AIM

Association for Information and Image Management

1100 Wayne Avenue, Suite 1100
Silver Spring, Maryland 20910

301/587-8202

Centimeter

Inches

MANUFACTURED TO AIM STANDARDS
BY APPLIED IMAGE, INC.

1 of 1

An Informal Introduction to Parallel Processors*

by

Kevin W. Hopkins
Southwest Baptist University
Bolivar, MO 65613

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

*Work performed at Argonne National Laboratory, a contract Laboratory of the United States Department of Energy, while a Faculty Research Participant.
Program administered by the Argonne Division of Educational Programs with funding provided by the U. S. Department of Energy.

MASTER

The submitted manuscript has been authored by a contractor of the U. S. Government under contract No. W-31-109-ENG-38. Accordingly, the U. S. Government retains a nonexclusive, royalty-free license to publish or reproduce the published form of this contribution, or allow others to do so, for U. S. Government purposes.

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

875

1. Introduction. In the summer of 1992, I had the opportunity to participate in a Faculty Research Program at Argonne National Laboratory. During that time I came in contact with some developments in computer science that were unfamiliar to me as a mathematician. Since most mathematicians have some exposure to computers, but certainly are not computer scientists, I felt it was appropriate to write a paper summarizing my introduction to parallel processing. This paper is not meant to be a full explanation of the topic, but an informal introduction for the "mathematical layman."

2. My Work At Argonne. My project at Argonne the summer of 1992 was supervised by Dr. Jim Boyle. I worked on transforming code written in pure functional Lisp to Fortran code to run on distributed-memory parallel processors such as the 8-processor Intel iPSC (located at Argonne) and the 512-processor Intel Touchstone Delta machine (located at Caltech). To perform this project, I had to learn three things: the transformation system [6], the Lisp programming language [6], and the basics of distributed-memory parallel machines.

3. Parallel Processors. Parallel processing is a rather new and promising field of research in computer science. (As an indication of how current the field of parallel processing is, I note that a search of the Argonne Library's database revealed 488 titles matching the keyword "parallel"; most of those were produced since 1985.) Rick Stevens, director of Argonne's High-Performance Computing and Communications Program, stated the following about parallel processing:

"The past decade has seen the emergence of a new way of doing science and engineering. The new mode of 'computational science' is poised to join theory and experiment as a third approach to solving scientific and engineering problems. Its widespread use will fundamentally alter the way

modern researchers attack problems." [9].

To learning about parallel processors I had a series of discussions with Dr. Boyle and attended a few seminars. A parallel processor is a set of independent processors interconnected in such a way that communication can take place between the processors as each performs its separate tasks. A basic parallel processor that most are familiar with is a network of workstations or personal computers, with each processor doing the task assigned by its own user. More typical of a parallel machine is one that allows for breaking up one task and having each processor performing a portion of that task. Thus the job is finished more quickly.

There are many reasons an increase in speed may be important or even necessary to attempt a problem. Many numerical techniques trade off accuracy or significantly limit the model to achieve reasonable execution time. For many simulations, a "real-time" speed is required (see [11, p. 51] for a discussion of a real-time simulation). These applications have the need for faster execution than "ordinary" computers provide. A series of Grand Challenge problems have been proposed that demand the use of parallel computers.

"The national mandate to resolve fundamental problems in science and engineering requires the assimilation of computational science and high-performance computer technology ... Improved massively parallel systems must perform faster and more accurate arithmetic calculations." [1, p. 1]

The cost for increasing the speed of one machine is currently more than the cost of using several "low-cost" processors in parallel to achieve the same or better increase in speed. Some examples of the uses of parallel processing at Argonne are climate modeling (with a goal of producing in ten years climate models that are

10,000 times or more faster than the current model) and modeling type-II superconductors (for more examples, see [4, 8, 10]).

Ideally, if one doubles the number of processors, one halves the time of the task. In practice, such speedup is not achieved, especially as the number of processors increases [5]. There are two main reasons for this. One reason is that, in any task, there are subtasks that need to be performed sequentially (first one and then the second) rather than in parallel (both at the same time). The second reason is that for parallel programs there is extra program overhead that must be included to guarantee that two processors do not attempt to do the same work or alter the same piece of data at the same time (see [7] for a discussion of locking to ensure this does not happen). Both of these reasons detract from the theoretical speedup.

In the first case, the lack of parallel work means that some -- and, as the number of processors increases, most -- of the processors must sit idle until there is work to do. The loss of speedup is usually a consequence of the algorithm being programmed and can be lessened only by changing the algorithm. In the second case, the contention to get tasks and data grows as the number of processors grows. This situation can be improved by certain programming choices and hardware considerations, but still imposes an upper bound on the amount of speedup that is possible.

It is the programming choices of which algorithm to use and of how one divides the job that make programming in parallel far more challenging than programming a sequential machine. Dividing the job is called problem decomposition. Two methods are generally applied: domain decomposition, where the data is divided among the processors, and control decomposition, where the

tasks are divided among the processors. In the Lisp-to-Fortran conversion, all the parallelism is in control decomposition (for more information on parallel programming, see [12]).

