Abstract. We discuss the implementation of several classical methods for solving elliptic partial differential equations on the hypercube multiprocessor. The methods considered are the Alternating Directions Implicit (ADI) algorithm, a direct banded Gaussian elimination method and multigrid methods. The complexity analysis of these algorithms shows that high efficiencies can be achieved by carefully assigning the data to the processors and (sometimes) resorting to more parallelizable methods. The binary reflected Gray code plays an important role for both the multigrid and the ADI algorithms.

Solving Elliptic Partial Differential Equations on the Hypercube Multiprocessor

Tony F. Chan, Youcef Saad and Martin H. Schultz

Research Report YALEU/DCS/RR-373
March 1985

The first author was supported in part by the Department of Energy under contract DE-AC02-81ER10996 and by the Army Research Office under contract DAAG-83-0177. The second and third authors were supported in part by the Office of Naval Research under grant N00014-82-K-0184 and in part by a joint study with IBM/Kingston.

1. The hypercube multiprocessor

The hypercube is a loosely coupled multiprocessor with powerful interconnection features [1, 7, 9, 12]. An $n-$cube consists of 2^n nodes that are numbered by n-bit binary numbers, from 0 to $2^n - 1$ and interconnected so that there is a link between two processors if and only if their binary representation differs by one and only one bit. For the case $n = 3$, the 8 nodes can be represented as the vertices of a three dimensional cube. One of the main advantages of the hypercube is that it imbeds many of the classical topologies such as two-dimensional or three-dimensional meshes (in fact arbitrary dimension meshes can be imbedded) [12, 7]. The diameter of an n-cube is n: to reach a node from any other node one needs to cross at most n interprocessor connections. Another appealing feature of the hypercube is its homogeneity and symmetrical properties. Unlike many other ensemble architectures, such as tree or shuffle exchange structures, no node plays a particular role. This facilitates algorithms design as well as programming. On the other hand, each node has a fan-out of n, a logarithmically increasing function of the total number of processors, and so with increasing n, there will be increasing hardware difficulties to fabricate each of these nodes.

2. Banded Gaussian elimination on the hypercube

Large symmetric banded linear systems are among the most important problems encountered when solving elliptic partial differential equations. For problems that are not too large, direct methods might be considered for solving these systems.

In this section we assume that n is even. The number of processors, denoted by k, is of the form $k = 2^n$ and we define $\kappa = \sqrt{k} = 2^{n/2}$. Consider the linear system $Ax = f$, where A is a real N x N matrix whose half-bandwidth is ν, i.e. whose total bandwidth is $2\nu - 1$. The method presented in [10], consists in first mapping a κ x κ grid into the n-cube and then perform a grid algorithm in the imbedded grid. However, instead of using only the links of the imbedded $2 - D$ grid, the more advantageous interconnection features of the hypercube are expolited when moving data.

To map a 2-D grid into an n-cube, observe that an n-cube can be viewed as a "cross-product" of two $n/2$-cubes. We can consider the n-bit binary number of any n-cube node as the result of concatenating two $n/2$-bit binary numbers, say b_i and c_j. In other words we can write any node number as $a_{ij} = b_i \hat{} c_j$, where $\hat{}$ denotes the concatenation, and b_i, c_j are the first and second $n/2$ bits of the node number. From the properties of the n-cube it can be easily seen that when b_j is fixed the resulting $2^{n/2}$ nodes obtained by varying the second part of the binary number, i.e. by varying c_i, form an $n/2$-cube, i.e. a sub-cube of the n-cube. Similarly, when we fix the second part c_i and let b_j vary we obtain a $n/2$-subcube. This defines in a natural way the n-cube as a cross product of the two $\frac{n}{2}$-cubes. We refer to a vertical plane as an $n/2$-cube defined by the set of all a_{ij} where j is fixed. A horizontal plane is defined likewise by fixing i and letting j vary.

