In Proceedings of the High Performance Computing 1994 Conference,
La Jolla, CA, April 11–15, 1994.
Published by the Society for Computer Simulation, San Diego, CA.

# PARADIGMS AND STRATEGIES FOR SCIENTIFIC COMPUTING ON DISTRIBUTED MEMORY CONCURRENT COMPUTERS

Ian T. Foster [‡]
David W. Walker [†]

[‡] Mathematics and Computer Science
Argonne National Laboratory
Argonne, IL 60439-4844

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

Corresponding author:
David W. Walker
Oak Ridge National Laboratory
P. O. Box 2008
Oak Ridge, TN 37831-6367
(615) 574-7401 (office)
(615) 574-0680 (fax)
walker@msr.epm.ornl.gov (email)

# PARADIGMS AND STRATEGIES FOR SCIENTIFIC COMPUTING ON DISTRIBUTED MEMORY CONCURRENT COMPUTERS [1]

Ian T. Foster
Mathematics and Computer Science
Argonne National Laboratory
Argonne, IL 60439-4844

David W. Walker
Mathematical Sciences Section
Oak Ridge National Laboratory
Oak Ridge, TN 37831-6367

ABSTRACT

In this work we examine recent advances in parallel languages and abstractions that have the potential for improving the programmability and maintainability of large-scale, parallel, scientific applications running on high performance architectures and networks. This paper focuses on Fortran M, a set of extensions to Fortran 77 that supports the modular design of message passing programs. We describe the Fortran M implementation of a particle-in-cell (PIC) plasma simulation application and discuss issues in the optimization of the code. The use of two other methodologies for parallelizing the PIC application are considered. The first is based on the shared object abstraction as embodied in the Orca language. The second approach is the Split-C language. In Fortran M, Orca, and Split-C the ability of the programmer to control the granularity of communication is important is designing an efficient implementation.

## 1. INTRODUCTION

Distributed memory concurrent computers would appear to be ideal for large-scale applications with large computational, memory, and/or storage requirements. However, many researchers are deterred from using these machines by the perception that they are hard to program, and this has hindered their more widespread use as general-purpose computational resources. To improve the programmability and maintainability of large-scale parallel applications requires languages, abstractions, and mechanisms that hide details of how an architecture exploits parallelism, in addition to the development of tools to aid in the parallelization of code, and reusable software. In recent years, many potentially useful parallel languages and abstractions have been proposed, but these have generally been ignored by mainstream computational scientists, who have preferred to use sequential languages, augmented by explicit data distribution and message passing routines. One of the reasons that some good ideas have been overlooked has been the poor performance of prototypes — in general, computational scientists are not prepared to sacrifice much performance in return for improving ease of programming.

This paper describes the early stages of an ongoing project to investigate how well different programming methodologies are able to exploit the power of high performance architectures and networks. The aim is to study and evaluate different approaches for a small number of applications. For each approach and application, factors such as performance, ease of programming, portability, modularity, and maintainability are assessed and where appropriate, quantified. The applications to be considered are characterized by different types of interprocessor communication, ranging from very regular to completely irregular.

Three different methodologies for parallelizing scientific applications will be considered in this paper, and their use in implementing a particle-in-cell (PIC) algorithm for simulating plasmas will be discussed. The first is Fortran M, a set of extensions to Fortran 77 that supports the modular design of message passing programs, and is the main focus of the paper. The other approaches use the Orca language, which is based on the shared object abstraction, and the Split-C language, a set of parallel extensions to C. In this paper we do not attempt to give a detailed description of Fortran M, Orca, or Split-C. Instead, the reader is referred to more complete documentation (Foster and Chandy 1992, Foster *et al.* 1993, Bal *et al.* 1992, Culler *et al.* 1993). The PIC algorithm has been chosen for study because it involves gather and scatter-with-add operations that result in interesting communication and load balance tradeoffs.

