# A Case for Visualization-integrated System-level Design Space Exploration

Andy D. Pimentel

Computer Systems Architecture group
Informatics Institute, University of Amsterdam
Kruislaan 403, 1098 SJ, Amsterdam, The Netherlands
Email: andy@science.uva.nl

**Abstract.** Design space exploration plays an essential role in the system-level design of embedded systems. It is imperative therefore to have efficient and effective exploration tools in the early stages of design, where the design space is largest. System-level simulation frameworks that aim for early design space exploration create large volumes of simulation data in exploring alternative architectural solutions. Interpreting and drawing conclusions from these copious simulation results can be extremely cumbersome. In other domains that also struggle with interpreting large volumes of data, such as scientific computing, *data visualization* is an invaluable tool. Such visualization is often domain specific and has not become widely used in evaluating the results of computer architecture simulations. Surprisingly little research has been undertaken in the *dynamic use* of visualization to guide architectural design space exploration. In this paper, we plead for the study and development of generic methods and techniques for run-time visualization of system-level computer architecture simulations. We further explain that these techniques must be scalable and interactive, allowing designers to better explore complex (embedded system) architectures.

## 1 Introduction

Chip technology continues to advance along the path predicted by Moore without any saturation in the exponential growth of transistor density foreseen within the next five years [1]. This growth is required to satisfy the demands of many diverse computer applications and has resulted in a steady increase in the complexity of today's computer architectures. A noticeable trend illustrating this increase in complexity is the move towards architectures that exploit parallelism at multiple levels of granularity (e.g. bit-level, instruction-level, and task-level) by partitioning the system into a multitude of specialized resources supporting a given level of parallelism. In the embedded systems domain, this trend is also clearly noticeable with the emergence of *Systems on a Chip* (SoCs) – or rather Multi-Processor SoCs (MPSoCs) – that can integrate an entire parallel system onto a single chip and start to play a vital role in the embedded systems market. The complexity of these MPSoCs is aggravated by the fact that – besides offering parallel computing resources – they often have a heterogeneous architecture, consisting of components that range from fully programmable processor cores to fully dedicated hardware blocks. Programmable processor technology is used for realizing flexibility, for example to support multiple applications and future extensions, while dedicated hardware is used to optimize designs in time-critical areas and for power and cost minimization. Because of the complexity of these MPSoC architectures, it is crucial to have good tools for exploring design decisions during the early stages of design. In recent years, much work has been undertaken in design space exploration and this paper explores an area that promises to increase the productivity of designers still further.

System-level simulation frameworks that aim for early design space exploration can create large volumes of simulation data in exploring alternative architectural solutions. Interpreting and drawing conclusions from these copious simulation results can be extremely cumbersome. In other domains that also struggle with interpreting large volumes of data, such as scientific computing, *data visualization* is an invaluable tool. Such visualization is often domain specific and has, surprisingly enough, not become widely used in evaluating the results of computer architecture simulations. Here, results are usually still presented graphically as a post-mortem but very little research has been undertaken in the *dynamic use* of visualization to guide design-space explorations. In this paper, we advocate the study and development of generic methods and techniques for run-time visualization of system-level computer architecture simulations. Specifically, we focus on those simulations that target architectural design space exploration. We explain that the visualization techniques must be scalable and interactive, allowing designers to better explore complex architectures that may be heterogeneous in nature and may exploit various levels of concurrency. To summarize, rather than presenting concrete research results, this paper tries to identify a challenging new research area.

The remainder of the paper is organized as follows. The next section provides an introductionary overview of the field of system-level design space exploration. In Section 3, we observe that hardly any research is performed on run-time visualization for architecture simulations, and that this is especially true from the perspective of design space exploration. In Section 4, we therefore plead for visualization-integrated design space exploration, in which generality, scalability, and interactiveness are key ingredients. Finally, Section 5 concludes the paper.

