1. Motivation

Emerging applications as well as software and hardware technologies have always been a driving force for architectural developments and shared-memory multiprocessors are not an exception. Because multiprocessors take advantage of mainstream hardware and software technologies, they provide a particularly cost-effective solution to high-performance computing. While research into this, what seems to be a promising technology, has received considerable attention, machines are today mainly used as servers to increase throughput and only in rare cases to speed up individual applications. We all know that the main reason is that it is not sufficiently easy to program them.

The purpose of this task force is to identify fruitful research directions in architecture, programming support, and applications for shared-memory multiprocessors.

2. Current state of research

 Needless to say, shared-memory multiprocessing has been, and still is, an active research area. Research activities fall into the following main categories:

- applications,
- programming support (parallelizing compilers, performance debugging tools), and
- architecture

Application domains that traditionally have driven the research efforts are from the scientific and engineering domains. Unfortunately, industry has criticized academia for focusing too much on this domain. A key question is what application domains are and will become interesting and how they will change our ways of conducting research in this area.

It is generally agreed that the major breakthrough for multiprocessors comes if we ever solve the parallel programming dilemma; to provide application programmers with appropriate support to parallelize her/his applications. While some progress has been made in this area, compiler technology and performance tuning methods are immature so a question is what can be done in this area.

Finally, architecture research has for the most part been concerned with how we can overcome the shared-memory bottleneck of machines ranging from small bus-based multiprocessors to distributed memory systems. Is this topic exhaustively researched or is there still room for major breakthroughs. Or, should we pay more attention to other issues such as dependability and predictability issues?

Here follow some reflections on these issues from some of the participants in this task force.

3. Position statements

David J. Lilja, University of Minnesota:

As a community of computer engineers and scientists, we have (naturally) tended to focus our greatest efforts and resources on improving the performance of scientific and engineering applications. However, by taking this narrow focus we have ignored the economic analog of Amdahl's Law, which says that we should concentrate our efforts on the segment of the market with the largest monetary return. Since significantly more money is spent to purchase computer systems that perform commercial applications, such as transaction-processing and database operations, we should now turn our attention to applying what we have learned in developing shared-memory multiprocessor systems for numerical applications to these widespread commercial applications.

The development of architectural ideas is strongly driven by the application domains of interest. Our single-minded focus on benchmarks such as SPEC, SPLASH, and the Perfect Suite has led us to study specific topics in shared-memory architectures, such as cache memory design, coherence protocols, and false-sharing. However, there is growing evidence that these factors are less important for commercial applications than for numeric applications. For instance, there are indications that a cache shared among the processors provides better performance than private caches for transaction-like applications. As a result, false-sharing is eliminated, along with the cache coherence problem itself. Another example is data prefetching. Numerous prefetch mechanisms have been developed that work well for highly-regular application programs, but fail miserably with less structured programs. We need to understand more deeply how this different application domain affects the architectural decisions we make when designing the next generation of shared-memory multiprocessor systems.

In addition to expanding our horizons to include more commercially important application domains, we need to
understand how to exploit parallelism at multiple granularities simultaneously. The industry continues to move towards higher integrated circuit transistor densities making feasible the integration of the equivalent of several processors on a single chip. These devices will be incorporated into shared-memory multiprocessor systems that will, in turn, be incorporated into larger networks of heterogeneous systems. Significant research is needed to determine how best to exploit the resulting hierarchies of parallelism that will be made possible by these new configurations. For instance, it is likely that the fine-grained instruction-level parallelism should be exploited automatically by the compiler and the hardware within each cluster of processors, but what about the coarse-grained parallelism that potentially can be exploited between these highly parallel nodes? Furthermore, within these cluster systems, should special-purpose networks be introduced that are tuned to different types of communication, or should a single, standardized network be used for all types of intertask communication?

In summary, we have done a good job of developing architectures that support well-structured numeric application programs, but it is now time to turn our attention to supporting the application programs and system configurations on which the real money is spent.

Margaret Martonosi, Princeton University:
To obtain good performance on parallel computers, software must be written and compiled to take advantage of some specific characteristics of the underlying machine, and to avoid others. For example, exploiting an appropriate parallelism granularity and making efficient use of communication mechanisms can be crucial in approaching the peak performance of the machine. For this reason, it has become increasingly important to develop performance monitoring systems that allow programmers and system designers to identify and tune the portions of their design that are limiting application and system performance.

Key hurdles to efficient performance monitoring are: (i) having physical access to observe the "interesting" performance events, and (ii) monitoring these events with low overheads and little perturbation. A major issue pertaining to the observability of performance events lies in the degree of integration of the system being monitored. As more and more functionality of a parallel system is implementable on a single chip, it will become important to integrate hardware performance monitoring support on-chip as well. Furthermore, since monitoring a system may affect the execution interleaving of parallel processes, the goal of a monitoring system should be to minimize its impact on the execution time, caching, and communication behavior of the machine being monitored. Rather than rely on ad-hoc hardware and software monitoring designs, a dialog between architects, application programmers, performance tool developers and compiler writers can help drive the development of efficient, widely-useful performance monitoring support.

Steve Woo, Rambus Inc.:
Although multiprocessors have been the focus of considerable research effort, these machines remain difficult to program and are often relegated to being used as compute servers to increase job throughput rather than to increase the performance of individual applications. Several issues need to be addressed in order to reverse this trend. First, it is important to understand what difficulties applications designers face during the creation of parallel programs. Aside from the difficulties of creating and implementing an algorithm also seen when writing uniprocessor programs, parallel programs have the added hurdle of requiring well-planned synchronization and communication to avoid race conditions that can be hard to detect and diagnose. Parallel applications development would benefit significantly from compiler feedback that detects potential pitfalls such as race conditions, in much the same way that vector programming is enhanced by having the compiler alert programmers of dependencies that inhibit vectorization. Research efforts in this direction will continue to improve the process of developing parallel programs. Performance tuning often presents programmers with several additional obstacles. It is often difficult to isolate the source of performance bottlenecks (CPU, memory, I/O), but application developers would benefit from integrated performance tuning environments that combine hardware event counters with information on memory system performance (hot-spotting, cache conflicts) and I/O performance into a set of graphical tools that can provide quick feedback during the performance tuning process. Important research has already been conducted in each of these areas individually, and combining key research ideas in each of these areas into an integrated tool would significantly enhance the development process.

In terms of multiprocessor architecture, future research must continue to look at emerging applications to assess needs at both the hardware and software levels. An important issue that needs to be addressed is the identification of application domains that research should focus on. Finally, one possible reason why parallel programming is not more widely used is that it remains largely a topic covered only at the graduate level. Interest in writing parallel programs may be broadened by expanding undergraduate engineering and computer science curricula to include parallel programming.