

A Performance Evaluation of 2D-Mesh, Ring, and Crossbar Interconnects for Chip Multi-Processors

Jesús Camacho Villanueva¹, José Flich¹,
José Duato¹

¹Universidad Politécnica de Valencia
{jcamavil, jflich, jduato}@gap.upv.es

Hans Eberle², Nils Gura², Wlodek Olesinski²

²Sun Microsystems
{hans.eberle, nils.gura,
wlodek.olesinski}@sun.com

ABSTRACT

As the number of processing nodes on chip multi-processors (CMPs) keeps increasing, providing efficient communication with the on-chip interconnect becomes increasingly critical. With 32-core CMP designs on the drawing table of engineers, there is a demand for accurate simulation models that capture all the complexities and interactions of the different design layers including the application, operating system, cache hierarchy, coherency protocol, and other on-chip resources. These components cannot be modeled anymore in isolation as unpredicted performance anomalies may arise once all the system variables are taken into account.

In this paper, we present a simulation framework for CMP systems, focusing our attention on the on-chip network. We show preliminary results for the choice of key network parameters (topology, flit size) with respect to the behavior and performance of applications running on top of different network configurations. This paper tries to convey the need for an overall CMP system simulator as a way to accurately characterize the actual behavior of the on-chip network.

Categories and Subject Descriptors

B.4.4 Performance Analysis and Design Aids - Simulation

General Terms

Measurement, Performance, Design.

Keywords

Interconnects, on-chip networks, chip multi-processors, multiprocessor simulations.

1. INTRODUCTION

As the number of cores on a CMPs is increased, the on-chip core interconnect becomes an increasingly critical component. Here, we are mainly interested in a sub-class of CMP architectures that allow for tightly-coupled symmetric multiprocessing (SMP). Fig. 1 shows a typical organization of such a system, e.g. as exemplified by the UltraSPARC® T1 processor [4]. There are n cores each containing a processor and a first-level (L1) cache.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

NoCArc '09, December 12, 2009, New York City, New York, USA
Copyright © 2009 ACM and Sun Microsystems, Inc 978-1-60558-774-5... \$10.00

The cores are connected to a second-level (L2) cache through a crossbar interconnect¹. The L2 cache is shared and split into m banks. In the example of the UltraSPARC® T1 processor, $n = 8$ and $m = 4$. CMPs with 8 cores are shipping in systems now [4], CMPs with 16 cores have been announced, and CMPs with 32 cores are already on the drawing table. The use of a crossbar as shown in Fig. 1 is attractive as it simplifies the overall design of the system thanks to properties like fixed latency and the capability to broadcast packets. By offering a fixed latency between any core and any L2 bank, memory accesses can be easily pipelined and interleaved. Further, by being able to broadcast packets, operations such as cache invalidate operations can be efficiently implemented.

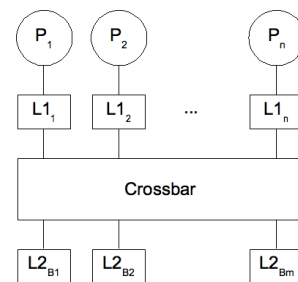


Fig. 1: Typical CMP system with Crossbar Interconnect.

As denser process technologies allow for integrating more and more cores on a single die, the question arises how well crossbar architectures that rely on a single monolithic switch scale. In [8], the authors report on their experience with synthesizing a crossbar switch in 65 nm technology and conclude that for switches with a radix greater than 30 the area overhead and degradation in clock frequency become intolerable.

Given the limited scalability of crossbar switches we have been examining alternative interconnect topologies that make use of smaller switches that can be distributed and integrated into the cores thus allowing for a tiled organization. In particular, we looked at other popular interconnect choices like 2D-meshes and rings. Both 2D-meshes and rings are advantageous when interconnecting larger numbers of nodes as the length of the link connecting two neighboring nodes does not depend on the number of nodes (which is not true for the crossbar). Further, it is

¹ This configuration with the processor cores and the private L1 cache on one side and the banked shared L2 cache on the other side of the crossbar is sometimes referred to as “dancehall” configuration.

straightforward to lay out 2D-meshes and rings in a plane unlike other interconnects such as tori or trees. Though the ring exhibits a longer average distance between two nodes than the mesh, it has its advantages such as an extremely simple routing protocol.