## 2 System-level Design Space Exploration

The sheer complexity of modern (embedded) computer architectures forces designers to start with modeling and simulating system components and their interactions in the very early design stages. This is an important ingredient of *system-level design* [2, 3]. System-level models typically represent workload behavior, architecture characteristics, and the relation (e.g., mapping, hardware-software partitioning) between workload(s) and architecture. These models are deployed at a high level of abstraction, thereby minimizing the modeling effort and optimizing the simulation speed required to explore large parts of the design space. This high-level modeling allows for the early verification of a design and can provide estimates of the performance, power consumption and cost of a design.

Design space exploration (*DSE*) plays a crucial role in system-level architecture design. It is imperative therefore, to have good evaluation tools for efficiently exploring different design choices during the early design stages, where the design space is largest. Consequently, considerable research effort has been spent in the last decade on developing frameworks for system-level modeling and simulation that aim for early architectural exploration. Examples are Metropolis [4], MESH [5], Milan [6], Artemis [7, 8] and various SystemC-based [9] environments such as the work of [10]. This research has produced significant results in various disciplines of system-level modeling and simulation. With respect to application modeling, or workload modeling, much work has been performed in the area of models of computation (e.g.,[11–14]). System-level modeling and simulation of architectures and their performance constraints has been addressed by a large number of research groups (e.g., [4, 8, 15, 6, 16]). In many of these efforts, transaction-level models [17] are applied in which transactions between architecture components are modeled by atomic transfers of high-level data and/or control. Various research groups also recognize an explicit mapping step between application (workload) models and architecture models and subsequently proposed different mapping mechanisms (e.g., [8, 18, 4]).

Research on the refinement of (abstract) system-level architecture performance models to gradually disclose more implementation details is gaining interest but is still in its infancy. There are several attempts being made to address this issue, such as in the Metropolis [19], Artemis [8], and Milan frameworks [6], the work of [20], and in the context of SystemC (e.g., [10]). In [20], for example, a methodology is proposed in which architecture-independent specification models are transformed (i.e., refined) into architecture models to facilitate architectural exploration. The majority of the work in this field, however, focuses on communication refinement only (e.g., [21–23]).

Finally, different methods have been proposed for helping designers to quickly find good candidate architectures that can subsequently be further evaluated and explored by means of simulation. These methods usually apply multi-objective optimization techniques (e.g., [24–27]), or in the case of [6], symbolic analysis.

## 3 Visualization, or the lack of it

System-level simulations may exhibit vast amounts of simulation data on various characteristics (validity, performance, power consumption, reliability, etc.) for the architecture(s) under investigation. As mentioned before, interpreting and drawing the right conclusions from such copious simulation results may be extremely cumbersome. Because of exactly this reason, other domains that also struggle with interpreting massive amounts of data (or code), such as scientific computing, have embraced data (code) visualization (both at run-time and post-mortem) as a real aid for analysis and interpretation. As a result, visualization has become a research field in its own right in these domains. The same is not yet generally true for the computer architecture domain. Run-time visualization can, however, be extremely helpful to a system designer in identifying or analyzing dynamic effects that may occur during simulation and which may affect static performance but which cannot be analyzed at post-mortem. Here, one can think of, for example, synchronizations, cache behavior and coherency traffic in MPSoCs, or network contention and congestion in Network-on-Chip [28] based MPSoCs. To briefly illustrate the usefulness of run-time visualizations in the context of computer architec-

