Sapphire: A Framework to Explore Power/Performance Implications of Tiled Architecture on Chip Multicore Platform

Aparna Mandke Keshavan Varadarajan Basavaraj Talwar
Bharadwaj Amrutur Y. N. Srikant

IISc-CSA-TR-2010-3
http://archive.csa.iisc.ernet.in/TR/2010/3/

Computer Science and Automation
Indian Institute of Science, India

June 2010

Sapphire: A Framework to Explore Power/Performance Implications of Tiled Architecture on Chip Multicore Platform

Aparna Mandke, Keshavan Varadarajan, Basavaraj Talwar, Bharadwaj Amrutur, Y. N. Srikant

Abstract

Sapphire is a multi-processor/multicore simulator where the memory hierarchy, interconnect (network on chip) and offchip DRAM are parametrized and can be configured to model various configurations. Sapphire addresses shortcomings in SESC by integrating it with Ruby from the GEMS framework. Sapphire also integrates Intacte, an interconnect power model. DRAMSim models off-chip DRAM in great details. This has been integrated with Ruby. Power consumed by DRAM is modelled using MICRON power model. Thus Sapphire allows users to explore power and performance implications of all main system components like processor, interconnect, cache hierarchy and offchip DRAM.

1 Introduction

Multi-core processors are becoming ever more popular. We are currently seeing multi-core processors connected through simple 1-D interconnections such as buses, rings etc. Newer architectures that employ 2-D interconnection structures like mesh based architectures will soon be available viz. Intel 80 tile architecture. These kind of architectures provide a large design space which an architect would want to explore better and efficient architectures. The design space includes (but not limited to) exploring performance optimization and energy efficiency in the context of different network topologies, network flit size, routing algorithms, router design, choice of each node (spanning from single compute elements to full processors), size of caches at various levels, co-scheduling algorithms for applications etc. The simulation framework for multi-cores must be endowed with sufficient configurability so as to enable exploration of some or all of these parameters for energy efficiency/performance optimization.

Multi-core simulators available today primarily fall under two categories, namely processor simulators and system simulators. Processor simulators model the processor pipeline in detail and are suitable for microarchitectural exploration. Simulators such as SESC[15], Opal[12] are processor simulators which are popular. These simulators cannot model the system-level impact of a certain change. System simulators such as Simics [11], M5 [4] simulate the entire system (including the processor, caches, memory hierarchy, I/O devices). Most system simulators allow booting of complete operating systems and perform end-to-end simulations. Such simulators are useful for modeling the system level impact of a macro architectural change or changes to the system software. But this makes these simulators very slow.

To make the simulator faster, various techniques have been applied like simulation of an interval from every phase of the program, instead of simulating the whole program[16], abstracting various components of the system architecture or selectively simulating on the native platform and on the cycle-accurate simultor. Yet another popular tool is CMP-$im [7]. Unlike other simulators, CMP-$im uses dynamic instrumentation to track certain events of interest. The execution happens on the host processor and instructions are added to the application binary to capture events of interest. While such a technique can be used to perform cache modeling, the scheme cannot be applied to explore diverse cache configurations like distributed caches [5], tiled caches [21] and molecular caches[18] and wide range of interconnect topologies from bus-based to point-to-point and Network-on-Chip based choices like torus and meshes.

2 Motivation

SESC [15] is an open-source processor simulator that is capable of modeling multi-cores or chip multiprocessors(CMPs), simultaneous multi-threading and symmetric multi-processors. The simulator models the MIPS processor instruction set. The simulator consists of a functional emulator which performs the functional simulation. The trace of instructions executed by the functional emulator are passed through the timing model. It models minute details of the processor pipeline and processor components. Such a decoupled implementation lends high efficiency to the SESC processor implementation. The simulator also models the memory hierarchy and the interconnect. However, these models cannot be easily extended to perform various architectural explorations with regard to caches and network on chip. Also, the interconnection implemented in SESC is not advanced and does not model 2-D interconnections. As shown subsequently, the network on chip is an important component which contributes to performance and energy efficiency. Since the source and sink of the data on the network are the various elements of the memory hierarchy, it is essential to model these accurately. As per [9], interconnects consume significant amount of power and area on the chip, compared to caches and cores on CMPs. Hence interconnects should be co-designed along with caches on CMPs. However, implementations of these components in SESC needs substantial improvement to support configurability to the extent available in Ruby, the memory hierarchy simulator[12].

