ORNL/TM--12071

DE92 016802

Engineering Physics and Mathematics Division

Mathematical Sciences Section

THE HIERARCHICAL SPATIAL DECOMPOSITION OF THREE-DIMENSIONAL PARTICLE-IN-CELL PLASMA SIMULATIONS ON MIMD DISTRIBUTED MEMORY MULTIPROCESSORS

David W. Walker

Mathematical Sciences Section
Oak Ridge National Laboratory
P.O. Box 2008, Bldg. 6012
Oak Ridge, TN 37831-6367

Date Published: July 1992

Research was supported by the Applied Mathematical Sciences Research Program of the Office of Energy Research, U.S. Department of Energy.

Prepared by the
Oak Ridge National Laboratory
Oak Ridge, Tennessee 37831
managed by
Martin Marietta Energy Systems, Inc.
for the
U.S. DEPARTMENT OF ENERGY
under Contract No. DE-AC05-84OR21400

MASTER

Contents

1 Introduction 1
2 The PIC Algorithm 1
3 Decomposing PIC Problems 2
4 Communication of Guard Layer Data 4
5 Other Decomposition Methods 7
6 Conclusions 9
7 References 9

THE HIERARCHICAL SPATIAL DECOMPOSITION OF THREE-DIMENSIONAL PARTICLE-IN-CELL PLASMA SIMULATIONS ON MIMD DISTRIBUTED MEMORY MULTIPROCESSORS

David W. Walker

Abstract

The hierarchical spatial decomposition method is a promising approach to decomposing the particles and computational grid in parallel particle-in-cell application codes, since it is able to maintain approximate dynamic load balance while keeping communication costs low. In this paper we investigate issues in implementing a hierarchical spatial decomposition on a hypercube multiprocessor. Particular attention is focused on the communication needed to update guard ring data, and on the load balancing method. The hierarchical approach is compared with other dynamic load balancing schemes.

1. Introduction

For the last two decades the particle-in-cell (PIC) algorithm has been the method of choice for many types of plasma simulation computation [3, 10]. Current state-of-the-art, three dimensional, PIC simulations involve millions of particles, and several hundred thousand grid points, and thus impose heavy computational requirements. However, it is difficult to implement the PIC algorithm efficiently on most advanced architecture computers because it performs gather/scatter operations and recurrent data accesses [16, 17]. On MIMD distributed memory concurrent computers (or *multicomputers*) these difficulties manifest themselves as load imbalance and communication overhead. These two types of overhead can be traded off against each other, depending on the type of data decomposition scheme used. Thus, selecting the best decomposition scheme is of crucial importance in designing an efficient parallel PIC application code.

This report discusses issues concerned with implementing a PIC code that uses a unitary hierarchical spatial decomposition on a multicomputer. In this type of decomposition scheme the problem domain is subdivided over each coordinate direction in turn so that the total computation arising from the particle and field updates in each time step is balanced among the processors. In section 2 a brief description of the PIC algorithm is given. Section 3 outlines approaches to decomposing PIC problems, and describes the hierarchical method in more detail. Section 4 is concerned with the communication necessary to update a processor's "guard data." In section 5 other decomposition schemes are considered. Section 6 presents a summary and some conclusions.

2. The PIC Algorithm

The fundamental equations governing the evolution of a plasma system are the equation of motion for the particles, that relates the acceleration of a particle to the electromagnetic field at its position, and Maxwell's equations, that determine the evolution of the electromagnetic field. In the PIC method the field equations are solved on a regular tensor product grid. However, the particles are not constrained to lie on the grid and can be located at any position in the problem domain. The particle motion is driven by the electromagnetic field, so in advancing the particle positions it is, therefore, necessary to interpolate EM field values on the grid to each particle's position. Similarly, the evolution of the EM fields is determined by the current density generated by the motion of the particles, so when updating the EM fields by Maxwell's equations the current density at each grid point must be found by summing the contributions from all particles in neighboring grid cells. Thus, the coupled equations of particle motion and EM field evolution are solved by a time stepping algorithm, each step of which consists of four phases:

(A) The scatter phase, in which each particle scatters its contributions to the current density to the vertices of the cell in which it lies (its home cell) using a linear weighting scheme.

(B) The field solve phase, in which the current density on the grid evaluated in the scatter phase is used to integrate Maxwell's equations forward one time step.

(C) The gather phase, in which each particle gathers contributions to the electric and magnetic fields at its position by summing over the vertices of its home cell.

