**Abstract.** This paper examines several ways of implementing multigrid algorithms on the hypercube multiprocessor. We consider both the standard multigrid algorithms and a concurrent version proposed by Gannon and Van Rosendale. We present several mappings of the mesh points onto the nodes of the cube. The main property of these mappings, which are based on binary reflected Gray codes, is that the distance between neighboring grid points remains constant from one grid level to another. This results in a communication effective implementation of multigrid algorithms on the hypercube multiprocessor.

# Multigrid Algorithms on the Hypercube Multiprocessor

Tony F. Chan and Youcef Saad

Research Report YALEU/DCS/RR-368
March 1986

This work was supported in part by the Office of Naval Research under grant N00014-82-K-0184, by IBM/Kingston under a joint study grant, by the Department of Energy under contract DE-AC02-81ER10996 and by the Army Research Office under contract DAAG-83-0177.

**Keywords:** *Multigrid algorithms, hybercube multiprocessor, Gray codes, parallel computing, partial differential equations, domain decomposition.*

## 1. Introduction

In the recent years there has been a growing interest in the design of parallel algorithms for solving various mathematical problems that arise in scientific computing. One of the major objectives is to construct algorithms, either by restructuring existing ones designed for sequential machines or by inventing new algorithms, that can efficiently exploit the parallelism available in a given machine architecture. This task is often complicated by the need for efficient communication between the processors in a parallel architecture. This communication overhead, which is absent in analyses of sequential algorithms, can be a bottleneck if not handled efficiently. Thus, a successful implementation of an algorithm in a parallel architecture must not only decompose a given problem into appropriate smaller pieces for each individual processors to process, but also must arrange this assignment so that the communication overhead is kept at a minimum.

In this paper, we study in detail a particular algorithm-architecture combination: the implementation of the class of multigrid algorithms for solving elliptic partial differential equations on the hypercube multiprocessors. Multigrid algorithms are among the most efficient methods for solving partial differential equations. In addition to their immediate effects on applications, multigrid algorithms are also of theoretical interest because it can be proven that they can compute a solution to truncation error accuracy in time proportional to the number of unknowns. This optimality result, together with the fact that many aspects of the multigrid algorithms are highly parallelizable, makes it natural to consider parallel implementation of multigrid algorithms. There have been a few papers published on this topic: Grosch [5, 6], Brandt [2] Gannon and Van Rosendale [4] and Chan and Schreiber [3].

The hierarchy of grids in multigrid algorithms presents a special challenge in minimizing the communication overhead in a parallel implementation. For even though it is possible to map the grid points of the finest grid onto many architectures such that neighboring grid points are mapped into neighboring processors, it is generally much more difficult to preserve this proximity property for the coarser grids required in the multigrid algorithms. Grosch [6] asserts that a perfectlty shufled nearest neighbor array is a suitable architecture. Gannon and Van Rosendale [4] propose a concurrent variation of the standard multigrid algorithms and consider implementations on mesh-connected arrays, permutation networks and direct VLSI imbeddings. In this paper, we consider the hypercube multiprocessor [12]. Of the many parallel architectures that have been proposed in the past few years, the hypercube multiprocessor is one of a few that are commercially available and one for which there has been some experience on its usage. We show that the hypercube architecture is also ideally suited to the implementation of multigrid algorithms, especially regarding the communication overhead.

In Section 2, we briefly describe the architecture of the hypercube and in Section 3 we describe the essential features of the multigrid algorithms with regard to their parallel implementations. In Section 4, we consider various mappings of the grid structures behind the multigrid algorithms into the nodes of the hypercube in a communication-effective way. In Section 5, we briefly analyze the arithmetic and communication complexity of the proposed parallel algorithms and we present some concluding remarks in Section 6.

## 2. The hypercube multiprocessor

The hypercube is a multiprocessor array with powerful interconnection features introduced under different names (Cosmic cube, boolean $n$-cube, $n$-cube, etc.. See [1, 7, 12] for references). 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, see Figure 1.

