Cache Tracker: A Key Component for Flexible Many-Core Simulation on FPGAs

Jonathan Woodruff, Greg Chadwick, Simon Moore

Cache Tracker: A Key Component for Flexible Many-Core Simulation on FPGAs

Jonathan Woodruff, Greg Chadwick and Simon Moore
Computer Laboratory University of Cambridge

***Abstract*—This paper presents a cache tracker, a hardware component to track the cache state of hundreds of caches serving processors modeled using threads on a single MIPS64 processor. This host-multithreading approach allows a single, low-cost FPGA to model large systems to allow quick and broad architectural exploration with reasonable simulation performance. The cache tracker stores all state in DRAM to allow maximum scalability in both number of processors and in cache sizes. We describe our approach of scalability versus simulation performance, our implementation in Bluespec SystemVerilog, and give a sample study of a parallel merge-sort over various processor numbers, cache sizes and arrangements.**

Introduction: FPGA Simulation and Our Approach

Our system follows others to adopt an FPGA simulation approach to multi-processor architecture exploration but differs from past efforts by stressing simplicity and scalability over simulation performance while still maintaining orders of magnitude performance advantage over software simulation. Academic computing efforts began with fabricating actual implementations but turned to software simulation as transistor densities and logic complexity soared. Recently researchers have turned back to hardware models to explore large-scale computing systems using FPGAs. FPGA modeling is cheap and fast but limited in flexibility as the entire system must often span multiple chips. A new approach, FPGA simulation, strikes a balance between the flexibility of software simulation and the performance of full-system FPGA modeling [1], [3], [5]. Two notable current efforts in this vein are RAMP Gold from Berkeley [1] and Protoflex from Carnegie Mellon [2]. RAMP Gold uses a very similar approach to ours but is so far quite limited in scalability due to their sharp focus on performance. The RAMP Gold architecture supports 64 processors per core and has limited support for simulating cache sizes due to their use of on-chip FPGA memory to store cache tags. For larger systems these tags must be stored in DRAM, as the paper mentions, but as these tags must be accessed every cycle, performance would suffer and the option was not pursued in their report. We intend to simulate thousands of processors on a single FPGA. We use the performance advantage of hardware simulation to produce a faster system than is possible in software simulation and a more scalable system than is possible using direct FPGA modeling. For this purpose, a performance loss within an order of magnitude would not reduce our field of exploration while a fixed limit to scalability certainly would. In this spirit, all development has been in Bluespec SystemVerilog, a high-productivity hardware description language, and all designs have been focused on maintainable and extensible functionality and are generally free from obscure optimizations.

The Cache Tracker Architecture

Our system is composed of two components. The first is a thread-flexible user-level in-order MIPS64 core written in Bluespec. The second is the cache tracker which tracks a cache state for each thread in DRAM. The simulated system behaves as if all caches are on a shared bus with direct access to the transactions of all other processors. The shared bus model represents the best case for multi-processor shared memory access and should provide a valuable reference point. Simulating different interconnects and cache coherency models is future work.

Data Structures

The cache tracker maintains two data structures in DRAM: the cache tags and the directory. The first data structure contains the cache tags for each cache. Each set of lines in the modeled caches is allocated 64 bits. Each set may contain 4 lines. Fourteen bits are allocated for each address with 2 bits to record the age of each entry (see figure 1). Since 14 bits are allocated per record, each record may only disambiguate 16,384 lines. This limitation implies that if processor memory increases, cache sizes must maintain a minimum capacity to ease the disambiguation burden of each line. We hope to alleviate this requirement in the future but this approach allows a simplified design at this stage.

