Future computer applications will be targeted toward highly parallel architectures because of their attractive cost-to-performance ratios. Developing software for these machines will be even more troublesome than it is for traditional computers due to synchronization issues and new algorithms and languages, yet there is currently little software support for highly parallel architectures. Some vendors minimize the problems in developing quality software for non-sequential machines, but lack of an adequate software base is already hindering experimentation with this class of machines.

Software for many of today's high performance computer architectures can only be developed by elite programmers (i.e., gurus) who fully understand the architecture of the target machine. Before future needs can be adequately met for high quality software in a cost-effective manner for large applications, better software engineering techniques must be available for parallel architectures. The following is a list of issues that need to be resolved to facilitate advances in software development for parallel architectures:

The proper software development paradigm is not established.

The traditional life-cycle model is being questioned for some sequential computer applications, and its use for parallel architectures is even more in doubt. “The emergence of parallel architectures...offers the motivation and opportunity for a new paradigm for software development,” believes J. C. Browne. New approaches will be object-oriented and will use more graphic, higher-bandwidth interaction between man and machines (1).

Problem visualization and animation techniques are deemed useful by many experts but are immature technologies.

To help monitor program processing and understand huge volumes of data, visual techniques such as algorithm animation are recommended but need to be more fully developed.

Language issues for parallel architectures for future systems have not been resolved.

There are many unanswered questions on whether new languages should be developed to address parallel issues at the language level. Additional experience in large-scale application are needed before "best" solutions can be determined. Software support tools are lacking for parallel object-oriented design and implementation at the language, compiler, and runtime package levels.

Approaches to problem decomposition are important to map applications to the proper parallel architecture.

Coarse-grained architectures require a very close fit between algorithms or data and the hardware to achieve performance gains. If that fit isn't there to begin with, extracting the proper parallelism is likely to fail. For example, unless vectors contain tens of elements for a SIMD machine, performance may not exceed a sequential machine (2).

Algorithm design and evaluation require architecture understanding.

Extracting maximum parallelism for algorithms is necessary to achieve performance gains. Each architecture type is likely to need its own algorithms in order to take advantage of its individual strength. Knowledge is also needed to implement the algorithm effectively on different parallel machines.

Techniques for software design partitioning and allocation are lacking.

General methodologies for decomposing algorithms for parallel architectures are not available. Partitioning (dividing an algorithm into procedures, modules and processes) and allocation (assigning these units to available processors) are among the most difficult and most important decomposition issues (3).
Algorithms and protocols are needed for scheduling parallel processors.

More support is needed to help tell what the hardware should do in parallel—not to be confused with designing more parallelism than already exists in a given algorithm.

Execution monitoring of concurrent processes requires new tools and techniques.

To properly debug parallel software, programmers must gain access to information about the run-time state of the machine. Monitoring use of CPUs and buses is also necessary to insure performance bottlenecks are not occurring.

Software testing is inadequate to provide necessary software reliability levels.

Reliability will be affected by the occurrence of unresolved errors due to asynchronous behavior, and this will be especially critical in non-numeric applications with unrepeatable behavior patterns. Besides typical software errors, there will be additional abnormalities due to synchronization issues that testers must uncover.

Techniques for evaluating the quality (reliability, maintainability, etc.) of parallel software are almost non-existent.

While attention is currently focused on performance levels, there has been little regard paid to quality or productivity issues. Parallel software certainly can increase performance but explicit user-level parallelism (handcoding) is likely to be complicated and expensive. Implicit parallelism (parallel compilers or sophisticated languages) may be more transparent to the user but these types of tools are still immature and may not achieve highest levels of performance.

Fault avoidance and detection, reliability prediction and reliability estimation have not been seriously studied for parallel architectures.

Achieving high performance may be paramount for some applications, but in other cases parallel architectures will be used because of their attractive cost-to-performance ratios. In these cases, having control over the quality of the software becomes an increasingly critical issue. Just as software reliability is a serious problem in the serial world, it will be even more of a problem for parallel architectures.

Software reusability will be limited by parallel architectures.

Software reusability is seen by many to be a prerequisite to productivity improvements. Although it, too, is a problem that has not been effectively solved in the serial world, it will be even more difficult to address for parallel machines because of the close relationship of algorithms to hardware. If software applications can not be easily ported, fewer machines will be sold and customized software will have to be developed. Both conditions lead to higher acquisition costs for parallel systems and will slow their use in mainstream computing.

Fault tolerance models for fault-tolerant computing are not adequate for complicated parallel computation.

With more processors, there is an increasing chance that a processor might fail. Recovery may be more critical because order of computation may not be known. Another question that needs to be answered is "What happens to a hypercube, for example, when a processor is lost and you no longer have a hypercube architecture?"

References

