Proceedings of the 1997 Winter Simulation Conference
ed. S. Andradóttir, K. J. Healy, D. H. Withers, and B. L. Nelson

EXECUTION-DRIVEN SIMULATORS FOR PARALLEL SYSTEMS DESIGN

Anand Sivasubramaniam

Department of Computer Science and Engineering
The Pennsylvania State University
University Park, Pennsylvania 16802, U.S.A.

ABSTRACT

Evaluating, analyzing and predicting the performance of a parallel system is challenging due to the complex inter-play between the application characteristics and architectural features. The overheads in a parallel system that limit its scalability have to be identified and separated in order to enable performance-conscious parallel application design and the development of high-performance parallel machines. We have developed an evaluation framework that uses a combination of experimentation, simulation and analytical modeling to quantify these parallel system overheads. At the heart of this framework is an execution-driven simulation testbed called SPASM which uses a suite of real applications as the workload. We discuss our experiences in using this simulator in a wide range of architectural projects in this paper.

1 INTRODUCTION

High Performance Computing is becoming increasingly important to scientific advancement and economic development and is at the point of significantly improving our standard of living. With the inherent limitations of sequential computing, parallel machines have been proposed as the solution for high-performance computing. Despite their promise and attractiveness to the research community, parallel machines have not been very successful in the commercial world due to two main reasons. First, their delivered performance often falls short of the projected peak performance. Second, the cost of these machines is high compared to their sequential counterparts. For the success of parallel computation, *we should build machines that bridge the gap between projected and delivered performance over a spectrum of important real-world applications in a cost-effective manner.* Performance evaluation of parallel systems plays a crucial role towards this goal.

Applications exhibit different characteristics thus imposing diverse demands on the underlying hardware, while parallel machines also come in several flavors. To find out how good a job a machine does of meeting an application's demands, we need a way of evaluating the match between an application and an architecture. Evaluating the performance of an application-architecture combination has widespread applicability in parallel systems research. The results from such an evaluation may be used to: select the best architecture platform for an application domain, select the best algorithm for solving the problem on a given hardware platform, predict the performance of an application on a larger configuration of an existing architecture, predict the performance of large application instances, identify application and architectural bottlenecks in a parallel system to suggest application restructuring and architectural enhancements, and evaluate the cost vs. performance trade-offs in important architectural design decisions. But evaluating and analyzing the performance of parallel systems pose several problems due to the complex interaction between application characteristics and architectural features.

Performance evaluation techniques have to grapple with several more degrees of freedom exhibited by parallel systems compared to their sequential counterparts. Experimentation and measurement on actual hardware, analytical modeling and simulation are three well-known performance evaluation techniques. But each technique has its own limitations. Experimentation requires the hardware to be built, analytical models often make unreasonable assumptions about the underlying system to keep the modeling tractable, and simulation requires immense resources in terms of storage and time.

In this paper, we summarize our previous and ongoing effort in developing a framework for evaluating the performance of parallel systems and using this framework to develop cost-effective platforms that meet the demands of numerous real-world applications. First, we identify performance metrics which are essential to understand the intrinsic algorithmic and architectural artifacts that impact the performance of a parallel system. Next, we outline an evaluation framework that we have developed to quantify these metrics. The framework uses all three performance evaluation techniques to alleviate their individual limitations. At the heart of this framework lies SPASM (Simulator for Parallel Architectural Scalability Measurements) which provides detailed performance profiles for applications on a range of parallel hardware platforms. This simulator helps identify, isolate and quantify the algorithmic and architectural bottlenecks in an execution, that can be used for application restructuring and to suggest architectural enhancements. In the rest of the paper, we illustrate the utility of an execution-driven simulator such as SPASM in several architectural projects.

The rest of this paper is organized as follows. In Section 2, we identify performance metrics that we require from evaluating a parallel system and discuss different evaluation techniques. Section 3 outlines our evaluation framework and the SPASM simulator. Section 4 summarizes our experience and results in using execution-driven simulators for several architectural projects. Finally, Section 5 presents concluding remarks.

2 EVALUATING PARALLEL SYSTEMS

In conducting any evaluation, we need to identify a set of performance metrics that we would like to measure and the techniques and tools that will be used to gather these metrics.