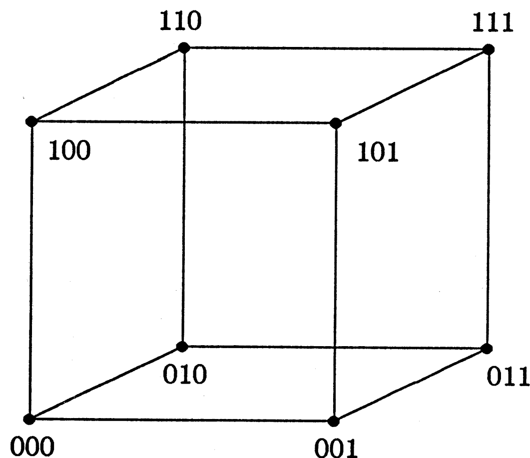



**Figure 1:** 3-D view of the 3-cube.

There are many reasons for the recent growing interest in the hypercube configuration. One of them, perhaps the most important, is that the hypercube imbeds many of the classical topologies such as two-dimensional or three-dimensional meshes (in fact arbitrary dimension meshes can be imbedded) [12, 11]. 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.

Seitz [12] describes a real hypercube machine which is utilized at Caltech, and for the first time presents some details on software and applications of a machine based on the hypercube architecture. Saad and Schultz [11] analyze the intrinsic topological properties of the hypercube regarded as a graph, and propose several algorithms for transferring data between its nodes[10]. A generalization of the hypercube topology has been proposed by Bhuyan and Agrawal [1].

## 3. Multigrid Algorithms

In this section, we describe briefly the essential features of the multigrid algorithms that are relevant to their implementation on the hypercube.

Multigrid algorithms are generally used to solve continuous problems defined by differential or integral equations on a given region in space. To facilitate our presentation, we restrict our attention to the important case of the solution of linear elliptic differential equation:

$$Lu = f$$

on a $d$-dimensional square in $R^d$. Such problems are usually solved approximately by discretizing the problem via a $d$-dimensional grid, say with $m$ grid points in each coordinate direction. After applying an appropriate discretization technique (e.g. finite difference or finite element), the differential equation is transformed into a set of linear algebraic equations of size $m^d$. Many different methods can be used to solve this set of algebraic equations, e.g. Gaussian elimination or conjugate gradient methods. The class of multigrid methods is distinguished from others by their use of a hierarchy of coarser grids, in addition to the one on which the solution is sought, in order to improve efficiency. 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 have been annihilated. 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.

To describe the method more precisely, we denote the hierarchy of grids by $G^i$, with $G^1$ being the coarsest grid, the discretization of the elliptic operator on $G^i$ by $L^i$ and the corresponding solution and right-hand-side by $u^i$ and $f^i$, the operation of projection from $G^i$ onto a coarser grid $G^{i-1}$ by $P^i$ and the interpolation back onto $G^i$ from a coarser grid $G^{i-1}$ by $I^i$. With this notation, the classical multigrid algorithm can now be described succintly as:

**ALGORITHM MG($L^i, f^i, u^i$)**

Comment: Solve $L^i u^i = f^i$.
    IF $G^i$ is the coarsest grid THEN
        Solve $L^i\ u^i\ =\ f^i$ exactly.
    ELSE
        Perform $s$ smoothing iterations.
        Compute the residual :
            $r^i\ :=\ f^i\ -\ L^i\ u^i.$
        Project the residual onto the next coarser grid :
            $g^{i-1}\ :=\ P^i\ r^i.$
        Solve the coarse grid problem $L^{i-1}v^{i-1} = g^{i-1}$ recursively by $c$ iterations of MG:
            Repeat $c$ times:
                MG($L^{i-1}$, $g^{i-1}$, $v^{i-1}$), starting with $v^{i-1} := 0$.
        Interpolate the correction from $G^{i-1}$ back to $G^i$ :
            $u^i\ :=\ u^i\ +\ I^i\ v^{i-1}.$
    ENDIF

The parameter $c$ controls the flow of the algorithm, in particular the frequency with which the various grids are visited. The larger $c$ is, the more time the algorithm spends on the coarser grids. The most common cases of $c = 1$ and $c = 2$ are often referred to as the V-cycle and the W-cycle respectively.

