LA-UR-82-456

Conf-820606--2

LA-UR--82-456

DE82 008174

TITLE: PARALLEL PROCESSING A LARGE SCIENTIFIC PROBLEM

AUTHOR(S): Robert Hiromoto

DISCLAIMER

SUBMITTED TO National Computer Conference, Houston, TX, June 1982

MASTER

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

FORM NO. 836 R4

PARALLEL PROCESSING A LARGE SCIENTIFIC PROBLEM

by Robert Hiromoto

Los Alamos National Laboratory

MS 265, C-3

Los Alamos, New Mexico

Phone (505) 667-7028

ABSTRACT

We discuss a parallel-processing experiment that uses a particle-in-cell (PIC) code to study the feasibility of doing large-scale scientific calculations on multiple-processor architectures. A multithread version of this Los Alamos PIC code was successfully implemented and timed on a UNIVAC System 1100/80 computer. Use of a single copy of the instruction stream, and common memory to hold data, eliminated data transmission between processors. The multiple-processing algorithm exploits the PIC code's high degree of large, independent tasks, as well as the configuration of the UNIVAC System 1100/80. Timing results for the multithread version of the PIC code using one, two, three, and four identical processors are given and are shown to have promising speedup times when compared to the overall run times measured for a single-thread version of the PIC code.

Keywords: Parallel processing, activities, processors, common memory, algorithm, multiple-processor architecture.

Robert Hiromoto
Parallel Processing

INTRODUCTION

Anticipating a need for increased computational speed[1] for laboratory codes (which is unlikely to be attained by single processor systems), we have initiated studies to test the feasibility of doing parallel processing on multiple-processor architectures.[2] In part, our hope is to learn about multiple-processor architectures, the compatibility of algorithms with particular parallel processing environments, parallel processing speedups as a function of the number of processors, and the desirable characteristics of multiple-processor architectures in general.

This paper presents the results of our investigation concerning the feasibility of parallel processing a specified scientific problem on a commercially available multiple-processor system and to determine the computational speedups as a function of the number of processors employed. The problem used in this experiment involves a particle-in-cell (PIC) method for simulating the electrostatic interactions of a collisionless plasma. We first outline the PIC algorithm and graphically describe its parallel-processing structure as implemented in our experiment. A general description of the UNIVAC System 1100/80 is then given, followed by a discussion of the implementation of the PIC code on that system. Results of our experiment are given, showing overall computational speedups as a function of the number of processors and the equivalent number of parallel acti ies.

PARTICLE-IN-CELL

The problem selected for our parallel-processing experiment models the collisionless, electrostatic interaction between two superimposed plasma beams with a relative drift velocity.[3] The code uses a particle-in-cell method for studying the interaction and resulting motion of the charged particles in this simulation.[4] This code is of general interest to us because it represents a class of algorithms exhibiting limited vector capabilities for implementation on our vector computers. Due to the PIC algorithm (discussed below), the conversion to parallel processing was made with relative ease.

PIC algorithm

The particle-in-cell method used in this study decomposes a region of space into a collection of cells. These cells are then used for tracking particle movement, and assist in evaluating relevant physical properties. An initialization stage sets up two ensembles of charged particles (we shall use particles to mean charged particles throughout this paper) constituting the two superimposed, collisionless plasma beams. During this initialization, the particles are distributed uniformly in space and randomly in velocity. The movement of particles is discretized in a time step (dt).

During each computational time step of the simulation (see
Figure 1 Figure 1), cell-centered charges (C) are calculated by linearly weighting each particle's charge contribution to the four nearest-neighbor cell centers. Using this charge distribution, Poisson's equation with periodic boundary conditicns is solved for the associated electrostatic potential (ϕ) on the grid of cell centers, with the resulting electric (E) field interpolated to individual particle positions. Under this E field, each
Figure 2 particle's position and velocity (see Figure 2) are advanced (pushed).

PIC parallel-processing structure

The computational structure of the PIC algorithm, as implemented on the UNIVAC System 1100/80, takes advantage of the large, natural computational divisions of the particle initialization and aspects of the particle-in-cell calculations.
Figure 3 Figure 3 graphically displays the multi/single-thread diagram of our PIC code, with accompanying definitions of the respective calculation(s) each thread performs.

