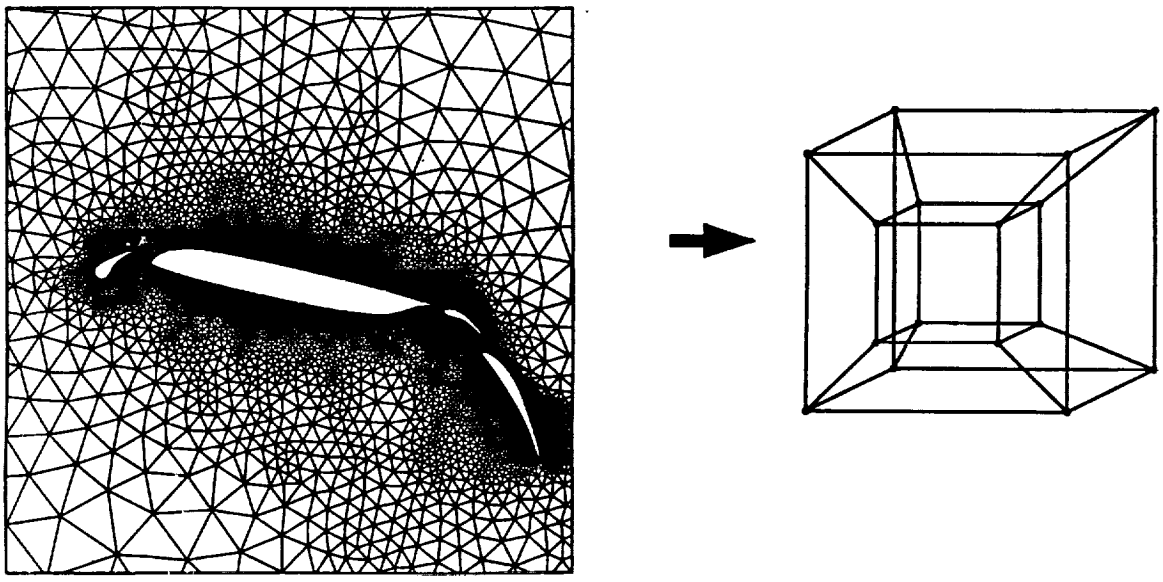
Research Institute for Advanced Computer Science
NASA Ames Research Center

Mapping Unstructured Grid Problems to the Connection Machine

Steven W. Hammond **Robert Schreiber**



RIACS Technical Report 90.22 **October 1990**

Mapping Unstructured Grid Problems to the Connection Machine

Steven W. Hammond Robert Schreiber

The Research Institute of Advanced Computer Science is operated by Universities Space Research Association, The American City Building, Suite 311, Columbia, MD 244, (301)730-2656

Work reported herein was supported by the NAS Systems Division of NASA and DARPA via Cooperative Agreement NCC 2-387 between NASA and the University Space Research Association (USRA). Work was performed at the Research Institute for Advanced Computer Science (RIACS), NASA Ames Research Center, Moffett Field, CA 94035.

Mapping Unstructured Grid Problems to the Connection Machine

Steven W. Hammond * Robert Schreiber †

Aug 1990

Abstract

We present a highly parallel graph mapping technique that enables one to efficiently solve unstructured grid problems on massively parallel computers. Many implicit and explicit methods for solving discretized partial differential equations require each point in the discretization to exchange data with its neighboring points every time step or iteration. The time spent communicating can limit the high performance promised by massively parallel computing. To eliminate this bottleneck we map the graph of the irregular problem to the graph representing the interconnection topology of the computer such that the sum of the distances that the messages travel is minimized. We show that, in comparison to a naive assignment of processors, our heuristic mapping algorithm significantly reduces the communication time on the Connection Machine CM-2.

*Visiting Research Associate, Research Institute for Advanced Computer Science, NASA Ames Research Center, Moffett Field, CA 94035 and Ph.D. Student at Rensselaer Polytechnic Institute, Troy, NY 12180.

†Senior Scientist, Research Institute for Advanced Computer Science, NASA Ames Research Center, Moffett Field, CA 94035.

1 Introduction

In fluid dynamics, structural mechanics, electromagnetics, combustion, and many other applications, the problems to be solved are nearly always initial-boundary value problems for coupled systems of partial differential equations (PDEs), quite often nonlinear. Many projects have demonstrated that massively parallel architectures are very effective at solving PDEs when grids are fixed and topologically simple, Cartesian for example [1, 16, 17]. Here we investigate the use of these architectures for more difficult problems, where the grid is arbitrary, but static. One very important problem is to map the unstructured grid to the architecture so that grid neighbors are not distant. Such a mapping allows for fast communication of data stored at grid points to neighboring grid points. In general, graph mapping problems of this type are difficult (hypercube embedding is NP-complete). We can, however, use heuristics.

We have developed a highly parallel heuristic graph mapping algorithm and implemented it on the Connection Machine CM-2. For very large, very irregular problems arising in 2D fluid flow problems it has achieved excellent results. The method has outperformed methods based on simulated annealing. By this we mean that we need far less time to do the embedding and the results obtained are as good. Compared with naive or random embeddings, we are able to reduce communication time threefold, even for realistic, large, highly irregular and stretched meshes.

