A Run-time Task Migration Scheme for an Adjustable Issue-slots Multi-core Processor

Fakhar Anjam, Quan Kong, Roel Seedorf, and Stephan Wong

Computer Engineering Laboratory,
Delft University of Technology,
Mekelweg 4, 2628 CD, Delft, The Netherlands
Email: {F.Anjam, R.A.E.Seedorf, J.S.S.M.Wong}@tudelft.nl,
kongquanquan@hotmail.com

Abstract. In this paper, we present a run-time task migration scheme for an adjustable/reconfigurable issue-slots very long instruction word (VLIW) multi-core processor. The processor has four 2-issue ρ-VEX VLIW cores that can be merged together to form larger issue-width cores. With a task migration scheme, a code running on a core can be shifted to a larger or a smaller issue-width core for increasing the performance or reducing the power consumption of the whole system, respectively. All the cores can be utilized in an efficient manner, as a core needed for a specific job can be freed at run-time by shifting its running code to another core. The task migration scheme is realized with the implementation of interrupts on the ρ-VEX cores. The design is implemented in a Xilinx Virtex-6 FPGA. With different benchmarks, we demonstrate that migrating a task running on a smaller issue-width core to a larger issue-width core at run-time results in a considerable performance gain (up to 3.6x). Similarly, gating off one, two, three, or four cores can reduce the dynamic power consumption of the whole system by 24%, 42%, 61%, or 81%, respectively.

Keywords: Softcore, VLIW processor, Interrupts, Multi-core, Task migration

1 Introduction

Reconfigurable processors have filled the gap between general-purpose (GP) cores and application-specific integrated circuits (ASICs) in such a way that higher performance can be achieved without losing flexibility. A softcore processor is a processor that can be parameterized at design time and/or reconfigured at run-time when implemented in a field-programmable gate array (FPGA). It provides an efficient way to adapt to large number of applications.

The ρ-VEX [24] is a reconfigurable and extensible softcore very long instruction word (VLIW) processor. The processor can be adapted to different applications. The entire or partial datapath of the processor can be reconfigured. Multiple smaller issue-width cores can be combined to create a larger issue-width

core to exploit the instruction level parallelism (ILP) available in an application and improve the performance [3].

Task migration among different cores has been studied in the context of multi-core architectures. There could be different reasons for task migration. For example, the task migration is used for balancing workload, power, and thermal characteristics [21][17][13][5][4]. A fair task distribution results in lower power consumption in all cores, lower network and memory traffic, and lower heating of the overall system. In case of a fault at a core, its running code can be migrated to another core as well. We utilize the task migration for possible improvement in performance, workload balancing, and power reduction.

Figure 1 depicts the timeline for a task migration example. At a time instance, *core1*, a 2-issue core is running *task1* and requires time *t1* to finish the task. *Core2*, which is a 4-issue core is running *task2* and requires time *t2* to finish the task. At *t2*, *core2* is free, and in a time Δt, *task1* can be migrated from *core1* to *core2*. Since *core2* is a larger issue-width core, it can boost the performance and hence finishes *task1* at $t3 < t1$. Similarly, shifting from a larger issue-width core to a smaller issue-width core at run-time and turning off the larger issue-width core can reduce the power consumption of the overall system.

We first present the design and implementation of the interrupts system. The interrupts system is parameterized to support different applications. Parameters include the number of interrupt vectors, the interrupt priority for each vector, and the interrupt service routine (ISR) location address in the instruction memory. We implemented the interrupts in an reconfigurable issue-slots multi-core processor [3]. The processor has four 2-issue cores. Each core can be run independently. Multiple 2-issue cores can be combined to make a larger issue-width core. Building on the interrupts system, we developed a mechanism for task switching or task migration between different cores when these cores are combined or split for increasing the performance or reducing the dynamic power of the system. Because different issue-width VLIW cores require different codes, we assume that different code versions are available with defined switching points.

