Adaptive Zone-Aware Multi-bank on Chip Last Level L2 Cache Partitioning for Chip Multiprocessors

Nitin Chaturvedi
Electrical Electronics & Engineering
Birla Institute of Technology & Science,
Pilani, India-333031

Jithin P Thomas
Electrical Electronics & Engineering
Birla Institute of Technology & Science,
Pilani, India-333031

S Gurunarayanan
Electrical Electronics & Engineering
Birla Institute of Technology & Science,
Pilani, India-333031

ABSTRACT

This paper proposes a novel efficient Non-Uniform Cache Architecture (NUCA) scheme for the Last-Level Cache (LLC) to reduce the average on-chip access latency and improve core isolation in Chip Multiprocessors (CMP). The architecture proposed is expected to improve upon the various NUCA schemes proposed so far such as S-NUCA, D-NUCA and SP-NUCA[9][10][5] in terms of average access latency without a significant reduction in the hit rate. The complete set of L2 banks is divided into various zones. Each core belongs to one particular zone which is the closest to it. Consequently, adjacent cores are grouped into the same zone. Each zone individually follows the SP-NUCA scheme [5] for maintaining core isolation and sharing common blocks. However, blocks that need to be shared by cores which belong to different zones are replicated. This scheme is much more scalable than the SP-NUCA scheme and bounds the maximum on-chip access latency to a lower value as the number of cores increases.

This paper merely details the proposed scheme. The claims made regarding the benefits of the scheme shall be substantiated through simulations and a detailed comparative study in the future. The intended simulation methodology and architectural framework to be used in this regard have also been mentioned.

General Terms

Computer Architecture

Keywords

Chip Multiprocessor (CMP), Non-Uniform Cache Architecture (NUCA) and Shared Last level Cache (LLC)

1.INTRODUCTION

Present sub-micron integrated circuit technologies have fueled microprocessor performance growth. Each new process technology increases the integration density thus allows for higher clock rates and also offers new opportunities for micro-architectural innovation. Both of these are required to maintain microprocessor performance growth. Micro-architectural innovations employed by recent microprocessors include multiple instruction issue, dynamic scheduling, speculative execution, instruction level parallelism and non-blocking caches. In the past, we have seen the trend towards CPUs with wider instruction issue and support for larger amounts of speculative execution but due to fundamental circuit limitations and limited amounts of instruction level parallelism, the superscalar execution model provides diminishing returns in performance for increasing issue width. Faced with this situation, building further a more complex wide issue superscalar processor was not at all the efficient use of silicon resources and a better utilization of silicon area. So researchers came up with a new Novel architecture which was constructed from simpler processors then superscalar and multiple such processors are integrated on a single chip popularly known as chip multiprocessor or multi-core processor. Researchers faced two important challenges for next generation microprocessors are the slow main memory and the limited off-chip bandwidth. Efficient management of the last level on-chip cache is therefore important in order to accommodate a larger number of cores in future multi-core architectures. Previous research has shown that a last-level multi-core cache can be organized as private partitions for each core or having all cores sharing the entire cache.

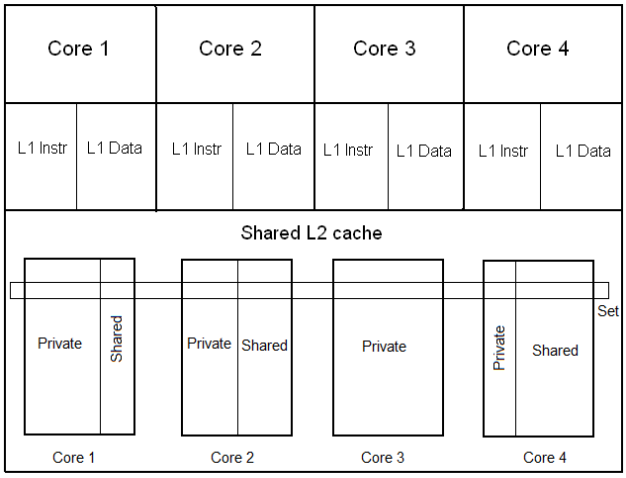

Figure 1. Chip Multiprocessor

Previous results show that the shared cache organization can be utilized more flexibly by sharing data between cores. However, it is slower than a private cache organization. In addition, private caches do not suffer from being polluted by accesses from other cores by which we mean that other cores displace blocks without contributing to a higher hit rate. Non-uniform cache architectures (NUCA) are a proposed hybrid private/shared cache organization that aims at combining the best of the two extreme organizations [1, 2, 3, 4, 6] by combining the low latency of small (private) caches with the capacity of a larger (shared) cache. Typically, frequently used data is moved to the shared cache portion that is closest to the requesting core (processor); hence it can be accessed faster. Recently, NUCA organizations have been

