Simulating the Future kilo-x86-64 core Processors and their Infrastructure

Antoni Portero[1], Alberto Scionti[1], Zhibin Yu[1], Paolo Faraboschi[2], Caroline Concatto[3], Luigi Carro[3], Arne Garbade[4], Sebastian Weis[4], Theo Ungerer[4], Roberto Giorgi[1]
[1]University of Siena
[2]HP labs
[3]UFGRS Universidade Federal do Rio Grande do Sul
[4]Augsburg University
[1]{portero, scionti, zhibin, giorgi}@dii.unisi.it,
[2]paolo.faraboschi@hp.com,
[3]{concatto, carro}@inf.ufrgs.br
[4]{arne.garbade, sebastian.weis, ungerer}@informatik.uni-augsburg.de

Abstract

The continuous improvements offered by the silicon technology enables the integration of always increasing number of cores on a single chip. Following this trend, it is expected to approach microprocessor architectures composed of thousands of cores (i.e., kilo-core architectures) in the next future. To cope with the increasing demand for high performance systems, many-core designs rely on integrated network-on-chips to deliver the correct bandwidth and latency for the inter-core communications. In this context, simulation tools represent a crucial factor for designing architectures at such scale of integration. The efficient simulation of the interconnection network along with the overall architecture (i.e., cores, cache memories, accelerators, etc.) still represents a complete open issue. This paper proposes a framework based on the COTSon simulator, able of scaling towards heterogeneous kilo-core architectures. Compared with current state-of-the-art architectural simulators, our framework provides not only a full-system architectural simulator, but a full-integrated accurate network-on-chip simulator. The framework shows a well balanced trade-off between simulation speed and accuracy, supporting the power consumption estimation. Experimental results demonstrate the ability of our framework to correctly simulate a large many-core machine and its interconnection network, considering different traffic patterns.

Keywords: Performance Analysis and Design Aids, Simulation, Verification, Verification, Worst- case analysis.

1. INTRODUCTION

Looking at the Moore's Law, the continuous improvements offered by silicon technology make possible placing always increasing number of transistors on a single chip. Following this trend, it is expected that future high performance microprocessor systems will be composed of thousand of cores (i.e., *kilo-core* architectures). Matching the performance request with the power consumption requirement will lead to a massive adoption of heterogeneous architectures. This huge amount of processing units requires an adequate interconnection subsystem in order to provide enough bandwidth for both inter-core and external to the chip communications. Among the possible systems (i.e., bus, crossbar switch, network-on-chip), networks-on-chips, or simply NoCs, have demonstrated to be highly scalable and reliable interconnections, delivering large bandwidth and low latency communications. Further, NoC infrastructures can meet the increasing stringent power consumption requirements of modern many-core architectures.

More than in the past, the adoption of architectural simulators has become essential for assuring the correctness of any design. Architectural simulators historically suffered from low simulation speed and accuracy, imposing serious limitations on the ability of predicting correct behaviors of the designed architecture, especially in the many-core era. The adoption of always complex interconnection systems, and in particular of NoCs, has led to the creation of tools specifically devoted to the accurate timing simulation (i.e., the tool can provide an accurate simulation of the behavior of the network with an estimation of the communication bandwidth and latency and the energy consumption) of these communication infrastructures. Although the existence of such tools providing functional, timing and energy models, there is a lack of frameworks that integrate both many-core architecture and network-on-chip simulators.

With the aim of providing a tool characterized by a high simulation speed and accuracy for an heterogeneous kilo-core architecture integrating an accurate network-on-chip simulator, this paper proposes a framework based on the COTSon infrastructure and the Noxim simulator [5]. Compared with current state-of-the-art simulation platforms, the proposed one offers a complete environment for a many-core full-system simulation, for accurate NoC timing simulation and energy characterization, and for the power consumption estimation of the many-core architecture. In order to guarantee fast simulations, COTSon implements a functional-

directed approach, where functional emulation is alternated to a complete timing-based simulation. The result is the ability of supporting the full stack of applications, middlewares and OSs. The modular approach on which COTSon is based, allow us the adoption of both the proprietary AMD SimNow [4] (available now) emulator and the open source Qemu [10] (in progress) based functional emulator, opening the door to the support of several different micro-architectures. Finally, the integration of the proposed framework with the McPAT tool [18], provides the ability of predicting the power consumption for a given design.

