Using Problem and Algorithm Topology for Parallelization

Lorie M. Liebrock
Darrell L. Hicks
Ken W. Kennedy
Jack J. Dongarra

CRPC-TR91166
July, 1991

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

Revised November, 1991.

Using Problem and Algorithm Topology for Parallelization

by

Lorie M. Liebrock

Department of Computer Science
Rice University
Houston, Texas 77251

Advisors: Dr. J.J. Dongarra & Dr. K.W. Kennedy

Abstract

Efficient parallelization for distributed memory parallel processors requires that some parallel decomposition of the program be found and that communication be encoded which satisfies all dependences in the decomposed program. With the use of the Rice Fortran D compiler, the parallelization problem is reduced to finding "proper" data structure partitions (in the form of decomposition, alignment and distribution statements). Our conjecture is that understanding problem topology is useful in optimizing parallel programs in computational physics and other computational sciences. In particular, the notion of topology can be used in the compilation process to assist with parallelization. This thesis will be explored by: developing a graphical tool, TopView, that uses problem topology to provide recommendations for the alignment and distribution of data structures; and extending Fortran D's capability to express and take advantage of topology for the important case of irregularly coupled regular meshes.

1. Introduction

Current automatic parallelizers have difficulty finding truly efficient global parallelism. They are unable to analyze programs well enough to produce global parallelism the way a computational scientist with extensive application and parallel programming experience does. For this reason, many computational scientists are currently must "hand parallelize" their applications programs for every parallel processor they wish to use. It has been observed that this process of "hand coding for each machine" is reminiscent of the early practice of writing machine or assembly code. We propose to study the use of problem topology as a means to assist in writing efficient, portable programs for parallel machines. We will focus on: 1) the development of a tool, called TopView, based on the use of problem topology for parallelization; and 2) extension of the notion of topology in Fortran D to support applications with irregularly coupled regular mesh topologies.

Although TopView may be most useful for distributed memory machines, it's use is not restricted to such machines. In our experience, programs developed using global parallelism tend to run faster than those using "doacross parallelism" on most parallel architectures [McGrath, et al., 1986]. TopView should also be useful in the context of the weakly coherent virtual shared memory machines currently under development, e.g., the EDS system in the ESPRIT project, the Willow project at Rice University, and other hierarchical memory machines or distributed shared memory machines. Researchers working on the Willow project say that two of the most important problems to solve to obtain good performance from their machines are data partitioning and process to processor mapping.

In a program with global parallelism, processes are created at the very beginning of execution and run in parallel (with synchronization and communication) until program termination. In global parallelism the default mode of execution is parallel. This mode may be compared with forms where the program runs sequentially except for the parallel execution of some loops. This approach is seen in most parallel languages based on the "doacross construct" where the default mode of execution is sequential. Basically, parallelism must be the rule not the exception for optimal performance.

Applications considered here are physical simulation applications such as material dynamics, continuum mechanics, quantum physics, etc.

2. Terminology

The following terminology applys within the scope of this document.

A *process* is a unit of computation - e.g., an independently executable Fortran subroutine together with the calling sequence parameters, common data and externals. A main program is also a process. A *processor* is a physical device capable of executing a process.

An *inter-process (cross-processor) data dependence* exists whenever one process, α, needs a data value, χ, from another process, β. On a shared memory multiprocessor, the data dependence is satisfied when β has written χ to a shared variable and α has read χ. On a distributed memory multiprocessor, the data dependence is satisfied when β has communicated χ to α, i.e., β has sent a message containing χ to α and α has received that message. A *global dependence* is one in which a single value is dependent on all of the values of an array.

Communication includes both synchronization operations and actual exchange of data. On a shared memory multiprocessor, communication involves reads and writes of shared data and/or synchronization operations. On a distributed memory multiprocessor, communication involves sends and receives for both data exchange and synchronization. On weakly coherent shared memory multiprocessors, communication is proposed to be reads and writes of shared data and the associated required synchronization operations.

The *topology* of a physical system is the essence of its connectedness. In problems such as the simulation of physical phenomenon, the connectedness of the physical system being simulated may be reflected in the algorithm topology, which may further be reflected in the program's data structures. Topology or *neighborliness* may be used to specify which computational processes will communicate (be neighbors) in a parallel algorithm. For example, the natural topology of an n-dimensional fluid flow problem for computational simulation purposes is an n-dimensional mesh for n = 1, 2, 3. For a discussion of problem topology (or geometry) and its significance in parallelization see Heerman and Burkitt, 1991. The primary application characteristic that we are restricting our consideration to are those applications based on mesh topologies. The most important research component of this work will entail extending the notion of topology in Fortran D to support problems with *irregularly coupled regular meshes* (ICRMs; [Crowley & Saltz, 1991]). These problems have a number of components with regular mesh topologies, but the components are connected in irregular ways. An important example in this problem class is the simulation of water cooled nuclear reactors. This simulation applies ICRMs as the reactor core may be simulated using a 3-dimensional mesh but the pipes are usually simulated using 1-dimensional meshes for efficiency. For this research we will be considering problems with regular mesh topologies and irregularly coupled regular meshes.

A *process dependence graph* is a graph in which the nodes represent processes and the edges represent inter-process data dependences. In some programs for simulation of physical systems, when the nodes in the process dependence graph are arranged according to the topology of the physical system, dependence edges exist only between neighbors in the graph. In this case, edges then represent communication lines when the processes are assigned to processors connected in the same topology as the processes. Thus use of topology can help to minimize communication overhead whether the topological information is used by a programmer or compiler.

The *computation communication graph* (CCG) is a form of dependence graph that shows how values are related in a program fragment. Each node in a CCG represents an array storage location. Each edge represents the dependence of the value at the sink on the value at the source of the edge. The CCG for an entire program is called the *global computation communication graph* (GCCG).

3. Related Work

Although the amount of related work is far too great to discuss all of it here, we will briefly discuss some of the major thrusts in parallelization research and their relationship to the proposed research. In each area only a few of the relevant works will be mentioned to give an idea of what is being done in the related areas of partitioning and parallelization.