The rest of this paper is organized as follows. In Section 2 we define the mapping problem and discuss related and previous work. The architecture of the Connection Machine CM-2 is described in Section 3. In Section 4 we discuss the parallel mapping algorithm. Section 5 discusses the experiments and results of using the heuristic.

2 The Mapping Problem

The mapping problem arises because we want to minimize the communication time for an application whose tasks have been distributed over many processors of a massively parallel local memory computer. These distributed tasks must repeatedly exchange locally stored or computed data with their neighbors. The tasks and their associated communications form a graph $G = (V_G, E_G)$ where there is a vertex, V_G for each task. Additionally, for each task i that communicates with another task j there is an edge

$(i, j) \in E_G$. For simplicity we assume that the computation is composed of unit tasks and and unit communications. This is a reasonable assumption for a single instruction multiple data (SIMD) computer since all processors do the same thing at the same time.

The processors and their interconnection network can be represented as a graph also. Let the graph $H = (V_H, E_H)$ represent the parallel computer where there is a vertex, V_H for each processor. Additionally, for each processor p that is directly connected with another processor q there is an edge $(p, q) \in E_H$.

Let $\psi : E_G \rightarrow E_H$ and $u \in V_G$. Define

$$\lambda(u, \psi) = \sum_{(u,v) \in E_G} (distance(\psi(u), \psi(v)))$$

where *distance* is the length of the shortest path in H from $\psi(u)$ to $\psi(v)$. For a hypercube, this distance is the Hamming distance. $\lambda(u, \psi)$ is the contribution one vertex makes to the total communication load on the system – the sum of the distances that all messages originating at a single vertex must travel. Also, we define the total communication load, Λ, to be

$$\Lambda(G, H, \psi) = \sum_{u \in V_G} (\lambda(u, \psi)).$$

We seek to reduce Λ by a good choice of the map ψ. When the choices of G, H and ψ are clear we will simply use Λ.

One might wonder whether it would be better to minimize the maximum message distance rather than the sum of message distances. It is true that the maximum message distance is a lower bound on the communication time. Experiments on the CM with our problems have shown that the latter is better correlated to the actual communication time.

When grids are created the grid points are usually given some numbering such as the order that they are generated – the first grid point generated would be labeled number 1. We call the mapping of grid point 1 to processor 1, grid point 2 to processor 2, and so forth, a *naive* mapping.

2.1 Related Work

Hypercube networks [26] have attracted much attention because of their powerful topological properties [21] and their development into products by several manufacturers. Also, it has long been known that multi-dimensional

grids of suitable dimension can be embedded as subgraphs of the hypercube by means of gray codes [12].

Many people have considered mapping applications to hypercubes and other parallel processors [2, 3, 5, 6, 7, 9, 10, 11, 14, 18, 19, 20, 22, 23, 25, 24, 28]. But, most of these efforts were sequential algorithms focused on mapping graphs with hundreds of vertices to machines with tens of processors, not thousands of processors. Here we consider parallel techniques for mapping graphs with tens of thousands of vertices to a computer with thousands of processors.

One might also consider first partitioning the graph and then mapping partitions to processors. But, performing the two operations in isolation can lead to poor mappings and much less than optimal communication time [9].

Dahl [8] has shown that simulated annealing to minimize Λ can be effective at reducing the communication time for the class of problem considered here. Simulated annealing is attractive because it can achieve good results if run long enough and typically avoids getting stuck in local minima. On the other hand, it cannot take advantage of the fact that one often knows the spatial locations of the grid points. This information can be used to form an initial guess that reduces the time to obtain a good mapping. Also, it requires the user to specify parameters of the algorithm (beginning and ending "temperatures"). Optimal choice of these inputs differ for every graph and if chosen incorrectly can greatly increase the running time of the heuristic.

3 Connection Machine Characteristics

The Connection Machine CM-2 is a massively parallel SIMD computer with 64K processors. It's underlying topology is an 11-dimensional hypercube of *sprint nodes*. A sprint node is composed of 32 processors (two processor chips containing 16 processors each), a Weitek floating point chip and memory. Neighboring sprint nodes are connected by two bi-directional 1-bit paths. Communication between the 32 processors on a sprint node is very inexpensive relative to communication with processors on other sprint nodes. In order to analyze communication time, one ignores the fact that sprint nodes contain 32 processors and focuses on reducing the cost of inter-sprint-node communication.

Communication is much slower than computation on the CM-2. Nearest neighbor communication (often referred to as NEWS communication) rates vary between 1.6×10^9 and 7.3×10^7 floating point words communicated

per second, depending on the virtual processor ratio. The realizable peak computation rate is 5.7×10^9 flops [15]. Using the router for collision-free distance 1 communication is approximately a factor of 8 slower than the NEWS network [4].

When algorithms and architectures match then the performance is very good. Many people have shown that well matched computations on the Connection Machine can be implemented using regular (nearest neighbor) communications and achievable rates are close to the realizable peak rate [4, 10, 15, 17, 27]. On the other hand, if an algorithm requires general communication between processors (using the router) then there can be 3 orders of magnitude (or greater) difference between realized and peak performance [4].

