From Plasma to BeeFarm: Design Experience of an FPGA-based Multicore Prototype

Nehir Sonmez[1,2], Oriol Arcas[1,2], Gokhan Sayilar[3], Osman S. Unsal[1], Adrián Cristal[1,4], Ibrahim Hur[1], Satnam Singh[5], and Mateo Valero[1,2]

[1] Barcelona Supercomputing Center, Spain
[2] Computer Architecture Department, Universitat Politècnica de Catalunya
[3] Faculty of Engineering and Natural Sciences, Sabanci University, Turkey
[4] IIIA - Artif. Intelligence Research Inst. CSIC - Spanish National Research Council
[5] Microsoft Research Cambridge, United Kingdom

Abstract. In this paper, we take a MIPS-based open-source uniprocessor soft core, Plasma, and extend it to obtain the Beefarm infrastructure for FPGA-based multiprocessor emulation, a popular research topic of the last few years both in the FPGA and the computer architecture communities. We discuss various design tradeoffs and we demonstrate superior scalability through experimental results compared to traditional software instruction set simulators. Based on our experience of designing and building a complete FPGA-based multiprocessor emulation system that supports run-time and compiler infrastructure and on the actual executions of our experiments running Software Transactional Memory (STM) benchmarks, we comment on the pros, cons and future trends of using hardware-based emulation for research.

1 Introduction

This paper reports on our experience of designing and building an eight core cache-coherent shared-memory multiprocessor system on FPGA called BeeFarm to help investigate support for Transactional Memory [11, 17, 23]. The primary reason for using an FPGA-based simulator is to achieve a significantly faster simulation speed for multicore architecture research compared to the performance of software instruction set simulators. A secondary reason is that a system that uses only the FPGA fabric to model a processor may have a higher degree of fidelity since no functionality is implemented by a magical software routine. Another way to use FPGA-based emulation is to offload infrequent or slow running instructions and I/O operations to a software simulator but retain the core functionality in FPGA hardware [7]. In our work we model the entire multiprocessor system on reconfigurable logic, although commercial simulator accelerators like Palladium and automated simulator parallelization efforts also take advantage of reconfigurable technology [19].

Recent advances in multicore computer architecture research are being hindered by the inadequate performance of software-based instruction set simulators which has led many researchers to consider the use of FPGA-based emulation.

Although sequential software-based simulators are mature and it is relatively fast to make changes to the system in a high-level environment, they turn out to be slow for the simultaneous simulation of the cores of a typical multiprocessor.

The inherent advantages of using today's FPGA systems are clear: multiple hard/soft processor cores, fast SRAM blocks, DSP units, more configurable logic cells each generation on a more rapidly growing process technology than ASIC and already-tested Intellectual Property (IP) cores. There are various synthesizable open-source Register Transfer Level (RTL) models of x86, MIPS, PowerPC, SPARC, Alpha architectures which are excellent resources to start building a credible multicore system for any kind of architectural research. Furthermore, various IPs for incorporating UART, SD, Floating Point Unit (FPU), Ethernet or DDR controllers are easily accessible [1]. Although FPGA-based multiprocessor emulation has received considerable attention in the recent years, the experience and tradeoffs of building such an infrastructure from these available resources has not yet been considered. Indeed, most infrastructures developed were either (i) written from scratch using higher level HDLs, such as Bluespec, (ii) using hard cores such as PowerPC, or (iii) using proprietary cores e.g. Microblaze.

Therefore, in this work we choose a new approach: We take an existing MIPS uniprocessor core called Plasma [2] and we heavily modify and extend that to build a full multiprocessor system designed for multicore research. To obtain the Honeycomb core, the basic building block for the BeeFarm, we designed and implemented two coprocessors, one providing support for virtual memory using a Translation Lookaside Buffer (TLB), and another one encapsulating an FPU; we optimized the Plasma to make better use of the resources on our Virtex-5 FPGAs; we modified the memory architecture to enable addressing for 4 GB; we implemented extra instructions to better support exceptions and thread synchronization and we developed the BeelibC system library to support the BeeFarm system. Additionally, we designed coherent caches and developed a parameterizable system bus that accesses off-chip RAM through a DDR2 memory controller [21]. Finally, we developed a run-time system and compiler tools to support a programming environment rich enough to conduct experiments on Software Transactional Memory (STM) workloads. A hypothesis we wish to investigate is the belief that an FPGA-based emulator for multicore systems will have better scalability compared to software-based instruction set simulators. We check this hypothesis on 1–8 cores using our flexible BeeFarm infrastructure, obtaining performance speedups of up to 8x. The key contributions of this paper are:

