Towards Realizing Scalable High Performance Parallel Systems*

Umakishore Ramachandran **H. Venkateswaran**

College of Computing
Georgia Institute of Technology
Atlanta, GA 30332-0280.
{*rama, venkat*}*@cc.gatech.edu*

To appear in a book entitled, "Suggesting Computer Science Agenda (s) for High-Performance Computing", ACM Press

1 Introduction

Computing has permeated a wide variety of disciplines ranging from medical information processing to economic forecasting to distributed decision making. These new demands on computing coupled with the globalization of information that needs to be processed in a decentralized manner calls for widening the scope of *high performance computing*. High performance computing should encompass novel computing paradigms to meet these new computational challenges in addition to meeting the increased processing requirements of applications from the scientific domain. It is clear that sequential computing is inadequate to meet these challenges, and it becomes imperative that we focus our efforts on parallel computing. The term parallel computing is used here broadly to include tightly-coupled multiprocessing to geographically distributed processing of information. Research issues in high-performance computing span distributed organization, accessing, and processing of information, to system architecture issues that lead to the development of high-performance systems.

In developing high-performance computing systems, one has to look at a broad spectrum of issues that include theoretical models for algorithm design, tools for parallel program development, design and evaluation of architectural enhancements for parallel programming, and detailed performance issues of large parallel systems. From the standpoint of addressing high performance computing challenges, a parallel architecture is useful only if it helps in solving problems that are infeasible to solve on sequential machines. Therefore, a fundamental issue which has to be addressed by the research community to make HPCC initiative viable is the *scalability* of parallel systems. By a parallel system we mean an application-architecture combination.

In this position paper, we begin by focusing on the issues relating to the scalability of parallel systems, and the mechanisms and policies for realizing scalable parallel machines. We develop a novel approach to characterizing the scalability notion; develop a framework for implementing this approach; and enumerate the system architecture issues in realizing scalable parallel systems. We then present a more global research agenda for the high-performance computing systems architecture.

*This work has been funded in part by NSF grants MIPS-9058430 and MIPS-9200005, and an equipment grant from DEC.

2 Scalability

2.1 What is Scalability?

From an application's perspective, one would like to see that the efficiency of the parallel system does not degrade as the problem size is increased. Similarly, from an architecture's perspective, the system should deliver increased performance with increased computational resources. The term *scalability* is used to capture this relationship between an application and an architecture. Any set of metrics used to quantify scalability should at least be useful to: select the best architecture platform for an application domain, predict the performance of an application on a larger configuration of an existing architecture, and glean insight on the interaction between an application and an architecture to understand the scalability of other application-architecture pairs.

Performance metrics such as speedup [4], scaled speedup [12], sizeup [30], experimentally determined serial fraction [13], and isoefficiency function [15] have been proposed for quantifying the scalability of parallel systems. While these metrics are extremely useful for tracking performance trends, they do not provide adequate information needed to understand the reason why an application does not scale well on an architecture. At the very least, one would like to know if the problem is with the application or the architecture. In order to do this, we should be able to identify application and architectural bottlenecks in a parallel system that lead to poor scalability. It is important to devise metrics for scalability that separate and quantify such bottlenecks.

2.2 Our Approach

Architectural studies generally address low-level issues such as latency, contention and synchronization. The scalability of synchronization primitives supported by the hardware [5, 21] and the limits on interconnection network performance [2, 23] are examples of such studies. While such issues are extremely important, it is necessary to put the impact of these factors into perspective by considering them in the context of overall application performance. Recognizing this importance, there have been recent studies that use real applications to address specific architectural issues [6, 8, 26].

Adhering to the RISC ideology in the evolution of sequential architectures, we would like to use *real world applications* in the per-

all three which uses the best attribute of each is a good framework for implementing the top-down approach, and is the one we take (see Figure 2). Experimentation is used in conjunction with simulation to understand the performance of real applications on real architectures, and to identify the interesting kernels that occur in these applications for subsequent use in the simulation studies. We use the datapoints obtained from simulation to derive new models as well as refine existing models for predicting the scalability of larger systems. Refined models of system abstractions thus derived can then be used to speed up the simulation.

We have developed a simulation platform (SPASM) [29], that provides an elegant set of mechanisms for quantifying the different overheads we have identified. *Constant problem size* (where the problem size remains unchanged as the number of processors is increased), *memory constrained* (where the problem size is scaled up linearly with the number of processors), and *time constrained* (where the problem size is scaled up to keep the execution time constant with increasing number of processors) are three well-accepted scaling models used in the study of parallel systems. The overhead functions embodied in the simulation platform can be used to study the growth of system overheads for any of these scaling strategies.

The algorithmic bottlenecks identified using the framework can be used as feedback for algorithm redesign. Similarly, the architectural bottlenecks may suggest new architectural primitives to enhance performance. The framework can also be used to validate, develop, and refine models for specific algorithmic and architectural artifacts.

