Group-caching for NoC Based Multicore Cache Coherent Systems

Wang Zuo, Shi Feng, Zuo Qi, Ji Weixing, Li Jiaxin, Deng Ning, Xue Licheng, Tan Yuan, Qiao Baojun
High Performance Embedded Computation Lab
School of Computer Science and Technology, Beijing Institute of Technology
Beijing, P. R. China
wuchanjian.wang@gmail.com

***Abstract*—Most CMPs use on-chip networks to connect cores and tend to integrate more simple cores on a single die. Low-radix networks, such as 2D-MESH, are widely used in tiled CMPs since they can be mapped to on-chip networks efficiently. However, low-radix networks introduce high network latency caused by long diameter. In this paper, we propose the use of group-caching design in NoC based multicore cache coherent systems. In our design, on-chip L2 banks are organized to form multiple groups. Each cache group behaves like a shared L2 cache for the cores inside cache group while the cache coherence between cache groups is maintained by coherence messages. Besides, group-caching also adopts the new cache replacement policy to improve the inefficient use of the aggregate L2 cache capacity. Compared to banked and shared L2 design, as most L2 accesses are served by local cache group, the hop count is significantly reduced. Experiment results based on full-system simulation show that for 2D-MESH, group-caching can increase the performance by 2%~8% compared to banked and shared L2 design, with network energy consumption reduced by 11%~13%. Experiment results also show that the communication overhead inside cache group plays an important role in the performance of group-caching.**

Keywords-CMP; NOC; network latency; L2 banks; cache coherence; group-caching; performance; power

I. INTRODUCTION

In order to utilize the increasing number of transistors available in modern VLSI technology, processor designs tend to integrate more simple cores instead of complicated processor on a single die. As the number of cores increases in CMPs, on-chip networks are widely used to provide efficient communication between cores. There are two different ways to implement on-chip networks. One is to use low-radix networks, such as 2D-MESH found in RAW processor [1], TRIPS processor [2], Intel's Teraflops [3] and Tile64 processor [4]. Because of simple topologies, low-radix networks can be mapped to on-chip networks with compact physical layout using conventional routing in horizontal and vertical directions. Especially in RAW processor, the longest wire in the system is no greater than the length or width of a tile [1]. Although low-radix networks can be mapped efficiently, they also lead to several disadvantages, such as high network latency caused by long diameter. Soteriou [7] presented the on-chip networks' traffic model based on real-life traffic traces obtained from full system simulations. They derived the hop count distribution model for 2D-MESH on the assumption that an optimal mapping should place communicating nodes as close as possible. However, such optimal mapping is hard to implement in reality. The other option is to use high-radix networks. Balfour and Dally [6] proposed the use of concentrating MESH to reduce the hop count. James [5] also demonstrated how the flattened butterfly topology can be applied to on-chip networks. However, wiring requirements of high-radix networks complicate the layout which leads to higher area cost [6].

To our knowledge, most of current designs tend to use low-radix networks as the connection substrate and develop effective cache managements to reduce the hop count. A hybrid of private, per-processor tag arrays and a shared data array was used to realize controlled replication, in-situ communication and capacity steeling in paper [20]. Zhang [21] presented a new cache management policy, which combines the advantages of private and shared schemes. Beckmann [22] developed L2 cache designs that incorporate three latency management techniques, including block migration, Transmission Line Caches (TLC) and traditional stride-based hardware pre-fetching. Jichuan [10] Chang used private L2 caches to cooperate caching just like a shared L2 cache. Liqun Cheng [19] presented the work that combined efforts on both network design and cache management to improve the performance of CMPs. They proposed the interconnections composed of wires with varying latency, bandwidth and energy characteristics, and advocate intelligently mapping coherence operations to the appropriate wires.

In this paper, we propose the use of group-caching design in NoC based multicore cache coherent systems. The principle of group-caching is to attract the frequently accessed data in local cache group so as to reduce remote on-chip references. In our design, on-chip L2 banks are organized to form multiple groups. Each cache group behaves like a shared L2 cache for the cores inside cache group while the cache coherence between cache groups is maintained by coherence messages. Besides, in order to improve the inefficient use of the aggregate L2 cache capacity, group-caching uses new cache replacement policy to keep data on-chip as much as possible.

