PROTOFLEX: FPGA-accelerated Hybrid Functional Simulation

Eric S. Chung, Eriko Nurvitadhi, James C. Hoe, Babak Falsafi, Ken Mai
Computer Architecture Laboratory at Carnegie Mellon (CALCM)
{echung, enurvita, jhoe, babak, kenmai}@ece.cmu.edu
http://www.ece.cmu.edu/~simflex

Abstract. *PROTOFLEX is an FPGA-accelerated hybrid simulation/emulation platform designed to support large-scale multiprocessor hardware and software research. Unlike prior attempts at FPGA multiprocessor system emulators, PROTOFLEX emulates full-system fidelity—i.e., runs stock commercial operating systems with I/O support. This is accomplished without undue effort by leveraging a hybrid emulation technique called transplanting. Our transplant technology uses FPGAs to accelerate only common-case behaviors while relegating infrequent, complex behaviors (e.g., I/O devices) to software simulation. By working in concert with existing full-system simulators, transplanting avoids the costly and unnecessary construction of the entire target system in FPGA. We report preliminary findings from a working hybrid PROTOFLEX emulator of an UltraSPARC workstation running Solaris 8.*

We have also started developing a novel multiprocessor emulation approach that interleaves the execution of many (10s to 100s) processor contexts onto a shared emulation engine. This approach decouples the scale and complexity of the FPGA host from the simulated system size but nevertheless enables us to scale the desired emulation performance by the number of emulation engines used. Together, the transplant and interleaving techniques will enable us to develop full-system FPGA emulators of up to thousands of processors without an overwhelming development effort.

1. INTRODUCTION

After years of focus on uniprocessor performance, the "power wall" has overnight driven the microprocessor hardware and software industry down the multicore path. This abrupt transition has left everyone at a loss about the designs of future multi-core and multiprocessor hardware and software. Recently, full-system multiprocessor simulators have emerged as the research vehicle of choice in response to the shift towards studying and evaluating experimental multiprocessor systems [MCE02, MSB05, RHWG95]. These simulators have nevertheless been developed to run on single-threaded hosts. This severely limits the speed and maximum size of simulated systems, even on high-end workstations.

Conveniently, today's FPGAs are large and fast enough to provide a scalable alternative to software simulation. Renewed interest in FPGA-based solutions has led to initiatives such as RAMP [AAC05] to develop large-scale (1000-way) emulation platforms. By supplying a platform that is fast and large enough, experimental systems can be rapidly evaluated and co-developed by software researchers, a practice that is infeasible with simulators for systems larger than several processors.

The transition from software-based simulation into FPGA-based emulation carries a non-trivial price. Today's state-of-the-art multiprocessor research demands first-class support for *full-system* simulation—the ability to run commercial applications such as databases and web servers with unmodified operating systems and I/O subsystems. For example, existing software simulators (e.g., Virtutech Simics [MCE02]) are capable of modeling complete, enterprise-level computer systems including CPUs, memory, disks, and networks. A complete port from full-system software simulation into FPGA emulation necessitates reproducing every hardware unit in detail (e.g., SCSI controller), which significantly exceeds current FPGA capabilities and requires detailed design knowledge of each system component.

Furthermore, in a conventional FPGA emulation approach (mapping the target system directly onto the FPGA host), the development complexity scales commensurately with the size of the simulated system. Developing and integrating 1000 processors in an FPGA platform is an overwhelming effort compared to a simple parameter change in a software-based simulator. Together, the *full-system* and the *scaling* complexities prohibitively elevate the threshold-of-entry for practical FPGA emulation. In reconciling these complexity challenges, the PROTOFLEX FPGA-accelerated hybrid simulation project is developing two important enabling techniques: hybrid transplant simulation and multiple-context emulation engines.

Hybrid transplant simulation. Only a small subset of total system behaviors contributes most of actual runtime in a full-system simulation. Many complex behaviors (e.g., disk I/O) are exceedingly rare and benefit little from FPGA acceleration. As a result, we have developed a hybrid **functional**[1] simulator that attains the performance benefits of FPGA hardware concurrency for the common operations (e.g., ALU instructions), while maintaining full-system fidelity with software simulations of infrequent, complex behaviors (e.g., I/O). Our "transplant" technology enables a simulated processor to switch dynamically between the FPGA and the simulator host at runtime. A processor can always fallback to simulation for behaviors unimplemented in FPGA. This enables the FPGA hardware development effort to focus *solely* on

[1] Fast, large-scale **functional** emulators provide important benefits for accelerating large-scale architecture and software research. In particular, a fast functional simulator addresses the key bottleneck in well-established cycle-accurate sampling-based techniques [WWF06]. Section 5 explains in more detail.

