A Complexity-Effective Architecture for Accelerating Full-System Multiprocessor Simulations Using FPGAs

Eric S. Chung*, Eriko Nurvitadhi*, James C. Hoe*, Babak Falsafi*†, Ken Mai*

* Computer Architecture Laboratory at Carnegie Mellon (CALCM), Carnegie Mellon University
† School of Computer and Communication Sciences, École Polytechnique Fédérale de Lausanne
{echung, enurvita, jhoe, babak, kenmai}@ece.cmu.edu
http://www.ece.cmu.edu/~simflex

ABSTRACT

Functional full-system simulators are powerful and versatile research tools for accelerating architectural exploration and advanced software development. Their main shortcoming is limited throughput when simulating systems with hundreds of processors or more. To overcome this bottleneck, we propose the PROTOFLEX simulation architecture, which uses FPGAs to accelerate simulation. Prior FPGA approaches that prototype a complete system in hardware are either too complex when scaling to large-scale configurations or require significant effort to provide full-system support. In contrast, PROTOFLEX reduces complexity by virtualizing the execution of many logical processors onto a consolidated set of multiple-context execution engines on the FPGA. Through virtualization, the number of engines can be judiciously scaled, as needed, to deliver on necessary simulation performance. To achieve low-complexity full-system support, a hybrid simulation technique called transplanting allows implementing in the FPGA only the frequently encountered behaviors, while a software simulator preserves the abstraction of a complete system.

We have created a first instance of the PROTOFLEX simulation architecture, which is an FPGA-based, full-system functional simulator for a 16-way UltraSPARC III symmetric multiprocessor server hosted on a single Xilinx Virtex-II XCV2P70 FPGA. On average, the simulator achieves a 39x speedup (and as high as 49x) over comparable software simulation across a suite of applications, including OLTP on a commercial database server.

Categories and Subject Descriptors

C.4 [**Performance of System**]: Modeling Techniques.

General Terms: Design, Experimentation, Performance

Keywords

FPGA, simulator, emulator, prototype, multicore, multiprocessor

1. INTRODUCTION

The rapid adoption of parallel computing in the form of multicores has re-energized computer architecture research. The research focus has now shifted toward architectural mechanisms for enhancing programmability and scalability in future, highly-concurrent computing platforms. Architectural research of this kind will require close collaborations between hardware and software researchers.

To co-develop new systems successfully, two conflicting dependencies must be resolved. First, software researchers cannot afford to wait until the hardware development cycle is complete. Second, developing new hardware requires feedback from software research which is difficult to attain before a design can be finalized. Fast and flexible functional full-system simulators will play a vital role in helping to resolve these dependences.

In the past and particularly in the uniprocessor setting, full-system functional simulators (e.g., [MCE02, MSB05, RHW95, WWF06]) have stayed within a 100x slowdown relative to the real system, a performance deemed acceptable by many hardware and software researchers. However, when simulating multiprocessor systems, the slowdown grows at least linearly in proportion to the number of simulated processors. Additionally, up to 10x or more slowdown is incurred with any meaningful instrumentation [NFS04]. As a result of these effects, modern architectural simulators are limited to simulating only tens of processors at speeds practical for functional exploration. The recent multicore scaling trends only exacerbate this problem further.

Parallelization efforts for improving multiprocessor simulation performance have not been fruitful due to the managing of communication overheads between distributed simulated components. The scalability of a distributed simulation is limited if the simulated components must interact at an interval below the communication granularity supported by the underlying host system [LW95, PFH06]. As a result, prior attempts to parallelize software simulators have been met with only limited success.

In this paper, we present the PROTOFLEX simulation architecture, which aims to leverage fine-grained concurrency in the form of FPGA acceleration to overcome the software simulation bottleneck. The goal of the PROTOFLEX simulation architecture is to simulate the functional execution of a multiprocessor system using FPGAs while lowering the development effort and cost to a justifiable level in a computer architecture research setting. This goal is distinct from prototyping the accurate structure or timing of a multiprocessor system. Specifically, the requirements for the architecture are to (1) functionally simulate large-scale multiprocessor systems with an acceptable slowdown (<100x), (2) model full-system fidelity [MSB05] to execute realistic workloads including operating systems, and (3) provide fast, low-overhead

FPGA'08, February 24-26, 2008, Monterey, California, USA.

instrumentation for architectural-level research. Below, we outline how PROTOFLEX achieves these goals.

Addressing large-scale system complexity. Prior FPGA simulators—even functional simulators (e.g., [WCN07])—implemented their models in a way that maintain structural correspondence to the simulated target system. In other words, 16 independent cores are instantiated and integrated together to deliver the functionality of a 16-way multiprocessor. While evidently feasible with small-scale systems, building an FPGA-based simulator with adherence to structural correspondence becomes difficult to justify when studying configurations with hundreds or thousands of processors.

We observe that in the case of providing fast, functional simulation, re-implementing the target system exactly in an FPGA is unnecessary. In the PROTOFLEX simulation architecture, the logical execution and resources of many processors can be virtualized onto a consolidated set of host execution engines on the FPGA. Within each engine, multiple processor contexts are interleaved onto a multiple-context high-throughput instruction pipeline. Through virtualization, the simulator architect is able to judiciously scale or consolidate the number of engines in the FPGA host, as needed, to deliver on necessary simulation performance.

Addressing full-system complexity. Apart from limitations in scale, prior FPGA efforts also typically forgo full-system fidelity and are limited to user-level custom-compiled applications (e.g., [OBI95]). Full-system fidelity enables a simulator to model real machines to a level of functional detail (e.g., devices) such that the simulated software, including the BIOS and operating systems, cannot make the distinction between the simulator and a real machine. This capability is supported in software simulators today (e.g., [MCE02, MSB05, RHW95, WWF06]) and is especially important for evaluating unmodified applications on real operating systems. Implementing the same capabilities entirely in FPGAs would require detailed hardware design knowledge and effort typically beyond the resources of the simulator architect.

To address this challenge, we observe that the great majority of behaviors encountered dynamically in a full-system simulation make up a very small subset of total system behaviors. It is this small subset of behaviors that primarily determines the overall simulation performance. To exploit this observation, the PROTOFLEX simulation architecture incorporates a hybrid simulation technique called **transplanting**, which allows a simulated entity such as a processor to be dynamically reassigned from FPGA hardware into software-based simulation. The simulator architect is now given the option to select only a subset of common-case behaviors for implementation on the FPGA while still retaining the abstraction of a complete system.

Supporting fast functional instrumentation. As mentioned previously, any form of substantial instrumentation (e.g., memory address tracing) can introduce 10x or more slowdown to software-based functional simulation. One of the advantages with using FPGA-based simulation is the ability to instrument the hardware with little overheads in performance (e.g., many passive counters in parallel). As we discuss in Section 2, this capability allows us to overcome the functional software simulation bottleneck for cycle-accurate simulation sampling methodologies.

Contributions. We developed an instantiation of a functional full-system, FPGA-based simulator called **BlueSPARC** that incorporates the two key concepts of the PROTOFLEX simulation architecture: (1) time-multiplexed interleaving and (2) hybrid simulation with transplanting. BlueSPARC is our first step toward simulating large-scale multiprocessor systems and currently models the architectural behavior of a 16-CPU UltraSPARC III SMP server. BlueSPARC is hosted on a single 16-way multithreaded instruction-interleaved pipeline running on the BEE2 FPGA platform [CWB05], while Virtutech Simics provides the backing software substrate during hybrid simulation. In the future, we envision combining multiple BlueSPARC pipelines to simulate even larger systems. BlueSPARC currently executes real applications on an unmodified Solaris 8 operating system, including On-Line Transaction Processing (OLTP) on Oracle. Our performance evaluation shows that we achieve a 39× speedup average over comparable software simulation using Simics. The majority of the BlueSPARC development effort has been carried out by a single graduate student in a little over a year.