The rest of this paper is organized as follows. Group-caching design is proposed for NoC based multicore cache coherent systems in section 2. We evaluate group-caching in section 3 and conclude in section 4.

This paper is partially supported by Beijing Key Discipline Program. This paper is also supported in part by High-Tech Research and Development Program of China under grant No. 2007AA01Z109 and NSFC under Grant 60806014

978-3-9810801-5-5/DATE09 © 2009 EDAA

II. ON-CHIP MEMORY SYSTEM

A. Bankground

Although the on-chip memory systems of CMPs have borrowed heavily from that of traditional multiprocessors built with multiple chips, there are still some differences between them. First, most traditional multiprocessors use directory to maintain the cache coherence between nodes. Either directory or snooping is used to maintain the cache coherence inside node. Node here indicates a processor or a group consisting of multiple processors. For most current CMPs, the on-chip systems often use the shared L2 design to maintain the cache coherence between cores. The information of data sharing between private L1 caches is recorded in the shared L2 cache. Second, when an L2 cache read misses, the memory systems of traditional multiprocessors would not try to fetch clean data from L2 caches in other processors since the latency of getting clean data from memory would typically be lower than getting it from another L2 cache. The situation for on-chip memory systems is very different: the transfer of clean data between on-chip L2 caches (or L2 banks) can be done relatively easily and efficiently while off-chip access is costly. Paper [10] shows that for commercial workloads, sharing on-chip clean data between L2 caches can reduce 10-50% of off-chip misses, with only a slight increase in traffic for on-chip networks.

Banked and shared L2 cache design is widely used in tiled CMPs. In order to reduce the conflict of accessing the same bank, the physical address of shared L2 cache is interleaved across L2 banks. However, interleaving also leads to some disadvantages. First, when a thread tries to access a physical memory page, the cache system will divide this page to several blocks and load these blocks into different L2 banks. The overheads of accessing the different parts of the same page differ from each other: some may be low, such as accessing local L2 bank, while others may be quite high. Second, if two or more threads share the same block, these threads have no choice but to access the unique block (since all L2 banks form a logical shared L2 cache, any data has the only copy in L2 cache) no matter how high the overheads are. Thus, banked and shared L2 cache design is a good choice when the latency of accessing remote L2 banks is not significant. For example, paper [8] and [9] use the low latency crossbars to connect distributed L2 banks. But, if we use low-radix networks as the connection substrate, the disadvantages of banked and shared L2 cache will be exaggerated. As data are spread evenly across all banks, only a small fraction of references are satisfied by the local bank while other references may need to jump several hops to access remote L2 banks. Thus, the efficiency of cache system is heavily influenced by the network diameter in banked and shared L2 design.

We think that there are two policies to reduce the hop count. First, when a thread tries to load a page into L2 cache, this page should be loaded into the L2 banks which are close to the core running this thread. Second, if two or more threads share the same data, multiple data copies should be introduced into L2 cache with each copy placed close to its corresponding core. Besides, we also need to control the number of data copies properly, otherwise large number of copies will prick up the inefficient use of the aggregate L2 cache capacity so as to increase the number of off-chip accesses.

B. Group-caching Design

In order to implement the two policies referred above, we propose the use of group-caching in NoC based multicore cache coherent systems. Taking 2D-MESH for an example, we create multiple cache groups by the L2 banks located in 4-node rectangle. As shown in Fig.1, 16 L2 banks are divided into 4 cache groups. We choose banked and shared L2 design for the cache organization inside cache group while maintaining the cache coherence between cache groups by query messages and invalidation messages. Query messages are used to fetch data from other cache groups while invalidation messages are used mark other data copies invalid.

