POLYBLAZE: FROM ONE TO MANY. BRINGING THE MICROBLAZE INTO THE MULTICORE ERA WITH LINUX SMP SUPPORT

Eric Matthews, Lesley Shannon

School of Engineering Science
Simon Fraser University
ematthew@sfu.ca, lshannon@ensc.sfu.ca

Alexandra Fedorova

School of Computing Science
Simon Fraser University
fedorova@sfu.ca

ABSTRACT

Modern computing systems increasingly consist of multiple processor cores. From cell phones to datacenters, multicore computing has become the standard. At the same time, our understanding of the performance impact resource sharing has on these platforms is limited, and therefore, prevents these systems from being fully utilized. As the capacity of FPGAs has grown, they have become a viable method for emulating architecture designs as they offer increased performance and visibility into runtime behaviour compared to simulation. With future systems trending towards asymmetric and heterogeneous systems, and thus further increasing complexity, a framework that enables research in this area is highly desirable.

In this work, we present PolyBlaze: a multicore MicroBlaze based system with Linux Symmetric Multi-Processor (SMP) support on an FPGA. Starting with a single-core, Linux supported, MicroBlaze we detail the changes to the platform, both in hardware and software, required to bring Linux SMP support to the MicroBlaze. We then outline the series of tests performed on our platform to demonstrate both its stability (e.g. more than two weeks of up time) and scalability (up to eight cores on an FPGA, with resource usage increasing linearly with the number of cores).

1. INTRODUCTION

Computing system complexity is growing rapidly due to the increasing adoption of multicore systems and the trend toward asymmetric and heterogeneous platforms. Unfortunately, our ability to optimize runtime scheduling on these systems has not advanced enough to deal with the added complexity. This has lead to increasingly variable performance on these systems, where application performance can be degraded by as much as 150% [1]. Without greater visibility into the system, these runtime interactions cannot be predicted, and a variety of performance characteristics, from latency to throughput, can be negatively impacted.

Current commercial systems, which are only now starting to include more hardware counters, still focus on microarchitectural details. This tends to result in the ability to detect symptoms (e.g. high cache miss rates), but does not provide an easy means to diagnose the source of the problem (e.g. cache access patterns). Current multicore systems are limited by their fixed nature and do not allow us the flexibility to tune different parameters or provide us with a means of examining future asymmetric and heterogeneous systems.

There is a wide range of simulators that model, to varying levels of detail, different parts of the system hierarchy [2, 3]. For performance reasons, any given simulator will typically model only a specific area in detail; however, few target cycle level accuracy. Simulators also typically face performance scaling problems when the breadth and depth of information collected is expanded as all software data collection has some level of overhead.

Ideally, the target platform for enabling systems research would be a highly configurable and scalable multicore platform with SMP support for a modern Operating System (OS) such as Linux. Given the configurable nature of an FPGA's fabric and the acceleration it provides over software simulation, both researchers and commercial vendors are now using FPGAs to emulate processors for prototyping and research [4, 5, 6, 7]. However, the existing multicore research frameworks that utilize FPGAs either do not run a standard OS, they do not scale well, or they are too costly due to resource usage, to be a general research platform. As such, the focus of this paper is to create a framework for systems research that meets all of these criteria.

In this paper, we present PolyBlaze, a multicore system featuring multiple MicroBlazes and SMP OS support through the Linux 2.6.37 kernel. Starting with the existing support for a single MicroBlaze running Linux [8], we describe the principle changes to the hardware and software required to enable a Linux-based multicore system derived from the MicroBlaze platform.

The specific contributions of this paper include:

- hardware extensions to the MicroBlaze processor to support SMP;
- Linux support for the multicore MicroBlaze system; and
- a scalable Timer and Interrupt Controller for the multicore MicroBlaze system

The remainder of this paper is organized as follows. Section 2 discusses the related work in the area of multicore platforms and FPGAs as a contrast to the overview of the PolyBlaze platform discussed in Section 3. The transformation of the MicroBlaze into the PolyBlaze platform is described in Section 4 and a system analysis is provided in Section 5. Finally, Section 6 concludes the paper and summarizes future work.

