A major purpose of the Technical Information Center is to provide the broadest dissemination possible of information contained in DOE's Research and Development Reports to business, industry, the academic community, and federal, state and local governments.

Although a small portion of this report is not reproducible, it is being made available to expedite the availability of information on the research discussed herein.

LA-UR--84-3118

DE85 000653

NOTICE

PORTIONS OF THIS REPORT ARE ILLEGIBLE.

It has been reproduced from the best available copy to permit the broadest possible availability.

Los Alamos National Laboratory is operated by the University of California for the United States Department of Energy under contract W-7405-ENG-36

CONF - 841242 -- 2

TITLE: PARALLEL PROCESSING EXPERIENCES ON THE DENELCOR HEP COMPUTER

AUTHOR(S): Ann H. Hayes

SUBMITTED TO: International Workshop on Modelling and Performance
Evaluation of Parallel Systems
Grenoble, France
December 13-18, 1984

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.

By acceptance of this article the publisher recognizes that the U.S. Government retains a nonexclusive royalty-free license to publish or reproduce the published form of this contribution or to allow others to do so, for U.S. Government purposes.

The Los Alamos National Laboratory requests that the publisher identify this article as work performed under the auspices of the U.S. Department of Energy.

Los Alamos Los Alamos National Laboratory
Los Alamos, New Mexico 87545

MASTER

gsw

PARALLEL PROCESSING EXPERIENCES ON THE DENELCOR HEP COMPUTER

by

**Ann H. Hayes
Los Alamos National Laboratory
Computer Research and Applications Group
MS B265
Los Alamos, New Mexico 87545**

ABSTRACT

Recent experiments conducted on a Denelcor HEP (Heterogeneous Element Processor) computer are discussed in this paper. Algorithm research was done on four types of problems of interest to the Los Alamos National Laboratory: 1) Monte Carlo, using GAMTAB, in which the interaction of photons with matter is analyzed; 2) Hydrodynamics; 3) Reactor Safety, in which the operation of a nuclear reactor is simulated and 4) Particle-in-Cell, in which electrostatic interaction of plasma beams are studied. Means of maximizing programming efficiency are analysed with ways of speeding up processing determined.

I. REASONS FOR USING A PARALLEL PROCESSOR

As single-processor computers reach the limit of their capabilities, computer scientists must develop alternative strategies to meet the computational needs of physicists and other scientists. Parallel processing, or the application of several processing elements to solve a problem, may provide some of the performance gains necessary to meet these needs.

In July 1983, Los Alamos National Laboratory leased a Denelcor HEP (Heterogeneous Element Processor) computer to do algorithm research on problems of

interest to the Laboratory. Our goal is effective use of multiprocessor supercomputers as they become available in the marketplace.

Our purpose has been to determine optimal algorithms and techniques suitable for parallel processing computations. Although multiprocessor supercomputers are now entering the marketplace, techniques for their efficient utilization are still in their infancy. There are two modes of operation that may be associated with multiple-CPU computers. First, as a means to increase throughput (that is, by executing different jobs in each processor), and second, by executing, through several processors, those jobs whose complexity or execution time precludes their currently being run. It is the latter use that we see as being the most important for the Laboratory's future computing needs.

II. MATCHING NEEDS AND CAPABILITIES

For several years, we have had a project in the Computing and Communications Division at Los Alamos National Laboratory to identify and quantify the computing workload of Laboratory scientists.[1] By developing and using software tools to measure program parameters such as vector length and instruction sequences, as well as the actual amount of computer time accounted for by various codes, we have been able to identify heavily used applications and concentrate our efforts on those areas of greatest concern to Los Alamos users. Using the Denelcor HEP computer as a research tool to develop optimal algorithms for a parallel environment, four problem areas, representative of the Laboratory workload, have been selected: Monte Carlo, hydrodynamics, reactor safety, and particle-in-cell calculations.

Objectives for this effort include

- * To develop algorithms and software tools to optimize use of parallel computation.

- * To determine the difficulty of writing code for parallel processors.
- * To obtain quantitative data through experimentation on Laboratory codes.
- * To provide exposure for experimental computers that appear promising.

Our approach has been to test small benchmark versions of prototypical Laboratory applications on multiprocessor machines, determining the problems and benefits found in each generic area. After we establish the algorithm's feasibility, we test larger, more realistic versions of these applications.

We obtain measurements of code execution, using various code parameters and system configurations. We are primarily interested in the speedup of an algorithm executed in parallel mode over a comparable algorithm run in serial mode.