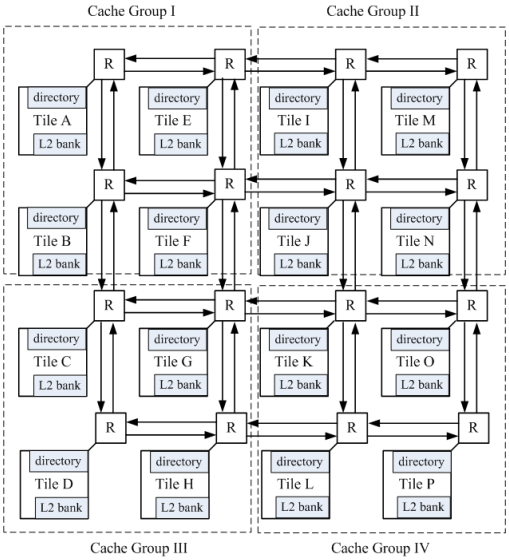

Figure 1. Applying group-caching in 2D-MESH

When an L2 cache read access is missed in local cache group, query messages will be multicasted to the other cache groups to check whether the data (dirty or clean) is already loaded into chip. If so, an off-chip memory access is avoided by obtaining the data from other cache groups. If there are more than one data copies in different cache groups, we will choose the nearby cache group walking clockwise to reply query messages. As shown in Fig.1, when cache group I, II and III both have the data block which required by cache group IV, we will choose cache group III to reply the query message. When writing L2 cache occurs, invalidation messages are also multicasted to the other cache groups to mark other data copies invalid. In order to reduce the interruptions of accessing the L2 banks by the cores in local cache group, we implement distributed directories by duplicating the tag architecture of L2 caches. For example, when an L2 cache read misses in the L2 bank of tile B, query messages will be multicasted to the directories of tile J, tile D and tile L to judge whether to carry an off-chip access or not. Because the physical address of cache group is interleaved across its 4 L2 banks, the data, which can be cached in L2 bank B, can only be cached in the L2 banks of tile J, tile D or tile L. We should note that the way, multicasting query messages to other cache groups, differs

from traditional snooping used in bus-based CMP. First, query messages are not required to broadcast to all directories. Second, each directory does not really "snoop" network but behaves message-driven.

As duplicated L2 tags do not hold the information of data sharing between cache groups, writing L2 cache will always trigger invalidation messages multicasted to other cache groups. In order to reduce the overhead of multicasting invalidation messages, we introduce a bit vector for each duplicated tag to indicate the data copies' location. In the case of 4 cache groups, we use a 3-bit vector to indicate whether there are data copies in other cache groups walking clockwise. For example, the vector value "001" in cache group I indicates there is a data copy in cache group III while "010" indicates there is a data copy in cache group IV. When writing L2 cache occurs, the local tag's bit vector is checked whether there are on-chip data copies. If so, invalidation messages are sent to the target cache groups; otherwise, the local block is updated directly.

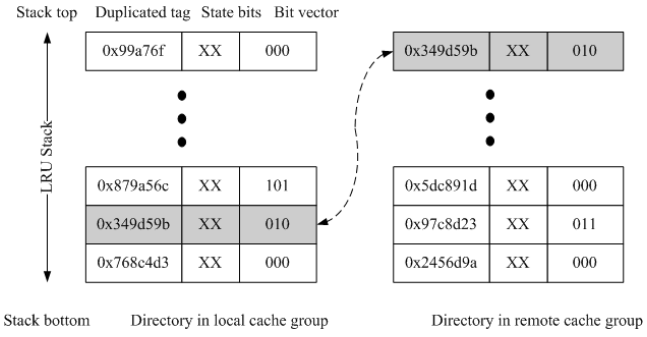

Figure 2. Architecture of distributed directory

Except for reducing the overheads of maintaining the cache coherence between cache groups, the bit vectors are also utilized to improve the cache replacement. Because off-chip access is costly, evicting a unique data block (has no copies in other cache group) from local cache group may lead to large overhead caused by reloading this block back. However, evicting a non-unique data block does not have such problems since fetching this block from another cache group is efficient. Thus, when we evict a data block from local cache group, first check local tags: if the block, selected by LRU, does not have copies in other cache groups, we may choose another block which is non-unique. As shown in Fig.2, although block 0x768c4d3 is at the bottom of LRU stack, we will first evict block 0x349d59b from L2 cache because it has a copy in another cache group.