The remaining part of the paper is organized in five sections as follows. In Section 2. the state-of-the-art for the simulation frameworks and tools supporting both architectural and NoC simulation is analyzed. Section 3. presents the architecture of the proposed simulation framework, detailing on the functional-directed architectural simulator based on COTSon, while Section 4. describes an instance of the proposed architecture with the integrated Network-on-chip simulator. Section 5. provides the main results of a large experimental campaign obtained with the proposed simulation framework. Finally, Section 6. summarizes the main contribution and results of our work.

2. PREVIOUS WORK

Historically, architectural simulators, such as TraceFactory [14] and GEMS [22], have been used to accurately simulate the behavior of always more complex processor designs. Moving towards multi-core architectures requires the integration of different components that allow to correctly analyze the power consumption of the target system and its interconnections. Recent works proposed different approaches aiming at the simulation of kilo-core systems and of network-on-chip infrastructures. In [20] the authors present HORNET, an architectural simulator based on a highly configurable network-on-chip engine. It may use processor traces, execute a MIPS-based timing simulation or execute native code using the Pin instrumentation tool [21]. Although it targets kilo-core systems, it exploits a network oriented design, with a limited support for accurate heterogeneous core timing simulations and the inability of running full-system simulations. Similarly, in [28] the authors describe Multi2Sim, a framework for super-scalar, multi-threaded and multi-core processors simulations. It is composed of a functional emulation and a timing simulation layers.

One of the main characteristics distinguishing this tool from the others, is the implementation of a timing model also for graphic processors, allowing the simulation of GPU-based systems. However, there is a complete lack for the interconnection network model, thus limiting the design space exploration only to the architectural side. Other limitations are given by the only support to the MIPS architecture and the inability of supporting full-system simulation. MARSS-x86 [25] is a fast cycle-accurate full-system simulator for heterogeneous architectures. It combines the functional emulation provided by Qemu with the timing simulation offered by PTLsim [31]. While the Qemu emulation makes possible to target several core architectures including chip-sets and peripheral devices, PTLsim only offers accurate timing models for the x86-64 micro-architecture, requiring additional code to target different systems. Furthermore, PTLsim has not been designed to work on a kilo-core scale, thus it does not integrate any model (i.e., neither functional, nor timing models) for the interconnection network of such systems. A similar approach has been explored by the authors of MPTLsim [32]. It is a cycle-accurate, full-system simulator targeting x86 and x86-64 multi-core architectures. MPTLsim uses the hardware abstraction provided by the Xen hypervisor [3] to fast-forward execution and reach a given point where simulation can start. MPTLsim provides a significant faster simulation rate compared to simulators such as GEMS, which use an off-the-shelf sequential emulator (Intel's Simics) with a timing-first simulation approach (i.e., the timing model drives Simics one instruction at a time, thus resulting in low performance). MPTLsim makes use of a cycle accurate out-of-order core design implementing the x86-64 ISA.

However, the fast-forwarding approach is completely opaque to the simulator and during the Xen execution nothing can be inferred about memory accesses, instructions or I/O operations. MANIFOLD [30] is a simulator designed with a similar approach to our proposed framework. It decouples functional and timing simulations. Functional simulation is based on the Qsim library, while the timing model is based on a dedicated module. It supports also the integration of thermal and power modeling, however it appears to a premature stage of development. Finally, GRAPHITE [24] is a multi-core simulator designed to provide high level of simulation performance by distributing the simulation workload. Although its ability of scaling up to kilo-core machines keeping a fast simulation process, it only provides accurate estimation methods, while completely lacking of timing and functional models for the interconnection system.

