On the Implementation of Virtual Array Using Configuration Plane[*]

Yong-Sheng Yin [1], Li Li [2], Ming-Lun Gao[1,2], Gao-Ming Du[1], and Yu-Kun Song[1]

[1] Institute of VLSI Design, Hefei University of Technology, Hefei, Anhui 230009, China
YinYongSheng@hfut.edu.cn
[2] Intitute of VLSI Design, Nanjing University, Nanjing, Jiangsu 210093, China
{LiLi,GaoMingLun}@nju.edu.cn

Abstract. A new method of designing and using virtual array in pipeline reconfigurable system is presented. This method is based on the partition of the configuration data. Using this method not only is helpful to design the virtual hardware, but also is necessary to investigate the application algorithms oriented this virtual hardware. Basing on the analysis of the space-time graph and the configuration plane, this paper explores the structure and application of virtual array integrated in the MPRS (Multi-Pipeline Reconfigurable System), an in-house developed reconfigurable computing system that utilizes virtual pipeline. Finally, the design procedure of mapping the application to the virtual array and the programming procedure of using the MPRS are illustrated by examples. The experiment results show that the method is feasible and the performance of the MPRS with the virtual array nearly reaches the expected level.

1 Introduction

The fixed size of reconfigurable resource restricts the computing capability of reconfigurable system, which is one of the most important problems in reconfigurable computing. Based on this fact the concept of virtual hardware [1,2,3,4] have been presented, which means satisfying infinite resource requirement of algorithms by time division of finite hardware resource. Paper [1] surveyed a collection of important projects in this field. The virtualization of hardware is one of the basis objectives of studying dynamic reconfiguration.

In fact, similar restriction also exists in systolic array. The fact that one array can only be used to solve the applications under certain fixed size limits the application range of systolic array. Therefore some methods [5] including emulation method, partition method, LPGS (Local Parallel, Global Sequential) method, and LSGP (Local Sequential, Global Parallel) method have been proposed to resolve this problem.

[*] This work is supported by the National Natural Science Foundation of China under grant No. 90307011, 60373076.

The traditional LPGS/LSGP methods have been used as references when implement virtual hardware in reconfigurable systems, though some key processes must be changed to adapt the reconfigurable factors. Several projects show that the virtualization of hardware can be implemented by introducing *incremental reconfigurable* into the compute pipeline, designing buffers for intermediate data and creating corresponding control mechanism [4, 6]. In despite of some papers mentioned that the systems supporting virtual hardware have been completed, but most of them focused on describing the corresponding changes on the hardware and few of them explained what preparations of the target applications should be made for the virtualization of hardware, which is the key process when execute a algorithm using virtual arrays [3, 7].

On the other hand, there are some realistic difficulties existing in other projects to support the proposed method of designing virtual hardware. For instance, RaPiD implements large scale applications by storing multiple configuration data in local memory and then cyclically processing the compute data within the processing elements. This method makes the control logic in each processing unit too complex. More importantly, the data propagating between the neighbor units loses the inherent systolic rhythm because of the repeatedly unit-inside processing. All of these make the design of virtual array more difficult.

2 MPRS Architecture

We have implemented the MPRS that incorporates multiple 1-D arrays as coprocessor with a main processor. MPRS supports virtual array and the multiple 1-D arrays can optionally work in chained mode or parallel mode to explore the loop-level parallelism.

The structure of MPRS reconfigurable arrays is shown in Fig.1. The torus chain and the hierarchy buses are used as the interconnection backbone: torus chain connects the arrays and the buses connect the arrays with the storages. The first-level buses connect the main memory with the inner buffer. The second-level buses consist of the intra-array buses and the inter-array buses. The intra-array buses are used to connect the units in the same array with the buffer corresponding to that array, and the inter-array buses are used to connect the units in different arrays with buffer corresponding to that unit. In short, the hierarchy buses transport the input/output operation data, the reconfiguration data and the intermediate results.

In Fig. 1, the shadowed rPUs belong to one single linear array. S_FIFO represents the buffer used to store the intermediate result; D_FIFO represents the buffer used to store the operation data; C_FIFO represents the buffer used to store the reconfiguration data. The inter-array input buses make each unit have the capability of inputting data, but only the last unit of each array has the capability of outputting data.

