Three Tier Proximity Aware Cache Hierarchy for Multi-core Processors

Akshay Chander, Aravind Narayanan, Madhan R and A.P. Shanti
Department of Computer Science & Engineering,
College of Engineering Guindy, Anna University, Chennai, India

Abstract - **Unlimited amounts of fast memory is a dream for computer programmers. An economical way to achieve that desire is a memory hierarchy which takes advantage of locality and cost–performance ratio of various memory technologies. Multi-core processors currently in use today do not take full advantage of the possible memory hierarchies available. In this paper, we propose a three-tiered cache hierarchy for a chip multiprocessor with at least 32 or 64 cores, with a high-bandwidth shared Level 3 cache for small groups of, say 8, cores, and private Level 2 and Level 1 caches for each core, to improve cache performance in multi-core processors. We make use of a proximity-aware directory-based coherence algorithm to improve the performance of the shared Level 3 caches. This architecture will benefit tasks where a shared cache is desirable, such as inter-core communication and performing operations on shared data structures. This method maximizes the cache hit rate thereby improving the overall performance of the memory hierarchy and the multi-core processor.**

Key Words – **Multi-core, CMP, Cache Coherence, Proximity aware, directory**

1. Introduction

While manufacturing technology continues to improve, reducing the size of single gates, physical limits of semiconductor-based microelectronics have become a major design concern. Some effects of these physical limitations can cause significant heat dissipation and data synchronization problems.

The demand for more capable microprocessors drives CPU designers to explore various methods of increasing performance. Some instruction-level parallelism (ILP) methods like superscalar pipelining are suitable for many applications, but are inefficient for others that tend to contain difficult-to-predict code. Many applications are better suited to thread level parallelism (TLP) methods, and utilizing multiple independent CPUs is one common method used to increase a system's overall TLP. A combination of increased available space due to refined manufacturing processes and the demand for increased TLP is the logic behind the creation of multi-core CPUs. Thus, multi-core processors (also known as chip multiprocessors) are increasingly being used to enhance the throughput and power efficiency of chips.

Initial implementations of chip multiprocessors made use of two or four cores. However, the number of cores per chip is increasing rapidly. For example, IBM manufactures the eight-core Cell Processor [1], and Tilera has recently introduced the TILE64 [2], a processor featuring 64 cores on a single chip. Intel's Core Duo and Core 2 Duo [3] platforms use a shared L2 cache and a private L1 cache for each core. However, IBM's Cell Processor and Tilera's Tile64 feature two private levels of cache per core, with no shared level at all. As the number of cores increase, a shared cache becomes less viable, as the size and cost required to implement such a shared cache for all the cores becomes prohibitive, traffic congestion on the chip and the speed gains lessen.

Hence, we propose to implement a shared Level 3 cache for small groups of cores. This does not lead to any of the drawbacks of implementing a shared cache for a large number of cores, while still maintaining the positive effects of a shared cache, such as application pipelining and cache under-utilization, among others. Thus, in a 64 core chip multiprocessor, each core would have a private Level 1 and Level 2 cache. Moreover, each group of, say, 8 cores will have a shared Level 3 cache to themselves. Thus, we implement 8 shared Level 3 caches in all for the 64 core processor. We use a modified proximity aware directory based cache coherence algorithm detailed in

Brown et al. [4]. Thus, the presence of a shared Level 3 cache among a group of 8 cores will facilitate quick inter-core communication, as the cores can use the shared Level 3 cache as a communication mechanism to pass messages. It will enable the group to act in tandem on particular data, thus enabling collaborative processing. For example, some cores can pre/post-process the data, while others perform the actual computation. Hence, it enables application pipelining among the group of cores.

2. Related Work

A proximity-aware directory-based coherence algorithm has been proposed in [4]. It has been proposed for private Level 2 caches. We tweak this algorithm to make it suitable for our architecture. That is, we use the algorithm to maintain cache coherence among the shared Level 3 caches in our architecture, as well as to improve the performance of the caches and reduce congestion on the chip network.

Brown et al. [4] propose an algorithm which is proximity aware. While a cache block may be present in other cores, there is no guarantee that the block will be present in its home core's cache. Instead of making a core always retrieve the block from very slow off-chip memory if it is not present in the home node's cache, proximity awareness is called in to enable the closest sharer of the block to source the required data to the home core. The advantages of this method are decreased latency time for accessing a block and optimized bandwidth utilization. Furthermore, even if the cache block is present in the home node's cache, it might also be present in a cache which is closer to the requestor. Hence, retrieving it from the closer source can help save valuable bandwidth and time.