- A description of extending Plasma for implementing the BeeFarm multiprocessor system on the BEE3 platform [9] and discussions on the tradeoffs and an analysis of the FPGA resource utilization of our approach.
- Experimental results for three benchmarks investigating support for Transactional Memory and an analysis of the performance and scalability of software simulators versus the BeeFarm system.
- An experience reporting the pros and cons of using FPGA-based multicore emulation and identification of specific challenges that need to be overcome to better support FPGA-based emulation in the future.

2 The BeeFarm System

The synthesizable MIPS R2000-compatible soft processor core Plasma was designed for embedded systems and written in VHDL. It has a configurable 2-3 stage pipeline (no hazards), a 4 KB direct-mapped L1 cache, and can address up to 64 MB of RAM. It was designed to run at a clock speed of 25 MHz, and it includes UART and Ethernet cores. It also has its own real-time operating system with some support for tasks, semaphores, timers etc. Although the original Plasma core is suitable for working with diverse research topics, it has some limitations that makes it unsuitable as the processing element of the BeeFarm system, such as the lack of virtual memory (MIPS CoProcessor 0), exceptions and floating point arithmetic (MIPS CoProcessor 1). Furthermore, the design only infers optional on-chip resources for the register file, and there is no support for multiprocessing or coherent caches.

The BEE3 platform contains four Virtex5-155T FPGAs, where each FPGA controls four DDR2 DIMMs, organized in two channels of up to 4 GB each. The DDR2 controller [21] manages one of the two channels, performs calibration and serves requests, occupying a small portion of the FPGA (around 2%). Using one controller provides sequential consistency for our multicore since there is only one address bus, reads are blocking and stall the processor pipeline.

2.1 The Honeycomb core: Extending Plasma

On Honeycomb processor (Figure 1) is implemented in a 3-stage pipeline with an optional stage for data accesses. Instructions and data words are 32-bit wide, and data can be accessed in bytes, half words (2 bytes) or words (4 bytes). In a typical ALU instruction, the program counter passes the program counter to the memory control unit, which fetches the 32-bit opcode from memory. In the next stage, the memory returns the opcode which is passed to the control unit that converts it to a 60-bit control word and forwards it to the appropriate entities through a central multiplexer (bus-mux). Cache accesses pause the CPU and can take various cycles in case of a miss.

The MMU (CP0): In order to support virtual memory, precise exceptions and operating modes, we implemented a MIPS R3000-compatible 64-entry TLB, effectively upgrading the R2000 core to an R3000, which we named Honeycomb. It provides memory management and exception handling intercepting the memory control unit datapath. There exist various approaches to implement an efficient Content Addressable Memory (CAM) on FPGAs, with configurable read/write access times, resource usage, and the technology utilized, where general-purpose LUTs or on-chip block memories can be used[5]. The use of LUT logic for medium and large CAMs and multi-cycle access are inappropriate since the TLB must translate addresses each cycle on our design. We implemented this unit with on-chip BRAM configured as a 64-entry CAM and a small 64-entry LUTRAM. Virtual patterns that are stored in the CAM give access to an index to the RAM that contains the physical value. It is driven by a half-cycle shifted clock that performs the translation in the middle of the

memory access stage so a dedicated pipeline stage is not needed. This 6-bit deep by 20-bit wide CAM occupies four BRAMs and 263 LUTs.

Double-Precision FPU (CP1): The MIPS 3010 FPU implemented in Coprocessor 1, which was designed using Xilinx Coregen, can perform IEEE 754-compatible single and double precision floating point operations. It takes up 5520 LUTs and 14 DSP units, performing FP operations and conversions in variable number of cycles (4–59). We used only 4 of the 6 integer-double-float conversion cores to save space. This optional MIPS CP1 has 32x32-bit FP registers and a parallel pipeline. The integer register file was extended to include FP registers implemented as LUTRAM. For double precision, two registers represent the low and high part of the 64-bit number and the register file was replicated to allow 64-bit (double precision) read/write access each cycle.