2.4 Research Issues

- It is important to develop *metrics* that adequately capture the performance bottlenecks in the scalability measurements of parallel systems.
- Scalable parallel systems, owing to the several degrees of freedom, are complex to design and evaluate. Therefore, there is a need to develop powerful tools for this purpose. These tools would include architecture simulators, analytical models,visualizers, and performance debuggers. Novel techniques need to be developed to study the interaction between applications and architectures.
- Sequential machines have well accepted set of benchmarks for performance evaluation. It is imperative to develop a similar set of benchmarks for parallel systems evaluation.

3 System Architecture

3.1 Architecture Trends

Parallel machines have evolved along two broad architectural classification: shared memory, and message-passing based on the inherent communication and synchronization model underlying the basic architecture. Shared memory machines are characterized by a set of processors connected to a globally shared memory; all interprocessor communication and synchronization are effected via this shared memory. Message-passing machines are characterized by a set of processors each with its own private memory; there is a global interconnection network that allows processors to communicate and synchronize with one another via messages. In shared memory machines communication is implicit (e.g. updating shared locations); thus a set of mechanisms for synchronization is usually needed. On the other hand, in message-passing machines communication is explicit (via messages). Synchronization and communication are intertwined in that information exchange among the processors serves both. Conventional wisdom suggests that programming shared memory machines is easier due to the global name space provided by the architecture. Message-passing architectures have the potential for providing high performance since the communication can be direct and explicit. Given the relative merits of the two architectural styles, it is clear that future parallel machines should have features that combine the benefits of the two.

3.2 Shared Memory Issues

As an exposition of low level issues in realizing scalable shared memory machines we will first focus on some specifics. Use of private caches as a latency tolerating technique has been successfully used in realizing scalable shared memory machines. In realizing such machines there are two main design issues: memory consistency model, and coherence management. Early designs used sequential consistency, which presents a uniformly consistent view of shared memory for all processors at all times, for the memory model [27, 10]. Recently, use of some form of relaxed memory consistency model has been proposed as a means to improve performance of cache-based shared memory multiprocessors [19, 1, 9, 18]. The basic premise is that most shared memory applications follow some synchronization model and expect consistent views of data only at well-defined synchronization points. Therefore, architectures based on such relaxed memory models, overlap communication with computation by allowing global operations (such as invalidations) to go on in the background and only requiring such operations complete before a synchronization operation.

In addition to the specific memory consistency model is the issue of coherence management in a cache-based shared memory multiprocessor. This involves when and how coherence actions are actually performed in the architecture. The “when” question has to do with whether the hardware or the software triggers the coherence actions; the “how” question relates to whether the software or the hardware keeps the necessary information for taking the coherence actions. Systems such as Alewife [3], KSR-1 [25], and DASH [19] leave the “when" question entirely to the hardware. That is, the coherence action is triggered as soon as the hardware recognizes that there is a potential for inconsistency in the view of a data that is actively shared. Since the compiler/runtime has a better knowledge of the programmer’s intent it makes sense to let the compiler/runtime determine when coherence actions should be initiated. However, for efficiency most of the state information needed to carry out the coherence action should be in the hardware. This is the familiar policy/mechanism argument, with the hardware providing the right set of mechanisms (the “how" question) and the policy (the “when" question) being implemented in the software. We have specified an architecture called *Beehive* [17, 18, 28], wherein we partition the responsibilities between the hardware and software for consistency management. The hardware provides a simple set of mechanisms which are to be used by the compiler/runtime to effect the consistency management.

3.3 Research Issues

- Since shared memory and message-passing each have their own advantages, it is clear that the logical next step is to combine the two in future parallel machines. While there have been some initial investigation in this direction [3, 16, 11], considerably more effort is needed to evolve the right set of mechanisms that are usable from the applications perspective. Further, appropriate programming interfaces need to be developed that exploit the fast communication capability of the underlying hardware while providing shared memory view of the system.
- Analyzing the program semantics can give us useful infor-

mation regarding the sharing pattern and synchronization in a parallel program. Such information can be used to shift some of the hardware functionality into the software. This can be beneficial both from the point of view of reducing hardware complexity as well as improving performance. The research questions to be addressed include deciding the dividing line between hardware and software in terms of policy/mechanism separation; and development of appropriate tools for detailed performance evaluation of this separation. A related issue is devising appropriate hardware mechanisms for runtime/compile-time management of memory consistency; and the design of compile-time/runtime algorithms for data allocation, synchronization, and concurrency management.

- The current trend in building multiprocessors is to use off-the-shelf processors that are not usually designed with multiprocessing in mind. As we get better understanding of the system architecture issues in high-performance computing, it is important to re-engineer uniprocessor architectures. The same is true of networking hardware as well.

