NASA Contractor Report 172195

ICASE

NASA-CR-172195
19830026341

PARALLEL ARCHITECTURES FOR ITERATIVE METHODS ON ADAPTIVE, BLOCK STRUCTURED GRIDS

Dennis Gannon

and

John Van Rosendale

Contract Nos. NAS1-17070, NAS1-17130
August 1983

INSTITUTE FOR COMPUTER APPLICATIONS IN SCIENCE AND ENGINEERING
NASA Langley Research Center, Hampton, Virginia 23665

Operated by the Universities Space Research Association

Parallel Architectures for Iterative Methods on Adaptive, Block Structured Grids.

Dennis Gannon
Department of Computer Sciences, Purdue University

John Van Rosendale
ICASE, NASA Langley Research Center

ABSTRACT

This paper proposes a parallel computer architecture well suited to the solution of partial differential equations in complicated geometries. Algorithms for partial differential equations contain a great deal of parallelism. But this parallelism can be difficult to exploit, particularly on complex problems. One approach to extraction of this parallelism is the use of special purpose architectures tuned to a given problem class. The architecture proposed here is tuned to boundary value problems on complex domains. An adaptive elliptic algorithm which maps effectively onto the proposed architecture is considered in detail.

Two levels of parallelism are expolited by the proposed architecture. First, by making use of the freedom one has in grid generation, one can construct grids which are locally regular, permitting a one to one mapping of grids to systolic style processor arrays, at least over small regions. All local parallelism can be extracted by this approach. Second, though there may not be a regular global structure to the grids constructed, there will still be parallelism at this level. One approach to finding and exploiting this parallelism is to use an architecture having a number of processor clusters connected by a switching network. The use of such a network creates a highly flexible architecture which automatically configures to the problem being solved.

Research supported by the National Aeronautics and Space Administration under NASA Contract Nos. NAS1-17070 and NAS1-17130 while the authors were in residence at the Institute for Computer Applications in Science and Engineering, NASA Langley Research Center, Hampton, VA 23665. Additional support for the first author was provided by NSF Grant MCS-8108512.

I. INTRODUCTION

The development of larger and faster computers is being driven by a number of application areas where todays computers are not adequate. One of the most important of these areas is the numerical solution of partial differential equations. No computer in existence could tackle the problem of accurately modeling air flow over realistic aircraft configurations. This is so even if one neglects all time dependent effects. Similar remarks could be made about partial differential equations arising in a number of applications areas.

Faced with these needs, one could hope for computers similar to present designs, but with memory size and speeds thousands of times greater than current models. While no major technological barriers impede the development of such memories, the speed of switching elements seems to be reaching a plateau.

The solution must lie in the use of parallelism. With device densities rising rapidly, and prices dropping correspondingly, computers having hundreds or thousands of processors will be practical within a decade. Such computers will present a challenge to both the computer architect and numerical analyst. From the architects point of view, current problems like processor communication and synchronization will become more troublesome. But the changes facing the numerical analyst may be much more sweeping.

In the past, the programmer or numerical analyst was able to design algorithms taking relatively little cognizance of the architectures on which they would run. With the advent of highly parallel computation, this situation may be drastically altered. It may be that the available parallelism can be exploited only by special purpose architectures tuned to a given problem or problem class. Too, the algorithms employed will probably be highly architecture dependent, chosen as often for architectural reasons as for traditional numerical analysis reasons. More precisely, the impact of error analysis on algorithm design should change little, but complexity questions will tend to revolve around architectural considerations.

With the exception of the FEARS project (Zave, 1979) much of the recent work in computer architectures has been stimulated by a small set of simple numerical problems. Great attention has been paid to such problems as finding efficient architectures for Fourier transforms or band solution algorithms. Less tractable, and equally important problems, such as Lagrangian hydrodynamics, have received scant attention.

One class of numerical algorithms which is well understood, yet offers a great deal of the complexity frequently present in real world applications, is adaptive iterative solution of elliptic partial differential equations. This paper proposes an algorithm-architecture pair for the adaptive solution of elliptic problems in complicated domains. An attempt is made to point out and extract as much of the parallelism present as possible. We also attempt to retain as much flexibility in both the algorithm and architecture as possible.

The plan of the paper is as follows. Section II and III present the data structure and numerical algorithm being considered. Section IV describes our proposed architecture. The final section draws some tentative conclusions and suggests directions for further research.