I gained familiarity with two main types of parallel machines: global memory and distributed memory. The reason I needed to be familiar with both types of machines is that the distributed-memory version of the transformations was developed by starting with the global-memory version (see [3] for a discussion of the results of a program that was transformed to a global-memory version). Global-memory machines have one memory that any of the processors can access at any time, within the constraints of locking as mentioned in the preceding paragraph. A diagram of this setup is in Figure 1. The distributed-memory machines are such that each machine has its own local memory. This cuts down on much of the data locking overhead, but increases message-passing overhead, as now one processor may need the data that resides on another processor and so must ask for it by sending a message to that processor and waiting for a response. A diagram of this setup is in Figure 2.

4. Results. My main task was to test and improve the transformations (written mainly by Jim Boyle; David Paik, a student at Argonne in summer 1991; Jeanine Gild, a student at Argonne in spring 1992; and Peter Schuetz, a foreign visitor to Argonne in 1988) to transform pure Lisp to Fortran for distributed-memory parallel machines. In the distributed-memory case, one must know not only the address of the data, but what processor, or node, has the data. Thus the data must be split into two pieces, one piece indicating the node and the other the address on that node. I tested the split derivation on sequential machines to be sure it was working properly before the parallelism issues were added. Time did not allow me to test

the programs on any parallel machines, although I did spend time testing the transformations to produce the code.

One task that faced me was to address the added "cost" to access data in the distributed-memory case. For a Lisp programmer, the function calls `car(l)` and `cdr(l)` are relatively "cheap" (do not take a great deal of time to perform), and so they are used freely in the program. When the program is converted to Fortran for distributed-memory machines, however, these calls are potentially much more "expensive," as the data may not reside on that processor.

To deal with this problem, we introduced a pseudofunction `carncdr` that would return both the `car` and the `cdr` whenever it was called. This `carncdr` function was placed as early in the code as possible, so the data retrieved from another processor (the plan is for Ken Dritz to write the routines for accessing `car` and `cdr` from the other processor) could be used at later times from the processor's own memory, rather than making numerous requests to another processor for the same pieces of data. This situation is especially prevalent in `if/then/else` statements, where `car`, `cdr`, or both may appear in many places.

5. Conclusions. The field of parallel computing is still open for many research topics, in terms of both software and hardware development. The two machines mentioned in this paper had 8 and 512 processors. Machines with 1000 or more processors, the so-called massively parallel machines are being studied separately from the "smaller" parallel machines (in October 1992 the Fourth Symposium on the Frontiers of Massively Parallel Computation was held in McLean, Virginia). There is a feeling among some that the real future of parallel computing lies in these massively parallel machines:

"A machine that is massively parallel - that is, one that can handle very large

numbers of operations concurrently - is essential for creating the advanced climate model that scientists hope to complete by the end of the decade." [2, p. 46]

Argonne has initiated a High-Performance Computing and Communications Program that involves collaboration with IBM to produce a massively parallel machine capable of 1 teraflop (one trillion floating point operations per second) by 1995, about 1000 times faster than the Intel iPSC. Other DOE laboratories have or are proposing other High Performance Computing and Communications Programs. The dividends in parallel are just beginning to come in. The future of this area is bright. I think we will all be amazed at the advances that will come about in the near future.

Figure 1
Global-Memory Parallel Processor

Figure 2
Distributed-Memory Parallel Processor

References

1. Argonne National Laboratory, *Computational Science and Engineering at Argonne National Laboratory*.
2. Argonne National Laboratory, *Research Highlights 1991-92*.
3. J. M. Boyle, "Abstract programming and program transformation-an approach to reusing programs," *Software Reusability*, Vol I, ed. Alan J. Perlis, ACM Press, Addison-Wesley Publishing Company, Reading, Massachusetts pp. 361-413 (1989).
4. E. Chiricozzi and A. D'Amico, eds., *Parallel Processing and Applications: Proceedings of the International Conference on Parallel Processing and Applications*, North Holland, New York (1987).
5. K. W. Dritz and J. M. Boyle, "Beyond "speedup": performance analysis of parallel programs," ANL-87-7, Argonne National Laboratory, Argonne Illinois (Jan. 1989).
6. K. W. Hopkins, "An informal introduction to program transformation," Preprint MCS-P319-0892, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Illinois (August 1992).
7. E. Lusk, R. Overbeek, J. Boyle, R. Butler, T. Disz, B. Glickfeld, J. Patterson, and R. Stevens, *Portable Programs for Parallel Processors*, Holt, Rinehart and Winston, Inc., New York (1987).
8. R. W. Numrich, *Supercomputer Applications*, Plenum Press, New York (1985).
9. R. L. Stevens, "Argonne's high performance computing program," Seminar

- held July 16, 1992 at Argonne National Laboratory, Argonne, Illinois (1992).
10. J. L. Sanz, *Opportunities and Constraints of Parallel Computing*, Springer-Verlag, New York (1989).
 11. L. Smarr and C. E. Catlett, "Metacomputing," *Communications of the ACM*, 35(6) pp. 45-52 (June 1992).
 12. *Using the Intel iPSC/860*, Mathematics and Computer Science Division Technical Memorandum ANL/MCS-TM-143, Argonne National Laboratory, Argonne, Illinois (1990).

DATE

FILMED

8 / 31 / 94

END