4 Agenda

Until now we focussed on two specific issues that we have been looking at in our research for developing high-performance computing systems. We now take a more global perspective and present a research agenda pertaining to system architecture issues.

- **Scalability** studies and **architectural mechanisms** for scalable systems are central to realizing high performance computing systems of the future. It is important to continue development of performance-conscious theoretical and analytical **models** for parallel program development [7]. Another important aspect of the system architecture that needs to be emphasized in the design of parallel systems is **input/output** Given that several large applications have considerable amount of input-output requirements, it is clear that this aspect should be integral to both the design of parallel algorithms as well as high performance system architectures. There have been several recent attempts both from the architecture point of view (such as the parallel disk array technology [14]), and from an algorithmic point of view [22] to address this issue.

- To make parallel computing more accessible and affordable, it is important to investigate off-the-shelf processing and networking technologies for the realization of **low-cost parallel machines**. In this context, the research issues center around **operating systems mechanisms** (such as distributed shared memory [20, 24]) for efficient interprocessor communication, and language-level support for coarse-grained parallel programming.

- The heavy computational demands posed by HPCC applications such as weather modeling has served as an ideal impetus for focusing on the fundamental issues in the development of high-performance systems. There are also several other **application domains**, such as optimization and game theory that has implications on economic forecasting, that can benefit from high performance computing systems. Further, there are computer science applications such as simulation of large parallel systems and networks, and the development of novel user interfaces that need to be investigated in the context of high performance computing.

- It is clear that the computational space occupied by applications from different domains display different and often multiple levels of granularity (i.e. computation to communication ratio). Parallel machines (both commercial and university prototypes) tend to be mostly fixed points in the granularity spectrum ranging from fine-grain (e.g. Maspar MP-2), to medium grain (e.g. KSR-2, CM-5), to coarse-grain (e.g. DEC AXP). Given the diversity in application characteristics in terms of granularity spread, a very interesting research question is to see how to map applications across a **multi-granular** architecture platform. The issues to be investigated include decomposition of a large high performance computing application into parts that fit that well on to specific points in the granularity spectrum; tools that aid in such analyses; the **communication infrastructure** for supporting such a multigranular platform; and portable programming environment across the granularity spectrum.

- From the context of distributed information processing, there is a need for investigating **integrated resource management** issues. These issues include location transparency, security, authentication, and privacy for information access; end-to-end guarantees for low-latency high-throughput reliable communication; and making available the resources that are needed to get the job done without the user having to specify them.

5 Putting it all together

What are reasonable expectations as deliverables from such a research agenda? The ultimate parallel machine? Clearly not! Although parallel computing has been around for a long time, it is not clear whether there is a single programming model or machine characteristic that is appropriate for all applications. However we believe that the following is a reasonable list of expectations:

- Specification and realization of fairly low-cost parallel architectures that give adequate performance across a wide variety of applications.

- A clear understanding of trends in architecture (both uni- and multi-processor), operating systems, and parallel programming in the context of high performance computing.

- Development of parallel programming and performance debugging tools.

- A set of metrics (other than just space and time) for evaluating the performance of parallel systems.

- A set of benchmarks for evaluating parallel machines.

- Performance-conscious algorithm design techniques for applications spanning a wide variety of domains.

- A better understanding of the algorithmic characteristics and their relationship to architectural features.

- A better understanding of the scalability of parallel systems.

6 Acknowledgments

The research ideas and issues presented in this position paper have been generated as part of a larger research project concerned with the system architecture of high performance computing. Many individuals have contributed to these ideas. In particular, we would like to thank Martin Davis Jr., Joonwon Lee, Walter B. Ligon III, Gautam Shah, Aman Singla, and Anand Sivasubramaniam for their contributions.

References

[1] Sarita V. Adve and Mark D. Hill. Weak Ordering - A New Definition. In *Proceedings of the 17th Annual International Symposium on Computer Architecture*, pages 2–14, May 1990.

[2] A. Agarwal. Limits on Interconnection Network Performance. *IEEE Transactions on Parallel and Distributed Systems*, 2(4):398–412, October 1991.

[3] A. Agarwal et al. The MIT Alewife machine : A large scale Distributed-Memory Multiprocessor. In *Scalable shared memory multiprocessors*. Kluwer Academic Publishers, 1991.

[4] G. M. Amdahl. Validity of the Single Processor Approach to achieving Large Scale Computing Capabilities. In *Proceedings of the AFIPS Spring Joint Computer Conference*, pages 483–485, April 1967.

[5] T. E. Anderson. The Performance of Spin Lock Alternatives for Shared-Memory Multiprocessors. *IEEE Transactions on Parallel and Distributed Systems*, 1(1):6–16, January 1990.