II. DATA STRUCTURE

In this section we present our data structure for the finite element resolution of complex three dimensional domains. The data structure chosen permits the use of adaptive refinement and multigrid solution techniques. Our focus is on simple scalar elliptic partial differential equations, such as the Poisson or Helmholtz equations, and finite element grids consisting of trilinear isoparametric elements.

The proposed data structure is based on body fitted finite element grids made of macro-elements or cells. Each of our cells is an 8 x 8 x 8 grid of trilinear isoparametric elements. The use of such cells has advantages from both the architectural and data structure point of view. From the data

structure side, it creates and attractive homogeneity, and requires less storage than completely general grids. From the architectural side, it enables the use of fast special purpose hardware designed to handle the fixed size blocks of data occurring.

To create such cellular grids, we begin by assuming the domain is covered by curved *rectangular* or hexahedral regions having disjoint interiors. Each of these curved rectangular regions will become a cell in the initial grid. Associated with each curved rectangular region, we have a mapping from the canonical rectangle [0,1] x [0,1] x [0,1] onto the given curved rectangular region, as in Figure 1.

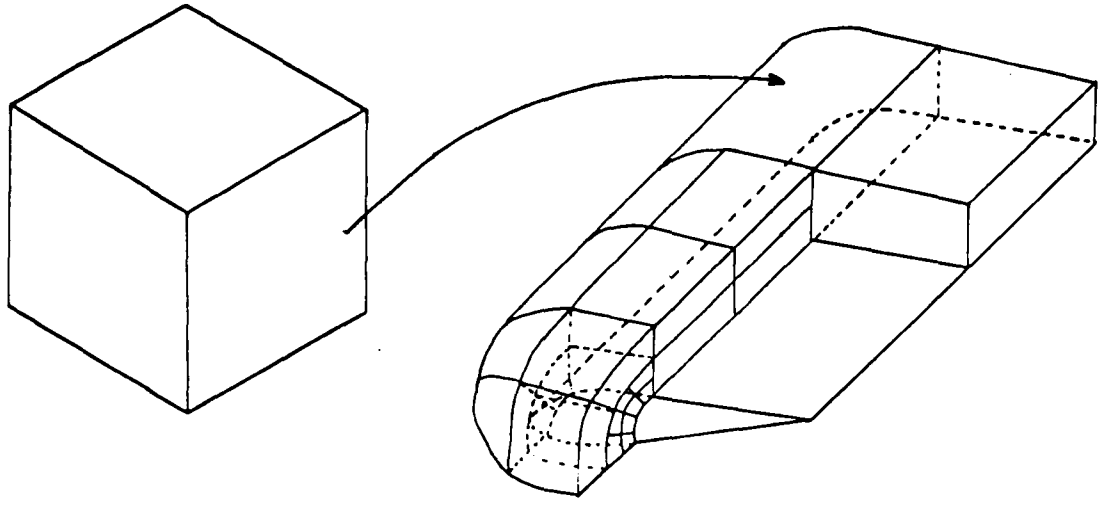

Figure 1. Mapping from Reference Cube to Physical Domain

The subdivision of a given domain into curved rectangular regions, defined by mappings like that in Figure 1, is a complex computer aided design problem. The necessity of having well defined and invertible Jacobians, usual in finite element theory, applies here as well, greatly complicating this problem. At a minimum, this requirement forces the angles at which the sides of curved rectangular regions meet along edges to be bounded away from 0 degrees and 180 degrees. These grid generation problems, though extremely difficult, are in principal solvable. Figure 2 shows the decomposition of a tetrahedron into four curved rectangular regions, showing that there is no loss of generality in restricting to domains composed of curved rectangular regions.

Associated with each of these curved rectangular regions we have an ***ancestral cell***. The collection of all ancestral cells form the initial grid used

