Weixia Xu Liquan Xiao
Pingjing Lu Jinwen Li
Chengyi Zhang (Eds.)

Communications in Computer and Information Science 337

Computer Engineering and Technology

16th National Conference, NCCET 2012
Shanghai, China, August 2012
Revised Selected Papers

Springer

中国计算机学会
CCF

Communications in Computer and Information Science 337

Weixia Xu Liquan Xiao Pingjing Lu
Jinwen Li Chengyi Zhang (Eds.)

Computer Engineering and Technology

16th National Conference, NCCET 2012
Shanghai, China, August 17-19, 2012
Revised Selected Papers

Springer

Volume Editors

Weixia Xu
Liquan Xiao
Pingjing Lu
Jinwen Li
Chengyi Zhang

National University of Defense Technology
School of Computer Science
Changsha, Hunan, P.R. China, 410073

weixia_xu@263.net
marshell.xiao@gmail.com
pingjinglu@gmail.com
lijinwen@sina.com
chengyizhang@nudt.edu.cn

ISSN 1865-0929 e-ISSN 1865-0937
ISBN 978-3-642-35897-5 e-ISBN 978-3-642-35898-2
DOI 10.1007/978-3-642-35898-2
Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2012954675

CR Subject Classification (1998): C.1.2, C.1.4, B.7.1, B.4.3, B.3.2, B.2.4, B.8.2

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Preface

We are pleased to present the proceedings of the 16th Annual Conference on Computer Engineering and Technology (NCCET 2012). Over its short sixteen-year history, NCCET has established itself as one of the major national conferences dedicated to the important and emerging challenges in the field of computer engineering and technology. Following the previous successful events, NCCET 2012 provided a forum to bring together researchers and practitioners from academia and industry to discuss cutting-edge research on computer engineering and technology.

We are delighted that the conference continues to attract high-quality submissions from a diverse and national group of researchers. This year, we received 108 paper submissions, among which 27 papers were accepted. Each paper received three or four peer reviews from our Technical Program Committee (TPC) comprised of a total of 55 TPC members from academia, government, and industry.

The pages of this volume represent only the end result of an enormous endeavor involving hundreds of people. Almost all this work is voluntary, with some individuals contributing hundreds of hours of their time to the effort. Together, the 55 members of the TPC, the 12 members of the External Review Committee (ERC), and the 10 other individual reviewers consulted for their expertise wrote nearly 400 reviews.

Every paper received at least three reviews and many had four or more. With the exception of submissions by the TPC, each paper had at least three reviews from the TPC and at least one review from an outside expert. For the second year running most of the outside reviews were done by the ERC, which was selected in advance, and additional outside reviews beyond the ERC were requested whenever appropriate or necessary. Reviewing was "first read double-blind", meaning that author identities were withheld from reviewers until they submitted a review. Revealing author names once initial reviews had been written, allowed reviewers to find related and previous material by the same authors, which helped greatly in many cases in understanding the context of the work, and also ensured that the author feedback and discussions at the PC meeting could be frank and direct. For the first time in many years, we allowed PC members to submit papers to the conference. Submissions co-authored by a TPC member were reviewed exclusively by the ERC and other external reviewers, and these same reviewers decided whether to accept the PC papers; no PC member reviewed a TPC paper, and no TPC papers were discussed at the TPC meeting.

After the reviewing was complete, the Program Committee met at the National University of Defense Technology, Changsha, on June 15th and 18th, to select the program. Separately, the ERC decided on the PC papers in email and phone discussions. In the end, 27 of the 108 submissions (25%) were accepted for the conference.

First of all, we would like to thank all researchers who submitted manuscripts. Without these submissions, it would be impossible to provide such an interesting technical program. We thank all PC members for helping to organize the conference program. We thank all TPC members for their tremendous time and efforts during the paper review and selection process. The efforts of these individuals were crucial in constructing our successful technical program. Last but not least, we would like to thank the organizations and sponsors that supported NCCET 2012. Finally, we thank all the participants of the conference and hope that you have a truly memorable NCCET 2012 in Shanghai, China.

October 2012

Xu Weixia
Fu Yuzhuo
Zhang Minxuan
Xiao Liquan

Organization

Organizing Committee

General Co-chairs

Xu Weixia	National University of Defense Technology, China
Fu Yuzhuo	Shanghai Jiao Tong University, China
Zhang Minxuan	National University of Defense Technology, China

Program Chair

Xiao Liquan	National University of Defense Technology, China

Publicity Co-chairs

Lu Pingjing	National University of Defense Technology, China
Zhang Chengyi	National University of Defense Technology, China

Local Arrangement Co-chairs

Jiang Jiang	Shanghai Jiao Tong University, China
Li Jinwen	National University of Defense Technology, China
Huang Yan	Shanghai Jiao Tong University, China

Registration and Finance Co-chairs

Yu Meijuan	Shanghai Jiao Tong University, China
Zhang Junying	National University of Defense Technology, China

Program Committee

Han Wei	Xi'an Aeronautics Computing Technique Research Institute, China
Jin Lifeng	Jiangnan Institute of Computing Technology, China
Xiong Tinggang	Wuhan Digital Engineering Institute of China Shipbuilding Industry, China
Zhao Xiaofang	Institute of Computing Technology Chinese Academy of Sciences, China
Yang Yintang	Xi Dian University, China
Li Jinwen	National University of Defense Technology, China
Jiang Jiang	Shanghai Jiao Tong University, China

Technical Program Committee

Chen Shuming	National University of Defense Technology, China
Chen Yueyue,	Hunan Changsha DIGIT Company, China
Chen Zheng	Xi'an Aeronautics Computing Technique Research Institute, China
Du Huimin	Xi'an University of Posts and Telecommunications, China
Fan Dongrui	Institute of Computing Technology Chinese Academy of Sciences, China
Fan Xiaoya	Northwestern Polytechnical University, China
Fang Xing	Jiangnan Institute of Computing Technology, China
Gu Tianlong	Guilin University of Electronic Technology, China
Guo Donghui	Xiamen University, China
Hou Jianru	Institute of Computing Technology Chinese Academy of Sciences, China
Huang Jin	Xi Dian University, China
Ji Liqiang	Cesller Company, China
Jin Jie	Hunan Changsha Fusion Company, China
Li Ping	University of Electronic Science and Technology of China, China
Li Qiong	Inspur Information Technology Co. Ltd., China
Li Yuanshan	Inspur Information Technology Co. Ltd., China
Li Yun	Yangzhou University, China
Lin Kaizhi	Inspur Information Technology Co. Ltd., China
Lin Zhenghao	Tongji University, China
Lv Chunyang,	Jiangnan Institute of Computing Technology, China

Table of Contents

A Method of Balancing the Global Multi-mode Clock Network in Ultra-large Scale CPU

Zhuo Ma, Zhenyu Zhao, Yang Guo, Lunguo Xie, and Jinshan Yu

School of Computer,
National University of Defense Technology,
Changsha, Hunan,
410073 China
{ouhzam,zhaozhenyu,guoyang,xielunguo,yjs}@nudt.edu.cn

Abstract. It is a long-time discussed problem that the balancing of global multi-mode clock tree is. And there are many potential problems caused by the unbalanced clock tree, such as timing violations, density and power comsuption. In this article, an innovative balance method is opened by adopting the redundance clock mux. The basic idea of it is to maximize the reuse of the clock tree for other modes and keep the sub-clock tree within the sub-blocks unchanged. A demo chip on 40nm process has this balance skill verified, and makes the density, leakage and power comsuption deeply decreased.

Keywords: Ultra-large Scale, CPU, clock tree, balancing, multi-mode.

1 Introduction

As the chip size increasing and the function of chips to complicate, the design of clock network becomes more and more difficult.

For example, for a chip on 40nm process of which the die size is about 10x10mm2, the latency of global clock tree is more than 3ns. And the point is not only for the latency, for such a big chip, to build a global balanced clock tree is almost unachievable in general. Because the maximum skew of the clock could not be well-controlled, and also this issue may cause a lot of big hold violations[1].

Facing this problem, for most of large scale chips, building a group of local clock domains is a solution which is frequently used[2]. But the clock skew between two local clock domains is still big enough, especially in multi-mode designs[3].

The researching in this article fouced on openning a method, with which to build a global balanced clock network for each mode within a big chip is available. In section 2, the detailed problems now faced are list, and the deep-seated impacts are also provided. In section 3, the main skills of this paper are opened. In section 4, a detailed real chip is provided to be the demostrate. And finally, the conclusion is drawn in section 5.

2 Problem Faced

Chip size increasing is quantitative change to qualitative change. To make it clearly, Fig.1 shows a typical clock network with multi-mode. There are several blocks belong to three clock domains, and three clock sources drive these loadings through three clock mux.

As a routine method, there should be synchronizers between each clock domains[4]. The following lists three cases:

Mode 1: set [ModeSelect0/1/2] to 000. Obviously, the clock latency of BlockA is 4.7ns, while that of BlockB is 4.4ns, so a lot of 0.3ns delays should be inserted after hold timing fixing. On the other hand, BlockC to other two domains is asynchrounous and to be false path, so it is no need to fix the timing between them.

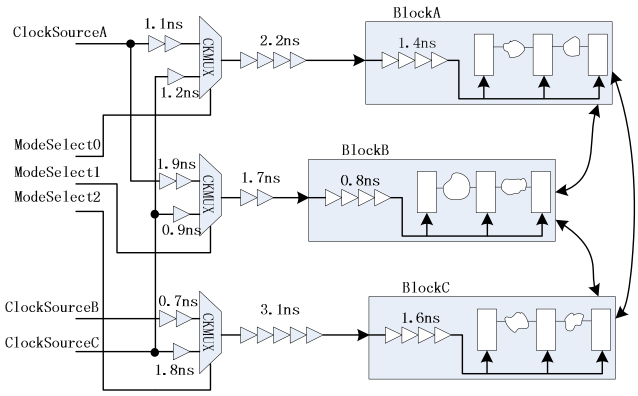


Fig. 1. A structure of Mult-mode Clock Network

Mode 2: set [ModeSelect0/1/2] to 111. Under this mode, all the sinks get the clock from ClockSourceC. This means all the registers on the chip belong to a same clock domain. It is important that the clock latency to each register need to be well matched. But with Fig. 1, it could be found that gaps of clock latency between these blocks are extreamly huge for fixing.

Then, inconvenient things will come if the design is carried on based on this structure.

First of all, because there are three clusters of registers in the design, of which the clock latency is 4.8ns, 3.4ns and 6.5ns respectively, the max clock skew for those registers is about 3.1ns and hard to be fix by ajusting the clock path only. In 40nm process, the delay of BUF1X is no more than 10ps in fast corner, and the delay of DELAY500 is no more than 500ps in fast corner, so 6 DELAY500 cells and 7 BUF1X cells are going to be inserted to fix the hold violation caused by that 3.1ns skew with signal integrity considered. If the number of datapath violations are massive, the cell count of inserted cells is enormous.

On the other hand, a lot of inserted cells make the density increasing fast, also the leakage power and dynamic power increasing fast. For the demo chip released in this paper, the increase of density will be 12 percents, which of leakage power will be 11.7 percents, and that of the total power consumption will be almost 20 percents. Fig. 2 shows the difference of the same area between preFixed stage (partA) and postFixed stage (partB). Because of the unbalanced clock structure, the density increased fast obviously.

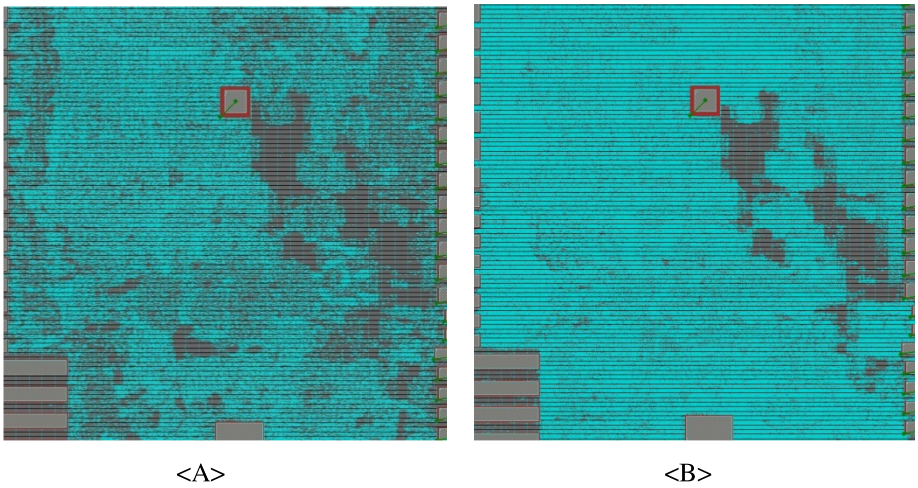


Fig. 2. Density Comparison between preFixed and post-Fixed

To settle the above problems, a better way is to balance all the clock sink in global[5]. But this seems to be a very difficult mission[6]. A simplified mode of this issue is shown in Fig. 3. This mode is consist of two registers, two clock muxes and a couple of clock buffer chains. When ModeSelect is 0, both of these two flip-flop are driven by ClockSource0, otherwise they are driven by ClockSource1.

Because Reg0 and Reg1 are both the startpoint and endpoint of true pathes, these two clock sinks should be balanced. So under mode 0 (set ModeSelect to 000), the delay of two clock pathes should be the same, which is shown in equation (1).

$$t_{preDelay0} + t_{postDelay0} = t_{preDelay2} + t_{postDelay1} \tag{1}$$

According this, the delay chains, including preDelay0, postDelay0, preDelay2 and postDelay1, are fixed after the clock pathes has been balanced. When the clock mode switches, another relation needs to be matched, which is shown in equation (2).

$$t_{preDelay1} + t_{postDelay0} = t_{preDelay3} + t_{postDelay1} \tag{2}$$

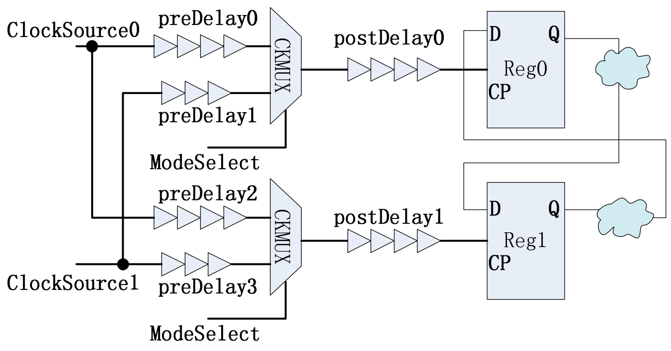


Fig. 3. A Simplified Mode of Multi-mode Clock Tree

To balance the clock skew under mode 1 (set ModeSelect to 1), only chains preDelay1 and preDelay3 could be adjust for the reasons list above, while chains preDelay0 and preDelay1 are fixed. For a small scale design, manually adjusting the preDelay* for each clock muxes is achievable. Once the scale of the design getting larger, it will be more than the ability designers could handle.

3 Balancing Method

A better way to settle this problem is to build a wide balanced clock tree for multi-modes, and this structure is called Multi-Mode Balanced Clock Tree (MMBCT). A simplified structure of MMBCT is shown in Fig. 4.

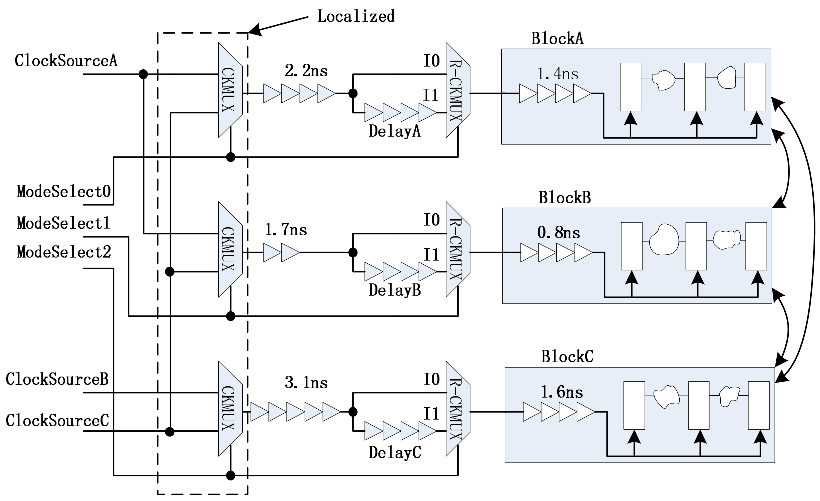


Fig. 4. The Multi-Mode Balanced Clock Structure

To summarize the MMBCT, the key points could be list as following:

- Use hierarchical design flow, and build the local clock tree for one mode;
- Localize all the intrinsic mode-select clock mux at the start point of the clock tree, and no any clock mux in blocks;
- Add redundance mode-select clock mux, and placed these muxes close to the block;
- Compensate the gap of the latency between clocks at the I1 pin of the redundance mux.

There is no difference when implement the block design. All the clock tree within the blocks will be reused for all the modes.

For details, the structure in Fig. 4 could be an example. Use the same analysis to the above. when the mode is set to Mode 1 [ModeSelect is set to 000], the clock latencies of BlockA/B/C are 3.6ns/2.5ns/4.7ns respectively, because those three clock muxes CKMUX are localized. In general, the redundance muxes R-MUX have no impact on the clock function of this mode.

But the exact scenario is in the other mode, when the mode is set to 0 (set ModeSelect to 111), the clock pathes to all the registers are going to be balanced as well. All the clock chains from top to the blocks boundary and internal the blocks are fixed. There is another flexible point at the I1 pin of R-MUX, which is used to compensate the clock skew under this mode. To modify the latency of DelayA/B/C is easy to balance all the clock pathes, which is described in equation (3).

$$2.2ns + t_{DelayA} + 1.4ns = 1.7ns + t_{DelayB} + 0.8ns = 3.1ns + t_{DelayC} + 1.6ns \tag{3}$$

In this case, the latencies of DelayA/B/C could be set to 1.1ns/2.2ns/0ns, then the latency from ClockSourceC to all the sinks are the same.

There are some advantages by adopting such a methodology. First of all, the clock tree under the other mode are fully reused, there is no need to have the original clock pathes or devices modified. Then, because almost all the clock sinks are balanced, there will be little hold violations within inter-blocks. And the third, the increased power is very small.

4 Demo and Result

A demo chip was build based on the above idea. Fig. 5 shows the snapshot of the fullchip, and the red line highlights the clock tracks. The logical structure of this demo chip is very similar to what is shown in Fig. 4. A pll here plays the role of clock source, and the mode select mux is placed close to it. There are ten R-MUXes which are placed near the clock port of each blocks and are highlighten in red frame respectively. An important parameter should be empasized that the size of this demo chip is over 12mmx13mm.

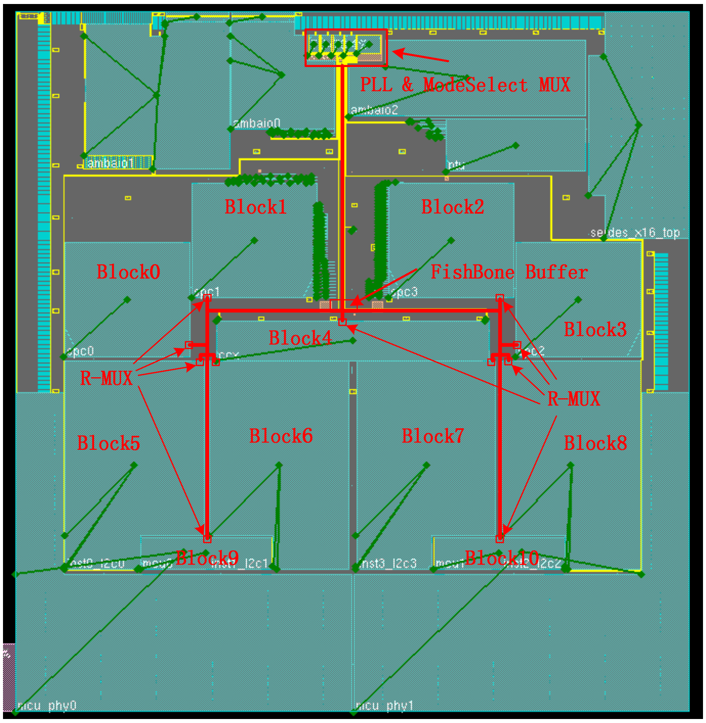


Fig. 5. The Snapshot of the Demo Chip

Table 1. Detailed Clock Information of the Demo Chip

Block Name	Internal latency	TOP lantency	Compensate Delay	Talk Relations
Block0	0.8ns	2.3ns	2.2ns	Block4
Block1	0.8ns	2.1ns	2.4ns	Block4
Block2	0.8ns	2.1ns	2.4ns	Block4
Block3	0.8ns	2.3ns	2.2ns	Block4
Block4	0.6ns	1.6ns	3.1ns	Block1~8
Block5	2.8ns	2.5ns	0ns	Block4
Block6	2.8ns	2.5ns	0ns	Block4
Block7	2.8ns	2.5ns	0ns	Block4
Block8	2.8ns	2.5ns	0ns	Block4
Block9	1.1ns	3.7ns	0.5ns	Block5/6
Block10	1.1ns	3.7ns	0.5ns	Block7/8

In Table. 1, the detailed information of this demo case is list. The "Internal latency" column shows latencies of the sub-clock tree for each blocks. While the "TOP latency" column shows the delay from clock source to the clock port of each blocks. And because there are R-MUXes beside clock ports, to compensate the latency differency is possible, and the compensatd delay values are list in column “Compensate Delay”.

According to Table. 1, the implementated clock structure is the same to the original mode 0, but all the clock sinks have been balanced under the other mode 1. As a result, because there is few of hold violations at inter-blocks, the increases of density, leakage power and total power are 7%, 5.5% and 11%. Compared to the calculated result which are shown in section 2, the MMBCT method impoves the performance of ultra-large scale chips rapidly.

5 Conclusion

The balance issue is a traditional problem on multi-mode clock trees. In most of the cases, it is not easy to satisfy all the requirements for each mode, and this will cause the problems, such as density, leakage, power. In this article, an innovative method, MMBCT, was opened to build a balanced clock tree in multi-mode applications. The basic idea of MMBCT is to maximize the reuse of the clock tree under each mode, and then the issues of density, leakage and power will be relieved.

Acknowledgements. Financial supports from the Natural Science Foundation of China (NSFC) under award 61076036 and 61076025 are greatly acknowledged, also greatly thankful to the KJ-11-04 project.

References

1. Xiao, L.F., Xiao, Z.G., Qian, Z.C., Jiang, Y., Huang, T., Tian, H.T., et al.: Local clock skew minimization using blockage-aware mixed tree-mesh clock network. In: 2010 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 458–462 (2010)
2. Fu, Q., Luk, W.S., Zhao, W.Q., Chen, S.J., Zeng, X.: Local refinement method for optimizing clock tree topology. In: 7th International Conference on ASIC, ASICON 2007, pp. 1110–1113 (2007)
3. Lee, H., Paik, S.W., Shin, Y.S.: Pulse Width Allocation and Clock Skew Scheduling: Optimizing Sequential Circuits Based on Pulsed Latches. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 29(3), 355–366 (2010)
4. Tsai, C.C., Lin, T.H., Tsai, S.H., Chen, H.M.: Clock planning for multi-voltage and multi-mode designs. In: 2011 12th International Symposium on Quality Electronic Design (ISQED), pp. 1–5 (2011)
5. Chen, Y.P., Wong, D.F.: An algorithm for zero-skew clock tree routing with buffer insertion. In: Proceedings of the European Design and Test Conference, ED&TC 1996, pp. 230–236 (1996)
6. Sulaiman, M.S.: A balanced clock network design algorithm for clock delay, skew, and power optimization with slew rate constraint. In: Proceedings of the IEEE International Conference on Semiconductor Electronics, ICSE 2002, pp. 62–66 (2002)

Hardware Architecture for the Parallel Generation of Long-Period Random Numbers Using MT Method

Shengfei Wu, Jiang Jiang, and Yuzhuo Fu

School of Microelectronics Shanghai Jiao Tong University,
Shanghai, P.R., China
{wushengfei,jiangjiang,fuyuzhuo}@ic.sjtu.edu.cn

Abstract. Random numbers are extremely important to the scientific and computational applications. Mersenne Twist(MT) is one of the most widely used high-quality pseudo-random number generators(PRNG) based on binary linear recurrences. In this paper, a hardware architecture for the generation of parallel long-period random numbers using MT19937 method was proposed. Our design is implemented on a Xilinx XC6VLX240T FPGA device and is capable of producing multiple samples each period. This performance let us obtain higher throughput than the non-parallelization architecture and software. The samples generated by our design are applied to a Monte Carlo simulation for estimating the value of π, and we achieve the accuracy of 99.99%.

Keywords: MT 19937 method, Hardware architecture, parallel generation, FPGA.

1 Introduction

High quality random numbers are epidemical applied in the domains of scientific applications[9]. One of the extremely important instances is the Monte Carlo method, such as particle transport in computational physics, quantum thermodynamic calculations and aerodynamic calculation. However, with the increasing complexity of the calculation model and calculation accuracy, the amount of Monte Carlo calculation to be done is growing. As all we known, the quality of random numbers has a significant impact on the accuracy of the Monte-Carlo simulations results.

Due to the advantages of high performance and reproducibility, pseudo-random number generators (PRNGs) based on linear recurrences modulo 2 are widely adopted in such simulations. One prevalent F2-linear PRNG is the Mersenne Twister (MT) [3], which is highly suited to simulation for its long period, multi-dimensional equidistribution, and high performance.

For large scale computing, one trend is the development of the parallelism of applications and building structure with the available parallel hardware resources, such as in FPGAs[10][11]. In modern system, providing fast streams of statistically reliable random numbers is the key component of many scientific applications, and designing parallelism PRNGs structure is the critical part.

Most hardware implementations of MT19937 are straightforward non-parallelized implementations of the original C-code [7], [8]. Shiro[1] proposed a hardware framework implement MT19937 in parallel by using 624 registers, but this is not area-efficient. Ishaan L.Dalal and Deian Stefan[2] developed a hardware architecture for generating parallel pseudo-random numbers, this method decrease the cost of hardware but the structural details of the multi-ported RAM and pipeline registers are not revealed.

In this paper, we develop a parallelized hardware framework for MT19937. More specifically, we make the following contributions:

1. We design a hardware architecture for parallel generating long-period random numbers using MT19937.
2. We implemented various degrees of parallelization architecture on a Xilinx Virtex 6 device.
3. We evaluate the proposed architecture using Monte-Carlo simulation for estimating the value of π.

2 Algorithmic Backgrounds

2.1 Mersenne Twister Method

The Mersenne Twister Method, which is a pseudorandom number algorithm based on a matrix linear recurrence over F2, is developed by Makoto Matsumoto in 1997[3]. The sample generation process of MT method is illustrated in Fig.1.

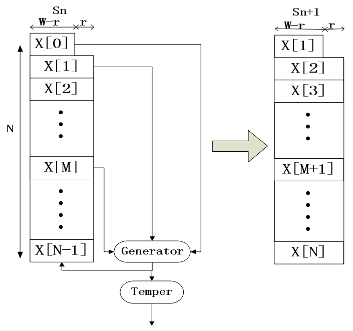


Fig. 1. Generation process of MT method

Where Sn represents the current state and Sn+1 is the next state. The state vector contains N words, each words contains w-bit. Three words in current state go through the Generator transform to the next state with a series of logic operation XOR and shift.

We could express the transform process in Fig.1 in the form of Eqn. (1)

$$x[N] = (x[0]^{w-r} \mid x[1]^{r}) \otimes x[M] \cdot A \tag{1}$$

In the equation $x[0]^{w-r}$ stands for "the upper w-r bits" of x[0], and $x[1]^{r}$ means "the lower r bits" of x[1]. So $(x[0]^{w-r}|x[1]^{r})$ means the new words consisted of the "the upper w - r bits" of x[0] and "the lower r bits" of x[1] .

The matrix A in the Eqn. (1) is the twist transformation matrix and defined in the form of Eqn. (2).

$$A = \begin{pmatrix} & 1 & & \\ & & 1 & \\ & & & \\ & & & 1 \\ a_{w-1} & a_{w-2} & \cdots\ \cdots & a_0 \end{pmatrix} \tag{2}$$

In order to get the long period and good equidistribution, the Mersenne Twister is cascaded with a tempering transform to compensate for the reduced dimensionality of equidistribution, the temper is defined in the case of Mersenne Twister as Eqn. (3)

$$\begin{cases} y = x_n \oplus (x_n >> u) \\ y = y \oplus ((y << s) \,\&\, b) \\ y = y \oplus ((y << t) \,\&\, c) \\ y_n = y \oplus (y >> l) \end{cases} \tag{3}$$

Where b and c are bitmasks and u, s, t and l are constant integers. Period reaches the theoretical upper limit 2^{Nw-r}-1.

In this paper, we use MT19937 algorithm and the coefficients for MT19937 are N=624,w=32,r=31,M=397,the period of MT19937 is $2^{19937-1}$.Thease characters could meet the demands for the simulation.

3 Hardware Architecture for MT19937

Fig.2 illustrates the overview of hardware architecture which we propose for MT19937 algorithm. It is composed of four components, i.e. the Address Unit, the Transform Unit, the Temper Unit, and the parallelized configuration consisted of BRAMs and registers.

The Transform Unit and the Temper Unit correspond to the Transform and Temper processes described in Eqn. (1) and Eqn. (3), respectively. The responsibility of the Address Unit is producing the appropriate addresses for the BRAMs. The connection and configuration of BRAMs and registers are the key points of parallelized architecture. They ensure the correctness of the generating results. We introduce the design details of BRAMs and registers in the next sub-section.

Two advantages of our parallel architecture are shown below:

1. Due to the capability of producing multiple random samples per clock cycle, we can obtain the high throughput.
2. The whole system can be built inside a single FPGA device so that the external resources such as off-chip memory are not required.

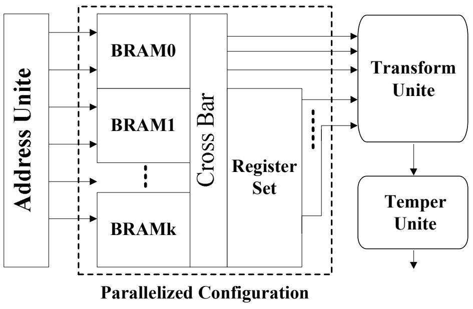


Fig. 2. The hardware architecture for MT19937

3.1 Structural Details for BRAMs and Registers

We use dual-port BRAMs in FPGA for the implementation and 3 degrees parallelization will be introduced as an example.

624 seeds are needed at the initial state of MT19937, we use one 206×32-bit, two 207×32-bit dual-port BRAMs and four registers to ensure state consistency for the given parallelized configuration. The initial vectors are distributed as Fig.3.

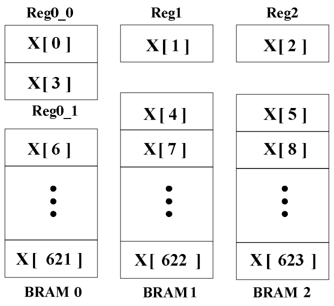


Fig. 3. The initial state of BRAMs and registers

It cost 3 read operations and 1 write operation for generating one random number using MT19937 method. We use two dual-port BRAMs in READ_FIRST mode to achieve. In READ_FIRST mode, data previously stored at the write address on the output latches, while the input data is being stored in memory (read before write).

Based on this character, the R/W details during the first 3 clock cycles is illustrated in Fig.4, vectors in bracket are the output data of the corresponding port. During the initialization, all I/O port of three BRAMs are in read mode. The numbers of address in the BRAMs (addr0,addr131,etc.) of Fig.4 correspond to the R/W addresses, which are controlled by the address unit and mapped to the proper BRAMs .During runtime, one port of BRAM is in read model and the other one is in write mode. Each of BRAMs R/W addresses is updated by the different counter synchronously.

In a single clock, because the BRAM is in READ_FIRST mode, the output data of the port in write mode is the previous data which was stored in the current write address and will be transformed to the appreciate register. Therefore, no more than 2 accesses will be required for each BRAM in a single clock cycle and we get all the vectors needed in one generation. Fig.5 depicts the hardware implementation of our proposed architecture.

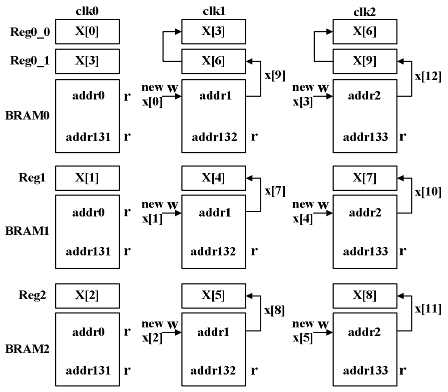


Fig. 4. R/W details during first 3 clocks

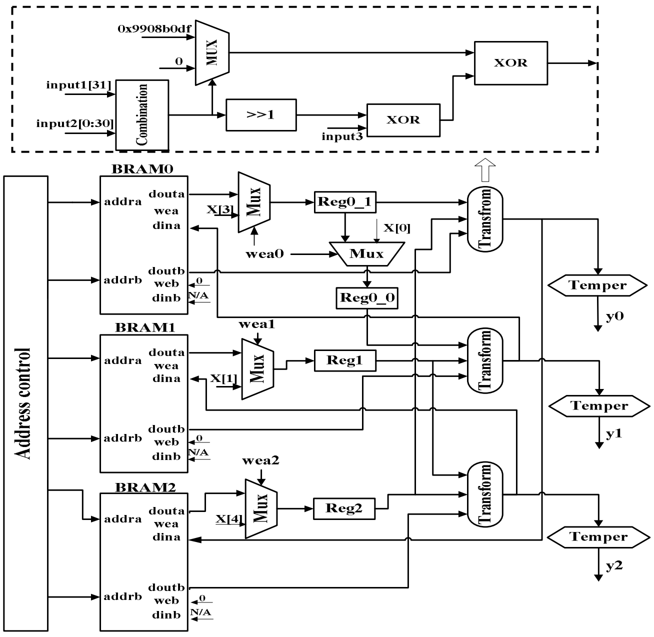


Fig. 5. Hardware architecture of 3 degree parallelization for MT19937

4 Implementation and Testing

This section presents the implementations and evaluations of the proposed architecture and framework using FPGA technology.

4.1 Implementation of Hardware Architecture

We implemented the architecture described in Section 3 on a Xilinx Virtex-6 XC6VLX240T (hosted on the ML605 evaluation board) FPGA device. The designs were coded in Verilog HDL and synthesized with Xilinx ISE 12.2. The initial design is simulated in Modelsim SE 6.5 to ensure functional correctness.

The BRAMs are configured in READ_FIRST mode. The output data of the port in write mode of BRAM which will be used in next iteration is transformed to the corresponded registers, meanwhile the output data of the other port and the old data in register were bypassed to the generator directly. The whole system is fully pipelined.

We have implemented a non-parallelized and four different (of degrees 2,3,4,6) long-period RNGs in various parallelization configurations for MT19937 .

Table 1 summarizes the resource usage and maximal performance statistics for each of these configurations. Hardware implementations have a better performance in throughput than the software. And as the result shown in Table 1, when we want to get more than one random number in one cycle, simply duplicating multiple instances of non-parallelized implementation will consume more resource than the parallelized architecture.

Table 1. Resource usage for various degrees of parallelism for MT19937

	Software	None	2-d	3-d	4-d	6-d
Flip-Flops	N/A	94	102	148	194	264
LUTS	N/A	152	167	245	323	405
BRAMs	N/A	2	2	3	4	6
Freq(MHZ)	N/A	312.0	260.3	253.9	238.3	232.2
Thruput(Gbps)	6.03	10.0	16.7	24.4	30.5	44.6

We also plot corresponding throughput vs. area for 2-d, 3-d, 4-d, 6-d as shown in Fig. 6. We can see that the throughput/area efficiency roughly remains constant as the degree of parallelism increases.

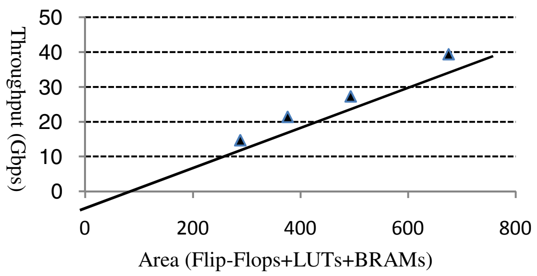


Fig. 6. Throughput vs Area for various parallel degrees

4.2 Application for Testing

Two standard statistical test suites, Diehard[11] and Crush which is from TestU01[12] were applied to test the quality of random numbers generated by the framework presented in Fig.5.

The implementations passed all the Diehard tests. All tests in Crush were passed except for two linear complexity tests. The reason is that all F_2-linear PRNGs producing bit sequences follow liner recurrences, thus they cannot have the linear complexity of a truly random sequence [4] and will fail the two tests.

For the purpose of checking the framework further , we apply the architecture to a practical application that Monte-Carlo simulation to estimate the value of π for further test. The basic principle of it is to generate points in the quarter unit square randomly, and then count the number of points falling inside the quarter unit circle. The value of π can be estimated by the Eqn. (12)

$$\pi = 4 \times \frac{\text{points inside the circle}}{\text{total points in simulation}} \tag{4}$$

The implementation of this Monte-Carlo application is shown in Fig. 7, and we chose 10000 points generated by the architecture proposed as the test samples. The estimated value of π is 3.141593 and this already achieve the accuracy of 99.99%. We believe that if we increase the number of sample points, the estimated value will be more accuracy.

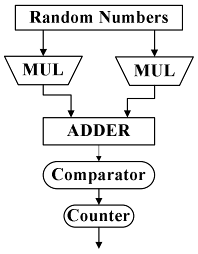


Fig. 7. Framework for estimating the value of **π**

5 Conclusion

In this paper, we have developed a hardware architecture for generating parallel long-period pseudo-random numbers using MT19937 method.

We implemented various degrees of parallelization architecture and a non-parallelization architecture for comparison on a Xilinx XC6VLX240T FPGA device. Due to the capability of capable of producing multiple samples each period, our architecture obtain higher throughput than the no parallelization architecture and software.

Finally, we apply the Monte-Carlo simulation for estimating the value of π for testing our architecture successfully.

Acknowledgment. Our work is supported by the IBM Shared University Research (SUR), System-level Virtualization Based on Reduced Hyper Multi-core Architecture, SUR201102X.

References

1. Konuma, S., Ichikawa, S.: Design and evaluation of hardware pseudo-random number generator MT19937. IEICE Trans. Info. Systems 88(12), 2876–2879 (2005)
2. Dalal, I.L., Stefan, D.: A Hardware Framework for the Fast Generation of Multiple Long-period Random Number Streams. In: Proc. 16th ACM Int. Symp. FPGAs, pp. 245–254 (February 2008)
3. Matsumoto, M., Nishimura, T.: Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. ACM Trans. Modeling and Computer Simulation 8(1), 3–30 (1998)
4. L'Ecuyer, P., Panneton, F.: Fast random number generators based on linear recurrences modulo 2: overview and comparison. In: Proc. 37th Conf. Winter Simulation, pp. 110–119 (December 2005)
5. Knuth, D.E.: The Art of Computer Programming, Seminumerical Algorithms, 3rd edn., vol. 2. Addison Wesley, Reading (1998)
6. Kurokawa, T., Kajisaki, H.: FPGA based implementation of Mersenne Twister. Sci. Eng. Rep. Nat. Def. Acad (Japan) 40(2), 15–21 (2003)
7. Pasciak, A.S., Ford, J.R.: A new high speed solution for the evaluation of monte carlo radiation transport computations. IEEE Trans. Nuclear Science 53(2), 491–499 (2006)
8. Sriram, V., Kearney, D.: An area time efficient field programmable Mersenne Twister uniform random number generator. In: Proc. Int. Conf. Eng. of Reconfigurable Systems & Algorithms, pp. 244–246 (2006)
9. Yuan, L., et al.: Software/Hardware Framework for Generating Parallel Long-Period Random Numbers Using the WELL Method. In: Field Programmable Logic, pp. 110–115 (2011)
10. Wenqi, B., et al.: A reconfigurable macro-pipelined systolic accelerator architecture. In: 2011 International Conference on Field-Programmable Technology (FPT), pp. 1–6 (2011)
11. Jiang, J., Mirian, V.: Matrix Multiplication based on Scalable Macro-Pipelined FPGA Accelerator Architecture. In: International Conference on Reconfigurable Computing and FPGAs, ReConFig 2009, pp. 48–53 (2009)

MGTE: A Multi-level Hybrid Verification Platform for a 16-Core Processor[*]

Xiaobo Yan, Rangyu Deng, Caixia Sun, and Qiang Dou

School of Computer Science,
National University of Defense Technology,
Changsha 410073
{xbyan,rydeng,cxsun,douq}@nudt.edu.cn

Abstract. With the widely application of multi-core multi-thread processor in various computing fields, simulation and verification of processors become increasingly important. In this paper, a multi-level hybrid verification platform called MGTE is designed and developed for a 16-core processer PX-16. MGTE supports software simulating and hardware emulating in module level, subsystem level or full-chip level, which is capable of verifying the processor during all the design periods from details to the whole. Also, MGTE supports the hybrid verification of behavior models, RTL codes and net lists, which is capable of improving the simulation performance. It's proved that MGTE can effectively ease the functional verification and preliminary performance evaluation of PX-16 processor.

Keywords: Multi-core Multi-thread Processor, Multi-level Hybrid Verification, Verification platform

1 Introduction

Processor is the core of a modern computer system. With the widely application of multi-core multi-thread processors in various computing fields, the growing size and complexity make the verification of the multi-core multi-thread processor more and more important [1]. Meanwhile, as the cost of processor manufacture keeps rising, simulation and verification of the processor should preliminarily evaluate the performance and power consumption before taping-out the initial assessment.

In this paper, a multi-level hybrid verification platform called MGTE (Making a Good Testing the Environment) is proposed for a 16-core processor PX-16. MGTE is based on the existing simulation and emulation tools, and is capable of either verifying the function or evaluating the performance and power of the processor.

[*] This work is supported by National Natural Science Foundation of China (NSFC) under grant No.61070036 and No.61103011, and HGJ Project of China under grant No. 2009ZX01043-003-002.

PX-16is a 16-core processor, which focuses the design on enhancing the system throughput. It employ sin-order issue mechanism to reduce the complexity of the processor core, so that the chip can integrate more processor cores. Fig. 1 shows a block diagram of the PX-16 processor. 16 general-purpose multi-thread processor cores (Core) with private L1 and L2 caches share a 4-bank L3 cache. Each bank of L3 cache connects to a directory controller (DCU) and a memory controller (MCU). PX-16 also integrates an on-chip interconnection network(NOC), a clock and reset control unit (CRU),a partial mode unit (PMU) and an IO controller (IOU).

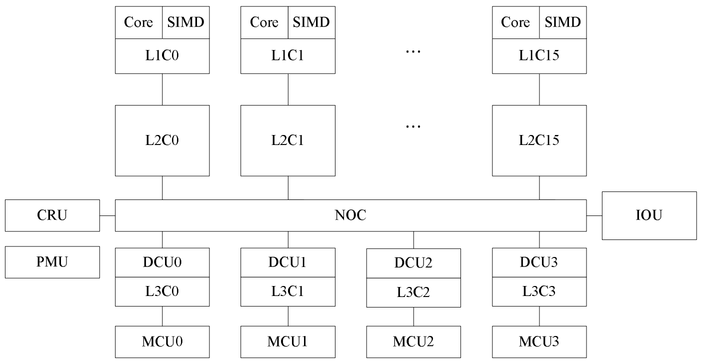


Fig. 1. PX-16 Block Diagram

2 Related Work

Verification is an important method for the successful implementation oftoday's CPUs. According to the objective of verification, it can be divided into functional verification and timing verification. According to the stage of verification, it can be divided into pre-silicon verification and post-silicon verification. Moreover, pre-silicon verification can be subdivided into pre- synthesisverification and post-synthesis verification.According the verification level,it can be divided into the architecturelevel verification, behavioral level verification, register transfer level (RTL) verification, netlist level verification, and prototype-chip level verification.According to themethodologyof verification, it can be divided into formal verification, simulation verification, FPGA verification, prototype system verification, coveragedriven verification, assertion verification, pipeline behavior verification, and hardware and software co-verification. A single verification method is notenough to guarantee the correctness of the processor design, so a variety of verification methods are requiredthrough the processor design [1].

Simulation verification ensures the correctness of processor design by modeling the design at different abstraction levels and observing the response of the model under the external test stimulus.

Simulation verification tools can model the processor with high-level language (such as the C language, System-Verilog language, etc.), such asSAM [2] architecture simulatorfor OpenSPARC T2[3] and SimpleScaler architecture simulator for general-purpose processor.SAM models not only the processor design but also the disk, off-chip memory, multi-processor system etc. SimpleScaler [4] is able to simulate the CPU, cache, memory system, computer architecture etc. It adopts MIPS/DLX-like instruction set, provides compile tool chain and a number of test programs, and supports the analysis of the processor and cache system. High-level languagesimulation cannot accuratelyreflect the details of the processor architecture.But it is usually fast and the simulation speedcan reach MHz or more.Therefore,the simulation at this level is usually used for software development.

Simulation verification can also model the processor with hardware description language (i.e. Verilog, VHDL, etc.). For example, Sims [5] is a simulator for OpenSPARC T2 including thousands of assembly test programs, which is capable of modeling real hardware architecture with the supports of specific EDA tools (i.e. VCS [6] from Synopsys inc. or IUS [7] from Cadence inc.). Hardware logic-level simulation serves a relatively low level abstraction, which can accurately reflect the details of the processor architecture, but the simulation speed is usually slow, which is usually only a few hundred Hz to several thousand Hz.

Simulation verification can also be done on hardware emulator (i.e. PXP from Cadence inc., EVE from EVE inc., etc.). The processor design can be loaded into the hardware emulator to run a real operating system and applications.The log and waveform records the hardware status and results for further analysis. Compared with the previous two verification methods, hardware emulation verification can accurately reflect the detail structure of the processor, but also has a fast simulation speed around MHz-scale. However, the hardware emulation platform is usually expensive, and limited by the number of logic gates. Moreover,the debug process is relatively complex and more time-consuming.

During the whole process of CPU design, these three verification methods will be collaborative at different stages. MGTE simulation verification platform supports both logic-level simulation on common computer servers and hardware emulation on hardware emulators.

3 MGTE Verification Platform

MGTE verification platform is mainly composed of the software environment and the hardware environment. Thesoftware environment is responsible for configuring the processor, compiling the test cases and controlling the simulation process.The hardware environment is responsible for the synthesis of the processor design codes, off-chip DDR memory model, FLASH memory model and the off-chip I/O device model, as well as loading the PLI libraries, the initialization codes, the hardware monitor/checker and simulating on the hardwareemulator platform.

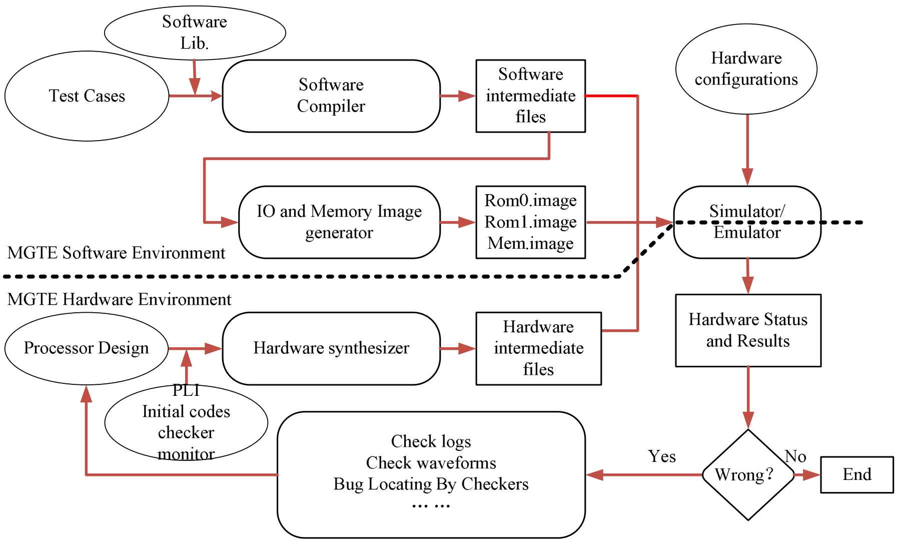

Software Lib.
Test Cases
Software Compiler
Software intermediate files
Hardware configurations
IO and Memory Image generator
Rom0.image Rom1.image Mem.image
Simulator/ Emulator
MGTE Software Environment
MGTE Hardware Environment
Processor Design
Hardware synthesizer
Hardware intermediate files
Hardware Status and Results
PLI Initial codes checker monitor
Check logs Check waveforms Bug Locating By Checkers … …
Yes
Wrong?
No
End

Fig. 2. The workflow of MGTE simulation verification platform

3.1 Workflow of MGTE

Fig. 2 shows the workflow of MGTE verification platform.

At first, the software compiler converts the test case and the software function library into some intermediate files, including binary executable files, the symbol table, and the disassembled codes with instruction and data addresses. The symbol table contains the instruction addresseswhere the program exits normally and abnormally.The disassembled codes are used to debug errors. IO and memory image generator converts the binary executable files to the machine codes and data and divides them according to the address space.The codes and data in IO space are stored in two files: Rom0.image and Rom1.image. The codes and data in memory space are stored in Mem.image. At the same time, the hardware design codes of the processor, together with the initialization code, PLI libraries, checkers and monitors,are compiled by the hardware synthesizerand the hardware intermediate codes are generated.

After compiling, the verification can begin. When the hardware codesstart to work, the IO and memory image files and the arguments extracted from the symbol table will be loaded into the simulator or emulator. The hardware status and results will be printed to the server screen and the log files. Once the real-time errors in hardware design are detected by the checkers, the simulator will pause and invoke a debug interface to check the errors. After the simulation or emulation is finished, the software environment will re-analyzethe logs and hardware dump files.If the test case exits from the bad address or if the hardware status is inconsistent, MGTE platform will report the sequence of the error instructions from the logs or the address sequence of the inconsistentdata for troubleshooting.

3.2 MGTE Hardware Environment

MGTE hardware environment refers to the PX-16 processor full-chip design codes and simulation related codes, including the global flow control module, processor model, memory model, Flash model, the IO reference design model, global initialization module and global checkers and global monitors, as shown in Fig. 3.

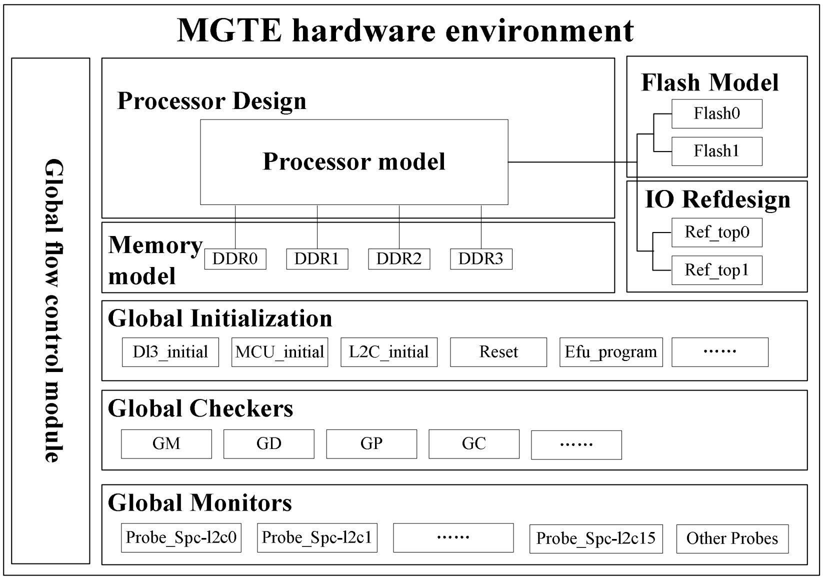


Fig. 3. MGTE the hardware environment structure

The global flow control module is mainly responsible for controlling the work flow of the processor, loading the test cases, initializing the processor, starting the global checker and the global monitors.

The processor design codes contain RTL codes and some behavior models. The RTLcodesare real hardware design, and the behavior models are used for fast simulation.

The off-chip DDR memory model, abbreviated as the memory model, contains not only the clock accurate DDR models, but also a fast DDR behavioral model. Its initial data are loaded from Mem.image.

The flash model is responsible for the simulation of the off-chipflash chips, which supplytwo extraFlash chips to store the system initialization machinecodes and binary data. The flash model also contains a real clock accurate model and a fast behavioral model.Its data areloaded fromRom0.image and Rom1.image.

The IO reference design model is responsible for the simulation ofthe off-chip IO devices, which can supply two simple IO devices.

The global initialization module is responsible for the fast initialization of the full system. In a real system, the initialization process is done by hardware instructions, which is extremely slow.In order to speedup the initialization process, most units of the processor can be fast initialized by forcing a set of signals or embedding some hardware behavioral initial codes. The fast initialization units includes Dl3, MCU, L2Cache, CRU, PMU and other miscellaneous modules such as the memory model, the FLASH model and the IO reference design model.

The global checkersare basically responsible for fast error detection and hardware debug by checking the status and results of the simulation. For examples, Golden Memory checker (GM) uses a memory reference model to check the correctness of memory subsystem; Golden Directory checker (GD) usesL1C directory model to check the protocol correctness of the L1 Cache coherency and consistency; Golden Packet checker (GP)checks the correctness of the input and output packets through NOC; Golden Cache Coherencychecker (GCC)usesa hardware modelto check the protocol correctness of L2Cache coherency and consistency; Golden MCU checker uses a behavioral DDR model under L3Cache to check the correctness of MCU; Golden DCU checker checks the correctness of the DCU internal protocol; Golden L2C checker checks the correctness of the L2Cache internal protocol.

The global monitorsare responsible for not only monitoring the processor simulation but also collecting the statistical system performance, including the average on-chip memory access bandwidth, the average off-chip memory access bandwidth, the DMA-accessbandwidth and processor computing resource utilization. Based on these statistics, programmers or hardware designers can analyze the characteristics of the applications and find out the hardware and software bottlenecks for improvement.

3.3 Advantages of MGTE Platform

MGTE is a multi-level hybrid simulation verification platform that covers from simulation to emulation. It can meet the processor verification requirements at different design stages,and make a lot of optimizations for simulation performance.

1) Supporting Multi-level Cross-Platform Verification

MGTE platform supports multi-level simulation verification.It supports both full-system simulation but also sub-system simulation, including the processor core sub-system simulation, the memory sub-system simulation, and the IO sub-system simulation.It also supportsmodule-level simulation, including the Core module simulation,the L2Cache module simulation, the NOC module simulation, the L3Cache module simulation, the MCU modulesimulation and the IOU module simulation. All the simulation and emulation levels can be selected by a run-time parameter.

MGTE platform also supports cross-platform verification. It not only supports software simulation on IUS environment or VCS environment, but also supports emulationon the PXP or EVE platform. Through the support of the cross-platform simulation and emulation verification, MGTE platform can process pseudo-assembly-level test cases on module level and subsystem level software simulation, assembly-level test caseson full-chip simulation and real applications on full-chip emulation after loading operating system.

Multi-level cross-platform simulation verification method makes the processor verification unified, structured and parameterized and eases the verification of the PX-16processor from the details to the whole system.

2) Supporting Hybrid Simulation Verification

MGTE platform uses different configuration files to achieve hybrid simulation of the behavior models, RTL codes and netlists. The configuration entryfollows the form "mod_select: <*mod_name=mod_type*>", where*mod_name*can be the name of the module with multiple implementations (such as "mcu" for the module MCU), the name of the subsystem with multiple implementations (such as "mem_subsys"for the memory subsystem) or the name of the common components with multiple implementations (such as "regfile"for the register files,"ram" for theon-chip rams, "clockgate" for the clock gating modules, etc.); *mod_type* is the configuration of the hardware modules, subsystems or components: "real" means thereal RTL codes areused, "model"means a null model is used, "sim"means a behavioral model is used, and "netlist"means the netlist model is used. For example, if MCUis required to be configured as a behavioral model, the line "mod_select: <mcu=sim>" should be insert into the configuration file. In addition, the number of processor cores can be configured too.For example, "mod_num: <corenum=1>" means the processor is configured to be a single-core processor, and the other 15 coreswill usenull models.

MGTE platform analyzes the hardware configuration file, and dynamically generates a compile file list, whichis further compiled by the IUS or VCS hardware compiler. The parameter conf is used to select the hardware configuration file, which is customized according to the simulation requirements. For example, the hardware configuration file"fastcmp16.conf" is used for the hybrid fast simulation of a 16 core system by setting the parameter conf as "fastcmp16".

The hybrid simulation of behavioral models, RTL codes and netlists greatly increase the simulation performance.The fast behavioral modelsare used for the modules whose functional correctness isguaranteed, and the empty models are used for the unused modules, so that thechip complexity is reduced, and the bugs in the hardware logic can be exposed rapidly.

In addition to these two important characteristic, the MGTE platform also has the following advantages:

- **Easy to use.** MGTE platform provides a few parameters to flexiblyselect a part or whole test cases for convenient regression testing.In addition, MGTE platform can simulate multiply tasksparallelly.Most operations of MGTE platform need only onesingle command, which greatly simplifies the human-computer interaction.
- **Convenient to troubleshoot.**MGTE platform supports automatic checkpoint, which automatically saves the state of a long-run test case in a fixed time interval for re-running and troubleshooting. In addition, MGTE platform supplies a rich number of online and offline checkers to check the status and results of the simulation. The online checkers includea lot of PSL assertions andhardware checking logic. The offline checkers include the L1/L2 coherency and consistency checking toolsbased on the dumpedinternal RAM data and the error path analysis tools based onthe log files.

4 SimulationExperiments

MGTE platform is designedto guarantee the correctness of PX-16 processor and to evaluate its performance preliminarily. Three kinds of test cases are used on MGTE platform: the pseudo-assembly-level test cases,the assembly-level test casesas well as the application-level test cases.

Pseudo assembly-level test casesare used to observe the system status through forcing the internal interface signals, which are mainly for the subsystem level and module level verification, as shown in Table 1.

Table 1. Pseudo assembly-level test cases running on MGTE platform

Name	Count	Functionality
core_sim	726	test the processor core and core module
mem_sim	253	test on-chip memory subsystem
io_sim	126	test IO subsystem and IO module
l2csim	19	test L2 Cache module
Nocsim	37	test NOC module
l3csim	39	test L3 Cache module
Mcusim	25	test MCU Module

Assembly-level test cases are used mainly in the full-chip software simulation, which are designed to test the traditional functions, to verify the new features of PX-16 processor, to evaluate the performance preliminarily of PX-16 processor as well as to do some random tests.Thesetest cases are shown in Table 2.

Table 2. The assembly-level test cases running on MGTE platform

Type	Name	Count	Functionality
traditional function	Cmp1	881	Single-core, 8-core and 16-core basic functionality; core,ISA, virtual memory management, error handling, performance management, exception handling, memory access, and other miscfunctionality.
	Cmp8	654	
	Cmp16	276	
directed function	SIMD	68	SIMD functionality
	DMA	36	DMA functionality.
	CReg	10	Control registersfunctionality.
	Reset	1	Chip reset functionality.
	PM	16	Partial mode functionality.
	IPI	2	Inter-processorinterruption functionality.
	HardAtom	2	Hardware atomic instructions functionality.
Performance Evaluation	Stream	12	Performance of Stream program.
	Ld	1	Memory accesslatency at different levels.
	Dd	18	DMA performance under different situations.
	Linpack	1	Performance of LINPAC.
	Atom	12	Performance of locks and barrier
Random Test	MixISA	*	Random instruction sequence.
	MixCC	*	Random critical region and randomsynchronization.

Application level test casesare mainly used on PXP and EVE emulation platform after loading the operating system.These benchmarks contain SPEC CPU2006, SPEC OMP2001, NPB, Stream, Matrix multiplication and LINPACK.

4.1 Simulation Interface

The entrance program of MGTE platform is**mgte_run**.There are two types of parameters for the program: action parameters and options parameters. There are three action parameters: "run" indicates to run some testcases; "help" indicates to print help information; "merg" represents to merge the coverage data; "clr" is used to empty the temporary directory.

Somebasic option parameters are as follows:

sim=*: * indicates the type of simulation, such assim=chip,chip_vcs, pld_chip, eve_chip, core_sim, mem_sim, io_sim, coresim, l2csim, nocsim, l3csim, mcusim,iosim and so on.

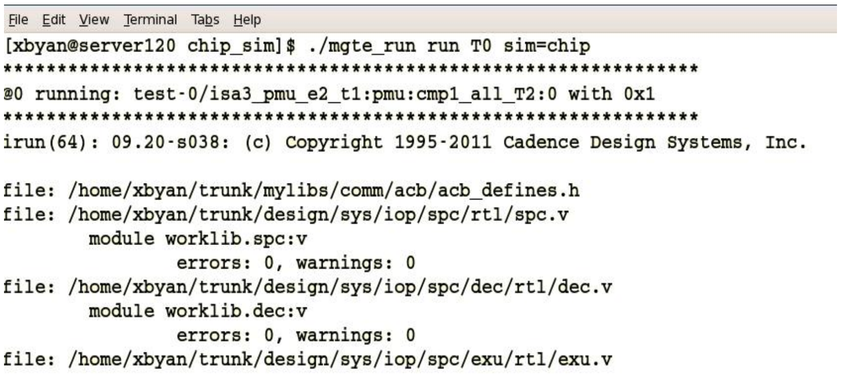
conf=*: * specifies a particular hardware configuration name.The default value is the norm, meaning to use RTL codesfor the whole system.

T [n]: n is a number or a rangelike"m-n", indicating which test group(s) will be tested. The bracket is not required when inputting the command line.

test=*: * indicates the names of the test cases, where wildcards can be used.

nproc=*: * indicates the number of processes to run the test task. The default value is 1.

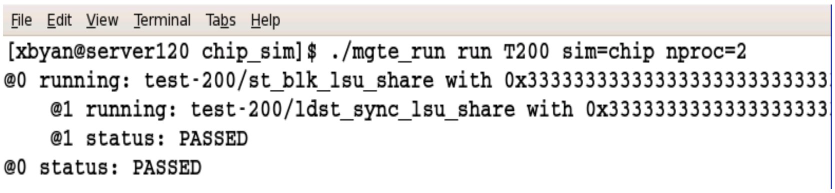
Fig. 4 and 5 give the consoles of the MGTE platform for serial single-task simulation and parallel multi-tasking simulation.



```
[xbyan@server120 chip_sim]$ ./mgte_run run T0 sim=chip
****************************************************************
@0 running: test-0/isa3_pmu_e2_t1:pmu:cmp1_all_T2:0 with 0x1
****************************************************************
irun(64): 09.20-s038: (c) Copyright 1995-2011 Cadence Design Systems, Inc.

file: /home/xbyan/trunk/mylibs/comm/acb/acb_defines.h
file: /home/xbyan/trunk/design/sys/iop/spc/rtl/spc.v
        module worklib.spc:v
                errors: 0, warnings: 0
file: /home/xbyan/trunk/design/sys/iop/spc/dec/rtl/dec.v
        module worklib.dec:v
                errors: 0, warnings: 0
file: /home/xbyan/trunk/design/sys/iop/spc/exu/rtl/exu.v
```

Fig. 4. The console for single-task test on MGTE



```
[xbyan@server120 chip_sim]$ ./mgte_run run T200 sim=chip nproc=2
@0 running: test-200/st_blk_lsu_share with 0x3333333333333333333333333
    @1 running: test-200/ldst_sync_lsu_share with 0x333333333333333333
    @1 status: PASSED
@0 status: PASSED
```

Fig. 5. The console for parallel multi-task test on MGTE

4.2 Simulation Results

MGTE platform is used through all the stages of PX-16 processor verification, from module level to full-chip level. With the help of MGTE platform, 16459 bugs are fixed and the bug distribution is shown in Table 3.

Table 3. The bug distribution found by MGTE platform

Simulation level	**Bug count**	**Bug type**
Module Level	11895	Syntax, logic design, new function, pipeline, environment, etc.
Subsystem Level	4321	Subsystem interface, L2Cache coherency and consistency, logic design, environment, etc.
Full-Chip Level	243	Full-chip interface, L2/L1 Cache coherency and consistency, logic design, IO consistency, environment, etc.

MGTE platform greatly increases the simulation speed bymixing the real RTL codes and behavioral modelsin simulation verification. Table 4 shows the simulation speed in the hybrid mode compared with the real RTL codes mode. Only the real RTL codes emulation is performed on PXP and EVE platform. AIntel Xeon server is used in the software simulation, configured with 4 Xeon E5540 CPUs operating at 2.53GHz and 8MB on-chip Cache within each CPU.

Table 4. Simulation performance atdifferent levels

Simulation level	**Simulation Speed(Hz)***		**Test cases**
	Real	**Hybrid**	
Module level	≈200	≈2000	Pseudo assembly benchmarks
Subsystem level	≈50	≈500	Pseudo assembly benchmarks
Full-chip level (soft)	≈10	≈100	Assembly benchmarks
Full chip on PXP	300k~1.5M	-	OS and applications on full-chip mode
Full chip on EVE	1M~3M	-	OS and applications on partial mode

* thespeed is measured as number of cycles per second in processor core clock domain.

5 Conclusions

MGTE is a multi-level hybrid verification platform based on hardware design codes, which supportsboth software simulation and hardware emulationfor the module level, sub-systemlevel and full-chiplevel.It eases the verification during all the stages of processor design, from the details to thewhole system. It also supports the hybrid simulation of behavioral models, RTL codes and netliststo improve the simulation performance. MGTE effectively supports the functional correctness verification and preliminary performance evaluation of the PX-16 processor.

References

1. Hu, J., et al.: A Study on CPU Chip-Oriented Verification Technology. Microelectronics 37(1), 16–23 (2007)
2. OpenSPARC^{TM} T2 Core Microarchitecture Specification, Revision A. Sun Microsystems, Inc. (December 2007)
3. Nussbaum, D., Fedorova, A., Small, C.: An overview of the Sam CMT simulator kit. Sun Microsystems, Inc., Mountain View (2004)
4. Austin, T.M., Larson, E., Ernst, D.: Simplescalar: An infrastructure for computer system modeling. IEEE Computer 35(2), 59–67 (2002)
5. OpenSPARC^{TM} T2 Processor Design and Verification User's Guide, Revision A, Sun Microsystems, Inc. (2008)
6. Synopsys VCS^{TM} training, Synopsys, Inc. (2008)
7. Verilog Simulation User Guide, Product Version 9.2, Cadence, Inc. (July 2010)

An Efficient Parallel SURF Algorithm for Multi-core Processor

Zhong Liu, Binchao Xing, and Yueyue Chen

Microelectronics and Microprocessor Institute, School of Computer,
National University of Defense Technology,
Changsha, China
{zhongliu,yychen}@nudt.edu.cn,
xbch8511@163.com

Abstract. In this paper, we propose an efficient parallel SURF algorithm for multi-core processor, which adopts data-level parallel method to implement parallel keypoints extraction and matching. The computing tasks are assigned to four DSP cores for parallel processing. The multi-core processor utilizes QLink and SDP respectively to deal with data communication and synchronization among DSP cores, which fully develops the multi-level parallelism and the strong computing power of multi-core processor. The parallel SURF algorithm is fully tested based on 5 different image samples with scale change, rotation, change in illumination, addition of noise and affine transformation The experimental results show that the parallel SURF algorithm has good adaptability for various distorted images, good image matching ability close to the sequential algorithm and the average speedup is 3.61.

Keywords: parallel, SURF, image matching, multi-core processor.

1 Introduction

With the rapid development of digital image processing and microelectronics technology, image matching is used widely in aided navigation of aircraft, face recognition, image stitching, image retrieval, medical diagnosis, natural resource analysis, and weather forecasting[1][2]. These applications require highly efficient and even real-time image matching algorithm. Lowe's SIFT (Scale Invariant Feature Transform)[3] algorithm is a nice image matching algorithm for scale change, rotation, illumination, noise and even affine transform images, but the SIFT suffers from high computation complexity. Herbert Bay presents SURF (Speed Up Robust Features)[4] algorithm with fast computing speed, robust keypoints and high parallelism. However, the SURF algorithm on a single chip can not meet the needs of real-time applications. With the emergence of multi-core processors, it becomes an effective method to improve performance of algorithms by developing parallel algorithm based on multi-core processors.

2 The Architecture of Multi-core Processor

YHFT-QDSP [5] is a high performance heterogeneous multicore processor, which is shown in Fig.1. It combines four enhanced YHFT-DSP/700 cores and a RISC (Reduced Instruction Set Computing) core in one chip. All of the cores can play their roles quite well with the advantage of high bandwidth and low delay of the multi-core on-chip structure. The memory space of each core is independent individually. The RISC core runs embedded operating system and provides a number of external interfaces for dealing with user interaction, peripheral device management, while the DSP(Digital signal processing) cores are mainly used to improve data processing.

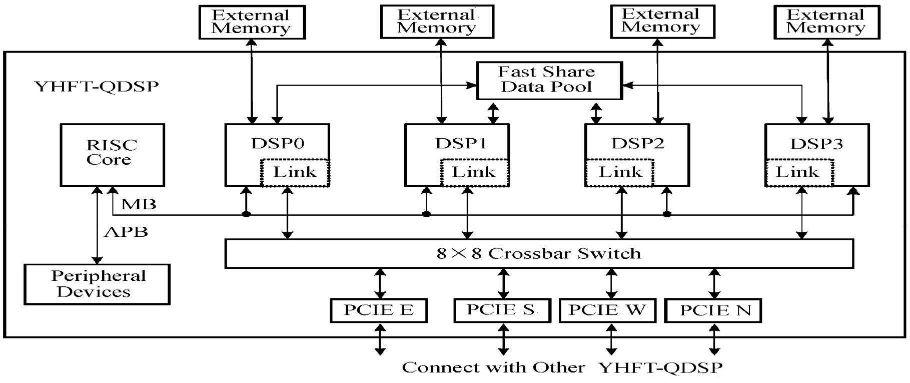


Fig. 1. Architecture of YHFT-QDSP

A multi-core system that runs parallel applications well needs a highly efficient communication mechanism. YHFT-QDSP provides two data communication between DSP cores: (1) A FSDP (fast shared data pool) is proposed for scattered data transfer between different DSP cores, it is designed for high bandwidth communication with hardware synchronization and suitable for data flow applications.(2) A link-crossbar Switch-PCIE named QLink is designed for bulk data transfer among the DSP cores. The Qlink data path consists of a link port, a crossbar switch and a PCIE. A data path is established by linking ports from the source to the destination through the crossbar switch. The applications of loosely coupling computation can benefit from Qlink, since Qlink does not disturb the CPU during data transmission process.

3 The Serial SURF Algorithm

The basic principle of SURF is similar to that of the SIFT algorithm. But compared to SIFT algorithm, the computational speed of SURF algorithm is improved greatly, mainly due to the use of the box filers and haar wavelet based on integral image, which makes the computation time a constant independent of the changing of filter size. Integral image further saves a lot of redundant computing.

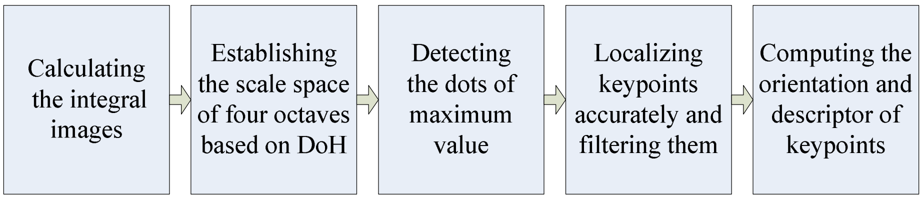

Calculating the integral images
Establishing the scale space of four octaves based on DoH
Detecting the dots of maximum value
Localizing keypoints accurately and filtering them
Computing the orientation and descriptor of keypoints

Fig. 2. The implementation process of SURF algorithm

As shown in figure 2, the implementation process of SURF algorithm includes: calculating the integral images, establishing scale space, detecting maximum value, localizing keypoints and filtering, and computing the orientation and descriptor of keypoints.

3.1 Calculating the Integral Image

Though the integral image theory is proposed by Viola and Jones[6], it is firstly applied to box filters by Simard [7]. The value of integral image element is the sum of the rectangle area pixel values between the point and the origin, that is, the sum of value of all pixels in the upper left. For instance, for an image I and a point P (x, y), the integral image value of the point P is the sum of pixel values within the rectangular area in which the origin point and point P are on the diagonal, shown as formula 1:

$$I_{\Sigma}(x, y) = \sum_{i=0}^{i \le x} \sum_{j=0}^{j \le y} I(x, y) \tag{1}$$

In an integral image, the sum of all pixels is calculated using four values within the rectangular area, and the computation complexity is independent of the rectangle size. For example, to calculate the sum of pixels of the rectangular region whose vertices is A, B, C, D, the result is shown as formula 2. Regardless of the size of the rectangle, all four values can be calculated, and the computational time is a constant. That explains how integral image can greatly improve SURF computing speed, especially for the large-scale images.

$$sum = A + D - B - C \tag{2}$$

3.2 Establishing the Scale Space

Similar to the SIFT, SURF presents image scale space using image pyramid. In a certain sample step, a series of boxes with different sizes are used to calculate DoH(Determinant of Hessian) at each sample point, which generates the first octave of scale space; then the sample step is increased, and a series of boxes with different size are used to calculate DoH at each sample point, which generates the second octave of scale space. In a similar way, the third and the fourth octave of scale space are obtained. SIFT algorithm's new octave is obtained by sampling the middle image layer of old octave, while SURF generates new octave through increasing the sample step for the integral image. The size of the box filter is shown in formula 3:

$$l = 3 * (2^{(o+1)} * (i+1) + 1) \tag{3}$$

Where l is the size of the filter, o is the octave order of scale space, *i* is image layer order of octave. When the filter box size is 9, corresponding to Gaussian standard deviation 1.2, if different sizes of box filers have to maintain a certain layout of the filter, the corresponding standard deviation of the Gaussian can be calculated as shown in formula 4:

$$\sigma_{approx} = \frac{1.2}{9} * l \tag{4}$$

3.3 Detecting Keypoints

SIFT adopts DoG approximately to present the LoG, while SURF algorithm uses the box filters to simplify the DoH. DoH is a keypoint detecting operator, which is the determinant of formula 5.

$$H(f(x,y)) = \begin{bmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial xy} \\ \frac{\partial^2 f}{\partial xy} & \frac{\partial^2 f}{\partial y^2} \end{bmatrix} \tag{5}$$

Where f (x, y) is continuous function. The determinant of HESSION is capable of detecting the minimum and maximum values of function f (x, y). SURF applies it to detect the extreme of an image: the function f (x, y) is replaced by the image I (x, y), correspondingly, the second derivative of f (x, y) is replaced by the convolution of the image and the second derivative of Gaussian kernel, which are shown in formula 6:

$$H(I(x,y,\sigma)) = \begin{bmatrix} L_{xx}(x,y,\sigma) & L_{xy}(x,y,\sigma) \\ L_{xy}(x,y,\sigma) & L_{yy}(x,y,\sigma) \end{bmatrix} \tag{6}$$

Where $L_{xy}(x,y,\sigma)$, $L_{xx}(x,y,\sigma)$ and $L_{yy}(x,y,\sigma)$ present the image convolution at point (x, y) with the second derivative of Gaussian kernel. SURF algorithm uses a box filter with weights to approximate the Laplace operation of HESSION matrix.

3.4 Keypoints Detecting, Localizing and Filtering

When scale space construction is completed, the local maximum value point can be detected in each octave. Only if the DoH of a sample point is greater than 8 adjacent sample points in the same layer image and 9 sample points respectively in the adjacent scale correspondingly, the sample point is treated as candidate keypoint.

Localizing and filtering keypoints: in scale space, the Taylor expansion at the candidate keypoint (x, y, σ) is shown in formula 7.

$$H(x) = H + \frac{\partial H^T}{\partial x} x + \frac{1}{2} x^T \frac{\partial^2 H}{\partial x^2} x \tag{7}$$

The derivation of the formula 7 is done and makes it zero, find the extreme position of the keypoint, as shown in formula 8:

$$x' = -\frac{\partial^2 H}{\partial x^2}^{-1} \frac{\partial H}{\partial x} \tag{8}$$

If three directions offset between the extreme point (x, y, σ) and the candidate keypoints are all less than 0.5, the interpolation ends; Otherwise, adjusting the position of the keypoint interpolation continues until the iteration times are greater than the 5 or three offsets are all less than 0.5. If interpolation is successful, the location of keypoint is accurate than before in scale space. If the DoH value of a sample point is smaller than a certain threshold, the keypoint is removed.

3.5 Computing the Orientation

Similar to SIFT algorithm, SURF feature also needs orientation for ensuring rotation invariance, while computing the orientation of SURF is based on Haar wavelet. Haar wavelet computes the gradient for image area, which enjoys very good resistance to noise.

In a circle using keypoint as the center and 6s (s presents the scale of keypoint) as radius, calculate Haar wavelet with the size of 4s, denoted as haarX, haarY; and multiply the weight w (as in formula 9) with factor 2s, denoted as WhaarX, WhaarY, making the sample point's haar wavelets that close to keypoint have great contribution to orientation. Similarly, the sample point far from keypoint has little contribution to orientation. The direction of the keypoint is the arc tangent of WhaarX and WhaarY.

$$w = \frac{1}{2 * \pi * \sigma * \sigma} e^{-\frac{x*x+y*y}{2*\sigma*\sigma}} \tag{9}$$

First, in the above circle that uses keypoint as the center and 6s as radius, the direction of sample point within $0 \sim \frac{\pi}{3}$, Haar wavelet is weighted and summed, donated as $\sum WhaarX$, $\sum WhaarY$. Then, making fan-shaped area rotate 0.15 radians, or, the direction of sample point being ranged within $0.15 \sim 0.15 + \frac{\pi}{3}$, its weighted haar wavelets are accumulated. Repeating the above process, when the fan-shaped area around the keypoint is in one circle, select the largest group accumulated sum $(\sum WhaarX)^2 + (\sum WhaarY)^2$, and the arc tangent between $\sum WhaarX$ and $\sum WhaarY$ is the orientation of keypoint.

3.6 Computing Keypoints'descriptors

Similar to the calculation of the keypoint, keypoint's descriptors need to calculate haar wavelets which use the keypoint as the region center. Compute keypoint's descriptor as follows:

First, a 20s×20s region with the keypoint as the center is taken, which is divided into 16 sub-regions with a size of 5s × 5s, and coordinate axes is rotated counterclockwise to orientation of keypoint. Then, haar wavelets relative to orientation are calculated at each sample point in each sub-region, and multiply Gaussian weight with the factor 2.5s, denoted by dx, dy; the dx, dy are accumulated to four components $\sum dx$, $\sum dy$, $\sum |dx|$, $\sum |dy|$. When the sampling points of the entire sub-regions are traversed, four components are multiplied by Gaussian weight relative to 4 × 4 sub-regions. For the other sub-regions, similar calculation is done; therefore, each keypoint can form 16*4 = 64 dimensions descriptors. Finally, it was normalized to remove its sensitivity to light.

After keypoints' extraction is completed, the traversing method is used to find matching point, that is, each keypoint of the real-time image finds the keypoints that minimum Euclidean distance or near minimum Euclidean distance in the reference image; If the ratio of two distance is less than 0.7, then the keypoint in real-time image and keypoint in reference image are taken for matching points.