Section 2 gives an overview of some of the approaches

[1] This work was supported in part by the Applied Mathematical Sciences Research Program, Office of Energy Research, U.S. Department of Energy under contract DE-AC05-84OR21400 with Martin Marietta Energy Systems Inc., by the Office of Scientific Computing of the U.S. Department of Energy under contract W-31-109-Eng-38, and by the National Science Foundation's Center for Research in Parallel Computation under contract CCR-8809615.

to exploiting parallelism on high performance architectures and networks. In Section 3 a brief description of the PIC algorithm is given. The implementation of the PIC algorithm using Fortran M is discussed in Section 4. Two implementations, which differ in how the gather phase is performed, are described. In Section 5, our future plans for optimizing the Fortran M PIC code and for developing shared object and Split-C versions of the code are described. Section 6 presents some concluding remarks.

## 2. PROGRAMMING METHODOLOGIES

The most common approach to programming distributed memory machines is to use a sequential language and a library of message passing routines. Such libraries provide for point-to-point communication between processes, and usually some collective communication, such as broadcast and reduction routines. Examples include native message passing systems such as NX/2 on the Intel family of parallel supercomputers, commercially available products such as Express, and portability interfaces of varying degrees of sophistication such as PICL, p4, PARMACS, PVM and Zipcode. The recently developed MPI message passing interface in intended to become the standard for explicit message passing on distributed memory systems (MPI 1993, Walker 1994). Many of these message passing systems are described in a recent journal publication (Parallel Computing 1994). We shall refer to this approach as "sequential language plus explicit message passing" (SL+EMP).

A major reason for the popularity of the SL+EMP approach has been the tradeoff between performance and ease of programming. Although researchers would like distributed memory machines to be easy to program, for many at the forefront of computational science performance is the preeminent requirement. For a large class of applications SL+EMP results in high performance, scalable applications. However, as software technologies advance, we expect the balance of the tradeoff between performance and programmability to change. Indeed, by allowing preprocessors, compilers, and/or runtime systems to schedule resources and perform necessary data movement, we expect alternatives to the SL+EMP approach to become increasingly attractive and to rival or exceed the SL+EMP approach in performance. The work described here is directed at investigating and furthering this trend.

Other approaches to parallel programming have been reviewed by Bal (Bal 1990). Coordination languages represent an important alternative to the SL+EMP approach. These are specialized languages for specifying concurrency, communication, and synchronization, and examples include Occam, Strand, PCN, and Fortran M. Fortran M provides a set of extensions to Fortran 77 to support modular message passing programs. The input and output of different modules (or processes) are connected by typed channels. Message passing is performed over channels by means of nonblocking send and receive routines. A process encapsulates common data, subprocesses, and internal communication channels. The use of Fortran M is illustrated for the PIC algorithm in Section 4.

Another type of alternative approach involves software support for shared memory, such as the shared virtual memory (Li and Hudak 1989) and shared object (Bal *et al.* 1992) paradigms. Orca provides quite general support for shared objects, while MetaMP (Otto 1991, Otto 1994) provides for globally indexed arrays and certain types of weakly coherent shared data objects. The use of Orca in the PIC algorithm is outlined in Section 5.

Other important alternatives to the SL+EMP approach are not addressed in this paper, but will be studied in the future. ParAlfl, Sisal, and ID are implicit parallel languages employing functional parallelism. Sequential languages with data parallel extensions, such as Fortran-D, Vienna Fortran, and High Performance Fortran, make use of directives for specifying data distribution, and computation is performed using the "owner computes" rule.