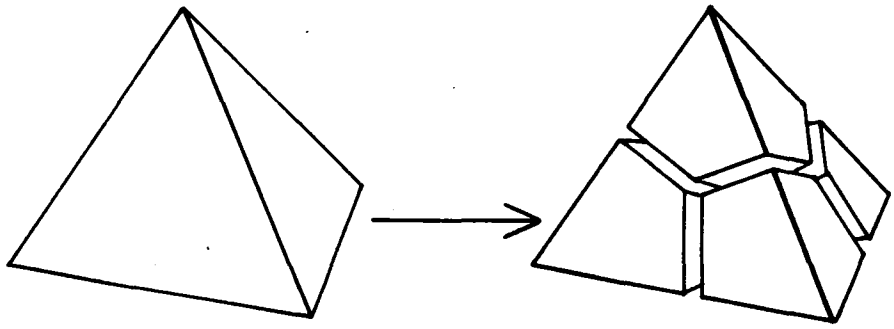

Figure 2. Decomposition of a Tetrahedron into Curved Rectangular Solids

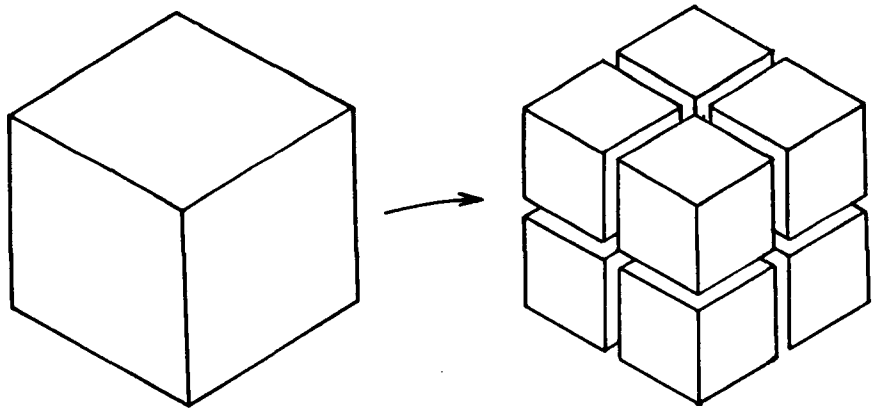

Figure 3. Refinement of a Cell into Eight Children

by the multigrid solution algorithm. This initial grid will be refined adaptively during the course of the computation. Figure 3 shows the refinement of an ancestral cell into eight children.

Associated with each cell, which is really an 8 x 8 x 8 macro-element composed of 512 trilinear elements, we need three kinds of information:

1. geometry information describing the region covered by the cell and the grid inside it.

2. numerical information, such as the local stiffness matrix.

3. data structure information to keep track of neighborhood relations between cells.

One of the advantages of the use of cells of this type is that items 1 and 3 here need be stored only once per cell, rather than separately for each of the trilinear finite elements.

We turn now to a closer examination of these three types of information associated with each cell.

A. Geometric Information

The geometry information associated with each cell is the region in the physical domain it occupies, and the location of all mesh points within it. Each cell must be either an ancestral cell or the child of an ancestral cell created by adaptive refinement. If it is ancestral, it covers the curved rectangular region p([0,1] x [0,1] x [0,1]), where p is its associated mapping, as in Figure 1. If this cell is a child it covers the region p($[r_0, r_1]$ x $[s_0, s_1]$ x $[t_0, t_1]$), for parameters r_0, r_1, s_0, s_1, t_0, t_1, and where p is the coordinate transformation mapping associated with its ancestor.

B. Numerical Information

Associated with each cell we need solution, residual and data vectors, and also the set of matrix coefficients. The three vectors here can be stored as 9 x 9 x 9 arrays of reals. This implies redundant storage along the boundaries, since boundary mesh points will be shared with other cells. The matrix coefficients are best stored as 27 point difference formulas associated with each of the 729 mesh points in the 9 x 9 x 9 grid. The alternative, storing the element stiffness matrices for each element is wasteful of storage.

C. Data Structure Information

The data structure must permit dynamic allocation of cells, in order to allow adaptive refinement and must also keep track of a number of kinds of relationships between cells. We must be able to extract neighborhood relationships, including the relative size and orientation of neighbors, in order to perform relaxation iterations. We must also maintain parent-child relation ships, as these are required in multigrid iteration. Finally, we must keep track of storage allocation in the computer architecture. Storage allocation

will be discussed briefly in section 4.

These data structure issues for locally refined grids are inherently complex, and made more so here by the need to rapidly access data structure information in the course of the computation. The data structure chosen is based on a global numbering of nodes, and requires that mesh size vary by at most a factor of two from a cell to its neighbor. Space does not permit detailed description of this data structure here, but it will be described in detail when we present the results of the completed study.

III. ADAPTIVE SOLUTION ALGORITHM

An adaptive multigrid solution algorithm can be based on the data structure described in the last section. Such a solution algorithm requires six basic numerical operations.

