A Practical FPGA-based Framework for Novel CMP Research

Sewook Wee, Jared Casper, Njuguna Njoroge, Yuriy Tesylar, Daxia Ge
Christos Kozyrakis and Kunle Olukotun

Computer Systems Laboratory
Stanford University
tcc_fpga_xtreme@lists.stanford.edu

ABSTRACT

Chip-multiprocessors are quickly gaining momentum in all segments of computing. However, the practical success of CMPs strongly depends on addressing the difficulty of multithreaded application development. To address this challenge, it is necessary to co-develop new CMP architecture with novel programming models. Currently, architecture research relies on software simulators which are too slow to facilitate interesting experiments with CMP software without using small datasets or significantly reducing the level of detail in the simulated models.

An alternative to simulation is to exploit the rich capabilities of modern FPGAs to create FPGA-based platforms for novel CMP research. This paper presents *ATLAS*, the first prototype for CMPs with hardware support for *Transactional Memory* (TM), a technology aiming to simplify parallel programming. ATLAS uses the BEE2 multi-FPGA board to provide a system with 8 PowerPC cores that run at 100MHz and runs Linux. ATLAS provides significant benefits for CMP research such as 100x performance improvement over a software simulator and good visibility that helps with software tuning and architectural improvements. In addition to presenting and evaluating ATLAS, we share our observations about building a FPGA-based framework for CMP research. Specifically, we address issues such as overall performance, challenges of mapping ASIC-style CMP RTL on to FPGAs, software support, the selection criteria for the base processor, and the challenges of using pre-designed IP libraries.

Categories and Subject Descriptors: C.4 [Performance of System]: Modeling Techniques

General Terms: Design, Experimentation, Performance

Keywords: Chip Multi-processor, Transactional Memory, FPGA-based emulation

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
FPGA'07, February 18–20, 2007, Monterey, California, USA.
Copyright 2007 ACM 978-1-59593-600-4/07/0002 ...$5.00.

1. INTRODUCTION

Processor vendors are turning *en masse* towards chip-multiprocessors (*CMPs*) as a practical way to turn increasing transistor budgets into scalable performance. Nevertheless, the practical success of CMP-based systems is limited by the difficulty of parallel programming [1]. While in some systems we can utilize CMP cores by running many programs concurrently, parallel programming is necessary in order to reduce the execution time of one program. Existing models for multithreaded programming using locks are challenging for most programmers as they introduce a trade-off between performance and correctness. The use of coarse-grain locks makes it easy to write a correct parallel program but it limits concurrency. The use of fine-grain locks reveals additional concurrency but often leads to races, deadlocks, livelocks and other difficult bugs [2].

Transactional Memory (*TM*) [3] has been proposed as a promising technology that can simplify concurrency management. TM allows programmers to define coarse-grain parallel tasks (transactions) that will be executed atomically and in isolation. Using optimistic concurrency, TM executes these tasks in parallel on a CMP, providing performance similar to that of fine-grain locks. Recently, there has been significant research on efficient software and hardware TM implementations from both academia and industry [4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]. Hardware support for TM (*HTM*) is considered necessary in order to eliminate the overheads of managing concurrent execution of transactions and to allow TM programs to use existing software libraries without the need for recompilation.

To develop a technology like TM, collaborative research on both hardware and software aspects is necessary. Unfortunately, software researchers are waiting for real hardware with fast TM support before they develop and evaluate a significant volume of TM-based software. Similarly, hardware researchers cannot justify the introduction of new architectures before sufficient software is available. Moreover, it is difficult to optimize new hardware designs without feedback from their software evaluations. To date, software simulation has been the primary tool to allow collaborative hardware and software research. However, simulation is not a practical environment for parallel software development which typically involves large programs and datasets, for which debugging and performance tuning on a simulator is too slow. Experimental chip prototypes can be used to speedup TM software research, but due to their fixed nature, they cannot help with the exploration of hard-

ware alternatives. Additionally, prototype chips take long to develop, hence they cannot help with collaborative hardware/software research in the early stages of a project.

Conveniently, FPGA systems are now a viable platform for CMP research, both hardware and software. Today's FPGAs integrate one or more processor cores, multi-ported SRAM blocks, and rich I/O interfaces within a dense fabric of hundreds of thousands of reconfigurable logic cells. Given the availability of multi-FPGA boards and the wide variety of pre-designed IP cores, it is possible to prototype novel CMP architectures in a full-system environment. Such prototypes allow for innovative hardware research, where designers can "tape out" an improved version of the system on a daily basis. Additionally, FPGA-based CMP prototypes can be fast enough (50 to 200MHz) to allow software research at speeds that are orders of magnitude faster than simulation. Overall, the advantages of FPGAs for CMP research has attracted significant attention and has lead to initiatives such as RAMP [15].

In this paper, we present the architecture and design experiences from ATLAS, the first FPGA-based framework for research on CMPs with hardware support for transactional memory. ATLAS is also the first operational design from the RAMP initiative. ATLAS uses the BEE2 multi-FPGA board [16] to provide a full system with 8 PowerPC cores and a memory hierarchy with TM support. The system runs Linux and exhibits good performance efficiency on a range of parallel applications. ATLAS' primary goal is to provide an excellent framework for research on TM software. Despite running at 100MHz, it is 100 times faster than a simulator running on high-end workstations. Hence, it can significantly shorten the application development and evaluation cycle, even when large datasets are used. Moreover, ATLAS provides extensive support for performance monitoring and tuning which helps software developers identify performance bottlenecks and generates detailed insights for hardware researchers.

In addition to presenting and evaluating ATLAS, this paper presents our observations and insights on using FPGA systems to create flexible frameworks for CMP research. The specific lessons from ATLAS are:

- **Lesson 1** Despite the lower clock frequency as compared to modern workstations, FPGA-based CMP environments can outperform software simulators by two orders of magnitude.

- **Lesson 2** The major challenges when mapping ASIC-style RTL for a CMP system on an FPGA are highly associative memory structures, large lower-level caches, and interconnect fabrics that span across FPGAs.

- **Lesson 3** For projects with a heavy focus on software research, it is important to select a base processor core with rich software support and features. Other criteria, such as flexibility and size, are secondary.

- **Lesson 4** The use of IP core libraries is crucial to shortening the system development time. However, there is a need for cores with improved interfaces for debugging and performance profiling purposes.

The rest of the paper is organized as follows. Section 2 overviews TM and related work. Section 3 presents the ATLAS hardware and software architecture. Section 4 quantitatively compares ATLAS with a software simulator modeling the same target architecture. Section 5 summarizes the major lessons from the ATLAS development and Section 6 concludes the paper.

