LA-UR -83-413

Conf-8209107--1

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

LA-UR--83-413

DE83 007558

TITLE TWO PARALLEL FORMULATIONS OF PARTICLE-IN-CELL MODELS

AUTHOR(S) B. L. Buzbee

SUBMITTED TO Purdue Workshop on Algorithmically-Specialized Computer Organizations (WACO), Purdue University, West Lafayette, Indiana
September 27 - October 1, 1982

To be published by Academic Press

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

TWO PARALLEL FORMULATIONS OF PARTICLE-IN-CELL MODELS

B. L. Buzbee[1]

Computing Division Los Alamos National Laboratory Los Alamos, New Mexico

Particle-in-cell (PIC) models are widely used in fusion studies associated with energy research. They are also used in certain fluid dynamical studies. Parallel computation is relevant to them because

1. PIC models are not amenable to a lot of vectorization--about 50% of the total computation can be vectorized in the average model;
2. the volume of data processed by PIC models typically necessitates use of secondary storage with an attendant requirement for high-speed I/O; and
3. PIC models exist today whose implementation requires a computer 10 to 100 times faster than the Cray-1.

This paper discusses parallel formulation of PIC models for master/slave architectures and ring architectures. Because interprocessor communication can be a decisive factor in the overall efficiency of a parallel system, we show how to divide these models into large granules that can be executed in parallel with relatively little need for communication. We also report measurements of speed-up obtained from experiments on the UNIVAC 1100/84 and the Denelcor HEP.

---

[1]This work was supported in part by the Applied Mathematical Sciences Program, Office of Basic Energy Sciences of the US Department of Energy and the Air Force Office of Scientific Research.

PARTICLE-IN-CELL MODELS

We discuss particle-in-cell models (PIC) in the context of studying the behavior of plasmas in the presence of force fields [7]. We assume a two-dimensional region that has been discretized with N cells per side for a total of N^2 cells in the region. The discretization is illustrated in Fig. 1. The approach is to randomly distribute particles over the two-dimensional region and then study their movement as a function of time and forces acting on them. Typically, the average number of particles per cell will be $O(N)$ and particle information includes position, velocity, charge, etc. Thus, the total particle information will be $O(N^3)$. In its simplest form, the plasma simulation proceeds as follows.

1. "Integrate" over particles to obtain a charge distribution at cell-centers (a cell center is denoted by "X" in Fig. 1),
2. Solve a Poisson equation for the potential at cell-centers,
3. Interpolate the potential onto particles for a small interval of time Δt; i.e., apply force to the particles for a small time interval, recomputing their positions, velocities, etc.

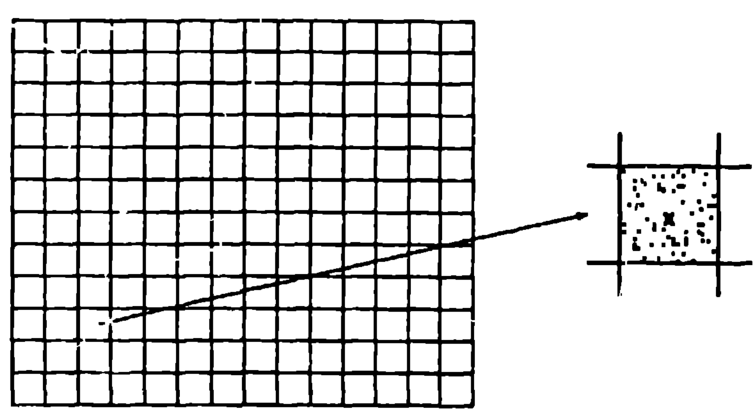

FIGURE 1. Relationship of region, mesh, and particles.

Step 2 requires $O(N^2)$ operations. Steps 1 and 3 require $O(N^3)$ operations and thus dominate the overall computational process. Generally, the particle information is stored in a large array and there is no correlation between particle position in that array and particle position in the rectangle. Thus, Step 1 is a many-to-one mapping of random elements from the list onto a cell center. Conversely, Step 3 is a one-to-many mapping of information at the cell center onto random elements of the particle list. These mappings from and to random elements in a list generally preclude efficient vector implementation. In general, only about 50% of the total operations in a PIC model are subject to efficient vector implementation. Of course, to achieve the highest level of performance from a vector processor, one needs to vectorize 90% or more of the total work in a computation [9]. Further, some PIC simulations used within the fusion energy research community require a computer that is about 100 times faster than the Cray-1 to successfully model phenomena of interest [4]. This need for higher performance combined with difficulties in implementing PIC efficiently on vector processors motivates our interest in asynchronous parallel (MIMD) formulations of them.