Until recently, general communication on the Connection Machine required the router and was excruciatingly slow. This is particularly true for solving unstructured grid problems where the communication pattern does not match the interconnection topology of the parallel computer. A feature of the communication we focus on here is that the communication pattern, although irregular, remains static throughout the duration of the computation. Dahl [8] has developed communication software called the "Communication Compiler" to take advantage of the fixed pattern. It is a software facility for scheduling completely general communications on the Connection Machine. The user specifies a list of source locations and destinations for messages and enables one to fully utilize the large communication bandwidth of the machine. The communications are scheduled once at the beginning of the program and then the message routing pattern is used repeatedly throughout the duration of the program. The schedule is a set of one or more *message cycles*. A message cycle is a single communication step when messages can be moved across all bi-directional paths connecting sprint nodes. A single sprint node on an 8K processor CM-2 has 16 bi-directional connections. In one message cycle each sprint node can send and receive 16 messages.

Communication using the compiler is a factor of $5 - 10$ faster than using the router for general communication. An intelligent mapping of tasks to processors as proposed here results in further improvements.

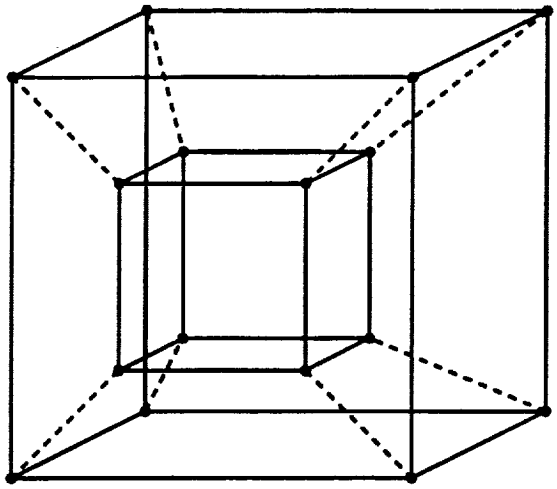


Figure 1: 4-dimensional hypercube split into two 3-cubes.

4 A Parallel Mapping Algorithm

In this section we discuss the heuristic used to reduce Λ and thus the communication time. Our heuristic consists of an iterative improvement of the initial mapping by performing parallel pairwise exchanges of the processors to which the vertices are mapped. The heuristic loops over each dimension of the hypercube. At each iteration of the loop, the chosen dimension partitions the hypercube into two equal size subcubes. Figure 1 shows a 4-dimensional hypercube partitioned into two 3-cubes. The dashed line represents the chosen dimension and highlights pairs of sprint nodes. The algorithm picks two vertices from each sprint node pair, one from each, and swaps their location. Recall that on the CM-2 there are 32 processors on a sprint node. If we assume that we map one vertex to each processor then each sprint node holds 32 vertices. We choose the vertex from each sprint node that causes the largest reduction in Λ when moved. Let ψ be the current mapping and let ψ' be the new mapping if some vertex v was moved across the dimension highlighted by the dashed line. For each vertex v compute the reduction in Λ

$$reduction_v = \lambda(v, \psi) - \lambda(v, \psi').$$

Each sprint node chooses the vertex with the largest reduction. Let vertices m and n be two such vertices from neighboring sprint nodes. We perform

the exchange if and only if

$$0 \leq reduction_m + reduction_n.$$

These computations and vertex exchanges are performed in parallel by every node in the hypercube of sprint nodes. For a given dimension the maximum possible reduction in Λ is made every iteration. We call one complete loop over all dimensions of the hypercube of sprint nodes a *sweep*.

Not every grid comes with the same number of vertices as the number (or some multiple of the number) of processors in the computer system. We fill in the difference with *wild card vertices*. A wild card vertex is a vertex with no neighbors. Wild card vertices provide a uniform way to handle special cases on a SIMD computer. Since there are no neighbors, the function $\lambda(u, \psi)$ always has the value 0 for these vertices.

5 Results

We tested our heuristic on 2 grids arising in computational fluid dynamics. The graph show in Figure 2 is part of a triangular discretization around a 4 component airfoil. It has 15606 vertices and 45878 edges. The grid shown in Figure 3 is part of an unstructured mesh around a 3 component airfoil. The grid has 4720 vertices and 13722 edges. These are particularly hard test cases because they have multiple areas of refinement, holes for the wing body and flaps, and the ratio of the longest edge length to the shortest is greater than 1000.

A data parallel mesh-vertex upwind finite-volume scheme for solving the Euler equations on the grids show in Figures 2 and 3 has been developed [13]. The problem is mapped to an $8K$ processor CM-2 (8-dimensional hypercube of sprint nodes). A common operation in the application is for the processor associated with each vertex to communicate with the processors assigned to a subset of its neighboring vertices. There is one communication for each edge. We call this operation a `get`.

