Design space exploration in the microthreaded many-core architecture

Irfan Uddin
University of Amsterdam, The Netherlands
mirfanud@uva.nl
May 22, 2014

Abstract

Design space exploration is commonly performed in embedded system, where the architecture is a complicated piece of engineering. With the current trend of many-core systems, design space exploration in general-purpose computers can no longer be avoided. Microgrid is a complicated architecture, and therefore we need to perform design space exploration. Generally, simulators are used for the design space exploration of an architecture. Different simulators with different levels of complexity, simulation time and accuracy are used. Simulators with little complexity, low simulation time and reasonable accuracy are desirable for the design space exploration of an architecture. These simulators are referred as high-level simulators and are commonly used in the design of embedded systems. However, the use of high-level simulation for design space exploration in general-purpose computers is a relatively new area of research.

Contents

1 Introduction 2
2 Design space exploration in embedded systems 2
3 Application-dependent and-independent DSE 3
4 Design space exploration in general-purpose systems 4
5 Design space exploration in the Microgrid 7
6 Conclusion 8
1 Introduction

Simulators with high simulation speed and less complexity are desirable for early Design Space Exploration (DSE) of the architecture. Any decision to improve the architecture becomes more expensive and requires more effort at the later stage and requires more effort, time and budget. DSE is performed in all kinds of computer systems. However, in the embedded systems domain the use of high-level simulation for DSE purposes has been accepted as an efficient approach for more than a decade. In that sense, the DSE in embedded systems pioneered the high-level simulation techniques. Therefore, in this paper we will give details about DSE in embedded systems using high-level simulators. We will explain that general-purpose computers are getting more complex and therefore high-level simulators are also required for the DSE. Since Microgrid is a complex architecture therefore we need to know its design space before we present the high-level simulation techniques for the Microgrid.

The rest of the paper is organized as follows. In Section 2 we give an explanation of DSE in embedded systems. In Section 3 we differentiate the application dependent DSE and independent DSE. We give the DSE for the Microgrid in 5 and conclude the paper in 6.

2 Design space exploration in embedded systems

Embedded systems perform predefined tasks and therefore have particular design requirements. They are constrained in terms of performance, power, size of the chip, memory etc. They generally address mass products and often run on batteries, and therefore need to be cheap to be realized in silicon and power efficient. Modern embedded systems, typically have a heterogeneous MultiProcessor-System-on-Chip (MP-SoC) architecture, where a component can be fully programmable processor for general-purpose application or a fully dedicated hardware for time critical applications. This heterogeneity makes embedded systems more complex and therefore designers use high-level simulator to perform DSE at an early stage, because high-level simulators take less effort to develop and less time in executing applications. In this section we describe the high-level simulation technique used for the DSE in embedded systems.

Different high-level simulation techniques are introduced for the DSE in embedded systems, and are often based on the separation of concerns [26] between application, architecture and mapping functions. DSE in embedded systems is generally application-dependent or scenario-based. Traditional embedded systems are targeting one particular architecture and application, the aim is to explore the design for improvement based on certain objective. Scenario-based DSE [43] is the process of mapping every individual process of an application to every architecture component with different configurations. This mapping results in an exponential number of mapping choices i.e. design space. We show an example in fig. 1 (taken from [21]) to demonstrate that only three processes are mapped to three architecture components, but the resulted design space is large.

Ideally DSE would like to considers all possible mappings, but an exhaustive search is infeasible. Therefore computer architects use design pruning to optimize the search through the design space to speed up the DSE. A smart DSE intelligently evaluates a small fraction of the design space to come up with a sub-optimal solution. These choices have a crucial impact on the success of the final product. DSE addresses multiple objectives [15] e.g. maximum performance, minimum power consumption and less complex components. It is very difficult to have a single solution that meets all the objectives simultaneously. The main problem is that the objectives are conflicting e.g. low power generally means bad performance or good performance means high power usage. Therefore a
application events, divided in two groups: 
execute events for computational behavior 
and read and write events for communication behavior.

The architecture models in Sesame are cycle-approximate TLM models and simulate the performance consequences of the computation and communication events generated by an application model. Architecture models are constructed from building blocks provided by a library containing template models for processing cores, and various types of memories and interconnects.

