A User's Perspective on the State of Parallel Processing

Jon G. Kuhl
Department of Electrical and Computer Engineering
University of Iowa
Iowa City, IA 52242

1. Motivation

The comments in this section reflect the experiences and perspectives of a researcher actively involved in the development of large-scale, highly parallel applications hosted on a variety of parallel processing architectures. For the past 18 months the Centers for Computer-Aided Design and Mechanical System Simulation and Design Optimization at the University of Iowa have been developing and implementing parallel algorithms for the real-time simulation and visualization of the dynamic performance of multibody mechanical systems. The resulting algorithms, based on new recursive dynamics formulations developed by Bae and Hwang [Bae87, Hwa88], can simultaneously exploit parallel processing at several different levels, ranging from fine-grained vector processing to moderately fine-grained (millisecond and sub-millisecond) heterogeneous tasking, to large granularity (up to several seconds) parallel tasks suitable for distributed computation on a local network. A number of different parallel processing platforms have been employed including an 18 processor Encore Multimax multiprocessor, an Alliant FX/8 mini-supercomputer with 8 computational elements (CEs), a 32 processor distributed memory Transputer system, and a network of Apollo workstations.

A number of prototype projects have been completed including a real-time simulation and on-line animated visualization of a complex wheeled vehicle, detailed in [Dub88]. This project involved the creation of several tens of thousands of lines of parallel code to support the real-time simulation on the Alliant FX/8 and the on-line interactive visualization using a network of workstations.

Tools and paradigms employed in the design of the application include vectorization and loop concurrency supported by the Alliant's Fortran compiler, a locally written lightweight tasking environment to efficiently support small-grained, heterogeneous tasking on the Alliant, and remote procedure call facilities to support distributed computing on the workstation network.

2. Position Statement

Based on the considerable, and often frustrating, experience gained during the work on these projects, this position statement offers several observations about the current and future state of parallel processing.

Whereas the design of sequential software involves mapping a problem onto a more-or-less fixed computing architecture, parallel software design often requires substantial design effort to map the architecture to fit the problem. This is particularly true when dealing with parallel processor architectures that are reconfigurable at the hardware or communications architecture level. However, it also seems to be true for parallel architectures with a fixed physical structure. For instance, in our use of the Alliant FX/8 architecture, we find it appropriate, even within a single application, to utilize the architecture at times as a conventional vector processor, at other times to use the CEs for executing heterogeneous parallel tasks, and at still other times to run distinct communicating processes on the CEs. Indeed, a large degree of the power and utility of general parallel processing seems to accrue from the ability to control and configure the processing components of the system to suit the requirements of the problem.

2. In our experience, no single set of paradigms, abstractions, descriptive models, or programming tools is sufficient to support the specification and design of parallel applications. As noted above, the nature of the problem seems, in many cases, to dictate the appropriate models and abstractions. At the programming level, abstractions generally imply overheads and performance compromises that may be unacceptable for applications with hard performance requirements.

3. General tools for automating the creation of parallel applications are, and will perhaps always be, of limited value. Beyond the class of so-called "embarrassingly parallel" applications, development of efficient parallel applications must start at the most fundamental levels of algorithm design. Even at this level, specific architectural issues can have significant impact on the structure of the algorithm. For instance, in our experience with mechanical system dynamics algorithms, the switch from a shared memory multiprocessor with vector processing abilities to a distributed memory transputer system has required fundamental rethinking and redesign of the dynamics formulations. This may appear somewhat in contrast to the comments above, about mapping the processing architecture to the problem. In fact, the parallel algorithm design process seems to involve an intertwined process of tailoring the computing architecture to match the problem and designing the algorithm to be compatible with the inherent architectural characteristics and limitations of the system.

4. All of the problems discussed above notwithstanding, parallel software design is not as intimidating or hopelessly difficult as some would have us believe. Once we free ourselves from our sequential biases and let the basic properties of the problem itself guide the design process, things generally flow along in a more or less intuitive fashion. At our site, we have seen personnel with little previous software design experience become competent parallel software designers in a little over a year's
time. More importantly, we have seen considerable success with the ability of disciplinary engineers to develop new problem formulations that are highly amenable to parallel processing. The basic hurdle seems to be not the lack of consistent models or paradigms, but rather just acquiring the instincts to "think parallel".

5. By far the most difficult and painful part of the parallel software design process is the lack of effective support tools for performance analysis, testing, and debugging. Even the most rudimentary tools in the sequential programmer's tool box, such as profilers and interactive debuggers do not exist for most parallel processing environments. Furthermore, the effective provision of these tools is thwarted by the fact that even modest perturbation or instrumentation of parallel code can significantly alter its execution-time behavior. Even without the effects of code instrumentation, errors in parallel software are often not consistently repeatable due to the asynchronous nature of the control and/or data flows in the application. A software designer, working blind, without benefit of effective tools, can find it nearly impossible to identify and fix subtle problems that invariably plague large parallel software projects. Of course these problems are not entirely new. They have plagued the concurrent and event-driven software communities for a long time. However the severity of the problem is obviously exacerbated by the large number of parallel control and data flows in a parallel application. Some sort of architectural level support to allow the monitoring, capture and perhaps recreation of control and data flows might ultimately be called for to assist with these difficult problems.

3. Summary

We should not become so caught up in the search for general and consistent models and programming tools for parallel processing that we forsake the ability to tailor the processing resources of the system to the requirements of the problem and vice versa. High level abstractions that hide architectural details may obscure performance issues that are important to the algorithm design process. They may also constrain or force compromises in the manner in which the underlying architecture can be applied to the problem. Powerful tools are wonderful and desirable, but these tools should be tailored for, and compatible with, the specific architecture, or class of architectures which they support. The parallel processing community does not need a single, uniform set of models, tools, or paradigms to achieve success. All that is needed is the same sort of basic design and analysis tools that have served the sequential processing community for a number of years.

References