In Tables 1 and 2 we compare 4 different forms of this `get` operation for the grids shown in Figures 2 and 3. We compare 1) random initial mapping and the router, 2) hueristic mapping and the router, 3) random initial mapping and the communication compiler, and 4) hueristic mapping and the communication compiler. Each test is executed 1000 times. The first column shows the value of Λ. The second column shows the number of sweeps of the heuristic applied. The third column lists the number of

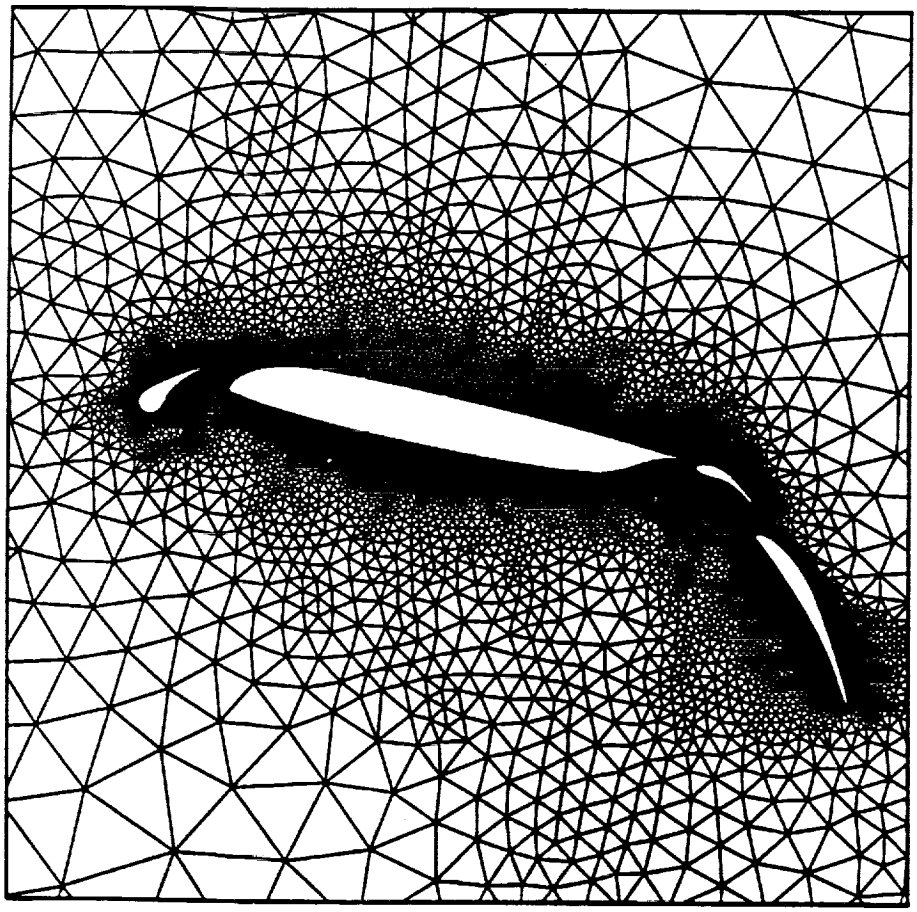


Figure 2: Closeup of mesh about 4 component airfoil with extended flaps.

	Λ	sweeps	message cycles	time in seconds for 32-bit message
random + router	183521	0	–	30.14
heuristic + router	48851	30	–	30.17
random + compiler	183521	0	53	4.12
heuristic + compiler	48851	30	17	1.49

Table 1: Comparison of router and Communication Compiler for random mapping and heuristic mapping on 4 component grid.

