Int J Parallel Prog (2008) 36:478–492
DOI 10.1007/s10766-008-0079-0

Scalability and Parallel Execution of Warp Processing: Dynamic Hardware/Software Partitioning

Roman Lysecky

Received: 4 March 2008 / Accepted: 23 July 2008 / Published online: 19 September 2008

Abstract Warp processors are a novel architecture capable of autonomously optimizing an executing application by dynamically re-implementing critical kernels within the software as custom hardware circuits in an on-chip FPGA. Previous research on warp processing focused on low-power embedded systems, incorporating a low-end ARM processor as the main software execution resource. We provide a thorough analysis of the scalability of warp processing by evaluating several possible warp processor implementations, from low-power to high-performance, and by evaluating the potential for parallel execution of the partitioned software and hardware. We further demonstrate that even considering a high-performance 1 GHz embedded processor, warp processing provides the equivalent performance of a 2.4 GHz processor. By further enabling parallel execution between the processes and FPGA, the parallel warp processor execution provides the equivalent performance of a 3.2 GHz processor.

Keywords Warp processing · Hardware/software partitioning · Dynamically adaptable systems · Embedded systems

1 Introduction

Field programmable gate arrays (FPGAs) are increasingly becoming more popular and mainstream. FPGAs have moved from primarily being used as a prototyping and debugging platform to being incorporated within many computing domains, from high-performance supercomputers to consumer electronics. FPGAs can implement any hardware circuit simply by downloading bits for the hardware circuit, much in the

R. Lysecky (✉)
Department of Electrical and Computer Engineering, University of Arizona, Tucson, AZ 857821, USA
e-mail: rlysecky@ece.arizona.edu

same way that microprocessors can execute any software program simply by downloading a new application binary. The programmability, flexibility, rapid prototyping, and debugging advantages that FPGAs offer over ASICs have enabled more designs to be implemented using FPGAs. This trend will likely continue as FPGAs become more advanced with larger capacities, higher performance, domain specific architectural features, and decreasing costs.

With the continuing evolution of FPGAs, single-chip microprocessor/FPGA devices have emerged, incorporating one or more microprocessors and an FPGA into a single device along with efficient mechanisms for communication between the microprocessor, FPGA, and shared memory. Such devices, available from Xilinx, Altera, and Atmel, are ideally suited to hardware/software partitioning.

Hardware/software partitioning is the task of partitioning an application between software executing on a microprocessor and hardware coprocessors, which can be efficiently implemented within an FPGA. Extensive research has demonstrated the benefits that can be obtained by re-implementing a software application's critical kernels as a custom hardware coprocessor on an FPGA. For many applications, researchers and commercial vendors have observed overall application speedups of 10X–100X [1–11], and approaching 1000X [12,13] for several highly parallelizable applications. Hardware/software partitioning can also reduce energy consumption by up to 99% [14–17].

Programming an FPGA is fundamentally the same as programming a microprocessor. We can program an FPGA by downloading a bitstream into the FPGA's memory, just as we program a microprocessor by downloading a software binary. Thus, conceptually, a compiler can perform hardware/software partitioning, creating a software part executing on a microprocessor and a hardware part executing on an FPGA. Such compilers do exist, although mostly in the research domain. Unfortunately, while hardware/software partitioning offers the potential for tremendous speedups, automated partitioning compilers require a significant departure from mainstream software tools. First, the compiler must determine the critical regions of a software application, which typically requires profiling. Profiling, while conceptually straightforward, is often not a standard part of compilation—especially in embedded systems, where executing an application often involves complicated time-dependent interactions with the application's environment, making setting up simulations difficult. Second, the compiler must generate a binary for the microprocessor and a binary—or bitstream—for the FPGA, where the latter is by no means standard. Thus, partitioning compilers lose the important concept of a standard binary and the associated benefits of portability and backwards compatibility. Furthermore, such partitioning tools often require extensive design expertise and significant effort to maximize results.

Recently, researchers showed that by using decompilation techniques, designers could perform desktop hardware/software partitioning starting from software binaries rather than from high-level code, with competitive resulting performance and energy consumption [18]. Binary-level partitioning opens the door to dynamic hardware/software partitioning, in which an executing binary is dynamically optimized by moving software kernels to on-chip configurable logic.

We previously demonstrated the feasibility of dynamic hardware/software partitioning, which dynamically and autonomously improves the speed and energy

consumption of a software binary executing on a microprocessor, a process called warp processing [19]. A warp processor dynamically detects a software binary's critical kernels, re-implements those kernels as a custom hardware circuit in an on-chip FPGA, and replaces the software kernel by a call to the new hardware implementation of that kernel, all without any designer effort or knowledge thereof. While not all applications can be improved using warp processing, many can, with application speedups of 2X–10X and average energy reductions of 66%, compared to software only execution on a low-power embedded processor.

While warp processing enables a new computing domain through advances in embedded processing architectures in which software kernels can be dynamically and autonomously converted to hardware, previous warp processing work focused on low-end, low-power embedded processors as the main processor, such as a low-power ARM processor. Furthermore, the current warp processor architecture utilizes a mutually exclusive HW/SW execution framework, in which the software execution and hardware execution after partitioning are executed one after the other, thereby not leveraging the potential for executing software and hardware in parallel.

