LEWIS

NASA Technical Memorandum 89823

The Hypercluster: A Parallel Processing Test-Bed Architecture for Computational Mechanics Applications

Richard A. Blech
Lewis Research Center
Cleveland, Ohio

Prepared for the
Summer Computer Simulation Conference
sponsored by the Society for Computer Simulation
Montreal, Canada, July 27–30, 1987

NASA

THE HYPERCLUSTER: A PARALLEL PROCESSING TEST-BED ARCHITECTURE FOR COMPUTATIONAL MECHANICS APPLICATIONS

Richard A. Blech
National Aeronautics and Space Administration
Lewis Research Center
Cleveland, Ohio 44135

E-3469

ABSTRACT

The development of numerical methods and software tools for parallel processors can be aided through the use of a hardware test-bed. The test-bed architecture must be flexible enough to support investigations into architecture-algorithm interactions. One way to implement a test-bed is to use a commercial parallel processor. Unfortunately, most commercial parallel processors are fixed in their interconnection and/or processor architecture. In this paper, we describe a modified n-cube architecture, called the hypercluster, which is a superset of many other processor and interconnection architectures. The hypercluster is intended to support research into parallel processing of computational fluid and structural mechanics problems which may require a number of different architectural configurations. An example of how a typical partial differential equation solution algorithm maps on to the hypercluster is given.

INTRODUCTION

Two research areas which are critical to the future progress of aerospace technology are computational fluid mechanics (CFM) and computational structural mechanics (CSM). The practical limits of applications in both of these areas are set by the state-of-the-art in computer architecture and software techniques. Parallel processing is an architectural concept which has the potential for vastly improving the performance of future computer systems. However, the use of parallel processing architectures will require a reassessment of numerical methods and software techniques that are currently used for CFM/CSM. Likewise, CFM/CSM requirements may impact future parallel architectures.

Most CFM/CSM problems require the numerical solution of a system of nonlinear partial differential equations (PDE). There are many algorithms for solving systems of PDE's on computers. The ideal algorithm for a given application minimizes computation time and the amount of memory required. A considerable amount of research has been done in this area for uniprocessor computers, resulting in many accepted approaches for solving various PDE systems. The continuing demand for more computing power and the emergence of supercomputer architectures employing parallel processors has prompted research into new approaches to solving systems of PDE's (Ortega and Voigt 1985). The goal of that research is the development of higher performance CFM/CSM codes that can effectively utilize the new parallel architectures.

The development of algorithms for parallel processors is not a straightforward task. Algorithms for parallel processors must be able to be partitioned into independent tasks that can be allocated to multiple processors for simultaneous execution. A high degree of parallelism does not guarantee higher performance, however. The development of parallel algorithms can be complicated by the hardware aspects of parallel processors. The communication mechanism between processors is one example. The algorithm should be analyzed to determine if fast, tightly coupled communication between processors is required, or if a slower, loosely coupled mechanism will suffice.

The individual processor architecture can also impact the performance of an algorithm. For example, a vector processor architecture operates most effectively by performing a single mathematical operation on large arrays of data. The performance of a vector processor is dependent on the length of the data arrays, or vectors. Therefore, it is desirable to develop algorithms which make use of long vector operations. If parallel vector processors are used, then any partitioning of the numerical method should avoid shortening the vector length to the point of degrading performance.

The memory hierarchy employed in a parallel processor is another consideration in the development of parallel algorithms. The use of local processor memory and/or global shared memory are examples of memory hierarchy within a parallel architecture. Cache memory, interleaved memory and mass storage are levels of the memory hierarchy local to the processors in a parallel processing system. An efficient parallel algorithm must make optimum use of the existing memory hierarchy. This requires maximizing the amount of computation occurring in the lowest (i.e., fastest) level of the hierarchy.

