CellSim: A Validated Modular Heterogeneous Multiprocessor Simulator

Felipe Cabarcas[1], Alejandro Rico[1], David Rodenas[1,2], Xavier Martorell[1,2], Alex Ramirez[1,2], Eduard Ayguade[1,2]

[1]Departament d'Arquitectura de Computadors
Universitat Politècnica de Catalunya
{cabarcas, arico, xavim, aramirez}@ac.upc.edu

[2]Barcelona Supercomputing Center –
Centro Nacional de Supercomputación
{david.rodenas, eduard.ayguade}@bsc.es

Abstract

As the number of transistors on a chip continues increasing the power consumption has become the most important constraint in processors design. Therefore, to increase performance, computer architects have decided to use multiprocessors. Moreover, recent studies have shown that heterogeneous chip multiprocessors have greater potential than homogeneous ones.

We have built a modular simulator for heterogeneous multiprocessors that can be configure to model IBM's Cell Processor. The simulator has been validated against the real machine to be used as a research tool.

1 Introduction

There is an agreement between the industry and the research community about the importance of Heterogeneous Chip Multiprocessors (HCMPs) in the future. On the one hand the computing industry led by IBM, Toshiba and Sony has developed the Cell processor [10], Intel has recently announced the Larrabee project [2], and the embedded industry has been developing heterogeneous multiprocessors for several years, like the TI's OMAP [7], or the Nexperia by NXP. On the other hand, the research community [3, 16] has shown that the future multiprocessors should be heterogeneous.

However, the success of HCMPs depends on the ability of software designers and computer architects to develop strategies to extract parallelism from applications and to use them to effectively drive the available hardware [3]. Hence, it will be required that simulators not only model application traces, or statistics, but also the dynamic behavior of programs. Researchers, then, can find bottlenecks and strategies to exploit more parallelism.

As the number of processors and the complexity of the systems grow, researchers will require a modular simulation infrastructure for their work. They will then be able to build simulators by connecting independent modules as if they were Lego bricks. A modular infrastructure gives researchers the ability to create and modify components without the worry of generating unexpected behavior on other modules. In order to construct a modular infrastructure, it is convenient to create a common interface for the communication of modules. Therefore, the granularity, or smallest size of modules, is an important characteristic of the infrastructure since it would determine the common interface characteristics and the simulation speed. Therefore, we have decided to established the granularity at the level of processor units, memories and interconnection networks.

A modular simulator is not enough to obtain a flexible and efficacious simulator, it is also important to have it validated [5, 9]. This would allow researchers to compare the simulation results against a known hardware, in addition to reduce the possibility of hiding bugs

and bottlenecks with the proposed improvements of the architecture.

This paper presents a validated Cell Broadband Engine [12] simulator (CellSim). CellSim is a modular simulator that not only simulating the Cell Processor but also a wide range heterogeneous multiprocessor architectures. In section 2 we survey some of the existing simulators. Section 3 contains the description of CellSim. The validation process we have done for CellSim is exposed in section 4. Finally section 5 contains our conclusions.

2 Related Work

There exist a vast collection of systems to simulate heterogeneous chip multiprocessors. The differences between them depend on the objectives of researchers. However, to the best of our knowledge, there are not simulators with all the necessary characteristics to develop detailed HCMP research.

The existing simulators range from architecture detailed prototype build on FPGAs like the RAMP [20] project —where the hardware infrastructure and program development are considerable—, to program execution cost models [11] —which do not require to program or develop software, just change parameters— . Nonetheless, the majority of research groups have developed heterogeneous multiprocessor simulators based on uniprocessors simulators, like Kumar et al. [13] who have modified SMTSIM [19] for their work (and therefore inherits SMTSIM's non-modular properties), or GEMS [15] that uses Simics [14] to create a modular simulator, which is highly tied to the implemented ISA. SimFlex [21] also adds to Simics multiprocessor, microarchitecture and checkpoint simulation capabilities.

There are other simulators like SimOS [18] which lacks the modularity we believe it's necessary for heterogeneous multiprocessor research. MESH [17] simulates the performance of heterogeneous chip multiprocessors applications requiring the user to instrument the applications with some information for the simulator.

Mambo [6] is IBM's Cell Broadband Engine Simulator. It does not provide the sufficient configuration flexibility for architecture research, but we have used it for Cellsim SPU's functional validation.

The M5 simulator [4] presents most of the characteristics we have presented in this paper, however their focus is modeling networked systems, while ours is computer architecture.