The GEMS [12] simulation framework includes Opal, the processor simulator and Ruby, the memory hierarchy simulator. Opal simulates the SPARC instruction set and performs detailed pipelining modeling. Opal too uses a functional simulator for actual application execution and the timing is obtained from the pipelining. For functional simulation, it employs Simics [11]. The GEMS simulation framework can operate in two modes: Simics+Ruby and Simics+Opal+Ruby. The use of Simics helps GEMS to do system-level modeling while not trading off the accuracy of pipeline modeling. However, due to the use of Simics, this approach incurs the overhead of simulating the operating system and other I/O devices. Even though Simics provides an efficient implementation of all components, it increases simulation time. Also a user intending to evaluate the micro-architectural changes does not need the completeness of system-level modeling. Ruby simulates the memory hierarchy and interconnect in great details. It is highly configurable. Ruby allows specification of various coherence protocols, in terms of states, events and transitions, through a language called SLICC. This specification is transformed into a C++ implementation by the framework. This allows accurate modeling of the coherence protocol, which is an important contributor of network traffic. The network in GEMS can be modeled at various levels of details. Garnet [2], the network model included within GEMS can be configured to specify the exact interconnect topology, the link latency and bandwidth of specific links etc. It models various stages of network router pipeline, namely, route computation, virtual channel allocation, switch allocation and switch traversal. In Sapphire, we try to leverage the advantages of both simulation frameworks by integrating the SESC processor with Ruby, the memory and interconnect model.

3 Integrating SESC with Ruby

Ruby is designed to work independent of the Opal processor (or Simics). Due to this reason, Ruby has a well defined interface that specifies the list of functions exported by Ruby and the list of processor functions expected by Ruby. This clear interface definition integrating with Ruby is not an engineering intensive task. For integration, we created a `SescInterface` class which implemented all the relevant interfaces to interact with SESC. In SESC, a new memory object was created. The memory object, referred to as `RubyMemObj` forwards the requests to Ruby, while ensuring that two requests belonging to the same processor, do not refer to the same cache line (which is a requirement of Ruby). The detailed block diagram is shown in Figure 1.

As can be observed from Figure 1, an object of the `RubyMemObj` class is created for every cache (i.e. L1-I, L1-D). These objects enqueue the incoming requests that can be processed in this cycle. Other requests that cannot be processed are placed in a separate data structures. The Request Dispatcher (which is a callback function) deques as many requests as the number of ports from each processor and forwards the requests to Ruby. The request is received by the SescInterface object in Ruby. The request is forwarded to the appropriate Sequencer (one per processor) for subsequent processing by the caches. When the data is available, Ruby invokes the call back function for hit, which triggers the corresponding function in SESC.

Figure 2 shows the sequence of steps involved in making a request to Ruby. An `IMemRequest` object is created by the `FetchEngine` of SESC. This request is forwarded to the `RubyMemObj` object, which in turn checks whether Ruby can accept this request. If yes, the request is stored in the respective queue for this processor. The control is returned to the `FetchEngine`. When the clock cycle is incremented then the callback function to forward all requests from `RubyMemObj` is invoked. This issues the requests to Ruby, through `SescInterface`. The `SescInterface` forwards the request to the appropriate `Sequencer` object. The `Sequencer` object then forwards the request to the appropriate L1 Cache Controller within Ruby. The request flows through the memory hierarchy and when the data is available, the hit callback of SESC is invoked to inform SESC about the completion of the request. The `RubyMemObj` in turn invokes the `goUp` function to inform the Load/Store Queues that the data is now available.

This simulator now addresses all the shortcomings of SESC. The technique described in the above section can be used to attach any processor front end to the Ruby memory hierarchy.