Interconnecting neighbor rPUs (dashed arrows in Fig. 1) that belong to different arrays enhances the generality of MPRS to wider field of applications. In this way, the interconnection of MPRS extends from the torus chain to the torus mesh, which can utilize those mature algorithms based on 2-dimension mesh.The detail description about MPRS architecture and mapping method can be referred to the paper [8].

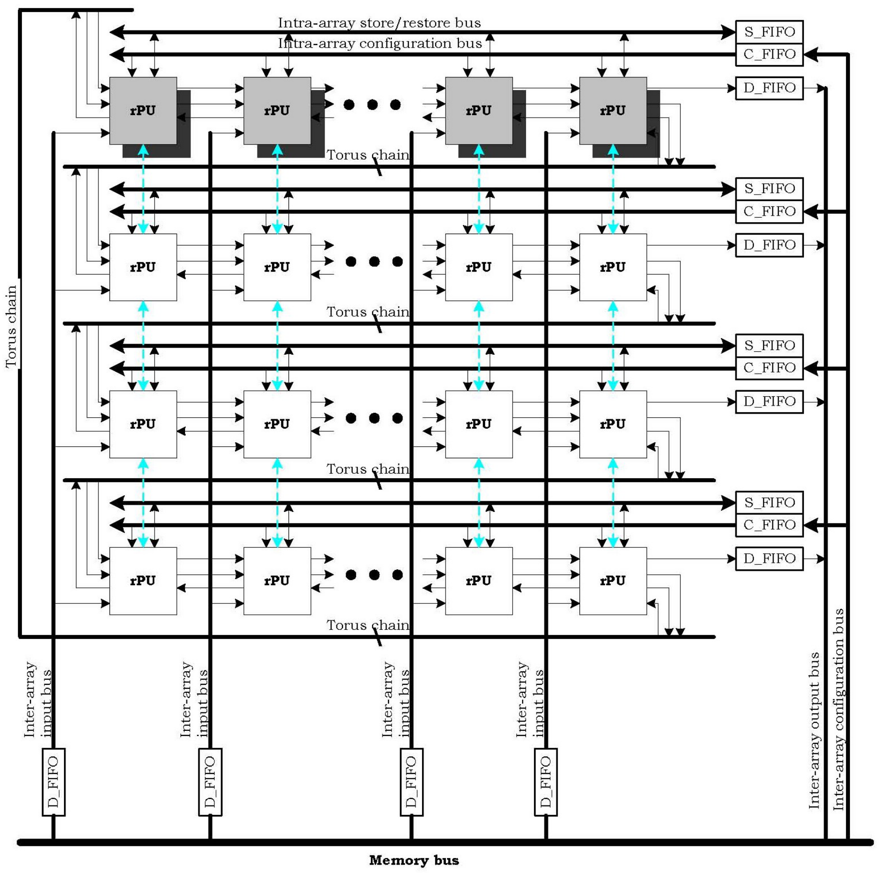
Intra-array store/restore bus
Intra-array configuration bus
rPU
Torus chain
S_FIFO
C_FIFO
D_FIFO
Inter-array input bus
Inter-array output bus
Inter-array configuration bus
Memory bus

Fig. 1. MPRS architecture

3 Designing Virtual Array Using Configuration Plane

In the reconfigurable system utilizing virtual pipeline, the intermediate data between two sequent reconfiguring operations must be stored/restored in the right time, and the external data (including the configuration data and the computing data) for the pipeline must be arranged in correct order. In other word, we must predefine the organization and timing of these data.

The requirements on the various sequences of the input/output data should be met to compute various algorithms using pipeline array, and the same thing should happen to compute one same algorithm using different pipeline arrays. In fact, these requirements are decided by the specific configuring and executing of the physical array. The distinction of the data sequences is caused by two reasons: one is the data flow direction; the other is the data flow speed. The data flow direction can be transformed to fit MPRS anyway, so the data flow speed becomes the main factor need considering when design virtual arrays.

The data flow speed is corresponding with the number of pipeline registers in the rPU (reconfigurable Processing Unit). It is difficult to decide the data sequence when there are two or more registers in the Rpu's data path. The structures of MPRS array when compute matrix-vector multiplication and 1-D Convolution are shown in Fig.2 respectively. Only one pipeline register is used in the rPU of the former, while three registers (two is in the output data path and one is in the input data path) is used in the rPU of the latter. The design of virtual arrays for the latter will be more difficult than that for the former, because it is more difficult to figure out the organization and timing of input/output data, even more difficult to decide the time of storing/restoring the intermediate data.