(D) The push phase, in which the equation of motion of each particle is advanced one time step using the EM field values found in the gather step.

3. Decomposing PIC Problems

In general, in a parallel PIC plasma simulation code there are two distributed data objects: the set of particles and the computational grid. In seeking an efficient implementation on MIMD distributed memory concurrent computers we wish to distribute (or decompose) these two data objects over the nodes of the concurrent computer so that communication and load imbalance overhead are simultaneously controlled to within reasonable limits, so that the total overhead is close to minimum. The push phase involves no interprocessor communication, while the field solve phase only requires communication of boundary values between processors. Thus, we are mainly concerned with controlling communication overhead in the scatter and gather phases of the parallel PIC algorithm.

If the distribution of particles remains fairly homogeneous throughout the simulation then load imbalance is not a problem, and communication overhead can be minimized by applying the same uniform spatial decomposition to the particles and the grid. This is referred to as an *Eulerian decomposition*. Clearly, if the particle distribution is inhomogeneous an Eulerian decomposition will suffer from significant load imbalance, as some processors will contain more particles than others. In this case, an alternative approach would be to decompose the grid spatially as in the Eulerian approach, but to apply a nonspatial decomposition to the particles by placing approximately the same number in each processor without regard to their location. This is referred to as a *Lagrangian decomposition*, and ensures good load balance. However, a high communication overhead is incurred in the scatter and gather phases of the PIC algorithm as particles do not, in general, lie in the same processor as the grid points with which they must interact.

Other more sophisticated decompositions attempt to keep communication costs low while simultaneously controlling load imbalance by dynamically changing the decomposition as the system evolves (see [5, 18] for an overview of some dynamic load balancing methods). In the adaptive Eulerian decomposition [11], a static uniform spatial decomposition is applied to the grid, and a non-uniform dynamic spatial decomposition is applied to the particles. The two decompositions are coupled so that some degree of data locality is maintained, and by dynamically changing the particle distribution reasonably good load balance can be achieved. In the scatter and gather phases it is necessary to transfer pieces of the grid between nearby processors.

In the unitary, hierarchical, spatial (UHS)decomposition [4] the same dynamic spatial decomposition is applied to both the grid and the particles. Again the dynamic decomposition ensures good load balance, and by applying the same spatial decomposition to the particles and the grid communication costs in the gather and scatter phases are kept low. Hierarchical spatial decomposition has previously been independently proposed as a load balance method by McCormick and Quinlan [13]. McCormick and Quinlan call their load balance method the "multilevel load balancer," and have used it in solving partial differential equations using the asynchronous fast adaptive composite (AFAC) grid method [8, 14]. Since the use of the UHS decomposition in PIC problems is the main concern of this report, we shall describe it in more detail.

A spatial domain is first divided into n_z subdomains by partitioning the domain orthogonal