2. BACKGROUND

The increasing capacity of FPGAs has fueled their growth in areas of research such as the simulation of traditional architectures [7], as well as research into soft-processor architectures [9]. Multicore processor architecture research projects utilizing FPGAs include: the Research Accelerator for Multiple Processors (RAMP) [10] project, which uses multiple lightweight processors on tens and hundreds of FPGAs [11]; the Beehive Project [12], which enables multiple lightweight processors on a single FPGA; the Hthreads project [13], which provides support for threads in hardware and software along with a hardware based OS; a xilkernel based multicore MicroBlaze system [14]; and emulation platforms used during the design of commercial multicore processors [4, 5, 6]. While the commercial emulation platforms do support an OS, they are proprietary and their cores are too large to easily instantiate multicore systems on platforms affordable to researchers. By contrast, the RAMP, Beehive, Hthreads and xilkernel multicore MicroBlaze platforms all use lightweight processing cores, enabling multicore systems to be instantiated on affordable research platforms. However, none of these systems support an OS with virtual memory as is desirable for most systems research.

More recently, the emergence of platforms such as the OpenSPARCT1 [15] and LEON3 [16] has enabled new approaches into systems research. Their open-source nature enables visibility into all details of the system, providing access to cycle accurate data, while still being able to run a modern OS on the system, many times faster than software simulators. However, each OpenSPARC processor core is large enough to map to a separate Virtex 5 110LXT and the memory controller is emulated using firmware on a MicroBlaze. Thus, scaling the system beyond one core not only requires multiple FPGA boards due to the size of the processor, but the MicroBlaze-emulated memory controller significantly limits the system's scalability as all memory operations are handled in MicroBlaze firmware, which makes the MicroBlaze the bottleneck in system performance. The LEON3 is another SPARC-based platform that supports a multi-core setup, however, there is limited memory controller support for different boards. PolyBlaze allows the user to instantiate up to eight cores on a Virtex 5 110LXT and, by using Xilinx's Multi-Port Memory Controller, maps more easily to different Xilinx research boards.

In addition to the OpenSPARC and LEON3 platforms, we are aware of only two other soft-processors with full support for Linux 2.6 or later. They are the NIOS II [17] and the MicroBlaze [18]. While both processors support Linux in a single-core configuration, the NIOS II's support is more recent compared to the MicroBlaze, which has supported Linux since the 2.6.30 kernel.

3. POLYBLAZE PLATFORM

A major benefit of choosing the MicroBlaze for our multicore system is that we immediately benefit from the high configurability of the processor design. This will facilitate future work into asymmetric systems, allowing for configurations with different size caches and varied arithmetic units. For example, a mix of processors with and without Floating Point Units (FPUs). While transforming the MicroBlaze into a PolyBlaze processing core requires some architectural changes, none of these changes restrict the existing configurability of the MicroBlaze.

As PolyBlaze will be used for systems research, a more complex memory hierarchy is desired than the direct mapped caches of the MicroBlaze. However, adapting the MicroBlaze's architecture and the existing Linux kernel support to support multicore configurations does not require a complex cache hierarchy. Therefore, a detailed discussion of cache configurations and hierarchies is left as future work due to space limitations.

Figure 1 illustrates PolyBlaze, our multicore MicroBlaze system on an FPGA. Modules highlighted in blue indicate the hardware components that have been designed/redesigned for our framework. The *Hardware Profiler* is used to validate the system and evaluate its performance and is not required for normal system operations. The system consists of one to eight MicroBlaze cores, along with support for timers, interrupts and extended support for atomic instructions through the Lock Arbiter[1]. Section 4 will detail the changes in implementation required to achieve this transformation.

