Hardware and Software Architecture for Irregular Problem Architecture

Geoffrey Fox

CRPC-TR91164
July, 1991

Center for Research on Parallel Computation
Rice University
P.O. Box 1892
Houston, TX 77251-1892

SCCS–111

CRPC-TR91164

Hardware and Software Architectures for Irregular Problem Architectures

Geoffrey C. Fox

Syracuse University

Northeast Parallel Architectures Center

111 College Place

Syracuse, New York 13244-4100

gcf@nova.npac.syr.edu

Invited Talk at ICASE Workshop on

Unstructured Scientific Computation

on Scalable Microprocessors

Nags Head, North Carolina October 29–31, 1990

Hardware and Software Architectures for Irregular Problem Architectures

Geoffrey C. Fox
Northeast Parallel Architectures Center
111 College Place
Syracuse, New York 13244-4100
gcf@nova.npac.syr.edu

Abstract

We study the hardware and, especially, software support for unstructured scientific simulations in the context of an application classification in terms of a problem architecture. We suggest that extensions of Fortran 90 may provide an overall framework for handling general loosely synchronous problems. These include over 90% of current scientific simulations.

1 Introduction

We will discuss the parallel computer software and hardware architectures needed to support "unstructured scientific simulation" or, more precisely, what we have termed irregular loosely synchronous problems. We will focus on software which we believe is the main limitation to the rapid adoption of parallel machines in the mainstream of computing. Unstructured problems are particularly challenging to both hardware and software. They require more flexible hardware and usually faster communication (compared to calculation) performance than simpler regular computations. Many types of unstructured problems have been successfully parallelized, but I suggest that the software has been difficult to write. Expressing the irregularities and efficient mapping onto particular machines currently needs tedious nonportable code. Thus, this class of problem is a very important one for motivating new approaches to parallel software. We do not need a new language to express matrix multiplication or the solution of Laplace's equation in a rectangular domain on a parallel machine. Rather, it is the unstructured problems which should be one of the major targets of work on new software approaches to scientific simulations.

We suggest that the close coupling between hardware, software, and problem structures emphasizes the value for interdisciplinary research. This was a feature of the Caltech Concurrent Computation Program (C^3P) which I have described elsewhere [Fox:87d, Fox:88oo, Fox:90h]. This project taught me that "parallel computing works," but also that we did not yet have the correct software tools to tackle unstructured problems. We also found it quite hard, at times, to express the "obvious" parallelism in the problem. We will see examples of this in Section 5. We are continuing this work at Syracuse with major portable parallel software projects described in Section 4 and Section 6. Interdisciplinary research is centered on a new computational

science program at Syracuse offering degrees at the undergraduate, masters and Ph.D. level. We are also emphasizing applications of parallel computing to industry. We find more irregular problems here than in the academic applications which have been the dominant domain of parallel computing up to now. The irregularity in industrial applications is a natural consequence of simulating the details of the "real world". As industrial use of parallel computing is essential for the field to advance, we see again the importance of examining and developing tools for unstructured problems.

In Section 2, we review the current status of our problem architecture classification, and Section 3 a corresponding approach to software. In Section 4, we discuss the Rice-Syracuse proposal of Fortran D as a portable language for synchronous problems. Section 5 and Section 6 describe the unstructured "skeletons in the programming closet," and a proposed software approach based on extensions of Fortran 90D.

2 Problem Architectures

We have introduced three broad classes of problem architectures [Fox:88b, Denning:90a, Angus:90a]. These were deduced from our experience at Caltech combined with a literature survey involving 400 papers, which was reasonably complete up to the middle of 1989. At Caltech, we developed some 50 applications on parallel machines of which 25 led to publications in the scientific literature describing the results of simulations performed on our parallel computers [Fox:87d, Fox:88a, Fox:88oo, Fox:89n]. This analysis led us to introduce three broad classes of problem architectures that technically describe the temporal (time or synchronization) structure of the problem [Fox:88b]. Further detail is contained in the spatial structure or computational graph describing the problem at a given instance of simulation time [Fox:88tt]. Here, we only need to consider "embarrassingly parallel" problems, where there is little or no spatial connection between the individual parallel program components. For embarrassingly parallel problems, the synchronization (both software and hardware) issues are greatly simplified.

We have recently realized that we need to extend our classification, and that our original proposal should only be applied to individual program modules [Fox 91c, Fox:91d].

The three general temporal structures for program modules are called synchronous, loosely synchronous, and asynchronous, which we sometimes shorten to Classes I, II, and III, respectively. The temporal structure of a problem is analogous to the hardware classification into SIMD and MIMD. The spatial structure of a problem is analogous to the interconnect or topology of the hardware. The detailed spatial structure is important in determining the performance of an implementation [Fox:88a], but it does not affect the broad programming issues discussed here.

Synchronous problems are data parallel in the language of Hillis [Hillis:87a] with the restriction that each data point is evolved in time with the same procedure. The problem is synchronized microscopically at each computer clock cycle. Such problems are particularly common in academia, as they naturally arise in any description of

some world in terms of identical fundamental units. This is illustrated by quantum chromodynamics (QCD) simulations of the fundamental elementary particles which involve a set of gluon and quark fields on a regular four-dimensional lattice. These computations form the largest use of supercomputer time in academia [Baillie:89e, Baillie:90f, Baillie:90n].

Loosely synchronous problems are also typically data parallel, but now we allow different data points to be evolved with distinct algorithms. Such problems appear whenever one describes the world macroscopically in terms of the interactions between irregular inhomogeneous objects evolved in a time synchronized fashion. Typical examples are computer or biological circuit simulations where different components or neurons are linked irregularly and modeled differently. Time-driven simulations and iterative procedures are not synchronized at each microscopic computer clock cycle, but rather only macroscopically "every now and then" at the end of an iteration or a simulation time step.

Loosely synchronous problems are spatially irregular, but temporally regular. The final *asynchronous* class is irregular in space and time. A good example is an event-driven simulation which can be used to describe the irregular circuits we discussed above, but now the event paradigm replaces the regular time-stepped simulation. Other examples include computer chess [Felten:88i] and transaction analysis. Asynchronous problems are hard to parallelize unless they are "embarrassingly parallel" (termed Class III-EP). More general asynchronous applications require sophisticated software and hardware support to properly synchronize the nodes of the parallel machine, as is illustrated by the time-warp mechanism for event-driven simulations [Wieland:89a].

Unstructured problems could be either loosely synchronous or asynchronous. Clearly, a "war game" is an unstructured simulation that is usually implemented as an asynchronous event-driven simulation. However, we will *not* discuss asynchronous problems in Sections 4–6; we will concentrate on the irregular loosely synchronous case that is, in fact, a possible implementation of most war games, and the dominant methodology for irregular scientific simulations.

Synchronous or loosely synchronous problems parallelize on systems with many nodes. The algorithm naturally synchronizes the parallel components of the problem without any of the complex software or hardware synchronization mentioned above for event-driven simulations. 90% of the surveyed applications fall into the classes that parallelize well. This also includes the embarrassingly parallel I, II, III-EP classes. It is interesting that massively parallel distributed memory MIMD machines that have an asynchronous hardware architecture are perhaps most important for loosely synchronous unstructured (scientific) problems.

As described in [Fox:91c, Fox:91d], many important problems have heterogeneous architectures, illustrated in Figure 1, which points out the analogy between heterogeneous problems and networks of heterogeneous machines. We term this mixed class IIICG–IIFG to indicate it contains an asynchronous coarse grain mixture of loosely synchronous (data parallel) fine grain modules. This figure uses an example com-

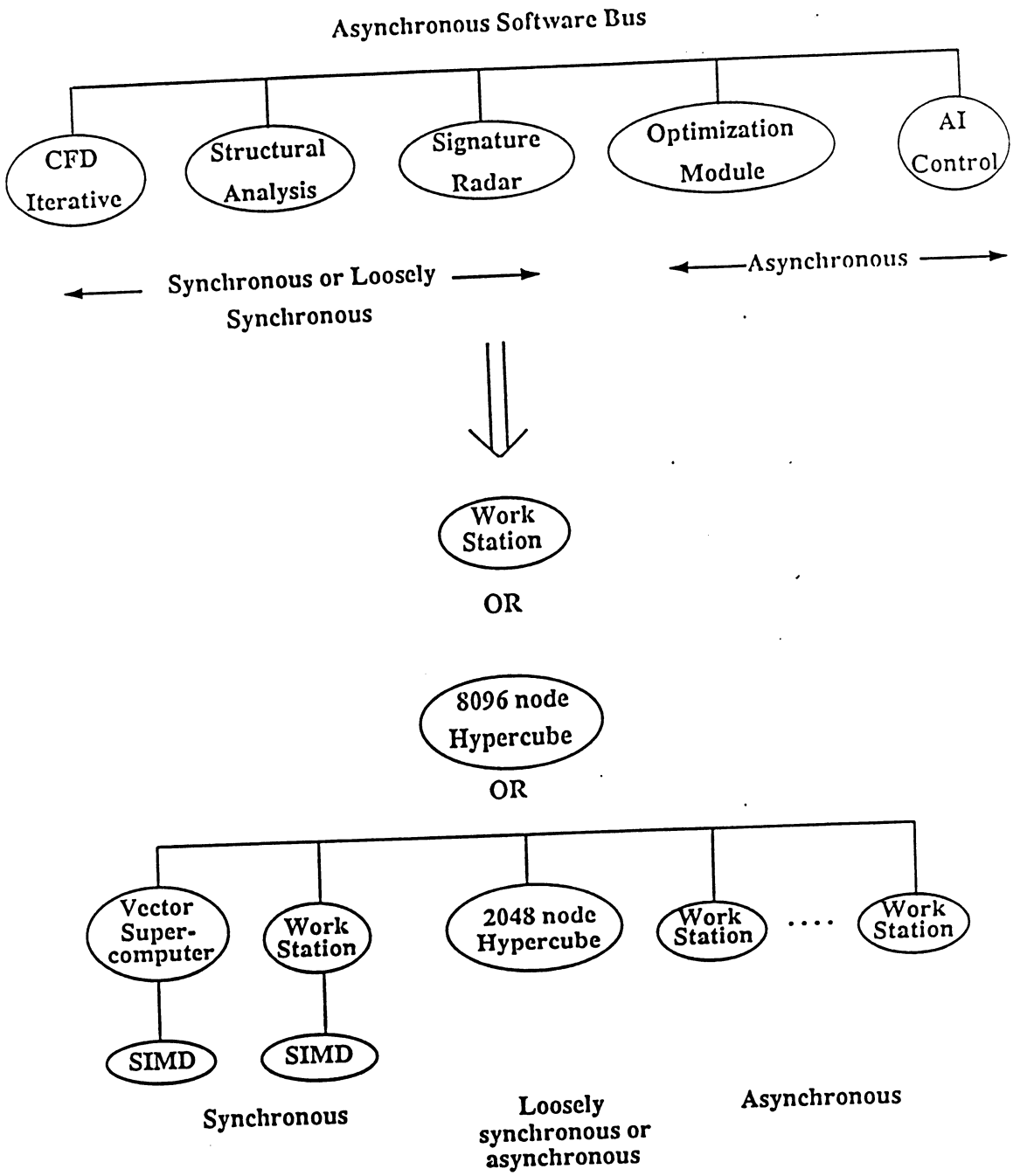

Figure 1. The mapping of heterogeneous problems onto heterogeneous computers illustrated for an integrated approach to the design of a new airframe