In this paper, we perform a thorough analysis of warp processing for several warp processor implementations, including a low-end AMR7 based warp processor and high-end XScale based warp processor. We further analyze the scalability of warp processors across many possible implementations with a processor independent analysis framework, utilizing ratios to characterize the processor and FPGA within the warp processor implementation. Finally, we analyze the potential of a parallel warp processing execution framework, in which the software and hardware execution after dynamic partitioning can overlap and execute in parallel, thereby achieving greater application speedups.

2 Warp Processing Overview

Figure 1 provides an overview of a warp processor, highlighting the steps performed during dynamic hardware/software partitioning. A warp processor consists of a main processor with instruction and data caches, an efficient on-chip profiler, our warp-oriented FPGA (W-FPGA), and an on-chip computer-aided design module (OCM). Initially, a software application executing on a warp processor will execute only on the main processor. During execution of the application, the profiler monitors the execution behavior of the application to determine the critical kernels within the application. After identifying the critical regions, the OCM re-implements the critical software regions as a custom hardware component within the W-FPGA.

We include a profiler within each warp processor to determine the critical kernels within the executing application that the warp processor could implement as hardware. This non-intrusive profiler monitors the instruction addresses on the instruction memory bus and maintains relative execution frequencies of application kernels using a cache of 16 entries [20]. Through simulations of our warp processor design, we have found that the profiler can accurately determine the critical regions of an application within 10 branch frequency register saturations. Using this methodology, the profiler

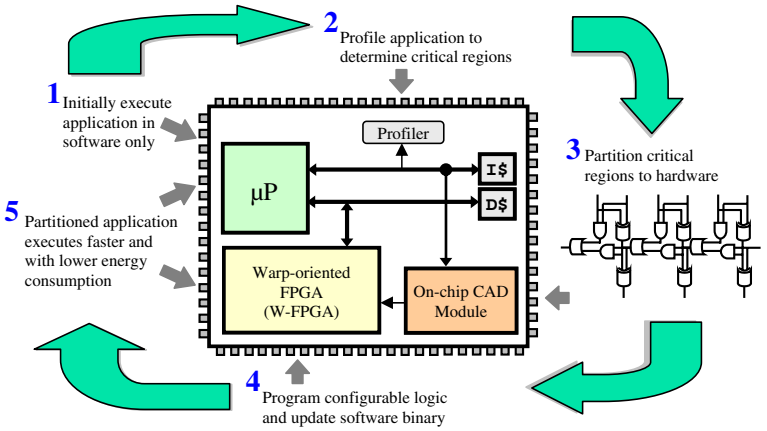

Fig. 1 Warp processor architecture overview

is able to properly select the correct critical kernels for partitioning for all applications we considered.

After profiling the application to determine the critical regions, the on-chip CAD module executes partitioning, synthesis, mapping, placement, and routing tools to re-implement the critical software kernels in hardware, producing a bitstream for programming the W-FPGA. The OCM first analyzes the profiling results for the executing application and determines which critical region the warp processor should implement in hardware. After selecting the software region to implement in hardware, the OCM decompiles the critical region into a control/dataflow graph and synthesizes the critical kernel to produce an optimized hardware circuit that is then mapped onto our custom FPGA through technology mapping, placement, and routing. Finally, the OCM configures the configurable logic and updates the executing application's binary code to utilize the hardware within the configurable logic fabric.

Currently, we implement the on-chip CAD module as lean software tools executing on a separate ARM7 processor including caches and separate instruction and data memories, which can either be located on-chip or off-chip depending on what is acceptable for a given warp processor implementation. Alternatively, we could eliminate the need for the OCM by executing the CAD tools as a software task on the main processor sharing computation and memory resources with the main application.

Significant previous research has been conducted detailing the warp processor architecture and on-chip CAD tools [19,21,22] and are beyond the scope of this paper.

3 Performance and Energy Benefits of Warp Processing

We first consider two alternative warp processor designs, one incorporating a low-power ARM7 [23] processor executing at 100 MHz and the other incorporating a high-performance XScale processor [24] executing at a maximum frequency of 624 MHz. Both warp processor implementations include a W-FPGA executing at 250 MHz, and a low-power ARM7 processor executing the on-chip CAD tools.

Table 1 Embedded benchmark applications

Benchmark	Benchmark suite	Description
brev	Powerstone	Bit reversal
g3fax	Powerstone	Group three fax decode
g721	Powerstone	CCITT voice decoding
matmul	Powerstone	Matrix multiplication
mpeg2	MediaBench	MPEG-2 decoder
pktflow	EEMBC	IP header validation
bitmnp	EEMBC	Bit manipulation
canrdr	EEMBC	Controller area network (CAN)
tblook	EEMBC	Table lookup and interpolation
ttsprk	EEMBC	Engine spark controller
matrix	EEMBC	Matrix operations
idct	EEMBC	Inverse discrete cosine transform
fir	EEMBC	Finite impulse response filter
url	NetBench	URL-based switching
rocm	Warp	Logic minimizer

We simulated the warp processor execution for a number of standard embedded benchmark applications collected from several different benchmark suites, including Powerstone [25], EEMBC [26] MediaBench [27], and NetBench [28], in addition to a lean logic minimization algorithm [21]. Table 1 presents a summary of the embedded benchmark applications considered. We only considered those applications amenable to speedup using FPGA, namely applications whose critical kernels do not use floating point arithmetic, dynamic memory allocation, recursion, function pointers, and regular pointers (other than for array accesses).

