Studying Data Forwarding on a Non-Coherent Tiled Multicore

Asbjørn Djupdal, Magnus Jahre, and Magnus Själander
Norwegian University of Science and Technology, NTNU
firstname.lastname@idi.ntnu.no

ABSTRACT

Evaluation of experimental multicore architectures using a software simulator can be very time-consuming. This paper presents the Single-isa Heterogeneous MAny-core Computer (SHMAC) platform for doing design space exploration and performance evaluation of multicores. SHMAC makes it easy to custom build a tiled multicore architecture using simple RISC-V CPU cores in a mesh interconnect. This can then be synthesized for an FPGA where programs can be evaluated much faster than with a simulator.

Interconnect utilization is an important consideration for multicores. Congestion at busy hot spots leads to increased latency and reduced performance. Unnecessary traffic represents wasted energy. This paper investigates several different techniques to reduce interconnect traffic and congestion in a tiled multicore, using SHMAC as the experimental platform.

1. INTRODUCTION

Modern hardware description languages (HDLs), such as Chisel, increases productivity and the largest FPGAs can fit an ever larger number of simple CPU cores for each new generation. These combined effects makes FPGA prototyping of multicores more accessible. Large multicores simulate very slowly in software, while they run at acceptable speeds in an FPGA. We have, therefore, developed SHMAC, a framework for doing hardware prototyping, design space exploration, and performance analysis on tiled multicores

This paper presents the SHMAC framework and uses it to investigate various solutions to improve interconnect utilization in a tiled multicore. We first present the SHMAC framework (Sec. 2) in detail: CPU core (Sec. 2.1), interconnect (Sec. 2.2), and memory (Sec. 2.3). We then present the interconnect utilization study (Sec. 3) where we describe the studied techniques (Sec. 3.1) and the various configurations under test (Sec. 3.2). We finish the paper with discussing the interconnect results (Sec. 3.5) and conclusions (Sec. 4).

2. SHMAC: A CONFIGURABLE TILED MULTI CORE

SHMAC is a tile based architecture resembling the Tilera and Epiphany multicores [1, 2]. A top-level view of SHMAC is shown in Fig. 1. The top-level architecture is a 2D rectangular mesh, with each tile communicating with each of the four neighbors. Most of the tiles contain a CPU core (marked "CPU" in the figure), the other tiles are special purpose providing other capabilities to the SHMAC system, such as memory or I/O.

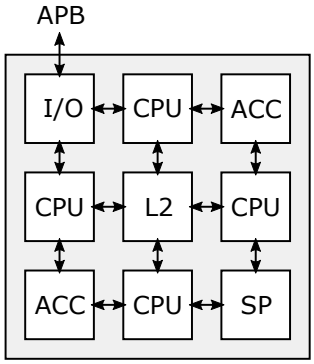

Figure 1: SHMAC top-level architecture, showing an example 3x3 configuration with four CPU cores, two accelerators, one L2 cache tile, one scratchpad memory tile, and one I/O tile for external communication over AMBA APB.

SHMAC is highly configurable, properties such as mesh dimensions and the type and capabilities of the different tiles can be chosen prior to synthesis.

2.1 Processor Core

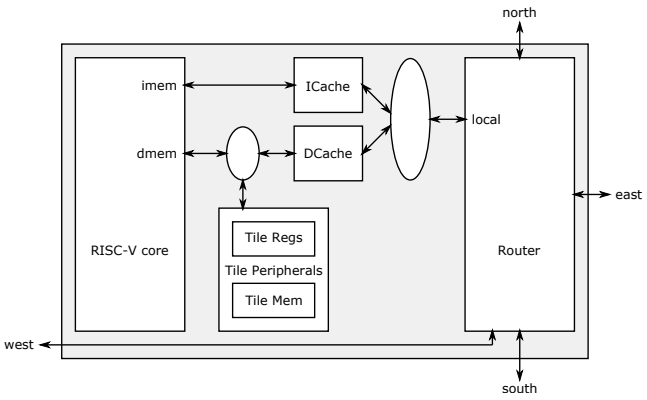

Figure 2: SHMAC CPU tile architecture

The CPU tile is shown in Fig. 2. The CPU tile, like any other SHMAC tile, contains a router and a local core, which in this case is a RISC-V CPU core.