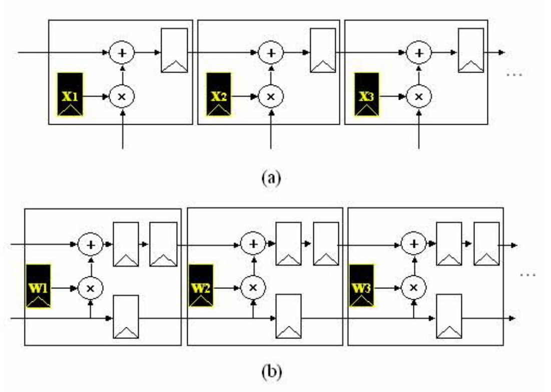

Fig. 2. Different data flow speed according to the different algorithms implemented on MPRS (a) matrix-vector multiplication; (b) 1-D convolution

The concept of "*configuration plane*" is proposed in this paper for those reasons mentioned above, and the corresponding design method based on the partition of the configuration plane is presented also. Using this method we can describe the configuration and execution of reconfigurable arrays directly, which make it easier to analyze the influence of the data flow speed on the virtual array, and to define the organization/timing of external data, and to decide the exact time of storing/restoring intermediate data. Only after all these key problems have been considered, can we design hardware structure correctly.

Suppose that an n-stage virtual pipeline is realized with an actual array including m-stage rPUs, as shown in Fig. 3. The whole array on the top of Fig. 3 is the virtual array: the real line shows the actual array, and the dashed line shows the virtual array simulated with the actual array. The space-time plane of the virtual array on the bottom of Fig. 3 shows the data operation of each rPU in each time step.

We partition these operations into several groups marked by a set of horizontal parallel lines in the time axis. The interval of these horizontal lines indicates the basic time step of the computing pipeline, and is called as "*virtual time slice*". All the operations in one same time slice should be done in one same time step if the actual array is large enough. However, if that actual array is smaller than the virtual array, those operations must be processed in the different partitioned time steps. We hope make such partition clear. According to the maximal number *k* of the pipeline registers in one rPU, a set of directed biases whose slope is *k* can be drawn. All the operations along one same bias can be processed in the pipelined sequence, and then the final results can be achieved. We call this directed bias "*computing slice*". All the operations can be partitioned into many groups marked by the parallelograms along the computing slice, and those operations contained in one same parallelogram are the all operations needed when configure/execute the whole actual array one time. We call the parallelogram "*configuration plane*". It should be noted that only (*m*-1) operations on one configuration plane can be processed by the actual array simultaneously at the same time slice, and the remained one rPU is being configured at the same time. That is the cost that must pay out for the reconfiguration of array.

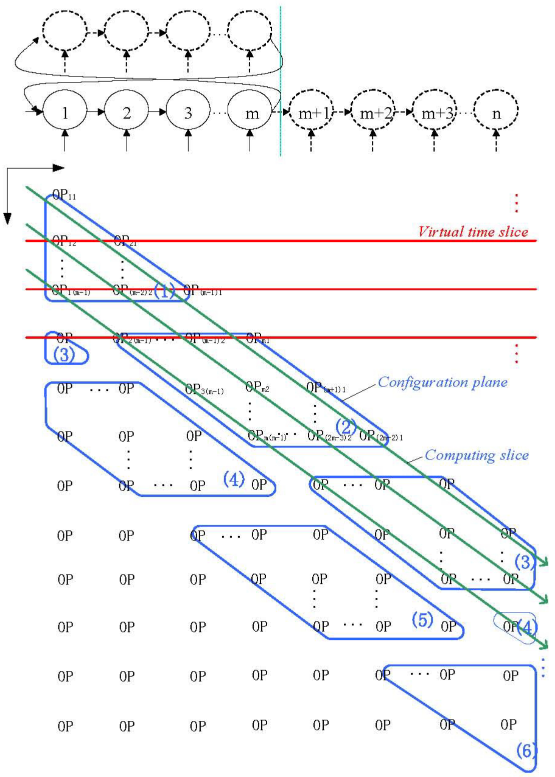

Fig. 3. Space-time plane of virtual array

The operation OP_{ij} of the rPU represents all possible operations that include inputting external data, executing arithmetic and logical operation (among the input data, local data and intermediate data) and outputting the results. A space-time plane of the actual array (Fig. 4.) can be achieved by arranging all these operations according to