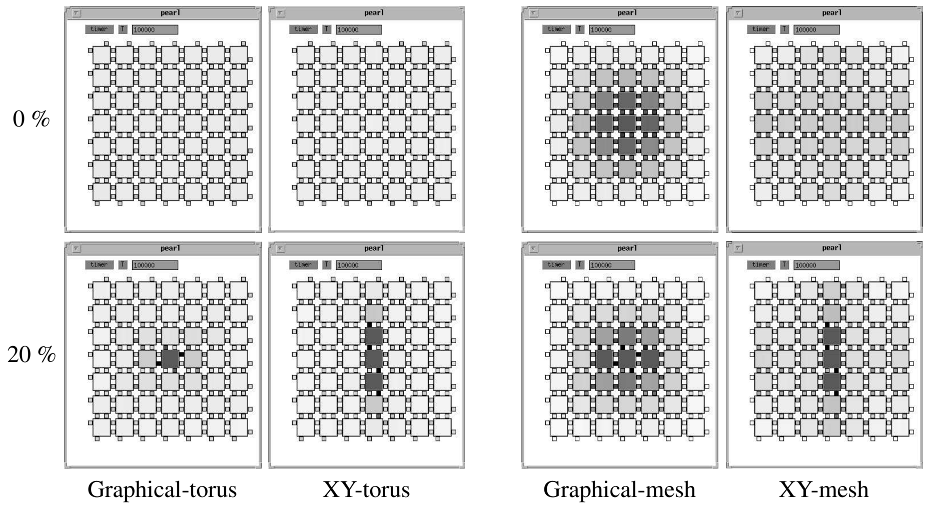

**Fig. 1.** XY and Graphical routing in torus and mesh networks

ture simulation, Figure 1 shows a case study from some early visualization work by our group [29]. The pictures show visualizations of network simulations for eight different network configurations, i.e., for two types of networks (7×7 torus and mesh networks), two types of routing mechanisms (XY routing based on dimension ordering and Graphical routing based on Bresenham's line-drawing algorithm), and two network loads (a uniform network load (= 0% hotspot) and a network load in which 20% of the traffic is directed towards a hotspot in the middle of the network). The darker cells in Figure 1 indicate a higher network contention. The visualizations show at a glance the different behaviors of, for example, the two routing mechanisms. While Graphical routing performs well in a torus, it clearly suffers from problems in a mesh network.

Despite the clear benefits, very little research is being conducted into generic methods and techniques for run-time visualization of (system-level) computer architecture simulations. Even more so, research on run-time visualization support to aid the DSE process is basically non-existing. Existing visualization work in the context of computer architecture simulation mainly focuses on visualization technology for educational purposes (e.g., [30, 31]), tightly couples visualization to one particular, often lower than system-level, architecture simulation environment (e.g., [32–34]), or only provides support for post-mortem visualization of simulation results (e.g., [35, 36]). To the best of our knowledge, only the recent research efforts of [37, 38] and especially [39] target generic visualization support in the domain of computer systems' analysis. Although the work of [37, 38] provides generic visualization support, it does so for a wide range of computer system related information which may not necessarily be applicable to computer architecture simulation, with its own domain specific requirements. Here, the data is generated dynamically and the goals are normally to refine a design space with minimum computer resources and elapsed time.

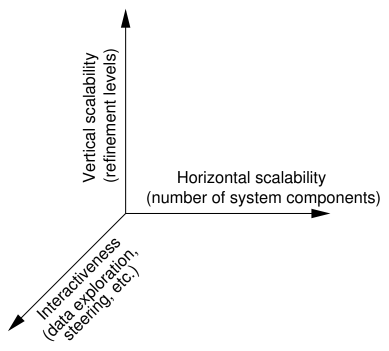

**Fig. 2.** Evaluation criteria for visualization methods for DSE.

The Vista work from [39] aims at generic support for visualization of computer architecture simulations, but it does not target system-level simulation of systems that may comprise a large number of architectural components. As will be pointed out in the next section, this will have impact on the scalability requirements of the visualization. In addition, none of the above research efforts addresses the needs for visualization from the perspective of DSE, in which different trade-offs regarding, for example, performance, power, area, etc., need to be studied. Finally, the aforementioned visualization efforts do not provide the level of interactivity that is desired for effective DSE. In this respect, we envision a visualization-integrated DSE process in which the designer is allowed to provide (interactive) feedback to the simulation environment in order to actively explore and investigate the simulation results, or even to steer the simulation (as will be discussed later on). We believe that such visualization-integrated DSE is critical for improving the effectiveness of (future) system-level DSE approaches, which will eventually lead to reductions of design times.