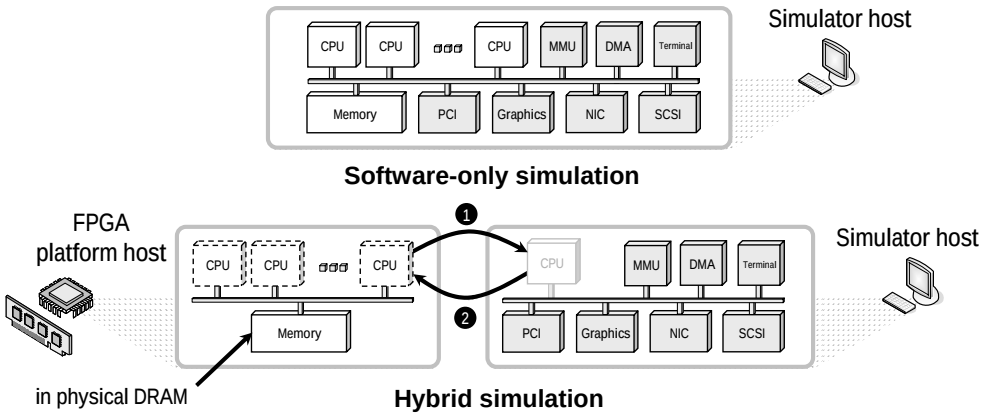

Figure 1 Partitioning a simulated target system across FPGA emulation and software simulation in the PROTOFLEX hybrid simulator.

common-case behaviors that would actually benefit from FPGA acceleration.

Multiple-context emulation engines. We address *scaling complexity* by decoupling the size and complexity of the FPGA host system from the size of the target multiprocessor system. This is achieved by mapping multiple simulated processors onto single multiple-context emulation engines hosted on FPGAs. In contrast to conventional approaches, our technique allows integrating as many emulation engines as needed *solely* based on target emulation performance and *not* the size of the simulated target system. Our initial estimates show that a 1000-way emulation system performing at 1000 MIPS (sufficient to conduct research) can be built out of merely tens of emulation engines hosted on a small number of FPGAs.

Paper outline. Section 2 provides details of our transplant technology. Section 3 describes our multiprocessor multiple-context processor emulation approach. Section 4 reports our current status on a working hybrid PROTOFLEX emulator of an UltraSPARC workstation running Solaris 8. Section 5 discusses applications, and we conclude in Section 6.

2. TRANSPLANT TECHNOLOGY

Figure 1 offers a high-level view of the PROTOFLEX hybrid-simulation technology. We begin with an existing complete software simulator (e.g., Virtutech Simics [MCE02]) that already supports stand-alone full-system execution executing on a workstation. From this full-system simulator (top), we select the performance-dominating components to implement for FPGA-emulation in a hybrid-simulation (bottom). For example, **Figure 1** shows that main memory is completely implemented in hardware (bottom-left). When a software-simulated CPU or DMA I/O device accesses memory, it is in fact accessing a hardware memory module through a memory controller on the FPGA emulation platform. Furthermore, in the example, the CPUs are shown as emulated in an FPGA, but using an incomplete model (dotted CPU modules). When an FPGA-hosted CPU model encounters an unimplemented behavior (e.g., a page table walk following a TLB miss), the emulated CPU instance is suspended; thereafter, its state is "transplanted" (❶ in **Figure 1**) to its fall-back instance in the software simulator (which is a slow but complete CPU model in the software simulator). The software-simulated CPU instance performs the unimplemented behavior before transplanting the state back to the FPGA emulated CPU instance (❷ in **Figure 1**). The remaining components are simulated entirely in software (e.g., disk storage and network interfaces, etc).