Speedup (S) is computed as

$$S = \frac{\text{execution time/1 processor}}{\text{execution time/n processors}} .$$

To obtain greatest speedup, processors must be uniformly used; that is, applications must be designed so that most processors are doing useful work most of the time. Results show that parallelism must exceed 90% for cost-effective speedup.

III. HEP ARCHITECTURE

The Denelcor HEP computer is a general-purpose MIMD (Multiple Instruction/Multiple Data) computer capable of parallel processing within a single processor. Both its instructions and word size are 64 bits, with floating point capability and a Fortran compiler available. A control unit, functional units, and four types of memory (program, data, register, and constant) are associated with each process execution module (PEM). Each PEM may access additional shared memory through a hardware switch, and up to sixteen PEMs may be interconnected through the switch. The Los Alamos HEP consists of one PEM, with 256K words of program memory and 512K words of data memory.[2, 3] User programs can be executed in parallel by dividing them into sections, called *processes*. A maximum of 50 user processes may be executing concurrently in each PEM.

Parallelism is implemented on the HEP at the instruction level. At each segment of the eight-stage pipeline, a different process may be executing. Instructions enter the pipeline from a hardware queue, where they are decoded and operands obtained. Since each instruction (except for divides and memory fetches) requires eight stages to complete, theoretically eight active processes will keep the pipeline filled and thus assure full CPU utilization. In reality, 12-13 processes per PEM are required, since memory fetches exit from the pipeline immediately and are routed to a secondary scheduling queue. For a data memory fetch, the second queue accesses a memory module through the switch. At completion of the memory access, the process state is readmitted to the first queue for later execution. Since the basic machine cycle is 100ns, it is possible to achieve a performance speed of 10 Mflops/PEM. Figures 1 and 2 show, respectively, the general HEP architecture and the architecture of a PEM.

IV. FORTRAN EXTENSIONS FOR THE HEP COMPUTER

Programming the HEP for parallel operations is straightforward. Parallel constructs are added by the user with two Fortran extensions. Variables that are to be shared by several processes and that must be updated asynchronously are prefixed with a dollar sign (\$). In the HEP hardware, data variables have an extra bit, designated as the status bit. Whenever data is stored in a variable, the bit is set to full; conversely, the bit is set to empty when data is read. A process may read an asynchronous variable only when the status bit is set to full and may write into a variable only when the bit is set to empty. The status bit value is held in reserve during the instruction execution. There are two sets of memory instructions on the HEP, one set ignores the status bit and regards data as if it is accessible to all processes equally; a second set is invoked whenever variables prefixed with a \$ are specified, and operations proceed based on the status bit. If an instruction cannot proceed because of an adverse condition of the status bit, the instruction is rescheduled for the next time the process is serviced. These instructions guarantee that synchronization is effected when necessary. Processes are thus blocked from accessing data until proper updating has occurred. These barriers, as they are called, guarantee that critical synchronization points are handled properly, usually at process termination. For parallel operation to obtain maximum performance, synchronization points must be kept to a minimum. In the HEP, testing of the status bit is done in the hardware, not through operating system intervention, and process synchronization is very efficient.

The second Fortran extension is the CREATE syntax, used to spawn processes. This is analogous to the CALL statement in Fortran, except that the calling program continues to execute along with all CREATE processes.[4]

DO loops may be executed either by prescheduling or self-scheduling techniques. In the latter mode, independent tasks are executed from a queue, which is reexamined upon each task's completion to determine if there are further calculations to be performed. For DO loops that contain variable run times for different indices, this assures that the distribution of work is equitable among the processes.

V. RESULTS

Brief descriptions of the selected codes and the results we obtained follow.

Monte Carlo: this class of problem employs a random-number generator to produce a model of statistically valid entities simulating a natural process. Typically, Monte Carlo codes do not vectorize well because of the decision-making nature required of the computations, which results in program branches. However, this asynchronous feature makes execution on an MIMD computer especially practical.

GAMTAB, an 800-line Monte Carlo transport code, was used for our studies. This code analyzes the interaction of photons with matter. As a result of various physical processes during the lifetime of a photon particle, offspring are produced, photons are paired or split, and histories are generated. Each history generation may be considered an independent task.

Because of the asynchronous nature of the calculations, results were nonreproducible until a pseudorandom-number generator was written and added to the code. This algorithm includes a tree structure, in which unique random-number seeds are assigned to each particle, and traversal of the tree assures reproducibility, regardless of timing variations. (While Monte Carlo is statistically valid with any sequence of random numbers, for tracking purposes, code developers require identical answers over successive runs.)