For all applications, we simulated the warp processor implementations utilizing an ARM and XScale port of the SimpleScalar simulator [29] to determine software execution cycles and gate-level simulations of the W-FPGA to determine hardware execution cycles for the partitioned critical kernels. All software and hardware execution times include the communication cycles required to communicate and synchronize between the processor and FPGA.

We calculated the energy consumed before and after dynamically partitioning the critical kernels to hardware using the following equations:

$$
\begin{aligned}
E_{\text{Total}} &= E_{\text{Proc}} + E_{\text{FPGA}} \\
E_{\text{Proc}} &= P_{\text{Proc(active)}} t_{\text{Proc(active)}} + P_{\text{Proc(idle)}} t_{\text{Proc(idle)}} \\
E_{\text{FPGA}} &= P_{\text{FPGA(active)}} t_{\text{FPGA(active)}} + P_{\text{FPGA(static)}} t_{\text{Total}}
\end{aligned}
$$

The total energy consumption (E_{Total}) is the sum of the energy consumed by the processor (E_{Proc}) and the energy consumed by the FPGA (E_{FPGA}). The energy consumed by the processor consists of the energy consumed executing the software applications and the idle energy consumed by the processor when the FPGA is active. The active energy

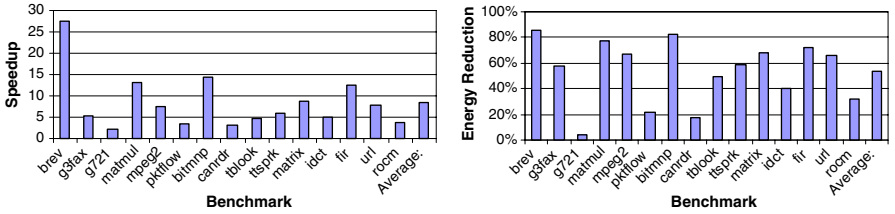

Fig. 2 Speedup and percentage energy consumption reduction of a 100 MHz ARM7 based warp processor compared to software execution alone

consumption of the processor is computed as the processor's active power consumption ($P_{\text{Proc(active)}}$) multiplied by the processor's active execution time ($t_{\text{Proc(active)}}$). The idle energy consumption of the processor is computed as the processor's idle power consumption ($P_{\text{Proc(idle)}}$) multiplied by processor's idle time ($t_{\text{Proc(idle)}}$). The energy consumed by the FPGA consists of the active energy consumed while the FPGA is executing the partitioned hardware kernels and the FPGA's static power consumption. The active energy consumption of the FPGA is computed as the FPGA's active power consumption ($P_{\text{FPGA(active)}}$) multiplied by the FPGA's active time ($t_{\text{FPGA(active)}}$). The static energy consumed by the FPGA is computed as the FPGA's static power consumption ($P_{\text{FPGA(static)}}$) multiplied by the total execution time (t_{Total}). While the static power consumption of the FPGA is consistent across all benchmarks and frequencies, the active power consumption of the FPGA is calculated for each partitioned critical kernels and operating frequency using a synthesizable VHDL specification of the W-FPGA design. We note that the functionality of the W-FPGA design was verified through post-layout simulation in collaboration with the Intel Research Shuttle.

Figure 2 presents the speedup and energy reduction benefits of a 100 MHz ARM7 based warp processor for several embedded benchmark applications compared to software only execution. The warp processor achieves an average application speedup of 8.4X, with applications speedups ranging from 2.3X for *g721* to over 25X for *brev*, indicating that warp processing is extremely effective in improving embedded system performance for low-end embedded systems. On average, warp processing provides the equivalent performance of an 840 MHz ARM processor. In addition, warp processing results in an average energy reduction of 53%, with a maximum reduction of 85% for *brev*. However, because the active power consumption of the FPGA is approximately 3X higher than the active power consumption of the ARM7 processor, several applications only achieve small reduction in energy consumption. For example, warp processing only reduces energy consumption by 3.7% for the application *g721*.

Figure 3 presents the speedup and energy reduction benefits of our alternative high-performance 624 MHz XScale based warp processor for several embedded benchmark applications compared to software only execution. The warp processor achieves an average application speedup of 3.2X over the already high-performance XScale processor. Applications speedups range from 1.4X for *canrdr* to almost 7X for *bitmnp*, indicating that warp processing is quite effective in improving embedded system performance, even for high-performance systems. In addition, warp processing results in an average energy reduction of 75%, with a maximum reduction of 92% for *brev*.