Recently, with the large adoption of NoCs as the primary interconnection, several simulation tools have been proposed to evaluate the impact of the interconnection network on the overall system. In [12], the authors proposes GARNET, a detailed cycle-accurate interconnection network model integrated within GEMS. The tool provides a model for all the main components of the interconnection: flit-level input buffers, routing logic, allocators and the crossbar switch. Although it replaces the original simple network model coming with GEMS, using a more sophisticate one, it continues to suffer of the main limitations of the micro-architectural GEMS simulator. DARSIM [19] is a tool that offers a highly

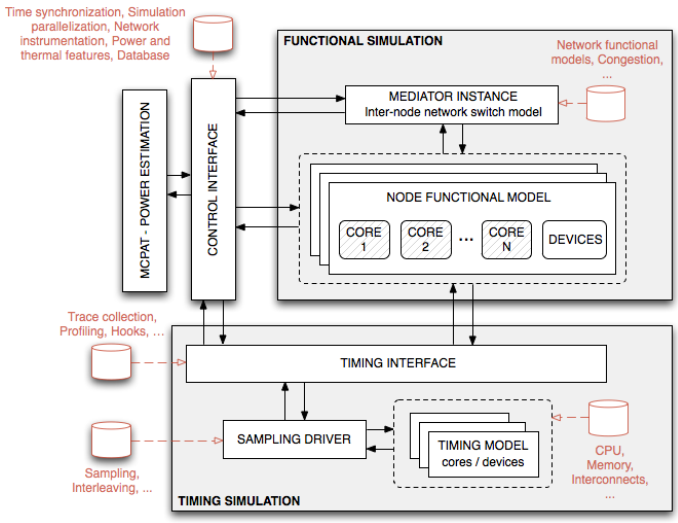

Figure 1. Many-core architectural simulation framework overview.

configurable, parallel network-on-chip engine based on an ingress-queued wormhole router architecture. Most hardware parameters are configurable, including geometry, bandwidth, crossbar dimensions, and pipeline depths. It comes with the possibility to feed the network with synthetic patterns or MIPS based application traces. Similarly to the HORNET simulator, it mainly lacks of the support for more common architectures (i.e., essentially x86 and x86-64). In [9, 29], authors present a modular, extensible, open-source C++ library for implementing network simulators. The library offers several advantages to the simulation framework such as traceability and debug features. It also provides the support for parallel implementation of the simulator frameworks. Although its potential scalability and flexibility (i.e., with respect to the previous cited tools, the library allows the definition of any arbitrary topology), it does not provide any feature for the architectural simulation, for which an external tool is still required. The following sections present our approach for the integration of the Noxim simulator with the COTSon framework.

3. SIMULATION FRAMEWORK

Figure 1 sketches the structure of the proposed simulator. It is composed of three main blocks connected each other. The up-center side of the figure shows the functional block, formed by the functional emulator (without timing annotation). This functional emulator can be the proprietary AMD SimNow (currently available) tool that targets x86-64 architectures or the open source Qemu (in the near future) full-system emulator that offers a larger architectural support (e.g., x86-64, Microblaze, MIPS, PowerPC, etc.). In this work we opted for the former, since our focus is the integration of the interconnection simulator, rather than the support for heterogeneous architectures. The down side of the figure shows the timing simulation block. For each component (i.e., cores, caches, memory, disks and networks interfaces) of the target architecture, there is a specific timing model. The third block is represented by the control interface, which allows the user to set the simulated guest system, control the simulation, and to run power consumption analysis. Power consumption of the simulated guest system is analyzed recurring to the McPAT tool. The simulation trace (e.g., instruction counting, memory accesses, etc.) of the guest system is stored in a database. Subsequently, McPAT tool combines the guest system configuration and the database traces to estimate the power consumption of the guest system. Although this mechanism can be adapetd to provide periodic power consumption estimation during the guest system simulation, this task is out of the scope of this work.