Freescale [5] has proposed an architecture featuring a shared Level 3 cache. Each core will have a private Level 1 cache on chip, and a private Level 2 cache attached as a back-side implementation, which can significantly improve performance. In order to facilitate the tasks for which a shared cache would be desirable, they have also proposed a multi-megabyte high-bandwidth shared Level 3 cache, which improves the hit rates, while also providing fast memory access for input/output and accelerator blocks.

While the Intel Core 2 Duo [3], or AMD Opteron [6] chips also use a shared cache Level 2 cache, they only consist of two or four cores, which does not complicate the design, or pose problems in cache coherence, as a shared cache is easy to implement for small number of cores. As the number of cores per chip grows, it becomes less viable, as we can see in current chip designs such as the ones used by IBM's Cell Processor (8 cores) and Tilera's Tile64 (64 cores), which do not feature shared caches.

However our implementation ensures that the Level 3 cache is shared among 8 cores at most, thus bringing the benefits of a shared cache to a chip featuring a large number of cores, while at the same time keeping the negative aspects of a shared cache, such as the high bandwidth required, traffic congestion and access time increases, at bay.

The rest of this paper is organized as follows. Section 3 explains in detail the system architecture. Section 4 explains our proposed modifications to the system, and their detailed functioning. Section 5 features our implementation and benchmarking plans. Section 6 concludes.

3. Architecure

3.1 Existing Architecture

Our proposed architecture consists of a 64 core chip multiprocessor, such as the Tilera Tile64 (Fig. 1). It consists of 64 identical processor cores (tiles) interconnected with an on-chip network. Each tile is a comlete full-featured processors, including integrated Level 1 and Level 2 caches.

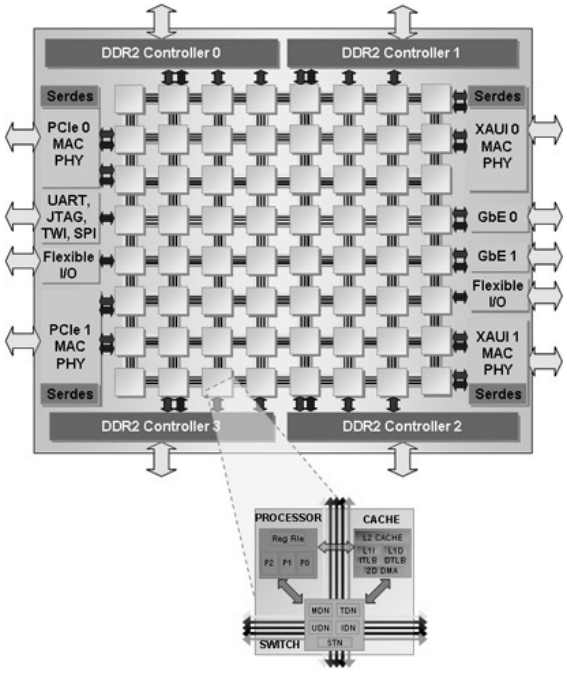

Fig. 1 – The Tilera Tile64 chip multiprocessor block diagram

3.2 Proposed Architecture

In addition to the private Level 1 and 2 caches, that it includes, we add a Level 3 cache for each group of 8 cores.

All the cores of a group will be clustered around the Level 3 cache so as to minimize the cache access time. In addition the Level 3 caches have a means of communication amongst themselves. We note that the non-uniformity of the latencies when access the Level 3 caches. This non-uniformity (particularly the difference between the latency for accessing a nearby node and a node far away) will increase with the increase in the number of cores on the chip. Accesses to Level 3 caches other than a core's home cache will have to be done over the on-chip network, and will incur varying latencies depending on the proximity of the cache, as well as the congestion of the links.

Each Level 3 cache, similar to the one shown below, is connected to the on chip network which enables it to communicate with the other shared caches on the chip.