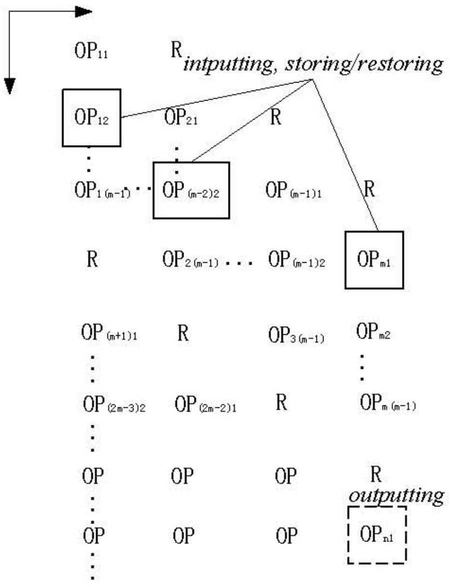

Fig. 4. Space-time plane of actual array

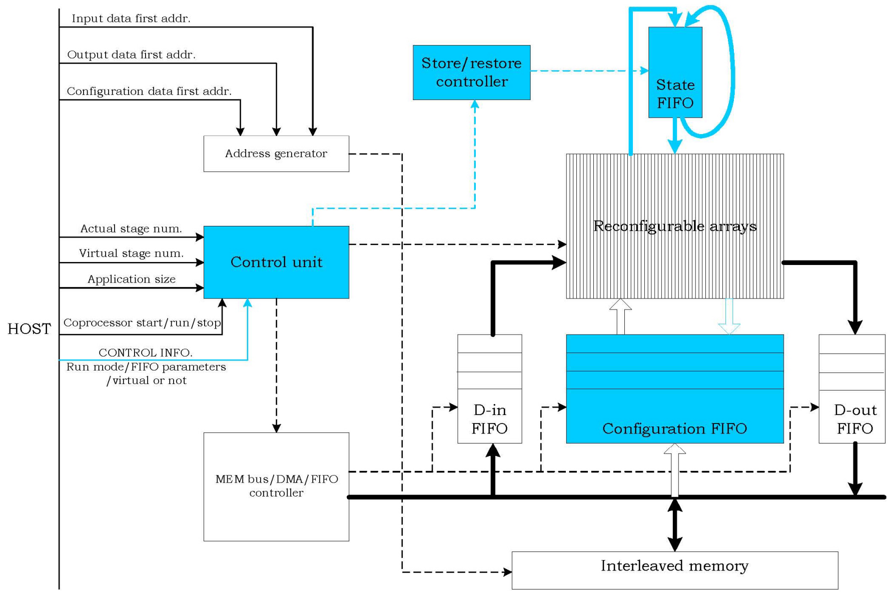

Fig. 5. Architecture of MPRS considering virtual array

the sequence number of configuration plane. From this new plane, we can decide when the operations of the same original virtual time slice are processed respectively; also can we decide the organization and timing of input/output data, and the storing/restoring time and contents of the intermediate data.

According to the analysis of configuration plane, we know that some special components should be designed in the MPRS if this system supports virtual array. All these components are marked with blue color in Fig. 5. The state FIFO is used to store the intermediate data and the store/restore controller decides when the store/restore operations start and finish. It should be noted that there are no additional storage devices for the storing/restoring of configuration data, and the configuration FIFO is reused for this purpose instead. As the precondition of this method, the reconfigurable array should work in a pure pipelined mode and the length of the virtual array should be smaller than the depth of the configuration FIFO. These two conditions can be met in our current MPRS implementation.

The computing time spend on processing the pipelined task using virtual array is determined by the scale of the actual array and the requirement of the task. Using the "configuration plane" method, it is easy to figure out (by the control unit of MPRS) the number of time slices needed to complete the given pipelined task. That number can be used to program the specific control register automatically, which is necessary to be definite for the designer of the system supporting virtual hardware.

4 Examples and Results

Matrix operation and motion estimation belong to the uniform linear recurrence applications fitting the MPRS array. Here we illustrate the design procedure and the programming procedure for the MPRS virtual array with these two examples and give the results finally.