Outline. Section 2 describes work related to FPGA-based simulation and prototyping. Section 3 presents the PROTOFLEX simulation architecture in detail. Section 4 presents an instantiation of the PROTOFLEX simulation architecture called the BlueSPARC simulator. Section 5 presents an evaluation and a performance model for the BlueSPARC simulator. Section 6 discusses current limitations of the BlueSPARC simulator and suggests future developments for the PROTOFLEX simulation architecture. We conclude in Section 7.

2. RELATED WORK

FPGA prototyping. Recent FPGA efforts such as [LYK07,VH07] have produced full-system prototypes involving a CPU implemented in an FPGA, while real motherboards and PC components are used to provide the CPU-external full-system environment. In both examples, the CPU designs are commandeered from existing low-level RTL models developed originally for standard cell technologies. While the implementation effort only requires a porting effort to the FPGA, neither the CPU internals nor the system environment is easily amenable to the modifications needed for architectural design exploration.

Other approaches such as [WCN07, OBI95] implement functional multiprocessor models with approximate timing fidelity—that is, they are not cycle-accurate but retain timing similarities to the target system. However, these approaches forgo full-system fidelity and also maintain structural correspondence to the target system in the number of processors implemented in the FPGA.

Cycle-accurate FPGA-based simulation. UT-FAST [CSK07] is another form of hybrid simulation that uses FPGAs to accelerate cycle-accurate simulation of uniprocessor microarchitectures. UT-FAST is currently divided into a software functional partition [BE05] and a timing model implemented in the FPGA. Their simulation performance is constrained in part by the speed of the full-system functional partition—an issue the PROTOFLEX simulation architecture potentially addresses.

Simulation sampling. Note that functional simulation speed is not only important to studying architecture issues but also to microarchitectural-level performance studies. Our prior research in the SIMFLEX project has developed a sampling-based timing simulation methodology that leverages fast architectural simulation [WWF06]. Simulation sampling research has shown that it is possible to achieve very accurate estimation of uniprocessor and multi-processor performance by simulating a small fraction of the

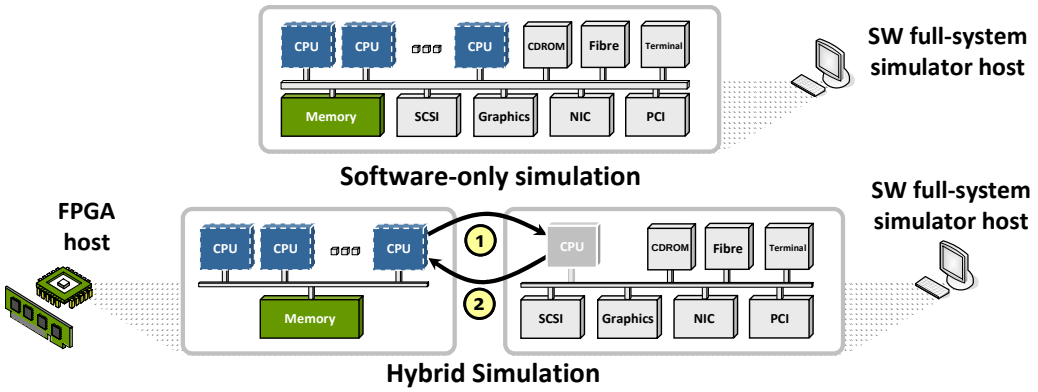

Figure 1 Partitioning a simulated target system across an FPGA and software simulator during hybrid simulation

total benchmark in cycle-accurate mode. In short, the performance of the detailed cycle-accurate simulator is itself inconsequential to the latency of performance data collection. The performance bottleneck now lies in the amount of time it takes to advance the system's architectural state across the large gaps between sampled sections of the benchmark and to perform functional warming of microarchitectural structures (e.g., caches). As opposed to software, a key benefit of using FPGA-based simulation is the ability to carry out this instrumentation activity at-speed without a performance overhead.

3. PROTOFLEX

In this section, we expand on the PROTOFLEX simulation architecture for FPGA acceleration of functional full-system simulation. We discuss both hybrid simulation with transplanting and interleaving of multiple processor contexts [CNH07]. Note that the PROTOFLEX simulation architecture is not a specific instance of hardware or infrastructure but embodies a set of general design concepts when approaching an FPGA-based simulation design.

3.1 Hybrid Simulation

Hybrid Simulation is motivated by the observation that the great majority of behaviors encountered dynamically in a simulation during runtime are contained by a small subset of total system behaviors. It is this small subset of behaviors that primarily determines the overall simulation performance. Thus, to improve software simulation performance while minimizing the hardware development, one only applies FPGA acceleration to components that exhibit the most frequently encountered behaviors.

A key ingredient in developing an FPGA-based simulation platform is to have an existing software reference model of a target system at hand. Not only is this the most tractable way to capture and debug the wide range of behaviors necessary, but it is also a crucial enabler in validating the FPGA hardware.

Hybrid simulation. Figure 1 offers a high-level depiction of the hybrid simulation of a multiprocessor system. In the figure, all components are either hosted on the FPGA or simulated by the reference simulator; specifically, the main memory and CPUs are hosted in the FPGA while the remaining components are hosted in the reference simulator (e.g., disks and network interfaces, etc). When a software-simulated DMA-capable I/O device accesses memory, it accesses a hardware memory module on the FPGA platform. In a simulation, both the FPGA-hosted and software-simulated components are advanced concurrently to model the progress of a complete system (e.g., simulating disk activity in parallel with processors running on the FPGA).

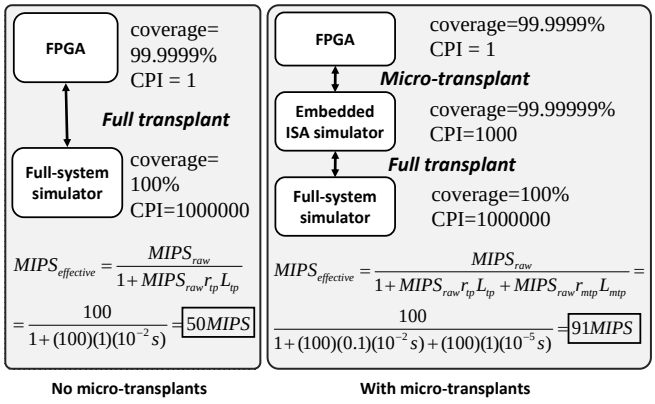

Figure 2 Improving performance with hierarchical transplanting

Transplanting. A component such as the CPU can be partitioned into a small core set of frequent behaviors and an extensive set of complicated or rare behaviors. Assigning the complete set of CPU behaviors statically to either software simulation or FPGA results in either the simulation being too slow or the FPGA development being too complicated. The conflicting goals can be reconciled by supporting "transplantable" components which can be re-assigned to the FPGA or software simulation at runtime. In the CPU example, the FPGA would only implement the subset of the most frequently encountered instructions. When the partially implemented CPU encounters an unimplemented behavior (e.g., a page table walk following a TLB miss), the FPGA-hosted CPU component is suspended. Immediately after, its state is "transplanted" to its corresponding software-simulated component in the reference simulator (❶ in Figure 1). The software-simulated CPU component is then activated to carry out the unimplemented behavior before becoming de-activated again. Finally, the CPU state is then transplanted back to the suspended FPGA-hosted CPU component to resume acceleration on the FPGA (❷ in Figure 1).