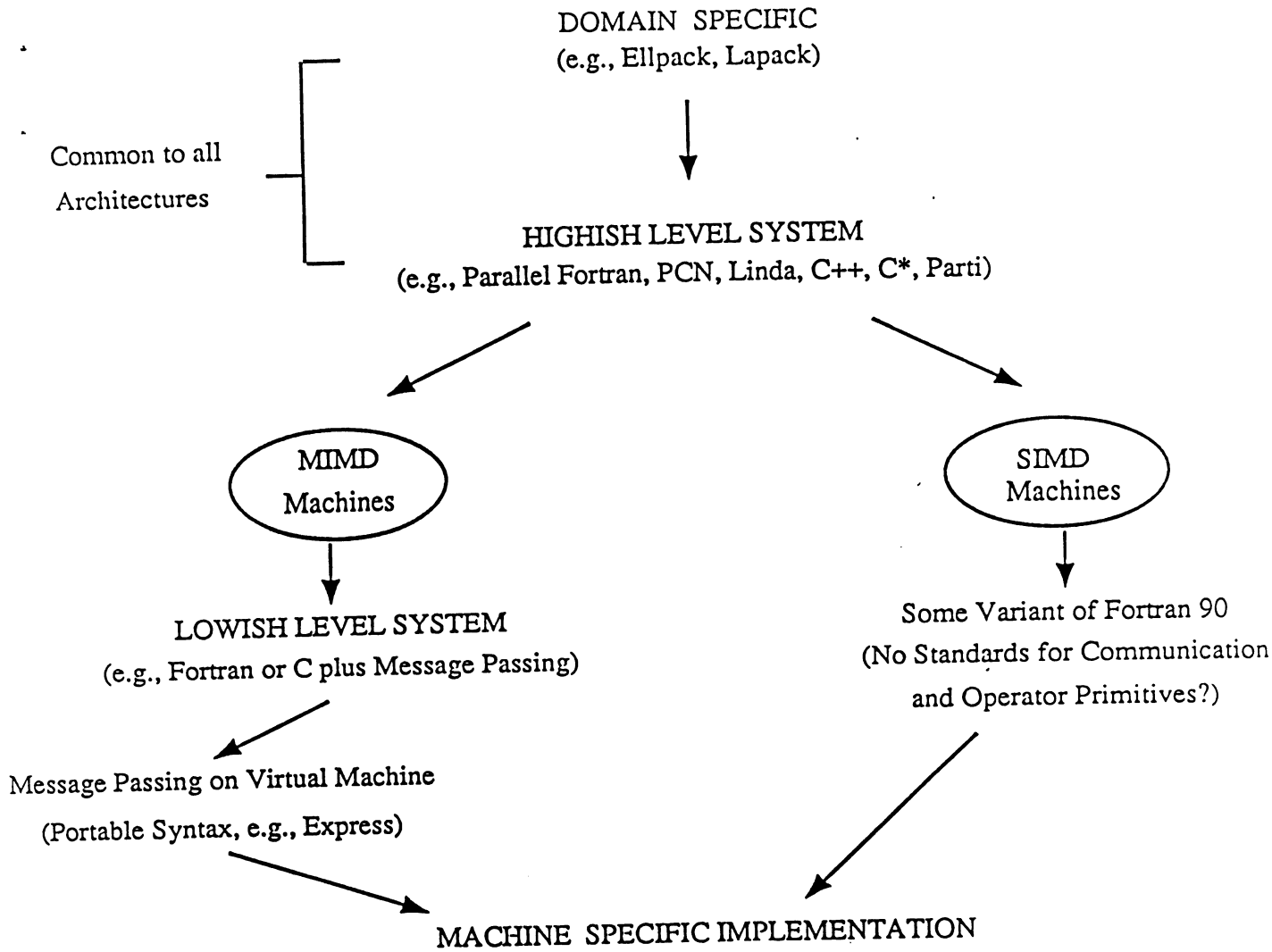

Figure 2. Software layers for parallel machines

ing from an integrated approach to the optimization of a new airframe. This could involve the integration of airflow, structural and radar signature modules with an artificial intelligence optimization module. The fields of vision (and related areas, such as robotics), or more generally information systems, produce such heterogeneous problems. We will not need to study this here, but concentrate on irregular loosely synchronous problems. Note that both asynchronous and loosely synchronous problems are irregular. In asynchronous problems, the irregularities are usually dynamic and cannot, and need not, be exploited; subtle mappings that minimize communication are not required, but rather one sees statistical methods of load balancing underlying the decomposition. In the loosely synchronous case, the irregularities usually come from an underlying data parallelism. Expressing this properly leads to important performance gains. This difference is reflected in the software support. In the asynchronous case, one stresses flexible communication in an object oriented approach. In the loosely synchronous case, we search for high-level data structures, and runtime support for optimal geometric decompositions. We also note that loosely synchronous problems parallelize, albeit with software difficulties but that for asynchronous problems, parallelism is a serious difficulty as well as the software.

3 A Portable Software Strategy

In Figure 2, we follow K. Kennedy and divide software into several layers where, here, we are interested in the higher levels where problems can be portable to a broad range of machines, in particular, having a SIMD or MIMD architecture. The domain specific levels include environments aimed at linear algebra (Lapack) or differential equations (Ellpack, Pcgpack ...). Here, we will discuss the next lowest level, which has broad applicability to many problems but is still portable over a range of machine architectures. As expanded in Figure 3, we view software as a mapping of problems onto machines. We expect that portable user-friendly software will reflect the general characteristics of the problem architecture, and *not* the machine architecture. This is what we mean by portable. From this point of view, we consider Fortran (or C) plus message passing as a low-level non-portable approach to parallel programming. This software model reflects the machine architecture of MIMD nodes communicating with messages. We should emphasize that this approach (mainly C plus message passing) dominated our work at Caltech, and is currently the only generally effective approach to MIMD programming. This is especially true for unstructured problems, where we have successfully parallelized the applications described in Section 5 in C plus message passing, but as revealed in Section 6, have only some suggestions as to possible portable higher level systems.

We suggest that the problem architecture classification introduced in Section 2 provides a reasonable framework to describe possible portable software models. Just as we found three classes for problem modules, we will not propose a single high-level software model. Rather, we will need several approaches, including one that allows the integration of modules of disparate architectures in heterogeneous problems.

We now discuss how, for program modules of our three temporal classes, we can suggest portable software paradigms.

Class I Synchronous Problems

These problems are tightly coupled synchronous problems which are regular in space and time. Their data, or geometric parallelism, can be naturally expressed in Fortran 90D (appropriately extended Fortran 90) [Fox:91c, Fox:91e] or similar languages such as CM Fortran, Crystal [Chen:88b], C* [Quinn:90a, Quinn:90b], or even APL. This allows the user to specify the problem structure in a natural high-level fashion using the vector and matrix constructs of Fortran 90. The compiler can take care of mapping this onto different machines, including those of SIMD and MIMD architecture [Fox:90h, Wu:91a]. We will describe this in more detail in Section 4 where we note that we have defined extensions to Fortran 77, called Fortran 77D, which will allow Fortran 77, as well as Fortran 90, to be used in this problem class.

Class III Asynchronous Problems

This class is irregular in space and time and often exhibits functional or process parallelism. One good example of this class is the simulation of sensors and control platforms involved in your favorite SDI battle management architecture [Meier:90a, Fox:91d]. Here, the basic components communicate with messages in the real world and so, at this, we can hardly complain about software models involving message passing! Thus, in this problem, we see a natural breakup into processes and message passing at the problem level, and software engineering approaches, such as object oriented programming, ADA, C++, Strand [Foster:90a], PCN [Chandy:90a], ISIS [Birman:91a] or Linda [Gelernter:89a] are possibilities. In many cases, we do not need to use carefully optimized decompositions but, rather, use statistical load balancing and decomposition methods. This problem class includes distributed computing and the software, such as ISIS, designed to support it. As well as this loosely coupled category, we also see the event-driven simulations with their specialized software, which we mentioned in Section 2.

The heterogeneous problem of Figure 1 is supported by similar software models, but we will need an important extension. Each module can be viewed as an object to be supported in an asynchronous software model. However, we must further allow parallelization internal to the module. Thus, we see the need for systems, such as Strand, supporting Fortran 90 processes or ADA supporting calls to data parallel C*.

Class II Loosely Synchronous Problems

These problems are irregular in space, but regular in time. Often, their spatial structure changes dynamically, and adaptive algorithms are needed. As already mentioned, this class is difficult because the tightly coupled spatial structure demands the same kind of detailed optimizations provided by the Fortran 90 compiler for Class I. However, the irregularities make this hard to implement.

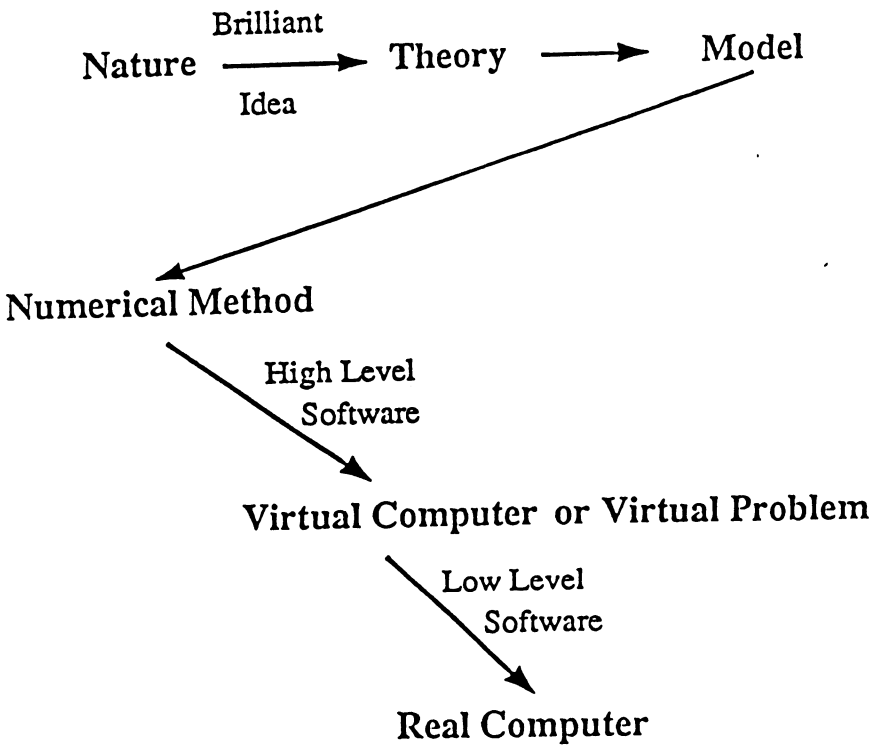

Figure 3. Computation and simulation as a series of maps

Table 1A. Gaussian Elimination (256 x 256 matrix)