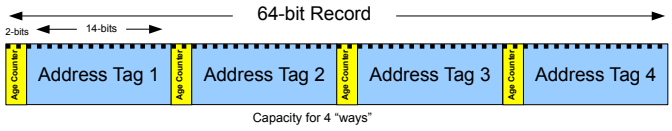

Figure 1. Cache Line Record

The second data structure is a directory of memory lines accessible to the processor. There is one entry in the directory for each line in memory. Each entry is a bit-vector with one bit for each cache in the system. Each bit represents the presence of that line in that cache in the system. This is a simple but inefficient approach as a cache line size equal to the number of processors in the system, i.e. 64 byte lines and 512 processors, will result in this directory occupying the same space as system memory. As DRAM is quite inexpensive, this solution is acceptable for this simple system.

Cache Tracker State Machine

The cache tracker requires a single DRAM access for each host memory access that hits the cache but will require 5 DRAM accesses for each cache miss. When the processor reports a memory transaction to the cache tracker, the cache tracker performs the following steps:

1) Recall directory bit-vector
2) If the bit for this thread is already set, do nothing more. If the bit for this thread is not selected, set it, and recall the set of tags for this thread and this line. If this is a write, clear all other bits in the vector to maintain coherence.
3) Check the age tags of the lines in this set, note and replace the last loaded line and update the age tags for all other tags.
4) Recall directory bit-vector for the replaced line.
5) Clear the bit for the replaced line and store the new bit-vector.

Parameters

There are four parameters: cache number, cache size, line size and associativity. The cache number is currently equal to the number of processors being modeled as caches and processors are only modeled in a one-to-one arrangement. Associativity is restricted to 1, 2 or 4 way for simplicity as these are sufficient to demonstrate any strong trends an algorithm may exhibit toward associativity [4].

Experimental Setup

Our system runs on an Altera DE2-70 board, available academically for $269 and commercially for $495. The Cyclone II FPGA board contains 484 Kbits of BRAM which is used almost entirely for the register-file of the MIPS64 processor at 128 contexts. We have recently developed a user-level ARM core which runs 512 contexts on the same board as each ARM register file is $1/4$ the size of the MIPS64.

A minimal Altera NIOS II processor with a debug module is also synthesized into the design to allow DRAM updates from a desktop computer and to allow direct inspection of DRAM contents. This arrangement allows our multi-threading processor to be as simple as possible while still allowing full debug accessibility to DRAM.

Sample Study: Choosing an Architecture for Parallel Merge Sort

In this section we will use our cache tracker system to select a many-core architecture to perform a high-performance merge-sort. We ran the algorithm across a matrix of cache sizes from 1 to 64 kilobytes per core, line sizes from 64 bytes to 1024 bytes, associativity from 1 way to 4 way and processor number from 8 to 64. For a high level view we will analyze performance versus area and performance versus power. From an architectural exploration perspective, we will analyze the effects of cache sharing in two configurations on the parallel merge-sort algorithm.

Performance vs. Area Estimate

In figure 2 we sorted configurations by estimated area required and colored them by processor number and cache size. We assume that a processor core and 16 kilobytes of cache both occupy $0.5mm^2$. We will estimate performance simply by assuming that every cache miss results in a 100 cycle delay in execution and ignore the complex interaction of delays among the processors. At this point we will assume independent caches and a line size of 64 bytes and in the next sections we will analyze the effect of inter-cache sharing and various line sizes. Finally we have fixed a performance requirement of 50 sorts per second at 100 MHz for the system.

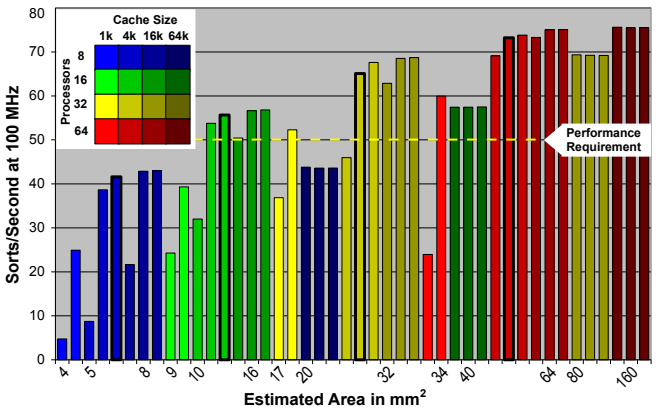

Figure 2. Area vs. Performance

The large variations within a group of the same processor number and cache size in figure 2 are due to differences in associativity. It is apparent that associativity strongly affects performance when total cache sizes are very small. We have highlighted four nodes that we will follow through the remaining tests.