The contribution of the paper is summarized as follows:

- design and implementation of the interrupts system tailored for the ρ-VEX VLIW processor;
- setting run-time adaptation of the issue-slots by utilizing the ρ-VEX processors with interrupts in a reconfigurable issue-slots multi-core processor;

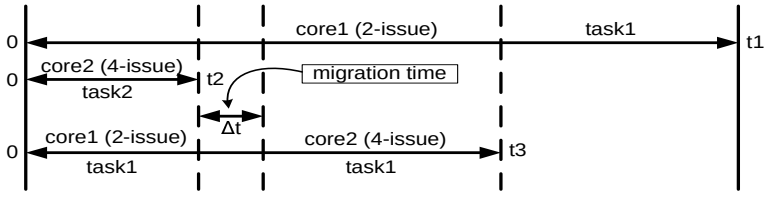

Fig. 1. A task migration example

– design and implementation of a task migration scheme targeting performance improvement, power reduction, and workload balancing.

The remainder of the paper is organized as follows. Section 2 presents some related work. The ρ-VEX processor and the related toolchain are briefly discussed in Section 3. The proposed task migration scheme for the ρ-VEX based adjustable/reconfigurable issue-slots multi-core processor is presented in Section 4. Experimental results are discussed in Section 5. Finally, Section 6 presents the conclusions.

2 Related Work

There are several softcore VLIW processors discussed in literature. Spyder [10] appeared as the first softcore VLIW processor. The toolchain was not complete and there was no interrupts system on the processor. An FPGA-based design of a softcore VLIW processor based on the ISA of the Nios-II soft processor is presented in [12]. Due to the licensed Nios-II, this VLIW design is not much flexible and not open-source. The design can use the interrupts system of the Nios-II architecture. In [18], the micro-architecture of a customizable softcore VLIW processor is presented. The limitation is the absence of a compiler. The processor does not have an interrupts system. All of these processors cannot adjust their issue-slots at run-time to exploit the available ILP.

Several interrupts handling schemes to reduce the size of contexts to be switched to minimize the interrupt latency for VLIW and DSP processors are presented in [15][9][22]. All these mechanisms need the support of a relative compiler and even processor architecture. The ρ-VEX softcore is a parameterized open-source softcore VLIW processor and can be adapted to different applications [23]. In this paper, we implemented the interrupts system on the ρ-VEX processor to further enhance its capabilities, and use it for task migration.

Task migration is used in multi-core architecture to balance the workload and network congestion. An unbalanced workload may result in excessive power consumption and thermal hot-spots and unbalanced network congestion may result in missed deadlines. [21] and [17] present different task or process migration mechanisms and algorithms. The authors in [13] discuss different policies for real-time task migration in embedded multi-core architectures. [2] assesses the impact of task migration on embedded soft real-time streaming multimedia applications. Here, a middleware infrastructure at operating system (OS) level supporting dynamic task allocation for non-uniform memory architectures (NUMA) is presented. [11] presents a context-aware run-time adaptive task migration mechanism to reduce the task migration latency in multi-core architectures. A task migration between two cores results in cache warm-up overheads on the target core, which can result in missed deadlines for tight real-time schedules. [19] proposes a micro-architectural support for migrating cache lines that enables real-time tasks to meet their deadlines in the presence of task migration.

[5] and [8] present policies for task migration to control the thermal characteristics in multi-core systems. Energy-efficient real-time task scheduling and

migration in multiprocessor systems is discussed in [25] and [20]. In [4], the authors discuss the impact of task migration in network-on-chip based MPSoCs for soft real-time systems. [16] presents techniques to selectively migrate the code/data to reduce communication energy in embedded MPSoCs. [14] presents a fault-and-migrate mechanism for asymmetric multi-core architectures which traps a fault when a core executes an unsupported instruction, migrates the faulting thread to a core that supports the instruction, and allows the OS to migrate it back when load balancing is necessary.

We implemented a run-time task migration scheme for a reconfigurable and adjustable issue-slots VLIW multi-core processor. This processor has four 2-issue ρ-VEX cores. An interrupts system has been implemented for these cores, and hence a running core can be interrupted, its state saved, and can be transferred to another core. Multiple cores can be merged at run-time to increase the performance. We do not utilize caches rather we implement local memories. We utilize task migration for improving the performance or reducing the power consumption.