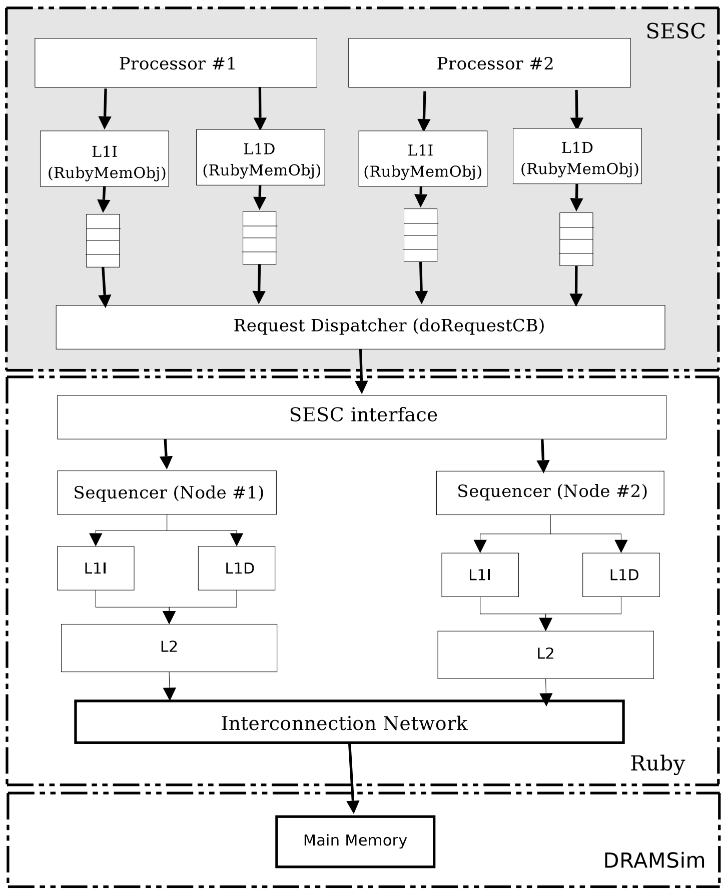

Figure 1: Detailed block diagram showing the integration of SESC with Ruby. The components shown within the box are new components that have been added for integration.

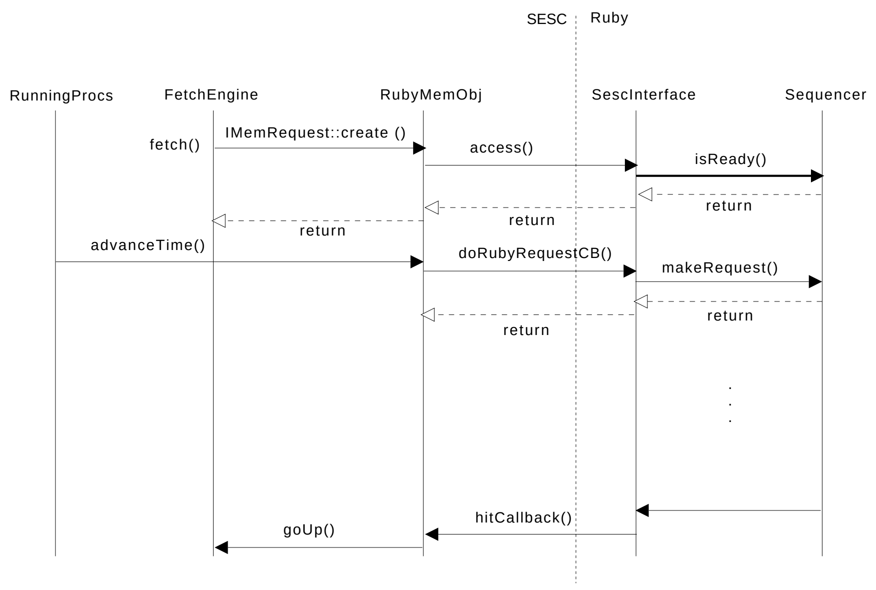

Figure 2: Interaction between various modules of SESC and Ruby for processing an instruction memory request. A similar sequence of calls are executed for a data request.

4 Modeling the Interconnect

While the simulator by itself exposes sufficient configurability to be able to model the interconnect, most architectural simulations use single cycle communication distance between two routers or between the router and end point (i.e. cache/memory in this case). With technology scaling, while the effective length of the wire decreases, the resistance of the wire increases on account of lower wire width [6]. Due to these reasons the interconnects need to be appropriately enhanced through increased wire pitch, use of repeaters, appropriate selection of repeater size, use of pipe stages etc. While many of these choices are made while transforming a design into silicon, current available high-level simulators do not capture the impact of these parameters during architectural simulation. These parameters determine power consumed and latency incurred by the network link. Our framework fills this gap by integrating Intacte [14], a low level tool to explore interconnect links.

Intacte [14] is an interconnect exploration tool which given the range of input frequencies, wire length and technology parameter, gives the least power configuration of the link by iterating over the following set of parameters: wire width, wire spacing, repeater size and spacing, number of repeaters, degree of pipelining, supply voltage and threshold voltage. These values are obtained using conservative estimates of activity and coupling factors. However, the tool can take these as input to estimate the best configuration.