1. Form the stiffness matrix on a cell.

2. Form scaled data vectors on all cells.

3. Perform a Jacobi smoothing iteration on a multigrid level.

4. Perform a multigrid projection from one multigird level to the next coarser level.

5. Perform a multigrid injection from a multigird level to the next finer level.

6. Form an estimate of the local truncation error on a cell.

Designing multigrid algorithms built from these operations is reasonably straight forward and is discussed in detail in the report (Gannon and Van Rosendale 1983). Mathematically, operation 6, estimating the truncation error, is the most subtle. But from a data structure or architectural point of view, operations 2, 3 and 4 are the most difficult, because of the complex data movements involved. The basic structure of operations 3, 4 and 5 are similar. Operation 3 is described in more detail in section IV.C.

IV. ARCHITECTURE

The computations described in the previous section contain two distinct levels of parallelism which may be exploited. These computations are, at their lowest level, composed of array operations on the cells of the global mesh. The matrix product Au and the multigrid injection and projection operators are highly parallel array computations on each cell. These matrix

computations form a core of operators that dominate the serial complexity of our adaptive multigrid algorithm. Since the cells here are relatively small and since these computationally intensive operations are quite simple, a large systolic array (Kung and Leiserson, 1980) or an SIMD processor like the Burroughs BSP would be well suited to these computation.

Parellelism is also available at a higher level, since operations on different cells in the global mesh can be done concurrently. A virtual machine that would be ideal for the algorithms here consists of a network of processors organized as a complete graph with one cell per processor. More realistically, a system of processors communicating through a high bandwidth switching network, such as in the TRAC system (Kapur, Lipovski, Premkumar 1980) or PASM (Siegel, 1979) could be used, with grid cells mapped onto these processors in a many to one manner. Then each processor would be responsible for computation on each of its assigned cells.

The primary difficulty with this scheme is that an operation, such as a relaxation iteration, introduces data dependencies in the form of partially computed residuals along shared faces or edges of adjacent cells. If these adjacent cells are assigned to different processors, then the partial result data must be passed from one processor to the other over the switching network. The problem of optimally scheduling the computations on a general grid can be extremely difficult if network delays are significant. If on the other hand, computation, task scheduling and communication can be sufficiently overlapped, then it would be possible to use arrival of partial result data and the availability of hardware (such as a free floating point unit) as a mechanism for dynamically scheduling local cell computations. This is known as the "data driven" approach to task scheduling.

Both levels of parallelism can be simultaneously exploited if the algorithms and architecture are carefully matched. As one approach to achieving this, we propose a system architecture based on low level SIMD or systolic array computations, governed by a data flow synchronized, distributed operating system. The proposed hardware is organized as 16 "local clusters" communicating with each other through a message switching network. A global control processor also communicates with all local clusters via a shared bus as illustrated in Figure 4. The overall system organization is similar to that of the Cedar project (Gajski, Kuck, Lawrie, Sameh 1983).

Each local cluster consists of four units.

1. A local control processor (LCP) which communicates with the global controller and coordinates the activities of the other devices in the cluster.

2. A configurable mesh of floating point processing elements (PEs) known as the array unit (AU).
4. A network input processor called the receive unit (RU).
4. A network output processor know as the send unit (SU).

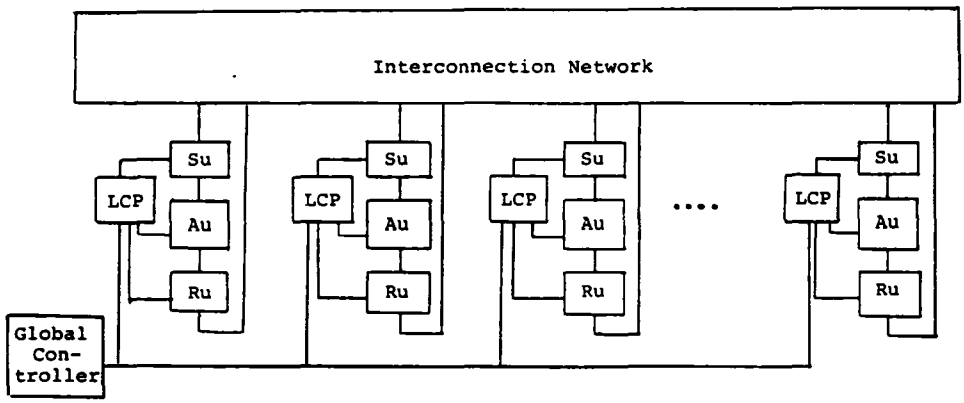

Figure 4. System Organization.