There are many variants of the above multigrid algorithms, with different numerical properties, e.g. ones more suitable for nonlinear problems. The major steps, however, remain the same as in the above algorithm. In particular, the computations on a given grid and the transfers between

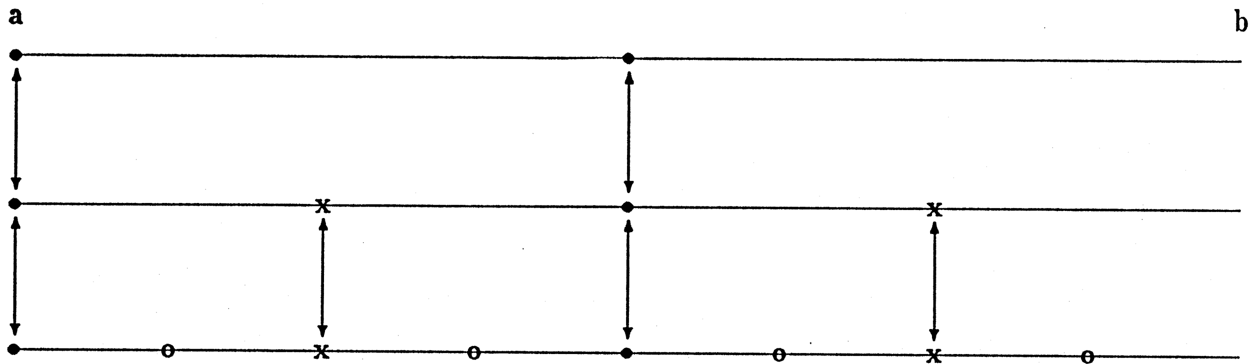



**Figure 2:** One dimensional pyramid.

neighboring grids as illustrated in the algorithm are typical. It is therefore sufficient, for the purpose of implementation on parallel systems, to consider the above algorithm.

## 4. Parallism in Multigrid Algorithms

Now we consider implemention on a parallel machine. The major steps of Algorithm MG are:

1. Smoothing;
2. Computation of the residual;
3. Projection onto a coarser grid;
4. Interpolation from a coarser grid.

The first two are *intra*-grid operations while the last two are *inter*-grid ones. There are generally two opportunities for exploiting parallelism: performing operations on a given grid in parallel and performing operations on all grids in parallel. The first approach is more straightforward. Since the operator $L^i$ is usually a local operator, both the smoother (e.g. Jacobi or red-black SOR) and the computation of the residual can be computed in parallel. Similarly, the operators $P^i$ and $I^i$ are also local operators and thus the projection and interpolation steps can also be performed in parallel. Such parallel multigrid algorithms have been considered by Grosch [5, 6], Brandt [2], and Chan and Schreiber [3]. The major drawback of this class of parallel multigrid methods is that at any given instance, only one grid is active and therefore the processors assigned to the other grids are idle. In order to have more efficient processor utilization, Gannon and Van Rosendale [4] considered a *concurrent* multigrid algorithm in which all processors on all grids are active all the time. Note that this approach represents a deviation from the standard multigrid algorithm and the extra operations must be designed carefully so as to improve and not diminish the rate of convergence.

For simplicity, we assume throughout this paper that the number of grid points in each coordinate direction on $G^i$, denoted by $m_i$, is given by $m_i = 2^{i-1}m_1$ and that the $L^i$'s are nearest-neighbor stencils (e.g. the standard second order centered difference stencil for the Laplacian). For such problems, the natural architecture for the multigrid algorithm is the *pyramid*, a one dimensional version of which is shown in Figure 2. The ' •' points correspond to the coarsest level, the 'x' points to the second level and the 'o' points to the third and finest level. This topology completely describes the communication pattern between each grid point in the grid hierarchy. In other words, each connection in the pyramid represents a data flow path required by the algorithm. Unfortunately, for two or three dimensional elliptic problems, the corresponding pyramid is three dimensional

and therefore cannot be easily implemented in two dimensional architectures without creating long communication delays. For example, long wires are required in a direct VLSI implementation.