A (timing) simulator can use different approaches to run a simulation, depending on the relationship between the "functional model" (fm) and the "timing model" (tm) [23]. In the "functional-first" or "trace-driven" approach, the *fm* is run first and separately, while the *tm* is run later on, in a completely decoupled fashion. In the "timing directed" or "execution driven" methodology, the *fm* and *tm* are closely coupled (no decoupling), while in the "timing-first" approach, the *tm* drives the *fm*. In this case, the *tm* and the *fm* are completely decoupled, but the functional execution has to be checked later on and eventually undone. As previously mentioned, COTSon explored the "functional-directed" approach, in which the *fm* drives the *tm*. Both the functional and the timing models are completely decoupled. The functional model is always assumed to be the right one, but a timing feedback from *tm* to correct the timing behavior is introduced, so that it becomes visible to the applications being simulated [11].

Assuming a functional-directed approach, periodically the functional emulation is paused in order to perform the timing simulation. The timing simulator block collects a set of events (e.g., instruction counts, memory accesses, etc.) from the functional emulator, and uses its accurate internal models to adjust the global simulation time and the speed of the functional emulation. With the aim of decoupling functional emulation from timing simulation, in COTSon [8] a communication interface is designed, in order to simplify exchanging events and feedbacks information. For correctly managing events produced by a many-core target architecture, a set of event queues are implemented. Thus, the sequential instruction stream coming out of each emulated core is interleaved to account for the correct time order before running the timing simulation.

The support for a kilo-core system is given by interconnecting a large number of nodes (i.e., currently SimNow in-

stances). Each node is compose of a certain number of cores with their cache hierarchies, memory blocks, network interfaces and disks. Nodes are connected each other through their network interfaces. Properly setting the timing model of the network interfaces it is possible to correctly simulate the presence of a network-on-chip. As previously mentioned, one of the main issues, when design an architectural simulator, concerns the trade-off between simulation speed and accuracy. Since timing simulation is a time consuming process that causes the main slowdown, COTSon [7] adopts a sampling strategy to choose the timing simulation only for those application phases of interest. This goal is accomplished by implementing a sampling mechanism: the functional emulation is monitored, and whenever one of the phases is reached the timing simulation is enabled.

Currently, COTSon can perform multi-(guest-) machine simulations but only if the applications are written assuming a distributed memory machine (e.g., using a messaging library like MPI). Based on this premise, in this work we want to target the simulation of a future many-core microprocessor architecture, where standard x86-64 cores are grouped into several asymetric nodes and the communication is guaranteed by an integrated network-on-chip. Noxim simulator [5] is used to correctly model the communication infrastructure, and can be extended in order to support specific applications. It provides both the functional emulation and the timing simulation models. Since the tendency of this kind of architectures is to support the execution of a huge number of threads, in this work we target also a new execution model for the simulated architecture. The following section will detail about the architecture of the target microprocessor.

4. TARGET KILO-CORE PROCESSOR

As mentioned in the Section 3., the proposed simulation framework has been designed to support the simulation of complex microprocessor architectures composed of thousands of cores, and their interconnection infrastructures. Starting from this premise, the processor architecture we want to target is based on several x86-64 cores grouped into asymmetric nodes. Currently, COTSon support timing models for x86-64 architecture, but it can be extended in order to provide models for different architectures enabling the simulation of heterogeneous systems. Each node can have a different number of cores, and the interconnection among the nodes is based on an embedded Network-on-Chip. It is expected that such type of machines will be able to support the execution of massive set of threads. Looking at this processor architecture tendency, we also propose an extension in the x86-64 ISA allowing threads in the processor to be executed in a Data-Flow fashion.