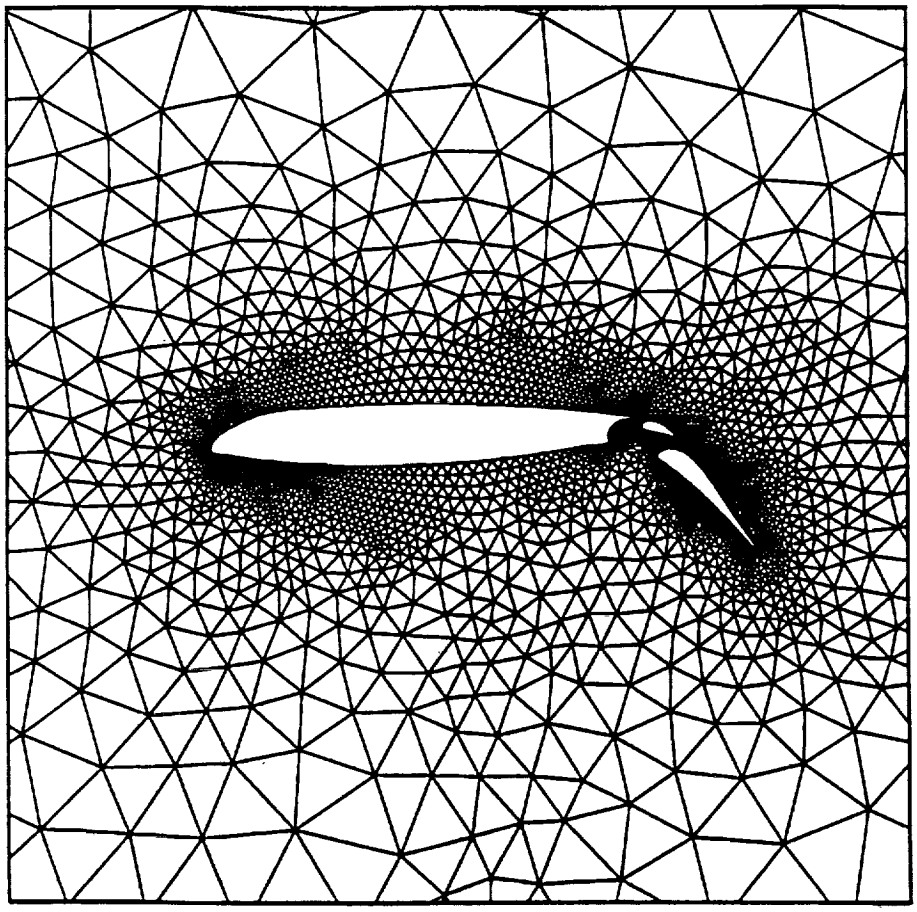


Figure 3: Closeup of unstructured mesh about 3 component airfoil with flaps down.

	Λ	sweeps	message cycles	time in seconds for 32-bit message
random + router	55222	0	–	13.68
heuristic + router	17488	9	–	12.77
random + compiler	55222	0	30	1.52
heuristic + compiler	17488	9	9	0.78

Table 2: Comparison of router and Communication Compiler for random mapping and heuristic mapping on 3 component grid.

message cycles required for the **get** operation. Message cycles only apply to the communication compiler and not to the router. The fourth column shows the delivery times in seconds for 32-bit data. Note that the compiler alone reduces the time by a factor of 9 in Table 1 and a factor of 7 in Table 2 compared to the router. This is primarily due to the fact that the compiler knows *a priori* the total communication load and schedules messages to use all wires in parallel. The router does not have the same information available at any one time and is therefore less efficient. Replacing the random mapping and the compiler with the heuristic mapping and compiler produces almost a threefold reduction in communication time. Even though we reduce the number of message cycles by a factor of 3 for both test graphs the communication time is reduced 2 to 2.5-fold. The time reduction is not threefold because the system overhead becomes significant as the number of message cycles is reduced. For problems with higher levels of communication than these two relatively small and sparse test graphs we expect to see a close correlation between reduction in the number of message cycles and the reduction in communication time.

30 sweeps of the heuristic on the 4 component mesh took 35 seconds and 9 sweeps on the 3 component mesh took 7.2 seconds on the CM-2. The timings were done on a CM-2 hosted by a Sun-4. The program was written in *lisp.

Now we further illustrate the effectiveness of the heuristic for reducing Λ and thus the communication time. In Figures 4 - 6 we show the reduction in Λ and the reduction in the number of routing cycles as a function of the number of sweeps of the heuristic. The graphs in Figures 4 and 5 correspond to the test case in Figure 2. The graphs in Figure 6 refer to the test grid in Figure 3. We plot two graphs for each test case. The top graph shows the reduction of Λ as a function of the number of sweeps. The bottom graph shows the reduction in the number of message cycles as a function of the sweeps.

For the data in Figure 4 and 6, we begin with a naive mapping (vertex 1 goes to processor 1, ..., vertex n goes to processor n) and then run the heuristic. (Note that Λ is also reduced threefold. This confirms that our objective function is a predictor for the communication time.)

In Figure 5 we started with 3 random initial assignments of vertices to processors and then used the heuristic on each initial guess. It is interesting to see that random initial guesses produce a very large Λ, but initially require fewer message cycles to complete the communication than the naive initial guess. This is because the communication load is distributed over