PIC ON A MASTER/SLAVE CONFIGURATION

Assume that we have an MIMD processor with a master/slave control schema as illustrated in Fig. 2. In practice a single processor may execute the function of both the master and one of the slaves, but for purposes of discussion we assume that they are distinct. The key to achieving efficient parallel implementation of PIC on a master/slave configuration is to divide the particles equally among the slaves and to keep all particle-related information within the slaves. Assuming that the master has the total charge distribution in its memory, the computational procedure is as follows.

Step 2B. Master solves potential equation and broadcasts potential ($O(N^2)$) to each slave.
Step 3. Each slave applies the potential for Δt (moves its particles).
Step 1A. Each slave integrates over its particles to obtain their contribution to total charge distribution at cell centers.
Step 1B. Each slave ships its charge distribution ($O(N^2)$) to the master.
Step 2A. Master sums charge distribution from slaves.

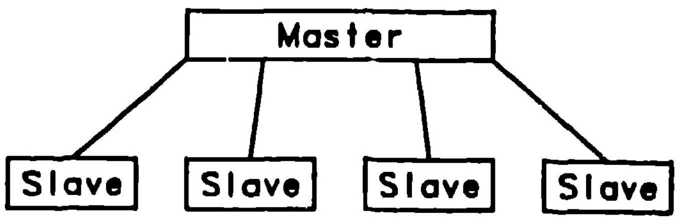

FIGURE 2. Master/slave communication geometry for four processors.

Note that in this approach the "particle pushing" ($O(N^3)$) portion of the computation is shared equally among the slaves. The amount of computation done by the master is $O(N^2)$ and the amount of interprocessor communication is $O(N^2)$. Further, the potential calculation is amenable to parallel implementation [2], but because the particle pushing dominates the overall calculation, we will not concern ourselves with parallel processing the potential calculation.

The key to efficient parallel implementation of PIC on a master/slave configuration lies in dividing particles equally among the slaves irrespective of particle position in the region. This was not our first approach in attempting to parallel process PIC. Rather, our initial approachs considered dividing the region into subregions and having a processor assigned to particles in each of the subregions. Such an approach produces a number of complications. For example, at the end of each time step some particles will migrate to its neighboring subregion. Thus, there must be an "exchange" of particles between processors at each time step. This exchange will necessitate garbage collection within the particle list of a given processor and, should the particles eventually concentrate in a small region, a single processor will do most of the computation while the others sit idle. To rectify such a situation, the region must be resubdivided, particles reallocated, etc. The computational cost of such processes is significant.

A similar phenomenon seems to occur in the parallel solution of elliptic equations. Again, the natural approach is to subdivide the region and to assign a processor to a subregion. It is extremely difficult to do this in a fashion that will yield a net gain in computational efficiency [5]. The point is that efficient implementation involves techniques that are somewhat counterintuitive.

PARALLEL PROCESSING PIC ON A RING CONFIGURATION

PIC can also be efficiently implemented on a MIMD machine with a ring control/communication organization. For purposes of discussion we assume a four-element ring with communication from left to right as indicated in Fig. 3. The key to success in this environment is again to divide particles equally among the processors but, in addition, have processors do a significant amount of redundant computation. Assuming that each processor has the total charge distribution at cell centers in its memory, the computational process is as follows.

Step 2. Each processor solves the potential equation.
Step 3. Each processor moves its particles.
Step 1A. Each processor integrates over its particles to obtain their contribution to the total charge distribution.

Step 1B. For 1 = 1, 2, 3, 4: pass partial charge distribution to neighbor; add the one received to "accumulating charge distribution."

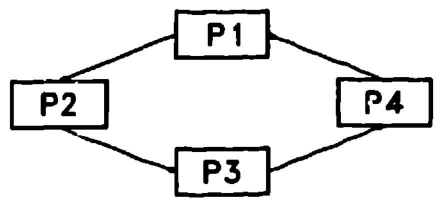

FIGURE 3. A four-element ring configuration.