IMPLEMENTATION ON A UNIVAC SYSTEM 1100/80

Our parallel-processing version of the PIC code was implemented on a UNIVAC 1100/80 multiple-processor system.* The System 1100/80 may be configured with from one to four processors. UNIVAC's designation for its System 1100/80 with a one-, two-, three-, or four-processor configuration is denoted by 1100/81, 1100/82, 1100/83, or 1100/84, respectively.

---

*Provided for our use by the Computer Operations Department, Sandia National Laboratories, Albuquerque, New Mexico.

A global software manager (EXEC) executes out of all processors and, coupled with hardware devices, drives the multiple-processor architecture of the System 1100/80. The aggregate of processors share a common memory, which allows for multiple-program execution for tasks written in Fortran or Cobol. A principal feature of the System 1100/80 is its ability to multiprocess a single Cobol instruction stream. This capability, though not incorporated by UNIVAC for processing Fortran programs, was essential for our particular experiment.

Implementation

The PIC code was written entirely in Fortran, and implemented with a single copy of the instruction stream. The management of data addressing and the mechanics of parallel-processing synchronization were devised and implemented by Dave Hammer of Sandia National Laboratories, Albuquerque, NM.*

---

*By devising an address mapping and a synchronization scheme for multithread activities, Hammer essentially converted the System 1100/80 into a Fortran parallel-processing machine for our use.

Figure 4

Figure 4 represents a simplified diagram of a UNIVAC 1100/84 (four-processor) system, on which our PIC timing runs were made. Although not indicated in the diagram, the processing of each activity is not necessarily handled by only one physical processor. In fact during the complete computational cycle of such an activity, all processors may timeshare the execution of the activity. A distinction, therefore, is made between activities and processors.

All relevant particle-in-cell data were put into various common blocks and partitioned for use by specific activities. Due to software addressing limitations, the PIC code was restricted to a maximum of 262k (decimal) words of total memory. For each particle, five data quantities (two for position and three for velocity) were required. Three mesh quantities, constituting a 34 X 34 mesh size, were required and duplicated for a maximum of eight (particle-push) activities. A total of 37k particles were initiated for processing, requiring 213k words of memory (particle plus mesh data). A further 47k of memory was used for the instruction stream, address mapping, and activity synchronization scheme.

PIC PARALLEL PROCESSING RESULTS

A multithread version of the PIC code was executed on a UNIVAC System 1100/80* with one, two, three, and four identical processors. Overall run times were measured and the results are given in Table I and Figure 5. The speedup values are the ratios of the overall execution time of a single-thread version of PIC (running on one processor) to the overall execution time of a multithread PIC code running on two, three, and four processors. We found that a maximum speedup of three was attained when using four processors with four activities spawned for each task.

able I
igure 5

---

*Provided for our use through the courtesy of SPERRY UNIVAC, Roseville, Minnesota.

Because the multithread PIC was not totally parallel (see Figure 3), the speedup for four processors may not indicate the full potential of the PIC algorithm. The times recorded and used for the parallel-processing speedup calculations were based on wall clock times, with timing runs made in a dedicated mode. Due to resource and time limitations, actual CPU times were not measured; therefore, no estimates could be determined for losses in effective processing time during the synchronization stage of each multithread activity.

CONCLUSIONS

Our results strongly suggest the possibility of significant computational speedups for a multiple-processor architecture similar to the UNIVAC System 1100/80. The coupling between algorithm and processing architecture illustrates not only the seemingly high degree of compatibility between our particular code and the computing environment, but also the need to distinguish those algorithms for which specific multiple-processor architectures are most effective.

The straightforward use of Fortran in coding the multi-thread PIC algorithm greatly simplified the overall task of implementing our parallel-processing experiment. Programming in Fortran is certainly a characteristic of Laboratory codes, and would be a desirable feature to retain when converting such codes from serial- to parallel-processing systems.

Encouraged by our results, we currently are studying the possibility of a totally parallel version of the PIC algorithm. We also plan to investigate parallel processing on multiple-processor architectures possessing as many as 16 processors.

REFERENCES

1. Buzbee, B.L., W. J. Worlton, G. Michael, and G. Rodrigue. "DOE Research in Utilization of High-Performance Computers." Los Alamos Scientific Laboratory report LA-8609-MS, December 1980.