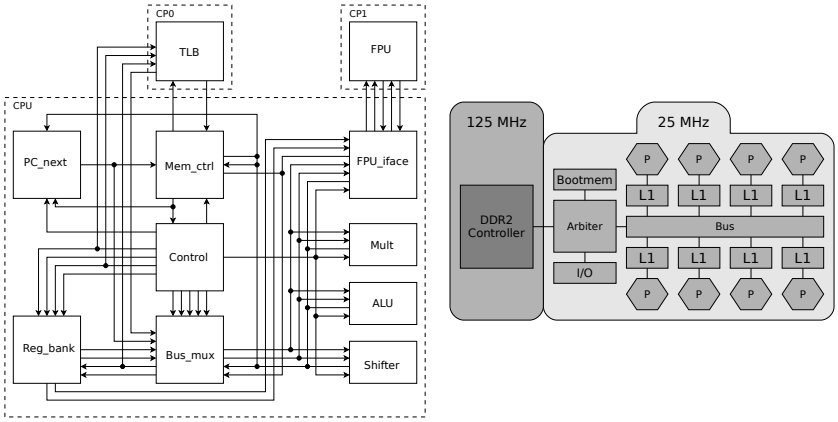

Fig. 1. Left: the Honeycomb processor. Right: the BeeFarm multiprocessor system.

Memory Map and ISA Extensions: We redesigned the memory subsystem which could originally only map 64 MB of RAM, to use up to 4 GB with configurable memory segments for the stack, bootloader, cache, debug registers, performance counters and memory-mapped I/O ports. Furthermore, we extended the Honeycomb ISA with three extra instructions borrowed from the MIPS R4000: **ERET** (Exception RETurn), to implement precise exception returns that avoid branch slot issues, **LL** (Load-Linked) and **SC** (Store Conditional), which provide hardware support for synchronization mechanisms such as Compare and Swap (CAS) or Fetch and Add (FAA). This is useful for Software TM support, as we detail in Section 2.4.

2.2 The BeeFarm Architecture

Honeycomb's 8 KB write-through L1 cache design that supports the MSI cache coherency for both data and instructions in 16-byte, direct-mapped blocks, uses 2+1 BRAMs for storing data and cache tags. The BRAM's dual-port access enables serving both CPU and bus requests in a single cycle. Reads and writes are blocking, and coherence is guaranteed by the snoopy cache invalidation protocol

that we implemented. The caches designed are interconnected with a central split-bus controlled by an arbiter, as shown in Figure 1. They snoop on the system bus to invalidate entries that match the current write address, where write accesses are processed in an absolute order. This protocol can perform invalidations as soon as the writes are issued on the write FIFOs of the DDR and it serves to find an adequate balance between efficiency and resource usage. More complex caches that demand more resources would make it difficult to implement a large multiprocessor given the limited resources present on chip. We are not interested in using large, multithreaded soft cores like OpenSPARC and Leon3 because we concentrate on designing our own TM-capable manycore emulator by upgrading a popular soft core, and reflecting on such experience.

The bus arbiter implemented interfaces the FIFOs of the DDR controller, serving requests from all processors following a round-robin scheme. The boot-up code is stored in a BRAM next to the arbiter and mapped to a configurable region of the address space. I/O ports are also mapped, and the lowest 8 KB of physical memory give access to the cache memory, becoming a useful resource during boot-up when the DDR is not yet initialized. Furthermore, the cache can be used as stack thanks to the uncached execution mode of MIPS. Such direct access to cache memory is useful for debugging, letting privileged software to read and even modify the contents of the cache.

The arbiter, the bus, caches and processors can run at a quarter of the DDR frequency (25 - 31.25 MHz), the CPU's shallow pipeline being the main cause of this upper bound on the clock. Although bus and cache frequencies could be pushed to work at 125 MHz or at an intermediate frequency, it was not desirable to decouple this subsystem from the processor because partitioning the system in many clock domains can generate tougher timing constraints, extra use of BRAM to implement asynchronous FIFOs or additional circuitry for signals that cross different clock domains. Further optimizations to the Honeycomb are possible by clocking faster all special on-chip units and including such extra circuitry.