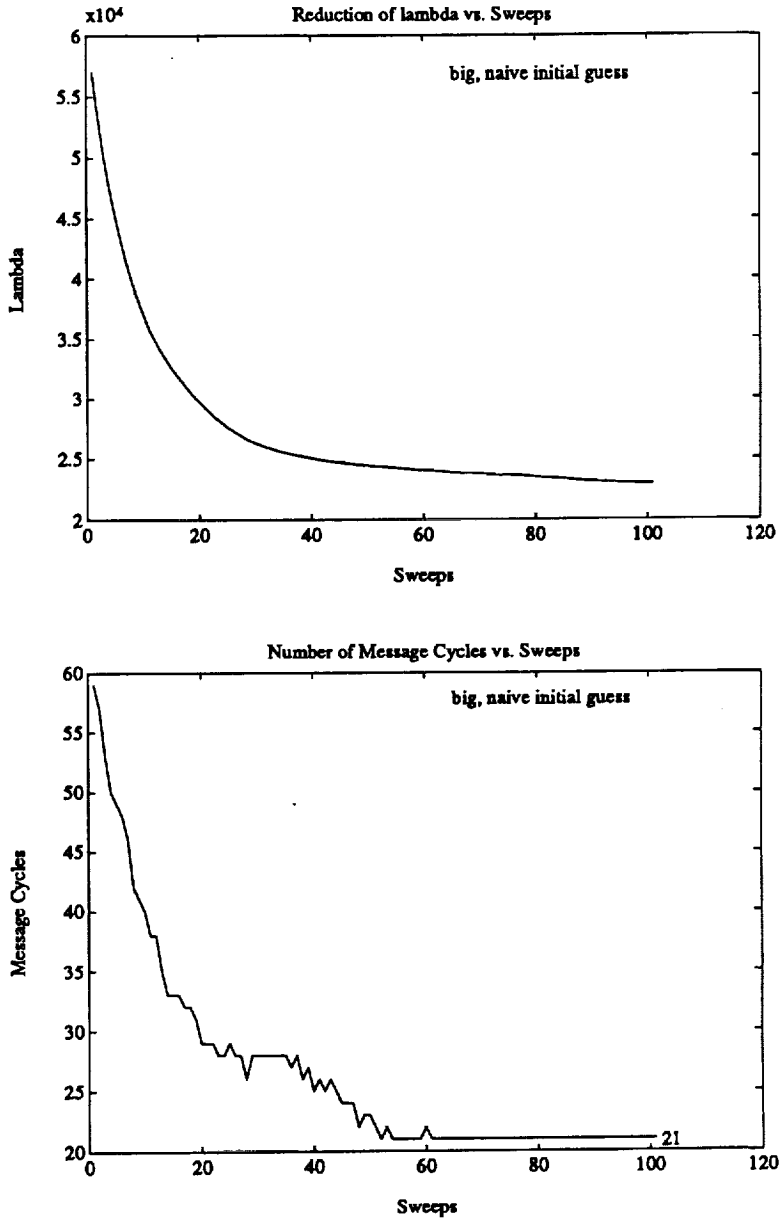


Figure 4: Reduction in Λ and message cycles versus sweeps for 4elt graph.

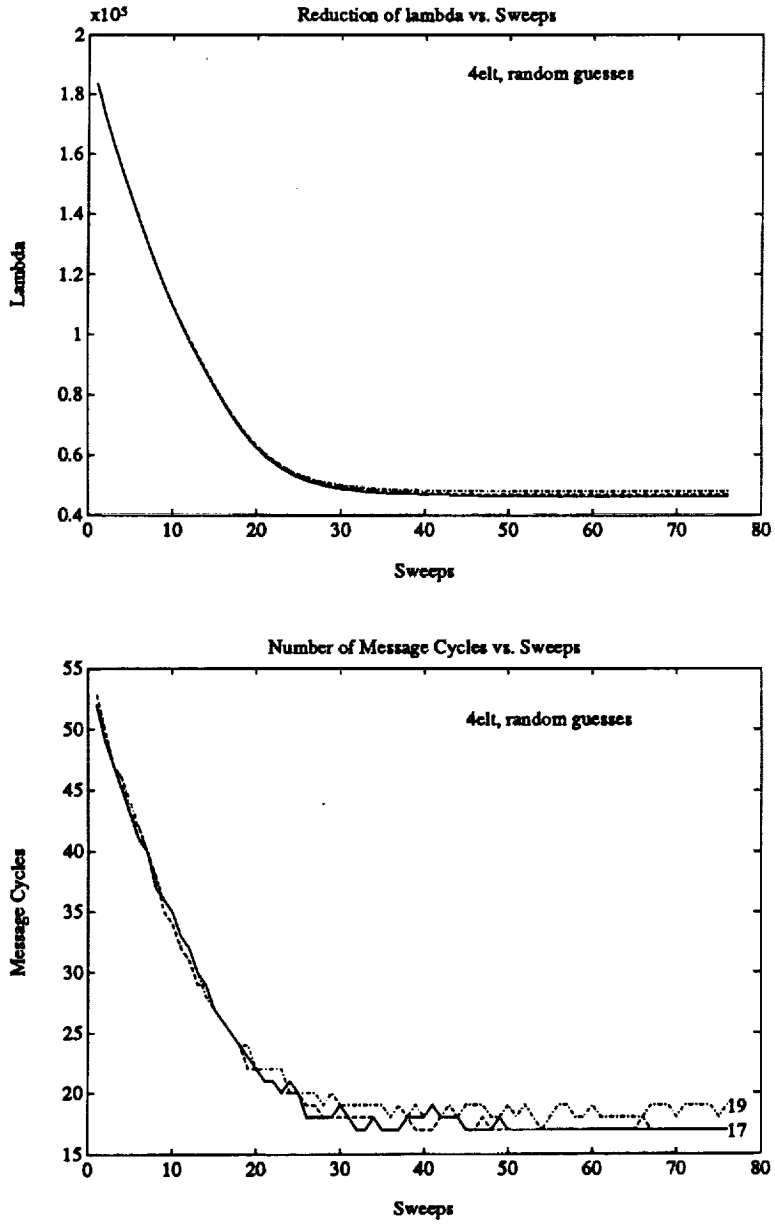


Figure 5: Reduction in Λ and message cycles versus sweeps for 4elt graph.

the machine. To some extent this is an artifact of the current state of the communication compiler. It does not choose alternative wires in its scheduling. For example, if 10 messages all have to go distance 1 across the same wires the compiler serializes the communication rather than seeking unused alternative paths.