3 The ρ-VEX VLIW Processor

The VEX instruction set architecture (ISA) is a 32-bit clustered VLIW ISA that is scalable and customizable to individual application domains. The VEX ISA is loosely modeled on the ISA of the HP/ST Lx (ST200) family of VLIW embedded cores [6]. Based on trace scheduling, the VEX C compiler is a parameterized ISO/C89 compiler. A flexible programmable machine model determines the target architecture, which is provided as input to the compiler. A VEX software toolchain including the VEX C compiler and the VEX simulator is made freely available by the Hewlett Packard Laboratories [1].

The ρ-VEX is a configurable (design-time) open-source softcore VLIW processor [23]. The ISA is based on the VEX ISA [7]. Different parameters of the ρ-VEX processor, such as the number and type of functional units (FUs), number of multiported registers (size of register file), number and type of accessible FUs per syllable, width of memory buses, and different latencies can be changed at design time. Figure 2 depicts the organization of a 32-bit, 2-issue ρ-VEX VLIW processor implemented in an FPGA. The ρ-VEX processor is a 5-stage pipelined processor consisting of fetch, decode, execute 0, execute 1/memory, and writeback stages. There are two arithmetic logic units (ALUs), two multiplication units (MUL), a control/branch unit (CTRL), and a load/store (LS) or memory unit (MEM). There are two multiported register files: a 64×32-bit general-purpose register (GR) file and an 8×1-bit branch register (BR) file. The instruction and data memories for the processor are implemented with block RAMs (BRAMs). The data memory is also utilized for storing the state or context of a program when an interrupt is serviced. The ρ-VEX processor supports reconfigurable operations, as the VEX compiler supports the use of custom operations via pragmas inside the application code.

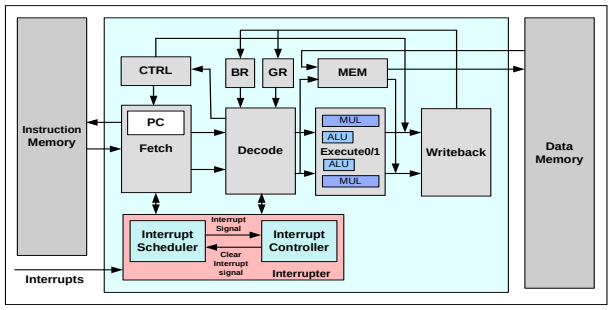

Fig. 2. A 2-issue ρ-VEX processor with the Interrupter

4 Task Migration Scheme

In this section, we first discuss the interrupts system for the ρ-VEX processor. Subsequently, we explain how we utilize the interrupts system in a ρ-VEX based multi-core system for task migration among different cores.

4.1 Interrupts System

Figure 2 depicts our interrupts system called the *interrupter* embedded into the ρ-VEX processor. It can be easily plugged in or out of the ρ-VEX core. The interrupter receives input signals from interrupt pins and then generates control signals to the fetch stage to reschedule instructions such that an ISR could be executed. At the same time, the necessary context is stored. When a *return from interrupt (RFI)* instruction is decoded, a signal is passed to the interrupter to indicate the end of an ISR. After that the context is restored back to the core and the core resumes the original execution. The interrupter has two sub modules: *interrupt scheduler* and *interrupt controller*.

Interrupt Scheduler: The interrupt scheduler has the knowledge of the task that is being executed and the requests that are issued to the ρ-VEX processor. The interrupt scheduler is responsible for (1) receiving the interrupt input signals from different interrupt sources, (2) scheduling different tasks into the task queue, and (3) enabling interrupt requests to the interrupt controller when the priority of the requested task is higher than the current task. There are two inputs for the interrupt scheduler: external *interrupt in* signals from the outside world and the internal *clear* interrupt-flag signal from the interrupt controller. The *interrupt in* signal adds tasks to the task queue and the *clear* signal removes it from the task queue. Only if an interrupt with a higher priority enters, or a higher priority task is finished, a waiting task can become active. The interrupt vector table records information such as the interrupt vectors (type of interrupts) and their priorities, interrupt flags which show the status of each interrupt request, ISR addresses, and the interrupt enable bits to mask the interrupts.