3 CellSim: a validated modular heterogeneous multiprocessor simulator

CellSim is a modular simulator for heterogeneous multiprocessors. Its modularity is based on the fact that it is composed by modules, which are connected among them through signals. We have used UNISIM [1] as the supporting infrastructure for module description and connection handling. UNISIM requires that modules are C++ classes with methods sensitive to the modules communication signals, and it is UNISIM who takes care of calling the methods and synchronize the modules. The UNISIM version we have used requires a clock signal, common to all modules, through which the simulator executes the functionality of modules each cycle.

We have implemented the modules corresponding to the current version of the Cell Broadband Engine Architecture, the Cell Processor. Using these modules we have configured CellSim in order to be able to validate it against the real processor. We give a brief introduction to the Cell Processor in the first part of this section and then we present the structure of CellSim and its modules.

3.1 The Cell Processor

The Cell Broadband Engine Architecture (CBEA) is a joint initiative of Sony, Toshiba and IBM. The first hardware implementation of the CBEA is the Cell Processor. Thanks to the fact that it is a real hardware implementation and it is being used as the main processor of a popular product, the PlayStation 3, it has a complete software ecosystem. That means that we have had an available compiler,

new applications and, even, existing applications that are being rewritten specifically to benefit the Cell Processor features. This gave us the necessary tools to develop CellSim using existing applications, test it comparing the execution results and validate it against the real hardware.

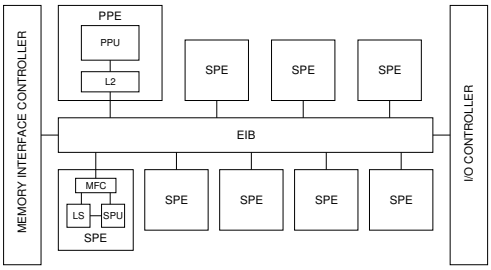


Figure 1: Cell Processor block diagram.

As shown in Figure 1, this processor is composed by one Power Processor Element (PPE), and eight Synergistic Processor Elements (SPE), connected through an Element Interconnection Bus (EIB), that also connects them with the memory and the I/O controllers. The main component of the PPE is a dual-threaded, dual-issue, 64-bit PowerPc Processor Unit (PPU). Each SPE is composed by three elements: a Synergistic Processor Unit (SPU), a Local Store (LS) and a Memory Flow Controller (MFC).

The PPU is responsible to run the operating system, and acts as the master of the system. The SPU has a different ISA, completely SIMD. The dataflow path is 128-bit wide and the register file contains 128 general-purpose registers of 128-bits each one (that can be accessed as 16 bytes, 8 half-words, 4 words, 2 double-words or 1 quad-word).

The memory-mapped LS provides 256 Kbytes for both instructions and data. Each SPU has only direct access to its own LS, to communicate with other elements outside the SPE, it has to perform a direct memory access (DMA) transaction through its MFC.

The MFC is fundamentally a DMA controller. Each MFC contains memory mapped control registers that allow the PPE, and other devices, to control the state of the SPU, configure DMA operations, etc.

Table 1: CellSim configuration parameters.

	Parameters	Cell	CS
Gen.	Number of PPEs	1	1
	Number of SPEs	8	1-8
PPE			
PPU	Issue bandwidth	2xth	1
Cache	Number of lines	512	512
	Line size	128	128
	Number of ways	8	8
SPE			
SPU	Issue bandwidth	2	2
LS	Size (KB)	256	256
	Latency (cycles)	6	6
	Number of ports	1	3
MFC	DMA cmd queue size	16	16
	DMA cmd processing delay (cycles)	...	30
EIB	# of rings/buses	4	3-4-5
	BW (B/bus cycle)	8	8
	# outstanding transfers/node	16	16

3.2 CellSim Structure

The modules of the simulator, as it is shown in Figure 2, represent each hardware component in the Cell Processor in Figure 1. The modules that compose the simulator are the following: PPU, Cache, SPU, MFC, Memory and Interconnection Network. It is possible to connect a PPE, 8 SPEs, and a Memory module to an Interconnection Network to obtain the same configuration with respect to the Cell Processor. However, it is also possible to connect more PPEs and SPEs keeping the limits due to the Interconnection Network.

In order to create a common interface for the simulator, we have developed the *MemoryAccess* class, which is the information packet that is transmitted between modules. A MemoryAccess object includes the type of access (*Load* or *Store*), the target and source addresses, the number of bytes and, if it is a store, the data. This common interface combined with the fact that each module has a physical page range assigned, provides the necessary flexibility to have a modular simulator.