Since Sesame makes a distinction between application and architecture models, it needs an explicit mapping step to relate these models for co-simulation. In this step, the designer decides for each application process and FIFO channel a destination architecture model component to simulate its workload. Here, Sesame provides support for modeling a variety of scheduling policies in case multiple application processes are mapped onto a single architectural processing element. Mapping applications onto the underlying architectural resources is an important step in the design process, since the final success of the design can be highly dependent on these mapping choices. In Figure 1, we illustrate this mapping step on a very simple example. In this example, the application model consists of three Kahn processes and FIFO channels. The architecture model contains two processors and one shared memory. To decide on an optimum mapping, many instances need to be considered (and thus simulated). In realistic cases, in which the underlying architecture can also be varied during the process of design space exploration, simulation of all points in the design space is infeasible. Therefore, analytical models are needed to prune the design space, steering the designer towards a small set of promising design points which then can be simulated. The remainder of this section provides an outline of the basic analytical performance model [3, 4] we use in Sesame for design space pruning, after which the subsequent sections present our signature-based mechanism to calibrate this analytical model.

The application models in Sesame are represented by a graph $KPN = (V_K, E_K)$ where the sets $V_K$ and $E_K$ refer to the Kahn processes and the directed FIFO channels.

Figure 1: The mapping of three application processes to three architecture components resulting into a large number of design space to be explored.

set of solutions are selected based on a Pareto optimal front [1], where solutions are not dominated by any other solution looking for the same objectives.

2.1 Related work

High-level simulators have been used for the DSE in embedded systems domain for more than a decade, and are used in the research of academia and industries. Below are some of the research groups using high-level simulation for DSE in embedded systems. There might exist other areas of research in using high-level simulation for the DSE of embedded systems.

- Sesame, University of Amsterdam [16].
- (Metro)Polis, University of California, Berkeley [50].
- Mescal, University of California, Berkeley [12].
- Milan, University of Southern California, Los Angeles [2].
- The octopus toolset, University of Eindhoven [4].
- SystemC-based environment, STMicroelectronics [47].

3 Application -dependent and -independent DSE

We want to clearly distinguish between application-dependent DSE in traditional embedded systems and application-independent DSE in modern embedded system or general-purpose computers. In

3
traditional embedded systems, applications are statically mapped to different configurations of an architecture using some mapping functions. Based on the simulation results, innovative ideas can be generated which can improve application, mapping and architecture separately.

In modern embedded systems we do not have one particular application or scenario, but a range of applications targeted to a different configurations of the architecture. For instance in smart phones it is not only one type of application that can statically be mapped, but a range of different types of applications are required to be explored on the different configuration of the architecture. In a way modern embedded systems are converging to the general-purpose systems. The range of applications increases in general-purpose computers, where a variety of applications can be executed on the given architecture. In these situations, the mapping of the application to architectural component can not be analyzed statically but instead the code patterns in algorithms are analyzed, and then different processes of an application are dynamically mapped to different parts of the chip based on certain objectives. Because of the dynamic mapping, application-independent DSE is not as trivial as scenario-based DSE.

Design pruning is more structured in traditional embedded systems. For instance, genetic algorithms, simulated annealing etc. are some of the structured techniques that are commonly used in design pruning. However, for design pruning in modern embedded systems or general-purpose systems, there exists no structured solution that can dynamically determine a reduction in the design space to optimize the search.

4 Design space exploration in general-purpose systems

The growing number of cores and size of the on-chip memory are creating significant challenges for evaluating the design space of future general-purpose computers. We need scalable and fast simulators for the exploration of large number of cores on a chip within limited development time and budget. Commercially available processors available in the market have few cores on a chip e.g. Intel E708800 Series, IBM’s POWER7 and AMD’s Opteron 600 Series. In the near future we believe there will be hundreds of cores per chip and DSE at the early stage can no longer be avoided in general-purpose computers, as the number of mapping an application explodes as the number of cores increases.

The use of high-level simulators for the DSE in general-purpose computers is relatively new compared to embedded systems domain. A number of simulation techniques are in research to develop high-speed simulators for the DSE of general-purpose computers with less complexity and shorter development time then conventional cycle accurate simulator. These simulation techniques are diverse and do not follow one particular pattern. In this section we give details of some high-level simulation techniques. There might exist other high-level simulations targeting general-purpose computers.