We now assume that the matrix A is a block-banded matrix of block-bandwidth κ. Each block A_{ij} of the matrix is an $\frac{\nu}{\kappa}$ x $\frac{\nu}{\kappa}$ submatrix of A, and we have $A_{i,j} = 0$ for $|i - j| \geq \kappa$. Let us assign the submatrix A_{ij} to the node numbered $h(i)\hat{}h(j)$ where $h(i) \equiv Binary[\text{Mod}(i - 1, \kappa)]$. This is illustrated in Figure 1 where we have used a decimal encoding of $h(i)$ for simplicity. Thus the block A_{54} which is labelled by 0-3 in the figure is assigned to the node 0011.

Implementing the Gaussian elimination algorithm with this scattering of the data is straightforward. At the i-th step of the elimination process, the i-th row of the matrix, i.e. the pivot row, is distributed among the κ nodes of one horizontal plane, while the i-th column, i.e. the column of multipliers, is distributed among the nodes of one vertical plane. To perform the elimination we first compute the multipliers that will be needed to perform the linear combinations of rows. This requires moving the element a_{ii} from top to bottom and dividing the column i by it. This operation is in fact of a negligible cost. Then to perform the elimination step, we need to move the part of

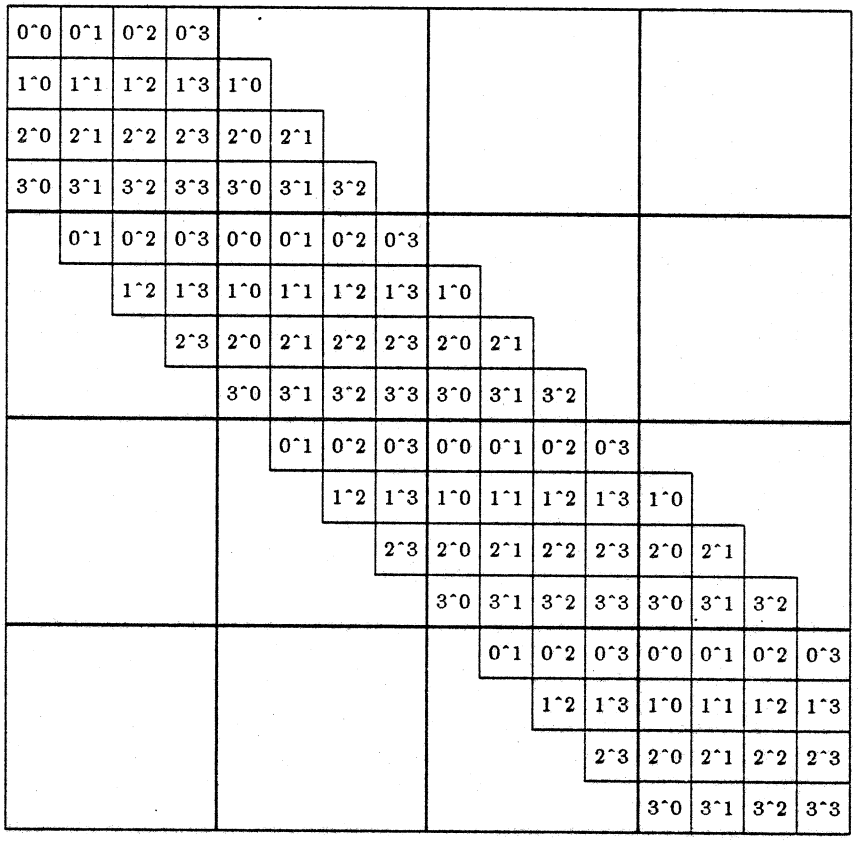

Figure 1: Block interleaving of a banded system in a 4-cube. The numbers 0, 1, 2, 3 represent the decimal encoding of the 2-bit binary numbers 00, 01, 10, 11. The symbol ^ refers to concatenation.

the pivot row located in each node T of the horizontal plane to all processors that are on the same vertical plane as T. Similarly, we need to move the multiplier blocks in the horizontal direction. After these two data permutation operations, each processor will end up with the data that is necessary to perform the i-the step of the elimination. Note that we assume that communication is not overlapped with arithmetic. A better algorithm using pipelining can be described for the case where overlapping of communication and arithmetic holds.