Many researchers are working on partitioning loops for parallel execution [King, et al., 1989, Smith, et al., 1989, Smith, et al., 1990, D'Hollander 1989, Ikudome, et al., 1990], but this tends to lead to high communication/computation ratios and low parallel efficiency. A variation on this theme partitions data structures for parallelism [Ramanujam, et al., 1989], but requires constant and regular dependence distance vectors. This work is not general enough to efficiently handle implicit material dynamics codes which also

have non-constant dependence distance vectors. A somewhat more sophisticated approach is taken in Parafrase-2 [Polychronopoulos, et al., 1990] where initial tasks are outer loops, subroutine calls, and basic blocks. These initial tasks may then be merged to increase granularity. Although this may be more promising, the parallel code generated will still not, in general, be as efficient as that produced by an experienced parallel programmer. A great deal of work has been done on data dependence analysis for parallelization (e.g., see Li, et al., 1990 and Wolf, et al., 1987) which forms a basis for all of this work. All of this research is based solely on dependence analysis and not on the algorithm designer's understanding of the application. Although TopView is also heavily reliant on dependence analysis, it also makes use of the computational scientist's knowledge of the application. In particular, although theoretical algorithms have been developed [Li and Chen, 1990] for data structure alignment they have not been put into use. We intend to use previous research results [Li and Chen, 1990, Knobe, et al., 1990, Knobe and Natarajan, 1990, Chapman, et al., 1991] as a starting point, with the extra information we are obtaining from the user, for the development of TopView and the extension of Fortran D.

Some techniques have been presented for "optimally partitioning" problems for large granularity systems. [Agrawal, et al., 1988] performs data partitioning in the restricted setting where a master-slave model is used and 1) data can be arbitrarily divided, and 2) there is no need for communication between slave tasks. This approach is too costly in overhead for applications such as material dynamics where communication on each time step implies that the master reassigns tasks to each slave on every time step. That further implies reinitialization and completion of slaves on every timestep which is far too expensive for practical use. Another work on optimization of partitions [Ni, et al., 1989] requires that the user specify (among other things) a set of design parameters which includes the parameters to define the partition scheme. Of course, this means the algorithm designer must have already figured out generally how to partition the problem but just does not know the optimal values for the partition parameters. In general, for all of these approaches either the approach requires that the user already have the basic partitioning strategy in mind or the approach is too restrictive for the problems we are interested in supporting. TopView is proposed to assist with actually discovering a proper partition, not just optimizing one that is already known.

Some systems have been developed which focus on large granularity parallelism. Schedule [Dongarra, et al., 1987] was designed to assist users who wanted to develop functionally parallel programs. The user must perform functional partitioning and then is assisted in constructing and verifying the dependence relations for all of the subroutines. This approach does not work well for applications like material dynamics which have natural process-oriented or data parallelism but not much functional parallelism. Another system for developing explicitly parallel, large granularity programs is the FORCE [Jordan, et al., 1989]. The FORCE is basically a portable parallel language based on the parallel Fortran that was used on the Denelcor HEP. Although this system does provide a portable (across shared memory machines only) approach to parallel programming, it does not assist the user in developing the parallel program. In particular the user must not only partition the data structures but must also explicitly perform the synchronization.

The distributed memory compiler at Rice University [Hiranandani, et al., 1991] requires that the user provide a data structure partition. The compiler then completes the task of parallelizing the program. This saves the user a great deal of work including coding for only one process to perform input/output and explicit specification of communication. Unfortunately, this leaves a most crucial part of the parallelization task for the user to perform without assistance: data structure partitioning. Suprenum [Gerndt, et al., 1987] uses a similar approach. They state that the parallelization system relies heavily on the user to make the strategic decision in partitioning. Some new parallel languages such as DINO [Rosing, et al., 1990] also require that users specify the distribution of data for parallel execution on distributed memory machines. Another such system is BLAZE [Koelbel, et al., 1987] in which "the only extra information that the programmer must provide is a general statement of the data distribution pattern". TopView can assist the user in constructing the partitions (or distributions) for parallelization systems such as these.

Other proposed research approaches to assisting with data structure partitioning and distribution include the following. The use of static performance analysis is proposed in [Balasundaram, et al., 1990 and 1991] to guide data partitioning decisions. The notion of a training set of kernel computations is used to "train" the performance estimator. The performance estimator is then intended to predict the performance of code generated by the Fortran D compiler for a particular target machine. This is to allow the interactive

data partitioning tool to statically explore different partitioning schemes without actually running the code. This exploration may be an exhaustive search of "reasonable" distributions for each array. Currently, data partitioning and mapping is specified statically by the user in SUPERB, but a current research thrust is to use program analysis and heuristic rules with pattern matching to automate the process where possible [Chapman, et al., 1991]. [Gupta and Banerjee, 1991] present an approach to automatic data partitioning based on distribution constraints. In this approach, each loop is analyzed to determine constraints (restrictions) on distribution of data structures. Loop based constraints are then to be merged so that the overall execution time is minimized. These approaches are all in the category of research in progress and therefore have not been shown to be complete solutions to the problem of data structure partitioning. At best the approaches have been validated by hand-simulation on a few benchmarks. TopView seems a reasonable compromise between having the user do all of the work (as with current distributed memory compilers discussed above) and not allowing the user to take advantage of their knowledge of the problem (as with automatic partitioners).

4. Thesis and Research Problems

THESIS: An understanding of problem topology is useful in optimizing parallel programs (both by hand and with a compiler).

Currently, the notion of problem topology in Fortran D is weak and limited to simple meshes. In Fortran D, topology can be expressed as the number of dimensions in decompositions, but there is no guarantee that the neighborliness defined by the topology will be preserved. Therefore the notion of neighborliness must be strengthened both in terms of expressing more general problem topologies and ensuring expressed problem topologies are preserved whenever possible in mapping to any given hardware topology. Fortran D also requires the user to specify alignment and distribution of data structures.

This dissertation will investigate the hypothesis that specification of problem topology alone is sufficient to automatically generate alignments and distributions. A tool, TopView, is to be developed based on this hypothesis.

An important class of problems that are not currently supported by Fortran D is characterized by the need for irregularly coupled regular meshes. Irregularly coupled regular mesh decompositions are more general than simple regular meshes in that they allow the final decomposition in Fortran D to be composed by connecting regular meshes of varying sizes and dimensions. Irregularly coupled regular mesh decompositions are simpler than irregular decompositions thereby allowing more precise compiler analysis to provide better performance. The second hypothesis, that topology provides the key to extending Fortran D to support irregularly coupled regular meshes, will be investigated through an exploration of requisite compiler technology.