4. BRINGING MULTICORE SUPPORT TO THE MICROBLAZE PROCESSOR

Adding a second MicroBlaze processor to a system is a straightforward process; however, the resulting system is still far from having the necessary infrastructure to support booting an SMP OS. To achieve this requires many changes both to the OS support and the hardware platform. Some support can be provided with minimal changes, such as processor identification, which can be implemented through the existing Processor Version Registers (PVR). Other support,

[1]Due to the proprietary nature of the MicroBlaze, we are looking into the possibility of making available an encrypted version of the PolyBlaze framework to other researchers.

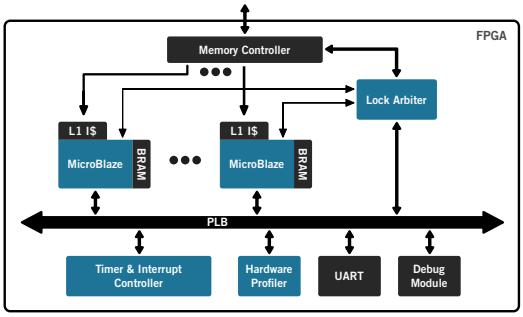


Fig. 1. PolyBlaze: the Multicore MicroBlaze Platform

requires more extensive changes as is the case for interrupts, timers, atomic operations and Memory Management Unit (MMU) support.

4.1. Interrupts

For a single processor system, interrupt support is relatively straightforward. To support multiple interrupts, a standalone interrupt controller is used to abstract each interrupt's type and multiplex the multiple interrupts to the MicroBlaze's single interrupt input (IRQ pin). This solution is insufficient for multicore systems as we may want interrupts to be served only by a single (specific) processor or, conversely, allow any processor to be the recipient of an interrupt. Further complicating matters is the need for asynchronous communication between processors, which is typically provided through Inter-Processor Interrupts (IPIs).

Although the interrupt controller provided by Xilinx (the LogiCORE IP XPS Interrupt Controller v2.01a [19]), has support for handling multiple interrupts and types of interrupts (level/edge), it only supports a single interrupt output. Simply duplicating the interrupt controller per processor is not a viable option, as in addition to the poor scaling of this approach, it would also complicate software handling of interrupts and still would not provide a mechanism for IPIs.

4.2. Timers

To support the dynamic clock tick timekeeping functionality in the Linux kernel [20], a free-running counter and per processor timers are required. For the single processor system, this role is adequately served by the Xilinx LogiCORE IP XPS Timer/Counter peripheral [21]. The timer peripheral supplies a 32-bit free-running counter and a decrementer, with an interrupt connected to the interrupt controller. While this is sufficient for a single core system, it does not scale well with multiple processors for a few reasons. First of all, the decrementers are used on a per processor basis, and while the timer peripheral supports two timers they share an interrupt output. In addition, each timer would consume a system wide interrupt pin on the interrupt controller, but would only ever be sent to a single processor which, is an inefficient use of resources.

4.3. PolyBlaze Timer and Interrupt Controller

As discussed in the previous section, an integrated approach to timers and interrupts provides for better system scalability, therefore, in PolyBlaze they are combined into a single Timer and Interrupt Controller (TIC) peripheral as shown in Figure 2.

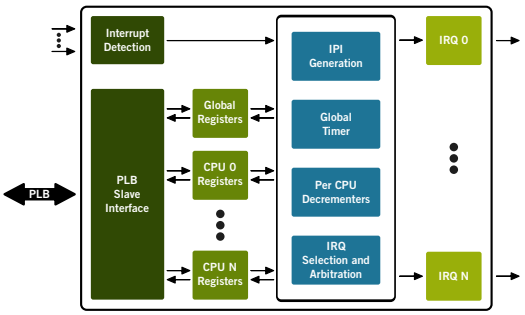


Fig. 2. High-level Timer and Interrupt Controller Diagram

The TIC provides each processor with a dedicated 32-bit decrementer and a per processor interrupt used to schedule wake-ups in the kernel. There is also an independent, global, free-running 64-bit counter used for timekeeping purposes and a configurable number of IPIs. Like the Xilinx interrupt controller, interrupt priority is fixed by connection order, with IPIs and timers having higher priority than all other system interrupts. A master enable exists for each interrupt (represented in the global registers block in Figure 2), as well as on a per processor basis (the CPU X register blocks), thus support for both distributing interrupts and fixed interrupt assignment exists. On every cycle, the selection logic determines the highest priority interrupt as well as which processor(s) have this interrupt priority level enabled. The interrupt is then assigned to a processor that is not already servicing an interrupt based on round-robin arbitration.