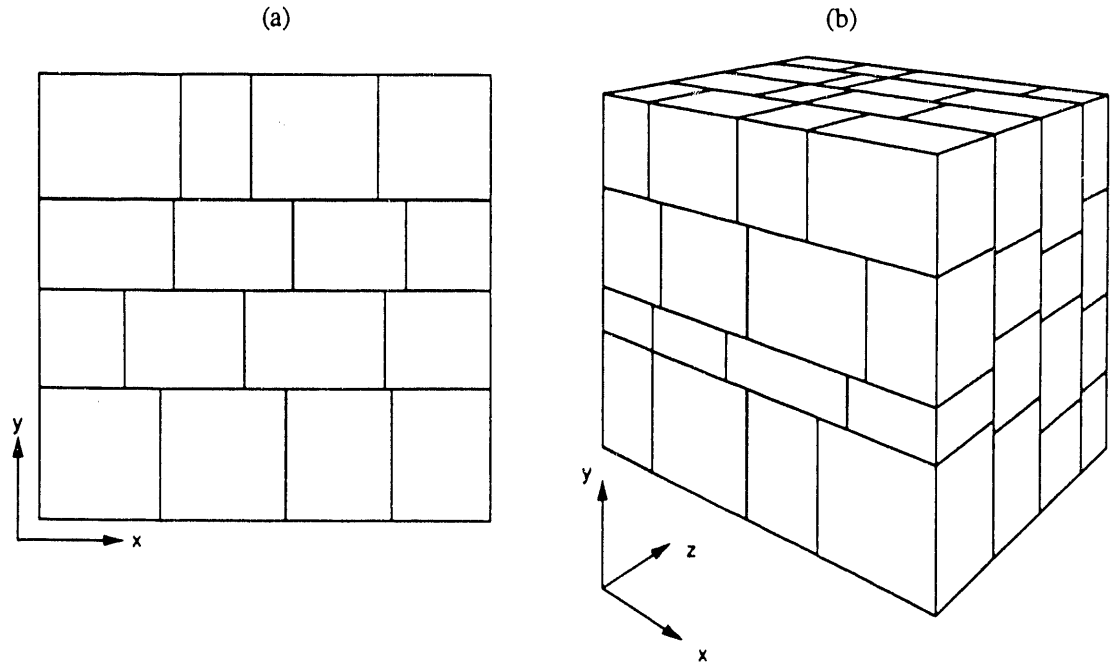

Figure 1: Hierarchical spatial decompositions of (a) a two dimensional domain decomposed onto a 4×4 processor mesh, and (b) a three dimensional domain decomposed onto a $4 \times 4 \times 4$ processor mesh.

to the z-axis. Each subdomain is then further partitioned orthogonal to the y-axis into n_y pieces. At this stage we have $n_y \cdot n_z$ subdomains, each of which is next partitioned orthogonal to the x-axis to produce $n_x \cdot n_y \cdot n_z$ subdomains. Examples of the UHS partitioning of two- and three-dimensional domains are given in Fig. 1. The subdomains are then assigned to processors, that can be regarded as forming a logically regular processor mesh. In this report we shall assume that one subdomain is assigned to each processor, although it may be desirable to overdecompose the domain so that each processor is assigned more than one subdomain. This overdecomposition permits communication and calculation to be overlapped. At each stage of the decomposition process the partitions are placed so that the workload in each of the resultant subdomains is approximately equal. It should be noted that no assumptions have been made about how the partitioning orthogonal to each coordinate direction is done. Fast heuristic methods based on a smooth approximation to the workload distribution across the coordinate direction of interest provide an attractive way of positioning the orthogonal partitions [4, 12].

In a strict unitary decomposition the workload in a subdomain consists of the work done processing both particles and grid points, and communication between processors is permitted only at the start of each time step. The basic unit in the spatial decomposition is a cell of the computational mesh, where a cell consists of the particles within the cell plus the root grid point of the cell (i.e., the first grid point in the cell as we sweep first over the x, then the y, and finally the z direction). The decomposition method described above divides the domain into $n_x n_y n_z$ rectangular blocks of cells that are then assigned to processors. In the gather phase we need the grid point data at each cell vertex in order to find the EM fields at the particles within a given cell. This can be done without additional communication if at the start of each

time step a "guard layer" of grid points is added to the three positive faces of each processor's block of cells. This requires communication with neighboring processors. To do the field solve phase without additional communication we need to be able to update each grid point that takes part in the gather phase in a processor. If we are using a method that requires data from grid points G grid spacings away, then the guard layer must be extended to $G+1$ grid points on the positive faces, and G grid points on the negative faces. For the simple finite difference scheme often used $G = 1$. Finally, to perform the scatter phase to find the current density at each grid point used in the field solve phase requires a guard layer of particle data $(G+1)$ cells thick on each face.

A strict unitary decomposition requires both grid point and particle data to be exchanged between processors at the start of each time step. This can result in significant overhead, particularly in the transfer of particle data. If we relax the restriction that there be only one communication phase in each time step, and allow grid point data to be communicated between the scatter and field solve phase then no particle data need be transferred to update the guard layer. This avoids some redundant computation in the scatter phase, and reduces the communication overhead. In this case we only require a guard layer of grid points that is $G+1$ and G grid points wide on the positive and negative faces of each subdomain, respectively. If we further permit grid point data to be communicated between the field solve and gather steps then the communication overhead is reduced even more since then the thickness of the guard layer need only be G grid points on each face. This of course introduces a synchronization point in the PIC algorithm between the field solve and push phases that destroys the unitary load balance of the grid and particle based update. However, this loss of unitary load balance may be worthwhile if the relative cost of the field solve phase is small, and so does not significantly contribute to the workload.