2. Bucher, I. Y., P. O. Frederickson, and J. W. Moore. "Experience with a Multiprocessor Based on Eight FPS 120B Array Processors." Los Alamos Scientific Laboratory report LA-UR-81-1082 (unpublished), 1981.

3. Morse, R. L., and C. W. Nielson. "One-, Two-, and Three-Dimensional Numerical Simulation of Two Beam Plasmas." Physical Review Letters 23 (10 November 1969), 19, pp. 1087-1090.

4 Morse, R. L., and C. W. Nielson. "Numerical Simulation of Warm Two Beam Plasma." The Physics of Fluids 12 (November 1969), 11, pp. 2418-2425.

FIGURE CAPTIONS

Figure 1. A distribution of particles (dots with attached arrows--denoting position and velocity, respectively) contained within the four nearest-neighbor, cell-centers (+) from which the charges C_i, C_{i+1}, C_j, and C_{j+1} are in part calculated.

Figure 2. Distribution of particles pushed under the influence of the four nearest-neighbor, cell-centered (+) electric fields E_i, E_{i+1}, E_j, and E_{j+1} (determined by solving Poisson's equation ($-\nabla^2\phi = C$) for ϕ) and the uniform background electric/magnetic fields.

Figure 3. A multithread version of PIC as implemented on a UNIVAC System 1100/80 with two parallel-processing tasks (1 and 4), where A_n = total number of parallel activities (multithread), n = total number of particles, n_i = number of particles for activity i, C = total charge (distribution), and C_i = charge computed for activity i.

Figure 4. A simplified diagram of the UNIVAC System 1100/80 with four processors (P), designated 1100/84.

Figure 5. Plot of number of processors versus speedup corresponding to Table I.

Table I. Run times and speedups as a function of number of processors and number of activities for each parallel task spawned.

Number of Activities Per Parallel Task	Number of Processors	Average run Time (millisecond)	Speedup
1	1	102631	1
2	2	57110	1.80
3	3	42214	2.43
4	4	33263	3.09

