Design Guidelines and Visualization Support for Developing Parallel Real-Time Applications

Lars Lundberg
Department of Computer Science and Business Administration, University College of Karlskrona/Ronneby,
Soft Center, S-372 25 Ronneby, Sweden, Phone: +46-(0)457-787 30, Fax: +46-(0)457-271 25
Lars.Lundberg@ide.hk-r.se

Proposal for a 2+2 year research project within PAMP

1. Short summary

Large telecommunication applications running in real-time environments need to have high, predictable and scalable performance. Parallel (multithreaded) programs and multiprocessors is one way to achieve this. However, the experience of writing parallel programs is still limited and a lot of issues need to be addressed. Moreover, tools and methods are needed to predict and visualize the performance of a parallel program when using a multiprocessor, e.g. how will performance scale when the number of processors is increased, and where are the serialization bottlenecks.

As a part of this project we will extend an early prototype of a performance prediction and visualization tool. By studying a set of real telecommunication applications, provided by our industrial partner Ericsson Software Technology, we will also obtain a set of guidelines for writing parallel programs with high, predictable and scalable performance. These guidelines will also consider maintainability and reusability aspects.

The collaboration with the academic partners within PAMP will make it possible to use the same tools for performing the research. Our research will be at the application level, and will thus complement the research at the operating system and hardware level performed by other nodes in PAMP. Although Ericsson Software Technology is our main industrial partner, we hope that we will be able to use applications from the other industrial partners as well.

2. Problem statement

Many real-time applications, especially in the telecommunication domain, are *transaction oriented*. Failure to meet the performance requirements, e.g. due to transient overloads, often results in loss of value for the company or organization operating the real-time system, i.e. *the quality of the service provided by the real-time system to the company operating it may seriously deteriorate if the timing requirements are not met*. One example is that telephone network operators may lose information about billable calls, resulting in loss of money. The real-time demands on transaction oriented applications are increasing, e.g. there is a need for including *hot billing* in the billing systems. This means that the cost for making a phone call is calculated in real-time, making it possible to disconnect subscribers which have exceeded there cost limit. Consequently, there is a need for *high and predictable performance*.

Due to the rapid growth of the telecommunication sector, most real-time systems are expected to scale up in the near future, e.g. network operators want to be able to handle more subscribers. Due to their size and complexity, many real-time telecommunication systems are extremely costly to develop and maintain. Consequently, one would like to increase the capacity of such systems in a modular fashion (*scalable performance*), i.e. one would not like to redesign large parts of the system when the performance requirements increase.

In the near future, we expect to see multiprocessors that share the same address space, so called *symmetric multiprocessor systems* (SMPs). Besides the performance and cost advantages of this emerging technology, one can (at least in theory) incrementally increase throughput and decrease response time by simply adding more processors in a modular fashion. In order to reduce the cost for developing and maintaining large real-time systems, one is often forced to use standard system software, such as standard operating and database systems. Consequently, the SMP platform consists not only of the multiprocessor hardware; it also includes system software.

In order to benefit from multiprocessors, the application program must be structured in a parallel way. One way of doing this is to write multithreaded programs using Sun Solaris threads. Today, the experience of writing parallel applications is quite limited, especially if high, predictable and scalable performance is the goal. Moreover, preliminary results indicate that there can be a conflict between maintainable and reusable software design on the one hand and high, predictable and scalable performance on the other hand [5].

Consequently, there is a need for tools and guidelines for developing parallel real-time systems with high, predictable, and scalable performance. Due to the considerable costs for developing and maintaining large systems, there is also a need for understanding the trade-offs between performance and reusability/maintainability.