	Number of Processors on iPSC2 Hypercube				
	1	2	4	8	16
Original Hand Coded F77+MP	85.4	58.1	31.1	16.0	8.42
F77+MP from Fortran 90	80.0	50.2	26.6	13.8	7.72
Revised Hand Coded F77+MP	73.4	50.1	26.9	13.8	7.53

Table 1B. N-body Simulation (1024 Particles)

	Number of Processors on iPSC2 Hypercube				
	1	2	4	8	16
Hand Coded F77+MP	71.7	35.9	17.9	8.98	4.83
F77+MP from Fortran 90	139.6	69.1	35.5	18.1	9.40

Table 1C. Fast Fourier Transform (16384 Points)

	Number of Processors on iPSC2 Hypercube				
	1	2	4	8	16
Original Hand Coded F77+MP	36.8	23.3	14.2	8.32	4.82
Optimized Hand Coded F77+MP	13.0	6.67	3.42	1.75	0.91
Optimized F77+MP from Fortran 90	18.8	10.1	5.36	2.84	1.50

We know that Fortran (C) plus message passing works for this problem class, but we need a more portable user friendly approach. This can involve new data structures to extend languages like Fortran 90. It needs sophisticated run time support, such as that provided by the PARTI system from ICASE [Saltz:91b]. In particular, we need dynamic load balancing modules for which the basic research has been done, but no general implementations are yet available [Fox:88mm]. We will expand this brief discussion in Section 6.

4 Fortran D as a Portable Parallel Software Environment for Synchronous Problems

Although direct parallelization of Fortran 77 has proven to be very difficult, we believe that one can build an excellent portable Fortran environment for synchronous (Class I) problems. This is the goal of a collaboration with K. Kennedy's group at Rice, my group at Syracuse, and the Parasoft Corporation, and Figure 4 illustrates our strategy. One needs to "help" the compiler disentangle the problem architecture by, for instance, specifying how the Fortran arrays are distributed over the distributed memory parallel machines. These extensions to Fortran 77 or Fortran 90 are called Fortran D [Fox:91e].

The success of CM Fortran as the programming environment for the CM-2 suggests that it is a good approach for our synchronous Class I applications. As discussed earlier, we view Fortran 90D (CM Fortran, C*) as programming systems for "SIMD" (synchronous) problems and not as languages for SIMD machines. Compilers can map Fortran 90D effectively into all parallel architectures suitable for this problem class including MIMD, SIMD parallel machines, systolic arrays, and heterogeneous networks. Fortran 90 was not originally designed as a massively parallel programming system, but it has one key attribute that makes it effective. It uses high-level data structures explicitly (as vectors and matrices), and so the problem architecture is clear and not hidden in values of pointers and DO loop indices. It is portable as high-level constructs—such as A = B*C with A, B, and C matrices—can be optimized by the compiler for each new machine. Our experience has been that, in many cases, users prefer Fortran 90 to Fortran 77 even for sequential applications, as it expresses applications naturally with much shorter code. Often, one finds a factor of two to three reduction in code size for Fortran 90 compared to Fortran 77.

We can illustrate the success of Fortran 90D by some test examples, shown in Table 1, which compare it with direct user coding of Fortran 77+ message passing [Fox:90h, Wu:91a].

For a problem with a simple topology, LU decomposition in Table 1A, the Fortran 90 code produced essentially as good a code as the direct Fortran 77+ Message Passing. Indeed, the "automatic" Fortran 90 procedure pointed out a possible improvement in our handed codes F77+MP; this is the difference between lines one and three of Table 1A. In Table 1B, our current automatic approach for the N-body problem loses a factor of two compared to the best parallel implementation; this is

due to inefficient communication, and one may need to change the Fortran 90 implementation to allow the compiler to optimize this. In this sense, the user will need to understand some issues of parallelism, even when writing "explicitly parallel" code as with Fortran 90. Note the example in Table 1B is the simple $O(N^2)$ algorithm and not the more interesting and challenging $O(N(\log N))$ approach discussed in Section 5. In Table 1C, we find a 50% degradation in performance on the FFT for the Fortran 90 approach. This indicates that Fortran 90 does not optimally support the hierarchical data structures found in the FFT. As we discuss in Section 6, we expect that the final Fortran 90D language will include new data structures—over and above the arrays and vectors in Fortran 90.

Finally, in Table 1D we come to a "real," albeit small in code size, problem. The original climate modeling code has been used in production [Keppenne:89a] on CRAY and SUN computers. We saw, in this project, an interesting division of labor. The first rewriting from C to Fortran 90 was performed by the application expert. The further conversions of Fortran 90 into Fortran 77 and Fortran 77+ message passing were performed by "computer scientists" without deep knowledge of the application [Keppenne:90a]. In this case, we believe that no automatic method could have parallelized the original C code, but that our planned automatic approach would be able to perform the MIMD parallelization from Fortran 90. The result of this project is a portable code running well on the CRAY, Connection Machine and hypercubes. Note that we even improved the sequential performance (line one vs. line three of Table 1D) by an order of magnitude. The original C code made extensive use of pointers which had several repercussions. It made vectorization hard on the CRAY; it made the code impossible to automatically parallelize as the "structure of problem" was expressed in dynamic pointer values; it made the code hard to port except by the domain expert.

Our initial experiments are sufficiently encouraging that we believe that a language like Fortran 90 will become an efficient vehicle for synchronous Class I applications. However, the use of Fortran 77 is, of course, critical due to the large amount of existing code and experience in this language. However, we can discuss Fortran 90D more straightforwardly as its array extensions are easier for the compiler to parallelize efficiently. In Figure 4, we can view Fortran 90D as a "permanent annotation language" for user assisted parallelization of Fortran 77. It will require more experimentation with real application codes to compare the relative merits of parallelizing Fortran 77D versus Fortran 90D.

5 Irregular Loosely Synchronous Problems

5.1 Overview

In the previous section, we have discussed software support for "simple" or synchronous problems. We can ask if the following problem classes are roughly identical:

- problems suitable for SIMD machines, such as the Maspar and CM-2;
- problems whose data structure is an array;
- problems that can be expressed easily in Fortran 90;

Table 1D. Climate Modeling Code [Keppenne:90a]

Implementation	Size of Code (lines)	Machine	Performance (megaflops)
Original C Code	1500	CRAY X-MP (1 C.P.U.)	~ 1
Fortran 90	600	8K CM-2	66
F77 Produced by Hand from Fortran 90	1500	CRAY Y-MP (1 C.P.U.)	20
F77+MP by Hand from Fortran 90	1650	NCUBE-1 16 node hypercube	3.3
		NCUBE-2 16 node hypercube	20
		INTEL i860 16 node hypercube	80

- problems that a semi-automatic compiler can parallelize from good Fortran 77 code;
- problems that run well on the Cray.

Now we start our collection of "skeletons in the programming closet" which fall outside the above classes; those irregular loosely synchronous problems which we will use to challenge and motivate our parallel software environments.

At the conference, we saw several examples of irregular meshes for partial differential equations. These could be adaptive and involve multigrid techniques. Another large class of irregular problems involve particle dynamics, where we describe a particularly hard example in Section 5.3. Other problems in our collection will be irregular circuit simulations from many fields including biology (cortex models), chip simulation, and electric and gas power distribution. Many of these applications can be handled by systems which offer decomposition tools, such as PARTI which support the distribution of irregular sparse matrices [Saltz:87a], [Saltz:91b].

What other applications should we use to challenge our software system? In Section 5.2, we will review Monte Carlo studies of materials near the critical point where dynamic irregular domains are formed and must be treated explicitly by the algorithm. This raises different issues that are similar to those in high-level image processing where regions need to be identified. A final difficult example can be seen in the generation of adaptive meshes; R. Williams has successfully parallelized this in two dimensions, but the tricky parallel algorithm and use of linked lists are not obviously expressible in a high-level fashion [Williams:88a], [Williams:89b], [Williams:90b].

These examples share certain characteristics.

1) They exhibit natural parallelism (loosely synchronous irregular "data parallelism").
2) They involve either hierarchical data structures or, more simply, can be expressed as an irregular sparse matrix.
3) They typically run more straightforwardly on MIMD than SIMD architectures.
4) We have been able to parallelize these applications "by hand"; however, both the parallel algorithm and software development have been hard.

In Section 6, we will study implications of these examples and characteristics for the parallel software support. Here we discuss, in more detail, two physics applications mentioned above. The multigrid method is familiar and successful in the solution of partial differential equations and can be viewed as an algorithm that properly treats all length scales in the problem by refining them on the same iteration time scale. The methods of the next two sections reflect a similar physical idea, but for different applications; statistical physics and particle dynamics.

5.2 Clustering Models for Statistical Physics

We will discuss the concepts for a caricature of the real problem shown in Figure 5 as a 12×10 array of spins (the Ising model)—where each spin can be up or down. The conventional iterative Monte Carlo approach to such problems involves cycling through each spin, one at a time, deciding whether or not to change its value. Meth-