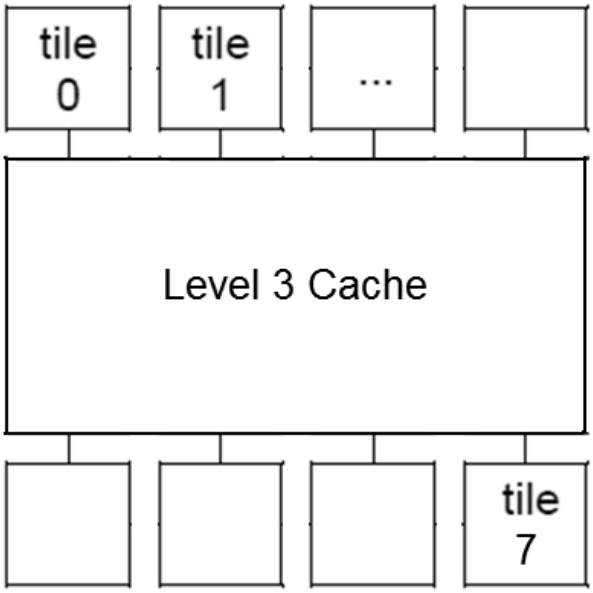

Fig. 2 – The architecture of a group of cores sharing a Level 3 cache

A generic MESI protocol [7], modified for chip multi-processors can be used as the baseline cache coherence protocol upon which the proximity aware algorithm [4] is built. Each cache line is marked with one of four states: (i) M - Modified: The cache line is present only in the current cache, and is dirty; (ii) E - Exclusive: The cache line is present only in the current cache, but is clean; (iii) S - Shared: Indicates that this cache line

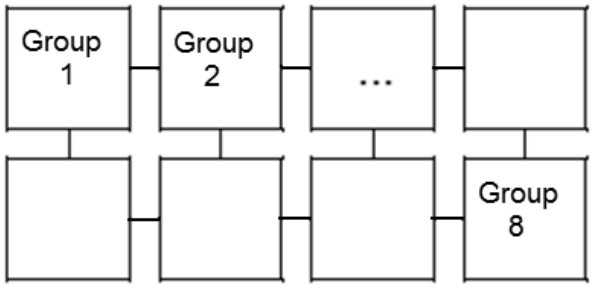

Fig. 3 – Overall architecture for the proposed system

may be stored in other caches of the machine; (iv) I - Invalid: Indicates that this cache line is invalid. Various rules are imposed on the access of a cache line, based on its current state. A cache may satisfy a read from any state except Invalid. An Invalid line must be fetched (to the Shared or Exclusive states) to satisfy a read. A write may only be performed if the cache line is in the Modified or Exclusive state. If it is in the Shared state, all other cached copies must be invalidated first. This is typically done by a broadcast operation known as Read for Ownership (RFO). A cache may discard a non-Modified line at any time, changing to the Invalid state. A Modified line must be written back first. A cache that holds a line in the Modified state must snoop (intercept) all attempted reads (from all of the other CPUs in the system) of the corresponding main memory location and insert the data that it holds. This is typically done by forcing the read to back off (i.e. to abort the memory bus transaction), then writing the data to main memory and changing the cache line to the Shared state. A cache that holds a line in the Shared state must also snoop all invalidate broadcasts from other CPUs, and discard the line (by moving it into Invalid state) on a match. A cache that holds a line in the Exclusive state must also snoop all read transactions from all other CPUs, and move the line to Shared state on a match.

The directory is implemented in memory, but each core will also have a directory cache, which caches directory state. Thus, time is saved, as instead of access the off-chip memory based directory each time, it can access the directory cache itself. Only when there is a miss in the directory cache does the controller need to access the off-chip directory.

4. Proposed Solution

The private cache model used by current chip multi-processors which have a large number of cores suffers from certain innate disadvantages such as the lack of

easy inter-core communication, inability to operate simultaneously on shared data and cache under-utilization, etc.

Our system aims to overcome these disadvantages by use of a shared Level 3 cache. Because such a shared cache does not scale well as the number of cores increase, we partition the cores into groups of, say, 8 cores each, and implement a shared Level 3 cache for each group (Fig. 3). However, this poses the problem of cache coherence. Hence, we use a proximity-aware directory-based cache coherence algorithm, detailed in [4], to maintain the coherence between the shared Level 3 caches. This gives rise to efficient bandwidth utilization and savings both during access to main memory, and access to the other shared caches. It also saves time, as it cuts back on frequent and unnecessary memory accesses.

Proximity aware coherence asserts that if data is on the chip multiprocessor, a request can be satisfied without resorting to an off-chip access. This results in lower access latency and more efficient bandwidth utilization. The algorithm put forth by *Brown et al.* [4] differs from the baseline MESI protocol in a few special cases as follows. If a block is clean, and a request is issued, the request is forwarded to the home node. If the block is not present at the home node during a read miss, but it is present at other nodes, the home node sends a message to the closest sharer, requesting it to forward the required block to the original requestor. The sharer then sends an ACK to the home node to indicate that it has forwarded the block. If all the sharers that the home node sends a message to reply with a NACK, then it quits and falls back to a main memory access to service the request. When a remote node receives a forward request from a home node, it either sends the block if that is possible, and then sends an ACK back to the home node, or it replies straight away with a NACK, if it is not able to forward the block. During a read miss, if a block is not in the home node's cache and is in shared state, forward-exclusive requests are sent to potential shares, with invalidate request being sent in parallel to any potential shares who were not sent forward requests. The remote node's activities are similar to the case of read miss.