2.1 Performance Metrics

Metrics which capture the "available" compute power (MFLOPS, MIPS etc.) are often not a true indicator of the performance actually "delivered" by a parallel system. Metrics for parallel system performance evaluation should quantify this gap between available and delivered compute power since understanding application and architectural bottlenecks is crucial for application restructuring and architectural enhancements. Many performance metrics such as speedup, scaled speedup and isoefficiency, have been proposed to quantify the match between the application and architecture in a parallel system. While these metrics are useful for tracking overall performance trends, they provide little additional information about where performance is lost. Some of these metrics attempt to identify the cause (the application or the architecture) of the problem when the parallel system does not scale as expected. However, it is essential to find the individual application and architectural artifacts that lead to these bottlenecks and quantify their relative contribution towards limiting the overall scalability of the system. Traditional metrics do not help further in this regard.

Parallel system overheads may be broadly classified into a purely algorithmic component (*algorithmic overhead*), a component arising from the interaction of the application with the system software (*software interaction overhead*), and a component arising from the interaction of the application with the hardware (*hardware interaction overhead*). Algorithmic overheads arise from the inherent serial part in the application, the work-imbalance between the executing threads of control, any redundant computation that may be performed, and additional work introduced by the parallelization. Software interaction overheads such as overheads for scheduling, message-passing, and software synchronization arise due to the interaction of the application with the system software. Hardware slowdown due to network latency (the transmission time for a message in the network), network contention (the amount of time spent in the network waiting for availability of network resources), synchronization and cache coherence actions, contribute to the hardware interaction overhead. Each of these components would cause the performance to deteriorate from the available compute power (potential peak performance) of the hardware. To fully understand the scalability of a parallel system, it is important to isolate and quantify the impact of each of these components on the overall execution. In our earlier research, we have proposed the notion of an *overhead function* (Sivasubramaniam et al. 1994) that tracks the growth of a particular system overhead with respect to a specific system parameter.

2.2 Evaluation Techniques

Experimentation, analytical modeling and simulation are three well-known techniques for evaluating parallel systems. Experimentation involves implementing the application on the actual hardware and measuring its performance. Analytical models abstract hardware and application details in a parallel system and capture complex system features by simple mathematical formulae. These formulae are usually parameterized by a limited number of degrees of freedom so that the analysis is kept tractable. Simulation is a valuable technique which exploits computer resources to model and imitate the behavior of a real system in a controlled manner. Each technique has its own limitations. The amount of statistics that can be gleaned by experimentation (to quantify the overhead functions) is limited by the monitoring/instrumentation support provided by the underlying system. Additional instrumentation can sometimes perturb the evaluation. Analytical models are often criticized for the unrealism and simplifying assumptions made in expressing the complex interaction between the application and the architecture. Simulation of realistic computer systems demand considerable resources in terms of space and time.

3- THE FRAMEWORK

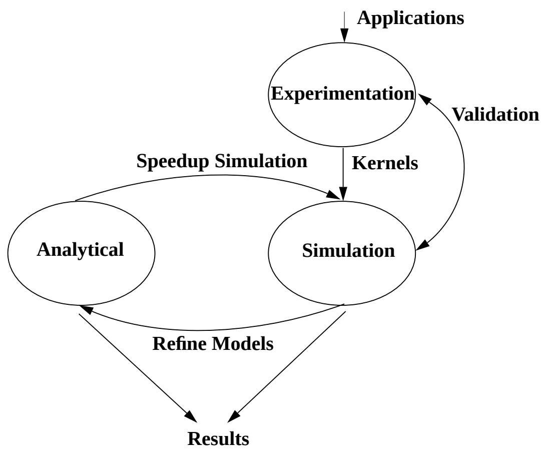

Figure 1: The Framework

We have developed an evaluation framework that uses a combination of the three techniques to avoid some of their individual drawbacks. Experimentation is used to implement real-world applications on parallel machines, to understand their behavior and extract interesting kernels (abstractions of applications that capture representative phases of the execution) that occur in them. These kernels are fed to an execution-driven simulator called SPASM which faithfully models the details of the parallel system interactions. The statistics that are drawn from the simulation are used to develop new analytical models or to validate and refine existing models. Simulation is used for detailed study of smaller systems in a non-intrusive manner. Analytical models are used to complement the simulation results to project the performance and overheads for larger systems (than those that can be simulated). When an analytical model is sufficiently validated/refined, it may be possible to use this model in the simulator itself to abstract details in the simulation to ease resource requirements. Using this approach, we have illustrated (Sivasubramaniam et al. 1995a) how the details of cache simulation and the details of interconnection network simulation may be abstracted by suitable models to gain substantial savings in the simulation time.