Hydrodynamics: the benchmark code, SIMPLE, was used for this investigation. This code contains the basic features of a hydrodynamic code in an abbreviated form. An alternating direction algorithm (ADI) to solve the heat diffusion equations is the principal algorithm used in this calculation. In this method, a mesh is swept first in one direction, then in the other, to achieve maximum independence of operations. By self-scheduling boundary calculations along with interior mesh calculations, maximum parallelism was achieved. Synchronization was imposed, to assure that interior mesh calculations preceded boundary calculations. This code is written almost entirely in parallel.

Reactor Safety: a subset (4,000 lines) of TRAC, a highly used Laboratory code simulating the operation of a water-cooled nuclear reactor, was used. This code consists of several modules, each representing a portion of the reactor, such as a ring or a pipe. Because of the large granularity and independence of the computations, this problem is inherently suited to parallel execution. Following an initialization calculation, each of the modules may be executed independently and concurrently with others, until a final synchronization point is reached, where data must be collected preparatory to the next iteration.[5]

The major difficulty in converting a code such as this is in locating variable dependencies that are not evident while the modules are executing serially.

Particle-in-Cell: Simulating the flow of particles through a plasma is a process found in many Los Alamos projects. Because of the irregular nature of the motion of the particles, this type of problem does not vectorize well on current supercomputers. However, by self-scheduling particles over several processes, it was found that parallel processing of this activity proceeds smoothly.[6]

For the selected codes, conversion to the HEP was done primarily by our dividing outer DO loops among several processes. This amounted to our partition-

ing the codes into sections of large granularity. Often the vectorized inner loops, optimized for Cray execution, were left intact. For future MIMD vectorizing multiprocessors, the programmer's controlled combination of parallelization over outer DO loops and vectorization of inner DO loops, may provide further significant speedups.

As an immediate way to efficiently use parallel processing computers, this simple conversion technique has proved effective. The large investment in Fortran codes at the Laboratory makes this an important conclusion. As we become more familiar and comfortable with parallel processing, new algorithms, specifically designed for parallel implementation, are certain to be developed.

The codes we studied had over 90% execution in parallel mode. Note, however, that we have not yet addressed the important issues of graphics or I/O—the codes are computation-intensive. Speedups were nearly linear in all cases for up to twelve processes, after which no further gains were evident. By our minimizing synchronization points, little overhead was introduced in the synchronization process.

While it is difficult for us to list firm rules for writing parallel code, one observation can be stated. When optimizing code for a vector computer, one begins optimization at the innermost structure, vectorizing to the outermost level. For one to obtain maximum performance and parallelism on a parallel machine, this process is reversed; that is, one initiates optimization from the outermost level on a global basis, and proceeds inward.

VI. CONCLUSIONS

Our initial results strongly support our belief that parallel processing may bring substantial performance gains. If one can obtain a high level of code parallelism, there is a potential for significant process speedup. Our experiments are valid

only for a modest number of processors (up to 8); further experimentation must be undertaken with systems containing a larger number of processors (up to 64). As processor communication becomes more complex, issues previously hidden are likely to surface.

Clearly, work must still be done in the development of software tools for parallel processors, both in code analysis and debugging. Program variable dependencies and non-reproducibility of results require that the user be supplied with at least semi-automatic tools if parallel processing is to be cost effective.

From our experiences on the HEP computer we conclude that a small number of extensions to the language, in addition to a small set of control structures, may suffice to provide the constructs necessary for writing and executing parallel code.

REFERENCES

1. J. L. Martin, I. Y. Bucher, and T. T. Warnock, "Workload Characteristics for Vector Computers: Tools and Techniques," Los Alamos National Laboratory document LA-UR-83-305 (August 1983).
2. *HEP Principles of Operation*, (Denelcor, Inc.; Aurora, Colo.: 1981).
3. B. J. Smith, "A Pipelined Shared Resource MIMD Computer," International Conference on Parallel Processing (Bellaire, Mich., 1978).
4. J. W. Moore, "The HEP Parallel Processor," *Los Alamos Science* (Los Alamos, N.M., Fall 1983).
5. J. W. Moore, "Experience Reprogramming TRAC for the HEP Computer," Los Alamos National Laboratory document LA-UR-83-27 (1983).

6. A. H. Hayes, "Quarterly Status Report of the Deneicor HEP at Los Alamos," Los Alamos National Laboratory memorandum (May 1984).

FIGURE CAPTIONS

Figure 1. A high-level view of the HEP architecture.

Figure 2. PEM architecture.

HEP ARCHITECTURE

Los Alamos