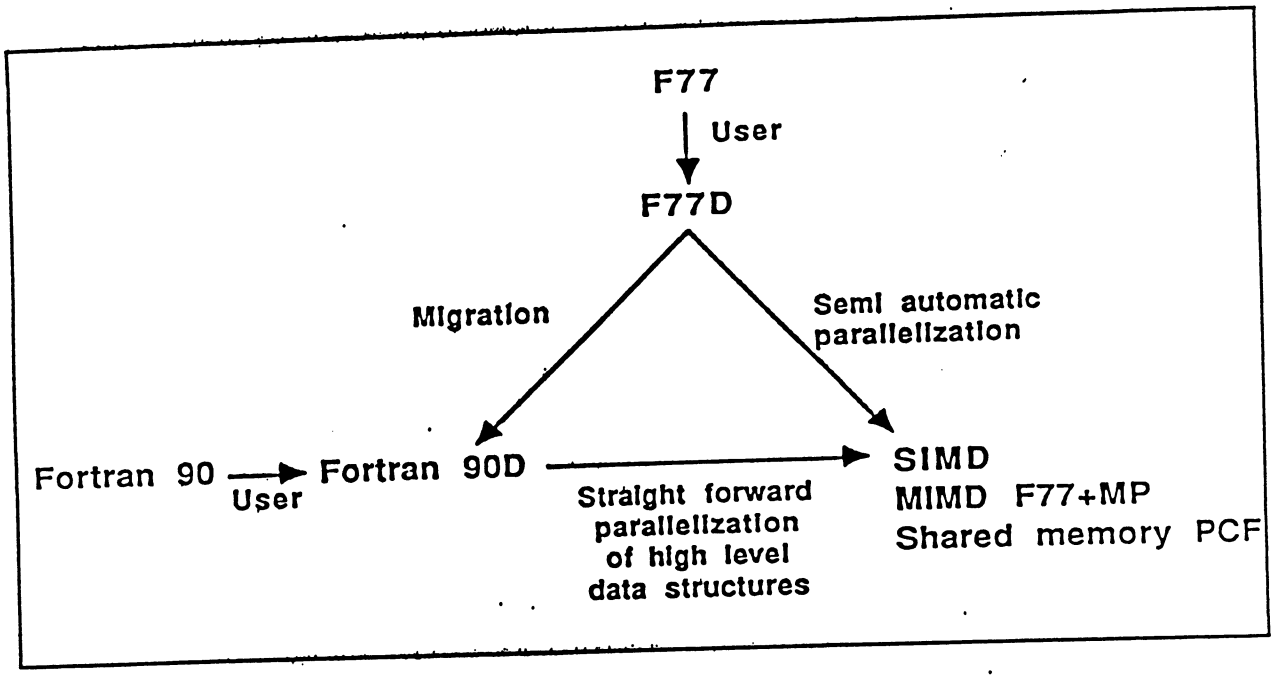

Figure 4. An Integrated Fortran Environment

ods such as Heat Bath or the Metropolis algorithm are used in this decision. Near a phase transition, neighboring spins are correlated and often domains form in which all the spins have the same direction. We can introduce a correlation length, τ, such that spins are correlated over this length scale. The traditional methods are still valid in this interesting region (which contains most of the important physics) where τ is large, but slow down because (roughly) the simulation time grows like τ^2; this can be seen intuitively from a random walk argument. Unfortunately, behavior near a critical point is a major goal of simulations, and the decreasing effectiveness of single-site updates is called "critical slowing down." We cannot directly apply multigrid as there is no continuity from site to site in the statistical configuration. However, the idea is the same; we must design an algorithm that properly treats effects on their physical length scale. Several ingenious algorithms have been found that remove the critical slowing down [Brower:91a], [Sokal:88a], [Swendsen:87a], [Wang:90a], [Wolff:90b]. Typically, one would find clusters, such as those shown in Figure 5 with blocks of dots or crosses. Clusters are then manipulated as a single entity. In our original work, the major difficulty was finding and labeling independent clusters. This was solved on a MIMD machine by using an optimal sequential algorithm within nodes, and a less efficient but parallelizable method to identify clusters between nodes. The results on the MIMD Symult-2010 are shown in Figure 6 [Baillie:91a]. Recently, we have extended the methods to the Connection Machine using the "scan" operations [Apostolakis:91b].

There are some interesting hardware implications. The basic single-site Monte Carlo runs well on SIMD machines, and the CM-2 is being used extensively for production QCD calculations of this type. However, the cluster algorithms are much more sensitive. They might need either MIMD or better communication/calculation bandwidth to perform well. Interestingly, nobody has yet to find a good clustering method for QCD, even though the large lattices (100^4) anticipated for QCD teraflop calculations would clearly gain much from explicit treatment of correlations. So, we may find the need to change the target architecture for this application.

5.3 Large N-body Calculations

We now describe clustering algorithms for particle dynamics. This was perhaps our most interesting project at Caltech, as it combined beautiful physics with challenging parallel algorithms.

The basic idea was developed by Appel, Barnes, Hut and Greengard [Appel:85a], [Barnes:86a], [Fox:89t], [Greengard:86a], [Greengard:88a], [Salmon:90a] where we have used the particular version developed (on a sequential machine) by Barnes and Hut. The essential point is shown in Figure 7. With a long-range force, as we get with the gravitational interaction between stars, the simple algorithm has a time complexity of $O(N^2)$ for N stars. However, we note that we can replace the interaction of complexity M in Figure 7, by a single interaction between a star and the centroid of the cluster. This idea can be applied recursively and refined by the use of multipole

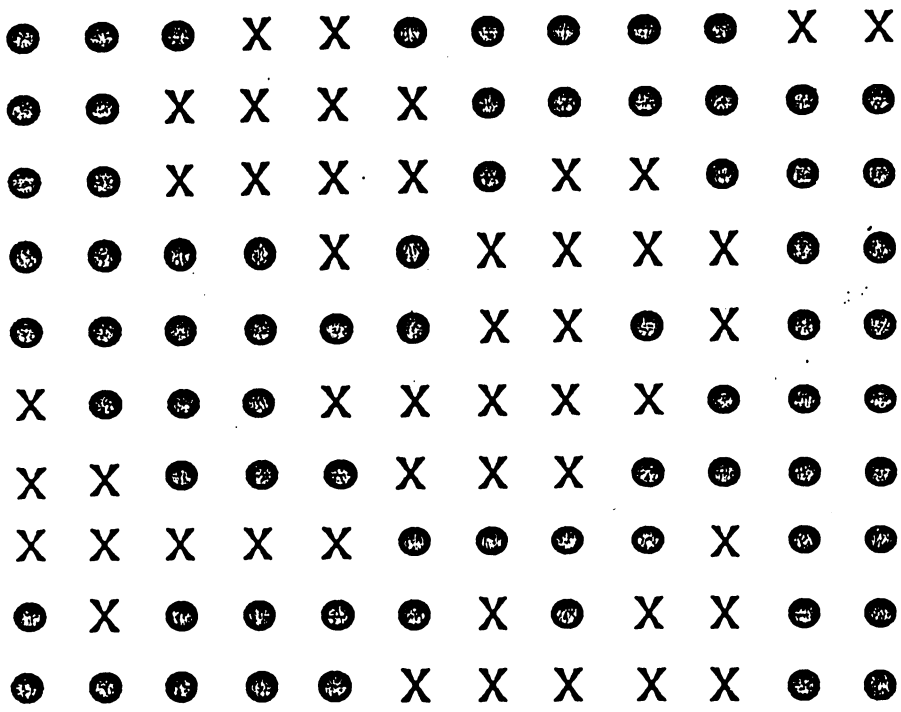

Figure 5. A 12 × 10 array of spins which are up (.) or down (x)

expansions instead of the single centroid term. J. Salmon has shown that if the original unaccelerated method has a time complexity of $T_{\text{naive}} = 0.5\,N^2\,t_{2\,\text{particle}}$, then the Barnes-Hut clustering algorithm has $T_{\text{cluster}} = (20-50)N \log N\, t_{2\,\text{particle}}$. Here $t_{2\,\text{particle}}$ is the time taken to calculate the interaction between two stars. For $N > 1000$, the clustering approach is more effective than the naive approach, and for the interesting case $N = 10^6$, the clustering problem is a thousand times faster. This acceleration allows a new class of calculation in the areas of

- galaxy structure and collision. Here, we ran several 2×10^5 "star" simulations on the 512-node NCUBE-1. The excellent speedup measurements for this are shown in Figure 9 [Salmon:89b].
- cosmological studies of the growth of fluctuations in the early universe. Some scientific results from this are shown in Figure 8 [Warren:91a].
- globular cluster simulations involve precisely of order million stars, but we need more powerful machines to handle the short-range interactions (binaries) found in this case.

Our original NCUBE-1 calculations were, at the time, the largest simulations performed in this field as they exploited the large (albeit distributed) memory available on the parallel machine. We noted [Fox:89i], [Fox:89n], [Fox:90o] that the NCUBE-1 was 50% efficient in this production run, but this unusually "bad" result (typically we find 80% or better efficiency on loosely synchronous problems) allowed the 512-node NCUBE-1 to deliver about twice the performance of the CRAY X-MP which only realized 5% efficiency. Often, one finds that irregular problems parallelize naturally but are hard to vectorize. The same approach and lessons can be used in other fields, such as the vortex approach to fluid dynamics [Pepin:90a], [Pepin:90b], molecular dynamics [Ding:90m] and plasma physics.

We now examine why this problem is hard to parallelize and challenging to the software environment.

Sequential Particle Clustering Algorithm

This builds up a tree by successive division of space—three dimensions in our real application, but shown in two dimensions in Figure 10. This illustrates an essential feature of this algorithm—namely, the data structure is an irregular dynamic tree that is rebuilt at each time step. On a sequential machine, tree building is only 2% of the total execution time, and so unimportant. As detailed by Salmon, the tree building is performed in steps.

- start with null tree
- add particles one at a time
- refine tree as necessary to ensure, at most, one particle in each cell (leaf node of tree)

The tree building is followed by the major computational part of the algorithm—namely, calculation of the force on each particle. Here, each force calculation starts at the root of the tree and traverses the tree downward stopping on each branch when