Interrupt Controller: The interrupt controller's main jobs are (1) receiving interrupt request signals from the interrupt scheduler, (2) storing the context, (3) loading the ISR address, (4) restoring the context, and (5) resuming the main program again from the point where it was left before the interrupt. The interrupt controller is designed as a finite state machine (FSM). An interrupt queue is implemented to record information of ISR addresses, return addresses, and interrupt vectors received from the interrupt scheduler along with the interrupt request signal. We utilize the pipeline of the processor to perform the context storing and restoring and the instructions for storing and restoring the context are generated in the software.

ISA Support Software Interrupt and Interrupt Enable/Disable: Since in the original VEX ISA there is no instruction that can generate an interrupt, consequently, we extended the instruction set to support this functionality. We designed and implemented a custom instruction for the ρ-VEX processor called *INT_SOFT*. With this instruction, a software code can interrupt the core.

We extended the instruction set of the ρ-VEX processor with two more instructions, one for enabling and one for disabling the interrupts. When these instructions are decoded, enable/disable signals are sent to the interrupts system, and hence the interrupts can be masked.

4.2 Run-time Task Migration for VLIW Multi-core Processor

In [3], the authors presented an architecture where a group of small 2-issue VLIW cores can be combined to make a larger issue-width core to exploit the available ILP in an application. Figure 3 depicts the general view of this adjustable issue-slots processor. A group consists of four 2-issue cores. Each core can be run independently. Multiple 2-issue cores can be combined or split at run-time. Within a group of four 2-issue cores, following are the possible configurations of VLIW cores:

1. four 2-issue cores;
2. one 4-issue and two 2-issue cores;
3. two 4-issue cores;
4. one 8-issue core.

When multiple cores are configured, some of them can be gated off and thereby adding more configurations and flexibility to the system. The issue-width of the cores is reconfigured by writing to a configuration register, and it takes only one cycle to configure/adjust the issue-width of the cores. The methodology utilized in [3] was that the cores could only be combined or split when these were idle (i.e., had finished their current execution). In this paper, we enhanced that architecture. We provided another level of control to that multi-core architecture with the development of the interrupts system. Each core is now able to pass on its environment (execution state) to another core of the same or different type in order to manage the cores utilization at run-time.

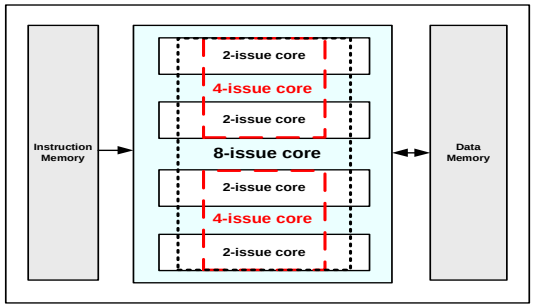

Fig. 3. The adjustable issue-slots VLIW Multi-core processor

We can now combine or split cores that are even not idle. We implemented an environment shifting or task migration mechanism for the cores utilizing the interrupts system. The environment shifting is needed in different situations. For example, if a larger issue-width core becomes available, it might be needed to switch an application running on a smaller issue-width core to the larger issue-width core for performance reasons. Similarly, one might need to switch a code running on a larger issue-width core to a smaller issue-width core and turn the larger issue-width core off to reduce the dynamic power consumption of the whole system at run-time. We assume here that different code versions of the same application are accessible and there are defined switching points available in the codes. These code versions are manually generated at the assembler level.