To sum up, group-caching introduces the "group" design used in memory systems of traditional multiprocessors to the on-chip memory systems. L2 banks are organized to form multiple groups. Each cache group behaves like a banked and shared L2 cache while the cache coherence between cache groups is maintained by query message and invalidation messages. Besides, group-caching also adopt the new cache replacement policy to improve the inefficient use of the aggregate L2 cache capacity. We compare group-caching to banked and shared L2 design: for applications with low data sharing between threads, because private data of a thread is loaded into local cache group which is close to the core running this thread, most L2 accesses are severed by local cache group and the hop count is reduced; for applications with high data sharing between threads, because multiple data copies are loaded into different cache groups which corresponds to cores running these threads, most L2 accesses are also severed by local cache group and the hop count is also reduced. Compared to cooperative caching [10], group-caching still has some advantages. Cooperative caching uses Central Coherence Engine (CCE) to backup the tags of all private L2 cache and adopts special point-to-point network to connect CCE and cores. Even if we ignore the extra cost of special network and the possible performance bottle-neck of CCE, placing every core in the same distance to CCE is a hard work. Thus, paper [10] only implemented cooperative caching in 4-core and 8-core CMP. However, group-caching does not have such layout problems as distributed directories can be easily integrated into each tile. Adding in bit vector overhead, the total capacity of distributed directories is 832KB (a 16-core CMP with a 4-way associative 2MB per-core L2 cache, assuming a system with 16G total physical memory and a 128-byte block size), increasing the on-chip cache capacity by 2.4%. This ratio is similar to Piranha [8] and cooperative caching [10].

When the size of network increases, L2 read misses will get more expensive as they must contact directories that could be anywhere on-chip. Taking 64-core CMP for an example, an L2 read miss will trigger query messages multicasted to 16 different directories assuming that each cache group is formed by the L2 banks located in 4-node rectangle. Thus, for large-scale CMPs, we need to increase the size of cache group. For example, cache groups can be formed by the L2 banks located in 16-node 2D-MESH for 64-core CMP.

It is worth noting that group-caching can also be applied to other topologies. We describe 9-node triplet-based topology in Fig.3 and its physical layout in Fig.4. As shown in Fig.4, the tiles placed vertically in a row construct a cache group, such as tile A, tile B and tile C. Compared with the cache group formed by 4-node rectangle, the communications between the nodes inside a 3-node triplet only need one hop while diagonal nodes in a 4-node rectangle need 2 hops to interact with each other. We will show the negative impact of an extra hop inside cache group on the performance in our experiment.

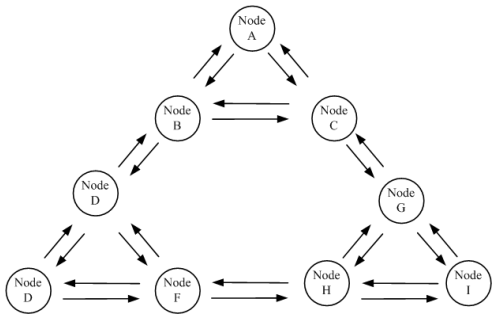

Figure 3. 9-node triplet-based topology

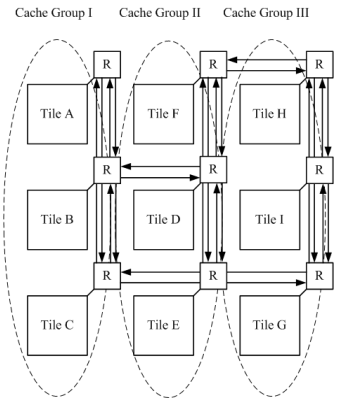

Figure 4. The physical layout of 9-node triplet-based network

III. EXPERIMENT METHODOLOGY AND EVALUATION

A. Methodology

In order to describe the latencies of accessing different L2 banks accurately, we use Princeton's Garnet network simulator [11] to simulate current on-chip networks. We select Garnet's fixed pipeline model in our experiment. Fixed pipeline model is intended for low-level interconnection network evaluations and models the detailed features of a state-of-the-art network. Table I lists parameters used by the fixed pipeline model. Flit is the smallest item transferred on on-chip networks. The size of flit also indicates bandwidth of links. Buffer size indicates the number of buffers inside routers. Garnet models the pipeline router which consists of RC (route computing), VA (virtual channel allocation), SA (switch allocation), ST (switch transfer) and LT (link transfer) five stages. Virtual network is a group of VCs (virtual channels) which are used to transfer the same type of coherence messages. As seen from Table I, Garnet supports 5 virtual networks, each of which consists of 4 VCs