Hierarchical transplanting. While transplanting is a straightforward way to achieve full behavior coverage of a complex component like the CPU, there is a significant performance overhead associated with the hand-off between FPGA and software simulation. This includes both the time to transfer data between the FPGA and the software hosts, and the time for the software simu-

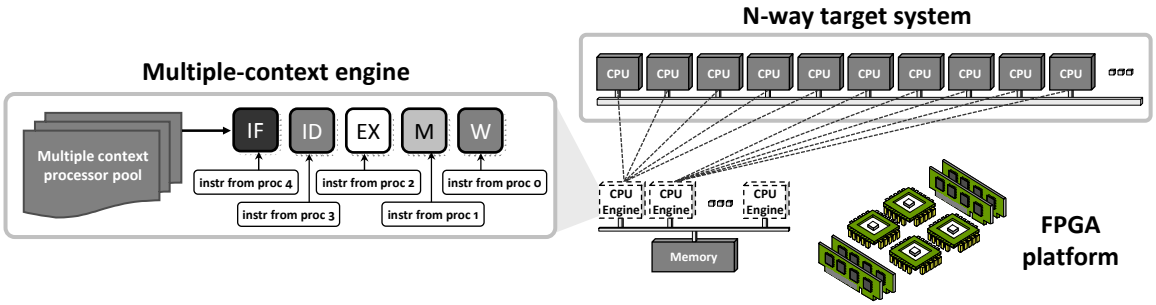

Figure 3 Large-scale multiprocessor simulation using a small number of interleaved engines

lator to carry out the unimplemented behavior on behalf of the FPGA. This exchange could take several milliseconds if the FPGA and the software simulation hosts are separated far apart (e.g., over Ethernet).

The effective hybrid simulation throughput for a 1-CPU pipeline with transplanting is

$$MIPS_{effective} = MIPS_{raw} / (1 + MIPS_{raw} \cdot rate_{tplant\text{-}per\text{-}million} \cdot L_{tplant})$$

where $MIPS_{raw}$ is the simulation throughput of a single FPGA-hosted CPU; $rate_{tplant}$ is the number transplants per million instructions; and L_{tplant} is the latency cost of a transplant in seconds.

Figure 2 (left) illustrates an example calculation assuming a 100MHz FPGA-hosted CPU with CPI=1. Even if the FPGA-hosted CPU transplants just once every 1,000,000 instructions, 50% of the 100 MIPS raw throughput potential would be lost due to the high transplant penalty (10msec=1 million cycles). Because $MIPS_{raw}$ appears in the denominator of the discount factor, attempting to improve performance by increasing $MIPS_{raw}$ would yield diminishing returns. Attempts to raise performance by reducing $rate_{tplant}$ would also face diminishing returns because increasingly more instructions would have to be built in the FPGA-hosted CPU to reduce the transplant rate further.

A better solution is to introduce a hierarchy of CPU transplant hosts with staggered, increasing instruction coverage and performance costs. For example, an embedded processor (e.g., the embedded PPC405 in the Xilinx Virtex architecture or even a synthesized soft core) can support a simple software-based simulation kernel for the CPU. The simple software-based simulation kernel would still be slow relative to the FPGA-hosted CPU, but would be much faster than a full transplant to the reference software simulator. The embedded simulation kernel should also be able to reach, with relative ease, a higher instruction coverage in software than in the FPGA-hosted CPU, which reduces the rate of expensive transplants to the reference software simulator.

Returning to the example in Figure 2, suppose we introduce an embedded software-based simulation kernel as an intermediate CPU transplant host with CPI=1000. Further suppose that the intermediate simulation kernel transplants to the full reference simulator only once every 10 million instructions. Then, 90% of the 100 MIPS raw throughput potential would be preserved.

3.2 Interleaved Multiprocessor Simulation

The most important goal in FPGA-accelerated multiprocessor simulation is to provide simulation performance beyond the capabilities of software-based simulators. It is first important to understand how fast a useful multiprocessor simulator needs to be.

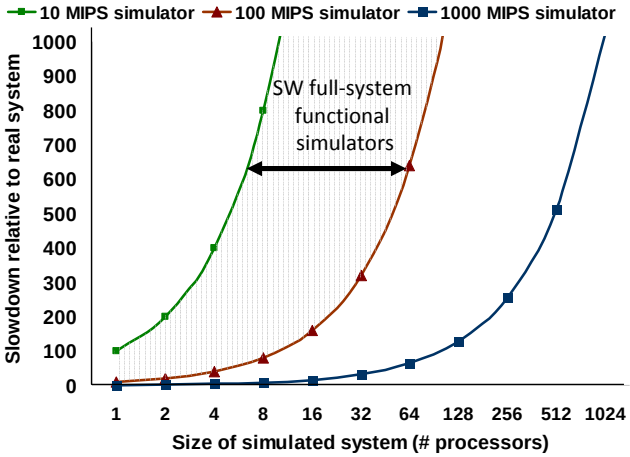

Figure 4 Simulation slowdown versus system size (assuming a single real processor performs at 1 GIPS)

Simulation performance requirement. Interactive software research can tolerate no more than 100x slowdown. Batched performance simulation studies can tolerate as much as 1000x to 10,000x slowdown. Assuming three simulators with aggregate simulation throughputs of 10, 100 and 1000 MIPS, respectively, Figure 4 plots the user perceived simulation slowdown (y-axis) of the given simulator relative to a real system. As the system size increases, the aggregate simulation throughput is divided by the number of processors being simulated (x-axis). We assume a "real" multiprocessor system consists of 1000-MIPS processors.

Today's fastest full-system software simulators have less than 100 MIPS aggregate throughput. As seen in Figure 4, a 100-MIPS simulator represents only a 10x perceived slowdown in a uniprocessor simulation, but is only practical for studying up to tens of processors before the slowdown is no longer acceptable. Figure 4 also suggests that a 1000-MIPS simulator is already adequate for effective hardware and software research of a 100-way or even 1000-way multiprocessor system.

The straightforward approach to construct a 1000-way multiprocessor FPGA-based simulator is to replicate 1000 cores and integrate them together in a large-scale interconnection substrate. While this meets the requirement of simulating a large-scale system, the development effort and required resources would be excessive or prohibitive and the final aggregate throughput would be 100 to 1000x faster than needed. A performance-driven approach would trade the excess simulation performance for a more tractable hardware development effort and cost.