Many proposals have appeared that improve the performance of the on-chip interconnect. Proposals span from topology selection to router architecture, and many other topics have been covered like flow control, routing algorithms, arbiter design, etc. For a general overview refer to [2][3]. However, most of the proposals have evaluated the network in isolation by mostly using synthetic traffic patterns or traces.

In a recent publication [1], the effect of the interconnect on the performance of real applications executed on a detailed full-system simulator was shown. By using a detailed network simulator (GARNET) in a full-system simulator, the authors demonstrated that the network influenced the execution time of the application significantly, and, most importantly, that not considering the network component may lead to incorrect conclusions.

In this paper, we follow the same approach used in [1] by adding a detailed network simulator in GEMS, which stands for the Wisconsin Multifacet General Execution-driven Multiprocessor Simulator [6]. However, we extend the research by exploring the effects of key network parameters on the execution of real applications. In particular, we explore the effect of topology selection and network capacity. While the only topology considered in [1] is the mesh, we examine four interconnect types including an ideal fixed-latency network, a mesh, a ring, and a crossbar.

In summary, the contributions of the paper are twofold: First, we present a detailed network simulator and its integration into a full-system CMP simulator. Second, we explore the effects of two key network parameters on CMP performance.

The remainder of the paper is organized as follows. In Sections 2 and 3, we describe the simulation tool and model, respectively. In Section 4, we present the simulation results, and, in Section 5, we conclude the paper.

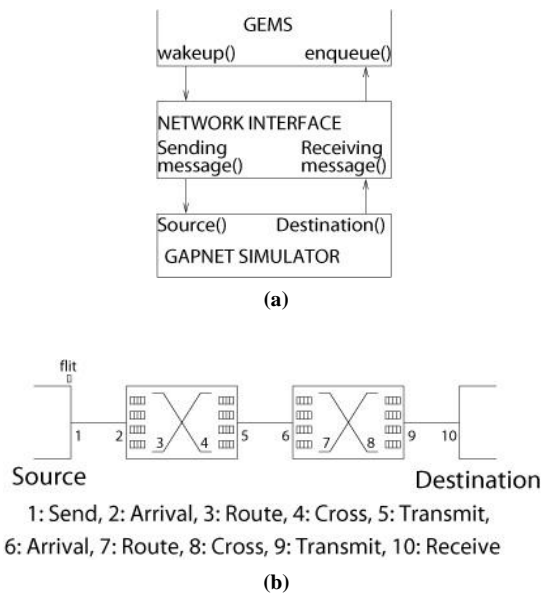


Fig. 2: Gap Network Simulator (a) and Simulation Events (b).

2. NETWORK SIMULATOR

Our work is based on Virtutech Simics [5] which is a full-system simulation framework, extended with the Wisconsin GEMS toolset [6]. In this section, we describe the network simulator as well as its connection to the GEMS system.

We have added our own network simulator, referred to as *gapnet*, to GEMS. GEMS already provides two network simulators, *simple* and *garnet*. *simple* is the original simulator (with no concept of network contention) which was later complemented with *garnet* allowing for a more accurate network model [1]. The *gapnet* simulator adds new functionality not present in *garnet*, e.g. support for collective communication, wormhole and virtual cut-through switching, virtual channels and virtual networks. The topologies that can be simulated include 2D-mesh, ring, torus, crossbar, and irregular topologies.

Gapnet is an event-driven simulator, initially providing a set of several events, some of them are shown in Fig 2.b. An event models a particular action within the network and triggers one or several new event with a parameterizable delay. The behavior of the network is modeled as a set of consecutive events, mimicking the advancement of flits through the network. The list of events is fully customizable and can be extended by new ones.

The GEMS system is also event-driven and provides a cycle-accurate simulation of events. Whenever GEMS advances the simulation time by a cycle, a function in *gapnet* is called that processes newly arrived events. In this way, *gapnet* stays fully synchronized with GEMS, acting as a slave module.

The link between GEMS and *gapnet* is implemented by means of an interface module (see Fig 2.a) with two entry points (in each direction) for both GEMS and *gapnet*. Whenever GEMS has a message ready to be sent, it calls the interface. The interface, then, processes the message and takes the required actions, for instance packetizing the message (if required), building a collective primitive (if required) and computing the address of the destination entity. The interface uses a memory space shared with *gapnet* in order to maintain state information (packet state). In the same way, whenever *gapnet* delivers a packet, the interface is called and, if required, the corresponding destination in GEMS (a blocked process) is woken up.