3. Main ideas

In this project we will extend an early prototype [1] of a visualization and performance prediction tool for parallel applications written using Sun Solaris threads. The tool consists of three major parts, the Recorder, the Simulator, and the Visualizer (see figure 1). The developer writes the multithreaded program, compiles it, and an executable binary file is obtained. After that, the program is executed on a uni-processor, with the *Recorder* placed *between* the program

and the standard thread library. Each time the program uses the thread routines, the call passes through the *Recorder* which records information about the call. The *Recorder* then calls the original routine in the thread library. When the execution of the program finishes all the collected information is stored in a file, the *recorded information*.

The *Simulator* simulates a multiprocessor execution. The main input for the simulator is the *recorded information*. The simulator also takes as input the hardware configuration and scheduling policies. The output from the simulator is information describing the simulated execution. Using the *Visualizer* the simulated parallel execution of the program can be inspected. When visualizing a simulated execution, it is possible for the developer to click on a certain interesting event, get the source code displayed, and the line making the call that generated the event highlighted. Hitherto, the tool has been used for small parallel programs, but not for large real-time industrial applications. In this project we will use the tool on a large application, provided to us by Ericsson Software Technology.

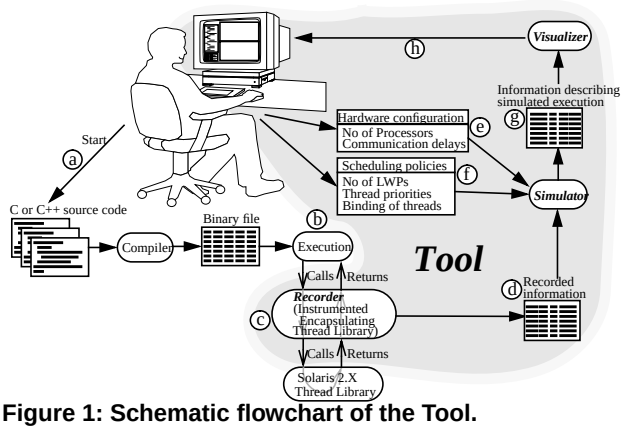

Figure 1: Schematic flowchart of the Tool.

Our tool does not capture the impact of I/O, and the memory systems (e.g. cache hit ratio). To address this problem, we need to monitor the system at a very detailed level, e.g. by using simulation tools. Unfortunately, the detailed simulation tools available today cannot handle large parallel applications, mainly due to efficiency reasons. A group at SICS has (with some help from Chalmers and Karlskrona/Ronneby) developed a highly efficient simulation environment, where parallel applications can be run [8]. In addition, detailed execution statistics can be obtained from the simulation environment. We will use this tool in our research.

In order to obtain the guidelines for developing parallel programs with high, predictable and scalable performance we are going to perform a number of case studies. In each case study we will look at a parallel application. There are at least 3-4 relevant applications at Ericsson Software Technology, and we also hope that we will be able to look at applications from other industrial partners in PAMP. For each application we will work in the following way:

1. *Identify performance bottlenecks*
   In order to identify the bottlenecks we will use the simulation tool developed at SICS (and to some extent Chalmers and Karlskrona/Ronneby) [8]. We may also acquire the source code for Solaris.
2. *Develop a set of guidelines for avoiding these bottlenecks*
3. *Redesign the application according to the guidelines*
4. *Evaluate the performance* (depending on the outcome of this steps 1 to 4 may be reiterated)
5. *Access the impact the guidelines will have on reusability and maintainability.*

As described in the research plan for PAMP, each PAMP-project contains four phases: (1) application analysis, (2) concept development, (3) evaluation, and (4) refinement/generalization. Step one above clearly corresponds to the first phase, steps two and three correspond to phase two, and steps four and five correspond to phase three. Having developed guidelines for a number of different applications, we will merge these into a more general set of guidelines, i.e. we will perform phase four (generalization) for a bunch of applications. Consequently, phases one, two and three are reiterated for each application, whereas phase four is performed for a bunch of applications (see section 5).