In order to estimate the latency (in cycles) of a certain wire, we need to estimate the wire length. We estimate the interconnect length between router-router and router-end point by constructing the floor plan of each node on the network. Each network node contains a processor, L1 Cache Controller connected to the L1 instruction(L1-I) and L1 data cache(L1-D), L2 Cache Controller and L2 Cache, a router and an optional memory controller. The memory is placed off-chip and is not a part of the node. We henceforth refer to the network node as a tile. The floorplan of a typical tile is shown in Figure 3. The area of the processor is same as the size of a single core of Intel Nehalem [Internet]. The area occupied by the L2 cache is obtained using an approximate estimate of 7.5mm^2 per MB of cache, which is the density achieved on the AMD Shanghai processor [Internet]. This estimate is more conservative than the estimates obtained by CACTI [13]. The area of the router is estimated to be quite negligible at 32nm [8]. The L1 cache is of size 32KB whose area is very small and included in the processor area and hence is not depicted in the figure. ‘ The wire lengths were determined for

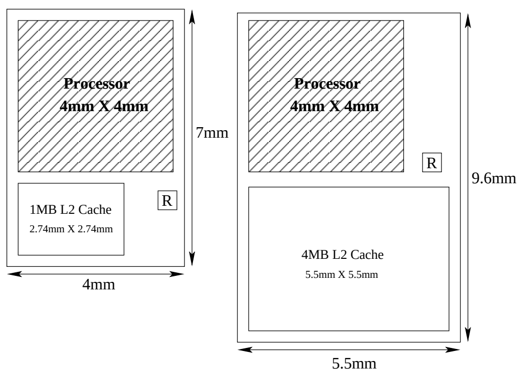

Figure 3: The floorplan of a tile with 1MB L2 cache and 4MB L2 cache. *R* refers to the router.

the floorplans in Figure 3 at a frequency of 3GHz and 32nm technology. This is tabulated in Tables 1 and 2. As is evident from the table, some of these latencies are quite high and hence cannot be ignored during architectural simulations. In order to account for these in simulation, we modify the network configuration file and input the specific latencies as obtained from Intacte. It can be observed from these tables that the latency increases with increase in cache size, due to the larger area occupied by the 4MB L2 cache.

Table 1: Tile Link Latencies and Power Estimations using Intacte for 1MB L2 cache; *R*: Router, *M*: Memory Controller, *L1*: L1 Cache Controller, *L2*: L2 Cache Controller.

Link Type	Length	PipeLineStages	Power
L1-R	1.26mm	2	1.083 mW
L2-R	4mm	8	3.4091 mW
R-R Horizontal	4mm	8	3.4091 mW
R-R Vertical	7mm	10	6.1998 mW
M-R	0.2mm	1	0.2545 mW

Table 2: Tile Link Latencies and Power Estimations using Intacte for 4MB L2 cache.

Link Type	Length	PipeLineStages	Power
L1-R	1.5mm	2	1.39 mW
L2-R	5.6mm	10	4.7437 mW
R-R Horizontal	5.5mm	10	4.6526 mW
R-R Vertical	9.6mm	19	8.14 mW
M-R	0.2mm	1	0.2545 mW

4.1 Modelling Power in Interconnect

Yet another aspect that needs to be considered in NoC based architectures is the power dissipated in the interconnect. This aspect is crucial for evaluating architectural solutions for energy efficiency, since interconnect power is no longer a negligible aspect [17]. In order to address this, we compute the link activity and coupling factors for all request and response messages on the network. One of the requirements for performing this analysis is the ability to pass correct memory data over the network. While, Ruby implements exact data transfers when simulated with Simics, the feature is not available otherwise.

In order to implement this, a detailed understanding of the SESC memory model and update procedures need to be understood. The instructions in SESC are obtained from the MIPS executable. However the data portions i.e. stack, global and heap memories are allocated by the simulator and then passed on to the running application. The simulator updates the memory upon receipt of memory writes. This memory is not accessible to Ruby. The request structure between SESC and Ruby is modified to pass the expected data along with the request (since SESC performs the write in functional simulation before it is simulated through the pipeline). The memory implementation in Ruby is modified to return this data (available along with request) instead of looking up its copy of the memory. The activity computation function appropriately accounts for the data, upon the return path and not along the forward path.