4. Communication of Guard Layer Data

In this section we discuss how each processor determines with which processors to communicate when updating guard layer information. We begin with some notation. We shall denote the set of consecutive integers running from I_1 to I_2, inclusive, by $\{I_1, I_2\}$. Regular grids are represented in terms of tensor products. For example, a $N_x \times N_y \times N_z$ grid, $\mathcal{G}$, is written as,

$$\mathcal{G} = \{1, N_x\} \otimes \{1, N_y\} \otimes \{1, N_z\} \tag{1}$$

This representation explicitly shows that the grid points are indexed starting at 1. The extent of each subdomain can be expressed in terms of the global index range in each coordinate direction. Thus, the subdomain at position $\mathbf{m} = (i, j, k)$ in the logical processor mesh can be described by,

$$\mathcal{D}(\mathbf{m}) = D_1(\mathbf{m}) \otimes D_2(\mathbf{m}) \otimes D_3(\mathbf{m}) \tag{2}$$

where,

$$D_t(\mathbf{m}) = \{d_t(\mathbf{m}), d_t(\mathbf{m}) + n_t(\mathbf{m}) - 1\} \tag{3}$$

and $t = 1$, 2 and 3 corresponds to the x, y, and z coordinates, respectively. Here, $\mathbf{d}(\mathbf{m}) = (d_1(\mathbf{m}), d_2(\mathbf{m}), d_3(\mathbf{m}))$ refers to the global index of the first grid point in the subdomain at position $\mathbf{m}$ of the processor mesh, and similarly, $\mathbf{n}(\mathbf{m})$ is a vector whose elements are the number of grid points in each direction in that subdomain. We can succinctly refer to the index set of a subdomain by means of a vector whose elements are the global index ranges in

each coordinate direction. Thus,

$$\mathbf{D}(\mathbf{m}) = \{\mathbf{d}(\mathbf{m}), \mathbf{d}(\mathbf{m}) + \mathbf{n}(\mathbf{m}) - \mathbf{1}\} \tag{4}$$

is just a vector whose elements $D_t(\mathbf{m})$ are given by Eq. (3).

We shall use the term *augmented subdomain* of a processor to refer to that processor's subdomain together with its associated guard layer. The augmented subdomain of the processor at position $\mathbf{m}$ in the processor mesh can be represented as,

$$\mathbf{A}(\mathbf{m}) = \{\mathbf{d}(\mathbf{m}) - \mathbf{G}_1, \mathbf{d}(\mathbf{m}) + \mathbf{n}(\mathbf{m}) - \mathbf{G}_2 - \mathbf{1}\} \tag{5}$$

where $\mathbf{G}_1$ and $\mathbf{G}_2$ are vectors giving the width of the guard layer on the negative and positive faces, which in general can be different on each face. If we use the expression $\mathbf{S}(\mathbf{m} \rightarrow \mathbf{m}')$ to refer to the data that the subdomain at position $\mathbf{m}$ must send to the subdomain at position $\mathbf{m}'$ to update the guard layer, then we may write,

$$\mathbf{S}(\mathbf{m} \rightarrow \mathbf{m}') = \mathbf{D}(\mathbf{m}) \cap \mathbf{A}(\mathbf{m}') \tag{6}$$

meaning that we must send the intersection of the subdomain at $\mathbf{m}$ and the augmented subdomain at $\mathbf{m}'$. Similarly, the data to be received by the subdomain at position $\mathbf{m}$ from the subdomain at position $\mathbf{m}'$ can be expressed as,

$$\mathbf{R}(\mathbf{m} \leftarrow \mathbf{m}') = \mathbf{D}(\mathbf{m}') \cap \mathbf{A}(\mathbf{m}) \tag{7}$$

The hierarchical spatial decomposition first divides the problem domain into slabs. These slabs are then divided into rows, and the rows are divided into the subdomains and assigned to processors. It therefore follows that when updating the guard layer a processor will only have to communicate with processors in the same slab as itself, and in the slabs that lie above and below it. Within the slabs above and below, a processor may have to communicate with any processor. Within the same slab a processor may have to communicate with any processor in the adjacent two rows of processors, in addition to communicating with the adjacent two processors in the same row.

We have written a prototype parallel code that updates the guard layer for a hierarchical decomposition, and runs on the Intel iPSC/2 hypercube. The update for some processor, P, is done in the following steps:

1. Processor P gathers together information about the decomposition in its slab. This information consists of the positions of the boundaries between the processor subdomains in the x and y directions, i.e., $d_1(i, j, k)$ and $d_2(j, k)$, where here we have shown that for a hierarchical spatial decomposition the positions of the y boundaries depend only on the location in the processor mesh in the y and z directions.

2. Processor P exchanges information about its slab with the processors above and below it in the processor mesh. P now has the decomposition information for the three slabs with which it must communicate.