TABLE I. PARAMETERS OF GARNET'S FIXED-PIPELINE MODEL

Network configuration	Parameters
Flit size	16 bytes
Buffer size	4
Pipeline stage	5-stage
VCs per virtual network	4
Number of virtual network	5

Papers [5] [12] use synthetic batch traffics, such as uniform random (UR), to simulate the traffic of on-chip networks. However, these benchmarks do not represent real-life traffics. In this paper, we run multithreaded commercial benchmarks on multi-core simulator GEMS [13] to generate practical traffics and simulate these traffics on Garnet. GEMS leverages an existing full-system functional simulation infrastructure (Simics [14]) as the basis around which to build a set of timing simulator modules for modeling the timing of memory systems and microprocessors. Table II lists the configurations of the processor and cache/memory used in our experiment.

TABLE II. PROCESSOR AND CACHE/MEMORY PARAMETERS

Component	Parameters
Processor	UtraSPARCIII+
L1 I/D cache	64KB, 4-way, 3 cycles
L2 cache bank	2MB, 4-way, 12 cycles
Memory bank	1GB, 4KB/page, 158 cycles
distributed directory	52KB, 6 cycles

We create the network configuration file for 9-node triplet-based network via the user-defined network interface provided by GEMS. This configuration file records the topology and the link latencies (long links' latency in Fig.4 is set to 2 while short links' latency is set to 1), which are used by Garnet as network configuration. We also create the network configuration file for 16-node 2D-MESH and set all links' latency to 1.

We use work-related throughput [15] as the metric to evaluate the performance of group-caching and use Orion [16] to evaluate the energy consumption. Orion is a power model integrated with Garnet to evaluate network energy consumption using 100nm technology. The commercial benchmarks used in our experiment include Apache (a static web server benchmark based on SURGE [17] running on top of Apache web server), SPECJBB2005 (a java server benchmark), OLTP (a scaled down TPC-C benchmark used to capture the performance behavior of OLTP workloads [18]) and DSS (a scaled down TPC-H benchmark used to capture the performance behavior of DSS workloads [18]). Workload parameters for benchmarks are reported in Table III.

TABLE III. WORKLOADS PARAMETERS

Benchmark(transactions)	Environment
Apache(10000)	2000files,90 clients,0 think time
JBB(20000)	SPECjbb2005, 14 warehouses
OLTP(200)	IBM DB2, mbenchkit
DSS(100)	IBM DB2, mbenchkit

There are two cache coherence protocols used in our experiments. GEMS provides MESI_SCMP_bankdirectory protocol (referred as MESI below), which is widely used in banked and shared L2 design. We implement group-caching protocol via the user-defined interface of GEMS.

B. Evaluation

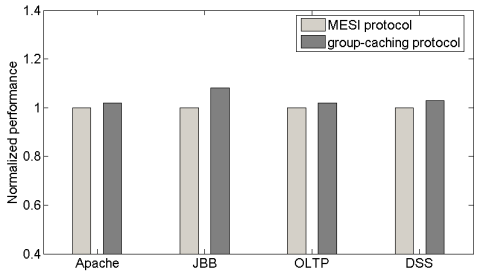

Figure 5. Performance comparisons between MESI and group-caching for 16-node 2D-MESH

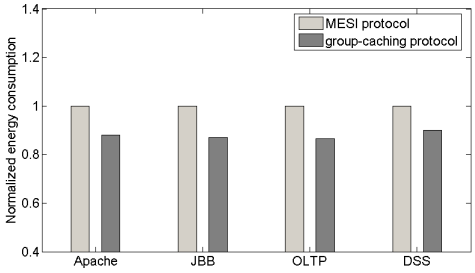

Figure 6. Energy consumption comparisons between MESI and group-caching for 16-node 2D-MESH

Work-related throughputs and energy consumption are both normalized to that of using MESI protocol. As shown in Fig.5 and Fig.6, group-caching increases the performance of 16-node 2D-MESH by 2%~8% on Apache, SPECJBB, OLTP and DSS benchmarks compared to banked and shared L2 design, with energy consumption reduced by 11%~13%. We should note that the performance increment of SPECJBB is significant than other benchmarks, which means SPECJBB is more sensitive to network latency.