4.1. Alignment and Distribution Recommendation in Fortran D

The owner computes rule used in Fortran D says that the owner of a datum will compute and store it's values throughout the execution of the program. This generally means that all of the values referenced on the right hand side of an assignment must be located on the processor which stores the value for the left hand side. Thus the essence of data structure alignment is that variables occuring in the same statement should be allocated to the same processor. For alignment in Fortran D, all of the arrays are aligned at the level of the problem mapping [Fox, et al., 1990] which reflects the fine granularity computational nature of the program. A few of the complications arising in alignment are: for every array, it must be determined which dimensions are sequential; for every pair of arrays, A and B, it must be determined which parallel dimensions of A correspond to which parallel dimensions of B (these are aligned dimensions); for every pair of arrays in each of their aligned dimensions, it must be determined what offset is necessary. Once all data structures have been aligned, they must be distributed across the parallel machine. The primary

tradeoff in distribution is between load balancing and communication overhead. Correct distribution decisions require analysis to determine: which parallel dimensions of the aligned arrays should be distributed; which parallel dimensions, if any, should be sequentialized; and how the distributed dimensions should be mapped to the processors (i.e., block, cyclic, etc.).

We want to avoid requiring a parallel programmer to supply all of the decomposition, alignment, and distribution statements necessary for the Rice Fortran D compiler to generate a distributed memory program. We hypothesize that by strengthening the notion of neighborliness in Fortran D, decomposition statements (or some other form of topology specifications) will suffice to allow automatic recommendation for alignment, and distribution of data structures. To this end, a tool, TopView, will be developed which, given topology specifications for a program segment (a loop nest, subroutine or program), generates a set of recommended alignments and distributions. This will specifically allow the user to specify changes in topology, or independence of topology in different parts of the program at whatever level they deem appropriate. TopView will display the recommendations to the user, possibly in multiple formats such as the CCG or a process dependence graph. TopView will support graphical modification of the recommended alignments and partitions. When the user has completed modifications, TopView will generate Fortran D code which reflects the finalized alignment and distribution specifications for the program segment.

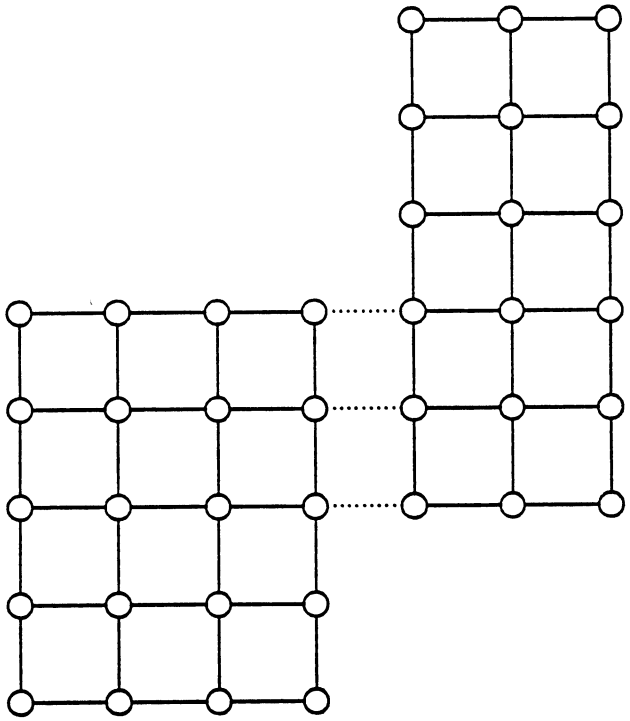

Figure 1: Simple irregularly coupled regular meshes.

To assist in choosing alignments we will begin by considering previous research [Li and Chen, 1990, and Knobe, et al., 1990]. The unifying idea in this previous research is to use affinities or preferences in alignment based on dependence analysis, where two different data elements prefer to be aligned if they appear in the same statement. We will develop compiler algorithms based on such previous research, modified to take advantage of our understanding of problem topology. Similarly, we will develop a decomposition strategy based on some combination of an exhaustive search of valid decompositions restricted to the problem topology and previous research [Knobe and Natarajan, 1990] modified to take advantage of compiler understanding of problem topology. Since we are only considering up to 3-dimensional regular mesh topologies an exhaustive search may be reasonable as a first cut algorithm.

For a brief discussion of the GCCG and the current prototype of TopView see Appendix 1.

Upon completion, TopView will provide a powerful tool to assist with parallelization of physical simulation applications.

4.2. Support for Irregularly Coupled Regular Meshes

There are critical applications which do not have simple mesh topologies as supported in Fortran D, e.g., simulation of water-cooled nuclear reactors and aerodynamics of airplanes. Many such critical applications can much more efficiently be simulated using irregularly coupled regular meshes. In simulation of water-cooled nuclear reactors, the scientist's primary interest is in simulating the state of the reactor core so the core may be modelled in three dimensions. It is inefficient to model the entire reactor system in three dimensions, especially as one 3-dimensional mesh. Therefore, 1-, 2-, and 3-dimensional meshes are used for the various components of a reactor and special boundary conditions are used at the interfaces between the components. Similarly, in simulation of the aerodynamics of airplanes, scientist's are interested in the air flow over the surface of the plane and sometimes model the various surface components as 2-dimensional meshes which are connected in irregular ways. These applications are critical in the sense that human lives may depend on the speed with which correct results of the simulations can be obtained. For example, a reactor operator may be able to avoid a meltdown of the reactor if a simulation can predict what action can correct the problem which is leading to meltdown. This can only occur with faster than real-time simulation capabilities. To support such applications, it is important to extend Fortran D to efficiently handle irregularly coupled regular meshes.

One way of extending Fortran D for irregularly coupled regular meshes is to add explicit coupling of decompositions with statements of the form:

$$\text{Couple}(A(d_{A_1}, l_{A_1} : u_{A_1}; \cdots; A(d_{A_n}, l_{A_n} : u_{A_n}), B(d_{B_1}, l_{B_1} : u_{B_1}; \cdots; B(d_{B_m}, l_{B_m} : u_{B_m}))$$

where d_i is a dimension place holder and l_i (u_i) is the lower (upper) bound for the coupling region in dimension d_i. Of course, $l_i : u_i$ may be written l_i if l_i and u_i are equal.

For example,

```
Decomposition(A(100, 50))
Decomposition(B(100, 100))
Couple(A(I, 3:100; J, 50), B(J, 1; I, 1:98))
```

joins decompositions A and B. The coupling created is A(3,50) to B(1,1), A(4,50) to B(1,2), ..., A(100,50) to B(1,98). We could in this way specify arbitrary connections. It is yet to be determined how general the supported connections should be and whether embeddings, for example, are necessary.