3. For each of the three slabs we check to see which augmented subdomains overlap with the subdomain of processor P. This is done by first checking y boundaries, and then x

boundaries. That is, for each slab, $k = K-1, K, K+1$, we find all j' such that,

$$D_2(j, K) \cap A_2(j', k) \quad \text{is not empty} \tag{8}$$

and then for each of these j' we find for each of the three slabs the i' such that,

$$D_1(i, j, K) \cap A_1(i', j', k) \quad \text{is not empty} \tag{9}$$

The set of indices $(i'(j', k), j'(k), k)$ for $k = K-1, K, K+1$ gives the set of subdomains to which processor P must send data, and the overlap between the augmented subdomains of target processors and the subdomain of processor P can easily be found to determine which data should be sent.

4. For each of the three slabs we check to see which subdomains overlap with the augmented subdomain of processor P. For each slab $k = K-1, K, K+1$ we find all j' such that,

$$D_2(j', k) \cap A_2(j, K) \quad \text{is not empty} \tag{10}$$

and then for each of these j' we find for each of the three slabs the i' such that,

$$D_1(i', j', k) \cap A_1(i, j, K) \quad \text{is not empty} \tag{11}$$

The set of indices $(i'(j', k), j'(k), k)$ for $k = K-1, K, K+1$ gives the set of subdomains from which processor P must receive data. Again, the overlap for the communicating processors can be found to determine where in its guard layer each processor must store the data received.

5. Processor P sends the appropriate data to its target processors, and then receives data from its source processors. The data received are copied into the guard layer.

In the prototype code the hierarchical decomposition is produced by perturbing the boundaries of a perfectly regular decomposition by a random amount. In a full code the data accumulated in steps 1 and 2 could be determined and stored while performing the hierarchical load balancing.

In the prototype code each processor sends its messages independently when updating guard layer data. An alternative approach would be the use a "dimensional router". In such a scheme each processor would first send all the messages destined for the slabs above and below in just two big messages. The messages are now in the same slab as their destination processor, and are next moved in the y direction until each is in the correct row of the processor mesh. Finally, the messages are moved along the rows to their final destinations. Thus, in the dimensional router messages are routed synchronously over each spatial dimension in turn until they reach their destination, and this may result in a lower communication cost than the independent routing strategy, particularly if the topology of the communication network contains a three-dimensional grid. In addition, dimensional routing avoids congestion in the communication network that could occur if independent routing generates too much message traffic. Otto and Felten [7] found that a type of dimensional router called the "Crystal Router" was more efficient than the independent routing method for sufficiently high message volumes on the nCUBE/3200. More recent multiprocessors have different computation and communication characteristics, so this type of work needs to be on machines such as the Intel iPSC/860 hypercube, to determine the best message routing strategy. A third approach would be to employ a hybrid strategy

in which a dimensional router is used for messages that must be sent to processors within a certain distance in the processor mesh, and independent routing is used for the more distant (and hopefully fewer) processors.

5. Other Decomposition Methods

It is important to note that at the data parallel level it is not possible to split the PIC problem as a whole into independent subtasks. The finest granularity that one might use corresponds to that of the grid cells, which are interdependent because of the interaction between grid points and particles in the gather/scatter phases, and because of the interaction between grid points in the field solve phase. This spatial interdependency between the grid and the particles is a fundamental characteristic of the PIC method, and precludes the use of load balancers that assume independence between the tasks to be balanced. In the push phase the particle positions are updated independently, and so such methods could be used just for this phase. In such a scenario a regular Eulerian decomposition would be used for the scatter, gather and field solve phases, and the push phase would be run under control of the load balancer. At the end of the push phase it would be necessary to "write back" the particle information to the original Eulerian decomposition. On currently available multiprocessors the communication overhead needed to do this would be too large.

Baden and Kohn [2] have found orthogonal recursive bisection (ORB) to be an efficient way of maintaining load balance. As they point out, the imposition of a CFL condition ensures that load imbalance grows slowly, with the result that only a relatively small number of particles need be transferred between processors when the load balancer is run sufficiently frequently. Baden and Kohn found in their experiments on a 32-node Intel iPSC/860 hypercube that updating the guard layer is a much larger source of overhead than load balancing. Baden [1] has developed a programming abstraction called GenMP that simplifies the task of developing applications, such as PIC simulations, that require the problem domain to be dynamically partitioned on MIMD concurrent computers.

Dragon and Gustafson [6] have proposed a load balancing method that is efficient both in terms of concurrent overhead and memory usage. In this approach subcubes exchange individual particles, based only on knowledge of the positions of the two particles in each subcube nearest the partition between the two subcubes. By successively performing particle exchanges at finer spatial scales the domain can be partitioned exactly.