3. SIMULATION MODEL

In this section, we briefly describe the organization and characteristics of the simulated CMP system.

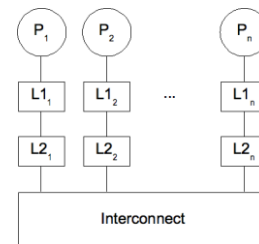


Fig. 3: Simulated CMP.

3.1 CMP Model

The simulated CMP system is shown in Fig. 3. It has 32 cores with each core containing a SPARC® CPU, an L1 cache and a L2

cache. The L1 cache is private and shared by both instructions and data. The L2 cache is shared though physically distributed. The chosen parameters for the caches are given in Table 1. With this organization, the CMP can be partitioned into tiles each containing a processor core, a private L1 cache, and a slice of the L2 cache. A tiled organization as outlined scales much better to higher core counts than the dancehall configuration described earlier. The reason is that the latter configuration makes it more and more difficult to provide fixed interconnect latencies between any core and any L2 location.

The cache coherency protocol is a two-level directory protocol and referred to as MESI_CMP_directory in the GEMS environment [6]. Caches are non-inclusive and blocking.

The cache parameters are derived from published numbers for CMPs such as the SPARC64® VIII processor [7] which has a 32 kB L1 data cache, 32 kB L1 instruction cache, and a 5 MB L2 cache – it is our assumption that a next-generation process technology will allow doubling the capacity of the cache memories.

3.2 Interconnect

Four interconnect types are being considered: an ideal interconnect, a crossbar, a 2D-mesh, and a bidirectional ring. The ideal interconnect is characterized by fixed delays for the transport of messages from any node to any other node; it is free of any contention and, thus, provides an unlimited amount of bandwidth. The crossbar is included in our analysis even though we know that its implementation is questionable given the size of CMP system considered; it does, however, provide a reference with a well-understood and widely used interconnect topology. The simulated 2D-mesh is organized as a 4 by 8 array. Routing is based on X-Y dimension order routing, that is, messages are first sent in X-direction and then in Y-direction. The ring topology is a bidirectional ring, where the direction is chosen based on the shortest distance between source and destination.

Table 1: Cache Parameters.

	L1 cache	L2 cache
Size	128 kB	8 MB
Associativity	8-way	16-way
Line Size	64 B	64 B
Hit Latency	3 cycles	6 cycles

The latency characteristics of the interconnect are given in Table 2. The link lengths are derived from our assumption that the cores are laid out with a pitch that corresponds to one clock cycle. Thus, for the 2D-mesh and the ring, the length of the link between two neighboring cores corresponds to the pitch of one cycle. For the crossbar, we assumed that the 32 cores are laid out as a tiled 2D-array. We assumed a link latency of 5 cycles which corresponds to the rounded Manhattan distance between a corner core and the center of the array (this calculation ignores the die area that the actual crossbar switch consumes).

Given the different radices of the switches used, we assumed one cycle latency for the smaller switches used for the ring and the 2D-mesh, and two cycles latency for the more complex 32-port crossbar switch. In case of the crossbar, the multi-cycle latencies assumed for the links and the switch translate into a pipelined design with the links and the switch broken up into several pipeline stages.

Table 2: Interconnect Parameters.

	Ideal	Crossbar	2D-Mesh	Ring
Link Latency [cycles]	-	5	1	1
Switch Delay [cycles]	1..128	2	1	1

A critical parameter of any interconnect is its transporting capacity. In our analysis, we are using different flit sizes as a way to change the capacity. The flit is the minimum amount of data information that can be flow-controlled through a link. The flit size is an important parameter of the network at two levels. At the architectural level, and assuming the use of wormhole switching, different flit sizes will lead to different contention levels within the network, and thus to different performance levels. At the design level, a larger flit size will lead to more expensive router designs that consume more area and power. Also, other issues such as wiring complexity and crosstalk will arise. In this paper, we are interested in the architectural side of the choice of flit size, in particular, its influence on performance.

3.3 Benchmarks

We have chosen a subset of the SPLASH-2 benchmarks consisting of FMM, LU, Radix, Radiosity, FFT, Barnes and Raytrace [9]. The characteristics of these benchmarks are listed in Table 3. The benchmarks were chosen such that we have a good mix of applications in terms of their usage of numerical types and synchronization primitives. Our characterization of the benchmarks’ use of synchronization operations is based on the characteristics given in [9]: “little” translates to an average time spent in synchronization operations of less than 10% and “moderate” corresponds to 30%.