To illustrate some of the issues involved in supporting irregularly coupled regular meshes consider two 2-dimensional meshes connected as illustrated in Figure 1 (coupling is shown in dotted lines). These might, for example, be just two of the meshes used in simulating the flow of air over an airplane. The primary problem that we are attempting to solve is how to map this built up decomposition to an arbitrary parallel machine while respecting the problem topology. The Fortran D for this example, using the syntax described above, is:

```
Decomposition(A(5, 4))
Decomposition(B(6, 3))
Couple(A(I, 1:3; J, 4), B(I, 4:6; J, 1))
```

Even in this simple case where the problem topology, including the length of the connections between the meshes, is explicitly specified, the optimal mapping is dependent on many other variables including: the size of each dimension of each mesh; the relative amount of computation in the elements of the meshes; the relative amount of communication in the meshes in their respective topologies (both in the number of communications and their sizes); the hardware topology; the number of processors; the communication parameters of the machine (e.g., whether communication can be performed in parallel in all dimensions of the hardware topology, whether communication can be overlapped with computation, and what the relationship is between the startup cost and communication cost per byte); and whether the computation's dependences allow pipelining of communication. Even with all of the variables fixed in this simple example we must consider the tradeoff between load balancing and communication that can only be equalized with precise understanding of the entire computation.

One approach to this problem, assuming that the computation and communication performed for each element of both meshes takes approximately the same amount of time, is to use *folding* to minimize the 2-dimensional area while preserving the topology and then distribute the resulting single 2-dimensional mesh (possibly having holes). If we have a 2-dimensional (or higher) torus architecture then we can fold

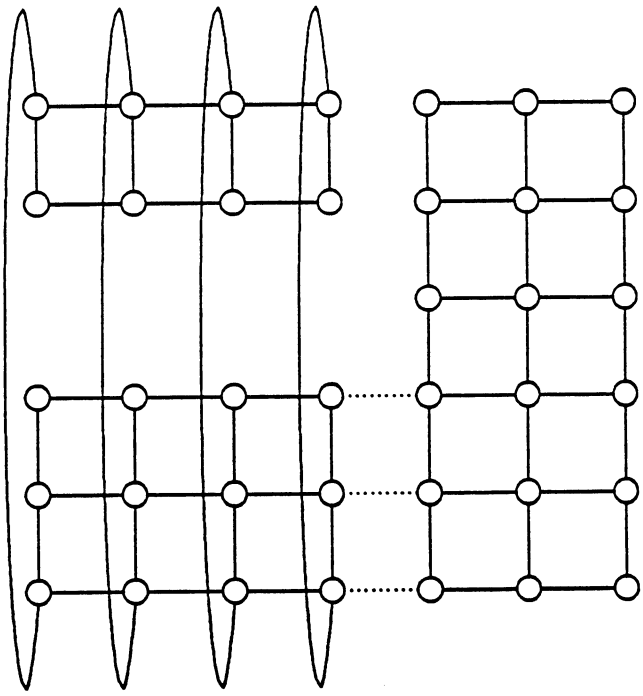

Figure 2: Folded irregularly coupled regular meshes.

the coupled meshes to reduce the area without increasing communication. This helps balance the load when the built up mesh is distributed. Folding results in the coupled (with a hole) mesh in Figure 2 which has a minimum achievable area in the plane while maintaining the neighborliness implied by the topology. Thus folding has reduced the area of the built up mesh from 7x8 to 7x6. This mesh would then need to be distributed; let's consider a 2x3 torus architecture. It is clear that even with this simple example we must carefully analyze the computation since the built up mesh is not full, the number of processors does not evenly divide the number of elements with work, and we must trade off communication against balancing computation times.

We will explore what can be done to assist with distribution in the more general case of this type of problem.

5. Research Plan

We first used topology specification explicitly in an experimental material dynamics code on a network of Sun workstations and a Sequent parallel processor. That use of topology allowed the user a more abstract means of specifying communication as it provided abstract names for neighbors in the topology rather than having the user specify the process/processor ids and the mapping of processes to processors.

TopView currently constructs and displays the GCCG for simple programs and allows the user to select distributions. This includes support for 1-dimensional and 2-dimensional regular meshes.

5.1. Alignment and Distribution Recommendation

Research will continue with algorithm development and analysis, design completion and further implementation of the TopView prototype. The prototype, given topology specifications, will recommend alignments and distributions automatically but allow the user to graphically modify these.

Algorithmic development and analysis will begin with aligned CCG construction. TopView will work at whatever level (loop, subroutine and program) topology is specified. Algorithmic development and analysis will continue with use of the CCG, topology, skeleton array and other user supplied information to partition the data structures semi-automatically.

TopView is being developed in the ParaScope programming environment. This allows us to take advantage of work that has already been done. Some of the advantages of using this environment are: 1) the abstract syntax tree is generated, which saves the work of lexical analysis, parsing, etc.; 2) and dependence analysis (intra-array only) is performed. Dependence analysis will have to be extended to represent all dependences, inter-array and intra-array, instead of just the intra-array dependences currently represented. Using ParaScope also simplifies the generation of Fortran D since statements can be inserted into the abstract syntax tree. For the Rice Fortran D compiler, also being developed in the ParaScope programming environment, the code will then be ready for compilation.

To test the performance of the recommendations we generate, we will use a suite of benchmarks being assembled by Geoffrey Fox. This suite provides us with versions of algorithms coded in: Fortran 77, message passing Fortran for the iPSC/860, CM Fortran for the Thinking Machines CM2, and Fortran D. To the Fortran 77 version we will add topology specification and compare the performance of the code generated by the Fortran D compiler using the automatically generated alignments and distributions with the parallel versions mentioned above. The most important and fairest comparison will be between the alignments and distributions that we generate and the optimal ones in the Fox Fortran D codes.

We will provide the prototype tool to users for experimentation. User comments and experiences will be recorded for evaluation. A variety of users, with different physical simulation applications, will be considered for evaluating the tool. The user group is intended to include representatives at different experience levels; from students having little application experience and no experience in parallel programming

to computational scientists who are experienced parallel programmers who have worked on their applications for many years.

Through interviews with computational scientists we will explore the types of problems that could be parallelized using TopView and applications having irregularly coupled regular meshes. We will also investigate how scientists think about these problems and what information is natural for them to provide the compiler. Interviews appear to be the best way to get insight into how computational scientists think [Norman, 1983]. Although the analysis of these interviews is subjective, the results will be summarized and quotations will be used to reflect the essence of the various views of computational scientists.