2. BACKGROUND

2.1 TM at a glance

To parallelize an application, a programmer must break the code up into multiple threads that can execute in parallel. The programmer must also synchronize the threads when they potentially operate on the same data in memory. The conventional synchronization approach is to use locks. If the system supports transactional memory, the programmer can synchronize threads simply by wrapping all the code that operates on the potentially shared data into a transaction. Transactions are guaranteed by the system to appear to execute atomically and isolated, even if multiple transactions are executed concurrently. TM systems achieve high performance through optimistic concurrency. A transaction runs without acquiring locks, optimistically assuming no other transaction operates concurrently on the same data. If this is true at the end of its execution, the transaction commits its writes to shared memory. If not, the transaction violates, its writes are rolled back, and it is re-executed.

A TM system must implement the following mechanisms: (1) isolation of stores until the transaction commits; (2) conflict detection between concurrent transactions; (3) atomic commit of stores to shared memory; (4) rollback of stores when conflicts are detected. Conflict detection requires tracking the addresses read (read-set) and written (write-set) by each transaction. A conflict occurs when the write-set of one transaction intersects with the read-set of another concurrently executing transaction. These mechanisms can be implemented either with hardware support (HTM) [4, 5, 6, 7] or in a software only manner (STM) [8, 9, 10, 11, 12, 13, 14]. HTM systems support transactional mechanisms at minimal overheads and make the implementation details transparent to software.

HTM systems implement speculative buffering and track read- and write-sets in caches [4]. The data caches in CMP cores are large enough to buffer the state for the transactions in most applications [17] and simple virtualization techniques can handle the rare case of a cache overflow [6, 18]. For conflict detection, a HTM system can utilize the cache coherence protocol. As messages are exchanged between cores to locate data on cache misses, it is possible to detect the overlap between a read-set and a write-set that triggers a conflict. Simulation studies have shown that such hardware TM implementations can allow multithreaded programs to synchronize in a high performance manner using fairly simple, coarse-grain, transactions in their code [19].

2.2 Related Research

FPGA-based emulation platforms for parallel systems date to the RPM project [20]. Given the shift towards CMPs and the increasing FPGA densities, there is renewed interest for such systems. FAST [21] is an FPGA-based framework for modeling CMPs with MIPS cores. While it is a suitable framework for CMP memory system research, it lacks support for significant software development. The RAMP[15] project strives to develop a general framework for FPGA-

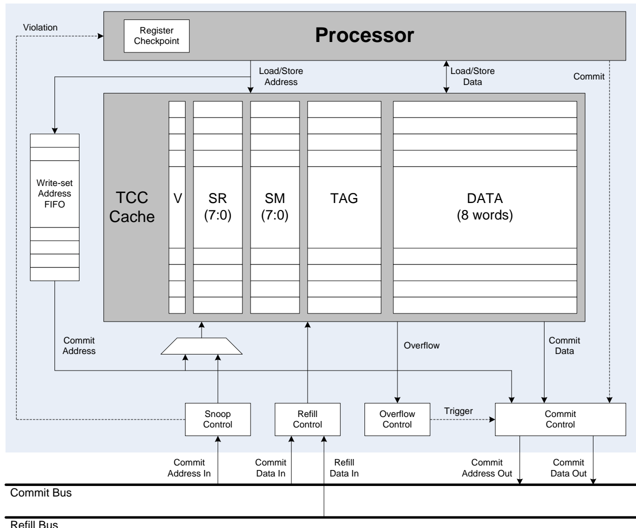


Figure 1: The data cache organization for ATLAS.

based hardware and software CMP research and currently uses the BEE2 board [16].

There are also efforts to use FPGAs to accelerate simulation systems. The idea is to move the time-consuming detailed modeling of hardware structures into the FPGA, while functional modeling is still simulated [22, 23, 24]. Even though this provides a faster simulator, it sits far from a full system prototype. Hong et al. have used FPGAs in a working system to quickly simulate various configurations for lower level caches [25]. Finally, there are efforts to map large out-of-order processors to FPGAs as a prototyping medium [26].

3. THE ATLAS CMP SYSTEM

ATLAS is the first full-system framework for a CMP with transactional memory support. This section presents its basic architecture, the hardware design, and its software environment. The prototype is currently operational at 100MHz, runs the GNU/Linux operating system, and runs multithreaded applications that use transactional memory.

3.1 The TCC TM Architecture

ATLAS prototypes the Transactional Coherence and Consistency (*TCC*) architecture for hardware-based transactional memory [19]. TCC assumes a set of processor cores with private first-level caches that are connected through a snooping bus to the shared memory (shared caches and DRAM). The cores execute transactions speculatively, while tracking the read- and write-sets in the data cache organization shown in Figure 1. As a core performs loads and stores within a transaction, it sets the speculatively-read (*SR*) and speculatively-modified (*SM*) bits to indicate that the corresponding word is now part of the transaction's read-set or write-set, respectively. Also, the first time a word is written in a specific cache line, the cache pushes a pointer to it into the write-set address FIFO. The cache operates as a write-buffer, isolating all writes from shared memory until the transaction completes.

At the end of the transaction, the core arbitrates for permission to commit the write-set to the shared memory. While the system executes multiple transactions in parallel, only one is allowed to commit at any point in time. Once granted the commit token, the core uses the write-set address FIFO to traverse the cache and commit the transaction's writes. To detect conflicts, all other cores snoop the commit messages, searching their data caches for the committed addresses. If a word currently committed belongs to the read-set of another transaction (SR bit is set), a violation is triggered for the snooping transaction. When a transaction violates, the cache undoes its effect by invalidating its write-set from the cache (lines with SM bits set). Register checkpointing at the boundaries of transactions is also necessary to undo register updates. Because there is only one transaction committing, the committing transaction is always guaranteed to complete. Before the commit is over, the processor clears the SR and SM bits in the cache.

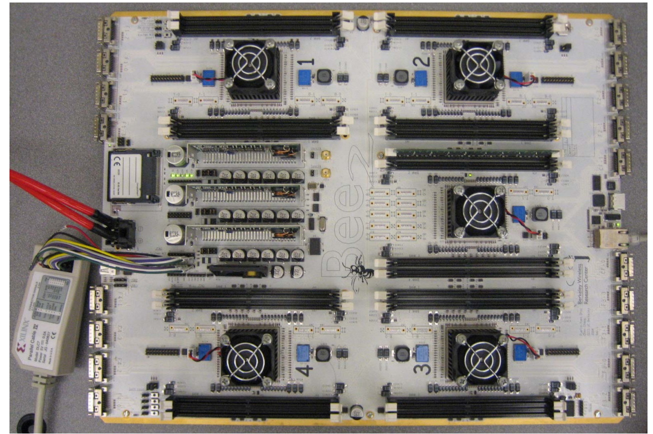


Figure 2: The BEE2 multi-FPGA board used to prototype ATLAS.