Other partitioning strategies result in approximate load balance, and typically are based on an approximation to the workload distribution obtained by averaging over some spatial scale, such as over cells [9]. Approximate load balance can also be achieved by approximating the work load density at the boundaries between partitions [15], although the accuracy of such methods for unitary decompositions is reduced since the work load density is not a smooth function of position. Another approach that might be worthwhile investigating is similar to that taken by Dragon and Gustafson, but instead of exchanging individual particles, subcubes exchange slabs of cells one cell thick. This has the advantage of keeping processor boundaries aligned with the computational grid.

Hinz [9] has discussed the use of N4 nets for dynamically maintaining unitary load balance of spatial two-dimensional domains. An N4 net is a regular rectangular mesh of quadrilaterals such that any two neighboring quadrilaterals in any row or column have only one boundary in common, as is illustrated in Fig. 2(a). To maintain approximate load balance each processor

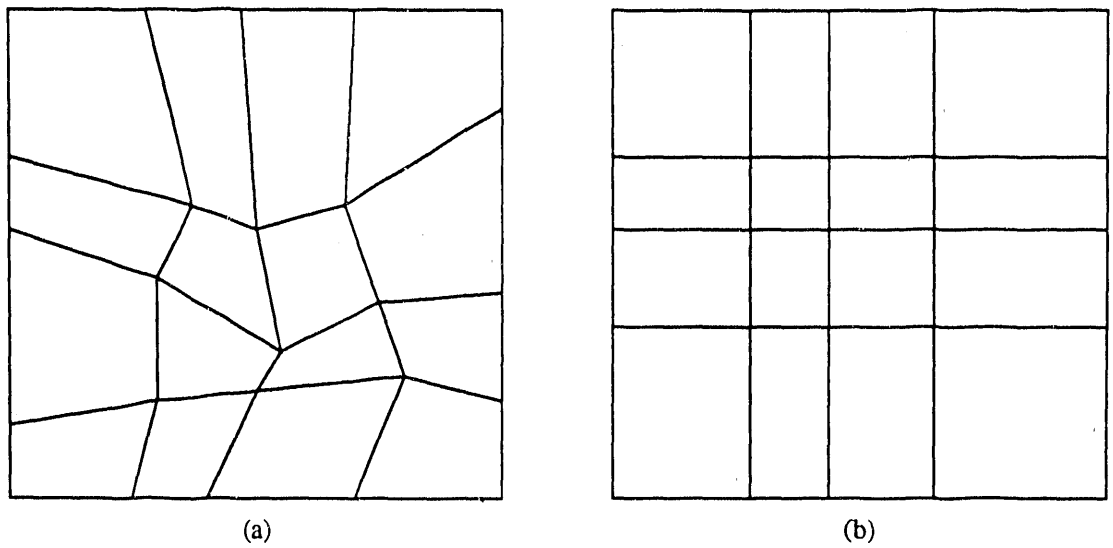

Figure 2: Example of the use of (a) an N4 net, and (b) rectilinear partitioning in decomposing a two dimensional domain onto a 4 x 4 processor mesh.

is responsible for moving the top lefthand corner of its subdomain. Processors lying along the bottom and righthand boundaries may move two points. The four corners of the domain are fixed. An attractive feature of the N4 net approach is that each processor needs to communicate only with its nearest neighbors in the processor mesh – the complications discussed in Section 4 that arise from non-neighboring communication do not occur. A disadvantage is that some additional bookkeeping is necessary to deal with the non-rectangular subgrids in each processor. Hinz's results with N4 partitioning of two-dimensional domains are encouraging, and it would be interesting to apply the N4 method to three-dimensional problems.

Another interesting way of dynamically maintaining unitary load balance of spatial domains has been proposed by Nicol [15]. In this approach we seek a rectilinear partitioning of the domain, such as that shown in Fig. 2(b). As for the N4 net method, this rectilinear approach has the advantage of requiring only simple communication patterns between processors when exchanging grid and/or particle guard layer data at the start of each time step. Nicol presents an algorithm for the one-dimensional Rectilinear Partitioning Problem (RPP), and uses this to find a conditionally optimal rectilinear partitioning of a two-dimensional domain. This is done by an iterative refinement procedure in which the optimal partitioning in one dimension is found with the partitioning in the other dimension held fixed. The resultant partitioning in the first dimension is then held fixed, and the partitioning in the second dimension is optimized. This procedure is iterated until no further change in the partitioning occurs. Nicol compares load balance and communication metrics for regular and irregular grids for partitionings produced by the rectilinear, binary recursive, and hierarchical spatial (called "jagged" by Nicol) decompositions. The general conclusions of this work were that rectilinear partitioning is most useful for grids that are not too irregular and when global communication is expensive. The rectilinear partitioning strategy may be appropriate for the Connection Machine.