studied in the context of multi-core systems as a replacement for a private last-level cache organization [3, 4]. The cache is statically organized into private partitions but a partition attached to one core can also keep blocks requested by other cores. When a block is installed in a certain partition, a replaced block from that partition will be installed in a neighbor's partition, picked by random. As a result, on a miss in one partition, all other partitions are first checked before accessing main memory. While this hybrid scheme provides fast access to most blocks, it can suffer from pollution because of the uncontrolled way by which partitions are shared among cores.

This paper provides a novel NUCA design for multi-cores based on private partitioning zones in which the sizes of the core-local partitions that are shared are chosen adaptively to maximize the overall performance. We will show that our adaptive scheme outperforms the uncontrolled sharing of blocks in private zones with reduced latency.

The rest of the paper is organized as follows: Related work is described in Section 2, Section 3 described the motivations for this work and section 4 provides proposed implementation details and explains the novel adaptive partitioning scheme SPR_NUCA with Architecture and Coherency protocol to improve performance limits of CMP. Section 5 provides Simulation methodology. Section 6 discusses the Benchmark Suite to be used and Section 7 concludes.

2. RELATED WORK

Kim et al. [4] introduced the concept of Non-Uniform Cache Architecture (NUCA). They observed that increasing wire delays would mean that cache access times were no longer constant. Instead, latency would become a linear- function of the line's physical location within the cache. On the basis of this observation, they designed several NUCA architectures by partitioning the cache into multiple banks and using a switched network to connect these banks. Two main alternatives have been proposed: Static NUCA (S-NUCA) and Dynamic NUCA (D-NUCA). Both designs organize the multiple banks into a two-dimensional switched network. The difference between the two architectures is the *Placement Policy* they manage. While in S-NUCA architecture, data are statically placed in one of the banks and always in the same bank, in D-NUCA architecture data can be promoted to be placed in closer and master banks. Although this promotion allows D-NUCA to potentially outperform S-NUCA, the D-NUCA benefit is significantly diminished by the quality of the bank-searching algorithm within the cache. Two alternative bank replacement policies are proposed: *zero-copy* and *one-copy*.

CMPs present additional challenges for on-chip cache management. First, a cache on a CMP requires multiple ports to provide appropriate bandwidth. Secondly, multiple threads mean multiple working sets, which compete for limited on-chip storage. Finally, sharing code and data interfere with block migration, since one processor's low latency bank is other processor's high latency bank. Beckmann and Wood [1] gathered the current proposals for managing wire delays and combined them with Chip Multiprocessors. They demonstrated that block migration is less effective for CMP because 40-60% of hits in commercial workloads were satisfied in the central banks. Block migration effectively reduced wire delays in uni-processor caches. However, to improve CMP performance, the capability of block migration relied on a smart search mechanism that was difficult to implement.Huh et al. [9] introduced the concept of the sharing degree in a NUCA bank. The sharing degree is the number of cores that share a specific bank, so a sharing degree of one signifies a private cache. Larger sharing degrees reduce the number of misses, thus optimizing the cache capacity usage. Unfortunately, smaller sharing degrees reduce hit latencies. An ideal design would capture the benefits of both reduced misses and reduced hit latencies. Although D-NUCA performance potential dramatically outperforms that of other mechanisms, the benefits currently offered by D-NUCA organization do not justify the complexity of the design. They also concluded that the simplest design– an S-NUCA organization with a small sharing degree–was probably the best. Muralimanohar and Balasubramonian [15] proposed a different approach in NUCA architectures. These authors proposed the use of two different physical wires to build NUCA architectures. One of these wires provided lower latency and the other higher provided bandwidth. They then proposed two different bank searching algorithms Chishti et al. [11] proposed an alternative to NUCA architecture named Non-uniform access with Replacement and Placement using Distance associativity (NuRAPID). This architecture is based on decoupling data and tag placement. NuRAPID stores tags in a bank close to the processor, optimizing tag searches. Whereas NUCA searches tag and data in parallel, NuRAPID and D-NUCA achieve similar results, although NuRAPID heavily outperforms DNUCA in power efficiency. The NuRAPID version for CMP is known as CMP-NuRAPID and was also proposed by Chishti et al. [2]. This proposal mitigates some of the effects described by Beckmann and Wood [1].Our work is inspired by earlier work on dynamic partitioning of the resources in shared caches among cores [3, 6, 7]. In the NUCA setting, the new issue becomes how to select the size of the shared partition which is not addressed in the earlier work. We combine and extend several existing mechanisms intended for solving problems in other contexts. The contribution of this paper is the unique combination and usage of these mechanisms and the novel architecture for the last-level cache. In our organization, hits to the private partitions are fast, while hits to neighboring partitions are slower.