The assessment of the impact our guidelines have on maintainability and reusability, i.e. step five above, will be done together with researchers from the ARCS research group at Karlskrona/Ronneby. This group is headed by Dr. Jan Bosch and the group has been working on reusability and maintainability aspects for a number of years.

Finally, our main research vehicle is an eight processor SMP that we will use as a test bench to do performance measurements, as a platform to run simulations on, and as platform for parallel program development.

4. Expected results and impact

We expect to obtain a set of guidelines on how to design parallel real-time applications with high, predictable and scalable performance. The guidelines will also consider reusability and maintainability aspects, We expect that the guidelines can be grouped into three categories:

- General guidelines which should be used for all parallel programs developed for SMPs

- Guidelines for applications where performance is extremely important, i.e. systems where one is prepared to compromise reusability and maintainability in order to obtain high performance.
- Guidelines for applications where reusability and maintainability are just as important as performance.

We also expect to obtain a tool for visualizing and predicting the performance of parallel programs. The tool will make it possible to predict the execution time of and identify performance bottlenecks in parallel applications. The tool will make it possible to see how a parallel application will behave when using a certain multiprocessor, defined by a number of parameters. The parameters describing the multiprocessor include hardware aspects, e.g. the number of processors, and operating system aspects, e.g. binding threads to processors or not. In order to help the programmer to remove bottlenecks, there are hooks from the graphical display showing the behaviour down to the source code. The first version of the tool will be developed for multithreaded Solaris programs.The final version will support other environments as well, e.g. Windows NT, making it possible to do performance tuning across SMP platforms.

5. Project plan for 1998-2002

Two Ph.D. students - Magnus Broberg and Daniel Häggander - will perform most of the work in this project. We present individual plans for Magnus and Daniel:

Magnus Broberg (performance prediction and visualization tool):
Below is a (reasonably) detailed plan for Magnus. We will now explain how this plan relates to the four phases of a PAMP-project ((1) application analysis, (2) concept development, (3) evaluation, and (4) refinement/generalization).

The analysis of the main application - the Billing Gateway (BGw) - has already been done and it is clear that the tool must be extended in order to handle parallel real-time applications such as BGw, i.e. we will start with the concept development phase during the fall of 1998. During the spring of 1999 we will go into the evaluation phase, and we will enter the refinement phase during the fall of 1999. This will conclude the first part of this project and Magnus will get his Lic. The second phase of the project will reiterate the four phases and we expect that the analysis will show that there is a need for extending the tool to other environments, e.g. Windows NT.

Activity 980701-981231: *Develop techniques to trace and model I/O system calls in multithreaded applications.*
In the current version of our tool, I/O cannot be modeled for the following reason. The tool relies on a trace of a uniprocessor execution when it does a prediction. Our current tracing technique requires that all user-level threads are interleaved on one single kernel-thread. Unfortunately, the operating system may create several kernel-threads when handling I/O requests. Therefore, we must extend the tracing technique in the tool to handle several kernel-threads. Ericsson will provide a case application (BGw).
Milestone: *A report describing our new approach to handle I/O in performance predictions.*

Activity 990101-990630: *Evaluate the tool when developing a (reasonably) large real-time system.*
We will supply the tool to a group of 4-5 students which will be developing a parallel real-time system as a part of their study program [9]. The project corresponds to 10 study points and it will be done together with Ericsson.
Milestone: *A report describing the experience of the evaluation*
First year checkpoint: *Here we will provide the reports defined in the ARTES project process.*
Annual progress presentation according ARTES project process.
Since a part of the first year checkpoint is to define the rest of the project, the plan for 990701-020630 is preliminary.