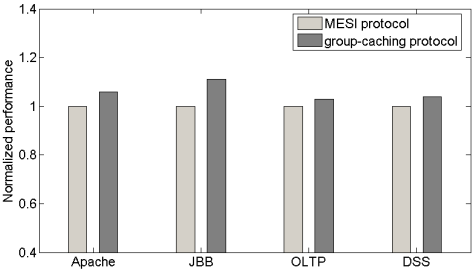

Figure 7. Performance comparisons between MESI and group-caching for 9-node triplet-based network

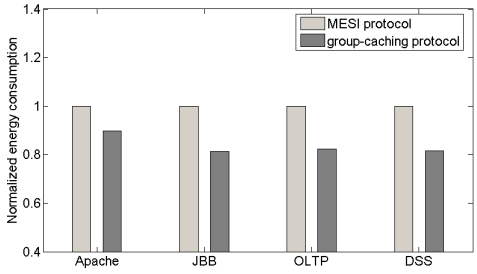

Figure 8. Energy consumption comparisons between MESI and group-caching for 9-node triplet-based network

As shown in Fig.7 and Fig.8, group-caching achieves more significant improvements on performance as well as energy consumption (performance is increased by 3%~11%, energy consumption is reduced by 13%~16%). We think the reason is the different communication overheads inside cache group: for triplet-based network, communications inside cache group only need one hop while communications inside cache group in 2D-MESH may need more than one hop.

IV. Conclusion and Future Work

In this paper, we propose the use of group-caching in NoC based multicore cache coherent systems. The principle of group-caching is to attract the frequently accessed data in local cache group so as to reduce remote on-chip references. In group-caching design, L2 banks are organized to form multiple on-chip cache groups. Each cache group behaves like a shared L2 cache while the cache coherence between cache groups is maintained by query message and invalidation messages. For applications with low data sharing between threads, because private data of a thread is loaded into local cache group which is close to the core running this thread, most accesses to private data are severed by local cache group and the hop count is reduced. For applications with high data sharing between threads, because multiple data copies are loaded into different cache groups corresponding to cores running these threads, most accesses to share data are also severed by local cache group and the hop count is also reduced. Experiment results show that for, when using 2D-MESH topology, group-caching increase the performance by 2%~8% compared with banked and shared L2 design, with network energy consumption reduced by 11%~13%. Group-caching can also be applied to other topologies. Experiment results show that group-caching helps 9-node triplet-based network to achieve more significant improvements on performance due to triplet's full-connected property.

Since the overheads of communications inside cache group play an important role in the performance of group-caching, our future work will focus on improving the efficiency of the intra-group communications. We plan to use the combination of crossbars and low-radix networks as the connection substrate. Taking 16-core CMP for an example, we use 4 low-latency crossbars to deliver the coherence messages inside cache group (assume that we divide 16 core into 4 cache groups) while using 16-node 2D-MESH to transfer the coherence messages between cache groups.

Acknowledgment

We sincerely thank Xu Qiang and Zhang Luping for their contributions on Solaris environment configuration.

References

[1] M.B. Taylor, J. Psota, A. Saraf, N. Shnidman, V. Strumpen, M. Frank, "Evaluation of the Raw microprocessor: an exposed-wire-delay architecture for ILP and streams" 31st Annual International Symposium on Computer Architecture, 2004. Proceedings. pp.2 – 13.

[2] P. Gratz, Kim Changkyu, K. Sankaralingam, H. Hanson, P. Shivakumar, S.W. Keckler, D. Burger, "On-Chip Interconnection Networks of the TRIPS Chip" Micro, IEEE Volume 27, Issue 5, pp.41 – 50, 2007.

[3] Y. Hoskote, S. Vangal, A.Singh, N. Borkar, S. Borkar, "A 5-GHz Mesh Interconnect for a Teraflops Processor"Micro, IEEE Volume 27, Issue 5, pp. 51 – 61, 2007.

[4] D. Wentzlaff, P. Griffin, H. Hoffmann, Bao Liewei, B. Edwards, C. Ramey, M. Mattina, Miao Chyi-Chang J.F. Brown, A. Agarwal, "On-Chip Interconnection Architecture of the Tile Processor" Micro, IEEE Volume 27, Issue 5, pp.15 – 31, 2007.