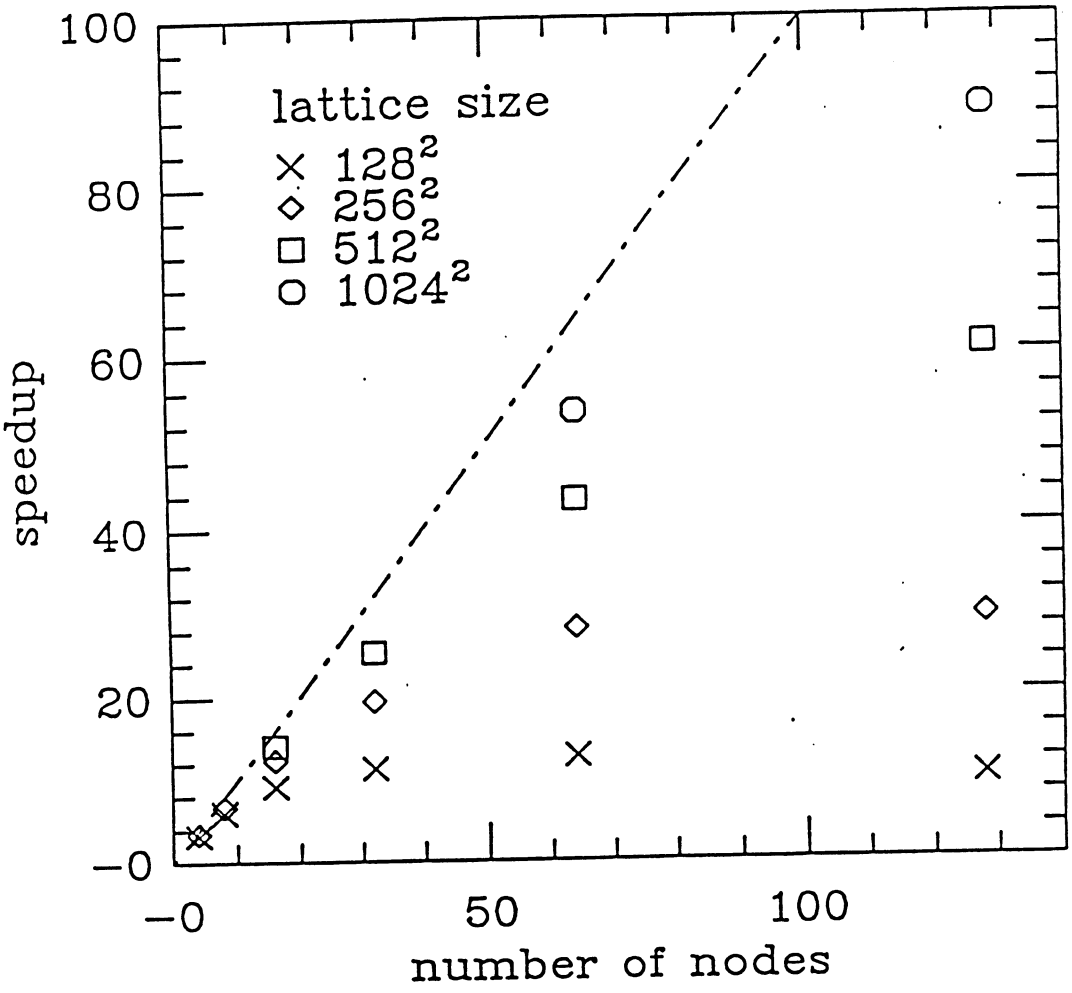

Figure 6. Speedup of cluster algorithm on the Symult-2010

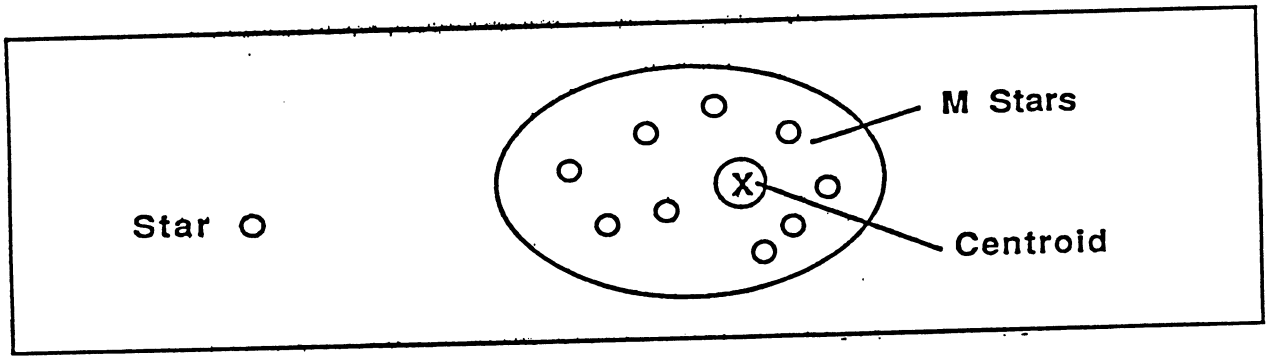

Figure 7. A Cluster of Stars Replaced by Their Centroid

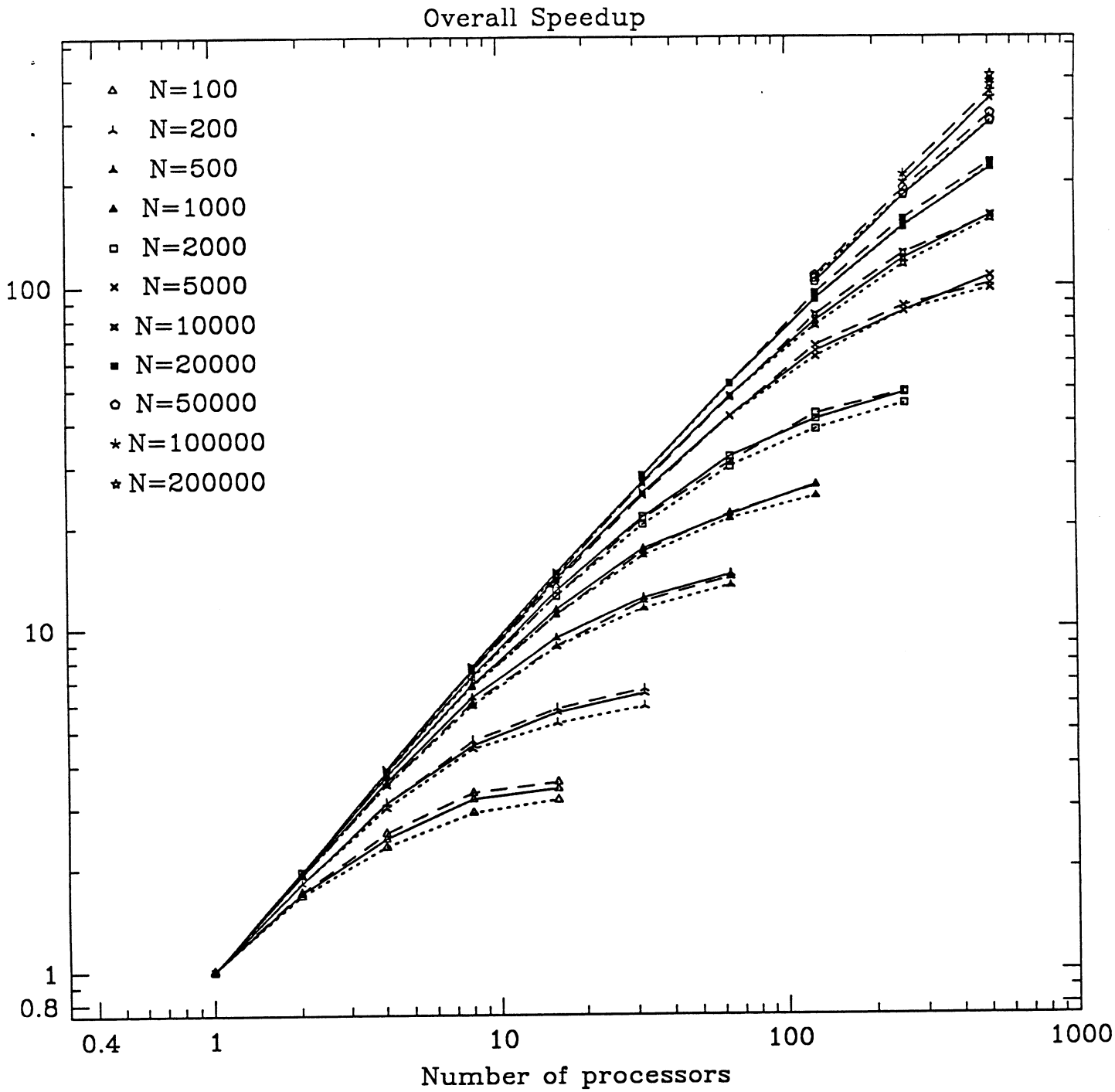

Figure 9. Speedup for three different simulations on the 512-node NCUBE-1 as a function of simulation and machine size

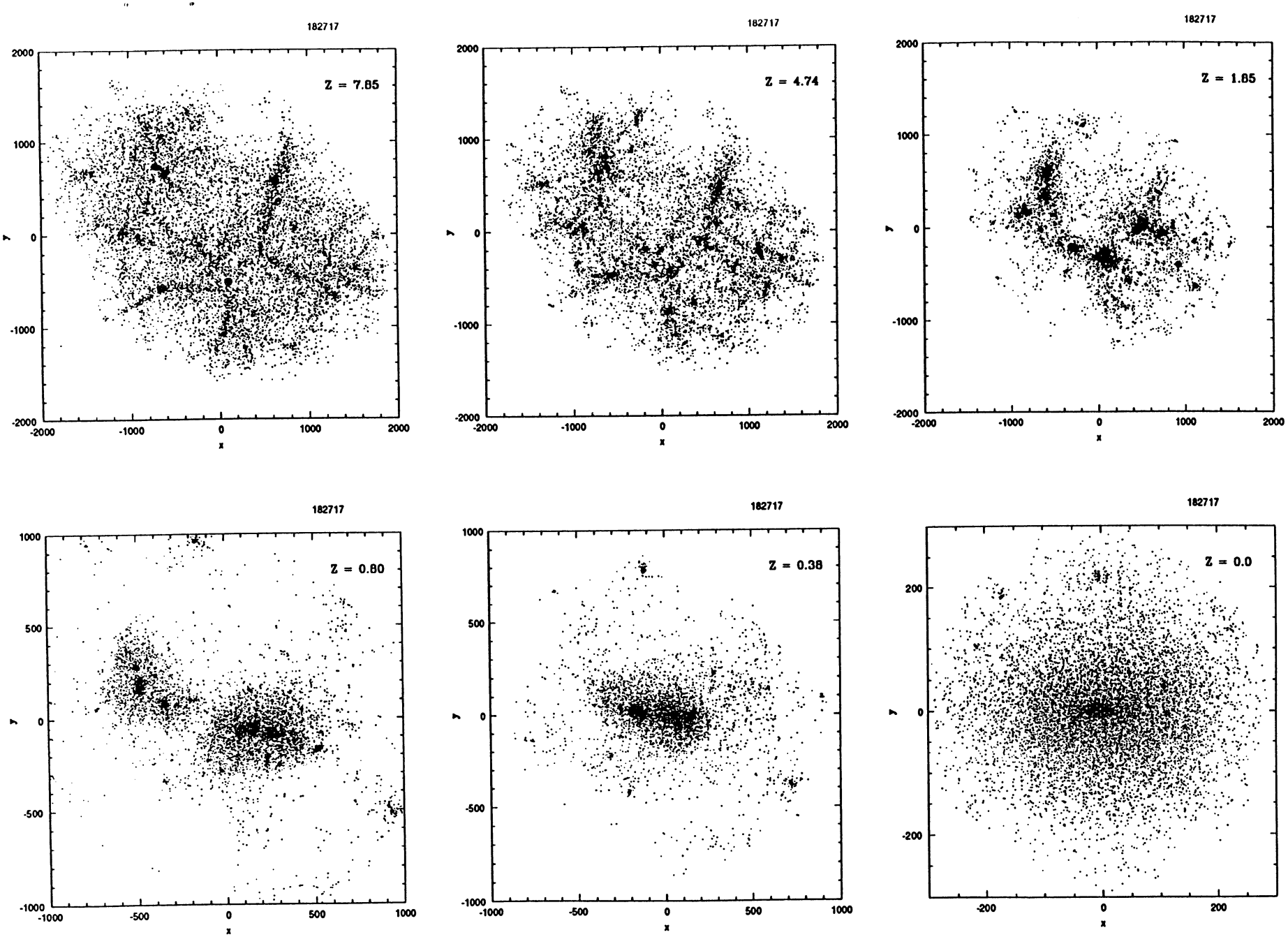

Figure 8. One Million Particle Simulation of Growth of Galaxies on the 64-node JPL Mark IIIfp Hypercube. We show the 20,000 particles that will form a galaxy.