## 4 Visualization-integrated DSE

We plead for the development of *generic* methods and techniques to provide *scalable* and *interactive* run-time visualization of system-level computer architecture simulations for DSE. This section will shed some light on what we exactly mean with each of these requirements – generality, scalability, and interactiveness – for the run-time visualization methods. More specifically, we propose to evaluate visualization methods for DSE according to three quantifiable criteria, illustrated as three separate dimensions in Figure 2. Two of these dimensions relate to scalability, while the third refers to interactiveness. The aforementioned requirement of generality can be seen as a fourth criterion, but this one is less easy to quantify.

### 4.1 Generic visualization support

To optimize re-use of visualization building blocks, it must be identified what types of generic visualization building blocks are required to compose run-time visualizations for a wide range of computer architecture simulations. Because characteristics other than just performance, like power consumption and even cost, also are major design goals in the embedded computing domain, it should be taken into account that a visualization can be applied to different dimensions of the data produced. That is, visualization building blocks must allow for effective employment in the context of performance analysis, the analysis of power consumption, etc. The choice of visualization building blocks is also influenced by the type of data values to be visualized. In this respect, a simple classification of two types of values can be made [29]: *snapshot* values that relate to a particular moment in simulation time (e.g., the signaling of a cache hit/miss) and *integrated* values that relate to the simulation over some time interval (e.g., the cache hitrate). In many occasions, a designer will be interested in, for example, performance behavior over some period of time. This may therefore require that snapshot values from the simulator are first transformed into integrated values before visualizing them. This can be established by defining a number a basic transformations on (raw) data values. Examples of such transformation are *smooth* (smooth a value by calculating the weighted average of the old smoothed value and the current value), *history* (keep a history of value samples), and *average* (calculate the average of a history of value samples). Flexible support for composing a wide range of such transformations on data before they are visualized, is therefore needed.

Furthermore, an efficient and effective mechanism, including a generic API definition, is needed that allows the placing of *probes* in an architecture simulator to capture the events or variables that need to be visualized. The probing mechanism should minimize intrusion and pollution of the target architecture simulation code. This could be achieved by devising a descriptive language with which a designer can describe how the events/variables captured from a simulator need to be *transformed* (applying the aforementioned transformations) and *visualized* (the type of visual used). We believe that an XML-based language may be a good candidate for such visualization descriptions since XML is already used extensively to describe the structure of (system-level) models [40, 41]. By applying visualization descriptions, we establish a strict *decoupling* of the visualization and the simulator code, which minimizes intrusion and pollution of simulator code and maximizes the potentials for re-use of the composed visualizations. The latter includes both re-use within a single simulation environment as well as sharing visualization components between different simulation environments. This should considerably improve the current situation in which designers typically need to develop their own proprietary visualization support.

## 4.2 Scalable visualization

In the envisioned run-time visualization technology, the visualizations should be highly scalable, both *horizontally* and *vertically* (see also Figure 2). By horizontal scalability we mean that the visualization methods and techniques should be capable of visualizing simulations of architectures with possibly very large numbers of computational elements, memory and communication components (assuming that sufficient computational resources are provided to perform the visualization). This is an important criterion since future MPSoCs may scale up to systems that integrate hundreds to thousands of processing elements.