The SHMAC CPU core is a small 32-bit integer-only RISC-V [3] compatible core built on the Sodor Z-scale [4]. The core has the same general architecture as the original Z-scale core, but with the following modifications:

- Added instruction and data L1 caches and modified pipeline to support variable memory access latency
- Added support for integer multiplication and division
- Added support for external interrupts
- Added support for atomic instructions

As shown in the figure, the tile also contains tile-local registers and scratchpad memory. The registers are used to provide the local CPU with information such as mesh location and CPU ID. This version of SHMAC does not support virtual memory, so to avoid concurrency conflicts when using libc functions, a local scratchpad is provided for newlibs _reent data.

2.2 Interconnect

The SHMAC interconnect is 2D and mesh based, implementing XY-routing. Every tile contains a router that is connected to the local core and to the routers in the four neighboring tiles. The router implements store-and-forward switching in a typical multi-stage architecture [5]. To reduce latency through the router, there are only two stages: (1) Route computation and (2) switch traversal. In addition, the last stage can forward data directly to its output port if the port is ready, resulting in a minimum latency of only one cycle.

Each router link has two virtual channels; one is used for request messages and one for response messages. This ensures deadlock free routing in all SHMAC configurations.

2.3 Memory

The SHMAC memory hierarchy has three levels: L1 instruction and data caches located on the CPU tiles, L2 caches located on dedicated L2 tiles, and DDR memory that exists outside of the SHMAC toplevel but reachable by the L2 tiles. All caches are set-associative, and the L2 cache is fully out-of-order to support requests from several cores at once.

There is currently no support for hardware cache coherence in the SHMAC system. Proper access to shared data must be coordinated in software. Likewise, there is no support for virtual memory. All addresses in the system are physical addresses.

2.4 Performance Analysis and Debugging

As the SHMAC platform is intended for research, an extensive set of performance counters are included, all accessible by software running on the SHMAC CPU tiles. These counters provide runtime information about the CPUs and both levels of cache. CPU pipeline performance counters are implemented as RISC-V CSR registers. Cache performance counters are implemented as memory mapped registers, handled by the caches themselves. In addition, there is a separate debug interface that enables control of the cores using GDB running on the external host computer.

3. IMPROVING INTERCONNECT UTILIZATION

The rest of this paper is dedicated to investigations into interconnect utilization in tiled multicores, where SHMAC is used as the experimental platform.

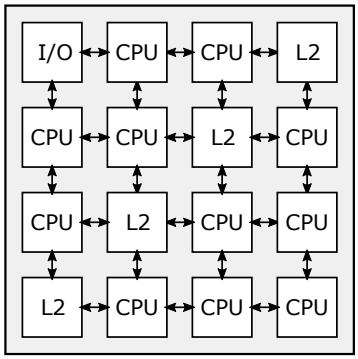

Figure 3: SHMAC configuration with four distributed L2 tiles

3.1 Investigated Techniques

3.1.1 Distributed L2 Cache

A centralized L2 cache, as shown in Fig. 1, is simple and area efficient, but has the large disadvantage of quickly becoming a bottleneck as the number of L1 caches in the system increases. A typical L2 cache can only accept at most one access per cycle, much less if the L2 hit rate is low.

One solution is to distribute the L2 cache memory across different tiles, each having its own instance of the L2 control logic [6]. One example of this is shown in Fig. 3 where there are four L2 cache tiles, each handling approximately $\frac{1}{4}$ of the total number of L2 accesses. The L2 tiles are all connected to the single memory controller in a round-robin arbitrated star network. The placement of the L2 tiles is important. To spread the traffic as evenly as possible across the mesh, no two L2 tile should share the same X or Y coordinate. Placement along the diagonal, as shown in the figure, is a good solution.