In this paper we are primarily interested in ensemble architectures consisting of a large number of identical processors interconneted to one another according to some convenient pattern. There is no global memory and no global bus. Although some designs may incoporate a global bus this does not constitute the main way of intercommunication. One advantage of such types of architectures is the simplicity of their design. The nodes are identical and can be produced at relatively low cost. Communication in ensemble architectures is done by message passing: data or code are transferred from processor $A$ to processor $B$ by traveling across a sequence of nearest neighbor nodes starting with node $A$ and ending with $B$. For a given mapping of the pyramid onto the ensemble architecture, grid points that are direct neighbors on the pyramid may be assigned to processors that are far away from each other which may result in an increase in communication overhead. Therefore, given a machine on which the multigrid algorithm is to be implemented, the first task is to map the pyramid architecture into the architecture of the given machine in such a way that the communication paths represented in the pyramid are as *short* as possible on the machine. Notice that these communication paths involve both grid points on the same grid and on other grids. While preserving locality on the finest grid for the *intra*-grid communications is relatively easy on many parallel architectures, the *inter*-grid communications and the need to preserve locality on coarser grids often present difficulties because neighboring grid points on the coarser grids may not be mapped onto neighboring processors. Grosch[5, 6] proposes a perfectly shuffled nearest neighbor array to handle this problem. Gannon and Van Rosendale [4] consider a concurrent multigrid algorithm and suggested mappings for various parallel architectures, such as the nearest-neighbor mesh-connected arrays, the mesh-shuffle connected network and the omega network, but most of their mappings do not succeed in preserving the locality of coarser grids communication paths in the pyramid. In the next section, we will consider mappings of the pyramid onto the hypercube and show that it is possible to map the pyramid into the hypercube so that all communication paths in the pyramid are mapped into communication paths in the hypercube with length bounded by 2.

In designing such mappings, it is important to note that if the concurrent multigrid algorithm is to be implemented, then each grid point of the pyramid must be mapped into distinct processors. If only the standard multigrid algorithm is to be implemented, then points on different grids can be mapped into the same processor. This consideration may impose different constraints on the mapping.

## 5. Mesh to hypercube mappings

In this section, we are concerned with the problem of mapping general grids in 1-D, 2-D or 3-D into the hypercube. Clearly there are numerous ways to map grid points into the $2^n$ processors in general. For reasons of communication efficiency we are interested in those mappings that have the property that any two neighboring points of the grid belong to neighboring processors. This is a fairly easy problem to which we will bring a solution in Section 4.1.

As mentioned in the introduction, a more critical requirement for the multigrid algorithms is that two grid points that are neighbors in the finer grid should remain either neighbors or close to each other when they are considered as grid points of a coarse grid. In other words it should not only be inexpensive to access a neighboring point of the same grid but also a neighboring point of a grid of a different level.

### 5.1. One dimensional meshes and Gray codes

Consider the problem of assigning $N = 2^n$ mesh points which discretize some real interval, into an $n-$cube in such a way as to preserve the proximity property, i.e. so that any two neighboring

mesh points belong to neighboring nodes. Another way of viewing the problem is that we are seeking a path of length $N = 2^n$ that crosses each node once and only once. In graph theory terminology, we are looking for a Hamiltonian path.

If we number the nodes of the hypercube according to its definition, i.e. so that two neighboring nodes differ by one and only one bit, a Hamiltonian path simply represesents a sequence of $n$-bit binary numbers such that any two successive numbers have only one different bit and such that all $n$-bit binary numbers are represented in the sequence. Binary sequences with these properties are called Gray codes, and have been extensively studied in coding theory (c.f. [9].)

One of the simplest methods for generating Gray codes is as follows. Let $G_i = \{g_0, g_2, \ldots, g_{2^i-1}\}$ be the $i$-bit Gray code, with $G_1 \equiv \{0, 1\}$ and denote by $G_i^R$ the sequence obtained from $G_i$ by reversing its order, and by $0G_i$ (resp. $1G_i$) the sequence obtained from $G_i$ by prefixing a zero (resp. a one) to each element of the sequence. Then Gray codes of arbitrary order can be generated by the recursion:

$$G_{i+1} = \{0G_i, 1G_i^R\}, \quad i = 1, \ldots \tag{5.1}$$

For example,

$$G_2 = \{00, 01, 11, 10\},$$
$$G_3 = \{000, 001, 011, 010, 110, 111, 101, 100\}. \tag{5.2}$$

There are many possible Gray codes; the particular one generated by the above algorithm is called a *binary reflected Gray code.*

We now describe a second algorithm for generating the binary reflected Gray code which will turn out to be more useful later. Given the $n$-bit binary reflected Gray code

$$G_n = \{g_0, g_1, \ldots g_{2^n-1}\},$$

one can generate the $(n + 1)$-bit Gray-code as follows:

$$G_{n+1} = \{g_0\,0, g_0\,1, g_1\,1, g_1\,0, g_2\,0, g_2\,1, g_3\,1, g_3\,0, \ldots g_i\,0, g_i 1, g_{i+1} 1, g_{i+1}\,0, \ldots\}.$$

In other words, the general pattern for generating $G_{n+1}$ is to expand any two successive nodes $a, b$ of $G_n$ into the nodes $a0, a1, b1, b0$. It can be proved that this algorithm indeed generates the binary reflected Gray code [9].