Activity 990701-991231: *Extend the tool to model more details in the system architecture, e.g. the implications of decreased cache hit-ratio when threads are migrated and the implications of I/O contention.*
In order to do these extensions of the tool we need to monitor the system at a very detailed level. We plan to do this by using the simulation environment developed at SICS [8] (extended by Chalmers and Karlskrona/Ronneby).
Milestone: *Licentiate degree for Magnus.*

Activity 000101-020630: *Extend the tool to handle execution environments other than Sun Solaris.*
When Magnus has finished his Lic. we expect to have a useful and well functioning tool for Solaris. At this time we would like to generalize our approach to cover also other execution environments, such as Windows-N/T, Java threads, Ada tasks etc. One interesting question is how much of the tool and methodology that are specific for Solaris and how much that can be applied to other execution environments.
Milestone: *Final report and doctoral degree for Magnus.*

Daniel Häggander (Guidelines):
Below is a (reasonably) detailed plan for Daniel. We will now explain how this plan relates to the four phases of a PAMP-project ((1) application analysis, (2) concept development, (3) evaluation, and (4) refinement/generalization).

Within this project we will reiterate the first three phases for each application (see section 3). The generalization phase (phase four), which is done during the spring of 2000 (see below), ends the first part of the project and Daniel will get his Lic. The last two years will start with a new application analysis. We expect that this analysis will show that there is a need for considering other environments, e.g. Windows NT.

Activity 980701-981231: *Evaluate different software architectures for a Fraud Detection system.*
The Fraud Detection system is a transaction oriented parallel real-time application developed by Ericsson The most interesting aspect of this system is that it uses a large database system and that it operates under real-time constraints. We expect that the guidelines will mostly concern the interaction between the application and the database system.
Milestone: *Report on the evaluation and a set of design guidelines for this application.*

Activity 990101-990630: *Evaluate different software architectures for a second parallel real-time system.*
We have not yet selected this application, one alternative is the BGw. There are however a number of interesting alternatives. We would like to select an application which includes a lot of network communication, since this is an important factor in real-time applications particularly for the telecommunication domain.
Milestone: *Report on the evaluation and a set of design guidelines for this application.*
First year checkpoint: *Here we will provide the reports defined in the ARTES project process.*
Annual progress presentation according ARTES project process.
Since a part of the first year checkpoint is to define the rest of the project, the plan for 990701-020630 is preliminary.

Activity 990701-991231: *Evaluate different software architectures for a third parallel real-time system.*
We would like to select an application which can be structured as a mixture of traditional operating system processes and threads. Having this kind of structure makes it possible to increase performance by splitting the application on a number of SMPs and it also improves reliability since processes can be restarted on other SMPs in case of failure. One interesting question is if there is a significant performance loss when using a number of operating system processes compared to one multithreaded program.
Milestone: *Report on the evaluation and a set of design guidelines for this application*

Activity 000101-000630: *Define a set of guidelines for designing parallel real-time systems.*
At this time we plan to merge our experience from the different applications into a set of guidelines for developing parallel real-time applications.
Milestone: *Licentiate degree for Daniel.*
Second year checkpoint: *Here we will provide the reports defined in the ARTES project process.*
Annual progress presentation according ARTES project process.

Activity 000701-020630: *Extend the guidelines to execution environments other than Sun Solaris.*
When Daniel has finished his Lic. we expect to have a useful set of guidelines. At this time we would like to extend and generalize the guidelines by considering other executing platforms than Sun Solaris, e.g. Windows NT.
Milestone: *Final report and doctoral degree for Daniel (the degree may be delayed until 021231).*