4.1 Design and Programming Steps

The design steps of MPRS array list as follow, and the detail steps can be referred to another submitting paper "Mapping Algorithms to Multi-Pipeline Reconfigurable System" for limited space.

(1) Design the serial algorithm;

(2) Design the single assignment program by extending the index of the input variable;

(3) Construct the DG (Dependency Graph) according to the extended space-time index;

(4) Draw the DGRV (DG with Reconfigurable Variable) by localization and reconfigurablization processes;

(5) Design SFG (Signal Flow Graph) through projecting and scheduling the DGRV;

(6) Mapping the SFG into the multiple MPRS arrays simultaneously when the MPRS works in the parallel mode or mapping the SFG to the single chained MPRS

array when works in the chained mode. Working in the virtual mode is transparent for the mapping process;

(7) Reflect the mapping results into the different fields of the configuration word that will be used to reconfigure the MPRS arrays dynamically when perform computing.

The programming steps of MPRS list as follow.

(1) Prepare the configuration data achieved from above processes.

(2) Prepare the computing data achieved from the target application.

(3) Create the configuration/computing data file that will used by the main program. Firstly, draw the space-time plane of virtual array (like Fig. 3) and the space-time plane of actual array (like Fig. 4) respectively. Secondly, arrange the configuration data in proper sequence according to the configuration plane and organize the computing data properly according to the space-time plane of actual array. Finally, create the data files.

(4) Program the specific registers in MPRS to provide enough information for the system to run properly. This is done by creating main program.

(5) Compile the main program and run the executable code on the MPRS.

4.2 Matrix-Vector Multiplication

After implementing the procedures mentioned above, all the needed data and program are ready for running on the MPRS that works in the virtual mode.

Suppose that x is a 8-dimention vector, A is a $m\times 8$ matrix, and $y=Ax$ is a m-dimension vector. Here, m represents the scale of matrix-vector multiplication, and its value changes from 32 to 4096.

In this experiment, various scales of array are used to compare the efficiency of the virtual array with that of the normal array. In the first situation, the array consists of 4 rPUs; in the second situation, the array consists of 8 rPUs. The execution periods of SimpleScalar [9] and MPRS on different application scales are shown in Fig. 6(a). In this figure, the horizontal axis represents the scale of matrix-vector multiplication, and the vertical axis represents the number of execution periods. MPRS_A shows the performance of MPRS using normal array, and MPRS_V shows the performance using virtual array. It can be seen that the number of periods spent by MPRS is much smaller than SimpleScalar. Fig. 6(b) shows the speedup factor of MPRS vs. SimpleScalar. It can be seen that the speedup factor when using virtual array is nearly equal to that when using normal array.

We can draw another conclusion by analyzing the periods spending respectively on the MPRS array and on the whole MPRS system. The system overhead should occupy the larger proportion of the total execution periods if the scale of applications is not large enough. In this situation, the number of rPU in the array can be effectively reduced by using virtual array. Inversely, when the application scale is large enough and the memory access time is short enough, the expense of speed caused by using virtual array should be considered fully.

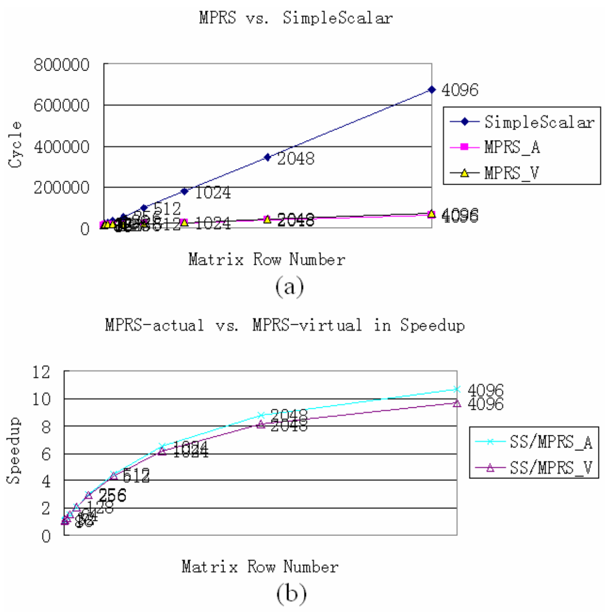

Fig. 6. (a) The number of execution periods of SimpleScalar and MPRS; (b) speedup factor