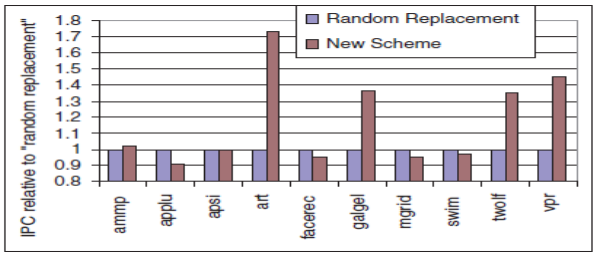

Figure 2. Relative Performance of the two schemes [7]

The size of the private partition is dynamically controlled and balanced against the other cores. A NUCA proposed by Dybdahl [7] works better than the hybrid scheme based on CMPs with private caches given by Sohi [16] except in cases where cores are competing for cache resources. They provide comparable

performance for many applications that do not use the last level cache.

3. MOTIVATIONS

With the evolution of technology, the number of cores that can be incorporated onto a single chip is constantly on the rise. However, the growth in off-chip bandwidth is progressing at a much a slower pace. Consequently, the pressure on the on-chip memory hierarchy to satisfy as many memory requests as possible is mounting. This is especially true at the last-level cache (L2/L3) where there is a strong contention between the various cores. Hence, an optimal dynamic partitioning scheme is required which would dynamically partition the cache according to the activity level of each core such that there is an overall reduction in the miss rate. An additional problem that arises with the increase in the number of cores is a corresponding increase in the size and associativity of the last level cache. A high associativity would directly translate into an increased latency per cache access. A multi banked cache would be a better option in this regard in comparison to a monolithic high associativity cache.

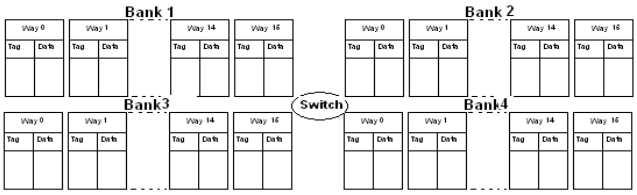

Figure 3. Multi-banked shared LLC

Thus, frequently accessed blocks could be placed in those banks closest to the processors that use them. However, this could also lead to these blocks ping-ponging between banks. Thus, blocks which are actively used by two or more cores should be placed in an optimally positioned bank such that the average access latency for the concerned cores is minimized. This is the primary aim of Non-Uniform Cache Architecture (NUCA) schemes such as S-NUCA, D-NUCA and SP-NUCA [10][11][4]. However, these schemes do not support the replication of shared blocks. Thus, scalability is limited in terms of the average access latency. For example, consider a system with 16 cores. Now, if two cores placed at two extreme ends in the cache layout were to share a block, optimally it would have to be placed in one of the centrally positioned banks. But it would still take a considerable number of cycles for each of those cores to access the block from the central bank. If, however, replication is allowed, the block could be replicated and each core could keep a copy in a bank close to it. This is what SPR-NUCA attempts to do. It is based on the SP-NUCA scheme and improves its scalability by allowing the replication of blocks when necessary. It considers the distance between the cores sharing a particular block. If it exceeds a certain limit, then the block is replicated and each core keeps a copy in one of the banks closest to it.

4. SPR_NUCA Architecture and Coherency Protocol

In this section, we presented the details of the proposed SPR-NUCA scheme for the Last-Level Cache (LLC). The LLC is organized in a multi-banked manner. The total set of banks is divided into various zones. Each zone comprises of a set of banks which are adjacent to each other. Similarly, each core also belongs to the zone closest to it. The number of cores that would comprise a zone has to be decided through simulations. However, four cores per zone seem to be an optimal option.