An underlying transplant runtime system of hardware and software wrappers encapsulates all components and ensures that transplants and cross-host interactions are transparent so that the illusion of a complete system is preserved regardless of component-to-host associations. Since the target processor is partitioned between FPGA emulation and software simulation, it must transplant dynamically between the FPGA and the simulator hosts depending on the behavior it encounters at runtime. For instance, a processor emulated on the FPGA executes user-level instructions until it encounters a TLB miss handler that is only implemented in the simulator. The PROTOFLEX runtime transplant system handles this by transplanting the processor component's state from the FPGA-hosted processor model into its corresponding simulated processor in Simics. Next, the simulator executes and completes the TLB miss handling operation. Finally, the processor component's updated state is transplanted back to the FPGA-hosted model to resume accelerated emulation. It should be clarified that all components in the target system run concurrently on their respective hosts (processors hosted on FPGAs and disks hosted in simulation run concurrently).

Micro-transplants. While transplants provide a fall-back mechanism for arbitrary behaviors (complex instructions, I/O accesses, TLB misses, etc), they also incur expensive overheads—this includes time to marshal data over a cross-host connection and execute in sequential software simulation on a workstation (in our implementation, a total of 10ms per roundtrip or 1,000,000 cycles at 100MHz). **Figure 2** (left) illustrates an example. Suppose

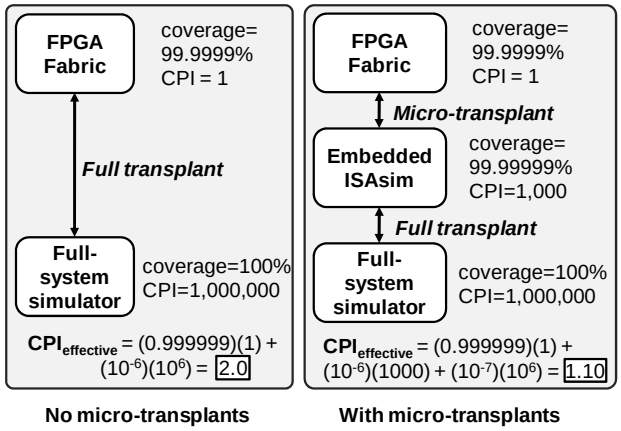

Figure 2 Improving performance with micro-transplants

a PROTOFLEX processor model in FPGA has CPI=1 and could handle 999,999 out of 1,000,000 dynamic instructions encountered (i.e., transplants just once every million instructions). Due to the high transplant penalty (1 million cycles), effective CPI would increase from 1 to 2 (losing half of ideal performance).

To overcome this severe limitation, we developed a technique called **micro-transplanting**, which substantially reduces the rate of expensive transplants. We observe that the majority of transplants (unimplemented complex instructions) can be filtered and serviced by a simple software simulation kernel running on a nearby embedded processor. The effect on performance is similar to cache hierarchies—the embedded processor is analogous to a "cache" that completes most of the unimplemented instruction behaviors to avoid the exorbitant penalty of transplanting to the full-system simulator (the "backing main memory").

Modern FPGAs conveniently provide embedded on-chip hard or soft service processors (e.g., Xilinx FPGA's embedded PowerPC405) which can be used to carry out micro-transplants (i.e., simulate unimplemented instruction behaviors). The software simulation kernel on an embedded processor could be developed with relative ease for capturing the complete instruction set behaviors; thus only processor interactions with simulated entities such as an I/O device necessitate transplants to the full-system simulator.

Figure 2 (right) illustrates how micro-transplants dramatically reduce the overhead of transplants. In the example, micro-transplants (now instead of transplants) occur once every million instructions but only incur 1000 cycles each time, resulting in negligible CPI increase. Because micro-transplants act as a filter, full transplants now only occur once every 10,000,000 instructions, resulting in a small effective CPI increase from 1 to 1.1.