6. Conclusions

In the implementation of parallel PIC simulations a common, unitary, spatial decomposition of the particles and grid is necessary to avoid the high cost of communication in the gather/scatter steps of the PIC algorithm. The hierarchical spatial approach promises to provide a partitioning strategy that dynamically maintains such a decomposition at low cost. Communication is necessary before and after each scatter step to update grid point values in each processor's guard layer. In general, this will involve communication between processors that are not nearest neighbors in the processor mesh. We have implemented this type of communication using independent routing, although other strategies involving dimensional routing may be more appropriate on different multiprocessors.

N4 nets and rectilinear partitionings provide other means of dynamically maintaining a unitary spatial decomposition, and require only nearest neighbor communication within the processor mesh to update guard layers. Further research is required to investigate the applicability of these methods to three-dimensional problems. A fast heuristic approach based on a smooth approximation to the workload distribution may provide a quicker way of performing rectilinear partitioning in three dimensions.

7. References

[1] S. B. Baden. Programming abstractions for dynamically partitioning and coordinating localized scientific calculations running on multiprocessors. *SIAM J. Sci. Stat. Comput.*, 12:145–157, 1991.

[2] S. B. Baden and S. R. Kohn. A comparison of load balancing strategies for particle methods running on mimd multiprocessors. Technical Report CS91-199, Department of Computer Science and Engineering, University of California, San Diego, May 1991.

[3] C. K. Birdsall and A. B. Langdon. *Plasma Physics Via Computer Simulation*. McGraw-Hill, New York, 1985.

[4] P. M. Campbell, E. A. Carmona, and D. W. Walker. Hierarchical domain decomposition with unitary load balancing for electromagnetic particle-in-cell codes. In D. W. Walker and Q. F. Stout, editors, *Proceedings of the Fifth Distributed Memory Computing Conference*, pages 943–950. IEEE Computer Society Press, 1990.

[5] George Cybenko. Dynamic load balancing for distributed memory multiprocessors. *J. Parallel and Distributed Comput.*, 7:279–391, 1989.

[6] K. M. Dragon and J. L. Gustafson. A low-cost hypercube load-balance algorithm. In J. L. Gustafson, editor, *Proceedings of the Fourth Conference on Hypercubes, Concurrent Computers, and Applications*, pages 583–589, 1989.

[7] E. W. Felten and S. W. Otto. A safe vertex. In G. C. Fox, editor, *Proceedings of the Third Conference on Hypercube Concurrent Computers and Applications*, pages 560–562. ACM Press, 1988.

[8] L. Hart and S. McCormick. Asynchronous multilevel adaptive methods for solving partial differential equations on multiprocessors: Basic ideas. *Parallel Computing*, 12:131–144, 1989.

[9] D. Y. Hinz. A run-time load balancing strategy for highly parallel systems. In D. W. Walker and Q. F. Stout, editors, *Proceedings of the Fifth Distributed Memory Computing Conference*, pages 951–961. IEEE Computer Society Press, 1990.

[10] R. W. Hockney and J. W. Eastwood. *Computer Simulation Using Particles*. Adam Hilger, Bristol, England, 1988.

[11] P. C. Liewer and V. K. Decyk. A general concurrent algorithm for plasma particle-in-cell simulation codes. *J. Comput. Phys.*, 85:302, 1989.

[12] P. C. Liewer, E. W. Leaver, V. K. Decyk, and J. M. Dawson. Dynamic load balancing in a concurrent plasma PIC code on the JPL/Caltech Mark III hypercube. In D. W. Walker and Q. F. Stout, editors, *Proceedings of the Fifth Distributed Memory Computing Conference*, pages 939–942. IEEE Computer Society Press, 1990.

[13] S. McCormick and D. Quinlan. Multilevel load balancing for multiprocessors – an outline. In *Preliminary Proceedings of the Third Copper Mountain Conference on Multigrid Methods, Copper Mountian, Colorado*. 1987.

[14] S. McCormick and D. Quinlan. Asynchronous multilevel adaptive methods for solving partial differential equations on multiprocessors: Performance results. *Parallel Computing*, 12:145–156, 1989.