Just as timer and interrupt support are merged in the PolyBlaze TIC, support for the timers and interrupts are also merged in the kernel. Interrupt support has been simplified within the kernel, with custom paths for edge and level interrupts replaced by a simple "acknowledge on completion," as the interrupt controller effectively hides the type of interrupt from the MicroBlaze. In moving from a 32-bit to a 64-bit free-running counter, the unreliable software approach to extending the counter to 64-bits (which did not work at certain operation frequencies) has been removed. However, as the MicroBlaze is a 32-bit processor this counter must now be snapshotted prior to reading and requires the protection of a global spinlock to ensure that the snapshotting and reads are performed by only one processor at a time.

4.4. Exception Handling

With support in place for interrupts and timers, another area of operation that differs between the single-core and multicore system is exception handling. When an exception or interrupt occurs during normal operation in the kernel some registers are needed for temporary processing. However, the MicroBlaze has no registers available for these operations. In the single core implementation, some registers are stored into fixed locations in memory using immediate mode addressing. Since the addresses generated through immediate mode addressing would not be unique for multiple cores this approach will not scale. To address this, we added a new set of registers called General Purpose Special Purpose Registers (GPSPRs) to the MicroBlaze, similar to those found in the PowerPC [22].

Support for these new registers is integrated into the MicroBlaze's Move To Special Purpose Register (MTS) and Move From Special Purpose Register (MFS) instructions. The modification to the Instruction Set Architecture (ISA) is shown in Figure 3. The current implementation supports four GPSPRs, two of which are currently needed for exception handling in the kernel; however the number is configurable and can readily be increased or reduced to meet the future needs of the PolyBlaze system.

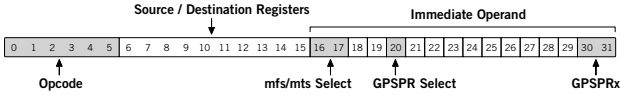


Fig. 3. Changes to the MFS/MTS Instruction Format to Support GPSPRs

4.4.1. Linux Support

Inside the kernel, the previously existing reads and writes to the fixed memory locations are replaced with local storage operations utilizing the GPSPRs. At present, this is accomplished with the insertion of the machine code for these instructions. Support has been implemented in Xilinx's version of the GNU assembler to provide an assembly level interface for these instructions, however, PetaLinux currently uses a different version of the assembler which is not compatible with these changes.

4.5. Atomic Operations

The final significant piece of infrastructure needed for multicore support is support for atomic operations. Despite the existence of a pair of conditional load/store instructions in the MicroBlaze ISA [18], the existing kernel implementation did not use these instructions for implementing synchronization primitives. Instead, to perform an atomic operation interrupts are disabled over the "protected" region. In a single-core system, this is sufficient to make an operation atomic as the operation cannot be interrupted. However, this also means that no form of kernel preemption can be supported, which can increase system latency. With the use of the MicroBlaze's Load Word Exclusive (LWX) and Store Word Exclusive (SWX) instructions, the number of circumstances where interrupts need to be disabled can be lessened and kernel preemption on even single-core systems can be enabled.

4.5.1. LWX/SWX Behaviour

To perform an atomic operation with the LWX/SWX instructions, first a conditional load is performed, which sets a reservation bit internal to the processor [18]. If this bit is still set when the next conditional store instruction is executed (to any address), the store will proceed. If the bit has been cleared, by events such as interrupts or exceptions, the store will be aborted [18]. While this design is suitable for a single-core system, it does not provide any guarantee that the operation will be atomic if there are other devices that could be modifying those memory locations. As such, a multicore system requires a more flexible approach.