4 Parallel SURF Algorithm

4.1 Parallel Extracting Keypoints

YHFT-QDSP consists of four DSP cores; and each core has independent storage space. The parallel algorithm based on YHFT-QDSP is mainly divided into two ways: task-level parallelism and data-level parallelism. According to characteristics of application, the corresponding parallel way can be adopted. Task-level parallelism is favored if the process of the algorithm is partitioned according to the consuming time of each sub-process, so the entire program is implemented in pipeline way. On the other hand, Data-level parallelism is suitable if the processed data are evenly distributed into four DSP cores, so data parallel processing is implemented.

It can be seen in section 3 that SURF algorithm has high computational complexity. Since all sub-processes of the algorithm are closely connected, even splitting the algorithm is difficult. Meanwhile, for the task-level parallelism of SURF algorithm, the intermediate result needed to be transferred to other core is large, which will hinder the speedup improvement. Therefore, we use the method of data-level parallelism to segment the integral image for parallel extraction of keypoints. Due to the extracted keypoints of SURF are the local features of image, the method of segmenting the integral image for extracting keypoints is feasible.

As shown in Figure 3, the basic idea of SURF algorithm for data-level parallelism is: First, all DSP cores calculate integral image for the loaded image; then, each DSP calculates DoH only on a quarter of the integral image: establishing scale space, detecting the maximum value points, generating descriptors. The calculation of coordinates of keypoints on four DSP cores are handled differently, that is, no special handling of keypoints on the DSP0 is needed; Keypoints' abscissa on the DSP1 must add the width of scale space where keypoint is located; Keypoints' ordinate on the DSP2 must add the height of scale space where keypoint is located; Keypoints' abscissa and ordinate on DSP3 must respectively add the width and height of scale space where keypoint is located. Therefore, SURF has well parallelism and is suitable for YHFT-QDSP multi-core architecture.

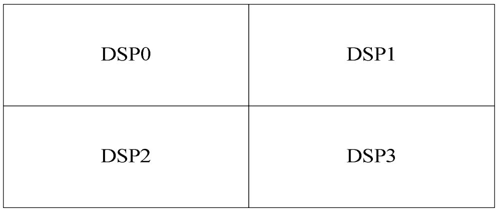


Fig. 3. Parallel extracting keypoints based on segmenting image

When parallel SURF algorithm is implemented on YHFT-QDSP, two images are loaded to storage space of each DSP core, which are called reference image and real-time image respectively. Each DSP core reads the corresponding part of the image according to its ID (the range is 0, 1, 2, 3), computes the integral image and extracts keypoints independently. When keypoints are extracted, the four DSP have a set of keypoint in reference image, denoted by A0, A1, A2 and A3, and a collection of keypoints in real-time image, denoted by B0, B1, B2 and B3, which are stored in some designated memory space in DSP core for the subsequent matching.

4.2 Parallel Matching Keypoints

In order to ensure the accuracy of image matching, each keypoint in real-time image searches its matching point in the whole keypoints set in reference image. Assuming that the number of real-time image keypoint set B0, B1, B2, B3 is n0, n1, n2, n3, respectively, and the total number is n; the number of reference image keypoint set A0, A1, A2, A3 is m0, m1, m2, m3, respectively, the total number is m, the total number of keypoint matching can be calculated as in formula 10:

$$\begin{aligned}
Total &= \sum_{i=1}^{n}\sum_{j=1}^{m} b_i a_j = \sum_{i=1}^{n0}\sum_{j=1}^{m} b_i a_j + \sum_{i=1}^{n1}\sum_{j=1}^{m} b_i a_j + \sum_{i=1}^{n2}\sum_{j=1}^{m} b_i a_j + \sum_{i=1}^{n3}\sum_{j=1}^{m} b_i a_j \\
&= (\sum_{i=1}^{n0}\sum_{j=1}^{m0} b_i a_j + \sum_{i=1}^{n0}\sum_{j=1}^{m1} b_i a_j + \sum_{i=1}^{n0}\sum_{j=1}^{m2} b_i a_j + \sum_{i=1}^{n0}\sum_{j=1}^{m3} b_i a_j) + (\sum_{i=1}^{n1}\sum_{j=1}^{m0} b_i a_j + \sum_{i=1}^{n1}\sum_{j=1}^{m1} b_i a_j \\
&+ \sum_{i=1}^{n1}\sum_{j=1}^{m2} b_i a_j + \sum_{i=1}^{n1}\sum_{j=1}^{m3} b_i a_j) + (\sum_{i=1}^{n2}\sum_{j=1}^{m0} b_i a_j + \sum_{i=1}^{n2}\sum_{j=1}^{m1} b_i a_j + \sum_{i=1}^{n2}\sum_{j=1}^{m2} b_i a_j + \sum_{i=1}^{n2}\sum_{j=1}^{m3} b_i a_j) \\
&+ (\sum_{i=1}^{n3}\sum_{j=1}^{m0} b_i a_j + \sum_{i=1}^{n3}\sum_{j=1}^{m1} b_i a_j + \sum_{i=1}^{n3}\sum_{j=1}^{m2} b_i a_j + \sum_{i=1}^{n3}\sum_{j=1}^{m3} b_i a_j)
\end{aligned}$$

As can be seen from the above formula, the matching keypoints can be distributed to four DSP cores for parallel processing. According characteristics of YHFT-QDSP architecture, we designed the parallel keypoint matching based on data-level method, that is, four DSP cores act as the master node in turn, and each DSP core implements parallel computing tasks and returns results to the master DSP core, as in Figure 4. It utilizes QLink and SDP respectively to deal with data communication and synchronization among DSP cores, where QLink is suitable for bulk data transfer and SDP is suitable for little fast data transmission and data synchronization. The multi-level parallelism of YHFT-QDSP is fully developed and hence image matching speed is improved.

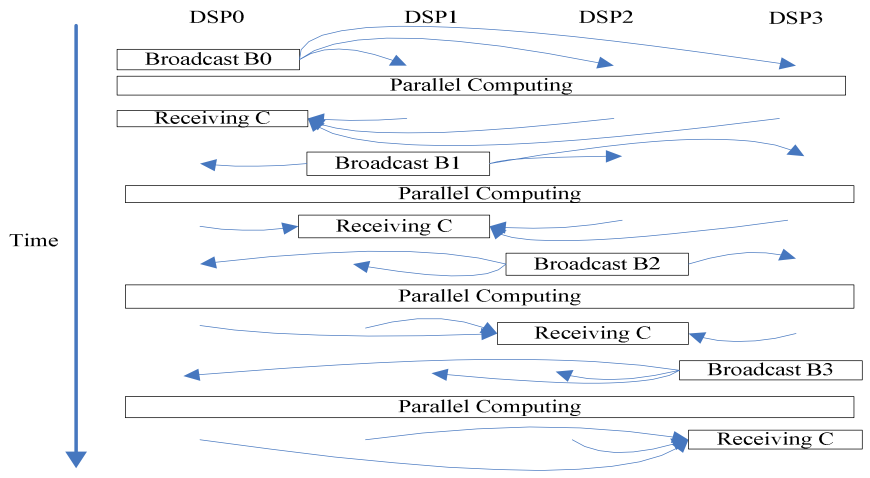


Fig. 4. Parallel keypoints matching based on YHFT-QDSP

The following is the process flow for DSP0 as the master node, which illuminates details of multi-core parallel keypoints matching:

(1)The descriptors set B0 is sent to DSP1, DSP2, DSP3 by DSP0 through QLink; when transmission is completed, the number of descriptors B0 is sent to the DSP1, DSP2, DSP3 by the SDP, meanwhile light signal to the three DSP cores to indicate data transmission is completed.

(2)The smallest Euclidean distance and the second smallest Euclidean distance of each keypoint of B0 are found in descriptors set A0, which are stored in C0. When DSP1, DPS2 and DSP3 detected signals sent by DSP0, read data and close signals; Then, DSP1, DSP2, DSP3 search the smallest Euclidean distance and the second smallest Euclidean distance of B0' keypoint in descriptors set A1, A2, A3, respectively. When calculation is completed, the results of three DSP cores are sent to DSP0 through the QLink, which are stored in the C1, C2, C3 in DSP0, respectively, and turn on the signals for DSP0.

(3)When DSP0 detects signal, read data and close signal; for each keypoint b in descriptors set B0, it has 8 keypoints' information that DSP cores have computed (the coordinates of keypoint and the distance between the keypoint and keypoint b).

Choosing the smallest distance and the second smallest distance, if the ratio less than 0.7, the two points are considered as the matching points; otherwise, they don't match. When DSP0 has processed all the keypoints completely, turn on signal to DSP1.

When DSP1, DSP2, DSP3 are acted as the master node, similar steps (1), (2), (3) are repeated. All the calculations are done; matching points of reference image and real-time image are obtained. Finally, mismatch points are removed through RANSAC algorithm.

5 Performance Evaluation and Analysis

The main criteria for evaluating the parallel image matching algorithm are the speedup and repetition rate. Speedup is the ratio of time consumed when the same task runs in the single-processor systems and in the multi-processor system, which is used to evaluate performance of parallel system result of parallel algorithm. In this paper, the speedup = T_{DSP} / $T_{YHFT\text{-}QDSP}$, where T_{DSP} is the time that SURF serial algorithm consumed in the single core DSP, $T_{YHFT\text{-}QDSP}$ is the time that parallel SURF algorithm consumed in the multi-core YHFT-QDSP. The repetition rate is the ratio between the matching points of two images and the average number of keypoints that two images extracted, which is used to measure the ability of image matching algorithms. repeatability $K = C(I_1, I_2) / N$, where $C(I_1, I_2)$ presents the number of matching points of two images, N refers to the average number of keypoints that the two images have extracted. The repetition rate = K_{DSP} / $K_{YHFT\text{-}QDSP}$, where K_{DSP} is the repetition of the serial SURF algorithm on single-core DSP, $K_{YHFT\text{-}QDSP}$ is the repetition of the parallel SURF algorithm on YHFT-QDSP. The repetition rate reflects the matching ability of the parallel algorithm in comparison with the serial algorithm. The closer the repetition rate is to 1, the closer the image matching ability of parallel algorithms is to sequential algorithm.

In this section, 5 deformation image samples with scale change, rotation, illumination changes in light and dark, noise and affine transformation are used to evaluate and test the designed parallel algorithm. We analyzed and evaluated the parallel SIFT algorithm according to the repetition rate and speedup. The image sample size is 360x144 and the format is pgm in the experiment.

Figure 5 (a) and (b) compare the image matching results of the serial and parallel SURF algorithm for scale change image. Figure 5 (c) and (d) compare the image matching results of serial and parallel SURF for rotation image. Figure 5 (e) and (f) compare the image matching results of serial and parallel SURF algorithm for illumination image. Figure 5 (g) and (h) compare the image matching results of serial and parallel SURF algorithm for noise image. Figure 5(i) and (j) compare the image matching results of serial and parallel SURF algorithm for affine transformation image.

Table 1 summarizes the matching performance results of 5 different deformation images. As can be seen from Table 1, the average speedup of parallel SURF algorithm is 3.61, where the noise image gets the maximum speedup, achieving remarkable acceleration. The average value of the repetition rate is 92%, which indicates that the image matching ability between parallel SURF algorithm and serial SURF algorithm is close, especially for the affine transform, illumination change and rotation image. It can be seen from Figure 5 that SURF parallel image matching algorithm works well in meeting the matching requirements.

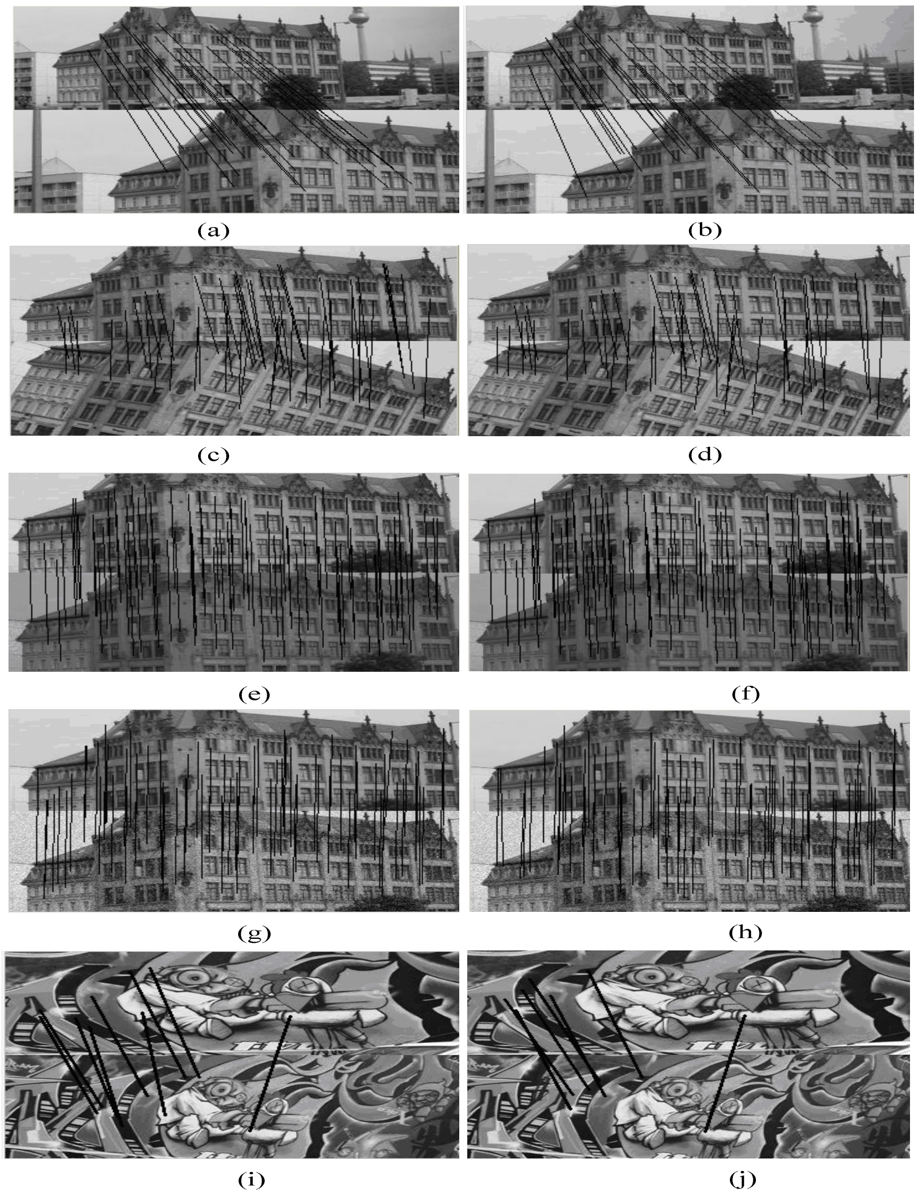


Fig. 5. Compare the image matching results of the serial and parallel SURF algorithm

Table 1. Matching performance result of different deformation images

Parameters \image type	scale change	rotation	illumination change	noise	affine transformation	average
speedup	3.56	3.65	3.57	3.69	3.57	3.61
repetition rate	87%	88%	97.8%	87.8%	100%	92%

6 Conclusions

This paper presents an efficient parallel SURF algorithm for multi-core processors, and data-level parallel algorithm for keypoint extracting and keypoint matching are designed. Based on the characteristics of YHFT-QDSP architecture, various computing tasks are processed in parallel in multiple DSP core. And Qlink is used in combination with SDP to deal with data communication among DSP cores, Keypoints are transferred by QLink when it is suitable for bulk data transmission, and the number of keypoints and data synchronization signals are transferred by SDP when it is suitable for light and fast data transmission. The method we have presented can fully develop multi-level parallelism of YHFT-QDSP architecture, and accelerate the speed of parallel SURF algorithm. The parallel SURF algorithm is fully tested using 5 different image samples for scale change, rotation, change in illumination, addition of noise and affine transformation. Experimental results show that the parallel SURF algorithm has good adaptability for various kinds of deformation images, and the matching capability of parallel SURF algorithm is close to the serial SURF algorithm with the average speedup of 3.61.

References

[1] Todorovic, S., Ahuja, N.: Scale-invariant Region-based Hierarchical Image Matching. In: Proc. 19th International Conference on Pattern Recognition (ICPR), Tampa, FL (December 2008)

[2] Toews, M., Wells III, W.M., Louis Collins, D., Arbel, T.: Feature-based Morphometry: Discovering Group-related Anatomical Patterns. NeuroImage 49(3), 2318–2327 (2010)

[3] Lowe, D.G.: Distinctive image features from Scale-Invariant Keypoints. International Journal of Computer Vision 60(2), 91–110 (2004)

[4] Bay, H., Tuytelaars, T., van Gool, L.: Speeded-up Robust Features (SURF). Computer Vision and Image Understanding (2007)

[5] Chen, S.M., Wan, J.H., Lu, J.Z., et al.: YHFT-QDSP: High-performance heterogeneous multi-core DSP. Journal of Computer Science and Technology 25(2), 214–224 (2010)

[6] Viola, P., Jones, M.: Rapid object detection using a boosted cascade of simple features. In: IEEE Conference on Computer Vision and Pattern Recognition, pp. 511–518 (2001)

[7] Simard, P., Bottou, L., Haffner, P.: Boxlets: a fast convolution algorithm for signal processing and neural networks. In: Advances in Neural Information Processing Systems (1999)

A Study of Cache Design in Stream Processor

Chiyuan Ma and Zhenyu Zhao

School of Computer, National University of Defense Technology,
Changsha, Hunan, 410073, China
{cyma,zyzhao}@nudt.edu.cn

Abstract. Stream architecture is a newly developed high performance processor architecture oriented to multimedia processing. FT64 is 64-bit programmable stream processor and it aims at exploiting the parallelism and locality of the applications. In this paper, first, we inspect the memory access characteristics of FT64 with cache and without cache. Second, we propose an improved cache design method. Making use of the feature of stream data type used by FT64, the improved method avoids loading data from memory when the stream store instruction fully modifying cache block misses. The experiments show the performance has been improved by 20.7% and 25.8% when a normal cache and an improved cache are used respectively. Finally, we study on the performance influence of cache capacity and associativity. The results show that better performance can be achieved when we use a small cache and an associativity of 2 or 4.

Keywords: stream processor, FT64, cache, memory access, fully modify.

1 Introduction

At present, stream application is becoming the main workload of processor, whereas the architecture of general purpose processors cannot satisfy the current demand. Most of the chip area has been used to conceal memory access latency, which results in deficiency of computing ability. In recent years, many stream architectures aiming at stream application have been proposed [1-6], and they have obvious advantages in domains of signal processing, multimedia and scientific computing. The main idea of stream processing is organizing the stream to data streams and computation kernels, utilizing the locality and parallelism of the application to conceal memory access latency. The key of stream architecture is to provide powerful computing ability and hide the memory access latency at the same time.

In order to conceal memory access latency, different stream processors use different methods. For example, VIRAM[7] puts DRAM into the chip, Imagine[1-2] and Merrimac[3] adopt multilevel memory structure. We design a 64-bit programmable stream processor FT64(Fei Teng 64)[8], which aims at exploiting the parallelism and locality of the applications in domains of signal processing, multimedia and scientific computing. FT64 inherits some characteristics of other stream processors, such as stream programming model and multilevel memory structure. FT64 adopts multi-level memory structure including inter-cluster local register file (LRF), four clusters shared stream register file (SRF), cache and off-chip memory to conceal memory access latency.

Memory access of typical stream applications has poor temporal locality. The same set of data is generally loaded from memory only once and sent to the computation unit for multiple times of process, then the output data are produced and stored in the memory, the input and output data will generally not be used again. Many papers have studied the cache behavior when the processor runs the stream application. In FT64, there are two stream buffers connected with the cache, what we are interested in is the cache's behavior under this condition and how to make use of the feature of stream programming model in the cache design.

To perform our evaluation, we collect results from several programs including two SPEC CPU2000 benchmarks and six important multimedia and scientific computing programs. These programs are reprogrammed with stream model to suit for FT64. The results are collected with a cycle-accurate simulator of FT64.

In the experiment, first, we inspect the characteristic of FT64's memory access without cache, the results show that the memory access latency cannot be concealed totally, and memory access is the bottleneck. Second, we study the performance when using a normal cache. Because huge speed gap exists between cache and off-chip memory, and every valid cache block can be hit several times, the advantage of using cache is apparent. We also notice that most stream store instructions write data to the continuous addresses. When a stream store instruction misses in the cache, it is meaningless to load the data from memory if the corresponding cache block will be fully modified by the store instruction, we can make use of the feature that FT64 supports stream data type, and store the data straight into the cache block when such stream store instruction appears. This improved cache design can further decrease the memory access latency. The results show the performance has been improved by 20.7% and 25.8% when a normal cache and an improved cache are used respectively. Finally, we study on the performance influence of cache capacity and associativity in the improved cache design. The results show that better performance can be achieved when a small cache is used and the associativity is 2 or 4.

2 Background

Our research is based on a 64-bit programmable stream processor FT64, which we design for exploiting the parallelism and locality of the applications in domains of signal processing, multimedia and scientific computing. FT64 supports stream programming model. In this model, an application is represented by a set of computation kernels which consume and produce data streams. Each data stream is a sequence of data records of the same type. Each kernel is a program which performs the same set of operations on each input stream element and produces one or more output streams. Stream applications consist of stream-level programs and kernel-level programs. A stream-level program specifies the execution order of all kernels and organizes data into sequential streams passed from one kernel to the next. A kernel-level program is generally a loop structure that processes record elements from each input stream and generates output streams.

FT64 is mainly composed of a stream controller (SC), a stream register file (SRF), a micro controller (UC), four ALU clusters, a stream cache (SC), a DDR memory controller (DDRMC), a host interface (HI) and a network interface (NI), as illustrated in Figure 1.

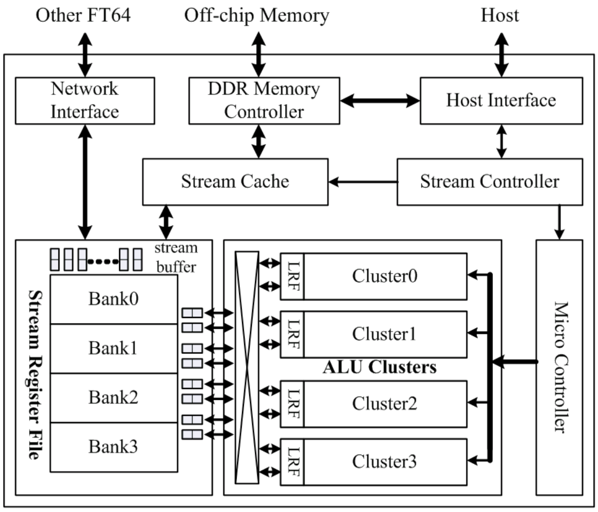


Fig. 1. Block diagram of FT64

FT64 has multi-level memory: LRF, SRF, cache and off-chip memory. LRF is distributed in the clusters' FU, and is used to buffer input and intermediate data. LRF of different ALU clusters can access SRF simultaneously through a crossbar. The total capacity of LRF is 19KB. SRF is used to buffer data streams and kernel-level programs. SRF has two ports, and both can access cache through a separate stream buffer. The total capacity of SRF is 256KB. The off-chip memory is controlled by DDRMC, which is in charge of loading and storing data streams.

3 Our Work

In FT64, the cache is between SRF and memory, it has two pipelines which are connected with the two ports of SRF, and each port has a stream buffer, so the cache can handle two accesses simultaneously. What we are interested in is the cache behavior. The experiment shows the performance has been improved by 20.7% when a normal cache is used.

The cache interfaced with off-chip memory usually adopts write allocate policy. When cache misses, the data will be loaded from memory to cache before load or store instruction operates. But for a processor that supports stream data type, if a stream store instruction writes data to the continuous addresses, and the data size is much larger than the size of cache block, then multiple cache blocks will be fully modified, it is meaningless to load data from memory to these cache blocks when the store instruction misses, we only need to load data to partial modified cache block. Next, we will explain how to realize this idea in FT64.

What is executed in SRF is stream instruction. Stream instruction supports stream data type which is a sequence of data records of the same type. The format of stream memory instruction is shown as Table 1.

Table 1. The format of stream memory instruction

valid bit	operating mode	start address	stream length	stride

The operating mode indicates what operation, load or store, the stream instruction will perform. Start address is the beginning address of the stream data. Stream length is the byte number of the stream data. Stride indicates the organization mode of the stream data, maybe it is an address continuous stream or an address jumped stream.

A stream instruction must access continuous addresses if its stride value is 1. If it is a stream store instruction, we can split the instruction into three parts at most: head, middle and tail. The head only exists when the beginning address of the store instruction doesn't align with that of some cache line. The tail only exists when the ending address of the store instruction doesn't align with that of some cache line. The middle handles the entire cache line and it only exists when at least one cache line will be fully modified by the store instruction. When a store instruction misses in the cache, we apply normal write allocate policy to head and tail parts, and store the stream data straight into corresponding cache lines for the middle parts. Certainly, the replaced cache lines shall be written back in advance if it is dirty. The algorithm is showed in Fig.2.

```
if ( instruction==store && stride==1 )
{           if ( beginning_address_not_align_with_cache_line )
            {           split_head_block;
                        store_head_block_with_write_allocate_policy;
            }
            if ( endding_address_not_align_with_cache_line )
            {           split_tail_block;
                        store_tail_block_with_write_allocate_policy;
            }
            if ( middle_block_number >= 1 )
            {           if ( coresponding_cache_line_is_dirty )
                        {           write_back_dirty_data;}
                        store_middle_block_directly;
            }
}
else
{           adopt_normal_cache_operation_policy;}
```

Fig. 2. Algorithm of stream store operation

The improved cache design described above is appropriate only when the stride value of the stream store instruction is 1, instructions with other stride values must be handled with normal cache operation. But we find that stride value equaling 1 is overwhelming for most of the programmers using stream model. This is profitable for our method to get better results.

The idea of the improved cache design is enlightened by the idea proposed by Shiwen Hu [9]. He adds a store fill buffer (SFB) in general purpose processor. If a store instruction misses, the data are put straight into the SFB. If some cache block is affirmed to be fully modified, it will be filled into cache from SFB. Supporting stream data type makes FT64 more feasible to realize the function than [9]. A stream store instruction of FT64 includes multiple successive store operations, and then we can make judgment that which cache line will be fully modified and prepare all necessary operations in advance. The hardware cost of our method is trivial.

In next section, we will compare the performances of using normal cache design and improved cache design, and study on the performance influence of cache capacity and associativity.

4 Performance Evaluations

4.1 Experimental Environment

To perform our evaluation, we collect results from several programs including two SPEC CPU2000 benchmarks (Swim and Lucas), two NPB benchmarks (EP and MG), a multimedia program (Mpeg2) and three important scientific application kernels (FFT, Laplace and NLAG-5). (NLAG-5 is a nonlinear algebra solver of two-dimensional nonlinear diffusion of hydrodynamics). These programs are reprogrammed with stream model to suit for FT64. The results are collected with a cycle-accurate simulator of FT64. The data sizes of some programs are too small, so we change their data sets and execute the programs several times to get the results.

4.2 Experimental Results

The overlap between computation and memory access is an important factor that impacts on a stream processor's performance. Fig.3 demonstrates the proportions of the kernel execution time and memory access time to the total execution time when cache is bypassed. The kernel execution and memory access of FT64 can execute simultaneously, but under this condition, memory access is still the bottleneck in most of the programs. For memory-intensive programs, such as Swim and NLAG-5, computation can be well hidden by memory access, thus memory access time is approximately the total execution time. For compute-intensive programs, such as EP, with huge computations and small memory accesses, memory access can be well hidden by computation, so kernel execution time is approximately the total execution time. To some programs with small proportions of memory access and kernel execution, most of the time is consumed in SRF allocation and memory access preparation.

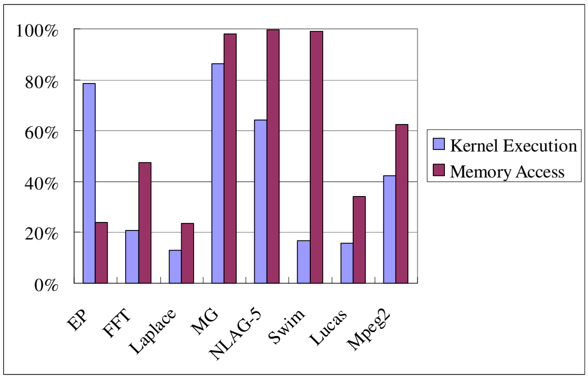


Fig. 3. Proportions of kernel execution time and memory access time

Fig.4 shows the proportions of access latency of LRF, SRF and memory when cache is bypassed. In average, the proportion is 23:2.4:1. The programs with large computations and good data locality can fully make use of LRF and have high proportions, such as MG, the proportion is 60:5:1, and EP, 33:2:1. Some programs, such as Laplace, NLAG-5 and Lucas, have low proportions of 12:2:1, 11:3:1 and 11:2:1 respectively, because their executions are limited by memory access. The access latency of LRF is in average 9.4 times that of SRF, and the access latency of SRF is just 2.4 times that of memory in average. The gap between SRF access and DRAM access is quite narrow, which can't make the memory access concealed in most of the cases, and leads the memory to be the bottleneck.

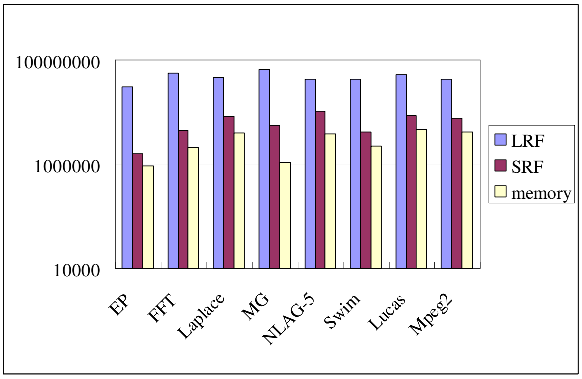


Fig. 4. Proportions of access latency of LRF, SRF and memory

Next, we will study the case in which cache is used. To what degree we can benefit from the improved method described previously is related to two factors, the proportion of store instructions to total memory instructions and the proportion of fully modified cache lines to all the cache lines modified by a store instruction. Fig.5 shows the distribution of memory access types. The proportion of the store instructions to all memory instructions is nearly 26% in average, Swim's is the biggest, 50.3%. Almost all the store related cache lines can be fully modified.

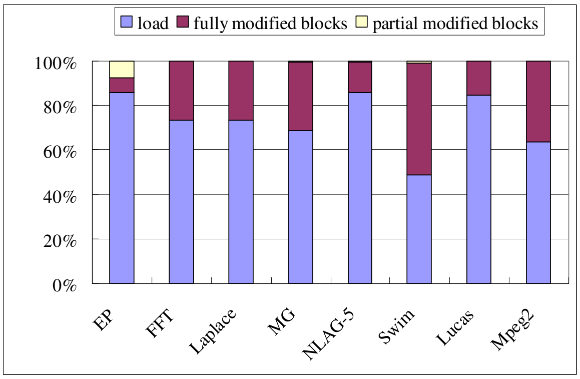


Fig. 5. Distribution of memory access types

Fig.6 shows the performance improvement by using a normal cache and an improved cache of 16K bytes respectively, in comparison with the performance of not using cache. (Not using cache doesn't mean that only one word will be loaded to SRF once memory is accessed, instead, a 64-byte buffer is used to buffer the loaded data temporally, this buffer can be thought as a 64 bytes, one set, direct mapped cache.) The cache's block size is 64 bytes and cache's associativity is 4. The performance of each program is related to its memory access proportion and the distribution of its memory access types. Among all the programs, Swim has the largest quantity of memory accesses, and the proportion of store instructions is very high, so it has the biggest performance improvement, up to 44% and 67%. Whereas EP, with small quantity of memory accesses, its performance improvement is no more than 1% after a cache is used. In average, the performance improvement is 20.7% and 25.8% respectively.

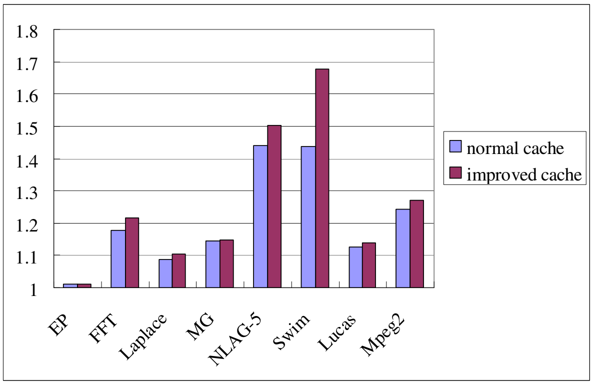


Fig. 6. Performance improvement by using a normal cache and an improved cache of 16K bytes

Now we can see the proportion variation of kernel execution and memory access after we use a 16K bytes improved cache in Fig.7. Convenient for comparison, the memory access latency in Fig.7 includes the cache and memory access latency. The proportion of kernel execution increases a little because of decrease of total execution time caused by the use of cache. Except Swim, the memory access latency of almost all the programs can be concealed.

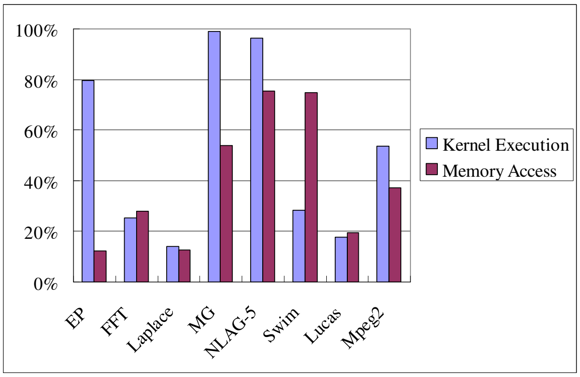


Fig. 7. Proportion of kernel execution time and memory access time when using an improved cache

Fig.8 shows the proportions of access latency of LRF, SRF, cache and memory after a 16K bytes improved cache has been used. In average, this proportion is 105:11.6:2.5:1. Compared with Fig.4, memory access latency has been shortened greatly. The hit rate of the cache is 89% in average. Since stream programs have poor data locality during memory access, this hit rate is comparatively low when compared with that of general purpose processor. The latency of cache hit is only one quarter of that of memory access, which makes the whole latency shorter.

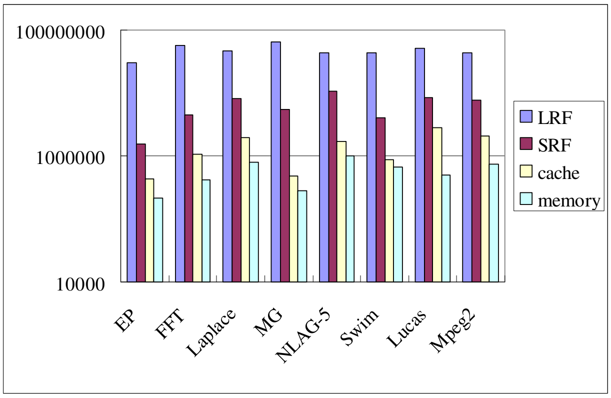


Fig. 8. Proportions of access latency of LRF, SRF, cache and memory

Fig.9 shows the performance influence of cache capacity. (In Fig.9, we suppose that the execution time of each program is 1 when cache capacity is 2K bytes, then we can compare the execution time with other capacities). Here, cache associativity is set to 4. The result shows that a cache of 8K or 16K bytes can attain the best performance for most of the stream programs. Since the data loaded into cache will seldom be used after initial continuous accesses, and the data stored to cache will scarcely be loaded again (data transferred between kernels is handled in SRF), a small cache in FT64 can achieve required result.

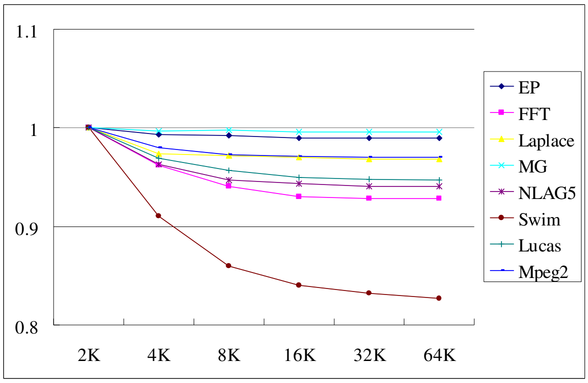


Fig. 9. Performance influence of cache capacity

Fig.10 shows the performance influence of cache associativity used in 16K bytes improved cache design. (We suppose that the execution time of each program is 1 when cache associativity is 1, then we can compare the execution time with other associativities). Associativity of 2 or 4 has the best results. It is because the stream number that most programs load or store simultaneously will not surpass 4, in most cases it is 1 or 2. The results get worst when cache associativity is increased to 8 or 16, because higher associativity will decrease the set number for a fixed cache capacity, and enlarge the conflict miss rate.

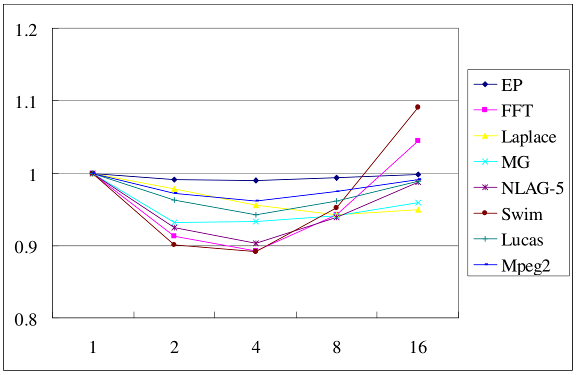


Fig. 10. Performance influence of cache associativity

5 Related Work

Stream models are studied and applied in domains of graphics, multimedia and signal processing, where many architectures and processors supporting stream models have emerged, such as AIFSP [4], SAT [6], Imagine[1], RAW [5], VIRAM [7] and TRIPS [10]. In some researches, stream models have been applied to scientific computing, such as Merrimac [3]. In addition, Cell [11] also supports stream models and is claimed to have tremendous computing ability.

Some novel ideas of the stream memory system are proposed during the researches of stream architectures. The VIRAM[7] architecture uses PIM technology to combine embedded DRAM with a vector co-processor for exploiting its large bandwidth potential. The Imagine[1-2] architecture provides a stream-aware memory hierarchy to support the tremendous processing potential of SIMD controlling VLIW clusters.

At the same time, there are many researches on stream applications in general purpose processor. Sermulins proposes a cache aware optimization method of stream programs [12]. Lee proposes a hardware prefetching technique that is assisted by static analysis of data access pattern with stream caches for multimedia applications[13]. Iacobovici analyzes the behavior of cache-missing loads in SPEC CPU2000 and proposes a multi-stride prefetcher that supports streams with up to four distinct strides[14].

6 Conclusion

In this paper, first, we inspect the memory access characteristics of FT64 with cache and without cache. Second, we propose an improved cache design method. Making use of the feature of stream data type used by FT64, the improved method avoids loading data from memory when the stream store instruction fully modifying cache block misses. The experiments show the performance has been improved by 20.7% and 25.8% when a normal cache and an improved cache are used respectively. Finally, we study on the performance influence of cache capacity and associativity. The results show that better performance can be achieved when we use a small cache and an associativity of 2 or 4.

In our research, we only focus on decreasing memory access through optimizing cache design, while other methods can obtain similar results, such as scheduling data streams during stream programming and stream-level compiling, which can be left for further researches.

Acknowledgements. Financial supports from the Natural Science Foundation of China (No.61076025), Hegaoji project of China (No.2009ZX01028-002-002) and project of China (KJ-11-04) are greatly acknowledged.

References

1. Kapasi, U., Dally, W.J., Rixner, S., Owens, J.D., Khailany, B.: The Imagine Stream Processor. In: ICCD 2002: Proceedings of 20th IEEE International Conference on Computer Design, pp. 282–288 (2002)
2. Kapasi, U.J., Rixner, S., Dally, W.J., Khailany, B., Ahn, J.H., Mattson, P., Owens, J.D.: Programmable Stream Processors. IEEE Computer 36(8), 54–62 (2003)
3. Dally, W.J., Hanrahan, P., Erez, M., Knight, T.J.: Merrimac: Supercomputing with Streams. In: SC 2003 (November 2003)
4. Wang, Y., Chen, S., Wan, J., Zhang, K., Chen, S.: AIFSP: An Adaptive Instruction Flow Stream Processor. IEEE Transactions on VLSI (2011)
5. Taylor, M., Kim, J., Miller, J., Wentzlaff, D., et al.: The RAW Microprocessor: A Computational Fabric for Software Circuits and General Purpose Programs. IEEE Micro 22(2), 25–35 (2002)
6. Yang, Q., Wu, N., Wen, M., He, Y., Su, H., Zhang, C.: SAT: A Stream Architecture Template for Embedded Applications. In: 10th IEEE International Conference on Computer and Information Technology (2010)
7. Kozyrakis, C.: Scalable Vector Media-processors for Embedded Systems. PhD thesis, University of California at Berkeley (2002)
8. Yang, X.J., Yan, X.B., Xing, Z.C., et al.: A 64-bit Stream Processor Architecture for Scientific Applications. In: ISCA 2007 (2007)
9. Hu, S., John, L.: Avoiding Store Misses to Fully Modified Cache Blocks. In: Proceedings of Performance, Computing, and Communications Conference (2006)
10. Burger, D., Keckler, S.W., McKinley, K.S., Dahlin, M., John, L.K., Lin, C., Moore, C.R., Burrill, J., McDonald, R.G., Yoder, W.: Scaling to the End of Silicon with EDGE Architectures. Computer 37(7), 44–55 (2004)
11. Pham, D., Asano, S., Bolliger, M., Day, M.N., Hofstee, H.P., Johns, C., Kahle, J., Kameyama, A., Keaty, J., Masubuchi, Y., Riley, M., Shippy, D., Stasiak, D., Suzuoki, M., Wang, M., Warnock, J., Weitzel, S., Wendel, D., Yamazaki, T., Yazawa, K.: The Design and Implementation of a First-Generation Cell Processor. In: ISSCC 2005, pp. 184–185 (2005)
12. Sermulins, J., Thies, W., Rabbah, R., Amarasinghe, S.: Cache Aware Optimization of Stream Programs. In: Proceedings of the 2005 ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems, vol. 40, pp. 115–126 (2005)
13. Lee, J., Park, C., Ha, S.: Memory Access Pattern Analysis and Stream Cache Design for Multimedia Applications. In: Proceedings of the 2003 Conference on Asia South Pacific Design Automation, pp. 22–27 (2003)
14. Iacobovici, S., Spracklen, L., Kadambi, S., Chou, Y., Abraham, S.G.: Effective Stream-Based and Execution-Based Data Prefetching. In: Proceedings of the 18th Annual International Conference on Supercomputing, pp. 1–11 (2004)

Design and Implementation of Dynamically Reconfigurable Token Coherence Protocol for Many-Core Processor

Chuan Zhou, Yuzhuo Fu, Jiang Jiang, Xing Han, and Kaikai Yang

School of Microelectronics,
Shanghai Jiao Tong University, Shanghai
{zhouchuan,fuyuzhuo,jiangjiang,
hanxing,yangkaikai}@ic.sjtu.edu.cn

Abstract. To efficiently maintain cache coherence in a many-core processor remains a big challenge today. Traditional protocols either offer low cache miss latency (like snoopy protocol) or not depending on bus-like interconnects (like directory protocol). Recently, Token Coherence has been proposed to capture the main characteristic of traditional protocols. However, since Token Coherence relies on broadcast-based transient request and inefficient persistent request, it is only suitable for small system. In order to make Token Coherence be scalable in many-core architectures, in this paper we introduce a dynamically reconfigurable mechanism to Token Coherence. Basing on sub-net, this mechanism can significantly reduce the average execution time and communication cost in 16-core processor. Therefore, this dynamically reconfigurable mechanism makes Token Coherence applicable in many-core architecture.

Keywords: cache coherence, token coherence, dynamically reconfigurable, sub-netting.

1 Introduction

Today, Chip multiprocessors have become a promising choice for keeping up performance with integration density [1, 2, 3]. The number of cores integrated on a chip has reached a hundred now, and in the near future, there will probably be thousands of cores. Therefore, the era of many-core processor is in close proximity.

To keep the view of memory coherent across different nodes, a many-core processor requires a cache coherence protocol which plays an important role on system performance. However, thanks to the large number of cores and the complexity of on-chip network, to efficiently support cache coherence in a many-core processor suffers from the problem of "coherency wall" [4]. Considering cache miss latency, network traffic and area overhead as the key attributes of cache coherence protocol [5], many-core processors need a protocol which has low cache miss latency and low network traffic. Traditional protocols either offer low cache miss latency (like snoopy protocol) or not depending on bus-like interconnects (like directory protocol). Snoopy protocol offers low cache miss latency due to direct communication between different cores, but it needs ordered network and the network traffic based on broadcast is also tremendous; Directory protocol does not rely on an ordered network because of the indirect node (directory) which otherwise increases the cache miss latency on average.

In order to gain the main characteristics of traditional protocols, recently, Token Coherence [6] has been proposed. Compared with snoopy protocol, Token Coherence does not rely on a totally ordered network which is not possible in many-core architecture. Meanwhile, Token Coherence avoids the indirect node (directory) in directory protocol, and therefore gain low cache miss latency. However, Token Coherence has some defects as well. First, the requests transmitted in the network are usually based on broadcast; As a result, the network traffic will be unacceptable if we introduce Token Coherence to many-core processors. Second, the persistent request mechanism used to resolve protocol races is inefficient and broadcast-based. The broadcast mechanism and persistent request make Token Coherence only suitable for small system. So, for the purpose of adapting Token Coherence to many-core processors, in this work, we propose a dynamically reconfigurable mechanism based on sub-netting. This mechanism can reduce the number of cores who will receive a broadcast message at one time. Actually, those messages will only reach a subnet which is a region in the network. In this way, we largely reduce the network traffic and the cost of using persistent request.

The rest of the paper is organized as follows: In Section 2, we give some background about Token Coherence. Section 3 presents our dynamically reconfigurable mechanism for Token Coherence. In Section 4, we describe the simulation environment and the results are shown in Section 5. Finally, Section 6 concludes the paper.

2 Token Coherence and Related Works

2.1 Token Coherence

Traditional protocols are based on basic coherence state transitions like MOESI, above which there are specific implementations of the protocol, so the protocol model includes two layers. Different from that, Token Coherence replaces the former model with a model which has three layers [6, 7]. Above MOESI, there are correctness substrate and performance policy. The correctness substrate ensures the accuracy of state transitions and prevents starvation. Since the correctness of protocol has been considered, the performance policy only aims at improving the efficiency of the protocol.

Token Coherence ensures the accuracy of state transitions through token counting. This mechanism can be summarized as following rules: At all time, every memory block in system holds a fixed number of tokens (usually equals to the number of cores). A core can write a block only if it gets all tokens. A core can read a block only if it has at least one token of that block.

In Token Coherence, cores request tokens or data by transient requests which are unordered messages. Since token counting does not ensure the completion of transient requests, they may cause starvation. Token Coherence uses persistent request to prevent starvation. When the starved node broadcasts a persistent request, the nodes who receive that message will finally give all the tokens and data of the destination block to the starver. Then the starved node will eventually get enough tokens and data to complete the operations.

Different from token counting and persistent request, the performance policy aims at efficiency no matter whether the protocol plays right. Currently, there are three kinds of performance policies: TokenB, TokenD and TokenM. In our work, we use the TokenB performance policy which has not any indirect nodes in system.

2.2 Related Works

Martin [6] proposed a prediction mechanism to convert broadcast messages to multicast messages, and it will obviously improve the scalability of Token Coherence.

Blas Cuesta, Antonio Robles and Jose Duato proposed several improvements to Token Coherence, including priority request [9] which aims at replacing persistent request, multicast coherence message [10] which can take advantage of the benefits offered by priority request, and message packing methods [8] to reduce the harm of broadcast message.

3 Dynamically Reconfigurable Token Coherence

In this section, we describe the main attributes of our dynamically reconfigurable mechanism and its implementation in the Gem5 Simulator [13]. First, we explain how this mechanism reduces the cost of broadcast. Then, we describe the implementations in Gem5 for the purpose of realizing the mechanism.

3.1 Reducing the Cost of Broadcast

In Token Coherence, when a core wants to store something, it will firstly search the L1 Cache for the related memory block. If the L1 Cache does not hold enough tokens and data for that block, then the L1 Cache will broadcast a transient request to all nodes including L2 Cache and memory (Fig. 1(a)). Those nodes will eventually send all tokens and data to the requestor. However, Broadcasting transient requests and persistent requests is obviously a huge cost.

According to the characteristics of parallel applications, we can dynamically reconfigure the whole network to several regions named subnet, each of which can run an application independently. Therefore, we can convert the broadcast-based transient requests and persistent requests to subnet-based broadcast messages. So, if a core in subnet 1 wants to store something, in the preceding case, the broadcast messages will be limited in a region (Fig. 1(b)). The NI in Fig. 1is able to filter the destination when L1 Cache broadcasts messages.

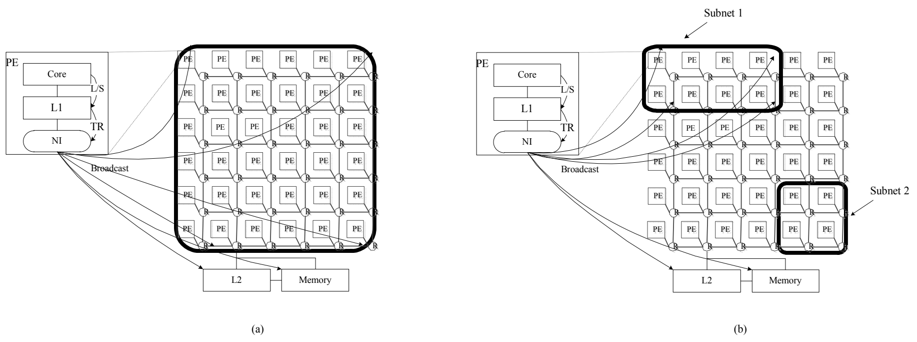


Fig. 1. Broadcast in Token Coherence, (a) before subnetting, (b) after subnetting, L/S=Load/Store, TR=Transient Request, NI=Network Interface

3.2 Implementations in Gem5

In order to realize the dynamically reconfigurable mechanism in Gem5, we have to change several specific components including message definition and coherence controllers.

Coherence Message

To support the filtering work of NI, we change the structure of coherence messages (Fig. 2) which are transmitted from one node to another in the network. Therefore, in the situation described above (Fig. 1(b)), the NI writes the "Subnet ID" when it receives the transient request from the L1 Cache, moreover, it change the "Destination" from "All nodes" to "Subnet 1". As a result, only those nodes in subnet 1 receive request message from the requestor.

Old Structure

Address	Requestor	Destination	Type	RetryNum	Other Info
0x000000ac	1	All nodes	GetX	0	……
0x00000074	2	All nodes	GetS	0	……
0x00000f3b	5	All nodes	GetX	0	……

New Structure

Address	Requestor	Destination	Type	RetryNum	Subnet ID	Other Info
0x000000ac	1	Subnet 1	GetX	0	1	……
0x00000074	2	Subnet 1	GetS	0	1	……
0x00000f3b	5	Subnet 1	GetX	0	1	……

Fig. 2. Structure of transient request message

Directory and L2 Cache

In Gem5, Token coherence is specified by SLICC (Specification Language for Implementing Cache Coherence) which describe the behavior of coherence controllers. In our work, there is only one L2 Cache Controller and one Directory Controller. So, as shown in Fig. 3, after subnetting, different subnet will share these resources.

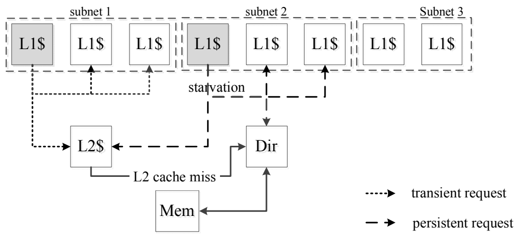
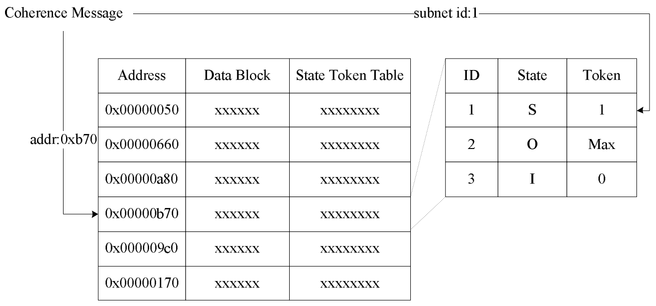


Fig. 3. Different subnets share l2 and directory, L1$=L1 Cache Controller, L2$=L2 Cache Controller, Dir=Directory Controller

The Directory Controller manages a directory which records the information of those memory blocks being used. Since it will receive messages from different subnet now, we should change the structure of directory. In our work, we add a "StateTokenTable" to each item recorded in directory. After the change, when directory receives a request message, it uses the "addr" and "subnet id" to get token and state information corresponding to a particular subnet (Fig. 4). In our system, as we assume that different applications work independently, we do not ensure the coherence among different subnets. Therefore, different "ID" corresponds to one "Data Block". We make a similar change to l2 cache memory managed by the L2 Cache Controller.



Address	Data Block	State Token Table
0x00000050	xxxxxx	xxxxxxxx
0x00000660	xxxxxx	xxxxxxxx
0x00000a80	xxxxxx	xxxxxxxx
0x00000b70	xxxxxx	xxxxxxxx
0x000009c0	xxxxxx	xxxxxxxx
0x00000170	xxxxxx	xxxxxxxx

ID	State	Token
1	S	1
2	O	Max
3	I	0

Fig. 4. Structure of directory

Data Structure

In coherence controllers, there are some structures such as "TBETable", "PersistentTable" and "TimeTable" which use "address" as key words. Since there are several subnets in our system now, we should make "subnet id" as key word simultaneously. As a result, we use two layer hash table to construct these structures instead of the original one layer hash table. Besides, we should also make all functions in SLICC adapt to the new data structures.

After the changes mentioned above, all subnets can get maximum resources (tokens and data) of a particular block at the same time, comparing to the old system. The changes do improve the performance of Token Coherence by means of limiting the destinations of broadcast.

4 Simulation Environment

We evaluate our mechanism by means of the System-Call Emulation (SE) Mode of GEM5 Simulator which offers a detailed memory system model and a precise network model. In our many-core simulation system, there are 16 cores which correspond to split L1 Caches and unified L2 Caches in the ruby memory system. Table 1 shows the system parameters.

Table 1. System parameters

GEM5 Parameters	
Processor frequency	2GHz
Cache hierarchy	Non-inclusive
Cache block size	64 bytes
Split L1 I&D Caches	L1I: 32KB, L1D:32KB, 2-way, 2 cycles
Unified L2 Cache	2MB, 8-way, 5 cycles
Directory latency	5 cycles
Memory controller latency	12 cycles

Taking "load/store ratio" and "local/share ratio" as the decisive input, we generate thousands of instructions which will be used in our system as the evaluation traces. Now, we have already found that the "load/store ratio" of SPECint2000 and SPECfp2000 ranges from 1.5 to 28 [11]. We also find that the "cache miss rate" in NAS Parallel Benchmark ranges from 5.2% to 16% when there are 16 cores in processor and the cache size is 32KB [12]. By approximately using "cache miss rate" to calculate "local/share ratio", we choose some numbers which falls in the above range as the input value. The traces correspond to these inputs will truly imitate the memory access of actual benchmark. In the next section, we compare the performance of our proposal with Token Coherence and Directory Coherence under such traces.

5 Evaluation Results

In this section, we compare our proposal with Token Coherence and Directory Coherence. First, we show our mechanism has great impact on average execution time. Second, we study the network traffic in each protocol. In the next section, we summarize these results and draw a conclusion.

5.1 Impact on Execution Time

Fig. 5 shows the average execution time for the traces which correspond to different inputs mentioned in Section 4. Because of the avoidance of indirection [5], Token Coherence reduces the average execution time when compared with Directory Coherence. In most situations discussed in Fig. 5, Token Coherence obtains improvements of 4% compared to Directory Coherence. On the other hand, since our proposal limits the broadcast messages to several regions, the persistent requests and transient requests can be completed easier than before. So, the average execution time is reduced too. In the following figure, we can see our mechanism (Token-DR) gains average improvements of 10% compared to Token Coherence.

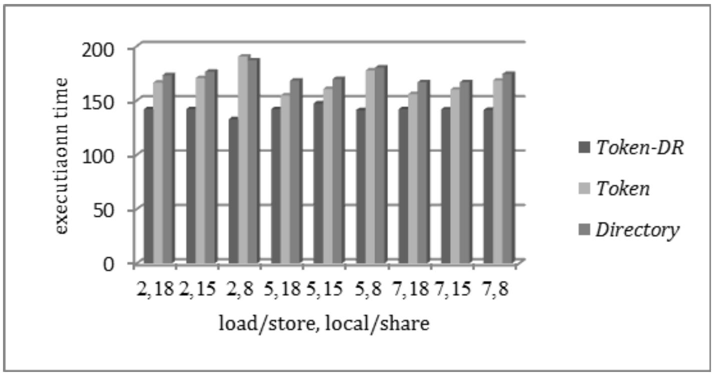


Fig. 5. Average execution time

5.2 Impact on Network Traffic

Under the testing environment mentioned in Section 4, Fig. 6 discusses the network traffic generated by protocols. We can find that Token Coherence need more network traffic than Directory Coherence, because it relies on broadcast messages. However,

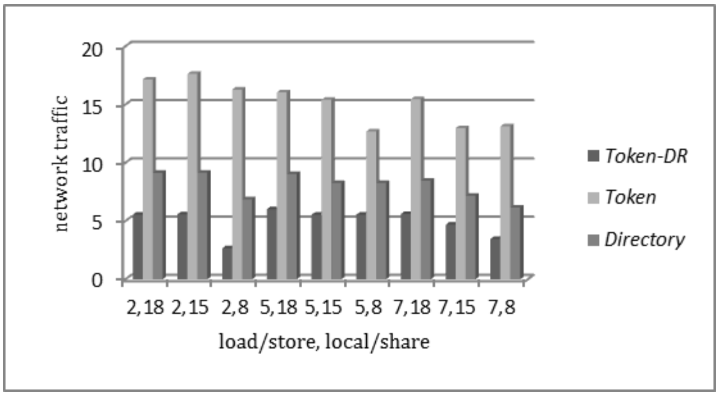


Fig. 6. Total network traffic

by the using dynamically reconfigurable mechanism, the cost of broadcast is largely reduced. Therefore, our mechanism shows tremendous improvements compared to Token Coherence.

6 Conclusions

Many-Core architecture has recently become a probable choice for designing processors. In this paper, we propose a dynamically reconfigurable mechanism for Token Coherence. Unlike the original Token Coherence, this mechanism limits the receivers of each broadcast message. This fact contributes to the reduction in network traffic and average execution time which compensate for the defects mentioned in Section 1. Therefore, combing with the dynamically reconfigurable mechanism, the cost of transplanting Token Coherence to many-core processors can be acceptable. It will gain main advantages of Token Coherence while avoiding its defects.

Acknowledgements. Our work is supported by the IBM Shared University Research (SUR201102X).

References

1. Hammond, L., et al.: A single-chip multiprocessor. IEEE Computer 30(9), 79–85 (1997)
2. Olukotun, K., et al.: The case for a single-chip multiprocessor. ASPLOS, 2–11 (1996)
3. Xu, Y., Du, Y., Zhang, Y., Yang, J.: A composite and scalable cache coherence protocol for large scale CMPs. In: Proceedings of the International Conference on Supercomputing, pp. 285–294 (2011)
4. Kumar, R., Mattson, T.G., Pokam, G., Wijngaart, R.: The case for Message Passing on Many-core Chips. University of Illinois Champaign-Urbana Technical Report, UILU-ENG-10-2203 (2010)
5. Ros, A., et al.: Cache coherence protocols for Many-Core CMPs. In: Parallel and Distributed Computing (2010)
6. Matin, M.: Token Coherence. University of Wisconsin-Madison (2003)
7. Martin, M.M.K., Hill, M.D., Wood, D.A.: Token Coherence: decoupling performance and correctness. In: 30th Annual International Symposium on Computer Architecture, pp. 182–193 (2003)
8. Cuesta, B., Robles, A., Duato, J.: Switch-Based Packing Technique for Improving Token Coherence Scalability. In: Ninth International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT 2008), pp. 83–90 (2008)
9. Saez, B.C., Robles, A., Duato, J.: Efficient and Scalable Starvation Prevention Mechanism for Token Coherence. IEEE Transactions on Parallel and Distributed Systems, 1610–1623 (2011)
10. Cuesta, B., Robles, A., Duato, J.: Improving Token Coherence by Multicast Coherence Messages. In: 16th Euromicro Conference on Parallel, Distributed and Network-Based Processing (PDP 2008), pp. 269–273 (2008)
11. Hennessy, J.L., Patterson, D.A.: Computer Architecture: A Quantitative Approach, 4th edn. Morgan Kaufmann, San Francisco (1996)
12. Wong, F.C., Martin, R.P., Arpaci-Dusseau, R.H., Wu, D.T., Culler, D.E.: Architectural requirements and scalability of the NAS Parallel Benchmarks. In: Proceedings of Supercomputing (1999)
13. Binkert, N., et al.: The gem5 simulator. SIGARCH Comput. Archit. News 39(2) (2011)

Dynamic and Online Task Scheduling Algorithm Based on Virtual Compute Group in Many-Core Architecture

Ziyang Liu, Yuzhuo Fu, Jiang Jiang, and Xing Han

School of Micro-electronic, Shanghai Jiao Tong University, Shanghai, China
{liuziyang,Fuyuzhuo,jiangjiang,hanxing}@ic.sjtu.edu.cn

Abstract. Efficient task scheduling for a series of applications on Mesh based many-core processors is very challenging, especially when resource occupation and release are required in some running task phases. In this paper, we present a dynamic and online heuristic mapping for efficient task scheduling based on Virtual Computing Group (VCG), and an algorithm managing free resources based on rectangle topology is proposed as well. This method quickly finds proper rectangle resources for a task, partitions processing elements (PEs) into a Virtual Computing Group by constructing a subnet, and maps communicating subtasks on adjacent PEs according to data dependency and communication dependency. Compared with the existing algorithms, our mapping algorithm can reduce the total execution time and enhance the system throughput by 10% in simulations.

Keywords: Many-Core architecture, Virtual Computing Group, Dynamic and online reconfiguration, Task mapping, Resources management.

1 Introduction

Many-core processors have a set of processing elements (PEs), and are capable for parallel computing in current applications. It's widely used for multi-threads parallel computation in large complex applications. However, as PEs in processors increase, the current core utilization is low and the performance can't rise significantly [1].

An efficient solution is task scheduling based on reconfigurable system [2][3]. According to different task, task scheduler divides free PEs into domains. To some extent, this reconfigurable method improves system performance. The decision where a task is mapped determines the fragmentation of the whole area and it may leads to resources competition between other tasks in some bad cases. So the efficient task scheduling algorithm which takes subtasks mapping and free resources management into account is the key point to reduce total execution time and enhance throughput of the whole system.

Mapping subtasks onto the many-core processors includes two aspects, (a) initial mapping: mapping at the time when tasks need to run at the initial time and (b) *Phase Mapping* when specific phase reconstruction of tasks happens, which will be discussed in Section 3.

In this paper, we focus on the phase mapping and the resulting irregular topology resources management, as there are lots of researches on dynamic initial mapping and regular resources management. The main contributions of this paper are as follows:

- Introduce a new task model, and the related task phase, data dependency and communication dependency of subtasks.
- Research on a new online task mapping method based on reconfigurable many-core architecture.
- Develop free resources management based on rectangles, especially in some irregular topology when online reconfiguration tasks place. This method is based on the partition way of [4].

Section 2 reviews related work. Section 3 introduces the system architecture and the task model. Section 4 and 5 discuss free space partition and task mapping, respectively. Section 6 presents simulation results and related analysis. Section 7 concludes the paper.

2 Related Work

Task mapping techniques in many-core processors are presented in many research papers. Static mapping in [9] find fixed placement of tasks at design-time with a well known computation and communication behavior between subtasks. Therefore, these mapping heuristics are not suitable for dynamic applications of which information is unknown. Briao et al.[10] present dynamic task allocation strategies based on bin-packing algorithms for real-time applications in homogeneous Multiprocessor Systems-on-Chip(MPSoC). While in the MPSoC platforms, Amit Kumar Singh et al. [11] proposed communication-aware mapping method to deal with the congestion in channels and workloads for traffic. Carvalho et al.[12] present on the fly mapping heuristic for dynamic mapping in NoC-based heterogeneous MPSoC. Wei Chen et al. [13] summarize an algorithm which proposes the systematic way in the area of task decomposition and partition and reconfiguration in many-core processor. And Cao YJ et al. [14] present an adaptive scheduling algorithm based on dynamic core resource partitions for many core processor systems. On the aspect of free resources management, the heuristic in [4] proposes a fast template placement for tasks in reconfigurable systems, and provide us an efficient way to manage free rectangle resources. Herbert Walder et al. [15] investigate task placement and footprint transform approach in their paper. Li Tao et al. [16] develop the TT_KAMER algorithm for more efficient placement in FPGA.

3 Architecture and Task Model

3.1 Architecture

The system used in this paper is a 2D mesh based many-core system, which is illustrated in figure 1(a). The architecture contains a Manager Processor (MP) and a series of PEs, which communicate with each other via NoC. And each PE is capable of supporting only one subtask (thread). The subnet reconstruction according to mapping is the key to construct a VCG for a task and isolate task crosstalk. The MP, running operating system, is responsible for task mapping, resources management, reconfiguration control and so on.

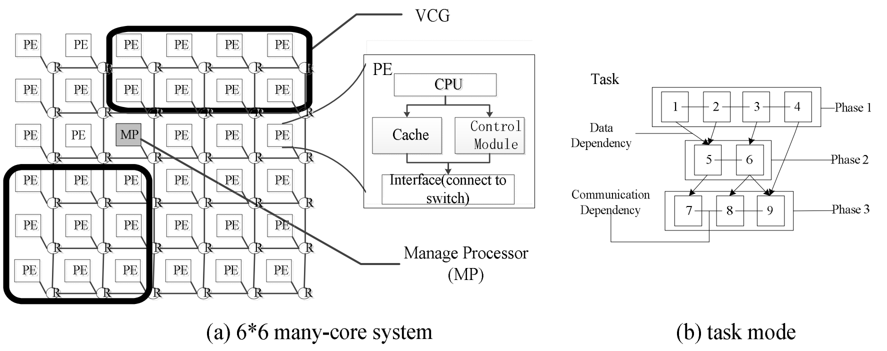


Fig. 1. 6*6 many-core system and task mode

3.2 Task Model

Tasks' phase property is clearly illustrated in [5][6][7][8]. That is, threads' behavior, interacting with each other in tasks or programs, can be divided into two categories, communication dependency and data dependency [17], explained in figure 1(b).

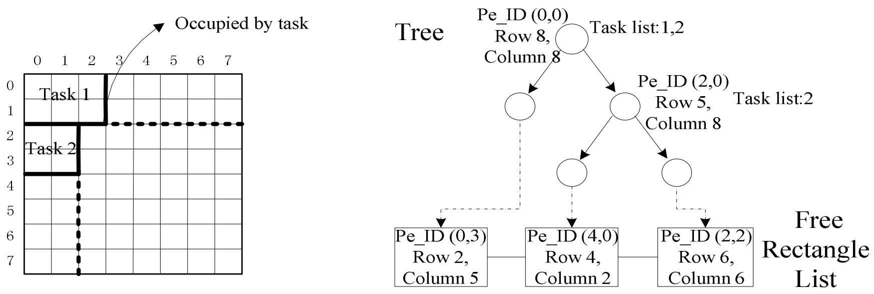
- Communication Dependency: The relationship between threads which communicate with others by using certain programming model such as MPI and so on.
- Data Dependency: The relationship between threads which may pass parameters to another when one is finished.

Because of communication and data dependency, all subtasks (threads) can't run in parallel at one time. And a set of subtasks which compose a *Task Phase* can run at the same time. Because phases have different number of subtasks, the resources may vary at different phases. Reconfiguration between phases, called phase reconstruction or phase mapping, can meet the flexible requirement. So it's necessary to research on phase reconstruction and the resulting resource management.

4 Free Resource Management Based on Rectangles

Our goal is designing a fast but not necessarily optimization method for management. Bazargan et al. [4] proposed an efficient Bin-Packing algorithm for placement. Based on it, we develop a more suitable way for our task model and phase reconstruction. And it takes O(n) time to research free rectangle resources and O(1) time for inserting them in the worst case. This method is consisted of two parts: (a) new task mapping and free rectangle division, (b) phase reconstruction, including releasing some nodes, adding nodes from task's free neighbors, and resources reclamation when a task finishes.

We use data structures of tree and list in this paper, explained in figure 2(b).


Occupied by task
0 1 2 3 4 5 6 7
Task 1
Task 2
Tree
Pe_ID (0,0) Row 8, Column 8
Task list:1,2
Pe_ID (2,0) Row 5, Column 8
Task list:2
Pe_ID (0,3) Row 2, Column 5
Pe_ID (4,0) Row 4, Column 2
Pe_ID (2,2) Row 6, Column 6
Free Rectangle List

(a) 8*8 resources (b) data structure

Fig. 2. Resources and the corresponding data structure after initial mapping

Only leaf node can be selected for task mapping. Each leaf symbols a free rectangle, and it's consisted of row and column size, PE ID at the top-left corner of a rectangle. And each node, which points to Class Rectangle, is the same as a node of list. So if a new task needs to map, MP will search a free rectangle in list with no necessary to search it from tree's root to suitable leaf. After top-left mapping and adding corresponding task ID from root to node selected, the selected rectangle resources will be divided into small rectangles. Then theses rectangles are the leaves of the original node.

4.1 New Task Mapping

New task mapping, of which details are mentioned in [4], is the top-left mapping heuristic with First Fit (FF), Best Fit (BF) and Worst Fit (WF) in this paper. And what‘s different with Bazargan is that irregular topology, such as L shape, illustrated in figure 3(c), should be taken into account, because of the number of subtasks in Phase One. As a result, the free rectangle remained should be divided into 3 small rectangles. And these leaves are inserted to the original node's sub-tree and free rectangle list after division.

We have tried different heuristics for how to partition the original free resources into several rectangles. Horizontal division is dividing a rectangle along task's the top/bottom edge, while vertical division is dividing rectangles along task's left/right edge, L division is dividing rectangles along the inner corner of L. Then the heuristics can be defined in terms of rectangle size as follows:

- Largest Rectangle (LR): the biggest one in one heuristic is bigger than the ones in the other two heuristic. Then this heuristic is selected, as the biggest one could accommodate more tasks than others.

- Square Rectangle (SR): the biggest one in one heuristic is closest to squares than these in the other two heuristic. The reason favoring SR is it's more likely to contain tasks with the irregular topology or a high ratio between its row and column.

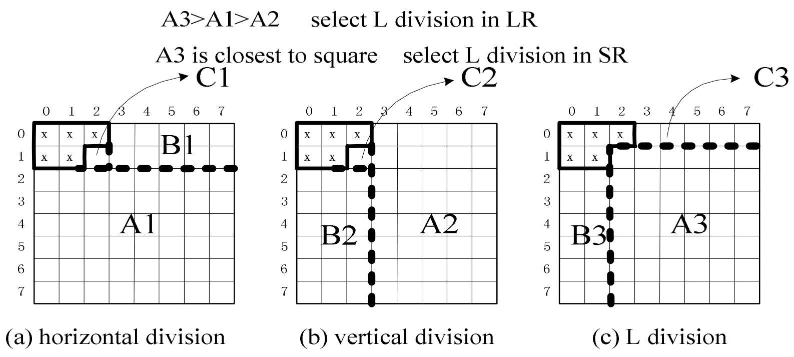

A3>A1>A2 select L division in LR
A3 is closest to square select L division in SR
C1
C2
C3
B1
A1
B2
A2
B3
A3
(a) horizontal division
(b) vertical division
(c) L division

Fig. 3. Split rules

4.2 Phase Reconstruction

Phase reconstruction includes two aspects: (a) release free PEs, and (b) add some PEs from its rectangle neighbors, which can be easily found in the task node's sub-tree.

Release Free Resources/PEs

When the previous phase is completed and a task needs to release nodes, according to task list, the MP will find the deepest task node in the tree. Only free PEs which is neighbors of the deepest node's leaves will be released, and then the adjacent nodes are merged into a small rectangle. After merging, all rectangles are inserted into the task node's sub-tree, and free rectangle list updates. In good cases, the leaf rectangle may become big one after merging PEs. In bad cases, all scattered neighbor nodes, which lead to plenty of fragment, are directly inserted into the deepest node's sub-tree. The process is explained in figure 4(a).

When all phases are completed, all resources in the task need to be reclaimed. Based on our data structure, it's easy to recycle all rectangles used in this task. What we should do are deleting task ID in the tree, treating child nodes with empty task list as leaves, and deleting all child nodes below them.

Add Free Resources and Split Rectangle Resources

When a task needs to add some resources/PEs, according to task list, the deepest task node will be picked. And one of its free rectangle leaves will be selected according to the rule First Fit (FF)/Best Fit (BF)/ Worst Fit (WF). Assume BF is needed in this paper. After picking a free rectangle, the neighbor nodes of the task in it are chosen as the edge of added topology, and the other edge length is computed like this: assume M is row, and S is the number of subtasks which need new PEs, then the column N and remainder R are

$$N = \left\lfloor \frac{S}{M} \right\rfloor \tag{1}$$

$$R=\mathrm{mod}(S, M) \tag{2}$$

If R is 0, the topology is a rectangle, then the column is N. Vice versa, the topology is L, and its column is N+1.

In bad cases, although the selected rectangle size is enough to accommodate all subtasks, its column is not bigger than the new topology's column due to the number of neighbors, M. Then our rule will change: Let the picked rectangle's row be the row of the added topology, and the column and remainder are computed according to the equation above. And the whole process is illustrated in figure 4(b).After deciding the new topology, the next step are adding task ID in this selected node, inserting sub-tree nodes into this node after rectangle division and update free rectangle list, which are mentioned in the section above.

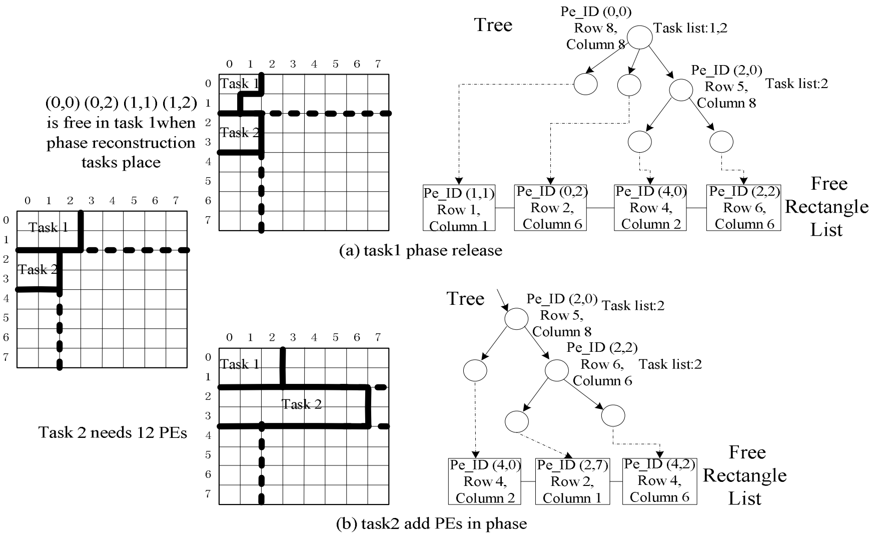

Tree
Pe_ID (0,0) Row 8, Column 8
Task list:1,2
Pe_ID (2,0) Row 5, Column 8
Task list:2
(0,0) (0,2) (1,1) (1,2) is free in task 1when phase reconstruction tasks place
Task 1
Task 2
Pe_ID (1,1) Row 1, Column 1
Pe_ID (0,2) Row 2, Column 6
Pe_ID (4,0) Row 4, Column 2
Pe_ID (2,2) Row 6, Column 6
Free Rectangle List
(a) task1 phase release
Tree
Pe_ID (2,0) Row 5, Column 8
Task list:2
Pe_ID (2,2) Row 6, Column 6
Task list:2
Task 2 needs 12 PEs
Pe_ID (4,0) Row 4, Column 2
Pe_ID (2,7) Row 2, Column 1
Pe_ID (4,2) Row 4, Column 6
Free Rectangle List
(b) task2 add PEs in phase

Fig. 4. Phase reconstruction

5 Mapping Strategies

Input is data/communication dependency graph and PE ID asking for one subtask. Output is subtask ID list and PE ID list, of which element corresponds to each other after mapping.

5.1 Initial Mapping

The initial mapping maps the subtasks of Phase 1 onto a block, which is consisted of free PEs. And the topology of block is decided by MP according to the number of subtasks. Then a block of PEs is occupied and VCG is formed by constructing a subnet. The initial mapping detail is as follows. First, map the subtask which has maximum communication with others in the center of VCG. Second, map its communication subtasks, according to amount of communication, from one hop to several hops around the center until the subtasks are totally mapped in the predetermined area.

5.2 Mapping before Phase Reconstruction

The subtask has characteristic of data dependency and communication dependency as discussed in Section 3. This feature directly decides the schedule order of subtasks. Our algorithm is illustrated as follows. When subtasks get scheduled, other subtasks which communicate with the scheduled ones must raise their communication priority. On the other side, in subtask pairs with data dependency, one must raise its data priority when the depended one gets scheduled. As a result, the free PE gets different subtask according to different priorities. We can choose one of them:

- Communication priority first: the subtask that has the largest amount of communication with those which were scheduled in the VCG gets schedule first, until all of them get scheduled, because one key subtask may lead to congestion and task pause in some situations.
- Data priority first: the subtask gets scheduled right after the depended one finished. Because of cache coherency, subtasks get data with no need to access memory off-chip.

5.3 Phase Mapping

The phase mapping takes place when the subtasks of previous phase are totally finished. And the MP checks which subtasks can be scheduled next. If the number of subtasks is smaller than the number of current free PEs in VCG, free PEs in this VCG will be released and VCG will be reconstruct. Vice versa, MP will decide which free rectangle is suitable for adding PEs. And the basic rule is the selected rectangle must be a neighbor of the task, because traffic workload and communication reduction must be taken into account. There are alternatives as follows, choose the biggest rectangle or the smallest rectangle.

If releasing resources is needed, those which will be released are the neighbors of free rectangles around the task, given topology must be kept as regular as possible and fragment should be little. What's more, the nodes released, along with their rectangle neighbor, may merge into a bigger rectangle if their size is equal to original neighbor's row or column. In the bad case, PEs released can merge into a big rectangle with each other if they are adjacent. Then VCG becomes smaller after reconstruction.

If there're not enough resources in the original task, MP will find a free rectangle, assume the smallest one, which has enough resources for phase reconstruction, among the task's neighbor. From the neighbor nodes to the other side of the selected one, subtasks will be mapped according to the initial mapping rules. Then new resources are added into the task and the former rectangle is divided into smaller one and VCG changes after mapping.