4.1 Interval simulation

Interval simulation [7,18] is a high-level simulation technique for the DSE of super-scalar single- and multi-core processors. It raises the level of abstraction from detailed simulation by using analytical models to derive the timing simulation of individual cores without the detailed execution of instructions in all the stages of the pipeline. The model is based on deriving the execution of an instruction stream in intervals. An interval is decided based on the miss events e.g. branch misprediction, cache misses, TLB misses etc. With interval analysis, execution time is partitioned
into discrete intervals using miss events. The analytical models of every core cooperate with miss events in the system, and can be extended to model the tight interleaving of threads in multi-core processors.

Interval simulation framework has two parts: functional simulation and timing simulation, and are connected with each other through a queue. The functional simulator feeds instructions into the tail of the queue and the timing simulator reads those instructions from the head of the queue. The functional simulator generates a dynamic instruction stream, including user-level and system-level code and is subsequently fed into the timing simulator. The timing simulator analyzes the code and advances the simulation time as per the time required to execute an instruction stream. In case of I-cache miss, branch misprediction and long latency load operations the simulation time is advanced by the miss latency, branch resolution time plus the front-end pipeline depth and long latency operations respectively.

Discussion

Interval simulator only simulates a small number of cores in super-scalar machines which disregards hardware microthreading and therefore the complexity of simulating latency tolerance is not encountered. In the Microgrid we can have more than 100 cores on the chip, and the architecture is completely different than super-scalar machine, as it provides fine-grained latency tolerance based on data-flow scheduling. The way programs can be written for the Microgrid is also different. Therefore interval simulation can not directly be used for the DSE of the Microgrid. However, we have learned some techniques from interval simulation and have used these in HLSim. For instance, in interval simulation in case of a cache miss the simulation time is advanced with the addition of cache miss latency. In HLSim we advance the simulation time with the cache miss latency but adjusted with a latency tolerance factor based on the number of active threads. Because in case of latency tolerance the cache miss latency can be shorter than the latency without any latency tolerance.

4.2 Statistical simulation

Statistical simulation has gained interest over the past few years, as it speeds up simulation by providing short running synthetic traces. The execution of the original benchmarks is profiled and the key execution characteristics are captured in a synthetic trace, which closely exhibits similar execution characteristic as original benchmarks. The key benefit of statistical simulation is that the synthetic trace clones the dynamic instruction count with several orders of magnitude smaller than in the original benchmarks, and therefore reduces the simulation time dramatically.

Nussbaum and Smith [29] and Hughes and Li [19] use statistical simulation paradigm to evaluate multithreaded programs running on shared-memory multiprocessor (SMP) systems. They have extended the statistical simulation to model synchronization and accesses to shared memory. Genbrugge and Eeckhout [13, 17] use statistical simulation to measure some execution characteristics in the statistical profile to be able to accurately simulate shared resources in multi-core processors.

Discussion

Statistical simulation is a trace driven simulation technique. A synthetic trace is generated which can be reduced to a shorter trace and is representative of the large trace of the benchmarks. The problem with this technique is that the original trace files can be very large which consume space and
this technique can not consider the dynamic adaptation of multiple applications on the chip. The high-level simulation of the Microgrid, is execution driven i.e we dynamically generate events which are representative of the instruction count in the basic block in a thread. These events are mapped to the architecture and represent the execution of the application with fine-grained interleaving. The events have information of a short piece of code and therefore statistical simulation techniques we are not a suitable choice to be used in HLSim.

4.3 Sampled simulation

The basic idea of sampled simulation is to simulate a number of sampling units rather than the entire dynamic instruction stream. The sampling units are selected either randomly [9], periodically [49] or based on phase analysis [33].

Different research in the multithreaded and multi-core processors simulation is using sampled simulation. Van Biesbrouck et al. [42] propose the co-phase matrix for speeding up sampled simultaneous multithreading (SMT) processor simulation running multi-program workloads. Stenstrom et al. [44] are researching the premise that fewer sampling units are enough to estimate overall performance for larger multi-processor systems than for smaller multi-processor system in case one is interested in aggregate performance only. Wenisch et al. [48] have obtained similar conclusions of throughput in server workloads. Barr et al. [3] propose the Memory Timestamp Record (MTR) to store micro-architecture state (cache and directory state) at the beginning of the sampling unit as a checkpoint.