All the simulation results have been obtained by averaging the results of 12 different simulation runs of which each used a different seed. The provided results are obtained only for the parallel part of the application.

Table 3: Benchmark Characteristics.

Benchmark	Instr (M)	FLOPS/Instr	Reads/Instr	Writes/Instr	Shared Reads/Instr	Shared Writes/Instr	Time spent in Synchron.
FMM	1250	0.34	0.18	0.03	0.17	0.02	little
LU	494	0.19	0.21	0.10	0.19	0.09	moderate
Radix	51	---	0.24	0.14	0.24	0.14	little
Radiosity	2832	---	0.18	0.10	0.09	0.00	moderate
FFT	35	0.17	0.11	0.09	0.11	0.09	little
Barnes	2003	0.12	0.20	0.16	0.11	0.05	little
Raytrace	829	---	0.25	0.10	0.19	0.03	little

4. PERFORMANCE ANALYSIS

We first evaluate the performance of a CMP system if an ideal network with a constant delay is used. Then, we look at the performance of the three interconnect topologies and show how it changes for different interconnect capacities.

4.1 Evaluation of an Ideal Network with a Constant Network Delay

We first analyze the performance of a CMP system that uses an ideal network. In particular, we model a network where all messages experience the same fixed latency, regardless of contention issues (thus, infinite network bandwidth is assumed). This analysis

shows the impact of the network speed on applications performance and also allows us to better understand the implications of the network subsystem on the CMP system in general and the coherency protocol in particular.

Table 4: Ideal Network. Results normalized to the 1c case.

App	netw delay	cycles	msgs	pages	L1D misses	instr
FMM	8c	1.14	1.02	1.01	1.02	1.11
	16c	1.30	1.02	1.02	1.03	1.23
	32c	1.61	1.03	1.05	1.05	1.46
	64c	2.71	1.09	1.10	1.10	2.32
	128c	7.09	1.22	1.27	1.27	6.51
LU	8c	1.29	1.10	1.09	1.13	1.26
	16c	1.68	1.19	1.18	1.27	1.63
	32c	2.64	1.33	1.28	1.40	2.37
	64c	4.50	1.46	1.39	1.56	4.08
	128c	29.02	2.82	2.55	2.75	28.32
Radix	8c	1.32	1.09	1.01	1.05	1.31
	16c	1.64	1.11	1.02	1.06	1.59
	32c	1.94	1.11	1.03	1.07	1.71
	64c	4.08	1.15	1.07	1.14	3.89
	128c	11.38	1.29	1.19	1.33	11.50
Radiosity	8c	0.99	1.01	1.52	1.02	1.02
	16c	2.30	1.74	2.41	1.64	2.40
	32c	1.19	1.10	3.39	1.08	1.15
	64c	1.48	1.18	5.37	1.21	1.36
	128c	2.49	1.47	4.30	1.64	2.60
FFT	8c	1.41	1.04	1.02	1.02	1.30
	16c	1.88	1.02	1.03	1.05	1.71
	32c	2.93	1.08	1.07	1.11	2.61
	64c	6.00	1.17	1.14	1.22	5.41
	128c	13.00	1.21	1.26	1.31	12.10
Barnes	8c	1.17	0.98	1.00	0.99	1.14
	16c	1.39	0.97	1.00	0.99	1.31
	32c	1.83	0.98	1.01	1.01	1.68
	64c	2.85	1.00	1.05	1.04	2.53
	128c	16.17	1.58	1.37	1.65	15.09
Raytrace	8c	2.71	1.16	1.36	1.36	2.40
	16c	4.20	0.97	1.03	1.04	3.98
	32c	7.20	0.90	0.98	1.00	7.67
	64c	13.29	0.86	0.81	0.86	16.84
	128c	23.72	0.80	0.80	0.87	29.88