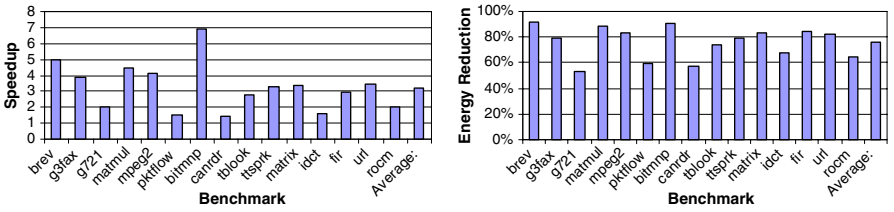

Fig. 3 Speedup and percentage energy consumption reduction of a 624 MHz XScale based warp processor compared to software execution alone

In general, the lower operating frequency of ARM based warp processor implementation will result in higher application speedups due to warp processing. Considering a fixed execution frequency for the on-chip FPGA, as the operating frequency of the main processor is increased, the speedup achieved by warp processing will decrease, due the higher base software execution of the faster processor. However, for some applications the performance improvements of warp processing are similar for both warp processor implementations. For example, the application speedup for *g721* is 2.3X and 2.0X for the ARM7 and XScale based warp processor implementations, respectively. Warp processing is able to partition four critical kernels within the *g721* application to hardware within the W-FPGA. However, those four loops comprise only 52% of the original software execution for both warp processor implementations. Thus, the maximum possible speedup that can be achieved by warp processing is 2.3X. Both the ARM7 and XScale based warp processors are able to achieve near optimal results as the average kernels speedup for the four kernels are 63X and 10X, respectively. For *g721*, the performance increase is limited by the size of the FPGA and number of kernels that can be supported within the FPGA.

On the other hand, the application speedup for *brev* is 27X and 5X for the ARM7 and XScale based warp processor implementations, respectively. *brev* performs an efficient bit reversal algorithm in which 99.5% application execution is within a single critical kernel. Thus, the reduction in speedup between the two implementations is related to the difference in original software execution. For *brev*, the ARM7 based warp processor is able to improve the performance of the critical kernel by 31X, while the XScale based warp processor only achieves a kernel speedup of 5X. The reduced kernel speedup of the XScale based warp processor can be attributed to the 6X faster software execution achieved by the warp processor, as both implementations result in the same hardware configuration within the FPGA.

3.1 Scalability

Application speedups and energy reduction of warp processing is dependent on the underlying processor and FPGA. For performance, this difference is due to the ratio of processor frequency to FPGA operating frequency, which can vary significantly across different systems. To analyze the scalability of warp processing across several possible warp processor implementations, we analyzed the performance and energy benefits of warp processing for several processor to FPGA frequency ratios.

Table 2 Ratios of processor to FPGA execution frequency and possible representative systems

μP:FPGA ratio	Representative systems (Processor MHz/FPGA MHz)
1:2.50	100 MHz/250 MHz (ARM7 Warp Processor)
1:2.25	85 MHz/200 MHz (MicroBlaze Warp Processor)
1:2.00	
1:1.75	
1:1.50	
1:1.25	200 MHz/250 MHz (ARM9 Warp Processor)
1:1.00	450 MHz/450 MHz (Virtex-4 FX)
1:0.75	
1:0.50	624 MHz/250 MHz (XScale Warp Processor)
1:0.25	1 GHz/250 MHz (High-end Embedded System)

Table 2 presents the processor to FPGA frequency ratios considered and possible representative system for several of those ratios. The ratios range from a low-end warp processor with a processor to FPGA ratio of 1:2.5, representing the ARM7 warp processor implementation previously considered, to a high-end warp processor with a ratio of 1:0.25. The high-end warp processor is representative of an implementation combining a 1 GHz processor with a 250 MHz FPGA. We note that the ratio of 1:1 is representative of several commercially available single-chip microprocessor/FPGA devices, such as the Virtex-4 FX device form Xilinx. The Virtex-4 FX FPGA incorporates a 450 MHz PowerPC and an FPGA capable of executing at a maximum frequency of 450 MHz—depending on the hardware implemented within the FPGA.

Figure 4 presents the speedup or warp processing for the various ratios of processor to FPGA frequency of Table 2, highlighting the speedup for several embedded application and the average speedup across all embedded application of Table 1. The three applications, *brev*, *g721*, and *mpeg2*, represent applications with the highest, lowest, and average speedups. Warp processor implementations in which the FPGA executes at a higher frequency than the processor can achieve extremely high applications speedups as the increased execution speed and parallelism of the FPGA allow for greatly improved execution of the critical kernels. Within this range of implementations, the application speedups range from 2.2X for *g721* with a ratio of 1:1.25 to 27X for *brev* with a ratio of 1:2.5. For *mpeg2*, the applications speedup ranges from 7.6X to 6.6X across the various warp processor implementations.

Starting with a ratio of 1:1, the performance improvements of warp processing are achieved primarily through the increased parallelism of the partitioned hardware implementation executing with the FPGA. If we consider an FPGA with a fixed operating frequency, the range of processor to FPGA frequency ratios from 1:1 to 1:0.25 corresponds to a 4X increase in the execution frequency of the processor. However, the increased parallelism of implementing critical kernels within the FPGA still provides good application speedups starting at 5.5X for a ratio of 1:1 to 2.4X and the high-end. Even considering a high-end embedded system with a 1 GHz processor and 250 MHz FPGA, warp processing is able to provide an average equivalent performance of a 2.4 GHz processor.