To summarize, the development of parallel algorithms requires cognizance of a large number of hardware and architectural parameters. This makes the evaluation of algorithm performance a critical step in the development process. To some extent, this can be done analytically. A detailed analytical performance evaluation would be cumbersome, however, especially if the number of hardware and architectural parameters is high. A preferable approach would be the evaluation using a hardware test-bed. Then hardware and architectural parameters could be directly implemented, or efficiently emulated.

A research effort at the NASA Lewis Research Center is devoted to studying the application of parallel processing to CFM/CSM. This effort is an outgrowth of work previously done on the Real-Time Multiprocessor Simulator (RTMPS) project (Arpasi 1985; Blech and Arpasi 1985; Cole 1985; Arpasi and Milner 1985). To facilitate the investigation of algorithm-architecture interactions and the evaluation of software tools, a reconfigurable hardware test-bed is

being assembled. This paper discusses the requirements driving the design of the parallel processing test-bed and describes the test-bed architecture being implemented at NASA Lewis. An example of how a typical PDE solution algorithm would map on to the architecture is presented.

Parallel Processing Test-Bed Requirements

In general, the purpose of a parallel processing test-bed is to support the development of parallel algorithms and the evaluation of software tools. Since many of the architectural requirements for a particular algorithm or software tool usually are not known, the test-bed must provide a degree of flexibility in configuration. This suggests some of the following desirable capabilities for any parallel processing test-bed.

(1) Ability to incorporate processors of various architectures within the parallel processing configuration. This allows evaluation of how the architecture and performance of the individual processing elements within a parallel system architecture can affect overall performance. Some processor architectural characteristics to be considered are vector processing capability, memory configuration (cache memory, interleaving), and specialized coprocessors (floating-point, graphics).

(2) Ability to emulate a wide variety of parallel processing architectures. The impact of interprocessor communication overhead is a critical issue in parallel processing research. The ability to vary the system architecture (and thereby the interprocessor communication paths) allows investigations into architecture-algorithm interactions.

(3) Ability to modify the I/O structure of the parallel processor. Input and/or output processing are the dominant time consumers for some applications. The ability to modify or augment the I/O structure allows researching of distributed database techniques and partitioning of the I/O task.

(4) Capability to expand to a large scale parallel system. This is necessary to evaluate algorithms requiring a large number of processors for effectiveness.

The usefulness of a parallel processing research test-bed having the above characteristics was recognized by researchers involved in IBM's Research Parallel Processor Project (RP3) (Pfister 1985). However, the RP3 architecture is neither commercially available nor easy to replicate. Commercial versions of some parallel processing architectures have recently become available. In most cases, the architecture is fixed and/or the user has limited capabilities for architectural or processor modifications. For example, Alliant's FX/8 machine (Alliant Computer Systems 1985) has multiple vector processors interconnected through shared memory. However, the current architecture is limited to eight processors. The BBN Butterfly (Crowther et al. 1985) has a large number of scalar processors communicating through shared memory, but lacks vector processing capability. Flexible Computers' FLEX/32 (Manuel 1985) combines both message passing and shared memory communication mechanisms, but again lacks vector processing capability.

The n-cube architecture, also known as the hypercube, is becoming a popular architecture due to its expandability and capability to emulate other architectures. A hypercube that has vector processing capability at each node (two commercial versions of which are discussed in Gustafson et al. 1986 and Robinson 1985) meets several of the requirements for a parallel processing test-bed. The strong points are: 1) an architecture which is expandable in a systematic manner, 2) vector and scalar processing capability at each node, and 3) the ability to emulate a limited number of other architectures. Emulation of shared memory architectures on the hypercube is difficult, however. This is especially true for applications exhibiting a fine-grained parallel structure, such as linear algebra. The difficulty can be traced to the interprocessor communications in the hypercube, which exhibit high overhead for two reasons. First, a routing algorithm is required for all but those applications which directly map on to the hypercube network. This consumes processor resources since the communication path from one processor to another must be calculated. Second, most commercial versions of the hypercube implement the network interconnections with fixed serial links. The net throughput rate on these links is relatively low when packetization and software protocol is taken into account. In addition, the link connections cannot be reconfigured.