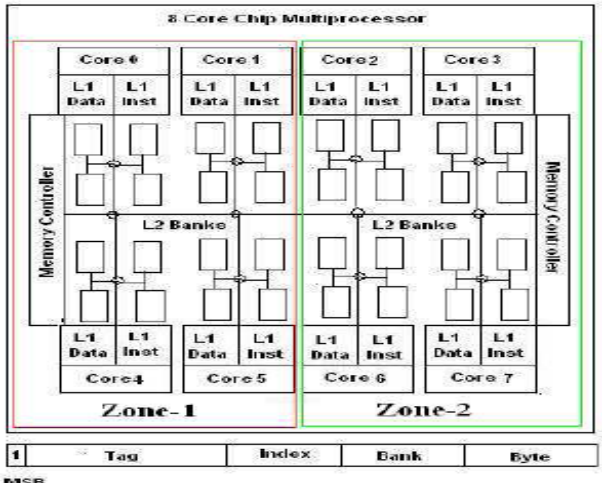

Figure 4. Layout of Chip Multiprocessor

The blocks in the LLC can be grouped into three types: private, shared and replicated. Private blocks are those which are used only by a single core. They are hence placed in one of the banks closest to that particular core. Both shared as well as replicated blocks are frequently used by two or more cores. However, shared blocks are used by processors which belong to the same zone. Replicated blocks, on the other hand, are accessed by cores belonging to different zones, ie, cores which are farther apart. Each bank can hold blocks of any of these three types. It is to be noted that within a single zone, a block can have only one copy. Replicas, if present, can reside only in different zones. Remember that a replicated block simply means that the block has more than one copy in the entire LLC. However, with respect to a particular zone, it could be either in the private bank of one of the cores or in the shared bank of that zone.

Since replication is also supported in SPR-NUCA, we intend to use a directory-based coherence protocol for keeping track of the zones which contain a copy of the block. The directory could either be a centralized or a distributed one. To reduce the directory width, the locations of the different copies of a block would be marked in terms of zones instead of individual banks. Once a request arrives at the LLC, a request is sent to the corresponding private bank of the requesting core as we. If this does not result in a hit, the request is forwarded to the remaining private banks in the zone as well as to the shared bank in which that block could reside in that particular zone. In case of a hit, there are two possibilities. If the block is found in the shared bank, the data is sent to the requestor. Else if it is found in one of the private banks, it is moved to the shared bank. In case of a miss, the directory entry of the requested block is searched. If it turns out that no zone contains the block, then it is an LLC miss and the request is forwarded to the main memory, following which, the block is brought into the private bank of the requestor. Else, the request is then forwarded to the nearest zone which

contains a copy of the block. Within that zone, the block is searched in all the private banks in which it can reside and also the corresponding shared bank in parallel. The block is then replicated into the private bank of the requestor. In addition to this, the bit corresponding to the requestor's zone in the directory entry of the block is set to one. When a block arrives from memory, it is placed in one of the private banks of the requesting core depending on the address. The block remains there as long as no other core in its own zone requests it. Even if cores from other zones access the block it is merely replicated and not removed from the current bank.

