1994

A Large-Grain Parallel Programming Environment for Non-Programmers

Lewis, Ted

IEEE

A LARGE-GRAIN PARALLEL PROGRAMMING ENVIRONMENT FOR NON-PROGRAMMERS

Ted Lewis
Computer Science Dept
Naval Postgraduate School
Monterey, CA. 93943-5118
lewis@cs.nps.navy.mil

Abstract -- Banger is a parallel programming environment used by non-professional programmers to write explicitly parallel large-grain parallel programs. The goals of Banger are: 1. extreme ease of use, 2. immediate feedback, and 3. machine-independence. Banger is based on three principles: 1. separation of parallel programming-in-the-large from sequential programming-in-the-small, 2. separation of programming environment from target machine dependency, and 3. instant feedback to user wherever possible.

INTRODUCTION

Great progress has been made in the maturity of parallel processor hardware, but parallel processor software has remained a challenging problem, i.e. parallel programming will remain a challenging research problem for computer scientists over then next decade. In the meantime, what about non-computer scientists? There are many disciplines of science and engineering that require the performance of parallel machines and the subject expertise (domain knowledge) of a non-computer scientist.

Such applications of parallel processors are typically small, short-lived, simple pieces of software. It is desirable to design, debug, and run these "quick-and-dirty" parallel programs within a few days rather than months or years, by one or two scientists rather than a large team of programmers. This mode of development is in sharp contrast to the large-scale software engineering projects most familiar to professional programmers.

The problem addressed by Banger is the problem faced by non-programmers in non-computer science disciplines: how to write a parallel program that solves a problem in some branch of science or engineering without becoming an expert in computer science or parallel programming.

The first step in using Banger is to draw a hierarchical dataflow graph of the application, leaving the coding details for later. Next, we define a target machine on which Banger automatically schedules the parallel tasks of the application. Third, we use a novel programmable pocket calculator metaphor to specify algorithms as small sequential tasks. Finally, we generate the code.

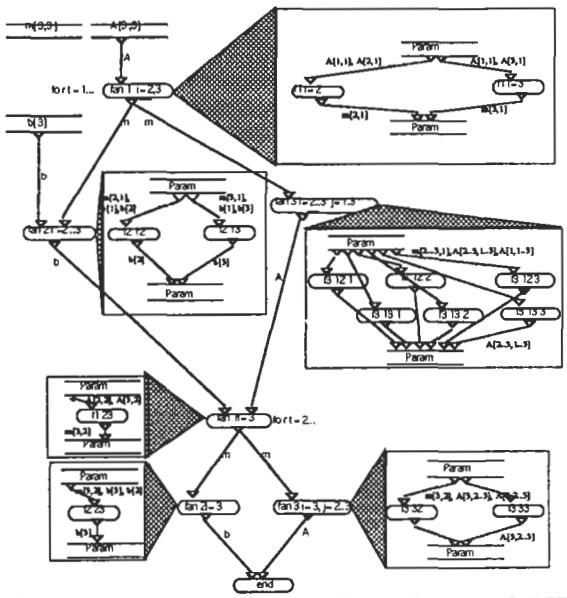

Figure 1. Hierarchical Dataflow Graph of PITL Design for an LU Decomposition of 3-by-3 System: Ax=b

Proceedings of the International Conference on Parallel Processing (ICPP'94)
0-8493-2493-9/0-6 $20.00 © 1994 IEEE

PROGRAMMING-IN-THE-LARGE

Programming can be divided into two major phases: programming-in-the-large (PITL), and programming-in-the-small (PITS). We use the principle of separating PITL and PITS to distinguish between parallel parts and sequential parts, in a parallel application. This is our first principle, which is expressed succinctly, below:

> A large-grained parallel program is a PITL hierarchical dataflow graph consisting of nodes and arcs. The nodes define hierarchical decompositions, or PITS sequential tasks, and the arcs define precedence relations between nodes.