Memory. The Memory module is used both as main memory and as LSs. It does not implement any specific internal distribution of memory banks, so sequential accesses

can be overlapped. However, the latency to access and the number of ports can be configured. Due to this fact the Memory module can be used also in the SPE to behave as the LS, which have several ports connected to both the SPU and the MFC.

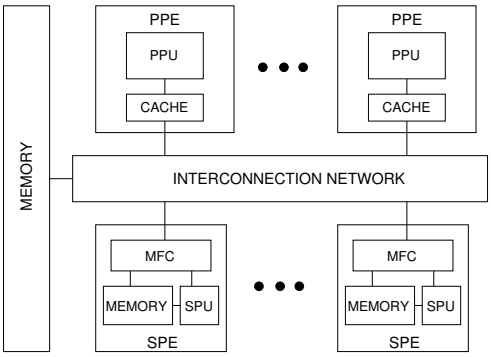


Figure 2: Simulator block diagram

PPE. The PPE which is formed by two modules: PPU and Cache. The PPU is a PowerPC 405 32-bit architecture compliant functional simulator, that executes a number of instructions per simulator cycle, and the instruction fetch and load/store instructions will be requested through the connection to the Cache.

The PPU module is an instruction set simulator (ISS) with a parametrizable issue bandwith and single threaded. Since the PPE is not a full system simulator, the PPU is responsible for emulating the operating system services: loading and execution of elf files, system calls, start and stop SPEs, mapping logical addresses onto physical ones, etc.

SPE. The SPE is composed of three modules: SPU, Memory and MFC. The SPU is a functional simulator, that executes a number of instructions per simulator cycle. It has two connections to the Local Store, one for instruction fetch and the other for Loads and Stores. Channel requests are services through one of the connections with the MFC, while the second connection is used to report the state of the SPU and to start and stop it when requested by the PPU using the memory map registers. The Local Store service the loads and stores requests of the SPU and the MFC. All the DMA command related services are handled by the MFC, it also contains the memory mapped registers of the SPEs.

Interconnection Network. The EIB of the Cell processor is a very efficient interconnection network, however it is not scalable. Therefore we have decided to implement the Interconnection Network (IN) module with a K-Bus topology. Although it is significantly different than the EIB, we have demonstrated that it is flexible enough to be configured and simulate a performance close to the Cell Processor (Section 4).

Modules connected to the IN are mapped in a global address space and have an input and an output port connected to the IN to send and receive MemoryAccess packets. These packets are routed using the memory map ranges (Table 2) for each module, which are related to the corresponding input and output ports. This configuration allows to connect one memory (up to 1 GB size) and an arbitrary number of PPEs (up to 4096) and SPEs (up to 112) to the IN.

Table 2: Interconnection Network routing table. Physical page range for each module and corresponding IN port. Physical page size: 64 kB.

Physical page range	Module	IN Port
[0x0000 - 0x4000)	Memory	0
[0x4000 - 0x4100)	LS0	NUM_PPEs+1
[0x4100 - 0x4200)	LS1	NUM_PPEs+2
...	...	...
[0xB000 - 0xB001)	PPE0	1
[0xB001 - 0xB002)	PPE1	2
...	...	...
[0xC000 - 0xD001)	MFC0	NUM_PPEs+1
[0xC001 - 0xD002)	MFC1	NUM_PPEs+2
...	...	...

This routing table depends on the architecture is wanting to be simulated. If one wants to test an architecture configuration not possible with the current memory ranges, these can be parametrized to assign different ports in the IN module to the routing table.

4 Simulator Validation

4.1 Functional Validation

Memory module. The functional validation of the Memory modules was performed using a stream of legal memory accesses. We verified the final state of the module with the correct data. This simulation was done with a specific configuration of the simulator, where it was composed by a module, which generates the memory accesses stream, directly connected to the memory module.

PPE module. In order to validate the PPU functionality, we implemented a special running mode. In this mode the PPU dumps its full state after the completion of each instruction. This generates a trace of all executed instructions and write it to a file. We have also created a *gdb* script that executes step by step an application and dumps the processor state with the same format than the PPU. The simulator configuration for the PPU functional evaluation consists of one PPU module connected directly to one Memory module. We have run the same application in CellSim with the mentioned configuration and in a PowerPC physical machine using the *gdb* script. We have verified that both files matches completely except for low level operating system responses, like the PID number. This evaluation process have been performed with a large number of micro benchmarks in order to test the implemented instruction set and system calls.