Discussion

Sampled simulation is also a trace-based simulation technique which suffers from the large trace files to be processed and changing an application results in producing and analysing a different trace file. Every time there is some optimization in the application, a new trace needs to be generated and analyzed.

4.4 Related works

There are other simulation techniques used in the design space exploration of general-purpose computers given below.

- FPGA prototypes: They have low little simulation time, high accuracy and are useful in DSE. However these simulations require more development time and are more complex. They also suffer from combinatoric explosion of considering many low level parameters during design space exploration. Some examples are: [30, 31, 8, 46].

- Trace simulation: These simulation techniques generate large execution traces from benchmarks, and are used for the evaluation of the architecture. They avoid the extremely large analysis of the application, by executing the program only one time, generating the trace and mapping it to the trace to different configuration of the architecture. However a large storage is required in order to store the large traces and a change in the application requires a different trace to be generated. Statistical simulations and sampled simulations are some of the techniques that addresses the reduction of the large trace files. Some example are: [9, 20, 27].
5 Design space exploration in the Microgrid

5.1 Microgrid

The Microgrid [24, 5, 22] is a general-purpose, many-core architecture developed at the University of Amsterdam which implements hardware multi-threading using data flow scheduling and a concurrency management protocol in hardware to create and synchronize threads within and across the cores on chip. The suggested concurrent programming model for this chip is based on fork-join constructs, where each created thread can define further concurrency hierarchically. This model is called the microthreading model and is also applicable to current multi-core architectures using a library of the concurrency constructs called svpt-ptl [45] built on top of pthreads. In our work, we focus on a specific implementation of the microthreaded architecture where each core contains a single issue, in-order RISC pipeline with an ISA similar to DEC/Alpha, and all cores are connected to an on-chip distributed memory network [23, 6]. Each core implements the concurrency constructs in its instruction set and is able to support hundreds of threads and their contexts, called microthreads and tens of families (i.e. ordered collections of identical microthreads) simultaneously.

A number of tools and simulators are added to the designer’s toolbox and used for the evaluation of the Microgrid from different perspective. The compiler for the Microgrid [25] can generate binary for different implementations of the Microgrid. We have software libraries that provide the run-time systems for the microthreading model on the shared memory SMP machines and referred as svpt-ptl [45] and distributed memory for clusters/grids and are referred as Hydra [28] and dsvpt-ptl [44] The SL compiler can generate binary for UTLEON3 [10, 11], MGSim [6, 32] and HLSim [37, 38, 39, 36, 40, 41, 34, 35].

HLSim is a high-level simulation technique aimed for the DSE of the Microgrid and is based on discrete event simulation technique. It is execution driven simulator and therefore does not suffer from the large size of trace files. The events are dynamically mapped to the architecture at run time. We have built the simulator from scratch without using any off-the-shelf code, but some simulation techniques from Sesame and Interval simulation were used during the development for inspiration.

5.2 Design space in the Microgrid

The Microgrid is a complex many-cores architecture and therefore has a huge design space for complex application. In order to have an efficient and validated system in the silicon we need to perform DSE in the Microgrid to explore the performance of different applications on the different configurations of the architecture. After DSE we can perform design pruning to change these parameters that affect the performance. We categorize the design space of the Microgrid as:

- Static architectural parameters:
  1. Thread table size
  2. Family table size
  3. Frequency of cores and memory
  4. Number of cores sharing an FPU
  5. Frequency of delegation and distribution network
  6. Size of L1-cache and L2-cache
7. Associativity of L1-cache and L2-cache
8. Number of L1-caches sharing L2-cache
9. Number of L2-caches in low-level ring
10. Number of low-level rings associated in the top-level ring
11. Distribution of address space of RAM into banks
12. Size of directory and root directory
13. The memory architecture
14. Synchronization-aware protocol

• Dynamic application parameters:
  1. Place size
  2. Window size
  3. Cold caches

There are some other parameters that are very low-level e.g. size of the chip, FPU frequency, pipeline stages, the way cores are distributed on the chip etc. We have shown only the parameters that we will simulate in the current implementation of HLSim for the design space exploration in the Microgrid.