Table 4 shows, for each application, several parameters: the number of cycles needed to run the application, the number of messages generated by the coherency protocol, the number of pages accessed by the application, the number of L1 data misses, and the number of instructions executed. These results have been obtained for a network with constant network delay (CND) ranging from one cycle to 128 cycles. Results are normalized to the 1c delay case (not shown). The first observation (column *cycles*) is that some applications are more insensitive to network latency than others. For FMM, LU, Radix, Radiosity, and Barnes, the increase in network delay leads to a moderate increase in execution time. However, for FFT and Raytrace, the increase is much more pronounced (most prominently for Raytrace). Another interesting observation is the impact of network delay on the flow of execution, in particular, on the behavior of the coherency protocol. For Radiosity, the number of page reclaims has increased more quickly than for the rest of applications. And in Raytrace, the L1 data misses become lower when increasing the latency. Notice that for LU the number of cache misses also increases sig-

nificantly. Illustrating a case of unpredictable interactions, we can see how in Radiosity the number of cache misses fluctuates as the network delay increases, with the 16c delay case being the singular case. The explanation of this behavior is that there are three times as many L1 data misses in supervisor mode than observed for other latencies. This increase is associated with synchronization operations generated by the application. This effect can also be seen in the increase of executed instructions due to synchronization primitives unsuccessfully testing locks.

4.2 Mesh vs. Ring

We first try to determine how execution time changes if we replace the crossbar with a 2D-mesh or a ring. Intuitively, we expect the 2D-mesh to enable higher performance as the average distance between two nodes is shorter in a 2D-mesh than in a ring. If we look at the execution time measured for applications FMM, LU, FFT, Radix, Raytrace, Radiosity, or Barnes in Fig. 4², this is exactly the case. In the best case, we see an improvement of 28% for *mesh16B* over *ring16B* (Fig. 4g). Note, that in this example, the execution time for the crossbar lies in-between the ones measured for the 2D-mesh and the ring. This finding is not necessarily true for all applications and configurations. For example, *mesh16B* and *ring16B* show very similar results for Radix and Radiosity – in the former case, the ring performs even slightly better than 2D-mesh.

4.3 Network Capacity

We looked at flit sizes of 4, 8, and 16 bytes³. Figs. 4a-g gives the execution times for each combination of flit size and interconnect topology. We expect to see a decrease in execution time with an increase in flit size. The simulation results, however, show varying degrees of this trend. For FMM, LU, FFT, Barnes, and Raytrace, a significant improvement in performance can be observed for all three topologies when increasing the flit size from 4 bytes to 8 bytes. There is, however, marginal improvement when the flit size is further increased to 16 bytes. This indicates, that the interconnects experience saturation for 4-byte flits but offer enough capacity for 8- and 16-byte flits to keep up with the cores.

When running applications Radix and Radiosity, it is only the ring and crossbar but not the mesh that show improved performance when the flit size is increased from 4 to 8 bytes. In all cases except for one case discussed in the next section, it is the mesh topology that achieves the best performance.

The network load seems to be low since the throughput (measured as the number of flits delivered by a node each cycle) is always less than 10% of the total capacity.

Looking at a flit size of 4B for which the interconnect shows some signs of saturation, the mesh topology allows to reduce execution time by 19% when compared to the ring, ranging from a minimum savings of 7% in Radiosity to a maximum savings of 32% in Radix. When compared to the crossbar, the average reduction is 26%, ranging from 12% in Radiosity to 42% in Raytrace.

We can further draw some interesting conclusions by comparing the performance of the different combinations given by topology and flit size. For example, we find that *ring16B* and *mesh4B*

² In this section, we consider an unsaturated interconnect which, as we will see later, is given by flit size 16B.

³ The capacity of the input buffers of the interconnect switches is always five flits.

provide similar performance, or that *ring16B* provides slightly better performance than *xbar8B*. These are interesting insights not seen when simulating the network only. We can see how a designer could decide, based on the provided results, to trade a 2D-mesh with a flit size of 4B with a ring with a flit size of 16B. Obviously, these decisions must be made by considering other additional input parameters relating to design constraints and the viability of implementation characteristics (wiring complexity, power consumption, and area requirements).

4.4 An Example of Unpredictable Performance

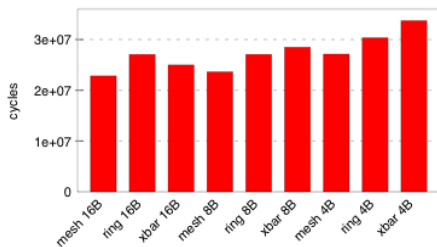
An interesting anomaly is observed when executing Radiosity for configuration *mesh8B*. As can be seen in Fig. 4d, the execution time for this case is about twice as much as for any other case. At closer inspection, we find that many more messages are generated, about twice more than for any other case (this data is not included in the paper due to space constraints).