Aside from reducing transplants, micro-transplants facilitate more flexible tradeoffs between FPGA hardware and software implementation. For example, implementing a rare double-precision divide instruction di-

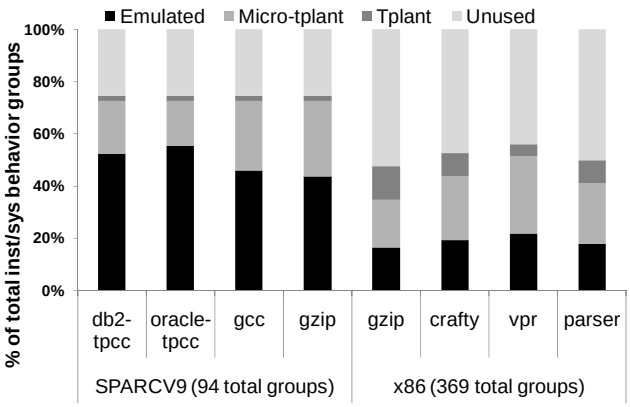

Figure 3 Partitioning results from linear programming (SPECINT apps are within 10% ideal performance) (TPCC apps are within 30% ideal performance)

rectly in reconfigurable logic is expensive in development effort and in FPGA logic resources. If the instruction occurs sufficiently rare enough in a given workload, simulating the instruction in software on the nearby embedded processor requires little effort and incurs a negligible overhead while saving resources and development effort. The added option of micro-transplants raises the question of how to systematically decide what instruction behaviors are implemented in FPGA and what behaviors are relegated to simulation (either micro-transplants or full-transplants). This assignment should be based on a multi-constraint optimization over the following factors:

1. How difficult is it to implement the behavior in FPGA (add⇒easy; pagetable walk⇒hard)
2. How much FPGA logic resource is required by the behavior (add⇒cheap; FP div⇒expensive)
3. What is the dynamic frequency of a behavior (add⇒frequent; memory-mapped I/O ⇒infrequent)
4. What is the performance gain between emulating, micro-transplanting, or transplanting a behavior?
5. What is the desired simulation performance target?

Figure 3 shows the breakdown of example assignments made by a linear programming solver that attempts to minimize the hardware development effort, subject to a performance floor. Examples are given in the context of both SPARCV9 and x86 full-system emulators. For these examples, we have classified the instructions and system-level behaviors into "equivalence" groups based on similarity (94 groups for SPARCV9 vs. 369 groups for x86). Implementing one behavior in a group in FPGA would effectively support all of the nearly-identical behaviors in the same group (e.g., "ADD" and "ADD with condition codes" belong in the same group). The vertical axis plots the percentage of all groups assigned to a specific host (i.e., FPGA emulated, micro-transplanted, fully-transplanted, or never encountered during profiling). We set our linear programming constraints to permit solu-

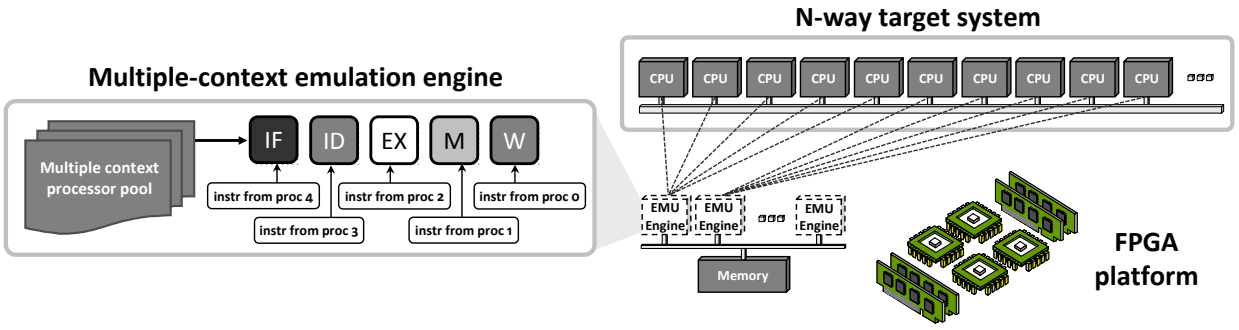

Figure 5 PROTOFLEX[MP] emulation platform

tions that are within 10% (30% for TPCC apps[2]) of ideal performance based on CPI estimates for our microarchitecture (see Section 4). Ideal performance assumes zero micro-transplant or transplant overheads. As expected, relaxing the target performance requirement allows us to implement only a subset of behaviors in FPGA hardware (approximately 50% for SPARCV9 and 20% for x86). In general, the transplant technology applied with our linear solver methodology facilitates a systematic way to trade away performance for lowered development effort and resources. This substantially mitigates the *full-system complexity* challenge. In the next section, we discuss a systematic technique for addressing *scaling complexity*.