At the heart of our evaluation framework lies a simulation platform called SPASM which is used to identify, isolate and quantify the individual parallel system overheads.

SPASM is an execution-driven simulator written in CSIM used for simulating the execution of a parallel program on a parallel machine. As with other recent simulators the bulk of the instructions in the parallel program is executed at the speed of the native processor (SPARC in our studies) and only instructions such as LOADs/STOREs on a shared memory platform, and SENDs/RECEIVEs on a message passing platform, that may potentially involve a network access are simulated. The rationale behind this approach is that since uniprocessor architecture is getting standardized with the advent of RISC technology, we can fix most of the processor characteristics (such as instruction sets, clocks per instruction, floating point capabilities, pipelining) by using a commodity processor as the baseline for each processor in our parallel system. A detailed simulation of the processor architecture is not likely to contribute significantly to our understanding of the scalability of a parallel system. The input to the simulator is parallel applications written in C. On a message passing system, the calls (SENDs/RECEIVEs) which trap to the simulator are inserted into the application program explicitly by the programmer. On a shared memory system, a pre-processor inserts code into the application program to trap to the simulator on a shared memory reference. On both systems, the compiled assembly code is augmented with cycle counting instructions which is used to keep track of the time spent in the application program since the last trap to the simulator. Finally, the assembled binary is linked with the rest of the simulator code.

A simulation platform like SPASM allows us to vary a wide range of hardware parameters such as the number of processors, the CPU clock speed, the network topology, the bandwidth of the links in the network, the network switching delays, and the cache parameters (the block size, cache size, associativity, etc). SPASM gives a wide range of statistics that isolate and quantify the contribution of each parallel system overhead on the overall execution time of the application. Further, these overheads can be quantified for different phases of the execution that can help in performance debugging for application restructuring and for suggesting architectural enhancements.

4- PROJECTS USING THE FRAMEWORK

We have used the above framework in a wide spectrum of architectural projects that are summarized below. Even though SPASM has been used to model and study message passing systems, the projects discussed here have used only its shared memory capabilities.

4.1- Validating Abstractions

Abstracting features of parallel systems is a technique often employed in performance analysis and algorithm development. For instance, abstracting parallel machines by theoretical models like the PRAM has facilitated algorithm development and analysis. Such models try to hide hardware details from the programmer, providing a simplified view of the machine. Similarly, analytical models used in performance evaluation abstract complex system interactions with simple mathematical functions, parameterized by a limited number of degrees of freedom that are tractable. Abstractions are also useful in execution-driven simulators where details of the hardware and the application can be captured by abstract models in order to ease the demands on resource (time and space) usage in simulating large parallel systems. Some simulators already abstract details of instruction-set simulation, since such a detailed simulation is not likely to contribute significantly to the performance analysis of parallel systems.

An important question that needs to be addressed in using abstractions is their validity. Our framework serves as a convenient vehicle for evaluating the accuracy of these abstractions using real applications. In (Sivasubramaniam et al. 1995a), we have illustrated the use of the framework to evaluate the validity and use of abstractions in simulating the interconnection network and locality properties of parallel systems. An outline of the evaluation strategy and results are presented below.

For abstracting the interconnection network, we have used the recently proposed *LogP* model that incorporates the two defining characteristics of a network, namely, latency and contention. For abstracting the locality properties of a parallel system, we have modeled a private cache at each processing node in the system to capture data locality. Shared memory machines with private caches usually employ a protocol to maintain coherence. With a diverse range of cache coherence protocols, it would become very specific if our abstraction were to model any particular protocol. Further, memory references (locality) are largely dictated by application characteristics and are relatively independent of cache coherence protocols. Hence, instead of modeling any particular protocol, we have chosen to maintain the caches coherent in our abstraction but do not model the overheads associated with maintaining the coherence. Such an abstraction would represent an ideal coherent cache that captures the true inherent locality in an application. Furthermore, if our abstraction closely models the behavior of a machine with a simple cache coherent protocol, then it would even more closely model the behavior of a machine with a fancier cache coherence protocol.