## 3. PARTICLE-IN-CELL APPLICATION

The particle-in-cell (PIC) method is commonly used for simulating the evolution of plasmas (Birdsall and Langdon 1985, Hockney and Eastwood 1988). The charged particles making up a plasma move under the influence of the electromagnetic fields that they generate, together with any external fields that may be present. Thus, in simulating the evolution of plasmas we seek to model the mutual interaction of all particles and fields in a self-consistent manner. In PIC simulations the particles are represented by a set of "superparticles" each of which models the action of a large number of physical particles, although for clarity we will drop the "super-" prefix. The dynamics of the system is followed by integrating the equation of motion of each of the particles in a series of discrete time steps. The particle motion is driven by the Lorentz force,

$$\frac{\mathrm{d}\mathbf{u}}{\mathrm{d}t} = \frac{q}{mc}\left(\mathbf{E}_p + \frac{1}{\gamma}\mathbf{u}\times\mathbf{B}_p\right), \qquad (1)$$

where $q$ and $m$ are the particle charge and rest mass, respectively, and $c$ is the velocity of light. $m\mathbf{u}$ is the particle's relativistic momentum divided by $c$ (i.e., $\mathbf{u} = \gamma\mathbf{v}/c$), where $\mathbf{v}$ is the velocity and $\gamma^2 = 1 + u^2$. $\mathbf{E}_p$ and $\mathbf{B}_p$ are the electric and magnetic fields at the particle's position. These fields are found on the grid by solving Maxwell's equations at each time step on a rectangular grid,

$$\begin{aligned} \frac{1}{c}\frac{\partial\mathbf{B}}{\partial t} &= -\mathrm{Curl}\,\mathbf{E} \\ \frac{1}{c}\frac{\partial\mathbf{E}}{\partial t} &= \mathrm{Curl}\,\mathbf{B} - \mathbf{j}, \end{aligned} \qquad (2)$$

where $\mathbf{j}$ is the current density due to the particles divided by $c$. The particle dynamics and the evolution of the electromagnetic field are coupled through the current density generated by the motion of the particles, and the Lorentz force arising from the electromagnetic field.

Each particle lies in a cell of the grid, and this cell will be referred to as the particle's *home cell*. To solve Maxwell's equations the current density, $\mathbf{j}$, which depends on particle charge and velocity, must be known at each grid point and is evaluated by having each particle contribute to the current density at the grid points lying at the corners of its home cell. Maxwell's equations can then be solved to give the electromagnetic fields on the grid at the next time step. Next the equation of motion (Eq. 1) is advanced one time step. The electric and magnetic fields at each particles are found by interpolating the values at the vertices of the particle's home cell. Thus, $\mathbf{E}_p$ and $\mathbf{B}_p$ are a weighted sum of the values of the $\mathbf{E}$ and $\mathbf{B}$ at the corners of the home cell, and once these are determined the equation of motion for each particle can be advanced. Thus, the PIC algorithm proceeds in a series of time steps each made up of four phases:

- The *scatter* phase in which particles scatter contributions to the current density to the corners of their home cells. This is a scatter-with-add operation.
- The *field solve* phase in which Maxwell's equations are advanced one time step. In the explicit scheme used the new values at each grid point depend only on values at neighboring grid points.
- The *gather* phase in which the particles use the electromagnetic field values at the corners of their home cells to evaluate the fields at their positions.
- The *push* phase in which the equation of motion of each particle is advanced one time step. The update for each particle is independent of all others.

There are two basic approaches to parallelizing PIC applications (Walker 1990, Walker 1991). In the first, both the computational grid and the particles are spatially decomposed into processes, and only data lying along process boundaries needs to be moved between processes. Good load balance is maintained if particles are distributed sufficiently homogeneously, but for heterogeneous particle distributions dynamic load balancing may be necessary. In the second approach, a regular spatial decomposition is applied to the grid, but a nonspatial decomposition is applied to the particles. The particles are decomposed into approximately equally-sized groups with no regard for their spatial location. Thus, the particle and grid decompositions are completely decoupled. This approach ensures good load balance, but the scatter and gather phases now require nonlocal communication. In this work the second type of decomposition will be used.

## 4. FORTRAN M IMPLEMENTATION

We have developed two Fortran M implementations of the PIC algorithm that differ primarily in how the gather phase is performed. In both cases four fundamental processes are created. These are the particle, scatter, grid, and gather processes. Only the particle and grid processes actually perform computations; the scatter and gather processes are for communicating data between the particles and the computational grid. The particle process spawns a specified number of child processes, referred to as subgroup processes, each of which handles a given set of particles. Similarly, the grid process spawns subgrid processes, each of which handles a rectangular 3D block of grid points.

