A Configurable Framework for Investigating Workload Execution

Eric Matthews[1], Lesley Shannon[1], Alexandra Fedorova[2]

School of Engineering Science[1], School of Computing Science[2], Simon Fraser University
8888 University Drive, Burnaby, BC Canada V5A 1S6
{ematthew,lshannon,fedorova}@sfu.ca

***Abstract*—Processor systems contain a limited number of hardware counters that provide some visibility for certain types of interactions, but do not support sophisticated analysis due to limited resources. By contrast, system software simulators provide multidimensional runtime data, but slowdown application execution, often resulting in an inaccurate picture of hardware/software interactions. The ideal solution to this problem is to create a dedicated hardware unit to "watch" the processor for these types of behaviours. In this paper, we present a hardware framework that leverages an FPGA's reconfigurable fabric to investigate of workload execution behaviours on processors using a hArdware-Based Analyzer for the Characterization of User Software (ABACUS). ABACUS is currently able to interface with the LEON3 processor using 1367 FFs, 1504 LUTs and 1 Block RAM on a Virtex 2Pro running at 144MHz.**

I. Introduction

As processor systems grow more complex, understanding workload performance characteristics becomes increasingly difficult. To obtain information on the run-time behaviour of threads, visibility into the processor's architecture is needed to monitor workload interactions. However, the greatest benefactors of this type of information are software designers of high performance applications (e.g. games). Due to performance and complexity issues, most simulators choose to focus on a particular level in the system hierarchy. Any simulator implemented in software that does provide this combined level of visibility will likely run much slower, a drawback when trying to understand thread interactions to improve software execution. Thus, a better way to gain the desired visibility into these types of behaviours is to create additional hardware to "watch" the processor for these types of behaviours.

In this paper, *we present a hardware framework for the investigation of workload execution behaviour on a processor using a FPGA* called the hArdware-Based Analyzer for the Characterization of User Software (ABACUS). ABACUS is a generic, configurable analyzer that allows designers to create and instantiate new profiling units to monitor specific behaviours to better understand thread interactions during execution and how this impacts performance. ABACUS is connected to the processor by only the minimal set of signals needed to collect the desired metrics; all profiling units are memory mapped so that users can change their configuration at run time to analyze the workload. This information can be used to improve the performance of software workload execution. The specific contributions of this paper include:

- An overview of the architecture of ABACUS;
- Examples of possible profiling units; and
- A set of experimental data demonstrating that ABACUS can be used to obtain the same types of results as Simics [1], a popular software simulator for researchers.

The remainder of this paper is organized as follows. Section II discusses the related work and Section III describes ABACUS's architecture as well as the existing profiling units. Section IV outlines the current platform into which we have integrated and ABACUS. the results obtained using ABACUS in contrast to Simics. Finally, Section V concludes the paper and summarizes future work.

II. Background

The previous work related to ABACUS falls into two main categories: traditional methods of obtaining useful data on thread behaviour (i.e. software simulators and hardware counters); and using FPGAs for processor architecture research and on-chip profiling.

A. Traditional Approaches to Understanding Workload Behaviour

To understand the behaviour of threads on processor systems, researchers traditionally use three methods: software profiling, hardware counters and software simulation. Software profiling relies on code instrumentation to associate certain hardware performance events with specific code segments (e.g. oprofile, gprof, Intel VTune, Sun Studio collect/analyze) or data structures [2]. Software profiling is conventionally regarded as an off-line method, because it slows down application execution. An alternative to software profiling is to use hardware performance counters directly [3], [4]. They can be programmed and queried without a noticeable slowdown of the application. Although hardware counters can collect some very useful information, their microarchitecture dependent nature and limited numbers prevents their adoption as a tool across platforms. In addition, hardware counters are designed to focus on predefined single events, limiting the scope of information they provide the exploration of new ideas. Software simulation (e.g. Simics [1], SimpleScalar [5]) as well as binary instrumentation tools (e.g. Pin [6]) offer greater visibility into microarchitectural interactions between the software and the hardware, such as modelling latencies and contention for shared resources. Unfortunately, they impose

978-1-4244-8983-1/10/$26.00 ©2010 IEEE

orders of magnitude of slowdown on the application especially when attempting to simulate all the fine-grained details of the microarchitecture.