We now return to our original problem of assigning the $N = 2^n$ mesh-points of a discretized interval to the $2^n$ nodes of an $n$-cube. A desirable property is that the mesh points are assigned to neighboring processors, in order to achieve locality in communication. According to our previous discussion on Gray codes, it is clear that the solution is to assign successive nodes of the mesh to the succesive nodes of a Gray code sequence, i.e. to the nodes of the cube whose binary numbers form a Gray code sequence. Thus, if there are 8 grid points numbered from 0 to 7 such that

$$x_0 < x_1 < x_2 < x_3 < x_4 < x_5 < x_6 < x_7,$$

they can be assigned successively to the nodes of the sequence (5.2) in that order; see Figure 3.

**5.2. Meshes in higher dimensional spaces**

One of the most attractive properties of the hypercube as a network, is that it can imbed meshes of arbitrary dimensions. This may in fact be the main reason for the success of the hypercube as a multiprocessor. Consider an $m_1 \times m_2 \ldots \times m_d$ mesh in the $d$-dimensional space $R^d$ and assume that the mesh size in each direction is a power of 2, i.e. it is such that $m_i = 2^{p_i}$. Let $n = p_1 + p_2 + \ldots p_d$ and consider the problem of mapping the mesh points into the $n$-cube, one mesh point per node.

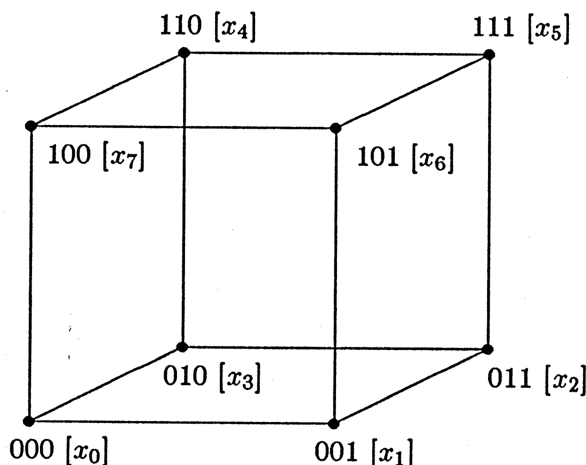



**Figure 3:** Grid point assignment for a one dimensional mesh of 8 points.

Observe that we have just enough nodes to accomodate one mesh point per node. We show next how to extend the ideas of the previous section to more than one dimension.

This is best illustrated by an example. Consider an 8 x 4 mesh of a 2-dimensional problem, i.e. $d = 2, p_1 = 3, p_2 = 2, n = p_1 + p_2 = 5$. A binary number $A$ of any node of the 5-cube can be regarded as consisting of two parts: its first three bits and its last two bits, which we write in the form

$$A = b_1 b_2 b_3 \; c_1 c_2$$

where $b_i$ and $c_j$ are binary bits. It is clear from the definition of an $n$-cube that when the last two bits are fixed, the resulting $2^{p_1}$ nodes form a $p_1$-cube (with $p_1 = 3$). Likewise, whenever we fix the first three bits we obtain a $p_2$-cube. The mapping then becomes clear. Choosing a 3-bit Gray code for the $x$-direction and a 2-bit Gray code for the $y$-direction, the point $(x_i, y_j)$ of the mesh is assigned to the node $b_1 b_2 b_3 \; c_1 c_2$ where $b_1 b_2 b_3$ is the 3-bit Gray code for $x_i$ while $c_1 c_2$ is the 2-bit Gray code for $y_j$. We refer to the mapping $(x_i, y_j) \longrightarrow b_1 b_2 b_3 \; c_1 c_2$ as the *cross product* of the two mappings $x_i \longrightarrow b_1 b_2 b_3$ and $y_j \longrightarrow c_1 c_2$. This mapping is illustrated in Figure 4 where the binary node number of any grid-point is obtained by concatenating its binary $x$-coordinate and its binary $y$-coordinate. Note that any one column of grid points forms a Gray code and any one row of nodes forms a Gray code.

