Scalable Shared-Memory Architectures

Introduction to the Minitrack

Josep Torrellas
Center for Supercomputing Research and Development
and Computer Science Department
University of Illinois at Urbana-Champaign, IL 61801, USA

1 Background

Shared-memory multiprocessing is emerging as a popular approach to increase computing power, over that of sequential computing, while maintaining programmability. What makes the base shared-memory paradigm attractive is the simplicity of the programming model: all memory is in a pool and is globally shared. Furthermore, architectures that do not require a broadcast channel for interprocessor communication can potentially scale to many processors and therefore provide large-scale computing power. Such systems are loosely termed "scalable".

There are many hardware and software research issues in scalable shared-memory multiprocessors. One of the most important ones is how to handle the increasing speed mismatch between fast processors and slow, far-off memory systems. Indeed, data needs to be fetched and returned to memory, and the processor may have to remain idle during this transfer. This problem is partially eliminated with the presence of caches or local memories, which try to keep the working set close to the processor. Alternative or complementary approaches to caches are prefetching [5, 9] and multithreading [2]. Prefetching consists of fetching instructions or data before they are needed while overlapping the accesses with other computation. Both hardware and software approaches are possible. Multithreading consists of switching processes when an instruction or memory access by a process would stall the processor. When the process is finally re-scheduled, the offending condition that forced the switch is likely to have disappeared. For example, if the offending condition is a remote memory read, the data is likely to be in the cache when the process is re-scheduled.

Unfortunately, not only do processors want to access memory, they want to communicate with each other as well. This gives rise to the synchronization problem [8]. Furthermore, given that, for performance reasons, data is usually allowed to replicate in the memory hierarchy, the popular problems of data coherence and memory consistency appear. Data coherence has been studied from many different angles. There are many hardware solutions [3] (variations of directory-based schemes), compiler solutions [6], and operating system-based approaches [4]. Memory consistency models focus on how the overlapping of memory accesses from different processors to the same set of variables affects the sequence of operations seen by the program [7, 1].

While communication resulting from the sharing of data (also called true sharing) is unavoidable, other non-essential sources of communication slow the machine further. Such sources are false sharing [10] and process migration [11]. The former can appear when the unit of coherence is larger than a single word; the latter is often necessary if we want to keep the machine load-balanced. These are also active areas of research.

Another active area of research focuses on the channels used by the processors to communicate, namely the interconnection networks. Many network configurations exist. The reason is that each of them satisfies different cost/performance requirements, which range from those of busses to crossbars.

In addition to the mostly hardware research issues that we mentioned, there are countless software issues that need to be explored in these architectures. Examples are task scheduling issues, operating systems issues, and many compiler/language issues. The shared-memory paradigm is attractive because it simplifies the development of complex software systems. For instance, there are several robust commercial multiprocessor operating systems and parallelizing compilers.

Last, but not least, there is the active field of application design for these machines. Many of the efforts done by developers of applications for these machines result in feedback for the architecture and system designers. Overall, the field of scalable shared-memory architectures is one of the most exciting and vast areas

of research in computer systems nowadays.

2 Scope of the Minitrack

The papers that have been accepted for this minitrack are good examples of the variety of research topics involved in building scalable shared-memory architectures. The first paper, "*Using Hints to Reduce the Read Miss Penalty for Flat COMA Protocols*", discusses a hardware optimization for the cache coherence protocol of cache-only memory architectures. Examples of such machines are the Kendall Square Research KSR-1 and the DDM-1 from the Swedish Institute of Computer Science. The second paper, "*Decoupled Pre-Fetching for Distributed Shared Memory*", discusses hardware support for data prefetching. The third paper, "*Modeling Load Imbalance and Fuzzy Barriers for Scalable Shared-Memory Multiprocessors*", focuses on the synchronization and load imbalance problems. It uses an analytical model to estimate the impact of these issues. Finally, "*A Survey of Software Solutions for Maintenance of Cache Consistency in Shared Memory Multiprocessors*" discusses schemes that use the compiler to keep data coherent.

Acknowledgments

I would like to thank all the authors that submitted papers to this minitrack; unfortunately, only a few papers could be published. Thanks also to the many reviewers who offered their time and expertise. Finally, thanks to my assistant, Donna Guzy, for her help and Trevor Mudge for his advice.

References

[1] S. Adve and M. Hill. Weak Ordering - A New Definition. In *Proceedings of the 17th Annual International Symposium on Computer Architecture*, pages 1–13, May 1990.

[2] A. Agarwal. Performance Tradeoffs in Multithreaded Processors. In *IEEE Transactions on Parallel and Distributed Systems*, volume 3, pages 525–539, September 1992.

[3] A. Agarwal, R. Simoni, J. Hennessy, and M. Horowitz. An Evaluation of Directory Schemes for Cache Coherence. In *Proceedings of the 15th Annual International Symposium on Computer Architecture*, pages 280–289, May 1988.

[4] W. Bolosky, R. Fitzgerald, and M. Scott. Simple but Effective Techniques for NUMA Memory Management. In *Proceedings of the 12th ACM Symposium on Operating System Principles*, pages 19–31, December 1989.

[5] T. F. Chen and J. L. Baer. A Performance Study of Software and Hardware Data Prefetching Schemes. In *Proceedings of the 21st Annual International Symposium on Computer Architecture*, pages 223–232, April 1994.

[6] H. Cheong and A. V. Veidenbaum. Compiler-Directed Cache Management in Multiprocessors. In *IEEE Computer*, pages 39–47, June 1990.

[7] K. Gharachorloo, D. Lenoski, J. Laudon, P. Gibbons, A. Gupta, and J. Hennessy. Memory Consistency and Event Ordering in Scalable Shared-Memory Multiprocessors. In *Proceedings of the 17th Annual International Symposium on Computer Architecture*, pages 15–26, May 1990.

[8] J. R. Goodman, M. K. Vernon, and P. J. Woest. Efficient synchronization primitives for large-scale cache-coherent multiprocessors. In *Proceedings of the 3rd International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 64–73, April 1989.

[9] T. Mowry, M. Lam, and A. Gupta. Design and Evaluation of a Compiler Algorithm for Prefetching. In *Proceedings of the 5th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 62–73, October 1992.

[10] J. Torrellas, M. S. Lam, and J. L. Hennessy. False Sharing and Spatial Locality in Multiprocessor Caches. In *IEEE Trans. on Computers*, pages 651–663, June 1994.

[11] J. Torrellas, A. Tucker, and A. Gupta. Evaluating the Performance of Cache-Affinity Scheduling in Shared-Memory Multiprocessors. In *IEEE Transactions on Parallel and Distributed Systems*. To appear 1994. A short version appeared in ACM Sigmetrics 1993.