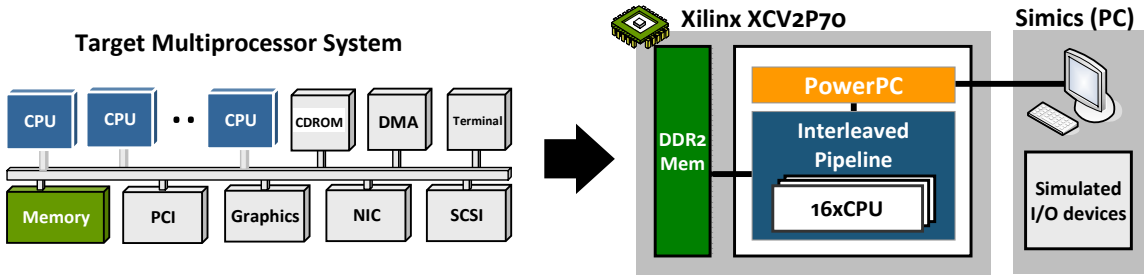

Figure 5 Allocating components for hybrid simulation in the BlueSPARC simulator

Table 1 BlueSPARC simulator characteristics

Processing nodes	16 64-bit UltraSPARC III contexts 14-stage interleaved pipeline
Caches	Split L1 I/D, 64KB, 64B, direct-mapped, writeback Non-blocking loads/stores 16-entry MSHR, 4-entry store buffer
Clock frequency	90MHz
Main memory	4 GB total memory
Resources (Xilinx V2P70)	33,508 LUTs (50%), 222 BRAMs (67%) 43,206 LUTs (65%), 238 BRAMs (72%) (+ monitors/stats/debug/assertions)
EDA tools	Bluespec System Verilog [Blue06] Xilinx EDK 9.2i, XST 9.2i
Statistics	25K lines Bluespec, 511 rules, 89 module types

Table 2 Selected partitioning in hybrid simulation

BlueSPARC (FPGA)	**Micro-transplant** (on-chip simulation)
• add/sub/shift/logical • multiply/divide • register windows • 38/103 SPARC ASIs • interprocessor x-calls • device interrupts • I-/D-MMU + tlb miss • Loads/stores/atomics • VIS block memory	• 65/103 SPARC ASIs • VIS I/II multimedia • FP add/sub/mul/div + traps • FP/INT conversion • trap on integer arithmetic • alignment • fixed-point arithmetic • tlb/cache diagnostics • tlb demap

Transplant (off-chip simulation)

• pci bus • isp2200 fibre channel • i21152 pci bridge • irq bus	• text console • sbbc pci device • serengeti IO prom • cheerio-hme NIC	• fibre channel SCSI disk • SCSI cdrom • SCSI bus

A performance-driven approach. The PROTOFLEX simulation architecture advocates *virtualizing* the simulation of multiple processor contexts onto a single fast engine through time-multiplexing. Virtualization decouples the scale of the simulated system from the required scale of the FPGA platform and the hardware development effort. The scale of the FPGA platform is only a function of the desired throughput (i.e., achieved by scaling up the number of engines). For example, Figure 3 illustrates the high-level block diagram of a large-scale multiprocessor simulator using a small number of interleaved engines. Multiple simulated processors in a large-scale target system are shown mapped to share a small number of engines.

Multiple-context engine design. Our decision to fulfill the role of a multiple-context engine using an instruction-interleaved pipeline is based on a number of reasons. An instruction-interleaved pipeline switches a processor context on every fetch cycle, allowing for multiple processor contexts to share a single pipeline. Such a design enjoys several implementation advantages as exemplified in prior designs such as the CDC Cyber and the HEP barrel processor. First, stalling and internal forwarding can be eliminated entirely if only at most 1 instruction from each context is allowed in the pipeline at any given time. Second, longer pipelines can be used to mitigate critical timing paths, which enables better overall clock frequency. Third, it is well known that such a design is highly tolerant to long-latency events such as cache misses and transplants since additional threads can execute in the shadow of a long-latency event. Finally, in an implementation with caches, multiple threads are automatically kept coherent in shared-memory as they interleave onto a single L1 cache; this helps to consolidate the number of nodes in a cache coherency design in larger implementations with multiple engines.

4. THE BLUESPARC SIMULATOR

4.1 Overview

In this section, we describe BlueSPARC, which is our first instantiation of the PROTOFLEX simulation architecture. BlueSPARC is one of our first steps towards simulating large-scale multiprocessor systems and currently models a 16-way symmetric multiprocessor (SMP) UltraSPARC III server (Sun Fire 3800). The BlueSPARC simulator combines Virtutech Simics on a standard PC as the reference simulator and only a single 16-context interleaved engine on an FPGA for acceleration. The FPGA portion is hosted on a Berkeley Emulation Engine 2 (BEE2) FPGA platform [CWB05]. The BlueSPARC simulator embodies the two key concepts of the PROTOFLEX simulation architecture: 1) hybrid simulation to achieve both performance and full-system fidelity with reasonable effort and 2) decoupling the logical system size from the physical design complexity. Future work will develop the cache coherency support needed for combining multiple engines to support simulation of larger multiprocessor systems.

The BlueSPARC simulator can serve the same functionality currently provided by simulators such as Simics. In fact, the BlueSPARC simulator uses the same simulated console as Simics, so a user can even "interact" with the simulated server. Furthermore, BlueSPARC is fully capable of generating and loading Simics-compatible checkpoints of the entire simulated system state.

High-level organization. Figure 5 shows a high-level block diagram of how we map the functionality of the target 16-way SMP server onto software simulation and hardware hosts. First, the main memory system is hosted directly by DDR2 DRAM on the

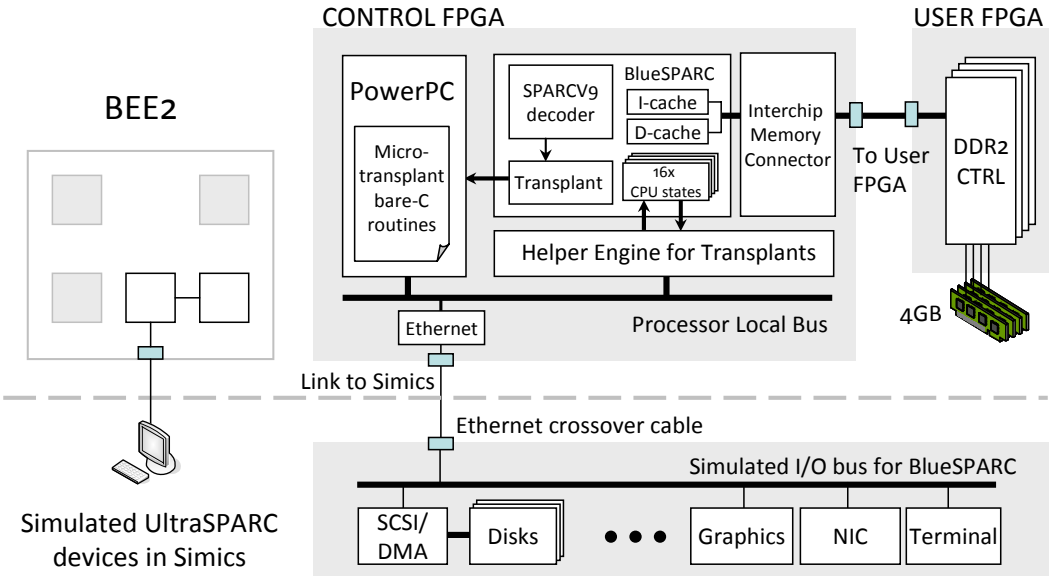

Figure 6 High-level Organization of the BlueSPARC simulator

BEE2. Next, all 16 target SPARC processors are mapped onto a single interleaved engine contained on one Xilinx Virtex-II XC2VP70 FPGA. The target SPARC processors are transplant-capable components. In addition to the interleaved engine, the PPC405 embedded processors serve as an intermediate transplant host for the target SPARC processors when they encounter unimplemented behaviors in the interleaved engine. The reference Simics simulator running on a PC workstation provides a third hosting option for the target SPARC processors. Last but not least, all remaining components of the simulated system are hosted by the reference Simics simulator. Table 1 summarizes the high-level characteristics of the BlueSPARC simulator (which we explain in more detail in the coming pages).