We can fit 8 Honeycomb cores without FPUs or 4 cores with FPUs on one Virtex5-155T FPGA. The system bus can become a bottleneck not only during system execution, but also when placing and routing the design. To use all four FPGAs on the BEE3 and to support up to 40 Honeycomb cores, a distributed directory scheme that will provide system-wide memory coherency is being designed and implemented. Each FPGA will have a quarter of the memory space and a directory that is responsible for local locations and a proxy directory with information about lines from the local memory that are cached on remote nodes.

2.3 FPGA resource utilization

One of the objectives of the design is to fit the maximum number of cores while supporting a reasonable number of features.The Honeycomb core without an FPU occupies 5712 LUTs (Figure 2) on a Virtex-5 FPGA including the ALU, MULT/DIV and Shifter units, the coherent L1 cache, the TLB and the UART controller, a comparable size to the Microblaze core. The functional blocks on Honeycomb can be categorized in three groups:

Compute-intensive (DSP): Eg. ALU, MULT/DIV, Shifter, FPU can take advantage of hard DSP units. In the original Plasma core these units fit the third category, since the ALU is a combinatorial circuit, while the MULT/DIV take 32 cycles to iterate and compute. The ALU can be mapped directly on a DSP block while a MULT can be generated with Xilinx Coregen in a 35x35 multiplier utilizing an acceptable 4 DSPs and 160 LUTs. The shifter can also benefit from these 4 DSPs thanks to dynamic opmodes, however, a 32-bit divider can take anywhere between 1100 LUTs to 14 DSP units: The optimal way of combining all these operations to have a minimal design is not yet clear, we leave this as future work.

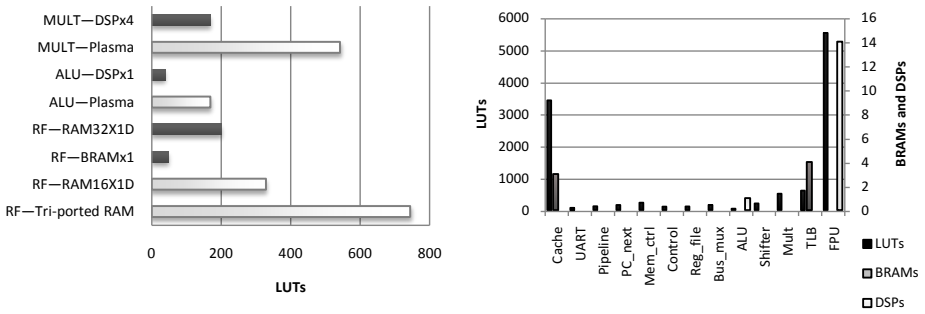

Fig. 2. Left chart: some of the available options (in 5-LUTs) to implement the Register File, the ALU and the Multiplier unit. The lighter bars indicate the choices in the original Plasma design. Right chart: LUT and BRAM usage of Honeycomb components.

Memory-intensive (BRAM/LUTRAM): Eg. Reg_Bank, Cache, TLB. The TLB is designed in a CAM, the cache and tags in BRAMs. The Reg_Bank in Plasma selects between using 4-LUT RAMs (RAM16), behaviorally describing a tri-ported RAM, or using a BRAM. The use of a large BRAM is inefficient, and the tri-ported RAM infers too many LUTs, as seen in Figure 2. When distributed LUTRAM is inferred, each 5-LUT can act as a 32-bit register on the Virtex-5, enabling two reads and one write per cycle assuming one of the read addresses is the write address. Few options to enable two reads and a write to distinct addresses on each CPU cycle are: (i) to do the accesses in two cycles, using one of the address inputs for reading or writing, (ii) to clock the register file twice as fast and do the reads and writes separately, or (iii) to duplicate the register file. Although we currently use the third approach, our design accepts either configuration. Other groups have proposed latency insensitive circuits which save resources by accessing the register file in a few cycles[25].

LUT-intensive: Eg. implementing irregular if/case structures or state machines: PC_next, Mem_ctrl, control, bus_mux, TLB logic, system bus and cache coherency logic. This category demands a high LUT utilization; one striking result in Figure 2 is that providing cache coherency occupies roughly half of the LUT resources used by the Honeycomb. Such complex state machines do not map well on FPGAs, however synthesis results show that our core would per-

form 43.7% faster on a Virtex-6 FPGA, so such irregular behavioral descriptions can still be expected to perform faster as FPGA technology advances.