In the first implementation the gather phase is performed by having the grid points scatter electromagnetic field information to particles in the surrounding cells. This requires each subgrid process to maintain lists that enable it to determine to which particle each of its grid points must send electromagnetic field information. The main disadvantage of this approach is that it introduces memory imbalance, since the list data structure in each subgrid process must be large enough to hold the maximum number of particles that interact with any given subgrid in the course of the simulation. In the alternative implementation the gather phase is performed by having each particle request electromagnetic field information from the grid points with which it interacts. This involves a two-phase communication procedure in which a subgroup process sends out a request for data, and subsequently receives that data. This approach avoids the memory imbalance problem, but results in additional communication overhead.

### 4.1. Implementation 1

In Fortran M, processes may communicate by means of typed, uni-directional channels. A channel connects the *outport* of the source process to the *inport* of the destination process. In both implementations, the main program establishes channels for the communication of data in the scatter, gather, and field solve phases and creates the particle, scatter, grid, and gather processes. In the first implementation, one channel is created to connect each subgroup process to the scatter process, and one channel is created to connect the scatter process to each subgrid process. Similarly, a set of channels is created to connect each subgrid process to the gather process, and another set is created to connect the gather process to each subgroup process. Thus, if there are $N$ subgroup processes and $M$ subgrid processes, the high-level structure of the first implementation is as shown in Fig. 1.

In the first implementation, each subgroup process passes current density data through a channel to the scatter process. The scatter process then routes the data to the correct subgrid process where it is accumulated at a grid point. In addition to sending the current density contribution, the indices of the grid point are also sent. These indices are used by the scatter router to direct the contribution to the correct subgrid process, and are also used by the subgrid process to determine at which grid point to accumulate the contribution. The detailed structure of the processes involved in the scatter phase is shown in Fig. 2. Each subgroup process, P, is connected to a router process, R, internal to the scatter process. When a router

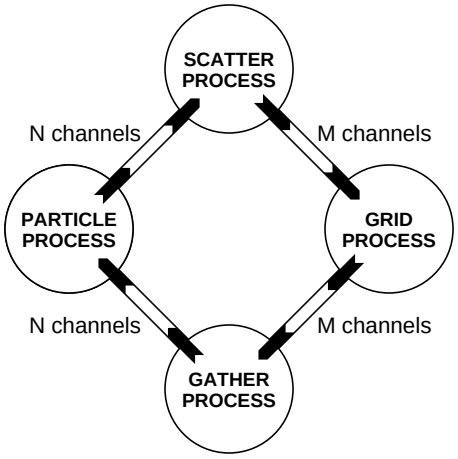


Figure 1: Schematic representation of processes and channels for Implementation 1. The black chevrons represent inports and outports that are connected to form channels between the processes. There are $N$ subgroup processes and $M$ subgrid processes.

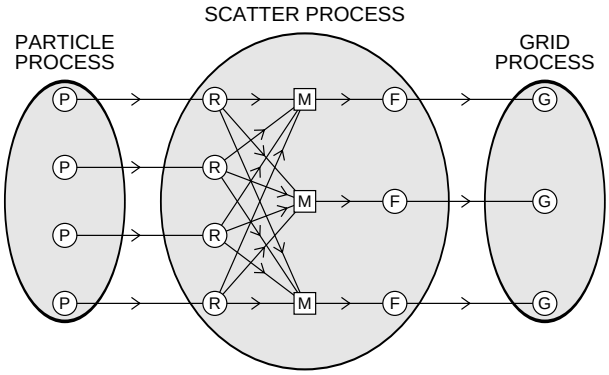


Figure 2: Schematic representation of processes and channels for Implementation 1 showing internal details of the scatter process. The circles labeled P, R, F, and G represent the subgroup, router, forward, and subgrid processes. The squares labeled M are mergers which the router processes use to route information. The arrowed lines represent channels. In this example, there are $N = 4$ subgroup processes, and $M = 3$ subgrid processes.

process receives data, it examines the grid point indices and uses this information to determine to which subgrid process to forward the current density contribution. The forwarding is done by means of a *merger*, labeled 'M' is Fig. 2. A merger is similar to a channel, but it connects several specified outports to a single inport. The scatter process contains one merger for each subgrid process, and each router process has an outport to each of these mergers. The inport of each merger passes data to a forward process, F, which passes the data on to the final destination subgrid process.