Implementation on BEE2. Figure 6 shows a structural view of our entire system mapped onto the BEE2 FPGA platform. The BEE2 board holds five Xilinx V2P70 FPGAs, which are connected to each other using low-latency, high-bandwidth 200MHz links. The FPGA-portion of the BlueSPARC simulator is hosted on two FPGAs. The center FPGA is used to host the BlueSPARC interleaved engine. This pipeline is connected over a high-speed inter-chip link to another FPGA, which hosts four independent DDR2 controllers for up to 4GBytes of main memory.

In the center FPGA, the PPC405 interacts with the BlueSPARC engine over the processor local bus (PLB) for servicing both transplants and I/O transactions. In addition to serving as an intermediate transplant host for the target SPARC processors, the PPC405 embedded processor also manages external communication with the simulation PC over Ethernet. The PC and the BEE2 are connected over a standard cross-over cable.

4.2 Hybrid Simulation and Transplanting

The BlueSPARC simulator fully incorporates the idea of hybrid simulation and transplanting as a methodology to reduce implementation complexity. The BlueSPARC engine only supports the minimum subset of the UltraSPARC III ISA required to achieve acceptable performance. We select instructions for inclusion in the FPGA following a quantitative profile of selected commercial and integer benchmarks. A general rule of thumb in the selection was to implement in FPGA the set of behaviors necessary to limit the rate of transplants to no more than 5 times in 10,000 instructions in any of the software workloads. This rule-of-thumb was determined through estimated initial design parameters and first-order software simulations of an interleaved pipeline augmented with transplanting.

In the current BlueSPARC simulator, the embedded software-based simulation kernel can capture the complete behavior of the UltraSPARC III processor. The only time execution is transplanted to the Simics reference simulation on the PC is when the CPU interacts with I/O devices. For clarity, we refer to a transplant to the embedded software-based simulation kernel as a **micro-transplant**, and we refer to a transplant to the Simics reference simulation as an **I/O-transplant**. Table 2 shows a detailed breakdown of how we partitioned the required behaviors of an UltraSPARC III processor for direct simulation in FPGA, micro-transplant, and I/O-transplant.

The BlueSPARC engine must support all of the user-level instructions as well as a minimal subset of privileged and implementation-dependent behaviors. A great majority of the time, the frequent behaviors that must be included are also the simple ones—leaving the most complicated instructions to the embedded software-based simulation kernel. It is worth mentioning that some complex behaviors do occur sufficiently frequently to warrant implementation in the simulation engine. Despite being a RISC ISA, the UltraSPARC III superset of SPARCV9 has a large number of complex implementation-dependent and legacy behaviors (e.g., SPARC ASIs). The explicitly-defined TLB size also poses an implementation challenge by consuming a large amount of BlockRAMs when replicated 16 times for the processor contexts.

However, despite these challenges, we found that our implementation was still justifiably easier than a full-blown prototype. Without micro-transplants or I/O transplants, implementing all of the behaviors in Table 2 in hardware would have required a tremendous amount of development effort.

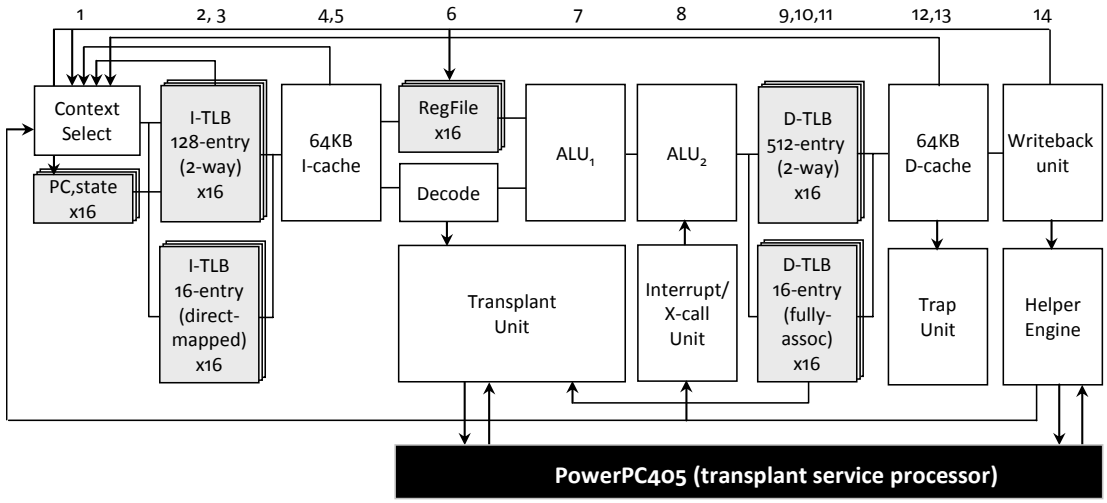

Figure 7 The BlueSPARC engine: a 14-stage instruction-interleaved pipeline

4.3 Interleaved Pipeline

The BlueSPARC engine is a 14-stage, instruction-interleaved pipeline that supports the multithreaded execution of up to 16 UltraSPARC III processor contexts (see Figure 7). The maximum retirement rate of our pipeline is one instruction per cycle, which in combination with the clock frequency dictates the peak throughput. The primary goals in developing the BlueSPARC engine are 1) to ensure correctness, 2) to maximize maintainability for future design exploration and 3) to minimize effort. In many cases we forgo complex performance optimizations in favor of a simpler, more maintainable design.

Pipeline operation. Figure 7 shows instructions beginning at the Context Select Unit (left), which is a simple fair scheduler that issues in round-robin order from different processor contexts. The scheduler is responsible for ensuring that all processor contexts make equal progress on average. This requirement is important in order to avoid unrealistic system behaviors (e.g., a processor holding a lock indefinitely). After being selected, each issued instruction is tagged with a unique context identifier used to index various structures and registers replicated for each processor context throughput the pipeline (see shaded components). Once the instruction is selected, a subset of its architectural registers (e.g., pstate, current window pointer) are scanned from a state register file and passed into the pipeline. After fetch, the decoding stage determines whether an instruction requires transplanting. Supported instructions in the common-case proceed through the pipeline stages until reaching the writeback stage. At writeback, the processor context re-enters the Context Select Unit for the next round of scheduling. A processor context can only have at most one instruction in the pipeline at a time, therefore register file RAW hazards cannot arise.

Micro-transplants and I/O-transplants. In the event a transplant operation is identified by the decoder, the unimplemented instruction word and the processor context ID are queued into the Transplant unit (see Figure 7) for service by the embedded PPC405 processor running a small UltraSPARC III ISA simulation kernel. With 16 processor contexts and 14 pipeline stages, while one transplanted context is being serviced, the remaining 15 contexts can still keep the pipeline fully utilized. Only one outstanding micro-transplant and one outstanding I/O-transplant are currently supported. Any additional I/O- or micro-transplant requests from other contexts are queued in the Transplant unit.