TABLE 1

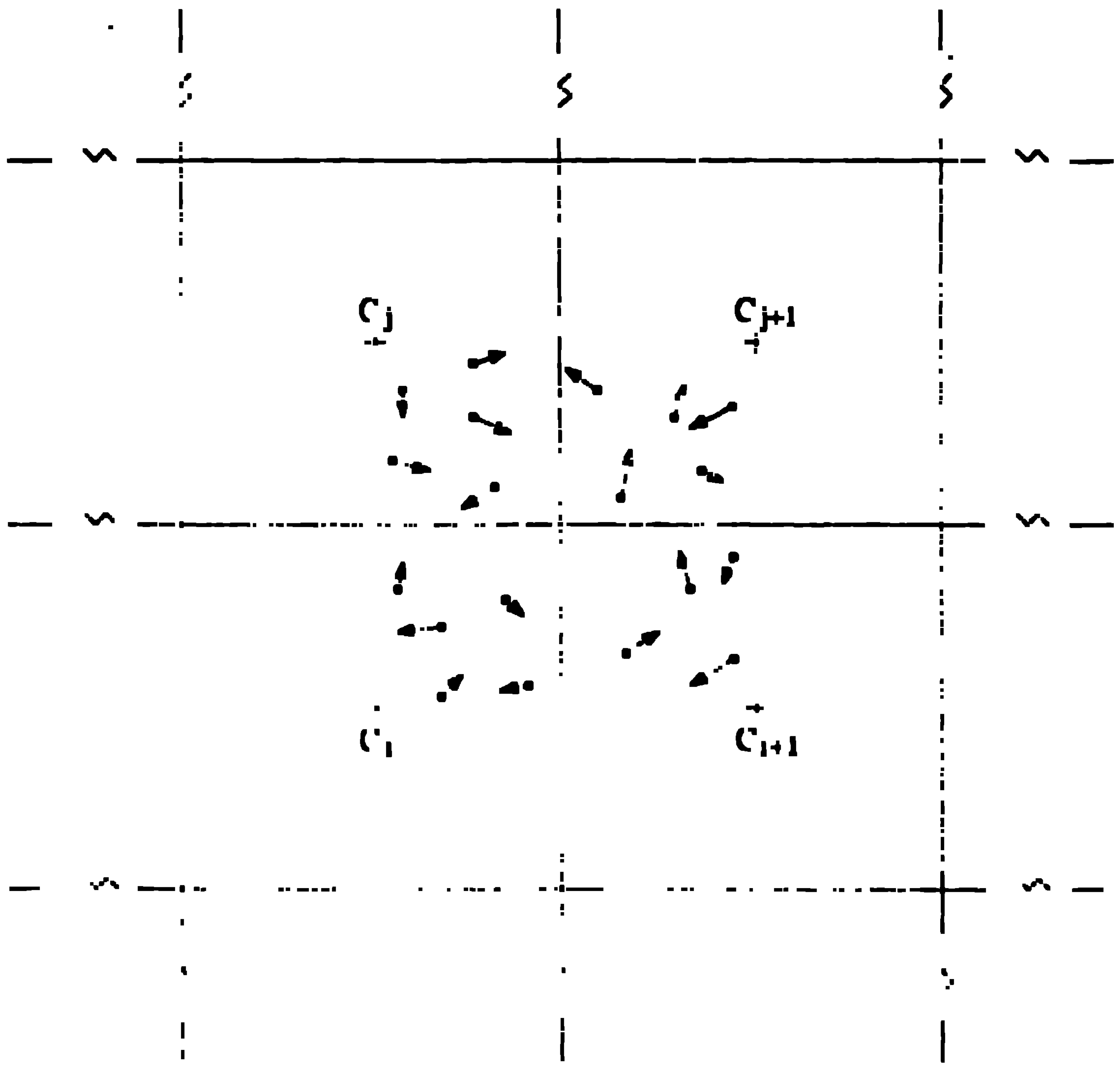

FIG. 1

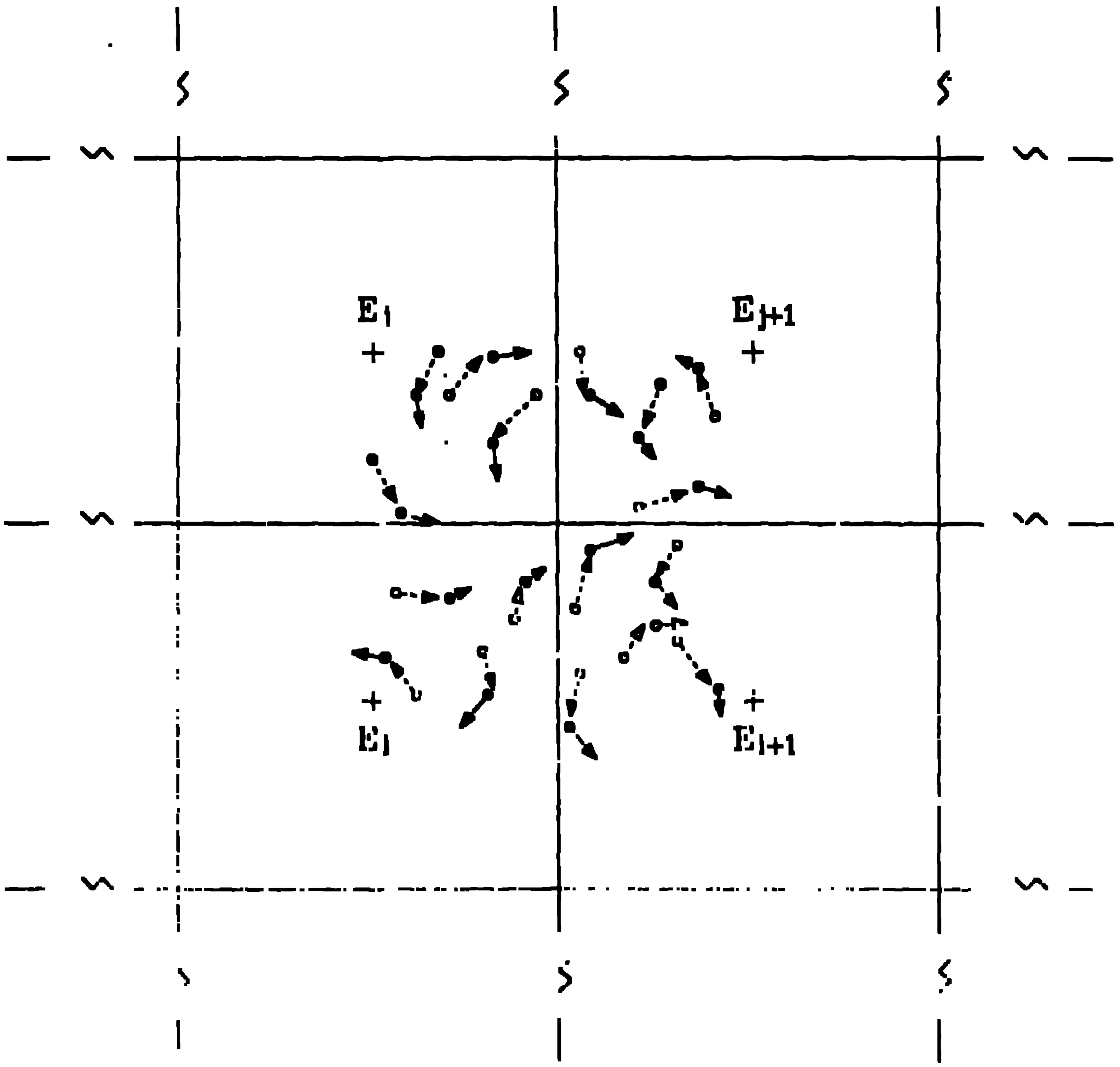