5 Modelling DRAM

Memory model of Ruby does not model DRAM latencies and power accurately. Hence, Sapphire includes a DRAM simulator, DRAMSim [19]. DRAMSim implements detailed timing models for variety of existing memories, including SDRAM, DDR, DDR2 etc. DRAMSim simulator is parameterized to support various row buffer management policies, address management policies and various layouts in terms of channels, ranks, rows and columns. This simulator uses MICRON power model [1] to estimate the power consumed in memory accesses.

6 Framework Evaluation

6.1 Simulation Procedure

Flow chart in Fig. 4 shows the experimental procedure. The first step of the procedure is to determine the optimal floorplan based on the processor, cache and router area. This is then used to compute the link lengths. These link lengths are then passed through Intacte to obtain the link parameters such as the number of pipe stages. To obtain these parameters, we assume conservative values of activity and coupling factor. Link parameters giving minimum power are used for the assumed activity and coupling factor. The number of pipe stages determines the latency of the link in cycles. This information is used in the network configuration file. The simulation is performed on Sapphire. Sapphire determines the activity and coupling factors of all the links. These are then passed through Intacte to determine the power dissipated in the NoC. DRAMSim model estimates power consumed in off-chip memory accesses. Cache power is estimated using CACTI power model [13].

6.2 Experimental Setup

In this section we present a results for a few experiments that were performed on *Sapphire*. Table 3 gives applications used during the experiments. We use 32KB of instruction and data L1 cache. L2 cache is shared distributed and each tile contains 1MB of single bank L2 cache. The various applications which were simulated are shown in Table 3. Results for few of them are

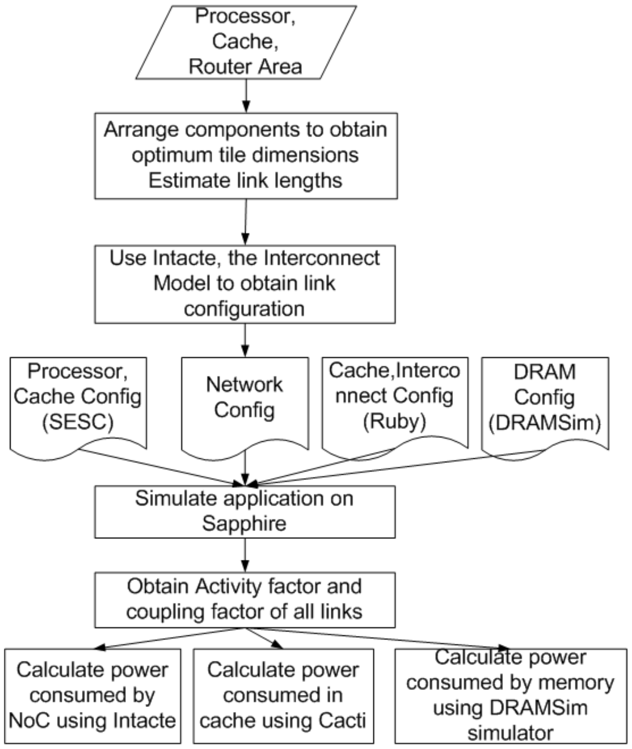
Processor, Cache, Router Area
Arrange components to obtain optimum tile dimensions Estimate link lengths
Use Intacte, the Interconnect Model to obtain link configuration
Processor, Cache Config (SESC)
Network Config
Cache,Interconnect Config (Ruby)
DRAM Config (DRAMSim)
Simulate application on Sapphire
Obtain Activity factor and coupling factor of all links
Calculate power consumed by NoC using Intacte
Calculate power consumed in cache using Cacti
Calculate power consumed by memory using DRAMSim simulator

Figure 4: **The figure shows the steps to be followed for performing detailed power-performance analysis on Sapphire.**

Multithreaded - PARSEC Benchmark[3]

H.264 Encoder	SimLarge i/p, Media Domain
swaptions	SimLarge i/p, Financial Domain
blackscholes	SimLarge i/p, Financial Domain
fluidanimate	SimMedium i/p, Animation

Splash2 Benchmark[20]

Ocean (continuous)	512x512 grid points, High Performance Computing

FFT
LU(Continuous)
LU(Non-Continuous)
RADIX
	64K Complex Data points
	256x256 Matrix
	256x256 Matrix
	Sorts 256K keys

Combination used for Multiprogramming Workload

Splash2 Benchmark[20]

LU(Non Continuous)	256x256 Matrix Factorization,B=16
RADIX	radix sort on 512K keys

Alpbench Benchmark [10]

MPGEnc(Enc)	Encodes 15 Frames of size 640x336
MPGDec(Dec)	Decodes 15 Frames of size 640x336

Multiprogramming Workload

ENC-LU Dec-RADIX	8 threads from each app., executed 1B instructions
Dec-LU	executed till completion

Table 3: **Applications used for experimentation**

presented here. The link lengths for the experiments are as shown in Table 1.