The global controller is viewed as a conventional, but reasonably powerful, front end processor which communicates with the local clusters by a system of message passing. Upon command from the global controller the local controllers initiate tasks such as Jacobi smoothing or truncation error estimation. Decisions about local refinement are made by the global controller based on information returned by the local controllers. The global controller also originates and distributes the data structure information to the rest of the system. In particular, when cell i is to be dynamically allocated, it is assigned to processor cluster $imod 16$. This approach to storage allocation equilizes, approximately, the storage and computation per processor cluster, though it induces more inter-processor communication than would be required by more subtle storage allocation schemes.

A. Local Cluster Hardware

The local controller and processing element array unit are modeled after one configuration of the CHiP system (Snyder, 1982) which is being designed for implementation on a single silicon wafer. The processors in the array are organized as a "nearest neighbor", toroidal lattice - each processor P_{ij} is connected to the 8 processors defined by $P_{(i \pm 1 mod\ 9)(j \pm 1 mod\ 9)}$. As shown

in Figure 5 (for a 4 by 4 sublattice), the PE array has two other networks interwoven with it. These networks are used for "parallel by row" loading and unloading of the local memories of the processing elements.

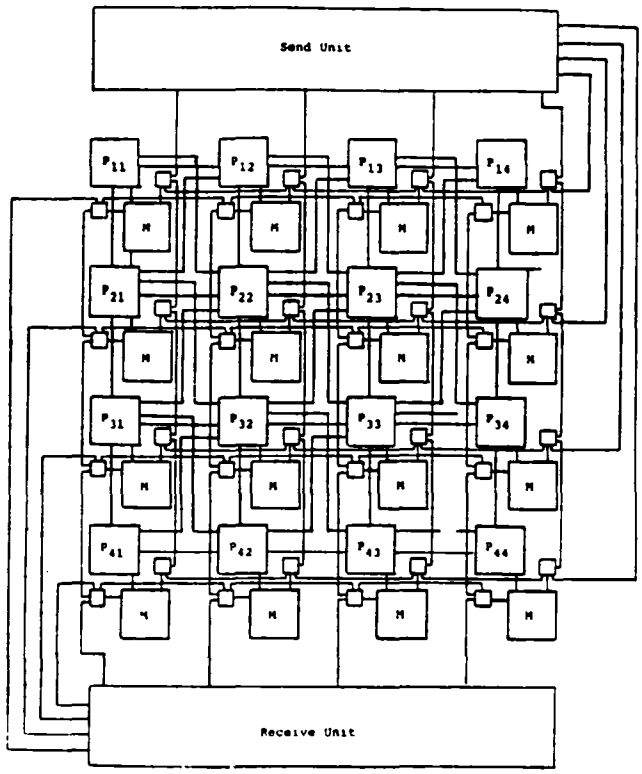

Figure 5. Array Unit Processing element organization

A moderately intelligent memory interface to the I/O networks permits the system to overlap the computation on one cell with data input on another cell and data output on a third.

Each processing element is assumed to have access to approximately 8K words of local storage which is sufficient to store the matrix components, solution vector and several temporary vectors for up to 20 cells per AU. This is also sufficient space to house the "microprograms" for each of the distinct computational tasks required in the multigrid algorithm. The array unit can be preprogrammed to perform any of the matrix operations required for computations on one cell. The 2 dimensional processor array can process a 3 dimensional "cube" of data as a sequence of 2 dimensional slices. Assuming an instruction cycle of $1\mu s$ a well designed PE can be shown to compute the matrix product Au on one cell in about 500 microseconds. This is assuming a cell size of 8 x 8 x 8. The speed here is limited both by the arithmetic

speed of the processor array and by the necessity of fetching the stiffness matrix elements from the processor cluster memory. About 30 add-multiplies are required , while 14 or 27 matrix elements must be fetched from memory per mesh point per iteration. These timing issues will be covered in more detail in a forth coming paper describing our architecture simulation in detail.

If a grid structure is composed of cells C_i, $i = 1..m$, then any linear operator A defined as a sum of cell integrals can be expressed as a nondisjoint sum of matrix products of the form

$$Au = \sum_{i=1}^{m} A^i u^i$$

where u^i is the vector of coefficients of u on C_i and A^i is the corresponding matrix defined by the integrations of A restricted to C_i. The summation takes place over the vector components corresponding to vertices of the grid that are shared by two or more cells. To extend the cell product computed as systolic array operations for individual cells to a "grid wide" product requires that the above summation be completed.