Figure 8 illustrates step-by-step how the interleaved pipeline interacts with the embedded PPC405 during a micro-transplant. After a detected unimplemented instruction is queued up in the Transplant unit, the PPC405 is interrupted (1) and begins to read a minimum amount of state from the requesting context to decode the unimplemented instruction (2). After decoding, the PPC405 continues to read from the engine the state required to complete the unimplemented instruction. Reading and writing processor state (e.g. register file and cache reads) may require issuing synthetic "helper" instructions through the engine. Once the input state is acquired, the ISA simulation kernel computes the state updates (3) and writes the changed state values back to the engine for the requesting context (4)—again with the help of helper instructions—before returning the blocked processor context back to scheduling.

Figure 8 also shows how I/O-transplants are handled. During simulation, Simics continuously polls the embedded PPC405 to examine for I/O activity queued in the transplant unit (1)/(2). If an I/O transaction (e.g., a memory-mapped load or store instruction) is pending, the PPC405 acquires the necessary state information from the engine and forwards the state to Simics in response to the polling (3)/(4). For I/O transplants, Simics simulates the memory-mapped I/O bus transaction to the target simulated devices (5) and returns the acknowledgement (6)/(7).

Other sources of serialization and latency. In addition to transplants**,** a variety of long-latency events can deter the progress of an instruction in the pipeline. The most frequently encountered example is a cache miss. In our memory system, the caches are non-blocking, and up to sixteen misses to independent cache sets are allowed at any given time. Other contexts may continue to utilize the pipeline while memory accesses are pending. To avoid conflicts between multiple contexts for cache sets in a transient state, each cache set is augmented with a pending bit in the tag array. Any subsequent context that attempts to access a pending cache set is removed from the pipeline and forced to **retry** from

the scheduler at a later time. As we show later in Section 5, retries can have a noticeable effect on performance.

To hide store misses, a 4-entry store buffer allows processor contexts to retire past a store miss; we currently do not implement store-to-load forwarding and thus any conflicting loads will stall until pending stores are drained into memory.

A number of special instructions cannot finish in a single pass through the pipeline. For example, atomics (e.g., ldstub) require a load followed by a store. Other examples of multi-pass instructions are Quad LDDs (atomic load of two doublewords), or block load/store instructions (64B loads into 8 consecutive registers). We implement a helper engine (see Figure 7) that issues helper instructions to facilitate multi-pass instructions. Note that helper instructions can steal pipeline slots but happen rarely in practice. As noted earlier, the helper engine also facilitates state access during micro-transplanting. Lastly, cache flushes and pipeline "freezes" can also introduce serialization into the pipeline. Because the i- and d-caches are incoherent, we conservatively flush the caches on any possibility of staleness in the I-cache. Pipeline "freezes" occur when the entire pipeline is stalled for a temporarily owned resource (e.g., during TLB replacement).

In Section 5, we quantify how all of these events can affect the overall pipeline performance. We now discuss strategies used to validate the BlueSPARC implementation.

4.4 Validation

Our validation methodology involves a combination of Verilog/C simulation and testing directly on the FPGAs. We rely on Simics to provide us with a complete and correct reference model for the UltraSPARC III server, including the pre-validated device models. In general, all of our testing (either in RTL simulation or running directly on FPGAs) must pass validation by matching up exactly with the architectural state that Simics generates. Our validation test suite comprises a subset of ported diagnostics from the OpenSPARC framework [VH07], auto-generated stress-test programs, hand-written test-cases, and real applications/benchmarks.

We relied on a variety of strategies to validate our design in the presence of non-determinism on the FPGA. In order to match architectural state with Simics, we implement a hardware "event recorder" that records the global commit order of instructions within the pipeline for each processor context. The event recorder also captures the timing of asynchronous events during runtime (e.g., the exact point of delivery of an interrupt). This ordering is scanned from the FPGA on the BEE2 and replayed in Simics to attain identical architectural and memory state for comparison.

Another validation strategy we adopted was to implement over 200 synthesizable assertion instances in our design. These assertions were monitored using the Chipscope built-in logic analyzer and responsible for detecting over 50% of the design bugs. In all, we have high confidence that BlueSPARC is validated to a level suitable for active use by end-users. We have successfully passed all of our test suites and have been able to validate billions of instructions in all of our workloads using the event recorder. We also have been able to interact with the hardware using a simulated console in Solaris and have been able to run console applications correctly without assertions firing or crashes occurring.

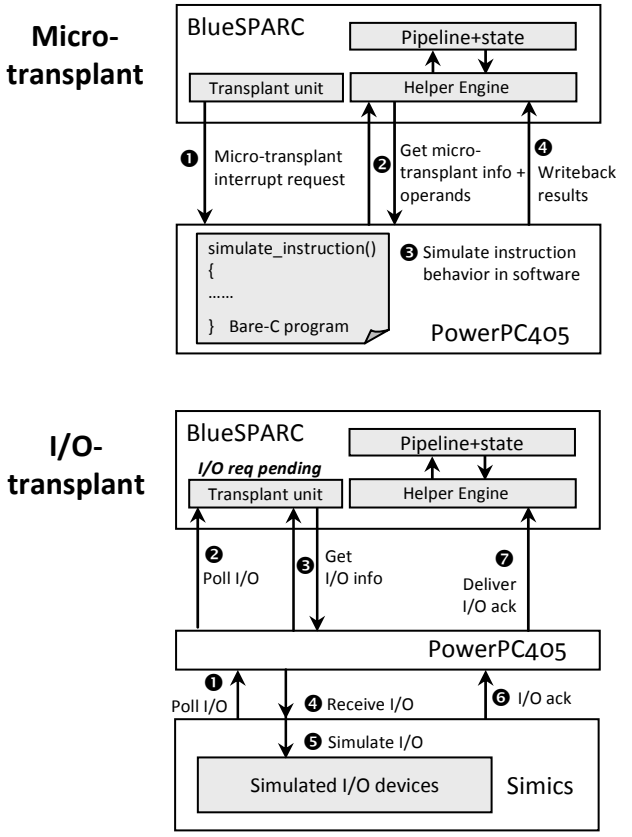

Figure 8 Detailed operation of Micro-transplants & I/O-Transplants

5. EVALUATION

This section reports the simulation throughput of the 16-way BlueSPARC simulator described in Section 4. The performance evaluation includes a comparison to the Simics simulator. Furthermore, we present a first-order model to understand the performance of multithreaded interleaving.

5.1 Simulation Throughput

Benchmarks. The evaluations in this section are based on simulating a 16-way symmetrical multiprocessing (SMP) UltraSPARC III server (Sun Fire 3800). The simulated server is running an unmodified Solaris 8 operating system. We exercise the simulated server using 16-cpu software workloads for saturating the pipeline. These consist of six SPEC2000 integer benchmarks (bzip2, crafty, gcc, gzip, parser, vortex) and a TPC-C On-Line Transaction Processing (OLTP) benchmark. For the SPECint workloads, 16 independent copies of the benchmark program are executed concurrently on the simulated 16-way server; we measure simulation throughput for 100 billion aggregate instructions (after the program initialization phase). Due to limited physical memory on our target configuration, it was necessary to run with test inputs and to omit benchmarks that exhibited excessive disk paging activity after program initialization (gap, mcf). Other SPECint benchmarks (eon, perlbmk, twolf, and vpr) were omitted due to a high presence of FP instructions, which we do not accelerate in FPGA. These instructions appear at sufficient rate (>1 in 1000) to incur prohibitive performance overheads if micro-transplanted. For the OLTP benchmark, the simulated server is running a commercial multithreaded database server—Oracle 10g

Table 3 Long-latency event rate (r_i) and latency (L_i). Percentages represent the rate of events per instruction. Numbers in parenthesis represent the latency per event in terms of 90MHz cycles.