6 Simulations

Experiments are performed on the GEM5simulation platform. Our simulations are evaluated on 4*4 and 6*6 many core platforms. The evaluated scenarios are randomly generated applications using Task Graph For Free [18], and each subtask contains instruction flow. And computation instructions and load/store instructions compose the instruction flow.

Figure 5 shows total execution time using different mapping heuristics (A) and division rules (B). It shows the total time in 6*6 is less than the one in 4*4. In most cases, the performance using Best Fit (BF) and First Fit (FF) are almost the same, and they are better than the one using Worst Fit (WF), about 10% gains totally. WF leads to reduction to PE utilization and the total throughput is lower than the others. Because of shape requirement, square division leads to lower throughput.

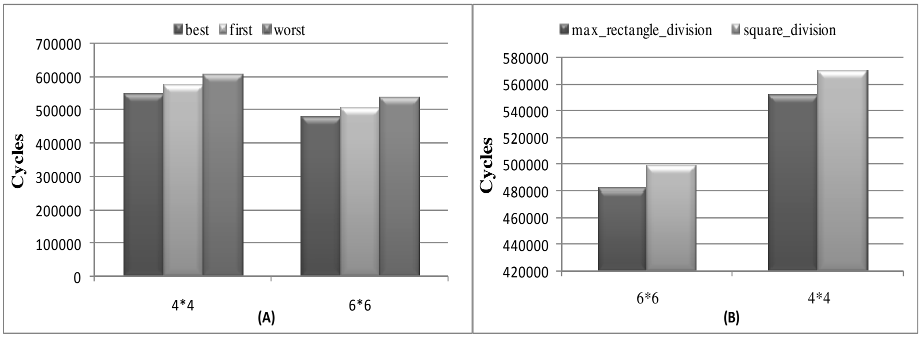


Fig. 5. Execution time

Because of different task sizes and mapping methods, BF can accommodate more tasks than WF and more PEs can run at one time. As a result, BF performs sometimes better than FF, and the difference between them is not significant. On average, BF improved the PE utilization by only 11.75%, illustrated in figure 6.The reason why

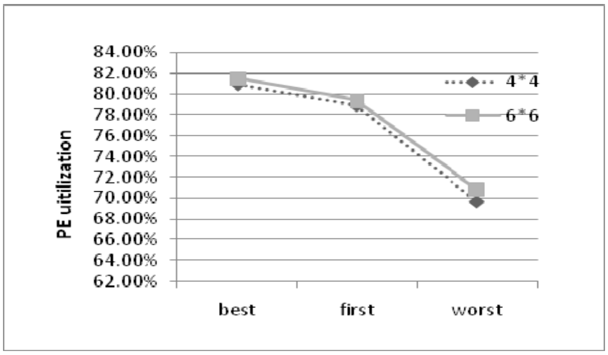


Fig. 6. PE utilization in 4*4 and 6*6 platforms

PE utilization is not very high is that the scatter PEs which are not neighbors of free rectangles in a task will not released in phase reconstruction, given that fragment won't increase dramatically using this heuristic.

Table 1 shows gains improved by many heuristics. And most heuristics mentioned above bring in system improvement compared to the random mapping algorithm on many-core systems without VCG and subnet.

Table 1. Improvement using different algorithms

Execution Time	Best Fit with VCG	Worst Fit with VCG	Best Fit without VCG	Worst Fit without VCG	Random mapping without VCG
Cycles	91054	97952	101341	102876	111254
Gains(%) compared to Random mapping without VCG	18.5	11.6	9.0	7.5	0

Execution Time	Best Fit with VCG	Worst Fit with VCG	Random mapping without VCG
Cycles	91054	97952	111254
Gains(%) compared to Random mapping without VCG	18.5	11.6	0

7 Conclusion and Future Work

This paper describes some new mapping strategies and a modified method based on free resources management based on rectangles for our task model. And we have evaluated several heuristics which enhance system throughput and PE utilization in 4*4 and 6*6NoC-based platform. We clearly demonstrate that the newly proposed method can consistently provide notable reduction in total execution time and communication overhead and enhance throughput in many-core processors.

Based on our mapping method, we plan to evaluate real time benchmarks on GEM5 and devise techniques for online task migration and improve our free resources management.

Acknowledgements. Our work is supported by the IBM Shared University Research (SUR),System-level Virtualization Based on Reduced Hyper Multi-core Architecture, SUR201102X.

References

1. Bhattacharjee, A., Contreras, G., Martonosi, M.: Parallelization libraries: Characterizing and reducing overheads. ACM Trans. on Architecture and Code Optimization 8(1), 5–29 (2011)
2. Maher, B.A.: Atomic block formation for explicit data graph execution architectures, PhD thesis, Department of Computer Sciences, The University of Texas at Austin (August 2010)
3. Govindan, S.M., Robatmili, Esmaeilzadeh, H., et al.: Scaling power and performance via processor composability, Technical report, 2010. UT Austin, Department of Computer Sciences TR-10-14 (2010)
4. Bazargan, K., et al.: Fast Template Placement for Reconfigurable Computing Systems. IEEE Design and Test of Computers 17, 68–83 (2000)
5. Purtilo, J.M., Hofmeister, C.R.: Dynamic Reconfiguration of Distributed Programs. Distributed Computing Systems, 560–571 (1991)
6. Adamo, J.-M., Bonello, C., Trejo, L.: Programming Environment for Phase-Reconfigurable Parallel Programming on SuperNode. In: Parallel and Distributed Processing (1992)
7. Sherwood, T., Perelman, E., Hamerly, G., Sai, S., Calder, B.: Discovering and Exploiting Program phases. IEEE Micro, 84–93 (2003)
8. Hauck, S.: Reconfigurable Computing: the Theory and Practice of FPGA-Based Computing, Section 9.2.2, p. 210. Elsevier Inc. (2008) ISBN 978-0-12-370522-8
9. Murali, S., et al.: A methodology for mapping multiple use-cases onto networks on chips. Proceedings of DATE, 118–123 (2006)
10. Briao, E.W., et al.: Dynamic task allocation strategies in mpsoc for soft real-time applications. Proceedings of DATE, 1386–1389 (2008)
11. Singh, A.K., Srikanthan, T., Kumar, A., Jigang, W.: Communication-aware heuristics for online task mapping on NoC-based MPSoC platforms. Journal of Systems Architecture: the EUROMICRO Journal Archive 56(4) (July 2010)
12. Carvalho, E., Moraes, F.: Congestion-aware task mapping in heterogeneous mpsocs. In: International Symposium on SoC, pp. 1–4 (November 2008)
13. Chen, W.: Task Partitioning and Mapping Algorithms for Multi-core Packet Processing Systems, p. 255, Masters Theses (2009)
14. Cao, Y.J., Qian, D.-P., Wu, W.-G., Dong, X.-S.: Adaptive Scheduling Algorithm Based on Dynamic Core-Resource Partitions for Many-Core Processor Systems. Journal of Software 23(2), 240–252 (2012)
15. Walder, H., et al.: Non-preemptive Multitasking on FPGAs: Task Placement and Footprint Transform
16. Li, T., Yang, Y.: Algorithm of Reconfigurable Resource Management and Hardware Task Placement. Journal of Computer Research and Development, 375–382 (2008)
17. Ferrante, J., Ottenstein, K.J., Warren, J.D.: The program dependence graph and its use in optimization. ACM Trans. Program. Lang. Syst. 9(3), 319–349 (1987)
18. Dick, R.P., et al.: Tgff: task graphs for free. In: Proceedings of Workshop on Hardware/Software Co-Design, pp. 97–101 (1998)

ADL and High Performance Processor Design

Liu Yang, Xiaoqiang Ni, Yusong Tan, and Hengzhu Liu

School of Computer Science
National University of Defense Technology
ChangSha, China
yangliujoy@gmail.com

Abstract. Architecture Description Language (ADL) can model many computer related problems and is widely used in software and hardware design. When used in processor design, lots of institutes and companies use ADL as processor quick prototype design language and use it to generate processor simulator, test-benches and compiler utilities. This paper analyzes and compares three processor description languages. We also give the disadvantages of modern ADL when used in high performance processor design and give some suggestions for further ADL development.

Keywords: ADL, High-Performance Processor Design, PML.

1 Introduction

Recently, the design of high performance processor becomes more and more complicate. The processor design techniques developed from scalar, single-core to super-scalar, multi-cores, multi-threads. The complexity of processor design increases. More and more I/O controllers are integrated into high performance processor and also bring forward higher challenge to design. Some other factors such as testability design, silicon area and power consumption also increase the complexity of the design. In summary, the design complexity of high performance processor becomes higher and higher than ASIP (Application Specific Integrated Processor) or SoC (System on Chip) processors.

The process of high performance processor design is: scheme design, logic design, logic verification, physical design and physical verification. Such process has some problems, such as: consideration of processor silicon area is not comprehensive, the set of performance parameters is not reasonable, logic design is complex, logic verification and physical verification are not comprehensive. There may be more challenges in designing multi-core, multi-threads processors, such as: memory protocol defects, verification of memory coherence, design and verification of NOC (Network on Chip). These problems cannot be solved completely by existing design methods or the cost of solving such problems is too high. So we need to change the methodology in high performance processor design. We need to consider various problems of design, verification and application of processors at the beginning of the processor design.

PML (Processor Modeling Language) is a kind of ADL. The concept of ADL was put forward in 1960s and it was used to implement software automation or software test. With the development of ADL and the increase of complexity of processor design, PML began to be used in processor design [1], from then on, the categories and application scope is developed vigorously [2]. In this paper, we use ADL to represent PML as a more general terminology.

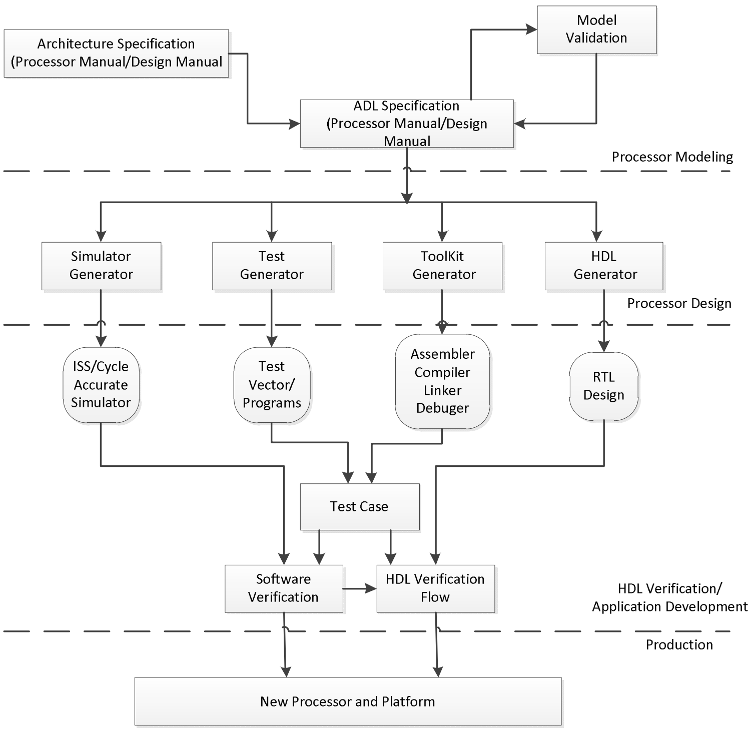

Architecture Specification
(Processor Manual/Design Manual
Model Validation
ADL Specification (Processor Manual/Design Manual
Processor Modeling
Simulator Generator
Test Generator
ToolKit Generator
HDL Generator
Processor Design
ISS/Cycle Accurate Simulator
Test Vector/ Programs
Assembler Compiler Linker Debuger
RTL Design
Test Case
Software Verification
HDL Verification Flow
HDL Verification/ Application Development
Production
New Processor and Platform

Fig. 1. ADL and its Usage in Processor Design

According to styles of description, ADL can be classified as structural description, hybrid description and behavioral description. According to the usage, ADL can be classified as synthesis oriented description languages, verification oriented description languages, compiling tools oriented description languages and simulation oriented description languages [3]. This paper will introduce and compare the ADLs used in high performance processor design.

In high performance processor design, ADL can be used in model simulation, automatic test and automatic generation of software tools and so on. With ADL, we can simulate the architecture and instruction set of processor at the beginning of processor design and explore the design space of processor. Using ADL, we can implement the automatic generation of test programs and test code; we can improve the test of processor and detect the bugs in processor design. In addition, current ADL can support the generation of tool chain and can automatically generate compiler, assembler, and linker. Developing processors with ADL can decrease the human cost in processor design and accelerate the time for coding, verification, taped out and the development of applications.

Currently, there are various ADLs to support processor design, such as MIMOLA[4], nML[5], LISA[6], BlueSpec[7], EXPRESSION[8], ASIP Meister[9], TIE (Tensilica)[10], MADL[11], ADL++[12], ArchC[13], MAML[14], GNR[15], RADL[16] and so on. These description languages can be used in modeling, automatic generation of compiling environment, test-bench and RTL (Register Transfer Language) code. The support is different for various languages. This paper focuses on three popular description languages and compares these three ADLs in description abilities and toolset functions.

Firstly, this paper introduces the application of ADL in high performance processor design. Section 2 introduces three popular ADLs. Section 3 compares these languages. Section 4 describes the lack of these languages in current high performance processor design. The last section is the conclusion of this paper.

2 Introduction of Three ADLs

2.1 LISA

LISA is a processor description language designed by CoWare and is used in the products named CoSy compiler design environment. LISA is an instructions driving language. It can describe general processors, RISC processors, DSP, ASIPs (Application Specific Instruction-Set Processors) and so on. LISA can describe cycle-accurate and non-clock-accurate processor models. It can also generate a set of software development environment, includes assembler, linker, compiler, simulator and debugger. It can generate RTL code. It can make the development, test and verification of processors more convenient. The original intention of LISA is to construct optimized compiler, so it provides good support to the description of instruction set and the definition of the function of instructions. The constructing tool of compiler can generate optimized compiler according to the function of instructions and the characteristics of processor pipelines. CoSy has various compiler optimizing modules. It can support the construction of compilers for CISC、RISC、VLIW and SIMD instruction system. Today, more than 100 companies' products use CoSy to construct compiling system, include the scope from 8bit single-chip to 256bit VLIW DSP. The CoSy express is integrated into PD (Processor Designer) of Synopsys. PD can be used to implement design and verification of processors, the construction of development environment and so on. On one hand we can use it to explore the design area of processors; on the other hand we can use it to implement iterative design, as shown in Fig.2.

Besides supporting description of instruction set, LISA supports various peripherals and interfaces. It also supports the timing model of underlying hardware.

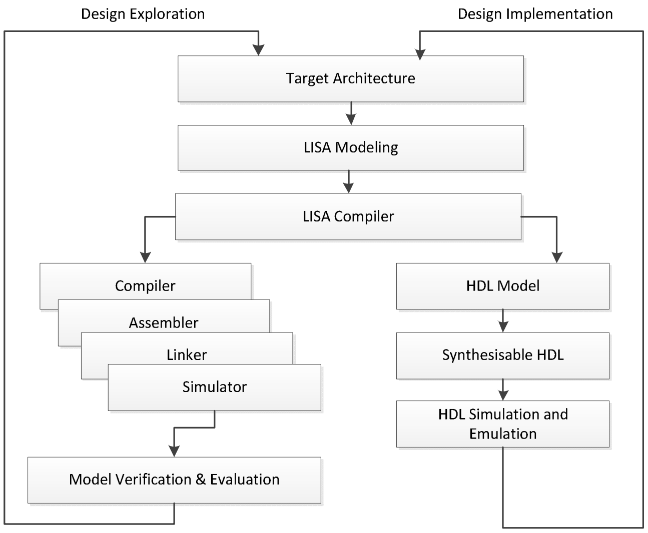


Fig. 2. Using LISA in Processor Design

2.2 ArchC

ArchC is an open source ADL designed by LSC (Computer System Laboratory) of Campinas. ArchC uses SystemC as the construct tool of simulator. ArchC uses GNU GCC and Binary Utilities to implement the mapping of ArchC description to compiler, assembler, linker and debugger. The design process is shown in Fig.3. ArchC is departed into two parts: AC_ISA and AC_ARCH. AC_ISA describes instruction set, instruction function, instruction pipeline and AC_ARCH describes the resources of

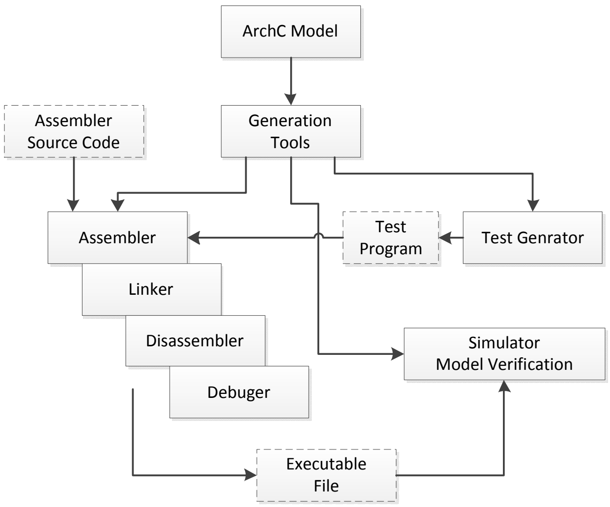


Fig. 3. Usage of ArchC in Processor Design

architecture. ArchC provides "acstone" to support the automatic generation of test-bench. Although the current version is not perfect, using of automatically testing tools can improve the coverage of test and decrease design risk that may occur.

Currently, ArchC implement the description of MIPS-R3000、PowerPC、LEON 、SPARC-V8、Intel8051 and PIC. With TLM extension, ArchC can describe multi-cores processors.

2.3 MAML

MAML（Machine Markup Language）is used in BUILDABONG system of Paderborn and it is a kind of ADL based on XML. It can be used to model, simulate and construct compiling environment for processors of special applications. The characteristics of MAML include the support to the description of multi processors architecture and the support to complete design exploration and optimization semi-automatically. In BUILDABONG system, lots of tools can be put together to complete the design of software and hardware.

MAML can be used to describe multi-processors model. Interconnection methods and communication models are also defined in MAML. It supports BUS, MESH, Cross-Switch, Fat-Tree and other network topology.

3 Comparing of Three ADLs

The comparing of the three kinds of ADL introduced in previous section is shown in Table 1. The comparing focuses on the abilities of description and the toolsets functions using these ADLs.

Table 1. Comparing of Three ADLs

	LISA	**ArchC**	**MAML**
Description Abilities			
Instruction Function	Yes	Yes	Yes
Pipeline description	Yes	Yes	Yes
Memory Hierarchy	Yes	Yes	No
System and IO	Yes	Yes	Yes
Multi-Cores	Yes	Through TLM	Yes
Multi-Threads	No	Through TLM	No
NoC	N/A	Through TLM	Yes
Toolsets Functions			
Instruction Set Simulator	Yes	SystemC	Yes
Cycle-Accurate Simulator	Yes	SystemC	Yes
Test bench Generation	Yes	Yes	N/A
Compiler Environment	Private	GCC	LCC
Debugger	Private[1]	GDB	N/A
Synthesizable RTL Code	Verilog/VHDL	N/A	Verilog/VHDL
Optimize for Area and Power	Yes	No	No

[1] Support generation of JTAG in hardware and software debugger.

We can conclude from Table 1 that the commercial ADL can all support automatic generation, but ArchC is not good at it. According to the abilities of description, the three kinds of ADL cannot provide full description for current multi-cores, multi-threads processors and the complex interconnection structure. With TLM, ArchC needs to program specific source code for different behaviors. Although it can compensate the lack of the description abilities, it violates the original intention of ADL.

4 Advantage and Disadvantage of ADL

Through the introduction of the challenge of high performance processor design and the comparing of various ADLs, we can see current ADL technology can support fast modeling, simulation, verification and construction of compiling environment and so on. It can describe instruction set, pipeline, peripheral and so on. But ADL has some disadvantages as follows:

1. The description abilities for multi-cores and multi-threads processors are not enough.
   Current ADLs cannot model multi-threads processors and cannot include threads execution model or virtual core/processor for virtualization. Although most modern processors provide virtualization support, the ADL cannot model the Hypervisor, Supervisor and User mode execution model.
2. The support to evaluate typical silicon process is limited.
   The silicon foundry provides Nano-meter techniques and supports more and more transistors in one chip. The Nano-effect affects the yield of single wafer and makes technologies faults more obvious, so the ADL should better provide the abilities to model Nano-meter silicon process for the silicon evaluation.
3. Cannot evaluate and optimize the power consumption for the lack of power model.
   More complicated processors consume more power and emit more heat; these affect the design of the package and the system. It also goes against the Green Computing. The ADLs can use foundry standard cell library and IP library to evaluate the power consumption before the implementation of the processor.
4. The auto design space exploration is not enough.
   Only LISA can evaluate multi design corner automatically for applications, the other ADLs need manual intervention. This function can greatly reduce the time of architecture design. The toolsets better support specific applications such as H.264 encode/decode and Soft-defined Radio to evaluate the ASIP and give the design parameters like cache size, buffer size, register size and computing resources etc.

From the analysis of the ADL advantages and disadvantages, ADL can accelerate processor design, coding, verification and the applications. It can also avoid design defects and the lack of verification induced by human factors.

In conclusion, designing of new high performance processors with ADL has been a feasible way. Describing the design of existing processors with ADL can also help decide the parameters for performance and generate test-bench automatically. It can improve the test coverage and provide more complicated compiling system earlier.

5 Conclusion

This paper firstly introduces ADL usages in processor design and chooses three typical ADLs to describe in detail. We compare LISA, ArchC and MAML in description abilities and toolsets functions. We summarize the advantages of current ADL in processor design and analyze the lack of ADL in modern high performance processor design. Using ADL can benefit various phases in the process of design of high performance processors and can simplify the human work greatly. According to the analysis above, we conclude the critical problems in the research of ADL and give some suggestions to the development and evolution of ADLs.

Acknowledgements. The research is supported by National Natural Science Foundation of China with Grant No.60970037, and by Doctor Program Foundation of Education Ministry of China with Grant No.20114307130003.

References

1. Barbacci, M.R.: Instruction set processor specifications (isps): The notation and its applications. IEEE Trans. Comput. 30(1), 24–40 (1981)
2. Abrar, S.S.: Advances in SoC and Processor Modeling Methodologies, `http://www.design-reuse.com/articles/20577/soc-processor-modeling-methodologies.html`
3. Mishra, P., Dutt, N.: Processor Description Languages-Applications and Methodologies. Morgan Kaufmann Publisher (2008) ISBN 978-0-12-374287-2
4. Marwedel, P.: The mimola design system: Tools for the design of digital processors. In: Proceedings of the 21st Design Automation Conference, DAC 1984, pp. 587–593. IEEE (1984)
5. Fauth, A., Van Praet, J., Freericks, M.: Describing instruction set processors using nML. In: Proceeding of European Design and Test Conference, ED&TC 1995, pp. 503–507 (1995)
6. Zivojnovic, V., Pees, S., Meyr, H.: LISA-machine description language and generic machine model for HW/SW co-design. In: Proceeding of VLSI Signal Processing, IX, pp. 127–136 (1996)
7. `http://www.bluespec.com/hardwaremodels/hardwaremodels.html`
8. Halambi, A., Grun, P., Ganesh, V., Khare, A., Dutt, N., Nicolau, A.: EXPRESSION: a language for architecture exploration through compiler/simulator retargetability. In: Proceedings of the Conference on Design, Automation and Test in Europe, DATE 1999. ACM, New York (1999)
9. Itoh, M., Higaki, S., Sato, J., Shiomi, A., Takeuchi, Y., Kitajima, A., Imai, M.: PEAS-III: An ASIP design environment. In: Proc. IEEE International Conference on Computer Design, VLSI in Computers & Processors (ICCD 2000), pp. 430–436 (September 2000)
10. TIE - The Fast Path to High Performance Embedded SOC Processing, `http://www.tensilica.com/products/literature-docs/white-papers/tie-the-fast-path.html`

11. Smeda, A., Oussalah, M., Khammaci, T.: MADL: Meta Architecture Description Language. In: Proceedings of the Third ACIS Int'l Conference on Software Engineering Research, Management and Applications (SERA 2005). IEEE Computer Society, Washington, DC (2005)
12. Pastel, R.: Describing vliw architectures using a domain specific language. Master's thesis, Michigan Technological University (2001)
13. ArchC - The Architecture Description Language, `http://archc.sourceforge.net/`
14. Fischer, D., Teich, J., Weper, R., Kastens, U., Thies, M.: Design space characterization for architecture/compiler co-exploration. In: Proceedings of the 2001 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES 2001), pp. 108–115. ACM, New York (2001)
15. Gorjiara, B., Reshadi, M., Chandraiah, P., Gajski, D.: Generic netlist representation for system and PE level design exploration. In: Proceedings of the 4th International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS 2006), pp. 282–287. ACM, New York (2006)
16. Siska, C.: A processor description language supporting retargetable multi-pipeline dsp program development tools. In: Proc. ISSS (December 1998)

The Design of the ROHC Header Compression Accelerator

Mengmeng Yan[*] and Shengbing Zhang

Dept. of Computer Science and Technology,
Northwestern Polytechnical University
Xi'an, P.R. China
yanmeng_meng@163.com

Abstract. ROHC (Robust Header Compression) packet header compression protocol could reduce the extra overhead, which introduced by the packetizing of the Internet transport protocol, and utilize the wireless bandwidth more effectively, so it is widely used. Previous studies are mainly focused on the software implementation and optimization of key parameters. This paper introduces the ROHC header compression scheme applied in the wireless environment, and designs the framework of ROHC header compression scheme in U-mode. The header compressor of IPv4/UDP/RTP header has also been realized according to the principle of ROHC under U-mode. The modules and the implementation of the compressor are described in this paper. The performances of ROHC header compression system is analyzed through experiments. The result shows that the hardware accelerator achieves the function of ROHC packet header compression protocol correctly, and significantly reduces the overhead of packet headers to effectively improve the link utilization; at the same time has good usability and flexibility.

Keywords: ROHC, hardware accelerator, compressor.

1 Introduction

TCP/IP framework is packetized based on every layer of the protocol stack, so a considerable part of the wireless bandwidth is used to transmit control information (i.e. packet header). These control information, which are useless to the end user, result in a low utilization of wireless channel bandwidth. Packet header will introduce redundancy, which is even more than half of the whole data packet [1]. The redundancy leads to the waste of wireless bandwidth and increases the probability of the packet to be discarded because of error. It's necessary to compress the packet header in order to improve the utilization of wireless bandwidth. The Compression of packet header also solves the problems of service quality and packets' overhead, which introduced by the real-time transmission on the wireless link.

[*] Corresponding author.

ROHC (Robust Header Compression) protocol for packet header compression aims at the wireless link environment to achieve a stable and efficient packet header compression. The studies of ROHC header compression protocol in domestic and international are as follows:

Li X.W., He X.S., and Wu X.S., etc used the ROHC in third generation mobile communication (3G) systems, such as Packet Data Convergence Protocol (PDCP) sub layer [2], mobile ad hoc network system [3] or Mobile IPv6 [4] networks, to improve the reliability of data transmission; Fracchia R. and Kim J. provided some new mechanisms to cope with ROHC to improve the ROHC performance [5], or for the dynamic adjustment of an optimistic parameter[6]; and Weng W. used hardware to implement the CRC calculation of ROHC protocol, which helps to solve the ROHC protocol hardware solutions [7]. However, as far as we know, there is no hardware accelerator that uses ROHC scheme.

The compressor in the ROHC packet header compression scheme, whose frame-work under the U mode is introduced in this paper. ROHC packet header compression protocol is applied to IPv4/UDP/RTP packet header in this framework. The compressor's function is implemented by Verilog HDL. The function of compressor is verified through simulations, and the compression efficiency is analyzed through experiments.

This paper is organized as follows. Section 2 introduces the ROHC compression protocol. Section 3 describes the structure of the compressor, including the structure of modules and the inter-module data flow diagram. The implementation of the compression module is described in Section 4. Section 5 presents the simulation results and the performance analysis. Finally, Section 6 presents our conclusion and future work.

2 ROHC Compression Protocol

When a new packet flow arrives, firstly, the compressor goes into the Initialization and Refresh (IR) state. The compressor stores the packet headers of this new stream in a new context, then divides every header into different regions (a packet header is divided into three groups: the Static Region, the Dynamic Region, and the Inferred Region). Secondly, the compressor sends the whole context of the stream to the de-compressor for several times. When the compressor is fairly confident the decompressor has received the correct context, it transits to the higher compression state and begins the sending of compressed packets. The compressor must update the content of the context, which belongs to the packet stream, after every sending, to ensure the content of the context is belonging to the latest packet header. Then the compressor determines the need for state transition according to current state. The correlation and the prediction between packets are used by the decompressor to rebuild the original packets.

2.1 Compressor States and State Transition Logic

In order to improve compression efficiency, RFC3095 defines three compressor states: initial state IR (Initialization and Refresh), the FO (First Order) state, and the SO (Second Order) state [8]. The compressor of the hardware accelerator has these three compression states too. The compressor always begins at the lowest state, and then transmits to the higher states gradually. The compressor transits to a lower state from a higher state occasionally. Fig. 1 is the state machine for the compressor in Unidirectional mode (U mode) [8]. Details of the transitions between states and compression logic are given subsequent to the figure.

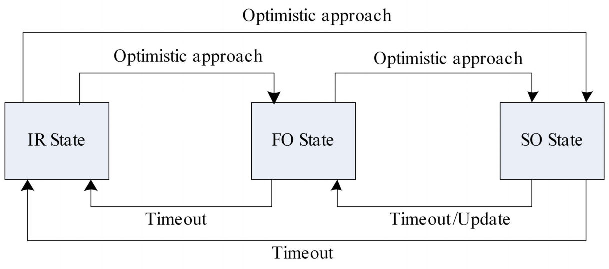


Fig. 1. Compressor transition logic

IR state: The compressor initializes and recovers the static part in the context from error in this state. In the IR state, the compressor sends an IR packet, and it will not transit to the higher compression state until it is fairly confident the decompressor has received the correct static information. An IR packet contains the complete original packet and the information of the created context, so it is longer than the original packet.

FO state: In this state, the compressor sends the irregular part of the packet flow. Compressor could transit to FO state from the IR state, or from the SO state when current packet format does not match the previous one.

SO state: This is the best compression state. The compressor just sends some additional information, such as the partially compressed Sequence Number (SN) and Context Identifier (CID), etc. If the current packet header is no longer in line with the previous format, and the context cannot be compressed independently from the previous format, the compressor transits to the FO state from the SO state.

2.2 The Packets Used by the Hardware Accelerator

IR packet: the compressor sends only IR packets when it is in the IR state. The IR packet is mainly used to inform the decompressor to initialize the context. The com-pressor creates a new context when it identifies a new packet flow, then it sends an IR packet to the decompressor. The compressor informs the de-compressor to create a corresponding decompression context for this new packet stream by this way.

IR-DYN packet: When the compressor is in the FO state, the compressor updates its compression context firstly when a changing is detected in the dynamic part of the context. Then the changed dynamic information, which organized in an IR-DYN packet, will be sent to the decompressor.

UO-0 packet: The compressor sends packets of this type when it is in the best compression state (SO).This packet type has the shortest length and is used to trans-mit the value of the encoded SN field.

UO-1 packet: The compressor sends packets of this type in the SO state to inform the decompressor when the increment of ID or TS changed.

UO-2 packet: This packet type adopts a stricter CRC check than the UO-1 packet. It transmits more dynamic information of the context to the decompressor. The com-pressor sends UO-2 packets to decompressor in the FO state.

3 The Structure of the Compressor

Firstly, the compressor of the hardware accelerator initializes all the registers, counters, and flags, etc. After the initialization, the compressor transits to IR state to wait for the data packets sent by the upper layer. The compressor stores the data packet sent by the upper layer, then divides every original header into several different regions, and saves them to the corresponding position in a temporary context. Then the compressor compares the quintuple (in ROHC, a context determined by the following five values: source IP address, destination IP address, source port, destination port and SSRC, packets have the same context belong to the same packet flow) of temporary context with all the existing contexts. The compressor determines whether the context of the current packet flow has been established or not. If an established context has the same quintuple with the current one, the compressor compresses the header sent to the decompressor according to the state of this context. If the current quintuple does not match any context that has been established, the compressor will creates a new context for the packet flow. After these done, the compressor determines whether to do a state transition. The compressor transits to a suitable compression state according to current condition.

3.1 The Structure of Modules

This design consists of four modules: the top module, the packet input module, the compression module and the packet output module. Here the role of each module is described.

Reading data packets, compressing packet header and outputting packets are implemented in the top module, so the packet input module, the compression module and the packet input module are instantiated in the top module. In order to make the packet input module, the compression module and the packet output module work in parallel, three FIFO queues: the Packet Payload (PP), the Uncompressed Packet Header (UPH), and the Compressed Packet Header (CPH), are designed in the top module. Three modules get their data from their corresponding FIFO for the purpose of parallel.

Packet Input Module. The function of the packet input module is transferring packets from upper layer to the corresponding FIFO queues. This module continuously receives data sent from the upper layer and does the following steps: divides the whole packet into two parts: one part is the header needs compression, and the other is the payload that doesn't need compression. And the compressor stores the data needn't to be compressed in the PP FIFO queue while stores the data need to be compressed in the UPH FIFO queue. As long as there is a packet need to be sent, the packet input module continues to receive data from upper layer.

Compression Module. After the packet header has been stored in the FIFO queues, the top module sends a signal to the compression module to inform it to compress the header of this packet. The compression module stores the compressed header into the CPH FIFO queue if it completes the compression.

Packet Output Module. The packet output module's function is organizing the compressed headers and their corresponding payloads together to form a compressed packet. This module constantly sends compressed packet to the decompressor at a rate of 2 Byte/cycle.

3.2 Inter-module Data Flow Diagram

The data flow between modules is shown in Fig. 2

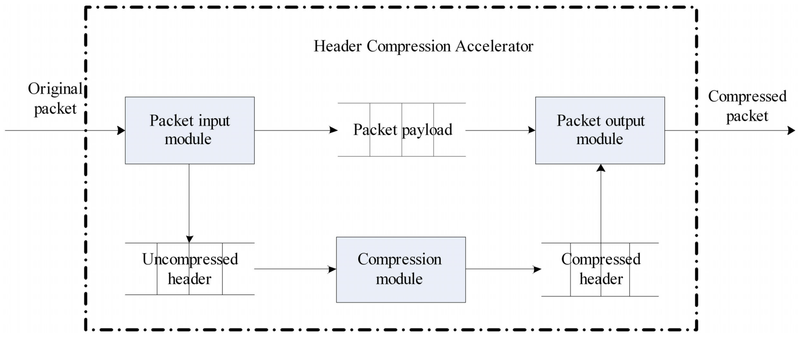


Fig. 2. Compressor transition logic

The packet input module transfers the data to the FIFO queues in the top module at a rate of 2 Byte/cycle, and it distinguishes the type of the data before storing them: stores the data in the PP FIFO queue if they are payload type; if the data are packet header type, stores them in the UPH FIFO queue.

If the compression module is idle, at the mean while the UPH FIFO queue has headers, the compression module reads a header from the UPH FIFO queue and compresses it, then stores the compressed header in the CPH FIFO after the compression, Then the compression module informs the top module it is idle again.

The packet output module determines which FIFO queue is selected to provide the data according to the current need of data. If the packet payload is needed, then the module gets data from the PP FIFO queue; if the compressed header is needed, then the module gets data from the CPH FIFO queue. This module will not be idle until there has no data to be sent to the decompressor.

4 Implementation of the Compression Module

The function of the compression module is compressing the packet header. The state machine of the compression module is shown in Fig. 3.

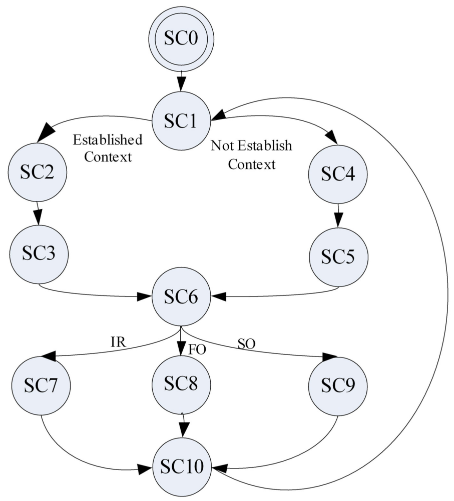


Fig. 3. Compressor transition logic

The states of the compression module are described as follows:

SC0: In this state the compressor initializes all of the registers, counters, state flags, etc.

SC1: In this state the compressor determines whether the context of current data packet has been established. If it has been established, the compressor will transmit to SC2, otherwise to SC4.

SC2: In this state the compressor compares the current context with all existing contexts, and sets the corresponding flags according to the comparison result, and then transmits to SC3.

SC3: In this state the compressor updates the context using the information of cur-rent packet, then transmits to SC6.

SC4: In this state the compressor assigns a usable CID to the current packet flow, and then transmits to SC5.

SC5: In this state the compressor establishes a new context for current packet, saves the various regions of this packet header to the corresponding position of the context, sets the corresponding flags, and then transmits to SC6.

SC6: In this state the compressor judges its current compression state, does the next steps according to the result: if current state is IR, the compressor transmits to SC7; if current state is FO, the compressor transmits to SC8; if current state is SO, the compressor transmits to SC9.

SC7/SC8/SC9: In these states the compressor determines the packet type to be sent in the IR, FO and SO state respectively then generates the corresponding compressed packet header.

SC10: In this state the compressor determines whether to do a state transition and saves the compressed packet header into the corresponding FIFO queue, then trans-mits to SC1 to wait for compressing the new packet header, which sent from the up-per layer.

5 Simulation and Performance Analysis

The header compression function is verified through simulation, and the analysis of the compression sufficiency is taken through experiments. The compression program runs on the compressor and the decompression program runs on the decompressor. The compressor compresses the packets immediately when it receives them, then the compressed packets sent to the decompressor to rebuild the original packets.

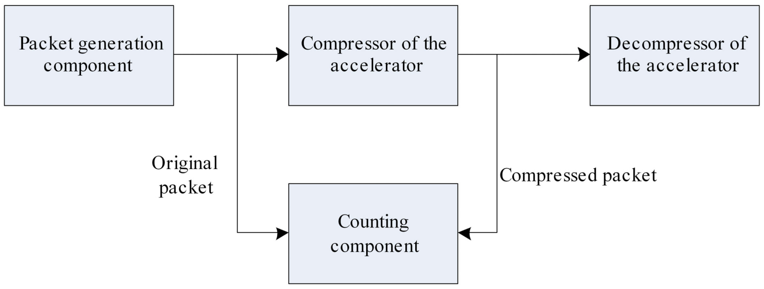

Packet generation component
Compressor of the accelerator
Decompressor of the accelerator
Original packet
Compressed packet
Counting component

Fig. 4. Compressor transition logic

Fig. 4 shows the structure used in the experiments. In this structure, the packet generation component generates packets continuously, and sends the packets to the compressor of the accelerator and the counting component. After compressing the packets, the compressor sends the compressed packets to the decompressor and the counting component. The function of the counting component is counting the number of the general packets (NGP) received by the compressor, the length of the general headers (LGH), the length of the general packets (LGP), the length of the packet (LCP) in each compression type, and

Table 1. The compression efficiency under U mode

Packet Type	NGP	LGH	LGP	LCH	LCP
IR	4	160	790	164	794
IR-DYN	0	0	0	0	0
UO-0	100	4000	20000	200	16200
UO-1	0	0	0	0	0
UO-2	24	960	4800	96	3936
Sum	128	5120	25590	460	20930

the length of the header (LCH) in each compression type. The result shown in Table 1 is provided by the counting component.

The compressor of hardware accelerator compresses the original IPv4/UDP/RTP packet header effectively, with the analysis of Table 1.

In the experiment, the length of original headers is 40 bytes, and the size of the packet is 200 bytes. The compressor could compress the header into 3.6 bytes on average. The *Brand_Gain* is defined by (1)

$$Brand\_Gain = \frac{H-h}{H+S}. \tag{1}$$

In (1) S is the size of the packet, H is the size of the original header, and h is the size of compressed header. Therefore, the *Brand_Gain* is 0.182. The *Header_Gain* of each packet is calculated by (2):

$$Header\_Gain = 1 - \frac{Payload}{Header + Payload} = \frac{Header}{Header + Payload}. \tag{2}$$

In the experiments, the *Header_Gain* is up to 0.91 when the bit error ratio of the wireless link is low. The compressor greatly reduces the header length of the packet, and saves a great many of wireless bandwidth resources. For instance, the length of IPv4/UDP/RTP packet that used in the voice communication is 40B, the length of payload is 33B. The *Brand_Gain* is up to 0.507 when these packets compressed by ROHC accelerator. It is obvious that compressed packets only use half of the original bandwidth. In our experiment, when the compressor and the decompressor run stably, the compressor could compress a 200-byte packet into only 163.5 bytes to transmit.

6 Conclusion

The ROHC header compression scheme is introduced in this paper, and the frame-work of ROHC header compression scheme in U-mode is designed, under which the compressor of the hardware accelerator has also been realized according to the principle of ROHC. A method has been programmed by Verilog HDL to compress IPv4/UDP/RTP packet headers. The analysis of the performances of ROHC header compression system is described in this paper. The result shows that ROHC header compression scheme reduces the overhead of packet header significantly (a 40-byte header could be compressed into 2 byte in the best compression state, i.e., header of the UO-0 packet), at the same time provides more effective link utilization. The accelerator works correctly and stably.

This accelerator will be developed to support more profiles, and the O-mode and R-mode of ROHC are going to be implemented in the recent future. In this frame, the accelerator can accommodates up to 16 contexts. However, the context space is not big enough to support more than 16 packet flows concurrently. So the replacement algorithm has to be accomplished to improve the performance of the accelerator.

Acknowledgments. We sincerely thank Lin Fu, Bo Liu, Yuhao Sun, and some other reviewers for their useful feedback and suggestions on earlier versions of this manuscript.

References

1. Mani, D., Gaubler, M., Frehner, C.: ROHC Implementation. University of Applied Sciences Rapperswil, USA (2005)
2. Li, X.W., Hu, X.L.: Research and realization of ROHC in PDCP sublayer in 3G system. Journal of Chongqing University of Posts and Telecommunications (Natural Secience Edition) 22, 174–178 (2010) (in Chinese)
3. He, X.S., Hu, H.Y.: Mechanism of Header Compression over AODV-based MANET. Communications Technology 41, 138–139 (2008) (in Chinese)
4. Wu, T.Y., Chao, H.C., Lo, C.H.: Mechanism of Header Compression over AODV-based MANET. Communications Technology 5, 138–139 (2008) (in Chinese)
5. Fracchia, R., Gomez, C., Tripodi, A.: R-ROHC: A Single Adaptive Solution for Header Compression. In: Proc. IEEE Symp. Vehicular Technology Conference (VTC Spring), pp. 1–5. IEEE Press, Budapest (2011)
6. Kim, J., Woo, H., Lee, H.: Dynamic Adjustment of Optimistic Parameter of ROHC for Performance Improvement. In: Proc. IEEE Symp. International Conference on Information Networking (ICOIN 2009), pp. 1–3. IEEE Press, Chiang Mai (2009)
7. Weng, W., Liu, S.P.: Hardware Implementation of the CRC Calculation in ROHC Protocol. Electronic Technology 6, 19–20 (2011) (in Chinese)
8. Bormann, C., Burmeister, C., Degermark, M.: Robust Header Compression (ROHC): Framework and four profiles: RTP, UDP, ESP and Uncompressed. Standards Track (2001)

A Hardware Implementation of Nussinov RNA Folding Algorithm

Qilong Su, Jiang Jiang, and Yuzhuo Fu

School of Microelectronics
Shanghai Jiao Tong University, Shanghai, 200240, China
{suqilong,jiangjiang,fuyuzhuo}@ic.sjtu.edu.cn

Abstract. The RNA secondary structure prediction, or RNA folding, is a compute-intensive task that is used in many bioinformatics applications. Developing the parallelism of this kind of algorithms is one of the most relevant areas in computational biology. In this paper, we propose a parallel way to implement the Nussinov algorithm on hardware. We implement our work on Xilinx FPGA, the total clock cycles to accomplish the algorithm is about half of using software in serial way, and we also partly resolve the limitation of fixed length requirement of existing hardware implementation with an efficient resource usage.

Keywords: Nussinov algorithm, vector operand, hardware implementation.

1 Introduction

Ribonucleic acid (RNA) molecule is one of the most important molecules in the biological systems. They can carry out diverse functions in living beings, plants and microorganisms. Though RNA is a single chain structure, the base will bond as pairs with another in the same chain. Under normal conditions, the RNA chain will twist, and the molecule then forms a coiled structure, this is called the secondary structure of the chain and the sequence of the bases is called the primary structure. The function of a RNA molecule is actually determined mainly by the folded shape of the secondary structure. Thus, determining the secondary structure is the key to analyze and to assign functions to RNAs.

At present, most important way to predict the secondary structure is the minimum free energy (MFE) method, and the most classical algorithm using the MFE method is the Nussinov algorithm, which was developed by Nussinov on 1978. The Nussinov algorithm uses the number of base pairs in the structure as a proxy to determine the minimum energy of the sequence. If one structure contains the maximum base pairs, it should be the one with the minimum free energy. More recent folding methods have used empirically learned models, in particular stochastic context free grammars (SCFGs) [5] , which directly assign probabilities to potential RNA structures. All these methods, however, use the dynamic programming recurrences of the same basic shape as Nussinov's algorithm [1].

Performing calculations of a RNA folding problem using present algorithms usually needs at least $O(n^3)$ complexity in time by software, which is really a time-consuming task for long RNA sequence folding or applications which will frequently launch this job. So use hardware approach to develop the parallelism in this algorithm is a very attractive idea. Many prior researchers [1] [2] have tried to solve this problem in an efficient form. In [1], authors have implemented Nussinov algorithm on FPGA, they built both GKT array and GJQ array which can operate with sequence of length 34 and 62 respectively. Document [2] presents an accelerating circuit on FPGA based on the Sankoff and Kruskal algorithm [9], which is very similar with the Nussinov algorithm. These two works both get a nice clock cycles performance on FPGA, but there circuit both requires fixed length of RNA sequence, and their modification for fitting to other lengths will be a huge work. This limits usage of their architecture in practical application.

In this paper, we propose an approach to develop the parallelism in the Nussinov algorithm by using a *vector* operand and present a sample implementation on Xilinx Virtex-6 FPGA. The result accords with theoretically expect and gives about half decrease of total cycles needed to accomplish the algorithm. Besides that, since we use block-rams instead registers to store temp data in the computing, our work partly resolves the needed of fixed length by previous hardware implementations, and that with a good hardware resource costs.

The rest of this paper is organized as follows. Section 2 shows the Nussinov algorithm and the *vector* operand version for hardware implementation. Section 3 presents the details of our sample implementation on FPGA. In section 4 we give the results of simulations and the resource usage, and the performance data. At last, in Section 5, we conclude from the results as the summarization, and propose the future work to adapt arbitrary length sequence and develop more parallelism.

2 Nussinov Algorithm

For the RNA sequence s of length N, the Nussinov algorithm finds a folding with the maximum pairs with a complexity of $O(n^3)$ in serial time. The algorithm is formally written as:

$$S(i,j) = max \begin{cases} max_{i<k<j}[S(i,k) + S(k+1,j)] \\ S(i+1, j-1) + e(i,j) \\ S(i+1,j) \\ S(i,j-1) \end{cases} \tag{1}$$

Function $\mathrm{e(i,j)}$ represents the matching condition for the pair$\mathrm{(i,j)}$. The value of $\mathrm{e(i,j)}$ is:

$$e(i,j) = \begin{cases} 1, & if\ (i,j) = (A,U) or\ (G,C) and\ j - i \geq 4 \\ 0, & else \end{cases} \tag{2}$$

Usually for implementation considerations, another simpler version of this formula which is widely used is as follows:

$$S(i,j) = max\begin{cases} max_{i\le k<j}[S(i,k) + S(k+1,j)] \\ S(i+1,j-1) + e(i,j) \end{cases} \quad (3)$$

According to the formula above, the Nussinov algorithm fills an $n \times n$ upper triangle matrix S with non-negative integers. After the matrix S is filled, $S(i,j)$ is the maximum number of pairs in subsequence $r_i r_{i+1} \dots r_j$ and in particular, $S(1,n)$ is the maximum pair number of the whole RNA sequence. The triangle matrix is shown in Fig.1.

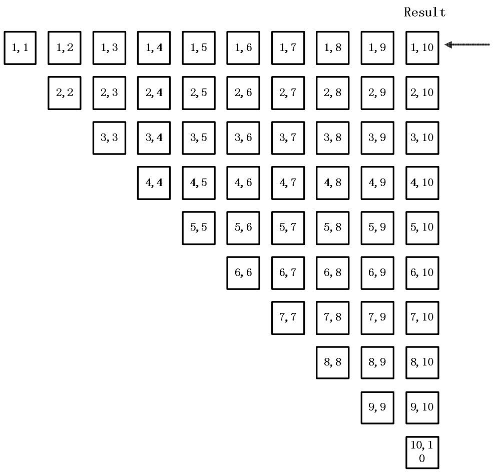


Fig. 1. Nussinov Algorithm Upper Triangular Matrix

The Nussinov algorithm based on software implementation in serial way can be described with the following pseudo-code as shown in Fig. 2.

```
for(j=0; j<N; j++)  // Diagonal Number
   for(i=0; j<N-j; i++)   // Row of Matrix S
   for(k=i; k<j; k++)  // Compute each grid
      Temp = B[i][k] + B[k+1][j];
      B[i][j] = max{Temp, B[i][k]}
```

Fig. 2. Nussinov algorithm based on software

According to the pseudo-code shown above, we can compute out the loop times of the algorithm is $\frac{1}{6}n(n+1)(n+2)$, where n is the length of the RNA sequence, and each loop performs 1 addition and 1 comparison. On hardware platform, we have adequate computing resource to perform no dependent operations at a time. So we can use a vector operand instead of the scalar operand on hardware platform to develop the parallelism within the algorithm. Based on this idea, we can describe the Nussinov algorithm based on *vector* operand with the pseudo-code shown in Fig. 3.

```
for(i=N-1; i>=0; i--)    // Rows of Matrix S
  for(j=i; j<=N; j+=2)    // Colomns of Matrix S
  for(k=j+2; k<N; k++)    // Compute each grid
     Temp1 = B[i][j] + B[j+1][k];
     Temp2 = B[i][j+1] + B[j+2][k];
    B[i][k] = max{Temp1, Temp2, B[i][k]}
```

Fig. 3. Nussinov algorithm based on vector operator

Here we use a *vector* operand of size 2 for simplicity but without loss of generality. By this way, the inner 2 recurrences reduce about half loop cycles each, so the total loop cycles should be around a half of the original scalar way. The more accurate result of this is $\frac{1}{12}n(n+1)(n+10)$ for average, which validates our approach theoretically.

3 Hardware Implementation

3.1 Overview

The key point to implement the Nussinov algorithm based on *vector* operand is the use of vector PE shown in Fig. 4(B). To achieve the maximum usage of vector PE, the data supplier must supplies 2n+1 operands each cycle to perform computing, where n is the size of the vector. In our implementation, we use vector size of 2 for simplicity.

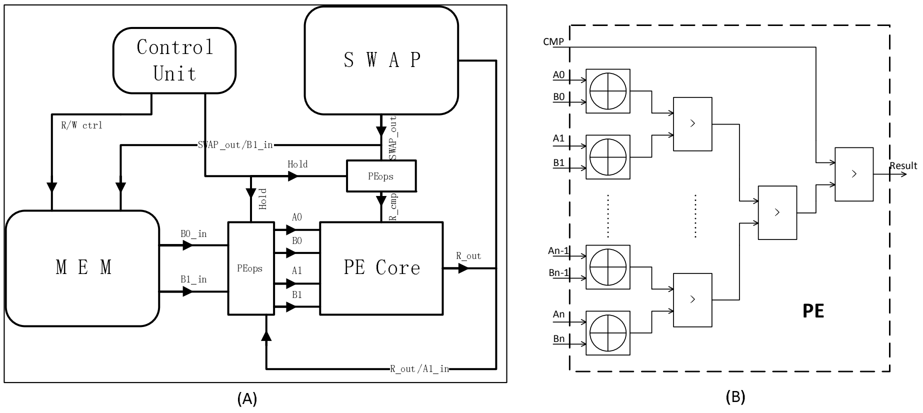


Fig. 4. (A) System structure. (B) Vector operator based PE

3.2 Data Path

Though the computing form applies to each grid of the matrix, the operands for each computation do not follow a uniformed data path. The implementation shown in document [2] proposes an approach use a register to store the value of each grid and a PE to compute each grid. The data path from grid register to PE is a static connection that can hardly be changed thus can only fit for a fixed length, which results not very good usability. In our work, we use a block-ram (MEM) used for operator supplier for PE, and also used to store the computed result for next usage. The process element (PE) gets operands from MEM, and performs appropriate operation according to the current working status. In most time, the PE need 5 operands input and compute 1 result out. To reduce the bandwidth pressure for MEM and avoid R/W conflict, we use another smaller block-ram to swap (SWAP) data. The control unit (CU) controls the stage transfer of the system, and generates appropriate address for MEM and SWAP, and controls the PE to perform right operation.

In our design, we use a dual-port block-ram resource on Xilinx Virtex-6 FPGA as MEM. Since we need perform 2 additions each cycle, the dual-port block ram could supply 2 operands at a time that meets our requirement. Review the algorithm shown in Fig. 3. B[j+1][k] and B[j+2][k] is supplied by MEM. Notice that B[i][j] and B[i][j+1] do not change along in the inner loop, so we can use 2 temp registers, REG0 and REG1, to hold these two operands within one loop. B[i][k] is stored in SWAP, each cycle SWAP read B[i][k] out for PE to perform computation, and writes the result back to SWAP. When the 2 inner loops are finished, a swap operation is performed to transfer result in SWAP to MEM, also next initial value from MEM to SWAP. The system structure and data path are shown in Fig. 4(A).

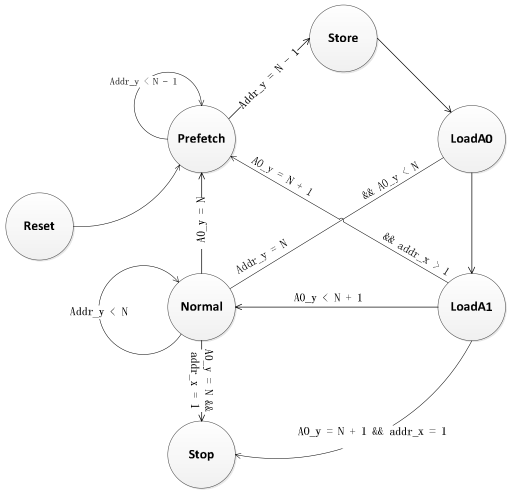


Fig. 5. Working stage transfer graph

3.3 Control Flow

The system working stage graph is shown in Fig. 5. After reset, stage is set to *Reset*, and then change to working stages. After the algorithm is accomplished, the stage changes to *Stop.* All the working stages are described as follows:

- *Prefetch*: In this stage, the swap between SWAP and MEM is performed. $e(i,j)$ is read from MEM and $S(i+1,j-1)$ is read from SWAP, PE performs $S(i+1,j-1)+e(i,j)$ as the current result of $S(i,j)$. The result of $S(i,j)$ is written into SWAP and $S(i+1,j-1)$ is written into MEM.
- *Store*: After the *Prefetch* stage is finished, there's still one result left in the SWAP. This stage use one cycle to transfer this result into MEM.
- *Load0*: Simply read current $S(i,j)$from SWAP and load it into REG0 as A0
- *Load1*: Read current $S(i,j)$ from SWAP, and PE compare larger of $S(i,j)$and A0 as current $S(i,j)$ result. Load this result into REG1 as A1and also write it back to SWAP.
- *Normal*: Performs normal computation. PE read B0 and B1 from MEM, and performs addition with A0 and A1 from REG0 and REG1 respectively. Current $S(i,j)$is read from SWAP, and PE compares the largest of $S(i,j)$ together with 2 addition results as current $S(i,j)$, and writes it back to SWAP.

4 Results

We have implemented our design in Verilog HDL, verified them in simulation with ModelSim 10.1, and synthesized it on a Xilinx Virtex-6 XC6VLX240T FPGA (hosted on the ML605 evaluation board) with Xilinx ISE 12.2. We generate 4 RNA sequences randomly with length of 10, 20, 30 and 50 respectively to test the performance of our implementation. The comparison results with software approach are shown in Table 1 and 2. In Table 1, the total clock cycles needed by our implementation and by software. Cycles needed by software are calculated according to the formulation we used in Section 2. We can see the results accord with our theoretically expect in a rough analogy. The deviation lies in that *Normal* stage is not the only stage in the system. But we can see the trend is that, as with the length increases, the effect of this factor decreases.

Table 1. Clock cycles on Hardware/Software

Sequence length	10	20	30	50
Our work(cycles)	140	955	2950	12450
Software (cycles)	220	1540	4960	22100
Speedup Ratio (%)	63	62	59	56

In Table 2 we compared the execution time of above 4 test cases, both for our hardware implementation and a C based software implementation. Even software implementation can be improved; we can see the difference in time of our hardware implementation is remarkable, since the CPU's frequency is much faster than our implementation. Besides that, compared with other implementations on hardware, our work is more flexible and could handle sequences with different length up to 200. We have shown the comparison result in Table 3. We can see that compared the implementation in Reference [2], our design could have efficient usage of logic resources but covers a large range of lengths. This partly resolves the problem for different length needed in practical application.

Table 2. Time elapsed on Hardware/Software

Sequence length	10	20	30	50
Our work on Virtex 6 at 180Mhz (ns)	770	5,252	16,225	68,475
Intel Core 2 Duo 2.93GHz CPU (ns)	2,793	13,130	37,434	147,784
Speedup Ratio	3.63	2.5	2.3	2.16

Table 3. Logic resource usage comparison with Reference [2]

	Reference[2]			Our work
Sequence length	10	20	30	From 10 to 200
Slices	367	5365	23252	492
Flip-Flops	159	1398	5427	188
Inputs/outputs	18	20	20	10
Frequency (MHz)	63.39	38.69	87.75	180.94

5 Conclusions and Future Works

In this work we have proposed a parallel approach to compute the Nussinov algorithm by using *vector* operands, and presented a sample implementation on Xilinx Virtex-6 FPGA. The result accords with theoretically expect and gives about half decrease of total cycles needed to accomplish the algorithm. Besides that, since we use block-rams instead registers to store temp data in the computing, our work partly resolve the needed of fixed length by previous hardware implementations, as well as with a good resource usage. Next stage in the future, we will focus on 2 points for further researching. One is that though we decrease a half cycles from the original, still it is a large time cost when the length grow larger. So much more parallelism should be developed to accelerate the computing. Besides that, we will further works on reduce the memory usage for the system while handling sequences with arbitrary length.

Acknowledgements. Our work is supported by the IBM Shared University Research (SUR)，System-level Virtualization Based on Reduced Hyper Multi-core Architecture, SUR201102X.

References

1. Jacob, A., Buhler, J., Chamberlain, R.D.: Accelerating Nussinov RNA secondary structure prediction with systolic arrays on FPGAs. In: Proceedings of 19th IEEE International Conference on Applications-Specific Systems, Architectures and Processors, Leuven, Belgium (2008)
2. Diaz-Perez, A., Garcia-Martinez, M.A.: FPGA Accelerator for RNA Secondary Structure Prediction. In: Proceedings of 12th Euromicro Conference on Digital System Design/Architectures, Methods and Tools (2009)

3. Nussniov, R., Pieczenik, G., Griggs, J.R., Kleitman, D.J.: Algorithms for loop matchings. SIAM Journal on Applied Mathematics 35(1), 68–82 (1978)
4. Zuker, M., Stiegler, P.: Optimal Computer Folding of Large RNA Sequences Using Thermodynamics and Auxiliary Information. Nucl. Acids Res., 133–148 (September 1981)
5. Sakakibara, Y., Brown, M., Hughey, R., Mian, I.S., Sjolander, K., Underwood, R.C., Haussler, D.: Stochastic context free grammars for tRNA modeling. Nucleic Acids Research 22, 5112–5120 (1994)
6. Chang, D.J., Kimmer, C., Ming, O.Y.: Accelerating the Nussinov RNA Folding Algorithm with CUDA/GPU. In: Signal Processing and Information Technology (2010)
7. Vidal, M.T.: Estrategias de particionamiento paralelo para el problema de RNA. Master's thesis, CINVESTAV-IPN, Mexico, D.F. (2002)
8. Cruz, G.J.: Particionamiento paralelo eficiente del algoritmo con complejidad O(n^4) para el problema de RNA. Master's thesis. CINVERSTAV-IPN, Mexico, D.F. (2005)
9. Sankoff, D., Kruskal, J.: Time warps, string edits, and macromolecules: The theory and practice of sequence comparison. Addison-Wesley, Reading (1983)
10. Durbin, R., Eddy, S., Krogh, A., Mitchison, G.: Biological sequence analysis. Cambridge University Press (1998)

A Configurable Architecture for 1-D Discrete Wavelet Transform

Qing Sun, Jiang Jiang, and Yuzhuo Fu

School of Microelectronics, SJTU, Shanghai, P.R. China
{sunqing,jiangjiang,fuyuzhuo}@ic.sjtu.edu.cn

Abstract. This work presents a novel configurable architecture for 1-dimensional discrete wavelet transform (DWT) which can be configured into different types of filters with different lengths. The architecture adopts polyphase filter structure and MAC loop based filter (MLBF) to achieve high computing performance and strong generality of the system. Loop unrolling approach is used to eliminate the data hazards caused by pipelining. The hardware usage of the configurable architecture is fixed for any kind of wavelet functions.

Keywords: 1-D DWT, Configurable circuit, VLSI, FPGA.

1 Introduction

In the last few years, discrete wavelet transform (DWT) has been used for a wide range of applications including signal analysis, image coding and compression, pattern recognition, and computer vision. The multiresolution feature and improved compression compared to existing methods such as the discrete cosine transform (DCT)-based compression schemes adopted in the old JPEG standard makes DWT a leading role in today's signal processing area.

Most major VLSI architectures to implement the DWT (both 1-D and 2-D) can be categorized into filter bank approach and lifting approach[1]. The filter bank approach (or the convolution approach) is an intuitive implementation of Mallat's multiresolution theory on digital circuit[2]. It has advantages of regular structure and good scalability, and is easy to be pipelined. The lifting approach[3] , on the other hand, uses Euclidean algorithm to decrease the computation complexity of DWT and its maximum speedup compared to the filter bank approach is 100%.

The filter bank approach had been deeply researched and widely used before the appearance of the lifting algorithm. Efforts had been made to fully exploit the parallelism of this structure, such as the recursive pyramid algorithm[4] , and the polyphase structure. The lifting structure is a polyphase structure in itself, and is much more hardware saving compared to the filter bank approach, which makes it very attractive to the users and designers. Drawbacks of the lifting structure are the relative long critical path and irregular structure. Although the critical path problem has been overcome by the Flipping method[5], poor scalability and irregular circuit structure remain to be obstacles for the researchers to exploit the generality of this structure.

Despite the fact that lots of researches has been done to optimize the VLSI architecture for specific kinds of wavelets, the generality of the circuit were rarely mentioned. In this paper, a configurable architecture for DWT with strong generality is introduced, this architecture can deal with 1-D DWT with any kind of wavelet functions. In other words, the architecture can be configured into filter banks of different lengths using fixed hardware. This configurable architecture is mainly based on the filter bank approach, because in order to adapt the architecture to every kind of wavelet, the circuit structure should be as regular as possible. Polyphase method is employed in our design to optimize the throughput, and MAC loop based filter (MLBF)s were used as the basic processing units to imitate the computing process of filters with different lengths. To optimize the critical paths and static timing performance, the architecture is pipelined and the loop unrolling method is used to solve the data hazard problems

This paper is organized as follows. In Section 2, concepts of filter bank approach and polyphase structure are introduced. Section 3 describes the structure of MLBF. The architecture of the configurable circuit is illustrated in Section 4, while Section 5 gives performance analysis and comparisons. Section 6 describes the FPGA implementation of the system and the experimental results. Section 7 concludes the paper.

2 Filter Bank Design for 1-D DWT

2.1 Filter Bank For 1-D DWT

The arithmetic computation of 1-D DWT can be expressed as filter convolutions and downsamplings as follows:

$$x_L(n) = \sum_{i=0}^{P-1} h(i) \times x(2n-i)$$

$$x_H(n) = \sum_{i=0}^{P-1} g(i) \times x(2n-i) \tag{1}$$

Where h can be considered as a lowpass filter and g as a highpass filter. Therefore, DWT can be viewed as the multiresolution decomposition of a sequence[6].

2.2 Polyphase Structure

Since half of the results of the filters would be sub-sampled, this part of "ineffective" computations should be replaced with some "effective" computations from the subsequent levels, and this is the basic idea of the recursive pyramid structure (or the folded structure)[4]. Another way to exploit that "ineffective" half of the computation is to replace the normal filters with polyphase filters, which is illustrated in Fig.1. The original non-polyphase filter is divided into an odd filter and an even filter, and the results of the polyphase filters are exactly the same as the results of the non-polyphase filters.

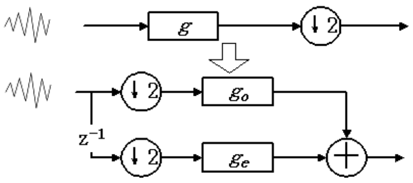


Fig. 1. Polyphase filter structure

By eliminating the "ineffective" computations, the throughput of the filter is doubled without any additional hardware.

3 MAC Loop Based Filter

3.1 Structure of MAC Loop Based Filter

The most common filter structure is shown in Fig.2.

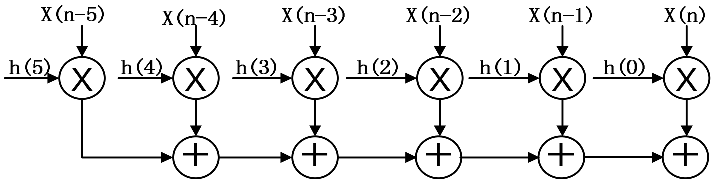


Fig. 2. Traditional convolution based filter

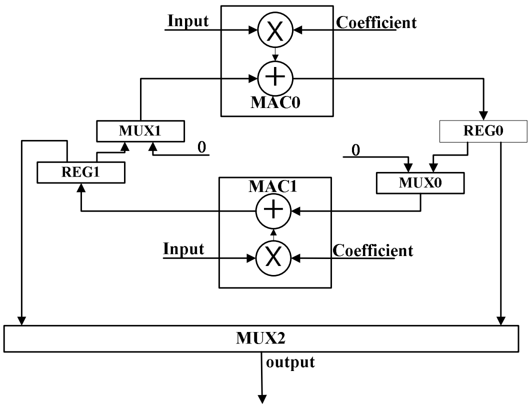


Fig. 3. Structure of a loop based MAC pair

The main drawback of the traditional convolution based filter is that the length of the filter depends on the number of the MACs. To overcome this problem, a MAC loop based filter(MLBF) is proposed to imitate the computation of an N-tap filter, where N can be any positive integer.

As it shows in Fig.3, the MLBF is composed of two MACs which are connected in a loop style. Here, we use the word "loop" to represent the physical MAC loop and the word "round" to represents the following operations:

- The multiplier in MAC0 multiplies an input data with a filter coefficient and get the result M0
- The adder in MAC0 adds M0 to the result from the previous round (if it is not the first round) or zero (if it is the first round), and gets A0.
- The multiplier in MAC1 multiplies an input data with a filter coefficient and gets the result M1.
- The adder in MAC1adds M1 to A0, and gets the result A1. A1 will be used as a input data for the next round (if it is not the last round), or be outputted as the result of the filter.

The MLBF can imitate any kind of filter just by controlling the number of rounds. For instance, to emulate the operation of a 6-tap filter, the input sequence should be multiplied and accumulated in the loop for 3 rounds.

3.2 Data Hazards in Pipelined MLBF

In order to minimize the critical path of the MLBF, the multipliers and the adders are pipelined. However, when applying the pipeline structure to the adders, some data hazards happen because the latency of the loop is linearly related to the number of stages in the pipeline structure. An example is given to illustrate the data hazard. To simplify the analysis, the pipeline stages of the adders are assumed to be one, which means the adders are pipelined by adding an output register as shown in Fig.3.

Table 1. Schedule for pipelined MLBF

T	M0	M1	A0	A1
1	*x(k-5)h(5)*			
2	*x(k-3)h(3)*	*x(k-4)h(4)*	*0+x(k-5)h(5)*	
3	*x(k-1)h(1)*	*x(k-2)h(2)*	*Data hazard*	*R0+x(k-4)h(4)*

Table 1 shows the schedule of the computations of every adder and multiplier in the loop. M0 stands for the multiplier in MAC0 and A0 stands for the adder in MAC0. R0 is the result of A0 which was ready in previous cycle.

The data hazard happens in the third cycle, because when the result of M0, which is x(k-3)h(3), is ready, it's expected to be added to x(k-5)h(5)+x(k-4)h(4). However, at the exactly same cycle, x(k-5)h(5)+x(k-4)h(4) is being computed by A1 and won't be ready until the next cycle. Such read-after-write data hazard can be overcome by loop unrolling approach.

Table 2 presents the new schedule for the pipelined MLBF after adopting the loop unrolling approach. The result of the nth point and n+1th point will be ready in cycle 7 and cycle 8.

Although the example above only discussed the situation of one-stage pipeline, the idea to solve the data hazard problems for any n-stage pipeline is quite same. The only change is to put more subsequent computations forward.

Table 2. Schedule for pipelined MLBF after adopting loop unrolling

T	M0	M1	A0	A1
1	*x(n-5)h(5)*			
2	*x(n-4)h(5)*	*x(n-4)h(4)*	*0+x(n-5)h(5)*	
3	*x(n-3)h(3)*	*x(n-3)h(4)*	*0+x(n-4)h(5)*	*R0+x(n-4)h(4)*
4	*x(n-2)h(3)*	*x(n-2)h(2)*	*R1+x(n-3)h(3)*	*R0+x(n-3)h(4)*
5	*x(n-1)h(1)*	*x(n-1)h(2)*	*R1+x(n-2)h(3)*	*R0+x(n-2)h(2)*
6	*x(n)h(1)*	*x(n)h(0)*	*R1+x(n-1)h(1)*	*R0+x(n-1)h(2)*
7		*x(n+1)h(0)*	*R1+x(n)h(1)*	*R0+x(n)h(0)*
8				*R0+x(n+1)h(0)*

4 Configurable Architecture for 1-D DWT Based on MLBF

Combining pipelined MLBF with polyphase approach, a novel configurable architecture for 1-D DWT is proposed. Fig.4 is a diagram of the overall architecture. CC stands for a single configurable circuit. Four pipelined MLBFs work as the filters in the filter bank. Since the MLBF can be expanded to filters of different lengths, the generality of the architecture is well guaranteed.

As described in Fig.4, the even-(odd-) filters, which are MLBF0 (MLBF1) and MLBF2 (MLBF3), share the same local memory, because the data feed for the two even-(odd-) filters are exactly the same. Therefore, the memory size requirement is same in the polyphase structure as in the non-polyphase structure. The local memories

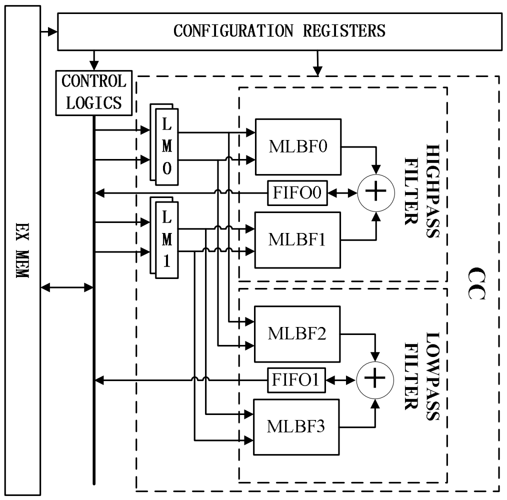


Fig. 4. Overall architecture for configurable 1-D DWT circuit

are implemented using dual-port RAMs to satisfy the bandwidth requirement. To avoid write-after-read hazards, the local memories are double buffered.

If the length of the original non-polyphase filter is odd, the length of the two derivative polyphase filters won't meet. In such circumstance, the length of the filters should be increased to an even-number by adding a 0 as the new coefficient of the filter. The drawback of the expansion of filter length is that it will cause some useless operations due to the multiplications of the inputs with 0. Configuration registers store the configuration parameters set by the users.

5 Performance Analysis And Comparison

Assuming the input size is *N*, the number of the levels of the DWT is *J*, and length of the filter bank is *L*. Theoretically, $\frac{N \times L}{2^{k-1}}$ multiplications and $\frac{N \times (L-1)}{2^{k-1}}$ accumulations should be done to accomplish the *k*th level DWT decomposition, therefore $\sum_{k=1}^{J} \frac{N \times L}{2^{k-1}}$ multiplications and $\sum_{k=1}^{J} \frac{N \times (L-1)}{2^{k-1}}$ accumulations are needed in a *J*-level DWT. Recall that the length of the original non-polyphase filter should be expanded when it's an odd number, therefore the actual length of the filter bank is $N+1$when N is odd. Since data hazards are totally avoided in the proposed architecture by employing loop unrolling and double buffering, the cycles needed for the DWT is approximately equal to

$$\frac{theoretical \quad number \quad of \quad multiplications}{number \quad of \quad multiplications \quad per \quad cycle} + cycles \quad to \quad fullfil \quad pipeline \tag{2}$$

In our design, the number of pipeline stages is 1 for the adders inside the MLBF, 2 for the multipliers inside the MLBF and 1 for the adder outside the MLBF), and the number of the multipliers is 8. Therefore, the overall number of cycles needed for the proposed architecture to finish a *J*-level DWT decomposition is $\sum_{k=1}^{J} \frac{N \times L}{2^{k+2}} + 4$, when L is even, and $\sum_{k=1}^{J} \frac{N \times (L+1)}{2^{k+2}} + 4$, when L is odd.

To fairly evaluate the performance of the MLBF based configurable architecture, we compare our design with some other related works. Since the proposed architecture utilizes fixed hardware regardless of the length of the filter bank, it's unfair to compare its throughput or number of cycles with other circuits, of which the hardware consumption is positively related to *L*. Therefore, instead of throughput, efficiency is a much more appropriate figure of merit here. Since the multipliers are much more area hungry than the adders, we will only analyze the efficiency of the multipliers.

The efficiency of the multipliers is evaluated by the work load of every single multiplier, which is defined as follow:

$$E_{mul} = number \quad of \quad multipliers \times number \quad of \quad cycles \tag{3}$$

Since the theoretical number of multiplications is fixed for a given DWT, which is $\sum_{k=1}^{J}\frac{N\times L}{2^{k-1}}$, the difference between the actual work load and the theoretical number is the number of redundant computations.

Table 3. Comparison with related works

Architecture	N_{MUL}	N_{CLK}	E_{MUL}
Parallel[7]	$2L$	$N+JL$	$2LN+2JL^2$
Systolic[8]	L	$2N+2JL$	$2LN+2JL^2$
Pipelined[9]	$\sum_{k=1}^{J}\frac{L}{2^{k-2}}<4L$	$N/2+JL/2$	$\sum_{k=1}^{J}\frac{LN+JL^2}{2^{k-1}}<2(LN+JL^2)$
DRU[10]	$\sum_{k=1}^{J}\frac{L}{2^{k-1}}<2L$	$N+2J$	$\sum_{k=1}^{J}\frac{LN+2JL}{2^{k-1}}<2LN+4JL$
[11]	$2L$	$N+J$	$2LN+2LJ$
Proposed (L is odd)	8	$\sum_{k=1}^{J}\frac{N\times(L+1)}{2^{k+2}}+4$	$\sum_{k=1}^{J}\frac{N\times(L+1)}{2^{k-1}}+32<2(L+1)N+32$
Proposed (L is even)	8	$\sum_{k=1}^{J}\frac{N\times L}{2^{k+2}}+4$	$\sum_{k=1}^{J}\frac{N\times L}{2^{k-1}}+32<2LN+32$

Table 3 gives the comparisons between the proposed architecture and the architectures introduced in other papers. When L is even, the efficiency of the multipliers of the proposed architecture is better than that of the other 5 architectures in most cases. When L is odd, the efficiency of our architecture deteriorates because of the redundant computations brought by the expanding of the filter bank. However, these redundancies are well paid off by the generality and configurability of the architecture. Fig.5 plots the work load of the multipliers in every structure as functions of L and N when J is 4. To ensure the clearance of the figure, only the lines of the Systolic and Pipeline will be plotted along with the proposed architecture.

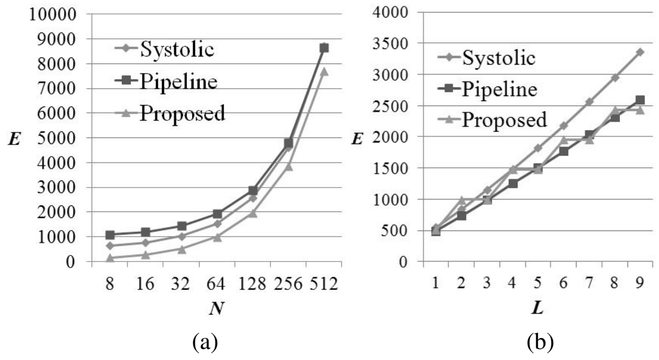


Fig. 5. (a) E versus N (L=8) (b) E versus L (N=128)

6 FPGA Implementation and Experimental Results

6.1 FPGA Implementation

To verify the functionality of our design, we implemented a configurable 1-D DWT circuit on ML605 development board. The Virtex-6 ML605 evaluation board contains a Virtex-6 LX240T FPGA device. Ethernet interface and DDR3 SDRAM are also available on the board. The bit-widths of the circuit is 16 bits and the depth of the coefficient registers in the configuration registers is 32, which means the upper limit of the filter bank that the circuit can be configured into is 32.

Fig.6 give the overview of the test system implemented on FPGA

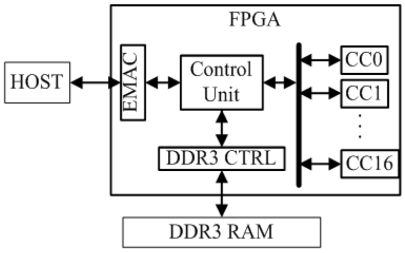


Fig. 6. Test system implemented on FPGA

Host and Interface. The host is implemented with C code on a PC to provide test data to and receive test result from the FPGA board. Communication between host PC and FPGA board is through 1 Gigabit Ethernet. The Ethernet interface is implemented by utilizing the on-chip embedded tri-mode Ethernet MAC (EMAC). A DDR3 interface was generated by Core Generator tool in the Xilinx ISE tool chain. The 16-bit 200MHz DDR3 on the development board is used as the external memory of the system to buffer the data fed by the host PC. The bandwidth of 16-bit 200MHz DDR3 is up to 6.4GB/s.

Control Unit. The control unit was built to control the data flow among host, DDR3 RAM and CCs. At the beginning of the experiment, the control unit reads the source data from the FIFO inside the EMAC and stores it in the DDR3 RAM. Then, the control unit works as a bridge between the DDR3 RAM and the local memories inside the CCs.