In a multicore system, in addition to the possibility of there being multiple concurrent memory requests in progress, it is also possible for the atomic operations to be targeting different memory addresses. Therefore, a system that is address based will allow for greater scalability. For our system, we have chosen to implement a conditional load/store approach with a different set of semantics based on the ordering of the stores across all processors in the system.

Our system provides a centralized location (the lock arbiter in Figure 1) that acts as the synchronization point for all reservations in the system. While all data requests pass through the arbiter, additional per-processor signals are utilized to communicate whether a request is atomic and whether the request was put through to memory. When a processor performs a conditional load operation, a reservation bit is set for that processor and the address is stored. Note, there is no restriction on multiple processors having reservation bits set for the same address. Any store to that address after that point will clear the reservation bit for any processor with a reservation for that address. If another processor then attempts a conditional store operation to the same address, the store will be terminated upon reaching the arbiter and will not be committed to memory.

Inside the Linux kernel, there are many types of atomic primitives from spinlocks to atomic versions of basic operations such as add/subtract and bit manipulations. As was previously discussed, the existing MicroBlaze Linux implementation did not provide custom implementations for any of these operations as it simply disabled interrupts for atomic operations. For PolyBlaze, we wrote the custom as-

sembly functions required to implement the full set of these operations.

4.6. MMU

As the MicroBlaze utilizes a fully software managed MMU, similar to the PowerPC 405 [22], hardware changes are not required to support a symmetric multicore system. Instead, only updates to the kernel are required; software contexts must now be tracked across multiple MMUs and managed appropriately. Other members of the PowerPC 4xx/8xx series processors [22] do support SMP and have fully software managed MMUs, and it was this support that provided the base for the SMP MMU support for our PolyBlaze platform; merging it into the existing MicroBlaze support and modifying it to support MicroBlaze specific behaviours where necessary.

4.7. Booting Linux on the PolyBlaze

In a system with multiple processors, it is important that the secondary processors be isolated from the system until they are brought online. This can be readily achieved by having the MicroBlaze run in firmware, located entirely in its local BRAM, until brought online. Since the secondary processor is executing code in its own BRAM, the only mechanism available to wake the processor is an IPI. As such, part of the systems firmware requires the secondary processors to enable a single IPI and then sleep until they receive that interrupt. Once the IPI is received, the processors jump to the base address of the kernel and begin their initialization process.

5. SYSTEM VALIDATION

To demonstrate both the scalability and stability of the PolyBlaze system, an 8-core configuration, as shown in Figure 1, is used for our testing. Single-core through eight-core configurations are tested on the same 8 core build as inactive processors do not access any shared resources. All MicroBlazes are identical and configured with the necessary parameters to support Linux including: the MMU, exception support and, additionally, a 4KB instruction cache. The FPU is not included (so as to lessen the demands on placement and routing). We have also excluded data caches, in an effort to avoid duplicating work, as the existing direct-mapped caches would need significant work to properly support locks and cache coherency and will be replaced in the future (see Section 3).

We consider two different stress tests for validating the stability of the system: the first is to repeatedly boot up the system; and the second is to place the system under heavy load for a prolonged period of time. Resource usage is presented for both the Timer and Interrupt Controller and the changes to the MicroBlaze to provide insight into the scalability of the system, along with data collected by a hardware profiler that shows the impact of bus/memory controller contention in the system. Finally, we compare the performance of a similarly configured MicroBlaze single-core system to a PolyBlaze single-core system to demonstrate that the system's performance has not been negatively impacted.

5.1. Stress Tests

Our first stress test was to repeatedly boot-up the 8-core system. Each test begins with the configuration of the FPGA, followed by the copying of the Linux system image into DDR RAM and then finally, by the boot-up of the system. To ensure that the system is still alive and responsive after boot-up, the login process is scripted and the contents of `/proc/interrupts` to the terminal. In our testing, this sequence of steps was repeated well over 200 times (251), all of which were successful boot ups. While such a test does not guarantee there are no issues with the system, it is our experience that even rare corner cases in lock behavior, interrupts, and virtual memory support are hit frequently during boot up and even a few successful boot-ups is indicative of the system being stable.