6 Conclusion

DSE is required in all kind of computer systems. The use of high-level simulators for DSE is pioneered in embedded systems and getting popular in general-purpose systems. As the Microgrid has a huge design space therefore, low-level simulators are not justifiable to be used for design space exploration. We need high-level simulators for the efficient design space exploration.

Acknowledgement

The author would like to thank Dr. Raphael Poss, Dr. Michiel van Tol and Prof. dr. Chris Jesshope.

References

[1] M.A. Abido. A niched pareto genetic algorithm for multiobjective environmental/economic dispatch. *International Journal of Electrical Power and Energy Systems*, 25(2):97 – 105, 2003.

[2] A. Bakshi and A. Ledeczi. Milan: A model based integrated simulation framework for design of embedded systems. In *ACM SIGPLAN Notices*, pages 82–93, 2001.

[3] K. C. Barr, H. Pan, M. Zhang, and K. Asanovic. Accelerating multiprocessor simulation with a memory timestamp record. In *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software, 2005*, ISPASS ’05, pages 66–77, Washington, DC, USA, 2005. IEEE Computer Society.
[4] Twan Basten, Emiel Van Benthum, Marc Geilen, Martijn Hendriks, Fred Houben, Georgeta Igna, Frans Reckers, Sebastian De Smet, Lou Somers, Egbert Teeselink, Nikola Trčka, Frits Vaandrager, Jacques Verriet, Marc Voorhoeve, and Yang Yang. Model-driven design-space exploration for embedded systems: the octopus toolset. In Proceedings of the 4th international conference on Leveraging applications of formal methods, verification, and validation - Volume Part I, ISOLO’10, pages 90–105, Berlin, Heidelberg, 2010. Springer-Verlag.

[5] Thomas A. M. Bernard, Clemens Greck, Michael A. Hicks, Chris R. Jesshope, and Raphael Poss. Resource-agnostic programming for many-core microgrids. In Proceedings of the 2010 conference on Parallel processing, Euro-Par 2010, pages 109–116, Berlin, Heidelberg, 2011. Springer-Verlag.

[6] K. Bousias, L. Guang, C. R. Jesshope, and M. Lankamp. Implementation and evaluation of a microthread architecture. J. Syst. Archit., 55:149–161, March 2009.

[7] Trevor E. Carlson, Wim Heirman, and Lieven Eeckhout. Sniper: exploring the level of abstraction for scalable and accurate parallel multi-core simulation. In Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis, SC ’11, pages 52:1–52:12, New York, NY, USA, 2011. ACM.

[8] Derek Chiou, Dam Sunwoo, Joonsoo Kim, Nikhil A. Patil, William Reinhart, Darrel Eric Johnson, Jebediah Keefe, and Hari Angepat. Fpga-accelerated simulation technologies (fast): Fast, full-system, cycle-accurate simulators. In Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 40, pages 249–261, Washington, DC, USA, 2007. IEEE Computer Society.

[9] Thomas M. Conte, Mary Ann Hirsch, and Kishore N. Menezes. Reducing state loss for effective trace sampling of superscalar processors. In Proceedings of the 1996 International Conference on Computer Design, VLSI in Computers and Processors, ICCD ’96, pages 468–477, Washington, DC, USA, 1996. IEEE Computer Society.

[10] M. Danek, L. Kafka, L. Kohout, and J. Sykora. Instruction set extensions for multi-threading in leon3. In Design and Diagnostics of Electronic Circuits and Systems (DDECS), 2010 IEEE 13th International Symposium on, pages 237–242, april 2010.

[11] M. Daněk, L. Kafka, L. Kohout, J. Sýkora, and R. Bartosinski. UTLEON3: Exploring Fine-Grain Multi-Threading in FPGAs. Circuits and Systems. Springer, November 2012.

[12] Yves Denneulin. Mescal.

[13] Lieven Eeckhout, Sebastien Nussbaum, James E. Smith, and Koen De Bosschere. Statistical simulation: Adding efficiency to the computer designer’s toolbox. IEEE Micro, 23:26–38, September 2003.