Interviews will also be used to evaluate TopView and determine how it could be improved in future development. These improvements may include changes to make TopView more easily used. User interfaces can best be improved by observing user interaction [Monk, 1984]. See Appendix 2 for an example of such suggestions. Summaries and specific comments will be included in the dissertation as appropriate.

5.2. Irregularly Coupled Regular Mesh Support

Most importantly, in parallel with TopView development we will extend Fortran D to support irregularly coupled regular meshes and explore the implications of these extensions to compiler technology. This will include:

(1) language extension to express irregular coupling between regular meshes in Fortran D;

(2) algorithm development and analysis for mapping irregularly coupled regular meshes to parallel architectures;

(3) code generation design;

(4) validation of the designed algorithms via simulation by hand.

We will also produce a rough prototype and experiment with its performance.

6. Summary

We will develop and analyze algorithms for generation, alignment, and partitioning of CCGs through the use of algorithm topology and other user information. TopView will be provided as an implementation of the algorithms to allow experimentation with the use of topology in semi-automatic parallelization.

We will extend the notion of topology in Fortran D to include irregularly coupled regular meshes in order to support such important applications as water-cooled nuclear reactor simulation. We will also develop and analyze algorithms to map such meshes to different architectures.

The by-products of this research will be: a tool to assist with parallelization in Fortran D; new compiler technology to expand the applicability of Fortran D to include applications with irregularly coupled regular mesh topologies; and greater understanding of what extensions to Fortran D are needed to support more applications conveniently.

The primary contributions of this research are: in combination with a Fortran D compiler, to provide a very powerful tool which supports semi-automatic parallelization for distributed memory machines of an important class of problems; and to develop the important research area in compilation technology of support for irregularly coupled regular meshes. In addition, the research illustrates how to take advantage of user knowledge, i.e., problem topology, to simplify the difficult task of automatic parallelization.

7. Appendix 1

Graphical representation of topology and program data structures is helpful in alignment and distribution of data structures for parallelization in Fortran D. When an experienced parallel programmer is parallelizing an algorithm, consideration of the topology of the problem can quickly lead to finding a very good approach to decomposing the problem for parallel execution. We have used this insight to develop a prototype tool that assists the user in partitioning the data structures in their sequential program via a graphical program representation. In the future the tool will recommend alignments and distributions but allow the user to override those recommendations where appropriate.

One representation of programs that may assist in the use of topology for parallelization is the computation communication graph (CCG). For example, for the program fragment in Figure 3 the CCG has nodes for each array element of all of the arrays and edges: from U(I) and U(I+1) to Q(I); from Q(I), Q(I+1), P(I), and P(I+1) to U(I+1); from U(I+1) to X(I+1); from U(I), U(I+1) and ZM(I) to V(I); from V(I) to P(I). The topology of the problem is a one-dimensional mesh. We use this example for the description of the TopView prototype.

For distributed memory machines, two of the most challenging problems faced by the programmer are: 1) determining an efficient problem partition; and 2) correctly incorporating communication and synchronization. Problem partitioning is often done in the form of data structure partitioning. Here a TopView prototype is presented that assists with parallelization in Fortran D. After data structure partitioning has been performed, using TopView, the Rice Fortran D compiler may be used to complete the parallelization task for distributed memory parallel processors.

Starting with a serial program, one way to parallelize is to build the GCCG to see relationships between variables, then partition the data structures and computations for assignment to parallel processes.

```
do 100 It = 1, NumStps
  do I = Istrt,Ifins
    Q(I) = - ( Clav*AI(I) + Cqav*ABS(U(I+1)-U(I))/V(I) )
1         * ( U(I+1) - U(I) )
  enddo
  do I = Istrt,Ifins-1
    U(I+1) = U(I+1) - Dthlf * ( P(I+1) + Q(I+1)
1           - P(I) - Q(I)) / ((ZM(I+1) + ZM(I)) * 0.5)
    X(I+1) = X(I+1) + Dt * U(I+1)
  enddo
  do I = Istrt,Ifins
    V(I)=V(I)+Dt*(U(I+1)-U(I))/ZM(I)
  enddo
  do I = Istrt,Ifins
    P(I) = P0 - Aisq0 * (V(I) - V0)
  enddo
  PHist(It) = P(K)
enddo
```