To complement our first stress test, our second test repeatedly launches sixteen instances of the Dhrystone [23] benchmark after booting to place the system under heavy load with frequent context switching. The selection of the benchmark is arbitrary as, without data caches, even traditional compute intensive workloads become memory intensive. After more than 14 days the system was still up and running as can be seen by the console output from /proc/uptime in Figure 4. The first number is the system time under load (in seconds) and the second is the aggregate sum of processor idle time (also in seconds). Also included is the output of /proc/interrupts, which provides the total interrupt counts for all interrupts across all processors in the system.

5.2. Resource Usage

An analysis of the platform would not be complete without reporting on the hardware cost of bringing SMP support to the MicroBlaze processor.

The changes to the MicroBlaze include the addition of four GPSPRs and the changes to support external atomic instruction synchronization. Together, these two changes incur an overhead of an additional 64 LUTs (2%) and 204 FFs (8%) compared to an unmodified MicroBlaze. The changes to the MicroBlaze also did not affect the critical path of the design and it is still capable of operating at 125MHz on our platform.

```
~ # cat /proc/uptime
1319997.79 4271.02
~ # cat /proc/interrupts
           CPU0       CPU1       CPU2       CPU3       CPU4       CPU5       CPU6       CPU7
0:         28         37         24          9         25         19         25     202035   per_cpu  ipi reschedule
1:        335        361        333        160        365        455        251        308   per_cpu  ipi call function
2:         31         22         24        188         18         18         16         14   per_cpu  ipi call function single
4:  131948134  131946708  131947585  131945393  131947445  131946235  131946852  131945275   per_cpu  timer
5:        387        289        229        240        189        498        510        588   per_cpu  serial
```

Fig. 4. Output of /proc/uptime and /proc/interrutps after more than 14 days of uptime

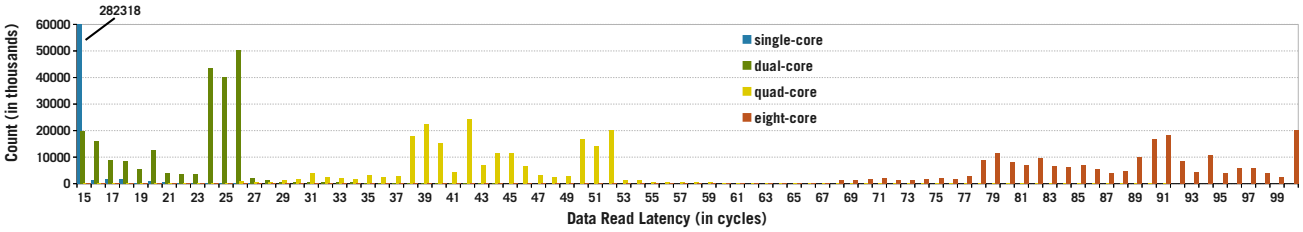
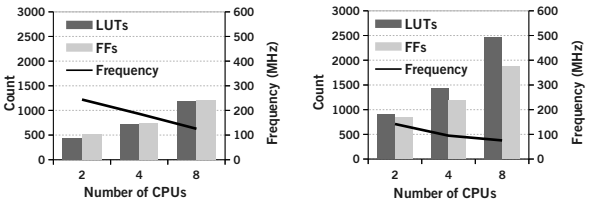


Fig. 5. Memory Bus Read Latency

5.2.1. Timer and Interrupt Controller

Our current design for the Timer and Interrupt Controller is intended to be scalable up to 8 cores at 100MHz. Figure 6 illustrates how the resource usage and frequency of the controller scales across the number of cores for, a) the minimum number of interrupts (4 IPIs, one timer, and one external), and b) the maximum number of interrupts supported (which is 32). For the 8-core system, after 12 interrupts the operating frequency drops below 100MHz to 91MHz. Currently the critical path in the design is the interrupt selection, arbitration and assignment operation, which occurs in a single cycle. The design could be readily modified to pipeline this process by splitting up the interrupt selection and arbitration phases without otherwise impacting the behaviour or operation of the TIC. However, at present we do not have a system with these requirements so this change will implemented in the future if it becomes necessary.