An interesting feature of TCC is that communication between cores occurs only at transaction boundaries. This point is the only place where the caches are kept coherent and access ordering is enforced. TCC uses this simple commit mechanism to replace the complex cache coherence protocols like MESI. This feature is possible because well synchronized programs should access truly shared data only within user-defined transactions. To handle legacy codes with benign races, TCC uses periodic commits outside of transactions to keep caches coherent.

A user-defined transaction may be large enough to overflow the data cache. In this case, the core arbitrates immediately for the commit token, performs a partial commit, and does not release the token until it reaches the real end of the transaction. Such overflow events slowdown commits but are rare in tuned programs. In Section 3.3, we discuss how ATLAS provides explicit support to help programmers identify and quickly eliminate overflows and other bottlenecks in their parallel code.

3.2 The ATLAS Hardware Design

ATLAS implements the TCC architecture for CMPs with transactional memory support. ATLAS includes 8 PowerPC 405 cores that run multithreaded code for applications and a ninth core that handles the operating system and I/O devices. The design has been mapped onto the BEE2 multi-FPGA board shown in Figure 2 [16]. ATLAS is also part of the RAMP project that aims at developing FPGA-based

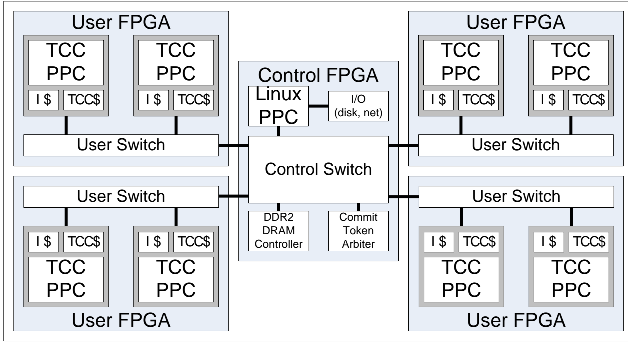
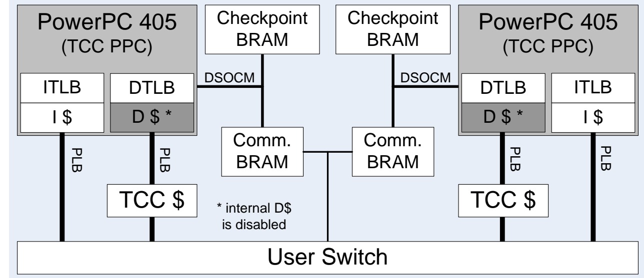
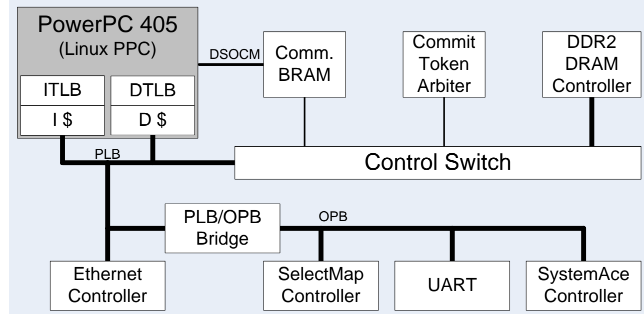


Figure 3: Block diagram of the ATLAS system.



(a) User FPGA



(b) Control FPGA

Figure 4: Block diagram of the User (a) and Control (b) FPGAs.

technology for prototyping modern CMP systems [15].

Figure 3 illustrates how ATLAS is mapped to the BEE2 board. The four outer FPGAs, labeled as User FPGAs, are connected in a star topology through the 5th FPGA, designated as the Control FPGA. Figure 4(a) shows the block diagram of the User FPGA. Each FPGA includes two PowerPC 405 cores enhanced with a TCC data cache (*TCC PPCs*). We use the hardcore PowerPC cores in the Virtex II-Pro FPGAs. The data cache design, written in synthesizable Verilog, is attached to the PowerPC cores through the IBM Processor Local Bus (*PLB*). The cache has 32 byte lines and can be 1, 2, or 4-way set associative (each way is 8 KB, resulting in cache sizes of 8, 16, or 32 KB). The write-set address FIFO can be configured to be 0.5, 1, 2, 4 or 8 KB. The internal data cache in the PowerPC cores is disabled. The TCC cache is in turn connected to a network switch (shared by two cores) that forwards cache misses and commit requests to the control FPGA through a central switch. For instructions, we use the built-in instruction cache in each PowerPC core. Like the TCC caches, the local switch forwards the cache refill requests to the Control FPGA. As we run Linux on ATLAS, each core activates the built-in TLB (unified, fully-associative, 64 entries). We also connect two SRAM blocks to the processor through the On-Chip Memory (*OCM*) bus. The first SRAM (*Checkpoint BRAM*) serves for register checkpointing (in software), while the second SRAM (*Communication BRAM*) serves to coordinate system calls and exceptions for Linux (refer to Section 3.3). The two SRAMs are implemented on the FPGA using the on-chip block SRAM (*BRAM*) blocks available.

The Control FPGA is a hub that connects all processors to the shared memory and I/O devices. The current design does not use secondary caches, though this is not fundamental. As depicted in Figure 4(b), the Control Switch interfaces with the commit token arbiter, the DDR controller and the PowerPC 405 core that runs Linux (*Linux PPC*). The Linux PPC uses its built-in caches but its memory writes are broadcast to all other processors for coherence purposes. All traffic sent through switches is packetized with a single packet format to simplify routing around the system.

Another function of the Control FPGA is to provide I/O capabilities. A number of I/O peripherals are connected to Linux PPC, including an Ethernet controller, a UART controller, a SelectMap controller for programming the user FPGAs through a Linux device driver, and a SystemACE controller that interfaces to a CompactFlash card. Using Network File System (NFS), ATLAS can access external storage over the Ethernet.

3.3 The ATLAS Software Design

The ATLAS software stack consists of an API for programming with transactions and the system software.

For application programming, the user partitions work into parallel threads and defines atomic transactions within each thread. As explained in [27], this approach allows for both non-blocking coarse-grain synchronization of parallel code and speculative parallelization of sequential code. We currently provide an API for C-based transactional programming, which permits users to define transaction boundaries. The application is compiled with a regular C compiler (gcc in our case) and linked with the ATLAS library that provides optimized assembly-based implementations of the API calls. At the start of a transaction, the API call checkpoints the processor's register file, clears SM and SR bits in the data cache, and registers the transaction with the commit token arbiter. When the transaction completes, the API call first requests the commit token and, when granted, triggers the commit of the write-set to shared memory.