Unlike the cache coherence protocol and the shared system bus that map poorly, compute-intensive units and the register bank are good matches for distributed memory that use 5-LUTs, although one still can not do a single cycle 3-ported access. BRAMs and DSP units must be used carefully, to match better the underlying FPGA architecture. Regular units that match a compute-and-store template rather than complex state machines must be fashioned. In general, we believe that caches are a bottleneck and a good research topic for multicore prototyping. There is little capacity for larger or multi-level caches on our FPGA, and it would not be easy at all to provide high levels of cache associativity.

2.4 The BeeFarm Software

Since we are not running a full Linux with all system calls implemented, we can not use the standard C library libC, so we developed a set of system libraries called BeelibC for memory allocation, I/O and string functions. Other groups process system calls and exceptions falling back to a host machine or a nearby on-chip hard processor core [20, 7]. A MIPS cross-compiler with GCC 4.3.2 and Binutils 2.19 is used to compile the programs with statically linked libraries. The cores initially boot from the read-only Bootmem that initializes the cores and the stack and then loads the RTOS kernel code into memory. One of the most attractive proposals for shared-memory CMPs has been the use of atomic instructions in Transactional Memory (TM), a new programming paradigm for deadlock-free execution of parallel code without using locks, providing optimistic concurrency by executing atomic transactions in an all-or-none manner. In case of a data inconsistency, a conflict occurs and one of the transactions has to be aborted without committing its changes, and restarted. Transactional Memory can be implemented in hardware (HTM) [11], which is fast but resource-bounded while requiring changes to the caches and the ISA, or software (STM) [10] that can be flexible at the expense of weaker performance. Specifically, we are interested in the intermediate approaches, or Hardware-assisted STM (HaSTM) which aims to accelerate an STM implementation for which we provide a framework that could be easier for conducting architectural studies. Towards this goal, we have successfully ported TinySTM [10], a lightweight and efficient word-based STM library implementation, and ran TM benchmarks on the BeeFarm, we present these results in Section 3.

3 Comparison with SW Simulators

The multiprocessor system presented in this work was designed to speed up multiprocessor architecture research, to be faster, more reliable and more scalable than software-based simulators. Its primary objective is to execute real applications in less time than popular full-system simulators. Our tests: (i) Measure the performance of the simulator platform, and not of the system simulated. (ii) Abstract away from library or OS implementation details, so that external functions

like system calls would not significantly affect the results of the benchmark. (iii) Can be easily ported to different architectures, avoiding architecture-specific implementations like synchronization primitives. (iv) Pay special attention to the scalability of the emulation, a key weakness of software multiprocessor simulators. Our emulations are not affected by the number of processors in other ways than the usual eg. memory bandwidth, contention, cache protocols.

M5 [4] is an easily modifiable "full-system simulator" that can simulate Alpha processors with cache and bus details. We believe that despite the fact that MIPS and Alpha are distinct architectures, this can be a fair comparison: Both architectures are 32 register 32-bit RISC, operate on fixed-size opcodes and the only operations that access the memory are load and store. We executed the test in M5 fast mode (with optimizations, minimum timing and profiling options), and additionally for ScalParC in a slower profiling mode with timing. The compilers used to obtain the test programs for the BeeFarm and the M5 both use GCC version 4.2, compiled with the -O2 optimization level or -O3 when possible on a 64-bit Intel Xeon E5520 server with 2x quad-core processors running at 2.26 GHz with 64 GB of DDR3 RAM and 8 MB of L3 cache memory. All results were obtained using Xilinx ISE 12.2 running on RHEL5.

3.1 BeeFarm Multicore Performance of STM Benchmarks

To test multicore performance with STM benchmarks running on the BeeFarm, we have run ScalParC from RMS-TM [13], Intruder and SSCA2 TM benchmarks from STAMP [16] which are very commonly used for TM research. We modified ScalParC with explicit calls to use TinySTM. In our experiments, ScalParC was run with a dataset with 125K records, 32 attributes and 2 classes, SSCA2 was run with problem scale 13 and Intruder with 1024 flows.