Table 4. Resource usage of XC6VLX240T FPGA for a single CC(configurable circuit)

Resource	Number used	Total number available	Percentage used
Slice Registers	273	301440	0%
Slice LUTs	344	150720	0%
Block RAM./FIFO	2	416	0%
DSP48Es	18	768	2%

CC. The fixed point multipliers and the adders inside the configurable circuits are constructed by using the XtremeDSP slices, and the local memories are implemented based on the Block RAM primitives. 16 CCs are included in the system.

6.2 Experimental Results

To evaluate the performace of the system, input sequences range from 2^{12} points to 2^{17} points are generated by the host. Table 5 lists the respective number of cycles needed by 16 CCs to finish a 3-level DWT. The length of the filter is 8.

Table 5. Execution timefor 1-D DWT

Size(points)	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}
Time(cycles)	518	1030	2052	3844	7428	14596

Fig.7 indicates that the execution time increase linearly with input size. The slight difference between the execution time and the theoretical time which discussed in section 5 is the time used to preload the data into the ping-pong buffers in every CU.

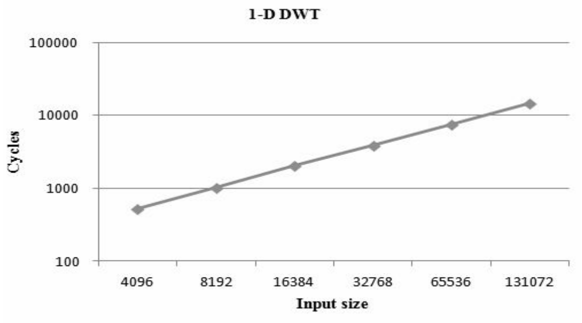


Fig. 7. Execution time in logarithmic scale

7 Conclusion

This paper proposed a novel configurable arichitecture for 1-D DWT which has the following features:

- By using the polyphase structure, the throughput of the circuit is doubled without additional hardware consumption
- The computational units of the circuit are the proposed pipelined MLBFs, which imitate the computation process of any type of filter bank with fixed hardware resource and thus give the circuit good generality and configurability
- Compared with other designs, our architecture is not only more general, but also multiplier-efficient, because data hazards are eliminated by loop unrolling and double buffering.

Acknowledgement. Our work is supported by the IBM Shared University Research (SUR).

References

1. Kotteri, K.A., et al.: A comparison of hardware implementations of the biorthogonal 9/7 DWT: convolution versus lifting. IEEE Circuits and Systems II: Express Briefs
2. Mallat, S.G.: A theory for multiresolution signal decomposition: the wavelet representation. IEEE Transactions on Pattern Analysis and Machine Intelligence 11, 674–693 (1989)
3. Daubechies, I., Sweldens, W.: Factoring wavelet transforms into lifting steps. Journal of Fourier Analysis and Applications 4, 247–269 (1998)
4. Vishwanath, M.: The recursive pyramid algorithm for the discrete wavelet transform. IEEE Transactions on Signal Processing 42, 673–676 (1994)
5. Chao-Tsung, H., et al.: Flipping structure: an efficient VLSI architecture for lifting-based discrete wavelet transform. IEEE Transactions on Signal Processing 52, 1080–1089 (2004)
6. Mallat, S.G.: Multifrequency channel decompositions of images and wavelet models. IEEE Transactions on Acoustics, Speech and Signal Processing 37, 2091–2110 (1989)
7. Chakrabarti, C., Vishwanath, M.: Efficient realizations of the discrete and continuous wavelet transforms: from single chip implementations to mapping on SIMD array computers. IEEE Trans. Signal Process. 43(3), 759–771 (1995)
8. Grzesczak, A., Mandal, M.K., Panchanathan, S.: VLSI implementation of discrete wavelet transform. IEEE Trans. Very Large Scale Integr. (VLSI) Syst. 4(4), 421–433 (1996)
9. Marino, F., Guevorkian, D., Astola, J.: Highly efficient high-speed/low-power architectures for 1-D discrete wavelet transform. IEEE Trans. Circuits Syst. II, Exp. Briefs 47(12), 1492–1502 (2000)
10. Park, T.: Efficient VLSI architecture for one-dimensional discrete wavelet transform using a scalable data recorder unit. In: Proc. ITC-CSCC, Phuket, Thailand, pp. 353–356 (July 2002)
11. Chengjun, Z., et al.: A Pipeline VLSI Architecture for High-Speed Computation of the 1-D Discrete Wavelet Transform. IEEE Transactions on Circuits and Systems I: Regular Papers 57, 2729–2740 (2010)

A Comparison of Folded Architectures for the Discrete Wavelet Transform

Jia Zhou and Jiang Jiang

School of Microelectronics,
Shanghai Jiao Tong University
clanga@ucla.edu,
jiangjiang@ic.sjtu.edu.cn

Abstract. The multi-level discrete wavelet transform (DWT) for multiresolution decomposition of a signal through the cascading of filter banks, employs a folded architecture to enhance hardware utilization. This work compares folded architectures for DWT based on three filter structures, the direct form filter, the linear systolic array, and the lifting structure. We generalize the design of these architectures in terms of DWT levels, filter taps and pipeline insertion in critical path. A figure of merit for assessing all the three architectures under different specifications is proposed. A detailed quantitative comparison among the architectures is presented with different combinations of specification. The result shows that variations in DWT levels, filter taps and pipeline insertions have different impacts on the three architectures. Overall, the folded architecture based on lifting structure gives the most desirable figure of merit and the one based on linear systolic array demonstrates the best scalability.

Keywords: VLSI, Discrete Wavelet Transform, Multiresolution Decomposition.

1 Introduction and Background

The multiresolution decomposition performed by wavelet transform offers a significant scalability in the scope of signal analysis, where either local transitions or long-term behaviors can be identified at the corresponding decomposition levels(Mallat 1989; Herley September. 1992). This approach, as depicted in Fig.1, is done by iteratively decomposing a signal at an approximation level into the orthogonal sum of a signal at a coarser approximation level and a detail signal through a pair of quadrature mirror filters, each having constant relative bandwidth, which means that the signal is break up into different octave bands. The application of this transform ranges from image compression in JPEG2000 to geophysical signal processing(Christopoulos et al. 2000; Goupillaud et al. 1984; Meyer 1992), and its implementation is made through either convolution by filter banks or through lifting structure.

As Fig.1 illustrates, due to decimation, the utilization of filters in each level of DWT keeps degrading as the level increases. The first level thus has an efficiency of fifty percent; the second has a quarter and etc.

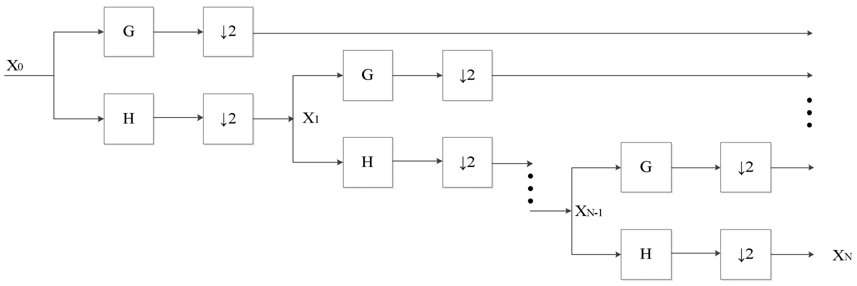


Fig. 1. A multiresolution decomposition by an N-level DWT

In an N-level wavelet, during the same time period the hardware utilization sum-up of the same G and H quadrature mirror filter pair is

$$\text{Hardware Utilization} = \frac{1}{2} + \frac{1}{4} + \cdots + \frac{1}{2^N} = 1 - \frac{1}{2^N} \tag{1}$$

It is upper-bounded by one. This implies that the maximum number of quadrature mirror pairs needed in a one-dimensional wavelet of arbitrary levels is one as long as the computations of different levels are carefully arranged, known as the Recursive Pyramid Algorithm that schedules a computation as soon as possible(Vishwanath 1994). The process of executing all operations in one processing unit is referred to as folding (Parhi 1992; Parhi and Nishitani 1993). Two folded architectures for multi-level DWT based on convolution have been proposed in (Parhi and Nishitani 1993) and (Vishwanath et al. 1995), which incorporates direct form filter and linear systolic array for a three-level DWT, respectively.

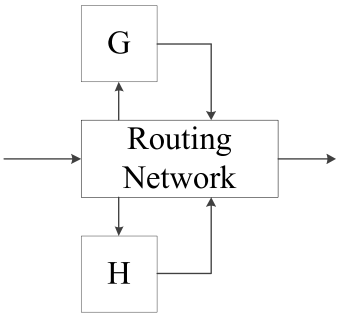


Fig. 2. Folded architectures for multi-level DWT

In this paper, we design and evaluate fixed-point implementations of multi-level DWT based on the direct form filter, the linear systolic array and the lifting structure proposed in (Sweldens 1996). For all designs, pipeline stages for critical path, filter taps and DWT levels are generally assumed and a figure of merit for assessment is proposed for all architectures to reveal the corresponding impact of different specifications on the performance of the three architectures. The structures of each filter are depicted in Fig. 3.

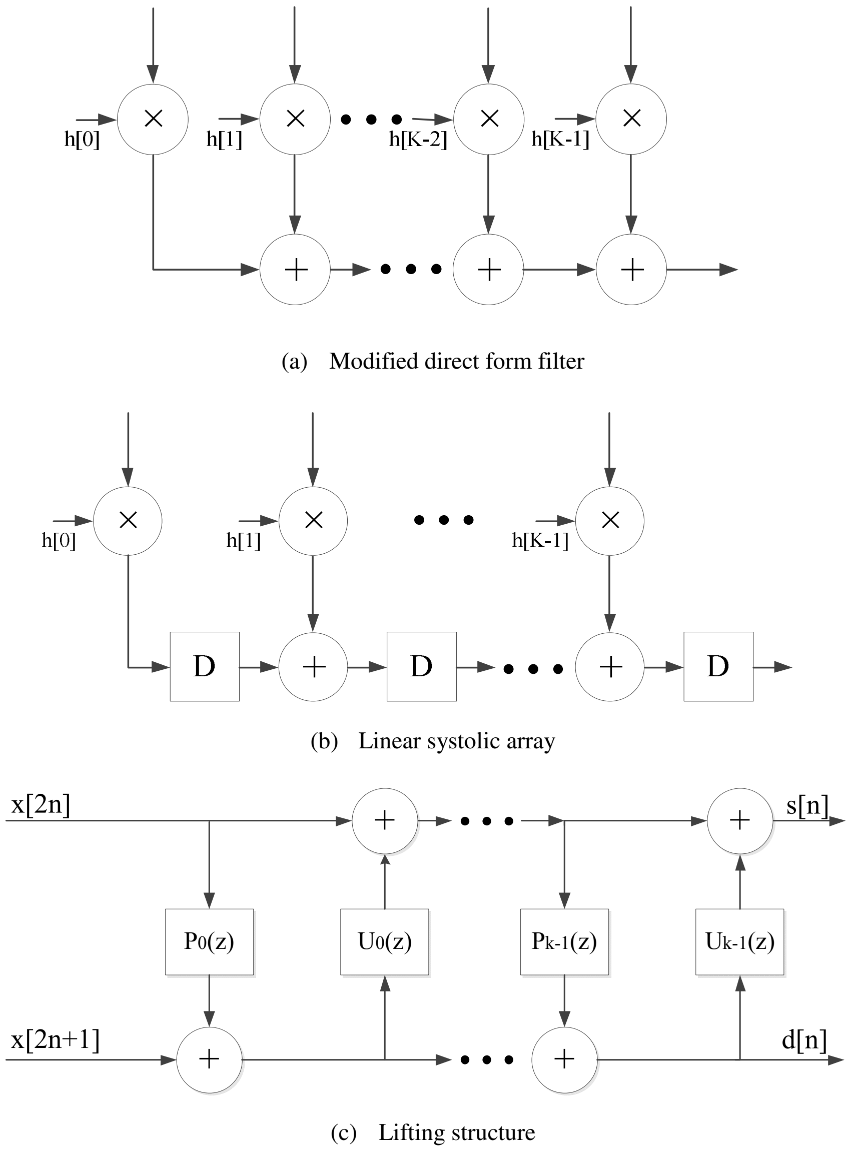


Fig. 3. The structures of the filter utilized in multi-level DWT

2 Generalized Folded Architectures and the Evaluation Method

2.1 Figure of Merit

The comprehensive evaluation of architectures for DWT mentioned above is often troubled with difficulties in quantitative analysis because of the preferences in design concerns and objective applications. Thus the evaluation usually involves too many qualitative terms that can't provide accurate design insights for scalability in system specifications, which reasonably include hardware parameters like DWT levels, filter taps, pipeline stages, or implementation cost like area and power.

We propose a figure of merit that defines the desirability of generalized architectures for multi-level DWT under different combinations of system specifications for three architectures, three folded architectures based on the modified direct form filter, the linear systolic array, and the lifting structure.

For an architecture that computes up to Nth octave of an sequence of 2^N discrete values, we define the figure of merit (FOM), as

$$\mathrm{FOM} = \frac{\text{Number of multipliers} \times \text{Latency for finishing all computations}}{\text{Number of pipeline stages of the multiplier} \times 2^N} \tag{2}$$

The evaluation principle is that smaller value of FOM is more desirable.

In this equation, the first multiplier appearing at the numerator is the number of multipliers, which represents the resource of computation processors and the cost of area and power since multipliers are much more area hungry than any other components in all architectures, and therefore they will consume most of the power.

The second term in the numerator, the latency for finishing all computations is an indicator for processing capability of completing the 2^N-point calculation. A smaller latency means that the computation capability of the architecture is better. The trade-off between latency and number of multipliers is conspicuous, as more multipliers will roughly bring shorter processing latency. The latency is strongly affected by the stages, or levels of discrete wavelet transform, and thus it should be normalized by 2^N, which is the period of optimal latency for the whole computation.

The final term appearing as the denominator, number of pipeline stages of the multiplier, is a representation of critical path and potential throughput of the architecture. Since the critical path consists mainly of a multiplier's processing time, and as a result the pipelined multiplier will enhance the throughput of the system by a factor of number of pipelines. The trade-off between pipelined multipliers and latency lies in the fact that the pipeline of multiplier will lead to more cycles of latency, and due to the data-correlation and causality between the input sequences of a discrete wavelet transform, the pipeline might be forced to stall or filled with bubbles in order to wait for the proper input data.

In a word, a smaller figure of merit means 1) smaller area and power cost under the roughly equivalent processing capability, or 2) higher computation capability with close cost of area and power, including either fewer cycles for finishing processing and higher throughput situations.

2.2 The Folded Architecture Based on the Modified Direct Form Filter

The design of this architecture consists of two parts as depicted in Fig. 2, the routing network and the modified direct form filter as illustrated in Fig. 4. The filter is initially pipelined after the cut set of all multipliers, and thus the critical path is apparently the delay of a multiplier. The pipeline stages inserted in the multipliers other than the initial design, is denoted as M as introduced above. The filter tap parameter K directly decides the number of multipliers. Whereas, the DWT level N is introduced and expressed in the schedule of filter computation, where 2^N-point computation should be arranged for properly to maximize pipeline utilization. As an example in Table 1, it is apparent that the computations of different levels, i.e. $x_1(n)$, $x_2(n)$, $x_3(n)$, are interleaved, and the pipeline is nearly full. For a detailed design procedure for this example, the reader are referred to (Parhi and Nishitani 1993). The total time required to finish the computation is 11, denoted as latency.

We then generalize the design. Assume an N-level folded architecture based on the modified direct form filter with K taps. The architecture has $2K$ multipliers with M pipeline stages for the multiplier, and the level of the discrete wavelet transform is$N$. Therefore the latency for computing all the output variables is

$$\mathrm{M} + 2^{\mathrm{N}} + \mathrm{M} + 1 \tag{3}$$

This latency is obtained by the facts that the amount of computation, 2^Npoints, is fixed, and the pipeline of the filter is mostly occupied despite few blanks at the starting period, denoted as M, and the time to compute the last point, denoted as the pipeline length, M+1.

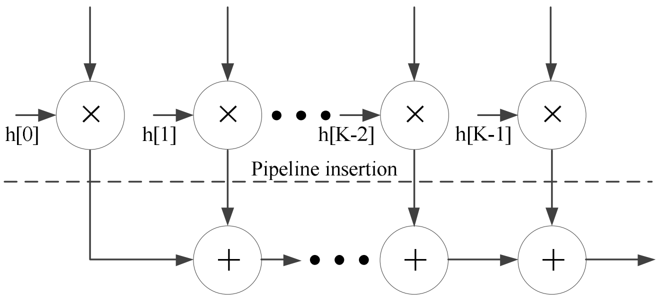


Fig. 4. A pipelined design of a four-tap modified direct form filter

Table 1. Computation schedule for a three-level DWT with a four-tap filter and one multiplier pipeline stage (N=3, K=4, M=1)

Cycle	Filter Schedule	Filter Output
0	$x_1(0)$	
1	-	
2	$x_1(2)$	$x_1(0)$
3	$x_2(0)$	-
4	$x_1(4)$	$x_1(2)$
5	$x_3(0)$	$x_2(0)$
6	$x_1(6)$	$x_1(4)$
7	$x_2(4)$	$x_3(0)$
8		$x_1(6)$
9		$x_2(4)$

Then the generalized form of figure of merit for the folded architecture based on poly-phase filter is

$$\mathrm{FOM} = \frac{2\mathrm{K}\times[\mathrm{M}+2^{\mathrm{N}}+\mathrm{M}+1]}{\mathrm{M}\times 2^{\mathrm{N}}} \qquad (4)$$

2.3 The Folded Architecture Based on the Linear Systolic Array

The design of this architecture also consists of two parts as depicted in Fig. 2, the routing network and the linear systolic array as illustrated in Fig. 3. The filter is initially pipelined by processing element, i.e. the multiply-accumulate element, and thus the critical path is mostly the delay of a multiplier. The pipeline stages inserted in the multipliers other than the initial design, is denoted as M as introduced above. The filter tap parameter K decides the number of multipliers. For an N-level DWT, the computation is strictly scheduled according to the Recursive Pyramid Algorithm, and an example of filter utilization is offered in Table 2. It is obvious that the computations of different levels, i.e. $x_1(n)$, $x_2(n)$, $x_3(n)$, are separated in difference cells, and the pipeline is nearly full. For a detailed design procedure for this example, the readers are referred to (Vishwanath et al. 1995). The total time required to finish the computation is 8, denoted as latency.

Table 2. Snapshot of H filter in one period(K=4, M=1, N=3)

Cycle	Stage 1 - h_3	Stage 2 - h_2	Stage 3 - h_1	Stage 4 - h_0	Output variable
0	$x_0(1)$	$x_2(-8)$	$x_0(1)$	$x_1(0)$	$x_1(0)$
1	$x_1(-2)$	$x_0(2)$	$x_2(-4)$	$x_0(2)$	$x_2(0)$
2	$x_0(3)$	$x_1(0)$	$x_0(3)$	$x_2(0)$	$x_1(2)$
3	-	$x_0(4)$	$x_1(2)$	$x_0(4)$	$x_3(0)$
4	$x_0(5)$	-	$x_0(5)$	$x_1(4)$	$x_1(4)$
5	$x_1(2)$	$x_0(6)$	-	$x_0(6)$	$x_2(4)$
6	$x_0(7)$	$x_1(4)$	$x_0(7)$	-	$x_1(6)$
7	$x_2(-4)$	$x_0(8)$	$x_1(6)$	$x_0(8)$	-

Then we can generalize the design. For an *N*-level folded architecture for DWT based on the linear systolic array with K cells, it has 2K multipliers with a number of pipe line stages of multiplierM. Therefore the latency for computing all the output variables is

$$\mathrm{M}\times 2^{\mathrm{N}} \qquad (5)$$

The equation means that the output variable is computed one after another with the interval of 2^{M-1} cycles. Therefore the generalized form of figure of merit for the folded architecture based on the linear systolic array is given by

$$\mathrm{FOM} = \frac{2\mathrm{K}\times\mathrm{M}\times2^{\mathrm{N}}}{\mathrm{M}\times2^{\mathrm{N}}} = 2\mathrm{K} \tag{6}$$

2.4 The Folded Architecture Based on the Lifting Structure

Though its folded structures have been proposed in (Chung-Jr et al. 2001), the aim of it is at the scalability for most of the bi-orthogonal wavelets configuration rather than at multi-level folding. The design of the folded architecture based on lifting structure consists of two parts as depicted in Fig. 2, the routing network and the lifting structure as illustrated in Fig. 3. The filter is initially pipelined by lifting stage, and thus the critical path is mostly the delay of a multiplier. The pipeline stages inserted in the multipliers other than the initial design, is denoted as M as introduced above. The filter tap parameter K decides the number of multipliers. For an N-level DWT, the computation is scheduled according to Pyramid Algorithm without interleaving different levels of DWT.

In general, an N-level folded architecture based on lifting structure has K lifting stages with identical number of multipliers that is M pipelined. The latency of cycles for computing all the output variables is

$$2^{\mathrm{N}} - 1 + (\mathrm{M}+1)\times\mathrm{K} \tag{7}$$

The generalized expression of FOM for this architecture is

$$\mathrm{FOM} = \frac{\mathrm{K}(2^{\mathrm{N}}-1+(\mathrm{M}+1)\times\mathrm{K})}{\mathrm{M}\times2^{\mathrm{N}}} \tag{8}$$

3 Results and Analysis

We provide a table of comparison of figure of merit of the three architectures under different combinations of (K, M, N), which covers the common situations and reveals the impacts of variations of specifications on the efficiency of the architectures.

3.1 Impact of Variations on Filter Taps or Lifting Stages

From the last three rows of data comparison, it can be conclude that all the three architectures will be increasing in sizes if the taps of filters or lifting stages go up. The folded architectures based on poly phase and the linear systolic array increase linearly with K; however, the one based on lifting structure goes up in quadratic style, and therefore its application with larger K is not as desirable as the other two architectures.

Table 3. FOM of three architectures of different specificatioons

(K, M, N)	FOM of Folded architecture based on polyphase filter	FOM of Folded architecture based on linear systolic array	FOM of Folded architecture based on lifting structure
(4, 1, 3)	11	8	7.5
(4, 2, 3)	6.5	8	5.25
(4, 3, 3)	5	8	3.83
(4, 1, 2)	14	8	11
(4, 1, 4)	9.5	8	5.75
(5, 1, 3)	13.75	10	10.625
(6, 1, 3)	16.5	12	14.25
(7, 1, 3)	19.25	14	18.375

3.2 Impact of Inserting Pipeline Stages in Critical Path

From the data of the first three rows of Table 3, we can conclude that the folded architectures based on linear systolic array is not affected by the variations in pipeline insertion in critical path, while the one based on direct form will gain the most significant benefits from this approach.

The reasonable explanation for this phenomenon is that in the architecture based on linear systolic array, the data path though pipelined, is subjected to the correlation between neighboring stages, but the direct form filter and lifting structure will not suffer due to their parallelism in input scheme.

3.3 Impact of Variations on DWT Levels

Data from Table 3 indicate that the variation in DWT levels will not affect the architecture based on linear systolic array, unlike the other two. More levels will bring the other two architectures benefits of smaller figure of merit due to the availability of interleaving computations of different DWT levels.

4 Conclusion

The generalized design of three folded architectures based on the direct form filter, the linear systolic array and the lifting structure is compared through a proposed figure of merit. For the different sets of DWT levels, filter taps and pipeline insertions, the three architectures behave at different sensitivities: the architecture based on direct form filter is favorable for pipeline insertion; the architectures based on linear systolic array is not affected by the levels of DWT and pipeline insertion; the filter taps will directly affect the sizes of all architectures, and mostly the one based on lifting structure. The architecture based on linear systolic array demonstrates the best scalability as parameters shift, which is most desirable for design concerns. The architecture based on lifting structure gives the best figure of merit.

References

1. Christopoulos, C., Skodras, A., Ebrahimi, T.: The JPEG2000 still image coding system: an overview. IEEE Transactions on Consumer Electronics 46(4), 1103–1127 (2000), doi:10.1109/30.920468
2. Chung-Jr, L., Kuan-Fu, C., Hong-Hui, C., Liang-Gee, C.: Lifting based discrete wavelet transform architecture for JPEG2000. In: The 2001 IEEE International Symposium on Circuits and Systems, ISCAS 2001, May 6-9, vol. 442, pp. 445–448 (2001), doi:10.1109/iscas.2001.921103
3. Goupillaud, P., Grossmann, A., Morlet, J.: Cycle-octave and related transforms in seismic signal analysis. Geoexploration 23(1), 85–102 (1984), doi:10.1016/0016-7142(84)90025-5
4. Vetterli, M., Herley, C.: Wavelets and Filter Banks: Theory and Design. IEEE Trans., Signal Processing 40(9), 2207–2232 (1992)
5. Mallat, S.G.: Multifrequency channel decompositions of images and wavelet models. IEEE Transactions on Acoustics, Speech and Signal Processing 37(12), 2091–2110 (1989), doi:10.1109/29.45554
6. Meyer, Y.: Wavelets and Applications: Proceedings of the International Conference, Marseille, France, Masson. Recherches en mathématiques appliquées = Research notes in applied mathematics, vol. 20. Springer, Paris (1989)
7. Parhi, K.K.: Systematic synthesis of DSP data format converters using life-time analysis and forward-backward register allocation. IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing 39(7), 423–440 (1992), doi:10.1109/82.160168
8. Parhi, K.K., Nishitani, T.: VLSI architectures for discrete wavelet transforms. IEEE Transactions on Very Large Scale Integration (VLSI) Systems 1(2), 191–202 (1993), doi:10.1109/92.238416
9. Sweldens, W.: The lifting scheme: A custom-design construction of biorthogonal wavelets. Appl. Comput. Harmon. Anal. 3(2), 14 (1996)
10. Vishwanath, M.: The recursive pyramid algorithm for the discrete wavelet transform. IEEE Transactions on Signal Processing 42(3), 673–676 (1994), doi:10.1109/78.277863
11. Vishwanath, M., Owens, R.M., Irwin, M.J.: VLSI architectures for the discrete wavelet transform. IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing 42(5), 305–316 (1995), doi:10.1109/82.386170

A High Performance DSP System with Fault Tolerant for Space Missions

Kang Xia, Ao Shen, Yuzhuo Fu, Ting Liu, and Jiang Jiang

The School of Micro Electronics, Shanghai Jiao Tong University
Shanghai, 200240, China
{xiakang,shenao,fuyuzhuo,liuting,jiangjiang}@ic.sjtu.edu.cn

Abstract. Space missions are very demanding on system reliability. As the development of space-based remote sensor technologies, space missions are increasingly high required on system performance. Conventional techniques mainly focus on the system reliability, at the expense of system performance.

In this paper, a flexible, DPS-based, high-performance system is presented. The system could dynamically adapt the system's level of redundancy according to varying radiation levels. A compare-point and fast recovery mechanism is proposed to improve system performance. Besides, some design ideas and implementation methods also be mentioned. In this paper, the system performances are evaluated and analyzed. With running of the correlation function benchmark in this system, it is shown that the system provides high performances under the premise of certified reliability.

Keywords: compare-point, fast recovery, high-performance, space missions, fault tolerance.

1 Introduction

In recent years, large-scale space technology research projects have been conducted and commercially adopted by quantities of countries. With the increasing requirements of space technologies, the reliability and processing capacity of on-board processing systems become more and more critical.

Previously, most of the data collected by satellites and spacecrafts could be transmitted to ground stations without delay. However, owing to the development of space-based remote sensor technologies, more data will be collected [1].The spacecrafts will lose the ability of transmitting the data in due time. Thus, the processing capability of on-board computer must be enhanced in order to meet live transmission constraints. Additionally, as the radiation hazards in outer space are always changing during the mission, the traditional on-board systems, which are fixed and designed for the worst case, are unnecessary and they will definitely influence the system performance.

In order to solve the problems mentioned above, a High Performance Fault Tolerant System (HPFTS), which will significantly improve the system performance, is implemented and discussed in detail in this paper. The basic hardware elements of this system are one control unit and three processing units. According to the environment information, system status and application requirements, system can perform dynamic switch between redundancy mode and parallel mode. The compare-point and fast recovery mechanism will also be described in this paper. Finally, the reliability and performance of the system will be evaluated and analyzed.

2 Background and Related Works

Single Event Effects (SEEs) caused by space radiation will generate deviations in the expected functionality and performance of the on-board systems [2]. Traditional fault tolerant techniques include devices radiation-hardened technology, self-test technology, redundant technology, etc. Generally speaking, the structure of conventional redundant systems is fixed. But as the requirements for on-board systems become higher, it is urgent to enlarge the operating capability of on-board systems. Fortunately, this subject has attracted a lot of attention in large numbers of institutions, and it has already been scheduled as extensive research.

The Dependable Multiprocessor project conducted by NASA NMP ST8 is about an environmentally adaptive fault tolerant system, which ensures that the processors and the improvement in system processing capability can be effectively used and reflected [3][4]. Generally, this system implements three operation modes, and the selection of the operation mode is determined by environment, system level requirements and system health status. However, as the use of PowerPC in this system, the overall performance has become less satisfied with signal processing.

J. Yang et al. have proposed a Reconfiguration Space Information Processing Platform, the key technology of which is the system reconfiguration [5]. In order to improve the performance of computing, this information processing platform has employed high performance DSP and SRAM-based FPGAs for data processing. Its high reliability unit can dynamically reconfigure the system in a SEE event and also can reconfigure the system function by applying the signals that delivered by the ground stations.Although dynamic partial reconfiguration can improve performance of system, it is not so flexibility as dynamic switching, the performance and algorithm are closely related.

3 System Overview

The Environmentally Adaptive High-performance Fault-tolerant System (HPFTS) is an application and environment oriented on-board computer system that consists of a high reliability FPGA connected to a number of high performance COTS DSPs. The excellent application and environment oriented architectural characteristics can provide an extensible system to meet variable space requirements. In this section, a brief overview of the HPFTS will be provided.

3.1 System Units

The HPFTS involves the control unit and the processing unit, as shown in Fig. 1. The former is environment sensitive and the latter is application sensitive.

Control Unit. The high reliability FPGA plays the role of the system controller that will configure the system operation mode according to time, position, sensor signals and external commands. In case of Signal Event Effect, the control unit will detect the error, decrease the system redundant level and reset the faulty DSP at the same time. In case of operation mode conversion, the control unit will reconfigure the data pass and order the DSP to change the processing mode.

Processing Unit. The processing unit consists of high performance COTS DSPs and an independent memory. The DSP has the characteristics of high performance, low cost, market availability and radiation sensitivity. Via special fault tolerant measures, the DSPs can focus on large amounts of on-board signal processing. The DSPs are independently programed and can be custom assigned in redundant and parallel operation modes.

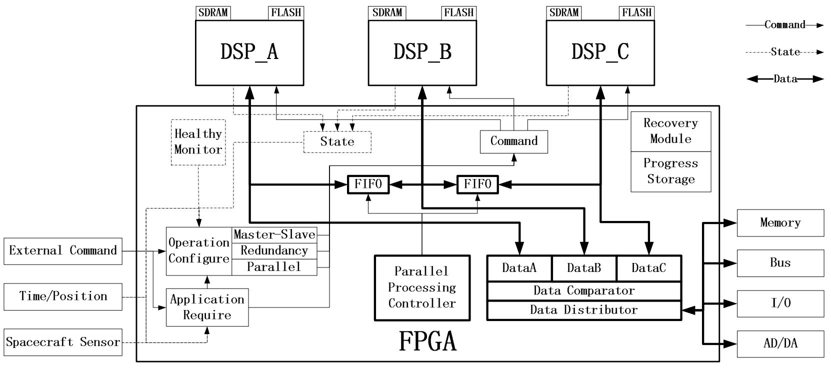


Fig. 1. The HPFTS Architecture

3.2 FPGA Logic Components

In order to accomplish the whole system redundant and parallel processing control, FPGA needs to get the processors and environment status, give accurate configure commands and control the data flow. Therefore, the relevant state, command and data logic components for FPGA have been designed.

The environment signals, including time, spacecraft position and a variety of sensors signals, determine the on-board computer system's application requirement and operating mode. Healthy monitor will get DSPs healthy state via DSP's heartbeat detection and give critical recommendation to the redundancy level.

FPGA will gather the states with external commands together and make judgments to deliver different configuration commands to DSPs. The configuration information contains master or slave role, application number, redundancy level, parallel level and parallel processing part.

The data processing paths of redundant mode and parallel mode are different. The redundant mode mainly focuses on DSPs data comparison, which is supported by Data Register for each DSP, Data Comparator and Data Distributor components. For the failure DSP's fast recovery, the processing progress will be stored in Recovery Module Component. The parallel mode mainly focuses on different DSPs data sharing and transmission, which is supported by the data FIFOs between DSPs and Parallel Processing Controller components.

4 System Implementation

The HPFTS implementation should meet the requirements of space applications that demand high reliability and high performance. So the system operation status and key mechanisms will both consider the reliability and performance.

4.1 System Operation States

In the system, the high reliability FPGA checks the environment signal and external command real-time to switch system state between standby and operation states, as shown in Fig. 2. In consideration of radiation total dose effect and low power, DSPs will be powered off in standby state. In operation state, the system will process applications and tolerate faults.

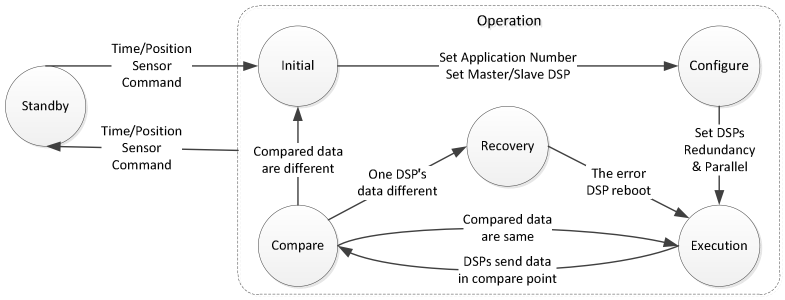


Fig. 2. System Operation States

All applications are programmed in DSPs memories. The FPGA configures applications redundancy level by activating the numbers of DSPs and telling them the same application number and the same parallel part. The FPGA configures parallel level by informing DSPs of different parallel parts for a same application.

In execution state, DSPs become the protagonist for the large amounts of data processing. The high performance feature of this system is shown in this state. Various digital signals processing algorithm for the space application can be realized in the DSPs and even in parallel processing mode.

For critical data that contains key intermediate results or system output signals, the system must make sure its correctness. Simultaneously, considering the high performance demand of the whole system, we design a faster hardware Compare-point mechanism in DSPs corresponding to the system's compare state. In compare-point, each DSP sends the data to FPGA and waits for result. FPGA uses the TMR voting mechanism to select the right data.

When a DSP's data have error, the system will decrease the redundant level. For high reliability requirement, the fault DSP needs to recover and catch up with the system progress as soon as possible. In order to fulfill this function, the Fast Recovery mechanism has been designed. In recovery state, the fault DSP needs to store its last right stage progress in FPGA and reboots itself to the initial state and load it.

4.2 Compare-Point Mechanism Design

Compare-point mechanism is the basis of fault tolerance mechanism in HPFTS. By using compare-point mechanism, the faults can be tolerated efficiently. Actually, it is not necessary to compare all the data produced in DSP because not all the data are critical and satellites are not always work in the harshest environment [6]. When processing, the HPFTS only compare the critical data which determine the applications functions and output information. Thus, it is important to ensure high reliability of these data.

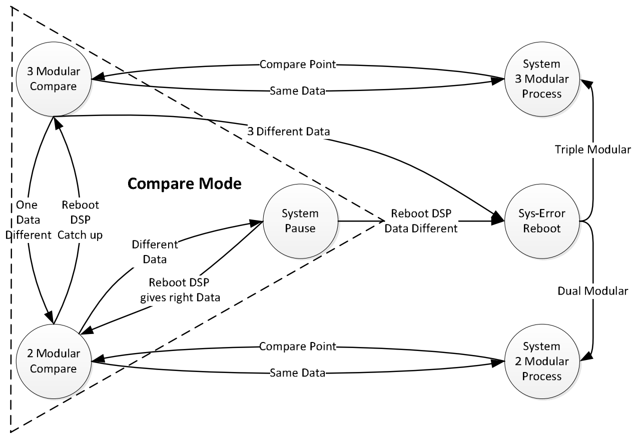


Fig. 3. Compare-point States

According to the application demands, programmers can decide what data need to be compared and how many compare-points need to be settled in program flexibility. Thus, the reliability of the system can be guaranteed and the waste of comparing time can be reduced effectively.

When a DSP runs to a compare-point, it will call the response function to convert comparing data into 32-bit and send it to FPGA, then it will wait for the feedback. After all the running DSPs' data have been received, FPGA will compare the data and send the results back to DSPs. When the system's redundancy level is 3, and the three comparing data are same, all the DSPs will be set to continue to process. If there is one data different with the other two, which means the corresponding DSP is error, this DSP will be set into recovery state by FPGA. If all the data are different, FPGA will reboot the whole system, as shown in Fig. 3.

When a DSP has been detected failure and is recovering now, FPGA will not compare its' data and only compare the data of the two remaining DSPs. If the data of the two remaining DSPs are consistent, these two DPSs will continue to run. Otherwise, the system will wait for the recovering DSP's data. Also, before catching up with the schedule of the other two DSPs, the recovering DSP is running in the fast recovery mechanism that will be mentioned in the next section.

Performance is another important consideration of the compare-point mechanism. The data's comparison will improve the reliability but decrease the performance of the system. So we need to reduce the compare-point's cycle cost as much as possible. The DSP-to-FPGA's data writing will stay several cycles. Different data writing interface configuration will affect the performance significantly, which will be shown in the results and analysis section. What's more, the FPGA should receive the compared data before the end of DSP write timing and return the results back immediately as long as the data are valid, which could definitely shorten the DSPs waiting time and improve the system performance efficiency.

4.3 Fast Recovery Mechanism Design

Based on the compare-point mechanism, the system will reduce the redundancy level and then reboot the error DSP when failure occurs. It is expected that the error DSP could recover and catch up the system as fast as possible. To improve the recovery speed, it is required to design a fast recovery mechanism to help the error DSP travel back to the track at the nearest station and pursue with the processing system. These are so-called progress store/load mechanism and station report mechanism.

Considering the save/load cycles cost and the hidden errors in registers and memories, we will save the basic configuration information and the processing progresses that have been finished in our system. For example, in the situation of video processing, we will save the number of frames that have been processed. Basically, the progress is only recorded by the DSP after every frame processing finishing. Only if the DSP has been detected faulty will the last correct record data be stored in the FPGA, which can reduce time consumption and decrease the interaction between DSPs and the FPGA.

When the DSP reboots after failure, the starting point of its application is decided by the loaded processing information instead of the original point. It can be the recent accomplished station in the system to pursue the progress or even the next station to wait for the progress arriving. Taking the video processing again as an example, when the system is processing at the N frame in the video, the recovered DSP can start at the N frame to catch up the system or start at N+1 frame to wait for the system arriving.

In rebooting process, in order to catch up with the system processing, the DSP must run faster than the system. Thus, the DSP needs to cut off the data comparing procedure and only change one output port signal at the compare-point to inform the FPGA that it has passed this point. When catching up with the system processing, the FPGA will tell the DSP to output the compared data again.

5 Results and Analysis

In this section, we describe the experimental results that present the performance and reliability of the HPFTS by the mechanisms of compare-point and fast recovery. The system will run a typical application for image/video processing with SEE soft errors.

We verified the functionality of the HPFTS on a fundamental experiment environment. The DSP is TI TMS320DM642 with 600MHz CPU clock. The FPGA is Altera Cyclone EP1C6Q240C6 with 20MHz clock. The connections between the FPGA and DSPs are I/O ports and EMIFA interfaces. The debug environment is CCS v5.0.3 and emulator is SEED XDS560PLUS. The benchmark is TI TMS320C64x IMGLIB's IMG_corr_gen function in C code [7].

5.1 Performance of Compare-Point

More compare-points mean higher reliability and longer processing time. Programmers should know the relationship between compare-points and DSP cycles to insert proper compare-points in critical applications parts.

We use the IMG_corr_gen benchmark. Calling the correlation function to process an image will produce 719 data. We call it for 10 times and compare the results after every calling. The every time comparing percent of 719 results is changing to draw the relationship of compare-points and DSP cycles.

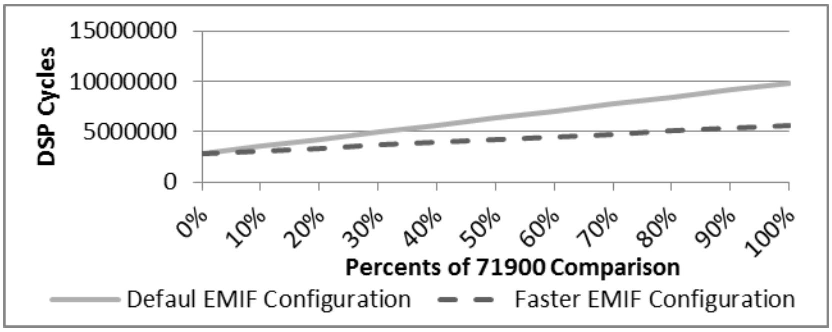


Fig. 4. Performance of Compare-points

We configure the DSPs asynchronous EMIFA in default mode and faster mode and then get the two lines in Fig. 4. The faster's EMIFA write timing is shorter by reducing the setup time, gating time and hold time to meet the frequency of the FPGA. We can see that 10 times correlation function without comparison costs about 2821320 cycles of DSP. And as comparison numbers increasing, the DSP cycles increase linearly. So each data's comparison costs 978 DSP cycles in default mode and 392 DSP cycles in faster mode.

This result indicates that the compare-point's transmission speed is the bottleneck of the system's performance. When the faster EMIFA write timing decreases 85.7% (from 168 to 24 EMIF clock) from the default mode, each compare-point cost cycles decrease 60.0%. However, the synchronous EMIF is much faster than the asynchronous EMIF, which is planned as future work.

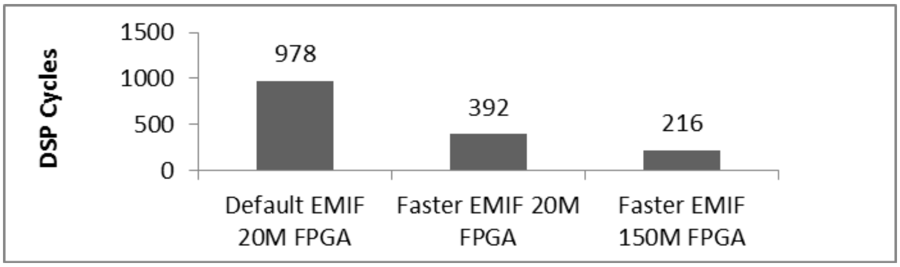


Fig. 5. Performance of different Transmission Speed and FPGA Frequency

Another bottleneck of the performance is FPGA's frequency. The current FPGA's 20MHz frequency is much slower than the DSP's. The 392 DSP cycles for each data's comparison contain 144 cycles of EMIF transmission, 120 cycles of FPGA comparison and feedback and 128 cycles of DSP pipeline lossless in waiting comparison results. A faster FPGA can need less EMIF write time and cost less time in comparison and feedback. With the using of 150MHz FPGA, the EMIF transmission can be decreased to 72 (from 24 to 6 EMIF clock) DSP cycles and the comparison and feedback can be decreased to 16 (120 cycles * 20MHz / 150MHz) DSP cycles. So we can calculate that one data's comparison costs 216 (72 + 16 + 128) DSP cycles by using 150MHz FPGA, as shown in Fig. 5.

By using hardware TMR architecture, the system's performance will be better than the software TMR. Running the IMG_corr_gen benchmark again, we assume that the 71900 result data are all critical data and need to be correct. In the HPFTS, the 71900 data will be transmitted from 3 DSPs to FPGA for comparing. In software TMR system, the correlations for 10 images will run 3 times and the results will be compared in DSP [8]. The performance result is shown in Fig. 6.

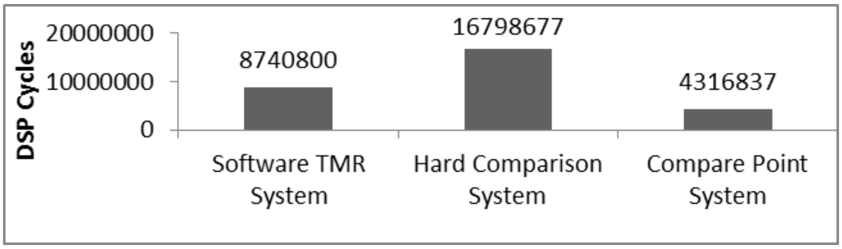


Fig. 6. Performance of different Redundancy Mechanisms

By inserting the compare-points manually, the data's comparison amount is smaller than the hard comparison system. So the system's performance will also be improved. Using the IMG_corr_gen benchmark again, there are 9 intermediate results for a signal final result. So in hard comparison system [9], there will be 71900*9 data to be compared, which are much larger than the compare-point mechanism system, as shown in Fig. 6.

5.2 Reliability of the System and Performance of Fast Recovery

For SEE in single DSP, the HPFTS can easily tolerate the fault because of the 3-DSPs redundancy architecture. By ejecting the errors in one DSP's registers and memories and even by cutting off one DSP's power, the system will execute with no pause and error.

Rarely, the system will meet the situation that two DSPs continuously have error. To verify the HPFTS's reliability, we inject 2 errors in ten pictures correlation processing. One error is in DSP_A's 10th picture and another's location changes from the DSP_B's first to the last picture. And the DSP_C is correct. So when the system runs IMG_corr_gen benchmark, DSP_B will be checked fault first and recover. Then DSP_A and DSP_C's comparing data will be different in the 10th picture. At that compare-point, they will pause and wait the recovery DSP_B to catch up and give the correct comparison data to judge which DSP is faulty.

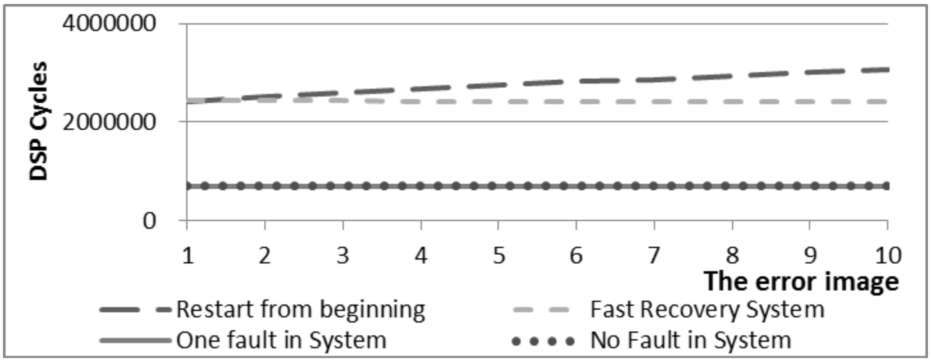


Fig. 7. System Performance for Errors in different Images

The system's pause because of two DSPs' errors will affect the performance. Our system's fast recovery mechanism will be helpful for it. For 10 images correlation, when a DSP is checked fault, it will save its processing states to the FPGA and recovery in the nearest image of the system's progress.

From the result, we can see that one fault in the system will not affect's system's normal execution. Continued two faults will make the system pause and the pause time is determined by the DSP's initial program. When the system restarts from the application's beginning, the pause time will increase by the error's location, as shown in Fig. 7. For the HPFTS, the DSP's fast recovery takes about 1.37M DSP cycles.

6 Conclusion

The system consists of three high performance COTS DSPs and a high reliability FPGA. The experimental results show HPFTS could efficiently tolerate single DSP's SEE errors with no system pause and bear continuously two DSPs' failures with a short pause. Therefore, reliability of HPFTS is trustworthy.

Moreover, many methods have been adopted to improve system performance. Compare-point mechanism not only reduces comparing amount but also shortens comparing time. Fast recovery mechanism will speed up the failure DSP's recovery and improve the overall efficiency of the system.

Future work includes implementation of parallel mode, error self-detection and comparing speed improvement. We also plan to improve the system devices and mechanisms to pursue a higher performance of the system.

Acknowledgements. Our work is supported by the Cisco Research Center Requests for Proposals (RFPs), Virtualization for NoC fault tolerance, 2011-90403 (3696).

References

1. Jacobs, A., George, A.D., Cieslewski, G.: Reconfigurable fault tolerance: A framework for environmentally adaptive fault mitigation in space. In: Field Programmable Logic and Applications (FPL), pp. 199–204 (August 2009)
2. Yousuf, S., Jacobs, A., Gordon-Ross, A.: Partially reconfigurable system-on-chips for adaptive fault tolerance. In: Field-Programmable Technology (FPT), pp. 1–8 (December 2011)
3. Ramos, J., Brenner, D.W., Galica, G.E., Walter, C.J.: Environmentally Adaptive Fault Tolerant Computing (EAFTC). In: Aerospace Conference, pp. 1–10 (March 2005)
4. Samson, J., John, R., Ramos, J., George, A.D., Patel, M., Some, R.: Technology Validation: NMP ST8 Dependable Multiprocessor Project II. In: Aerospace Conference, pp. 1–18 (March 2007)
5. Yang, J., Xing, K.F., Zhang, C.S.: Design of Reconfigurable Space Information Processing Platform. In: 7th National Information Gathering and Processing Meeting, pp. 723–726 (June 2009)
6. Abate, F., Sterpone, L., Lisboa, C.A., Carro, L., Violante, M.: New Techniques for Improving the Performance of the Lockstep Architecture for SEEs Mitigation in FPGA Embedded Processors. In: Nuclear Science, pp. 1992–2000 (August 2009)
7. Texas Instruments: TMS320C64x Image/Video Processing Library Programmer's Reference. SPRUEB9 (March 2006)
8. Yang, X.J., Gao, L.: Software Implemented Fault Tolerance Based on COTS for Space Explorations. Computer Engineering & Science 29(8), 82–87 (2007)
9. Matthew, J.W.: Using Commercial Off the Shelf (COTS) Digital Signal Processors (DSP) for Reliable Space Based Digital Signal Processing. Naval Postgraduate School (2001)

The Design and Realization of Campus Information Release Platform Based on Android Framework

Jie Wang[1], Xue Yu[1], Yu Zeng[2], and Dongri Yang[3]

[1] School of Management, Capital Normal University,Beijing 100089, China
wangjie@cnu.edu.cn
[2] Beijing Computing Center, Beijing 100094, China
[3] College of Computing & Communication Engineering,
Graduate University of the Chinese Academy of Sciences,Beijing 100049, China

Abstract. With the popularity of the mobile terminal, there appears a new trend to release all kinds of campus information by intelligent mobile terminals. The efficient, intelligent and popular features of Android smart phone platform will be combined with the campus information system to achieve the synchronization and convenience of all types of campus information release and to strengthen the communication between the various campuses of the same university. In this paper, we design and realize a campus information release platform based on Android framework. This campus information release platform can effectively reduce the complexity of the information release system and strengthen the real-time performance of information, which thereby promote the information construction of the campus.

Keywords: Android framework, Campus information release, Mobile information system.

1 Introduction

In the information society, universities have entered the era of digital information campus. , and campus information release system has become a necessary product. Android platform is an open-ended system to support a variety of scalable user experiences, with a very rich graphics system, multimedia support and powerful browser[1]. If we combine these advantages and the portability of phone, then apply them to the campus information release platform, we can not only solve the jumbled and cumbersome problem of information release platform, but also can integrate the various characteristics of universities to develop the campus information system with its own characteristics and promote the innovative construction of the campus information technology innovation[2].

Android refers to the original meaning of "robot", the name of the open source mobile operating system Google announced on November 5, 2007. Android's biggest feature is that it is an open-ended system, with a very good development and debugging environment and it supports a variety of scalable user experiences, and it has a very rich graphics system, multimedia support features and a very powerful browser [3]. Android is a Linux-based open source operating system, mainly used in portable devices. In the first quarter of 2011, Android caught up with symbian system and ranked first in the global market share for the first in the world. In February 2012, Android occupies 52.5% of the global smart phone operating system market share, and the Chinese market share is 68.4% .

In this paper, we firstly do an overall analysis of campus information release platform, and then design and realize a real campus information release platform from System functional module, system architecture and technology road map aspects. Furthermore, we also give the initial interface of our realization. This campus information release plat-form can effectively reduce the complexity of the information release system and strengthen the real-time performance of information, which thereby promote the information construction of the campus. Our research is a useful attempt and this will also be a big breakthrough in the construction of university digital information.

2 Overall Analysis of Campus Information Release Platform

In this part, we do an overall analysis of campus information release platform, which is based on the status of the our own universities: the university administrators releasing the information, teachers and students using the Android smart phone client, by mobile or other telecommunications signals to access the campus information server.

For this platform to fit the needs of students, we design a questionnaire surveying college students across the country, and the findings are more representative. According to the statistics, the proportion pie chart of the information which the students are most interested in is in figure 1-3:

For the respect of the life, the statistics result is shown in figure 1:

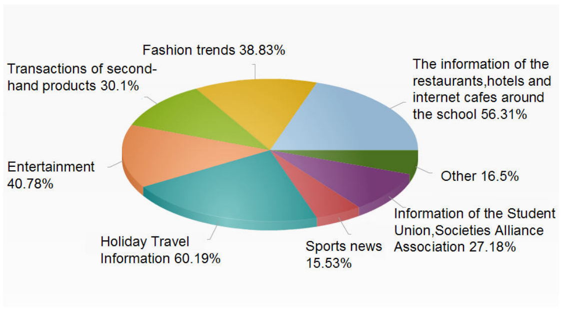


Fig. 1. The proportion pie chart of the life information

The respect of the study and work are shown in figure 2 and figure 3 respectively:

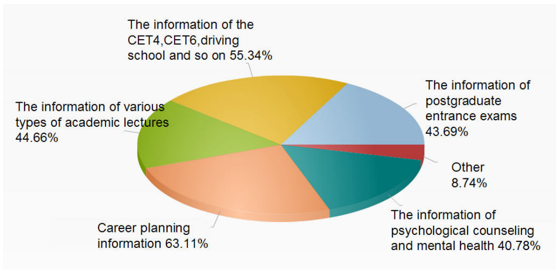


Fig. 2. The proportion pie chart of the study information

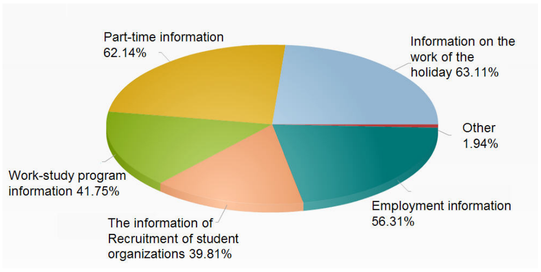


Fig. 3. The proportion pie chart of the work information

According to the statistics, about 60% of the students get access to the school information by monitor, but only 30% of them catch the school information through the campus network. More than half of the students said that they communicate little with the teachers and their seniors, and they hope to communicate more with them. In terms of idle goods, 50 percent of the students tend to put them aside. More than half of the students hope to be able to set up trading in second module in this information release platform. Most of the students said they support this platform.

In the end of the questionnaire, many students give us a lot of good suggestions on the module. According to the statistics result, the modules they are most interested are AC modules, message modules, enterprises recruitment in campus, students questions, current events, part-time information, second-hand goods transactions, campus affairs, notifications and so on. The requirements of interface are mostly clear and concise.

3 The Design and Realization of Campus Information Release Platform Based on Android Framework

3.1 System Functional Module Design

According to the statistical results in the need analysis,we design the overall system functional module and it is shown in figure 4:

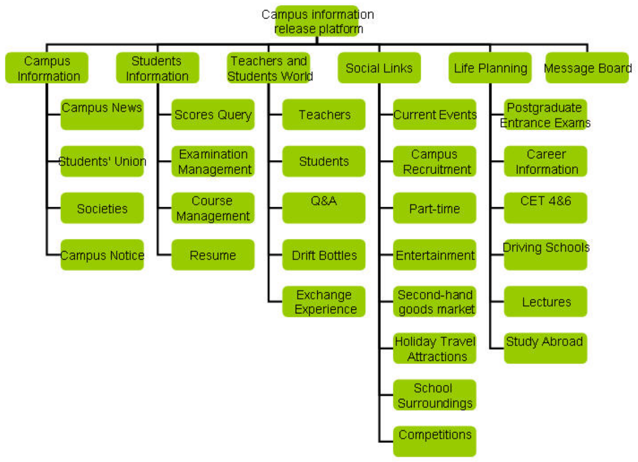


Fig. 4. The System Function Module

As shown in figure 4, the information release platform includes campus information module, students information module, teachers and students world module, social links module, life planning module and message board module.

The following describes each function module of the system in detail:

1. Campus Information module includes: Campus News—campus affairs, event information and interesting news; Students Union—students unions recruitment and activities information; Societies—the Introduction of the various communi-ties, the information of recruitment and the activities; Campus Notice— all kinds of notification information, such as holiday, course selection notification, research and reporting and so on.
2. Students Information module includes: Score Query—for students to log and query all subjects; Examination Management—for students to query recent infor-mation about the examinations and related examination registration and test center notice; Course Management—for students to log on to choose and withdraw the courses and query curriculum; Resume—to query the award, sanctions and other information obtained during the campus.
3. Teachers and Students World module includes: Teachers—teachers biographical and research information; Students— students' style, including the selection of out-standing students and the Scholarships; Q&A—it is in the form of the Forum, where teachers and students can ask questions and give answers, to enhance the communication between teachers and students at all levels; Drift Bottles—anonymous users can send information to unknown users, the users who receive the information bottles can accept or refuse; Exchange Experience—for teachers and students to show experience and to provide a reference for others, and users can also ask for help.
4. Social Links module includes: Current Events— to update community news for the students to browse; Campus Recruitment—the information of

campus recruitment , the time, the place and some tips; Part-time—part-time information; Entertainment—to update the audio and video entertainment information; Second-hand goods market—for teachers and students to publish trading information to buy or to sell; Holiday Travel Attractions—holiday travel routes, attractions and other characteristic information; Campus Surroundings—supermarkets, shopping malls, cafes and other information around the campus; Competitions—the information of the competitions for college students.

5. Life Planning module includes: Postgraduate Entrance Exams—the registration, time, counseling agencies and the various college entrance information of the Post-graduate Entrance Exams; Career Information—the information of various occupational profiles, income ranking, proportion of graduate recruitment and so on; CET 4&6—the information of registration, counseling agencies of the CET 4&6; Driving Schools—driving campus quote and enrollment information; Lectures—campus lecture information; Study Abroad—for exchange students to apply for foreign graduate students and other related information.
6. Message Board module is for the users to give comments and suggestions to the publishing platform and campus. Users can leave messages freely in this module.

The logistical departments, administrative and student groups can be arragend to manage these modules reasonably.

3.2 System Architecture Design

As is shown in figure 5, the Android client establishes a connection with the server through the GSM (Global System for Mobile Communications) base station. The switchboard is used to connect the server and database and to establish the electrical signal path. The Physical Isolation Network Gateway is used to ensure the safe and moderate data exchange in the network [4].

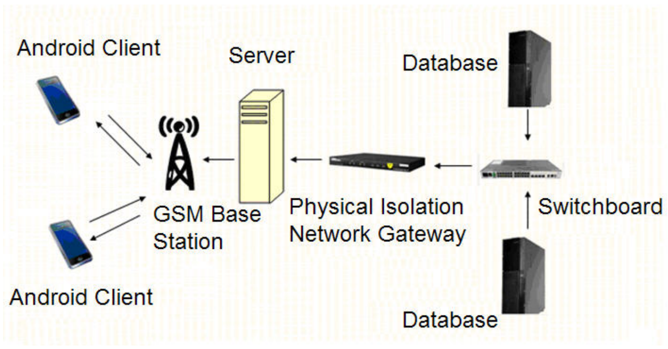


Fig. 5. The System Architecture

3.3 Technology Road Map Design

As like the realization in paper[5], we use the C / S mode, which is helpful to reduce the servers running load, to optimize the data storage management functions, and reduce the complexity of the client runs. This mode is also helpful to reduce the occupancy of mobile resources,client traffic, and it may be useful to lower problems of mobile phones [6].

As the Android client, we use Java language to develop and use the HTTP network protocol for Network communication. Data is encapsulated in XML format in the transfer process, so as to reduce the data flow and speed up response. At the same time, we use the Struts2 framework which is safer and easier to use [7].The Spring Framework is respond for the requests of customer for the Web server. Spring uses the low-intrusive design, and the code contamination is very low. It is independent from the various application servers, and can reduce the complexity of the business object to replace [8].The Hibernate framework is used to access the database, which can be more convenient for the programmer to operate the database [9].The Web Logic Server is used for the Web server, which has the leading standard, the higher scalability and the more flexible deployment [10].

We use the Oracle database to store data, which is efficient for big data and is easy to deal with data recovery problems [11].

4 User Interface Design

According to our need analysis and survey, interface design should be clear and concise. The initial interface of the client is shown in figure 6.

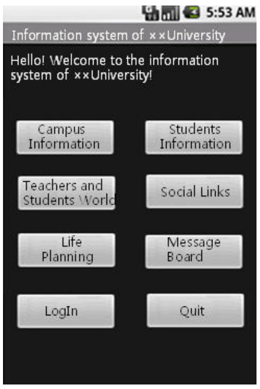

5:53 AM
Information system of ××University
Hello! Welcome to the information system of ××University!

Fig. 6. The initial interface of the system

The above six buttons in the user interface correspond to the link to the six modules, and you can click on the link to enter the lower branches of modules and perform operations in accordance with the system functional design. There are two buttons under the six buttons in the initial interface ”login” and ”exit”. They correspond to the link to the log-in screen and exit of the program. For example, click on ”Campus Information System”, there will be four branches in four button links in the middle of the interface shown in figure 7, the interface is the module, and then you can click a button which is linked to information browsing, voice mail and other operations. The ”Back” button in the bottom right corner corresponds to the link to return to the layer interface.

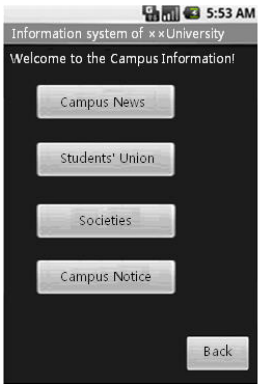

5:53 AM
Information system of ××University
Welcome to the Campus Information!
Campus News
Students' Union
Societies
Campus Notice
Back

Fig. 7. The interface after entering the module of Campus Information System

5 Conclusion

In today’s information society, the campus information system has become the main way for universities to publish information. Its contact with Android and the mobile terminals—smart phone, and the development of a more convenient real-time information dissemination platform will undoubtedly have great prospects.

In this paper, we firstly do an overall analysis of campus information release platform, and then design and realize a real campus information release platform from system functional module, system architecture and technology road map aspects. We also show the initial client interface of our realization. Our research is a useful attempt and this will also be a big breakthrough in the construction of university digital information.

References

1. Wang, S., Suo, G.: Google Android Development Guide, 2nd edn. Posts & Telecom Press (2009) (in Chinese)
2. Li, X.: Design and Implementation of the student housing system based on the Android platform. Wireless Internet Technology 1, 33–34 (2011) (in Chinese)

3. Information on Android website, http://www.android.com/
4. Information on Baike of Baidu, http://baike.baidu.com/view/1241829.html
5. Wang, C.: Campus information release system based on the Android platform. Digital Technology and Application 8, 123–124 (2010) (in Chinese)
6. Information on Baike of Baidu, http://baike.baidu.com/view/45170.html
7. Husted, T., Dumoulin, C., Franciscus, G., Winterfeldt, D.: Struts in Action. Manning Publications Corp. (2003)
8. Johnson, R., Holler, J., Arendsen, A.: Professional Java Development with the Spring Framework. Wiley Publishing (2005)
9. Gao, A., Wei, W.-X.: Application of Java data persistence with Hibernate and Struts framework. Computer Applications 12 (2005)
10. Alapati, S.: Oracle WebLogic Server 11g Administration Handbook 1st. McGraw-Hill Osborne Media (2011)
11. Oracle Database, http://www.oracle.com

A Word-Length Optimized Hardware Gaussian Random Number Generator Based on the Box-Muller Method

Yuan Li[1], Jiang Jiang[2], Minxuan Zhang[1], and Shaojun Wei[3]

[1] School of Computer
National University of Defense Technology
Changsha, Hunan, P.R. China, 410073
[2] Institute of Microelectronics
Shanghai Jiao Tong University
Shanghai, P.R. China
[3] Institute of Microelectronics
Tsinghua University
Beijing, P.R. China, 100084
csliyuan@hotmail.com, jiangjiang@ic.sjtu.edu.cn,
mxzhang@nudt.edu.cn, wsj@tsinghua.edu.cn

Abstract. In this paper, we proposed a hardware Gaussian random number generator based on the Box-Muller method. To reduce the resource complexity, an efficient word-length optimization model is proposed to find out the optimal word-lengths for signals. Experimental results show that our word-length optimized Fixed-Point generator runs as fast as 403.7 MHz on a Xilinx Virtex-6 FPGA device and is capable of generating 2 samples every clock cycle, which is 12.6 times faster compared to its corresponding dedicated software version. It uses up 442 Slices, 1517 FFs and 1517 LUTs, which is only about 1% of the device and saves almost 85% and 71% of area in comparison to the corresponding IEEE double & single Floating-Point generators, respectively. The statistical quality of the Gaussian samples produced by our design is verified by the common empirical test: the chi-square (X^2) test.

Keywords: Box-Muller Method; Hardware Gaussian Random Number Generator; FPGA; Word-Length Optimization.

1 Introduction

High quality Gaussian random numbers is essential in a large number of computationally intensive modeling and simulation applications, especially Monte-Carlo simulations. It is well known that in Monte-Carlo simulations the quality of the underlying random numbers plays a key role for the accuracy of the final computation results. Currently, due to recent advances in reconfigurable hardware in the form of Field-Programmable Gate Arrays (FPGAs), hardware-based simulations are getting increasing attention because of their huge performance advantages over traditional software-based methods [1]. Being the critical component of many scientific applications, designing a hardware Gaussian Random Number Generator (GRNG) that is capable of providing high quality Gaussian distribution sequence to fully utilize the parallel nature of hardware is also becoming more and more important in modern systems.

Despite the importance, most of the research that has been done on the Gaussian generation method concentrates on algorithms and relevant software implementations [2] [3] [5] [6]. Hardware implementations are usually not sufficient in the literature. Lee et al. [1] proposed a hardware GRNG based on the Box-Muller method. The word-length of signals in this generator is determined using mathematical analysis method. Zhang et al. [7] presented a ziggurat-based hardware structure for generating Gaussian samples. It used the Combined Tausworthe method [4] as the fundamental Uniform Random Number Generator (URNG) and evaluated the speed and size advantages on a Xilinx Virtex-2 FPGA device.

In this paper, we propose a hardware GRNG for efficient implementation of Box-Muller method. The WELL19937 algorithm [10] is adopted as the basic URNG and a simulation-based method is used for the word-length optimization. More specially, we make the following contributions.

- We design a hardware architecture for the Box-Muller method that can achieve a throughput of 2 sample per clock cycle, with modest resource overhead.
- We develop a simulation-based model to determine the optimal word-length of signals, which maximize the performance/cost efficiency of the GRNG system.
- We adopt the WELL19937 algorithm as the basic URNG in our design, which is proven to be capable of generating uniform distribution samples with high quality. Thus guaranteeing the Gaussian distribution property of the variables generated by the Gaussian system.
- We evaluate the proposed architectures using the standard test suite, the chi-square (X^2) test, and implement it on a Xilinx Virtex-6 FPGA device.

The rest of the paper is organized as follows. Section 2 gives a brief introduction of the algorithmic background of the Box-Muller method. Section 3 presents our Box-Muller-based hardware architecture, and Section 4 presents technique-specific implementations, discusses evaluations and results. Finally, we conclude this paper in Section 5.

2 The Box-Muller Method

The Box-Muller method [3] is one of the most widely used Gaussian sample generation method. It is an exact transformation method and is capable of producing a pair of Gaussian sample from a pair of Uniform random numbers via a series of transformation. The pseudo-code for implementing this method is described in Fig. 1. Where U_1, U_2 are the pair of basic uniform random samples and g_1, g_2 are the two generated Gaussian random samples. One can see that the transformation involves four elementary functions, i.e., the logarithmic, the square root, the sine and the cosine functions, and in each iteration when it is executed, two independent Gaussian samples can be generated.

3 Hardware Architecture for the Box-Muller Method

Fig. 2 illustrates our hardware structure for Box-Muller method. It consists of two main parts: the Uniform Random Number Generators (URNGs) and the elementary function approximation units.

The URNGs: One prevalent URNG is the Mersenne Twister [9], which is widely used for its high quality, long-period and high performance. However, this generator is proven to have some serious weaknesses [10]. The Well Equidistributed Long-period Linear (WELL) method [10] overcomes these drawbacks while retaining an equal period length and achieving better quality. So it is preferable for applications

```
Algorithm 1. The Box-Muller Method
1. $a \leftarrow \sqrt{-2\ln U_1}, \mathrm{b} \leftarrow 2\pi\, \mathrm{U}_2$
2.
return $\mathrm{g}_1 \leftarrow a\sin b, \mathrm{g}_2 \leftarrow a\cos b$
```

Fig. 1. The pseudo-code for implementing the Box-Muller method

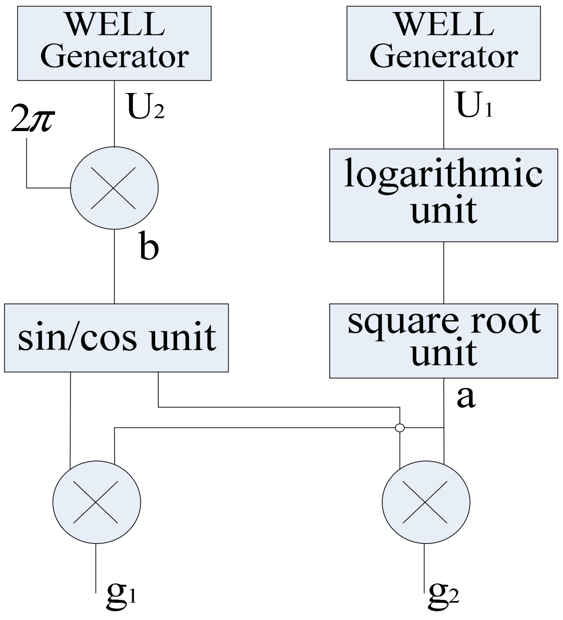


Fig. 2. The overview of our proposed Box-Muller architecture

such as ours which require samples with extremely stringent randomness. For this reason, we choose WELL as the basic URNGs in our design and adopt the WELL-based structure we proposed in [11], which is area-efficient and capable of generating one sample every clock cycle.

The Elementary Function Approximation Units: The accurate and efficient estimation of elementary functions (i.e., the logarithmic, sin/cos and square root functions) are necessary for efficient implementation of the Box-Muller generator. Consider an elementary function *f*(*x*), where *x* is in the range [a, b]. The evaluation *f*(*x*) typically consists of three steps [1] [12]: 1) range reduction: reducing x over the interval [a, b] to a more convenient y over a smaller interval [a', b']; 2) function approximation on the reduced interval, and 3) range reconstruction: expansion of the result back to the original result range.

We adopt the methods presented in [1] to estimate the elementary functions in our design (i.e., the logarithmic, the sin/cos and the square root functions). Detailed structures for these approximation units are illustrated in Fig. 3, Fig. 4 and Fig. 5, respectively. For the logarithmic function and the square root functions, a Leading Zero Detector (LZD) and some combinatorial logics such as shifter and multiplexer are used to perform the Range Reduction based on the equivalent transformations in the form of (1) and (2), respectively. Where y is the reduced argument and k is the output of the LZD unit. For the sin/cos functions, because of their periodicity, we only needs to reduce the range into [0, $\frac{\pi}{2}$].

$$\log(x) = \log(y) + k \cdot \log(2) \quad y \in [0.5,1] \tag{1}$$

$$\sqrt{x} = \begin{cases} \sqrt{y} \cdot 2^{\frac{k}{2}} & \{k \mid k = 2n \mid n \in Z\},\ y \in [0.5,1) \\ \sqrt{y} \cdot 2^{\frac{k-1}{2}} & \{k \mid k = 2n - 1 \mid n \in Z\},\ y \in [0.25,0.5) \end{cases} \tag{2}$$

We use the table-with-polynomial method [1] [12] to approximate the functions over the reduced range. The reduced intervals [0.5,1] and [0.25, 0.5] of the logarithmic and the square root functions are further split into 32 equally sized segments, and 5 bits in proper positions of the argument y are served as the indexes into the tables. The reduced range [0, $\frac{\pi}{2}$] of the sin/cos functions are split into 101 segments and the 7 most significant bits are served as the index. The tables contain all the coefficients for each interval, which are obtained via a minimax approximation that minimizes the maximum absolute error using MAPLE [1] [12]. We found that for each interval, a polynomial with a degree-1 is enough to meet the precision requirement for our system.

Word-Length Optimization Model: One critical point in designing a hardware GRNG is the transformation from Floating-Point to Fixed-Point, or word-length optimization for signals in other words. This process is performed to reduce the implementation complexity of the system. There are typically two categories of approaches for word-length optimization: the analytical method [13] [14] and the simulation-based method [15] [16]. Which adopt numerical analysis and program simulation to determine the word-length of signals, respectively. Compared to the analytical method, the simulation-based method is capable of obtaining better solutions and can be applied to more complicated system including loops. So we choose this method to perform the word-length optimization in our system.

The algorithm of the word-length optimization for our design is described in Fig. 6. A software simulator is developed, this simulator is programmed to be bit-accurate to the actual hardware realization and the word-length of signals can be flexibly configured. The starting point for the search is determined using the base point method proposed in [16] and the Sequence Search [16] is used as the search engine. In each iteration, 10^9 samples are produced for the X^2 testing [17]. The search process finished when the sequence passes the X^2 testing.

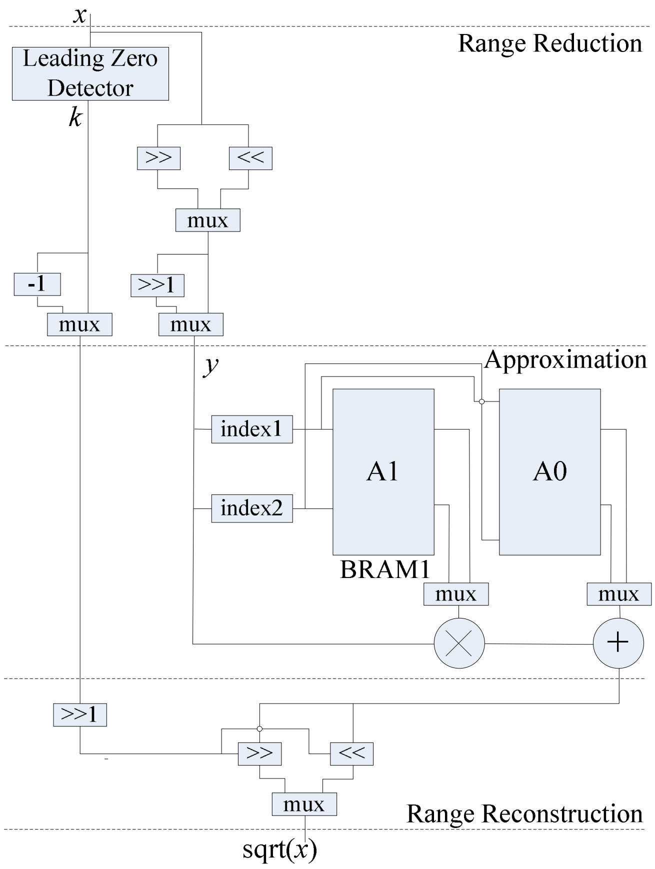


Fig. 3. Hardware structure for the approximation of the square root function

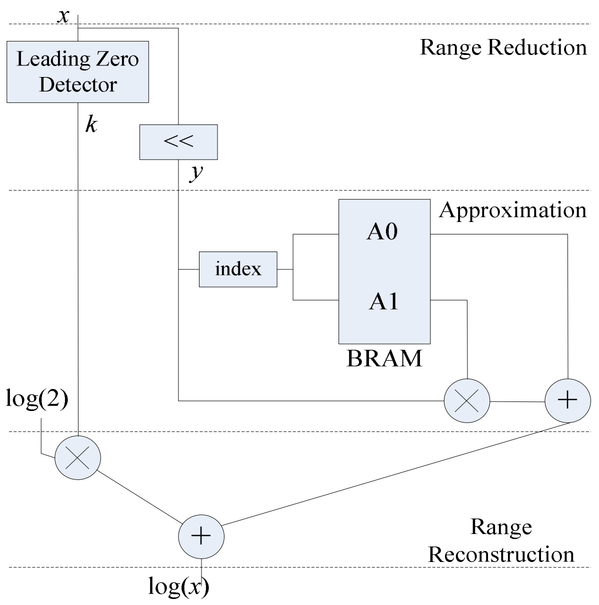


Fig. 4. Hardware structure for the approximation of the logarithmic function

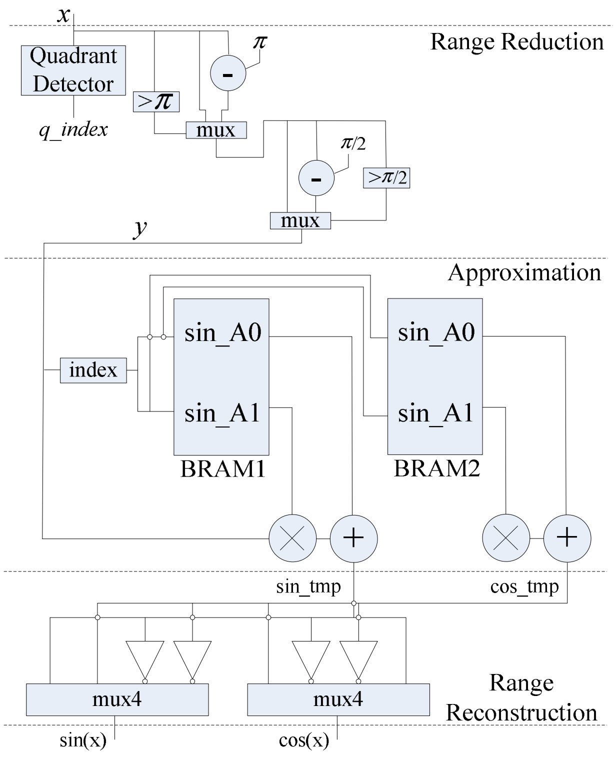


Fig. 5. Hardware structure for the approximation of the sin/cos functions

```
Algorithm 2. Word-length optimization process for Box-Muller

1. Develop a software simulator for Box-Muller in which the word-length of
   signals can be configured.
2. Determine the starting point.
3. Find the next word-length vector $W_k$ using the Sequence Search [16].
4. Update the software simulator using $W_k$, run and produce $10^9$ samples
5. if the sample sequence pass the $X^2$ testing then
6.     output $W_k$ as the optimized word-length vector, the search is finished.
7. else
8.     goto step 3.
```

Fig. 6. Algorithm of word-length optimization for Box-Muller

4 Implementation and Evaluation

We implement the architecture described in Section 3 on a Xilinx Virtex-6 XC6VLX240T FPGA device (hosted on the ML605 evaluation board). The design is described in Verilog HDL and synthesized & implemented using Xilinx ISE 12.1. The initial design is simulated in Modelsim SE 6.5 and the software simulator is used to ensure the functional correctness. Optimization techniques, such as register-retiming, are adopted to improve the clock speed of the system.