3.1.2 Forwarding Invalidation for Synchronization

One common way of implementing synchronization between cores is to have a synchronization variable that is written by the signalling core and read by the waiting core. The waiting core spins over this variable and does not advance until the variable has been changed by the other core. This is simple and works well, but has implications for traffic on the interconnect.

In a system with a cache coherence protocol, the waiting core can spin locally on its L1 cache until an invalidation message appears from the L2 cache. A full cache coherence protocol implemented in hardware in a chip multi processor (CMP) might not always be the preferred solution. In a system without cache coherence, the spinning must effectively be done on the L2 cache, resulting in increased L2 traffic. Exponential backoff reduces the traffic significantly, but does not eliminate it.

For this paper we have added a forwarding instruction [7] to the ISA that can be used to send invalidation messages directly to the L1 cache of a different core. These messages are routed along the shortest path in the mesh and are not required to pass through the L2 cache. By using such forwarding messages, the waiting core can spin on its local L1 cache until the signaling core executes its forwarding instruction, thus reducing both interconnect traffic and L2 utilization.

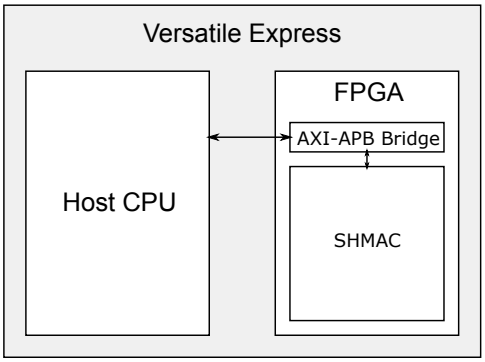

Figure 4: Host and SHMAC interaction

3.2 Evaluated Configurations

In order to compare the techniques mentioned in Sec. 3.1, several combinations of hardware and software configurations must be evaluated. The following is a list of the combinations chosen for the experiments in this paper:

BO0: Pthread synchronization variable uses self-invalidated spinning, no exponential backoff. Centralized L2 cache.

BO14: Pthread synchronization variable uses self-invalidated spinning, with exponential backoff MAX_DELAY of $1 << 14$ cycles. Centralized L2 cache.

FC: Pthread synchronization variable uses forward invalidated spinning. In addition, dedup queue synchronization is also forward invalidated. Centralized L2 cache.

FWD (dedup only): Pthread synchronization variable uses forward invalidated spinning and forwarding write is used for data exchange between software pipeline stages. Centralized L2 cache.

BO14 + 4xL2: Pthread synchronization variable uses self-invalidated spinning, with exponential backoff MAX_DELAY of $1 << 14$ cycles. Distributed L2 cache with four L2 tiles.

3.3 FPGA Synthesis and Core Execution

The SHMAC multicore is synthesized for a Xilinx Virtex7 2000T FPGA in an ARM Versatile Express development system. Fig. 4 shows the host computer and its connection to the the FPGA containing SHMAC. SHMAC acts as an AMBA APB slave and its registers are directly accessible through loads and stores on the host ARM processor. The host runs Linux and custom software on this host is used for controlling and monitoring the SHMAC multicore.

3.4 Benchmarks and Software Runtime Environment

The hardware and software configurations in this paper are evaluated on Dedup, Canneal and StreamCluster, which are selected from the PARSEC benchmark suite [8]. Some porting was needed, both due to a lack of cache coherency, but also to incorporate the techniques under investigation.

All benchmarks run "bare-metal", using only libc, a custom version of pthreads and a minimal OS kernel providing necessary I/O. No thread scheduling is performed on the individual cores, each core is running only one thread. This puts higher demand on benchmark load balancing but makes system software simpler.

3.5 Experimental Results

Fig. 5 shows a performance comparison of the benchmarks mentioned in Sec. 3.4. Each benchmark has been run in all the different configurations mentioned in Sec. 3.2 with a 16 kB L1 instruction and 32 kB data cache. The L2 cache has a constant size of 512kB and is in one central tile, except for in the configuration marked BO14+4L2 where the L2 is distributed across four tiles. Performance here is runtime of the measured section of the benchmarks. Runtime is normalized to the baseline configuration BO0 for each benchmark.