ESTIMATING PERFORMANCE OF THE MASTER/SLAVE IMPLEMENTATION

The key issue in parallel processing is speedup as a function of the number of processors used. We define speedup as

$$S_p = \frac{\textit{execution time using one processor}}{\textit{execution time using } p \textit{ processors}}$$

To estimate performance of the master/slave formulation, we use a model of parallel computation introduced by Ware [8]. We normalize the execution time using one processor to unity.

Let

p = number of processors,

and

α = percent of parallel processable work.

Assume at any instant that either all p processors are operating or only one processor is operating; then

$$S_p = \frac{1}{(1-\alpha) + \frac{\alpha}{p}} .$$

Also

$$\frac{dS_p}{d\alpha}\Big|_{\alpha=1} = p^2 - p .$$

This model is unrealistic because the basic assumption will seldom, if ever, be realized in practice. However, with a little averaging, a lot of reality can be mapped onto this model. Note the behavior of the derivative of S_p in the neighborhood of $\alpha = 1$. This rapid and "last minute" growth as a function of α is displayed for a 4-processor, an 8-processor, and a 16-processor system in Fig. 4. Thus, successful realization of the potential performance of a parallel processor necessitates parallel formulation of at least 90% of the total computation. Therein lies the challenge in research in parallel processing. In 1970 Minsky [6] conjectured that average speedup in parallel processing would go like $\log p$. Indeed, if only 60% or 70% of the total computation is implemented in parallel, then he will be correct. However, for the master/slave implementation of PIC, recall that we are parallel processing the $O(N^3)$ component of the calculation and sequentially processing the $O(N^2)$ component. Thus, we have the possibility of achieving relatively high efficiency, at least on systems with a few processors.

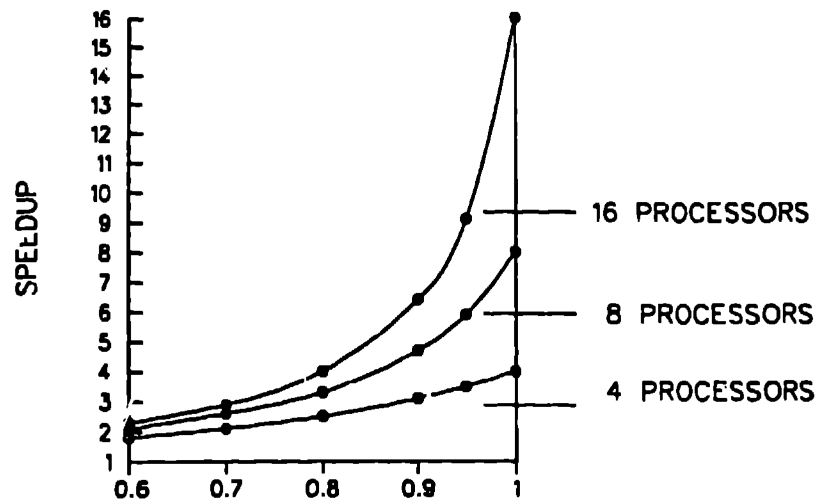

FIGURE 4. Ware's model of speedup for 4, 8, and 16 processors.

To estimate S_p for PIC in the master/slave environment, let

$$T = \textit{Total Operation Count}$$

$$= \underset{\substack{\uparrow \\ \text{Poisson} \\ \text{Solve}}}{C_1 N^2 \log N} + \underset{\substack{\uparrow \\ \text{Mesh} \\ \text{Transmission}}}{C_{2p} N^2} + \underset{\substack{\uparrow \\ \text{Particle} \\ \text{"Push"}}}{C_3 K N^2}$$

and

$$\alpha = \frac{particle\ push\ operations}{T}$$

$$= \frac{1}{1 + \dfrac{C_1 \log N + C_2 p}{C_3 K}}$$

$$\cong 1 \text{ if } C_1 \log N + C_2 p \ll C_3 K \ .$$

where K = average number of particles/cell.

If we further assume that each of processors has performance comparable to the Cray-1, then

$C_1 = 0.300\ \mu$ s/cell,

$C_2 = 0.075\ \mu$ s/cell, and

$C_3 = 5.500\ \mu$ s/particle.