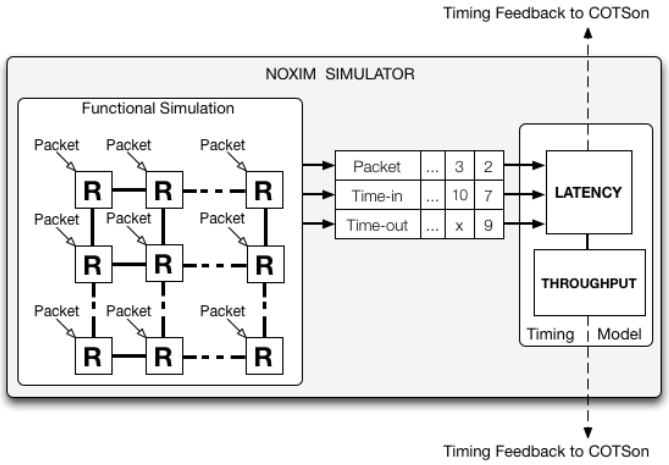

Figure 2. Noxim instance and its interface with the architectural simulation framework.

4.1. ISA extension supporting Data-Flow threads

The basic execution model aims at supporting the execution of a very large number of fine-grained threads that are generated after the compilation process. Moreover, we also want to target threads that have a "Data-Flow behavior" [17, 16], in the sense that they are repeatable with no-side effects and their inputs and outputs are clearly identified. We call these threads DF-Threads (Data-Flow threads). However, in order to make possible the execution of DF-threads, a set of special instructions called T* ISA Extension [27, 13, 15, 26] are required. In particular, the proposed extension consists in two instructions for generating/stopping DF-Threads (TSCHEDULE, TDESTROY), two instructions for operating on input/output data of the DF-Threads (TWRITE, TREAD), two instructions for allocating/freeing (TALLOC, TFREE) special purpose memory (even if the memory model is out of the scope of this document, it is an essential part of our execution model). Besides the ISE, we introduced the architectural support: an integrated Distributed Thread Scheduler Unit (D-TSU) is responsible for allocating tasks for processors inside the node and for maintaining balanced workload on each of them, depending on the execution of the T* based instructions.

4.2. The NoC interconnection

The NoC model is based on the Noxim simulator. Noxim is cycle-accurate simulator, developed using C++ and SystemC languages, and it can be downloaded from SourceForge under GPL license terms. The NoC model consists of a low-level interconnection network modeling the detailed features of a real network-on-chip infrastructure. Researchers interested in investigating different network-on-chips can easily modify the NoC architecture. The NoC follows, somehow, the strategy

used by COTSon, that is to split the overall simulation process into a functional and a timing simulation processes, as depicted in figure 2. The functional block (i.e., the mediator) is responsible for emulating the communication process among the cores. The emulation process produces a set of events (i.e., mainly packet identifier, time entering a router, etc.) that is temporarily stored in event queue. Events in the queue are processed by the timing simulation block, extracting the latency and throughput of the communication infrastructure. Noxim makes available these information to the architectural simulator, in order to correctly adjust the overall simulation process. The timing model exported by the NoC simulator is called by the user whenever the computation of the application performance in presence of a NoC is required. In this work we configured the NoC configuration as a 2D-Mesh NoC.

4.3. Simulation platform

In this subsection, we expose the architecture of the platform able to support the simulation of a very high number of cores. In order to achieve this goal, we need a powerful simulation system. We define the host machine as the computer where we run the simulated processor, and the guest machine as the proper simulated one. Currently, we use a host machine equipped with 64 cores and 1TB of main memory. There is a trade-off between complexity of the guest system and the time required by the simulation. A good trade-off is to use one host-core for each functional instance. Each node can have till 32 cores but we have experimented that 16 cores per node can better scale up. The simulation of larger architectures can be achieved distributing the simulation on more than one host. However since we want to focus on the simulation of a 1K-core system, considering a single host machine is sufficient.