Figure 3: A Segment of a Simple Material Dynamics Code

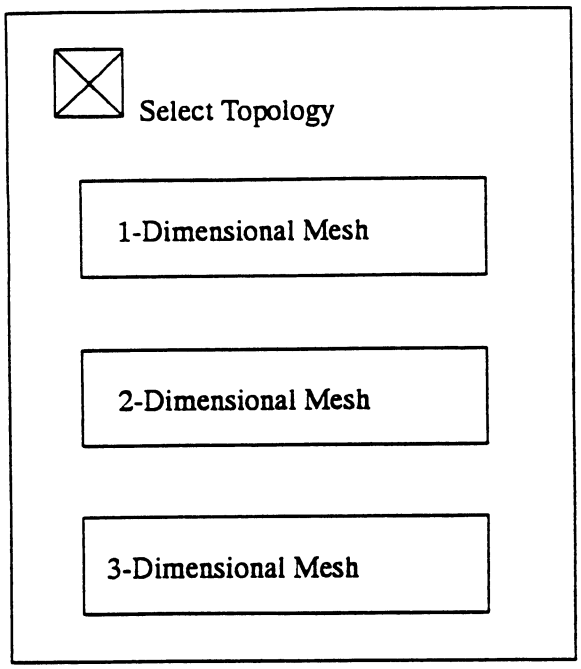

Figure 4: Topology Selection Menu

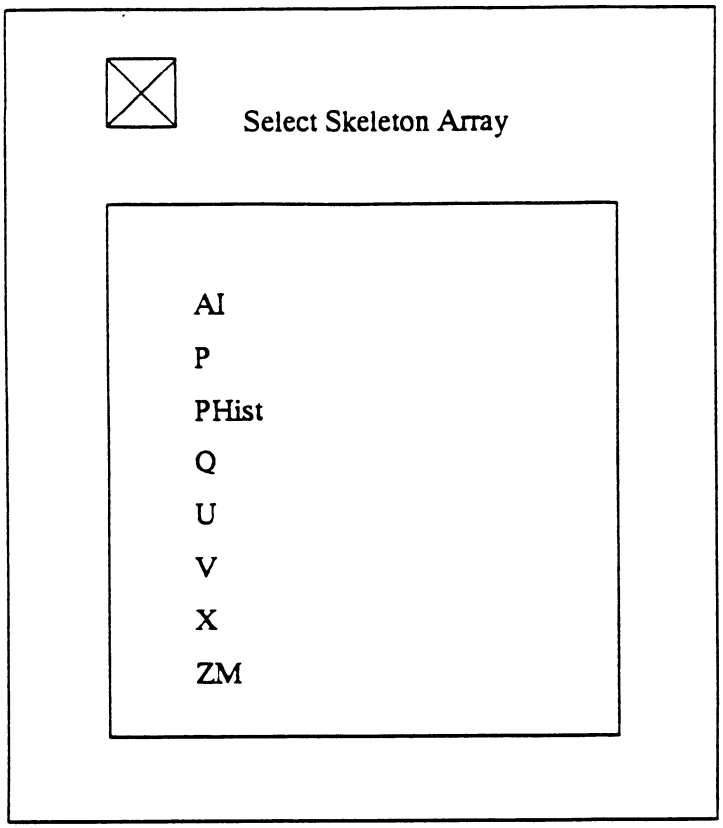

Figure 5: Skeleton Selection Menu

The GCCG may also be of use in debugging programs as it gives the user a different view of the variable interactions.

After the user selects the problem topology (see Figure 4), the user will be given a list of the arrays that conform to the topology (see Figure 5), and may therefore be used as a basis for displaying and partitioning the GCCG. Once the user has selected one of the arrays to be the skeleton array, the construction of the GCCG is performed automatically by TopView (see Figure 6).

Specification of the topology and the skeleton array provides a basis for laying out the GCCG for display to the user. The skeleton array is a structural foundation on which we build by attaching all neighboring nodes (array elements), laying them out according to their relationship to the skeleton and other arrays.

Currently, once a GCCG has been constructed and displayed, it remains for the user to partition the GCCG. In particular, the user must partition the graph into some number (the number is dependent on the topology) of subgraphs in which the amount of associated computation is approximately the same for each and there are a minimal number of edges cut in the partitioned graph. TopView supports user partitioning in an interactive, visual way, i.e., uses a mouse select edges to be severed. Future development of TopView will focus on recommending alignments and distributions to provide an initial (modifiable) partitioning. Even though the alignment problem is NP-Complete, a heuristic algorithm [Li and Chen 1990] is available

as a starting point for TopView development. Although the general problem of partitioning weighted graphs is NP-Complete [Gusfield & Irving 1989], given a graphical picture of the GCCG, not only is it often easy for the user to see what the correct partition strategy should be, but topology restricts reasonable distributions so that development is simplified for this problem also.

TopView helps a novice parallel programmer start the parallelization task with a major portion of the work done but later will allow them to improve on the performance when the alignment or partitions are suboptimal. This provides partitioning options for programmers who may not have enough understanding of the problem underlying their program.

8. Appendix 2

The use of interviews has already been fruitful as a potential user has suggested that a graphical representation of the supported topologies should be available. This seems very reasonable as it gives a visual clue to the actual topology that should be used for problem decomposition. One way of incorporating this comment is to add a graphical representation of the topology directly to the topology selection menu. This approach is illustrated in Figure 7. A possible problem with this approach is that for more complicated topologies it may be very difficult to fit a reasonable diagram in the menu. Therefore, we may instead use pop-up diagrams which will only be displayed when the user wants to see them.

9. Bibliography

R. Agrawal and H.V. Jagadish, "Partitioning Techniques for Large-Grained Parallelism" **IEEE Transactions on Computers**, V37, N12, December 1988, pp. 1627-1634.

V. Balasundaram, G. Fox, K. Kennedy and U. Kremer, "An Interactive Environment for Data Partitioning and Distribution", **Proceedings of the 5th Distributed Memory Computing Conference**, April, 1990.

V. Balasundaram, G. Fox, K. Kennedy and U. Kremer, "A Static Performance Estimator to Guide Data Partitioning Decisions", **Proceedings of the Third ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming**, April, 1991.

C.D. Callahan, K.D. Cooper, R.T. Hood, K. Kennedy and L. Torczon, "ParaScope: A Parallel Programming Environment", **The International Journal of Supercomputer Applications**, V2, N4, Winter 1988, pp. 84-99.

David Callahan & Ken Kennedy, "Compiling Programs for Distributed-Memory Multiprocessors", **The Journal of Supercomputing**, N2, 1989, pp. 151-169.

B. Chapman, H. Herbeck, and H. Zima, "Automatic Support for Data Distribution", **Proceedings of the 6th Distributed Memory Computing Conference**, April, 1991.

M. Chen, J. Li, and Y. Choo, "Compiling Parallel Programs by Optimizing Performance", **Journal of Supercomputing**, V2, 1988, pp. 171-207.