	Micro-tplant	I/O-tplant	Load miss	Fetch miss	Scheduler	Retry	Flushes	Multi-pass
Oracle	0.015% (7448)	0.00066% (420354)	2.94% (58)	1.43% (56)	100% (2.16)	3.10% (75)	0.0037% (3352)	0.808% (21)
Bzip2	0.000% (3149)	0.00007% (151695)	8.68% (96)	0.30% (105)	100% (1.2)	11.49% (96)	0.0042% (3256)	0.096% (13)
Crafty	0.000% (3234)	0.00001% (153707)	7.07% (130)	2.49% (133)	100% (0.66)	8.35% (46)	0.0010% (3380)	2.369% (13)
Gcc	0.005% (7784)	0.00007% (149454)	1.92% (114)	0.95% (114)	100% (2.08)	2.49% (118)	0.0057% (3319)	0.077% (30)
Gzip	0.000% (16945)	0.00007% (105389)	6.82% (113)	0.26% (126)	100% (1.32)	8.44% (75)	0.0035% (3284)	0.084% (33)
Parser	0.000% (2690)	0.00008% (149826)	5.33% (122)	2.50% (123)	100% (1.93)	7.83% (99)	0.0091% (3244)	0.471% (52)
Vortex	0.007% (1806)	0.00018% (118308)	8.97% (173)	3.90% (168)	100% (2.02)	15.56% (84)	0.0045% (3294)	0.150% (32)

Enterprise Database Server—configured with 100 warehouses (10GB), 16 clients, and 1.4 GB SGA as in [WWF06]. We measure simulation throughput for 100 billion aggregate instructions in a steady-state region (where database transactions are committing steadily). As we will see next, the workload has a large impact on the simulation throughput of both Simics and BlueSPARC.

Simics performance. When Simics is invoked with the default "fast" option, it can achieve many tens of MIPS in simulation throughput. However, the fast mode is a purely black-box system—that is it does not support instrumentation (e.g., memory address tracing) or augmentation of behavior (e.g., adding a new opcode). There is approximately a factor of 10x reduction in simulation throughput when Simics is enabled with trace callbacks for instrumentation (this observation is also noted by [NFS04]). In Figure 9, the bars labeled *Simics-fast* and *Simics-trace* report our measured Simics simulation throughput (total simulated instruction per second) for the simulated 16-way SMP server. Simics simulations were run on a Linux PC with a 2.0 GHz Core 2 Duo and 8 GBytes of memory. The performance most relevant to architecture research activities is reported in *Simics-trace*. Here, an approximately 10× difference in simulation throughput is seen when Simics is run without the fast-mode.

BlueSPARC performance. The simulation throughput of the BlueSPARC simulator is reported in the left-most bars in Figure 9. The interleaved engine is clocked at 90MHz. The BlueSPARC simulator with acceleration from just a single Virtex II XCV2P70 FPGA achieves speed comparable to *Simics-fast* on the SPECint and Oracle-TPCC workload. In comparison to the more relevant *Simics-trace* performance, the speedup is more dramatic, an average speedup of 39x and as high as 49x on GCC. As mentioned in Section 2, fast functional instrumentation is a key capability in accelerating cycle-accurate simulation sampling activities. Although we do not discuss in detail in this paper, we have successfully implemented an FPGA-based functional 16-way CMP cache simulator that can accept memory reference traces at full rate for the purposes of fast functional warming in the SIMFLEX project. In the next section, we present a detailed analysis explaining the reported performances of BlueSPARC.

5.2 Performance Analysis

The performance model we present is based on a simple accounting of pipeline utilization by the interleaved processor contexts. Under ideal conditions, the instructions from multiple processor contexts would be issued into the pipeline in round-robin order, without stall or interference. Let N be the number of interleaved processor contexts and P be the depth of the interleaved pipeline ($N > P$ in our design). Suppose I is the average interval between the instructions issued from one processor context into the pipeline (in cycles). The pipeline utilization by one context is then $1/I$,

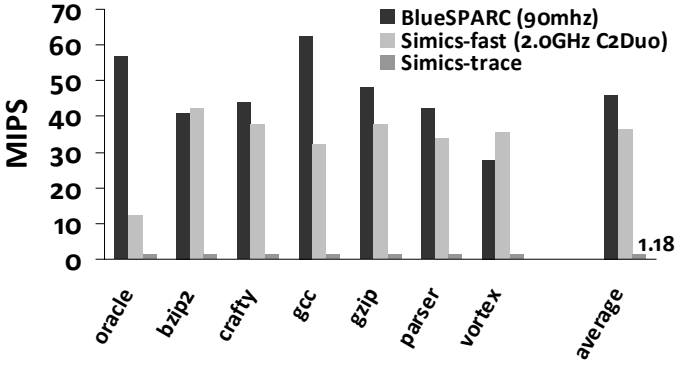

Figure 9 Performance comparison of BlueSPARC and Simics

and the total utilization by N contexts is N/I. In the ideal case, $I=N$ when the pipeline is fully utilized. I can be further decomposed into two components in the ideal case, where $I=P+S$ cycles. The first component represents the number of cycles traversed in a P-stage pipeline (constant). The S component refers to cycles spent waiting in the scheduler in a fully occupied pipeline. $S=N-P$ when the pipeline is fully utilized.

Under real conditions, events such as cache misses, micro-transplants and I/O-transplants increase the average interval I between the instructions issued from one processor context. We can express I_{avg} as

$$I_{avg} = P + S + r_1L_1 + r_2L_2 + r_3L_3 + \ldots$$

where r_i is the rate (event per instruction) of a specific long latency event that delays the completion of an instruction by L_i cycles. During long-latency events, the blocked processor context is removed from the pipeline and does not consume pipeline issue slots. Thus, a high pipeline utilization can be sustained even in the presence of these long latency events provided 1) $N>P$ and 2) the number of ready-to-execute contexts is at least P.

Table 3 gives the rate r_i (events per instruction) and average latency L_i (in 90MHz cycles) for the event classes that impact I_{avg}. All of these statistics are collected using performance counters built into the interleaved engine. Using the measured data as input, the I_{avg} model agrees well with wall-clock measurements and is able to predict pipeline utilizations for all of the workloads within 1% error. Figure 10 illustrates the measured fractional contribution of each event class to the overall $I_{avg.}$ Each percentage component from an event class is calculated as $(r_iL_i)/I_{avg.}$ The components also include percentage contribution from time spent in the pipeline ($14/I_{avg}$) and time in the scheduler (S/I_{avg}).

Based on Figure 10, the three most dominant sources of performance overhead are memory latency, retried instructions, and I/O-transplant latency. For the majority of the workloads, main mem-

ory stalls contribute a significant fraction of the average instruction interval; this is expected as we interleave 16 processor contexts onto single, direct-mapped caches. Vortex and crafty experience the highest rate of memory misses and are also additionally penalized by increased queuing latency in the memory system. Recall that our memory controllers are hosted on a second FPGA—the interchip link between the two FPGAs limits the rate at which multiple outstanding memory requests can be serviced.