Figure 4 depicts the mechanism for migrating a task from ρ-VEX1 to ρ-VEX2. Here ρ-VEX1 and ρ-VEX2 could be any issue-width cores (2-issue, 4-issue, or 8-issue). A scheduler (currently implemented in a hardware, but in future may be a process of an operating system running on a certain core) controls the task migration. When an application is running on a certain issue-width core and a request to migrate to another core (a request to change the

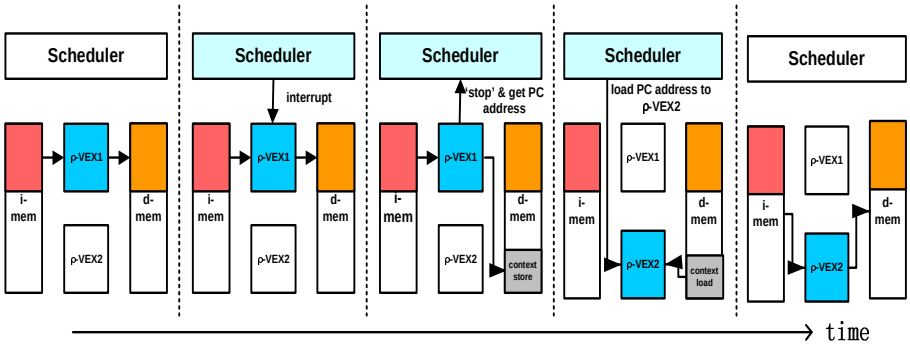

Fig. 4. Mechanism for task migration

issue-width) is received, the application is first allowed to execute through to the switching point on the current core, and then the process for migrating the task to the new core is started. When shifting a code running on ρ-VEX1 to ρ-VEX2, the following steps are performed by the scheduler as depicted in Fig. 4:

- generate an interrupt on ρ-VEX1 core
- a special ISR is called on ρ-VEX1 which stores the context into the data memory (shared memory accessible to all cores) and the program counter (PC) address with respect to a defined switching point where the currently running program was stopped is recorded
- reconfigure the issue-width of the core (now called ρ-VEX2) by changing the configuration register values [3]
- generate an interrupt on ρ-VEX2 core
- a special ISR is called on ρ-VEX2 that restores the context from the data memory
- load the PC address into ρ-VEX2
- start ρ-VEX2 to resume execution of the remaining code

Here, we only store the general-purpose registers and the branch registers. We implement the stack in the data memory accessible to both cores and hence, we do not store and restore the stack while moving the task from one core to another. The new and the previous cores should know the address where the stack is implemented, and it is done at compile/assemble time. This reduces the migration time among different cores.

5 Experimental Results

Table 1 presents the hardware resource utilization for the adjustable issue-slots processor with the interrupts system. There is a marginal increase in the hardware resources and the critical path remains the same. The processor has four 2-issue cores, each having 2 ALUs, 2 multipliers, 1 load/store (LS) unit, a 64×32-bit general-purpose register file and an 8×1-bit branch register file. These cores can be run independently or combined at run-time to make larger issue-width cores. We utilized the Xilinx ISE release version 12.4 for synthesis and implementation and the target FPGA device is Virtex-6 *XC6VLX240T-1-FF1156* available on the *ML605* development board. The blockRAMs (BRAMs) utilized are 36 kbits BRAMs.

In the ρ-VEX architecture, the general-purpose register number 0 ($r0.0) is hardwired to value zero, and cannot be written. When read, it will always

Table 1. Hardware resource utilization and maximum frequency for the adjustable issue-slots VLIW multi-core processor with the interrupts system

Processor	Registers	LUTs	DSPs	BRAMs	Max. Frequency
Original adjustable processor	2880	15600	128	64	110 MHz
New adjustable processor	4528	16281	128	64	110 MHz

return value zero. Therefore, it is not stored during context store. The interrupt response time for our interrupter is 76 cycles. It includes 4 cycles for completing the currently fetched instruction and stopping the pipeline, 1 cycle for scheduling the interrupt, 63 cycles for moving the general-purpose registers and 8 cycles for moving the branch registers to the memory.