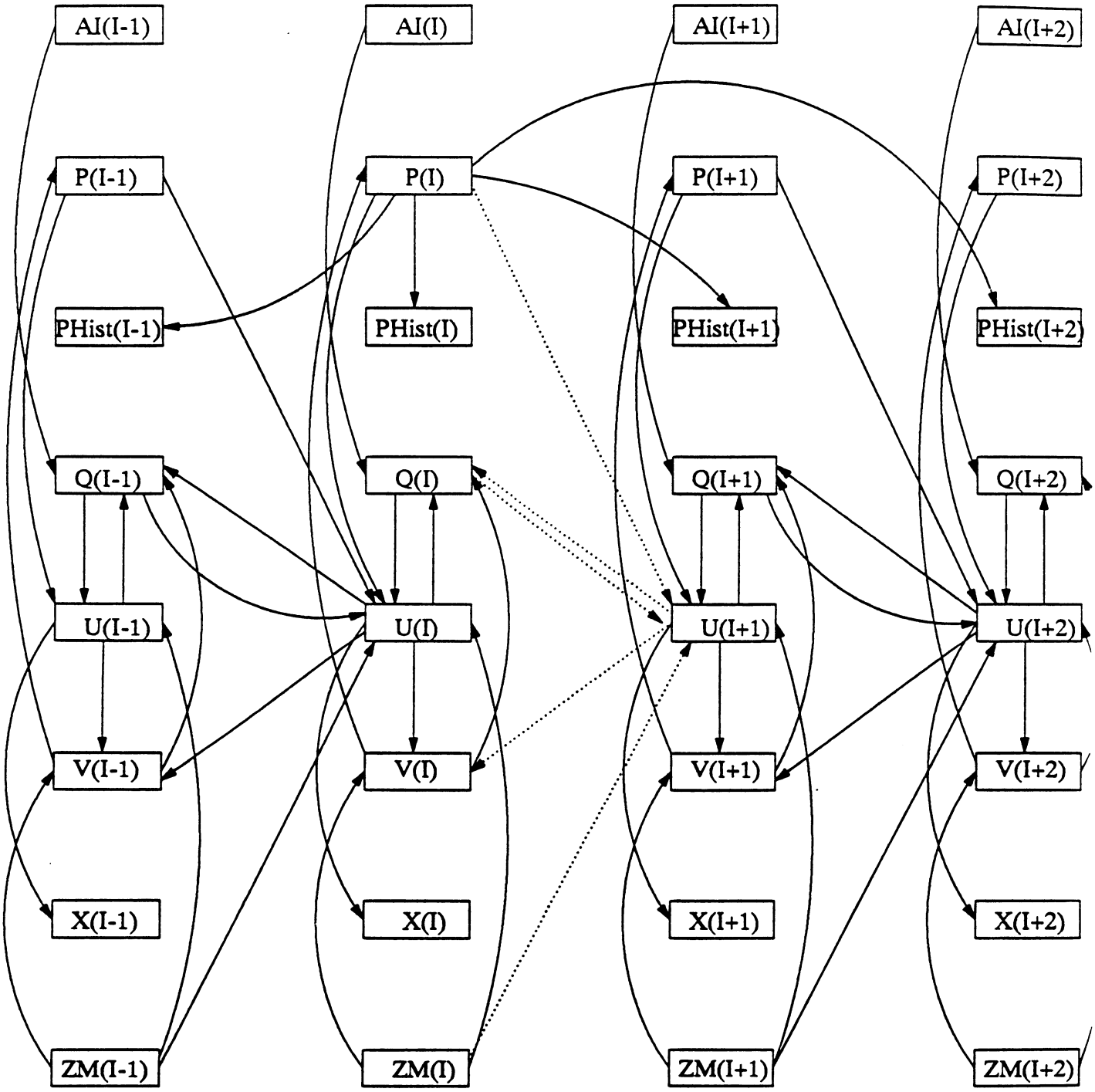

Figure 6: Partitioned Material Dynamics GCCG

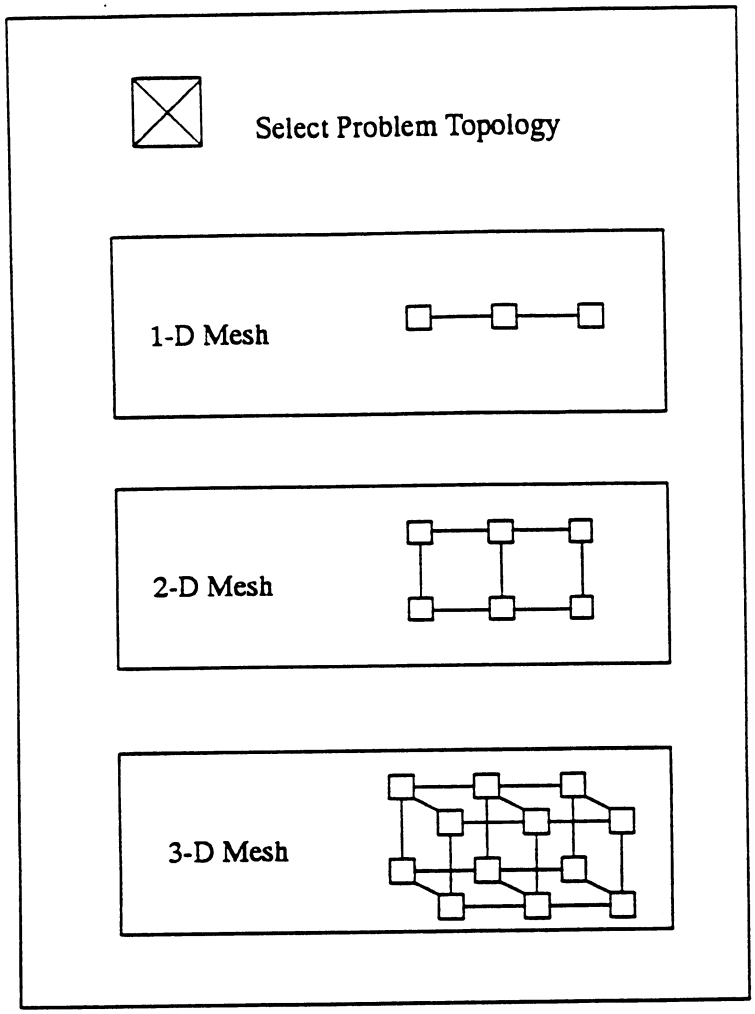

Figure 7: Topology Selection Menu with Graphical Topology Representation

K. Crowley, and J. Saltz, "A Manual for the Block Structured PARTI Runtime Primitives", Institute for Computer Applications in Science and Engineering, NASA Langley Research Center, Computer Science Department, Yale University, 1991.

E.H. D'Hollander, "Partitioning and Labeling of Index Sets in Do Loops with Constant Dependence Vectors", **1989 International Conference on Parallel Processing,** Vol. II, 1989, pp. 139-144.

J.J. Dongarra and D.C. Sorensen, "SCHEDULE: Tools for Developing and Analyzing Parallel Fortran Programs", **The Characteristics of Parallel Algorithms**, Eds. Leah H. Jamieson, Dennis B. Gannon, and Robert J. Douglass, MIT Press, Cambridge, Massachusetts, 1987, pp. 363-394.

G. Fox, S. Hiranandani, K. Kennedy, C. Koelbel, U. Kremer, C. Tseng, and M. Wu, "Fortran D Language Specification", Technical Report 90-141, Computer Science, Rice University, December, 1990.

M. Gerndt and H.P. Zima, "MIMD-Parallelization for Suprenum", **Supercomputing**, June 1987, Ed. E.N. Houstis, T.S. Papatheodorou, and C.D. Polychronopoulos, Springer-Verlag, pp. 278-293.

D. Gusfield and R. Irving, **The Stable Marriage Problem: Structure & Algorithms**, 1989, MIT Press, 240 pp.

V.A. Guarna, D. Gannon, P. Jablonski, A.D. Malony, and Y. Guar, "Faust: An Integrated Environment for Parallel Programming", **IEEE Software**, July 1989, pp. 20-26.

M. Gupta and P. Banerjee, "Automatic Data Partitioning on Distributed Memory Multiprocessors", **Proceedings of the 6th Distributed Memory Computing Conference**, April, 1991.

D.W. Heerman and A.N. Burkitt, **Parallel Algorithms in Computational Science**, Springer Verlag, Berlin, Heidelberg, 1991, 183 pp.