In all of the workloads, retried instructions (including discarded computation) contribute a noticeable fraction to the average instruction interval. In our measurements, the majority of retries are caused by conflicting accesses to the L1 data cache sets in a transient state (e.g., pending cache fill). Due to the direct-mapping of the caches and a long window of vulnerability (e.g., long-latency cache fills), this event occurs frequently enough to contribute a significant overhead.

As expected, none of the user-dominated SPECint benchmarks experience any noticeable I/O except for Oracle, which experiences disk activity and queuing for I/O-transplants (note the higher I/O latency for Oracle in Table 3). One interesting result is that micro-transplants contribute little if no overhead in almost all of the workloads. This outcome is due to the fact that we were conservative in our initial chosen instruction coverage in the FPGA; these results show that more opportunity exists to omit even further behaviors from the hardware.

Despite these effects, the interleaved pipeline is still remarkably fast given the modest implementation in hardware. Future work to improve the pipeline efficiency will be to reduce latency and increase bandwidth to the DDR2 memory and to optimize the FPGA-PC link (for I/O-intensive applications). To reduce the rate of contention in the d-cache (retries), more intelligent scheduling decisions can be used to schedule conflict-free memory accesses.

6. CONCLUSION

In this paper we presented the PROTOFLEX simulation architecture for FPGA-accelerated functional full-system multiprocessor simulation. The BlueSPARC simulator is the first instantiation of this architecture that simulates a 16-way UltraSPARC III SMP server. The resulting simulator embodies the two key concepts of PROTOFLEX: hybrid simulation with transplanting and multiprocessor interleaving. Our evaluation of BlueSPARC showed a simulation throughput as high as 63 MIPS. The evaluation further showed a significant speedup, 39x on average and 49x best-case, relative to Simics with instrumentation enabled.

In the long run, we plan to scale up the BlueSPARC simulator to combine tens of interleaved engines to achieve an aggregate of 1000 MIPS to support the simulations of large (>100-way) multiprocessor systems. To succeed at that scale, we also need to extend the PROTOFLEX simulation architecture with means to virtualize the required main memory capacity expected of a multiprocessor system at that scale. We are investigating the feasibility and performance impact of employing demand-paging against a larger but slower backing storage (e.g., disk or FLASH memory) to provide the illusion of the required main memory capacity.

Acknowledgements. Funding for this work was provided in part by grants from the C2S2 Marco Center, NSF CCR-509356, and an IBM Faculty Award. We thank Xilinx for their generous FPGA and tool donations. We also thank Bluespec for their tool donations and support. We thank our colleagues in the RAMP [WPO07] and TRUSS projects for their interaction and feedback.

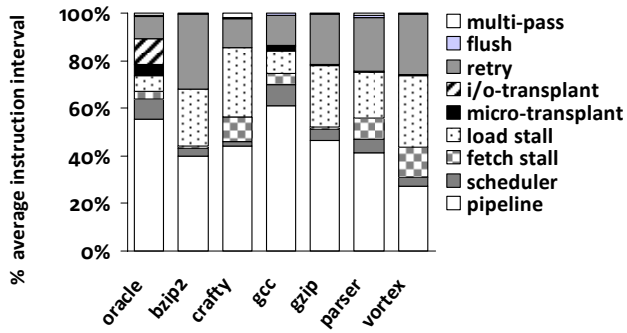

Figure 10 Average contribution of each long-latency event class to the average instruction interval (I_{avg}).

7. REFERENCES

[BE05] F. Bellard. QEMU, a Fast and Portable Dynamic Translator. In Proceedings of USENIX 2005, FREENIX Track, April 2005.

[BDH06] N. L. Binkert, R. G. Dreslinski, L. R. Hsu, K. T. Lim, A. G. Saidi, S. K. Reinhardt. The M5 Simulator: Modeling Networked Systems. IEEE Micro, vol. 26, no. 4, pp. 52-60, July/August, 2006.

[Blue06] Bluespec, Inc. http://www.bluespec.com, 2006.

[CNH07] E. S. Chung, E. Nurvitadhi, J. C. Hoe, K. Mai, B. Falsafi. ProtoFlex: FPGA-Accelerated Hybrid Functional Simulator. NSF NGS Workshop at IPDPS, March 2007.

[CSK07] D. Chiou, D. Sunwoo, J. Kim, N. Patil, W. Reinhart, E. Johnson, J. Keefe, and H. Angepat. FPGA-Accelerated Simulation Technologies (FAST): Fast, Full-System, Cycle-Accurate Simulators. In Proceedings of the International Symposium on Microarchitecture, December 2007.

[CWB05] C. Chang, J. Wawrzynek, R. W. Broderson. BEE2: A High-End Reconfigurable Computing System. IEEE Design and Test of Computers, March 2005.

[LW95] U. Legedza and W. E. Weihl. Reducing synchronization overhead in parallel simulation. Workshop on Parallel and Distributed Simulation, May 1996.

[LYK07] S. L. Lu, P. Yiannacouras, R. Kassa, M. Konow, T. Suh. An FPGA-based Pentium® in a complete desktop system. In Proceedings of the International Symposium on Field Programmable Gate Arrays, February 2007.

[MCE02] P. S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt, and B. Werner. Simics: A full system simulation platform. IEEE Computer, 35(2):50-58, February 2002.

[MSB05] M. M. K. Martin, D. J. Sorin, B. M. Beckmann, M. R. Marty, M. Xu, A. R. Alameldeen, K. E. Moore, M. D. Hill, and D. A. Wood. Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset. Computer Architecture News, September 2005.

[NFS04] F. D. Nussbaum, A. Fedorova, and C. Small. An overview of the Sam CMT simulator kit, Technical Report TR-2004-133, Sun Microsystems Research Labs, February 2004.

[OBI95] K. Öner, L. A. Barroso, S. Iman, J. Jeong, K. Ramamurthy, M. Dubois. The Design of RPM: An FPGA-based Multiprocessor Emulator. In Proc. of International Symposium on Field Programmable Gate Arrays, February 1995.

[PFH06] D. A. Penry, D. Fay, D. Hodgdon, R. Wells, G. Schelle, D. I. August and D. Connors. Exploiting Parallelism and Structure to Accelerate the Simulation of Chip Multi-processors. In Proceedings of the 12th International Symposium on High-Performance Computer Architecture, February 2006.

[RHW95] M. Rosenblum, S. A. Herrod, E. Witchel, and A. Gupta. Fast and accurate multiprocessor simulation: The SimOS approach. IEEE Parallel and Distributed Technology, 3(4), Fall 1995.

[VH07] D. Vahia, P. Hartke. OpenSPARC T1 on Xilinx FPGAs - Updates. Given on 6/14/2007 at RAMP Retreat, June 2007.

[WCN07] Sewook Wee, Jared Casper, Njuguna Njoroge, Yuriy Tesylar, Daxia Ge, Christos Kozyrakis, Kunle Olukotun. A practical FPGA-based framework for novel CMP research. In Proceedings of the International Symposium on Field Programmable Gate Arrays, February 2007.

[WPO07] J. Wawrzynek, D. Patterson, M. Oskin, S. L. Lu, C. Kozyrakis, J. C. Hoe, D. Chiou, K. Asanovic. RAMP: Research Accelerator for Multiple Processors. IEEE Micro, 27(2):46-57, March-April 2007.

[WWF06] T. F. Wenisch, R. E. Wunderlich, M. Ferdman, A. Ailamaki, B. Falsafi, and J. C. Hoe. SimFlex: statistical sampling of computer system simulation. IEEE Micro, 26(4):18-31, Jul-Aug 2006.