[14] M. Ekman and P. Stenstrom. Enhancing multiprocessor architecture simulation speed using matched-pair comparison. In Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software, 2005, ISPASS ’05, pages 89–99, Washington, DC, USA, 2005. IEEE Computer Society.
[15] Cagkan Erbas, Selin C. Erbas, and Andy D. Pimentel. A multiobjective optimization model for exploring multiprocessor mappings of process networks. In Proceedings of the 1st IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis, CODES+ISSS '03, pages 182–187, New York, NY, USA, 2003. ACM.

[16] Cagkan Erbas, Andy D. Pimentel, Mark Thompson, and Simon Polstra. A framework for system-level modeling and simulation of embedded systems architectures. EURASIP J. Embedded Syst., 2007:2–2, January 2007.

[17] Davy Genbrugge and Lieven Eeckhout. Chip multiprocessor design space exploration through statistical simulation. IEEE Transactions on Computers, 58:1668–1681, 2009.

[18] Davy Genbrugge, Stijn Eyerman, and Lieven Eeckhout. Interval simulation: Raising the level of abstraction in architectural simulation. In HPCA, pages 1–12, 2010.

[19] Clay Hughes and Tao Li. Accelerating multi-core processor design space evaluation using automatic multi-threaded workload synthesis. In 2008 IEEE International Symposium on Workload Characterization, pages 163–172. IEEE, October 2008.

[20] V. S. Iyengar and Trevillyan. Evaluation and Generation of Reduced Traces for Benchmarks. Technical Report RC20610, IBM T. J. Watson, October 1996.

[21] Stanley Jaddoe and Andy D. Pimentel. Signature-based calibration of analytical system-level performance models. In Proceedings of the 8th international workshop on Embedded Computer Systems: Architectures, Modeling, and Simulation, SAMOS ’08, pages 268–278, Berlin, Heidelberg, 2008. Springer-Verlag.

[22] Chris Jesshope. A model for the design and programming of multi-cores. Advances in Parallel Computing, High Performance Computing and Grids in Action(16):37–55, 2008.

[23] Chris Jesshope, Mike Lankamp, and Li Zhang. The implementation of an svp many-core processor and the evaluation of its memory architecture. SIGARCH Comput. Archit. News, 37:38–45, July 2009.

[24] Chris R. Jesshope. Microgrids - the exploitation of massive on-chip concurrency. In Lucio Grandinetti, editor, High Performance Computing Workshop, volume 14 of Advances in Parallel Computing, pages 203–223. Elsevier, 2004.

[25] Raphael ‘kena’ Poss. SL—a “quick and dirty” but working intermediate language for SVP systems. Technical Report arXiv:1208.4572v1 [cs.PL], University of Amsterdam, August 2012.

[26] Kurt Keutzer, Sharad Malik, Senior Member, A. Richard Newton, Jan M. Rabaey, and A. Sangiovanni-vincentelli. System-level design: Orthogonalization of concerns and platform-based design. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 19:1523–1543, 2000.

[27] Thierry Lafore, André Seznec, Erven Rohou, and François Bodin. Code cloning tracing: A “pay per trace” approach. In Proceedings of the 5th International Euro-Par Conference on Parallel Processing, Euro-Par ’99, pages 1265–1268, London, UK, UK, 1999. Springer-Verlag.
[28] Andrei Matei. Towards Adaptable Parallel Software - the Hydra Runtime for SVP Programs. November 2010.

[29] Sebastien Nussbaum and James E. Smith. Statistical simulation of symmetric multiprocessor systems. In SS ’02: Proceedings of the 35th Annual Simulation Symposium, page 89, Washington, DC, USA, 2002. IEEE Computer Society.

[30] Michael Pellauer, Muralidaran Vijayaraghavan, Michael Adler, Arvind, and Joel Emer. Quick performance models quickly: Closely-coupled partitioned simulation on fpgas. In Proceedings of the ISPASS 2008 - IEEE International Symposium on Performance Analysis of Systems and software, ISPASS ’08, pages 1–10, Washington, DC, USA, 2008. IEEE Computer Society.

[31] David A. Penry, Daniel Fay, David Hodgdon, Ryan Wells, Graham Schelle, David I. August, and Dan Connors. Exploiting parallelism and structure to accelerate the simulation of chip multi-processors. In Proc. of the Twelfth Int. Symp. on High-Performance Computer Architecture, pages 29–40, 2006.