6. Preliminary budget 980701-020630

Name	Title/function	Activity level	Requested Funding	Cost
Lars Lundberg	Ph.D.	50%	10%	27,000 SEK
Håkan Grahn	Ph.D	50%	20%	49,000 SEK
Magnus Broberg	Ph.D. student	80%	80%	117,000 SEK
Daniel Häggander	Ph.D. student	80%	80%	117,000 SEK
Publication costs (including conference trips), and travelling to other PAMP nodes				35,000 SEK
Expenses, computer equipment, and software				15,000 SEK
University overhead (23% adminstrativ avtaxering)				108,000 SEK
Total cost 1998 including overhead (6 months)				***468,000 SEK***
Lars Lundberg	Ph.D.	50%	10%	56,000 SEK
Håkan Grahn	Ph.D	50%	20%	102,000 SEK
Magnus Broberg	Ph.D. student	80%	80%	239,000 SEK
Daniel Häggander	Ph.D. student	80%	80%	239,000 SEK
Publication costs (including conference trips), and travelling to other PAMP nodes				65,000 SEK
Expenses, computer equipment, and software				25,000 SEK
University overhead (23% adminstrativ avtaxering)				217,000 SEK
Total cost 1999 including overhead (12 months)				***943,000 SEK***

Lars Lundberg	Ph.D.	50%	10%	57,000 SEK
Håkan Grahn	Ph.D	50%	20%	106,000 SEK
Magnus Broberg	Ph.D. student	80%	80%	246,000 SEK
Daniel Häggander	Ph.D. student	80%	80%	246,000 SEK
Publication costs (including conference trips), and travelling to other PAMP nodes				55,000 SEK
Expenses, computer equipment, and software				25,000 SEK
University overhead (23% adminstrativ avtaxering)				220,000 SEK
Total cost 2000 including overhead (12 months)				***955,000 SEK***

Lars Lundberg	Ph.D.	50%	10%	58,000 SEK
Håkan Grahn	Ph.D	50%	20%	110,000 SEK
Magnus Broberg	Ph.D. student	80%	80%	254,000 SEK
Daniel Häggander	Ph.D. student	80%	80%	254,000 SEK
Publication costs (including conference trips), and travelling to other PAMP nodes				55,000 SEK
Expenses, computer equipment, and software				25,000 SEK
University overhead (23% adminstrativ avtaxering)				226,000 SEK
Total cost 2001 including overhead (12 months)				***982,000 SEK***

Lars Lundberg	Ph.D.	50%	10%	30,000 SEK
Håkan Grahn	Ph.D	50%	20%	56,000 SEK
Magnus Broberg	Ph.D. student	80%	80%	132,000 SEK
Daniel Häggander	Ph.D. student	80%	80%	132,000 SEK
Publication costs (including conference trips), and travelling to other PAMP nodes				25,000 SEK
Expenses, computer equipment, and software				15,000 SEK
University overhead (23% adminstrativ avtaxering)				116,000 SEK
Total cost 2002 including overhead (6 months)				***506,000 SEK***

7. Related research

There are guidelines for designing software with high performance, e.g. the “Principles for creating responsive Software” in [12]. However, these guidelines consider neither the specific problems of parallel programs and multiprocessor systems nor maintainability and reusability aspects. Guidelines for designing maintainable and reusable software often take the form of design patterns, i.e. examples of best practice solutions [3]. In most cases, such design patterns do not consider the trade-offs between performance and maintainability/reusability. Experience shows that parallel real-time applications developed according to such guidelines may suffer from poor performance [5].

We are not the only ones that have realized that there is a need for doing trade-offs between different architectural aspects, e.g. performance versus maintainability/reusability. The ATA project (Architectural Tradeoff Analysis method) at Carnegie Mellon University also attacks this problem (see http://www.sei.cmu.edu/activities/ata/ata_init.html). However, their approach do not focus on parallel real-time systems.

Different tools for visualizing the behaviour of, and thus the bottlenecks in, parallel programs have been developed [2, 4, 6, 7, 10, 11, 13]. Some performance tools show the behaviour of one particular monitored multiprocessor execution of the parallel program [2, 4, 6, 7]. If we monitor the execution on a multiprocessor with four processors such tools make it possible to detect bottlenecks which are present when using four processors. The problem with this approach is that one cannot detect bottlenecks which appear when using another number of processors. There are a number of tools which make it possible to visualize the (predicted) behaviour of a parallel program using any number of processors. However, these tools are either developed for message passing systems [10] or for non-standard programming environments [11, 13], thus making it impossible to use them for industrial applications.