Performance vs. Power Estimate

In figure 3 we have sorted configurations by estimated power consumption. We estimate that the active logic of one processor core and the SRAM of 64 kilobytes of cache both consume 50mW at 100 MHz, the active logic consuming four times the power of the SRAM per unit area.

The order of figure 3 is similar to figure 2 and demonstrates that the cache-heavy designs are only slightly more power efficient than the compute heavy designs for this algorithm. Figure 7 at the end of this section gives the numbers for the four nodes of interest.

Performance and Bandwidth Requirement with Cache Sharing and Exclusive Caching

Finally we will examine performance and system bandwidth requirements with cache sharing. The previous data assumed independent caches which were supplied exclusively from external memory. Here we will also explore the benefit of

cache sharing, that is, allowing other caches to supply memory requests directly.

Figure 4 displays the external bandwidth requirements for each configuration and includes line size as well as cache size and processor number. The nodes of interest from the previous section are also noted. Cache sharing reduces external bandwidth requirements by 50.6%. There is also a strong correlation between line size and bandwidth requirements. While performance is not shown for the larger line sizes, it is within a few percent of the smaller ones.

Figure 5 displays the internal bus bandwidth requirements for the two configurations. As expected the sharing configuration requires more on-chip bandwidth, 55% on average, than the non-sharing case to deliver data directly between caches.

Figure 6 gives the performance for the nodes examined in the previous section with sharing. Performance is better in all cases and sharing allows even the best 8 processor option to meet the performance requirement. Figure 7 gives the precise numbers for our four nodes of interest.

DISCUSSION

Applications of an FPGA Host-multithreaded Simulation Engine

Algorithm Characterization – Software development teams do not have access to many-core processors on which their algorithms may be executing within a few years. A fast, inexpensive and flexible simulator environment gives deeper insight to software developers and allows them to avoid design decisions that would hurt the performance of their algorithms in a many-core environment.

Design Exploration for Application-specific Chips – Hardware development teams may need such a system for quick and broad architectural exploration. Hardware development teams would have the resources to purchase any simulation equipment required but the flexibility and performance of the host-multithreading simulator will allow characterization of a large number of configurations within the transistor budget of the project, even allowing auto-tuning with reasonable performance due to basic system parameters being controlled by simple register writes.

Architectural Feature Development for Many-core – The simplicity of the host-multithreaded model allows for