The functional validation of the Cache module has been done using the same methodology used for the Memory module, but including a Cache between the Memory module and the one that generates the memory accesses stream.

SPU module. The SPU functional validation was carried out using a set of programs composed by: (a) own custom programs and (b) some SPU tests extracted from the IBM SDK. Besides, the applications used for the MFC module functional validation and for the timing validation have also helped for validating the SPU module.

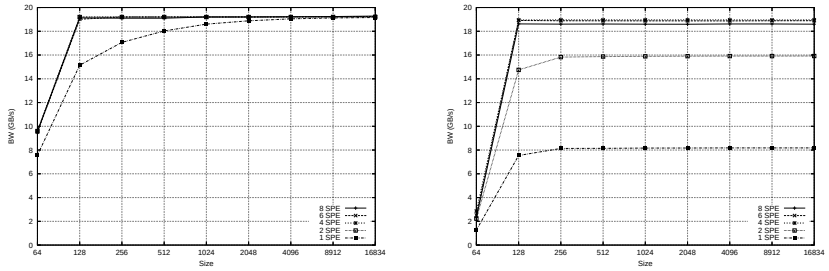
The first step was checking that the output of the programs run in CellSim were the same as the execution in the Cell Processor. Provided that, we performed some extra debugging, instruction per instruction, comparing the SPU module's state (registers and LS values) with the data provided by the IBM Cell Simulator executing the same applications.

MFC module. The functionality of the MFC module was validated using also own and SDK programs both exploiting specific functionalities such as channels, MMIO registers, mailboxes, signals, DMA-transfers and DMA List transfers. Similarly as in the methodology followed to validate functionally the SPU, we compared the results against the real machine and, step by step, we checked the values of the SPU registers, the MFC channels and MMIO registers, the LS and the main memory.

4.2 Timing Validation

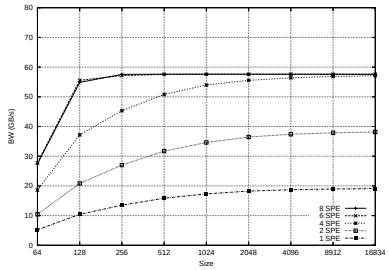
The timing validation has been performed in two phases. In the first phase we validated the SPU timing and, we did an overall validation in the second. For both phases, we used the timebase registers in the PPE (*Incrementer*) and in the SPEs(*Decrementers*). These hardware counters decrement their values at a specific frequency. On one hand, The PPU can access to the PPE decrementer using the *mtspr* and *mfspr* instructions of the PowerPC ISA. On the other hand, the SPE decrementer is accessed from the SPU through the *rdch* and *wrch* instructions of the SPU ISA.

The SPU timing validation was carried out using SPU benchmarks that count the number of *ticks* (Decrementer value changes) they spend executing. First we execute these benchmarks in a real Cell Processor to extract the number of ticks. Secondly, using CellSim, we extracted the number of instructions of each benchmark to calculate the achieved IPC. Then, we tuned the *issue bandwidth* parameter in the SPU to run at the same IPC as the actual machine. Having executed the benchmarks in CellSim, we can state that if the SPU module execute at the same IPC than the actual Cell, the number of ticks is exactly the same for both executions. As a result, the simulator is able to provide the same timing

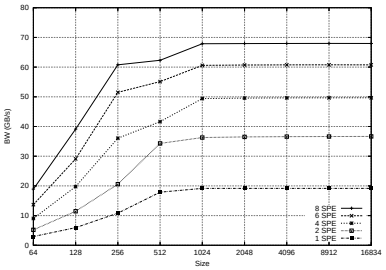


(a) CellSim SPE-MEM

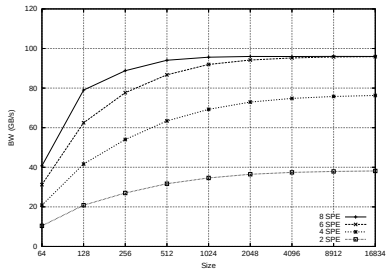
(b) Cell SPE-MEM



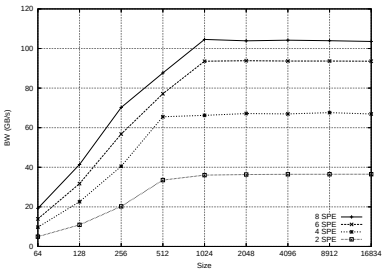
(c) CellSim SPE-SPE cycle



(d) Cell SPE-SPE cycle



(e) CellSim SPE-SPE couple



(f) Cell SPE-SPE couple

Figure 3: Interconnection network bandwidth results of Daniel Jimenez' programs. The SPE-MEM program transfers data between SPEs and main memory. SPE-SPE programs transfer data between SPEs. SPE-SPE cycle distributes SPEs in a cycle (SPE $i \rightarrow$ SPE $i+1$) and SPE-SPE couple in pairs.

information for the SPU as the real Cell as long as the IPC of the SPU module can be set using the issue bandwidth parameter.