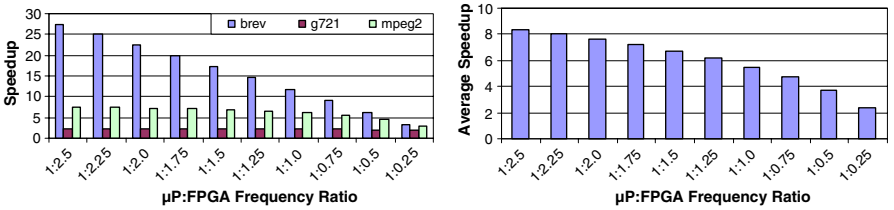

Fig. 4 Speedup or warp processing for various ratios of processor to FPGA frequency of Table 2, highlighting the speedup for several embedded application and average speedup across all embedded applications considered

Fig. 5 Average energy consumption reduction of warp processing for various processor:FPGA frequency ratios

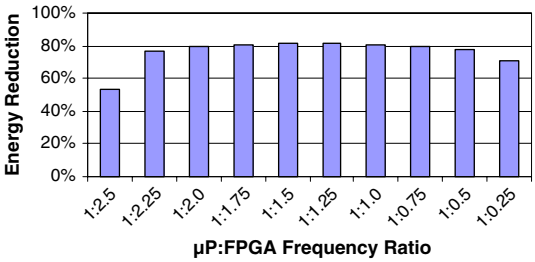

Figure 5 presents the energy consumption benefits of warp processors for the various ratios of processor to FPGA frequency of Table 2. As the ratio of processor to FPGA frequencies can correspond to very different warp processor implementations, we chose to fix the FPGA for all ratios, utilizing a 250 MHz FPGA for all implementations. Thus, the ratio of processor to FPGA frequency corresponds to an increase in processor frequency from 100 MHz to a maximum frequency of 1 GHz. The estimated processor power consumption increases linearly from 31 mW, at 100 MHz, to 988 mW, at 1 GHz. At the low-end of processor frequencies, the power consumption of the FPGA is over 3X higher than the low-power embedded processor, such as the ARM7. Although the application execution time is significantly reduced, the overall energy consumption reduction is only 53%. However, as the processor frequency increases relative to the FPGA, the energy consumption reduction quickly increases to a maximum reduction of 81%, achieved by the ratios of 1:1.75 to 1:1, corresponding to processor frequencies of approximately 150–250 MHz. At the high-end of processor frequencies the significantly increased power consumption of the processor results in decreased energy consumption of 71%. However, across the entire range of processor to FPGA frequency ratios, warp processing is able to achieve average energy reductions of at least 50%, ranging from 53% to 81%.

4 Parallel Warp Processing

The current warp processing implementation utilizes a mutually exclusive execution model, whereby either the main processor or the FPGA is active at any given time. Using this implementation, the main processor and FPGA can access the same data cache, thereby avoiding any cache coherency and/or data consistency issues. However,

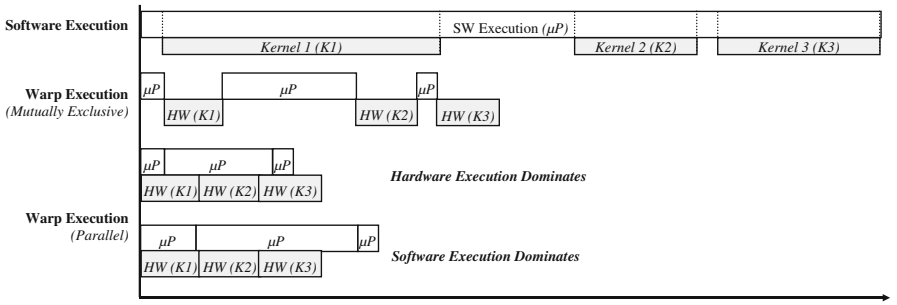

Fig. 6 Timeline comparing software only execution and warp processing execution, considering both a *mutually exclusive* warp processing execution and an ideal *parallel* warp processing execution (*Not to Scale*)

the potential for parallel execution in which both the microprocessor and FPGA can execute at the same time may result in increased application performance. Furthermore, the benefits of parallel warp processing depend both on the application and the warp processor implementation.