Each local cluster contains, for each of its cells, a list of adjoining cells and a description of the corresponding intersections (faces, edges and vertices). The send unit, upon command from the local controller, accesses the partial result data from the array unit memories one row at a time until a complete face or edge has been buffered. It then forms a network message using the adjoining cell record as a network address tag.

The receive unit at the destination local cluster stores the data in the array unit memories and signals the local controller that a message has arrived for the targeted cell.

B. The Switching Network

The processor clusters communicate asynchronously through a crossbar message switching network. In spite of the large number of processor elements, the demands on the switching network turn out to be quite modest: one can easily show that over 95 percent of the communication takes place within processor clusters, rather than between them. Even so, transmitting the remaining 5 percent is a non-trivial task if the machine is expected to perform in the gigaflop range.

A reasonable network for the system described here is illustrated in Figure 6a. Figure 6b gives a detail of each switch node. This network implements a full crossbar switch by bucket brigading. Messages at the inputs percolate upward along columns until they reach the row corresponding to

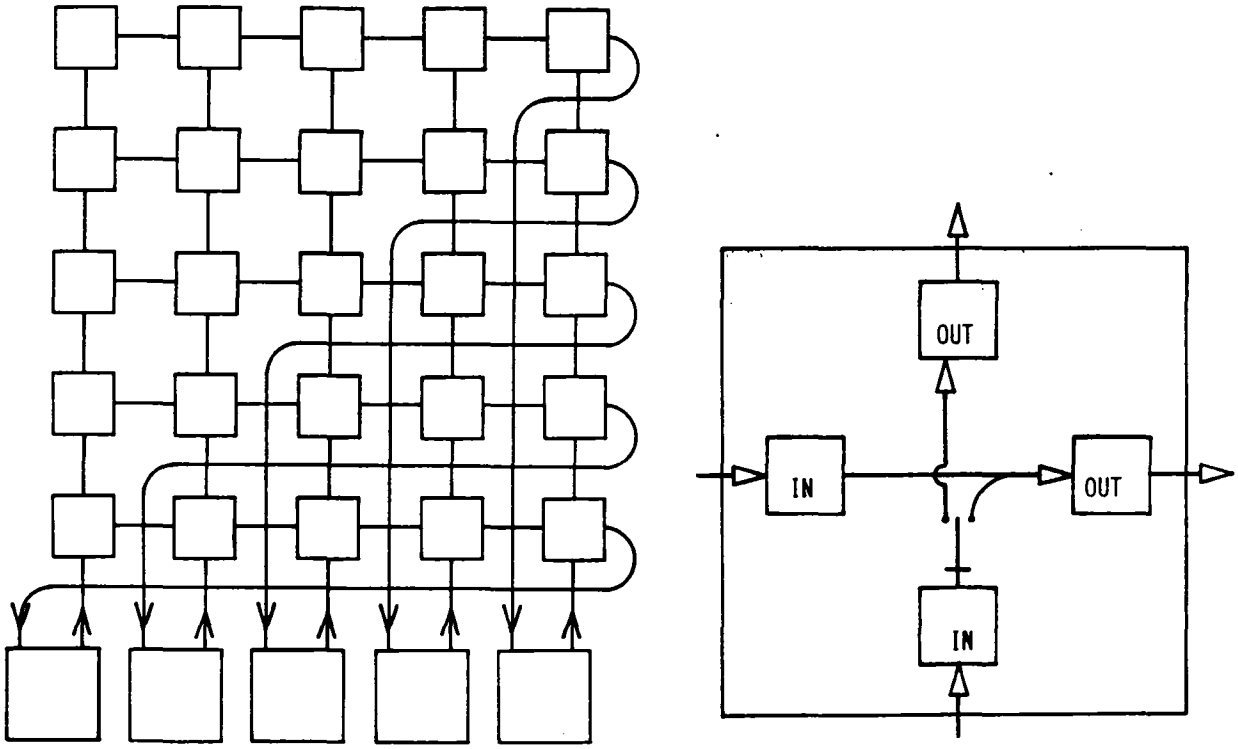

Figure 5. a). Communication Network. b) Node detail.

their target address. Then they are switched onto this row and bucket brigaded out to the right.

C. The Local Cluster System Software.

The local controller has the responsibility for managing the resources and scheduling the computations on each of the cells assigned to it. Each cell is viewed as having a virtual processor which may execute any of several "cell processes".