This definition is illustrated in Figure 1, where we show a 2-level hierarchical PITL dataflow graph representing the major tasks in a 3-by-3 LUD algorithm for solving a linear system, Ax=b when n=3. Storage is shown as open rectangles labeled with the data they contain. Tasks are shown as oval-shaped nodes labeled with a comment, e.g. fan1, f121, etc. Bold-lined nodes are decomposable into lower-level dataflow graphs shown as expansions, e.g. lower-level graphs. Arcs establish precedents created by either control flow or dataflow dependencies, and are labeled with the names of the variables containing data which flows in/out of each node.

In general, any large-grained dataflow calculation, not involving loops or branches, can be expressed as a hierarchical dataflow diagram as illustrated by this small example.

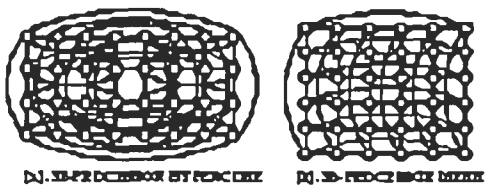

Figure 2. Two Examples of Network Interconnection Topologies Supported by Banger

SEPARATION OF PROGRAM FROM MACHINE

Principle two, advocating separation of program from machine, is achieved as a byproduct of the PITL hierarchical dataflow design. But, how is runtime efficiency retained if the target machine is not considered during design? Banger recovers much of the potentially reduced efficiency by optimally scheduling the PITL design onto a specific parallel processor. This is the second principle governing the design of Banger:

> Machine-independent parallel programming can be made efficient by optimal scheduling heuristics which find the shortest elapsed execution time schedule for a specific parallel program, given a specific target machine.

Matching an arbitrary large-grained program design to an arbitrary parallel computer architecture is automatically performed by a scheduling heuristic that finds the shortest elapsed execution time schedule for a specific target machine. A program is tailored to a certain machine by considering the following characteristics of the target machine:

1. Processor speed
2. Process startup time
3. Message passing startup time
4. Message transmission speed

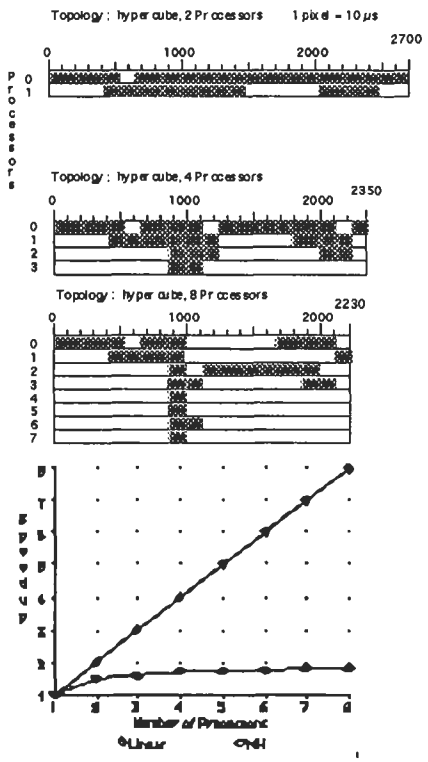

Figure 3. Some Gantt Chart Schedules and a Speedup Chart Automatically Generated by Banger.

Proceedings of the International Conference on Parallel Processing (ICPP'94)
0-8493-2493-9/0-6 $20.00 © 1994 IEEE

In addition, the target machine's interconnection network topology (if it is a distributed-memory machine) is entered by the user as another graph, see Figure 2. Banger supports hypercubes, meshes, trees, stars, and fully-connected topologies.

Banger uses the scheduling heuristics implemented in PPSE [1]. Figure 3 shows several schedules obtained from Banger when mapping the PITL design of Figure 1 onto various sized hypercube interconnected machines. Also shown is a speedup prediction graph obtained by mapping the PITL design onto 2,4, and 8 hypercube processors.