The current density contribution and the grid point indices are sufficient to perform the scatter phase of the PIC algorithm. However, in the first implementation, the global particle ID number must also be sent and stored in the subgrid process so that the subgrid processes can send back electromagnetic field information in the gather phase to the correct subgroup processes. After the subgrid processes have received and accumulated all the current density information, Maxwell's equations are solved. Then each subgrid process sends electromagnetic field data to the gather process which routes it to the correct subgroup process. The channels that connect a subgrid process to the gather process, and the gather process to the subgroup processes carry the three components of the electric and magnetic fields as well as the global particle ID which is used by the gather process to route the data. When the electromagnetic field information has been gathered by the subgroup processes, the equation of motion is solved and the particles are advanced by one time step.

Apart from the scatter and gather operations, the subgroup processes are not involved in any other communication as the particle push phase involves data entirely local to each subgroup process. However, the subgrid processes need to communicate with each other in the field solve phase to exchange information about grid points lying along their boundaries. In general, for a three-dimensional problem each subgrid process must have six channels to perform these exchanges — one to send, and one to receive, in each coordinate direction.

### 4.2 Implementation 2

The second implementation is the same as the first, except the gather phase is performed using a two-phase communication protocol. When a subgroup process needs electromagnetic field information it passes a request to the gather process, which routes the request to the subgrid process that has the information. This subgrid process then sends the information back to the subgroup process that requested it. The high-level structure of the second implementation is shown in Fig. 3.

A request for electromagnetic field information from a subgroup process is routed to the appropriate subgrid process using a set a $M$ mergers. This is similar to the scheme used in the scatter phase. However, the gather process makes use of Fortran M's dynamic channel capabilities to send the information back to the requesting subgroup process. Before a router process internal to the gather process sends on a request for information, it first sets up a channel to be used to return the information. The outport for this channel is then sent, along with the grid point coordinates, into the appropriate merger to a forward process. The forward process then sends on the grid point coordinates to the subgrid process containing the grid point. The subgrid process sends the electromagnetic field information for the

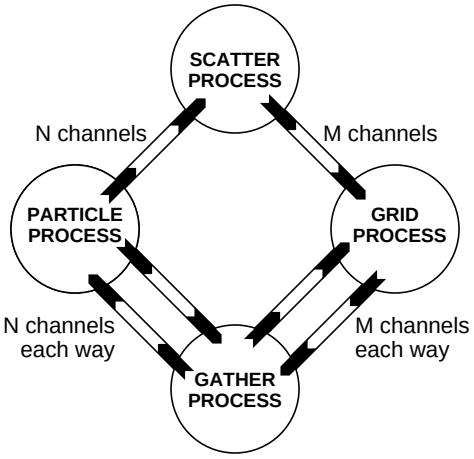


Figure 3: Schematic representation of processes and channels for Implementation 2 in which a request/response protocol is used to perform the gather phase. The black chevrons represent inports and outports which are connected to form channels between the processes. There are $N$ subgroup processes and $M$ subgrid processes.