A cell process consists of a sequence of tasks. For example, the cell process for a relaxation iteration that would be executed on each cell of a specified grid level goes roughly as follows:

```
Relaxation_process{ x: cell# }
    {
      1. request the A.U. to do a matrix multiply on
           cell x.
      2. request the send unit to transmit the partial
           result face data to all adjoining cells.
      3. wait for the arrival of partial result face data
           from adjoining cells.
      4. request the A.U. to add the partial results and
```

to compute $u := u + \lambda(f - Au)$.

5. signal the global controller that an iteration is complete on cell x.

}

A schematic representation of 3 processors executing a relaxation on a grid consisting of 9 cells is given in Figure 7. The dotted lines between processes represent the flow of messages through the network to waiting cells.

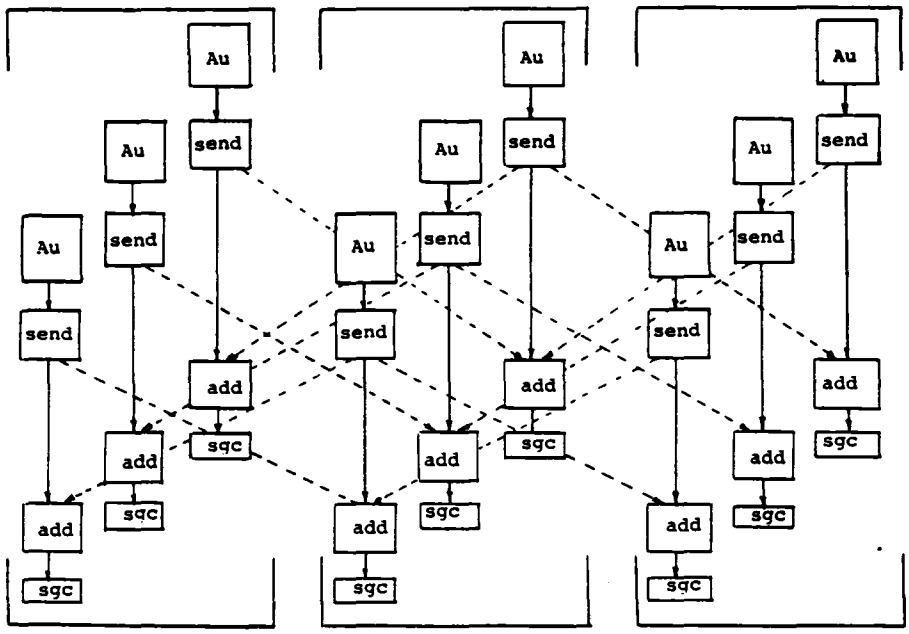

Figure 7. Nine Concurrent Relaxation Processes on Three Processors.

In a similar manner cell processes are constructed for the other basic solution operations, such as the matrix assembly and interpolations between grid levels.

The operating system for each local control processor is interrupt driven and is organized as a set of resource monitors (Hoare, 74) for the A.U., the send unit, the receive unit and for messages from the global controller. Each cell has a cell descriptor record which, in addition to its geometric attributes, lists its current process and task state. Each resource monitor maintains a queue of waiting cells (a list of descriptor records) and a pointer to a descriptor for the cell currently in control of the device. When a cell process of a cell requests service from a device, it invokes the appropriate monitor, which places that cells descriptor on the queue of waiting cells.

The global control message monitor maintains a queue of "jobs" for each cell. Messages that arrive from the global controller cause an interrupt and the LCP puts the message in the appropriate queue. When a cell completes a process it enters an idle state and the system initiates a new process by fetching the next command from the queue.

V. CONCLUSION.

Using simulation techniques, we are investigating several questions of system performance. The first of these is the problem of estimating the system overhead due to local controller task scheduling. Preliminary simulation results indicate that a system with a 1 microsecond cycle time for all local cluster operations would have software overhead that would consume 10-25% of total execution time, depending on the degree of multiprogramming. At this clock rate, a system with 16 array units, each consisting of a 9 by 9 array of processor elements, has a possible throughput of 1.3 gigaflops. Accounting for software overhead and a 100 microsecond message delay, the achieved performance is in excess of 800 megaflops. This result was obtained using a multigrid program on a model problem having 93 cells. The switching network needed to communicate 12 words per clock cycle on average. As the complexity of the problem increased the performance increased to over 900 megaflops with an increase of message traffic to only 14 words per clock cycle. The amount of data transmitted by the switching network turns out to be only about 2 percent as large as the amount of data transmitted between the processor elements and their attached memory. This low message traffic justifies our claim that a reasonably simple network will provide all the global communication required in this class of algorithms. A more complete analysis will be published when the study is complete.