Figure 6 presents timelines illustrating software only execution, a mutually exclusive warp processing execution, and a parallel warp processing execution. A mutually exclusive execution implementation transitions from executing software on the processor to the hardware executing on the FPGA whenever a partitioned critical kernel is encountered. Alternatively, a parallel execution implementation would initialize the FPGA to execute the partitioned kernel and continue to execute software in parallel with the hardware execution. Ideally, the software and hardware execution would be able to synchronize any data dependencies and resolve any data consistencies between the hardware and software execution without delaying the execution of hardware or software, thereby achieving maximum parallelism. For some applications, the synchronization and communication between the software and hardware is minimal or nonexistent. For example, in the *mpeg2* video decoding application, the inverse discrete cosine transform (IDCT) is performed on a block by block basis and no synchronization is required between blocks. As such, parallel software and hardware execution of the IDCT function can be implemented without any intermediate synchronization, in which the software and hardware can simultaneously execute the IDCT on independent blocks within the video stream. For other applications, the software and hardware may need to communicate and synchronize to ensure proper execution, resulting in reduced parallel execution. For multitasked systems, however, while the hardware is executing for a given task, the processor can execute another software task, which would increase the potential for parallel execution. In this paper, we consider an idealized abstraction—one in which no communication and synchronization is required—to evaluate the maximum potential benefits of parallel execution for warp processing.

Within a parallel warp processing execution framework, two possible execution scenarios exist. The first scenario is hardware dominated execution, in which the partitioned hardware kernels execute for a longer duration than the remaining unpartitioned software execution. In this scenario, the application speedup from warp processing is limited by the hardware execution. The second scenario is software

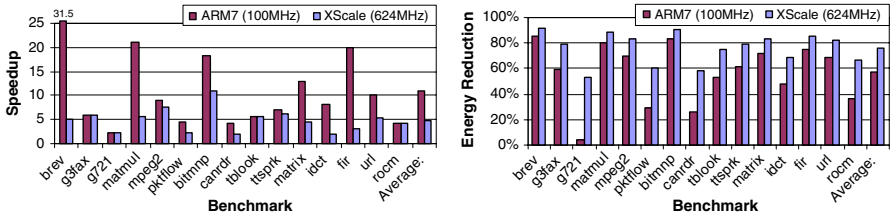

Fig. 7 Speedup and percentage energy consumption reduction of a *parallel* warp processing for 100 MHz ARM7 and 624 MHz XScale based warp processors compared to software execution alone

dominated execution, in which the remaining un-partitioned software executes longer than the hardware kernel execution. In this scenario, the application speedup is limited by the software execution, which is directly related to the total percentage of software execution of the critical kernels.

Figure 7 presents the speedup and energy reduction benefits for the ARM7 and XScale based warp processor implementations with an ideal parallel execution implementation. For the ARM7 warp processor implementation, the parallel execution model increases the average application speedup from 8.4X to 11X. For several applications, the benefits of parallel warp processing are almost negligible. The increase in performance from a parallel execution model for *g721* is only 0.04X, compared to the mutually exclusive implementation. *g721* is an example of software dominated execution, in which the high speedups of the partitioned critical kernels, 63X, do not allow for much parallel execution. After dynamic partitioning, the hardware execution of the critical kernels is only 2% of the total partitioned application execution, limiting the potential for parallel execution. On the other hand, *matmul* and *fir* are examples of application that greatly benefit from parallel execution, with increases from 13X to 21X and 13X to 20X, respectively. For *fir*, the partitioned hardware execution time if 63% of the total partitioned execution time, allowing a parallel warp processing implementation to potentially execute the remaining un-partitioned software in parallel with the hardware execution. The final parallel execution is an example of hardware dominated execution.

For the XScale based parallel warp processing, the average application speedup is increased from 3.2X, for a mutually exclusive execution, to 4.8X with parallel execution. For hardware dominated applications, the increase in performance from parallel execution is not as pronounced for higher performance processors due to the reduced software execution time. Faster software execution limits the amount of parallelism that can be achieved. This difference can be seen in the effects of parallel processing on the applications *matmul* and *fir*. The difference between mutually exclusive warp processing and parallel warp processing for these applications is less pronounced in the XScale based warp processor than for the ARM7 warp processor. Parallel execution results in a performance increase of 4.5X–5.7X and 2.9X–3.2X for *matmul* and *fir*, respectively. For *fir*, the partitioned hardware execution time is 91% of the total partitioned execution time, due to the increased execution speed of the remaining un-partitioned software, limiting the potential benefit of parallel execution. However, the resulting idle time of the software could be utilized to perform the communication and synchronization, thus helping to ensure the maximum parallelism can be achieved.

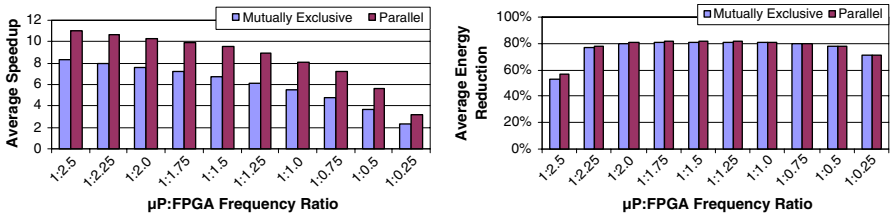

Fig. 8 Average speedup and percentage energy consumption reduction of a *mutually exclusive* and *parallel* warp processing for various processor:FPGA frequency ratios of Table 2

4.1 Scalability

We further analyze the performance and energy consumption of parallel warp processing compared to mutually exclusive warp processing for the processor to FPGA frequency ratios of Table 2 to further analyze the potential of parallel execution. Figure 8 presents the average speedup and energy consumption of mutually exclusive warp processing and parallel warp processing for various processor to FPGA frequency ratios. Figure 9 presents an alternative illustration of the benefits of parallel warp processing, plotting the average speedup and execution of warp processing with the ration of processor to frequency ratio for both mutually exclusive and parallel warp processing.