(a) 6 Interrupts (b) 32 Interrupts

Fig. 6. TIC Scalability Across two, four and eight CPUs

5.3. Memory Latency

In this test, we utilize a hardware profiler to collect data bus read latency measurements of the system at runtime. Across 1, 2, 4 and 8-core configurations, 8 instances of the Dhrystone benchmark were launched and then latency measurements were collected for a two minute runtime window, the results of which are presented in Figure 5. The samples begin at 14 cycles, which is the minimum memory latency in our system. Without data caches, we expect to see a significant increase in read latency as the number of cores is increased as is confirmed in the results in Figure 5. Average memory latency for one core is approximately 15 cycles and is, by far, the largest single bin for the single-core configuration extending well beyond the top of the graph. For two cores, the average latency increases to approximately 22 cycles. We now see a spread of the latencies as there is contention for the bus and memory controller. At 4 cores the latency doubles to around 44 cycles on average, and doubles again to about 87 cycles for 8 cores. For the larger systems, it is quite likely that there will always be multiple cores with outstanding memory requests and thus the distribution of latencies both spreads out and shifts to the right.

5.4. Single-core Performance

In our final test, we create a single-core MicroBlaze system with the same configuration options as our PolyBlaze cores (such as cache configuration, multiplier, divider, etc.). On each system, we run two instances of the Dhrystone benchmark (to ensure some context switching takes place) for one million runs. The Dhrystone's per second of the original MicroBlaze averaged 13466 across the runs, whereas the PolyBlaze core averaged 14106.

The 5% speedup is due to changes in the hardware, and not due to any kernel changes as the system reports less than a hundredth of a percent of runtime is from the kernel during the test. The source of the speed up is found in the Lock Arbiter, which passes memory requests to the memory controller with lower latency in some cases (most likely during consecutive writes) compared to the original PLB-to-

memory controller interface. Therefore, under normal operating conditions, we have observed no negative impact on runtime performance of the system as was expected. With the changes to the Linux kernel, we expect that with preemption support, throughput could decrease slightly, but only if the workload is primarily in the kernel itself, which is rare.

6. CONCLUSIONS AND FUTURE WORK

Today's multicore systems provide many challenges to the OS; however it is our goal to help reduce one of them, a lack of understanding into the system's actual runtime behaviour. In this paper, we presented our work in bringing multicore support to the MicroBlaze processor. This work includes details about the modifications that were necessary to achieve SMP support, both in hardware and in software. Through careful consideration of design parameters we were able to ensure that our design is scalable which we then verified through our test cases, demonstrating both the system's stability and scalability. Using this platform as a foundation for future work, our immediate focus will be to provide: a configurable memory hierarchy for the platform; conduct investigations into asymmetric and heterogeneous system configurations; and develop scheduling algorithms that can monitor runtime behavior and adapt to the existing workload.

7. ACKNOWLEDGEMENTS

The authors would like to thank Xilinx Inc, PetaLogix and the Natural Sciences and Engineering Research Council of Canada (NSERC) for supplying resources and/or funding for this project.

8. REFERENCES

[1] S. Zhuravlev *et al.*, "Addressing Contention on Multicore Processors via Scheduling," in *Int'l Conf. on Architectural Support for Programming Languages and OSes*, 2010.

[2] P. S. Magnusson *et al.*, "Simics: A full system simulation platform," *Computer*, vol. 35, no. 2, pp. 50–58, 2002.

[3] T. Austin, E. Larson, and D. Ernst, "Simplescalar: An infrastructure for computer system modeling," *Computer*, vol. 35, no. 2, pp. 59–67, 2002.