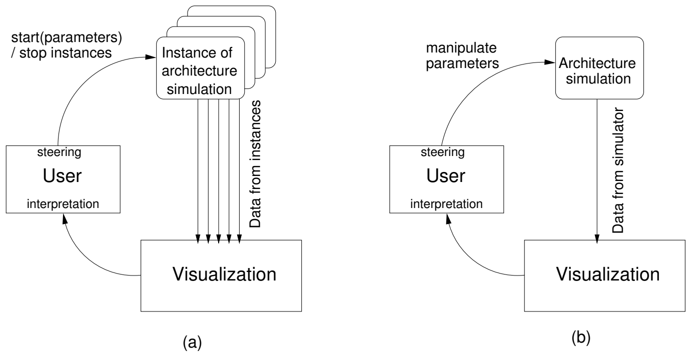

**Fig. 3.** Two types of steering in interactive visualization for DSE: (a) starting (and stopping) instances of an architectural simulation with different parameters, and (b) run-time manipulation of simulation parameters.

With vertical scalability, we refer to the capability of visualizing computer architecture simulations at multiple levels of abstraction. This is analogous to the gradual refinement of system-level architecture simulation models to exhibit more implementation details. It should, for example, be possible for visualizations to follow a similar refinement trajectory, i.e., gradually showing more detailed information. Therefore, it needs to be investigated whether or not such refinements can be formalized in transformations that move the visualization perspective through different levels of abstraction.

## 4.3 Interactive visualization

Since the objective is support for DSE, the envisioned visualization technology should not be restricted to a one-way flow of information, namely from simulation to visualization. Rather, the designer should be able to provide interactive feedback to the visualization environment, thereby allowing the designer to actively explore and investigate the simulation results and maybe even *steer* the simulation. In our view, three different types of interactive feedback can be provided by a designer. First, a designer can change the view of a visualization. This might be done by changing the way data is visualized but retaining the same abstraction level, or by changing the level of abstraction in a visualization (as discussed in the above).

The two remaining types of feedback both deal with steering the simulations. This steering is based on the idea of *computational steering* [42, 43] that is commonly applied in the field of scientific computing. In the second type of feedback, simulations can be steered – or orchestrated – by interactively starting up (and stopping) parallel instances of a simulation with different parameters, according to the findings of the designer. With the proper support for visualization, these parallel instances of a simulation and, in particular, the differences between them, aid the architect in the DSE process. This steering mechanism is illustrated in Figure 3(a).

The third type of feedback, which is illustrated in Figure 3(b), comprises the steering of a simulation by changing its parameters at run-time. For (relatively) long-running simulations, it may be too time-consuming to start up a new simulation with different parameters to reach a certain interesting point in execution again. Therefore, the potentials for manipulating a running architectural simulation (changing its parameters) also need to be investigated.

Finally, we would like to mention that both forms of interactive steering are orthogonal. So, they can complement each other in order to improve the process of DSE even further.

## 5 Conclusions

In this paper, we advocated the development of generic methods and techniques for run-time visualization of system-level computer architecture simulations. More specifically, we argued that especially visualization support to aid the process of architectural design space exploration deserves more attention. It was also explained that generality, scalability, and interactiveness are key ingredients in our envisioned visualization technology. Eventually, the proposed visualization-integrated design space exploration should lead to reductions in design times, and hopefully result in better designs. Of course, a logical next step is to give the ideas presented in this paper a more concrete form.