FIG. 7

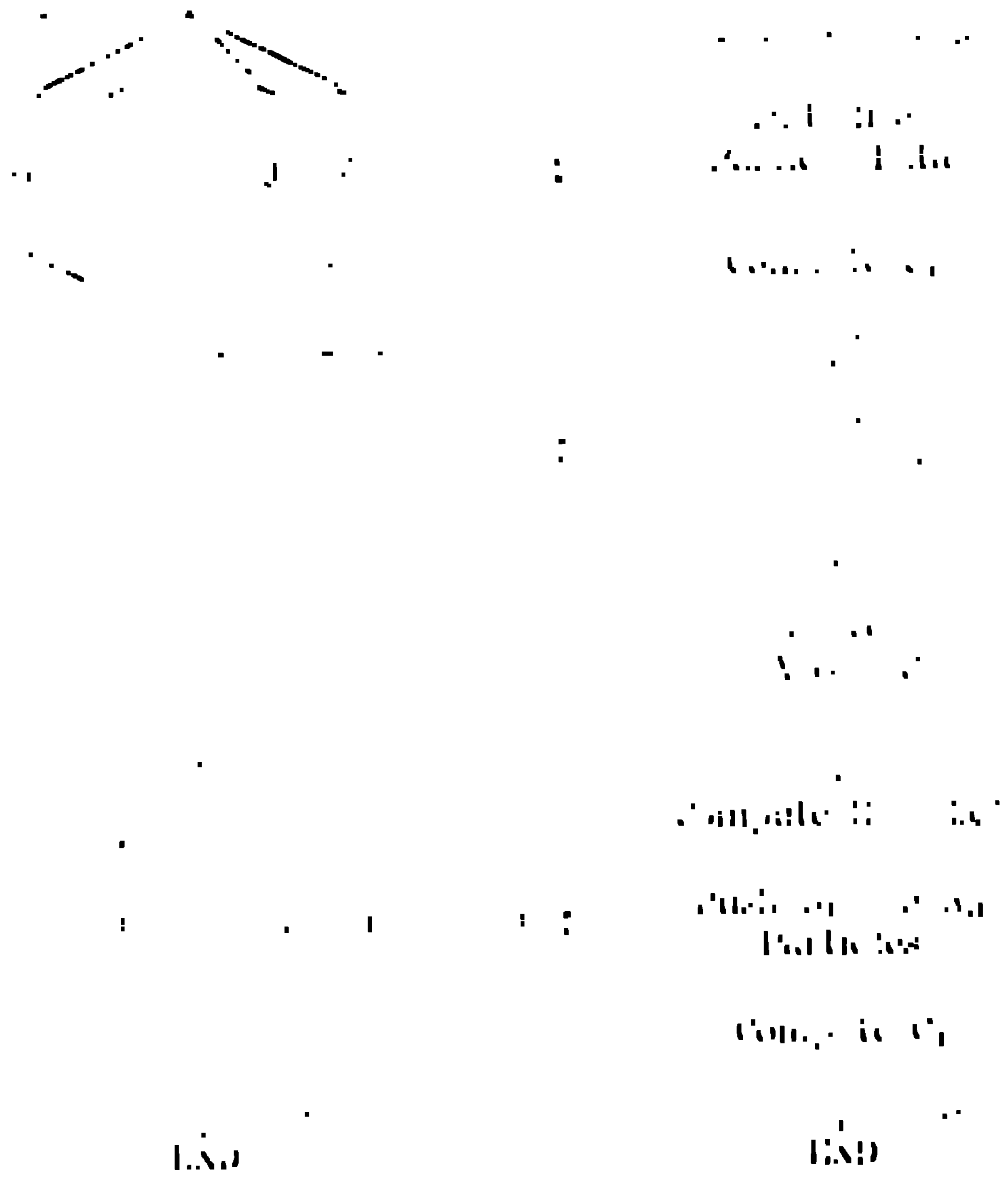

FIG. [illegible]

