Scalable Shared-Memory Architectures

Introduction to the Minitrack

Per Stenström

Department of Computer Engineering, Lund University
PO Box 118, S-221 00 LUND, Sweden

Background

The single address-space that shared-memory architectures offer simplifies programming, problem partitioning, and dynamic load balancing as compared to other programming models for parallel computing systems such as e.g. message passing. Unfortunately, as we scale shared-memory architectures to large configurations, the resulting memory system latencies may limit their performance potentials. Finding cost-effective solutions to the memory-system latency issue has become an important research objective and is the main focus of this minitrack. Loosely speaking, scalability for these systems refers to optimizing both the performance and the implementation cost. While it is not meaningful to strictly define the term scalability, it is an important intuitive goal when evaluating new shared-memory architectures.

For a shared-memory architecture to be performance-scalable, we must find methods to reduce or tolerate the memory-system latencies of large configurations. Because another important attribute of shared-memory architectures is their generality in terms of application domains, they are interesting to consider as general-purpose computing platforms. Therefore, low implementation cost of these systems is also an important objective; the performance of large-scale, shared-memory architectures should be traded off against their hardware complexity introducing several issues we discuss next.

A key technique for latency reduction is replication of shared data across the processing nodes. In a distributed system, the shared-memory view can be afforded by using the local memories as caches. By contrast, multiprocessor systems use private caches to effectively reduce the latencies and contention of the shared-memory interconnect. For both kinds of systems, an important issue becomes how to maintain a coherent view of the single address-space despite of the many copies of blocks/pages that are distributed in the system. In the context of distributed (virtually shared memory) systems (DVSM) this issue is referred to as the memory coherence problem, whereas in multiprocessors with private caches, it is called the cache coherence problem [6].

Despite of the countless number of published papers on the coherence issue, there are still issues that have not been fully explored. (For an excellent tutorial, see [5] edited by Tomasevic and Milutinovic.) In a multiprocessor, a hardware-based distributed protocol, called a cache coherence protocol, typically maintains consistency. One important issue to consider is which policy to use; write-validate, write-update, or a combination of both [6]. Another issue relates to how to limit the bandwidth needed for coherence traffic. While small-scale systems prefer to broadcast updates or invalidations, large-scale systems cannot afford this; updates or invalidations should be explicitly sent to the caches having a copy—these protocols are referred to as directory-based protocols [1]. Critical issues for directory-based protocols are the latencies for consistency actions, the bandwidth requirement, and the implementation cost.

Even if we can find effective solutions to the memory/cache coherence problem, the latency cannot be completely eliminated because the intrinsic communication need in the application puts an upper limit on the performance. To cope with these "intrinsic" latencies, another approach is to hide them through latency hiding techniques. The key idea is to overlap communication (e.g. consistency actions) with local computation. Important techniques that have been considered include memory consistency model relaxation, prefetching, and multithreading [4].

Relaxation of the memory consistency model is based on the intuition that the synchronization points in a program dictate where communication really has to take place; between two synchronization points, we
can relax the consistency requirement. Based on this fundamental observation by Dubois et al. [2], a large number of papers have addressed this issue. Although studies have demonstrated that relaxed memory consistency models have a potential to considerably enhance the performance [3], it is still an open question whether their added complexity to the software model justifies their performance advantages. Apart from the memory consistency model, efficient implementation of synchronization is an important issue, especially for DVSM systems in which memory coherence is supported by operating system primitives and coherence actions trap the processors attached to the nodes involved.

The importance of the area has been emphasized lately because research institutions as well as companies have managed to build cost-effective systems. Important examples of such systems are cache-coherent NUMA architectures [7], such as the Stanford DASH and the MIT Alewife, and COMA (cache-only memory) architectures, such as the Swedish Institute of Computer Science's Data Diffusion Machine and the Kendall Square Research's KSR1. Also in the domain of DVSM systems a notable example is the Minin system at Rice University.

Scope of the Minitrack

The papers that have been accepted for this minitrack span a spectrum of issues and demonstrate the variety of research topics involved in building scalable shared-memory architectures. The minitrack consists of seven papers in three sessions, and a panel discussion. In the first session, entitled “Cache and Memory Coherence Schemes”, there are three papers that address implementation as well as performance issues related to scalable solutions to the memory or cache coherence problem. In the second session, called “Memory Management and Fault-Tolerance”, two important problems are addressed. The first paper attacks the TLB (translation-lookaside buffer) consistency problem and considers the performance-cost tradeoff in where to place the TLB, whereas the second paper focuses on schemes to enforce fault tolerance in a DVSM system. Finally, the third session, “Synchronization and Memory Consistency Models” contains two papers that present new synchronization schemes as well as memory consistency models that offer a potential performance advantage over previously proposed schemes.

Acknowledgments

First, I would like to thank the authors for their contributions to this minitrack. I am also indebted to the large number of reviewers that not only helped me in the paper selection process; because of their constructive criticism, the authors of the accepted papers could improve them before sending them to the press. Finally, I would like to express my appreciation to my assistant Elisabeth Larson for being helpful with all practical matters related to the minitrack organization when I took off for a two months leave at USC in the middle of the minitrack preparation phase.

References