Generalizations to higher dimensions are straightforward. We can state the following general theorem.

**Theorem 5.1.** *Any $m_1$ x $m_2$... x $m_d$ mesh in the d-dimensional space $R^d$, where $m_i = 2^{p_i}$, can be mapped into an n-cube where $n = p_1 + p_2 + \ldots p_d$, with the proximity property preserved. The mapping of the grid points is the cross product $G_1$ x $G_2$ x ... x $G_d$ where $G_i$ is any one-dimensional Gray-code mapping of the $m_i$ points in the $i^{th}$ coordinate direction.*

### 5.3. One dimensional hierarchical Gray codes for standard multigrid methods

In the previous sections we have not addressed some of the important aspects of multi-level meshes that appear in multigrid methods. In particular, a crucial issue we would like to examine in this section is to find an efficient mapping of the mesh points of the different levels of refinement

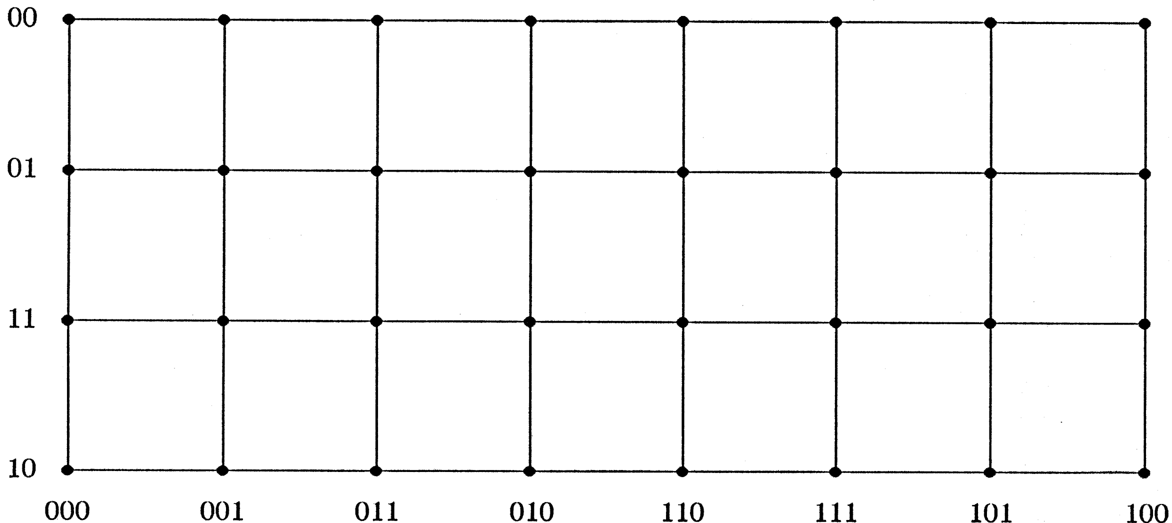



**Figure 4:** Two dimensional Gray-code for an 8 x 4 grid.

into a cube. In this section we consider only the classical multigrid approach. The concurrent approach will be examined in the next section.

As an illustration consider a one-dimensional mesh on the interval $[a, b]$ with the periodic boundary condition $u(b) = u(a)$ as is illustrated in Figure 2, which shows three different levels of meshes discretizing the interval, starting with two points. It is desirable to assign the mesh points of the finest mesh so that not only its neighboring points are assigned to neighboring processors but also so that the points of the coarser meshes be not too far from each other. The reason for this is clear: we wish to minimize the intercommunication not only when iterating at the finest level but at the coarser levels as well.

For example, we would wish to map the 8 points of the bottom mesh of Figure 2 into a 3-cube so that the points of the submeshes shown in levels 1 and 2 remain not too far from each other. Ideally, we would like them to be neighbors. However, this turns out to be impossible. This is easy to see because the binary representations of both $x_0$ and $x_2$ must differ from that of $x_1$ in one bit. Since $x_0$ and $x_2$ are distinct nodes, the bit in which they differ from $x_1$ must be different and therefore the distance between $x_0$ and $x_2$ is bigger than 1. More generally, if it were possible to find such a mapping it would mean that there are odd cycles in an $n$-cube, which is impossible [11]. For example, if we could have a connection between the nodes containing $x_0$ and $x_2$ we would have the cycle $x_0, x_1, x_2, x_0$ which is of length 3.