The Level 3 cache will use the Least Recently Used cache replacement policy, and will be organized as a set-associative cache.

The advantages of such a system would be to maximize hit rates, as an intervening level in between the Level 2 cache and main memory would reduce the number of required memory accesses. It would also greatly simplify the migration of a running thread to another core without having to resort to using the main memory, which would be very time consuming. The system also reduces cache under-utilization, since, when some cores in a group are idle, the other cores can have access to the whole of the shared resource. The shared cache also reduces the front-side bus traffic as effective data sharing between the cores via the shared Level 3 cache allows requests to be resolved at the shared cache level, instead of going all the way to the main memory. It also facilitates easier and faster inter-core communication and data sharing, which in turn enables the cores to operate simultaneously on shared data, as well as enabling application partitioning/pipelining. The proposed system can be combined with set prediction to increase the performance further.

4.1 Proposed Implementation

We propose to implement this architecture on a chip multiprocessor simulator, such as SESC [8] or CMPSim [9], both of which support a large number of cores per chip, in order to gauge the resulting performance gains quantitatively. We intend to run various benchmarks using this architecture, such as FFT computation, Ocean Simulation, Quick-sort, etc., provided by the SPLASH Benchmark suite [10], to compare it with competing systems.

Proximity aware cache coherence has two distinct advantages – elimination of unnecessary memory accesses, and the minimization of distance travelled by shared data. It will also reduce congestion on the on-chip network, hence improving the performance of the cores.

5. Conclusion

As the number of cores increase, the problems of cache coherence and maintaining high performance become important. We have proposed a system that solves these hurdles, using a three tier architecture, and a proximity aware directory based coherence protocol to maintain cache coherence. Our system improves the performance of the cores by reducing the overhead delays created by private caches, and enabling the cores to make use of shared caches to facilitate inter-processor communication and to work on shared data structures simultaneously, and gives rise to various other benefits.

References

[1] IBM Cell Architecure Project http://www.research.ibm.com/cell/ProductInformation/0,,30_118_8825,00.html

[2] Tilera's Tile64 Multi-core processor - http://www.tilera.com/pdf/ProBrief_Tile64_Web.pdf

[3] Intel's Core 2 Duo Processor Family - http://www.intel.com/products/processor/core2duo/index.htm

[4] Jeffery A. Brown, Rakesh Kumar, and Dean Tullsen. "Proximity-aware directory-based coherence for multi-core processor architectures" in the *Proceedings of the nineteenth annual ACM symposium on Parallel algorithms and architectures,* San Diego, California, USA, 2007, pp. 126-134

[5] A Smarter Approach to Multi Core: Freescale's Next Generation Communications Platform - http://www.freescale.com/files/32bit/doc/white_paper/MULTICOREFTFWP.pdf

[6] AMD Opteron Processor Family http://www.amd.com/us-en/Processors/

[7] J. Laudon and D. Lenoski. "The SGI Origin: A ccnuma Highly Scalable Server", in the In ISCA '97: Proceedings of the 24th annual international symposium on Computer architecture, pp 241–25, New York, NY, USA, 1997.

[8] SESC: cycle accurate architectural simulator - http://sesc.sourceforge.net/, Jose Renau, University of California, Santa Cruz.

[9] CMP-SIM: An Environment for Simulating Chip Multiprocessor (CMP) Architectures - http://www.utdallas.edu/~rama.sangireddy/CMP-SIM/. Dr. Rama Sangireddy, Assistant Professor, Department of Electrical Engineering, University of Texas at Dallas

[10] Steven Cameron Woo, Moriyoshi Ohara, Evan Torrie, Jaswinder Pal Singh, and Anoop Gupta. "The SPLASH-2 Programs: Characterization and Methodological Considerations", in the *Proceedings of the 22nd International Symposium on Computer Architecture*, pp. 24-36, Santa Margherita Ligure, Italy, June 1995

[11] M. M. Michael and A. K. Nanda. Design and performance of directory caches for scalable shared memory multiprocessors. In HPCA '99: Proceedings of the 5th International Symposium on High Performance Computer Architecture, page 142, Washington, DC, USA, 1999. IEEE Computer Society