The ATLAS system software is summarized in Figure 5. ATLAS runs Linux only on the Linux PPC. The eight TCC PPCs in the user FPGAs run a simple runtime kernel that coordinates with Linux. This approach is similar to the Intel MISP concept for efficient CMPs [28]. A user application starts on the Linux PPC as a regular user application under Linux. On the first call to the ATLAS library, the context of the application (private stack pointer, program counter, and process information) is transferred to the primary TCC PPC (core number 0). The primary TCC PPC executes the application and invokes the other 7 processors as needed on parallel regions in the program. The runtime kernel also contains a transaction rollback handler triggered when a data cache detects a violation for the currently executing transaction. The handler invalidates the write-set in

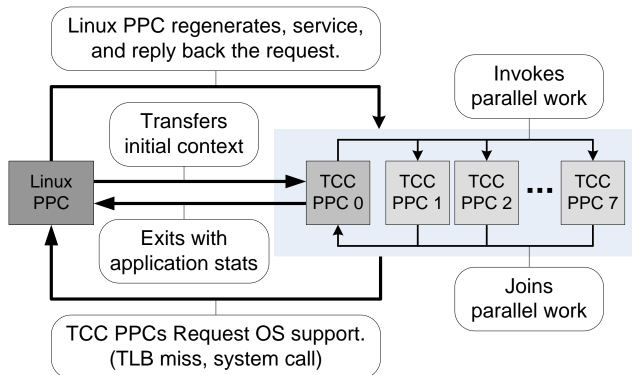


Figure 5: Overview of the ATLAS system software.

CPUs	8 PowerPC 405 cores (TCC) at 100 Hz 1 PowerPC 405 core (Linux) at 300 MHz 16 KB 2-way I-cache (9 cores) 32 KB 4-way TCC D-cache (8 cores) 16 KB 2-way PowerPC internal D-cache (1 core)
Main Memory	512 MB DDR2 at 200 MHz
I/O	10/100 Mbps Ethernet, RS232 UART, 512 MB Compact Flash
OS	Montavista 3.1 (Linux kernel ver. 2.4.30)
EDA Tools	Xilinx EDK 7.1i
User FPGA	Xilinx XC2VP70, 17,641 LUTs (26%), 212 KB BRAMs (32%)
Ctrl FPGA	Xilinx XC2VP70, 16,284 LUTs (24%), 66 KB BRAMs (10%)

Table 1: ATLAS design statistics.

the data cache, restores the register checkpoint, and restarts the transaction.

During the program execution, the Linux PPC core handles all interrupts due to external devices. It also handles any OS functions needed to support the execution on the TCC PPCs, like system calls or exceptions such as a TLB miss. For example, on a TLB exception, the TCC PPC sends the faulting address to the Linux PPC. The Linux PPC regenerates the exception, runs the corresponding OS code to resolve it (e.g. access the page table for the proper translation entry), and sends the information back to the requesting TCC PPC. We handle other exceptions and system calls in a similar manner (I/O, memory allocation etc.). Hence, the user code can run assuming full OS support at any point of the program. At the current scale of the ATLAS system, a single core running the OS is sufficient to serve eight cores running application code. Using a single core for the OS allows us to run conventional Linux without special consideration for concurrency in the OS code. In larger scale configurations of ATLAS, the number of OS cores may need to be scaled. In this case, we will have to port SMP Linux onto ATLAS. We will also have the opportunity to explore the use of transactional synchronization in system code.

An important part of the ATLAS system is the support for performance tuning of user applications. Transactional memory makes it easy to write a correct parallel program. Nevertheless, a program may still include performance bottlenecks such as frequent transaction violations or expensive overflows [27]. To help the user identify the most significant bottlenecks, ATLAS includes a profiler framework that utilizes performance counters built into the PowerPC cores and additional counters and filters introduced in the TCC cache [29]. The hardware tracks the occurrence of all violations and overflows, the corresponding instruction and data references that triggered them, and an approximation of their cost. The profiler software uses this information to identify the most important problems and pinpoint the offending variables or lines in the user source code. The accurate feedback on performance bottlenecks allows ATLAS users to quickly tune their applications as they can focus on the important issues and avoid the need to understand the whole application in detail.

4. EVALUATION

Table 1 presents the default configuration of the ATLAS design and its resource utilization. The design is currently operational at 100MHz and runs Linux.

4.1 Methodology

To evaluate ATLAS, we ran five applications on both ATLAS and TASSEL, an established software simulator for the TCC architecture. The applications include three scientific benchmarks (radix, mp3d, ocean); a hashtable microbenchmark that performs random insert, remove, and lookup operations on a hashtable; and vacation, a benchmark that emulates a travel reservation management and database system. All applications include multiple threads that operate on data in shared memory in an irregular manner. We use the following evaluation metrics:

- **Wall Clock Time**: We observe how much faster ATLAS executes each application compared to TASSEL by wall clock time (as seen by the user).
- **Speedup**: We examine the estimated architectural speedup of each application normalized to the uniprocessor execution time in each system. We compare the TASSEL and ATLAS speedup to verify that the FPGA design constraints do not affect the accuracy of execution results.
- **Execution Time Breakdown**: To fully understand the speedup trends, we measure and compare the execution time breakdown for each system. The execution time is divided into the following components: "Useful" time, stall cycles due to cache misses, time spent committing transactions, idle cycles due to thread imbalance (synchronization), and cycles lost due to violations. Such breakdowns are important for both verification purposes and to provide programmers and architects with further profiling information.

4.2 Comparing ATLAS against TASSEL

TASSEL is a PowerPC-based execution-driven simulator of TCC that runs on workstations with PowerPC G5 processors (2 GHz). TASSEL is coded in C++ and uses fast forwarding to avoid time-consuming simulation for the initialization code of each benchmark. During fast forwarding,

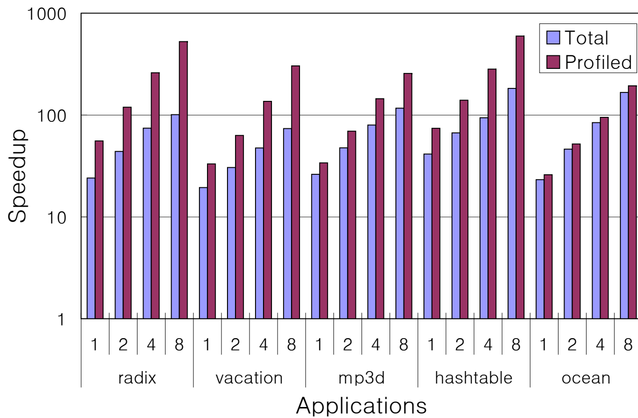


Figure 6: Speed up of the wall clock time to run the application on ATLAS vs. TASSEL. Total is the total time used by the process in Linux and Profiled is the user-specified interesting section of the application. Note the logarithmic scale on the axes.

