PRELUDE: A System for Portable Parallel Software

William Weihl Eric Brewer Adrian Colbrook
Chrysanthos Dellarocas Wilson Hsieh Anthony Joseph
Carl Waldspurger Paul Wang *

Large-Scale Parallel Software Group
MIT Laboratory for Computer Science

1 Introduction

We describe PRELUDE, a programming language and system for writing portable MIMD parallel programs. PRELUDE supports a methodology for designing and organizing parallel programs that makes them easier to tune for particular architectures and to port to new architectures.

A number of MIMD multiprocessors are now commercially available, and new large-scale machines are being developed. Given the variety of architectures and the cost of developing software, there is a clear need for support for writing high-performance programs that are portable and scalable across a broad range of MIMD architectures.

PRELUDE allows the programmer to write programs using an abstract model of computation that is independent of any particular architecture. A program can then be mapped onto a particular machine by attaching annotations to it. This allows the programmer to separate the description of the computation to be performed by a program from the description of how that computation is to be mapped onto a machine. We provide most of the power of approaches that give the programmer direct, low-level control while retaining most of the ease of use of purely automatic approaches.

Our goal in PRELUDE is to provide a comprehensive suite of mapping mechanisms that give the programmer sufficient power to implement a range of parallel programs efficiently on a wide variety of MIMD architectures. To this end, we have included many mapping mechanisms that have appeared in other systems, including remote procedure call, object migration, and data replication and partitioning. In addition, PRELUDE includes novel migration mechanisms for computations based on a form of continuation passing.

A complete version of this paper is available [1].

2 The PRELUDE System

PRELUDE provides the programmer with a computational model based on objects and threads that abstracts away from the underlying architecture, together with

*This work was supported by the National Science Foundation under grant CCR-8716884, by the Defense Advanced Research Projects Agency (DARPA) under Contract N00014-89-J-1988 and by an equipment grant from Digital Equipment Corporation. Individual authors were supported by an Office of Naval Research Graduate Fellowship, a Science and Engineering Research Council Postdoctoral Fellowship, National Science Foundation Graduate Fellowships, an IBM Graduate Fellowship and an AT&T Graduate Fellowship.

high-level annotations that allow the programmer to control the mapping of a program onto a particular machine. Concurrency is expressed explicitly in PRELUDE.

The annotations are used to describe and control the performance of the program, not its functionality. For example, annotations can be used to express the migration of objects and computations between processors in distributed memory architectures. Since annotations affect performance but not functionality, the annotations can be freely changed without introducing errors into the program. This separation of architecture-specific performance-related concerns from the rest of the program makes it relatively easy to port or tune the performance of a program.

PRELUDE provides three types of invocations: synchronous, unordered asynchronous, and ordered asynchronous. In a synchronous invocation, the calling thread performs the invocation. Unlike synchronous invocations, an asynchronous invocation conceptually forks a new thread to perform the invocation. Unordered asynchronous invocations avoid the software overhead required to maintain order and are therefore simpler and faster than ordered invocations. However, in many situations a thread can run concurrently with a sequence of calls it makes but these calls must be executed sequentially. A mechanism for ordered asynchronous calls leads to programs that are both simpler to understand and more efficient than ones in which the ordering is enforced by application-level synchronization. We introduce a new mechanism, pipe objects, to support ordered asynchronous calls.

The runtime system incorporates novel mechanisms for migrating data and computation in a distributed-memory multiprocessor. Existing systems have provided reasonable flexibility in mapping data onto parallel machines (via partitioning, replication, and migration), but have provided only simple mechanisms such as remote procedure calls for mapping logical threads. PRELUDE is designed to provide flexible control over the *migration* of computation, which allows a logical thread to be mapped onto a number of different physical threads as the computation represented by the logical thread migrates around the machine.

Parallel programs are difficult to test, debug, and tune. To accompany PRELUDE, we have built a retargetable simulator, PROTEUS [2], that provides extremely efficient instruction-level simulation for a wide range of MIMD multiprocessors. Because of its efficiency, accuracy and flexibility, PROTEUS has shown itself to be a useful tool for prototyping, testing, and tuning parallel programs. We have built prototypes of the PRELUDE compiler and runtime system using PROTEUS. We are experimenting with our implementation to evaluate the effectiveness of our current suite of mapping mechanisms and to understand what other mechanisms or changes to our current mechanisms are needed.

References

[1] W. Weihl, E. Brewer, A. Colbrook, C. Dellarocas, W. Hsieh, A. Joseph, C. Waldspurger and P. Wang. PRELUDE: A System for Portable Parallel Software. Technical Report MIT/LCS/TR-519, MIT Laboratory for Computer Science, October 1991.

[2] E.A. Brewer, C.N. Dellarocas, A. Colbrook and W.E. Weihl. PROTEUS: A high-performance parallel-architecture simulator. Technical Report MIT/LCS/TR-516, MIT Laboratory for Computer Science, September 1991.