[5] J. Kim, J. Balfour, W.J. Dally, “Flattened Butterfly Topology for On-Chip Networks” 40th Annual IEEE/ACM International Symposium on Microarchitecture, 2007. Preceedings, pp.172 – 182.

[6] J. Balfour, W.J. Dally, “Design tradeoffs for tiled CMP on-chip networks” 20th Annual International Conference on Supercomputing, 2006. Preceedings, pp.187-198.

[7] V. Soteriou, Wang Hangsheng, L. Peh, “A Statistical Traffic Model for On-Chip Interconnection Networks” 14th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, 2006. Preceedings, pp.104 – 116.

[8] L.A. Barroso, K. Gharachorloo, R. McNamara, A. Nowatzyk, S. Qadeer, B. Sano, S. Smith, R. Stets, B. Verghese, “Piranha: a scalable architecture based on single-chip multiprocessing” 27th International Symposium on Computer Architecture, 2000. Proceedings, pp.282 – 293.

[9] P. Kongetira, K. Aingaran, “Niagara: a 32-way multithreaded Sparc processor” Micro, IEEE Volume 25, Issue 2, 2005. Preceedings, pp.21 – 29.

[10] Chang Jichuan, G.S. Sohi, “Cooperative Caching for Chip Multiprocessors” 33rd International Symposium on Computer Architecture, 2006. Preceedings, pp.264 – 276.

[11] http://www.princeton.edu/~niketa/publications/garnet-tech-report.pdf.

[12] P. Guerrier, A. Greiner, “A generic architecture for on-chip packet-switched interconnections” Design, Automation and Test in Europe Conference and Exhibition 2000. Proceedings, pp.250 – 256.

[13] Milo M.K. Martin, Daniel J. Sorin, Bradford M. Beckmann, Michael R. Marty, Min Xu, Alaa R. Alameldeen, Kevin E. Moore, Mark D. Hill, and David A. Wood, “Multifacet's General Execution-driven Multiprocessor Simulator (GEMS) Toolset” Computer Architecture News (CAN), September 2005.

[14] M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt, B. Werner, “Simics: A full system simulation platform Magnusson” Computer Volume 35, Issue 2, 2002. pp. 50 – 58.

[15] A. R. Alameldeen, M. M. K. Martin, C. J. Mauer, K. E. Moore, M. Xu, D. J. Sorin, M. D. Hill, and D. A. Wood. “Simulating a $2M commercial server on a $2K PC. IEEE Computer” 2003. pp.50–57.

[16] Wang Hang-Sheng, Zhu Xinping, Peh Li-Shiuan, S. Malik, “Orion: a power-performance simulator for interconnection networks” 35th Annual IEEE/ACM International Symposium on Microarchitecture, 2002.Proceedings, pp.294 – 305.

[17] P. Barford, Mark Crovella, “Generating representative Web workloads for network and server performance evaluation” 1998 ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems table of contents. Proceedings, pp.151 – 160.

[18] Minglong Shao, Anastassia Ailamaki, Babak Falsafi, “DBmbench: Fast and Accurate Database Workload Representation on Modern Microarchitecture” Conference of the Centre for Advanced Studies on Collaborative Research 2005.

[19] Cheng Liqun, N. Muralimanohar, K. Ramani, R. Balasubramonian, J.B. Carter, “Interconnect-Aware Coherence Protocols for Chip Multiprocessors” 33rd International Symposium on Computer Architecture, 2006. Preceedings, pp.339 – 351.

[20] Z. Chishti, M.D. Powell, T.N. Vijaykumar, “Optimizing replication, communication, and capacity allocation in CMPs” 32nd International Symposium on Computer Architecture, 2005. Proceedings, pp.357 – 368.

[21] M. Zhang and K. Asanovic “Victim replication: Maximizing capacity while hiding wire delay in tiled CMPs” 32nd International Symposium on Computer Architecture, 2005. Preceedings, pp.336–345.

[22] B. M. Beckmann and D. A. Wood. “Managing wire delay in large chip-multiprocessor caches”, International Symposium on Microarchitecture, 2004. Preceedings, pp.319–330.