THE CALCULATOR METAPHOR

The third principle of our approach says to use simplifying metaphors wherever possible. Because Banger is aimed at scientific non-programmers, we adopted a calculator metaphor as a friendly user interface to Banger's PITS language. The calculator is used when defining a PITS sequential routine for each primitive node of the hierarchical dataflow graph. Principle three dictates an appropriate user interface for Banger:

> For scientific programmers, an acceptable programming metaphor is a simulated pocket calculator containing simple programming constructs, scientific and engineering functions, constants, and formulas, and some means of obtaining numerical results, upon demand.

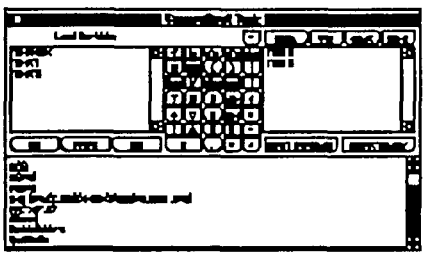

Figure 4. Calculator Panel for defining each sequential task. The SquareRoot Task uses Newton-Raphson approximation to compute: $x = \sqrt{a}$.

In Figure 4. a list of input/output variables for the node is shown in the upper right-hand window, a list of local variables is shown in the upper left-hand window, and a panel of programming buttons is shown in the upper middle of the calculator. The lower window holds a textual representation of the node routine, in a simplified programming language.

RESULTS

As far as the author knows, Banger is the only parallel programming environment for non-programmers. It is an entirely new approach to parallel programming.

While limited in its current form, Banger incorporates features that may be useful in more sophisticated parallel programming environments. For example, the ability to schedule parallel program designs onto target machines is useful in any parallel programming environment. The ability to perform trial runs of tasks or entire programs is certainly valuable in any programming environment.

More significantly, Banger demonstrates four principles that may be followed by designers of future parallel programming environments, regardless of the level of expertise of the user:

1. separation of parallel programming-in-the-large from sequential programming-in-the-small demonstrates that it is possible to reduce the complexity of parallel programming by this division scheme. This principle is made practical by the scheduling heuristics of Banger.

2. separation of programming environment from target machine dependency demonstrates that it is possible, within the large-grained paradigm, to achieve machine-independent parallel programming. While we have not shown that this principle applies to a broader range of parallel programming paradigms, we are confident that Banger can be extended to encompass fine-grained parallelism through the use of machine-independent data-parallel constructs.

3. use of simplifying metaphors such as the programmable calculator demonstrates the power of graphical user interface techniques in programming environment design and implementation. Users simply do not need to learn and recall arcane syntactic expressions, even when dealing with parallelism.

4. instant feedback to the user wherever possible, especially through graphical displays and animations, is invaluable to non-programmers because it is believed to be a major contributor to early defect removal.

Proceedings of the International Conference on Parallel Processing (ICPP'94)
0-8493-2493-9/0-6 $20.00 © 1994 IEEE

Banger is implemented as an experimental prototype on an easy-to-use, pervasive personal computer, so it is highly available to anyone, whether a computer scientist or not. Copies may be obtained directly from the author. However, Banger does not currently support automatic code generation. A number of program generators for a variety of systems are under development. These code generators will provide a final step in producing a production-level parallel program for specific target parallel computer systems.

Acknowledgements. Banger is an extension of PPSE, which was written by Rob Currey in 1991 [2]. Pongsaya Hongswadhi added the pocket calculator and interpreter capability in 1993, and improved the user interface. Scheduling heuristics were developed over a period of six years by Boontee Kruatrachue, Hesham El-Rewini, and the author.

REFERENCES

[2]. Currey, R. W. and T. G. Lewis, "User Manual for The Parallel Programming Support Environment," Technical Report (91-60-14). Department of Computer Science, Oregon State University, OR, 1991.

[1]. El-Rewini, H., and T. Lewis, "Scheduling Parallel Program Tasks onto Arbitrary Target Machines", *Journal of Parallel and Distributed Computing*, vol 9, pp. 138-153, (June 1990).

Proceedings of the International Conference on Parallel Processing (ICPP'94)
0-8493-2493-9/0-6 $20.00 © 1994IEEE