We have used our simulation framework for evaluating these abstractions. We have compared the results from simulating the five applications on a machine incorporating these abstractions with the results from an exact simulation of the actual hardware. Our results show that the latency overhead modeled by LogP is fairly accurate. On the other hand, the contention overhead modeled by LogP can become pessimistic for some applications since the model does not capture communication locality. The pessimism gets amplified as we move to networks with lower connectivity. With regard to data locality, results show that our ideal cache, which does not model any coherence protocol overheads, is a good abstraction for capturing locality over the chosen range of applications.

Apart from evaluating these abstractions in the context of real applications, the isolation and quantification of parallel system overheads has helped us validate the individual parameters used in each abstraction. The simulation of the system which incorporates these two abstractions is around 250-300% faster than the simulation of the actual machine. Using a similar approach, one may also use this framework to refine existing models (like reducing the pessimism in LogP in modeling contention), or even develop new models for accurately capturing parallel system behavior.

4.2- Synthesizing Network Requirements

For building a general-purpose parallel machine, it is essential to identify and quantify the architectural requirements necessary to assure good performance over a wide range of applications. Such a synthesis of requirements from an application view-point can help us make cost vs. performance trade-offs in important architectural design decisions. Our framework provides a convenient platform to study the impact of hardware parameters on application performance and use the results to project architectural requirements. We have conducted such a study in (Sivasubramaniam et al. 1995b) towards synthesizing the network requirements of the applications mentioned earlier, and the experimental strategy along with interesting results from our study are summarized here.

To quantify link bandwidth requirements for a particular network topology, we have simulated the execution of the applications on such a topology and vary the bandwidth of the links in the network. As the bandwidth is increased, the network overheads (latency and contention) decrease, yielding a performance that is close to the ideal execution. From these results, we have arrived at link bandwidths that are needed to limit network overheads (latency and contention) to an acceptable level of the overall execution time. We have also studied the impact of the number of processors, the CPU clock speed and the application problem size on bandwidth requirements. Computation to communication ratio tends to decrease when the number of processors or the CPU clock speed is increased, making the network requirements more stringent. An increase in problem size improves the computation to communication ratio, lowering the bandwidth needed to maintain an acceptable efficiency. Using regression analysis and analytical techniques, we have extrapolated requirements for systems built with larger number of processors.

The results from the study suggest that existing link bandwidth of 200-300 MBytes/sec available on machines like Intel Paragon and Cray T3D can easily sustain the requirements of some applications even on high-speed processors of the future. For the other applications studied, one may be able to maintain network overheads at an acceptable level if the problem size is increased commensurate with the processing speed.

The separation of the parallel system overheads plays an important role in synthesizing the communication requirements of applications. For instance, an application may have an algorithmic deficiency due to either a large serial part or due to work-imbalance, in which case 100% efficiency is impossible regardless of other architectural parameters. The separation of overheads enables us to quantify bandwidth requirements as a function of acceptable network overheads (latency and contention). The framework may also be used for synthesizing requirements of other architectural features such as synchronization primitives and locality capabilities from an application perspective.

4.3 Deriving Architectural Mechanisms

The single most important overhead limiting performance of parallel applications is the communication overhead. One solution is to make the network as fast as possible so that even though the application does not make any fewer network accesses, the overheads will not manifest as a significant component of the total execution time. But the resources to sustain the necessary bandwidth may simply not be available in some cases. The second approach is to reduce the network accesses incurred in the execution or to tolerate the communication overhead if these accesses are unavoidable. Cache coherence protocols, weak memory consistency models, prefetch, poststore, and multithreading are some of the proposed latency reducing and tolerating techniques in the context of shared memory architectures. It has been shown that no one technique is universally applicable for all applications. On the other hand, a close examination of the communication behavior of a range of applications can help derive a set of architectural mechanisms that may prove beneficial and we have conducted such a study in (Ramachandran et al. 1995) using our evaluation framework. By examining the communication properties of applications, we have proposed a set of explicit communication primitives that are generalizations of the poststore and prefetch mechanisms.