## References

1. International Technology Roadmap for Semiconductors: Executive summary. http://public.itrs.net/Files/2003ITRS/Home2003.htm (2003)
2. Keutzer, K., Malik, S., Newton, A., Rabaey, J., Sangiovanni-Vincentelli, A.: System level design: Orthogonalization of concerns and platform-based design. IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems **19** (2000)
3. Gajski, D.D.: System Level Design Flow: What is needed and What is not. Technical report, CECS, University of California at Irvine (2002) CECS-TR-02-33.
4. F. Balarin et al.: Metropolis: An integrated electronic system design environment. IEEE Computer **36** (2003)
5. Cassidy, A., Paul, J., Thomas, D.: Layered, multi-threaded, high-level performance design. In: Proc. of the Design, Automation and Test in Europe (DATE). (2003)
6. Mohanty, S., Prasanna, V.K.: Rapid system-level performance evaluation and optimization for application mapping onto SoC architectures. In: Proc. of the IEEE International ASIC/SOC Conference. (2002)
7. Pimentel, A.D., Lieverse, P., van der Wolf, P., Hertzberger, L.O., Deprettere, E.F.: Exploring embedded-systems architectures with Artemis. IEEE Computer **34** (2001) 57–63
8. Pimentel, A.D.: The Artemis workbench for system-level performance evaluation of embedded systems. Int. Journal of Embedded Systems (2005)
9. Grötker, T., Liao, S., Martin, G., Swan, S.: System Design with SystemC. Kluwer Academic Publishers, Dordrecht, The Netherlands (2002)
10. T. Kogel et al.: Virtual architecture mapping: A SystemC based methodology for architectural exploration of system-on-chip designs. In: Proc. of the Int. workshop on Systems, Architectures, Modeling and Simulation (SAMOS). (2003)
11. Buck, J., Ha, S., Lee, E.A., Messerschmitt, D.G.: Ptolemy: A framework for simulating and prototyping heterogeneous systems. Int. Journal of Computer Simulation **4** (1994) 155–182

12. de Kock, E.A., Essink, G., Smits, W.J.M., van der Wolf, P., Brunel, J.Y., Kruijtzer, W.M., Lieverse, P., Vissers, K.A.: Yapi: Application modeling for signal processing systems. In: Proc. of the Design Automation Conference (DAC). (2000) 402–405
13. Stefanov, T., Kienhuis, B., Deprettere, E.F.: Algorithmic transformation techniques for efficient exploration of alternative application instances. In: Proc. of the 10th Int. Symposium on Hardware/Software Codesign (CODES'02). (2002) 7–12
14. Turjan, A., Kienhuis, B., Deprettere, E.F.: Translating affine nested loop programs to process networks. In: Proc. of the Int. Conf. on Compilers, Architectures and Synthesis for Embedded Systems (CASES). (2004)
15. Mihal, A., Kulkarni, C., Sauer, C., Vissers, K., Moskewicz, M., Tsai, M., Shah, N., Weber, S., Jin, Y., Keutzer, K., Malik, S.: Developing architectural platforms: A disciplined approach. IEEE Design and Test of Computers **19** (2002) 6–16
16. Lahiri, K., Raghunathan, A., Dey, S.: System-level performance analysis for designing on-chip communication architectures. IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems **20** (2001) 768–783
17. Cai, L., Gajski, D.: Transaction level modeling: An overview. In: Proc. of CODES-ISSS. (2003) 19–24
18. Živković, V., van der Wolf, P., Deprettere, E.F., de Kock, E.A.: Design space exploration of streaming multiprocessor architectures. In: Proc. of the IEEE Workshop on Signal Processing Systems (SiPS). (2002)
19. Densmore, D., Rekhi, S., Sangiovanni-Vincentelli, A.: Microarchitecture development via Metropolis successive platform refinement. In: Proc. of the Design, Automation and Test in Europe (DATE). (2004)
20. Peng, J., Abdi, S., Gajski, D.: Automatic model refinement for fast architecture exploration. In: Proc. of the Int. Conf. on VLSI Design. (2002) 332–337
21. Abdi, S., Shin, D., Gajski, D.: Automatic communication refinement for system level design. In: Proc. of the Design Automation Conference (DAC). (2003) 300–305
22. Lieverse, P., van der Wolf, P., Deprettere, E.F.: A trace transformation technique for communication refinement. In: Proc. of the 9th Int. Symposium on Hardware/Software Codesign (CODES). (2001) 134–139
23. Nicolescu, G., Yoo, S., Jerraya, A.A.: Mixed-level cosimulation for fine gradual refinement of communication in SoC design. In: Proc. of the Int. Conference on Design, Automation and Test in Europe (DATE). (2001)
24. Haubelt, C., Mostaghim, S., Slomka, F., Teich, J., Tyagi, A.: Hierarchical synthesis of embedded systems using evolutionary algorithms. In: Evolutionary Algorithms for Embedded System Design. Kluwer Academic Publishers (2002)
25. Palesi, M., Givargis, T.: Multi-objective design space exploration using genetic algorithms. In: Proc. of the 10th Int. Symposium on Hardware/Software Codesign (CODES). (2002)
26. Thiele, L., Chakraborty, S., Gries, M., Künzli, S.: A framework for evaluating design tradeoffs in packet processing architectures. In: Proc. of the ACM/IEEE Design Automation Conference (DAC). (2002)
27. Erbas, C., Erbas, S.C., Pimentel, A.D.: A multiobjective optimization model for exploring multiprocessor mappings of process networks. In: Proc. of the IEEE/ACM CODES+ISSS Conference. (2003)
28. Benini, L., Micheli, G.D.: Networks on chips: A new SoC paradigm. IEEE Computer **35** (2002) 70–80
29. Kok, H.C., Pimentel, A.D., Hertzberger, L.O.: Runtime visualization of computer architecture simulations. In: Proc. of the Workshop on Performance Analysis and its Impact on Design (in conjunction with ISCA '97). (1997) 15–24