Hypercluster Architecture

A modified version of the hypercube architecture, called the hypercluster, is proposed to overcome the difficulties described above. The hypercluster retains the hypercube network structure between processor nodes, but each node now consists of multiple processors communicating through a shared memory. This concept is illustrated in Figure 1, for a dimension 2 (D-2) cube. Each circle labelled 'M' represents a shared memory at a node. Each square labelled 'P' is a processing element interconnected to the shared memory in some fashion. Processors can have local memory in addition to shared memory. Communication links between nodes form the hypercluster network.

The hypercluster supports both tightly coupled interprocessor communication via shared memory (within a node) and loosely coupled communication through the hypercube network (between nodes). The hypercluster is expanded in the same manner as the hypercube, with processor clusters replacing the normal single processor node. An arbitrary number of processors may be assigned to a cluster, limited only by the hardware constraints of the shared memory interconnect and/or power requirements.

Figure 2 shows a more detailed diagram of the D-2 hypercluster configuration being implemented at the NASA Lewis Research Center. The nodes consist of multiple board-level computers interconnected by a commercial bus. Although a bus is not the highest performance shared-memory interconnect mechanism available, it does allow for convenient implementation. In addition, the use of a commercial bus allows a variety of processor architectures to be incorporated within a node. Thus each node has an architectural 'personality' determined by the type of processor boards connected to the bus. The NASA Lewis D-2 hypercluster has three nodes with a vector personality and one node with a scalar personality. Each of the vector nodes uses four board-level vector processors, while the scalar node uses four general purpose microcomputer boards. The incorporation of

vector processors is crucial in the investigation of CFM/CSM algorithms because many CFM/CSM algorithms contain large arrays of independent computations that are best handled by a vector architecture. The availability of multiple vector processors allows very large arrays of calculations to be broken up and distributed for a parallel solution.

There are two types of communication links. Internode communication links form the hypercluster network as described before. Additional links provide communication paths between each node and a front-end processor (FEP). The FEP allows a user to interact with the hypercluster. Each communication link consists of two control processors (CP) interconnected by a dual-port memory. The CP's coordinate communication over the links and supervise the operation of processors within a node. Executive software in each CP performs these functions. For the D-2 hypercluster, it is both practical and advantageous to have a communication link between each node and the FEP. However, as the hypercluster is expanded to more nodes, the associated size and cost constraints make this approach impractical. In that case, most nodes will not have an FEP link. Software will then be necessary to route information from nodes with an FEP link to those without.

Shared memory within a node consists of memory boards connected to the node's bus, and/or dual-ported memory on the processor boards. Dual-ported memory has become a standard feature on many commercial computer boards, and is particularly useful in the hypercluster environment. Through software, memory segments can be allocated as local to a processor, global to all processors in a node, or a combination of local and global segments. This allows emulation of the different memory hierarchies used in parallel processing systems.

Each node of the hypercluster can have its own local I/O capability. For example, each node can have a disk control processor and hard disk drive. This arrangement would allow research in distributed I/O and database techniques, aimed at eliminating the I/O bottleneck present in many applications.

An Algorithm Example

The alternating direction implicit (ADI) algorithm is a technique commonly used for the solution of partial differential equations (PDE) (Gerald 1980). The two stages of the ADI algorithm are shown in Figure 3 for a 4 by 4 grid.

In the first stage, equations are formed which are implicit (i.e., depend on current time step information) in the X direction only. Thus a coefficient matrix A and vector b can be generated to form the system $Ax = b$ which describes one row of points. Several such systems are formed to describe the entire grid. The matrix A is a tridiagonal matrix (the matrix is block tridiagonal if several PDE's are solved at each grid point). The second stage of the ADI algorithm begins after the equations from the first stage are solved. It is identical to the first stage except that now the equations are implicit in the Y direction only.