To analyse the complexity of this algorithm, we make the simplifying assumption that a node can be crossed simultaneously by two streams of data, one going vertically, i.e. traveling in the vertical plane of the node, while the other travels in the horizontal plane of the node.

Denoting by β the start-up time for communication in any one link, by τ the time to transfer one word excluding start-up, i.e. the inverse of the communication bandwidth, and by ω the time to perform one multiplication and one addition it can be shown that the total time required to

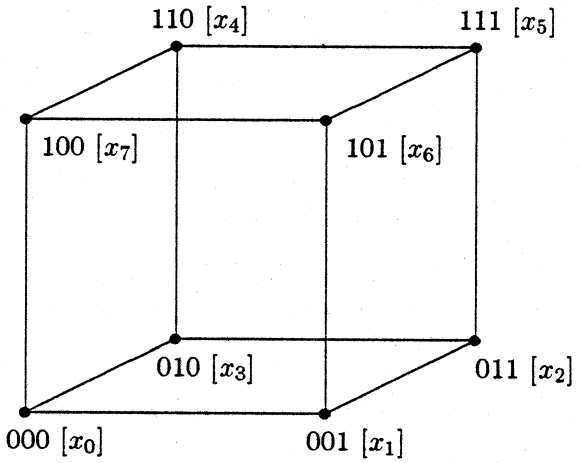

Figure 2: Grid point assignment for a one dimensional mesh of 8 points.

perform the above banded Gaussian elimination on a hypercube of $k \equiv \kappa^2$ processors where κ is a power of 2, is approximately [10]

$$t_{HBGE} \approx N\left(\frac{\nu}{\kappa}\right)^2 \omega + N\frac{\nu}{\kappa}\tau\left(1+\alpha\sqrt{2\kappa log_2\kappa}\right)^2 \tag{2.1}$$

$$\approx N\left(\frac{\nu}{\kappa}\right)^2 \omega + N\left[\frac{\nu}{\kappa}\tau + 2\sqrt{\nu\tau\beta\frac{log_2\ \kappa}{\kappa}} + \beta log_2\ \kappa\right]$$

where $\alpha = \sqrt{\frac{\beta}{2\nu\tau}}$.

Observe that as the number of processors increases, the overhead of communication increases only logarithmicly.

3. Multigrid algorithms

Multigrid methods [13] are distinguished from other elliptic problem solvers by their use of a hierarchy of coarser grids (in addition to the one on which the solution is sought) in order to improve the rate of convergence to the solution process. The basic idea is that if an iterative method (such as the Gauss-Seidel relaxation method) is used on the finest grid, convergence usually slows down after the high frequency components of the error has been annihilated and thus by transferring the problem onto a coarser grid, the lower frequencies become the high frequencies of the coarser grid and therefore can be annihilated more rapidly than on the fine grid. Employing this idea recursively, one eventually arrives at a grid that is coarse enough that the problem can be solved completely by either direct or iterative methods.

This hierarchy of grids in multigrid algorithms presents a challenge when attempting to minimize the communication overhead in a parallel implementation: although it is easy in many ensemble architectures to map the grid points of the finest grid so that neighboring grid points are mapped into neighboring processors, it is generally much more difficult to preserve this proximity property for the coarser grids. This proximity property is important in order that the time spent doing communication does not result in a serious degradation of speed-up.

In [2], it was shown that for the hypercube a mapping which has the desired property is the one based on the binary reflected Gray code. For the one-dimensional grid of points

$$x_0<x_1<x_2<\ldots<x_{2^n-1},$$

it suffices to map the point x_i to the node whose binary label is g_i, where $g_0, g_1, \ldots g_{2^n-1}$ is the binary reflected Gray code [6]. It can then easily be shown that g_i and g_{i+2^j} differ in exactly two bits, for all $j>0$ such that $i + 2^j \leq 2^n - 1$. This property means that the distance between neighboring mesh points at the finest level is one while if we work on the coarser levels the distance is exactly two. This is illustrated in Figure 2 for the case $n = 3$, i.e. for an 8 point mesh. The important fact here is that when we change levels we will not pay a heavy overhead in communication as is the case in schemes which do not preserve proximity. Higher dimensional grids can be mapped by using cross products of Gray codes [2] and present no special difficulty.