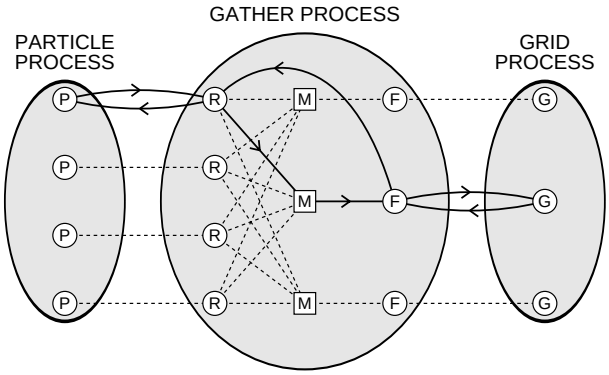


Figure 4: Schematic representation of processes and channels for Implementation 2 showing internal details of the gather process. The circles labeled P, R, F, and G represent the subgroup, router, forward, and subgrid processes. The squares labeled M are mergers which the router processes use to route information. Channels are shown as dashed lines. Solid arrowed lines are used to show an example of a particular request and response. Note how the forward process sends the response back directly to the router process. In this example, there are $N = 4$ subgroup processes, and $M = 3$ subgrid processes.

grid point back to the forward process. The forward process then uses the outport that it received in the request phase to return the electromagnetic field information back to the router process that routed the request. This avoids having to route the response through a set of mergers. The router process can then send the electromagnetic field information back to the subgroup process that requested it. An example of this two-phase request/response protocol is shown in Fig. 4.

## 5. DISCUSSION AND FUTURE WORK

We have implemented the gather and scatter-with-add kernels described in Section 4 in Fortran M, and are in the process of developing a complete PIC application. The Fortran M codes were simple to develop, but are currently unoptimized. We intend to investigate optimization strategies in the future. The key question here is whether the task of optimizing the Fortran M code significantly detracts from its programmability, which after all is the main reason for using it. One optimization strategy is to reduce the *volume* of interprocess communication. One way to do this is to eliminate the scatter and gather processes, and their child router and forward processes, and do all the routing of messages directly in the subgroup and subgrid processes. This makes the code less modular, and rather inelegant.

Another important optimization strategy is to reduce the *frequency* of interprocess communication. In the current code, communication in fine-grained. In the scatter phase each particle sends data to each of the corners of its home cell one at a time. Similarly, in the gather phase each particle receives data separately from each of these corners. In general, the corners of a particular cell all lie in the same subgrid process, except for cells on subgrid boundaries. Thus, performance could be improved by designating the lower, left, front corner of a cell as the root vertex for the cell, and sending current density contributions from each particle to this root vertex. For 3D grids this would reduce the frequency of communication by about a factor of 8. The gather phase could be performed in a similar way, by first gathering electromagnetic field information for each cell to the root vertex, and then gathering it to the particles.

A more general approach to increasing the granularity of communication is to buffer messages in the scatter and gather processes. The idea here is to defer sending data out of a router process outport until the amount of data to be sent exceeds a specified threshold. This threshold can be used to adjust the granularity of communication, and hence to optimize performance.

We are also planning to investigate other parallel programming methodologies to gauge their suitability for PIC and other applications. One approach of interest is based on the shared object abstraction as embodied in the Orca language. Orca is a procedural, type-secure language that supports shared objects. In Orca a shared object is an instance of an abstract data type consisting of a specification of the operations that can be applied to an object, and a declaration of variables and code for these operations. Shared objects are transparently replicated through the machine, and replication, consistency, and object placement are managed by the Orca runtime system. The obvi-

ous way to write a PIC application in Orca is to distribute the particles nonspatially, as in our Fortran M implementation, and to divide the grid into shared subgrids. To clarify the issues that arise, consider the extreme case in which there is just a single shared subgrid – namely, the entire grid. Since operations on shared objects are atomic, having just one shared grid essentially serializes the scatter phase. Memory constraints may also preclude sharing the entire grid. Increasing the number of shared subgrids increases the parallelism in the scatter phase, but having too many subgrids may place too great a burden on the Orca runtime system. As in the Fortran M implementation, improved performance may result if the granularity of communication is increased, for example, by scattering to all corners of a cell in a single operation, rather than scattering to each corner in separate operations. The granularity may be increased further by maintaining local copies of grid point information that are accessed and updated locally, and used to periodically update the shared subgrids. A drawback of this approach is that the data structures for storing the local copies are dynamic since as particles move the set of grid points with which they interact changes. A simpler approach is to maintain a local buffer for each shared subgrid and to store deferred remote operations in these buffers. This is essentially the same as buffering messages in the scatter and router processes in our Fortran M implementation.