"enough detail" has been exhibited. A typical search termination criterion is shown in Figure 11 where we only open up a cell and go further down the tree if the cell subtends a large enough angle at the star.

Parallel Clustering Algorithm

The parallel algorithm was developed by Salmon and Warren, and is described in Salmon's Ph.D. thesis and initially in Warren's undergraduate senior thesis report [Salmon:90a, Warren:88c].

The load balancing uses a heuristic, illustrated in Figure 12, which has been found useful in many other applications—namely, orthogonal recursive bisection along each direction of the hypercube. The "work" per star is very nonuniform as the density of particles can vary by a factor of a thousand between different regions of space—and there is much more calculation needed for a star in a dense region than in a quiet region. The work per star is found at each time step and used to balance the load for the next time iteration. Thus, we have a fully dynamic load balancing.

The calculation of the tree was the hardest part of the parallel algorithm. Although it only took 1/50th of the sequential time, we wanted to run on at least 512 nodes, and so we did not have the memory or time to compute the tree on a single node and broadcast to all others.

One essential idea can be expressed in the concept of *locally essential data*. This implies that each node will calculate and store only that part of the tree it will need to calculate the force on each particle stored in the node. The concept is quite general. We can view locally essential data as "the world as seen from the node" with fine detail at short distances and coarse information at long distances. Figure 13 shows that, for a simple partial differential equation, the locally essential data is that stored in the node plus the surround guard ring of grid points. In an iteration or time-stepped simulation, one would communicate between processors to ensure each has their locally essential data. Then the equation, or particles, can be calculated separately in each node without further communication.

Figure 14 shows the far subtler locally essential data for processor 0 in our tree problem defined in Figure 10 and Figure 12. Comparison of Figure 10 and Figure 14 shows that the latter has the detailed tree near the stars stored in node 0, but only the coarse levels of the tree for array. The top levels of the tree is now stored in all nodes and this "explains" why there is no sequential bottleneck at the root of the tree. However, how do we elegantly express the above argument in a portable efficient fashion? I do not know—Salmon's implementation proceeds as follows. The full tree of Figure 10 is never calculated in any one place—rather, we just calculate the locally essential trees which are naturally found in parallel. The algorithm for this involves a loop over hypercube channels where each processor sends to its neighbor in the hypercube any information that might be needed to calculate locally essential data by any processor on the "other" side of the particular hypercube channel. This step is followed by a tree update and pruning.

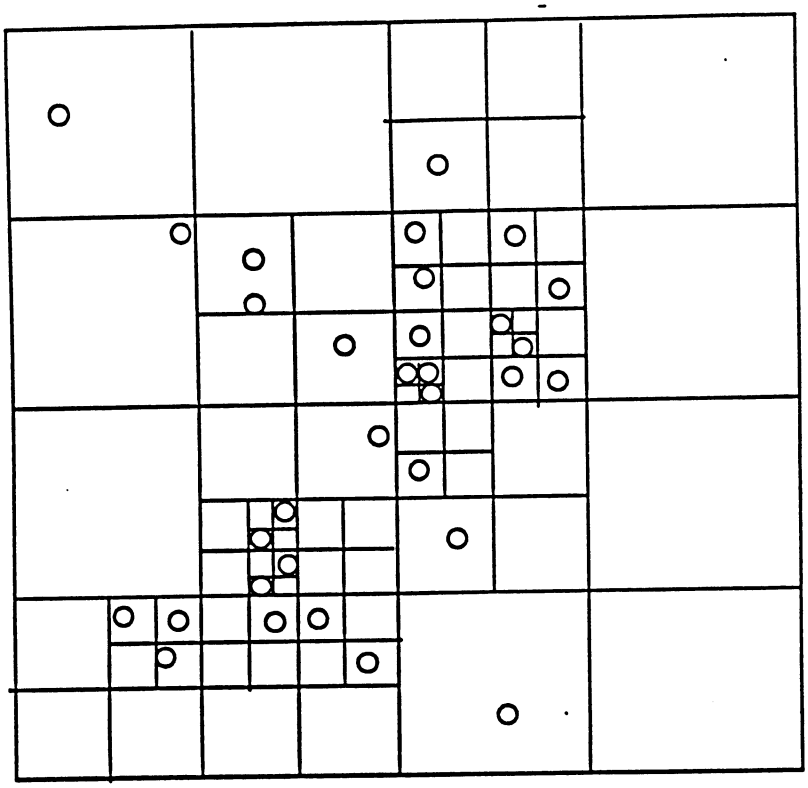

Figure 10. A Complex (tree-like) Data Structure not well expressed in current versions of Sequential or Parallel Fortran

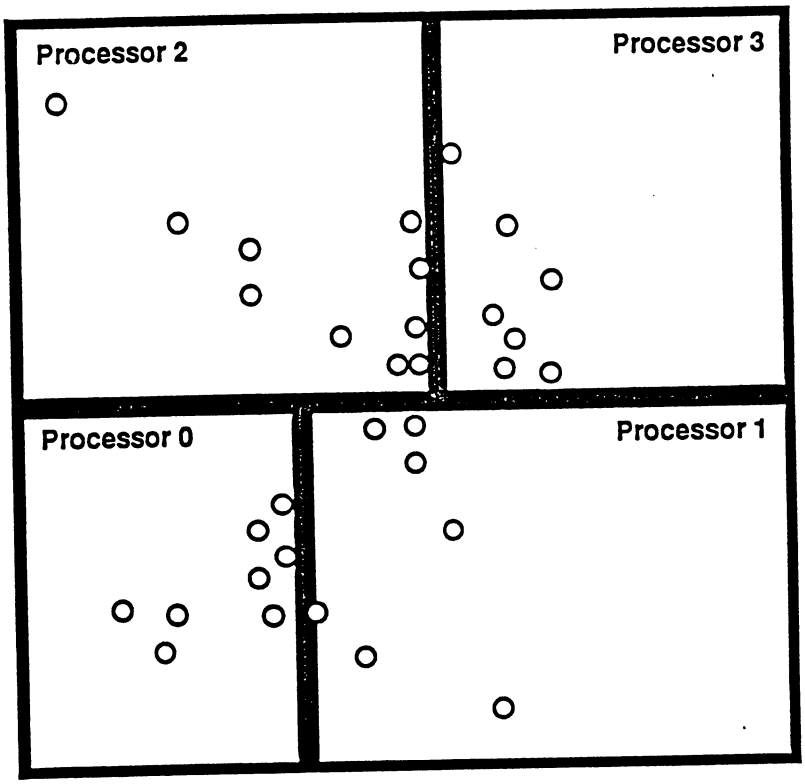

Figure 12. Dynamic Domain Decomposition Illustrated for Four Processors

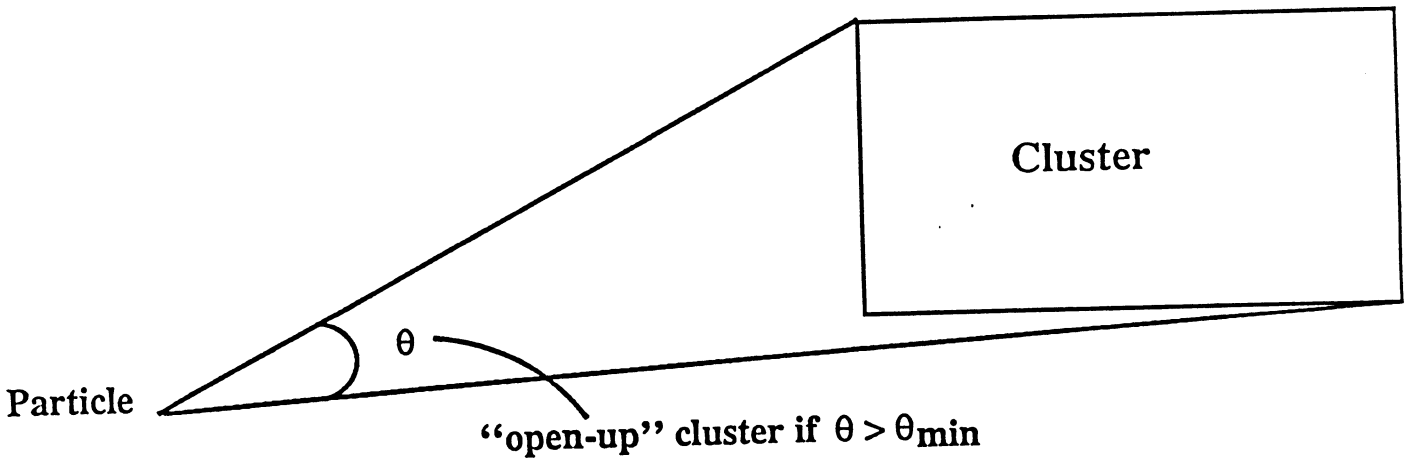

Figure 11. The Criterion for Terminating Search of Tree

Our discussion appears rather hardware specific and indeed the implementation was designed for the NCUBE-1 which is a hypercube. However, hierarchical data structures are naturally supported by a hypercube, and this is a "correct virtual machine" which can then be optimally mapped to the physical hardware topology.

There is a good analogy to the method used above to calculate the locally essential tree in the crystal accumulator or fetch and add algorithm [Fox:88a], [Fox:88g]. Consider

$$y_i = \sum_j A_{ij}\, x_j = \sum_j e_j^i$$

where x_j and y_i are both distributed. Then one calculates the sum over j incrementally as the components e_j^i cross hypercube channels, they are added "on the fly."

The last step of the parallel algorithm involves calculating the force on each particle, and this is straightforward. Indeed, by definition as each node is initialized with its locally essential data, it can proceed with the sequential algorithm without further node-to-node communication.

Summary

What did we see above that could have general significance?

We have an irregular loosely synchronous problem implemented as a complex communication step at the loose synchronization "barrier" defining the locally essential data followed by independent calculations in each node. Currently, we only have a good MIMD implementation and it is hard, but probably not impossible, to get a good SIMD version. There is a SIMD implementation of the easier case with a homogeneous particle distribution [Zhao:87a], [Zhao:89a].

We find a dynamic tree (or hierarchical) data structure which can also be seen in other applications such as quicksort, adaptive meshes, high-level image processing, and divide and conquer approaches to many problems. In the next section, we will discuss possible software lessons.

6 Fortran D Suspport for Unstructured Scientific Computations

In Table 2, we summarize how increasingly complex problem architectures require extensions to Fortran. As described in Section 4, the synchronous (Class I) and embarrassingly parallel (Class III-EP) applications can be handled by adding

a) decomposition directives defining data parallel arrays, their alignment with respect to each other, and separately their dynamic or static distribution over a particular machine;
b) *Forall* commands to express the asynchronous but uncoupled Class III-EP applications.