Each system of equations for a row or column is independent (in the current time step) of information from neighboring rows or columns. Only information from past time steps for neighboring rows or columns is used. This characteristic of the ADI algorithm makes it particularly attractive for solution on a parallel processor. The solution of rows or columns can be done in parallel. Each row or column can be allocated to a processor, if sufficient processors are available. Otherwise, groups of rows or columns must be formed, where the number of groups would equal the number of available processors.

The first stage of the ADI algorithm would map onto the hypercluster as shown in Figure 4. For the simple 4 by 4 grid and D-2 hypercluster shown, each row would be solved on a hypercluster node. If the grid were larger, groups of rows would be assigned to the node, or more nodes could be added. The processor allocation described thus far could be accomplished on any hypercube implementation. The advantage of the hypercluster architecture for the ADI algorithm is the ability to apply the tightly coupled multiple processors within each node to the simultaneous solution of the equation systems. The allocation of rows to hypercluster nodes results in one or more block tridiagonal equation systems which must be solved at each node. After the parallel solution of the equation sets is completed, information to and from neighboring rows is transmitted between nodes via the hypercube network. Then the second stage of the ADI algorithm can proceed with columns allocated to hypercluster nodes. After solution of both stages, the results are checked for convergence. If convergence has not been achieved, the whole process is repeated. The parallel ADI algorithm is outlined by the psuedocode in Figure 5.

The ADI algorithm is only one of many algorithms available to solve a system of partial differential equations. The partitioning of the calculations for the ADI algorithm described above is one of many possible methods. This example has been given only to demonstrate the usefulness of the hypercluster architecture for implementing a particular algorithm.

A number of parallel processing algorithms for solving partial differential equations have been proposed in Hockney and Jesshope 1981. Some of these algorithms are also vectorizable. Future work using the hypercluster as a test-bed will attempt to determine which algorithms (combined with the appropriate architecture) are optimum for CFM/CSM applications.

CONCLUDING REMARKS

The hypercluster architecture is intended to provide a reconfigurable test-bed on which various parallel processing algorithms, programming and operating tools can be developed. There is still a considerable amount of uncertainty as to the optimum parallel processing architecture for specific applications such as CFM and CSM. There is also a definite lack of programming and operating software that will allow researchers to easily take advantage of parallel processing. Future work using the hypercluster test-bed will attempt to address some of these issues. It will allow CFM/CSM research at the NASA Lewis Research Center to readily adapt to the rapidly developing discipline of parallel processing.

REFERENCES

1. Ortega, J. M. and Voigt, R G., "Solution of Partial Differential Equations on Parallel and Vector Computers," SIAM Review, Vol. 27, No. 2, June 1985.

2. Arpasi, D. J. "Real-Time Multiprocessor Programming Language (RTMPL) Users Manual," NASA Technical Paper 2422, June 1985.

3. Blech, R. A. and Arpasi, D. J., " Hardware for a Real-Time Multiprocessor Simulator, " NASA TM 83805, January 1985.

4. Cole, G. L., "Operating System for a Real-Time Multiprocessor Propulsion System Simulator," NASA Technical Paper 2426, January 1985.

5. Arpasi, D. J. and Milner, E. J., "Partitioning and Packing Mathematical Simulation Models for Calculation on Parallel Computers," NASA TM 87170, November, 1985.

6. Pfister, G. F., et. al., "The IBM Research Parallel Processor Prototype (RP3): Introduction and Architecture," Proceedings of the 1985 International Conference on Parallel Processing, August 1985.

7. Alliant Computer Systems, "FX/Series Product Summary," June 1985.

8. Crowther, W. et. al., "Performance Measurements on a 128-Node Butterfly Parallel Processor," Proceedings of the 1985 International Conference on Parallel Processing, August 1985.

9. Manuel, Thomas, "Parallel Machine Expands Indefinitely," Electronics Week, May 13, 1985

10. Gustafson, J. et. al., " The Architecture of a Homogeneous Vector Supercomputer," Proceedings of the 1986 International Conference on Parallel Processing, August 1986.