In order to correctly simulate a kilo-core architecture, we booted up 64 nodes, each one containing 16 cores. Each node runs a Linux distribution operating system. On top of this system, we are able to run several test benches based on both OpenMP and MPI programming models. One of the main modifications we did, has also been the implementation of the support of DF-threads through the ISA extension. DF-threads enable a different execution model based on the availability of data and opens the doors for many architectural optimizations not possible in current standard off-the-shelf cores.

5. EXPERIMENTS

Given the scenario exposed in the previous section, we analyzed the execution of different applications that use both OpenMP and MPI programming models on the top of the proposed target processor architecture. Experimentation and tight collaboration with the compiler and simulation tools will assess the merits of these programming models running on the target architecture.

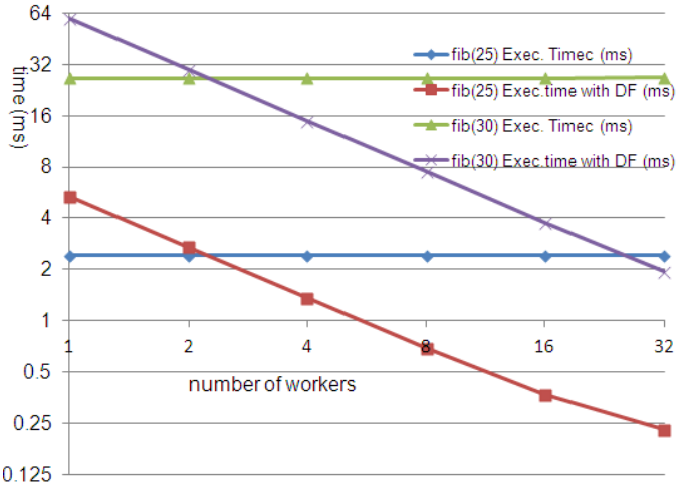

Figure 3. Execution time vs. number of workers for the Sequential Recursive Fibonacci application with input in the range of 25 to 30.

In the following subsections, we present the set of experiments performed. The experiments are intended to validate the simulation framework, when both the processor architecture and the communication infrastructure are considered. From this point of view, a first experiment is used to show the impact and scaling capability of the Data-Flow support. A second experiment presents the performance of different real benchmark applications running on the platform with 64 nodes and 16 cores for each node. In particular, over this cluster of nodes, we executed HPL 2.0-linpack [2] and Graph500 [1] to get performance of the system. The last experiment presents the network load distribution induced by sending and receiving packets from a server node to all the auxiliary cores nodes. All the experiments demonstrate the ability of the proposed framework to target the simulation of complex kilo-core systems and their interconnection NoC.

5.1. Data-Flow Threads experiment

As we have explained in subsection 4.1., we have modified the architecture allowing to integrate 6 new instructions that allow using DF-Threads. We used Sequential Recursive Fibonacci (SRF) in modified assembler. The node has 32 processors and we have added to the node architecture the model of the D-TSU. We can observe in figure 3, that execution time of SRF example with DF-threads can scale-down almost linearly. Compared to this result, standard processor architectures exhibit a constant execution time, regardless of the input parameter. Thus, this example shows that fine grain DF-Threads scale up linearly at node level.

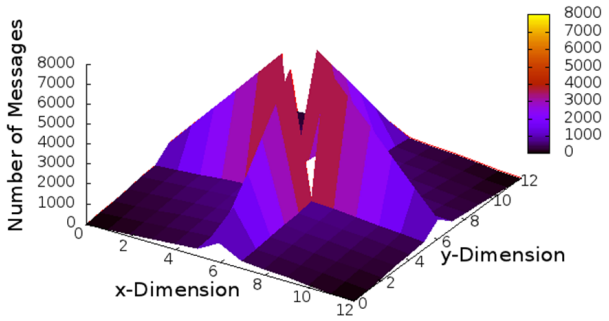

Figure 4. Network load distribution induced by test-bench example using XY routing.