Performance as measured in runtime is important, but depends on a lot of factors of the system under test. The main focus for this paper is interconnect utilization. To illustrate how the various configurations affect total interconnect traffic, Fig. 6 shows the total number of data transfers from the L2 cache to all the L1 data caches. The figure is otherwise organized as Fig. 5.

3.5.1 Centralized vs Distributed L2

The first insight from Fig. 5 is that a distributed L2 cache can provide a significant performance increase. When comparing the distributed variant (BO14+4L2) with the equivalent centralized variant (BO14), the distributed variant performs in general better or as good as the centralized variant. This is expected. By distributing the L2 cache across four different tiles, the L2 traffic is also distributed across the mesh, effectively increasing the bandwidth. In addition, up to four L2 accesses can be handled in parallel. This is well known by the community and the reason for the popularity of distributed L2 caches [1, 6].

3.5.2 Exponential Backoff vs Forward Invalidation

A second insight to be learned from Fig. 5 is regarding choice of synchronization implementation. It is clear that the BO0 configuration, the self-invalidation variant without exponential backoff, is a bad solution. For Dedup the performance is much worse, and for the other benchmarks it is marginally worse than the other variants. BO14 and FC do, however, not seem to differ significantly in performance.

Fig. 6 provides a more interesting view on the situation. In this figure it is immediately obvious that BO0 produces significantly more L2 traffic than any other configuration by a large margin. The number of L2 reads issued by the L1 data caches is dominated by the synchronization variable in the BO0 variant. It is also clear that the other configurations are more equal in their generated L2 traffic.

It is expected that the Canneal benchmark is relatively unaffected by the synchronization implementation; its barrier synchronization is too coarse grained. Likewise, Streamcluster spends too much time doing floating point for the differences between BO14 and FC to be of significance.

The exception is Dedup, where there is a notable difference between all variants. Fig. 7 shows the differences between BO14 and FC in more detail by plotting the combined number of reads per cycle for all L1 data caches in the system. Forwarding invalidation (FC) reduces the L2 read traffic by a modest 4% compared to using exponential backoff (BO14). The figure also shows that L2 congestion happens at around 0.25-0.30 data cache reads per cycle. The reason for not reaching one read per cycle is that instruction caches also generate traffic, and that many concurrent L2 misses (more than the available number of MSHRs) lead to the L2 cache stalling.