11. Robinson, Brian, "Hypercube Sprouts Vector Processors to Challenge Supercomputers," Electronic Engineering Times, April, 1985

12. Gerald, C. F., "Applied Numerical Analysis," Addison-Wesley Publishing Co., 1980.

13. Hockney, R. W. and Jesshope, C. R., "Parallel Computers," Adam Hilga Ltd., Bristol, 1981.

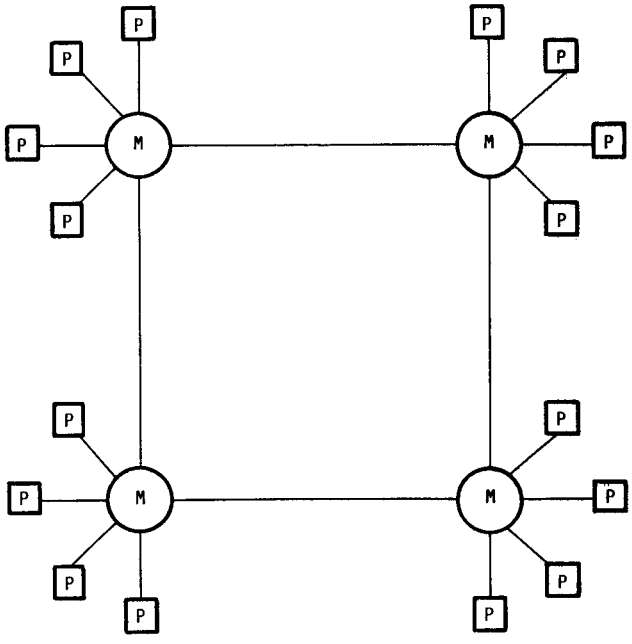

FIGURE 1. - TWO-DIMENSIONAL (D-2) HYPERCLUSTER CONFIGURATION.

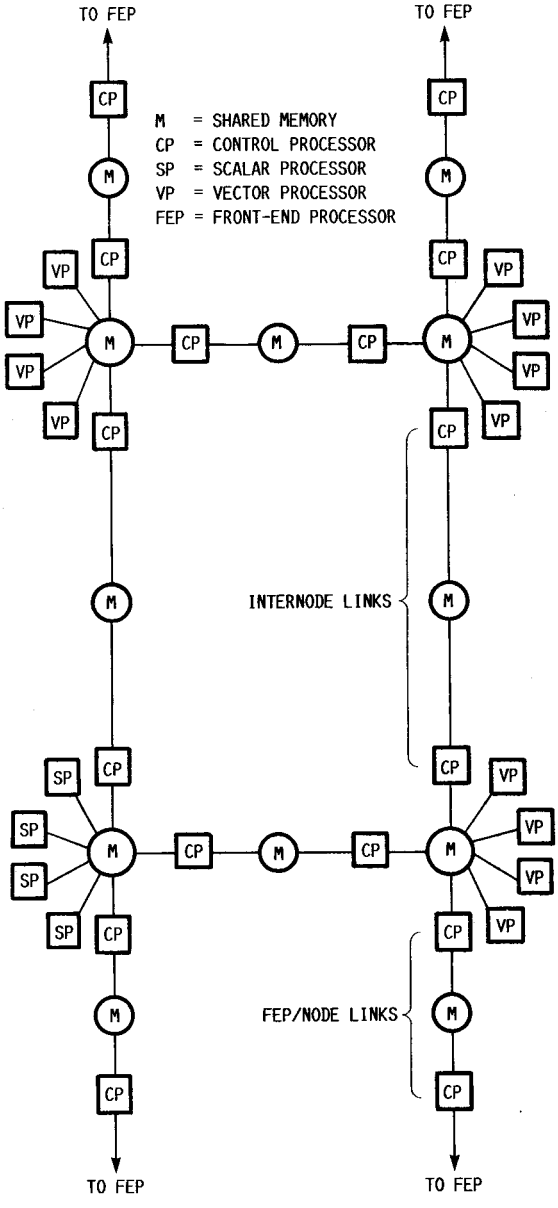

FIGURE 2. - NASA LEWIS IMPLEMENTATION OF D-2 HYPERCLUSTER.