We also mentioned in Section 4 how heterogeneous problems (Class IIICG – IIFG) can be expressed in a correspondingly heterogeneous software model involving an object oriented approach with Fortran D "objects." Here we concentrate on extensions

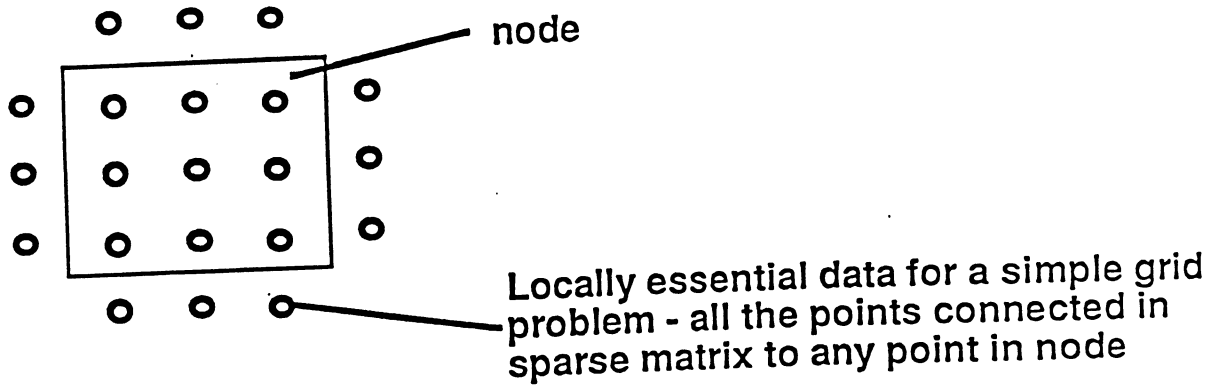

Figure 13. Locally essential data needed for solution of simple partial differential equations

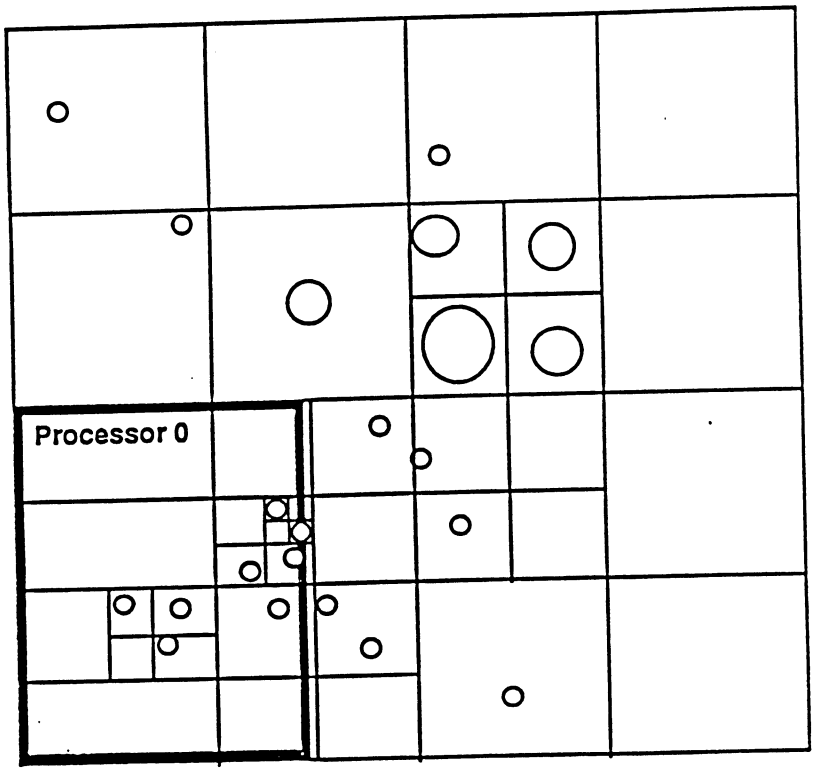

Figure 14. Locally essential data in processor 0 after communication cycle in parallel particle clustering algorithm

to Fortran D to allow it to express the unstructured scientific problems reviewed in Section 5. The success of irregular finite element codes on the Connection Machine [Johnsson:89b], [Johnsson:90a] illustrates how some sparse problems can be handled in Fortran 90 by using arrays of pointers. As an example, if we have an irregular shape but still four neighbors to each grid point, then four pointer arrays would just define the four neighbors. In Fortran D, we need to use the techniques of PARTI to handle these problems in general [Berryman:91a], [Saltz:87a], [Saltz:90b], [Saltz:91b]. These extensions of Fortran D allow the user to define runtime decompositions to optimize the distribution of the sparse matrix over the parallel machine. These are important, but not major changes, and we can say that Fortran D is adequate as the data structures are arrays of either values or pointers.

In Section 5.2, we described an application where we expect that the major extension needed in Fortran D would be new runtime support of generalizations of the Connection Machine's *scan* operation. In Section 5.3, we found a new challenge as the data structure is a dynamic tree which is expressed by the current code as linked lists in C. The user knows about the underlying physical structure shown in Figures 10, 12 and 14, and was able to parallelize the problem! However, the compiler (even with runtime support) would find the physical structure impossible to reconstruct from the hierarchy of pointers used in a linked list. Here, we need to extend Fortran D with new data structures, such as a tree [Mou:90a]. The astrophysics application of Section 5.3 also needs primitives to initialize, build, merge, and prune trees. We are currently investigating this and other applications with a hierarchical structure to see what general high level support is needed.

Acknowledgements

I would like to thank Alok Choudhary, Ken Kennedy, Sanjay Ranka, Joel Saltz, and John Salmon for many interesting discussions. This work was supported by the ASAS Program Office Techbase Program, and the National Science Foundation under Cooperative Agreement No. CCR-8809165—the Government has certain rights in this material.

Table 2.
Fortran 90D for Synchronous (SIMD) and Loosely Synchronous (MIMD)
Data Parallel Programming (about 90% of Scientific and Engineering Computations)

Program Class	Language and Environment Features
a) I – Regular Geometry e.g., full matrix e.g., finite difference e.g., Monte Carlo	"pure" Fortran 90 with arrays of values Need decomposition directives in Fortran D
b) I – Regular + III-EP e.g., chemical potential and dynamics problems Calculate matrix elements (needs *forall*); full matrix algebra (Class I) for energies and cross sections	Add *forall* to Fortran 90
c) I/II – Regular Topology but irregular geometry e.g., finite element	Add arrays of pointers to arrays of values. Need new run-time library as in PARTI.
d) "True" Loosely Synchronous (II) Irregular Problems e.g., High-level image processing e.g., Multiscale simulations Problem architectures are more general than that of array.	New data structures in Fortran 90D
e) IIICG –I, IIFG Complex System Simulations (See Section 2 and Figure 1)	Fortran 90D modules controlled by object oriented systems.

[Chen:88b] Chen, M., Li, J., and Choo, Y. "Compiling parallel programs by optimizing performance," *Journal of Supercomputing*, 2:171–207, 1988.

[Denning:90a] Denning, P. J., and Tichy, W. F. "Highly parallel computation," *Science*, 250:1217–1222, 1990.

[Ding:90m] Ding, H., 1990. Private Communication.

[Felten:88i] Felten, E. W., and Otto, S. W. "A highly parallel chess program," in *Proceedings of International Conference on Fifth Generation Computer Systems 1988*, pages 1001–1009. ICOT, November 1988. Tokyo, Japan, November 28 – December 2. Caltech Report C3P-579c.

[Foster:90a] Foster, I., and Taylor, S. *Strand: New concepts in Parallel Programming.* Prentice Hall, Englewood Clifs, New Jersey 07632, 1990.

[Fox:87d] Fox, G. C. "Questions and unexpected answers in concurrent computation," in J. J. Dongarra, editor, *Experimental Parallel Computing Architectures*, pages 97–121. Elsevier Science Publishers B.V., North-Holland, 1987. Caltech Report C3P-288.

[Fox:88a] Fox, G. C., Johnson, M. A., Lyzenga, G. A., Otto, S. W., Salmon, J. K., and Walker, D. W. *Solving Problems on Concurrent Processors*, volume 1. Prentice-Hall, Inc., Englewood Cliffs, NJ 07632, 1988.

[Fox:88b] Fox, G. C. "What have we learnt from using real parallel machines to solve real problems?," in G. C. Fox, editor, *The Third Conference on Hypercube Concurrent Computers and Applications, Volume 2*, pages 897–955. ACM Press, 11 West 42nd Street, New York, NY 10036, January 1988. Caltech Report C3P-522.

[Fox:88g] Fox, G. C., and Furmanski, W. "Hypercube algorithms for neural network simulation the Crystal_Accumulator and the Crystal_Router," in G. C. Fox, editor, *The Third Conference on Hypercube Concurrent Computers and Applications, Volume 1*, pages 714–724. ACM Press, 11 West 42nd Street, New York, NY 10036, January 1988. Caltech Report C3P-405b.

[Fox:88mm] Fox, G. C. "A review of automatic load balancing and decomposition methods for the hypercube," in M. Schultz, editor, *Numerical Algorithms for Modern Parallel Computer Architectures*, pages 63–76. Springer-Verlag, 1988. Caltech Report C3P-385.

[Fox:88oo] Fox, G. C. "The hypercube and the Caltech Concurrent Computation Program: A microcosm of parallel computing," in B. J. Alder, editor, *Special Purpose Computers*, pages 1–40. Academic Press, Inc., 1988. Caltech Report C3P-422.

References

[Angus:90a] Angus, I. G., Fox, G. C., Kim, J. S., and Walker, D. W. *Solving Problems on Concurrent Processors: Software for Concurrent Processors*, volume 2. Prentice-Hall, Inc., Englewood Cliffs, NJ 07632, 1990.

[Appel:85a] Appel, A. W. "An efficient program for many-body simulation," *Sci. Stat. Comput.*, 6:85, 1985.

[Apostolakis:91b] Apostolakis, J., Coddington, P., and Marinari, E. "A multigrid cluster labeling scheme." Technical Report SCCS-103, Syracuse University, June 1991.

[Baillie:89e] Baillie, C. F., Brickner, R. G., Gupta, R., and Johnsson, L. "QCD with dynamical Fermions on the Connection Machine," in *Proceedings of Supercomputing '89*, pages 2–9. ACM Press, November 1989. IEEE Computer Society and ACM SIGARCH, Reno, Nevada. Caltech Report C3P-786.

[Baillie:90f] Baillie, C. F. "Lattice QCD: Commercial vs. home-grown parallel computers," in D. W. Walker and Q. F. Stout, editors, *The Fifth Distributed Memory Computing Conference, Volume I*, pages 397–405, 10662 Los Vaqueros Circle, P. O. Box 3014, Los Alamitos, California 90720-1264, 1990. IEEE Computer Society Press. Held April 9–12, Charleston, South Carolina. Caltech Report C3P-878.