Our first thought was that a higher L1 miss rate is responsible for the additional messages. Looking at the miss rates in Table 5, we do, however, find that the miss rate is not too different for this configuration than for the other ones; in fact, it is slightly lower for *mesh8B* than for *mesh4B* and *mesh16B*.

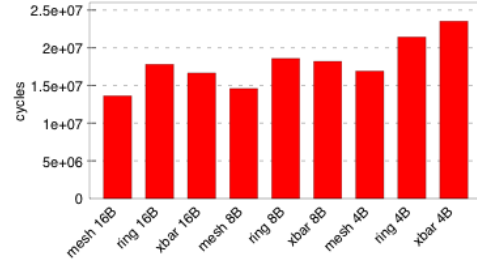
But analyzing the L1 misses further we did, however, find significant differences. Looking at Table 6, we notice that the absolute number of misses in user mode is about the same for all flit sizes, but that it differs significantly in supervisor mode, where the number of misses is about twice as much for *mesh8B* than for *mesh4B* and *mesh16B*. This observation leads to the conclusion that there is a significant additional overhead caused by synchronization operation for flit size 8B.

Our explanation for this phenomenon is that the particular latency parameters lead to timing characteristics of the interconnect that interact with the execution of the program and the resulting data access pattern in a way that results in more synchronization overhead, and, in turn, to an increase in generated messages.

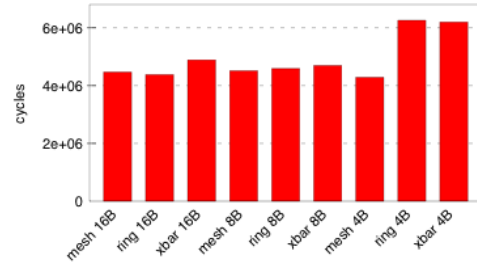
This anomaly should neither be treated as an error as we have explained, nor as the basis for selecting a given network topology or configuration. In fact, this anomaly should be treated as a demonstration that the system considered as a whole may have unpredictable performance variations for certain combinations of program behavior and network configurations. These effects are not captured adequately with simpler performance tools that isolate and analyze only parts of a system.



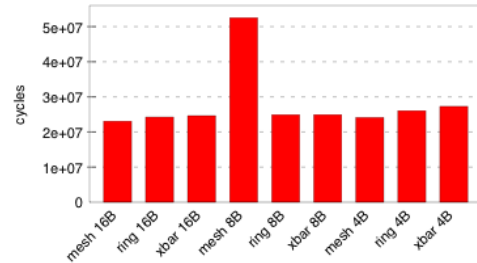
(a) FMM



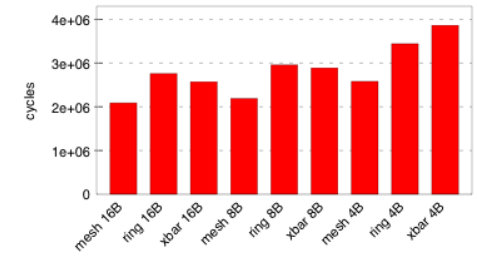
(b) LU



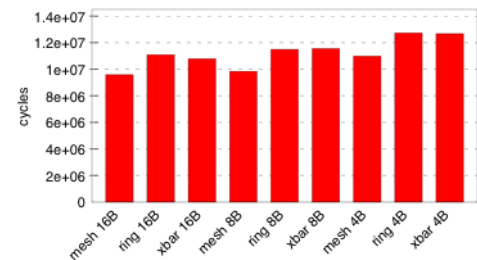
(c) Radix



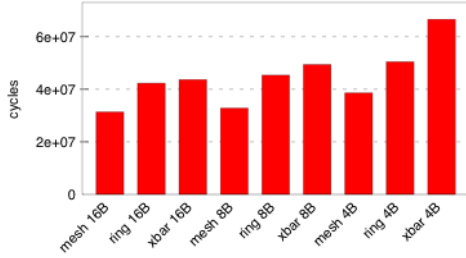
(d) Radiosity



(e) FFT



(f) Barnes



(g) Raytrace

Fig 4: Execution Times.