The objective for the ABACUS framework is to provide the best of both worlds: the richness and accuracy of the information that can be obtained from simulators, but at the real-time speed that can be delivered by hardware counters. Furthermore, by utilizing an FPGA's reconfigurability and the ABACUS framework, we can also easily investigate new types of profiling units and thread workload interactions.

B. *Using FPGAs for Processor Architecture Research*

Modern FPGAs are able to implement complex systems, including Systems-on-Chip. This has led to architectural research of both soft processors [7] for FPGAs as well as accelerating the simulation of more traditional architectures using FPGAs [8]; however, both these projects focus on the ISA of single core processors, whereas ABACUS is ISA independent. Comparable to the hardware counters available in commercial processors, researchers designing soft processors for FPGAs have also integrated dedicated profiling functionality into their architectures to investigate specific behaviours and interactions for multicore systems [9]. Although all the above works leverage FPGAs to include software profiling hardware circuitry, unlike ABACUS, they do not provide a generic framework for designing new profiling units or a user-level interface for run time reconfiguration.

III. ABACUS

ABACUS is designed to facilitate investigations into thread behaviour, specifically, to collect multi-dimensional metrics using a configurable *collection of profiling units*. ABACUS is external to the processor core, snooping processor and system infrastructure signals to collect data on different threads for various metrics. In designing ABACUS, we imposed a set of constraints to facilitate its integration with a variety of processor architectures. Specifically, ABACUS should:

- Be external to the processor core, and not integrated.
- Port easily to different processor architectures.
- Provide microarchitecture independent metrics, and
- Facilitate the inclusion of new profiling units.

Our main objective is that ABACUS remains a separate entity, not limited to integration with a specific microarchitecture. As such, we ensure that, outside of the system communication network interface, it connects to the minimal set of signals required to collect its metrics. This also reduces the effort needed to integrate ABACUS into new systems. To achieve portability, a standard interface is used, which also implies that the data it collects must focus on microarchitecture independent metrics. In addition, users should easily be able to add new profiling units to the ABACUS framework and instantiate any subset of the existing profile units to customize ABACUS for their purpose. The remainder of this section describes both the framework of the ABACUS architecture in greater detail and some example profiling units that have been created thus far.

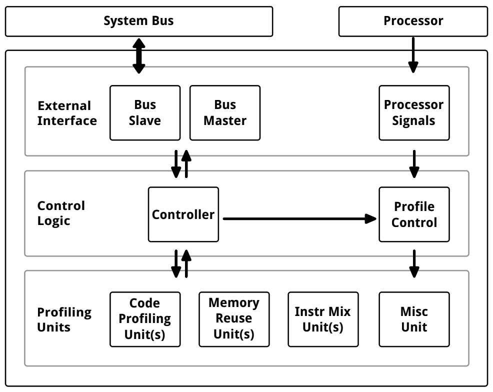


Fig. 1. High-Level ABACUS Block Diagram

A. *Architectural Overview*

As shown in Figure 1, ABACUS has a hierarchical design with three layers. The top layer is the *External Interface*, which consists of the logic required to interface it with the system. The middle layer implements the *Control Logic* provides a layer of support for controlling ABACUS through system software and provides a standard interface to the system. Finally, the lower layer is comprised of the *Profiling Units* that collect the various metrics. ABACUS's design has been structured to facilitate integration such that design changes are localized to the External Interface. Integration of ABACUS into a new system only requires that ABACUS have its address space mapped into the system and that the External Interface be adapted to interface with the system's communication network.

B. *Profiling Units*

The bottom layer is comprised of the *Profiling Units*. Each metric is implemented as a separate profiling unit, allowing the user to incorporate new profiling units into the existing framework and to instantiate only the desired profiling units into their design.