[6] D. Chen, H. Su, and P. Yew. The Impact of Synchronization and Granularity on Parallel Systems. In *Proceedings of the 17th Annual International Symposium on Computer Architecture*, pages 239–248, 1990.

[7] D. Culler et al. LogP : Towards a realistic model of parallel computation. In *Proceedings of the 4th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 1–12, May 1993.

[8] R. Cypher, A. Ho, S. Konstantinidou, and P. Messina. Architectural requirements of parallel scientific applications with explicit communication. In *Proceedings of the 20th Annual International Symposium on Computer Architecture*, pages 2–13, May 1993.

[9] M. Dubois, C. Scheurich, and F. Briggs. Memory Access Buffering in Multiprocessors. In *Proceedings of the 13th Annual International Symposium on Computer Architecture*, pages 434–442, June 1986.

[10] Encore Computer Corporation, 257 Cedar Hill St., Marlboro, MA 01752. *Multimax Technical Summary*, 1986.

[11] Matthew I. Frank and Mary K. Vernon. A hybrid Shared Memory/Message Passing parallel machine. In *Proceedings of the 1993 International Conference on Parallel Processing*, August 1993.

[12] J. L. Gustafson, G. R. Montry, and R. E. Benner. Development of Parallel Methods for a 1024-node Hypercube. *SIAM Journal on Scientific and Statistical Computing*, 9(4):609–638, 1988.

[13] A. H. Karp and H. P. Flatt. Measuring Parallel processor Performance. *Communications of the ACM*, 33(5):539–543, May 1990.

[14] Randy H. Katz, Garth A. Gibson, and David A. Patterson. Disk System Architectures for High Performance Computing. *Proceedings of the IEEE*, 77(12):1842–1858, 1989.

[15] V. Kumar and V. N. Rao. Parallel Depth-First Search. *International Journal of Parallel Programming*, 16(6):501–519, 1987.

[16] Jeffrey Kuskin et al. The Stanford FLASH multiprocessor. In *Proceedings of the 21th Annual International Symposium on Computer Architecture*, pages 302–313, April 1994.

[17] Joonwon Lee and Umakishore Ramachandran. Synchronization with Multiprocessor Caches. In *Proceedings of the 17th Annual International Symposium on Computer Architecture*, pages 27–37, 1990.

[18] Joonwon Lee and Umakishore Ramachandran. Architectural Primitives for a Scalable Shared Memory Multiprocessor. In *Proceedings of the Third Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 103–114, Hilton Head, South Carolina, July 1991.

[19] D. Lenoski, J. Laudon, K. Gharachorloo, W-D Weber, A. Gupta, J. Hennessy, M. Horowitz, and M. S. Lam. The Stanford DASH multiprocessor. *IEEE Computer*, 25(3):63–79, March 1992.

[20] Kai Li and Paul Hudak. Memory coherence in shared virtual memory systems. *ACM Transactions on Computer Systems*, 7(4):321 – 359, Nov 1989.

[21] J. M. Mellor-Crummey and M. L. Scott. Algorithms for Scalable Synchronization on Shared-Memory Multiprocessors. *ACM Transactions on Computer Systems*, 9(1):21–65, February 1991.

[22] M. H. Nodine and J. S. Vitter. Large-scale sorting in parallel memories. In *Proc. 3rd Annual ACM Symp. on Parallel Algorithms and Architectures*, pages 29–39, July 1991.

[23] G. F. Pfister and V. A. Norton. Hot Spot Contention and Combining in Multistage Interconnection Networks. *IEEE Transactions on Computer Systems*, C-34(10):943–948, October 1985.

[24] Umakishore Ramachandran and M. Yousef Amin Khalidi. An implementation of distributed shared memory. *Software Practice & Experience*, 21(5):443–464, May 1991.

[25] Kendall Square Research. Technical summary, 1992.

[26] E. Rothberg, J. P. Singh, and A. Gupta. Working sets, cache sizes and node granularity issues for large-scale multiprocessors. In *Proceedings of the 20th Annual International Symposium on Computer Architecture*, pages 14–25, May 1993.

[27] Sequent Computer Systems Inc., Oregon. *Sequent Guide to Parallel Programming*, 1987.

[28] Gautam Shah and Umakishore Ramachandran. Towards exploiting the architectural features of beehive. Technical Report GIT-CC-91/51, College of Computing, Georgia Institute of Technology, November 1991.

[29] A. Sivasubramaniam, A. Singla, U. Ramachandran, and H. Venkateswaran. 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*, pages 171–180, May 1994.

[30] X-H. Sun and J. L. Gustafson. Towards a better Parallel Performance Metric. *Parallel Computing*, 17:1093–1109, 1991.

[31] J. C. Wyllie. *The Complexity of Parallel Computations*. PhD thesis, Department of Computer Science, Cornell University, Ithaca, NY, 1979.