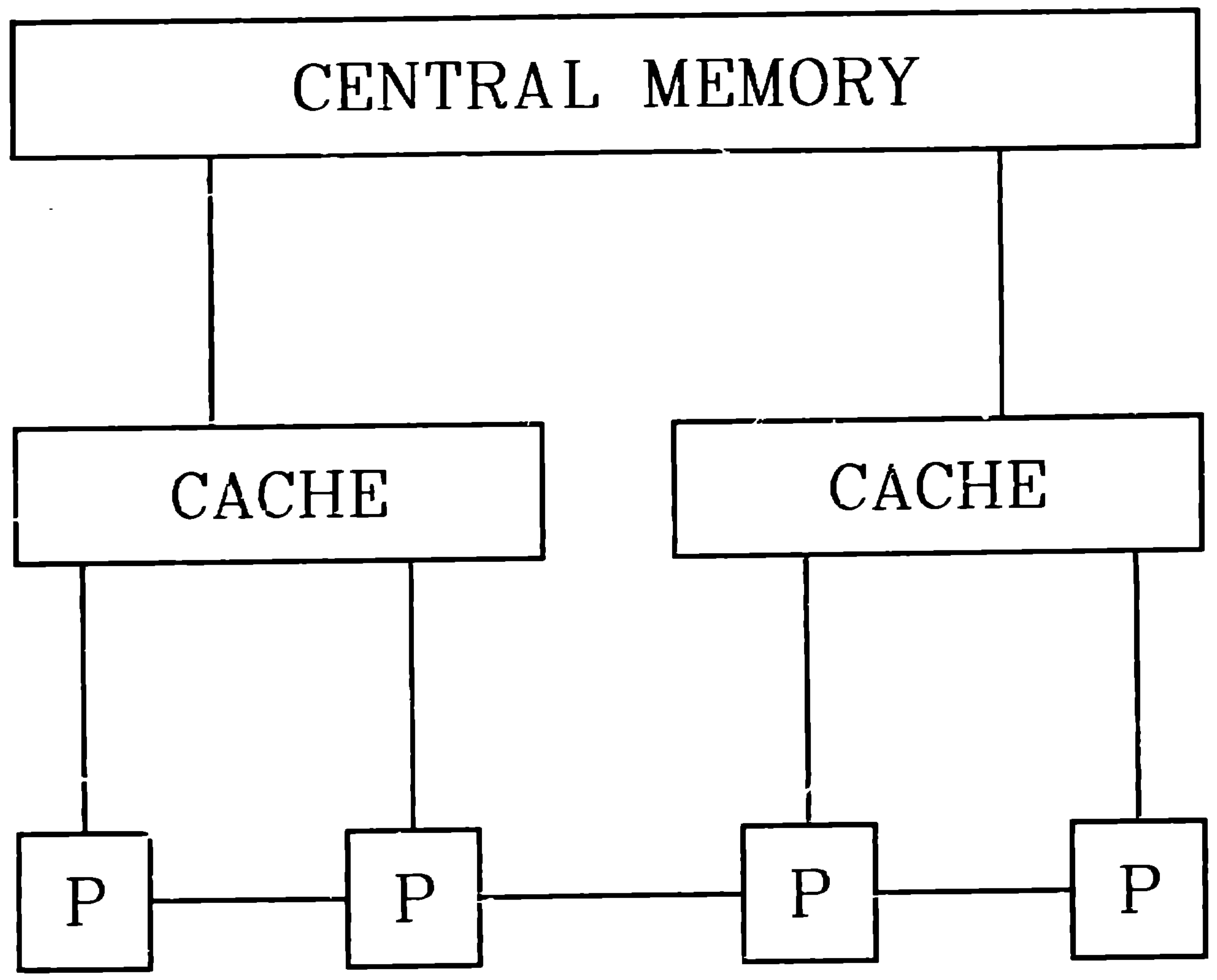

FIG. 4