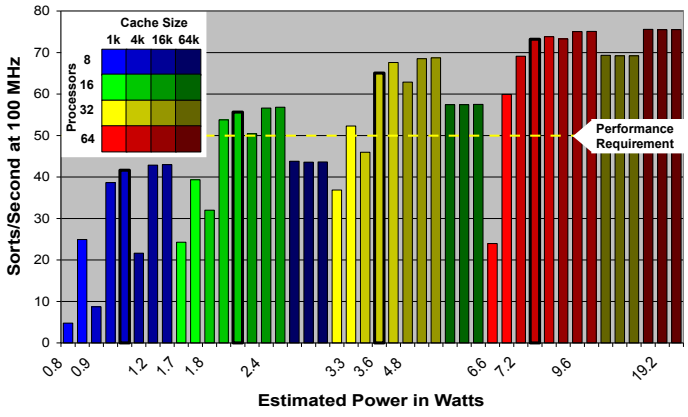

Figure 3. Performance vs. Power

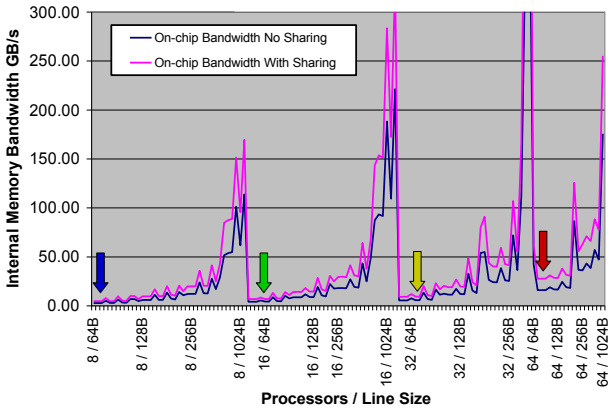

Figure 5. On-chip Communication Bandwidth

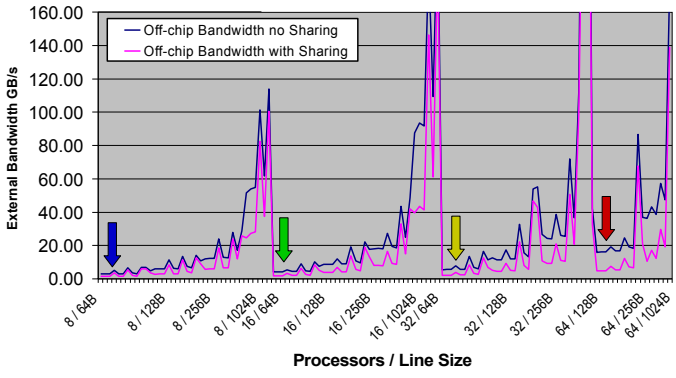

Figure 4. Off-chip Memory Bandwidth

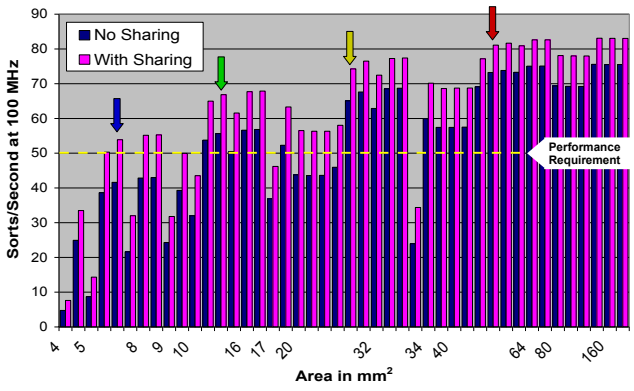

Figure 6. Sharing Performance

Cores	8	16	32	64
Cache Size in KB	4	4	4	4
Line Size in Bytes	64	64	64	128
Associativity	4	4	2	2
Sorts/Second at 100MHz with no Sharing	42	56	65	73
Sorts/Second at 100MHz with Sharing	54	67	74	81
Area in mm^2	5	10	20	40
Power in Watts	0.9	1.8	3.6	7.2
Cache Hit Rate with no Sharing	99.2	99.5	99.6	99.8
Cache Hit Rate with Sharing	99.7	99.8	99.8	99.9
Off-chip Bandwidth with no Sharing in GB/s	3.15	4.60	7.00	19.14
Off-chip Bandwidth with Sharing in GB/s	1.64	2.23	3.40	7.28
Internal Bandwidth with no Sharing in GB/s	3.15	4.60	7.00	19.14
Internal Bandwidth with Sharing in GB/s	5.03	7.34	11.01	31.71

Figure 7. Nodes of Interest

quick implementation and characterization of new architectural features for many-core processors. The performance advantage over software simulators allows researchers to gather more convincing data-sets and the complexity advantage over multi-FPGA models allows researchers to construct models more quickly.