The results that are normalized to the single-core M5 executions show that while the BeeFarm can scale in a near-linear way, the M5 simulator fails to scale and the performance rapidly degrades as the core counts are increased. Figure 3 shows that the gap opens with more cores and with only four, the BeeFarm with FPU just needs fifteen minutes to run the ScalParC benchmark, an eightfold difference. The scalability of our hardware is more obvious when the abort ratio between the transactions are low and little work is repeated, so the benchmark itself is scalable. SSCA2 also benefits from the inherent parallelism of the FPGA infrastructure and the good performance of the FPU: The two-core BeeFarm takes about half of the runtime of the M5 and it shows better scalability with more cores. In this sense our FPU which takes roughly the space of a single CPU core is clearly a worthy investment for the case of this particular benchmark. Other available hardware kernels such as a quicksort core [6] would be a very useful accelerator for this particular benchmark, and such specialized cores/execution kernels could further push the advantages of multicore emulation on reconfigurable platforms.

Intruder is a very high abort rate integer-only benchmark that scales poorly, and this can be seen on both M5 and BeeFarm results. We are able to run Intruder with 8 CPUs because this design does not use FP. It performs worse on

the BeeFarm for single processor runs, however for more than two cores, it runs faster than the M5, whose scalability again degrades rapidly. Certain benchmarks with certain configurations (eg. without an FPU) for a small number of cores could result in software simulators performing faster than FPGA devices. Mature simulators that take advantage of the superior host resources could still have advantages over FPGA emulators for simulations of a small number of cores.

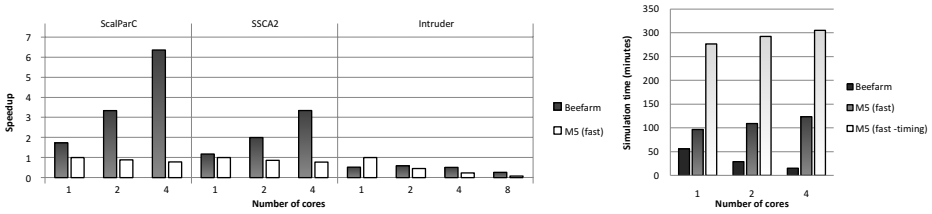

Fig. 3. BeeFarm vs M5. Left: Speedups for ScalParC, SSCA2 and Intruder (normalized to 1 core M5 run). Right: ScalParC simulation time.

4 The Experience and Trade-offs in Hardware Emulation

Although we achieved good scalability for our simulations with respect to the number of processor cores, we have observed several challenges that still face the architecture researcher that adopts FPGA-based emulation.

Place and route times can be prohibitively long, although newer synthesis tool versions have started to make use of the host multithreading capabilities. In the case of adding a simple counter to the design for observing the occurrence of some event, the resynthesis, mapping, placing and routing of an 8-core BeeFarm takes 2-3 hours on our 8-core server.

Another issue is the low level of observability of online debugging tools like ChipScope plus the resource overhead. This problem could be mitigated by an application specific debug framework that is tailored to capturing information about multiprocessor systems. Furthermore, in the example of working with a DDR controller that resides outside of the FPGA, a precise simulation model of the controller should be developed. While in its absence, a fully-working simulation of a design can fail to run when loaded onto the actual FPGA. For such cases, certain 'extra' versions (eg. one that substitutes the DDR with an on-chip memory of BRAMs to separate away all possible problems interfacing the DDR controller) prove to be extremely useful. We also observe an impedance mismatch between the speed of the off-chip DDR memory which runs much faster than the internal processing elements. This mismatch could be exploited by using the fast external memory to model multiple independent smaller memories which would better support architecture research where each core has its own local memory. Alternatively, one controller on each FPGA can be dedicated to model secondary-level caches.

Although the Virtex-5 FPGAs that we use in this work do not allow for implementing a greater number of cores or large multi-level caches, new FPGAs

that support 6-LUTs on 28 nm technology double the total number of gates available on chip while allowing the designer to have access to up to six MB of BRAM capacity and thousands of DSP units. Such abundance of resources might be more suitable for building larger and faster prototypes on reconfigurable infrastructures.

Finally, other researchers have advocated the use of higher level HDLs to improve programmer productivity. We undertook our design in VHDL/Verilog and based on our experience, we would also consider moving to a higher level representation because the design effort and debug difficulty of working with a lower level language when producing a system for emulation rather than production is probably not worthwhile compared to a system that offers rapid design space exploration, e.g. Bluespec.