Although the distance between neighboring mesh points at any level does not exceed two, a nonnegligible gain is realized by bringing that distance from two to one by an exchange operation described in [2]. The idea of this ***exchange algorithm***, is that whenever we pass to another level, we exchange the data of some nodes so as to make the mesh points of that level reside in neighboring processors,

It is clear that one weakness of the above parallel implementations of the standard multigrid algorithm is that many nodes are left inactive during coarse grid relaxations. A natural alternative is to assign the mesh points of different levels to different nodes and have the relaxation sweeps proceed at all levels in parallel. It is shown in [2] that the resulting concurrent multigrid algorithm [4] can also be mapped with minimum communication overhead onto the hypercube with the help of Gray codes.

4. Alternating Direction Method on the hypercube

Consider the partial differential equation:

$$\frac{\partial}{\partial x}\left(a(x,y)\frac{\partial u}{\partial x}\right) + \frac{\partial}{\partial y}\left(b(x,y)\frac{\partial u}{\partial y}\right) = f$$

on the domain $(x,y) \in \Omega \equiv [0,1]$ x $[0,1]$, with the Dirichlet boundary conditions:

$$u(\bar{x},\bar{y}) = 0 \quad \forall(\bar{x},\bar{y}) \in \partial\Omega.$$

A common approach to solve the above problem is the alternating direction implicit method (ADI). First the equations are discretized with respect to the space variables x and y using a mesh of $m + 1$ points in each direction. The result is the system of equations:

$$A_x\underline{u} + B_y\underline{u} = \underline{f} \tag{4.1}$$

in which the matrices A_x and B_y represent the 3-point central difference approximations to the operators $\frac{\partial}{\partial x}(a(x,y)\frac{\partial}{\partial x})$ and $\frac{\partial}{\partial y}(b(x,y)\frac{\partial}{\partial y})$ respectively.

The ADI algorithm consists of iterating by solving (4.1) alternatively in the x and y directions as follows:

$$(I - \frac{1}{2}\rho A_x)u^{i+\frac{1}{2}} = (I + \frac{1}{2}\rho B_y)u^i \tag{4.2}$$

$$(I - \frac{1}{2}\rho B_y)u^{i+1} = (I + \frac{1}{2}\rho A_x)u^{i+\frac{1}{2}}. \tag{4.3}$$

0	1	3	2
1	3	2	0
3	2	0	1
2	0	1	3

Figure 3: Domain decomposition and assignment of the square to the 2-cube. The numbers represent the decimal encoded labels of the processors where each subsquare of the domain is assigned.

Observe that if the mesh points are ordered by lines in the x direction, then (4.2) constitutes a set of m independent tridiagonal systems which is perfectly parallellizable. It is important to note that the system (4.3) can also be recast into a set of m independent triagonal systems by *reordering the grid points* by lines, this time in the y direction. This essentially amounts to transposing the matrix of m x m grid points and is an expensive data permutation operation which is often cited as the main drawback of the Alternating Direction Method in regard to its implementation on parallel machines. The other difficulty that has been traditionally associated with ADI is that classical algorithms for solving tridiagonal systems are sequential in nature.

Since it is the tridiagonal systems that are usually troublesome we will only consider the costs of the two tridiagonal system solutions in (4.2) and (4.3). For later comparison, recall that on a single processor the time for a half step on a single processor is approximately $T_1 = 8m^2\omega$.

A simple way of implementing ADI on the hypercube is to map a ring onto the hypercube and then perform an algorithm that is tailored for the ring. Consider the sequence of processors of the hypercube whose labels form the Gray code $g_0, g_1, \ldots g_{2^n-1}$. Recall that a n-bit Gray code is a sequence of binary numbers which represent all n-bit binary numbers and so that any two consecutive elements of the sequence differ in one and only one bit. Thus the sequence of nodes of the hypercube whose labels are successively $g_0, g_1, g_2 \ldots$ form a ring imbedded in the cube.