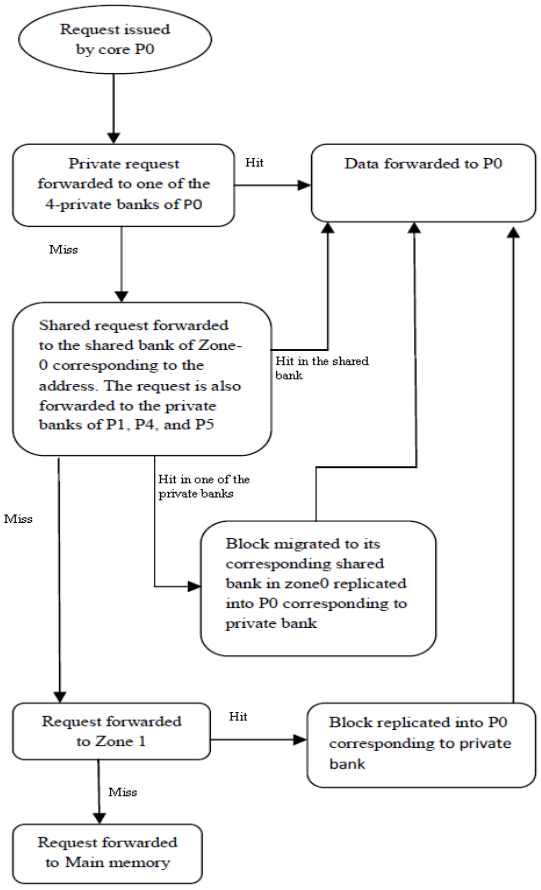

Figure 5. Search Algorithm

Hence, future accesses by the same core to the block would be faster. SPR-NUCA has a much better scalability than the other NUCA schemes such as S-NUCA, D-NUCA and SP-NUCA which do not support replication. In SP-NUCA [4], any block which is utilized by two or more cores is placed in a particular fixed shared bank. This does not cause a problem in terms of access latency if the number of cores is small (4 to 8). However, as the number of cores increases, the distance of each core to the bank farthest from it will keep increasing. Hence, such a block could be placed in a bank which is quite far away from the cores accessing it. This is why SPR-NUCA divides the LLC into small-sized zones. This ensures that a block frequently used by a particular core is not placed beyond a certain distance from the core. This limit remains the same as long as the size of the zone is a constant and does not change as the number of cores in the system increases, implying scalability.

Consider a system with 2^n banks per zone and 2^p cores per zone. Thus each core has 2^{n-p} private banks. Note that any of these 2^n banks can hold shared/replicated blocks. To interleave the addresses across the chip, the bank number is mapped using the lower bits of the address. The interpretation of an address is done in the same manner as in SP-NUCA. As shown in the figure, the address is divided as follows: The lowest *B* bits are used to select the byte in the cache block. Then, we use the *n-p* bits to select the private bank or the *n* bits to select the shared bank in that zone depending on whether it is a private or a shared request. The next *i* bits, the index, are used to select the corresponding set in the bank and the rest of the address is the tag. It is important to note from the figure that the address remains the same for both private and shared blocks. However, they are interpreted differently as shown. The private tag is *p* bits bigger than the shared one, but as they are to be stored in the same tag array, it must have the size of the private tag, increasing slightly the required area of LLC banks by *p* bits per line. In order to differentiate between private and shared blocks, an extra private/shared bit is also to added to the block. This bit would be present in the cache requests as well.

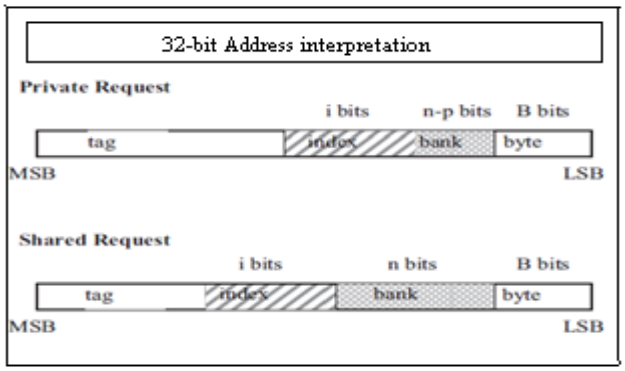

Figure 6. Address interpretation for private and shared requests

Replacement policy: Merino et al.[4] proposed three different LRU replacement policies for the SP-NUCA scheme – "always steal", "shadow-tag" based policy and a global LRU replacement policy – to partition each bank into shared and private portions in an optimal manner. However, simulation results showed that the global LRU policy gave the best overall performance. However, as pointed out by Kedzierski et al.[6], true LRU imposes extraordinary complexity and area overheads when implemented on high associativity caches. Thus, we intend to use a pseudo-LRU policy such as the Not Recently Used (NRU) policy to attain an LRU-like behavior without much degradation in performance.

5. SIMULATION METHODOLOGY

For evaluating the performance of the CMP requires a way of simulating the environment in which we would expect these architectures to be used in real systems. We will use Virtutech Simics [9] full system functional simulator extended with Multi-facet GEMS which is popularly used in the research community. The heart of GEMS is the Ruby memory simulator. The Ruby

Cycle is our basic metric of simulated time used in the Ruby module of the GEMS simulator. The Ruby module is the basic module for the memory system configuration and interconnection network design The Ruby cycles are the recommended performance metric in GEMS. The base line configuration to be used is given below

Number of cores	8
Core processor	Out-of-order SPARCv9
Main memory size	4 GBytes
Memory bandwidth	512 bytes/cycle
On-chip wire delay	1 cycle
Off-chip wire delay	20 cycles
Switch delay	1 cycle
Private L1 data caches	8 KBytes
Private L1 instruction caches	8 KBytes
Shared L2 NUCA cache	1 MB