3. PROTOFLEX[MP]

The most important challenge for PROTOFLEX is to achieve an unprecedented level of parallel functional simulation performance beyond the capabilities of today's software-based simulators. Before presenting our approach, we first analyze how fast a useful multiprocessor simulator needs to be.

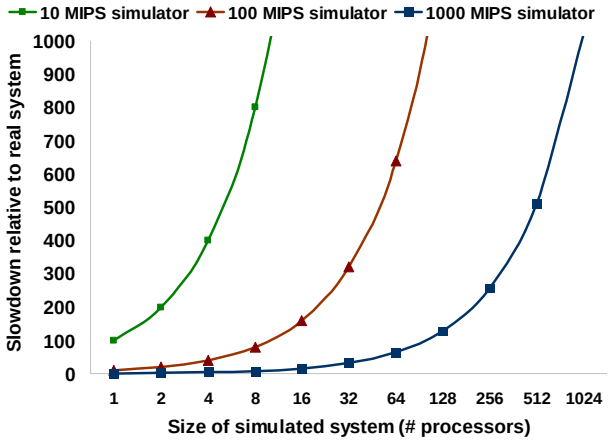

Figure 4 Simulation slowdown versus system size

[2] Our 16-way TPCC workloads exhibit a much wider and frequent range of behaviors. Specifically, the I/O rate is high and cannot be filtered by micro-transplants. As a result, no linear programming solutions exist with less than 30% overhead.

MP Simulation Performance. **Figure 4** plots the user perceived performance of a simulated target system relative to a real system as the number of simulated processors is increased. We assume "real" systems have processors with nominally 1000-MIPS performance each. We show three lines corresponding to simulators with aggregate instruction simulation rates of 10 MIPS, 100 MIPS and 1000 MIPS.

Typically, performance simulation studies have been possible at as much as 1000x to 10,000x slowdown; conducting interactive software research on new experimental platforms can tolerate no more than 100x slowdown. For example, Virtutech Simics is a commercial simulator that targets interactive software development at roughly a 10 to 100x slowdown.

The middle line in **Figure 4** corresponds to the fastest software-based simulators today (e.g., Simics) with aggregate instruction throughputs of up to 100 MIPS. This would yield a 10x perceived slowdown in uniprocessor simulation–but this aggregate throughput must be divided when the number of processors in the simulated system grows. From the figure, we observe that a 100-MIPS simulator (current single-threaded software-only simulation technology) at best is practical for studying up to tens of processors. The far-right line in **Figure 4** would suggest we need a 1000 to 10,000 MIPS simulator to support effective hardware and software research of a 1000-way multiprocessor system.

The naive approach to construct a multiprocessor emulator is to replicate 1000 FPGA 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 overwhelming and the final aggregate throughput would be 100 to 1000x faster than needed. Can we trade the excess simulation performance for a more realistic hardware development effort?

A Performance-Centric Approach. From the above insight, the goal of PROTOFLEX[MP] is to develop an emulator for a 1000-node system with an aggregate throughput of 1000 MIPS and with greatly reduced implementation complexity. Our approach is based on the idea of using multiple-context emulation engines to decouple the size

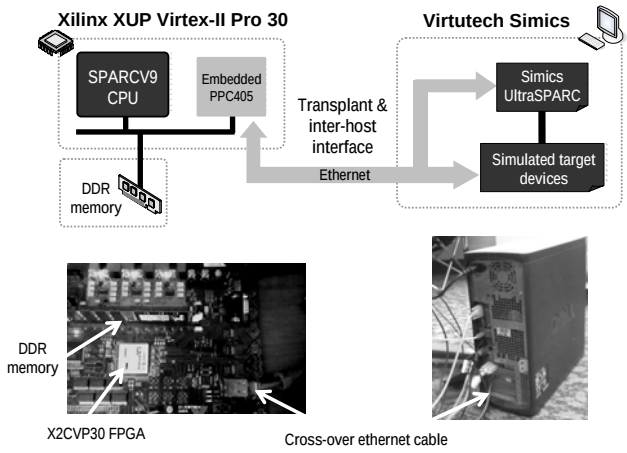

Figure 6 Initial uniprocessor proof-of-concept

of the simulated system from the required size of the FPGA host system. Instead, the required size of the FPGA host system should only be a function of the desired emulation performance.