To avoid transposing data in ADI as pointed out above, consider the special assignment of the grid points into the *ring* of processors proposed in [8] and shown in Figure 3 for the case $n = 2$ i.e. for a 4-processor cube . The numbers 0, 1, 3, 2 in the figure represent the decimal encoding of the 2-bit Gray code 00, 01, 11, 10. When iterating with ADI, the solutions of the systems (4.2) and (4.3) can be performed by a regular Gaussian elimination algorithm. Observe that all processors will be performing some work at any given stage of the iteration. Communication is greatly facilitated by the fact that all neighboring subsquares of the square are in neighboring processors and this is true in both the horizontal and vertical direction. Also the hypercube structure is not fully expolited since the hypercube is essentially regarded as a ring. A simple complexity analysis shows that the time for implementing such an algorithm on a ring of k processors is [8]

$$T(k) = 2(k-1)\beta + 4m\tau + \frac{8m^2}{k}\omega.$$

If k is small compared with m, the above formula shows that the optimal speed-up of k is nearly reached provided the communication constants β, τ are not too big. However, as the number of processors increases the communication time may become too high. In fact it is simple to show that the mimimal time that can be achieved on an arbitrarily large ring is $4(2\sqrt{\beta\omega} + 2\tau)m$, which

is linear in m. This is due to the very sequential nature of the Gaussian elimination algorithm on tridiagonal systems.

Since the hypercube also imbeds a two-dimensional grid of processors, one might consider mapping a grid algorithm into the hypercube instead of a ring algorithm, in order to reduce the execution time below the level of $O(m)$. In [7] it was shown that mapping the m^2 grid points of the square homographically into a κ x κ grid of processors, and using a substructured Gaussian Elimination [11, 3], the total time for one of the solves in ADI is of the form

$$T_G(k) \approx \alpha \frac{m^2}{k} + \delta \frac{m}{\sqrt{k}} + \gamma \sqrt{k} + Constant,$$

where α, δ, γ are constants inedependent of k. Moreover, the minimum time for an arbitrarily large processor grid is of the form $O(m^{2/3})$.

Observe that it was necessary to change algorithms in order to increase speed-up. The optimal number of processors to achieve the optimal time of $O(m^{2/3})$ is $m^{4/3}$. Many processors may theorefore be idle if the number of nodes of the hypercube is much larger than this number and one might wonder whether it is possible to still improve the above performance by resorting to alternative algorithms.

A natural candidate for solving tridiagonal systems on multiprocessors is the cyclic reduction algorithm [5]. Using the same mapping as for the 2-D grid, i.e. imbedding a grid into a hypercube and then assigning the small (m/κ) x (m/κ) squares in position (i,j) into processor (i,j) of the grid, it is clear that each of the solve phases in ADI amounts to solving in each row or column of the grid m/κ independant tridiagonal systems each of which is split into κ equal parts.

It is important to realize that when using the cyclic reduction algorithm the distance between the rows of the tridiagonal system will increase if the grid points are not assigned carefully into the nodes. The proper assignment of the rows is very similar to that used in multigrid algorithms and was described by L. Johnsson [5]. In fact the underlying problem is conceptually identical, since it consists of mapping a sequence of neighboring vertices of a graph so that not only these points are located in neighbor nodes but also every other point and every every other etc.., are at a constant distance from one another. In order to achieve this proximity preserving property observe that each column (or row) of processors is a subcube of the hypercube. Thus, one can consider using the mapping based on Gray codes, as suggested in [5] on each of the subcubes. Therefore, the κ different subcubes will solve in parallel a set of m/κ tridiagonal systems each of size m and spread in κ processors, m/κ equations per processor.

Consider the process on each of the m/κ tridiagonal systems separately. Each of the first $log_2\ (m/\kappa)$ steps of cyclic reduction requires only communication between neighboring processors in which a fixed number of elements is transmitted to neighbors namely 4 elements from each direction. The total time for arithmetic operations of the forward and backward sweep is $O(m/\kappa)$ since it is similar to that of performing the cyclic reduction algorithm on a tridiagonal system of size m/κ on a single processor. After these $log_2\ (m/\kappa)$ first steps are completed, each processor will end up with one equation of a κ x κ tridiagonal system. Cyclic reduction on such a system can be performed in time $O(log_2\ (\kappa))$ thanks to the fact that the distance between equations i and $i + 2^j$ is constant due to the assignment using Gray codes [5].