Assume

N = K = 128;

then

p	α	Sp
4	.99	~ 3.8
8	.99	~ 7.5
16	.99	~13.9

COMPUTATIONAL EXPERIMENTS

Because of the p^2 behavior in the slope of S_p as α approaches 1, the only way to be sure of how well a parallel implementation will work is to implement it and measure speedup experimentally. In other words, small perturbations in seemingly insignificant areas of the computation may, in fact, lead to large perturbations in overall performance. Thus, to confirm our analysis, we have implemented variants of the master/slave configuration of PIC on two parallel processing devices—the UNIVAC 1100/84 and the Denelcor Heterogeneous Element Processor (HEP).

The UNIVAC 1100/84 is a commercially available system whose typical use is to process four independent job streams. With the help of UNIVAC personnel, and a bit of ingenuity, Los Alamos personnel have devised ways to control all four processors in this machine and use them to process a single PIC model [5]. Speedup measurements as a function of p are given in Table I. These results compare favorably with our estimates and reflect the fact that indeed we have successfully parallel processed a large percentage of the total computation.

TABLE I. SPEEDUP MEASUREMENTS FOR A MASTER/SLAVE IMPLEMENTATION

Equipment	p	Speedup
UNIVAC 1100/84	2	1.80
	3	2.43
	4	3.04
Denelcor HEP		6.0

Recently, a PIC model was implemented on HEP. HEP is designed to do task switching on each instruction. The architecture of a single processor is reminiscent of the CDC 6000 series, PPU system. There is an eight-slot barrel with a task assigned to each of the slots, and the processor examines the slots sequentially, executing a single instruction from eight concurrent processes. Most instructions in the machine require about eight cycles for execution. Thus, loosely speaking, a single processor is analogous to an eight processor parallel system. Los Alamos personnel have implemented a PIC model on HEP, first as a single-process and then as a multiple-process calculation. The ratio of the associated execution time is given in Table I. Again reflecting the fact that a large percentage of the total computation is being done in parallel.

CONCLUSION

High-performance computer systems involving several vector processors that can operate in parallel have already been announced [3]. Our analysis and experiments indicate that these systems can be used to parallel process particle-in-cell calculations whose current computational demands exceed the ability of a single processor. Realizing the highest levels of performance of a parallel system requires that a large percentage of the total computation be done in parallel. In the case of particle-in-cell models we were able to realize such performance by taking software modules written for a uniprocessor and combining them with appropriate communication and data replication. Thus, parallel implementation of "off the shelf" particle-in-cell models is likely to be easier than their implementation on a vector processor.

ACKNOWLEDGEMENTS

I am endebted to Ingrid Bucher, Paul Frederickson, Robert Hiromoto, and Jim Moore, all of the Los Alamos National Laboratory, for the experimental results discussed herein.

REFERENCES

[1] D. Boley "Vectorization of Some Block Relaxation Techniques, Some Numerical Experiments," *Proceedings of the 1978 LASL Workshop on Vector and Parallel Processors*. Los Alamos National Laboratory report LA-7491-C (1978).

[2] B. L. Buzbee, "A Fast Poisson Solver Amenable to Parallel Implementation." *IEEE Trans. on Computers*, Vol. C-22, No. 8 pp. 793-796 (August 1973).

[3] Datamation, "Seymour Leaves Cray," pp. 52-59 (January 1980).

[4] D. Forslund, "Large Scale Simulation Requirements for Inertial Fusion," presented at the conference on High Speed Computing, Gleneden Beach, Oregon, 1981.

[5] R. Hiromoto, "Results of Parallel Processing a Large Scientific Problem on a Commercially Available Multiple-Processor Computer System," Los Alamos National Laboratory report LA-UR-82-862 (1982).

[6] M. Minsky, "Form and Content in Computer Science," ACM Lecture, JACM 17, pp. 197-215, 1970.

[7] R. L. Morse, C. W. Nielson, "One-, Two-, and Three-Dimensional Numerical Simulation of Two Beam Plasmas," *Phys. Rev. Letters* 23, 1087 (1969).

[8] W. Ware, "The Ultimate Computer," *IEEE Spect*, pp. 89-91 (March, 1973).

[9] W. J. Worlton, "A Philosophy of Supercomputing," *Computerworld*, (October 1981).