In our case, task migration from one core to another requires a total of 155 cycles. Out of these 155 cycles, 76 cycles are required for storing the context of the first core, 1 cycle for accessing the program counter of the first core, 1 cycle for reconfiguring the issue-width, 76 cycles for restoring the context on the newly configured core, and 1 cycle for loading program counter of that core. This means that switching a running application from one type of core to another core consumes 155 additional cycles, but then the execution time for the remaining part of the application could be reduced much.

We considered the following benchmark applications/kernels: advanced encryption standard (AES) encode and decode, inverse discrete cosine transform (IDCT), matrix multiplication, finite impulse response (FIR) filter, Hamming distance, secure hash algorithm (SHA), Huffman compression, data encryption standard (DES), and Sobel filter. Generally, these applications/kernels are part of some large applications such as H.264, and are repeated continuously or at least many times. Normally, these applications/kernels are implemented in the form of functions operating on a subset of data continuously; hence, the call to these functions within the code can be marked as a switching point. For example, the AES encode/decode takes 16 bytes input at a time and encrypt/decrypt them to generate 16 bytes output. When an application is running on a specific issue-width core, the application repeatedly call the kernel functions each time with different data set. The request for the change of the issue-width of a core can only be fulfilled when the current execution on that core has reached to a switching point in the code.

Figure 5 depicts the speedup for different benchmarks when the applications are migrated from a 2-issue core with 1 load/store (LS) unit to a larger issue-width core. As depicted in the figure, the compiler is able to extract more ILP, and hence the execution time is reduced. In our multi-core system, each of the four 2-issue cores has 1 LS unit. When multiple 2-issue cores are merged, the resulting larger issue-width core can also utilize the additional LS units to increase the data input (provided the data memory has multiple ports) and hence can further increase the performance for different applications. Figure 5 depicts the speedup for the benchmarks when the applications are migrated from a 2-issue core with 1 LS unit (2-issue-1-LS) to 4-issue-1-LS, 8-issue-1-LS, 4-issue-2-LS, and 8-issue-4-LS cores at different percentage of the total execution cycles for the 2-issue-1-LS core. The maximum speedup is at 0%, i.e., when the application has just started on the 2-issue-1-LS core. Hence, the speedup is more when the migration is done at the earlier stages of execution.

Similarly, when a code running on a larger issue-width core is shifted to a smaller issue-width core (e.g., from an 8-issue to a 2-issue), the unused issue-slots or 2-issue cores can be gated off to reduce the power consumption of the system.