5.2. Running benchmarks to assess the performance of a 1024 core processor

We executed HPL-2.0. linpack in the simulated platform composed of 64 nodes with 16 cores per node (with the parameters N, NB, P and Q respectively equal to 4000, 100, 1, 16). The maximum performance achieved per node is 2.1 Gflops. We have also executed Graph500, the example based on MPI programming model with a problem size equal to 15. The result obtained was 12.3 Million TEPS (Traversed Edges per Second) per node. If we ideally scale up to 64 nodes, we could get 788 Million TEPs. With this result, the reference MPI machine would rank in number 38, according to the graph 500 list (Nov. 2011). Although, we are running these benchmarks in this complex platform, we foresee that the inter-tile communication is our bottleneck since currently, we are not able to produce enough input data for each node to obtain this ideal performance. Therefore, we are working in the memory layer architecture and the software support to scale up the test-benches. But without any doubt, the simulator helps us to make an architecture exploration and detect problems that in another way it would be very difficult or even impossible. As we show in the following experiment, where it is depicted how the inter-tile communication produces the bottle-neck of the system.

5.3. Simulating NoC load Distribution

In this experiment, a server (master) node injects data packets through the NoC to the rest of the auxiliary (slave) cores. We show the communication pattern in the 2D-Mesh Noxim NoC simulator with 12 by 12 tiles. As we can observe in figure 4, the centralized communication overhead influences the communication capabilities directly, as the data message in the NoC interfere with other messages within the communication network.

6. CONCLUSIONS

This work introduces a first version of a simulation framework targeting future heterogeneous kilo-core architectures. The paper presents diverse scenarios where the x86-64 instruction set has been modified to support Data-Flow threads. In these scenarios we simulated a full-system with 1024 cores and the communication among nodes. The simulator framework serves to find the bottle-necks of the system and allow us to produce architectural explorations that would not be feasible in another way. Experimental results confirmed the ability of simulating future kilo-core systems.

COTSon trunk and the diverse branches with some of the developments explained in this paper can be found in sourceforge [6].

ACKNOWLEDGEMENTS

This work was partly funded by the European FP7 projects TERAFLUX id. 249013 http://www.teraflux.eu, ERA (Embedded Reconfigurable Architectures) id. 249059 (FP7) http://era-project.eu; HiPEAC IST-217068, and IT PRIN 2008 (200855LRP2).

REFERENCES

[1] The graph500 list. http://www.graph500.org.

[2] Hpl - a portable implementation of the hp linpack benchmark for distributed-memory computers. netlib.org/benchmark/hpl.

[3] Xen community overview. http://xensource.com/xen, 1999.

[4] *AMD SimNow Simulator 4.6.1 User's Manual*, November 2009.

[5] Noxim simulator. http://noxim.sourceforge.net/, 2009.

[6] Cotson sourceforge. svn https://cotson.svn.sourceforge.net/svnroot/cotson cotson, 2012.

[7] Patent: US 7,912,690 B2, Mar. 22, 2011.

[8] Patent: US 2008/0270959 A1, Oct. 30 2008.

[9] Al-Badi and et al. A parameterized noc simulator using omnet++. In *International Conference on Ultra Modern Telecommunications & Workshops*, 2009.

[10] F. Bellard. Qemu, a fast and portable dynamic translator. In *Proceedings of the 2005 USENIX Annual Technical Conference*, 2005.

[11] E. Argollo et al. Cotson infrastructure for full system simulation. *Operating Systems Rev*, 43:52–61, 2009.

[12] Niket Agarwal et al. Garnet: A detailed on-chip network model inside a full-system simulator. In *Performance Analysis of Systems and Software (ISPASS)*, pages 33–42, 2009.

[13] Roberto Giorgi et al. Pr d7.2 definition of isa extensions, custom devices and external cotson api extensions. Technical report, FET proactive 1: Concurrent Tera-Device Computing (ICT-2009.8.1) PRJ. NUM.: 249013, 2011.