[Baillie:90n] Baillie, C. F., Johnston, D. A., and Kilcup, G. W. "Status and prospects of the computational approach to high-energy physics," *The Journal of Supercomputing*, 4:277–300, 1990. Caltech Report C3P-800b.

[Baillie:91a] Baillie, C. F., and Coddington, P. D. "Cluster identification algorithms for spin models," *Concurrency: Practice and Experience*, 3(2):129–144, April 1991. Caltech Report C3P-855.

[Barnes:86a] Barnes, J., and Hut, P. "A hierarchical $O(N \log N)$ force calculation algorithm," *Nature*, 324:446, 1986.

[Brower:91a] Brower, R. C., Tamayo, P., and York, B. "Parallel multigrid algorithms for percolation clusters," *J. Stat. Phys.*, 63:73–88, 1991.

[Birman:91a] Birman, K., "Recent developments with ISIS," 1991. Presentation at DARPA Workshop, Providence, Rhode Island, February 28, 1991.

[Chandy:90a] Chandy, K., and Taylor, S. "A primer for program composition notation." Technical Report CRPC-TR90056, California Institute of Technology, June 1990.

[Gelernter:89a] Gelernter, D. "Multiple tuple spaces in Linda," in *Proceedings of Parallel Architectures and Languages Europes*, volume 2, page 366. Springer-Verlag, LNCS, June 1989.

[Greengard:86a] Greengard, L., and Rokhlin, V. "A fast algorithm for particle simulation." Technical Report YALEU/DCS/RR-459, Yale University, 1986.

[Greengard:88a] Greengard, L. "The rapid evaluation of potential fields in particle systems," in *ACM Distinguished Dissertation Series, Vol. IV*. MIT Press, Cambridge, Mass., 1988. Yale research report YALEU/DCS/RR-533 (April 1987).

[Hillis:87a] Hillis, W. D. "The Connection Machine," *Scientific American*, page 108, June 1987.

[Johnsson:89b] Johnsson, S. L., and Mathur, K. "Data structures and algorithms for the finite element method on a data parallel supercomputer." Technical Report CS89-1, Thinking Machines Corporation, 1989.

[Johnsson:90a] Johnsson, S. L., and Mathur, K. K. "Experience with the conjugate gradient method for stress analysis on a data parallel supercomputer." Technical report, Thinking Machines Corporation, 1990.

[Keppenne:90a] L., K. G., Ghil, M., Fox, G. C., Flower, J. W., Kolawa, A., Papaccio, P. N., Rosati, J. J., Shepanski, J. F., Spadaro, F. G., and Dickey, J. O. "Parallel processing applied to climate modeling." Technical Report SCCS-22, Syracuse University, November 1990.

[Meier:90a] Meier, D. L., Cloud, K. L., Horvath, J. C., Allan, L. D., Hammond, W. H., and Maxfield, H. A. "A general framework for complex time-driven simulations on hypercubes," in D. W. Walker and Q. F. Stout, editors, *The Fifth Distributed Memory Computing Conference, Volume I*, pages 117–121, 10662 Los Vaqueros Circle, P. O. Box 3014, Los Alamitos, California 90720-1264, 1990. IEEE Computer Society Press. Held April 9–12, Charleston, South Carolina. Caltech Report C3P-960.

[Mou:90a] Mou, Z. G. "Divacon: A parallel language for scientific computing based on divide and conquer," *Frontiers 90*, pages 451–461, IEEE Press, October 1990.

[Quinn:90a] Quinn, M. J., and Hatcher, P. J. "Data-parallel programming on multicomputers," *IEEE Software*, pages 69–76, September 1990.

[Quinn:90b] Quinn, M. J. "Compiling SIMD programs for MIMD architectures," in *Proceedings of the IEEE Computer Society 1990 International Conference on Computer Languages*, March 1990.

[Fox:88tt] Fox, G. C., and Furmanski, W. "The physical structure of concurrent problems and concurrent computers," *Phil. Trans. R. Soc. Lond. A*, 326:411–444, 1988. Caltech Report C3P-493.

[Fox:89i] Fox, G. C. "1989 — the first year of the parallel supercomputer." Technical Report CRPC-TR890010, California Institute of Technology, March 1989. Paper presented at the Fourth Conference on Hypercubes, Concurrent Computers and Applications; CCR-8809615.

[Fox:89n] Fox, G. C. "Parallel computing comes of age: Supercomputer level parallel computations at Caltech," *Concurrency: Practice and Experience*, 1(1):63–103, September 1989. Caltech Report C3P-795.

[Fox:89t] Fox, G. C., Hipes, P., and Salmon, J. "Practical parallel supercomputing: Examples from chemistry and physics," in *Proceedings of Supercomputing '89*, pages 58–70. ACM Press, November 1989. IEEE Computer Society and ACM SIGARCH, Reno, Nevada. Caltech Report C3P-818.

[Fox:90h] Fox, G. C. "Achievements and problems for parallel computing." Technical Report SCCS-29, California Institute of Technology, June 1990. Proceedings of the International Conference on Parallel Computing: Achievements, Problems and Prospects; held in Anacapri, Italy, June 3–9, 1990; to be published in *Concurrency: Practice and Experience*; CRPC-TR90083.

[Fox:90o] Fox, G. C. "Applications of parallel supercomputers: Scientific results and computer science lessons," in M. A. Arbib and J. A. Robinson, editors, *Natural and Artificial Parallel Computation*, chapter 4, pages 47–90. MIT Press, Cambridge, Massachusetts, 1990. SCCS-23. Caltech Report C3P-806b.

[Fox:91c] Fox, G. C. "Parallel problem architectures and their implications for portable parallel software systems." Technical Report C3P-967, Northeast Parallel Architectures Center, May 1991. CRPC-TR91120, SCCS-78, Presentation at DARPA Workshop, Providence, Rhode Island, February 28, 1991.

[Fox:91d] Fox, G. C. "FortranD as a portable software system for parallel computers." Technical Report SCCS-91, Syracuse University, June 1991. Published in the Proceedings of Supercomputing USA/Pacific 91, held in Santa Clara, California. CRPC–TR91128.

[Fox:91e] Fox, G. C., Hiranandani, S., Kennedy, K., Koelbel, C., Kremer, U., Tseng, C.-W., and Wu, M.-Y. "Fortran D language specification." Technical Report SCCS-42c, Rice Center for Research in Parallel Computation; CRPC-TR90079, April 1991.

Jefferson, D. "The performance of a distributed combat simulation with the time warp operating system," *Concurrency: Practice and Experience*, 1(1):35–50, 1989. Caltech Report C3P-798.

[Williams:88a] Williams, R. D. "DIME: A programming environment for unstructured triangular meshes on a distributed-memory parallel processor," in G. C. Fox, editor, *The Third Conference on Hypercube Concurrent Computers and Applications, Volume 2*, pages 1770–1787. ACM Press, 11 West 42nd Street, New York, NY 10036, January 1988. Caltech Report C3P-502.

[Williams:89b] Williams, R. D. "Supersonic flow in parallel with an unstructured mesh," *Concurrency: Practice and Experience*, 1(1):51–62, 1989. (See manual for this code in Caltech Report, C^3P-861 (1990)). Caltech Report C3P-636b.

[Williams:90b] Williams, R. D. "DIME: Distributed Irregular Mesh Environment." Technical Report C3P-861, California Institute of Technology, February 1990. Users Manual.

[Wolff:90b] Wolff, U. "Critical slowing down," *Nucl. Phys. B (Prop. Suppl.)*, 17:565–579, 1990.

[Wu:91a] Wu, M., and Fox, Geoffrey, C. "Compiling Fortran 90 programs for distributed memory MIMD parallel computers." Technical Report SCCS-88, Syracuse Center for Computational Science, April 1991. CRPC-TR91126.

[Zhao:87a] Zhao, F. *An $O(N)$ Algorithm for Three-dimensional N-body Simulations.* PhD thesis, Massachusetts Institute of Technology, 1987. October.

[Zhao:89a] Zhao, F., and Johnsson, S. L. "The parallel multipole method on the Connection Machine." Technical Report CS89-6, Thinking Machines Corporation, October 1989.

[Salmon:89b] Salmon, J., Quinn, P., and Warren, M. "Using parallel computers for very large N-body simulations: Shell formation using 180K particles," in A. Toomre and R. Wielen, editors, *Proceedings of the Heidelberg Conference on the Dynamics and Interactions of Galaxies*. Springer-Verlag, April 1989. Caltech Report C3P-780b.

[Salmon:90a] Salmon, J. *Parallel Hierarchical N-Body Methods*. PhD thesis, California Institute of Technology, December 1990. Caltech Report C3P-966.

[Saltz:87a] Saltz, J., Mirchandaney, R., Smith, R., Nicol, D., and Crowley, K. "The PARTY parallel runtime system," in *Proceedings of the SIAM Conference on Parallel Processing for Scientific Computing*. Society for Industrial and Applied Mathematics, 1987. held in Los Angeles, CA.

[Saltz:90b] Saltz, J., Crowley, K., Mirchandaney, R., and Berryman, H. "Run-time scheduling and execution of loops on message passing machines," *Journal of Parallel and Distributed Computing*, 8:303–312, 1990.

[Saltz:91b] Saltz, J., Berryman, H., and Wu, J. "Multiprocessor and runtime compilation," *Concurrency: Practice and Experience*, 3(5), 1991. Special Issue from International Conference on Parallel Computing, held in Anacapri, Italy June 3–9, 1990.

[Sokal:88a] Sokal, A. D. *Computer Simulation Studies in Condensed Matter Physics: Recent Developments*. Springer-Verlag, Berlin-Heidelberg, 1988. editors, D. P. Landau *et al.*

[Swendsen:87a] Swendsen, R. H., and Wang, J. "Nonuniversal critical dynamics in Monte Carlo simulations," *Phys. Rev. Lett.*, 58(2):86–88, 1987.

[Wang:90a] Wang, J., and Swendsen, R. H. "Cluster Monte Carlo algorithms," *Physica A*, 167:565–579, 1990.

[Warren:88c] Warren, M. "An $O(N \log N)$ hypercube N-body integrator." Technical Report C3P-639, California Institute of Technology, May 1988. Caltech Undergraduate Senior Thesis. WARNING: Internal

[Warren:91a] Warren, M. S., Zurek, W. H., Quinn, P. J., and Salmon, J. K. "The shape of the invisible halo: N-Body simulations on parallel supercomputers." Technical Report C3P-961, California Institute of Technology, 1991. submitted to Proceedings of After the First Three Minutes, ed. S. Holt, V. Trimble, and C. Bennetti, AIP, 1991; Los Alamos Technical Report LA-UR-90-3915.

[Wieland:89a] Wieland, F., Hawley, L., Feinberg, A., DiLoreto, M., Blume, L., Ruffles, J., Reiher, P., Beckman, B., Hontalas, P., Bellenot, S., and