6 Summary

We have introduced a highly parallel heuristic mapping algorithm for assigning processes to processors on a massively parallel computer. The proposed heuristic has been applied to unstructured grids typical of those used in solving discretized PDE's. We have shown that we can reduce the communication time by a factor of three by using our heuristic mapping together with the communication compiler compared to using the communication compiler with a naive or random mapping.

Acknowledgement

This work was supported by DARPA via Cooperative Agreement NCC 2-387 between NASA and the University Space Research Association (USRA). We would like to thank Dennis Jespersen and Timothy Barth of NASA Ames Research Center for supplying us with the unstructured grid. Also, Denny Dahl has provided us with valuable insight into the way messages are delivered on the CM and with the communication compiler software.

References

[1] R. K. Agarwal and J. L. Richardson. Development of an euler code on a connection machine. In Horst D. Simon, editor, *Proc. of the Conference on Scientific Applications of the Connection Machine, NASA Ames Research Center, Moffett Field, California*, pages 27–63, World Scientific, September 12-14, 1988.

[2] C. Aykanat, F. Özgüner, F. Ercal, and P. Sadayappan. Iterative algorithms for solution of large sparse linear equations on hypercubes. *IEEE Trans. Comp.*, 37(12):1554–1567, December 1988.

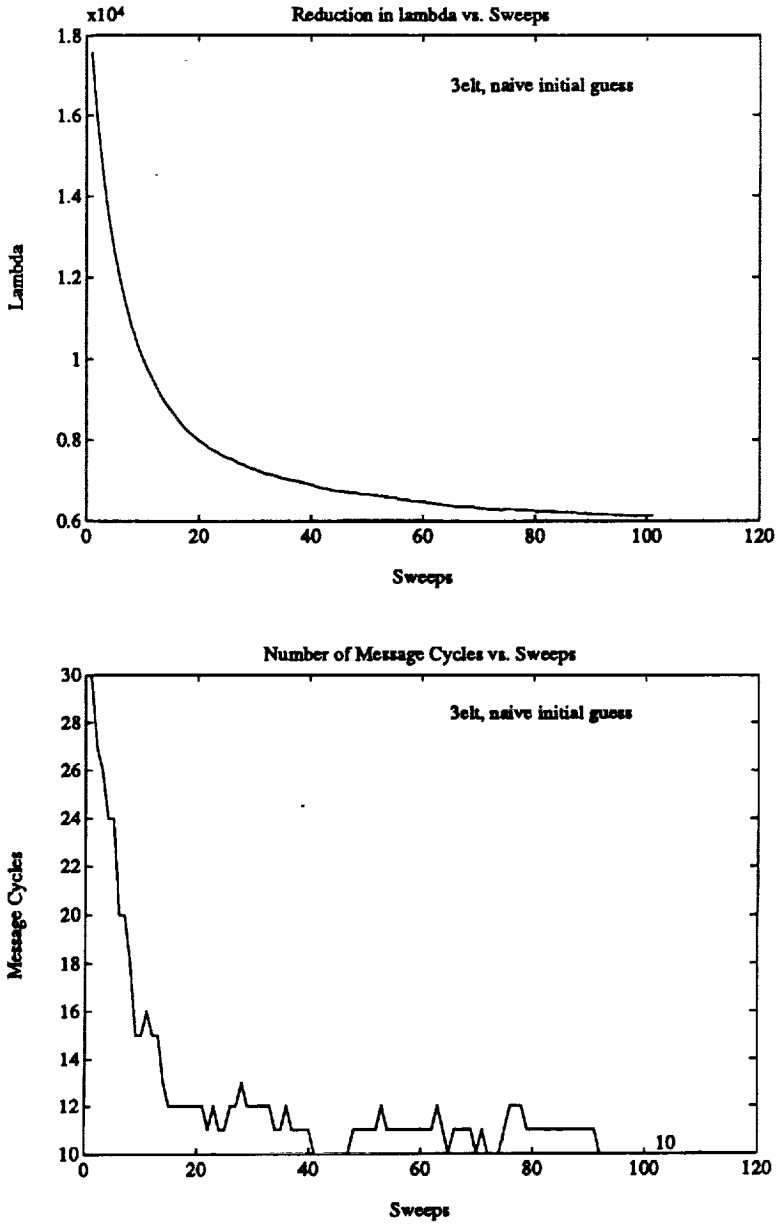


Figure 6: Reduction of Λ and message cycles versus sweeps for small graph.

[3] M. J. Berger and S. H. Bokhari. A partitioning strategy for nonuniform problems on multiprocessors. *IEEE Trans. Comp.*, 36(5):570–580, May 1987.

[4] H. Berryman, J. Saltz, and W. Gropp. *Krylov Methods Preconditioned with Incompletely Factored Matrices on the CM-2.* Technical Report TR-685, Department of Computer Science, Yale University, New Haven CT, March 1989.

[5] S. H. Bokhari. On the mapping problem. *IEEE Trans. Comp.*, 30(3):207–214, March 1981.

[6] M.-S. Chen and K. Shin. Embedding of interacting task modules into a hypercube. In M. Heath, editor, *Hypercube Multiprocessors 1987*, pages 122–129, SIAM, Knoxville, Tennessee, Sept 1987.