Cache coherence protocols broadly fall into two categories: *write-invalidate* and *write-update*. Invalidation-based schemes are more suited to migratory data and can become inefficient when the producer-consumer relationship for shared data remains relatively unchanged during the course of execution. On the other hand, update-based protocols can result in significant overheads due to repeated updates to the same data before they are used by another processor, as well as redundant updates when there are changes to the sharing pattern of a data item. The update and invalidation based schemes thus have their relative advantages and disadvantages, and based on application characteristics one may be preferable over the other. Invalidations are useful when an application changes its sharing pattern, and updates are useful to effect direct communication once a sharing pattern is established.

By examining the communication properties of a spectrum of applications, we have derived a set of explicit communication primitives that use sender-initiated communication within the context of an underlying invalidation-based protocol. The three proposed primitives intelligently propagate the data items to one or more consumers as soon as the data items are produced. The first primitive is intended for applications with static communication behavior where the consumer set of a data item is available at compile time. As a result, this set can be directly supplied to the hardware when the data item is produced. The second primitive is intended for variables governed by locks and it uses the lock structure to propagate data items to the processor next in line for the lock. The third primitive is for applications with dynamic communication behavior which detects the arrival of a new consumer to a current sharing pattern, and uses this information to intelligently mix invalidates with updates.

The execution-driven simulation of real applications has played an important role in this exercise. It has helped us identify and isolate typical communication scenarios in applications and derive a set of mechanisms that can optimize these scenarios. It has also helped us evaluate the cost-effectiveness of these primitives, and the benefits of these primitives over alternate mechanisms. A related study (Shah, Singla, and Ramchandran 1995) develops a realistic model for a shared memory machine, and using SPASM shows that for a spectrum of applications almost all the inherent communication in them may be overlapped with computation. This serves as the motivation for further research in developing explicit sender-initiated communication mechanisms.

4.4 Evaluating Network Designs

The complex interaction between a parallel architecture and an application makes it essential to use realistic workloads for evaluating parallel systems. Performance analyses of processors, caches, memory and I/O subsystems have therefore been conducted with parallel benchmarks. However, unlike other subsystems, the design and analysis of the interconnection network, which is perhaps the most crucial hardware component in a parallel machine, has rarely used the knowledge of workloads generated by parallel applications.

There are two differing perspectives of viewing the multiprocessor interconnection network. From the viewpoint of a software designer or an application programmer, it helps to make certain simplifying assumptions about the interconnection network such as assuming a constant delay or a simple model which does not take into account the details of message traversal within the actual network. These assumptions are sufficiently accurate when the objective is to minimize the communication required. By making these assumptions, performance evaluation of the system can be simplified and speeded-up. Interconnection network designers have a more network-centric viewpoint. From this viewpoint, improving the network performance is critical. Network topology, switching mechanism, routing, flow control, and communication workload, together determine the network performance. Until recently, network research has primarily focussed on the first four parameters to optimize network latency and throughput. Network designers have traditionally used synthetic benchmarks to evaluate their designs. At best, these benchmarks try to mimic some typical communication behavior in applications. The performance results derived from synthetic workloads can provide a general guideline or bounding values, while it may be difficult to make cost-performance architectural design decisions using these results.

SPASM is perhaps the first execution-driven simulator that has been used to integrate both these viewpoints into a single evaluation framework. It has been used extensively to study performance over a wide range of real applications and network parameters. For instance, in (Vaidya, Sivasubramaniam, and Das 1997a) we have used it to study the performance of a 2-dimensional mesh network for 5 shared memory applications. The specific aim in this study is to verify whether the promised performance improvement (for synthetic workloads) using recently proposed network enhancements, such as virtual channels and adaptive routing, is indeed obtained for real applications, and if so do these benefits override the cost of providing these enhancements.