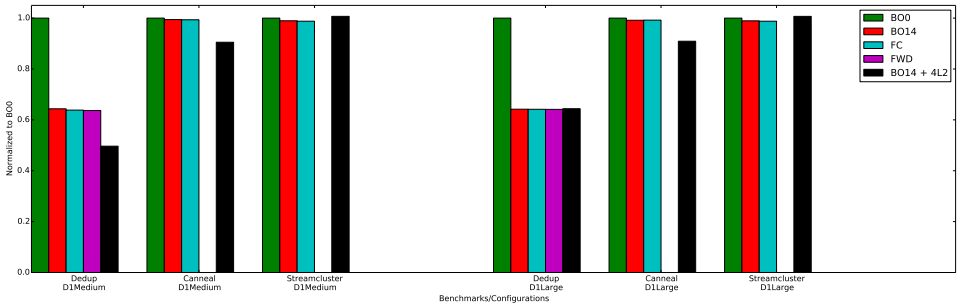

Figure 5: PARSEC performance. Smaller is better

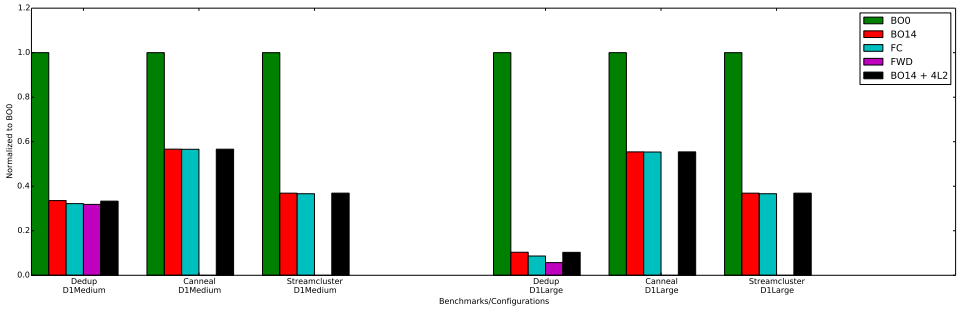

Figure 6: PARSEC L1D–L2 reads. Smaller is better

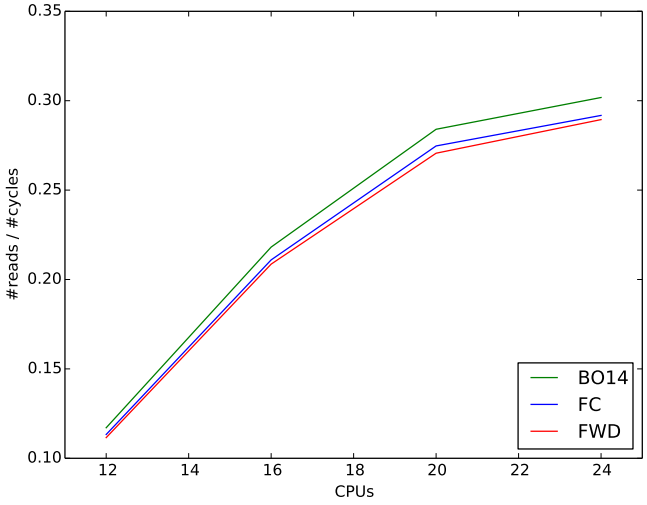

Figure 7: Dedup L1D–L2 reads per cycle.

4. CONCLUSION

We have presented the Single-isa Heterogenous MAny-Core Computer (SHMAC) evaluation framework, which is based on a tiled 2D mesh architecture similar to that of Tilera [9] and Epiphany [2]. The SHMAC framework is highly configurable and can easily be implemented on an FPGA. This makes the SHMAC framework suitable for performing design space explorations of multicore architectures that otherwise requires significant simulation efforts if performed in a software simulator. This is exemplified by the performed interconnect study where we show that data forwarding can reduce the interconnect traffic to the L2 cache by directly sending data to the awaiting L1 data cache.

Acknowledgement

We thank ARM Norway for their generous donation of the ARM Versatile Express development system. We also thank the master students and NTNU researchers that have contributed to the development of the SHMAC infrastructure.

5. REFERENCES

[1] *Tile Processor Architecture Overview for the TILEPro Series*, Tilera Corporation, 2013.

[2] *Epiphany Architecture Reference*, Adapteva, Inc., Mar. 2014, http://adapteva.com/docs/epiphany_arch_ref.pdf.

[3] A. Waterman, Y. Lee, D. Patterson, and K. Asanovic, *The RISC-V Instruction Set Manual*, 2014.

[4] C. Celio and E. Love, "The Sodor processor collection," https://github.com/ucb-bar/riscv-sodor.

[5] W. J. Dally and B. P. Towles, *Principles and Practices of Interconnection Networks*, 1st ed. Morgan Kaufmann, 2004.

[6] L. A. Barroso, K. Gharachorloo, R. McNamara, A. Nowatzyk, S. Qadeer, B. Sano, S. Smith, R. Stets, and B. Verghese, "Piranha: A scalable architecture based on single-chip multiprocessing," in *Proceedings of the International Symposium on Computer Architecture*, 2000, pp. 282–293.

[7] D. Poulsen and P.-c. Yew, "Data prefetching and data forwarding in shared memory multiprocessors," in *Proceedings of the International Conference on Parallel Processing*, vol. 2, aug 1994, pp. 280–280.

[8] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The PARSEC benchmark suite: Characterization and architectural implications," Princeton University, Tech. Rep. TR-811-08, 2008.

[9] *Tile Processor User Architecture Manual*, Tilera Corporation, 2011.