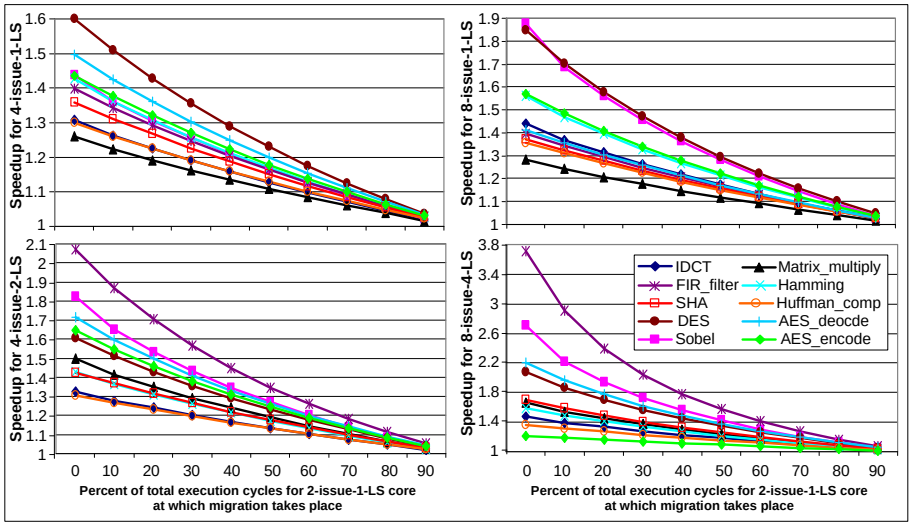

Fig. 5. Speedup compared to a 2-issue core with 1 load/store (LS) unit

We present the power consumption results for ASIC instead of FPGAs because clock gating is not effective in the current state of the art FPGAs, and techniques like gated-Vdd or power gating to reduce the leakage/static power cannot be applied. In the case of ASIC, all these techniques are possible and hence, task migration from a larger to a smaller issue-width core can reduce the power consumption. We implemented clock gating in our multi-core processor. When a core is not in use, it can be gated off to reduce the dynamic power consumption. Figure 6 presents the dynamic power consumption for our multi-core processor. We utilized the Synopsis Design Compiler and 90nm ASIC technology. From Fig. 6, it can be observed that gating off one, two, three, or four cores can reduce the dynamic power consumption of the whole system by 24%, 42%, 61%, or 81%, respectively.

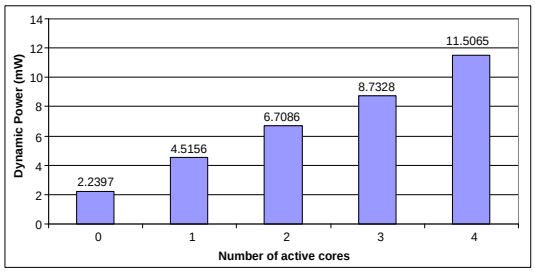

Fig. 6. Dynamic power consumption for the adjustable issue-slots processor with 90nm ASIC technology

6 Conclusions

In this paper, we presented a task migration scheme for a reconfigurable issue-slots VLIW multi-core processor having four 2-issue ρ-VEX cores. With the capability of task migration, the cores can be utilized more efficiently. A task running on a core can be migrated to a larger or a smaller issue-width core to increase the performance or reduce the power consumption, respectively. To realize the task migration scheme, a parameterized interrupts system is implemented for the processor. The design is implemented in a Xilinx Virtex-6 FPGA. There is a marginal increase in resources compared to the original processor. With different benchmarks, we demonstrated that migration from a smaller issue-width core to a larger issue-width core could result in considerable performance improvements (up to 3.6x). Additionally, we demonstrated that gating off one, two, three, or four cores in our multi-core processor could reduce the dynamic power consumption of the whole system by 24%, 42%, 61%, or 81%, respectively. For future work, we are considering the implementation of different migration policies for the scheduler.

Acknowledgment

This work is supported by the European Commission in the context of the ERA (Embedded Reconfigurable Architectures) collaborative project #249059 (FP7). The opinions expressed in this paper are of the authors only and in no way reflect the European Commissions opinions.

References

1. H. P. Labs. VEX Toolchain, http://www.hpl.hp.com/downloads/vex/
2. Acquaviva, A., Alimonda, A., Carta, S., Pittau, M.: Assessing Task Migration Impact on Embedded Soft Real-Time Streaming Multimedia Applications. EURASIP Journal on Embedded Systems. pp. 1–15 (2008)
3. Anjam, F., Nadeem, M., Wong, S.: Targeting Code Diversity with Run-time Adjustable Issue-slots in a Chip Multiprocessor. In: Design, Automation, and Test in Europe Conference. pp. 1358–1363 (2011)
4. Briao, E.W., Barcelos, D., Wronski, F., Wagner, F.R.: Impact of Task Migration in NoC-based MPSoCs for Soft Real-time Applications. In: International Conference on VLSI-SoC. pp. 296–299 (2007)
5. Cuesta, D., Ayala, J.L., Hidalgo, J.I., Atienza, D., Acquaviva, A., Macii, E.: Adaptive Task Migration Policies for Thermal Control in MPSoCs. In: International Symposium on VLSI. pp. 110–115 (2010)
6. Faraboschi, P., Brown, G., Fisher, J.A., Desoli, G., Homewood, F.: Lx: A Technology Platform for Customizable VLIW Embedded Processing. In: International Symposium on Computer Architecture. pp. 203–213 (2000)
7. Fisher, J.A., Faraboschi, P., Young, C.: Embedded Computing: A VLIW Approach to Architecture, Compilers and Tools. Morgan Kaufmann (2005)
8. Ge, Y., Malani, P., Qiu, Q.: Distributed Task Migration for Thermal Management in Many-Core Systems. In: Design Automation Conference. pp. 579–584 (2010)