For comparison, three reference designs are implemented: one software generator and two hardware generators of Box-Muller based on the IEEE double & single Floating-Point models, respectively. All hardware designs are built on exactly the same device with the same configurations and the software design runs on a 2.67-GHz Intel Core i5 processor with 6GB DDR3 SDRAM.

The comparisons between different implementations are summarized in Table 1. One can see that our proposed generator achieved a 12.6-fold speed up compared to

Table 1. Comparison of resource usage and performance for different implemenations

Type	Fixed-Point (proposed)	Floating-Point (Single)	Floating-Point (Double)	Software Version
Platform	HW	HW	HW	SW
Slices	442	1544	2956	-
Flip-Flops	1517	4958	11048	-
LUTs	1517	5014	9595	-
BRAMs	9	9	14	-
DSPs	31	35	96	-
Freq.(MHz)	403.7	386.7	335.0	-
Thouput(M/sec)	807.4	773.4	670.0	64.0
Speedup	12.6	12.1	10.5	1.0
X^2 test of g1	pass	pass	pass	pass
X^2 test of g2	pass	pass	pass	pass

its optimized software version. Moreover, our word-length optimized Fixed-Point generator achieves a great reduction in area usage versus the corresponding Floating-Point designs while retaining the highest performance.

5 Conclusion

We present a hardware Box-Muller GRNG which achieves high performance, high quality output while incurring low resource complexity. As for the performance, it is capable of producing 807.4 Million samples every clock cycle, which is 12.6 times faster than its dedicated software version. Moreover, by applying the word-length optimization model to minimize the word-length for signals, the resource cost is really low. It uses up 442 Slices, 1517 FFs and 1517 LUTs, which is only 1% of the device and saves about 85% and 71% area compared to the corresponding IEEE double & single based generators, respectively. The Gaussian samples generated by our design successfully pass the standard statistical test suite of the X^2 test, proving the correctness of our design. We expect our Gaussian structure apply to various hardware-based simulation systems.

Acknowledgments. This work was supported by the National Natural Science Foundation of China Grant No. 60970036, No. 61103011 and No. 61076025. The National Core-High-Base Major Project of China under Grant No. 2009ZX01028-002-002.

References

[1] Lee, D., Villasenor, J.D., Luk, W., Leong, P.H.W.: A Hardware Gaussian Noise Generator Using the Box-Muller Method and Its Error Analysis. IEEE Transaction on Computers 55(6), 659–671 (2006)

[2] Bell, J.R.: Algorithm 334: Normal random deviates. Comm. ACM 11(7), 498 (1968)

[3] Box, G.E.P., Muller, M.E.: A note on the generation of random normal deviates. Annals Math. Stat. 29, 610–611 (1958)

[4] Tezuka, S., L'Ecuyer, P.: Efficient and portable combined Tausworthe random number generators. ACM Transactions on Modeling and Computer Simulation 1(2), 99–112 (1991)

[5] Brent, R.P.: Algorithm 488: A Gaussian pseudo-random number generator. Comm. ACM 17(12), 704–706 (1974)

[6] Gebhardt, F.: Generating normally distributed random numbers by inverting the normal distribution function. Math. Computation 18(86), 302–306 (1964)

[7] Zhang, G., Leong, P., Lee, D., Villasenor, J., Luk, W.: Ziggurat-Based Hardware Gaussian Random Number Generator. In: Proc. 16th IEEE Int. Conf. Field-Programmable Logic and its Applications, pp. 275–280 (2006)

[8] Lee, D., Luk, W., Villasenor, J.D., Zhang, G., Leong, P.H.W.: A hardware Gaussian noise generator using the Wallace method. IEEE Transactions on Very Large Scale Integration (VLSI) Systems 53(12), 911–920 (2007)

[9] Matsumoto, M., Nishimura, T.: Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. ACM Trans. Modeling and Computer Simulation 8(1), 3–30 (1998)

[10] Panneton, F., L'Ecuyer, P., Matsumoto, M.: Improved long-period generators based on linear recurrences modulo 2. ACM Trans. Mathematical Software 32(1), 1–16 (2006)

[11] Li, Y., Jiang, J., Chow, P., Zhang, M.: Software/Hardware Framework for Generating Parallel Long-Period Random Numbers Using the WELL Method. In: Proc. 21st Int. Conf. Field Programmable Logic and Applications, pp. 110–115 (2011)

[12] Muller, J.: Elementary Functions: Algorithms and Implementation, 2nd edn. Birkhauser, Boston (2006)

[13] de Figueiredo, L., Stolfi, J.: Self-validated numerical methods and applications. In: Brazilian Mathematics Colloquium Monograph. IMPA, Brazil (1997)

[14] Lee, D., Gaffar, A.A., Mencer, O., Luk, W.: MiniBit: Bit-Width Optimization via Affine Arithmetic. In: Proc. ACM/IEEE Design Automation Conf., pp. 837–840 (2005)

[15] Sung, W., Kurn, K.: Simulation-based word-length optimization method for fixed-point digital signal processing systems. IEEE Trans. on Signal Processing 43(12), 3087–3090 (1995)

[16] Han, K., Eo, I., Kim, K., Cho, H.: Numerical word-length optimization for CDMA demodulator. In: IEEE Int. Symposium on Circuits and Systems, vol. 4, pp. 290–293 (2001)

[17] Snedecor, G.W., Cochran, W.G.: Statistical Methods. Iowa State University Press (1989)

DAMQ Sharing Scheme for Two Physical Channels in High Performance Router

Yongqing Wang and Minxuan Zhang

School of Computer Science, National University of Defense Technology,
410073 Changsha, China
yqwang@nudt.edu.cn

Abstract. Communication in large scale interconnection networks can be made more efficient by designing faster routers, using larger buffers, larger number of ports and channels, but all of which incur significant overheads in hardware costs. In this paper we present a dual-port shared buffer scheme for router. The proposed scheme is based on a dynamically allocated multi queue and four-port Register File. Two physical channels share the same input buffer space. This can provide a larger available buffer space per channel when load is unbalanced among physical channels and virtual channels. We give the detailed organization of shared buffer and management of idle buffer. Result of simulation shows that the proposed method has similar performance using only 75% of the buffer size in traditional implementation and outperforms by 5% to 10% in throughput with the same size.

Keywords: interconnection network, DAMQ, virtual channel, input buffer.

1 Introduction

Interconnection networks are widely used for inter-processor communication in both multiprocessors and supercomputers with multi-stage networks. Virtual channel (VC) [1][2] technology is extensively used to boost performance and avoid deadlock. Each VC is realized with a pair of buffers located on adjacent communicating nodes. When the flit-consumption rate of an input buffer is lower than the flit-arrival rate, the buffer congestion occurs.

Buffer congestion can also propagate throughout the network. When the arriving flits cannot be received by a downstream router due to the lack of buffer space, the flits must continue to be buffered at the upstream router. It directly reduces the transmission speed of flits among routers. Thus, the maximum throughput of a router depends directly on how efficient the router is at storing the flits that cannot be transmitted immediately and forwarding them when the appropriate output channel is no longer busy or congested.

The easiest solution for enhancing performance such as a higher throughput is to increase the buffer size, thereby reducing the severity of congestions. However, the buffer space cannot be effectively increased if a system is constrained by an upper limit on hardware resources.

Thus, the architecture and implementation of routers for these networks, particularly their internal buffers, are critical for achieving, at low cost, the performance goals of high throughput and low-latency communication. In a wormhole-switching [3] network with several virtual channels multiplexing a physical channel, routing algorithms tend to choose one set of virtual channels over others; thus the traffic load is not evenly distributed in the entire buffer space for a physical channel. A more efficient approach to use the available buffer space is to let the virtual channels belonging to a physical channel share buffer with virtual channels of another physical channel.

2 Related Works

Tamir and Frazier [4] classified the buffered switch architectures into four major types, FIFO, SAFC, SAMQ and DAMQ. This classification is based on how the input queues are manipulated and how data is stored.

Dynamically Allocated Multi-Queue (DAMQ) buffer [4] was proposed as a unified and dynamically allocated buffer structure. More precisely, within each buffer there are separate FIFO queues of packets destined for each output channel. Every read and write operation needs three cycles to complete, which might be excessive for high speed router. To enable intra-channel buffer sharing, DAMQ with self-compacting buffer was introduced [5] [6].A disadvantage of their approach is specific buffer customization.

A dynamic VC architecture was proposed by Lai et al. [2] to solve the head of line problem. However, their proposed architecture could not effectively use all the unused buffers of other input channels.

A Reliability Aware Virtual channel (RAVC) NoC router micro-architecture [7] was proposed. In particular, in the case of failure in routers, the virtual channels of routers surrounding the faulty routers can be totally recaptured and reassigned to other input ports.

The DAMQ-SHARED buffer that combined the buffer for virtual channels from two different physical channels was proposed by Liu [8]. It is also based on with self-compacting buffer. Though the DAMQ-SHARED buffer design can be used to enhance the utilization of inter-channel buffers, the design complexity is also increased due to the complex control logic of the DAMQ buffer.

Our new DAMQ-DP buffer combines the buffer for virtual channels from two different physical channels without shortcomings above.

3 DAMQ-DP (DAMQ for Dual Port) Scheme

In this paper, we assume the use of wormhole-switching and fixed length flow control unit (flit). And for simplicity, only two virtual channels exist in one physical channel.

For two physical channels can access the shared buffer simultaneously, we choose four-port RF (2R2W Register File) for the data memory. Four-port RF

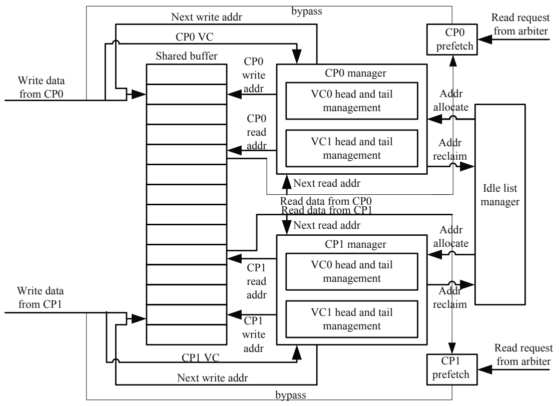


Fig. 1. DAMQ-DP structure

has separate address, data, and control inputs for each of the ports. Each physical channel uses one pair of these read/write ports to access the RF. All virtual channels belonging to one physical channel access memory with the same read/write port. To differentiate the router port and RF port, sometimes we name the router port as CP (channel port), and RF port as MP (memory port).

Fig. 1 presents the DAMQ-DP structure with 2 virtual channels in one CP. The main modules comprise: 1) shared buffer for data and pointer; 2) pre-fetching buffer for each VC to pre-fetch data from shared buffer as well as bypass function writing data from input channel into pre-fetching buffer directly; 3) head and tail pointer management for each VC; 4) idle list manager in charge of idle flit buffer; 5) multiplexer for multiple VCs access to the same read/write port of shared buffer in one CP (not show in this figure)

3.1 Shared Buffer with Pre-fetching and Bypass

Flits in one VC are organized with linked list and each slot in shared buffer is used to store a flit and a pointer to the next flit of the same VC.

Shared buffer is implemented with four-port RF, known as 2R2W Register File. It has two read ports and two write ports. Each CP uses one pair of read/write ports, usually with the same clock. The port, either read port or write port, often has similar signals, such as clock, enable, address bus and data bus. Read operation in this kind of buffer is different from the usual ways of accessing register, and is more like SRAM. Reading from port of the RF is synchronized by the read clock. At the next rising edge of clock, the data in the memory location indexed by address will appear at the output bus. That is, there is one-clock latency between issuing address and data availability. For the next read pointer is also included in the data, such characteristic will degrade performance greatly. The only way to pipeline the read access in the same VC is to get the data immediately when issuing read address.

To accelerate the read access of shared buffer, each VC is equipped with a private pre-fetching buffer with the depth set to three. Just as what the buffer name means, its function is to store data pre-fetched from shared buffer temporarily. It consists of three registers. The read request from crossbar arbiter is always directed to the pre-fetching buffer, so the corresponding data and the next address will be valid at the same rising edge of clock, and at the same time, we get the next flit address to read from. This avoids the three clock defect in original DAMQ.

The arriving flits from input channel can be written into shared buffer, or into private buffer by bypass. The choice depends on the states of both pre-fetching buffer and shared buffer. The rule is, if the destined VC has no data in shared buffer and pre-fetching buffer is not full, the arriving data should go into private buffer directly, else the data should be written into shared buffer.

If private buffer is not full and there are data in shared buffer, data read should be launched from shared buffer into private pre-fetching buffer. For there are more than one VC in a physical channel, and all of them access the shared buffer with the same read port. Some of them may want to fetch data from shared buffer simultaneously, thus there should be an arbiter to resolve the competition and multiplex the common port. Every time more than one VC needs to fetch data, only one of them can be admitted. Arbiter can use some kind of scheduling scheme such as round-robin to fulfill this task, choosing only one VC each time.

3.2 VC Head and Tail Pointers

Each VC has a head pointer (HP) and a tail pointer (TP) to organize the data in shared buffer into FIFO structure. HP points to the address of shared buffer the next read accesses, and TP points to the address the next write happens. When a flit arrives at input buffer, CP applies for a free slot from idle list manager. The applied address will be the TP value in next clock cycle. The data and the applied address are written into the shared buffer with the write address corresponding to current TP. When data are needed from shared buffer, the flit can be accessed from current HP pointing to, and at the same time HP is updated with the pointer contained in data. Finally the address of current HP is sent to idle list manager for reclaim. As we can see, HP and TP always point to the next operations happen, and flit can be read or written without incurring further latency. At the time the system is initialized, HP and TP of the same VC have the identical value, but they are different from other VC's.

3.3 FIFO Structure of Idle List Manager (ILM)

ILM organizes all free slot addresses for shared buffer into a FIFO, and puts them in SRAM, instead of linked list as others. Fig. 2 presents the structure of ILM. To hide the access latency of SRAM, it also has the similar bypass and pre-fetching circuits as shared buffer. An alternative choice is to use 1R1W Register File or Register Array, which can give data out immediately without waiting for one more clock period.

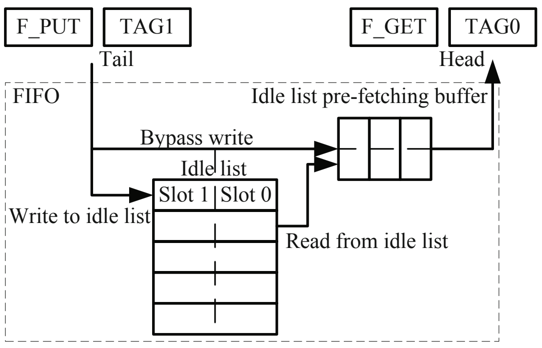


Fig. 2. Idle list manager

If there are no data in SRAM and the pre-fetching buffer is not full, the reclaimed address can be written into pre-fetching buffer directly, otherwise it should be written into SRAM. If pre-fetching buffer is not full and there is data in SRAM, data will be read out from SRAM and written into pre-fetching buffer. The depth of pre-fetching buffer is set to three too. According to the mechanism above, if data number is no more three, all data will exist in pre-fetching buffer. If data number is above three, the top three exist in pre-fetching buffer, and the remaining are in SRAM. This kind of internal FIFO structure is transparent to the outside of FIFO. The read access to FIFO involves no latency and can get the needed data immediately. The data must be gotten from pre-fetching buffer, and not from SRAM.

ILM needs initialization after system reset. Except for those addresses allocated to HP and TP of each VC, all other slot addresses are marked as idle and need to be included in SRAM. When a flit arrives, CP requests an empty slot from ILM for the flit; when a flit leaves, the corresponding slot is reclaimed to ILM. With the help of pre-fetching buffer, an idle slot can be gotten at the same time when needed, which promises zero latency and pipelining operation.

Allocation of Idle Slots. Each unit in idle list FIFO is a pair of shared buffer slot addresses, slot 0 address and slot 1 address. In addition to the FIFO, there are also two registers, F_GET and F_PUT, each of which can store a single slot address. And also, two tags, TAG0 and TAG1, mark the responding register's state. If a valid address is in F_GET, TAG0 is set true. If a valid address is in F_PUT, TAG1 is set true.

For two CPs share the same buffer space, at most two flits can arrive and apply for spare slots. If there is only one request, the first to do is to check if TAG0 is valid. If TAG0 is valid, the address in F_GET is sent to the corresponding CP, and TAG0 is set invalid. If TAG0 is invalid, the slot 0 in head position is sent to CP and address in slot 1 is sent to F_GET. And then, FIFO read address increases. If both CPs request for slots, slot 0 in head position is assigned to

CP0, and slot 1 to CP1. And then, FIFO read address increases. Following gives the description of allocation algorithm.

```
if(CP0_REQUEST && CP1_REQUEST)
begin
    CP0_REPLY = FIFO_HEAD[SLOT_0];
    CP1_REPLY = FIFO_HEAD[SLOT_1];
    READ_ADDR = READ_ADDR+1;
end
else if(CP0_REQUEST)
begin
    if(TAG0)
    begin
        CP0_REPLY=F_GET;
        TAG0     =FALSE;
    end
    else
    begin
        CP0_REPLY = FIFO_HEAD[SLOT_0];
        F_GET     = FIFO_HEAD[SLOT_1];
        TAG0      = TRUE;
        READ_ADDR = READ_ADDR+1;
    end
end
else if(CP1_REQUEST)
begin
    if(TAG0)
    begin
        CP1_REPLY=F_GET;
        TAG0     =FALSE;
    end
    else
    begin
        CP1_REPLY = FIFO_HEAD[SLOT_0];
        F_GET     = FIFO_HEAD[SLOT_1];
        TAG0      = TRUE;
        READ_ADDR = READ_ADDR+1;
    end
end
```

Reclaim of Idle Slot. Just as allocation of idle slot, the reclaim of idle slot has two possibilities, the number of released slots may be one or two. If one slot is released at a time, the first thing to do is to check if TAG1 is valid. If TAG1 is invalid, the slot address is written into F_PUT, and TAG1 is set valid. If TAG1 is valid, both addresses of released slot and the one in F_PUT are written into FIFO, and then, FIFO write address increases. If both CPs released their slots at the same time, both addresses of released slots are written into FIFO, and then, FIFO write address increases. Following gives the description of reclaim algorithm.

```
if(CP0_REQUEST && CP1_REQUEST)
begin
    FIFO_TAIL[SLOT_0] = CP0_ADDR;
    FIFO_TAIL[SLOT_1] = CP1_ADDR;
    WRITE_ENABLE;
    WRITE_ADDR        = WRITE_ADDR+1;
end
else if(CP0_REQUEST)
begin
    if(!TAG1)
    begin
        F_PUT    = CP0_ADDR;
        TAG1     = TRUE;
    end
    else
    begin
        FIFO_TAIL[SLOT_0] = CP0_ADDR;
        FIFO_TAIL[SLOT_1] = F_PUT;
        TAG1              = FALSE;
        WRITE_ENABLE;
        WRITE_ADDR        = WRITE_ADDR+1;
    end
end
else if(CP1_REQUEST)
begin
    if(!TAG1)
    begin
        F_PUT    = CP1_ADDR;
        TAG1     = TRUE;
    end
    else
    begin
        FIFO_TAIL[SLOT_0] = CP1_ADDR;
        FIFO_TAIL[SLOT_1] = F_PUT;
        TAG1              = FALSE;
        WRITE_ENABLE;
        WRITE_ADDR        = WRITE_ADDR+1;
    end
end
```

4 Experimental Results

The proposed scheme is simulated at the cycle-accurate level. The architecture we simulate is an 4-ary 2-cube message exchanging system without wrapped around channels. Each router is attached to one local end-node for injecting and sinking. Some of simulation configuration parameters are as follows:

1)Packets size is set to 8 flits
2)Switching technique used is wormhole
3)VC number in one physical channel is 4
4)Injection process is Bernoulli
5)Routing protocols are dimension-order(DOR) and Valiant [9]
6)Traffic patterns are uniform and transpose

The most basic performance measures of any interconnection network are its latency and throughput versus offered load. We set the total buffer size for each physical channel to 32 flits when SAMQ and normal DAMQ are used. In order to examine the performance of DAMQ-DP with regard to the relationship between buffer size and network performance, we use three different sizes, 24, 26 and 32 flits buffer respectively. By varying the injection rate, we study their impact on throughput and message latency of the network.

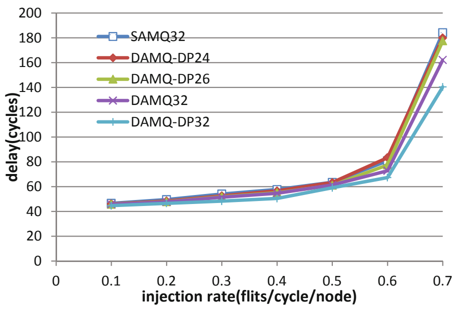
The simulation results of network throughput and message latency under uniform traffic with DOR routing is shown in Fig. 3.

There is hardly any difference for various buffer schemes while the network traffic is low. As to latency, there comes the forking point when the applied traffic load is increased to 0.5. As far as throughput is concerned, the forking point is at 0.3. Along with the network saturation process, DAMQ-DP has higher throughput and lower latency than both DAMQ and SAMQ when they all use same size 32-flits buffer. Comparing DAMQ-DP32 with SAMQ32 we can find the throughput is improved about 10% when load is a light higher. DAMQ-DP with 24-flit buffer achieves approximately the same latency and throughput as SAMQ32, but the buffer in former is only 75% of the latter.

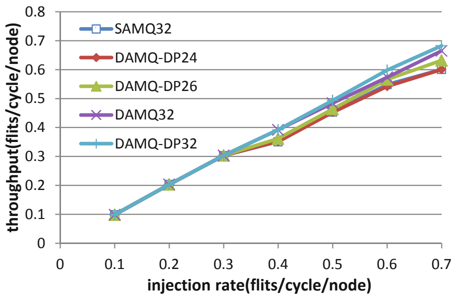
While under transpose traffic pattern, Fig. 4 and Fig. 5 the divarication comes out early when considering latency. There is no significant difference for different buffer schemes below load 0.3. After this point, the difference becomes very clear. Regarding the message latency and throughput, DAMQ24 managed to hold a similar performance as SAMQ32. With the same traffic load and buffer size, DAMQ-DP32 can provide a higher throughput about 5% than SAMQ32.

We also change the routing strategy to Valiant. Under uniform traffic pattern, we can find the similar conclusions as DOR except that throughput for different schemes discriminates earlier.

From the figures above we can find out that DAMQ-DP tends to provide a more efficient method for flits to share buffer space than DAMQ which has already shown advantages over traditional SAMQ scheme. DAMQ-DP achieves the best performance among the three buffer schemes we tested. It can utilize less buffer space without sacrificing the network performance.

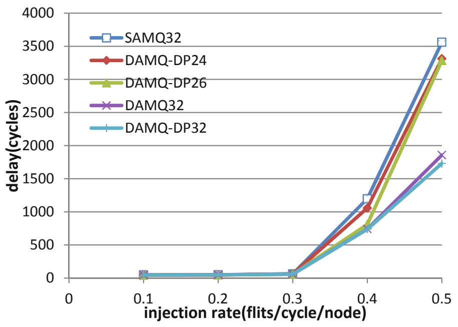


(a) Delay

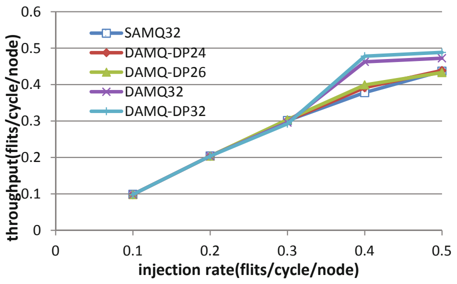


(b) Throughput

Fig. 3. Uniform traffic pattern, DOR

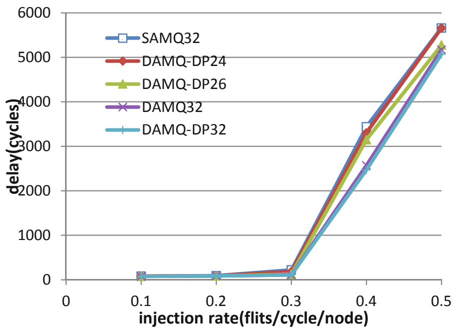


(a) Delay

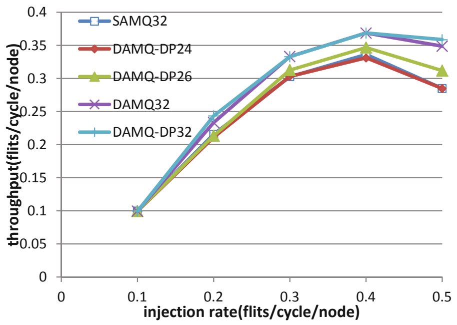


(b) Throughput

Fig. 4. Transpose traffic pattern, DOR



(a) Delay



(b) Throughput

Fig. 5. Uniform traffic pattern, Valiant

5 Conclusion

To enhance the performance of router with fewer buffers for higher throughput and lower communication latency, we propose a buffer sharing scheme that can share the common buffer space between ports, present the buffer structure and idle flit manager. The experiments show that the scheme supersedes SAMQ and DAMQ in terms of both throughput and latency. The scheme has the following features: 1)Less buffer space at similar performance. Under uniform traffic, DAMQ-DP gets the similar performance with 75% buffer. 2)Higher throughput. It outperforms SAMQ with 5% to 10% higher in uniform traffic simulations when same size buffer is used. Above all, DAMQ-DP is an excellent scheme to optimize buffer management providing a good throughput when the network has a larger load.

Acknowledgment. This work is partly supported by the 863 Project of China under contract 2012AA01A301.

References

1. Peh, L.-S., Dally, W.J.: A delay model and speculative architecture for pipelined routers. In: International Symposium on High-Performance Computer Architecture, pp. 0255–0266 (2001)
2. Lai, M., Wang, Z., Gao, L., Lu, H., Dai, K.: A dynamically-allocated virtual channel architecture with congestion awareness for on-chip routers. In: Proceedings of the 45th Annual Design Automation Conference, DAC 2008, pp. 630–633. ACM, New York (2008)
3. Ni, L.M., McKinley, P.K.: A survey of wormhole routing techniques in direct networks. Computer 26(2), 62–76 (1993)
4. Tamir, Y., Frazier, G.L.: Dynamically-allocated multi-queue buffers for vlsi communication switches. IEEE Trans. Comput. 41(6), 725–737 (1992)
5. Liu, J., Delgado-Frias, J.G.: A shared self-compacting buffer for network-on-chip systems. In: Proceedings of the 49th IEEE International Midwest Symposium on Circuits and Systems, vol. 2, pp. 26–30. IEEE Press, Piscataway (2006)
6. Delgado-Frias, J.G., Diaz, R.: A vlsi self-compacting buffer for damq communication switches. In: Proceedings of the Great Lakes Symposium on VLSI 1998, GLS 1998, pp. 128–133. IEEE Computer Society, Washington, DC (1998)
7. Neishaburi, M.H., Zilic, Z.: Reliability aware noc router architecture using input channel buffer sharing. In: Proceedings of the 19th ACM Great Lakes symposium on VLSI. GLSVLSI 2009, pp. 511–516. ACM, New York (2009)
8. Liu, J., Delgado-Frias, J.G.: A damq shared buffer scheme for network-on-chip. In: Proceedings of the Fifth IASTED International Conference on Circuits, Signals and Systems, CSS 2007, pp. 53–58. ACTA Press, Anaheim (2007)
9. Dally, W., Towles, B.: Principles and Practices of Interconnection Networks. Morgan Kaufmann Publishers Inc., San Francisco (2003)

Design and Implementation of Dynamic Reliable Virtual Channel for Network-on-Chip

Peng Wu, Yuzhuo Fu, and Jiang Jiang

The School of Micro Electronics, Shanghai Jiao Tong University
Shanghai, 200240, China
{wupeng,fuyuzhuo,jiangjiang}@ic.sjtu.edu.cn

Abstract. Reliability issue such as soft error due to scaling IC technology, low voltage supply and heavy thermal effects, has caused fault tolerant design be a challenge for NoC(Network-on-Chip). The router is a core element of the NoC, and the virtual channel based on flip-flop which occupies most of the area is the most sensitive element to soft error of the router. Focus on this problem, a dynamic reliable virtual channel architecture is proposed in this paper. It can detect the utilization of the virtual channel to adjust physical configuration to support for no-protection, dual redundancy and TMR (triple modular redundancy) requirements in flexibility. Compared with typical TMR virtual channel design, the synthesis results show that our method can achieve several fault tolerant structures switch with near 3 times resource utilization in ideal case and only 13.8% extra area cost.

Keywords: NoC, reliable, virtual channel, dynamic structure.

1 Introduction

With the rapid development of IC technology, more and more IP cores will be integrated on one chip. But the traditional bus based SoC design will meet many problems, such as the integration and reusability of IP cores. An efficient solution to these problems is NoC, which has regular structure and much higher bandwidth due to multiple concurrent connections. But the shrinking feature size makes more transient faults, including crosstalk, charge sharing and SEU (Single Event Upset) [1], which will degrade the NoC performance even result in system crash. So reliability research has become a hot spot of NoC design.

The router is one of the core elements of NoC, because the packet transmits in NoC through the router. Virtual channel, which is the main buffer of the router, is the most sensitive element of the router. We find that reliable virtual channel has become a challenge for NoC design since that virtual channel consumes about 46% of power [2] and 75% of area of the router [3]. At the same time, different applications or even the same application such as H.264 may have different reliability requirements. So it is important to design a reliable virtual channel which can support different reliability requirements.

In this paper, we propose a reliable virtual channel design, which can detect the utilization of the virtual channel to adjust the redundancy structure to support for no-protection, dual redundancy and TMR requirements. Major benefit of the proposed design over its counterparts is: it can exploit the inherent redundancy to meet different fault tolerant requirement.

The rest of this paper is organized as follows. Related work will be described in Section 2. Basic background and router structure will be described in Section 3. The proposed reliable virtual channel design will be described in Section 4. The results and comparison is presented in Section 5. Conclusion is drawn in Section 6.

2 Related Work

Reliability research of NoC has become a hot spot from different aspects, including retransmission mixed with error detection, spare link or router, triple modular voting structure and fault tolerant routing algorithm. Murali etc [4] proposed an error detection and recovery scheme for NoC design based on area, power and performance constraints. Yung-Chang Chang etc [5] proposed a fault tolerant NoC architecture using spare routers. Fault tolerant routing algorithms, including stochastic and adaptive, have been suggested in many papers [6-7].

The research and design above are mostly based on system level, and analyzed by the simulator. There are also some researches aiming at implementation of the element of NoC. Shih-Hsun Hsu etc implemented a router for ANoC [8]. M. H Neishaburi etc [9] proposed an enhanced reliability aware NoC router for permanent error.

In this paper, different from previous works, we design and implement a virtual channel support for different reliability requirements by the Verilog HDL and compare the area cost with the traditional designs.

3 Basic Router Architecture

In order to introduce proposed virtual channel design, we give the basic router microarchitecture as Fig 1 shows. In most of wormhole flow control mechanism based

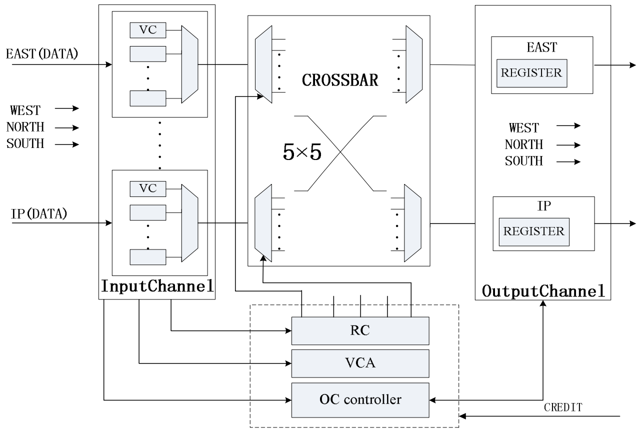


Fig. 1. Basic router microarchitecture

design, virtual channel is allocated in input channel. And it is the main buffer in the router. There are also some combinational logic part, such as routing computer part (RC), virtual channel allocator (VCA) and crossbar. The data packet will be divided into many flits to transmit in NoC, and one head flit will be added to control the data flow. The head flit will work in RC and VCA to decide which port and which virtual channel to flow. But the data flit, which carries the data payload, will stay in virtual channel for several cycles. So the virtual channel needs to be protected.

4 Proposed Reliable Virtual Channel Design

Our virtual channel design includes three aspects: 1) Flit format needs to be modified, which demands 3 extra bits. Two of them denote RR (Redundancy-Requirement), 00 means that the flit has no reliability requirement, 10 means the flit is dual redundancy data, 11 means the flit has TMR requirement. The other bit is parity check bit, which is support for dual redundancy data. And the new flit format also needs extended link, crossbar and FIFO bandwidth support. 2) The virtual channel needs to be re-designed, which is shown in Fig 2, including extra status registers (VC-Org and Friend-VC) to generate the redundancy structure signals and two auxiliary pointers to maintain two or three copies read and written in one cycle. 3) The router pipeline needs to be changed, which requires two stages, VC-in-configure and VC-out-configure to control the write and read of the virtual channel.

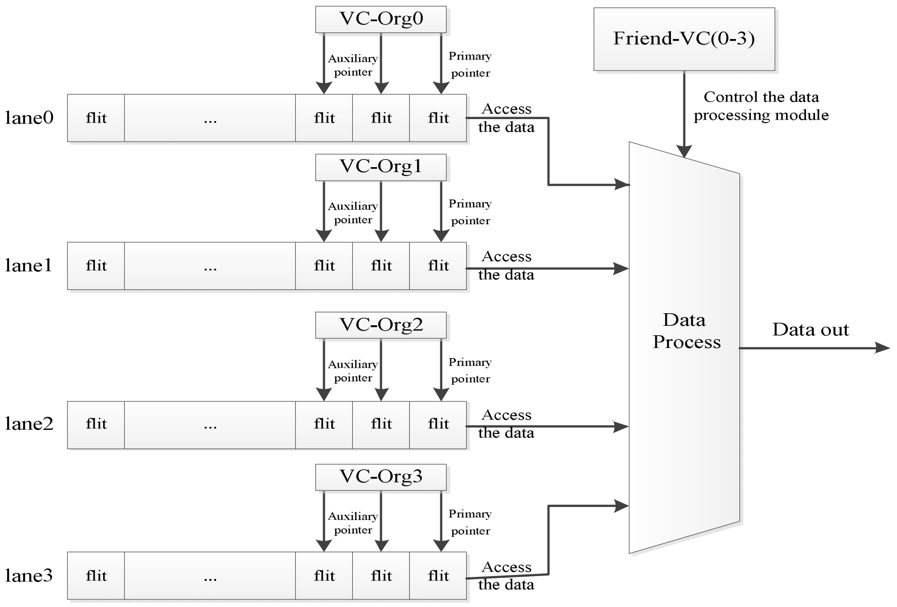


Fig. 2. Proposed Virtual Channel Architecture

VC-in-Configure: The function of this stage is to determine the redundancy structure of the virtual channel when the flit writes. The flowchart of the stage is shown in Fig 3. When a head flit will be written into the virtual channel, RR will be decoded at first.

- If RR is 00, which is region 1 in Fig 3. It means the packet (all flits after the head flit and before another head flit) has no reliability requirement. The flit will be stored in the virtual channel lane which is be allocated in the VCA, and the status registers of the lane will be the initial value.
- If RR is 10, which is region 2 in Fig 3. It means the packet has dual redundancy requirement (with parity check). Then we will check if there are two empty lanes in the virtual channel, just because only the empty lane is available in the wormhole flow control mechanism. If there are two empty lanes in the virtual channel, the flit will be stored in the two lanes. And each lane has a status register called Friend-VC as shown in Fig 2, which record the lane that stores the same flit with this lane. It achieves the transverse redundancy structure. If there is only one empty lane in the virtual channel, the flit will be stored in this lane twice with the help of the two auxiliary pointers of the FIFO as shown in Fig 2. It means the continuous two flits save the same data and it achieves the longitudinal redundancy structure, which is recorded in VC-Org.
- If RR is 10, which is region 3 in Fig 3. It means the packet has TMR requirement. Similar to dual redundancy, if there are three empty lanes in the virtual channel, the flit will be stored in the three lanes, otherwise the flit will be stored in the lane for three times. At the same time, Friend-VC and VC-Org will be update to record the redundancy structure of the virtual channel.

VC-Out-Configure: The function of this stage is to export the protected data when there is a read request of the virtual channel, as shows in Fig 4. When there is a read request of a lane, we will check its VC-Org at first.

- If VC-Org is 00, which is region 1 in Fig 4. It means it is single mode state. Then we will check the RR, if it is 00, it means the data has no reliability requirement, and export the data of lane directly. If RR is 10, it means the data is dual redundancy data, but the lane is single mode, so its redundancy data is in another lane. We will check the Friend-VC to take the redundancy data and use a dual mode voter and the parity check bit to get the protected data. If RR is 11, it means it is TMR data. We will check the Friend-VC to get other two redundancy data and use a triple mode voter to get the protected data.
- If VC-Org is 10, which is region 2 in Fig 4. It means the lane is dual mode state. We will use one auxiliary pointer to get two data in one cycle. Then we will use a dual mode voter and the parity check bit to check and export the protected data.
- If VC-Org is 11, which is region 3 in Fig 4. It means the lane is triple mode state. We will use the two auxiliary pointers to get three data in one cycle. Then we will use a triple mode voter to arbiter and export the protected data.

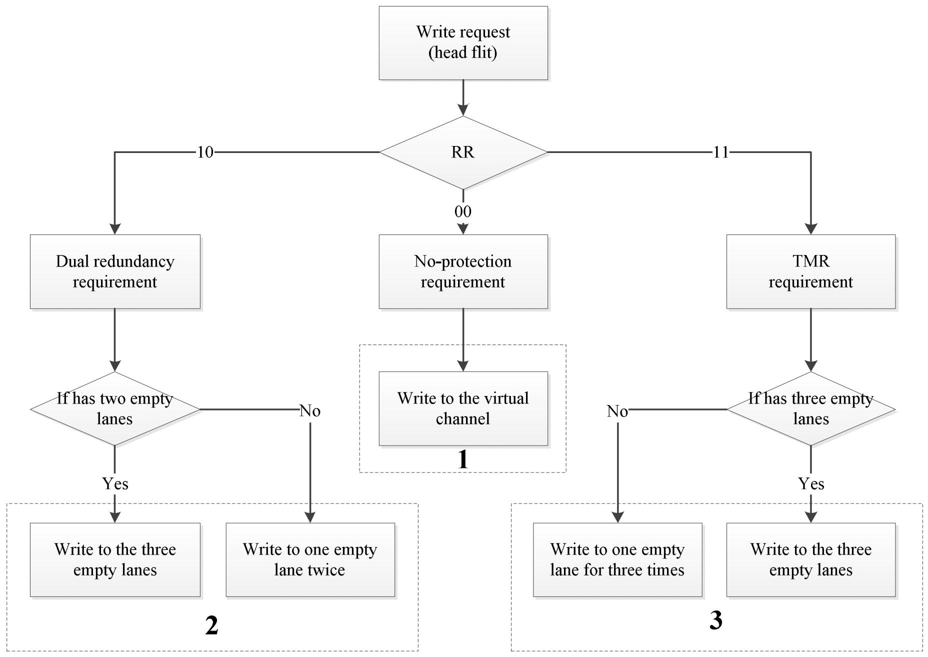

Write request (head flit)
RR
10
00
11
Dual redundancy requirement
No-protection requirement
TMR requirement
If has two empty lanes
No
Yes
Write to the virtual channel
1
If has three empty lanes
No
Yes
Write to the three empty lanes
Write to one empty lane twice
2
Write to one empty lane for three times
Write to the three empty lanes
3

Fig. 3. VC-in-configure flowchart

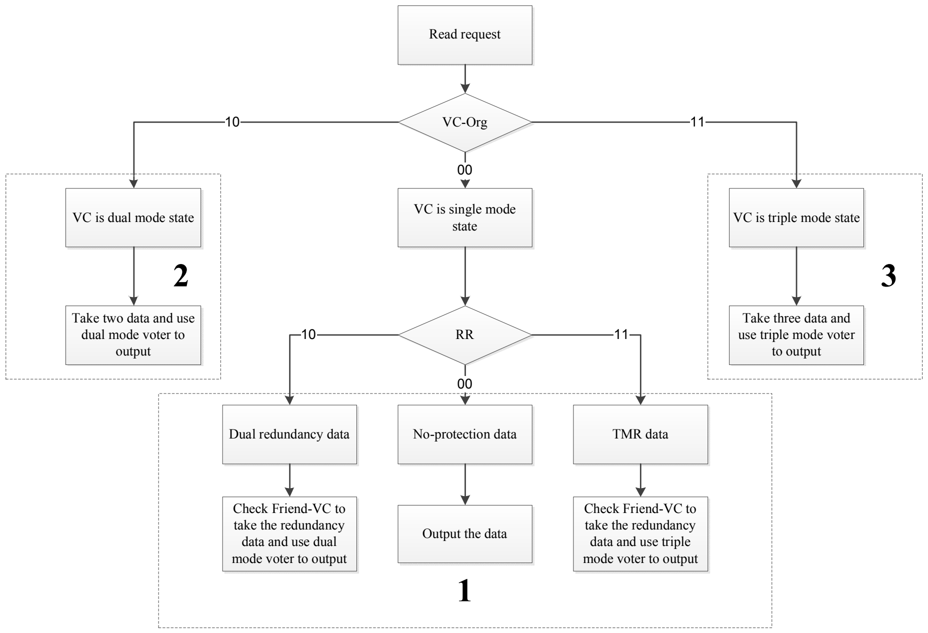

Read request
VC-Org
10
00
11
VC is dual mode state
2
Take two data and use dual mode voter to output
VC is single mode state
VC is triple mode state
3
Take three data and use triple mode voter to output
RR
10
00
11
Dual redundancy data
No-protection data
TMR data
Check Friend-VC to take the redundancy data and use dual mode voter to output
Output the data
Check Friend-VC to take the redundancy data and use triple mode voter to output
1

Fig. 4. VC-out-configure flowchart

5 Experimental Result

5.1 Efficiency Analysis

We have implemented three kinds of virtual channel by Verilog HDL, the basic design which has no protection [8], the typical TMR virtual channel and the proposed design. The typical TMR virtual channel uses three lanes to support TMR requirement. In our all designs, there are 4 lanes in one input port, and each size is 6 flits.

To evaluate reliability of the proposed design, we use the single error model, which means there is only one error at the same time. But the error can be one bit or several bits. Simulated by ModelSim, we find dual redundancy requirement can protect the data from the one bit error, and TMR requirement can protect the data from the multi-bit error in one flit. It confirms that proposed design has the expected goal.

At the same time, the proposed design can use transverse structure when the load of NoC is low to improve the throughput, and when the load is high it can be switched to vertical structure to reduce the congestion. For example, in ideal case, if the load is low enough, the typical TMR design based vertical structure need two more flits in the virtual channel to store the redundancy data, but at the same time the proposed design can be switched to transverse structure to get better performance. We extended cycle accurate simulator Nirgam [10] to support the proposed design. We select 4x4Mesh with typical XY routing. The packet accruing strategy is CBR (Constant Bit Rate). And each port includes 4 VCs and the buffer size per VC is 6. At the low load, the typical TMR design is like each VC size is 2 and the proposed design will be still 6. The simulation result shows that at low load the throughput of the proposed design is about 2.66 times of the typical TMR design and the average flit latency of the proposed design is about 41.1% of the typical TMR design. It means the resource utilization of our design may be near 3 times of the typical TMR design in ideal case.

5.2 Cost Evaluation

After verifying the fault tolerant performance of the proposed design, we synthesize the three Verilog RTL model based on TSMC 130nm cell library by Synopsys Design Compile. In the corner of typical operation, we have set up the constraint, with rise time, fall time and skew value of the clock of 0.1 ns, and input delay and output delay of 0.2ns, also we set the set_max_area of 0 to get the least area of the frequency, and compared with the traditional router design, we set the frequency as 200MHz and the buffer size is 6 flits. The parameters are shown in Table 1. We can find that the typical TMR virtual channel needs about 11.1% extra area cost to support the reliability requirement. And the proposed design achieves the dynamic structure to get better performance with about 13.9% area cost compared with the typical design.

Table 1. Synthesize parameter results

	Frequency	Buffer size	Area
Basic VC design	200MHz	6 flits	$89428.625\mu m^2$
Typical TMR VC	200MHz	6 flits	$99376.414\mu m^2$
Proposed design	200MHz	6 flits	$113127.18\mu m^2$

6 Conclusion

In this paper, we design and implement a new virtual channel of NoC which can achieve different reliability requirements with dynamic redundancy structure. The main contribution of the proposed design is it can detect the utilization of the NoC to adjust the redundancy structure to get better performance.

Acknowledgements. Our work is supported by the Cisco Research Center Requests for Proposals (RFPs), Virtualization for NoC fault tolerance, 2011-90403 (3696).

References

1. Kim, J., Park, D., et al.: Design and analysis of an NoC architecture from performance, reliability and energy perspective. In: Proceedings of the 2005 ACM Symposium on Architecture for Networking and Communications Systems, pp. 173–182 (2005)
2. Owens, J.D., et al.: Research challenges for on-chip interconnection networks. IEEE Micro 27(5), 96–108 (2007)
3. Gratz, P., et al.: Implementation and evaluation of on-chip network architectures. In: Computer Design, ICCD 2006, pp. 477–484 (2006)
4. Murali, S., et al.: Analysis of error recovery schemes for network on chips. Design & Test of Computers 22(5), 434–442 (2005)
5. Chang, Y.-C., et al.: On the design and analysis of fault tolerant NoC architecture using spare routers. In: Asia and South Pacific Design Automation Conference, pp. 431–436 (2011)
6. Hu, J., Marculescu, R.: Dyad: Smart routing for network-on-chip. In: Proceedings of Design Automation Conference (DAC), pp. 260–263 (2004)
7. Pirretti, M., et al.: Fault tolerant algorithms for network-on-chip interconnect. In: Proc. Int. Symposium Very Large Scale Integrate (ISVLSI), pp. 46–51 (2004)
8. SHHSU: Design and Implementation of a router for network-on-chip. Department of Electrical Engineering National Cheng Kung University Thesis for Master of Science (2005)
9. Neishaburi, M.H., et al.: ERAVC: Enhanced reliability aware NoC router. In: Quality Electronic Design (ISQED), pp. 1–6 (2011)
10. Nirgam: `http://nirgam.ecs.soton.ac.uk/`

HCCM: A Hierarchical Cross-Connected Mesh for Network on Chip

Liguo Zhang[1,2], Huimin Du[2], and Jianyuan Liu[2]

[1] School of Microelectronics, Xi Dian University, Xi'an, China
[2] School of Electronic Engineering,
Xi'an University of Posts & Telecommunications, China
{zhanglg,fv,liujianyuan}@xupt.edu.cn

Abstract. As the continuous development of semiconductor technology, more and more IP cores can be contained on the single chip. At this time the interconnected structure plays a decisive role on the area and performance of system on chip, and has a profound influence on the transmission capability of system. Based on the distributed routing lookup, we proposed a new kind of inerratic interconnection network is named HCCM (Hierarchical Cross-Connected Mesh), which is consisted of a $N \times N$ Mesh interconnection of N^2 subnets, every subnet comprised of 2×2 interconnection by full connection. Meanwhile, this paper comes up with a new hierarchical routing algorithm——HXY (Hierarchical XY), the simulation results demonstrate the HCCM topology is superior to the Mesh and the Xmesh topology on the performance of system average communication delay and normalized throughput.

Keywords: Network on Chip (NoC), Distributed Routing Lookup, Routing Algorithm, OPNET Modeler.

1 Introduction

The current decades have witness the rapid development of semiconductor technology, the significant increase of chip integration and the continuous development of the multi-core technology and parallel computing, The interconnection technology of Network-on-chip[2] are widely applied because of the excellent scalability, which the system can contain multiple processors、 memory. However, the network-on-chip has profound influence on the performance and scalability of the multi-processor. Taking into account the difficulty and cost of physical implementation, most of the existing single multi-processor using the classical topology, just as Mesh[3]、 Torus[4] etc.

The literature[5] analyzed seven different types interconnect features of different applications, including data-intensive matrix and vector computing, sparse matrix and vector operations、 transformation between the time domain and frequency domain etc, from which we can find: in some applications, which each node may have large volume of business with two or three adjacent nodes, under other situations, each node may communicate with more than four adjacent nodes, Based on the latter application, each node in Mesh network is just directly connected with four adjacent nodes, this characteristics of Mesh network can not satisfy the application. Therefore, a new kind scalable network on chip based on short-term interconnection, which is named HCCM (Hierarchical cross-connected Mesh) is proposed. HCCM network is consisted of several subnets, every subnet is seems as a single node, which is composed of four nodes by full connection, all the subnets is connected in Mesh structure. Through slight increasing in the cost of hardware, HCCM network adds the connectivity of each node, thereby increasing the effective interconnect bandwidth of each node.

The Xmesh network proposed by literature[1] is also based on Mesh structure by adding the straight path through main diagonals, what can decrease the network diameter and increases the average bandwidth, at the same time, which is likely to result in main diagonals block, which would significant affect the performance of network. Under the situation of main diagonal block, the performance of Xmesh network is close to Mesh network.

This paper presents the HCCM network. Compared with Mesh and Xmesh network, HCCM network has short network diameter and well localized, which the average delay is less than Xmesh and Mesh, is better than Xmesh and close to Mesh network on the load distribution. In the pattern of hot spot load, HCCM is better than both the Xmesh and Mesh on the normalized throughput.

The HCCM network is especially suitable for the high-performance router structure[6], which can blend the switching and forwarding organically: every subnet distributed store a complete routing table, four nodes in a subnet complete the lookup of routing table entry simultaneously[7-9]. According to the result of lookup, the exchange of packages will be completed in the global network. Due to the better performance of HCCM than Xmesh and Mesh which has been used in multi-core interconnection for NOC, therefore, we believe that the HCCM network can be used for the multi-core interconnection.

2 HCCM Networks

HCCM network is regular and symmetric which can be defined as two layers. The first is the subnet which is connected by full connection; the second is the mesh network, which every subnet can be seemed as a single node.

2.1 Subnet of HCCM Network

As shown in Fig.1, every subnet is consisted of four nodes by full connection, which can be represented as $N_i (0 \leq i \leq 3)$. Each node can reach to others only by one hop. (x, y, z) is used to located every node, in which x and y are located every subnet in the whole HCCM network, z represents different nodes inside the subnet which including four nodes (In simulation model, the coordinate includes two parts, which the fist part can be turned into x and y, the second part is the same as z). The packets are sent to take two diagonals as much as possible during transmission, which result in one hop shorter than mesh network. That is one of the reasons why HCCM is more efficient than Mesh network.

2.2 Global Network of HCCM

In Fig.2, The global network of HCCM is connected by $\sqrt{N}$ subnets in Mesh connection.

Based on the application of distributed routing lookup, the routing table can be divided into four parts, which of them are stored respectively in the four nodes of the subnet in HCCM network. From the view of the global network, every subnet has a complete routing table, each node can receives the IP packet. Once the packages arrive in any nodes, the node will begin to look up the table entry which have been stored in it, where the require table entry are stored in, where the package will be send to, then, completing the lookup and forwarding. There would be large amount in local traffic relatively, however, the interconnection of full connection can make that every package can reach to another one just in one hop inside the subnet. This approach can significant improve the efficiency of local forwarding. Since each node only stores a quarter of the entire routing table, the time for routing lookup will be drastically reduced. All these can dramatically promote the efficiency of route table lookup, at the same time, it would reduce the requirement space to store the route table in every node.

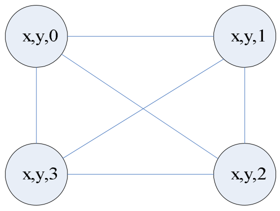


Fig. 1. The structure of subnet

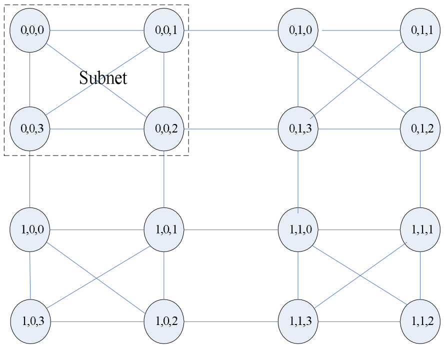


Fig. 2. The structure of global network

2.3 Topological Properties

In Table 1, the topological properties comparison for different network is illustrated, which of these are used to evaluated the complexity, communication efficiency and cost of one network, is the important criteria to decide which kind of network can be applied in the design. All nodes in the three kinds network have the different node degree. The diameter of a network[10] is the maximum distance between two nodes in a network. If there were N nodes in a HCCM network, then there were $\sqrt{N/4}$ subnets in every horizon or vertical line. The maximum distance in HCCM network is from the node (0,0) to ($N/4$ -1,2), There would be $2(\sqrt{N/4}-1)$ hops between these two nodes in the Mesh network with N nodes, $\sqrt{N/4}$ hops will be shorter in the HCCM network because of the diagonals. As a result the diameter of HCCM network is $2\left(\sqrt{N/4}-1\right)-\sqrt{N/4}=3/2\sqrt{N}-2$. A mesh with N nodes network have $2\left(N-\sqrt{N}\right)$ links, Two diagonals are added in every subnet, which cause that there will be $N/2$ links more than the number of links in Mesh network, so, $2\left(N-\sqrt{N}\right)+N/2=5/2\times N-2\sqrt{N}$ links exists in HCCM network , which have N nodes. Under the same scale, the number of links of Xmesh network is $2N$.

Table 1. Topological properties comparison

Topo	Diameter	Degree	Numbers of links
Mesh	$2(\sqrt{N}-1)$	4	$2(N-\sqrt{N})$
Xmesh	$\sqrt{N}-1$	6	$2N$
HCCM	$3/2\sqrt{N}-2$	5	$5/2N-2\sqrt{N}$

What Table.1 mirrored is that HCCM network has demand on the parity of the number of nodes because of there are four nodes in every subnets. Xmesh is the minimum in diameter, Mesh is the minimum and HCCM is the maximum in the number of links.

By adding some diagonal edges on the Mesh topology, the average distance of the Xmesh network is reduced sharply, which is the reason that Xmesh has the minimum diameter. Xmesh has a good network performance under small scale interconnection and light load. A simulation analysis based on OPNET simulator shows that heavy load will result in packets jams on the diagonal edges, which will seriously affect the performance of Xmesh network, especially the load distribution.

2.4 Ideal Throughput

Ideal throughput is the maximum throughput under the perfect flow control and route strategy. The literature[11] gives a formula for calculating the ideal throughput: $TH \le 2b\times B_C/N$, B_C represents the bisection channels when the whole network was divided into two identical parts. b is the data width of each channel, N represents the total number of nodes.

N=64, B_C =16 when the Mesh network under the scale of 8×8(There would be eight channels must be cut off when the network was divided into two parts). At the same scale, the TH value of HCCM, Xmesh and Mesh just as following:

$$THMesh \le b/2 \quad THHCCM \le b/2 \quad THXMesh \le 3b/4$$

The HCCM and Mesh have the same ideal throughput, Xmesh has the maximum ideal throughput.

2.5 HXY Routing Algorithm

HXY routing algorithm is base on the offset between the source and destination address. Firstly, the relative position of destination node away from the source node will be figure out based on the offset, Secondly, Choosing the shortest path inside the subnet.

The following definitions have to be made before describing the algorithm:

Definition 3: The source node and destination node are presented by C(id_region_C , id_C), D (id_region_D , id_D) . The id_region_C and id_region_D must be translated into coordinate:

x1=(int) id_region_C / $\frac{1}{2}\sqrt{N}$; y1=(int) id_region_C % $\frac{1}{2}\sqrt{N}$;

x2=(int) id_region_D / $\frac{1}{2}\sqrt{N}$; y2=(int) id_region_D % $\frac{1}{2}\sqrt{N}$;

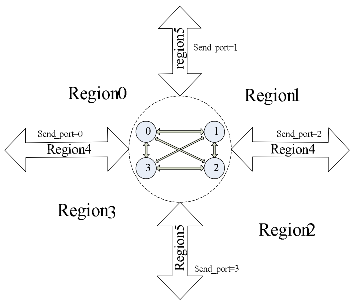
id_C and id_D are used to identify four nodes in a subnet, which the value of them are 0, 1, 2, 3.

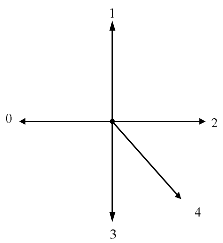
Definition 4: Routing regions: According to the relative position of destination node away from the source node, the destination node will falls in one of the six sections ,which just as Fig.3.

Definition 5: The send port: Every node has four ports to send the packages, from 0 to 4. The local port is defined as 5. As Fig.4.

Algorithm describing:

Xoffset=x2-x1, Yoffset=y2-y1 represent the offset on the X and Y axis. Every subnet is treated as a node, As Definition1 x1, y1, x2, y2 are used to locate the position of source and destination subnet in the whole network. Figuring out the send_port_x and send_port_y by XY algorithm and YX algorithm respectively, the routing region can be calculated by both of them. According to the routing region and the value of id_C and id_D , we can figure out the next hop based on the shortest path principle.





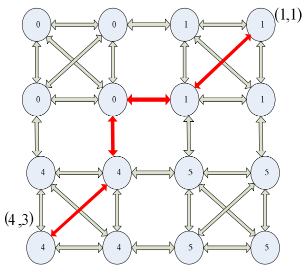


Fig. 3. Routing algorithm **Fig. 4.** Direction of send port **Fig. 5.** One of the longest path

This algorithm applies the XY or YX algorithm flexibly. Take the diagonals port as much as possible. Just as Fig.5 the red path from node (1,1) to (4,3) is one of the longest path in the scale of 4×4 HCCM network, which is two hops shorter than the Mesh network under the same scale.

3 Simulations and Evaluations

An OPNET-based NoC (Network-on-Chip) simulation environment, is adopted to evaluated the performance of the HCCM network and HXY algorithm, including the package loss rate, average end to end latency, port load distribution and normalized throughput. The uniform traffic patterns are used, in which each node sends packages to other nodes by a uniform of the destination node address.

The average end to end latency (ETE latency) is used to evaluate the latency performance of the proposed routing algorithm. The ETE latency is usually defined as the time elapsed from when the message header is injected into the network at the source node to when the last unit of information is received at the destination node. The package loss rate is used to evaluate the throughput when the network has been saturated. The normalized throughput is used to evaluate the network throughput of the proposed routing algorithm. The normalized throughput refers to the ratio of the number of packets successfully transferred by the network to that of the packets injected into the network.

Mesh, Xmesh, HCCM network with the same scale of 8×8, are used as comparison to evaluate these three kinds topological. Every package's length is 48 bits. The speed that the packages injected into network can be set up through the time interval to send packages.

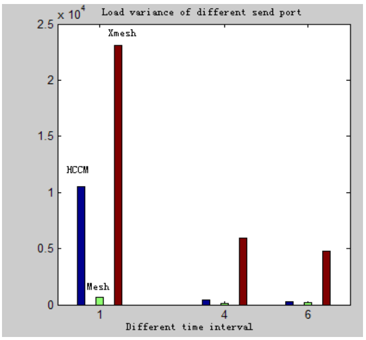


Fig. 6. The load variance of different send port

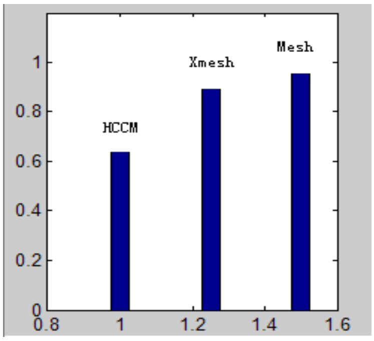


Fig. 7. The package loss rate

Under the traffic of balance load, the destinations of all nodes are random distributed in the entire network. The normalized throughput, the package loss rate, the port load distribution and the ETE delay are applied.

We collected different load of different ports from three kinds network in different traffic pattern, calculating the variance of the load of every network, which used to evaluated the load balance, the greater the standard deviation is, the more uneven the network load would be, what the Fig 6 mirrored just as following:

(1) The five direction load distribution of HCCM network is between Xmesh and Mesh network, it would be close to Mesh network in lighter load, there would be worse than Mesh in heavy load.

(2) The Mesh network has the most evenly load distribution in four directions.
(3) The Xmesh network owns the most uneven distribution in five directions.

In Xmesh network, the traffic in Northwest is maximum, which is just one of the diagonals, at the same time, north has the minimum load. Because that HCCM network takes the shortest route internal the subnet, there will be heavy load on the two diagonals inside a subnet, on the global network, HCCM is the same as Mesh network, and as a result, The HCCM network is more unevenly than Mesh network on average.

The same method is applied as Fig 7. What we can get from Fig 7 is that HCCM network is the minimum on the package loss rate, Mesh network is the maximum.

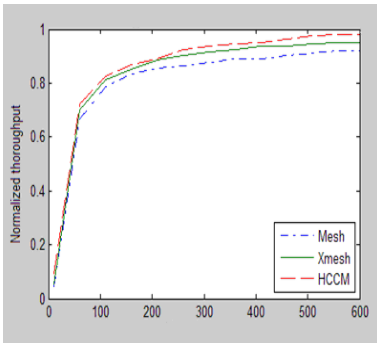


Fig. 8. The normalized throughput

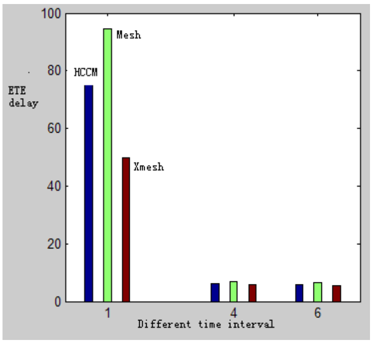


Fig. 9. The ETE delay of different network

As shown in Fig 8, there is not a very larger gap among three kinds networks. HCCM owns the largest normalized throughput, Mesh is the minimum one, Xmesh is just between these two networks.

On the ETE delay, the following conclusions can be draw.

(1) HCCM is slightly smaller than Mesh network in lighter load, which is between the Mesh and Xmesh network in heavy load.
(2) Mesh is the maximum network on the ETE delay no matter what kind traffic pattern.

Comparing all the statistics above, through analyzing the network diameter of three kinds networks, which the Mesh has the longest network diameter, Xmesh has the shortest one. Considering the normalized throughput, the load distribution, the package loss rate and the ETE delay of different networks, we can come up with these following conclusions:

(1) Due to the unevenly load distribution, the higher package loss rate and the traffic congestion on the diagonals of Xmesh, the normalized throughput of HCCM is higher than Xmesh in the simulation. The Mesh network owns the minimum normalized throughput because of the longest network diameter.
(2) Due to the high package loss rate and longest network diameter, Mesh network has the maximum ETE delay. In the heavy load, HCCM is smaller than Xmesh network on the ETE delay. In the lighter load, the ETE delay of HCCM is equal to Xmesh.

4 Conclusions

The characteristic of HCCM network is well localized, easy for expansion, is better than Mesh and Xmesh network on the aspect of throughput and load distribution. At present, we have implemented the HCCM network of four nodes by the distributed routing lookup in the Xilinx NetFPGA, and completed the test on the network consisted by NetFPGA[13]. In the next step, we will enable to complete the blending of switching and forwarding in multi-subnets of large network. When the network is large-scale, the HCCM network is superior to both Mesh and Xmesh in many aspects, so, we believe that HCCM network can be applied in multi-core interconnection.

Acknowledgement. This project is supported by the National Science Foundation of China (61136002, 60976020), The Ministry of Education Scientific Research Key Project (211180) and Science Research Program of Education Department of Shaanxi Province Government (11JK1063, 2010JK833, 2010JK826,) and Shaanxi Province Industry Tackles Key Problem Plan (2011K06-47) and The Natural Science Foundation of Shaanxi Province(2010JQ8014).

References

1. Zhu, X.J., Hu, W.W., Ma, K., Zhang, L.B.: Xmesh: A mesh-like topology for network on chip. Journal of Software 18(9), 2194–2204 (2007)
2. Zhang, X.P., Liu, Z.-H., Zhao, Y.-J., Guan, H.-T.: Scalable Route- Journal of Software, 7 (July 2008)
3. Duato, J., Yalamanchili, S., Ni, L.M.: Interconnection networks: An engineering approach. Morgan Kaufmann (2003)
4. Asanovic, K.: The Landscape of Parallel Computing Research: A View from Berkeley, Electrical Engineering and Computer Sciences University of California at Berkeley, p. 10 (2006)
5. Dong, Z.P.: High-performance router. Posts & Telecom Press-Computer Science (2005)
6. Chang, Y.K., Liu, Y.C., Kuo, F.C.: A Pipelined IP Forwarding Engine With Fast Update, Bradford, United Kingdom, pp. 263–269 (2009)
7. Jiang, W., Prasanna, V.K.: Parallel IP lookup using multiple SRAM-based pipelined, pp. 14–18 (2008)
8. Sun, Z.G., Dai, Y., Gong, Z.H.: MPFS: A truly scalable router architecture for next generation Internet. Science in China Series F: Information Science 51, 1761–1771 (2008)
9. Shen, Z.: Average diameter of network structures and its estimation. In: Proc. of the 1998 ACM Symp. on Applied Computing, pp. 593–597 (1998)
10. Dally, J., Towles, B.: Principles and Practices of Interconnection Network. Morgan Kaufman Publisher (2003)
11. Li, X., Ye, M.: Network Modeling and Simulation with OPNET Modeler. Xidian University, Xian (2006)
12. Wang, M., Du, H., Wang, Y.: An IPv4 Router based on Distributed Forwarding. In: The 3rd International Conference on Computer and Network Technology (ICCNT 2011), TaiYuan, pp. 10–15 (2011)

Efficient Broadcast Scheme Based on Sub-network Partition for Many-Core CMPs on Gem5 Simulator

Kaikai Yang, Yuzhuo Fu, Xing Han, and Jiang Jiang

School of Microelectronics,
ShangHai Jiao Tong University,
P.R. China
yangkaikai@ic.sjtu.edu.cn

Abstract. Networks-on-chip (Noc) is proposed to achieve extensible and higher bandwidth communication in many-core CMPs. To make full use of the IC resource efficiently, sub-network partitioning oriented to Noc is proposed, which divides the whole Noc into regions to achieve the traffic isolation demand that acquired by Cache coherence protocol. We take the region segmentation for mesh-based Noc, the task mapped PEs (processing elements) aggregate into the Logic sub-network, and routing between these PEs is implemented in the according Physical sub-network, in which an efficient tree-based broadcast scheme based on multicast XY routing algorithm is carried out. The Gem5 Simulator is used to promote the research, experimental results shows our approach have a quite less average packet latency compared with multiple unicast.

Keywords: Many-Core, Networks-on-chip (Noc), Broadcast, XY Routing, Sub-network.

1 Introduction

In the recent decades, owing to the improvement of integrated circuit (IC) manufacture technology and development of the microprocessor architecture, the performance of microprocessor was advanced by 60 percentages per year. The progress of IC manufacture provides massive resources for individual chip, how to make use of the IC resource efficiently to improve the properties and throughput of the microprocessor has become an important issue.

On-chip communication turns into a performance bottleneck of the many-core system. The reason is with the chip integrity improving constantly, the traditional bus or crossbar switch-based interconnect bring a poor communication ability and yield high power consumption, which is a serious constraint to the many-core architecture development. However, Noc interconnects plenty of cores merged in the many-core system, which provide scalable, high throughput communication architecture for SoC (System on Chip) design.

In additional, as mentioned above, how to make full use of the existing computation resource in the many-core system to attain the maximum extent parallelism in computation and processing become a key problem. Mapping to Noc level, sub-network partition method is proposed. The whole network will be divided into a few of different size region, and the communication between each node will be limited to the sub-network only, it indicates that traffic isolation among every two regions exits. Meanwhile, in the Cache coherence with sharing memory system, there is a certain amount of one-to-all communication demanded by Cache coherence protocol. A PE launch a broadcast action in the whole network bring high network latency and power consumption, however, a reasonable sub-network partitioning method will reduce much more latency and power but also attain the requirement asked for by the Cache coherence protocol.

An effective broadcast mechanism is also demanded to cut down the cost of the broadcast action. Basically, the routing manner plays the most important role in supporting the broadcast action. There are two routing methods can be achieved in the worm-hole mesh-based Noc, which called source routing and distributed routing. In source routing, packet header flit carry the pre-computed routing path message of each destination router node and start routing followed by other data flits, this manner brings large traffic to occupy extra network bandwidth. Otherwise, in distributed routing, packet header flit only storages the location information of each destination node, and the intermediate router node will automotive compute the output network link according to the information.

This paper organized as follows. Section 2 describes earlier work on multicast or broadcast mechanism in Noc. In Section 3, we make several definitions and assumptions related. In section 4， an effective broadcast scheme with sub-network partition method is adopted. Section 5 introduces the simulation environment. Section 6 reports the performance evaluation of our mechanism. Section 7 conclusion.

2 Related Work

In order to provide one-to-all broadcast communication hardware support, and sustain the Cache coherence protocol which carried out in the many-core system strongly, considering an effective broadcast scheme comes to be our research objective. As broadcast is a special case of the multicast, we review the former research about multicast approaches below, and only for wormhole mesh-based Noc.

Generally, multicast scheme could be implemented as three ways, unicast-based, path-based and tree-based. In the unicast-based way, source router break the multicast packet into several unicast packets and send them one by one, just in the unicast mode. This way could have a simple realization but bring more risk to produce network block, and exist a high network latency and power consumption. In the Path-based way, source node arrange all destination nodes in a path of a certain order, then send the multicast packet along this path, It is easy relatively to implement in hardware, but with the increasing of the number of router nodes, the path size will increase and bring high network latency accordingly. In the tree-based way, the source router constructs a spanning tree based on itself. This mode replicates the packet at the necessary time and form a branch of the tree. In comparison to the above two modes, tree-based mechanism has a much lower network latency and power consumption.

The virtual circuit tree-based multicast (VCTM)[1] algorithm implemented by building a virtual circuit tree in advance, and have a time-division multiplexing on the physical link in support of several numbers of trees. The region partition multicast (RPM)[2] algorithm make an improvement to the VCTM. On reaching an intermediary router, each multicast packet recursively divided into multiple regions based on the current node which looked as a starting node, and continue to routing in the destination region. Multicast Rotary Router (MRR) [3] adds a congestion detection mechanism based on the RPM, which weigh the depth and breadth of the multicast tree dynamically to reducing unnecessary packet replication operation, but the hardware implementation of this mechanism is more complex.

However, above multicast schemes cannot provide irregular mesh topology support. Broadcast Logic-Based Distributed Routing (bLBDR)[4][5] proposes a minimize path tree-based broadcast approach in the region, which ensure that each node will only receive a broadcast packet, meanwhile, a broadcast bit is added to distinguish between broadcast packet and unicast packet. Alternative Recursive Partition Multicasting(AL+RPM)[6] is proposed based on the RPM, adding a connectivity bit to indicate the connection situation of current node with other neighbor nodes, then judge the output direction according to the present circumstance. Although the aforementioned two approaches can applied in irregular sub-network, the shape is near convex only, which limits the scalable of the sub-network partition.

In this paper, we divide the whole network into any random shape topology, then carry out an effective broadcast scheme in each sub-network.

3 Preliminaries

Before putting forward our approach, some definitions and assumptions are made throughout the paper.

Definition 1. Oriented to the many-core processor, one PE is specified as the scheduling PE (SPE), which is responsible for real-time collection of the busy status information for all other PEs, then schedule the resource dynamically according to the phased resource requirements of the task.

Definition 2. Each task maps onto a Virtual Computing Group (VCG), which is composed of several PEs and the according Noc resources, and monopolize the VCG on the running phase of the task. Once individual PE carries out one thread of the task, all PEs in the VCG support Cache coherence protocol.

Definition 3. PEs in the VCG belong to a Logic sub-network, and in the physical implementation phase, the communication among all PEs may need to be routed through the regular sub-network, thus a Physical sub-network is formed.

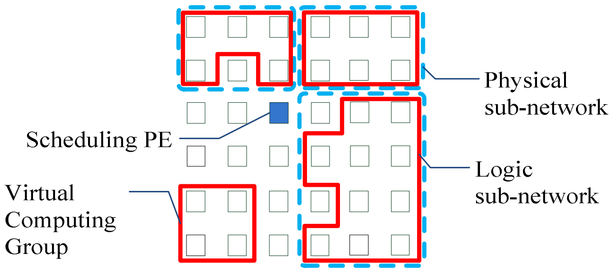


Fig. 1. The definition of Logic sub-network, Physical sub-network, SPE and VCG

Assumption 1. In each packet, destinations are encoded in bit string, as shown in Table 1, the bit which set 1 means according node is one of the destinations belongs to the Logical sub-network. At the beginning time of the routing, each router node initializes an routing table, which is used to be queried to find out a output port based on the destination list of the current packet.

Table 1. Destination list

Node	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	…
bit	1	1	1	1	1	1	0	0	1	0	1	0	1	0	0	1	…

Assumption 2. The target Noc router architecture is a 5-stage classic Virtual Channel router, which includes Buffer Write (BW)/Route Compute (RC), VC Allocation (VA), Switch Allocation (SA), Switch Traversal (ST) and Link Traversal (LT), and wormhole switching is used here. The replication unit is integrated into the basic router, which have the replication action to the coming flits of a multicast packet.

4 Deadlock-Free Broadcast XY Routing

Based on the Mesh topology, shortest path traversals are used to populate routing table at each router node. A deadlock-free broadcast wormhole method based on XY routing algorithm is proposed years before[7], which the target network is partitioned into four sub-networks, $N_{+X,+Y}$, $N_{-X,+Y}$, $N_{-X,-Y}$ and $N_{+X,-Y}$. Sub-network $N_{+X,+Y}$ include the [(i, j), (i+1, j)] and [(i, j), (i, j+1)], and the other sub-networks are just similarly to the $N_{+X,+Y}$ condition, which show as below (assumption the coordinate of source node is (x_o, y_o)),

$$D_{+X,+Y} = \{(x,y)|(x,y) \in D, x \geq x_o, y \geq y_o\}$$
$$D_{-X,+Y} = \{(x,y)|(x,y) \in D, x \leq x_o, y > y_o\}$$
$$D_{-X,-Y} = \{(x,y)|(x,y) \in D, x > x_o, y > y_o\}$$
$$D_{+X,-Y} = \{(x,y)|(x,y) \in D, x > x_o, y \leq y_o\}$$

An example illustrated in Fig.2 to show the operation of the algorithm. We set Node 21 as the source node, the other nodes which belong to the whole network are setting as the destination nodes.

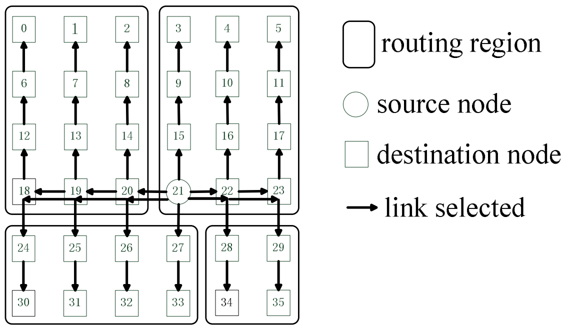


Fig. 2. Basic deadlock-free broadcast XY routing

Combining with the Logical and Physical sub-network we mentioned above, the broadcast XY routing algorithm has become more versatile. Before the final algorithm is proposed in detail, an example is given to explain the broadcast in a sub-network of any topology shape. As shown in Fig.3, Node 21 is still setting as the source node, the others as the destination nodes. The Logical sub-network is configured at the initial stage of the many-core processor system startup. According to the broadcast XY routing algorithm, the whole Logic sub-network is partitioned into four parts:

$$D\{node\ |node \in\ D_{+X,+Y}\} = \{(3,4,9,10,15,16,17,21,22,23)\}$$

$$D\{node\ |node \in\ D_{-X,+Y}\} = \{(1,2,6,7,8,12,13,14,18,19,20)\}$$

$$D\{node\ |node \in\ D_{-X,-Y}\} = \{(25,26,27,33)\}$$

$$D\{node\ |node \in\ D_{+X,-Y}\} = \{(28,29,34,35)\}$$

Notice that Node 12 and Node 13, which was not included in the Logic sub-network, still be intermediary nodes in the routing path, this is because the minimal path is chosen at each skipping according to the XY algorithm. That is to say, the mechanism achieved logical isolation of communication and broadcasting, but cannot attain physical isolation.

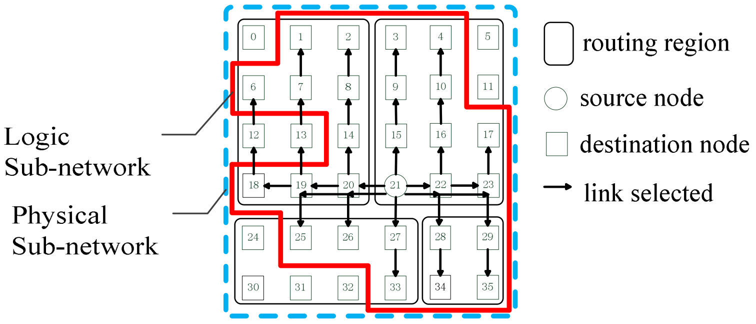


Fig. 3. Broadcast XY routing in sub-network

The whole broadcast process can be divided into three phases:

First, each routing table belong to the whole network is updated according to the configuration message, in particular, the nodes which are part of certain sub-network storage the information that which nodes are belong to the same Logical sub-network.

Second, according to the Logical sub-network information, up to four copies of the broadcast packet are generated at the source router node which each carries a destination list to show which nodes to be sent to.

Third, in each routing region, the broadcast XY routing algorithm is carried out as shown in Fig.4.

As in broadcast XY routing algorithm, replication is made as late as possible to reduce the number of replicated packets. It can be proved, this algorithm is deadlock-free, as a result virtual channels resource are saved which other related routing algorithm used more than one virtual networks to avoid deadlock for mesh-based networks.

```
//pseudo code for broadcast XY routing algorithm
//example for packets into the $N_{+X,+Y}$ region
Construct D = { $(x_i, y_i) \in D_{+X,+Y}$ }
Construct Dy = { $(x_i, y_i) \in D \mid (y_i = y_{current})$ }
Construct Dx = { $(x_i, y_i) \in D \mid (y_i \neq y_{current})$ }
If(D.sizeof() > 1)

  //replicate to current node
  If( (Dy $\neq$ 0)&& ($x_i = x_{current}$) )
      new_flit_cur = constructNewFlit(s_flit, cur_DestID);
      outport _cur = routeCompute(new_flit_cur);

  //replicate to y direction
  If( (Dy – ($x_{current}, y_{current}$) ) $\neq$ 0 )
      new_flit_y = constructNewFlit(s_flit, y_DestID);
      outport _y = routeCompute(new_flit_y);

  //replicate to x direction
  if(Dx $\neq$ 0)
      new_flit_x = constructNewFlit(s_flit, x_DestID);
      outport_x = routeCompute(new_flit_x);
```

Fig. 4. Pseudo code of broadcast XY routing algorithm

5 Simulation Environment

To evaluate the performance of our broadcast XY routing mechanism, the Gem5 Simulator System [8] is used, which is a modular platform for computer system architecture research, encompassing system-level architecture as well as processor micro-architecture. Gem5 integrates the cycle-accurate Garnet interconnection network model, which models a classic five-stage pipelined router with virtual channel (VC) flow control. The fixed-pipeline mode, which belongs to Garnet network [9], is intended for low-level interconnection network evaluations and models the detailed micro-architectural features of a 5-stage Virtual Channel router with credit-based flow-control, which is chosen as the research platform to continue our work.