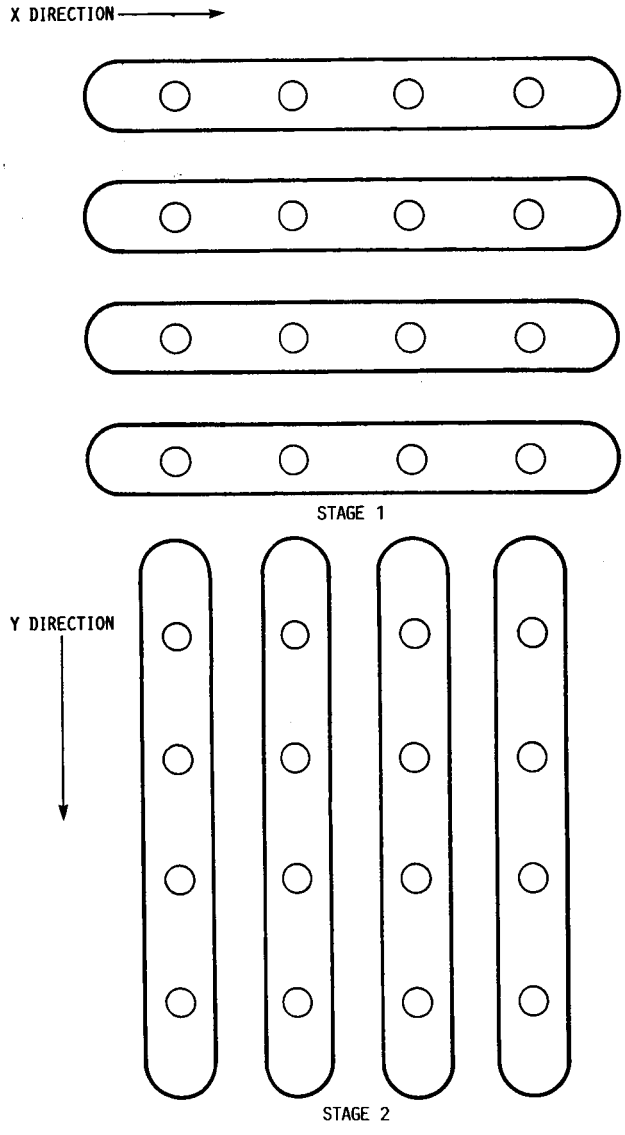

FIGURE 3. - SWEEP PATTERN FOR ALTERNATING DIRECTION IMPLICIT METHOD.