For each unit, various parameters can be configured at hardware instantiation time to match the characteristics of the system. In addition, some profile units may also have parameters that are runtime configurable. The visibility required into the processor or system-level architecture by each unit is dictated by the specific metric being profiled (e.g. Instruction Register, Program Counter, bus signals, etc.). Some examples of profiling units currently implemented in ABACUS are described in the following section.

1) Example Profiling Units: The initial set of profiling units created for ABACUS has primarily focused on replicating the types of data that can be obtained using system software simulators (e.g. Trace Profiling, Reuse Distance, and Instruction Mixes). This has facilitated both the initial verification of our design as well as developing an understanding of the tradeoffs of profiling on a software simulation platform versus an FPGA emulation platform.As previously stated, these are only examples of possible profiling units that have been created thus far within the ABACUS framework. The following paragraphs provide further details on a subset of the existing profiling units.

Instruction Mix: The Instruction Mix unit enables analysis of the type of instruction workload a thread is executing. It takes the Instruction Register (IR) as an input and maps the opcode to a runtime configurable look-up table stored in a BRAM to determine which counter in a set to update. In this way, instructions can be grouped together into any possible classification scheme. Although this unit is conceptually microarchitecture independent, simple alterations are required to adapt its implementation to new Instruction Set Architectures (ISAs), such as the width of the IR. While existing hardware counters can be configured to count instructions of a particular type, they must rely on sampling to estimate the entire mix; conversely, software simulators can keep track of the entire mix, but run very slowly. ABACUS can keep track of all the instructions and compute the entire mix at the operating frequency of the clock.

Reuse Distance: The Memory Reuse Distance Unit is designed to work with a set-associative cache that implements a Least Recently Used (LRU) replacement policy as the LRU stack contains a measure of the locality of memory accesses. Specifically, reuse distance profiles indicate how well a program reuses its cached memory; this can be used to analyze the locality of references as well as the nature of contention for the cache when multiple programs run on cores sharing a cache [10]. Currently, this unit relies on the existence of a LRU stack, however, there are also means of estimating the reuse distance in hardware with low overhead [11].

IV. Implementation and Results

In this section, we discuss how ABACUS is integrated into the LEON3 platform and describe the selected SPARC ISA model implemented using Simics. We then present the resource usage of ABACUS and an example reuse distance experiment run on the LEON3 system. These results are contrasted with the results from a Simics system simulation for the same single threaded workloads, highlighting the differences when an exact match for the ISA and kernel distribution cannot be used.

A. LEON3 and ABACUS vs Simics

The LEON3 soft processor [12] implements the SPARC v8 ISA [13] and offers a set of configuration parameters. The processor operates at 50MHZ on the XUP Virtex 2Pro board [14], has 256MB of DDR RAM, uses a networked disk for the filesystem, and runs the Debian Etch Linux Distribution with Linux kernel 2.6.21. A high level overview of the system is given in Figure 2. We use the reuse distance, in conjunction with the LEON3 system, to generate experimental data that can also be obtained using our Simics model in Section IV-C. The closest system supported on our Simics platform is an UltraSPARC II processor that supports the SPARC v9 ISA [15], which is backwards compatible with the v8 ISA while including additional support for a 64-bit datapath. The kernel is the 2.6.18 release and is built for the v9 ISA, however, all benchmarks are built for the v8 ISA.