6.3 Results

The plot in Figure 5 shows the interconnect power (excluding the router power). To verify correctness of these results, we compare these with the link utilization reported by Ruby. This is shown in Figure 6. The link power shows a very high degree of correlation with the link utilization. The link utilization measures the temporal activity of the links. Assuming that the bit level correlation between successive packets follows a statistical distribution (with a mean and variance), it is easy to see that the link utilization and activity factor have to be correlated. However the value of the mean and variance of this distribution are different for different applications, therefore the link utilization cannot be directly utilized.

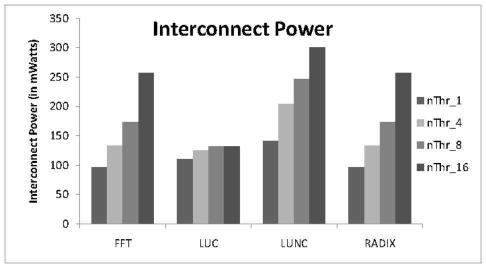

Figure 5: Graph shows variation in interconnect power consumed by various applications as the number of threads used for execution is varied.

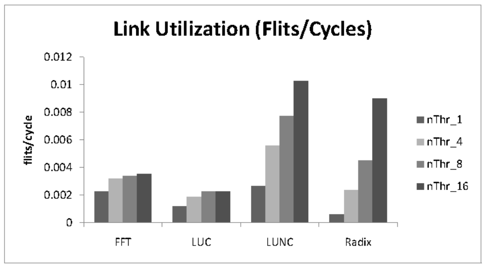

Figure 6: Graph shows variation in average link utilization for various applications as the number of threads used for execution is varied.

Figure 7 shows the time spent in transit by all the communication packets. While several of these communication overlap with each other and communication, this measure gives the total time it would have taken had the communication been done serially (say on a bus based architecture). As is evident from the graph, as the number of threads increases the communication drops, since each thread needs to perform a smaller piece of the task, while the cache size remains fixed at 1MB i.e. more resources are now available to perform the same amount of task. In such a scenario the communication is bound to reduce. After reaching a certain minima, the communication will begin to increase since the synchronization messages need to be sent higher number of entities.

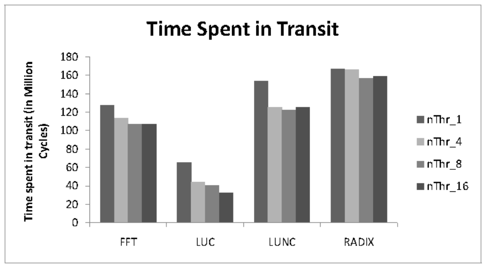

Figure 7: Graph shows variation in time spent in transit in term of clock cycles for various applications as the number of threads used for execution is varied.

7 Conclusion

Sapphire integrates configurable processor model using SESC with cache and 2-D interconnect model from Ruby and offchip DRAM model using DRAMSim simulators. Low level parameters of Interconnect link can be configured using Intacte. Power consumed by various components like cache, interconnect and DRAM can be estimated using CACTI, Intacte and DRAMSim.

References

[1] Micron DRAM power data sheet.

[2] N. Agarwal, L.-S. Peh, and N. Jha. Garnet: A detailed interconnection network model inside a full-system simulation framework. Technical Report CE-P08-001, Princeton University, 2008.

[3] C. Bienia and et. al. The PARSEC benchmark suite: Characterization and architectural implications. In *Conf. on Parallel Architectures and Compilation Techniques*, 2008.

[4] N. L. Binkert, R. G. Dreslinski, L. R. Hsu, K. T. Lim, A. G. Saidi, and S. K. Reinhardt. The M5 simulator: Modeling networked systems. *IEEE Micro*, 26(4):52–60, 2006.