The absolute performance benefits of parallel execution decreases as the ratio of processor to FPGA frequency increases. As the software execution speed increases relative to the hardware execution, the potential for executing software and hardware in parallel decreases for many applications. From Fig. 9 we can actually see that the benefits of warp processing from parallel execution compared to mutually exclusive execution increases from 1.3X increase in performance at the low-end to a maximum of 1.5X increase at a ratio of 1:0.75 (1.33), visible as the increase in separation of between the two plots. The benefits of parallel execution then decrease from 1.5X to 1.4X for the high-end warp processor design. While the average benefits of parallel execution only range from 1.3X to 1.5X, the decrease in performance at the low-end and high-end imply that parallel warp processing provides the greatest benefit when the processor and FPGA frequency are close. We note that the highest potential for parallel execution is for applications in which the partitioned execution is equally divided between hardware and software application. We further note that as the processor

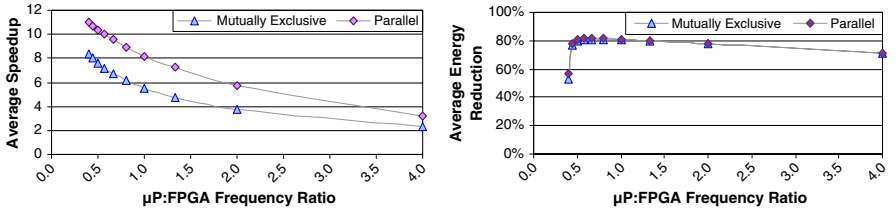

Fig. 9 Average speedup and energy consumption plotted against processor:FPGA frequency ratio for *mutually exclusive* and *parallel* warp processing

frequency increase relative to the FPGA, the higher performance of the processor will lead more applications into the category of being hardware dominated.

The energy consumption of mutually exclusive warp processing and parallel warp processing is almost identical. While parallel execution may reduce the application execution, the total energy consumed by the processor and FPGA will not significantly change as the active execution for both components is not reduced in a parallel execution framework.

5 Conclusions

Warp processing provides significant performance and energy benefits over software only execution by dynamically and transparently partitioning the critical kernels of an executing software applications to hardware implemented within an on-chip FPGA. We presented a thorough analysis of warp processing for two simulated warp processors, across several possible warp processor implementations, and considering a parallel execution framework for several embedded benchmark applications. Warp processing achieves excellent speedups across many different embedded processors, from low-end, low-power processors to high-performance embedded processors. On average a warp processing can provide a performance increase of 8.4X over a 100 MHz ARM7 processor and a 3.2X increase over a 624 MHz XScale processor. These performance benefits translate to an equivalent performance of a processor executing at 840 MHz to 2 GHz. Furthermore, while the benefits of warp processing decrease as the ratio of processor to FPGA frequency increases, the benefits of warp processing for high-end, high-performance embedded systems still achieve a 2.4X increase in performance and a 71% decrease in energy consumption, over a high-performance 1 GHz processor.

The benefits of warp processing can be further improved by enabling a parallel warp processing implementation in which the software executing on the processor and hardware executing within the FPGA are allowed to execute in parallel. While data and cache coherency issues will likely impose limits of the amount of parallelism that be achieved in practice, the potential for improving performance on average by 1.5X over the mutually exclusive warp processing execution is very promising.

6 Future Work

While warp processors already offer significant performance and energy advantages, warp processing is still in its infancy. Much research is needed to extend and improve warp processing, potentially to boarder application domains including desktops and servers. We are currently pursing further research in area of warp processors, specifically focusing on low-power, real-time, and desktop systems. We are currently pursuing research investigating the potential of utilizing voltage scalable processors and low-power modes to reduce power consumption of warp processing. Additionally, we are also evaluating the potential for leveraging the benefits of parallel warp processing by determining how to efficiently extract parallelism between the software and hardware components. Future work also include extending our dynamic partitioning

algorithms, warp processor architecture, and custom FPGA design to extend warp processing towards broader computing domains, including desktop, server, and scientific computing. Support for these domains requires supporting floating point arithmetic, extensive use of pointers, and dynamic memory allocations and represent three of the key challenges for future warp processing implementations.