5 Related Work

Some of the recent multicore prototyping proposals such as RAMP Blue [14] implement full ISA on RTL and require access to large FPGA infrastructures, while others such as the Protoflex [7] can use single-FPGA boards with SMT-like execution engines for simulator acceleration. Although multithreading leads to a better utilization on FPGAs, our focus is architectural emulation where it is generally desirable to keep as close to the original architecture that we are emulating. There is a wide variation of ISAs such as larger SMP soft cores [8], hard cores [26] and smaller ones like the Beehive [22], for prototyping shared memory as well as message-passing schemes [3]. Our work differs from these approaches in the sense that we model the cores only on reconfigurable logic and we effectively upgrade a full ISA open source soft processor core to better fit the architecture of modern FPGAs and to be able to closely examine STM applications and implementations. The only previous study on adapting an available soft core onto a commercial FPGA platform has been the LEON-3 core on the BEE2 [27]. Similar work that involves building a cache coherent MP with MIPS cores was presented in [24]. As for comparison of hardware emulation with software simulators, RAMP Gold [20] compares a SPARC V8 ISA design with three different Simics configurations: Functional, cache, and timing, reporting up to 250x speedup for 64 cores while the ATLAS design compares 8 on-chip 100 MHz PowerPC hard cores with HTM support on five TM applications with a execution-driven simulator at 2 GHz to show 40-200x speedup, also concluding that higher level caches and high associativity were problematic to implement on current FPGAs[26].

Transactional Memory has already drawn a lot of attention in the research community as a new easy-to-use programming paradigm for shared-memory multicores. However, so far mostly preliminary work has been published in the context of studying TM on FPGA-based multicore prototypes. The ATLAS emulator has read/write set buffers and caches augmented with transactional read-write bits and TCC-type TM support, and a ninth core for running Linux [26, 18]. A TM implementation that targets embedded systems which can work without caches, using a central transactional controller interconnecting four Microblaze cores was explained in [12].

Recent work that also utilizes MIPS soft cores focuses on the design of a conflict detection mechanism that uses Bloom filters for a 2-core FPGA-based HTM, however they do not consider/detail any design aspects on their infrastructure. They derive application-specific signatures that are used for detecting conflicts in a single pipeline stage. The design takes little area, reducing false conflicts, and this approach is a good match for FPGAs because of the underlying bit-level parallelism used for signatures[15].

6 Conclusions and Future Work

In this work, we have described a different roadmap in building a full multicore emulator: By heavily modifying and extending a readily available soft processor core. We have justified our design decisions in that the processor core must be small enough to fit many on a single FPGA while using the on-chip resources appropriately, flexible enough to easily accept changes in the ISA, and mature enough to run system libraries and a well-known STM library. We've presented an 8-core prototype on a modern FPGA prototyping platform and compared performance and scalability to software simulators for three benchmarks written to explore tradeoffs in Transactional Memory.

The BeeFarm architecture shows very encouraging scalability results which helps to support the hypothesis that an FPGA-based emulator would have a simulation speed that scaled better with more modelled processor cores than a software-based instruction set simulator. For small numbers of cores we find that software simulators are still competitive, however the gap widens dramatically as more cores are used as ScalParC runs suggest, where for 4 cores the BeeFarm system outperforms the M5 simulator in fast mode by 8x.

Our experience showed us that place and route times, timing problems and debugging cores are problematic issues working with FPGAs. We have also identified components of a typical multiprocessor emulator that map well on FPGAs, such as processing elements, and others that map poorly and consume a lot of resources, such as a cache coherency protocol or a large system bus. We are working on a ring bus that can substantially reduce routing congestion to fit more cores on an FPGA and a memory directory design to extend our infrastructure to use all four FPGAs on the BEE3.

Acknowledgements

We would like to thank Roberto Hexsel, Miquel Pericàs, Gokcen Kestor, Vasileios Karakostas, Steve Rhoads and all anonymous reviewers for their comments and valuable feedback. This work is supported by the cooperation agreement between the Barcelona Supercomputing Center and Microsoft Research, by the Ministry of Science and Technology of Spain and the European Union (FEDER funds) under contract TIN2007-60625, by the European Network of Excellence on High-Performance Embedded Architecture and Compilation (HiPEAC) and by the European Commission FP7 project VELOX (216852). The BeeFarm is available at `http://www.velox-project.eu/releases`.

References

[1] OpenCores Website. `www.opencores.org`.

[2] Plasma soft core. `http://opencores.org/project,plasma`.