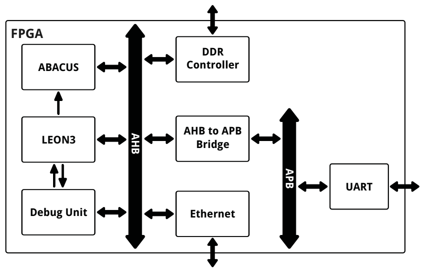


Fig. 2. ABACUS integrated into the LEON3 Platform

TABLE I
ABACUS Resource Usage and Operating Frequency

Component	Freq (MHz)	FFs	LUTs	BRAMs
Memory Reuse Unit	223	120	150	-
Instruction Mix Unit	170	248	230	1
Code Profiling Unit	257	646	554	-
Controller	238	86	174	-
AHB Master	299	81	76	-
AHB Slave	205	60	57	-

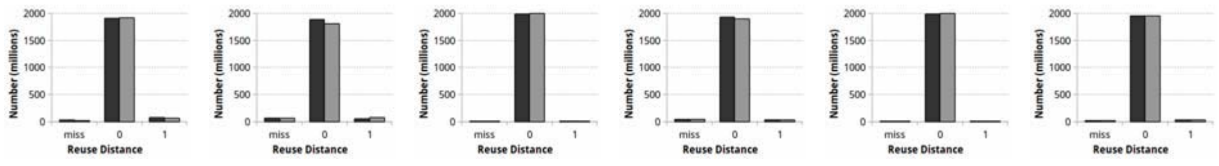
B. Resource Usage and Operation

In this section, we summarize the resource usage for ABACUS on an FPGA. Since ABACUS is modular in design with configurable parameters, we present a breakdown of the operating frequencies and resource usages for each major component for a default counter width of 40 bits.

Table I reports the breakdown for the configuration used in the LEON3 system on a Virtex 2Pro XC2VP30. The largest resource requirements are for the Code Profiling Unit due to the registers required to store the start and end addresses as well as the large number of comparators. The ABACUS configuration used in the LEON3 test platform consists of two Memory Reuse Units, an Instruction Mix Unit, and a Code Profiling Unit with an operating frequency of 144MHz on a Virtex 2 Pro utilizing 1367 FFs, 1504 LUTs and 1 Block RAM.

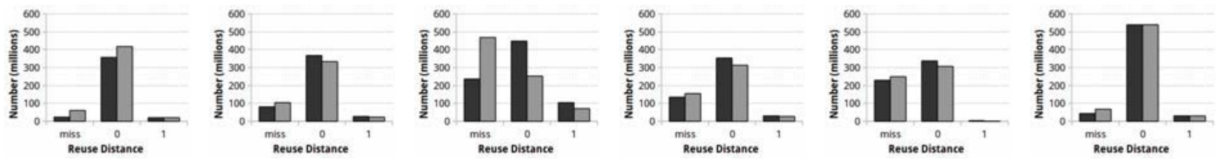
C. LEON3 Reuse Distance Experiment

To demonstrate the use of ABACUS, we measure the reuse distance for six benchmarks from the SPECCPU2006 benchmark suite with memory usage requirements that are within the 256MB memory limit of the LEON3 system. We also highlight the reduced runtime for obtaining these results and the importance of evaluating workload behaviour on the desired architecture and OS kernel, as opposed to the best approximation. For this, we use results obtained with Simics 3.0.26 configured to have the same L1 cache structure as the LEON3 system (a 2-way, 16KB instruction cache and a 2-way 8KB write-through, non-allocating data cache); neither Simics nor the LEON3 system is configured with an L2 cache. All benchmarks are run for 2 billion instructions, due to the runtime of the Simics simulations. Figure 3 shows the results for the instruction cache reuse profiles, using dark bars for Simics and light bars for ABACUS results. Since both platforms are executing the same binary, we expect the minimal variation ($<4\%$) seen in Figure 3. Although both platforms execute a 32-bit binary, minor differences in



(a) namd (b) gobmk (c) hmmer (d) sjeng (e) libquantum (f) h264ref

Fig. 3. Instruction Cache Reuse Distance Comparison



(a) namd (b) gobmk (c) hmmer (d) sjeng (e) libquantum (f) h264ref

Fig. 4. Data Cache Reuse Distance Comparison

the platform (e.g. the kernels) may cause these variations. Figure 4 illustrates the results for the data cache reuse profiles; again, the dark bars are for Simics and the light bars are for ABACUS. In this case, we find a significant variation in the results obtained on the two platforms. The larger discrepancy between the two sets of data reuse profiles is largely due to the fact that the SPARC v9 configuration on Simics has a 64-bit datapath whereas the LEON3's datapath is only 32 bits, although the difference in kernels may also cause some variation.