References

1. Balboni, A., Fornaciari, W., Sciuto, D.: Partitioning and exploration in the TOSCA co-design flow. International Workshop on Hardware/Software Codesign, pp. 62–69 (1996)
2. Berkeley Design Technology, Inc.: http://www.bdti.com/articles/info_eet0207fpga.htm#DSPEnhanced%20FPGAs (2004)
3. Chen, W., Kosmas, P., Leeser, M., Rappaport, C.: An FPGA implementation of the two-dimensional finite-difference time-domain (FDTD) algorithm. In: Proceedings of the International Symposium on Field-Programmable Gate Arrays (FPGA), pp. 97–105 (2004)
4. Eles, P., Peng, Z., Kuchchinski, K., Doboli, A.: System level hardware/software partitioning based on simulated annealing and Tabu search. Kluwer's Des. Automat. Embedded Syst. **2**(1), 5–32 (1997)
5. Ernst, R., Henkel, J., Benner, T.: Hardware-software cosynthesis for microcontrollers. IEEE Des. Test Comput. **10**, 64–75 (1993)
6. Gajski, D., Vahid, F., Narayan, S., Gong, J.: SpecSyn: an environment supporting the specify-explore-refine paradigm for hardware/software system design. IEEE Trans. VLSI Syst. **6**(1), 84–100 (1998)
7. Guo, Z., Buyukkurt, B., Najjar, W., Vissers, K.: Optimized generation of data-path from C codes. In: Proceedings of the Design Automation and Test in Europe Conference (DATE), pp. 112–117 (2005)
8. Henkel, J., Ernst, R.: A hardware/software partitioner using a dynamically determined granularity. In: Design Automation Conference (1997)
9. Keane, J., Bradley, C., Ebeling, C.: A compiled accelerator for biological cell signaling simulations. In: Proceedings of the International Symposium on Field-Programmable Gate Arrays (FPGA), pp. 233–241 (2004)
10. Stitt, G., Vahid, F., McGregor, G., Einloth, B.: Hardware/software partitioning of software binaries: a case study of H.264 decode. In: Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS), pp. 285–290 (2005)
11. Stitt, G., Vahid, F., Nematbakhsh, S.: Power Savings and speedups from partitioning critical loops to hardware in embedded systems. ACM Trans. Embedded Comput. Syst. (TECS) **3**(1), 218–232 (2004)
12. Böhm, W., Hammes, J., Draper, B., Chawathe, M., Ross, C., Rinker, R., Najjar, W.: Mapping a single assignment programming language to reconfigurable systems. J. Supercomput. **21**, 117–130 (2002)
13. Venkataramani, G., Najjar, W., Kurdahi, F., Bagherzadeh, N., Bohm, W.: A compiler framework for mapping applications to a coarse-grained reconfigurable computer architecture. In: Conference on Compiler, Architecture and Synthesis for Embedded Systems (CASES 2001) (2001)
14. Henkel, J.: A low power hardware/software partitioning approach for core-based embedded systems. In: Design Automation Conference, pp. 122–127 (1999)
15. Henkel, J., Li, Y.: Power-conscious HW/SW-partitioning of embedded systems: a case study on an MPEG-2 encoder. In: Proceedings of Sixth International Workshop on Hardware/Software Codesign, pp. 23–27, March 1998
16. Stitt, G., Vahid, F.: The energy advantages of microprocessor platforms with on-chip configurable logic. IEEE Des. Test Comput. **19**(6), 36–43 (2002)
17. Wan, M., Ichikawa, Y., Lidsky, D., Rabaey, L.: An power conscious methodology for early design space exploration of heterogeneous DSPs. In: Proceedings of the ISSS Custom Integrated Circuits Conference (CICC) (1998)
18. Stitt, G., Vahid, F.: New decompilation techniques for binary-level co-processor generation. In: Proceedings of the International Conference on Computer Aided Design (ICCAD) (2005)
19. Lysecky, R., Stitt, G., Vahid, F.: Warp processors. ACM Trans. Des. Automat. Electron. Syst. (TODAES) **11**(3), 659–681 (2006)
20. Gordon-Ross, A., Vahid, F.: Frequent loop detection using efficient non-intrusive on-chip hardware. In: Proceedings of the Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES), pp. 117–124 (2003)

21. Lysecky, R., Vahid, F.: On-chip logic minimization. In: Proceedings of the Design Automation Conference (DAC), pp. 334–337 (2003)
22. Lysecky, R., Vahid, F., Tan, S.: Dynamic FPGA routing for just-in-time FPGA compilation. In: Proceedings of the Design Automation Conference (DAC), pp. 954–959 (2004)
23. ARM Ltd.: ARM7 Processor Family. http://www.arm.com/products/CPUs/families/ARM7Family.html (2006)
24. Intel Crop.: XScale PXA27x Processor Family. http://www.intel.com/design/pca/prodbref/253820.htm (2006)
25. Malik, A., Moyer, B., Cermak, D.: A low power unified cache architecture providing power and performance flexibility. In: Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED), pp. 241–243 (2000)
26. EEMBC.: Embedded Microprocessor Benchmark Consortium. http://www.eembc.org (2005)
27. Lee, C., Potkonjak, M., Mangione-Smith, W.: MediaBench: a tool for evaluating and synthesizing multimedia and communications systems. In: Proceedings of the International Symposium on Microarchitecture (MIO), pp. 330–335 (1997)
28. Memik, G., Mangione-Smith, W., Hu, W.: NetBench: a benchmarking suite for network processors. In: Proceedings of the International Conference on Computer-Aided Design (ICCAD), pp. 39–42 (2001)
29. Burger, D., Austin, T.: The SimpleScalar tool set, version 2.0. SIGARCH Comput. Architect. News **25**(3), 13–25 (1997)