The performance results show that there is a modest performance benefit with these enhancements in the average network latency for the messages. However, with respect to the overall execution time, this improvement is dwarfed in comparison to the other components which constitute the execution time. When considered in the context of application scalability in terms of the number of processors and the problem considered, even though many of the considered applications inject a large number of messages into the network, their arrival into the network does not seem to generate any significant contention for network resources. Consequently, virtual channels and adaptive routing algorithms, which attempt to lower the network contention and not the raw network latency, do not show substantial saving in execution time. Further, our results suggest that the performance rewards may not justify the cost of these enhancements unless an application is highly communication intensive and potentially scaling poorly. On the other hand, if any of these enhancements were to slow down the network router, then there is a significant degradation in performance.

This study (Vaidya, Sivasubramaniam, and Das 1997a) has served has the motivation for yet another project (Vaidya, Sivasubramaniam, and Das 1997b) where we are trying to develop better routers for interconnection networks. In this project, we have formalized a pipelined model for the network router, and we have evaluated the trade-offs between different router designs using our simulator. We have also proposed and evaluated dynamically adaptable selection functions within the router to route messages along less congested paths.

4.5 Characterizing Communication Behavior

Characterization of the communication in parallel applications is essential in understanding their interplay with parallel architectures, to maximize the performance of existing architectures and to design better architectures in the future. The communication traffic of a parallel application can be captured by three attributes namely the temporal, spatial and volume components. Temporal behavior is captured by the message generation rate, spatial behavior is expressed in terms of the message distribution or traffic pattern, and volume of communication is specified by the number of messages and the message length distribution. These three attributes together define the communication workload and have been used extensively in many types of architectural evaluations. In particular, one of the most extensively studied areas of research in parallel architectures is the *interconnection networks*. A plethora of network topologies that support various types of switching mechanism and message routing algorithms have been proposed to design scalable parallel machines. Performance analyses of all these networks either via simulation or analysis require the above three communication attributes.

In the previous subsection, we discussed two studies that have studied the network for real applications using execution-driven simulation. But, such a detailed simulation of the network makes the evaluation exceedingly slow. Mathematical models, on the other hand, do not suffer from this drawback. However, most of these models for interconnection networks have been accused of making unrealistic assumptions about the communication workload. It is not clear what different traffic patterns are generated by parallel applications and how these traffic patterns can be captured by a distribution function for subsequent study. Therefore, the credibility of many model-based performance results has been questioned frequently.

It is thus crucial to develop some formal techniques to capture the communication properties of parallel applications. The novelty of such a characterization is that these attributes can be useful for many divergent studies: a system architect can use the communication information for better architectural design; an algorithm developer can use the communication cost for better algorithm design and analysis; and a system analyst can develop more accurate performance models using realistic workloads.

In (Chodnekar et al. 1997) and (Seed, Sivasubramaniam, and Das 1997), we have embarked on characterizing the communication traffic generated by a spectrum of applications using SPASM. We conduct a detailed execution-driven simulation on a chosen network configuration for each application. The network logs the arrival of messages along with the time of arrival, length and destination information. These logs are then presented to a statistical package (SAS) for regression analysis to calculate the message generation rate, the message length distribution, and the destination distribution.

The results obtained from the analysis of the application traces show that the inter-arrival times of all applications except one can be fitted to known probability distribution functions, which are variations of exponential distribution. Also, the average message generation rate can be obtained for the underlying distribution. For the spectrum of applications considered, the message generation distribution can be expressed in terms of exponential, hypoexponential or weibull distributions. Our results also confirm that the spatial distributions of parallel applications can be captured mathematically. For the applications considered, the spatial distributions are uniform, bimodal uniform and univariate polynomial. The sensitivity of these results to different application and hardware parameters has also been studied. We have found that only the means of the distributions change as we vary many of the parameters. These results lead us closer to the belief that it is possible to abstract the communication properties of parallel applications in convenient mathematical forms that have wide applicability.

5- CONCLUDING REMARKS

Performance evaluation is an integral part of any systems design process to: evaluate the cost-effectiveness of a given design, compare different designs, and derive alternate designs. This process is particularly made more difficult for parallel systems where the complex interaction between application and architecture introduces several more degrees of freedom compared to their sequential counterparts.