Figure 5 illustrates the anatomy of our emulation system, where multiple simulated processors in a large-scale target system are mapped to share a small number of emulation engines (i.e. 10 simulated processors map to 10 interleaved contexts in 1 emulation engine). These engines consist of ordinary instruction pipelines but are augmented with static interleaving multithreading support that issues an instruction from a different, rotating processor context each cycle.

This idea closely resembles statically interleaved multithreaded instruction pipelines (e.g. HEP barrel processor) and comes with the associated implementation advantages. First, with enough available contexts to keep the engine occupied, it is possible to design a deep pipeline without sacrificing emulation performance. Deeper pipelines can improve clock frequency, and reduce development complexity of emulation engines for complex ISAs such as x86 (recall that even though the pipeline is deep, there is no need to implement hazard detection or forwarding). Second, long-latency events such as accesses to main memory or transplant execution can be overlapped with other contexts doing useful work.

With static interleaving, an emulation engine should comfortably reach 100 MIPS (at 100 MHz on today's FPGAs). Thus, only 10 emulation engines are needed to deliver the desired aggregate throughput of 1000 MIPS. Combining the tens of emulation engines in a FPGA emulation system is far simpler than integrating together a bona-fide 1000-way system in the naïve multiprocessor emulation approach.

4. CURRENT STATUS

As a proof-of-concept, we have completed a fully-operational hybrid simulator of a uniprocessor UltraSPARC server (Sun Microsystems Sun Enterprise 3800, 1-CPU) running unmodified Solaris 8. By leveraging the Simics full-system simulator and the Bluespec high-level hardware design environment [Blue06], this development progressed quickly from start to operation in only six months by one student. Perhaps even more surprisingly, this proof-of-concept hybrid simulator runs on inexpensive commodity hardware—a standard Linux PC and a $300 Xilinx XUP demo board. The hybrid-simulator currently achieves as high as 16MIPS[3] on integer SPEC benchmarks running under Solaris 8.

Implementation. **Figure 6** (top) shows a block diagram of our proof-of-concept hybrid-simulator implementation. On the right, a standard Linux PC executes the Virtutech Simics full-system simulator. The emulation platform on the left is a Xilinx XUP demo board with an FPGA, Ethernet interface, and memory. The Linux PC and the FPGA are connected by standard 100BT Ethernet (see bottom). On the Xilinx X2CVP30 FPGA, we implement a partial UltraSPARC III model that shares a bus with physical DRAM and the embedded PowerPC405 hard core, which forms the FPGA-side of the transplant runtime system and service routines.

When starting a simulation from scratch, the booting of the simulated target system is hosted completely in simulation. This skips past "atypical" instructions used during the boot-up process that never appear again in the target application. Once booted, we use Simics's checkpointing facility to capture a snapshot of the architectural state. Subsequent PROTOFLEX hybrid simulations start directly from this checkpoint[4]. When starting from a checkpoint, the embedded PowerPC405 initializes the physical DDR memory with the target system's main memory (from a Simics checkpoint). Next the SPARC core state is transplanted from Simics to the FPGA host. From this point forward, full-system emulation proceeds with hybrid transplantation as described in Section 2.

Processor Core Development. Although a number of synthesizable SPARC RTL models are publicly available, they are described at a too low-level abstraction for practical use in a high-level functional simulator. Instead, we opted to develop our own UltraSPARC model using Bluespec, a high-level operation-centric hardware description language [Blue06]. Bluespec lets us capture ISA behaviors in a descriptive and human-readable format. This enables a more malleable processor model that can be maintained and modified by the user community at large. Our core model can be automatically compiled for FPGA synthesis. Our first implementation is a multi-cycle machine (CPI_{ideal}=6) and runs at 100MHz on the

[3] There are full-system simulators (Virtutech Simics) that can deliver 10 to 100 MIPS. However, Simics' performance erodes sharply if any instrumentation or modification is made to the default configuration. In contrast, our hybrid-simulator can be fully instrumented (e.g., cache or branch predictor simulation) with zero impact on performance.