Table 5: L1 Miss Rates.

		mesh	ring	xbar
FMM	16B	0.17	0.16	0.17
	8B	0.17	0.16	0.16
	4B	0.16	0.15	0.15
LU	16B	0.19	0.17	0.19
	8B	0.20	0.17	0.18
	4B	0.19	0.16	0.15
Radix	16B	0.35	0.33	0.33
	8B	0.35	0.37	0.36
	4B	0.38	0.30	0.29
Radiosity	16B	0.09	0.09	0.09
	8B	0.07	0.09	0.09
	4B	0.09	0.09	0.09
FFT	16B	0.36	0.29	0.32
	8B	0.36	0.28	0.29
	4B	0.31	0.26	0.22
Barnes	16B	0.15	0.13	0.14
	8B	0.15	0.13	0.13
	4B	0.14	0.13	0.13
Raytrace	16B	0.84	0.52	0.38
	8B	0.82	0.53	0.32
	4B	0.68	0.38	0.20

Table 6: L1 Miss Types in Radiosity.

	16	8	4
User	537,136	541,517	539,187
Supervisor	198,764	480,737	201,876
Total	735,901	1,022,255	741,063

5. SUMMARY AND FUTURE WORK

In this paper, we are advocating the use of a complete CMP system simulator that considers all the main system components in a single framework, including the application, the operating system, the cache hierarchy, the coherence protocol, and the network.

Several noteworthy conclusions are obtained from the evaluation of application performance and behavior when changing the topology and the network capacity (by varying the flit size). In some cases, the network delay induces unpredicted application and coherence protocol changes leading to a significant increase of the overall execution time. One main conclusion is that the 2D-mesh reduces application execution time when compared with a ring and a crossbar. Also, we have seen that trading off the two variables considered (topology and flit size) may lead to network configurations with comparable performance. As an example,

similar performance is achieved for wide rings (16 bytes) and narrow meshes (4 bytes).

We plan to extend the work presented in this paper in the following areas:

- Multicast support: We would like to better understand the performance improvement by providing support for multicast operations as used by the cache coherence protocol.
- Cache hierarchy: Our current simulation environment is limited with two levels of caches only. We realize that the increasing number of transistors on a chip die is not only used for adding cores but also for adding cache levels. Thus, we are planning to extend our simulator with the capability of an L3 cache.
- Commercial workloads: We would like to extend our analysis to commercial applications that exhibit different behavior with respect to cache misses and the use of the coherence protocol.

Acknowledgment

This work was supported by the Spanish MEC and European Commission FEDER funds under grant CONSOLIDER-INGENIO 2010 CSD2006-00046, and by the Spanish MITC, under Grant TSI-020400-2009-64 (COMCAS project).

References

- [1] N. Agarwal et al, "GARNET: A Detailed On-Chip Network Model Inside a Full-System Simulator," in IEEE International Symposium on Performance Analysis of System and Software (ISPASS), Boston, Massachusetts, April 2009.
- [2] Luca Benini, Giovanni De Micheli, "Networks on Chips: A New SoC Paradigm," Computer, vol. 35, no. 1, Jan. 2002, pp. 70-78.
- [3] Jose Duato, Sudhakar Yalamanchilli and Lionel Ni, "Interconnection Networks - An Engineering Approach", IEEE Computer Society Press, 1997. 01
- [4] P. Kongetira, K. Aingaran, K. Olukotun, "Niagara: a 32-way multithreaded Sparc processor," IEEE Micro, vol. 25, iss. 2, March-April 2005, pp. 21- 29.
- [5] Peter S. Magnusson et al. Simics: A full system simulation platform. Computer, 35(2):50-58, 2002. IEEE Computer Society Press.
- [6] M. Martin, D. Sorin, B. Beckmann, M. Marty, M. Xu, A. Almadeen, K. Moore, M. Hill, D. Wood, "Multifacet's general execution-driven multiprocessor simulator (gems) toolset," in Computer Architecture News, September 2005.
- [7] T. Maruyama, "SPARC64 VIIIfx: Fujitsu's New Generation Octo Core Processor for PETE Scale Computing," Hot Chips 21, Stanford, August 23-25, 2009.
- [8] A. Pullini, F. Angiolini, S. Murali, D. Atienza, G. De Micheli, L. Benini, "Bringing NOCs to 65 nm," IEEE Micro, vol. 27, iss. 5, Sept./Oct. 2007, pp. 75-85.
- [9] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, A. Gupta, A., "The SPLASH-2 programs: characterization and methodological considerations," 22nd Annual Int. Symposium on Computer Architecture (ISCA), Italy, June 22 - 24, 1995. pp. 24-36.

All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. in the US and other countries. Products bearing SPARC trademarks are based upon an architecture developed by Sun Microsystems, Inc.