[7] Z. Cvetanovic. The effects of problem partitioning, allocation, and granularity on the performance of multiple-processor systems. *IEEE Trans. Comp.*, C-36(4):421–432, April 1987.

[8] E. Denning Dahl. Mapping and compiled communication on the connection machine system. In *Proceedings of Distributed Memory Computer Conference*, Charleston, S.C., April 1990.

[9] F. Ercal, J. Ramanujam, and P. Sadayappan. Task allocation onto a hypercube by recursive mincut bipartitioning. In *Proceedings of the 3rd Hypercube Concurrent Computers and Applications Conference*, Pasadena, CA, January 1988.

[10] C. Farhat, N. Sobh, and K. C. Park. Dynamic finite element simulations on the connection machine. In Horst D. Simon, editor, *Proc. of the Conference on Scientific Applications of the Connection Machine, NASA Ames Research Center, Moffett Field, California*, pages 217–233, World Scientific, September 12-14, 1988.

[11] G. C. Fox, A. Kolawa, and R. Williams. The implementation of a dynamic load balancer. In M. Heath, editor, *Hypercube Multiprocessors 1987*, pages 114–121, SIAM, Knoxville, Tennessee, Sept 1987.

[12] E. N. Gilbert. Gray codes and paths on the n-cube. *The Bell System Technical Journal*, 815–826, May 1958.

[13] S. W. Hammond and T. J. Barth. An efficient massively parallel euler solver for unstructured grids. May 1990. Paper AIAA 91-0441 1991, 29th Aerospace Sciences Meeting.

[14] S.-Y. Lee and J. K. Aggarwal. A mapping strategy for parallel processing. *IEEE Trans. Comp.*, C-36(4):433–442, April 1987.

[15] M. Creon Levit. Grid communication on the connection machine: analysis, performance, and improvements. In Horst D. Simon, editor, *Proc. of the Conference on Scientific Applications of the Connection Machine, NASA Ames Research Center, Moffett Field, California*, pages 316–332, World Scientific, September 12-14, 1988.

[16] Lyle N. Long. A three-dimensional navier-stokes method for the connection machine. In Horst D. Simon, editor, *Proc. of the Conference on Scientific Applications of the Connection Machine, NASA Ames Research Center, Moffett Field, California*, pages 64–93, World Scientific, September 12-14, 1988.

[17] O. A. McBryan. Connection machine application performance. In Horst D. Simon, editor, *Proc. of the Conference on Scientific Applications of the Connection Machine, NASA Ames Research Center, Moffett Field, California*, pages 94–115, World Scientific, September 12-14, 1988.

[18] R. Miller and Q. Stout. Some graph- and image-processing algorithms for the hypercube. In M. Heath, editor, *Hypercube Multiprocessors 1987*, pages 418–425, SIAM, Knoxville, Tennessee, Sept 1987.

[19] D. A. Reed, L. M. Adams, and M. L. Patrick. Stencils and problem parititionings: their influence on the performance of multiple processor systems. *IEEE Trans. Comp.*, 36(7):845–858, July 1987.

[20] Y. Saad and M. H. Schultz. Data communication in hypercubes. *Journal of Parallel and Distributed Computing*, 6:115–135, 1989.

[21] Y. Saad and M. H. Schultz. *Topological Properties of Hybercubes.* Technical Report YALEU/DCS/RR-389, Yale University, New Haven, CT, 1985.

[22] P. Sadayappan, F. Ercal, and J. Ramanujam. *Cluster Partitioning Approaches to Mapping Parallel Programs Onto a Hybercube.* Technical

Report , Department of Computer and Information Science, Ohio State University, Columbus, Ohio, 1988. Submitted to Parallel Computing.

[23] K. Schwan, W. Bo, N. Bauman, P. Sadayappan, and F. Ercal. Mapping parallel applications to a hypercube. In M. Heath, editor, *Hypercube Multiprocessors 1987*, pages 141–151, SIAM, Knoxville, Tennessee, Sept 1987.

[24] C.-C. Shen and W.-H. Tsai. A graph matching approach to optimal task assignment in distributed computing systems using a minimax criterion. *IEEE Trans. Comp.*, 34(3):197–203, March 1985.

[25] J. B. Sinclair. Efficient computation of optimal assignments for distributed tasks. *Journal of Parallel and Distributed Computing*, 4(4):342–362, August 1987.

[26] J. S. Squire and S. M. Palais. Programming and design considerations for a highly parallel computer. In *AFIPS Cong. Proc.*, pages 395–400, 1963.

[27] Charles Tong. The preconditioned conjugate gradient on the connection machine. In Horst D. Simon, editor, *Proc. of the Conference on Scientific Applications of the Connection Machine, NASA Ames Research Center, Moffett Field, California*, pages 188–213, World Scientific, September 12-14, 1988.

[28] W. I. Williams. Load balancing and hypercubes: a preliminary look. In M. Heath, editor, *Hypercube Multiprocessors 1987*, pages 108–113, SIAM, Knoxville, Tennessee, Sept 1987.