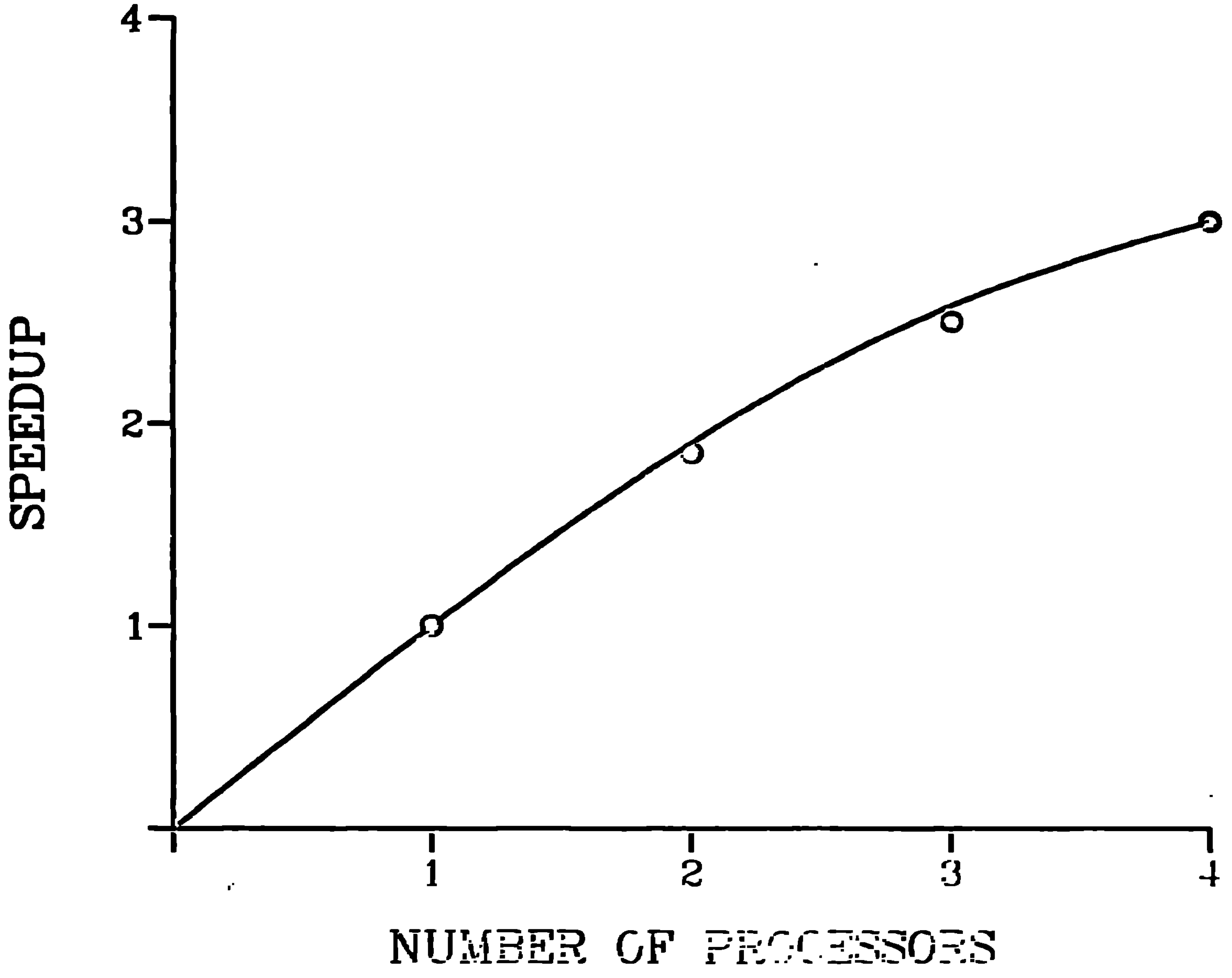

FIG. 5