30. Marwedel, P., Sirocic, B.: Multimedia components for the visualization of dynamic behavior in computer architectures. In: Proc. of the Workshop of Computer Architecture Education (WCAE'03). (2003)
31. Yehezkel, C., Yurcik, W., Pearson, M., Armstrong, D.: Three simulator tools for teaching computer architecture: Easycpu, little man computer, and rtlsim. Journal on Educational Resources in Computing (JERIC) **1** (2001)
32. P. S. Coe et al.: A hierarchical computer architecture design and simulation environment. ACM TOMACS **8** (1998) 431–446
33. Berkbigler, K., Bush, B., Davis, K., Moss, N., Smith, S.: À la carte: A simulation framework for extreme-scale hardware architectures. In: Proc. of the IASTED International Conference on Modelling and Simulation. (2003)
34. Stolte, C., Bosch, R., Hanrahan, P., Rosenblum, M.: Visualizing application behavior on superscalar processors. In: Proc. of the Fifth IEEE Symposium on Information Visualization. (1999)
35. Fang, W., Wang, C.L., Zhu, W., Lau, F.: Pat: A postmortem object access pattern analysis and visualization tool. In: Proc of the Int. Workshop on Distributed Shared Memory on Clusters (at CCGrid 2004). (2004)
36. Hlavacs, H., Kvasnicka, D., Ueberhuber, C.W.: Clue — a tool for cluster evaluation. In: Distributed and Parallel Systems (DAPSYS). (2000) 61–64
37. Bosch, R., Stolte, C., Tang, D., Gerth, J., Rosenblum, M., Hanrahan, P.: Rivet: A flexible environment for computer systems visualization. Computer Graphics **34** (2000)
38. Bosch, R.P.: Using Visualization to Understand the Behavior of Computer Systems. PhD thesis, Stanford University (2001)
39. Mihalik, A.: Vista: A visualization tool for computer architects. Master's thesis, Massachusetts Institute of Technology (2004)
40. Lee, E.A., Neuendorffer, S.: MoML - a Modeling Markup Language in XML, version 0.4. Technical Report UCB/ERL M00/8, Electronics Research Lab, University of California, Berkeley (2000)
41. Coffland, J.E., Pimentel, A.D.: A software framework for efficient system-level performance evaluation of embedded systems. In: Proc. of the ACM Symposium on Applied Computing (SAC '03). (2003) 666–671
42. Mulder, J., van Wijk, J., van Liere, R.: A survey for computational steering environments. Future Generation Computer Systems **15** (1999)
43. van Liere, R., Mulder, J., van Wijk, J.: Computational steering. Future Generation Computer Systems **12** (1997)