[3] H. Angepat, D. Sunwoo, and D. Chiou. RAMP-White: An FPGA-Based Coherent Shared Memory Parallel Computer Emulator. In *Austin CAS, March*, 2007.

[4] N. L. Binkert et al. The M5 simulator: Modeling networked systems. *MICRO06*.

[5] J.-L. Brelet. XAPP201: Multiple CAM Designs in Virtex Family Devices. `www.xilinx.com/support/documentation/application_notes/xapp201.pdf`, 1999.

[6] N. L. V. Calazans et al. Accelerating sorting with reconfigurable hardware. `www.inf.pucrs.br/~gaph/Projects/Quicksort/Quicksort.html`.

[7] E. S. Chung et al. A complexity-effective architecture for accelerating full-system multiprocessor simulations using FPGAs. In *FPGA '08*, pages 77–86, 2008.

[8] N. Dave, M. Pellauer, and J. Emer. Implementing a functional/timing partitioned microprocessor simulator with an FPGA. *WARFP*, 2006.

[9] J. Davis, C. Thacker, and C. Chang. BEE3: Revitalizing computer architecture research. *Microsoft Research*, 2009.

[10] P. Felber, C. Fetzer, and T. Riegel. Dynamic performance tuning of word-based software transactional memory. In *PPoPP*, pages 237–246, 2008.

[11] L. Hammond et al. Programming with transactional coherence and consistency (TCC). In *ASPLOS-XI*, pages 1–13, 2004.

[12] C. Kachris and C. Kulkarni. Configurable transactional memory. In *FCCM '07*, pages 65–72, 2007.

[13] G. Kestor, S. Stipic, O. Unsal, A. Cristal, and M. Valero. RMS-TM:a tm benchmark for recognition, mining and synthesis applications. In *TRANSACT 2009*.

[14] A. Krasnov et al. Ramp Blue: A message-passing manycore system in FPGAs. In *FPL 2007*, pages 27–29, 2007.

[15] M. Labrecque, M. Jeffrey, and J. Steffan. Application-specific signatures for transactional memory in soft processors. In *ARC 2010*, pages 42–54. 2010.

[16] C. C. Minh, J. W. Chung, C. Kozyrakis, and K. Olukotun. STAMP: Stanford transactional applications for multi-processing. In *IISWC*, pages 35–46, 2008.

[17] K. E. Moore, J. Bobba, M. J. Moravan, M. D. Hill, and D. A. Wood. LogTM: Log-based transactional memory. In *HPCA 2006*, pages 254–265, 2006.

[18] N. Njoroge, J. Casper, S. Wee, Y. Teslyar, D. Ge, C. Kozyrakis, and K. Olukotun. ATLAS: A chip-multiprocessor with TM support. In *DATE'07*, pages 3–8.

[19] D. A. Penry et al. Exploiting parallelism and structure to accelerate the simulation of chip multi-processors. In *HPCA*, pages 29–40, 2006.

[20] Z. Tan et al. RAMP gold: an FPGA-based architecture simulator for multiprocessors. In *DAC '10*, pages 463 – 468, 2010.

[21] C. Thacker. A DDR2 controller for BEE3. Microsoft Research, 2009.

[22] C. Thacker. Beehive: A many-core computer for FPGAs (v5). In *http://projects.csail.mit.edu/ beehive/BeehiveV5.pdf*. MSR Silicon Valley, 2010.

[23] S. Tomic, C. Perfumo, A. Armejach, A. Cristal, O. Unsal, T. Harris, and M. Valero. EazyHTM: Eager-lazy hardware transactional memory. In *MICRO 42*, 2009.

[24] J. Tortato Jr and R. Hexsel. A minimalist cache coherent mpsoc designed for fpgas. In *Int. J. High Performance Systems Architecture*, 2011.

[25] M. Vijayaraghavan and Arvind. Bounded dataflow networks and latency-insensitive circuits. In *MEMOCODE*, pages 171–180, 2009.

[26] S. Wee, J. Casper, N. Njoroge, Y. Tesylar, D. Ge, C. Kozyrakis, and K. Olukotun. A practical FPGA-based framework for novel CMP research. In *FPGA '07*, 2007.

[27] T. Wong. LEON3 port for BEE2 and ASIC implementation. `http://cadlab.cs.ucla.edu/software_release/bee2leon3port/`.