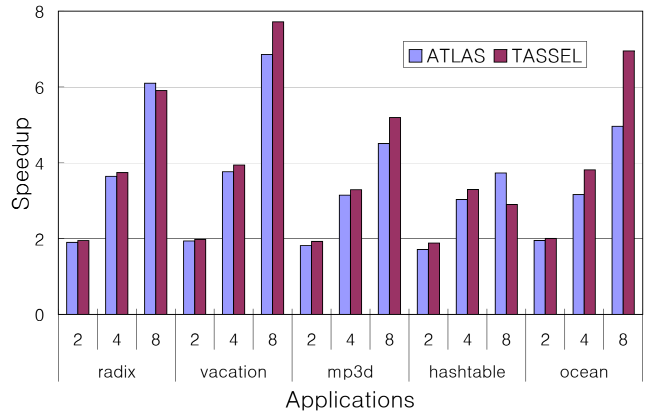


Figure 7: Speed up of real execution time in ATLAS and reported execution time from TASSEL. Numbers are normalized to the one processor case of each application and platform (ATLAS to ATLAS and TASSEL to TASSEL).

TASSEL uses direct execution on the G5 system. This feature reduces simulation time significantly. Since TASSEL is a fully-flexible simulator, the modeled hardware is not limited to the capabilities of FPGAs. Hence, there are some implementational features that ATLAS and TASSEL model differently. For instance, TASSEL assumes gang clears of the speculative state bits in the cache, while ATLAS has to clear each cacheline's speculative bits one cycle at a time. TASSEL assumes single-cycle L1 cache hits, while ATLAS takes 13 cycles to access the TCC cache from the PowerPC core. TASSEL's interconnection network is a bus-based model, whereas ATLAS is a hierarchical topology constrained by the inter-FPGA connectivity on the BEE2 board. In some cases, ATLAS provides more features than TASSEL. Most significantly, ATLAS applications run on top of Linux, so ATLAS uses virtual memory and TLBs. Naturally, this implies that ATLAS application runs include the time for TLB miss handling and page faults.

It is important to note that TASSEL is not fully cycle-accurate, hence we expect some variance when compared to the behavior of any real TCC implementation. Similarly, we do not claim that ATLAS is a cycle-accurate emulator of an ASIC TCC prototype since some FPGA constraints introduce additional latencies (longer L1 hit-time, multi-cycle gang-operations and so forth). However, we do claim that ATLAS is sufficiently accurate for software research on transactional memory, our primary target for the system. Specifically, software researchers are much more concerned with parallel efficiency and speedup rather than the precision of cycle counts. Moreover, these researchers are interested in the relative cost of various performance bottlenecks such as violations or cache misses as they determine the focus of performance tuning efforts. Hence, our evaluation focuses on comparing the speedups and execution time breakdowns reported by TASSEL and ATLAS. By comparing results from these two independent approximations of a real TCC system, the TCC architecture itself is further validated and the merits of transactional memory are exposed. Also note that the insights generated by software research are the basis for further research on improving the TCC hardware architecture.

4.3 Wall Clock Time Comparison

Figure 6 compares the wall clock execution time of ATLAS and TASSEL. The left bar shows the advantage of ATLAS over TASSEL for the total execution time, which is the total time spent running the application and is the amount of time the user waits for results. The graph demonstrates that, as the number of modeled processors increases, ATLAS is an increasingly faster tool to use. Moreover, most of the applications are in the range of two orders of magnitude in speedup, which from an user's perspective is the difference between hours and minutes of waiting for application results. The right bar in Figure 6 is ATLAS' advantage when we considered only the profiled portion of the application, which is the portion of the application that TASSEL actually simulates and does not fast-forward past. In this case, the ATLAS advantage is often higher than 100x. As applications for CMPs use increasingly larger datasets and more complex algorithms, we expect that the advantage of of ATLAS over TASSEL will be closer to that reported by the profiled bars.

The advantage of ATLAS over TASSEL highly depends on the amount of time each application spends on operating system services (TLB misses and page faults). TASSEL does not model OS overheads. ATLAS uses a single PowerPC 405 core to run the operating system requests for all threads and this can become a bottleneck. For example, radix works on a 2 MB dataset, so it has a large number of data TLB misses due to the small 4 KB pages our Linux kernel enlists. In fact, at eight processors, the contention for the Linux PPC increases by almost 2,000 cycles per TLB miss (up from 600 cycles in the uniprocessor case). This observation has prompted us to investigate methods to address these overheads that can otherwise cancel some of the benefits from parallelization and transactional memory. A software simulator alone would not allow us to identify this issue.

4.4 Speedup and Execution Time Breakdown

Figure 7 compares the reported speedups by ATLAS and TASSEL for each application run. The speedup is calculated by comparing an n-processor run to a sequential run

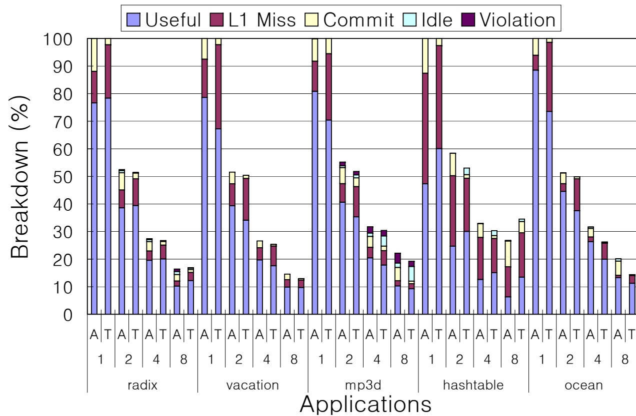


Figure 8: Breakdown of profiled real execution time in ATLAS and reported execution time from TASSEL. Numbers are normalized to the one processor case of each method.

in the same system. For TASSEL, we use the predicted execution time by the simulator, not the time the simulator took to run. ATLAS and TASSEL generally show similar speedup trends relative to their respective uniprocessor execution times. However, TASSEL tends to report that applications should scale better than ATLAS does, and the hashtable benchmark demonstrates some deviant behavior. Hashtable's anomalous behavior and ATLAS' predilection to serialize more at eight processor than TASSEL is explained by exploring the execution time breakdown chart depicted in Figure 8. One trend that is salient across the breakdown chart is that ATLAS' commit time occupies a larger share of execution time than TASSEL's. To explain this trend we note the following: First, ATLAS' commit overhead is larger because to clear the speculative state tags in the cache it takes almost 260 cycles in ATLAS instead of the one-cycle gang clear in TASSEL. Second, as the number of processors scales, the commit time to useful time ratio increases since the commit overhead remains constant while the useful work time decreases. Thus, at eight processors, ATLAS does not amortize its commit overhead (since it is larger) as well as TASSEL does, resulting in slightly less speedup.