Since the PPU and SPU performances can be tuned using parameters, the overall validation concentrates on the MFC and IN timings. For this task we chose the programs presented in Daniel Jimenez' EIB bandwidth study [8]. These programs stress the EIB by performing continuous DMA transfers to analyse the bandwidth the Cell's interconnection network provides. These experiments are classified depending on the sending and receiving elements. The SPE-MEM program implements DMA transfers between the SPEs and main memory. The SPE-SPE program sends data among LSs of several SPEs with the following configuration options: (a) the *cycle-elem* option performs DMA transfers from SPE i to SPE $(i+1)$ mod $total\_spes$; (b) the *cycle-dma* option communicates SPEs like cycle-elem but using DMA-list transfers; (c) in the *couple-elem* option SPEs are distributed in pairs and, finally, (d) *couple-dma* uses DMA-list commands for communicating couples.

To carry out the overall validation, we have executed Daniel Jimenez' programs in a Cell Processor at 2.4 GHz and and in CellSim using the parameters values seen in Table 1, column CS. Comparing the EIB and IN bandwidths we are not only validating the IN module but also the MFC, provided that the EIB bandwitdh depends on the way the MFC executes DMA commands. The right-hand side graphs show the EIB bandwidth achieved by the real machine executing the program for different DMA command sizes. The left-hand side graphs are the results when running in CellSim.

The results for SPE-MEM program's execution are shown in Figures 3(a) and 3(b). In both cases the maximum bandwidth is achieved provided that it is limited by memory bandwidth (19.2 GB/s). Since the simulator does not simulate the memory controller, there are differences for 1 and 2 SPEs. This problem can be solved by adjusting the number of buses in the IN as we show for the following experiments.

Figures 3(c) and 3(d) present the bandwidth for the cycle experiments. The Cell Processor assigns any physical SPE to a logical SPE, so it does not guarantee that SPE i is physically next to SPE $i+1$. This fact generates a lot of contention in the EIB for the cycle program, which results in a bandwidth lower than CellSim with 4 buses, where the physical SPE layout does not have any effects. In order to obtain a bandwidth closer to the real machine in this experiment, we adjust the number of buses in the IN to 3 so as to decrease CellSim's bandwidth.

In the *couple* experiments, we have the reverse situation, Figures 3(e) and 3(f). The EIB allows up to 3 non-overlapping transfers per ring but not more than 8 simultaneously in the 4 rings. Due to a better SPE layout with respect to the cycle programs the EIB achieves a higher bandwidth than the IN with 4 buses. Then, we performed the opposite action to the cycle case and we increased the number of buses in CellSim to 5, Figure 3(e) and get a close bandwidth results.

For the cycle and couple experiments we have performed the same adjustments for all the executions with different numbers of SPEs. However, each case can be tuned individually to minimize the error with respect to the real machine. For instance, in the SPE-MEM experiment, CellSim parameters does not need to be changed for 4, 6 and 8 SPEs, but one can decrease the number of buses or the individual bus bandwidth in the IN to compensate the lack of a memory interface controller.

5 Conclusions

The perspectives of both the industry and the research community show that the future of multiprocessors should be heterogeneous. We have developed a modular simulator (CellSim) for heterogeneous multiprocessors using as a first design the IBM's Cell Processor. We have validated CellSim against the Cell Processor using programs that stress the interconnection. In the validation process we have shown that CellSim is configurable enough to provide timing information similar to the real machine

and, therefore, be a valid tool for doing research.

6 Acknowledgements

This work has been supported by the European Commission in the context of the SARC integrated project (EU contract 27648-FP6), and the Spanish Ministry of Education under contract CICYT TIN2004-07739-C02-01. It has been also supported by the Programme Al-Ban, the European Union Programme of High Level Scholarships for Latin America, scholarship No. E05D058240CO; and by the Spanish Ministry of Education, scholarship No. AP2005-4245. Cabarcas is professor at *Universidad de Antioquia*

References

[1] https://unisim.org/site/. UNISIM.