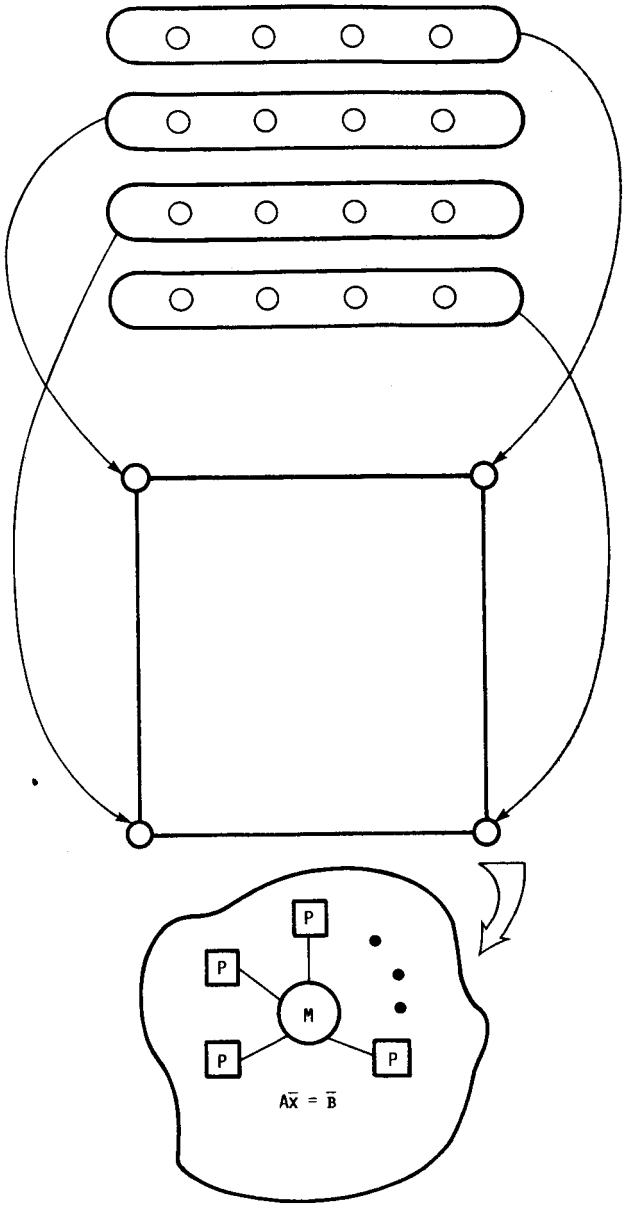

FIGURE 4.- MAPPING OF FIRST-STAGE OF ADI ALGORITHM ON HYPERCLUSTER.

```
REPEAT
   FOR ROW = 1 TO NROWS DO IN PARALLEL
      BEGIN
         CALCULATE COEFFICIENTS OF MATRIX A, VECTOR b̄
         SOLVE Ax̄^(k+1) = b̄ VIA PARALLEL ALGORITHM
      END
   TRANSFER DATA TO NEIGHBORING NODES
   FOR COLUMN = 1 TO NCOLUMNS DO IN PARALLEL
      BEGIN
         CALCULATE COEFFICIENTS OF MATRIX A, VECTOR b̄
         SOLVE Ax̄^(k+2) = b̄ VIA PARALLEL ALGORITHM
      END
   TRANSFER DATA TO NEIGHBORING NODES
UNTIL Δx̄ < ε
```

FIGURE 5. - PSEUDOCODE FOR PARALLEL ADI ALGORITHM.

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
NASA TM-89823		

4. Title and Subtitle	5. Report Date
The Hypercluster: A Parallel Processing Test-Bed Architecture for Computational Mechanics Applications	6. Performing Organization Code 505-62-21
7. Author(s)	8. Performing Organization Report No.
Richard A. Blech	E-3469
	10. Work Unit No.
9. Performing Organization Name and Address	11. Contract or Grant No.
National Aeronautics and Space Administration Lewis Research Center Cleveland, Ohio 44135	13. Type of Report and Period Covered Technical Memorandum
12. Sponsoring Agency Name and Address	14. Sponsoring Agency Code
National Aeronautics and Space Administration Washington, D.C. 20546	

15. Supplementary Notes

Prepared for the Summer Computer Simulation Conference sponsored by the Society for Computer Simulation, Montreal, Canada, July 27-30, 1987.

16. Abstract

The development of numerical methods and software tools for parallel processors can be aided through the use of a hardware test-bed. The test-bed architecture must be flexible enough to support investigations into architecture-algorithm interactions. One way to implement a test-bed is to use a commercial parallel processor. Unfortunately, most commercial parallel processors are fixed in their interconnection and/or processor architecture. In this paper, we describe a modified n-cube architecture, called the hypercluster, which is a superset of many other processor and interconnection architectures. The hypercluster is intended to support research into parallel processing of computational fluid and structural mechanics problems which may require a number of different architectural configurations. An example of how a typical partial differential equation solution algorithm maps on to the hypercluster is given.

17. Key Words (Suggested by Author(s))	18. Distribution Statement
Parallel processing; Fluid mechanics; Structural mechanics	Unclassified - unlimited STAR Category 62

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of pages	22. Price*
Unclassified	Unclassified	10	A02

*For sale by the National Technical Information Service, Springfield, Virginia 22161