The total time for all m/κ systems is of the form $O(\frac{m}{\kappa}^2) + O(\frac{m}{\kappa} log_2\ (k))$. This simplistic implementation of the cyclic reduction can be improved in several ways [5]. Observe that for the maximum allowable value of k, $k = m^2$ we get a time of the form $O(log_2\ k)$. Therefore, a logarithmic time in m is achievable with the hypercbe topology. We emphasize, however, that the constant in front of the logarithmic term is large and that when k is smal relative to the size m of the problem, a ring or a grid method may be less time consuming: in other words if it possible to reach

a speed-up of nearly k with Gaussian elimination, which is always possible if k is small compared with m, then why bother with a speed-up of k on the more expensive cyclic reduction algorithm? The consequence of this remark is that for a given architecture selecting the best algorithm should take into consideration the parameters of the architecture and the size of the problem relative to the total number of processors.

5. Conclusion

The intrinsic topological properties of the hypercube architecture allow highly efficient implementations of parallel algorithms. In this context the role of binary reflected Gray codes is crucial, as was shown in the implementations of multgrid algorithms and the ADI algorithm.

Another interesting fact revealed by the complexity analysis of various methods, is that the "best" algorithm for solving a certain problem is no longer fixed, as it depends on the relative size of the problem to the number of processors. This was made clear in the implementation of the Alternating Direction Method in which, depending on the size of the problem (relative to k), the ring algorithm or the grid algorithm or the cyclic reduction algorithm may perform best.

References

[1] L. N. Bhuyan, D.P. Agrawal, *Generalized Hypercube and Hyperbus structures for a computer network,* IEEE Trans. Comp., C-33 (1984), pp. 323–333.

[2] T.F. Chan, Y. Saad, *Multigrid Algorithms on the Hypercube multiprocessor,* Technical Report 368, Computer Science Dept., Yale University, 1985.

[3] D. Gannon, J. van Rosendale, *On the Impact of Communication Complexity in the Design of Parallel Algorithms,* Technical Report 84-41, ICASE, 1984.

[4] ————, *Highly Parallel Multigrid Solvers for Elliptic PDE's,* Technical Report 82-36, ICASE, 1984.

[5] L. S. Johnsson, *Odd-Even cyclic reduction on ensemble architectures.,* Technical Report YALEU/CSD/RR-339, Computer Science Dept., Yale University, 1984.

[6] E.M. Reingold, J. Nievergelt, N. Deo, *Combinatorial algorithms,* Prentice Hall, New-York, 1977.

[7] Y. Saad, M.H. Schultz, *Some topological properties of the hypercube multiprocessor,* Technical Report , Computer Science Dept., Yale University, 1985. In preparation.

[8] Y. Saad, M.H. Schultz, *The alternating direction algorithm on multiprocessors,* Technical Report , Computer Science Dept., Yale University, 1985. In preparation.

[9] ————, *Communication in the the hypercube multiprocessor,* Technical Report , Computer Science Dept., Yale University, 1985. In preparation.

[10] Y. Saad, M.H. Schultz, *Direct parallel methods for solving banded linear systems,* Technical Report , Computer Science Dept., Yale University, 1985. In preparation.

[11] A.H. Sameh, D.J. Kuck, *On Stable Parallel Linear System Solvers,* JACM, 25 (1978), pp. 81–91.

[12] C.L. Seitz, *The cosmic cube,* CACM, 28 (Jan. 1985), pp. 22–33.

[13] K. Stuben and U. Trottenberg, Multi-Grid Methods: Fundamental Algorithms, Model Problem Analysis and Applications, W. Hackbusch and U. Trottenberg eds., *Multigrid Methods,* Springer Verlag, Berlin, 1982.