Among the questions under current study are:

1. The effect of different degrees of multiprogramming on system performance. As the number of cells per local cluster increases, the utilization of the array units rises until the system is saturated. The exact form of this performance curve is being studied.
2. The design of efficient I/O algorithms compatible with parallel architectures. As computers become faster, extracting meaningful information from the reams of output produced becomes an increasingly difficult problem. Distributed graphics algorithms are being considered.

VI. REFERENCES.

Gajski, D, Kuck, D., Lawrie, D., Sameh, A., (1983) "CEDAR, A Large Scale Multiprocessor," Technical Report, Cedar project, Dept. of Comp. Science University of Illinois at Urbana-Champaign.

Gannon, D., Van Rosendale, J. (1983), "Highly Parallel Multigrid Solvers for Elliptic PDEs: An Experimental Analysis," ICASE Report 82-36, ICASE, NASA Langley Research Center, Hampton, Virginia.

Hoare, C. A. R. (1974), "Monitors: An Operating System Structuring Concept,", Comm. ACM 17, 10, pp.549-57, Oct.

Kapur, R., Premkumar, U., Lipovski, J.(1980), "Organization of the TRAC Processor-Memory Subsystem," AFIPS Conf. Proc. pp.632-629.

Kung, H. T., Leiserson, C. E. (1980), "Algorithms for VLSI Processor Arrays," in Mead and Conway, Introduction to VLSI Systems, Addison-Wesley, Reading, Ma., pp.271-292.

Siegel, H. J., et. al. (1979), "An SIMD/MIMD Multimicroprocessor System for Image Processing and Pattern Recognition," IEEE Conf. on Pattern. Recog. Image Proc. pp. 214-224.

Snyder, L. (1982), "CHiP: The Configurable Highly Parallel Computer," Computer, Jan. 1982.

Zave, P., Rheinboldt, W. (1979), "Design of an Adaptive, Parallel Finite-Element System," ACM Trans. on Math. Software. Vol. 5(1), 1979, pp.1-17.

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
NASA CR-172195		

4. Title and Subtitle	5. Report Date
Parallel Architectures for Iterative Methods on Adaptive, Block Structured Grids	August 1983
	6. Performing Organization Code

7. Author(s)	8. Performing Organization Report No.
Dennis Gannon & John Van Rosendale	83-39
	10. Work Unit No.

9. Performing Organization Name and Address	11. Contract or Grant No.
Institute for Computer Applications in Science and Engineering Mail Stop 132C, NASA Langley Research Center Hampton, VA 23665	NAS1-17070, NAS1-17130
	13. Type of Report and Period Covered
12. Sponsoring Agency Name and Address National Aeronautics and Space Administration Washington, D.C. 20546	Contractor report
	14. Sponsoring Agency Code

15. Supplementary Notes
Additional support: NSF Grant MCS -8108512,
Langley Technical Monitor: Robert H. Tolson
Final Report

16. Abstract

This paper proposes a parallel computer architecture well suited to the solution of partial differential equations in complicated geometries. Algorithms for partial differential equations contain a great deal of parallelism. But this parallelism can be difficult to exploit, particularly on complex problems. One approach to extraction of this parallelism is the use of special purpose architectures tuned to a given problem class. The architecture proposed here is tuned to boundary value problems on complex domains. An adaptive elliptic algorithm which maps effectively onto the proposed architecture is considered in detail.

Two levels of parallelism are exploited by the proposed architecture. First, by making use of the freedom one has in grid generation, one can construct grids which are locally regular, permitting a one to one mapping of grids to systolic style processor arrays, at least over small regions. All local parallelism can be extracted by this approach. Second, though there may not be a regular global structure to the grids constructed, there will still be parallelism at this level. One approach to finding and exploiting this parallelism is to use an architecture having a number of processor clusters connected by a switching network. The use of such a network creates a highly flexible architecture which automatically configures to the problem being solved.

17. Key Words (Suggested by Author(s))	18. Distribution Statement
parallel processing computer architectures adaptive methods	61 Computer Programming and Software Unclassified-Unlimited

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price
Unclassified	Unclassified	16	A02

End of Document