4.3 Motion Estimation

Since motion evaluation (ME) occupies the 98% processing time in video compressing and 42% in decompressing [10], it is important to enhance the execution speed of ME. One of most popular ME algorithm is the Full Search Block Matching (FSBM). FSBM can be expressed as follows:

$$MAD(m,n) = \sum_{i=0}^{N-1}\sum_{j=0}^{N-1} | R(i,j) - S(i+m, j+n) | \qquad -q \le m,n \le q$$

In the parallel work mode, MPRS can complete one FSBM of the standard MPEG scale (with N=8, q=8). Fig. 7 also shows the different results in other systems, in

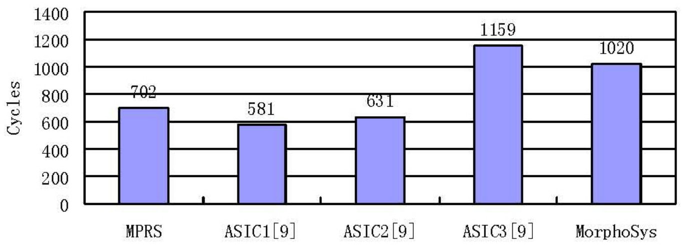

Fig. 7. Execution cycles of motion evaluation

which the ASICs have the special optimization for the FSBM [11]. Moreover, the same application in Pentium MMX needs 29000 cycles. It can be concluded that the speed of our MPRS in ME execution is 10 times faster than the general-purpose processor, also faster than the MorphoSys who is the similar reconfigurable computing system, and near the ASIC products.

5 Conclusions

This paper proposes a method of designing virtual hardware and exploiting target algorithms on it. The correct experiment results demonstrate that the method based on "configuration plane" is feasible. This new method can be applied to our MPRS system as well as to other systems using incremental reconfiguration.

References

1. Plessl, C., Platzner, M.: Virtualization of Hardware - Introduction and Survey. In: Proceedings of the International Conference on ERSA, pp. 63–69 (2004)
2. Hauck, S., Fry, T.W., Hosler, M.M., Kao, J.P.: The Chimaera Reconfigurable Functional Unit. IEEE Trans. on VLSI Systems. 12, 206–217 (2004)
3. Cronquist, D.C., Fisher, C., Figueroa, M., Franklin, P., Ebeling, C.: Architecture Design of Reconfigurable Pipelined Datapaths. In: Proceedings of the 20th Anniversary Conference on Advanced Research in VLSI, pp. 23–40 (1999)
4. Cadambi, S., Weener, J., Goldstein, S.C., Schmit, H., Donald, E.: Managing Pipeline-Reconfigurable FPGAs. In: ACM/SIGDA International Symposium on FPGAs, pp. 55–64 (1998)
5. Lorenzelli, F., Yao, K.: Integral Matrix-Based Technique for Systematic Systolic Design Integration. The VLSI Journal 20, 269–285 (1996)
6. Schmit, H., Cadambi, S., Moe, M., Goldstein, S.C.: Pipeline Reconfigurable FPGAs. The Journal of VLSI Signal Processing 24, 129–146 (2000)
7. Goldstein, S.C, Schmit, H., Budiu, M., Cadambi, S., Moe, M., Taylor, R.R.: PipeRench: A Reconfigurable Architecture and Compiler. IEEE Computer 33, 70–77 (2000)
8. Yin, Y.S., Li, L., Gao, M.L.: The Reconfigurable System Based on Multi-Pipeline (in Chinese). Microelectronics & computer 10, 88–91 (2005)
9. Burger, D., Austin, T.M.: The SimpelScalar Tool Set, version 2.0. University of Wisconsin-Madison Computer Sciences Department Technical Report #1342 (1997)
10. Miyamori, T., Olukotun, K.: REMARC: Reconfigurable Multimedia Array Coprocessor. IEICE Trans. on Inf. and Syst. E82–D, 389–397 (1999)
11. Singh, H., Ming-Hau, L., Lu, G., Kurdahi, F.J., Bagherzadeh, N., Chaves Filho, E.M.: MorphoSys: An Integrated Reconfigurable System for Data-Parallel and Computation-Intensive Applications. IEEE Trans. on Computers. 49, 465–481 (2000)