[15] D. M. Nicol. Rectilinear partitioning of irregular data parallel computations. Technical Report 91-55, ICASE, July 1991.

[16] D. W. Walker. Characterizing the parallel performance of a large-scale particle-in-cell plasma simulation code. *Concurrency: Practice and Experience*, 2:257–288, 1990.

[17] D. W. Walker. Particle-in-cell plasma simulation codes on the connection machine. *Int. J. of Computing Systems in Engineering*, 2(2/3):307–319, October 1991.

[18] M. Willebeek-LeMair and A. P. Reeves. Dynamic load balancing strategies for highly parallel multicomputer systems. Technical Report EE-CEG-89-14, School of Electrical Engineering , Cornell University, December 1989.

INTERNAL DISTRIBUTION

1. B. R. Appleton
2. C. Bottcher
3. B. A. Carreras
4-5. T. S. Darland
6. E. D'Azevedo
7. J. J. Dongarra
8. J. B. Drake
9. T. H. Dunigan
10. R. E. Flanery
11. J. N. Leboeuf
12. V. E. Lynch
13. C. E. Oliver
14-18. S. A. Raby
19-23. R. F. Sincovec
24. G. M. Stocks
25. M. R. Strayer
26-30. D. W. Walker
31-35. R. C. Ward
36. P. H. Worley
37. Central Research Library
38. ORNL Patent Office
39. K-25 Applied Technology Library
40. Y-12 Technical Library
41. Laboratory Records - RC
42-43. Laboratory Records Department

EXTERNAL DISTRIBUTION

44. Scott Baden, Department of Computer Sci. and Eng., University of California at San Diego, CSE 0114, 9500 Gilman Drive, La Jolla, CA 92093-0114

45. Colin Bennett, Department of Mathematics, University of South Carolina, Columbia, SC 29208

46. Dominique Bennett, CERFACS, 42 Avenue Gustave Coriolis, 31057 Toulouse Cedex, FRANCE

47. Roger W. Brockett, Wang Professor of EE and CS, Division of Applied Sciences, Harvard University, Cambridge, MA 02138

48. Captain Edward A. Carmona, Parallel Computing Research Group, Phillips Laboratory, Kirtland AFB, Albuquerque, NM 87117

49. John Cavallini, Acting Director, Scientific Computing Staff, Applied Mathematical Sciences, Office of Energy Research, U.S. Department of Energy, Washington, DC 20585

50. Alexandre Chorin, Mathematics Department, Lawrence Berkeley Laboratory, Berkeley, CA 94720

51. James Corones, Ames Laboratory, Iowa State University, Ames, IA 50011

52. John J. Dorning, Department of Nuclear Engineering Physics, Thornton Hall, McCormick Road, University of Virginia, Charlottesville, VA 22901

53. Richard E. Ewing, Department of Mathematics, University of Wyoming, Laramie, WY 82071

54. Geoffrey C. Fox, NPAC, 111 College Place, Syracuse University, Syracuse, NY 13244-4100

55. James Glimm, Department of Mathematics, State University of New York, Stony Brook, NY 11794

56. Hans Kaper, Mathematics and Computer Science Division, Argonne National Laboratory, 9700 South Cass Avenue, Argonne, IL 60439

57. Tom Kitchens, ER-7, Applied Mathematical Sciences, Scientific Computing Staff, Office of Energy Research, Office G-437 Germantown, Washington, DC 20585

58. Peter D. Lax, Courant Institute of Mathematical Sciences, New York University, 251 Mercer Street, New York, NY 10012

59. James E. Leiss, Rt. 2, Box 142C, Broadway, VA 22815

60. George McNulty, Department of Mathematics, University of South Carolina, Columbia, SC 29208

61. Neville Moray, Department of Mechanical and Industrial Engineering, University of Illinois, 1206 West Green Street, Urbana, IL 61801

62. Captain Mike Proicou, Parallel Computing Research Group, Phillips Laboratory, Kirtland AFB, Albuquerque, NM 87117

63. Mary F. Wheeler, Rice University, Department of Mathematical Sciences, P. O. Box 1892, Houston, TX 77251

64. Office of Assistant Manager for Energy Research and Development, U.S. Department of Energy, Oak Ridge Operations Office, P. O. Box 2001, Oak Ridge, TN 37831-8600

65-74. Office of Scientific & Technical Information, P. O. Box 62, Oak Ridge, TN 37831

DATE
FILMED
8/14/92