The graph indicates that hashtable does not scale to eight processors due to the poor locality it exhibits in its data references as it performs random accesses to a large hash table. The mediocre scaling can be attributed to the poor locality it exhibits in its data references as it performs random accesses to a large hash table. With a small number of processors, the memory bandwidth is not saturated by all the cache refills. However, at eight CPUs, the number of cache refills sent to the ICN cause significant contention. For ATLAS, the cache refills escalate from 57 cycles in the uniprocessor case (no contention) to 80 cycles at eight CPUs. TASSEL experiences an even larger refill time increase from 40 cycles in one processor to 146 cycles in eight processors. The discrepancies are due to the different ICN topologies. TASSEL models a shared-bus, which apparently saturates in hashtable, whereas ATLAS' network is a two-level arbitration ICN as depicted in Figure 3).

In summary, there is good correlation between the speedup trends observed in ATLAS and TASSEL, suggesting that ATLAS can confidently propel software research on transactional memory.

5. ATLAS DESIGN EXPERIENCE

The design experience from ATLAS allows us to make some observations and identify challenges for constructing FPGA-based frameworks for CMP research. This section summarizes the most important issues.

Lesson 1: Despite the lower clock frequency as compared to modern workstations, FPGA-based CMP environments can outperform software simulators by two orders of magnitude.

As demonstrated in Section 4, ATLAS outperforms the corresponding software simulator, in terms of wallclock time, by an average of 124x for the applications studied. This result is particularly interesting since the simulator is running on a 2GHz workstation while ATLAS runs at 100MHz. The main advantage of the FPGA-based approach is that it actually uses multiple processors instead of multiplexing multiple simulated processors on one actual processor, which implies that the advantage of FPGA-based systems will improve as the number of cores in CMPs increases. Second, the software simulator spends most of its time approximating the timing of the processors, the complex memory system, and the interconnect network. On the FPGA system, all these component exist with actual hardware implementations and operate at the system clock frequency. Overall, this result provides an interesting data point for the debate within the computer architecture community about the proper tools to use for CMP research, particularly given the importance of novel software development and the use of large datasets that can stress a CMP system. The combination of these factors provides an opportunity to benefit from the rich resources and the inherent parallelism of modern FPGAs.

Lesson 2: The major challenges when mapping ASIC-style RTL for a CMP system on an FPGA are highly associative memory structures, large lower-level caches, and interconnect fabrics that span across FPGAs.

Traditionally, FPGAs trade-off logic density for configurability. Nevertheless, modern FPGAs are increasingly denser and equipped with basic blocks such as dual-ported SRAMs, DSP blocks, and embedded hardcore processors to name a few. The improved capabilities of FPGAs allow for CMP designs of significant complexity to be mapped to a few FPGA chips. Nonetheless, there are some challenges faced when mapping certain components of ASIC CMP RTL on FPGA chips.

The first challenge is mapping highly-associative structures that are commonly used in CMP designs to implement content-addressable memories (*CAM*), hardware schedulers, victim caches, and state bits with gang set/reset capabilities. FPGA vendors now embed CAMs in their chips. Although CAMs have many useful applications for searching, they do not help with structures that use gang set or reset operations. In TCC, for example, we must clear all the SM and SR bits in the cache on transaction boundaries. On a violation, we must clear the valid bits for cache-lines with SM bits set (conditional gang reset). These operations are fast in ASIC prototypes using full-custom memory cell design. If

	PowerPC 405	MicroBlaze	Nios II	Leon (V3.0)	Tensilica	OpenRISC
Type	Hard	Soft	Soft	Soft	Soft	Soft
Software Support	A+	C	C	B+	B	B
MMU	Yes	No	No	Yes	Yes	Yes
Area usage*	Minimal	Small	Small	Large	Large	Medium
Max. Frequency**	Fast	Medium	Medium	Slow	Slow	Slow
FPU	No	Single Precision	Single Precision	Configurable	Configurable	No

* Small: 2∼3,000 LUTs, Medium: 5,000 10,000 LUTs, Large: 20,000∼30,000 LUTs
** Fast: ∼400MHz, Medium: ∼200MHz, Slow: 50∼70MHz

Table 2: Available Processor IPs

the same RTL is mapped to an FPGA, the result is highly sub-optimal. Our original TCC cache RTL used ASIC-style assumptions and used LUT flip-flops for the 17 kbits of SM and SR bits. This design consumed 112,000 LUTs or 160% of the total FPGA resources. To reduce utilization, we modified the RTL to allocate these bits into BRAMs. Clearing these bits now requires 257 cycles of pipelined read-modify-write operations using both BRAM ports. Nonetheless, an operation that the rest of the design expects to be atomic now takes multiple cycles, which created several design bugs. Hence, dealing with highly associative structures can lead to both density and correctness issues.

Second, very large memory arrays can be challenging for FPGAs. While L1 caches for CMP cores are small and can be handled easily by BRAMs, it is not the case for large secondary caches which range into the multiple Mbytes. To accommodate large secondary caches, one can use external memories, either SRAMs [21] or DRAMs, depending on the specific system used. Note, however, that the use of external memories may further complicate control and timing issues for the modeled architecture.

An additional challenge is porting designs that span multiple FPGAs. CMPs have a distinct advantage over large uniprocessor designs because they are naturally partitioned. Thus, the CMP cores themselves and their private caches are not as effected by inter-FPGA latencies. However, the interconnection network (*ICN*) is often constrained by the inter-FPGA network on the specific board used. For example, our initial design used a bus to interconnect the processor cores. To map to the BEE2 board, we modified the design to use a star-like interconnect that matched the board layout. Naturally, such changes can affect latency and bandwidth assumptions in the system as illustrated by the hash table application's behavior (described in Section 4.4).

There are a few techniques one can employ to effectively deal with such design challenges. One mechanism is to virtualize the inter-FPGA boundaries to provide the illusion that the design is mapped on one large FPGA. This virtualization involves tracking the cycle count for the modeled design separately from the cycles at which the FPGA system operates [30] and properly stopping/starting design modules to accommodate the desired latencies in modeled clock cycles [30]. Hence, an inter-FPGA communication that takes 5 real clock cycles can be accounted for more or fewer modeled clock cycles. This approach can be applied at any granularity but is more efficient to handle at the boundaries of large modules that are likely to operate asynchronously anyway. We did not employ this technique as cycle-level accuracy was not a priority for ATLAS.