8. Relation to the profile

The objective of PAMP is to provide design methods to make it possible to use symmetric multiprocessor technology to meet the performance demands of emerging high-performance real-time applications. PAMP is intended to develop systems design methods that shall drastically shorten the development time. Examples of research issues within the PAMP profile are: methods for parallelization of applications, and methods for performance predictions of multiprocessor systems.

The guidelines and tool produced in this project fit the PAMP profile. The tool will increase the productivity of the developer as well as the performance of the parallel real-time system. The guidelines will make it possible to parallelize an application in order to obtain high, predictable and scalable performance. The guidelines will also take the development time into consideration, since we will consider maintainability and reusability as well as performance.

9. Industrial relevance

The experience of writing parallel real-time applications is very limited an there is a need for guidelines and tools that will help the developer. Examples of questions that have been asked by our industrial partner Ericsson is:

- "How fast will the application run if the number of processors is increased?"
- "How can the application be restructured in order to achieve better speedup?"
- "What guidelines should be followed when writing parallel applications?"

The first two questions could be handled by the performance prediction and visualization tool, and the guidelines developed in this project are a direct answer to the last question. It is of particular industrial relevance that our guidelines will not only consider high capacity; they will also consider the important issues of system development cost (maintainability and reusability).

10. Relation to other SSF programmes

The Personal Computing and Communication (PCC) programme aims at providing "Mobile multimedia services to all at the same price level as fixed telephony today". This is obviously a challenging goal that will require high performance processing. One of the most promising ways to obtain high performance is parallel processing using SMPs, which is being studied in PAMP in general and this project in particular. The National Graduate School in Scientific Computing is also sponsored by SSF. High performance is a key factor in scientific computing and SMPs are often being used to obtain high performance. The first prototype of our performance prediction and visualization tool was evaluated on a set of scientific programs [1], and we believe that some of the guidelines obtain in this project will be applicable to scientific programs. The Excellence Center in Computer Science and Systems Engineering (ECSEL) aims at reducing the development time and cost for new products. Reduced development cost and time is also a major concern within PAMP. However, PAMP focuses on the particular problems of using SMPs in real-time applications.

11. Context

11.1 The research group

The research group consists of: Lars Lundberg, Håkan Grahn, Magnus Broberg, and Daniel Häggander.

11.2 Complementary activities and funding

We are currently seeking funding from KK-stiftelsen for a project called RiSE concerning Software Architectures. This is a joint research project between Dr. Jan Bosch, Dr. Håkan Grahn and Dr. Lars Lundberg. If the application is approved, we will recruit a number of Ph.D. students. The main focus for the project is the interaction and conflicts between different non-functional requirements, e.g. maintainability and performance. Consequently, cooperation between the RiSE project and the work done by Daniel Häggander in this project would be interesting and relevant.

11.3 Research cooperation

Professor Per Stenström (pers@ce.chalmers.se) acts as Ph.D. supervisor and examiner for Magnus Broberg. Daniel Häggander is currently being enrolled as a Ph.D. student at Uppsala and although this have not yet created a lot of cooperation we expect that it will in the very near future. If things turn out according to our plans, professor Hans Hansson (hans.hansson@mdh.se) will act as Ph.D. supervisor and examiner for Daniel.

The groups at Chalmers, SICS, and Karlskrona/Ronneby have jointly worked on a common simulation environment that enables us to run an operating system on top of a simulated model of a multiprocessor [8]. The environment will be a research vehicle when developing new performance prediction methodologies.

Consequently, we have research cooperation with Chalmers and SICS. When Daniel has been enrolled as a Ph.D. student at Uppsala we believe that this will create a lot of contacts and cooperation with the Uppsala node.