In Fig.5, we describe the detail communication process between Network Interface (NI) and classic Garnet Router in fixed-pipelined model, and the communication patterns between routers are just similar as this condition. As we can find, the current network model does not have hardware multicast support within the network, so a heavy work was done for adding the broadcast scheme into the Garnet network.

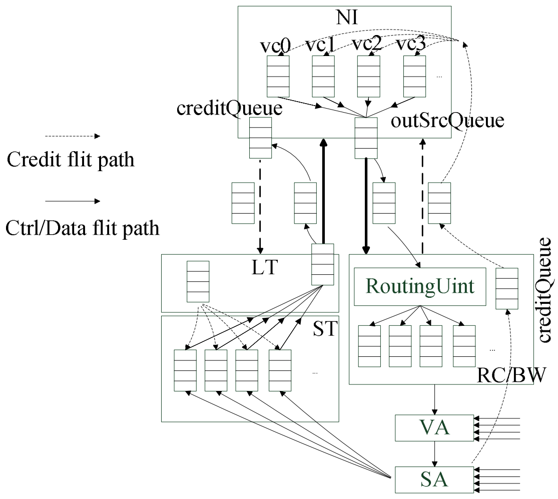


Fig. 5. Basic architecture of fixed-pipeline mode in Garnet

Besides, each module in the router launches a waking up action via creating a Ruby-Event in advance, thus a Ruby-Event-Queue which integrated into the fixed-pipeline mode need to be maintained. To add hardware multicast support in the router, we add several Ruby-Events artificially to let the modules wake themselves up.

6 Performance Evaluation

The Gem5 Simulator provides a framework for simulating the interconnection network with controlled inputs (Ruby Network Tester), which is especially useful for Garnet network testing/debugging. We set the multiple broadcast unicast as the comparison object with broadcast XY mechanism, building a 4×4 mesh-based Noc test environment, traffic ratio of broadcasting(B-ratio) to the whole Noc communication is ranging in {0.05:1, 0,1:1, 0.15:1, 0.2:1}, and focusing the average network latency as main contrasting parameter. As shown in Fig.6, when injection rate is low, the both broadcasting manners do not show much difference with the variation of B-ratio in the average packet latency. When the B-ratio increases, the broadcast XY increase much slower than that of multiple unicast, this is because less packets are replicated using the broadcast XY mechanism, thus less network congestion occurring.

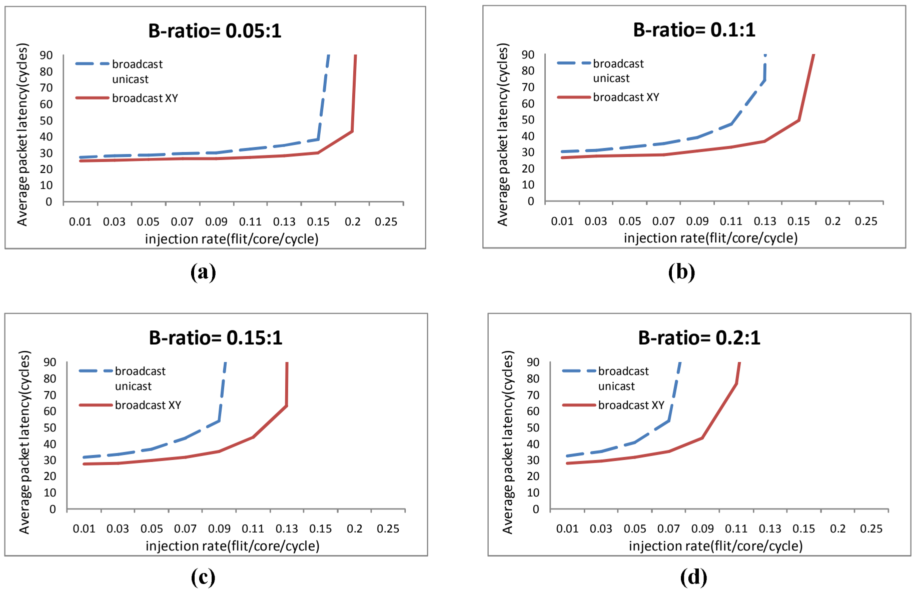


Fig. 6. Latency result with B-ratio set to (a) 0.05:1, (b) 0.1:1, (c) 0.15:1, (d) 0.2:1

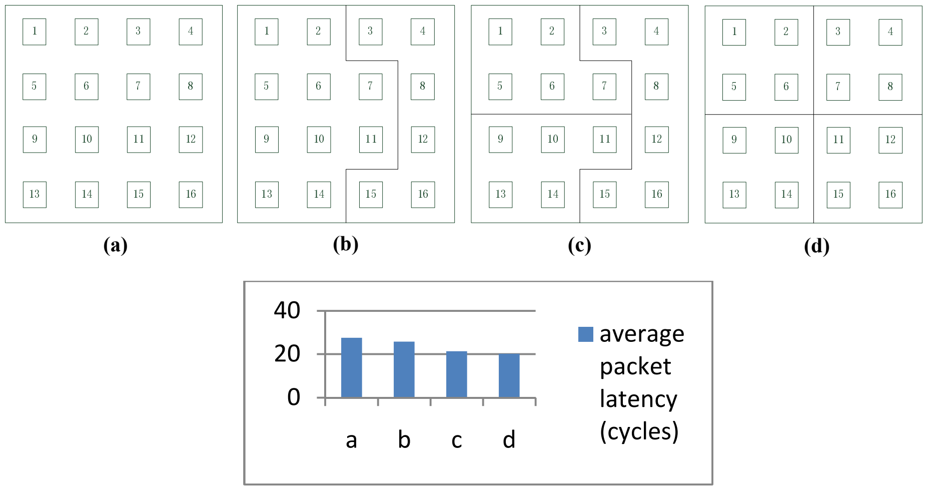


Fig. 7. Broadcast XY in sub-network (B-ratio = 0.1:1, injection rate = 0.05)

We observe the advantage which the sub-network partitioning method introduces into the Noc. Using the broadcast XY mechanism to the one-to-all communication, and for the network test mode, the B-ratio is setting to 0.1:1, injection rate is 0.05. For the below four sub-network approaches in Fig.7, the average packet latency shows a decreasing trend. As can be expected, with the increasing of the Noc scale, the effect will be more obvious.

7 Conclusion

The broadcast XY mechanism bases on multicast XY routing algorithm, which region segmentation is proceeding at the beginning of routing phase according to the current source node. Additionally, we introduce the concept of Logical sub-network and Physical sub-network, which the first one indicates the PEs which belong to the VCG and communication sets up among all of them, and the second one limits routing region for any message comes out from the PEs. For any irregular topology Logical sub-network, there always be a corresponding Physical sub-network wrapping it. As mentioned above, traffic isolation between nodes in and out of Logical sub-networks is not really achieved, at the routing stage, intermediary nodes in the routing path are only responsible for transferring the packets, which will not get one copy of the packet to PE. Our future research will focus on minimizing the impact of communication and broadcasting inside VCG to other PEs outside the VCG, thus a minimal Physical sub-network will be demanded.

Acknowledgements. Our work is supported by the IBM Shared University Research (SUR)，System-level Virtualization Based on Reduced Hyper Multi-core Architecture, SUR201102X.

References

1. Jerger, N.D.E., Peh, L.-S., Lipasti, M.: Virtual Circuit Tree Multicasting: A Case for On-Chip Hardware Multicast Support. In: 35th International Symposium on Computer Architecture, ISCA 2008, pp. 229–240 (2008)
2. Wang, L., Jin, Y., Kim, H., Kim, E.J.: Recursive partitioning multicast: A bandwidth-efficient routing for Networks-on-Chip. In: 3rd ACM/IEEE International Symposium on NoCS 2009, pp. 64–73 (2009)
3. Abad, P., Puente, V., Gregorio, J.-A.: MRR: Enabling fully adaptive multicast routing for CMP interconnection networks. In: IEEE 15th International Symposium on HPCA 2009, pp. 355–366 (2009)
4. Rodrigo, S., Flich, J., Duato, J., Hummel, M.: Efficient unicast and multicast support for CMPs. In: 2008 41st IEEE/ACM International Symposium on Microarchitecture, MICRO-41, pp. 364–375 (2008)
5. Flich, J., Rodrigo, S., Duato, J.: An Efficient Implementation of Distributed Routing Algorithms for NoCs. In: Second ACM/IEEE International Symposium on NoCS 2008, pp. 87–96 (2008)

6. Wang, X., Yang, M., Jiang, Y., Liu, P.: On an efficient NoC multicasting scheme in support of multiple applications running on irregular sub-networks. Microprocessors and Microsystems 35(2), 119–129 (2011)
7. Lin, X., McKinley, P.K., Ni, L.M.: Deadlock-free multicast wormhole routing in 2-D mesh multicomputers. IEEE Transactions on Parallel and Distributed Systems, 793–804 (1994)
8. Binkert, N., Beckmann, B., Black, G., Reinhardt, S.K., Saidi, A., Basu, A., Hestness, J., Hower, D.R., Krishna, T., Sardashti, S., Sen, R., Sewell, K., Shoaib, M., Vaish, N., Hill, M.D., Wood, D.A.: The gem5 simulator. SIGARCH Comput. Archit. News 39(2) (2011)
9. Agarwal, N., Krishna, T., Peh, L.-S., Jha, N.K.: GARNET: A detailed on-chip network model inside a full-system simulator. In: IEEE International Symposium on ISPASS 2009, pp. 33–42 (2009)

A Quick Method for Mapping Cores Onto 2D-Mesh Based Networks on Chip*

Zhenlong Song, Yong Dou, Mingling Zheng, and Weixia Xu

School of Computer,
National University of Defense Technology, Changsha China
songzhl@sina.com

Abstract. With the development of NoC, it becomes an urgent task to efficiently map a complex application onto a specified NoC platform. In the paper, an approach which is called constraint-cluster based simulated annealing (CCSA) is proposed to tackle the mapping problem in 2D-mesh NoC in order to optimize communication energy and execution time. Different from other methods, the relationship among cores that are patitioned into several clusters is considered in our method and according to the relationship constraints are set. Experimental results show that the proposed approach gets shorter execution time with lower energy consumption compared with others algorithms. In VOPD application (4x4), the reduction of execution time is about 75.64% combing the normal simulated annealing. In greater application (8x8 vodx4) the CCSA can save 68.89% . The energy consumption is the lowest among all the compared algorithms.

Keywords: network on a chip (NoC) Application mapping Simulated Annealing.

1 Introduction

With the advance of Semiconductor, the number of transistors available will be more than 4 billion at more than 10Ghzspeed[1].This allows that more than dozens of IP-blocks can be integrated in a single chip. Different from dedicated wires and shared-medium busses, the network on a chip (NoC) provides a high performance chip-level communication with regularity and modularity.

In literature[2] formulated the 32bit's ALU power consumption is about 0.3 pj,while translate 32 bits over 10mm need 17pj,the consumption of translation is more the 50 times than Alu. So how to decrease the energy consumption of a single chip is a challenging job in NoC platform. In the design flow for NoC architecture , the step of Mapping which determines which nodes host which cores is very important These choices have significant impacts on energy, area and performance metrics of the system[3],[4].

* This work was supported by the National High Technology Research and Development Program of China(No.2012AA01A301).

The core-to-node mapping problem is a NP-Hard optimization problem [3].For the node number of N, the result of mapping will be N! . The computation and search space are very huge. For example, mapping 16 cores onto 16 nodes has a search space of 16! ,that is about 2.1x1013.The count of computation of each map is about 435 Flop . Thus the computation is about 1016. Suppose the computer is Intel 3.0GHZ four cores, it will cost about 120 hours to fulfill the searching .So some efficient analytical model which can be used to find nearly optimal solutions in reasonable time must be presented.

In this paper, we present a rapid algorithm named Constraint-cluster based Simulated Annealing Approach (CCSA) to minimize the power consumed and overall runtime. The node can be partitioned into some cluster by calculating the communication difference between nodes , and according to the partition some searching restriction can be set . A cluster partition can make the problem more easy. Just as what has discussed aboved, for 16 cores/nodes, if they can be portioned into some clusters, the 16! problem would be a $N!\times C_1!\times C_2!\times\ldots\times C_N!$ problem,which is smaller than 16! to search. In the course of searching,the searching space is limited by the cluster restriction, By the partition and constraint the algorithm can resolve the mapping quickly without reducing the quality of solution. This paper chooses a two dimensional mesh interconnection which is very simple from a layout perspective and the local interconnections between resources and switches are independent of the size of the network. Moreover, routing in a two dimensional mesh is very easy resulting in potentially small switches, high bandwidth, short clock cycle, and overall scalability.

The rest of the paper is organized as follows, We define the mapping problem and give an overview of our Constraint-cluster based Simulated Annealing Approach(CCSA).Then the CCSA in details is described . The experimental results are reported in the last.

2 Realted Works

Now NoC mapping problem has become a broad topic of research and development. In [3], Hu and Marculescu presented a static mapping a branch-and-bound algorithm. The main goal of the approach is to reduce the overall power consumption by decreasing the consumed energy on communication. Literature [5] presents a two-step genetic algorithm to map an application, described on a mesh-based NoC architecture with the objective of minimal execution time. Zhou et al. apply a queue model to calculate the latency of net. In paper [7] a communication model is accept to calculate the latency of communication. In [6] another communication model is proposed to calculate the latency. In [12] PMAP, a two-phase mapping algorithm for placing the clusters onto processors is presented. The results of PMAP algorithm are shown to have low communication costs. In [11] Murali and De Micheli proposed a rapid algorithm NMAP that maps the cores onto mesh NoC architecture under bandwidth constraints. In general those algorithm dose not consider the characteristics of nodes in the no-heuristic search, so the search methods must balance the run time and the quality of result (such as branch-and-bound or back-tracking). A heuristic method may consider the character of node to search near-optimal solutions, which is different from the model to the NoC. Recently some paper want to accelerate the approach by using partition .In the paper [8], Lu presents a cluster-based Simulated Annealing according to the node "distance". By clustering, the method can make the search more efficiently. In paper [9],Lin presents an approach named HMMap that maps large numbers of ip cores onto 2d-Mesh. It bases hierarchical mapping method to speed up the approach .It also partitions the nodes into some clusters in order to minimize the run-time.In [18], propose a clustering based relaxation for integer Linear Programming determining the optimum mapping.

3 Approach Overview

The design flow of CCSA includes two parts: (1) getting the initial mapping and cores restrictions; (2) getting the solution by using the annealing step and the restrictions. The aim of our approach is to minimize the cost of communication and run time. An overview of the CCSA is shown in Figure 1.

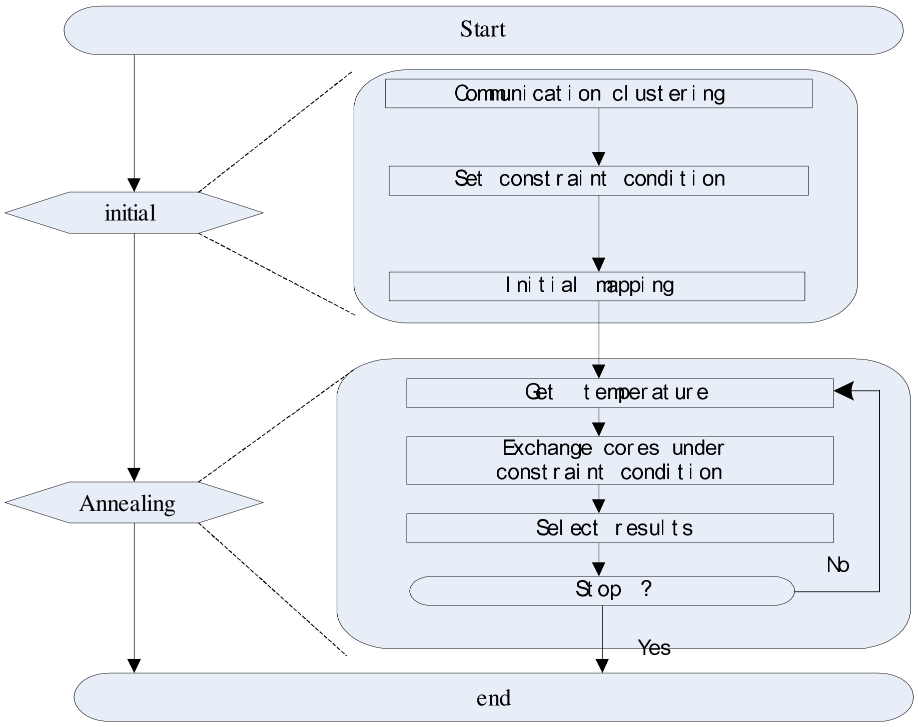


Fig. 1. The Constraint-cluster based Simulated Annealing (CCSA) Flow

The first part includes three steps .First we partition the core graph according to the communication volume between cores that is defined as the sum of the volume in an out. In the past they assumed that cores of large communication volume with each other should be placed on the neighboring tiles to optimize the communication energy and latency[17][13] .In this paper we say no. For example ,accroding to a 2*2 mapping ,assumed that V(AB) is the largest volume ,which core should be more closed to node A in Figure 2, B or D? (V(AB) is the sum of volume form A to B and from B to A).

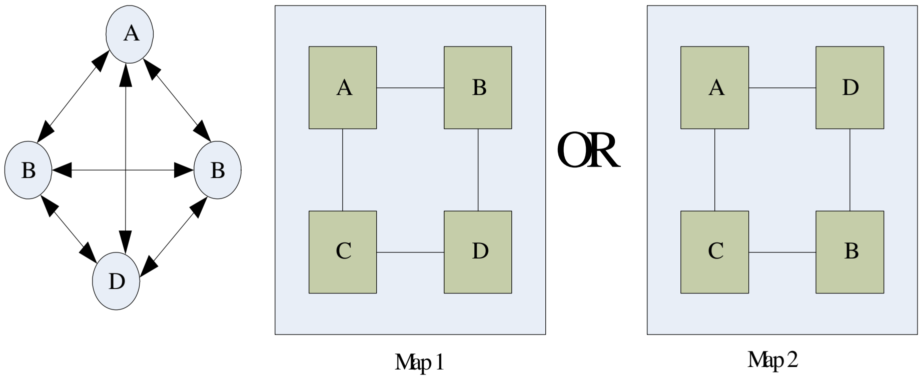


Fig. 2. The clustering example

The difference of communication cost between Map 1 and Map 2 is that:

$$\text{Cost(map1)-Cost(map2)= } (V_{(AD)}-V_{(DC)})-(V_{(AB)}-V_{(BC)}) \tag{1}$$

If (1) >0 ,That means the cost of map 2 is smaller than map1. even the volume of V(AB) is the biggest .The node D should be more close to node A than node B. so we assume that core l is more close to core k than core j if that :

$$(V(lk)-\sum_{i=0}^{i=n,i\neq j,i\neq k} V(il)\Big/\sqrt{L(l)})-(V(jk)-\sum_{i=0}^{i=n,i\neq l,i\neq k} V(ik)\Big/\sqrt{L(k)})\succ 0 \tag{2}$$

According to our assuming, the communication class is that the cores should be closed to each other. A core l should be partitioned to a communication class C(a,b,c…)if that :

$$\begin{aligned}&(\sum_{k\notin C_{(a,b,c\ldots)}} V(lk)\Big/\sqrt{(L_C(k))}-\sum_{i=0}^{i=n,i\notin C_{(a,b,c\ldots)}} V(il)\Big/\sqrt{L(l)-L_C(l)})-\\&(\sum_{k\notin C_{(a,b,c\ldots)}} V(jk)\Big/\sqrt{(L_C(j))}-\sum_{i=0}^{i=n,i\notin C_{(a,b,c\ldots)}} V(ij)\Big/\sqrt{L(j)-L_C(j)})>0\end{aligned} \tag{3}$$

According to the partition , some restrictions can be set . The detail description of this step is presented in the following Section. The last step is to get an initial mapping.

The Second part performs the annealing mapping to get optimal solutions .This annealing strategy cuts down the search space for the restrictions. That will be explained in the following section.

3.1 Problem Formulation

The core-to-node mapping problem is a specific graph embedding problem .To formulate our problem ,We assume that[8]:

- The network topology is 2D mesh. The mesh has bidirectional links with property of bandwidth between nodes.
- The number of cores is not greater than the number of nodes and one core is mapped to exactly one node.

Definition 1. A core application graph (CAG) which captures the communication between IP cores of the application is a directed graph,CAG(C ,A) , in which each vertex $c_i \in C$ represents an IP core and the directed arc $a_{(i--j)} \in A$ represents a direct communication from c_i to c_j . Each $a_{(i--j)} \in A$ is associated with a bandwith requirement $cb_{(i,j)}$ as its weight and has the following property : $V_{(i,j)} = V_{(j,i)}$ is the sum of arc volume from vertex c_i to c_j and from c_j to c_i,which represents the communication volume between c_i and c_j.

Definition 2. An architecture node graph (ANG) which reflects the architecture and connectivity of NoC is a directed graph ANG (T , R) , in which each vertex t_i represents one tile in the NoC architecture, and each directed arc $r_{(i--j)}$,represents a minimal path from tile t_i to tile t_j . Each $r_{(i--j)}$ has the following properties:

- $BW_{(i,j)}$ is the available bandwidth of link $r_{(i--j)}$.
- $e(r_{(i--j)})$ stands for the energy consumption of sending one bit from t_i to t_j.denoted as $e_{i,j}$.
- $d(r_{(i\rightarrow j)})$ represents the communication latency of one bit from t_i to t_j .
- $h_{i,j}$ denotes the distance from tile t_i to t_j.

Using these definitions, the mapping of the core application graph CA(C , A) onto the architecture node graph AG(T , R) is to find a mapping function[14]: map C→T. The routing algorithm ensures minimal path and deadlock free. The objective function minimizes overall communication volume and run-time.

$$\text{map } C \rightarrow T \text{ , s.t. } \text{map}(c_i)=t_j \quad (c_i \in C, \quad t_j \in T)$$

$$\text{aim min } \{E_{systme}, T_{runtime}\}$$

Where E_{system} represents the total system communication energy and $T_{runtime}$ is the approach running time.

3.2 Energy Model

The system communication energy is defined by the following equation:

$$E_{system} = \sum_{i=0,j=0}^{i=M\times N, j=M\times N} e_{i,j} \times v_{i,j} \Big/ 2 \quad (4)$$

$$e_{i,j} = E_{S_{bit}} \times (h_{i,j} + 1) + E_{L_{bit}} \times h_{i,j} \quad (5)$$

where E_{Sbit}, E_{Lbit} represent the energy consumed by sending one bit of data through the switch and on the links between tiles , respectively. Commonly the E_{Lbit} of a noc can be considered as a constant. The E_{Sbit} include two parts, One is the energy consumption of cross-bar, and another is energy consumption of the fifo for the date reading,writing and storing. If there is no any network contention the time of store can be considered a constant. So the energy consumption also can be considered as a constant. Thus the $e_{i,j}$ can be described as follows.

$$e_{i,j} = \alpha \times h_{i,j} + \beta \quad (6)$$

From the equation *(4)* and *(6)*,we can draw the conclusion of power consumption :

$$E_{system} \propto \sum_{i=0,j=0}^{i=M\times N, j=M\times N} h_{i,j} \times v_{i,j} \quad (7)$$

3.3 Constraint-Cluster Based Simulated Annealing

A Initial Mapping

As depicted in Figure 1,the initial mapping includes three steps, core clustering, clustering constraint condition and initial mapping.

The first step is to cluster communication cores by the communication volume of cores. According to inequation *(3)* , the core can be partitioned, but the computational complexity is high. In our partition we use the difference between the maximal volume and the second maximum volume to partition the communication cores. So every cluster has two parameters, the closest cluster and the communication difference:

$$C(a) = (C(A), D_{iff}) \quad (8)$$

$$D_{iff} = \sum_{i=0}^{i=M\times N(b_i \in C(A))} V(ab_i) - V_{second}(a) \quad (9)$$

The C(A) implies the core a would like to close with the communication cluster C(A), the Diff implies the possibility that the a will be partitioned to C(A). According to the core cluster some constraint conditions can be set for each core.

$$\text{Con(a)=(C(A),Dis)} \quad (10)$$

The C(A) implies the cores that core a would like to close with. Dis implies the max distance of core a and the cores belonging to C(A).

For example,as shown in Fig 3:

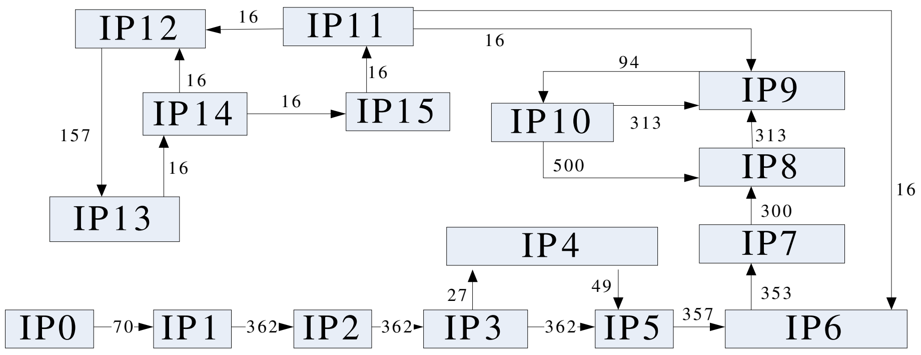


Fig. 3. The Video Object Plane decoder with bandwidth demand in MB/s

According to (8), we can get:

C(0)=(C(1),70), C(1)=(C(2),292), C(2)=(C(1),0)…C(8)=(C(10),187),C(10)=(C(8), 93)…so the first cluster that would be C(1,2).C(8,10) C(12,13) and some constraint can be set,just as:

*Con(1)=(C(2),1) Con(8)=(C(10),1),C(12)=(C(13),1),C(13)=(C(12),1).*Next the C(1,2) can be considered as one node, and *C(0)=(C(1,2),70),C(1,2)=(C(3),292), C3=(C(1,2),0)…* in this way the node can be partitioned and the constraint can be set again. Until there is no more than four clusters or there will be only on cluster in the next step.

According to core clusters and constraint conditions, an initial mapping can be obtained.

Annealing

Besides the initial mapping, the annealing will also be affected by the constraint conditions. We first describe the general annealing process and then the constraint-cluster based annealing.

To find the optimum solution of searching, the annealing technique is a cooling process[10]. Based on the initial map, the exchange of positions between two cores reflects the temperature, as described in the following equation:

$$\Delta E=\text{cont}(\text{Map}_i)-\text{cost}(\text{map}_{(i+1)}) \tag{11}$$

If ΔE is greater than zero, it is a lower temperature move and the result will be accepted. If ΔE is less than zero, it is a high temperature move and the result will be randomly accepted. For the problem of mapping cores onto 2D meshes, the number of annealing stages corresponds to the cores number. For a MxN mesh network ,the number of stages is equal to MxN.

Constraint-cluster based Annealing follows the annealing procedure described above .However in each stage. annealing must determine whether the cluster moves are allowed. For example core l core m is *Con(l)=(C(m),3)*, that implies the constraint condition is that the distance of *l* and *m* is less than 3, the core *l* only can be exchanged with the cores, that the distance between those and *m* less than 3. This implies that in each stage, the number of exchange is reduced and the near-optimal solutions can be got in a shorter time than annealing.

4 Experiments

To validate the advantages of our approach in runtime and quality of solution,in this section, we present the result of the execution of CCSA on benchmark application,Video Object Plane Decoder (VOPD[11]) with 16 IP cores shown in Fig.3.We also compare these results with those of previous mapping algorithms such as CGMAP[15] ,CSA[8] ,NMAP[11] ,PMAP[12],BMAP[16] ,PBB[3] C-ILP[18] .et.al using the same routing and scheduling characteristics, which also use VOPD as the benchmark application. In order to validate the advantages in runtime we have also implemented the simulated annealing without clustering and constraint condition. All these experiments are executed on a server with a Intel(R) Xeon(R) X5355 @ 2.66GHz processor and 6GB memory.

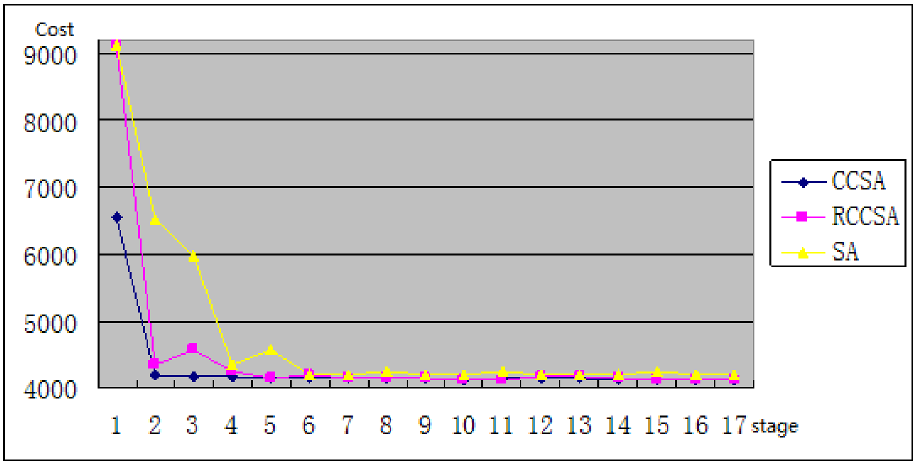


Fig. 4. CCSA RCCSA and SA mapping results

In this section we first compare the results of our approach mapping the VOPD application onto a 4x4 mesh. Three methods are implemented, including CCSA, RCCSA and SA. CCSA uses an initial mapping with the CCSA. RCCSA uses a random mapping with CCSA. SA uses the same random mapping with SA. In Fig.4, the

X axis represents the number of annealing stages, and the Y axis shows the average cost of accepted solutions. From the curve, we can see that the CCSA ,RCCSA and SA converge to the equilibrium state .The RCCSA and CCSA achieves the lower cost (4103) than the SA (4200) in the last annealing stage.

As shown in Fig 5, we compare our results with other eight mapping algorithm such as NMAP,CSA,CGMAP,PBB .et.al .Obviously our proposed algorithm performs better than the other nine mapping algorithms in VOPD ,considering the communication costs.

The CCSA takes 19 ms to finish .The RCCSA takes 28ms while the SA takes 78ms. Except [8][18], there is no running time reported. In [8] that CSA approach takes 387 seconds and SA takes 497 seconds. In [18], about 114.6 second was used to find the answer. For the different of computer we can not compare the running time directly. In our experiment a reduction of run time is about 75.64%..The main reason is that ,firstly the partition make the problem more easy to resolved,secondly the constraint conditions forbid a lot of switching and computing ,so the result can find so quickly.

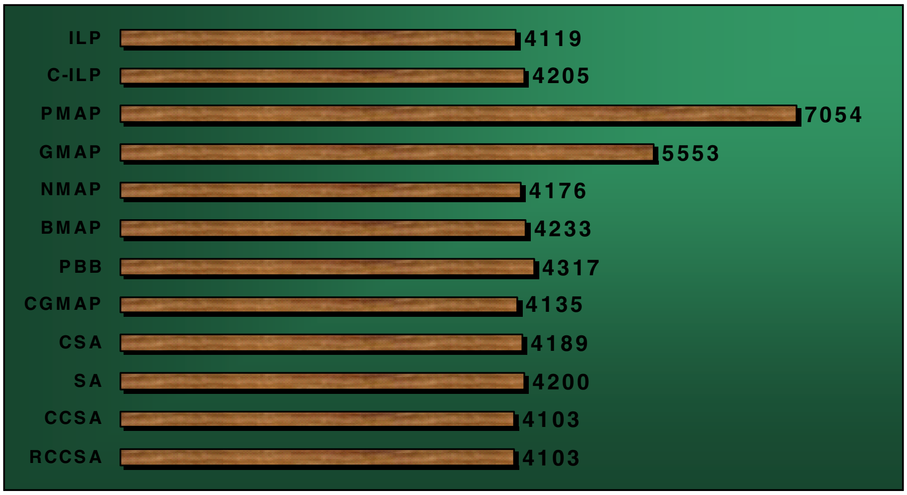


Fig. 5. Comparison between the communication costs of mapping algorithms

Except [8], the studied problem sizes are no more than 16 cores. To show the performance of constraint-cluster based approach to a bigger network size, we conduct experiments on an 8x8 mesh. We create an application by quadrupling the VOPD. The results are shown in Fig. 6.The CCSA takes 0.913 seconds to get the lowest cost 16498 while SA takes 2.935 seconds to get the cost 16514. The reduction of running rime is 68.89%. Paper [8] uses clusters and annealing too ,the reduction is only 22%.

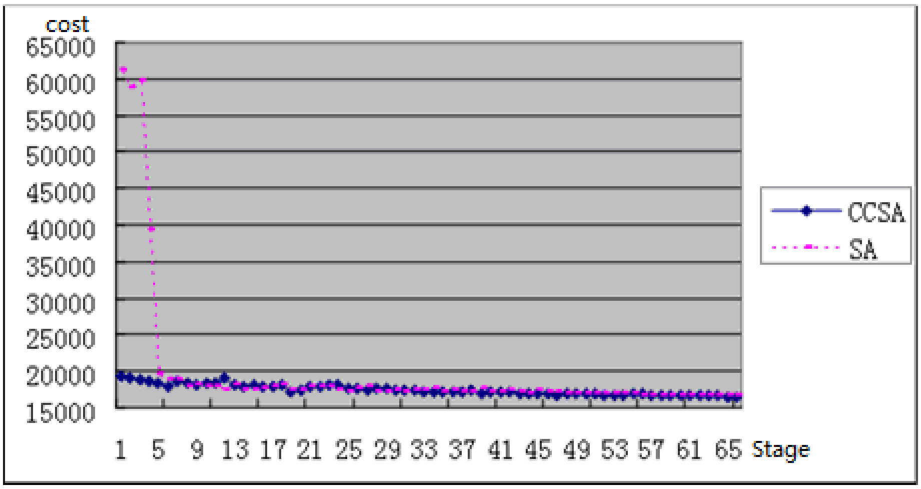
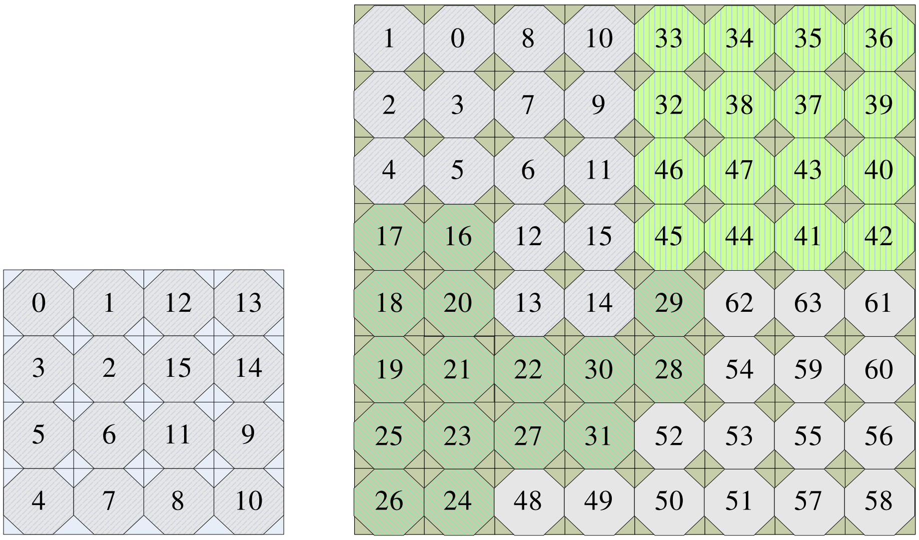


Fig. 6. CCSA and SA mapping results on 8x8 mesh

As the result shown , The result of 8x8 is not so efficient as 4x4 map, Include two aspect ,one is that the reduction of running time is lower than 4x4 mapping ,annother is that cost is higher than four VOPD mapping . The main reason is that the space of constraint conditions is large, and more time is spend on searching constraint. So some efficient method must be adopted to manage the constraint conditions.



a, VOPD mapping result

b, 4XVOPD mapping ressult

Fig. 7. Mapping result

5 Conclusion and Future Work

In the paper, we introduce the idea of combining the communication clustering technique with the simulated annealing to further leverage the performance of SA. The proposed algorithm CCSA appears to work quite better than other efficient mapping algorithms introduced in the previous work. However,with the increase of ip number the the constraint condition is becoming more and more difficult to set , In our future work we will do more works on how to set and manage the constraint conditions. So there is a rather huge design space to explore.

References

1. International Technology Roadmap for Semiconductors (ITRS) (2007), http://www.itrs.net/
2. Dally, W.J.: Computer Architecture is all about interconnect (it is now and will be more so in 2010), HPCA Panel (February 4, 2002)
3. Hu, J., Marculescu, R.: Energy aware communication and task scheduling for network on chip architectures under real time constraints. In: Proc. DATE 2004, pp. 234–239. IEEE, Paris (2004)
4. Nickray, M., Dehyadgari, M., Kusha, A.: Power and delay optimization for network on chip. In: ECCTD 2005, pp. 273–276. IEEE, Cork (2005)
5. Lei, T., Kumar, S.: A two-step genetic algorithm for mapping task graphs to a network on chip architecture. In: DSD 2003, pp. 180–187. IEEE, Antalya (2003)
6. Zhou, W.B., Zhang, Y., Mao, Z.G.: An application specific NoC mapping for optimized delay. In: DTIS 2006, pp. 184–188. IEEE, Gammarth (2006)
7. Kiasari, A.E., Hessabi, S., Sarbazi-Azad, H.: PERMAP: A Performance-Aware Mapping for Application-Specific SoCs. In: International Conference on Application-Specific System Architectures and Processors, ASAP 2008, pp. 73–78 (2008)
8. Cluster-based Simulated Annealing for Mapping Cores onto 2D Mesh Networks on Chip. In: 2008 11th IEEE Workshop Design on Design and Diagnostic of Electronic Circuits and Systems, pp. 1–6 (2008)
9. Lin, H., Zhang, L., Tong, D., Li, X., Cheng, X.: A Fast Hierarchical Multi Objective Mapping Approach for Mesh Based Networks on Chip. Acta Scientiarum Naturalium Universitatis Pekinensis 44(5) (September 2008)
10. Catoni, O.: Metropolis, simulated annealing, and iterated energy transformation algorithms: Theory and experiments. Journal of Complexity 12(4), 595–623 (1996)
11. Murali, S., Micheli, G.D.: Bandwidth-constrained mapping of cores onto NoC architectures. In: Proceedinlg of Design Automnation, and Test in Europe Conference, pp. 896–901 (2004)
12. Koziris, N., Romesis, M., Tsanakas, P., Papakonstantinou, G.: An Efficient Algorithm for the Physical Mapping of Clustered Task Graphs onto Multiprocessor Architectures. In: Proceedings of the 8th International Conference EuroPDP, Rhodos, Greece, pp. 406–413 (January 2000)
13. Marcon, C., Borin, A., Susin, A., et al.: Time and energy efficient mapping of embedded applications onto NoCs. In: Proceedings of Asia and South Pacific Design Automation Conference, Shanghai, pp. 33–38 (2005)

14. Hemani, A., Jantsch, A., Kumar, S., et al.: Network on a chip: An architecture for billion transistor era. In: Proceedings of the IEEE NorChip Conference, Turku, pp. 166–173 (2000)
15. Moein-darbari, F., et al.: Evaluating the Performance of a chaos genetic algorithm for solving the Network on Chip Mapping Problem. In: 2009 International Conference on Computational Science and Engineering, pp. 366–373 (2009)
16. Shen, W.T., Chao, C.H., KLien, Y., Wu, A.Y.: A New Binomial Mapping and Optimization Algorithm forReduced-Complexity Mesh-Based On-Chip Network. In: Proceeding of the 1st IEEE International Symposium on Networks-on-Chip (NOCS 2007), Princeton, New Jersey, pp. 317–322 (May 2007)
17. Rhee, C.E., Jeong, H.Y., Ha, S.: Many-to-Many Core-Switch Mapping in 2-D Mesh NoC Architectures. In: Proceedings of IEEE International Conference on Computer Design, San Jose, CA, USA, pp. 438–443 (October 2004)
18. Tosun, S.: Cluster-based application mapping method for Network-on-Chip. In: Advances in Engineering Software, vol. 42, pp. 868–874 (2011)

A Combined Hardware/Software Measurement for ARM Program Execution Time*

Liangliang Kong[1] and Jianhui Jiang[2]

[1] Department of Computer Science and Technology,
Tongji University, Shanghai 201804, China
liangliang_kong@yahoo.com.cn
[2] School of Software Engineering, Tongji University, Shanghai 201804, China
jhjiang@tongji.edu.cn

Abstract. In present there is no accurate end-to-end dynamic measurements for ARM program execution time, because the measurement results given by hardware counters in ARM microprocessors are not precise enough and the timing cost of instrument methods is difficult to be calculated. Therefore, this paper proposes a combined hardware/software measurement for ARM program execution time. It sets the precision of measurement in the system boot loader code, encapsulates the access to timers in the kernel of Linux, and then measures the execution time of the program by the timer and its corresponding interrupt during the execution of the program. Experimental results have shown that comparing with instrument methods and hardware counters, our method is an efficient way to obtain accurate and precise execution time measurements for ARM programs. Additional experiments performed by the combination of curve fitting techniques and our method have shown the method can be used to predict the execution time of program under different input data.

Keywords: execution time measurement, performance evaluation, real-time system, PWM timer, ARM microprocessor.

1 Introduction

Most embedded systems are real-time systems, so real-time is a very important performance metric for embedded systems. Real-time systems, as well as their deadlines, are classified by the consequence of missing a deadline. The goal of a hard real-time system is to ensure that all deadlines are met, but for soft real-time systems the goal becomes meeting a certain subset of deadlines. In the process of embedded system development, once the processor architecture is determined, accurate execution time estimation of programs is important for scheduling analysis and hardware/software partition [1-2].

* This work has been supported by the National High Technology Research and Development Program of China (Grant No.2009AA011705) and the National Natural Science Foundation of China (Grant No. 60903033).

The dynamic measurement is an important kind of techniques of execution time estimation. In end-to-end dynamic measurements, accessing to hardware counters like the time stamp counter (TSC) in X86 is a convenient and efficient way to obtain execution cycles of programs. However, there is no appropriate hardware counters like TSC to offer execution cycles for ARM programs. ARM only provides a real-time clock (RTC) to offer the system time, which works with the frequency of 32.768 KHz. So the precision of timing measurement RTC could give is only 10^{-4}s [3]. For measurements with higher precision or programs whose execution time is less than 10^{-4}s, the RTC can not meet the requirement of execution time measurement. The instrument method is another type of end-to-end measurement. It reads the system time by invoking system calls from the kernel space and calculates the execution time estimate in the user space. The precision of timing measurement the instrument method could give is 10^{-6}s. Though the instrument method could obtain more precise timing measurement than the RTC, the timing measurement is not accurate because of timing cost consumed by it. Therefore, in this paper we propose a combined hardware/software measurement for ARM program execution time. It sets the precision of measurement by configuring frequencies of hardware related to the pulse width modulation (PWM) timer in the system boot loader code, accesses the PWM timer in the kernel by transforming physical addresses of the timer into virtual ones, and measures the execution time of program by the PWM timer and its corresponding interrupt. Experiments have shown it is an efficient way to obtain accurate timing measurement with high precision for ARM programs.

2 Related Work

The study of the execution time estimation of programs began in 1980s. A lot of work has been done and they can be classified into three categories: static analysis, dynamic measurement and simulation [1-2]. The static analysis derives the execution time of program by analysis of target processor architecture features and the CFG of program. It is often used to estimate the worst-case execution time of program. It considers the optimization of compiler and features of the target architecture, so the execution time of program it estimates is accurate. But it depends on rich architecture experience of estimators [4-7]. The dynamic measurement executes the program under estimation on the given hardware for some set of input data or all input data and obtains timing measurements or their distribution [1]. Though in most cases it is impossible to perform exhaustive measurements to obtain the worst-case execution time or the best-case one, the dynamic measurement is also popular to be used for estimating execution time for soft real-time systems [8-9]. The simulation method generates a detailed model of the target processor architecture or the target system by tools to estimate the execution time of program. Though its estimation of execution time is accurate, it is resource and time costly [10-13].

According to the granularities of the objects to be measured, the dynamic measurement can be classifies into two categories: the end-to-end measurement and the measurement based on CFG partition.

In the measurement based on CFG partition, firstly, the CFG of program is partitioned into sub-paths which correspond to program segments. Secondly, generate input data for each sub-path to drive its execution and measure its execution time. Finally, according to the topological order of sub-paths in the CFG, calculate the execution time of program [14-17]. The algorithms of automatic input data generation include heuristics, model checking and so on [18-20]. Since the measurement based on CFG partition focuses on the execution time of sub-path, the dependent and mutually exclusive relationships between sub-paths have been ignored or may not be considered seriously in calculation. Thus, the execution time it estimates is often overestimated comparing with the observed timing measurement [18].

However, since the end-to-end measurement focuses on the execution time of a whole program and gives the coverage in terms of all relationships between sub-paths in CFG, the timing measurement it generates is more accurate. There are two types of end-to-end measurement, i.e. hardware counters and instrument methods. As described in Section 1, ARM provides RTC instead of TSC in X86 to offer the system time. Since the precision of timing measurement RTC could give is only 10^{-4}s, it can neither meet the measurement requirements of high precision nor measure the program whose execution time is less than 10^{-4}s. The instrument method reads the system time by invoking system calls from kernel space and calculates the execution time estimate in user space. In most cases of instrument, the system call is invoked by application program interface (API) in the C library or the library of the instrumentation tool. For example, the API gettimeofday in the C library invokes the system call sys_gettimeofday, which reads the system wall-clock time maintained by jiffies with the precision of 10^{-6}s in kernel mode and copies the result to the user space. Then the user process could read the system time in user mode. During the process, once the system call sys_gettimeofday is invoked, it will request a software interrupt that transfers control to the kernel code. After switch to kernel mode, the processor must save all its registers, copy the parameters of the user process in the user space to the kernel space, dispatch execution to the proper kernel function, read current system time, and copy the result to the user space. These operations take at lest 1×10^{-6}s. During the process of the software interrupt handle routine, if another interrupt request with higher priority occurs, the processor will halt current interrupt handle routine and transfer control to another interrupt handle routine. The execution of the halted routine will not be resumed until the routine of the interrupt request with higher priority is completed. In this case, it is often called the nested interrupts. The execution time of the software interrupt handle routine, the possible delay of time incurred by nested interrupts, and the execution time of the instrument instructions constitutes the timing cost of the instrument method [21]. The timing measurement of the instrument method is not accurate because of the timing cost. Since the instrumentation tools like PIN encapsulate the system calls in their APIs in the same way of the C library, the timing measurement they observed is not accurate as well.

Since the timing cost of the instrument method is hard to be calculated and derived from the measurement result and the precision of measurement provided by RTC is not high enough, we proposes a combined hardware/software measurement method for accurate execution time estimation of ARM programs. Its measurement precision can be set as 10^{-7}s.

3 Set Measurement Precision in Boot Loader

When a computer is first powered on, it usually does not have an operating system in ROM or RAM. The computer must execute a relatively small program stored in ROM, along with the bare minimum of data needed to access the non-volatile devices from which the operating system and data may be loaded into RAM. The small program that starts this sequence is known as a boot loader (i.e. Bootloader). This small program's only job is to locate and initialize hardware, and then find, load and start an operating system. The measurement method proposed in this paper is implemented by the PWM timer which is called timer for short, so the precision of measurement is determined by the frequency of the timer (i.e. TCLK). Since the timer is driven by advanced peripheral bus (APB), we set the precision of measurement by configuring frequencies of hardware related to the timer such as APB and CPU in the assembly code of hardware initialization in Bootloader. For the target microprocessor S3C2440A, to configure the timer frequency TCLK, several steps have to be followed:

Step 1. CPU bus mode configuration

In order to configure the frequency of APB, the CPU bus mode has to be changed from the fast bus mode to the asynchronous bus mode using following instructions:

```
mrc p15,0,r0,c1,c0,0
orr r0,r0,#0xc0000000      ; R1_nF:OR:R1_iA
mcr p15,0,r0,c1,c0,0
```

In the asynchronous bus mode, FCLK of CPU frequency, HCLK of advanced high performance bus (AHB) frequency and PCLK of APB frequency are allowed to be configured respectively.

Step 2. CPU frequency FCLK configuration

There are two clock generator phase-locked-loop (PLL) s in S3C2440A as follows: MPLL that is fed to FCLK, HCLK and PCLK, and UPLL that is fed to USB. According to the equation MPLL= ((M_MDIV<<12) + (M_PDIV<<4) +M_SDIV), write valid settings to the PLL control register MPLLCON as follows:

```
ldr r0,=MPLLCON
ldr r1,=0x0005c011    ; Fin=12MHz, Fout=400MHz
str r1,[r0]
```

FCLK can be configured as PLL output (i.e. MPLL) of 400MHz immediately after lock time.

Step 3. APB frequency PCLK configuration

S3C2440A supports selection of dividing ratio between FCLK, HLCK and PCLK. This ratio is determined by HDIVN and PDIVN of CLKDIVN control register. Chang the value of CLKDIVN register from default (1:1:1) to the divide ratio (1:4:8) as follows:

```
ldr r0,=CLKDIVN
ldr r1,=0x5        ; 0x5 = 1:4:8
str r1,[r0]
```

After lock time, the value setting for CLKDIVN is valid and the dividing ratio between FCLK, HLCK and PCLK is 1:4:8, so the value of PCLK is 50MHz.

Step 4. PWM timer frequency TCLK configuration

Because the timer is driven by APB, divide the APB frequency PCLK to obtain the timer frequency TCLK according to the following equation:

$$TCLK=PCLK/\ \{prescaler\ value\ +1\}/\ \{divider\ value\} \tag{1}$$

where {prescaler value +1}=0~255, {divider value}= 2,4,8,16. TCLK can be configured by setting the parameter of prescaler in timer configuration register$_0$ (TCFG$_0$) and the parameter of divider in timer configuration register$_1$ (TCFG$_1$). By writing valid settings to the TCFG$_0$ and TCFG$_1$ as follows: TCFG$_0$ &= 0xffffff00 and TCFG$_1$&= 0xff0fff0f, the prescaler is set as 0 and the divider is set as 2. Since PCLK has been configured as 50MHz, according to equation (1) TCLK is 25MHz. Then the precision of measurement of our method is 10^{-7}s.

4 Access PWM Timer in Kernel

We can not access to a timer by invoking a system call, because invoking the system call will request a software interrupt that transfers control to the kernel code and its interrupt handle routine may be interrupted by another interrupt request with higher priority. This process will take at lest 1×10^{-6}s. To avoid the timing cost taken by invoking a system call, we access to a timer by its virtual address which is encapsulated in macros. And we define the macros in the initialization code of the kernel. Then, when the kernel starts up, we can access to the timer. In S3C2440A, a timer is consisted of a sequence of registers. In this section, first we transform physical addresses of timer registers into virtual addresses. Then we encapsulate their virtual addresses in the macros to conveniently access to the timer in the kernel.

4.1 Address Transformation of PWM Timer Registers

For our target microprocessor S3C2440A, based on the analysis of mapping between physical addresses of I/O ports and their virtual addresses, the physical address domain 0x48000000 ~ 0x5effffff occupied by I/O ports is mapped onto the virtual address domain 0xe8000000 ~ 0xfeffffff [22-23]. And the trade-off between a physical address and its corresponding virtual address is always 0xa0000000. In S3C2440A, there are five PWM timers, i.e. Timer$_n$ (n=0,1,2,3,4). By the technical reference manual[3], we check the physical address of the first timer register begins at 0x51000000, and the address trade-off between same function registers which belong to neighbored timers (e.g. TCNTB$_1$ and TCNTB$_2$) is always 0xc.

According to these principles, if the address trade-off between a register of $Timer_n$ and its corresponding count buffer register (i.e. $TCNTB_n$) is $\triangle x$, its physical address can be expressed as $0x51000000+(n+1)\times 0xc+\triangle x$, and its corresponding virtual address as $(0x51000000+(n+1)\times 0xc+\triangle x) \mid 0xa0000000$.

4.2 Encapsulating Address Transformation in Macros

In order to conveniently access timer registers, according to the address transformation described in Section 4.1, we encapsulate the address transformation in macros which are defined in the initialization code of the kernel. All operations of access to timer registers in this paper will be performed by calling macros in kernel mode. Taking access to the register $TCNTO_1$ as an example, the macro REG_TCNTO_1 has encapsulated the address transformation between the physical address of $TCNTO_1$ (i.e. $0x51000000 +2\times 0x0c + 0x8$) and its virtual address (i.e. $(0x51000000+2\times 0xc+0x8) \mid 0xa0000000$).

The macros we defined have made the access to timer registers convenient and transparent to programmers and the address transformation readable and portable between different architectures. However, when accessing to registers by calling macros in kernel mode, actually, the operating system needs to transform virtual addresses defined in macros into physical ones according to their mapping relationships to access hardware.

5 Program Execution Cycle Measurement

In S3C2440A, there are five PWM timers. Since $Timer_0$ and $Timer_{2\sim4}$ have been occupied when the kernel initializes, we adopt $Timer_1$ and its corresponding interrupt INT_TIMER_1 to measure execution cycles of program, as illustrated in Fig. 1.

We use the instrument method described in Section 2 to measure the program many times, and obtain a range of measured execution time of the program, i.e. $T_{min} \le T \le T_{max}$ where T represents the timing measurement, T_{min} represents the shortest timing measurement and T_{max} represents the longest one. According to the range of measured execution time, we set the initial value of $Timer_1$ (i.e. the value of the count buffer register $TCNTB_1$) in two cases, as described in Section 5.1. Once start $Timer_1$ and the execution of the program, the down counter $TCNTO_1$ will count down from the initial value of $Timer_1$ by the step of one after every latency time within the timer frequency TCLK of 25MHz. If the initial value of $Timer_1$ is set according to Case One in Section 5.1, read $TCNTO_1$ when the execution of the program ends, and calculate the execution cycles according to equation (2); else the initial value of $Timer_1$ must be set according to Case Two. In Case Two, as illustrated by dash lines in Fig.1, when $TCNTO_1$ reaches zero, an interrupt request of INT_TIMER_1 will be generated and the value of $TCNTB_1$ will be automatically loaded into $TCNTO_1$ to continue the next operation. The interrupt handling routine will record the iteration count of $Timer_1$. This process will repeat until the execution of the program ends. Then read the value of $TCNTO_1$ and the iteration count of $Timer_1$ to calculate the execution cycles of the program according to equation (3) in Section 5.4.

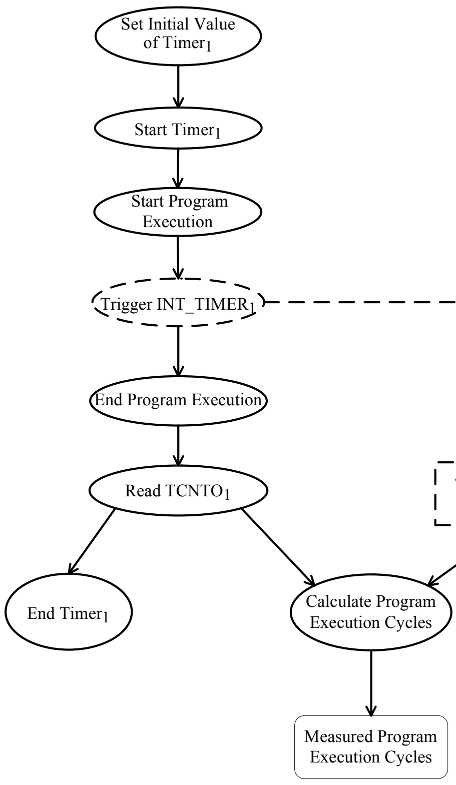


Fig. 1. Principle of measurement for execution cycles of program using $Timer_1$ and its interrupt INT_TIMER_1

5.1 Initial Value Setting For the Timer

As shown in Section 2, the instrument method implemented by the API gettimeofday can not obtain accurate timing measurements for programs due to the timing cost of at lest 1×10^{-6}s. However, the measured execution time by the instrument method will be considered in our method when setting the initial value of $Timer_1$. Since the clock frequency TCLK of $Timer_1$ has been configured as 25MHz in Section 3 and the maximum value of the 16-bit count buffer register $TCNTB_1$ is 65535, the maximum value of the timer cycle T_{timer1}[1]is 2621.4us according to $T_{timer1} = TCNTB_1/TCLK$. Assuming the program has been measured by the API gettimeofday many times and we obtain the measured execution time $T \in [T_{min}, T_{max}]$, classify two cases in our method to set the initial value of $Timer_1$:

[1] The timer cycle T_{timer1} represents the latency time during which the down counter $TCNTO_1$ counts down from the value of $TCNTB_1$ to zero.

Case One. If there is a set of values which could satisfy the condition as follows when assigned to the variable T_{timer1} in the condition:

$$s.t.\begin{cases} T_{max}\text{-}T_{min} \le T_{timer1} \le 2621.4us \\ (T_{max}\text{-}T_{min})/2 \le (T_{max}+T_{min})/2 - Int(T_{min}/T_{timer1})\times T_{timer1} \\ (T_{max}\text{-}T_{min})/2 \le (Int(T_{max}/T_{timer1})+1)\times T_{timer1} - (T_{max}+T_{min})/2 \end{cases}$$

Considering the effect of start and restart delay of the timer on timing measurements, pick out the maximum one from the set to be assigned to T_{timer1}, and then set the value of $TCNTB_1$ as $Int(T_{timer1}\times TCLK)$;

Case Two. Else there is no T_{timer1} which could satisfy the condition of Case One, considering the effect of start and restart delay of the timer on timing measurements, set the value of $TCNTB_1$ as 65535.

In Case One, the condition which T_{timer1} must satisfy has guaranteed the execution of the program ends in the same iteration of $Timer_1$ in any case, as shown in Fig.2.

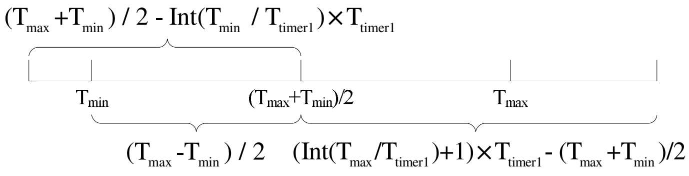

(T_max +T_min) / 2 - Int(T_min / T_timer1)×T_timer1
T_min
(T_max+T_min)/2
T_max
(T_max -T_min) / 2
(Int(T_max/T_timer1)+1)×T_timer1 - (T_max +T_min)/2

Fig. 2. Setting of T_{timer1} in Case One guarantees both T_{min} and T_{max} occur in the last iteration of $Timer_1$

The correlation of T_{min}, T_{max} and T_{timer1} shown in Fig.2 has been transformed into the condition in Case One. If only there is a value of the variable T_{timer1} which could satisfy the condition of Case One, the execution of the program will end in the same iteration (i.e. the last iteration) in any case. So there is no need to use the interrupt handling routine to record the iteration count of $Timer_1$ in Case One. The iteration count can be directly given by $Int(T_{max}/T_{timer1})$ or $Int(T_{min}/T_{timer1})$, both of them are equal. If we assume the delay of the execution of the instruction which reads the down counter $TCNTO_1$ is represented by T_{delay}, the execution cycles of the program can be calculated as follows:

$$ET\_TCLK = Int(T_{max}/T_{timer1})\times TCNTB_1+(TCNTB_1\text{-}TCNTO_1) - T_{delay} \quad (2)$$

where ET_TCLK represents the execution cycles of the program with the unit of clock cycle, the expression of ($TCNTB_1$-$TCNTO_1$) represents the remainder when the execution cycles is divided by the value of $TCNTB_1$.

If Case One could not be satisfied, Case Two must be satisfied. In Case Two, it is necessary to use the interrupt handling routine of INT_TIMER_1 to record the iteration count of $Timer_1$ as described in Section 5.3. In order to limit the effect of start and restart delay of the timer on timing measurements, the value of $TCNTB_1$ should be set as 65535, i.e. the maximum value of $TCNTB_1$.

5.2 Start and Run of the Timer

The count buffer register $TCNTB_1$ stores the initial value of the down counter $TCNTO_1$. And the compare buffer register $TCMPB_1$ stores the initial value of the compare register $TCMP_1$, which will be compared with the value of $TCNTO_1$.Start the timer, according to several steps as follows:

Step 1. Set the values of $TCNTB_1$ and $TCMPB_1$.

Step 2. Set the manual update bit of the control register TCON as 1 to update the values of $TCNTB_1$ and $TCMPB_1$.

Step 3. Set both the start bit and the auto reload bit of TCON as 1 to start the timer and clear its manual update bit.

After starting up $Timer_1$, $TCNTO_1$ starts counting down after every latency time within the timer frequency TCLK. In Case Two in Section 5.1, once $TCNTO_1$ reaches zero, the interrupt request of INT_TIMER_1 will be generated and the value of $TCNTB_1$ will be automatically loaded into $TCNTO_1$ to continue the next iteration of down counting. The process will repeat until the auto reload bit of TCON is cleared and $TCNTO_1$ reaches zero. Then the timer ends.

5.3 Interrupt Trigger and Execution Cycle Count

There are two types of interrupts in S3C2440A: interrupt request (IRQ) and fast interrupt request (FIQ). They are distinguished by the value of a corresponding bit of the interrupt mode register INTMOD [24]. Since FIQ has higher priority and there is only one interrupt can be set as FIQ simultaneously, in our method, we set INT_TIMER_1 as the only FIQ interrupt by configuring INTMOD. The principle of triggering the request of INT_TIMER_1 to record the iteration count of $Timer_1$ in the interrupt handling routine is shown in Fig. 3.

As shown in Fig.3, enable INT_TIMER_1 by clearing the F bit in the current program status register (CPSR); set the initial value of $Timer_1$ by setting the value of $TCNTB_1$; once start $Timer_1$, $TCNTO_1$ starts counting down after every latency time within the timer frequency TCLK; when $TCNTO_1$ reaches zero, an timer interrupt request of INT_TIMER_1 will be generated to inform the CPU to handle the interrupt routine; the iteration count of the timer will be recorded by the interrupt handling routine and the value of $TCNTB_1$ is automatically loaded into $TCNTO_1$ to continue the next iteration.

Since the performances of INT_TIMER_1 and $Timer_1$ are always concurrent in S3C2440A, there is no need to consider the interrupt response time and the timing cost of the interrupt handling routine when calculating the execution cycles of the program.

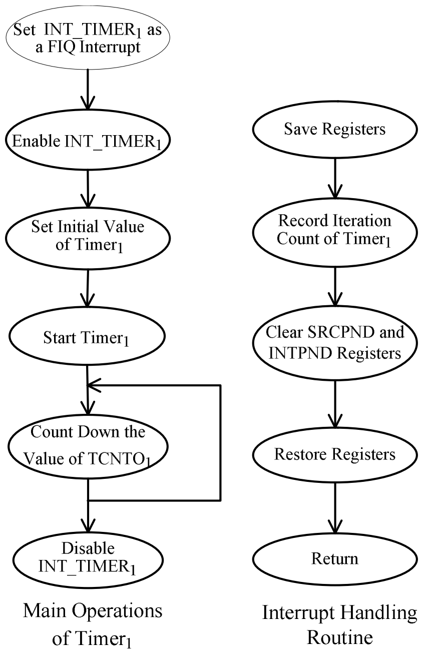

Set INT_TIMER1 as a FIQ Interrupt
Enable INT_TIMER1
Set Initial Value of Timer1
Start Timer1
Count Down the Value of TCNTO1
Disable INT_TIMER1
Main Operations of Timer1
Save Registers
Record Iteration Count of Timer1
Clear SRCPND and INTPND Registers
Restore Registers
Return
Interrupt Handling Routine

Fig. 3. Counting iterations of Timer_1 using the interrupt handling routine

5.4 Program Execution Cycles Calculation

As shown in Fig.1, when the execution of the program ends, read the value of TCNTO_1. The expression of (TCNTB_1-TCNTO_1) represents the execution cycles with the frequency TCLK in the last iteration of Timer_1, i.e. the remainder when the execution cycles of the program is divided by the value of TCNTB_1. Assuming C_{timer1} represents the iteration count of the timer, the delay of the execution of the instruction which reads TCNTO_1 is represented by T_{delay}, the execution cycles of the program can be calculated as follows:

$$\text{ET\_TCLK} = \text{TCNTB}_1 \times C_{\text{timer1}} + (\text{TCNTB}_1 - \text{TCNTO}_1) - T_{\text{delay}} \tag{3}$$

where ET_TCLK represents the execution cycles of the program with the unit of clock cycle.

6 Experiments and Analysis

The microprocessor of our target architecture ARM920T is S3C2440A. ARM920T has a five-stage pipeline, 16KB I-Cache/16KB D-Cache, and memory management unit (MMU). The configuration of Caches and MMU and access to them are performed in the coprocessor CP15. Benchmarks of our experiments come from the worst-case execution time (WCET) suite published by Mälardalen University. The

WCET suite consists of 32 benchmarks, which implement algorithms of signal processing, data compression, quick sort, impulse response filter, etc. It is often used to evaluate and compare different types of WCET analysis tools and methods in the domain of performance evaluation of embedded system.

6.1 Instrument Method and Timer Initial Value Setting

As described in Section 5.1, the execution time $T \in [T_{min}, T_{max}]$ of the benchmark measured by the instrument method was used to set the initial value of $Timer_1$ in two cases. In detail, the trade-off of the system time read by the API gettimeofday at the beginning and end of the execution of the benchmark is its timing measurement by the instrument method. We only measured 22 benchmarks of the WCET suite without floating point calculations, since ARM9 lacks hardware support for these. By the instrument method, each of the 22 benchmarks was measured 20 times. Among all the timing measurements of 20 times, we picked out the longest one T_{max}, the shortest one T_{min} and calculated the average one T_{ave} for each benchmark as shown in Fig.4.

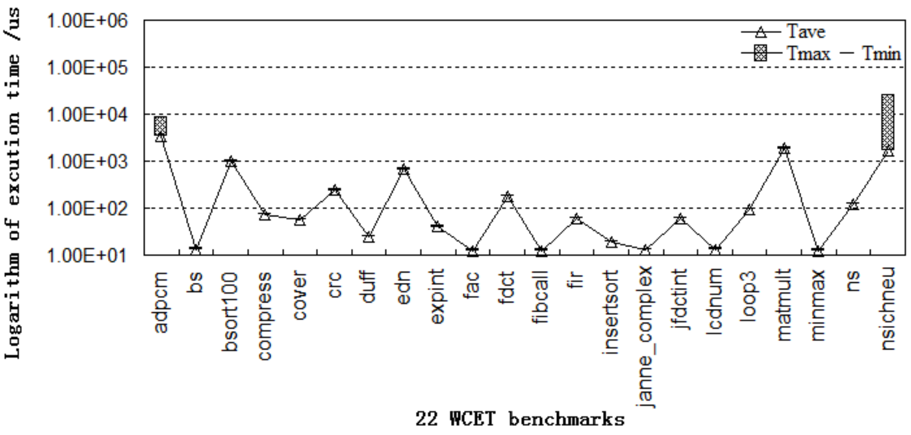


Fig. 4. Timing measurements of 22 WCET benchmarks by the instrument method

As illustrated in Fig.4, T_{min}, T_{max} and T_{ave} of the benchmark are almost equal to each other in most cases of the 22 WCET benchmarks expect for adpcm and nsichneu. For each of the two benchmarks, T_{max} is much longer than T_{min} or T_{ave}. This is because after the API gettimeofday invoked the system call sys_gettimeofday to read the system time at the beginning or end of the execution of the benchmark, a software interrupt request was triggered by sys_gettimeofday in kernel mode and the execution of its interrupt handle routine was interrupted by another interrupt request with higher priority. The execution of the interrupt handle routine was not resumed until the interrupt with higher priority was handled. In this case, T_{max} of each of the two benchmarks was much longer than T_{min} or T_{ave}. For each of 22 benchmarks, since its timing measurements of 20 times by the instrument method have meanings of statistics, T_{min}

and T_{max} were used to set the initial value of the timer and T_{ave} was used to compare with the timing measurement by our method in Section 6.4.

Considering the timing measurements of T_{min} and T_{max} of each benchmark in Fig.4, set the initial value of $Timer_1$ in two cases as follows. Since adpcm and nsichneu satisfied Case Two in Section 5.1, the value of $TCNTB_1$ was set as 65535 when measuring their execution time by our method. For each of other benchmarks, there was always a value to be assigned to T_{timer1} which could satisfy the condition of Case One, so the value of $TCNTB_1$ was set as $Int(T_{timer1} \times TCLK)$ according to Case One.

6.2 Enable Cache and MMU

In Section 6.1, we measured the execution time of the 22 benchmarks by the instrument method under the condition that Cache and MMU were enabled. In order to compare the measurement result of the instrument method with that of our method under the same condition, we enabled Cache and MMU before starting the timer. At first, to enable I-Cache and D-Cache, we set the Icr and Ccr bits of the control register c1 of the coprocessor CP15 as 1. Then we set the M bit of the register c1 as 1 to enable MMU. If we need to disable Cache, set the Icr and Ccr bits as 0.

6.3 PWM Timer Interrupt Setting and Iteration Count

For benchmarks of adpcm and nsichneu, they satisfied Case Two in Section 5.1, so their measurements for execution time were performed by combining $Timer_1$ and its interrupt INT_TIMER_1 according to Section 5.3.

Before their measurements, the value of $TCNTB_1$ and the type of $INT\_ TIMER_1$ were set according to Section 5.1 and Section 5.3. After $Timer_1$ started, its down counter $TCNTO_1$ started counting down with the frequency TCLK. When the value of $TCNTO_1$ reached zero, the timer interrupt request of $INT\_ TIMER_1$ was generated and the program counter (PC) jumped to the interrupt vector address 0x1c. By executing the jump instruction stored in 0x1c, the interrupt handling routine was found and executed. It maintained a global variable C_{timer1} that recorded the timer iteration count, as shown in Fig.3. Part of the interrupt handling routine was shown as follows:

```
__asm__ __volatile__(
" stmfd sp!, {r0-r4, lr}\n " // Save registers
" add %0,%0,#0x0001\n "      // Increase the timer itera-
                tion count by one each time
" mov r0, #0xffffffff\n "
" ldr r1, =rSRCPND\n "
" ldr r2, =rINTPND\n "
" str r0, [r1] \n "           //Clear register SRCPND
" str r0, [r2] \n "           //Clear register INTPND
" ldmfd sp!, {r0-r4, lr}\n " //Restore registers
" subs pc, lr, #4\n "        //Return
: "=r"(Ctimer1)
: "0"(Ctimer1) );
```

6.4 Execution Time Calculation and Method Effectiveness Analysis

When the execution of the benchmark ended, read the value of $TCNTO_1$ and the iteration count represented by C_{timer1} to calculate the execution cycles of the benchmark. In our experiment, we measured the delay of the execution of the instruction which reads $TCNTO_1$ many times and found that it was always equal to 64.5 clock cycles. According to two cases described in Section 5, the execution cycles ET_TCLK can be calculated by equation (2) or (3). The execution time of the benchmark with the unit of second was calculated by

$$ET\_s = ET\_TCLK / TCLK \tag{4}$$

The precision of the result is 10^{-7}s, which is much more precise than 10^{-4}s that RTC could provide.

According to equation (4), we got the timing measurements of 22 WCET benchmarks with the unit of second. In Fig.5, we compared the execution time measured by our method with the measurement results by the instrument method obtained in Section 6.1.

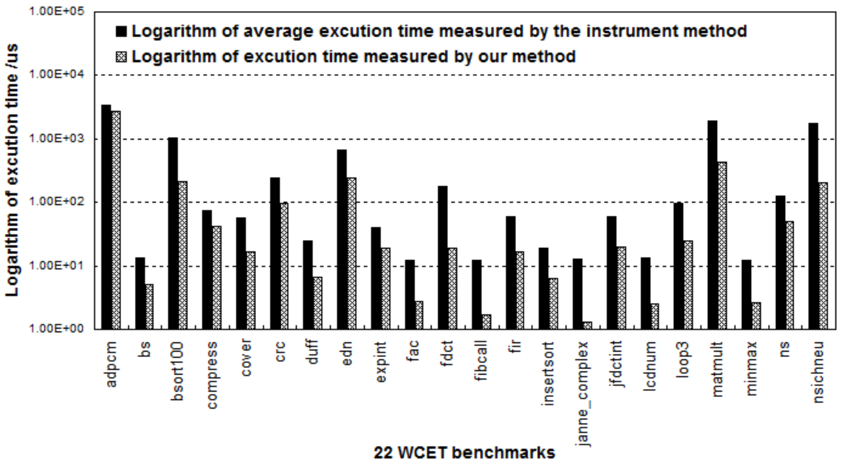
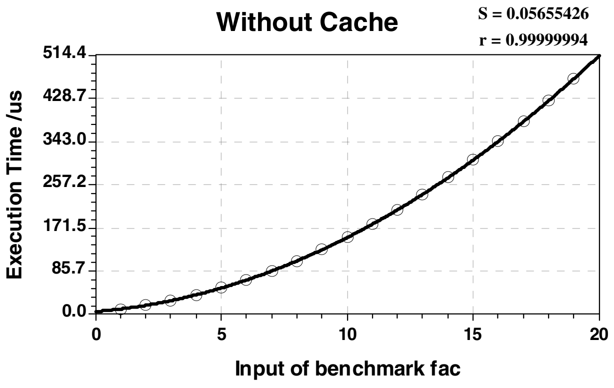


Fig. 5. Comparison of timing measurements by our method and those by the instrument method for 22 WCET benchmarks

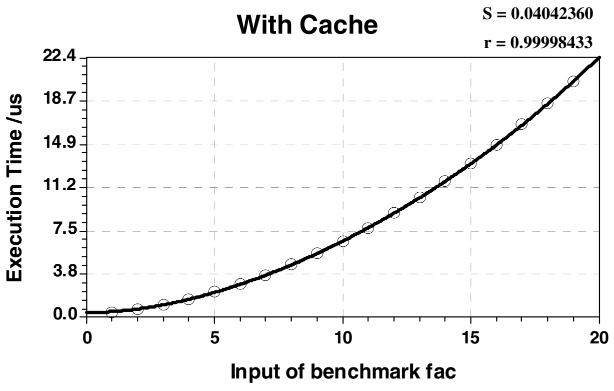
In Fig.5 it is shown that the execution time measured by our method is much shorter than the average execution time measured by the instrument method. The timing measurement by our method was only 28.46% of that by the instrument method on average. This is because after the API gettimeofday invoked the system call to read the system time, the kernel had to spend at least 1×10^{-6}s to handle the software interrupt which was triggered by the system call. Besides, the execution of the software interrupt handle routine might be interrupted by another interrupt request with higher priority. And the execution of the instrument instructions will delay the execution as well. The timing cost of the instrument method is so expensive that its timing measurements were much longer than those by our method. The experimental result has shown that our measurement method is an effective way to obtain accurate execution time of program.

6.5 Program Execution Time Prediction

In this section, firstly, we used our method to measure the execution time of three benchmarks (i.e. fac, fir and insertsort) which were randomly picked out of 22 WCET benchmarks. Each of the three benchmarks was measured 20 times under 20 inputs of it, and we got 20 pairs of (input data, timing measurement) for each benchmark. Secondly, we used curve fitting tool to fit these pairs of data and obtained the distribution of timing measurement for each benchmark. Finally, we used the distribution of timing measurement to predict the execution time of each benchmark under different input data and compare the predicted results with the measurement ones by our method.



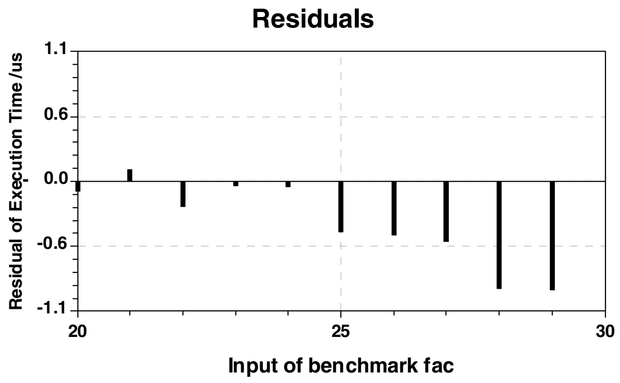
(a) Cache disabled



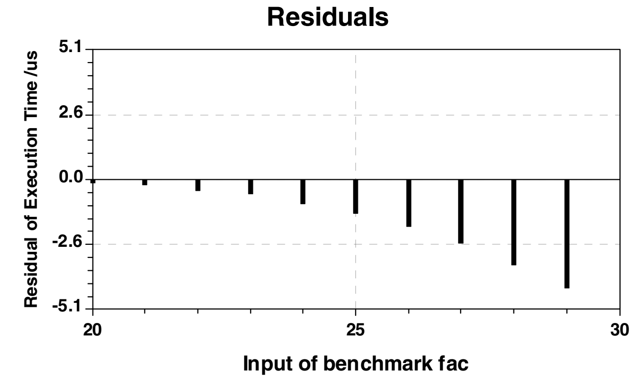
(b) Cache enabled

Fig. 6. Distribution of timing measurement of the benchmark fac under 20 inputs

Taking the benchmark fac as an example, in the case of Cache disabled, we firstly used our method to measure the execution time of fac under 20 inputs, and obtained 20 pairs of data as follows:(0, 5.45), (1, 10.63), (2, 17.94), … , (19, 468.66). Secondly, we fitted the 20 pairs of data by the curve fitting tool to get the timing measurement distribution which can be expressed as a function y=a+bx+cx^2+dx^3 where a =5.46072, b =4.10911, c =1.06182, d =0.00026 as shown in Fig. 6(a). The standard deviation of the fitting is 0.05655426 and the correlation coefficient is 0.99999994. Finally, we used the fitted distribution to predict the execution time of fac under its different 10 inputs (i.e. 20~29). Comparing the predicted results with the measurement ones by our method in Fig. 7(a), we got an average deviation of 0.04%.



(a) Cache disabled



(b) Cache enabled

Fig. 7. Comparison of the predicted execution time and the measured one of fac under 10 inputs

In the case of Cache enabled, we did the experiment following the same steps. Firstly, we measured the execution time of fac under the same 20 inputs and got pairs of (input data, timing measurement). Secondly, we fitted these pairs of data to obtain the distribution of the timing measurement (i.e. y=a+bx+cx^2+dx^3+ex^4 where a

=0.44485, b =0.01870, c =0.07512, d =0.00020, e =0.00005) as shown in Fig. 6(b). Finally, we used the distribution to predict the execution time of the benchmark under the same different 10 inputs. Comparing the predicted results with the measurement ones by our method, we got an average deviation of 4.13%.