[5] Z. Chishti, M. D. Powell, and T. N. Vijaykumar. Distance associativity for high-performance energy-efficient non-uniform cache architectures. In *MICRO 36: Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture*, page 55, Washington, DC, USA, 2003. IEEE Computer Society.

[6] R. Ho, K. W. Mai, S. Member, and M. A. Horowitz. The future of wires. In *Proceedings of the IEEE*, pages 490–504, 2001.

[7] A. Jaleel, R. S. Cohn, C. keung Luk, and B. Jacob. Cmp$im: A binary instrumentation approach to modeling memory behavior of workloads on cmps. Technical report, 2006.

[8] N. Joseph, C. R. Reddy, K. Varadarajan, M. Alle, A. Fell, S. K. Nandy, and R. Narayan. RECONNECT: A NoC for polymorphic ASICs using a Low Overhead Single Cycle Router. In *ASAP '08: Proceedings of the 19th IEEE International Conference on Application specific Systems, Architectures and Processors*, July 2008.

[9] R. Kumar, V. Zyuban, and D. M. Tullsen. Interconnections in multi-core architectures: Understanding mechanisms, overheads and scaling. In *Proc. of, Computer Architecture. ISCA '05. 32nd International Symposium on*, pages 408–418, 2005.

[10] M.-L. Li and et. al. The Alpbench benchmark suite for complex multimedia applications. In *Int. Symp. on Workload Characterization*, 2005.

[11] P. S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt, B. Werner, and B. Werner. Simics: A full system simulation platform. *Computer*, 35(2):50–58, 2002.

[12] M. M. K. Martin, D. J. Sorin, B. M. Beckmann, M. R. Marty, M. Xu, A. R. Alameldeen, K. E. Moore, M. D. Hill, and D. A. Wood. Multifacet's general execution-driven multiprocessor simulator GEMS toolset. *SIGARCH Comput. Archit. News*, 33(4):92–99, 2005.

[13] N. Muralimanohar, R. Balasubramonian, and N. P. Jouppi. CACTI 6.0: A tool to model large caches. 2009. http://www.hpl.hp.com/techreports/2009/HPL-2009-85.html.

[14] R. Nagpal, M. Arvind, Y. N. Srikanth, and B. Amrutur. Intacte: Tool for interconnect modelling. In *Proc. of 2007 Intl Conf. on Compilers, Architecture and Synthesis for Embedded Systems(CASES 2007)*, pages 238–247, 2007.

[15] J. Renau, B. Fraguela, J. Tuck, W. Liu, M. Prvulovic, L. Ceze, S. Sarangi, P. Sack, K. Strauss, and P. Montesinos. SESC simulator. January 2005. http://sesc.sourceforge.net.

[16] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder. Automatically characterizing large scale program behavior. In *ASPLOS-X: Proceedings of the 10th international conference on Architectural support for programming languages and operating systems*, pages 45–57, New York, NY, USA, 2002. ACM.

[17] S. Vangal, J. Howard, G. Ruhl, S. Dighe, H. Wilson, J. Tschanz, D. Finan, A. Singh, T. Jacob, S. Jain, V. Erraguntla, C. Roberts, Y. Hoskote, N. Borkar, and S. Borkar. An 80-tile sub-100-w teraflops processor in 65-nm cmos. *IEEE Journal of Solid-State Circuits*, 43:29–41, Jan. 2008.

[18] K. Varadarajan, S. Nandy, V. Sharda, B. Amrutur, R. Iyer, S. Makineni, and D. Newell. Molecular cache: A caching structure for dynamic creation of application-specific heterogeneous cache regions. In *MICRO 39: Proceedings of the 39th annual IEEE/ACM International Symposium on Microarchitecture*, 2006.

[19] D. Wang, B. Ganesh, N. Tuaycharoen, K. Baynes, A. Jaleel, and B. Jacob. DRAMsim: a memory system simulator. volume 33, pages 100–107, New York, NY, USA, 2005. ACM. http://doi.acm.org/10.1145/1105734.1105748.

[20] S. C. Woo and et. al. The SPLASH-2 programs: characterization and methodological considerations. In *22nd Int. Symp. on Computer architecture*, 1995.

[21] M. Zhang and K. Asanovic. Victim replication: Maximizing capacity while hiding wire delay in tiled chip multiprocessors. In *ISCA '05: Proceedings of the 32nd annual international symposium on Computer Architecture*, pages 336–345, Washington, DC, USA, 2005. IEEE Computer Society.