[4] P. H. Wang *et al.*, "Intel® atom processor core made fpga-synthesizable," in *The ACM/SIGDA Int'l Symp. on FPGAs*, 2009, pp. 209–218.

[5] G. Schelle *et al.*, "Intel nehalem processor core made fpga synthesizable," in *The ACM/SIGDA Int'l Symp. on FPGAs*, 2010, pp. 3–12.

[6] S. Asaad *et al.*, "A cycle-accurate, cycle-reproducible multi-fpga system for accelerating multi-core processor simulation," in *Proc. of the ACM/SIGDA Int'l symposium on FPGAs*, ser. FPGA '12. ACM, 2012, pp. 153–162.

[7] D. Chiou *et al.*, "Fpga-accelerated simulation technologies (fast): Fast, full-system, cycle-accurate simulators," in *Proc. of the 40th Annual IEEE/ACM Int'l Symp. on Microarchitecture*, 2007, pp. 249–261.

[8] "PetaLinux System Development Kit PetaLogix." [Online]. Available: www.petalogix.com/products/petalinux

[9] P. Yiannacouras, J. Rose, and J. G. Steffan, "The microarchitecture of fpga-based soft processors," in *2005 Int'l Conf. on Compilers, architectures and synthesis for embedded systems*, 2005, pp. 202–212.

[10] "RAMP - Research Accelerator for Multiple Processors." [Online]. Available: ramp.eecs.berkeley.edu/

[11] D. Burke *et al.*, "Ramp blue: Implementation of a multicore 1000 processor fpga system," in *Reconfigurable Systems Summer Institute*, Urbana, IL, 2008.

[12] Chuck Thacker, MSR Silicon Valley, *Beehive: A manycore computer for FPGAs (v6)*, 2010. [Online]. Available: research.microsoft.com/en-us/um/people/birrell/beehive/BeehiveV6.pdf

[13] J. Agron and D. Andrews, "Building heterogeneous reconfigurable systems with a hardware microkernel," in *Proc. of the 7th IEEE/ACM Int'l Conf on Hw/Sw codesign and system synthesis*, 2009, pp. 393–402.

[14] P. Huerta, J. Castillo, C. Sanchez, and J. Martinez, "Operating system for symmetric multiprocessors on fpga," in *Reconfigurable Computing and FPGAs, 2008. ReConFig '08. Int'l Conf on*, dec. 2008, pp. 157 –162.

[15] "OpenSPARC FPGA." [Online]. Available: www.opensparc.net/fpga/index.html

[16] *GRLIB IP Core User's Manual*. [Online]. Available: www.gaisler.com/products/grlib/grip.pdf

[17] Altera Inc., *(2011, May) The NIOS Soft CPU Family*. [Online]. Available: http://www.altera.com/literature/hb/nios2/n2cpu_nii5v1.pdf

[18] Xilinx Inc., *MicroBlaze Processor Reference Guide*. [Online]. Available: www.xilinx.com/support/documentation/sw_manuals/xilinx12_4/mb_ref_guide.pdf

[19] ——, *LogiCORE IP XPS Interrupt Controller (v2.01a)*. [Online]. Available: www.xilinx.com/support/documentation/ip_documentation/xps_intc.pdf

[20] "Clockevents and dyntick [LWN.net]." [Online]. Available: http://lwn.net/Articles/223185/

[21] Xilinx Inc., *LogiCORE IP XPS Timer/Counter (v1.02a)*. [Online]. Available: www.xilinx.com/support/documentation/ip_documentation/xps_timer.pdf

[22] IBM Corp., *PPC405Fx Embedded Processor Core Users Manual*. [Online]. Available: www-01.ibm.com/chips/techlib/techlib.nsf/techdocs/D060DB54BD4DC4F2872569D2004A30D6/$file/ppc405fx_um.pdf

[23] R. P. Weicker, "Dhrystone: a synthetic systems programming benchmark," *Commun. ACM*, vol. 27, pp. 1013–1030, October 1984.