[4] No modifications to Simics checkpoints are necessary. Therefore, our hybrid simulator is fully backwards-compatible with any existing software checkpoints generated by Simics. This allows us to leverage our large, existing library of tuned workloads originally created for software simulation.

Xilinx XC2VP30-7 FPGA. (We expect CPI to decrease to 1 when we transition next to the interleaved pipelined implementation described in Section 3.) With 8kB I/D caches, it currently consumes 16K LUTs (50% of the entire FPGA).

We are also pursuing an effort to develop an x86-based high-level functional model in Bluespec. The transplant technology will benefit our x86 implementation even further by allowing us to omit x86's large number of rarely-used legacy complex instructions.

5. APPLYING FAST FUNCTIONAL SIMULATION

Interactive architecture and software development. A new architecture requires a correctly matched software base to demonstrate its full potential. Software engineers (in applications, compilers, and operating systems) are generally unwilling to devote serious development effort when given only a software simulator. The PROTOFLEX hybrid simulator offers sufficient execution speed to support non-trivial software development activities. In a similar vein, in order to extract meaningful data using commercial workloads, commercially shipped application binaries (e.g., databases and web servers) require many rounds of tuning calibration to optimally match the hardware configuration and performance characteristics. This activity is excruciatingly slow at software simulator speeds.

Even when real hardware exists, developing multithreaded applications can be challenging due to nondeterminism and the lack of observability in the underlying execution platform. A PROTOFLEX hybrid simulator can be instrumented to expose relevant system internal activities (race-conditions, performance counters, etc.). A PROTOFLEX hybrid simulator can be made completely deterministic (in replay) to allow detailed investigation of timing-related bugs.

Simulation sampling. Although PROTOFLEX is a functional simulator, it addresses one of the key bottlenecks in performance studies using simulation sampling. Simulation sampling research has shown that it is possible to achieve very accurate estimation of uniprocessor and multi-processor performance by simulating an exceedingly small fraction of the total benchmark in cycle-accurate mode [WWF06]. In short, the performance of the detailed software cycle-accurate simulator is itself inconsequential to the latency of performance data collection. The performance bottleneck instead 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 micro-architectural structures (e.g., caches). For even medium-scale multiprocessor systems (16- to 32-way), this process is the bottleneck in simulation turn-around time (consuming days to weeks). As system sizes scale, the turnaround time grows commensurately.

PROTOFLEX is a perfect complement to sampling-based techniques by enabling accelerated functional warming over long periods of execution. PROTOFLEX provides enough flexibility to instrument and probe any architectural state (e.g., extracting warmed cache state) with virtually no slowdown (unlike software simulators). The level of performance offered by PROTOFLEX would enable for the first time, practical performance simulation studies of systems with 100s to 1000s of processors.

6. CONCLUSION

FPGA-based emulation is being pursued as an attractive alternative to software-based simulation. However, the transition from software to hardware introduces *full-system* and *scaling* complexities. PROTOFLEX addresses these challenges with hybrid simulation and multiple-context emulation. We have already completed an initial proof-of-concept for a uniprocessor full-system hybrid emulation. We are currently extending our infrastructure to support multiprocessor functional simulation by modifying the current processor core models to support multiple-context emulation as described in Section 3.

Beyond this, there are other practical issues that require addressing. For instance, unsynchronized host communications between simulated and emulated components can lead to nondeterministic behaviors. Deterministic replay is an invaluable feature in multithreaded software debugging and critical for experimental reproducibility in architectural studies. Another issue is memory capacity—how to support the scale of main memory capacity needed by a 1000-way system in a cost and performance effective way?

Acknowledgements. This work was supported in part by grants and equipment from two NSF Career awards, and NSF grant CCR-509356.

7. REFERENCES

[AAC05] Arvind, K. Asanovic, D. Chiou, J. C. Hoe, C. Kozyrakis, S.-L. Lu, M. Oskin, D. Patterson, J. Rabaey, J. Wawrzynek. RAMP: Research accelerator for multiple processors – a community vision for a shared experimental parallel HW/SW platform. *Technical Report UCB/CSD-05-1412*, September 2005.

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

[LW95] U. Legedza and W. E. Weihl. Reducing synchronization overhead in parallel simulation. In *Proc. Workshop on Parallel and Distributed Simulation*, 1995.

[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.

[RHWG95] 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.

[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.