Besides the research cooperation with the PAMP nodes, we will work with Jan Bosch and his group at Karlskrona/Ronneby. Dr. Jan Bosch leads a research group in software engineering with focus on object-oriented software architectures (see http://www.ide.hk-r.se/~bosch/ARCS.html). He will serve as a local expert on software development methods, particularly maintainability and reusability aspects.

11.4 Industrial cooperation

Our industrial partner is Ericsson Software Technology and we are currently working together with two business units. One unit responsible for a parallel application called BGw (Billing Gateway) and one business unit responsible for a Fraud Detection system, which is also a parallel application intended for multiprocessor systems. Currently, a number of similar systems are emerging at Ericsson Software Technology. The contact persons are: Mikael Roos, Ericsson Software Technology AB, Soft Center, S-372 25 Ronneby, Sweden, phone: 0457-77522, mikael.roos@epk.ericsson.se (BGw). Kennet Kjellsson, Ericsson Software Technology AB, Box 518, S-371 23 Karlskrona, Sweden, phone: 0455-395248, kennet.kjellsson@epk.ericsson.se (Fraud Detection).

12. References

[1] Magnus Broberg, Lars Lundberg, and Håkan Grahn, “VPPB - Visualization and Performance Prediction Tool for Multithreaded Solaris Programs”, in Proceedings of the 12th International Parallel Processing Symposium, March-April 1998.

[2] H. Chen, B. Shirazi, J. Yeh, H. Youn, and S. Thrane, “A Visualization Tool for Display and Interpretation of SISAL Programs”, Proc. ISCA Int’l Conf. on Parallel and Distributed Computing Systems, Oct. 1994.

[3] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides, “Design Patterns - Elements of Reusable Object-Oriented Software”, Addison-Wesley, 1995.

[4] J. K. Hollingsworth and B. P. Miller, “Dynamic Control of Performance Monitoring on Large Scale Parallel Systems”, Proc. Int’l Conf. on Supercomputing, pp. 185-194, Jul. 1993.

[5] Daniel Häggander, and Lars Lundberg, “Optimizing Dynamic Memory Management in a Multithreaded Application Executing on a Multiprocessor”, in Proceedings of the International Conference on Parallel Processing, August 1998.

[6] E. Kraemer and J. Stasko, “The Visualization of Parallel Systems: An Overview”, J. of Parallel and Distributed Computing, Vol. 18, pp. 105-117, 1993

[7] S. Lei and K. Zhang, “Performance Visualization of Message Passing Programs Using Relational Approach”, Proc. ISCA Int’l Conf. on Parallel and Distributed Computing Systems, pp. 740-745, Oct. 1994.

[8] Peter S. Magnusson, Fredrik Dahlgren, Håkan Grahn, Magnus Karlsson, Fredrik Larsson, Fredrik Lundholm, Anderas Moestedt, Jim Nilsson, Per Stenström, and Bengt Werner, “SimICS/sun4m: A Virtual Workstation”, in Proceedings of the 1998 USENIX Annual Technical Conference, June, 1998.

[9] Lennart Ohlsson, and Conny Johansson, “A Practice driven Approach to Software Engineering Education”, IEEE Transactions on Education, Vol. 38, No. 3, August 1995.

[10] V. Pillet, J. Laboarta, T. Cortes, and S. Girona, “PARAVER: A Tool to visualize and Analyse Parallel Code,” University of Politencia, Catalonia, CEPBA/UPC Report No. RR-95/03, Feb. 1995.

[11] S. R. Sarukkai and D. Gannon, “SIEVE: A Performance Debugging Environment for Parallel Programs,” *J. of Parallel and Distributed Computing*, Vol. 18, pp. 147-168, 1993.

[12] Connie U. Smith, “Performance Engineering of Software Systems”, Addison-Wesley, 1990.

[13] Z. Segall and L. Rudolph, “PIE: A Programming and Instrumentation Environment for Parallel Processing,” *IEEE Software*, 2(6):22-37, Nov. 1985.