We also intend to develop a version of the PIC application in the Split-C language, which is a parallel extension to C. The extensions that Split-C provides are few in number, but form the basis for a rich parallel programming environment. A shared memory programming style is supported through the use of global pointers. Split-C also provides split-phase access to remote data, allowing remote accesses to be overlapped with useful computation. The scatter phase of the PIC algorithm can be implemented using the split-phase *put* assignment operation, while the gather phase can be implemented using a *get* assignment operation. The signaling store operation, which stores to a remote location and signals the processor containing the location that the store has occurred, may also be used in the scatter phase. Finally, Split-C provides bulk data operations that can be used to increase the granularity of communication. Here again the idea is to maintain local copies of grid point information that can be used periodically to update grid points on remote processors.

## 6. CONCLUSIONS

We have described our progress in implementing the scatter and gather phases of the PIC algorithm in Fortran M, and discussed strategies for optimization. We have also sketched ideas for implementing the PIC algorithm using the Orca shared object language, and the Split-C parallel language. In all three of these programming methodologies we expect increasing the granularity of communication to be important for improving performance. The use of split-phase communication is also likely to prove a useful means of optimizing codes. Our future work in this area will be directed at investigating which optimization strategies are the most effective and how they impact the programmability and maintainability of the code.

## REFERENCES


Bal, H. E. 1990. *Programming Distributed Systems.* Silicon Press, New Jersey.

Bal, H. E.; Kaashoek, M. F.; and Tanenbaum, A. S. 1992. "Orca: A Language for Parallel Programming of Distributed Systems." *IEEE Trans. Software Engineering* 18, no. 3:190-205.

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

Culler, D. E.; Dusseau, A.; Goldstein, S. C.; Krishnamurthy, A.; Lumetta, S.; von Eicken, T.; and Yelick, K. 1993. "Parallel Programming in Split-C." In *Proceedings of Supercomputing '93* (Portland, OR, Nov. 15-19). IEEE Computer Soc. Press, Los Alamitos, CA, 262-273.

Foster, I. T. and Chandy, K. M.. 1992. "Fortran M: A Language for Modular Parallel Programming." Technical Report, Argonne National Laboratory.

Foster, I. T.; Olson, R.; and Tuecke, S. 1993. "Programming in Fortran M." Technical Report, Argonne National Laboratory.

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

Li, K. and Hudak, P. 1989. "Memory Coherence in Shared Virtual Memory Systems." *ACM Trans. Computers Systems* 7, no. 4 (Nov.): 321-359.

MPI. 1993. "Document for a Standard Message-Passing Interface." Technical Report CS-93-214, Department of Computer Science, University of Tennessee, Knoxville, TN.

Otto, S. W. 1991. "MetaMP: A Higher Level Abstraction for Message Passing Programming." Technical Report CS/E 91-003, Department of Computer Science and Engineering, Oregon Graduate Institute.

Otto, S. W. 1994. "Parallel Array Classes and Lightweight Sharing Mechanisms." *Scientific Computing* 2, no. 4, 203-216.

Parallel Computing. 1994. Special Issue on Message Passing.

Walker, D. W. 1990. "Characterizing the Parallel Performance of a Large Scale, Particle-in-Cell Plasma Simulation Code." *Concurrency: Practice and Experience* 2, no. 4 (Dec.): 257-288.

Walker, D. W. 1991. "Particle-in-Cell Simulations on the Connection Machine." *Computing Systems in Engineering* 2, no. 2/3: 307-319.

Walker, D. W. 1994. "The Design of a Standard Message Passing Interface for Distributed Memory Concurrent

Computers." *Parallel Computing* 20, no. 4 (April): 657-673.