D.L. Hicks, "Hydrocodes on the HEP", **MIMD Computations: HEP Supercomputer and its Applications**, Ed. J.S. Kowalik, MIT Press, 1985, pp. 309-330.

D.L. Hicks, L.M. Liebrock and J.J. Dongarra, "ParaPack/PLAS", in preparation, 1991.

D.L. Hicks, L.M. Liebrock, V.A. Mousseau, and G.A. Mortensen, "Parallelization of Hydrocodes on the Intel Hypercube, Part 2", INEL Report EGG-SPAG-7818, 1987, 38 pp.

S. Hiranandani, K. Kennedy, C. Koelbel, U. Kremer and C. Tseng, "An Overview of the Fortran D Programming System", Technical Report 91-154, Computer Science, Rice University, March, 1991.

S. Hiranandani, K. Kennedy, and C. Tseng, "Compiler Support for Machine-Independent Parallel Programming in Fortran D", Technical Report 90-149, Computer Science, Rice University, January, 1991.

K. Ikudome, G.D. Fox, A. Kolawa, J.W. Flower, "An Automatic and Symbolic Parallelization System for Distributed Memory Parallel Computers", DMCCS, March 1990, 10 pp.

H. Jordan, M.S. Benten, G. Alaghband, and R. Jakob, "The Force: A Highly Parallel Programming Language", **1989 International Conference on Parallel Processing**, Vol. II, Eds. K.P. McAuliffe & P.M. Kogge, 1989, pp. 112-117.

C. King, and L.M. Ni, "Grouping in Nested Loops for Parallel Execution on Multicomputers", **1989 International Conference of Parallel Computing,** Vol. II, 1989, pp. 31-38.

K. Knobe, J.D. Lukas, and G.L. Steele, "Data Optimization: Allocation of Arrays to Reduce Communication on SIMD Machines", **Journal of Parallel and Distributed Computing,** 8, 1990, pp. 102-118.

K. Knobe, and V. Natarajan, "Data Optimization: Minimizing Residual Interprocessor Data Motion on SIMD Machines", **Frontiers90: The 3rd Symposium on the Frontiers of Massively Parallel Computation,** 8, 1990, pp. 416-423.

C. Koelbel, P. Mehrotra, and J. Van Rosendale, "Semi-automatic Process Partitioning for Parallel Computation", **International Journal of Parallel Computing,** V 16, 1987.

J. Li, and M. Chen, "Index Domain Alignment: Minimizing Cost of Cross-Referencing Between Distributed Arrays", **Frontiers of Massively Parallel Computing,** IEEE Computer Society Press, 1990, pp. 424-433.

Z. Li, P. Yew, and C. Zhu, "An Efficient Data Dependence Analysis for Parallelizing Compilers", **IEEE Transactions on Parallel and Distributed Systems,** V1, N1, 1990.

L.M. Liebrock, J.F. McGrath, and D.L. Hicks, "Experiment in Concurrent Computational Dynamics on the CRAY X-MP, Elxsi and HEP Parallel Processors with Recommendations for Parallel Architectures and Languages", KMS Fusion Report KMSF-U-1784, 1987, 103 pp.

A. Monk, "How and When to Collect Behavioral Data", **Fundamentals of Human-Computer Interaction,** Ed. A. Monk, Academic Press, 1984.

J.F. McGrath, D.L. Hicks and L.M. Liebrock, "Concurrent Algorithms for Computational Continuum Dynamics", **Applied Mathematics and Computation,** V20, N2, 1986, pp. 145-173.

L.M. Ni and C. King, "On Partitioning and Mapping for Hypercube Computing", **International Journal of Parallel Programming,** V17, N6, 1988, pp. 475-495.

D.A. Norman, "Some Observations on Mental Models", **Mental Models,** Eds. D. Gentner and A.L. Stevens, Lawrence Earlaum Associates, Inc., 1983.

David Notkin, et al., "Experiences with Poker", **SIGPLAN Notices,** V 23, N 9, September 1988, pp. 10-20.

C.D. Polychronopoulos, M. Girkar, M.R. Haghighat, C.L. Lee, B. Leung, and D. Schouten, "Parafrase-2: An Environment for Parallelizing, Synchronizing, and Scheduling Programs on Multiprocessors", in Languages and Compilers for Parallel Computing, Editors: D. Gelernter and A. Nicolau and D. Padua, MIT Press, 1990, 29 pp.

J. Ramanujam and P. Sadayappan, "A Methodology for Parallelizing Programs for Multicomputers and Complex Memory Multiprocessors", **Proceedings of Supercomputing '89,** ACM Press, 1989, pp. 637-646.

A. Rogers and K. Pingali, "Process Decomposition through Locality of Reference", **SIGPLAN 89 Conference on Programming Language Design and Implementation,** June 1989.

M. Rosing, R.B. Schnabel, and R.P. Weaver, "The DINO Parallel Programming Language", University of Colorado at Boulder, Department of Computer Science, Technical Report CU-CS-475-90, April 1990, 41 pp.

J.H. Saltz, V.K. Naik, and D.M. Nicol, "Reduction of the Effects of the Communication Delays in Scientific Algorithms on Message Passing MIMD Architectures." **SIAM Journal of Sci. and Stat. Computing,** V8(1), January 1987, pp. 118-134.

K. Smith, and B. Appelbe, "Loop Parallelization Using the Intervariable Dependence Graph", 1990, 22 pp.

K. Smith, B. Appelbe, and K. Stirewalt, "Incremental Dependence Analysis for Interactive Parallelization", **Supercomputing,** 1989, pp. 330-341.

S. Sobek, M. Azam, and J.C. Browne, "Architecture and Language Independent Parallel Programming: A Feasibility Demonstration", **Proceedings of the 1988 International Conference on Parallel Processing,** Vol. II Software, Ed. H.E. Sturgis, 1988, pp. 80-83.

R. Tamassia, G. Di Battista, and C. Batini, "Automatic Graph Drawing and Readability of Diagrams", **IEEE Transactions on Systems, Man, and Cybernetics,** V18, N1, 1988, pp. 61-79.

M. Wolfe, and U. Banerjee, "Data Dependence and Its Application to Parallel Processing", **International Journal of Parallel Programming,** V16, N2, 1987, pp. 137-178.

Hans P. Zima, Heinz J. Bast, and Michael Gerndt, "Superb: A Tool for Semi-automatic SIMD/MIMD Parallelization", **Parallel Computing,** V 6, 1988, pp. 1-18.