Performance evaluation techniques should clearly isolate and quantify the different overheads in a parallel system execution that limit its scalability. Experimentation on the actual system, analytical modeling and simulation are three well-known techniques. But each has its own limitations. Execution-driven simulation offers the most promise because of its ability to study the parallel system accurately and in great detail in a non-intrusive manner. However, we need to confine ourselves to smaller systems with this technique, and complement the evaluation with mathematical models and experimentation to extrapolate performance for larger systems. In this paper, we have described one such simulator called SPASM that has been used extensively to study parallel architectures over a spectrum of applications. We have also briefly discussed five architectural projects that have used this simulator.

Recent trends show that a Network of Workstations (NOW) is a cost-effective solution for high performance computing. There are a wide range of architectural issues that need to be addressed if this platform is to become more prevalent. Our ongoing research is focusing on architectural projects towards this goal. We have recently implemented an execution-driven simulator called pSNOW (Kasbekar, Nagar, and Sivasubramaniam 1997) to specifically study hardware and system software issues for NOW platforms. We intend to use this simulator to design and evaluate architectural innovations concurrently with the development of an actual prototype in our laboratory.

ACKNOWLEDGMENTS

This research is supported in part by a NSF Career Award MIP-9701475 and equipment grants from NSF and IBM.

REFERENCES

Chodnekar, S., V. Srinivasan, A. Vaidya, A. Sivasubramaniam, and C. Das. 1997. Towards a communication characterization methodology for parallel applications. In *Proceedings of the Third International Symposium on High Performance Computer Architecture*, 310–319.

Kasbekar, M., S. Nagar, and A. Sivasubramaniam. 1997. pSNOW: A tool to evaluate architectural issues for NOW environments. In *Proceedings of the ACM 1997 International Conference on Supercomputing*, 100–107.

Ramachandran, U., G. Shah, A. Sivasubramaniam, A. Singla, and I. Yanasak. 1995. Architectural mechanisms for explicit communication in shared memory multiprocessors. In *Proceedings of Supercomputing '95*.

Seed, D., A. Sivasubramaniam, and C. Das. 1997. Communication in Parallel Applications: Characterization and Sensitivity Analysis. To appear in *Proceedings of the 1997 International Conference on Parallel Processing*.

Shah, G., A. Singla, and U. Ramachandran. 1995. The quest for a zero overhead shared memory parallel machine. In *Proceedings of the 1995 International Conference on Parallel Processing*, 194–201.

Sivasubramaniam, A. 1997. Reducing the communication overhead of dynamic applications on shared memory multiprocessors. In *Proceedings of the Third International Symposium on High Performance Computer Architecture*, 194–203.

Sivasubramaniam, A., A. Singla, U. Ramachandran, and H. Venkateswaran. 1994. An approach to scalability study of shared memory parallel systems. In *Proceedings of the ACM SIGMETRICS 1994 Conference on Measurement and Modeling of Computer Systems*, 171–180.

Sivasubramaniam, A., A. Singla, U. Ramachandran, and H. Venkateswaran. 1995. Abstracting network characteristics and locality properties of parallel systems. In *Proceedings of the First International Symposium on High Performance Computer Architecture*, 54–63.

Sivasubramaniam, A., A. Singla, U. Ramachandran, and H. Venkateswaran. 1995. On characterizing bandwidth requirements of parallel applications. In *Proceedings of the ACM SIGMETRICS 1995 Conference on Measurement and Modeling of Computer Systems*, 198–207.

Vaidya, A., A. Sivasubramaniam, and C. Das. 1997. Performance benefits of virtual channels and adaptive routing: An application-driven study. In *Proceedings of the ACM 1997 International Conference on Supercomputing*, 140–147.

Vaidya, A., A. Sivasubramaniam, and C. Das. 1997. The PROUD pipelined routers for high performance networks. Technical Report CSE-97-007, Department of Computer Science and Engineering, The Pennsylvania State University.

AUTHOR BIOGRAPHY

ANAND SIVASUBRAMANIAM is an Assistant Professor in the Department of Computer Science and Engineering at The Pennsylvania State University. He received his B.Tech in Computer Science from the Indian Institute of Technology, Madras, in 1989, and the MS and Ph.D. degrees in Computer Science from the Georgia Institute of Technology in 1991 and 1995 respectively. His research interests are in architecture, operating systems, performance evaluation and application aspects of high performance computing.