Figure 7. Base line Configuration

6. PARSEC BENCHMARK SUITE

The Princeton Application Repository for Shared-Memory Computers (PARSEC) has been recently released [10]. This benchmark suite includes emerging applications and commercial programs that cover a wide area of working sets and allow current Chip Multiprocessor technologies to be studied more effectively. In this paper we will evaluate a subset of the PARSEC benchmark suite. We also consider the simlarge inputs of PARSEC benchmarks in our simulations. We will fill the cache by executing 100 million instructions and finally we collect the statistics for the following 500 million instructions

7. CONCLUSIONS

The current advanced submicron integrated circuit technologies require us to look for new ways for designing novel microprocessors architectures and non-uniform cache architectures to utilize large numbers of gates and mitigate the effects of high interconnect delays. In this paper, we have discussed a novel dynamic partitioning scheme known as Adaptive Block Pinning which attempts to partition the last-level shared cache in a judicious and optimal manner thereby increasing overall system performance. Future work could be directed at modifying this scheme to work for multi-banked caches. It could also aim at reducing the latency penalties by attempting to place each cache block nearest to the core that most frequently uses it.

8. REFERENCES

[1] B. M. Beckmann and D. A. Wood. Managing wire delay in large chip-multiprocessor caches. In *37th International Symposium on Microarchitecture*, 2004.

[2] J.Chang and G.S.Sohi, "Cooperative Caching for Chip Multiprocessors", ISCA,2006.

[3] E.Herrero, J.Gonzalez and R.Canal, "Distributed Cooperative Caching", PACT,2008.

[4] H. Dybdahl and P. Stenstrom. An Adaptive Shared/Private NUCA Cache Partitioning Scheme for Chip Multiprocessors. In *Proceedings of the 13th Annual International Symposium on High Performance Computer Architecture, 2007.*

[5] J.Merino, V.Puente, P.Prieto and J.A.Gregorio, "SP-NUCA: A Cost Effective Dynamic Non-Uniform Cache Architecture", ACM SIGARCH Computer Architecture News, Vol. 36, No.2, May,2008

[6] B.M. Beckmann, M.R. Marty, D.A. Wood, "ASR: Adaptive Selective Replication for CMP Caches", MICRO 2006.

[7] K.Kedzierski, M.Moreto, F.J.Cazorla, M.Valero, "Adpating Cache Partitioing Algorithms to Pseudo-LRU Replacement Policies", IPDPS,2010

[8] M.K.Qureshi and Y.N.Patt, "Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches", MICRO, 2006

[9] M.Zhang and Krste Asanovic, "Victim Replication: Maximizing Capacity while Hiding Wire Delay in Tiled Chip Multiprocessors", ISCA, 2005

[10] J.Huh, C.Kim, H.Shafi and L.Zhang, "A NUCA Substrate for Flexible CMP Cache Sharing", ICS, 2005

[11] C. Kim, D. Burger and, S. W. Keckler, "An Adaptive, non-uniform cache structure for wire-delay dominated on-chip caches". ASPLOS X, pp. 211-222, October 2002.

[12] Z. Chishti, M. D. Powell, and T. N. Vijaykumar. Distance associativity for high-performance energy-efficient non-uniform cache architectures. In *36th International Symposium on Microarchitecture*,2003.

[13] J. Huh, C. Kim, H. Shafi, L. Zhang, D. Burger, and S. W. Keckler. A nuca substrate for flexible cmp cache sharing. In *19th ACM International Conference on Supercomputing*, 2005.

[14] C. Kim, D. Burger, and S. W. Keckler. An adaptive, non-uniform cache structure for wire-delay dominated on-chip caches. In *10th International Conference on Architectural Support for Programming Languages and Operating Systems*, 2002.

[15] N. Muralimanohar and R. Balasubramonian. Interconnect design considerations for large nuca caches. In *34th International Symposium on Computer Architecture*, 2007.

[16] J. Chang and G. S. Sohi. Cooperative caching for chip multiprocessors. In *33rd International Symposium on Computer Architecture*, 2006

[17] C. Bienia, S. Kumar, and K. Li. Parsec vs. splash-2: A quantitative comparison of two multithreaded benchmark suites on chip-multiprocessors. In *Procs. Of the IEEE International Symposium on Workload Characterization*, IISWC 2008.

[18] C. Bienia, S. Kumar, J. P. Singh, and K. Li. The parsec benchmark suite: Characterization and architectural implications. In *International Conference on Parallel Architectures and Compilation Techniques*, 2008.

[19] P. S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt, and B. Werner. *Simics: A Full System Simulator Platform*, volume 35-2, pages 50–58. Computer, 2002.