Lesson 3: For projects with a heavy focus on software research, it is important to select a base processor core with rich software support and features. Other criteria, such as flexibility and size, are secondary.

When architecting a CMP, one can design processor cores from scratch, or reuse an existing core design in order to reduce the overall project complexity. Table 2 presents the characteristics of the most significant cores available. Note that because most of the soft processors are configurable and FPGAs are actively evolving, numbers may vary depending on target device and configured features. Unfortunately, there is no core that universally satisfies the requirements of all users. Rather, researchers should carefully consider the priorities of their project before selecting a specific core.

For our project, rich and robust software support was the highest priority as the goal of the project is to enable software research. Thus, we considered issues such as stable OS, compiler, auxiliary libraries, and ample user group. Hence, cores without an MMU that cannot run full-featured Linux were of little interest (e.g., Nios and Microblaze). An additional consideration was the FPGA resources occupied by the cores. A compact core design allows for more resources to be used for the memory hierarchy, which is the major area of innovation for the TCC architecture. Consequently, the PowerPC 405 hardcore was an appropriate choice for ATLAS. An additional advantage of the 405 core is that it can be clocked significantly faster than any soft core.

Nevertheless, the choice of a hardcore can have disadvantages. First, we cannot modify the core in any way. This choice results in adding the TCC cache as an external data cache with a 13-cycle hit latency instead of 1-cycle. Similarly, we cannot add further instructions to facilitate faster interactions between the core and the cache. Another disadvantage of the 405 core in the Virtex-II Pro devices is that it does not provide a mechanism to connect it with a synthesized FPU. Therefore, all FP operations are emulated in software at a significant performance cost for scientific workloads. This limitation has been addressed in Virtex-4 devices. While these disadvantages are important, they were not sufficient to motivate the use of a softcore. We also took into account the design time necessary to modify and debug any new features we would introduce to the core.

We should point out that our experience with a hardcore CPU in FPGAs is not necessarily different than what many CMP designers deal with. In many cases, they must build a CMP using the pre-existing layout for a core that was not necessarily optimized for their system [31].

Lesson 4: The use of IP core libraries is crucial to shortening the system development time. However, there is a need for cores with improved interfaces for debugging and performance profiling purposes.

FPGA-based frameworks have been considered to require more effort than software simulators. This belief is mainly due to the time required for RTL development of the various smaller system components that are necessary for full-system modeling. The availability of pre-designed IP libraries has significantly reduced the design efforts for FPGAs. ATLAS makes heavy use of components available through Xilinx and RAMP such as the DRAM and ethernet controllers. Nevertheless, debugging and performance tuning with pre-designed IP is not necessarily easy. Most of these components lack debugging interfaces and profiling capabilities. Hence, it is difficult to identify and track bugs or performance bottlenecks without significant effort creating wrappers for such modules. In essence, each IP component should provide an Integrated Logic Analyzer (*ILA*) interface that allows the designer to capture detailed information and control its operation during debugging. Moreover, components should maintain statistics related to performance (stalls, contention, etc). To avoid unnecessarily penalizing the final design in terms of area or clock frequency, it should also be possible to disable the debugging and profiling features on each module. Most of the major components in ATLAS were developed with these guidelines in mind.

The PowerPC 405 core and the accompanying Xilinx software (Xilinx Microprocessor Debugger or XMD) provide good examples of hardware and software support for debugging and profiling. The capabilities of these systems provided a link through a JTAG chain from the PowerPC cores to the well-known GNU debugger. Additionally, XMD is built on top of a Tool Command Language (TCL) shell. This setup allowed us to develop custom scripts for bitstream download, system initialization, boot code download, profiling support, low-level TCC cache control, and memory range monitoring using XMD. Nevertheless, in several cases XMD does not provide execution information that would be valuable for external tools, such as program exit notification. In general, hardware and software systems for FPGA-based design should move towards increased transparency and flexibility to help researchers build aggressive full-system prototypes.

6. CONCLUSION

ATLAS is the first full-system prototype of a CMP with hardware support for transactional memory. Mapped on the BEE2 multi-FPGA board, ATLAS operates at 100MHz, runs Linux, exhibits good performance on parallel applications and outperforms our software simulator by two orders of magnitude. The ATLAS design indicates that FPGA-based frameworks can be an extremely useful tool for CMP research. Nevertheless, the ATLAS experience also indicates the challenges that researchers must face when mapping CMP designs to FPGAs. The issues include challenges in mapping ASIC-style CMP RTL on to FPGAs, the selection criteria for the base processor, and debugging and profiling support in pre-designed IP libraries.

7. ACKNOWLEDGEMENTS

This research was sponsored by the Defense Advanced Research Projects Agency (DARPA) through the Department of the Interior National Business Center under grant NBCH104009. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency (DARPA) or the U.S. Government. Additional support was also provided by the National Science Foundation Grant CCF-0444470, the Samsung Foundation of Culture, and the Stanford School of Engineering.

8. REFERENCES

[1] H. Sutter, "The free lunch is over: A fundamental turn toward concurrency in software," *Dr. Dobb's Journal*, vol. 30, March 2005.

[2] B. Lewis and D. J. Berg, *Multithreaded Programming with Pthreads*. Prentice Hall, 1998.

[3] M. Herlihy and J. E. B. Moss, "Transactional memory: Architectural support for lock-free data structures," in *Proc. of the 20th Intl. Symp. on Computer Architecture*, pp. 289–300, 1993.

[4] L. Hammond, V. Wong, M. Chen, B. D. Carlstrom, J. D. Davis, B. Hertzberg, M. K. Prabhu, H. Wijaya, C. Kozyrakis, and K. Olukotun, "Transactional memory coherence and consistency," in *Proc. of the 31st Intl. Symp. on Computer Architecture*, pp. 102–113, June 2004.

[5] C. S. Ananian, K. Asanović, B. C. Kuszmaul, C. E. Leiserson, and S. Lie, "Unbounded Transactional Memory," in *Proc. of the 11th Intl. Symp. on High-Performance Computer Architecture (HPCA'05)*, (San Franscisco, California), pp. 316–327, February 2005.

[6] R. Rajwar, M. Herlihy, and K. Lai, "Virtualizing Transactional Memory," in *ISCA '05: Proc. of the 32nd Annual Intl. Symp. on Computer Architecture*, pp. 494–505, June 2005.

[7] K. E. Moore, J. Bobba, M. J. Moravan, M. D. Hill, and D. A. Wood, "LogTM: Log-Based Transactional Memory," in *12th Intl. Conf. on High-Performance Computer Architecture*, February 2006.

[8] N. Shavit and D. Touitou, "Software transactional memory," in *Proc. of the 14th Annual ACM Symp. on Principles of Distributed Computing*, (Ottawa, Canada), pp. 204–213, August 1995.