While recognizing that the variations in the results may provide useful information to researchers, another benefit of using ABACUS is the speed at which these results are obtained. Running the first 2 billion instructions on the FPGA boards takes 1.5 minutes, compared to the 45 minutes it takes to obtain the results using Simics, approximately a 20x speed up. In this experiment only one metric is examined, and yet, a significant speedup is obtained over the software simulator. However, as an increasing number of metrics are collected, the runtime of the ABACUS system will remain constant; whereas each additional metric will increase the run time of Simics due to the increased complexity of the software simulator model.

V. Conclusions and Future Work

With the growing complexity of processor systems, better insight into the thread execution behaviour on these systems is needed to properly utilize their capabilities. Our proposed solution is a new, cycle-accurate, hardware framework that leverages an FPGA's reconfigurable fabric for the investigation of thread execution behaviours on processors.

We have demonstrated the performance advantage over a system-level software simulator by obtaining more than a 20x speedup and highlighted the better performance scalability of a hardware based approach. Furthermore, by performing new metric investigations in hardware, their impact on the system in terms of resource usage and operating frequency can be better understood.

Future work will include expanding the platform to support systems with shared caches. With a more complex system model, investigations into new profiling units that provide greater visibility of resource contention for current and future multicore systems can be performed.

Acknowledgements

The authors would like to thank the Natural Sciences and Engineering Research Council of Canada (NSERC), Xilinx Inc and the Canadian Microelectronics Corporation (CMC) Microsystems for funding and support of this project.

References

[1] P. S. Magnusson *et al.*, "Simics: A full system simulation platform," *Computer*, vol. 35, no. 2, pp. 50–58, 2002.
[2] A. Pesterev *et al.*, "Locating cache performance bottlenecks using data profiling," in *European Conf. on Computer Systems*, 2010.
[3] *Intel 64 and IA-32 Architectures Software Developers Manual: Volume 3B: System Programming Guide, Part 2.* . [Online]. Available: www.intel.com/Assets/PDF/manual/253669.pdf
[4] *BIOS and Kernel Developers Guide (BKDG) For AMD Family 10h Processors*. [Online]. Available: support.amd.com/us/Processor_TechDocs/31116.pdf
[5] T. Austin, E. Larson, and D. Ernst, "Simplescalar: An infrastructure for computer system modeling," *Computer*, vol. 35, no. 2, pp. 59–67, 2002.
[6] C.-K. Luk *et al.*, "Pin: building customized program analysis tools with dynamic instrumentation," in *PLDI*, 2005, pp. 190–200.
[7] 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.
[8] 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.
[9] G. G. F. Lemieux, "Hardware performance monitoring in multiprocessors," Master's thesis, ECE Dept. University of Toronto, 1996.
[10] C. Caşcaval *et al.*, "Estimating cache misses and locality using stack distances," in *in Int'l Conf. on Supercomputing*, 2003, pp. 150–159.
[11] M. K. Qureshi and Y. N. Patt, "Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches," in *MICRO 39: in 39th Annual IEEE/ACM Int'l Symp. on Microarchitecture*, 2006, pp. 423–432.
[12] *GRLIB IP Core User's Manual*. [Online]. Available: www.gaisler.com/products/grlib/grip.pdf
[13] *The SPARC Architecture Manual Version 8*. [Online]. Available: www.sparc.org/standards/V8.pdf
[14] Xilinx, Inc., *Xilinx UG069 XUP Virtex-II Pro Development System, Hardware Reference Manual*. [Online]. Available: www.xilinx.com/univ/XUPV2P/Documentation/ug069.pdf
[15] *The SPARC Architecture Manual Version 9*. [Online]. Available: developers.sun.com/solaris/articles/sparcv9.pdf