The experiments were also made in the same way for the other randomly picked benchmarks of fir and insertsort. In the case of Cache disabled, we got average deviations of 3.69% for fir and 1.77% for insertsort; in the case of Cache enabled, we got average deviations of 3.78% for fir and 4.69% for insertsort. The average deviation we got under enabled Cache was always larger than the one we got under disabled Cache. This is because in the case of Cache enabled, when the execution of the benchmark needs to access memory for instructions or data, it will be access I-Cache or D-Cache at first in which case either Cache hit or Cache miss occurs, then the timing cost due to accessing memory is undetermined and thus the execution time of the benchmark is not determined as well even under the same input.

These experiments have shown that our measurement method can be used to predict the program execution time especially in the case of Cache disabled. It provides a convenience way to predict program execution time.

7 Conclusions

Since the precision of measurement provided by the hardware counter RTC is not high enough and the timing cost of the instrument method is difficult to be calculated, we propose a combined hardware/software measurement for ARM program execution time, which could give the high measurement precision of 10^{-7}s. It first sets the precision of measurement by configuring frequencies of relevant hardware in the system boot loader code, next, encapsulates the access to the PWM timer in the macros defined in the kernel, and finally uses the PWM timer and its corresponding interrupt to measure the execution time of program. Experimental results show that it is accurate to measure the execution time of ARM program with high precision, comparing with current end-to-end dynamic measurements. Besides, combining with curve fitting techniques, the method can be used to predict the execution time of program under different input data. The measurement method proposed in this paper provides an effective and practical way for the execution time estimation of ARM programs.

References

1. Wilhelm, R., Engblom, J., Ermedahl, A., Holsti, N., Thesing, S., Whalley, D., et al.: The Worst-Case Execution-Time Problem-Overview of Methods and Survey of Tools. ACM Transactions on Embedded Computing Systems 7(3), 1–49 (2008)
2. Puschner, P., Burns, A.: A Review of Worst-Case Execution-Time Analysis. Real-Time Systems 18(2-3), 115–128 (1999)
3. Samsung Electronics: S3C2440A 32-Bit CMOS Microcontroller User's Manual, Revision 1. Technical report, Samsung Electronics (2004)
4. Engblom, J., Ermedahl, A., Sjodin, M., Gustafsson, J., et al.: Worst-Case Execution-Time Analysis for Embedded Real-Time Systems. Software Tools for Technology Transfer 4(4), 437–455 (2003)

5. Ermedahl, A., Stappert, F., Engblom, J.: Clustered Worst-Case Execution-Time Calculation. IEEE Trans. on Computers 54(9), 1104–1122 (2005)
6. Lim, S.-S., Bae, Y.H., Jang, G.T., et al.: An Accurate Worst Case Timing Analysis Technique for RISC Processors. IEEE Trans. on Software Engineering 21(7), 593–604 (1995)
7. Malik, S., Martonosi, M., Li, Y.-T.S.: Static Timing Analysis of Embedded Software. In: 34th Annual Conference on Design Automation, pp. 147–152. ACM Press, New York (1997)
8. Wegener, J., Mueller, F.: A Comparison of Static Analysis and Evolutionary Testing for the Verification of Timing Constraints. Real-Time Systems 21(3), 241–268 (2001)
9. Petters, S.M., Zadarnowski, P., Heiser, G.: Measurments or Static Analysis or Both? In: 7th International Workshop on Worst-Case Execution Time Analysis (2007)
10. Živojnovic, V.: Compiled Hw/Sw Co-Simulation. In: 33rd Annual Conference on Design Automation, pp. 690–695. ACM Press, New York (1996)
11. Burger, D., Austin, T.M.: The SimpleScalar Tool Set, Version 4.0. Technical report #1342, Computer Sciences Department of Wisconsin University (2001)
12. Andrews, J.: Co-Verification of Hardware and Software for ARM SoC Design. Translated by Zhou L.G., et al. BUAA Press, Beijing (2006) (in Chinese)
13. Lazarescu, M.T., Bammi, J.R., Harcourt, E., et al.: Compilation-Based Software Performance Estimation for System Level Design. In: IEEE International Workshop on High-Level Validation and Test, pp. 167–172. IEEE Press, New York (2000)
14. Wenzel, I., Kirner, R., Rieder, B., Puschner, P.: Measurement-Based Worst-Case Execution Time Analysis. In: 3rd IEEE Workshop on Software Technologies for Future Embedded and Ubiquitous Systems, pp. 7–10. IEEE Press, New York (2005)
15. Colmenares, J.A., Im, C., Kim, K.H., et al.: Measurement Techniques in a Hybrid Approach for Deriving Tight Execution-Time Bounds of Program Segments in Fully-Featured Processors. In: Real-Time and Embedded Technology and Applications Symposium, pp. 68–79. IEEE Press, New York (2008)
16. Zolda, M., Bunte, S., Kirner, R.: Towards Adaptable Control Flow Segmentation for Measurement-Based Execution Time Analysis. In: 17th International Conference on Real-Time and Network Systems, pp. 77–85 (2009)
17. Jean-Francois, D., Isabelle, P.: Safe Measuremnt-Based WCET Estimation. In: 5th International Workshop on Worst-Case Execution Time Analysis (2005)
18. Kong, L.L., Jiang, J.H.: A Safe Measurement-Based Worst-Case Execution Time Estimation Using Automatic Test-Data Generation. In: 16th IEEE Pacific Rim International Symposium on Dependable Computing, pp. 245–246. IEEE Press, New York (2010)
19. Kirner, R., Puschner, P., Wenzel, I.: Measurement-Based Worst-Case Execution Time Analysis Using Automatic Test-Data Generation. In: 3rd IEEE Workshop on Software Technologies for Future Embedded and Ubiquitous Systems, pp. 23–26. IEEE Press, New York (2005)
20. Wenzel, I., Rieder, B., Kirner, R., Puschner, P.: Automatic Timing Model Generation by CFG Partitioning and Model Checking. In: Conference on Design, Automation and Test in Europe, pp. 606–611. IEEE Press, New York (2005)
21. Wang, X., Ji, M.L., Wang, J., et al.: Trace Acquirement Technology of Real-Time Systems Based on WCET Analysis. Journal of Software 17(5), 1232–1240 (2006) (in Chinese)
22. Zhao, J.: Analysis of Linux Kernel, 1st edn. China Machine Press, Beijing (1998) (in Chinese)
23. Robert, L.: Linux Kernel Development. Translated by Chen, L.J., et al. China Machine Press, Beijing (2006) (in Chinese)
24. Yin, X.F., Yuan, S.H., Hu, J.B.: Experimental Research on Interrupt Latency Time of ARM Microprocessor. Computer Engineering 37(4), 252–263 (2011) (in Chinese)

A Low-Complexity Parallel Two-Sided Jacobi Complex SVD Algorithm and Architecture for MIMO Beamforming Systems[*]

Weihua Ding, Jiangpeng Li, Guanghui He, and Jun Ma

School of Microelectronics, Shanghai Jiao Tong University, Shanghai, China
{forever_06141,lijiangpeng,guanghui.he,majun}@sjtu.edu.cn

Abstract. Singular Value Decomposition (SVD) is a very important matrix factorization technique in engineering applications. In multiple-input multiple-output (MIMO) systems, SVD is applied in transmit beamforming which provides high diversity advantages. This paper proposes a low-complexity parallel two-sided Jacobi complex SVD algorithm and architecture which are suitable for any $\mathrm{m} \times n$ $(m \le 4, n \le 4)$ matrix. It performs two 2×2 complex SVD procedures in parallel, and employs master-slave CORDIC (coordinate rotation digital computer) to reduce the decomposition time. The proposed parallel algorithm for 4×4 complex SVD saves 52% decomposition time compared with the Golub-Kahan-Reinsch algorithm. Meanwhile, the Bit Error Rate (BER) performance of the proposed algorithm is almost the same with the ideal SVD.

Keywords: MIMO, Beamforming, Two-sided Jacobi, SVD, Master-slave CORDIC.

1 Introduction

Multiple-input multiple-output (MIMO) communication systems are employed in many wireless communication standards (e.g., IEEE 802.11n, IEEE 802.11ac) to increase data rates through spatial multiplexing or to improve reliability through diversity [1]. Beamforming is a technique that provides high diversity with the help of channel state information at the transmitter (CSIT). It corresponds to the transmit precoding and receiver shaping based on singular value decomposition (SVD) of the channel matrix [1]. To achieve high throughput in MIMO beamforming systems, precoding matrix derived from SVD should be sent from the receiver to the transmitter as soon as possible. Therefore, the decomposition time and the accuracy of SVD will significantly affect the beamforming performance.

Many research works have focused on SVD algorithm and architecture for MIMO beamforming applications. A time-shared SVD architecture was proposed in [2] using Golub-Kahan-Reinsch algorithm, which leads to low hardware cost but long

[*] This work was supported in part by Shanghai Natural Science Foundation under Grant No. 10ZR1416500 and the Research Fund for the Doctoral Program of Higher Education of China under Grant No. 20110073110055.

decomposition time. An adaptive algorithm is proposed in [3] while it requires large iterations to achieve an acceptable arithmetic precision. Jacobi-like algorithms are extremely suitable for parallel SVD to reduce the decomposition time. Two-sided Jacobi algorithm was applied in [4] to implement a 2x2 complex SVD with Givens rotation. It requires 7 serial pipeline coordinate rotation digital computer (CORDIC) stages. In [5], two-sided Jacobi algorithm for complex-valued matrix was implemented on systolic array. But the overall decomposition time is still much longer than real SVD array. To improve the decomposition time of SVD, we propose a low-complexity parallel two-sided Jacobi complex SVD algorithm for any $m \times n$ $(m \le 4, n \le 4)$ matrix. In this algorithm, if either row or column of the matrix is less than 4, it firstly expands the matrix to dimension 4×4 and then decomposes the 4×4 matrix by performing two 2×2 complex SVD in parallel. Finally, the derived SVD matrices $U_{m\times m}$, $\Sigma_{m\times n}$ and $V_{n\times n}$ correspond to m rows and n columns of the decomposed 4×4 matrices.

The rest of this paper is organized as follows. In section 2, the SVD-based beamforming system is outlined. The proposed parallel two-sided Jacobi complex SVD algorithm is described in section 3. VLSI architecture of this algorithm is introduced in section 4. Simulation results and the decomposition time comparison are presented in section 5 and 6, respectively. Finally, the conclusion is given in section 7.

2 SVD-Based Beamforming System

Consider a MIMO channel with a $N_r \times N_t$ channel matrix H which is known at both the transmitter and the receiver. For any matrix H its SVD can be formulated as

$$H = U\Sigma V^H \tag{1}$$

where the $N_r \times N_r$ matrix U and the $N_t \times N_t$ matrix V are unitary matrices and the singular matrix Σ is an $N_r \times N_t$ diagonal matrix containing singular values $\{\sigma_i\}$ of H. There are R_H nonzero singular values where R_H denotes rank of matrix H.

SVD-based beamforming is implemented by performing a transformation on the channel input x and output y via transmit precoding and receiver shaping, which is shown in Fig. 1.

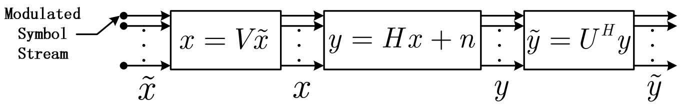


Fig. 1. Transmit precoding and receiver shaping

From the definition of SVD, it can formulate that:

$$\begin{aligned}\tilde{y} &= U^H(Hx+n) = U^H(U\Sigma V^H V\tilde{x}+n)\\ &= U^H U\Sigma V^H V\tilde{x} + U^H n = \Sigma\tilde{x}+\tilde{n}\end{aligned} \tag{2}$$

As multiplication by a unitary matrix does not affect the distribution of the noise, n and $\tilde{n}$ are identically distributed. SVD-based beamforming strategy separates the MIMO channel into R_H parallel independent subchannels that do not interfere with each other. It significantly improves the diversity gain of MIMO systems [1].

3 Proposed Parallel Two-sided Jacobi Algorithm

In this section, we introduce the proposed low-complexity parallel two-sided Jacobi complex SVD algorithm. It is suitable for any $m \times n\ (m \le 4, n \le 4)$ matrix. In this algorithm, if either row or column of the matrix is less than 4, it expands the original matrix to dimension 4×4 by inserting extra zero rows and zero columns. Then the 4×4 matrix is decomposed by iteratively performing two 2×2 complex SVD procedures in parallel. Finally, the derived SVD matrices $U_{m\times m}$, $\Sigma_{m\times n}$ and $V_{n\times n}$ correspond to m rows and n columns of the decomposed 4×4 matrices. Firstly, we introduce the 2×2 two-sided Jacobi complex SVD algorithm.

$$\mathbf{H} = \begin{bmatrix} \mathbf{C} & \mathbf{C} \\ \mathbf{C} & \mathbf{C} \end{bmatrix} \xrightarrow{\mathbf{LHS}} \begin{bmatrix} \mathbf{R} & \mathbf{C} \\ \mathbf{R} & \mathbf{C} \end{bmatrix} \xrightarrow{\mathbf{LHS}} \begin{bmatrix} \mathbf{R} & \mathbf{C} \\ \mathbf{0} & \mathbf{C} \end{bmatrix} \xrightarrow{\mathbf{RHS}} \begin{bmatrix} \mathbf{R} & \mathbf{R} \\ \mathbf{0} & \mathbf{C} \end{bmatrix} \xrightarrow{\mathbf{LHS}} \begin{bmatrix} \mathbf{R} & \mathbf{R} \\ \mathbf{0} & \mathbf{R} \end{bmatrix} \xrightarrow{\mathbf{TPR}} \begin{bmatrix} \mathbf{R} & \mathbf{0} \\ \mathbf{0} & \mathbf{R} \end{bmatrix} = \mathbf{\Sigma}$$

Fig. 2. Diagonalization of 2×2 complex-valued matrix

Consider a 2×2 complex-valued matrix H, which is shown in Fig. 2. Givens rotations are successively applied to H from the left-hand side (LHS) and from the right hand side (RHS), such that H is transformed to singular matrix Σ. Corresponding updates are also applied to unitary matrices U and V. The affected entries in each step are highlighted.

In the last step, TPR method [6] is employed to diagonalize the 2×2 real-valued matrix. It takes less rotations than two-sided Jacobi rotations.

Any 2×2 real-valued matrix can be reformulated as

$$A = A_1 + A_2 = \begin{bmatrix} p_1 & -q_1 \\ q_1 & p_1 \end{bmatrix} + \begin{bmatrix} -p_2 & q_2 \\ q_2 & p_2 \end{bmatrix} \tag{3}$$

This leads to the following reformulation of the last diagonalization step in Fig. 2.

$$\begin{aligned} \Sigma &= R(\theta_l)^T A R(\theta_r) = R(\theta_r - \theta_l) A_1 + R(\theta_r + \theta_l)^T A_2 \\ &= R(\theta_\Delta) A_1 + R(\theta_\Sigma)^T A_2 = \begin{bmatrix} r_1 & 0 \\ 0 & r_1 \end{bmatrix} + \begin{bmatrix} -r_2 & 0 \\ 0 & r_2 \end{bmatrix} \end{aligned} \tag{4}$$

where $\tan(\theta_\Delta) = q_1/p_1$, $\tan(\theta_\Sigma) = q_2/p_2$ and $R(\theta)$ denotes Givens rotation with angle θ. This means two-sided Jacobi rotations can be performed by two plane rotations [6] which only compute r_1 and r_2.

Then the 2×2 SVD procedure above is iteratively performed to diagonalize 4×4 complex-valued matrix. As each Givens rotation affects only two rows and two columns, we may actually perform two 2×2 two-sided Jacobi procedures in parallel. Then the 4×4 matrix could be divided into four 2×2 submatrices. As shown in Fig. 3, four entries are annihilated in each parallel 2×2 SVD procedures, which is called one iteration. Since there are 12 off-diagonal entries in a 4×4 matrix, 3 iterations are required to update all the off-diagonal entries, which is called one sweep. Multiple sweeps repeat for the convergence of all the off-diagonal entries. In Fig. 3, entry pairs under the arrows denote entries of the two parallel 2×2 submatrices (highlighted) in each iteration.

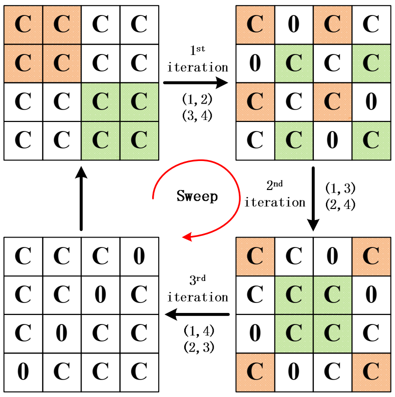


Fig. 3. Sweep process of 4×4 complex SVD

The proposed parallel two-sided Jacobi complex SVD algorithm for $m\times n\ (m\le 4, n\le 4)$ matrix is summarized in Table 1 where S denotes the number of sweeps, $I_{4\times 4}$ denotes 4×4 identity matrix. The algorithm performs two 2×2 SVD procedures and the corresponding updates on U and V in parallel.

Table 1. Proposed parallel two-sided Jacobi complex SVD algorithm

```
1. Initialization: $\begin{bmatrix} H_{m\times n} & 0 \\ 0 & 0 \end{bmatrix} \to H_{4\times 4}$, $U^H = I_{4\times 4}$, $V = I_{4\times 4}$
2. for s=1 to S do
3.   $i_1=\{1,1,1\}, j_1=\{2,3,4\};\ i_2=\{3,2,2\}, j_2=\{4,4,3\}$
4.   for k=1 to 3 do
5.     Divide $H_{4\times 4}$ into four 2×2 submatrices:
       $H_1=\begin{bmatrix} H(i_1(k),i_1(k)) & H(i_1(k),j_1(k)) \\ H(j_1(k),i_1(k)) & H(j_1(k),j_1(k)) \end{bmatrix}, H_3=\begin{bmatrix} H(i_1(k),i_2(k)) & H(i_1(k),j_2(k)) \\ H(j_1(k),i_2(k)) & H(j_1(k),j_2(k)) \end{bmatrix}$
       $H_2=\begin{bmatrix} H(i_2(k),i_2(k)) & H(i_2(k),j_2(k)) \\ H(j_2(k),i_2(k)) & H(j_2(k),j_2(k)) \end{bmatrix}, H_4=\begin{bmatrix} H(i_2(k),i_1(k)) & H(i_2(k),j_1(k)) \\ H(j_2(k),i_1(k)) & H(j_2(k),j_1(k)) \end{bmatrix}$
```

Table 1. (*continued*)

```
7.     Update $H_3$ and $H_4$: $U_1^H H_3 V_2 \rightarrow H_3$, $U_2^H H_4 V_1 \rightarrow H_4$
8.     Update the corresponding rows of $U^H$:
       $U_1^H \begin{bmatrix} U^H(i_1(k),:) \\ U^H(j_1(k),:) \end{bmatrix} \rightarrow \begin{bmatrix} U^H(i_1(k),:) \\ U^H(j_1(k),:) \end{bmatrix}$, $U_2^H \begin{bmatrix} U^H(i_2(k),:) \\ U^H(j_2(k),:) \end{bmatrix} \rightarrow \begin{bmatrix} U^H(i_2(k),:) \\ U^H(j_2(k),:) \end{bmatrix}$
9.     Update the corresponding columns of $V$:
       $[V(:,i_1(k))\ \ V(:,j_1(k))]V_1 \rightarrow [V(:,i_1(k))\ \ V(:,j_1(k))]$
       $[V(:,i_2(k))\ \ V(:,j_2(k))]V_2 \rightarrow [V(:,i_2(k))\ \ V(:,j_2(k))]$
10.   end iteration
11. end sweep
12. Order singular values and singular vectors
13. Derived SVD matrices: $\Sigma(1:m,1:n) \rightarrow \Sigma_{m\times n}$,
                          $U(1:m,1:m) \rightarrow U_{m\times m}$, $V(1:n,1:n) \rightarrow V_{n\times n}$
```

4 Parallel VLSI Architecture

Fig. 4 provides an overview of the parallel matrix decomposition architecture which supports MIMO beamforming systems with all antenna modes ($N_r \leq 4, N_t \leq 4$). The architecture consists of a matrix memory, two parallel 2×2 SVD cores and a finite state machine (FSM) which controls the memory and the SVD cores.

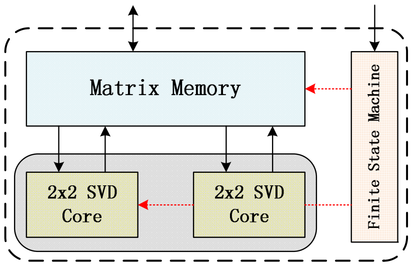


Fig. 4. Overview of the parallel matrix decomposition architecture

A. *Matrix Memory*

The matrix memory consists of one two-port SRAM macro cell that stores U , Σ and V . The singular matrix Σ is initialized by the channel matrix *H*. Both U and V are initialized by 4×4 identity matrices. All the decomposition results after each iteration are written back to the memory for the next iteration.

B. *Parallel* 2×2 *SVD cores*

Parallel 2×2 SVD cores perform the algorithm described in Table 1. As the decomposition mainly consists of two-dimensional Givens rotations, master-slave CORDIC [7] is employed in performing the rotations, which reduces decomposition time.

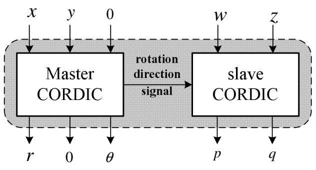


Fig. 5. The master-slave CORDIC

Consider the Givens rotation applied on a 2×2 real-valued matrix in (5). As shown in Fig. 5, the master CORDIC operates in vectoring mode to rotate the *1*st column vector $v_1=[x,y]^T$. The *2*nd column vector $v_2=[w,z]^T$ is fed into slave CORDIC which performs the same micro-rotations by the control of rotation direction signal.

$$\begin{bmatrix}\cos\theta & \sin\theta\\ -\sin\theta & \cos\theta\end{bmatrix}\begin{bmatrix}x & w\\ y & z\end{bmatrix}=\begin{bmatrix}r & p\\ 0 & q\end{bmatrix} \tag{5}$$

But for normal CORDIC, it first rotate v_1 to derive θ and then rotate v_2 with the derived angle. So the transformation of (5) with master-slave CORDIC requires half rotation time than that of normal CORDICs.

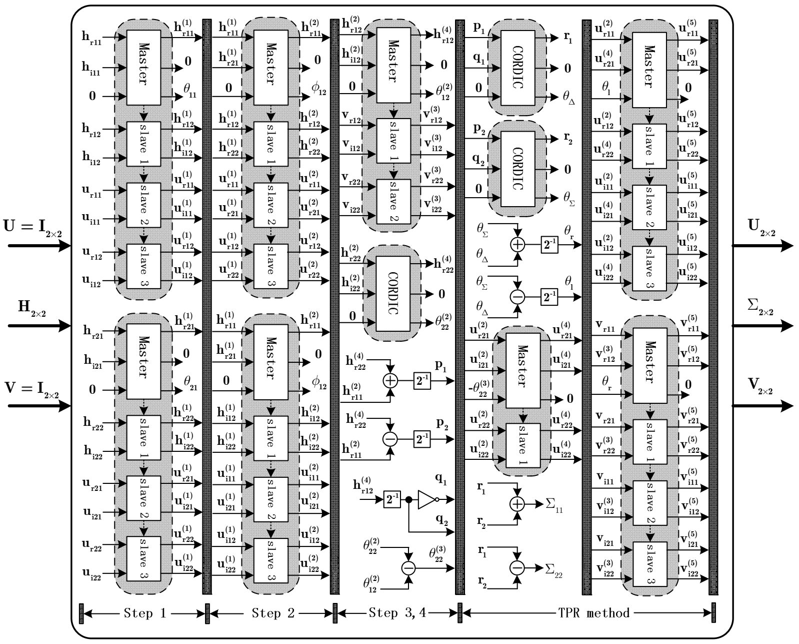


Fig. 6. VLSI architecture of 2×2 two-sided Jacobi complex SVD

VLSI architecture of 2×2 two-sided Jacobi complex SVD is illustrated in Fig. 6. To simultaneously rotate the corresponding entries in *U* and *V*, more slave CORDICs are specified with one master CORDIC. In the proposed architecture, each master-slave CORDIC contains 3 slave CORDICs at most. The serial pipeline CORDIC stages correspond to steps shown in Fig. 2.

5 Simulation Results

In this section we evaluate the beamforming performance with Bit Error Rate (BER) in a practical MIMO-OFDM system. Unitary matrix *V* is fed back to the transmitter for precoding and unitary matrix *U* is used for receiver shaping. Configuration of the specified MIMO-OFDM system is listed as follows.

- Beamforming technique : SVD
- Channel type : Ch. E [8], 802.11n, 4 spatial streams, 4×4 channel matrix
- Perfect channel state information is assumed
- Channel coding : code rate 2/3, convolutional code with constraint length 7, generator polynomial of [133 171]
- Modulation : 64-QAM
- MIMO-OFDM system with 256 tones
- Linear MMSE detection

As shown in Fig. 7, the proposed algorithm with 3 sweeps and 8 CORDIC micro-rotations degrades 1dB performance at $BER = 10^{-4}$ than the ideal SVD with Golub-Kahan-Reinsch algorithm [2]. As number of CORDIC micro-rotations increases, BER performance improves. With 3 sweeps and 12 CORIDC micro-rotations, BER performance of the proposed algorithm is almost the same with the ideal SVD.

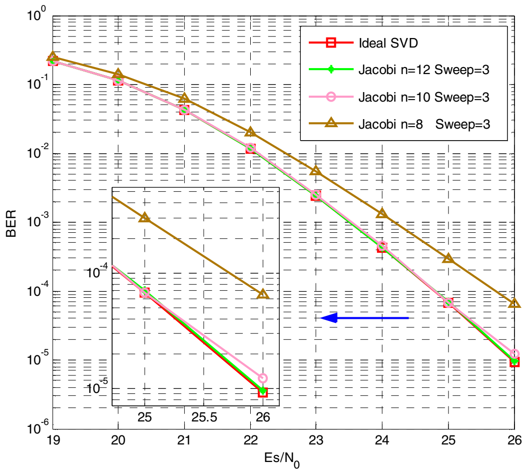


Fig. 7. BER performance of the proposed algorithm and the ideal SVD

6 Decomposition Time Comparison

The decomposition time is compared between the proposed parallel two-sided Jacobi algorithm and the method in [2] and [4]. We compare the decomposition time with the required number of clock cycles.

Firstly, we analyze the decomposition time of 2×2 complex SVD. To decompose a 2×2 complex-valued matrix, the proposed algorithm requires only one iteration. As shown in Fig. 6, it costs 5 serial pipeline CORDIC stages. However, the algorithm in [8] costs 7 serial CORDIC pipeline stages, which takes longer decomposition time. Assume that it takes 4 clock cycles to compensate the expansion factor K [9] with shift-add operations, and takes 5 clock cycles to order the two singular values, the proposed algorithm for 2×2 complex SVD requires $T_{2\times 2} \approx 5\times(n+4)+5$ cycles to complete one decomposition where n denotes number of CORDIC micro-rotations.

Decomposition time comparison of the 2×2 complex SVD is presented in Table 2 with different CORDIC micro-rotations.

Table 2. Decomposition time comparison of 2×2 complex SVD

2×2 complex SVD (clock cycles)			
CORDIC micro-rotations	**[4]**	**This work**	Improvement
16	173	105	39%
12	138	85	38%

To compare the decomposition time of 4×4 complex SVD, assume that it takes *20* clock cycles to order the four singular values and the corresponding singular vectors, the proposed parallel two-sided Jacobi algorithm for 4×4 complex SVD requires $T_{4\times 4} = S\times 3\times 5\times(n+4)+20$ cycles to complete one decomposition where S denotes the number of sweeps.

Decomposition time comparison of the 4×4 complex SVD is presented in Table 3. The 1539 clock cycles is computed with the maximum SVD time and the corresponding clock frequency in [2]. According to the simulation results in section 5, the number of sweeps is chosen to be 3. Because the BER performance of the proposed algorithm with 3 sweeps and 12 CORDIC micro-rotations is almost the same with the ideal SVD. As shown in Table 3., the proposed algorithm for 4×4 complex SVD saves 52% decomposition time compared with the Golub-Kahan-Reinsch algorithm [2].

Table 3. Decomposition time comparison of 4×4 complex SVD

4×4 complex SVD (clock cycles)			
CORDIC micro-rotations	**[2]**	**This work**	Improvement
12	1539	740	52%

7 Conclusions

In this paper, a low-complexity parallel two-sided Jacobi complex SVD algorithm and the parallel VLSI architecture are proposed for MIMO beamforming systems. The algorithm is suitable for any $m \times n (m \le 4, n \le 4)$ matrix by expanding it to dimension 4×4 . Then the 4×4 matrix is decomposed by performing two 2×2 complex SVD procedures in parallel. To reduce the decomposition time, master-slave CORDICs are employed in the parallel architecture. The proposed algorithm has low computational complexity which saves 52% decomposition time compared with the Golub-Kahan-Reinsch algorithm on 4×4 complex SVD. Furthermore, BER performance of the proposed algorithm is almost the same with the ideal SVD.

References

1. Goldsmith, A.: Wireless Communications, 1st edn. The People's Posts and Telecommunications Press, Beijing (2007)
2. Studer, C., Blosch, P., Friedli, P., Burg, A.: Matrix decomposition architecture for MIMO systems: Design and implementation trade-offs. In: Proceedings of the 41st Asilomar Conf. on Signals, Systems and Computers, pp. 1986–1990 (2007)
3. Poon, A.S.Y., Tse, D.N.C., Brodersen, R.W.: An adaptive multiantenna transceiver for slowly flat fading channels. IEEE Trans. Commun. 51, 1820–1827 (2003)
4. Wang, Y., Cunningham, K., Nagvajara, P., Johnson, J.: Singular Value Decomposition Hardware for MIMO: State of the Art and Custom Design. In: IEEE Conf. on ReConFig, pp. 400–405 (2010)
5. Hemkumar, N.D., Cavallaro, J.R.: A systolic VLSI architecture for complex SVD. In: Proceedings of the 1992 IEEE International Symposium on Circuits and Systems, ISCAS 1992, pp. 1061–1064 (1992)
6. Ahmedsaid, A., Amira, A., Bouridane, A.: Improved SVD systolic array and implementation on FPGA. In: Proc. IEEE Field-Programmable Technology (FPT), pp. 35–42 (2003)
7. Senning, C., Studer, C., Luethi, P., Fichtner, W.: Hardware-efficient steering matrix computation architecture for MIMO communication systems. In: Proc. of the IEEE Int. Symp. on Circuits and Systems, pp. 304–307 (2008)
8. Perahia, E., Stacey, R.: Next Generation Wireless LANs Throughput, Robustness, and Reliability in 802.11n. Cambridge University Press, Cambridge (2008)
9. Parhami, B.: Computer Arithmetic Algorithm and Hardware Designs. Oxford University Press, New York (2000)

A Thermal-Aware Task Mapping Algorithm for Coarse Grain Reconfigurable Computing System

Shizhuo Tang, Naifeng Jing, Weiguang Sheng, Weifeng He, and Zhigang Mao

School of Microelectronics,
Shanghai Jiao Tong University
Shanghai, China

Abstract. Ever growing power density has made thermal effects one of the most crucial issues for modern VLSI designs, e.g., reports have shown that more than 50% of IC failures are related to thermal issues. However, thermal issues for Coarse Grain Reconfigurable Architectures (CGRA) have been few addressed. In this paper, a thermal-aware task mapping algorithm called Max-Min algorithm is developed for the REmus reconfigurable architecture, which uses compact thermal model based on equivalent thermal circuit to iteratively optimize the power dissipation on the modern CGRAs. Experiments based on Hotspot simulation show that the algorithm can reduce the maximum temperature by 3~9 ℃ and narrow the temperature distribution range by 7~15 ℃. Compared to previous intuitive random algorithm, the Max-Min algorithm can significantly reduce the number of optimization iterations while reserving the same result.

Keywords: Reconfigurable computing system, thermal model, thermal management, Max-Min algorithm.

1 Introduction

Increasing power density has been one of the most urgent obstacles to the continuously scaling VLSI systems [1]. As power dissipation is spatially non-uniform across the entire chip, localized heating occurs much faster than chip-wide heating. The so-called "hot spot" and spatial gradients can cause timing errors or even physical damage to the chip. In order to avoid the problems caused by excessive heating, it is necessary to monitor the temperature and apply thermal management techniques to balance the heat across the chip.

For the thermal issues in modern microprocessor domain, the most common method is dynamic thermal management (DTM) [2]. It applies different techniques, e.g. dynamic voltage and frequency reduction (DVFS) [3] or clock gating to reduce the chip's power consumption with hardware cost and performance loss. To minimize the loss, software methods such as Dynamic Repartitioning algorithm, Dynamic Core Scaling algorithm [4] and power-aware real-time scheduler [5] are also proposed. However, the lack of accurate thermal model limits their practical use. To construct an accurate thermal model to tackle the issues, two general methods are widely used.

One is analytical thermal model, i.e., ATMI [6], which solves the heat equation based on physical structure to get the temperature. The other is compact thermal model, e.g. Hotspot [7], which uses thermal resistances and capacitances to assemble the model. The analytical thermal modeling approach does not take package into consideration and is not fast enough to solve heat equations. In contrast, hotspot modeling is able to provide detailed static and transient temperature information efficiently across the die and the package.

Recently, reconfigurable devices are becoming more and more popular due to the requirements for more flexibility and higher performance. Typically, by customized device reconfiguration, the CGRA [8] can reduce the non-recurring engineering cost of VLSI chips but still yields higher area efficiency. Though the thermal problem on reconfigurable device is not so serious as that in microprocessors, given high clock frequencies, and extreme operating environments, reconfigurable device can easily run overheated, and cause many issues on performance and reliability [9]. Unfortunately, so far there are only a few methods published to mitigate the thermal problems in reconfigurable device, i.e., dynamic thermal management for FPGA [10] and thermal-aware task mapping [11] for coarse-grain dynamic reconfigurable processor. However, these work aim to monitor the temperature on the targeted device, and so far no effective methods are proposed to solve the thermal issues in CGRA.

To this problem, a thermal aware task mapping algorithm called Max-Min is proposed, which is efficient to reduce the peak temperature and balance the heat distribution across a coarse-grain reconfigurable platform by taking different mapping effects into account. To accurately evaluate the temperature, our evaluation is based on a detailed thermal model of the target architecture using Hotspot. Hotspot is capable of solving large scale problems due to its simple model structure, and thus is widely used for thermal evaluation. The experiment results show that our algorithm is very effective to reduce the peak temperature and balance the heat distribution across the chip. In addition, we provided both optimality and complexity analyses on our proposed task mapping algorithm for its practical use, which demonstrate the effectiveness and efficiency of our proposed algorithm.

The rest of this paper is organized as follows: section II provides the dedicated reconfigurable device architecture and constructs the thermal model. Section III proposes the thermal optimization algorithm for the task mapping process. Experiments are carried out and results are analyzed in section IV. Section V concludes this paper and talks about the future work.

2 Thermal Model of the Target Device

REmus reconfigurable architecture mainly consists of an ARM processor, SRAM, DMA controller, interrupt request controller and a reconfigurable processing unit (RPU), as shown in Fig. 1(a). The RPU, designed to speed up the data-intensive multimedia applications, consists of constant memory, load/store FIFO, configuration register, controller and an 8×8 array of 64 reconfigurable cells (RC), which is called

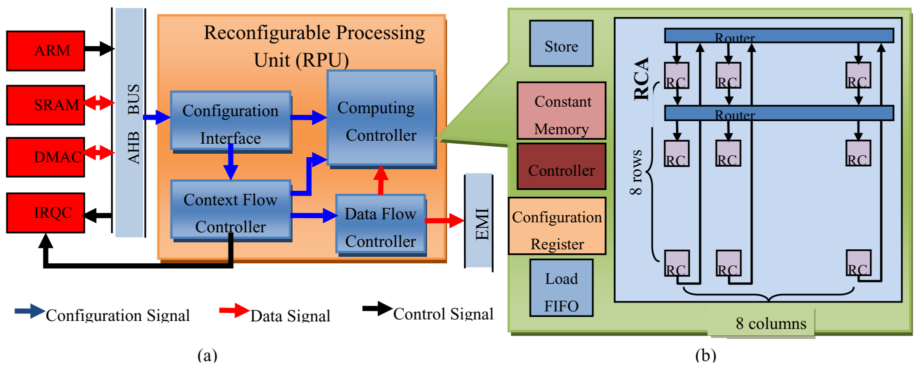


Fig. 1. Our Reconfigurable Processing Architecture

reconfigurable array (RCA) as shown in Fig. 1(b). Each RC is composed of computing unit, multiplexers and data registers, and can conduct different logic operations by configuring context words. Each row of RCs can access the results of previous row through the router. The router provides data access between any adjacent rows of RCs. Furthermore, the last row of RCs can also be accessed by the first row of RCs, so that the RCA formed a cyclic data path. Each RC can also read/write directly from the load/store FIFO, which is used to buffer the data from SRAM.

Generally speaking, targeting on a given reconfigurable architecture, the mapping problem is to place the logic operations onto RCA, and uses built-in interconnections to link the operations for desired functionality. Different mapping solutions will result in different performance and power. In this paper, we will focus on the thermal effects during mapping to reduce the maximum temperature and balance heat distribution of the design.

To construct the compact thermal model for thermal effect evaluation on RPU, detailed information of RPU is needed, such as the area, thickness and thermal conductivity of the chip. To make the model easy to construct, we have simplified the hardware architecture and focus on the RCA, because RCA occupied 80% area and power consumption of RPU.

The compact thermal model of RCA is shown in Fig. 2(a), according to the hotspot [7] model package, which includes the silicon layer, spreader, interface layer and heat sink, we first divide the silicon layer at the architecture-level. Note that RCA is an 8×8 array of same reconfigurable cells, 64 blocks are divided on silicon level, as shown in Fig. 2(b) in blue parts, and each of them is assigned to one node. For other layers (such as heat spreader and heat sink), we simply divide them as illustrated in Fig. 2(b), as their thermal information is enough. The central part is the area covered by another adjacent layer. This central part has the same number of nodes as its smaller neighbor layer or can collapse those nodes into fewer nodes, depending on the accuracy and computation speed requirements. The remaining peripheral part in Fig. 2(b) is then divided into four trapezoidal blocks, each assigned to one node.

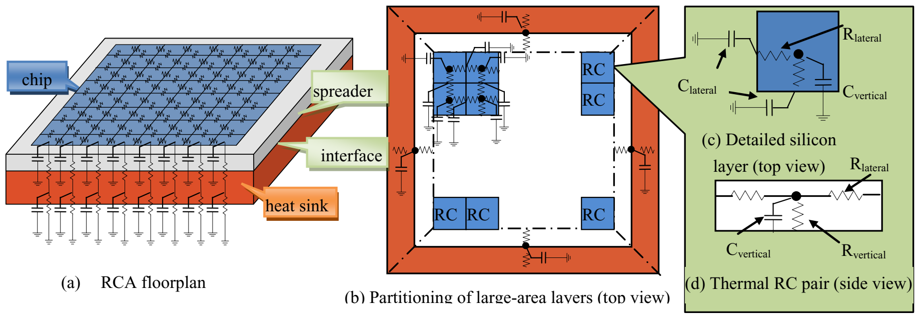

chip
spreader
interface
heat sink
(a) RCA floorplan
RC
RC
RC
RC
RC
(b) Partitioning of large-area layers (top view)
R_lateral
C_lateral
C_vertical
(c) Detailed silicon layer (top view)
R_lateral
C_vertical
R_vertical
(d) Thermal RC pair (side view)

Fig. 2. Compact model of RCA

The equivalent thermal circuit is structured as follow: every block in each layer has one vertical capacitance connected to the ground and several lateral capacitances between the centers of each shared edge and the ground, as shown in Fig. 2(c). And it also has one vertical thermal resistance connected to the next layer and several lateral resistances to its neighbors in the same layer, as shown in Fig. 2(d). The thermal capacitance C, calculated as in (1), is proportional to both thickness t of the material and the cross-sectional area A across which the heat is being transferred:

$$C = ctA \tag{1}$$

where c is the thermal capacitance per unit volume. The thermal resistance R, on the other hand, is proportional to the thickness t and inversely proportional to area A:

$$R = \frac{t}{kA} \tag{2}$$

where k is the thermal conductivity.

3 Thermal-Aware Task Mapping

Generally, the target application can be described by a Data Flow Graph (DFG). Since the number of RCs is limited, the task described by DFG has to be partitioned into small sub-graphs to meet the capacity of RC array, which is called task compilation in this paper. Our thermal aware task mapping is performed during task compilation. To monitor the temperature of the RCA when executing the divided sub-graphs, we compute the temperature by the above compact thermal model and the power dissipation on each block. Aiming to find an optimized mapping solution in terms of reduced maximum temperature and balanced heat distribution of the design, the problem can be mathematically formed as follows:

$$\begin{gathered} \mathit{minimize}\ \{\max T\},\ T \in T^n \\ GT = P,\ \ P \in \{P^n\} \end{gathered} \tag{3}$$

where T is a vector consists of 64 temperature values of each RC, and P is a vector consists of 64 power dissipation when RC is executing operations under different configurations and different inputs. $\{P^n\}$ is the set of all possible power vectors. G is the 64×64 matrix of thermal conductivity, which is parameterized and set by the file of model parameter, including the physical parameter such as the width, height of RC and thermal conductivity.

As reducing the power dissipation can reduce the temperature of RC, operations migration with higher and lower power dissipation operations is an intuitive way to balance the temperature. Inspired from this, we swap the operations between "cold" and "hot" RCs to reduce the peak temperature, and thus call it as Max-Min algorithm. During the swapping, we have to keep the data dependency among different operations on each RC, which means the critical path of initial sub-graph is not changed.

The pseudo-code of our Max-Min algorithm implementation is shown below:

```
for t = 1:rand_times
   Randomly mapping DFG on RCA
   //Main procedure
   for r = 1:row      //Search each row
       for n = 1:best_beg
           Do Max-Min operation in Row r
           Set Row n as the beginning row
           Solve the thermal equation
           if Tcur_max < Tmax
               Tmax = Tcur_max
           end
       end
   end
   for n = 1:col/2    //Max-Min operation
       for r = 1:row
           Set Row r as the beginning row
           Solve the thermal equation
       end
   end
   //Main procedure ends
end
```

where two nested loops are applied for an optimal solution. The main procedure of algorithm in the inner loop is performed as follows. First, starting from a random valid DFG mapping, the algorithm searches each row of the RCA in turns and swaps the operations with the maximum and minimum temperatures, i.e., Max-Min operation. Then, making uses of the structure of RCA with circular data path, it selects a row as the first row of the sub-graph, and calculates on-chip temperature when the position is changed. Next, search each row of RCA and swap the operations with the maximum and minimum temperatures on RC after getting a best beginning row. Finally, select the best mapping as one optimized solution.

Then, we use an outer loop for multiple optimal solutions by repeating the Max-Min algorithm in the inner loop. Due to the temperature of one RC is not only related to the computing power density but also related to the temperature of RC around, an initial mapping does not always get an optimal solution. So we repeat the Max-Min algorithm by randomly initializing the sub-graph of DFG, for several times and select the best mapping among them. Note that the initial random mapping of DFG have no correlation among each other, it can be executed in parallel to speed up the algorithm.

4 Experiment Results and Analysis

The dedicated RCA architecture well suits the multimedia applications by exploiting their explicit parallelism. In our experiments, we used several typical algorithms in multimedia applications to see the effectiveness of our proposed algorithm, e.g. 2x2 Matrix multiplication (Mat-2x2), 4x4 Matrix multiplication (Mat-4x4), 8 points Fast Fourier Transform (FFT8), 8 points Inverse Discrete Cosine Transform (IDCT8-row, IDCT8-col and IDCT8) and 32 points Discrete Cosine Transform (DCT32). We use (1) and (2) to get RC that forms G in (3), and use SPICE under different configurations and inputs to get the power dissipation P in (3). Then we can solve (3) by the hotspot simulator.

4.1 Results on Mapping Optimization

For the thermal evaluation, we perform analysis by executing the same configuration repeatedly, so that the temperature of the device can converge to a steady value. As shown in Fig. 3, the temperature distribution map of IDCT8_COL before and after optimization. Each rectangle in the plane represents a RC element. From the figure we can see that the peak temperature is reduced, and the heated area becomes more evenly distributed across the chip. For a clearer view of the peak temperature reduction, we plot the temperature distribution curve in Fig. 4, where we can see that the maximum temperature on RCA is reduced from 59.22℃ to 51.59℃. At the same time, the range

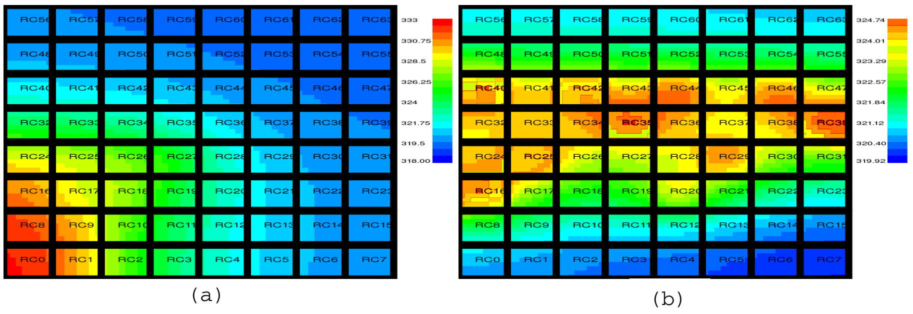


Fig. 3. IDCT8_COL temperature distribution map (a) before (b) after optimization

of concerned kernel is reduced from 20℃ to 10℃. Detailed statistical results are listed in Table 1, where the ratio η calculated by (4) shows the ratio of peak temperature reduction from the initial state by a typical ambience temperature of 45℃ [7]:

$$\eta = \frac{T_{opt} - T_{orig}}{T_{orig} - T_{amb}} \times 100\% \quad (4)$$

where T_{orig}, T_{opt} are the temperatures before and after optimization; T_{amb} is the typical ambience temperature. From the table, we can see that, according to different cases, the maximum temperature on the target device can be reduced by 3~9℃ lower after optimization. However, if the RCA has less spare computational units after mapping, the optimization space of our proposed algorithm is limited, as shown by DCT32.

Table 1. Maximum temperature of RCA with different algorithm

Algorithm	*Application*	*Mat 2x2*	*Mat 4x4*	*FFT8*	*IDCT8_ROW*	*IDCT8_COL*	*IDCT8*	*DCT32*
Origin	*Max Temp (℃)*	60.93	69.75	65.08	60.13	59.22	69.38	71.25
Random	*Max Temp (℃)*	56.25	67.01	59.02	51.34	51.83	66.54	71.22
	η	-29.38%	-11.07%	-30.18%	-58.43%	-51.97%	-11.65%	-0.11%
Max-Min	*Max Temp (℃)*	54.52	66.88	59.02	51.27	51.59	64.62	71.10
	η	-40.24%	-11.60%	-30.18%	-58.56%	-53.66%	-19.52%	-0.57%

We also do transient analysis to see the trend and power consumptions of different operations. Because the clock frequency of RCA is 200MHZ and the temperatures of every cycle are not needed, we sample the RCA temperature every 1000 cycles. And the configuration time, which is about ten thousand cycles, has also been considered, during which the temperature will be cooling down.

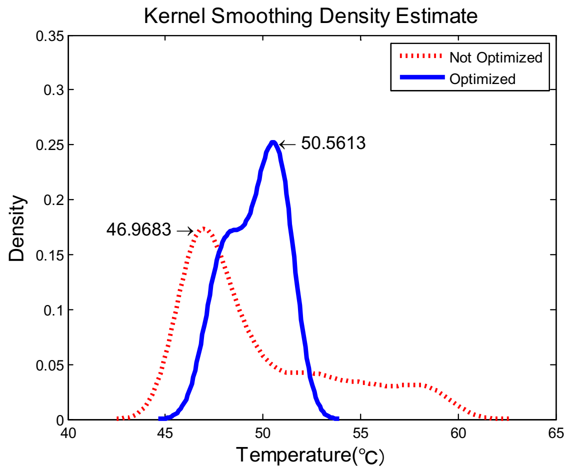


Fig. 4. IDCT8_COL temperature distribution curve

4.2 Optimality Analysis

In our experiments, we also studied the optimality of our Max-Min algorithm. Generally, given a DFG going to be mapped on a RCA with a dimension of $m \times n$, there are $m \times (n!)^m$ different mapping solutions with varied thermal results. For example, for an array of 64 RCs, the number of possible mapping may be 5.6e37. Table 2 listed the number of mapping in the second row for the 8 applications concerned in this paper.

To see the optimality of the Max-Min algorithm, we computed the minimum bound of the maximum temperature and compared to our result, which are also listed in Table 2. Considering the problem size, we can calculate the optimal solution by enumerating the mapping of Mat 2x2 and Mat 4x4. For other applications, we can approximate the minimum bound by unifying different operations on the RCs with the operation with the lowest power dissipation. By unifying the higher power dissipation operations to the lowest power dissipation operation, we can reduce the solution space for mapping, and hence we can get the minimum bound by enumerating. According to the number of nodes replaced (the more nodes replaced, the higher order approximation), we compute the one-order approximated solution of FFT8, IDCT8_ROW, IDCT8_COL and two-order approximated solution of IDCT8, DCT32. The minimum bounds are listed in the Table 2, where we can see that the Max-Min algorithm has almost achieved the optimal results, within less than 0.2% for all the cases.

Table 2. Minimum bound Maximum temperature of RCA

Application	*Mat 2x2*	*Mat 4x4*	*FFT8*	*IDCT8_ROW*	*IDCT8_COL*	*IDCT8*	*DCT32*
Num of Mapping	560	560	1,756,160	235,200	1.2e7	2.2e18	1.2e20
Bound Temp (℃)	54.52	66.88	59.02	51.26	51.58	64.50	71.01
Approximation	None	None	One	One	One	Two	Two
Compared to Max-Min	-0%	-0%	-0%	-0.0195%	-0.0194%	-0.186%	-0.127%

4.3 Complexity Analysis

Finally, we study our proposed Max-Min algorithm's complexity. Based on the RCA architecture, an accurate number of the swapping operations can be given in (5):

$$Complexity = (row^2 + \frac{1}{2} col \times row) \times rand\_times \tag{5}$$

where complexity is evaluated by the number of a 64×64 matrix multiplication, row is the number of row, *col* is the number of column, those values are 8, *rand_times* is the number of initial DFG mapping generated randomly.

Table 3 lists the correlations between computational complexity and optimization for these cases. Since initial mapping is generated randomly, the results will be a little different each time. When the *rand_times* is more than 100, the maximum temperature of each simulation with the same parameter varies less than 0.02℃. Considering the complexity and optimization results, we select *rand_times* = 100, which means the complexity is 9,600. While the random algorithm in [11] achieves a similar result requires 320,000 matrix multiplication, which is about 30 times larger than Max-Min algorithm.

Table 3. Max-Min Algorithm Correlations between Complexity and Optimization

Random Times	*Complexity*	*Maximum Temperature (℃)*		
		FFT8	*IDCT8*	*DCT32*
1	96	59.28	71,21	67.93
10	960	59.02	71.15	66.21
50	4,800	59.02	71.11	65.22
100	9,600	59.02	71.10	64.62
200	19,200	59.02	71.12	64.61
500	48,000	59.02	71.11	64.61
1,000	96,000	59.02	71.10	64.60

5 Conclusion and the Further Work

Targeting on a coarse-grained reconfigurable processor, this paper studies the thermal effects of different mapping solutions and proposed a thermal-aware task mapping algorithm to reduce the peak temperature while balancing the heat distribution across the chip. Starting from an initial random arrangement of the computational work load, the algorithm attempts to cool the device by swapping the cool and hot area when executing the work load after configuration. By heuristics, Max-Min algorithm can quickly converge to a near optimal solution, which is only about 1/30 time comparing with a random algorithm [11]. The experimental results show that Max-Min algorithm can reduce the average maximum temperature about 3~9℃ and narrow temperature distribution range about 7~15℃. According to the results above, Max-Min task mapping algorithm is efficient for thermal management in using coarse-grain reconfigurable computing system.

The current thermal-aware algorithm only focuses on RCA, and we will develop more comprehensive thermal mode for other components of the reconfigurable processor in the future.

Acknowledgement. The authors would like to thank for the support of the Hi-tech Research and Development Program (863) of China under Grant No. 2009AA011705.

References

1. Wei, H., Ghosh, S., Velusamy, S., Sankaranarayanan, K., Skadron, K., Stan, M.R.: HotSpot: a compact thermal modeling methodology for early-stage VLSI design. IEEE Transactions on Very Large Scale Integration (VLSI) Systems 14(5), 501–513 (2006)
2. Brooks, D., Martonosi, M.: Dynamic thermal management for high-performance microprocessors. In: The Seventh International Symposium on High-Performance Computer Architecture, HPCA 2001, pp. 171–182 (2001)
3. Kondo, M., Sasaki, H., Nakamura, H.: Improving fairness, throughput and energy-efficiency on a chip multiprocessor through DVFS. ACM SIGARCH Computer Architecture News 35(1), 31–38 (2007)

4. Seo, E., Jeong, J., Park, S., Lee, J.: Energy Efficient Scheduling of Real-Time Tasks on Multicore Processors. IEEE Transactions on Parallel and Distributed Systems 19(11), 1540–1552 (2008)
5. Bautista, D., Sahuquillo, J., Hassan, H., Petit, S., Duato, J.: A simple power-aware scheduling for multicore systems when running real-time applications. In: IEEE International Symposium on Parallel and Distributed Processing, IPDPS 2008, April 14-18, pp. 1–7 (2008)
6. Michaud, P., Sazeides, Y.: ATMI: Analytical model of temperature in microprocessors. In: Proc. MoBS, pp. 1–10 (2007)
7. Stan, M.R., Skadron, K., Barcella, M., Huang, W., Sankaranarayanan, K., Velusamy, S.: HotSpot: a dynamic compact thermal model at the processor-architecture level. Microelectronics Journal 34(12), 1153–1165 (2003)
8. Singh, H., Lee, M.-H., Lu, G., Kurdahi, F.J., Bagherzadeh, N., Chaves Filho, E.M.: MorphoSys: an integrated reconfigurable system for data-parallel and computation-intensive applications. IEEE Transactions on Computers 49(5), 465–481 (2000)
9. Jones, P.H., Lockwood, J.W., Cho, Y.H.: A Thermal Management and Profiling Method for Reconfigurable Hardware Applications. In: International Conference on Field Programmable Logic and Applications, FPL 2006, pp. 1–7 (August 2006)
10. Velusamy, S., Wei, H., Lach, J., Stan, M., Skadron, K.: Monitoring temperature in FPGA based SoCs. In: Proceedings of the 2005 IEEE International Conference on Computer Design: VLSI in Computers and Processors, ICCD 2005, October 2-5, pp. 634–637 (2005)
11. Xie, L., He, W., Jing, N., Mao, Z.: A thermal-aware task mapping flow for coarse-grain dynamic reconfigurable processor. In: 2011 IEEE International Symposium on Circuits and Systems (ISCAS), May 15-18, pp. 1952–1955 (2011)

DC Offset Mismatch Calibration for Time-Interleaved ADCs in High-Speed OFDM Receivers[*]

Yulong Zheng, Zhiting Yan, Jun Ma, and Guanghui He

School of Microelectronics, Shanghai Jiao Tong University, Shanghai, China
{zhengyulong,yanzhiting,majun,heguanghui}@ic.sjtu.edu.cn

Abstract. Zero Intermediate Frequency (zero-IF) receivers with two analog-to-digital converters (ADCs) in In-Phase and Quadrature (IQ) branches are widely used in emerging multi-Gigabit wireless Orthogonal Frequency Division Multiplexing (OFDM) systems. Because ordinary ADCs could not meet the demands of sampling rate in the system, two time-interleaved analog-to-digital converters (TI-ADCs) could be an attractive alternative for sampling speed improvement in the receiver. However, the mismatches among the parallel sub-ADCs can degrade the performance significantly without calibration. Targeting the DC offset mismatch of the TI-ADCs, this paper proposes calibration algorithm based on decorrelation least-mean-squares (LMS) and recursive-least-square (RLS) utilizing the comb-type pilots in OFDM frame, which could calibrate the two TI-ADCs in (IQ) branches simultaneously. The calibration algorithm has the property of fast convergence. Simulation results show that the BER performance is improved by the proposed algorithm.

Keywords: Multi-Gigabit, time-interleaved ADC, mismatch, calibration, parameter estimation, orthogonal frequency division multiplexing (OFDM).

1 Introduction

Realizing high-speed ($> 1GHz$), sufficient precision (8-10 bits), and low-power cost ($< 100mW$) analog-to-digital converters (ADCs) has been a bottleneck [1] in emerging multi-Gigabit wireless communication system (e.g. UWB, 60GHz wireless) with ultra large bandwidth. For instance, IEEE 802.15.3c requires the ADC sampling at 2.640GHz and 2.538GHz in High-Speed-Interface Orthogonal Frequency Division Multiplexing (HSI OFDM) mode and Audio Visual (AV) OFDM mode respectively [2]. Existing techniques and devices could not meet such a requirement or would work at the expense of unacceptable high costs. Time-interleaved (TI) architecture provides a promising approach to solve this problem by interleaving several slower sub-ADCs in parallel. As shown in Fig. 1, a time-interleaved analog-to-digital converter (TI-ADC) linearly increases its sampling rate with the number of interleaved sub-ADCs, and the resolution of the array is dependent on the resolution of sub-ADCs.

[*] This work was supported in part by Shanghai Natural Science Foundation under Grant No. 10ZR1416500 and the Research Fund for the Doctoral Program of Higher Education of China under Grant No. 20110073110055.

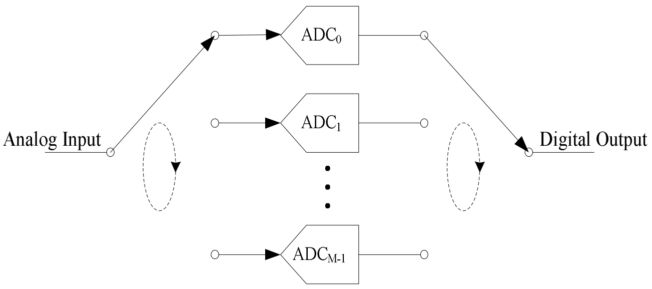


Fig. 1. A TI-ADC block diagram formed by M sub-ADCs

However, the mismatches among sub-ADCs degrade the performance of the TI-ADC. Typically, there are three major types of mismatches must be considered: DC offset, gain, and timing mismatches [3].

DC offset mismatch is one of the major mismatches in a TI-ADC. As a result, the mismatch calibration has received significant attention. For example, existing calibration methods employ the redundancy such as additional hardware or training sequences [5], [6].

OFDM technique is widely adopted by high-speed wireless communication such as MB-UWB, and IEEE 802.15.3c. TI-ADCs used in OFDM system can be calibrated by leveraging the training sequences such as preamble or pilot signals. The DC offset mismatch calibration in [3] constructs the objective function, use optimization method to estimate the parameter and then eliminate its effect. However, the method can only apply for one ADC in the receiver and it needs a large number of OFDM frames need to be averaged. Thus it will lead to slow convergence. The proposed calibration in [10] is based on LMS algorithm using block-pilot signals. However, block-pilot signals are adopted by few protocols. Targeting calibrating DC offset mismatch of two TI-ADCs in In-Phase and Quadrature (IQ) branches and improving the convergence speed, we extend the model and method proposed in [3], and then propose the calibration algorithm using comb-pilot based on decorrelation least-mean-square (LMS) and recursive-least-square (RLS) algorithms.

Section 2 describes the models of TI-ADC and OFDM system. Section 3 presents the calibration basis and the proposed algorithm. Simulation results are shown in Section 4, and finally, conclusions are drawn in section 5.

2 System Model

In this section, we first present the model of TI-ADC with DC offset mismatch. Secondly, in order to understand the attributes flow in the communication link, we describe the OFDM system that utilizes TI-ADCs both in IQ branches in the receiver. The RF front-end offers the analog input of the TI-ADC, and the baseband processor is fed with the digital output of the TI-ADC.

2.1 DC Offset Mismatch Model in TI-ADC

Taking DC Offset Mismatch into consideration only, we describe the model of a TI-ADC. Letting T_s denotes the nominal sampling period, and $1/T_s$ is the sampling rate, the output of TI-ADC is written as [4]:

$$v[m] = v(mT_s) + \mu_{m \bmod M} \tag{1}$$

Where $v[m]$ denotes the m_{th} digital quantized sample, sampled by the $(m \bmod M)_{th}$ sub-ADC, where mod denotes the modulo operation and M denotes the number of sub-ADCs. Sub-ADC with index of $m \bmod M$ has the DC offset $\mu_{m \bmod M}$.

As a result of the mismatch parameter drifting over hours, we set the mismatch parameter to constant values in our work. Since the quantization noise is small enough compared with the DC offset mismatch–induced interference and the thermal noise, we ignore the former in our analysis [4].

2.2 OFDM Model

Orthogonal frequency-division multiplexing (OFDM) is an advanced physical layer multicarrier modulation technique, which is widely used in many wireless standards. Fig. 2 illustrates the signal flow of OFDM both in transmitter and receiver.

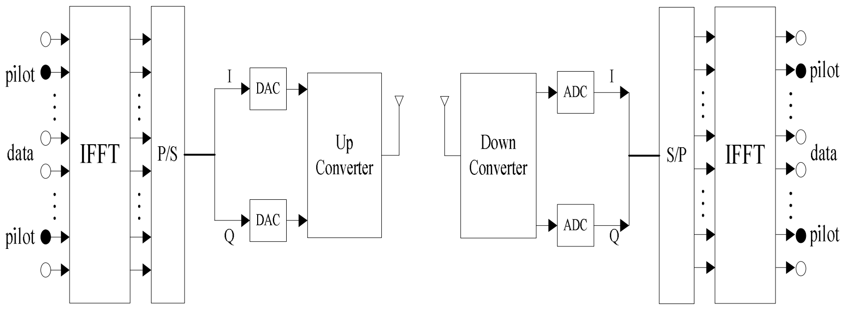


Fig. 2. OFDM transmitter (DAC = Digital-to-Analog Converter) and receiver

The basic unit of the data stream is called an OFDM frame, and the size of an OFDM frame (denoted by N) is identical to the size of IFFT and FFT in transmitter and receiver. A typical frame consists of information bearing subcarriers, pilot subcarriers, and virtual subcarriers. The pilot subcarriers are deterministic to make coherent detection robust against frequency offsets and phase noise and perform the channel estimation [7]. In this paper, we use the pilot signals to do the calibration work.

For simplicity, the calibration of DC offset mismatch for the TI-ADC in the OFDM receiver is implemented in an AWGN channel in the analysis below.

3 ADC Calibration Using Pilot Signals

In this section, we describe how we employ pilot signals to estimate and calibrate the DC offset mismatch between sub-channels of TI-ADCs in IQ branches. As the prior work [3] is about the case for one ADC, which cannot be directly applied in the direct down conversion receiver, we extend its mathematical model. Targeting the disadvantages of the speed of convergence in the algorithms proposed in [3], we propose the decorrelation LMS [8] and RLS algorithm [9] to solve the problem.

3.1 Constrains on Sub-ADC Numbers and FFT Size

It is proved in [3] that the number of sub-ADCs M must be prime with the FFT size N so that the error caused by DC offset mismatch could spread out over the entire subcarriers. If N is multiple of M, the error caused by mismatch could only affect the subcarriers with the index multiple of N/M, while others is not affected by the mismatch. Under such a circumstance, if pilot subcarrier is not with the index multiple of N/M, the calibration will not work. In our proposed algorithms, we adopt the former constrain that M is prime with N.

Note that, in branch I or Q, the number of sub-ADCs has the same value of M. Totally, there are $2M$ sub-ADCs in a direct down conversion receiver.

3.2 Relationship between DC Offset Mismatch and Its Effect on Pilot Signals

In the transmitter, the l_{th} transmitted OFDM symbol can be expressed as:

$$\mathbf{x}_l = IFFT(\mathbf{X}_l) \tag{2}$$

Where $\mathbf{X}_l$ denotes the l_{th} OFDM frame in frequency domain before IFFT operation, $\mathbf{x}_l$ denotes the l_{th} OFDM frame in the time domain, $IFFT(\bullet)$ represents an energy preserving IFFT operation.

After experiencing the additive white Gaussian noise (AWGN) channel, the signal in the receiver, which lies before the ADCs, can be written as:

$$\mathbf{x}_{l\ received} = \mathbf{x}_l + \mathbf{w} \tag{3}$$

Where $\mathbf{w}$ denotes the AWGN vector with the size of N with $E[\mathbf{w}\mathbf{w}^H] = \sigma^2\mathbf{I}$. σ^2 is the variance and N is the FFT size.

Adding the DC offset mismatch of TI-ADC, and the recovered OFDM frame after FFT operation can be expressed as:

$$\hat{\mathbf{X}}_l = FFT(\mathbf{x}_{l\ received} + \overline{\boldsymbol{\mu}}) \tag{4}$$

Where $\bar{\boldsymbol{\mu}} = [\mu_0 \quad \mu_1 \cdots \quad \cdots \quad \mu_{N-2 \bmod M} \quad \mu_{N-1 \bmod M}]^T$, μ_i has a complex value, the real part is the DC offset of the sub-ADCs with the index of i in I branch, and the imaginary part is the DC offset of the sub-ADCs with the index of i in Q branch, $FFT(\bullet)$ represents an energy preserving FFT operation.

Combining equation (2), (3) and (4), we can obtain:

$$\hat{\mathbf{X}}_l = \mathbf{X}_l + FFT(\bar{\boldsymbol{\mu}}) + \mathbf{W} \tag{5}$$

Where $\mathbf{W}$ is the FFT of $\mathbf{w}$. While FFT is an energy preserving transform, $\mathbf{W}$ and $\mathbf{w}$ has the same statistic characteristics.
Substituting FFT matrix for $FFT(\bullet)$, extracting the pilot signals, revising $\bar{\boldsymbol{\mu}}$ in equation (5) into $\boldsymbol{\mu} = [\mu_0 \quad \mu_1 \quad \cdots \quad \mu_{M-1}]^T$, and then subtracting the known pilot values, the equation (5) can be revised in matrix form as:

$$\mathbf{E} = \mathbf{F}\boldsymbol{\mu} + \mathbf{W} \tag{6}$$

Where in (6), $\mathbf{E}$ is the error column vector in pilot subcarriers caused by the offset mismatch with the length of L, the length is equal to the number of pilots in an OFDM frame, $\mathbf{W}$ is an AWGN noise vector of length of L, $\mathbf{F}$ is a $L \times M$ matrix , which is could be obtained by the column and row operations of the standard FFT matrix.

Equation (6) is the extended form proposed in [3]. In equation (6), two TI-ADCs embedded in IQ branches are taken into consideration.

As we shall show, the equation (6) gives the relationship between DC offset mismatch and its effect on pilot subcarriers. The vector $\mathbf{E}$ is the error observation we find in received pilot subcarriers, and $\boldsymbol{\mu} = [\mu_0 \quad \mu_1 \quad \cdots \quad \mu_{M-1}]^T$ is the offset that is need to be estimated in each sub-ADCs in the TI-ADCs in IQ branches. $\mathbf{W}$ is the noise vector, which could be reduced by averaging as in [3]. Thus, equation (6) is a classic linear model that could be solved by different methods [9].

3.3 Estimation Algorithm Based on Decorrelation LMS and RLS

Careful inspection of equation (6) suggests the LS [9] estimation, which is shown as:

$$\hat{\boldsymbol{\mu}} = (\mathbf{F}^H\mathbf{F})^{-1}\mathbf{F}^H\mathbf{E} \tag{7}$$

Where $(\bullet)^H$ and $(\bullet)^{-1}$ denotes the Hermitian transpose and inverse matrices. In (7), the estimation of offset vector $\hat{\boldsymbol{\mu}}$ only under the circumstance of $L \geq M$, which means that the number of the pilot is larger than the offset to be corrected, can be written as shown above. At the same time, $\mathbf{F}^H\mathbf{F}$ must be invertible, which means $rank(\mathbf{F}^H\mathbf{F}) = M$. Numerical simulation shows that under most circumstances,

$rank(\mathbf{F}^H\mathbf{F}) = M-1$. In order to meet the condition above, we need to add the error of DC subcarrier in the OFDM frame to the error observation vector $\mathbf{E}$, which is a $(L+1)$ vector, and additionally extract the first line of standard FFT matrix to make $\mathbf{F}$ a $(L+1)\times M$ matrix. This operation requires that, before ADC, the DC offset of the RF front-end must be cleaned and this can be achieved by AC-coupled or compensating in the analog domain in RF front-end.

Note that after adding the error of the DC tone in vector $\mathbf{E}$, the coordinate descend method proposed in [3] is also available without setting the offset of first unit to zero. But it converges too slowly.

As we can see, directly applying the LS estimation would require a prohibitively high cost, because matrix inversion requires a lot of computations. This can be avoided by using iterative algorithm like LMS and RLS algorithms.

A. Decorrelation LMS Algorithm

In decorrelation LMS algorithm, $\hat{\boldsymbol{\mu}}$ vector is computed as Table 1 shows [8]:

Table 1. Decorrelation LMS algorithm steps

```
Initial:
$\hat{\boldsymbol{\mu}}_0 = 0$;
Steps:
for $l = 1,2,3...$
1. $e_{l+1} = \mathbf{E}(l+1) - \mathbf{F}(l+1,:)\hat{\boldsymbol{\mu}}_l$
2. $a_{l+1} = \frac{\mathbf{F}(l,:)\mathbf{F}(l+1,:)^H}{\mathbf{F}(l,:)\mathbf{F}(l,:)^H}$
3. $\mathbf{P}_{l+1} = \mathbf{F}(l+1,:) - a_{l+1}\mathbf{F}(l,:)$
4. $\eta_{l+1} = \frac{e_{l+1}}{\mathbf{F}(l+1,:)\mathbf{P}_{l+1}}$
5. $\hat{\boldsymbol{\mu}}_{l+1} = \hat{\boldsymbol{\mu}}_l + \eta_{l+1}\mathbf{P}_{l+1}$
```

In Table 1, l represents the number of iteration, η is the step size, and the $\mathbf{F}(i,:)$ denotes the i_{th} row of $\mathbf{F}$. While updating the $\hat{\boldsymbol{\mu}}$ vector, the decorrelation LMS algorithm uses the correspondent previous and current rows of $\mathbf{F}$ in each iteration. Note that the algorithm we adopt is decorrelation LMS algorithm instead of the original LMS algorithm. In our application the correlation of the input vectors is relatively large, and the decorrelation LMS algorithm can remove the correlation of the input vectors to increase the convergence speed.

B. RLS Algorithm

In RLS algorithm, $\hat{\boldsymbol{\mu}}$ vector is updated as follows in Table 2 [9]:

Table 2. RLS algorithm steps

Initial:
$\hat{\boldsymbol{\mu}}_1 = 0$
$\boldsymbol{\phi}_1 = \frac{1}{\varphi_1}\mathbf{I}$
$0 < \lambda < 1$, λ is the forgetting factor
Steps:
for $l = 1,2,3...$
1. $\beta_{l+1} = \frac{\lambda^{-1}}{1+\lambda^{-1}\mathbf{F}(l+1,:)\boldsymbol{\phi}_l\mathbf{F}(l+1,:)^H}$
2. $\mathbf{m}_{l+1} = \beta_{l+1}\boldsymbol{\phi}_l\mathbf{F}(l+1,:)^H$
3. $\hat{\boldsymbol{\mu}}_{l+1} = \hat{\boldsymbol{\mu}}_l + \beta_{l+1}\boldsymbol{\phi}_l\mathbf{F}(l+1,:)^H(\mathbf{E}(l+1) - \mathbf{F}(l+1,:)\hat{\boldsymbol{\mu}}_l)$
4. $\boldsymbol{\phi}_{l+1} = \lambda^{-1}(\boldsymbol{\phi}_l - \mathbf{m}_{l+1}\mathbf{F}_{l+1}\boldsymbol{\phi}_l)$

In Table 2, the parameter is defined the same as in the classical RLS algorithm. RLS algorithm uses one row of $\mathbf{F}$ in each iteration. This algorithm provides more accurate and faster convergence speed at expense of much more complexity.

3.4 Calibration Architecture in the Receiver

The idea of the calibration is to use the algorithms described in section 3.3 to estimate the individual offsets in sub-ADCs first, and then eliminate the effects on frequency domain. Fig. 3 shows how the background calibration works. The OFDM frames are passing in path IQ 1, and are stored in Register 1. Pilot Tone Selection module gets the frame and selects the pilot tones. After averaging to reduce the AWGN channel noise, the LMS/RLS Estimator performs the decorrelation LMS or RLS algorithm to estimate the DC offsets of the TI-ADC in IQ branches. The DC offsets in frequency domain are stored in Register 2. The calibrated data is finally obtained by subtracting the contents in Register 2.

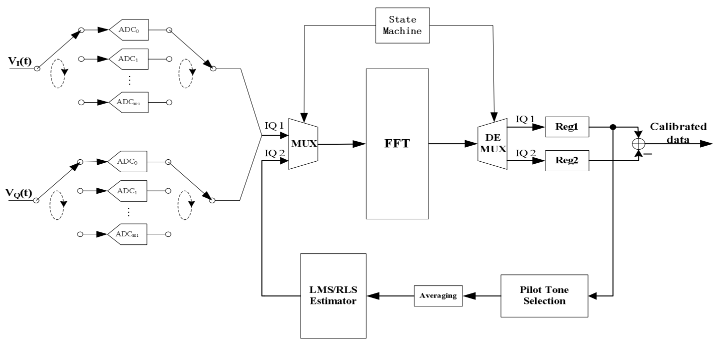


Fig. 3. Calibration Architecture using the proposed algorithm

It is also possible to perform a foreground calibration. The estimated DC offsets computed by LMS/RLS Estimator can be directly subtracted in the time domain before FFT operation.

4 Simulation Results

The calibration algorithms described above are evaluated in simulations developed in MATLAB. The OFDM system we use is the MB-UWB OFDM systems in the AWGN channel [7]. Two TI-ADCs sampling at 500MS/s are used in IQ branches, with 5 sub-ADCs in each one. The system uses QPSK modulation and there are 12 pilots in one OFDM frame, and the IFFT/FFT size is 128.

4.1 Offset Estimation and Comparisons

In section 4.1, we set the mismatch in the level of 10%. By 10% mismatch, we mean the values of DC offset mismatch are chosen uniformly in $[-A/10, A/10]$, where the A is the RMS value of the TI-ADC input signal's amplitude. The offset values of the estimation on the vertical axis are normalized by the RMS value in digital domain. For QPSK, the RMS value is equal to $1/\sqrt{2}$.

We extend the coordinate descend method which is proposed in [3]. After the numerical simulation, we find that in a noiseless channel the coordinate descend method needs about 8000 OFDM frames to achieve the convergence.

In order to make the calibration faster, we evaluate the decorrelation LMS and RLS algorithms. Fig. 4 and Fig. 5 show the process of estimation with LMS and RLS algorithm and the MSE comparison respectively:

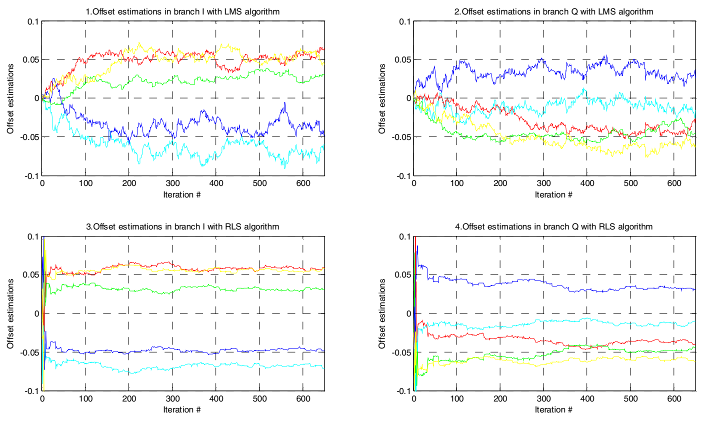


Fig. 4. Offset estimation using RLS algorithm

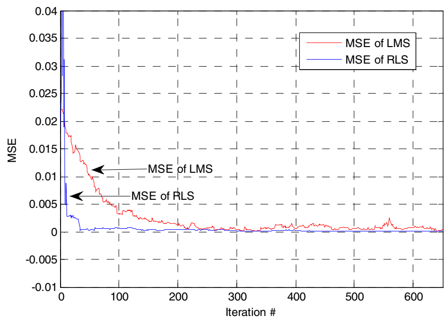


Fig. 5. MSE comparison between LMS and RLS

The decorrelation LMS algorithm and RLS algorithm in Fig. 4 and Fig. 5 are evaluated with $E_b / N_0 = 10dB$, and the averaging time is 500. The horizon axis represent the iteration number, which is not the same operation as in coordinate descend method. 13 iterations are updated in one OFDM frame. The figure shows LMS and RLS achieve the convergence after about the 200 and 40 iterations, which equally with 15 and 3 OFDM frames. Taking the averaging operation into consideration, LMS and RLS algorithm take 7500 OFDM frames and 1500 frames, respectively, to converge to acceptable estimated DC offset values.

Fig. 5 shows the mean square error (MSE) comparison between LMS algorithm and RLS algorithm. RLS algorithm takes fewer iteration times and is more accurate than LMS algorithm at expense of much more complexity.

4.2 BER Performance Comparison

Fig. 6 presents the BER performance among the calibrated, uncalibrated and the ideal MB-OFDM BER curve with E_b / N_0 from $5dB$ to $10dB$. The calibrated BER curves are nearly the same as the ideal one, which confirms the efficacy of the estimation algorithms proposed above.

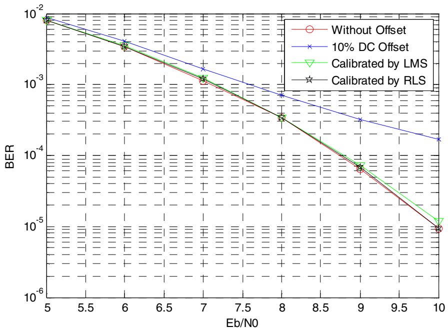


Fig. 6. BER performance comparison

5 Conclusions

In this paper, we propose a digital calibration in a high-speed OFDM system using the comb-pilot signals based on decorrelation LMS and RLS algorithms. The calibration could reduce the effect of the DC offset mismatch of two TI-ADCs in IQ branches simultaneously. And the calibration algorithm converges faster than the coordinate descend method. Furthermore, BER performance is improved by the proposed calibration. The calibration method may also be applied to 60GHz millimeter communication, optical communication and other high-speed communication systems.