Limitations of the Host-multithreading Approach

There are limitations to the host-multithreading approach. Firstly, simulation performance decreases linearly with the number of processors being simulated. While performance will be better than software simulation, scaling will be the same. An adjustment to the model to allow multiple multi-threaded processors operating concurrently should alleviate this problem and should not introduce the difficulties that have been encountered in parallelizing the highly complex software simulators. Secondly, out-of-order processors will be difficult to model with the host-multithreaded simulation approach. While a multi-threaded pipeline may efficiently and accurately simulate multiple in-order pipelines by simply multiplexing instructions, simulation of processors with more complex state may be intractable with this approach. However systems with hundreds to thousands of processors that are modelable with this method are likely to use in-order cores to meet density and power requirements.

Future Directions

Performance Improvement

The current system allows only a single context through the pipeline at a time due to stability problems. An instruction is issued every 10 cycles as each instruction requires 5 cycles to fetch and 5 cycles to proceed through the pipeline. The next instruction will only be issued if the previous transaction is being processed by the cache tracker. If the transaction hits the cache in the cache tracker the next instruction will not be delayed but a miss may delay the next instruction. As cache misses are rare the pipeline averages close to 10 cycles per instruction.

Feedback into Processor

Our first extension to the cache tracker system will be memory system performance feedback into the processor. The cache tracker will introduce delays to the individual threads running on the processor to simulate memory delays in the modeled system. This capability will allow the host-multithreaded system to model the complex interaction of delays in the various processors.

External Memory Banks

Actual systems cannot deliver multiple accesses to the same memory interface within the same cycle but often provide multiple memory banks to alleviate this contention. We will implement a simple memory bank model with interleaved lines such that only one access will be allowed per bank per cycle and the accesses for the other threads will be delayed thus more accurately modeling the shared bus model.

Multi-threaded Router for Network-on-Chip

A shared bus is not usually practical for a many-core system. Cores are normally connected to an on-chip network fabric of routers and transmission lines. The host-multithreading approach will also be useful for modeling this network. A single router will calculates the next state for each modeled router consecutively, storing state in DRAM. As with the cache tracker, this router only needs to manipulate routing tags and not actual data, thus easing pressure on the DRAM interface.

Conclusion

Our system allows software developers to quickly study the performance of their software across a broad range of scaling parameters. The low hardware cost of this system, about $500, makes it very accessible. For 1000 cores, the performance will be about 10,000 times slower than real-time due to context serialization and the frequency limitations of the FPGA but still significantly faster than software simulation as software simulation will require more than thirty clock cycles (the frequency difference between this core and high-performance processors) to update the state of each core and to track cache state. We also believe the emphasis on simplicity and flexibility in our system will make it useful as a basis for other projects in the many-core development community.

References

[1] R. Avizienis, Y. Lee, and A. Waterman. RAMP Gold: A High-Throughput FPGA-Based Manycore Simulator.

[2] E.S. Chung, E. Nurvitadhi, J.C. Hoe, B. Falsafi, and K. Mai. PROTOFLEX: FPGA-accelerated Hybrid Functional Simulation. 2007.

[3] E.S. Chung, E. Nurvitadhi, J.C. Hoe, B. Falsafi, and K. Mai. A complexity-effective architecture for accelerating full-system multiprocessor simulations using FPGAs. In *Proceedings of the 16th international ACM/SIGDA symposium on Field programmable gate arrays*, pages 77–86. ACM, 2008.

[4] MD Hill and AJ Smith. Evaluating associativity in CPU caches. *IEEE Transactions on Computers*, 38(12):1612–1630, 1989.

[5] A. Nanda, K.K. Mak, K. Sugarvanam, R.K. Sahoo, V. Soundararajan, and T. Smith. MemorIES3: a programmable, real-time hardware emulation tool for multiprocessor server design. In *Proceedings of the ninth international conference on Architectural support for programming languages and operating systems*, pages 37–48. ACM, 2000.