[32] Raphael Poss, Mike Lankamp, Qiang Yang, Jian Fu, Irfan Uddin, and Chris Jesshope. MGSim - A simulation environment for multi-core research education. SAMOS, 2013. (To appear).

[33] Timothy Sherwood, Erez Perelman, Greg Hamerly, and Brad Calder. Automatically characterizing large scale program behavior. SIGARCH Comput. Archit. News, 30(5):45–57, October 2002.

[34] Irfan Uddin. High-level simulation of the Microgrid. Master’s thesis, University of Amsterdam, Amsterdam, the Netherlands, August 2009.

[35] Irfan Uddin, Chris R. Jesshope, Michiel W. van Tol, and Raphael Poss. Collecting signatures to model latency tolerance in high-level simulations of microthreaded cores. In Proceedings of the 2012 Workshop on Rapid Simulation and Performance Evaluation: Methods and Tools, RAPIDO ’12, pages 1–8, New York, NY, USA, 2012. ACM.

[36] Irfan Uddin, Raphael Poss, and Chris Jesshope. Cache-based high-level simulation of microthreaded many-core architectures. Journal of System Architecture, 2013.

[37] Irfan Uddin, Raphael Poss, and Chris Jesshope. Multiple levels of abstraction in the simulation of microthreaded many-core architectures. Simulation Modelling Practice and Theory, 2013.

[38] Irfan Uddin, Raphael Poss, and Chris Jesshope. One-IPC high-level simulation of microthreaded many-core architectures. Simulation Modelling Practice and Theory, 2013.

[39] Irfan Uddin, Raphael Poss, and Chris Jesshope. Signature-based high-level simulation of microthreaded many-core architectures. Microprocessors and Microsystems, 2013. (Submitted, but not yet reviewed).

[40] Irfan Uddin, Raphael Poss, and Chris Jesshope. Analytical-based high-level simulation of microthreaded many-core architectures. In PDP, February 2014. (Submitted, but not yet reviewed).

[41] Irfan Uddin, Michiel W. van Tol, and Chris R. Jesshope. High-level simulation of SVP many-core systems. Parallel Processing Letters, 21(4):413–438, December 2011.
[42] M. Van Biesbrouck, T. Sherwood, and B. Calder. A co-phase matrix to guide simultaneous multithreading simulation. In Proceedings of the 2004 IEEE International Symposium on Performance Analysis of Systems and Software, ISPASS ’04, pages 45–56, Washington, DC, USA, 2004. IEEE Computer Society.

[43] P. van Stralen and A. Pimentel. Scenario-based design space exploration of mpsoecs. In Computer Design (ICCD), 2010 IEEE International Conference on, pages 305–312, oct. 2010.

[44] Michiel W. van Tol and Juha Koivisto. Extending and implementing the self-adaptive virtual processor for distributed memory architectures. ArXiv e-prints, abs/1104.3876, April 2011.

[45] M.W. van Tol, C.R. Jesshope, M. Lankamp, and S. Polstra. An implementation of the sane virtual processor using posix threads. Journal of Systems Architecture, 55(3):162–169, 2009. Challenges in self-adaptive computing (Selected papers from the Aether-Morpheus 2007 workshop).

[46] J. Wawrzynek, D. Patterson, M. Oskin, Shin-Lien Lu, C. Kozyrakis, J.C. Hoe, D. Chiou, and K. Asanovic. Ramp: Research accelerator for multiple processors. Micro, IEEE, 27(2):46 –57, march-april 2007.

[47] A. Wellig and J. Zory. Framed complexity analysis in systemc for multi-level design space exploration. In Digital System Design, 2003. Proceedings. Euromicro Symposium on, pages 416 –423, sept. 2003.

[48] Thomas F. Wenisch, Roland E. Wunderlich, Michael Ferdman, Anastassia Ailamaki, Babak Falsafi, and James C. Hoe. Simflex: Statistical sampling of computer system simulation. IEEE Micro, 26(4):18–31, July 2006.

[49] Roland E. Wunderlich, Thomas F. Wenisch, Babak Falsafi, and James C. Hoe. Smarts: accelerating microarchitecture simulation via rigorous statistical sampling. SIGARCH Comput. Archit. News, 31(2):84–97, May 2003.

[50] Guang Yang. Parallel simulation in metropolis.