

We use these computation graphs as the basis for representing programs for abstract machines. A computation graph is a program for some conceptual computation expressed in terms of the operations of an abstract machine. It is a directed graph where the nodes represent schedulable units of computation and arcs represent dependency relationships between source and sink pairs.

The computation is executed by transversal of the directed graph along the paths defined by the dependency relations associated with the arcs. Only the binding of operators to type instances is always associated with a node. Either the program defining the schedulable unit of computation or the type instances or both may arrive as values on arcs or be permanently bound to a node.

J.C. Browne,

Department of Computer Sciences, University of Texas, Austin, TX 78712

A node is therefore an abstraction for execution on a processor and memory. An arc is an abstract representation of a communication channel or a control channel that may include memory or execution of a synchronization protocol. The initial abstract machine will normally have a processor for every schedulable unit of computation and sufficient channels and control lines coupling the processors so that each dependency relation can be assigned its own set of resources.

The formulation proceeds through specification of successively more realizable abstract machines and mapping of program graphs to these abstract machines until a hardware-realizable machine is reached.

Performance is evaluated by establishing realizations, usually in software, for each abstract machine.

The framework given by graphical representation of programs allows a generic simulator, in which representations of schedulable units of computation and dependency relations can be implemented by parameterized templates.

Acknowledgment

The conceptual foundation for this research was established in the course of research on parallel computing sponsored by the Department of Energy, the National Science Foundation, and the Air Force Office of Scientific Research. Establishment of a research environment and experimental research is to be sponsored by DARPA as a part of the Strategic Computing Initiative.

References

- J.C. Browne, "Framework for Formulation and Analysis of Parallel Computation Structures," *Proc. 18th Hawaii Int'l Conf. System Science*, Jan. 1985, pp. 2-7.

Algorithm Design for Different Computer Architectures, Argonne National Laboratory

Within the last 10 years numerical analysts have realized the need to involve themselves not only in algorithm development, but also in the design of the software that embodies the numerical algorithms. Issues such as robustness, ease of use, and portability are now standard fare in any discussion of numerical algorithm design and implementation. The portability issue, in particular, becomes formidable as the evolution of new and exotic architectures makes a reality of the concepts of concurrent processing, shared memory, and pipelining, all of which are intended to increase performance. Ironically, it is tempting to assume that portability must always carry with it an unacceptable degradation in efficiency for any machine architecture. We

contend that this assumption is erroneous and that its widespread adoption could seriously hamper the effective use of future machines.

Future architectures promise a profusion of computing environments. The existing forerunners have already given many software developers cause to reexamine the underlying algorithms for efficiency. However, it seems an unnecessary effort to recast these algorithms with only one computer in mind, regardless of its speed. The efficiency of an algorithm should not be discussed in terms of its realization as a computer program. Even within a single architectural class, the features of one system may improve the performance of a given program, while features of another system may have the opposite effect.

Software developers should begin to identify classes of problems suitable for parallel implementation and to develop efficient algorithms for each area. With such a wide variety of computer systems and architectures in use or proposed, the challenge for algorithm designers is to develop algorithms—and ultimately software—that are both efficient and portable.

Research approaches. There appear to be three approaches to improving algorithm efficiency and portability. They are not mutually exclusive, and each can contribute to an effective solution.

The first approach is to express algorithms in terms of modules at a high level of granularity. When software is moved from one architecture