[14] R. Giorgi and et al. Trace factory: Generating workloads for trace-driven simulation of shared-bus multiprocessors. In *IEEE Concurrency*, pages 54–68, October 1997.

[15] R. Giorgi, Z. Popovic, and N. Puzovic. Dta-c: A decoupled multi-threaded architecture for cmp systems. In *Proc. IEEE SBAC-PAD, Gramado, Brasil*, Oct. 2007, pp. 263.

[16] Roberto Giorgi, Zdravko Popovic, and Nikola Puzovic. Implementing fine/medium grained tlp support in a many-core architecture. In *SAMOS*, pages 78–87, 2009.

[17] Krishna M. Kavi, Roberto Giorgi, and Joseph Arul. Scheduled dataflow: Execution paradigm, architecture, and performance evaluation. *IEEE Trans. Computers,, Los Alamitos, CA, USA*, 50, no. 8,:pp. 834–846, Aug. 2001.

[18] S. Li and et al. Mcpat: An integrated power, area, and timing modeling framework for multicore and manycore architectures. In *Proceedings of the 42nd Annual International Symposium on Microarchitecture*, pages 469–480. IEEE/ACM, December 2009.

[19] M. Lis and et al. Darsim: A parallel cycle-level noc simulator. In *Workshop on Modeling, Benchmarking, and Simulation*, June 2010.

[20] M. Lis and et al. Scalable, accurate multicore simulation in the 1000-core era. In *ISPASS*, pages 175–185, April 2011.

[21] Chi-Keung Luk and et al. Pin: Building customized program analysis tools with dynamic instrumentation. In *ACM SIGPLAN International Conference on Programming Language Design and Implementation*, 2005.

[22] M. M. K. Martin and et al. Multifacet's general execution-driven multiprocessor simulator (gems) toolset. *SIGARCH Comput. Archit. News*, 33(4):92–99, November 2005.

[23] C. J. Mauer, M. D. Hill, and D. A. Wood. Full-system timing-first simulation. *In SIGMETRICS 2002 Proceedings of the 2002 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, 1:118–116, 2002.

[24] J. E. Miller and et al. Graphite: A distributed parallel simulator for multicores. In *HPCA*, 2010.

[25] A. Patel and et al. Marss-x86: A qemu-based microarchitectural and systems simulator for x86 multicore processors. In *1st International Qemu Users' Forum*, pages 29–30, 2011.

[26] Antoni Portero, Zhibin Yu, and Roberto Giorgi. T-star (t*): An x86-64 isa extension to support thread execution on many cores. *ACACES Advance Computer Architecture and Compilation for High-Performance and Embedded Systems*, 1:277–280, 2011.

[27] Antoni Portero, Zhibin Yu, and Roberto Giorgi. Teraflux: Exploiting tera-device computing challenges. *Procedia Computer Science, Proceedings of the 2nd European Future Technologies Conference and Exhibition 2011 (FET 11), 4-6 May, 2011, Budapest*, 7:146–147, 2011.

[28] R. Ubal and et al. Multi2sim: A simulation framework to evaluate multicore-multithreaded processors. In *SBAC-PAD*, pages 62–68, 2007.

[29] A. Varga and et al. The omnet++ discrete event simulation system. In *In Proceedings of the European Simulation Multiconference (ESM'2001)*, pages 319–324, 2001.

[30] S. Yalamanchili. Manifold: Modeling and simulation of many core architectures. In *First Workshop on High Performance Computing Architectural Simulation (HPCAS)*, September 2009.

[31] M. T. Yourst. Ptlsim: A cycle accurate full system x86-64 microarchitectural simulator. In *ISPASS*, pages 23–34, April 2007.

[32] Hui Zeng and et al. Mptsim: A cycle-accurate, full-system simulator for x86-64 multicore architectures with coherent caches. *ACM SIGARCH Computer Architecture News*, 37(2), 2009.