9. Hsieh, K.Y., Lin, Y.C., Huang, C.C., Lee, J.K.: Enhancing Microkernel Performance on VLIW DSP Processors via Multiset Context Switch. Journal of Signal Processing Systems 51, pp. 257–268 (2008)
10. Iseli, C., Sanchez, E.: Spyder: A Reconfigurable VLIW Processor using FPGAs. In: FPGAs for Custom Computing Machines. pp. 17–24 (1993)
11. Jahn, J., Faruque, M.A.A., Henkel, J.: CARAT: Context-Aware Runtime Adaptive Task Migration for Multi Core Architectures. In: Design, Automation, and Test in Europe Conference. pp. 1–6 (2011)
12. Jones, A.K., Hoare, R., Kusic, D., Fazekas, J., Foster, J.: An FPGA-based VLIW Processor with Custom Hardware Execution. In: International Symposium on Field Programmable Gate Arrays. pp. 107–117 (2005)
13. Katre, K.M., Ramaprasad, H., Sarkar, A., Mueller, F.: Policies for Migration of Real-Time Tasks in Embedded Multi-Core Systems. In: Real-Time Systems Symposium. pp. 17–20 (2009)
14. Li, T., Brett, P., Hohlt, B., Knauerhase, R., McElderry, S., Hahn, S.: Operating System Support for Shared-ISA Asymmetric Multi-core Architectures. In: Workshop on the Interaction between Operating Systems and Computer Architecture. pp. 19–26 (2008)
15. Ozer, E., Sathaye, S.W., Menezes, K.N., Banerjia, S., Jennings, M.D., Conte, T.M.: A Fast Interrupt Handling Scheme for VLIW Processors. In: International Conference on Parallel Architectures and Compilation Techniques. pp. 136–141 (1998)
16. Ozturk, O., Kandemir, M., Son, S.W., Karakoy, M.: Selective Code/Data Migration for Reducing Communication Energy in Embedded MpSoC Architectures. In: Great Lakes Symposium on VLSI. pp. 386–391 (2006)
17. Richmond, M., Hitchens, M.: A New Process Migration Algorithm. ACM SIGOPS Operating Systems Review 31(1), pp. 31–42 (1997)
18. Saghir, M.A.R., El-Majzoub, M., Akl, P.: Customizing the Datapath and ISA of Soft VLIW Processors. In: International Conference on High Performance Embedded Architectures and Compilers. pp. 276–290 (2007)
19. Sarkar, A., Mueller, F., Ramaprasad, H., Mohan, S.: Push-Assisted Migration of Real-Time Tasks in Multi-Core Processors. In: Conference on Languages, Compilers, and Tools for Embedded Systems. pp. 80–89 (2009)
20. 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), pp. 1540–1552 (2008)
21. Smith, J.M.: A Survey of Process Migration Mechanisms. ACM SIGOPS Operating Systems Review 22(3), pp. 29–40 (1988)
22. Snyder, J.S., Whalley, D.B., Baker, T.P.: Fast Context Switches: Compiler and Architectural Support for Preemptive Scheduling. Microprocessors and Microsystems 19, pp. 35–42 (2000)
23. Wong, S., Anjam, F.: The Delft Reconfigurable VLIW Processor. In: International Conference on Advanced Computing and Communications. pp. pp. 242–251 (2009)
24. Wong, S., van As, T., Brown, G.: ρ-VEX: A Reconfigurable and Extensible Softcore VLIW Processor. In: International Conference on Field-Programmable Technologies. pp. 369–372 (2008)
25. Zheng, L.: A Task Migration Constrained Energy-Efficient Scheduling Algorithm for Multiprocessor Real-time Systems. In: International Conference on Wireless Communications, Networking and Mobile Computing. pp. 3055–3058 (2007)