[9] M. Herlihy, V. Luchangco, M. Moir, and I. William N. Scherer, "Software transactional memory for dynamic-sized data structures," in *PODC '03: Proc. of the twenty-second annual Symp. on Principles of distributed computing*, pp. 92–101, July 2003.

[10] T. Harris and K. Fraser, "Language support for lightweight transactions," in *OOPSLA '03: Proc. of the 18th annual ACM SIGPLAN Conf. on Object-oriented programing, systems, languages, and applications*, pp. 388–402, 2003.

[11] A. Welc, S. Jagannathan, and A. L. Hosking, "Transactional monitors for concurrent objects," in *Proc. of the European Conf. on Object-Oriented Programming* (M. Odersky, ed.), vol. 3086 of *Lecture Notes in Computer Science*, pp. 519–542, Springer-Verlag, 2004.

[12] B. Saha, A.-R. Adl-Tabatabai, R. L. Hudson, C. Cao Minh, and B. Hertzberg, "A high performance software transactional memory system for a multi-core runtime," in *PPoPP '06: Proc. of the eleventh ACM SIGPLAN Symp. on Principles and practice of parallel programming*, March 2006.

[13] M. F. Ringenburg and D. Grossman, "Atomcaml: first-class atomicity via rollback," in *ICFP '05: Proc. of the tenth ACM SIGPLAN Intl. Conf. on Functional programming*, pp. 92–104, 2005.

[14] V. J. Marathe, W. N. Scherer III, and M. L. Scott, "Adaptive Software Transactional Memory," in *19th Intl. Symp. on Distributed Computing*, September 2005.

[15] Arvind, K. Asanović, D. Chiou, J. C. Hoe, C. Kozyrakis, S.-L. Lu, M. Oskin, D. Patterson, J. Rabaey, and J. Wawrzynek, "RAMP: Research accelerator for multiple processors - a community vision for a shared experimental parallel HW/SW platform," tech. rep., 2005.

[16] C. Chang, J. Wawrzynek, and R. W. Brodersen, "BEE2: A

high-end reconfigurable computing system," *IEEE Design and Test of Computers*, vol. 22, pp. 114–125, Mar/Apr 2005.

[17] J. Chung, H. Chafi, C. Cao Minh, A. McDonald, B. D. Carlstrom, C. Kozyrakis, and K. Olukotun, "The Common Case Transactional Behavior of Multithreaded Programs," in *Proc. of the 12th Intl. Conf. on High-Performance Computer Architecture*, February 2006.

[18] J. Chung, C. Cao Minh, A. McDonald, H. Chafi, B. D. Carlstrom, T. Skare, C. Kozyrakis, and K. Olukotun, "Tradeoffs in transactional memory virtualization," in *ASPLOS-XII: Proc. of the 12th Intl. Conf. on Architectural support for programming languages and operating systems*, Oct 2006.

[19] A. McDonald, J. Chung, H. Chafi, C. Cao Minh, B. D. Carlstrom, L. Hammond, C. Kozyrakis, and K. Olukotun, "Characterization of TCC on Chip-Multiprocessors," in *PACT '05: Proc. of the 14th Intl. Conf. on Parallel Architectures and Compilation Techniques*, pp. 63–74, September 2005.

[20] K. Oner, L. A. Barroso, S. Iman, J. Jeong, K. Ramamurthy, and M. Dubois, "The design of RPM: an FPGA-based multiprocessor emulator," in *FPGA '95: Proc. of the 1995 ACM third Intl. Symp. on Field-programmable gate arrays*, pp. 60–66, 1995.

[21] J. D. Davis, S. E. Richardson, C. Charitsis, and K. Olukotun, "A chip prototyping substrate: the flexible architecture for simulation and testing (fast)," vol. 33, pp. 34–43, 2005.

[22] D. Chiou, H. Sunjeliwala, D. Sunwoo, J. Xu, and N. Patil, "Fpga-based fast, cycle-accurate, full-system simulators," in *2nd Workshop on Architecture Research using FPGA Platforms, 12th Intl. Symp. on High-Performance Computer Architecture (HPCA-12)*, February 2006.

[23] N. Dave, M. Pellauer, Arvind, and J. Emer, "Implementing a functional/timing partitioned microprocessor simulator with an fpga," in *2nd Workshop on Architecture Research using FPGA Platforms, 12th Intl. Symp. on High-Performance Computer Architecture (HPCA-12)*, February 2006.

[24] D. A. Penry, D. Fay, D. Hodgdon, R. Wells, G. Schelle, D. I. August, and D. A. Connors, "Exploiting parallelism and structure to accelerate the simulation of chip multi-processors," in *Proc. of the 12th Intl. Conf. on High-Performance Computer Architecture*, February 2006.

[25] J. Hong, E. Nurvitadhi, and S.-L. L. Lu, "Design, implementation, and verification of active cache emulator (ace)," in *FPGA '06: Proc. of the 2006 ACM/SIGDA 14th Intl. Symp. on Field programmable gate arrays*, pp. 63–72, 2006.

[26] F. J. Mesa-Martinez *et al.*, "SCOORE: Santa Cruz out-of-order RISC engine, FPGA design issues," in *Workshop on Architectural Research Prototyping (WARP), held in conjunction with ISCA-33*, 2006.

[27] L. Hammond, B. D. Carlstrom, V. Wong, B. Hertzberg, M. Chen, C. Kozyrakis, and K. Olukotun, "Programming with transactional coherence and consistency (TCC)," in *ASPLOS-XI: Proc. of the 11th Intl. Conf. on Architectural support for programming languages and operating systems*, pp. 1–13, October 2004.

[28] R. A. Hankins, G. N. Chinya, J. D. Collins, P. H. Wang, R. Rakvic, H. Wang, and J. P. Shen, "Multiple instruction stream processor," in *ISCA '06: Proc. of the 33rd Intl. Symp. on Computer Architecture*, pp. 114–127, 2006.

[29] H. Chafi, C. Cao Minh, A. McDonald, B. D. Carlstrom, J. Chung, L. Hammond, C. Kozyrakis, and K. Olukotun, "TAPE: A Transactional Application Profiling Environment," in *ICS '05: Proc. of the 19th Annual Intl. Conf. on Supercomputing*, pp. 199–208, June 2005.

[30] A. S. G. Gibeling and K. Asanović, "The RAMP architecture & description language," tech. rep., 2005.

[31] J. D. Gilbert, S. H. Hunt, D. Gunadi, and G. Srinivasa, "TULSA, A Dual P4 Core Large Shared Cache Intel Xeon Processor for the MP Server Market Segment, Intel," in *Conf. Record of Hot Chips 18*, 2006.