[2] Innovation, processing performance and cooperation key to moving forward faster. INTEL DEVELOPER FORUM, April 17 2007.

[3] K. Asanovic, R. Bodik, B. C. Catanzaro, J. J. Gebis, P. Husbands, K. Keutzer, D. A. Patterson, W. L. Plishker, J. Shalf, S. W. Williams, and K. A. Yelick. The landscape of parallel computing research: A view from berkeley. Technical Report UCB/EECS-2006-183, EECS Department, University of California, Berkeley, December 18 2006.

[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] B. Black and J. P. Shen. Calibration of microprocessor performance models. *j-COMPUTER*, 31(5):59–65, 1998.

[6] P. Bohrer, J. Peterson, M. Elnozahy, R. Rajamony, A. Gheith, R. Rockhold, C. Lefurgy, H. Shafi, T. Nakra, R. Simpson, E. Speight, K. Sudeep, E. V. Hensbergen, and L. Zhang. Mambo: a full system simulator for the powerpc architecture. *SIGMETRICS Perform. Eval. Rev.*, 31(4):8–12, 2004.

[7] J. Chaoui, K. Cyr, J.-P. Giacalone, S. de Gregorio, Y. Masse, Y. Muthusamy, T. Spits, M. Budagavi, and J. Webb. Omap: Enabling multimedia applications in third generation (3g) wireless terminals. Technical report, Texas Instruments, 2000.

[8] Daniel Jimenez-Gonzalez, Xavier Martorell, and Alex Ramirez. Performance Analysis of Cell Broadband Engine for High Memory Bandwidth Applications. In *ISPASS 2007: Proc. of the IEEE International Symposium on Performance Analysis of Systems and Software*, Apr. 2007.

[9] R. Desikan, D. Burger, and S. W. Keckler. Measuring Experimental Error in Microprocessor Simulation. In *ISCA '01: Proceedings of the 28th Annual International Symposium on Computer Architecture*, pages 266–277, New York, NY, USA, 2001. ACM Press.

[10] M. Gschwind, H. P. Hofstee, B. K. Flachs, M. Hopkins, Y. Watanabe, and T. Yamazaki. Synergistic processing in cell's multicore architecture. *IEEE Micro*, 26(2):10–24, 2006.

[11] M. D. Hill and M. R. Marty. Amdahl's law in the multicore era. Technical report, University of Winsconsin at Madison.

[12] J. A. Kahle, M. N. Day, H. P. Hofstee, C. R. Johns, T. R. Maeurer, and D. Shippy. Introduction to the Cell multiprocessor. *IBM Journal of Research and Development*, 49(4-5):589–604, July 2005.

[13] R. Kumar, K. I. Farkas, N. P. Jouppi, P. Ranganathan, and D. M. Tullsen. Single-ISA Heterogeneous Multi-Core Architectures: The Potential for Processor Power Reduction. In *MICRO 36: Proc. of the 36th annual IEEE/ACM International Symposium on Microarchitecture*, page 81. IEEE Computer Society, 2003.

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

[15] 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.

[16] Y. N. Patt. Guest Speaker at the IEEE Computer Society 60th Anniversary Reception in San Juan, Puerto Rico, 14 June.

[17] J. M. Paul, A. Bobrek, J. E. Nelson, J. J. Pieper, and D. E. Thomas. Schedulers as model-based design elements in programmable heterogeneous multiprocessors. In *DAC '03: Proceedings of the 40th conference on Design automation*, pages 408–411, New York, NY, USA, 2003. ACM Press.

[18] M. Rosenblum, S. A. Herrod, E. Witchel, and A. Gupta. Complete computer system simulation: The SimOS approach. *IEEE parallel and distributed technology: systems and applications*, 3(4):34–43, Winter 1995.

[19] D. Tullsen. Simulation and modeling of a simultaneous multithreading processor. In *22nd Annual Computer Measurement Group Conference*.

[20] J. Wawrzynek, M. Oskin, C. Kozyrakis, D. Chiou, D. A. Patterson, S.-L. Lu, J. C. Hoe, and K. Asanovic. Ramp: A research accelerator for multiple processors. Technical report, University of California at Berkeley, 2006.

[21] T. F. Wenisch, R. E. Wunderlich, M. Ferdman, A. Ailamaki, B. Falsafi, and J. C. Hoe. Simflex: Statistical sampling of computer system simulation. *IEEE Micro*, 26(4):18–31, 2006.