References

1. Singh, J., Sandeep, P., Madhow, U.: Multi-gigabit communication: the ADC bottleneck. In: Proc. 2009 IEEE International Conf. Ultra-Wideband, ICUWB (September 2009)
2. IEEE 802.15 WPAN High rate Alternative PHY Task Group 3c, `http://www.ieee802.org/15/pub/TG3c.html`
3. Oh, Y., Murmann, B.: System embedded ADC calibration for OFDM receivers. IEEE Trans. Circuits Syst.-I 53, 1693–1703 (2006)
4. Sandeep, P., Seo, M., Madhow, U., Rodwell, M.: Joint mismatch and channel compensation for high-speed OFDM receivers with timeinterleaved ADCs. IEEE Trans. Communications 58, 2391–2401 (2010)
5. Conroy, C.S.G., Cline, D.W., Gray, P.R.: An 8-b 85-MS/s parallel pipeline A/D converter in 1-μm CMOS. IEEE J. Solid-State Circuits 28(4), 447–454 (1993)
6. Tsai, T.H., Hurst, P.J., Lewis, S.H.: Correction of mismatches in a time-interleaved analog-to-digital converter in an adaptively equalized digital communication receiver. IEEE Trans. Circuits Syst. I. Regular Papers 56(2), 307–319 (2009)
7. IEEE 802.15 WPAN High rate Alternative PHY Task Group 3a, `http://www.ieee802.org/15/pub/TG3a.html`
8. Glentis, G.-O., Berberidis, K., Theodoridis, S.: Efficent least squares adaptive algorithms for FIR transversal filtering. IEEE Signal Processing Magazine 16, 13–41 (1999)
9. Haykin, S.: Adaptive Fitler Theory, 4th edn. Prentice-Hall (2001)
10. Jin, X., He, G., Ma, J.: Offset Calibration for Time-Interleaved Analog-to-Digital Converters in OFDM Systems. In: Signal Processing Theory and Application, APSIPA ASC (2010)

A Novel Graph Model for Loop Mapping on Coarse-Grained Reconfigurable Architectures

Ziyu Yang[*], Ming Yan, Dawei Wang, and Sikun Li

School of Computer, National University of Defense Technology, Changsha, China
{zyyang,zhaopeng,dwwang}@nudt.edu.cn,lisikun@263.net.cn

Abstract. Coarse-Grained Reconfigurable Architectures (CGRAs) provide more opportunities for accelerating data-intensive applications, such as multi-media programs. However, the optimization of critical loops is still challenging issues, since there is lack of application mapping tool of CGRAs. To address this challenge, we first take program feature analysis on the kernel loops of applications. And then we propose a novel graph model called PIA-CDTG containing these features. We implement an efficient task mapping method with a genetic algorithm based on the graph model. Experimental results show that the mapping method with PIA-CDTG is more effective than other features-unaware methods, and make the execution attains high efficiency and availability.

Keywords: PIA-CDTG, Program Feature Analysis, Loop Mapping, Coarse-grained Reconfigurable Architecture

1 Introduction

The multi-media applications, which emerged in the last decade, such as image and video processing, are date intensive. Coarse-grained reconfigurable architecture (CGRA) offers the capability for spatial, parallel, and specially computation through hardware customization, that is, hardware that can be reconfigured during runtime to substantially accelerate different kind of applications. These advantages in CGRA have led to the development of heterogeneous reconfigurable platforms for media processing applications, since applications realized in hardware whose execution is much faster than those realized in software. CGRA is essentially an array of processing elements (PEs), like ALUs and multipliers, interconnected with a mesh-like network. The complexity of hardware makes programming application is a challenging task since the compiler has to map the controller tasks of the application onto the host processor and map the data-intensive tasks onto the processing elements array. After the reconfigware/software partitioning process, the entire reconfigurable mapping flow involves several primary techniques such as code transformations, loop pipelining, scheduling and the generation of reconfigware configuration. Those efforts depend on the granularity of the reconfigurable tasks, which is effected by the

[*] Corresponding author.

computational model of target architecture. Mapping applications (mostly loop tasks) automatically and efficiently onto computation and storage resources is now one of the hottest topics among researchers [1]. A novel programming model is needed to cover the complexity of mapping applications on a CGRA. The representation must allow a variety of transformations and optimizations of applications to exploit the performance of the target reconfigurable architecture.

In this paper, we present a novel graph model called PIA-CDTG (Program Information Aided Control-Dataflow Task Graph) for efficient application mapping. Based with program features, PIA-CDTG aims at providing a new task granularity of the application mapping. With the graph model, we then focus on the critical loops mapping on CGRA. A genetic algorithm is proposed for mapping PIA-CDTG tasks, and experimental results and analysis prove its efficiency.

2 Related Work

Several CGRAs have been developed with varying degree of programmability, memory structure, and communication bandwidth, like Morphosys [2] and LEAP [3], and others. However, there is a lack of efficient tools for the full utilization of the performance or flexibility offered by these architectures. For instance, the SA-C [4] of Morphosys has been developed, which is a high-level single assignment language, to process media applications using data flow graph as its intermediate representation (IR). There have been many attempts to take the big challenge of the efficient reconfigurable mapping, several researches on discovering coarse-grained parallelism were proposed, like loop shifting [5], pipeline vectorization [6], and modulo scheduling [7].

Table 1. Graph Model Comparsion

IR Types	Supported Features				
	Control Dependency	Data Dependency	Granularity	Concurrency	Hierarchy
CFG	inaccurate	×	BB	×	√
DFG	×	Def-Use	Op.	Program	×
PDG	accurate	Flow-Anti-Out	Op.	Program	√
CDFG	inaccurate	Flow-Anti-Out	BB/Op.	BB	√
PIA-CDTG	accurate	All	BB/Loop	Program	√

A variety of graph model have been proposed to make the efficient multiple threads of concurrency over distinct data items. The graph should represent all the data and control dependencies of programs efficiently and enable the transformations of programs to take advantages of the target architecture. The traditional control/dataflow graph (CDFG) is now widely used. CDFG includes both control and data flow information from the input algorithm and embeds operations in its basic-block nodes. A DFG of each basic block then constructed for mapping on RPU. A PDG [8] is a combination of a CDG and a DDG, which usually incorporated with the

SSA for synthesis to reconfigurable devices. Both the CDFG and PDG are the underlying representation of hardware behavior, providing the hierarchy of loops, branches, and function calls. A variety of IRs proposed allow compiler to adjust the granularity of its data and computation partition with the target architecture characteristics, as shown in Table 1. In Table 1, BB stands for basic block, op. stands for operator. However, there is no a widely accepted programming model, high level language, or IR will allow automatic mapping on reconfigurable architectures up to date. In this paper, the proposed graph model is differ from others, since it focus on the critical loops mapping under the CGRA's resource constraints. With considering of program features, the genetic scheduling algorithm based on PIA-CDTG make the load-balanced execution more adaptive.

3 PIA-CDTG Mapping Model

As discussed, the main challenge of accelerating data-intensive applications on CGRAs is the mapping of applications with less execution time and power consumption. By analyzing the application's features, especially the critical loops, one can make the CGRA mapping more adaptive. In this section, we discuss the details of application features and the generation of PIA-CDTG from these analysis results.

3.1 Mapping data-intensive application on CGRA

Data-intensive application execution tends to spend most of the time in frequent nested loops. Since CGRA usually used as coprocessor for accelerating these application, the key part of parallel processing is mapping critical loops. Given a data-intensive application, we define the parallel mapping problem on CGRA as a four-stage process.

(1) Application analysis and profiling. We first transformed the application into intermediate code for profiling, and then analyze application features including computation/storage distribution, critical loops relevant information, etc.
(2) Critical loop computation/memory analysis. Get the critical loops based on the application analysis, generating the program information aided control-dataflow graph (PIA-CDFG) of these loops with information such as the number of computation/storage operation, iteration number, execution time and data dependences, etc.
(3) Loop tilling for task mapping on CGRA. It is important to minimize the communication cost when loop executed on the PEs array by choosing the optimized tile loop size to fit the current memory hierarchy.
(4) Loop task scheduling on CGRA. After loop tilling, the tiled tasks need to be mapped on the CGRA PEs array for execution. Considering the resource constraint offered by architecture model, it is important to the decrease memory access number to minimize the execution time with higher performance.

3.2 Application Features Analysis

When mapping an application, it is important to get the features of program to adapt the constraints of object hardware. However, the structure of data-intensive application is too complicated to analysis manually. And we need focus on the critical parts of these applications in order to analysis more fast and efficient, which means we take the kernel loops as the object of analysis. The overview of application analysis is shown in Fig.1.

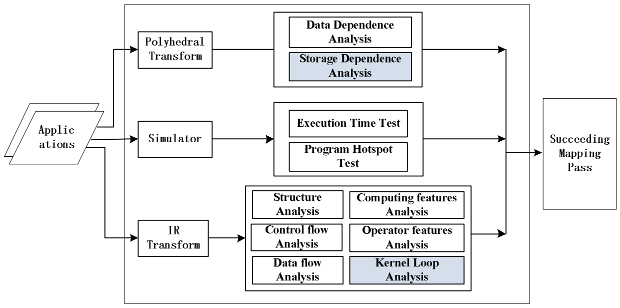


Fig. 1. Overview of the application analysis

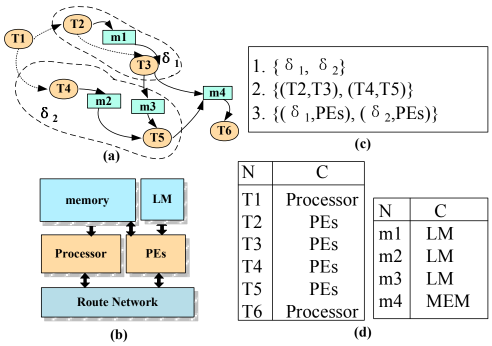
The kernel loops are the loops which take most part of the execution time in an application. Recent researches show that the optimization of kernel loops takes strong influence on performance of application. We first make the simulator to take the program hotspot test, and then mark the time-consuming parts in the application. We use the LooPo [9] to scan the annotations in the SUIF [10] IR of application and get the loop features, such like 1) Basic information, including type, index, upper-bound, limit, etc.;2) Hierarchy information, including nested level, innermost loop, etc.; 3) Loop data information, including arrays, data dependency, operator features, etc. If the loop's execution time is large than a threshold, we take it as a kernel loop.

3.3 Modeling of PIA-CDTG

Different from the traditional CDFG, the node in PIA-CDTG is no longer the simple operator but task. The property of these nodes includes the application features. With the description of hierarchy, control/data flow between nodes, PIA-CDTG can support the mapping, scheduling of application tasks.

Definition 1 *Given a PIA-CDTG = (N, E, P) consists of node N, edge E, and program profiling information P. N is the node set, including the task node set T and the storage node set M. E is the edge set with data/control dependences. Info P includes dataflow/control flow, computation/storage operations, the dependency and relevant of nodes.*

As shown in Fig. 2(a), the dotted lines indicate the control flow and the solid lines indicate the data flow over nodes. PIA-CDFG model defines architecture-independent information about field specific application program. The generation of PIA-CDTG includes four steps:



1. { δ 1, δ 2}
2. {(T2,T3), (T4,T5)}
3. {(δ 1,PEs), (δ 2,PEs)}

N	C
T1	Processor
T2	PEs
T3	PEs
T4	PEs
T5	PEs
T6	Processor

N	C
m1	LM
m2	LM
m3	LM
m4	MEM

Fig. 2. PIA-CDTG based mapping on RAG

1. Take the IR of application as input, which includes the application analysis results, perform task partitioning based on the granularity of program function, and then build the original task graph (TG);
2. Analysis the control dependences in TG, and generate the control-flow task graph (CTG);
3. Analysis the data dependences in TG, put the results into the CTG, and generate the control/data flow graph (CDTG);
4. Based on the CDTG, take the marked kernel loops as kernel task node, analysis these nodes' storage features. Generate the data dependences inside node; make the result as the info P of node. At last generate the PIA-CDTG.

The performance of application on CGRA depended on the mapping strategy. A good defined architecture model is helpful for the strategy construction. As shown in Fig. 2(b), an architecture graph describes the computing/communication/storage features of mapping object architecture. For CGRA, the architecture model is defined as:

Definition 2 *CRAG = (PR, MEM, PE, LM, RN) represents An N×M PEs array CGRA. PR is the main control processor, MEM is external memory, PE is the processing elements, LM is the local memory, and RN is the router network.*

PEs of the CGRA are classified by Computing PE (cPE) and Memory PE (mPE). The description of PR and PE consist of computing features, when the description of MEM and LM consist of storage features of CGRA.

3.4 Mapping PIA-CDTG on CGRA

Given an application, mapping it onto a CGRA is kind of looking for bindings of the node set N= (T, M) in PIA-CDTG and node set pair {(PR, PE), (MEM, LM)} in CRAG. Fig.2 (d) shows the mapping results. As shown in Fig. 2(c) the bindings must fit: 1.The application function constrains; 2.data dependences; 3.hardware constraints.

Definition 3 *Given a node V= (T/M, P) from PIA-CDTG, and the CRAG of CGRA R, find a mapping: V-> R, with the object function of minUoR and maxThO, under the meeting of resource constraints, such as computation/storage resource.*

If the node is task node T, the mapping called computing mapping, otherwise called storage mapping if the node is storage node M. The object function is showed in Eq. (1):

$$ObjFunction(V) = \begin{cases} \min UoR, & \text{if } DCP \geq 1 \\ \max ThO, & \text{if } DCP < 1 \end{cases} \quad (1)$$

Where *UoR* is the occupation ratio of PEs, *ThO* is the throughput of loop execution, and the *DCP* is the ratio of storage time T_D to computation time T_C of execution. When $DCP \geqslant 1$, we define the loop is memory-bounded, while the resource is sufficient, we prefer to optimize the objective *minUoR*.

In CRAG with n*m PEs and d LMs, if the task V needs i cPEs, j mPEs and s DMs, V consist of Num_V computation/storage operations and its execution time is $T_V=T_C+T_D$. The *UoR* and *ThO* of V are defined in Eq. (2):

$$\begin{cases} UoR_V = \dfrac{i+j+s}{m*n+dmn} \\ ThO_V = \dfrac{Num_V}{T_V} \end{cases} \quad (2)$$

3.5 Storage Mapping of PIA-CDTG

The object of storage mapping is the mapping strategy of M node of PIA-CDTG with minimum cost onto the CRAG. There are two kinds of storage node in CRAG, MEM stands for external memory, and LM stands for the local memory of PEs array. With program features analysis, the execution time of each task node is used to decide whether the current task is critical task or not. If an M node is connected to a non-critical task node, it will be marked for MEM node. Those M nodes which marked as LM will be gathered as the LM node set for succeeding computing mapping. If a M node's storage request is large than the LM size, which means the task nodes need to be tiled into sub-tasks in order to minimum the data communication.

3.6 Computing Mapping of PIA-CDTG

As the input, PIA-CDFG provides task nodes for computing mapping. Once these critical task nodes selected, we can generate a Dependency-Aided Graph (DAG). The nodes in DAG are critical tasks, and the edges denote the data dependences among these nodes. Fig. 5 shows a DAG includes 15 nodes. Before the mapping, the task nodes have been tiled for parallel execution with least data dependences [11]. Since the nested loop for accelerating are transformed into many tiled task with dependences, the parallel processing is now a scheduling for these tasks.

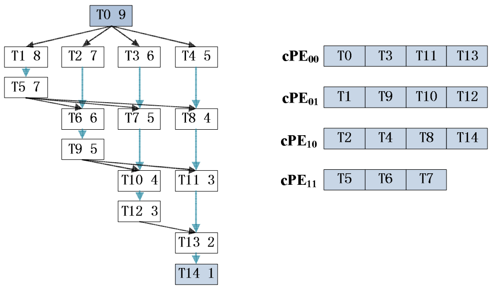


Fig. 3. DAG of LU decomposition and scheduled on a 2*2 cPEs array

We need to identify six steps as preliminary to the computing scheduling.

Step 1. Each node in DAG=<T, E> is numbered from the top down, and then the set of tasks is T= {T_0, T_1, … T_K}, the set of dependency edges is E. Edge e (Tx->Tu) denotes the dependence of Tx and Tu.

Step 2. Get each task Tx in the DAG, with its predecessor set PreTx, and its successor set SucTx.

Step 3. If the PreTx of Tx is NULL, Tx is called input node; if the SucTx of Tx is NULL, Tx is output node. The level of each node is the number of nodes in the path from itself to the output node. Apparently the level of input node is 1, and there is no dependence between the nodes with same level.

Step 4. The n*m PEs set P={PEij|i∈[0,n-1],j∈[0,m-1] }, the distance between PEij and PEst is |i-s|+|j-t|, the communication between PEs in direct proportion to their distance.

Step 5. For a node Tx, the execution time begins at STx, ends at ETx, as shown in Eq. (3). Where Tx is mapped to cPEij, and its father node Tp is mapped to cPEst. ω is constant, as the maximum of communication cost of two PEs in theory. The execution time of Tx is t.

$$\begin{cases} ST_x = \underset{T_p \in \mathrm{PreT}_x}{MAX} \{t + ST_p + \dfrac{\omega}{|n|+|m|} \times (|i-s|+|j-t|)\} \\ ET_x = t + ST_p \end{cases} \tag{3}$$

Step 6. Each element Q[i, j] in the n*m matrix Q[n,m] contrains a pair of number (x, q), which means the node Tx is mapped on the qth placement of cPEij's FIFO.

3.7 Task Scheduling

Task scheduling should consider two objectives: (1) to maximum the parallel level of the execution of the tasks; (2) to minimum the inter-task communication cost. However, these two objectives are mutually exclusive. In this case, our solution is to use inheritance algorithm to achieve a balance between these two objectives. For details, we determine the priority sequence of tiles by iterations on the candidate solutions. In each of the iterations, we first search for the entire problem space with a generic element, and then evaluate these candidate solutions with an adaptive function. This solution tries to find a scheduling, which first assigns the *k* tasks to the n*m processors, and then properly schedules the tasks within the queue of each processor to achieve the shortest execution time while the dependencies among tasks are fully respected. For details, for a assignment and schedule policy *S*, we aim to find a policy s to minimum the *ET*(*s*) that $\mathrm{ET}(s)=\max_{T_x \in T} \mathrm{ET}_x$ .Fig. 5 shows the dependency DAG with task numbers and level values. The detailed inheritance algorithm responsible for scheduling tasks is shown in Table 2.

Table 2. Task Scheduling Algorithm

```
Algorithm 1 Task Scheduling Algorithm
1 :  Procedure TSA
2 :  Input: DAG=<T,E> of tiled loops
3 :  Output:  Optimal solution with $ObjFunction(V)$
4 :  for each node T in DAG, compute T.level
5 :      if $T_i$.level= $T_j$.level, then $T_i$ and $T_j$ should assigned to different processors, where
     the tasks are sorted in descending order according to the level value of the task nodes.
6 :      if the requirement of the size of the initial population is satisfied, then break;
7 :  end for
8 :  Divide the initial population into several sub- population with the same size
9 :  for each policy unit, compute the adaptive function $F(s)=\sum_0^k ET_x - ET(s)$
10 :     if the iteration count reaches the maximum iteration number or the optimal solu-
     tion has not evolved more than 3% in the latest 10 generations, end procedure
11 :     end if
12 :        for each sub-population, select individuals with deploy the roulette wheel se-
     lection scheme with probability $F(i)/\sum_0^{SN} F(i)$ ,where SN is the size of population
13 :        if $T_i$.level= $T_j$.level && $T_i$ and $T_j$ from different processors
14 :        then take the selected individual $x$, perform uniform crossover on the two
     nodes
15 :          Assign the two nodes to the sub-task queues of two randomly selected $x$'s pro-
     cessors, assuring that the level values of task nodes in each queues are still in ascending
     order;
16 :     end if
17 :     Compute the level of these node, then move the same level nodes from the longest
     sub-task queue to the shortest sub-task queue;
18 :     compute the adaptive function for the individuals in each sub-population, and
     replace the low adaptability individual with high adaptability individual;
19 :        end for
20 :     end for
21 :     end procedure
```

4 Experiments

We evaluated three application kernels to do the experiments, including the discrete cosine transformation (DCT), matrix LU decomposition (LU), and matrix vector transpose (MVT). We did a task scheduling with a genetic algorithm to find the optimized strategy under the object functions. We use a reconfigurable SoC for experimental evaluation, which consists mainly of a 32-bit RISC microprocessor called EstarIII [12] and reconfigurable arrays called LEAP. We use EstarIII for common computing. It has 8KB instruction cache and 8KB data cache, 266M Hz and 220mW of CPU core. Meanwhile, we take LEAP PEs arrays as the accelerating coprocessor, whose total LMs size is 16 KB, and each size of LM is 4KB. We analyzed the features of loop in the three kernels, as shown in Table 3, in which the storage demand is count by the millisecond.

Table 3. Features of critical loops

Kernel	DCT(M=2000)		MVT(N=10000)		LU(N=2000)
Name	main.loop1	main.loop2	main.loop1	main.loop2	main.loop1
Type	for	for	for	for	for
Iteration Condition	i<M,j<M,k<M	i<M,j<M,k<M	i<N,j<N	i<N,j<N	i<N-k-1, j<N-k-1,k<N
Nested Level	3	3	2	2	3

Related Arrays	temp2d(0.43)
	block(1.02)
	cos1(1.02)
		sum(0.36)
		cos1(0.90)
		temp2d(0.90)
		block(0.36)
			x1(1.0)
			a(0.80)
			y_1(0.80)
				x1(1.0)
				a(0.80)
				y_1(0.80)
					a(1)

Index Relation	temp2d:(x,y)=(i,j)
	block:(x,y)=(i,k)
	cos1:(x,y)=(j,k)
		sum:(x,y)=(i,j)
		cos1:(x,y)=(j,k)
		temp2d:(x.y)=(k,j)
		block:(x,y)=(i,j)
			x1:(x)=(i)
			a:(x,y)=(i,j)
			y_1:(x)=(j)
				x2:(x)=(i)
				a:(x,y)=(i,j)
				y_2:(x)=(j)
					a:(x.y)=(k,j)
					a:(x.y)=(k,k)
					a:(x.y)=(i,j)
					a:(x.y)=(i,k)

Storage Demand	536,328bit		43,852bit		278,938bit

At first we evaluated the three kernels under several different conditions. We test four kinds of approaches, 1) paralleling the sequence program without optimization (baseline), 2) the PDG approach with sequence scheduling (PT), 3) multi-level DFG approach with sequence scheduling (DT), 4) multi-level PIA-CDTG with scheduling (PCT). Where the maximum generation of the generic algorithm is 1000, crossover is0.5, and mutation is 0.1. For the comparison of speedup with different approaches, we defined different tile sizes and different number of PEs. Fig. 4 shows the approach execution speedup with different size of PE arrays (from 2*2, 3*3, to 4*4) by changing the CRAG model configuration. Because of the tile size is 16KB which is just the data memory size; the scheduling gained a not very good workload balance among the PE arrays. But while the size of PE arrays getting bigger, the speedup ratio of PCT to PT are increasing much faster, that means the generic algorithm could find out the optimal solutions efficiently.

Fig.5 shows the affection taken by the increasing quantity of DCT. We set a 4*4 PEs arrays, and the loop tile size is 16KB. While the iteration number of these kernels increasing (from 2000 to 10000), the performance of the four approaches is getting dropping sharply. However, the declines of PCT and DT are much slowly than the PT and baseline. Therefore, the data access and communication of application nonlinear increase with the size of iteration number is the main bottleneck of parallelization. That's why we focus on the memory hierarchy to make the loop multi-level tiled and use the memory-aware object function for the genetic scheduling algorithm. Since the scheduling based on static analysis, we should take more dynamic analysis based on the architecture performance model as the future work.

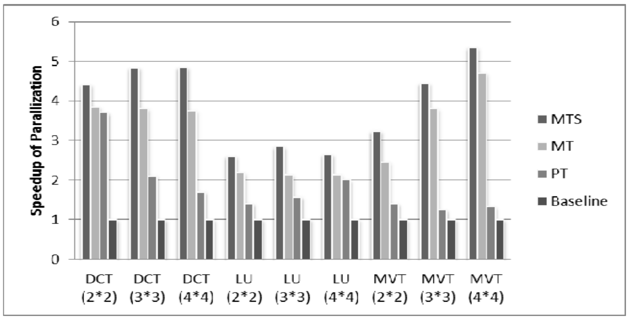


Fig. 4. Speedup comparison with different PE arrays

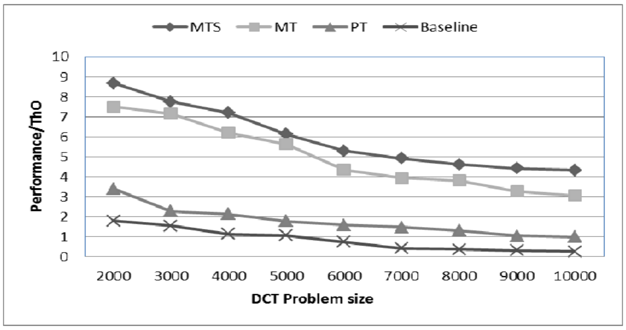


Fig. 5. Throughput comparison with different iteration size (DCT)

5 Conclusion

In this paper we proposed a novel graph model called PIA-CDTG for mapping da-ta-intensive applications on CGRA. With application features analysis, we can get all the information needed to generate the PIA-CDTG. By dividing application into task nodes and storage nodes, PIA-CDTG makes it an easier way to map. And then we used a generic algorithm with optimized object function to analyze depen-dences be-tween the task nodes. The experimental results shown the approach was efficient for data-intensive application acceleration on CGRA. The next challenges include developing a code generator for automatic compilation of loop nests and efficient power estimation models for the goal of an architecture/compiler co-exploration.

References

[1] Cardoso, J.M.P., Diniz, P.C., Weinhardt, M.: Compiling for reconfigurable computing: A Survey. ACM Computing Surveys 42(4), 1–65 (2010)

[2] Najjar, W., Bohm, W., Draper, B., Hammes, J., Rinker, R., Beveridge, J., Chawathe, M., Ross, C.: High-level language abstraction for reconfigurable computing. Computer 36(8), 63–69 (2003)

[3] Dou, Y., Wu, G., Xu, J., Zhou, X.: A coarse-grained reconfigurable computing architecture with loop self-pipelining. Science in China 38(4), 579–591 (2008)

[4] Rinker, R., Carter, M., Patel, A., Chawathe, M., Ross, C., Hammes, J., Najjar, W., Bohm, W.: An automated process for compiling dataflow graphs into reconfigurable hardware. IEEE Transactions on Very Large Scale Integration (VLSI) Systems 9(1), 130–139 (2001)

[5] Gupta, S., Dutt, N., Gupta, R., Nicolau, A.: Loop shifting and compaction for the high-level synthesis of designs with complex control flow. In: Proceedings of Design, Automation and Test in Europe Conference and Exhibition (DATE 2004), vol. 1, pp. 114–119. IEEE Computer Society (2004)

[6] Weinhardt, M., Luk, W.: Pipeline vectorization. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 20(2), 234–248 (2002)

[7] Park, H., Fan, K., Kudlur, M., Mahlke, S.: Modulo graph embedding: mapping applications onto coarse-grained reconfigurable architectures. In: Proceedings of the 2006 International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES 2006), pp. 1–11. ACM (2006)

[8] Gong, W.: Synthesizing sequential programs onto reconfigurable computing systems. PhD Thesis, University of California, Santa Barbara (2007)

[9] LooPo, Loop parallelization in the polytope model, `http://www.fmi.uni-passau.de/`

[10] Stanford University Intermediate Format Group. SUIF Compiler System Version 2, `http://suif.stanford.edu`

[11] Zhao, P., Yan, M., Li, S.: Performance Optimization of Application Algorithms for Heterogeneous Multi-Processor System-on-Chips. Journal of Software 22(7), 1475–1487 (2011)

[12] Yan, M., Shen, J., Zhao, P., Liu, L., Li, S.: Design and Implementation of an Embedded Visual Media Process SoC. Journal of Electronics 39(2), 249–254 (2011)

Memristor Working Condition Analysis Based on SPICE Model

Zhuo Bi[1], Ying Zhang[1], and Yunchuan Xu[2]

[1] School of Mechatronic Engineering and Automation,
Shanghai University, Shanghai, China
{zhuo.bi,yy_zhang}@shu.edu.cn
[2] Microelectronics R&D Center, Shanghai University Shanghai, China
xycxyc1989@foxmail.com

Abstract. Memristors are novel devices behaving like nonlinear resistors with memory. The concept was first proposed and described by Leon Chua in 1971. In 2008, HP lab proved its existence by announcing its first physical implementation as crossbar structures. A memristor has shown many advantages such as non-volatility and no leakage current. The logic value can be measured in terms of impedance and storing logic values without power consumption, which may cause significant effect on digital circuits. A detailed working condition of a nonlinear dopant drift model of a memristor is studied and a set of precise working condition has been found. The transition time between off and on states of a memristor is proposed as a kind of measurement of the switching behavior.

Keywords: memristor, digital circuits, SPICE simulation, working condition analysis.

1 Introduction

Memristor is considered to be the forth basic component in addition to resistors, capacitors and inductors. In 1971, Leon Chua [1] noticed the missing link of flux and charge (Fig.1) and predicted the existence of the forth device. He derived a set of formula describing the element and called it memristor. Instead of characterized by voltage and current, the memristor is characterized by the relationship between the charge and the flux-linkage [1].The resistance of a memristor is variable, depending upon the charge passing through the device.

In 2008, HP announced the first physical implementation of a working memristor in its lab [2]. The HP memristor is a crossbar structure and shown in Fig.2. Since the announcement of HP memristor, there is an increasing interest in the device and its applications. Already, the HP laboratory has several patents and papers related to memristor such as programmable logic design [5], neural networks [12] published, and proved the feasibility of the element. Besides, researchers are working on modeling a memristor required in circuit design and simulations. Memristor spice models are presented in [6], [7], [8] and [9].

A memristor has shown many advantages such as non-volatility and no leakage current. The logic values are measured in terms of impedance rather than voltages. And the feature that storing logic values without power consumption may cause significant effect on digital circuits.

The hysteretic and switch characteristic of memristor nonlinear dopant drift model is studied in this paper with SPICE model proposed in [6].The model is simulated under driven voltage source with different frequencies and amplitude.

The paper is organized as follows: Part I introduces memristor and its development. Part II is on the theory of memristor, the physical and electrical model of HP memristor. The Spice model adopted in this paper is briefly introduced in Part III and the memristor is simulated in Part IV. Finally, the experimental results are summarized in Part V.

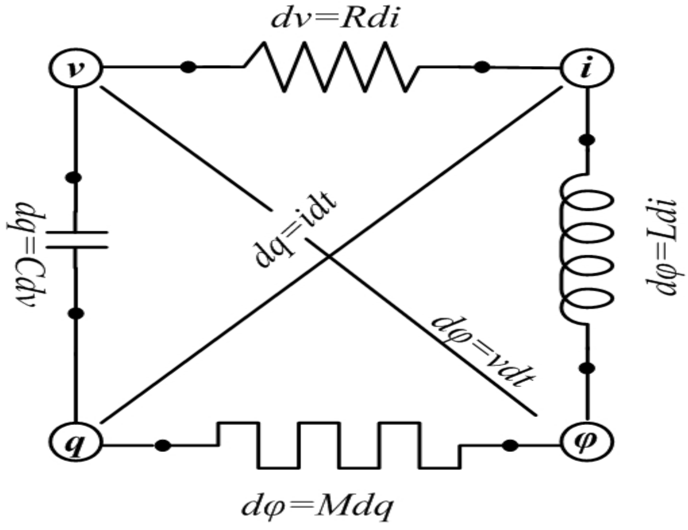


Fig. 1. The four fundamental circuit elements: resistor, capacitor, inductor and memristor

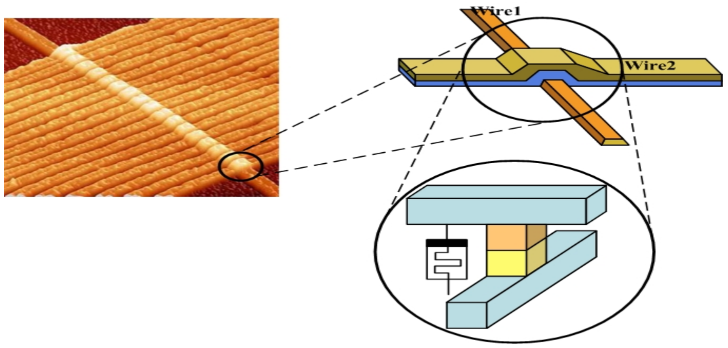


Fig. 2. HP Memristor Structures as Crossbars [3]

2 The Memristor

2.1 Theory of Memristor

A memristor is a two terminal passive element, combining the two variables of flux and charge. By assuming that flux is the function of charge, the following equation can be obtained:

$$\varphi = f(q) \tag{1}$$

By differentiating on both sides of the equation with respect to time, and using equation:

$$dq = idt \tag{2}$$

$$d\varphi = vdt \tag{3}$$

it can be derived that:

$$v = \frac{\partial f}{\partial q} \cdot \frac{dq}{dt} = \frac{\partial f}{\partial q} \cdot i = \frac{\partial \varphi}{\partial q} \cdot i \tag{4}$$

If the relationship between flux and charge is non-linear, then

$$v = M(q)i \tag{5}$$

with $M(q) = \frac{\partial \varphi}{\partial \mathrm{q}}$ a memristor.

Otherwise, a linear resistor is obtained if $\frac{\partial \varphi}{\partial \mathrm{q}}$ is a constant.

For （1）and（2）, by integrating over time, the following equation can be derived:

$$q(t) = \int_{-\infty}^{t} i(\tau)d(\tau) \tag{6}$$

$$\varphi(t) = \int_{-\infty}^{t} v(\tau)d(\tau) \tag{7}$$

By rewriting (1) with (6) and (7), the following equation can be obtained:

$$\int_{-\infty}^{t} i(\tau)d(\tau) = f(\int_{-\infty}^{t} v(\tau)d(\tau)) \tag{8}$$

which implies that the memristor is such an element whose relationship between the integrals of current and that of voltage is nonlinear.

2.2 HP Memristor

The HP Memristor is the first known fabrication of the device. It is a two-layer titanium dioxide (TiO2) cube of 40-nanometer between two crossed nanowires (Fig.2).This switch shown in Fig.3 is a voltage regulated device.

The memristance of a memristor can be given by the following equation [2]:

$$M(q) = R_{on}\frac{w(t)}{D} + R_{off}\left(1 - \frac{w(t)}{D}\right) \tag{9}$$

where D the total thickness of the two TiO2 layers, w(t) the thickness of the doped layer, w(t)/D the ratio of doped layers and the total thickness, which determines the resistance of a memristor. The resistances Ron and Roff are the limitation of a memristor when w=1 and w=0, respectively.

The speed of movement of the boundary speed between the two Tio2 layers is first given by HP as follows:

$$\frac{dw(t)}{dt} = \mu_v \frac{R_{on}}{D} i(t) \tag{10}$$

By applying a voltage pulse to the terminal of the switch, the resistance can be changed.

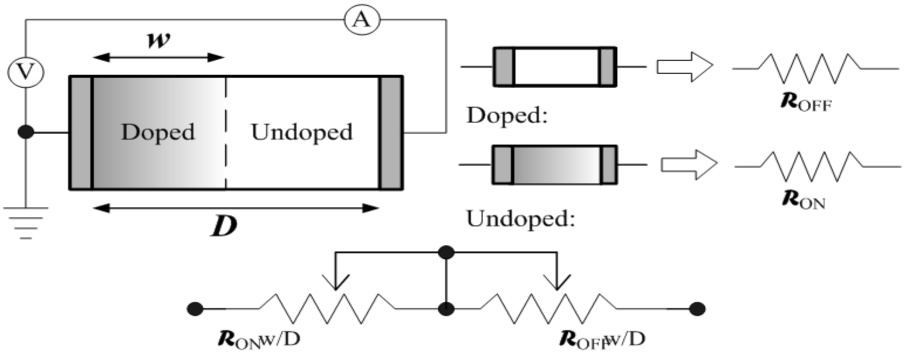


Fig. 3. The coupled variable- resistor model for a memristor [2]

2.3 Electrical Model[5]

A memristor as an ideal switch can be in either one of its two states: high impedance (open), or low impedance (closed), as shown in Fig.4.

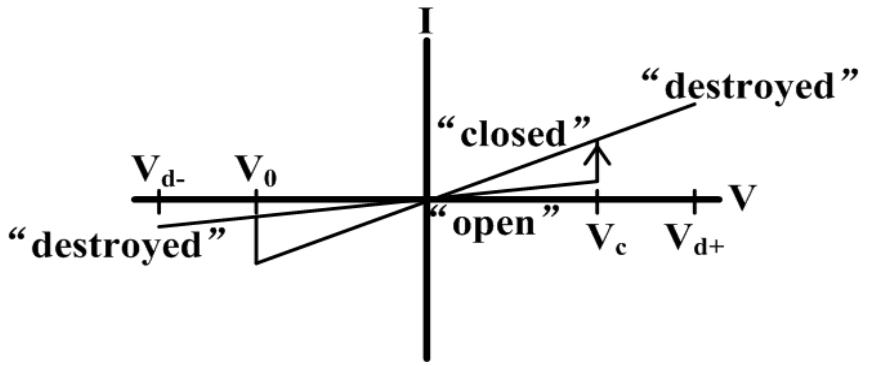


Fig. 4. Ideal Memristor switch model [5]

The element can hold its state unless the voltage drop across it exceeds the operating range. In other word, the memrister transitions from the open state to the closed one if the voltage drop exceeds Vc, while from the closed state to open if the voltage is less than Vo. Besides, an excessive positive voltage drop or negative voltage drop across a junction will destroy the device.

3 Memristor SPICE Model[6]

We experimented with SPICE model proposed in [6] to study the behavior of a single memristor. This model is the nonlinear ions drift model, and assumes a nonlinear dependence on voltage in the state variable differential equation. The model is proposed basing on the formula of (9) and (10), and its SPICE structure is shown in Fig.5. Window function, equation (11), is adopted hereby to model boundary conditions of a memristor.

$$f(x) = 1 - (2x - 1)^{2p} \tag{11}$$

where p is a positive integer.

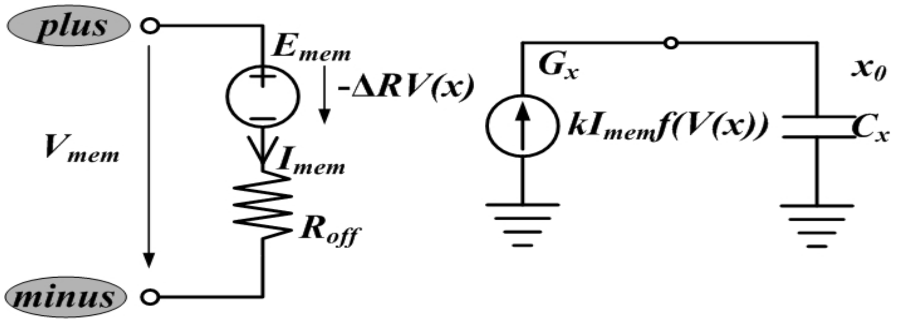


Fig. 5. Structure of the SPICE model adopted

4 Memristor Simulation

Calling the SPICE model as sub-circuit, the simulation can be done in Pspice. The simulation results are shown in the following figures. The memristor is driven by a sine wave voltage.

Fig. 6 demonstrates the typical I-V characteristic of a memristor with different Ron and Roff state. The simulation shows that memristor with high Roff resistance are with stronger hysteretic characteristics, and thus a better switch characteristic.

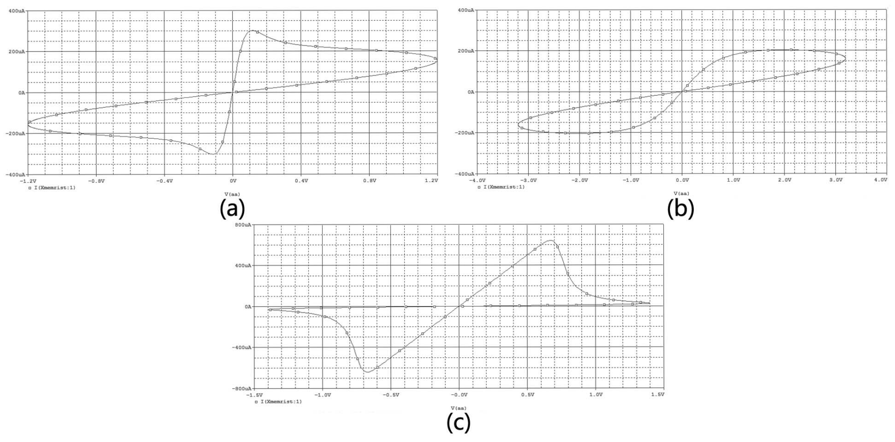


Fig. 6. Typical I-V curves of memristors, driven by sine voltages with frequency of 1Hz and amplitudes of 1.2V, 3.2V and 1.4V. The parameters are: a) Ron=100, Roff=16k, Rinit=11k, p=10 b) Ron=100, Roff=38k, Rinit=28k, p=10 c) Ron=1k, Roff=100k, Rinit=80k, p=1.The horizontal axis is the voltage applied and measured in Volts, where the longitudinal axis is current and measured by uA.

Although the memristor shows hysteretic characteristics when voltage is applied, it does not show very strong hard switching behavior (Fig.7 a, b). Switching characteristic gradually occurs and gets stronger with the increase of voltages. This could be resulted from Fig7 b, c and d. The right boundary deteriorates as voltage goes up. When voltage is up to 1.6v, the model is not able to hold switch state at right boundary (Fig.7 e). And the range of working voltages is 1.4v-1.5v (Table 1).

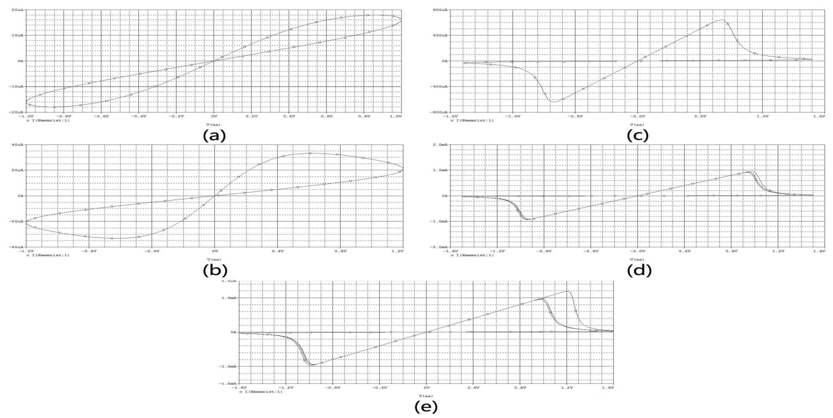


Fig. 7. I-V curves of memristor whose parameters are: Ron=1K, Roff=100k Rinit=80k p=1 f=1Hz.The amplitudes of the driven sine voltages are: a) 1.0v; b) 1.2v; c) 1.4v d) 1.5v e) 1.6v. The horizontal axis is the voltage applied and measured in Volts, where the longitudinal axis is current and measured by uA.in a), b) and c), by mA in c) and d).

Switching characteristic occurs at the voltage of 2.5v with 2Hz of voltage frequency, but not strong enough. This voltage is much higher than 1.4v compared with cases in Fig.7. The boundary begins to collapse at voltage of 4.5v (Fig.8 f).

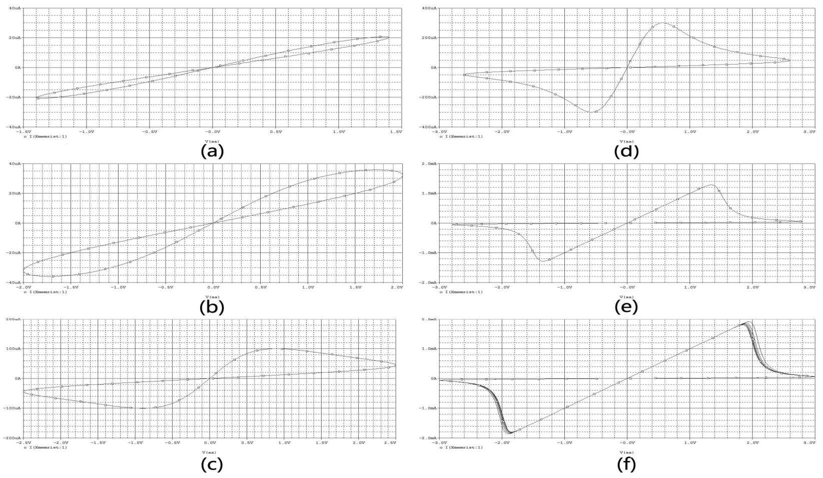


Fig. 8. IV curves of memristor whose parameters are: Ron=1K, Roff=100K Rinit=80K p=1 f=2Hz.The amplitudes of the driven sine voltages are: a) 1.4v; b) 2.0v; c) 2.5v d) 2.6v e) 2.8v f) 3.0v. The horizontal axis is the voltage applied and measured in Volts, where the longitudinal axis is current and measured by uA. in a), b) , c) and d) by mA in e) and f).

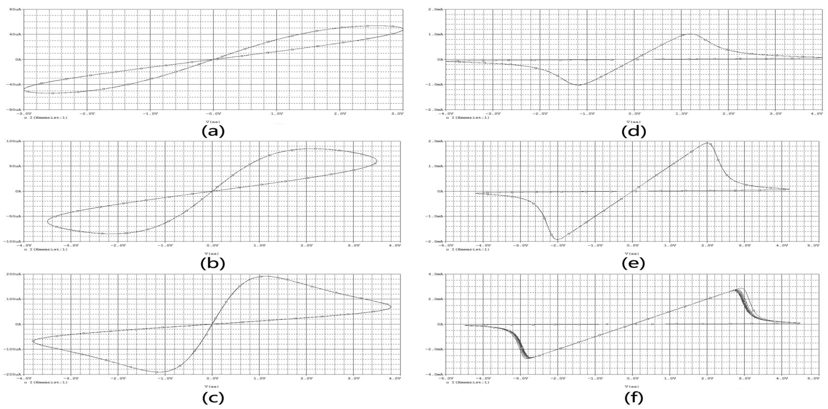


Fig. 9. I-V curves of memristor whose parameters are: Ron=1K, Roff=100K Rinit=80K p=1 f=3Hz.The amplitudes of the driven sine voltages are: a) 3v; b) 3.5v; c) 3.8v d) 4.0v e) 4.2v f) 4.5v The horizontal axis is the voltage applied and measured in Volts, where the longitudinal axis is current and measured by uA. in a), b) , c) and by mA in d), e) and f).

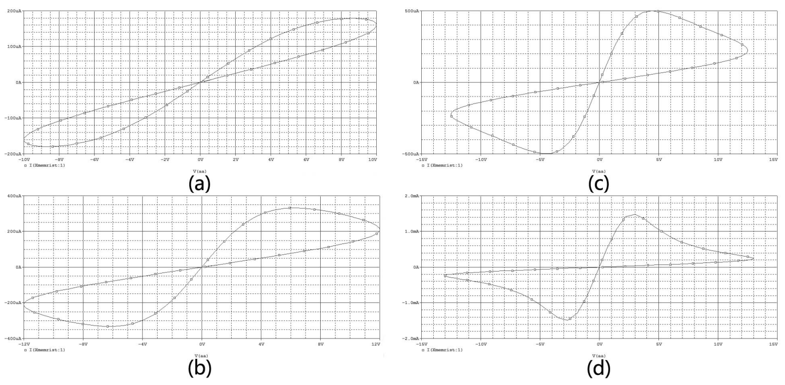


Fig. 10. I-V curves of memristor whose parameters are: Ron=1K, Roff=100K Rinit=80K p=1 f=10Hz.The amplitudes of the driven sine voltages are: a) 10v; b) 12v; c) 12.5v d) 13v. The horizontal axis is the voltage applied and measured in Volts, where the longitudinal axis is current and measured by uA.in a),b),c),and by mA in d).

Similar results can be deduced from Fig9 and Fig 10. The working voltage range of a memristor is summarized in Table 1, from which we can see that the widest working voltages occurs at the frequency of 2Hz and 3Hz.

Comparing Fig. 7, 8, 9 and 10, it can be found that the switch phenomenon occurs with higher applied voltages when frequencies are increasing. The can easily be seen from switching occurrence voltage summarized in Table 1. Thus, it may be concluded that higher voltage sources are required to maintain the switch characteristic of a memristor with the increase of frequency.

Table 1. Working Voltage at different frequences and switching occurance voltage

Frequency of sine wave	f=1Hz	f=2Hz	f=3Hz	f=10Hz
Working voltage	1.4v-1.5v	2.6v-3.0v	4.0v-4.5v	13v
Switching occurence voltage	1.4v	2.6v	4.0v	13v

Besides, it hints that the model can only work at extraordinarily low frequencies. Although the model still shows the hysteretic nature at frequency goes up to 10Hz, the amplitude of the sine wave applied has reached nearly 12V to maintain its switch characteristic, which is unsuitable for standard CMOS technology. Fig. 11 demonstrates the boundary deterioration when f=1, f=2, f=3(Hz).

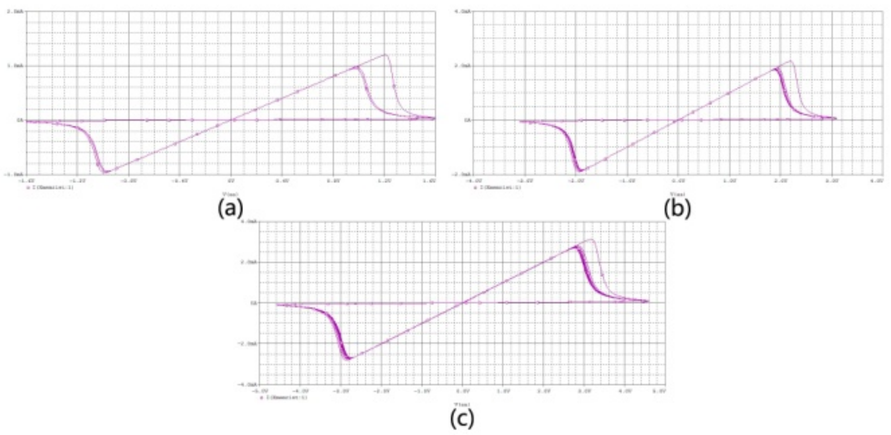


Fig. 11. Boundary deterioration when f=1Hz, f=2Hz, f=3Hz. The amplitude applied are 1.6v.3.1v.and 4.6v, respectively. The horizontal axis is the voltage applied and measured in Volts, where the longitudinal axis is current and measured by mA.

A memristor as an ideal switch will transform from one state to the other with suitable voltage applied. As an ideal switch, transition will be finished without time delay. Actually, a certain amount of time is required between the transitions of the states. Obviously, a faster switch can be obtained with shorter transition time. Thus, it can be said that the transition time between high impedance and low impedance is the one of the main measurement of memristor's switching behavior. By simulation of the non-linear drift model, it is possible that the transition time is related to conditions such as voltage applied and the frequency of driving source (see Table 2). It can be seen that 1) shorter transition time requires larger voltage; 2) higher frequency of the driving voltage source also shorten the transition time. Fig12 is the three-dimensional histogram of Table 2.

Table 2. Transition time at different voltages and frequencies

Frequency of sine wave	f=1Hz	f=2Hz	f=3Hz	f=10Hz
Voltage applied and state transition time	v=1.4v,t=0.425s	v=2.6v,t=0.230s	v=4.0v,t=0.150s	v=13v,t=0.046s
	v=1.5v,t=0.375s	v=2.7v,t=0.220s	v=4.1v,t=0.145s	
		v=2.8v,t=0.210s	v=4.2v,t=0.140s	
		v=2.9v,t=0.200s	v=4.3v,t=0.135s	
		v=3.0v,t=0.195s	v=4.4v,t=0.132s	
			v=4.5v,t=0.128s	

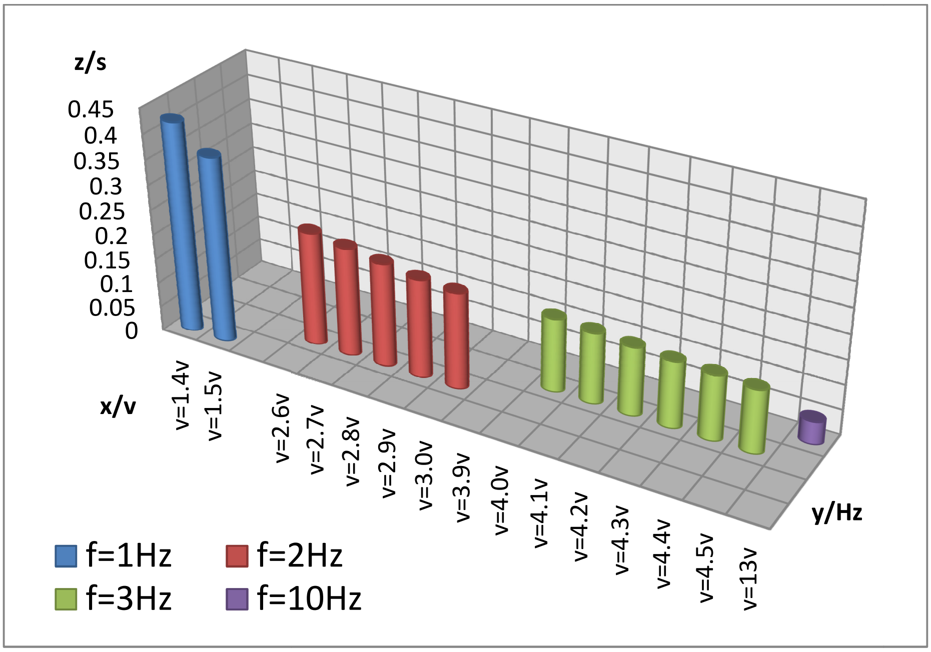


Fig. 12. Three-dimensional histogram of transition time at different frequencies and voltages: horizontal axis is the voltages applied at the memristor; the longitudinal axis is the frequency of the applied sine wave; the vertical is the stage transition time from high to low (transition time from low to high is exactly the same because the hysteretic I-V curve is centrally symmetric).

Table 3. Effection of parameter p in window function

Frequency Parameter P	f=1Hz	f=2Hz	f=3Hz

p=1	v=1.5v,t=0.470s
	v=1.6v,t=0.425s
		v=3.0v,t=0.237s
		v=3.1v,t=0.225s
		v=3.2v,t=0.213s
			v=4.5v,t=0.155s
			v=4.6v,t=0.152s
			v=4.7v,t=0.147s
			v=4.8v,t=0.142s
			v=4.9v,t=0.138s
			v=5.0v,t=0.135s

p=2	v=1.3v,t=0.475s
	v=1.4v,t=0.415s
		v=2.6v,t=0.238s
		v=2.7v,t=0.220s
		v=2.8v,t=0.208s
			v=3.9v,t=0.158s
			v=4.0v,t=0.150s
			v=4.1v,t=0.145s
			v=4.2v,t=0.138s

p=3	v=1.3v,t=0.438s	v=2.5v, t=0.238s
		v=2.6v, t=0.220s
			v=3.8v,t=0.155s
			v=3.9v,t=0.146s

p=4		v=2.5v, t=0.230s	v=3.7v,t=0.158s
p=5			v=3.7v.t=0.150s

Table 3 shows the effect of the parameter p in window function. The nonlinear dopant drift means that the speed of boundary between the doped and undoped regions gradually gets to zero. This is modeled by the window function described by equation (5). With the increase of p, equation (5) gradually becomes similar to a rectangular window, and thus non-linear characteristic disappeared. The memristor doesn't work with all p values, but just at some point. It also hints that transition time is getting shorter with the increase of p, if voltage and its frequency are fixed.

5 Summary

A nonlinear ions drift memristor SPICE model is simulated in this paper and a set of precise working condition of the device has been found. The switching characteristic of a nonlinear drift model is non-robust, and might be easily affected by factors such as external voltages and frequencies or internal p parameters. For the non-linear drift model, it is clear that higher external voltage is required to maintain the switch behavior as frequency goes up. To overcome this non-robustness for further study, one way is to renew the models, and another is to find equations better for describing the element.

References

1. Chua, L.O.: Memristor- the missing circuit element. IEEE Transitions on Circuit Theory CT-18(5), 507–519 (1971)
2. Strukov, D.B., Snider, G.S., Stewart, D.R., et al.: The missing memristor found. Nature 453, 80–83 (2008)
3. Williams, S.: How we found the missing memristor. IEEE Spectrum 45(12), 28–35 (2008)
4. Snider, G.S., Kuekes, P.J.: Nano State Machines Using Hysteretic Resistors and Diode Crossbars. IEEE Transactions on Nanotechnology 5(2), 129–137 (2006)
5. Snider, G.S.: Computing with hysteretic resistor crossbars. Applied Physics A 80, 1165–1172 (2005)
6. Biolek, Z., Biolek, D., Biolkova, V.: SPICE model of memristor with nonlinear dopant drift. Radioengineering 18(2), 210–214 (2009)
7. Batas, D., Fiedler, H.: A memristor SPICE implementation and a new approach for magnetic flux-controlled memristor modeling. IEEE Transactions on Nanotechnology 10(2), 250–255 (2011)
8. Rak, A., Cserey, G.: Macromodeling of the Memristor in SPICE. IEEE Transitions on Computer-aided Design of Intergated Circuits and Systems 29(4), 632–636 (2010)
9. Borghetti, J., Snider, G.S., Kuekes, P.J., et al.: 'Memristive' switches enable 'stateful' logic operations via material implication. Nature 464, 873–876 (2010)
10. Raja, T., Mourad, S.: Digital logical implementation in memristor-based crossbars a tutorial. In: 2010 Fifth IEEE International Symposium on Electronic Design, Test & Applications, pp. 303–309 (2010)
11. Snider, G.S.: Molecular-junction-nanowire-crossbar-based neural network. US Patent No.7359888 B2 (April 15, 2008)

On Stepsize of Fast Subspace Tracking Methods

Zhu Cheng[1,2], Zhan Wang[1], Haitao Liu[1], and Majid Ahmadi[2]

[1] School of Electron. Sci. & Eng.,
Nat. Univ. of Defense Technol.,
Changsha, China
Chengzhu88@gmail.com
[2] Department of ECE,
University of Windsor, Windsor,
Canada ON
Ahmadi@uwindsor.ca

Abstract. Adjusting stepsize between convergence rate and steady state error level or stability is a problem in some subspace tracking schemes. Methods in DPM or Oja class may sometimes show sparks in their steady state error, even with a rather small stepsize. By a study on the schemes' updating routine, it is found that the update does not happen to all of basis vectors but to a specific vector, if a proper basis is chosen to describe the estimated subspace. The vector moves only in a plane which is defined by the new input and pervious estimation. Through analyzing the vectors relationship in that plane, the movement of that vector is constricted to a reasonable range as an amendment on the algorithms to fix the sparks problem. The simulation confirms it eliminates the sparks.

Keywords: Array signal processing, subspace tracking, stepsize, convergence.

1 Introduction

Tracking a subspace is Estimating a projection matrix onto a space or a basis for that space, from a random vector sequence observed by a sensor array. It is a powerful tool in some signal processing fields such as: telecommunication, radar, sonar and navigation, serving as a measure of adaptive filter, DOA estimation, or interference mitigation. Subspace tracking methodology can be classified into two categories: the first is estimating the space where the signal is generated from, the second one is seeking the orthogonal complement of that space. The former is known as a principal subspace (PS, PSA) tracker or signal subspace tracker, the later is often referred to as minor subspace (MS, MSA) tracker or a noise subspace tracker. For earlier works on MUSIC, we are used to the term signal or noise subspace track.

N.L.Owsley developed the first algorithm for subspace tracking in [1]. Assuming the dimension of problem is N, the rank of the subspace we are interested in is L. Usually L << N. Complex of his solution proportions to N^2L, or namely $O(N^2L)$. Many schemes with less compute complex were developed after then. An excellent survey paper [20] outlined almost all of achievements on this topic before 1990, which cost O(N^{2}L) or O(NL^2) operations. Algorithms with O(NL) complexity were developed after it. The new class of algorithms is called as Fast Subspace Tracking method. Surveys on fast subspace schemes are presented in [3pp30–43] or [19pp221-270].

Let $x(k)$ is an N-dim observer vector from an N-element sensor array, as (1),

$$x(k)=\sum_{i=1}^{L} a_i s_i(k)+n(k) \tag{1}$$

Where a_i is N-dim vector with unit length, independent to each other, representing the manifold of one of the arriving signals, and $s_i(k)$ is a random variable independent to each other, representing the arriving signal from different source, $n(k)$ is a N-dim i.i.d random vector representing the sensor noises. Assuming $V=\{v_1,v_2,v_3.....v_L\}$ is an orthonormal basis of span(a_i). The signal subspace tracking is seeking a basis $W(k)=\{w_1(k),w_2(k),.....w_L(k)\}$ which spans a subspace same as the space spanned by V; the noise subspace tracking is searching a $W(k)$ which spans a subspace as orthogonal complement of span(V). From now, we may use the name of the basis as the name of the space, such as space W or space W(k) to span(W(k)), if there is no confusion.

One of criterion on subspace problem is the distance between the spaces. Majority of solutions[2-11] use the projection error power as criterion, as (2) for signal subspace tracking or (3) for noise subspace tracking.

$$e_p=\left\|P_V-P_W\right\|_F^2 \tag{2}$$

$$e_p=\left\|P_{V\perp}-P_W\right\|_F^2 \tag{3}$$

Where $P_V=VV^H$, $P_{V\perp}=I_N-P_V$, $P_W=WW^H$;

$\left\|\bullet\right\|_F$ means Frobenius Norm

I_N means N-dim identity matrix.

An additional criterion for orthonormality of W(k) as (4):

$$\eta=\left\|W^H(k)W(k)-I_L\right\|_F^2 . \tag{4}$$

DPM[21] class algorithms is started from optimization the coordinate length of input vector's image projected onto the subspace as (5) , while Oja[5] class schemes optimize the length of the projection image as (6).

$$E(\left\|W^H x(k)\right\|^2) \tag{5}$$

$$E(\left\|x(k)-P_W x(k)\right\|^2) \tag{6}$$

This work was partly supported by National Natural Science Foundation of China under Grant(11173068))

The routines of DPM and OJA type approaches are very similar. A typical DPM scheme routine is similar to (7).

$$\begin{aligned} q(k) &= W^H(k-1)x(k) \\ y(k) &= W(k-1)q(k) \\ T(k) &= W(k-1) \pm \boldsymbol{\beta} x(k)q^H(k) \\ W(k) &= \text{orthnorm}\{T(k\} \end{aligned} \tag{7}$$

orthnorm(•) means orthonormalization operation.
Plus sign(+) stands for signal subspace tracking,
Minus sign(-) is for noise subspace tracking.

The typical routine for an Oja scheme only replaces $x(k)$ with $p(k) = x(k) - y(k)$ in the temporary general basis $T(k)$ update in (7) line 3. The variety of DPM might include FDPM, FRANS, HFRANS[6], MFPDM[7] and a version of SOOJA[8]. The branches of Oja include OOJA[9], OOJAH [9], FOOJA[10], original version of SOOJA[11]. The different among schemes in same class is orthonormalization method.

Some unreasonable random sparks were observed when we apply these schemes, especially when the noise subspace tracking was realized. It happens to DPM class schemes under noise subspace tracking and all variety of Oja methods.

In this paper, we present the geometric relationship analysis among x(k), y(k), the old last estimated space W(k-1) and the next estimated space W(k) by update equations analysis. An amendment as needed on the schemes is made to fix the sparks problem by the applying a limiter on stepsize at every update step. The presented simulation confirms the amendment.

2 Analysis of the Update Equations

Considering the subspace tracking problem, the new arriving x(k) is projected onto the last estimated space W(k-1) to get y(k) in space W(k-1). Vector t(k) is the projection image of y (k) onto x(k)'s orthogonal complement as (8). And $t \perp x$.

$$t(k) = y(k) - x(k)x^H(k)y(k) / (x^H(k)x(k)) \tag{8}$$

The relation among those vectors and the last estimated space W(k-1) shows in Fig.1 with an omission of time index k. We omit the time index k in some of following equations if there is no confusion.

Assuming there is a vector set with L elements including y direction served as a orthonormal basis set for space W(k-1), noted as $(y / \|y\|, COM)$, where COM is an L-1 dim subspace of space $W(k-1)$, and $COM \perp y(k)$. COM is the intersection of span(x,COM) and W(k-1). Both $(y / \|y\|, COM)$ and W(k-1) are orthonormal base

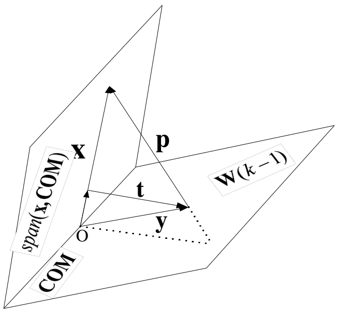


Fig. 1. Relationship of vector and space under discussion

sets for a same subspace. Therefore an L-by-L unitary matrix Q exists and meets (9,10);

$$(\mathbf{y}/\|\mathbf{y}\|, \mathbf{COM}) = W(k-1)\mathbf{Q}\,, \tag{9}$$

$$W(k-1) = (\mathbf{y}/\|\mathbf{y}\|, \mathbf{COM})\mathbf{Q}^H\,. \tag{10}$$

By left multiplication $W^H(k-1)$ to (9), it is easy to find the first row of Q is $q_0 = W^H(k-1)\frac{y}{\|y\|} = \frac{q}{\|y\|} = \frac{q}{\|q\|}$, therefore Q might be noted as $(q_0, \mathbf{Q}_{COM})$, with $q_0 \perp \mathbf{Q}_{COM}$, and $\mathbf{Q}_{COM}$ is an L row L-1 columns matrix, or L-1 number of L-dim vectors.

$$\begin{aligned} x^H COM &= x^H P_{W(k-1)} COM = x^H P^H_{W(k-1)} COM \\ &= (P_{W(k-1)} x)^H COM = y^H COM = O \end{aligned} \tag{11}$$

For (11), x is orthogonal to COM, therefore any linear combination of x and y is orthogonal to COM too, so is p.

Before the orthnorm(•) in (7), $\mathbf{T}(k)$ is a general basis of the newly estimated subspace. Right multiply the update equation of $\mathbf{T}(k)$ with $Q = (q_0, \mathbf{Q}_{COM})$. $\mathbf{T}(k)Q$ is another basis of the newly estimated subspace because $Q = (q_0, \mathbf{Q}_{COM})$ is a unitary matrix.

For Oja schemes:

$$\begin{aligned} \mathbf{T}(k)Q &= (\mathbf{w}(k-1) \pm \beta \mathbf{p}(k) q^H(k))(q_0, \mathbf{Q}_{COM}) \\ &= \mathbf{w}(k-1)(q_0, \mathbf{Q}_{COM}) \pm \beta \mathbf{p}(k) q^H(k)(q_0, \mathbf{Q}_{COM}) \\ &= (\mathbf{w}(k-1)q_0, \mathbf{w}(k-1)\mathbf{Q}_{COM}) \\ &\qquad \pm \beta(\mathbf{p}(k)q^H(k)q_0, \mathbf{p}(k)q^H(k)\mathbf{Q}_{COM}) \\ &= (y/\|y\|, COM) \pm \beta(\mathbf{p}(k)\|q(k)\|, \mathbf{0}, \mathbf{0}, ..\mathbf{0}) \end{aligned}$$

$$
\begin{aligned}
&= \left(y / \|y\| \pm \beta \mathbf{p}(k) \|q(k)\|, COM \right) \\
&\triangleq \left(h_1, COM \right)
\end{aligned} \tag{12}
$$

Or for DPM class:

$$
\mathbf{T}(k)Q = \left(y / \|y\| \pm \beta x(k) \|q(k)\|, COM \right) \triangleq \left(h_2, COM \right) \tag{13}
$$

After the multiplication, all newly estimated subspaces in Oja or DPM class schemes have a basis expressed in term of COM and a linear combinations of x and y. By comparison on newly estimated subspaces basis (12) or (13) with the predefined basis $(y/\|y\|, COM)$ for the last estimated subspaces at k-1 time, it can be found that in all update routines, COM keeps still while the basis vector updates only in the plane span(x,y) to move y to the new basis vector h1 or h2. Fig.2 shows the update happening in span(x,y) as a cross-section of Fig1.

From Fig.2, the difference between signal and noise subspace tracking method is only the move direction from y to new basis vector h is to or far from x. In signal subspace tracking it is to x, in noise subspace tracking it is far from x. In span(x,y), t is furthest vector from x in angle view, and the closest vector to x is itself. Therefore it is enough for the new vector h to move only between x and y for signal subspace tracking, outside that range cannot provide a better result than inside; for noise subspace tracking the boundary is between y and t.

We would like to control the new basis vector h in that range. The relation between h and y or x is controlled by the step size parameter Beta. It is needed to find a boundary for it.

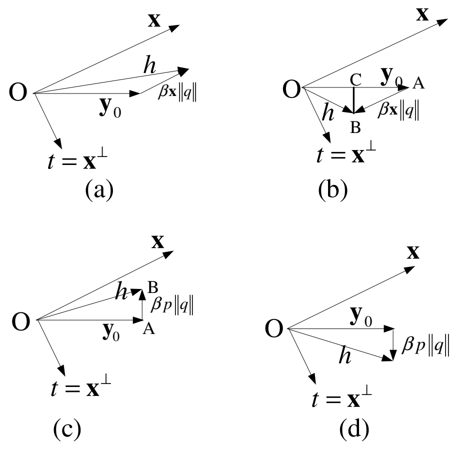


Fig. 2. The relation of new candidate replace vector for y in span(x,y)

a) DPM signal subspace b) DPM Noise subspace
c) OJA signal subspace d) OJA Noise subspace

3 Boundary of Beta

From Fig.2a, for DPM type algorithm, in signal subspace tracking, any positive value of Beta always will keep the new basis vector h between x and y, therefore any positive Beta value will not cause the problem of stability in this case.

But for DPM noise subspace tracking or all OJA classes, a fixed Beta value may move new basis vector h beyond the predefined boundary x or t at end of section 2, when the input is big enough, it might causes deviate or spark. To avoid this situation, we try to find a boundary of Beta for them.

Statement: To avoid sparks, in noise subspace tracking, condition for DPM class algorithm is $\beta \le 1/\|x\|^2$, condition for Oja class schemes is $\beta \le 1/\|p\|^2$; in signal subspace tracking, condition for OJA class algorithm is $\beta \le 1/\|y\|^2$. Or generally $\beta \le 1/\|x\|^2$ is sufficient to avoid sparks under all these situations.

Proof:

For DPM type noise subspace algorithm, with (13) and Fig2.b.

AO is vector $y/\|y\|$, therefore $|AO| = 1$

AB is vector $\beta x\|q\|$, so $|AB| = \beta\|x\|\|q\| = \beta\|x\|\|y\|$.

If the angle between x and y is θ,then

$$|BC| = \beta\|x\|\|y\|\sin\theta, |AC| = \beta\|x\|\|y\|\cos\theta,$$

therefore $|OC| = 1 - \beta\|x\|\|y\|\cos\theta$,

If the angle between y and new vector h is γ, then

$$\tan\gamma = \frac{\beta\|x\|\|y\|\sin\theta}{1-\beta\|x\|\|y\|\cos\theta}.$$

For we set new vector not over turn than the vector t in noise subspace track case, and the angle between y and t is $90^0 - \theta$, so $\gamma \le 90^0 - \theta$, and all the angles discussing here range from 0 to 90deg.

Therefore: $\tan\gamma \le \tan(90^0 - \theta)$.

It means $\tan\gamma \le \cos\theta / \sin\theta$.

$$\frac{\beta\|x\|\|y\|\sin\theta}{1-\beta\|x\|\|y\|\cos\theta} \le \frac{\cos\theta}{\sin\theta}.$$

Rearrange it we get

$$\beta \le \cos\theta / (\|x\|\|y\|) = 1/\|x\|^2 .\#$$

Therefore for FDPM type noise subspace tracking if $\beta \le 1/\|x\|^2$, then no overshoot will happen, and the sparks might cease.

Proof for OJA type scheme is very similar to this. We omit it to save space.

S.Attallah and his colleagues had found a similar boundary from a different aspect to search a more aggressive update stepsize and improve convergent rate [15-18] for some of their OJA and FRANS schemes. Unfortunately, there was no stable and fast subspace tracking scheme as FDPM or FOOJA for them to be used as a prototype at that time. From our view, aggressive stepsize is not the main target, the stability and the steady state error level of the scheme is. We are arming an amendment which can avoid the over tune phenomenon or eliminate the sparks.

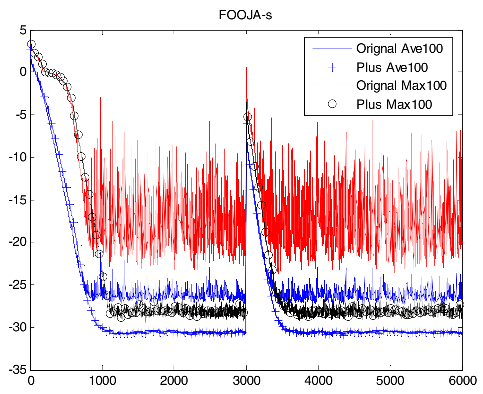
4 Amendment and Simulation

From the Statement in section 3, if we set Beta as minima of all possible $1/\|x\|^2$, the sparks might cease, but the convergence rate will be rather slow. If we apply beta as $1/\|x\|^2$ at any step, it will be too aggressive and make the steady state error rather high when the input is small.

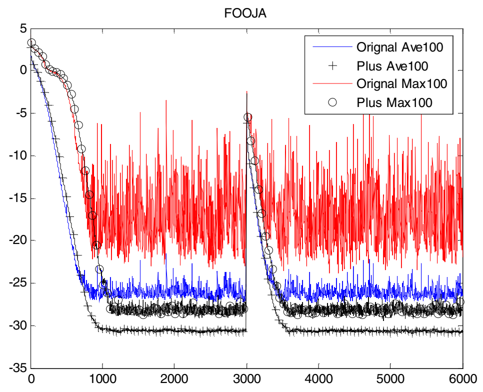
We decide to use a reasonable Beta as the original algorithms, but at any time if $\beta \le 1/\|x\|^2$, we set $\beta = 1/\|x\|^2$ for that step.

We present parts of our simulations to verify our amendment. The simulation setup is similar to [2,3], but differs in value of parameters. We consider a signal plus noise model with N =8, the random signal x(n) lies in an L=4 dimensional linear subspace, for convenience, assume manifolds of arriving signals a_i are orthogonal to each other to avoid the interaction between the arrival angle and convergence rate, arriving signals from different source $s_i(k)$ are random variables independent to each other with the powers equal to $[10,\ 1,\ 0.1,\ 0.1]$. The noise $n(k)$ is N-dim iid white and Gaussian random vector, each element with 10-3 variance. Beta=0.08 for all simulation. Duration of simulation is 6000 steps. For each single run, there is a break of the basis vector at 3000 step to introduce more projection error and destroy the orthonormality, by adding of random matrix on them, every element of it is a iid variable with 0.1 variance, to check the ability of orthonormal and projection error power convergence. Only the results of FOOJA are displayed in Fig.3, and results on FPDM or both version of SOOJA are similar to them. The results from all of schemes with amendment may follow almost a same track in simulation figures, especially in the steady state phase.

Projection error power (3) or (4) is applied for comparison in Fig.3. The y axis is in db scale. From Fig.3, average of projection error power for the original version(blue without mark) is higher than the amended one(with+), and maxima projection error power out of 100 runs for the original versions(red without mark) is much higher and noise than that of the amended ones(black, with o).



(a) FOOJA Signal Subspace



(b) FOOJA Noise Subspace

Fig. 3. Compare the Amended FOOJA and original ones

5 Conclusion

It is recommended to check relationship between the stepsize Beta and $1/\|x\|^2$ each iteration step for all mentioned schemes, if $\beta > 1/\|x\|^2$ then apply $1/\|x\|^2$ as a limiter on Beta.

References

1. Owsley, N.L.: Adaptive Data Orthogonalization. In: ICASSP 1978, pp. 109–112 (1978)
2. Doukopoulos, X.G., Moustakides, G.V.: Fast and Stable Subspace Tracking. IEEE Trans. on Signal Processing 56(4), 1452 (2008)

3. Doukopoulos, X.G.: Power techniques for blind channel estimation in wireless communication systems, Ph.D. dissertation, IRISA-INRIA, Univ. Rennes, Rennes, France (2004)
4. Attallah, S., Abed-Maraim, K.: Low-cost adaptive algorithm for noise subspace estimation. IEE Electronics Letters 38(12), 609–611 (2002)
5. Oja, E.: Principal components, minor components, and linear neural networks. Neural Networks 5, 927–935 (1992)
6. Attallah, S.: The generalized Rayleigh's quotient adaptive noise subspace algorithm: A Householder transformation-based implementation. IEEE Trans. Circuits Syst. II, Exp. Briefs 42(9), 3–7 (2006)
7. Linjie, Q., Zhu, C., et al.: Modification Algorithms for a class of subspace tracking methods. Signal Processing 26(5), 741 (2010) (Chinese)
8. Wang, R., et al.: A Novel Orthonormalization Matrix Based Fast and Stable DPM Algorithm for Principal and Minor Subspace Tracking. IEEE Trans. on Signal Processing 60(1), 466 (2012)
9. Abed-Meraim, K., Attallah, S., Chkeif, A., Hua, Y.: Orthogonal oja algorithm. IEEE Signal Processing Letters, 116–119 (2000)
10. Bartelmaos, S., Abed-Meraim, K.: Principal and minor subspace tracking: Algorithms & stability analysis. In: Proc. IEEE ICASSP 2006, Toulouse, France, vol. III, pp. 560–563 (May 2006)
11. Wang, R., et al.: Stable and Orthonormal OJA Algorithm With Low Complexity. IEEE Signal Processing Letters 18(4), 211–214 (2011)
12. Xiangyu, K., Changhua, H., Han, C.: A dual purpose principal and minor subspace gradient flow. IEEE Trans. on Signal Processing 60(1), 197 (2012)
13. Meyer, C.D.: Matrix Analysis and Applied Linear Algebra. SIAM (February 15, 2001)
14. Wang, R., Yao, M., Cheng, Z., Zou, H.: Interference cancellation in GPS receiver using noise subspace tracking algorithm. Signal Processing 91(2), 338–343 (2011)
15. Manton, J.H., Mareeks, I.Y., Attallah, S.: An analysis of the fast subspace tracking algorithm NOja. In: ICASSP, vol. 2, pp. 1101–1104 (2002)
16. Attallah, S., Abed-Meraim, K.: Fast Algorithms for Subspace Tracking. IEEE Signal Processing Letters 8(7), 203 (2001)
17. Attallah, S.: Revisiting Adaptive Signal Subspace Estimation Based on Rayleigh's Quotient. In: ICASSP, vol. 6, pp. 4009–4012 (2001)
18. Attallah, S., Abed-Meraim, K.: Subspace estimation and tracking using enhanced versions of Oja's algorithm. In: SPAWC 2001, pp. 218–220 (2001)
19. Delmas, J.P.: Ch.4 Subspace tracking for signal processing. In: Adali, T., Haykin, S. (eds.) Adaptive Signal Processing: Next Generation Solutions. Wiley Interscience (April 2010)
20. Comon, P., Golub, G.H.: Tracking a few extreme singular values and vectors in signal processing. Proceedings of the IEEE 78(8), 1327–1343
21. Yang, J.F., Kaveh, M.: Adaptive eigensubspace algorithms for direction or frequency estimation and tracking. IEEE Trans. Acoust. Speech Signal Processing 36, 241–251 (1988)

Author Index