The next best distance we can hope for is two. Observe that in order for the nearest neighbor mesh connections to be maintained on the finest level, the node assignment of the finest mesh must be a Gray code. A Gray code which has the desired property that the distance between any neighboring points of coarser submeshes is constant and equal to two will be referred to as a *hierarchical* Gray-code.

It is important to note that not every Gray code is hierarchical. Consider the following Gray code (obtained from the cross-product of two 2-bit Gray codes):

$$\{0000, 0001, 0011, 0010, 0110, 0111, 0101, 1101, 1111, 1110, 1010, 1011, 1001, 1000, 1100, 0100\}.$$

For this Gray code, the coarsest submesh would be assigned to the nodes $\{0000, 1111\}$ which are at a distance of 4 from one another.

Fortunately, the binary reflected Gray code described in Section 5.1 is hierarchical. More precisely one can prove the following property of the binary reflected Gray codes.

**Theorem 5.2.** *Let $G_n \equiv (g_0, g_1, \ldots, g_{2^n-1})$ be the sequence of n-bit binary numbers of the binary reflected Gray code. Then $g_i$ and $g_{i+2^j}$ differ in exactly two bits for all $j>0$ such that $i+2^j \leq 2^n-1$.*

*Proof.* See [9] and [7].

∎

We note that a similar problem was considered by L. Johnsson[7] in the context of odd-even cyclic reduction for solving tridiagonal systems.

As a consequence of the above theorem, if we map the mesh onto the cube using the binary reflected Gray code, 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. 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. This can be easily verified for an 8-point grid on the 3-cube as is illustrated in Figure 3. As a further illustration, if the total number of mesh points of the finest level is 16, we should assign the points

$$x_0 < x_2 < \ldots < x_{15}$$

successively to the nodes

$$0000, 0001, 0011, 0010, 0110, 0111, 0101, 0100, 1100, 1101, 1111, 1110, 1010, 1011, 1001, 1000.$$

We can verify on this example that the 8 points of the next coarser level, i.e. the points

$$x_0, x_2, x_4, x_6, \ldots x_{14},$$

are assigned to nodes that are distant by two, namely the nodes:

$$0000, 0011, 0110, 0101, 1100, 1111, 1010, 1001.$$

The same property can again be verified for the next level.

### 5.4. An Exchange Algorithm

Although it is satisfactory that the distance between neighboring mesh points at any level does not exceed two, it would be a nonnegligible gain to bring that distance from two to one by some exchange operation. When we pass to another level, we can exchange the data of some nodes, so as to make the mesh points of that level reside in neighboring processors. Relaxation on this level is then performed with improved efficiency since communication costs are divided by two. After the sweeps are done we can permute the data back to their initial assignments. This is effective if sufficiently many relaxation sweeps on each level are performed in order to pay off for the initial and the final data exchanges.

To explain how the exchange will be done, we need to use the second algorithm for generating the binary reflected Gray code described in Section 4.1. Let us start with the $n$-bit binary reflected Gray code

$$G_n = \{g_0, g_1, \ldots g_{2^n-1}\},$$

and generate the $(n+1)$-bit Gray-code as follows:

$$G_{n+1} = \{g_0\,0, g_0\,1, g_1\,1, g_1\,0, g_2\,0, g_2\,1, g_3\,1, g_3\,0, \ldots g_i\,0, g_i 1, g_{i+1} 1, g_{i+1}\,0, \ldots\}.$$

Recall that the general pattern for generating $G_{n+1}$ is from any two successive nodes $a, b$ to insert $a0, a1, b1, b0$. Consider now every other node, which corresponds to the next coarser grid. We get the sequence

$$\{g_0\,0, g_1\,1, g_2\,0, g_3\,1, \ldots g_{i-1}\,0, g_i 1, g_{i+1}\,0, \ldots\} \tag{5.3}$$

As shown before, every two successive nodes differ by two bits: the last bit and the bit that differs between $g_i$ and $g_{i+1}$. Now suppose that in the sequence $G_{n+1}$ we permuted the pairs of the form $g_i 1, g_i 0$ wherever they occur (these occur only for $i$ odd). We get the transformed sequence:

$$\hat{G}_{n+1} = \{g_0\,0, g_0\,1, g_1\,0, g_1\,1, g_2\,0, g_2\,1, g_3\,0, g_3\,1, \ldots g_i\,0, g_i 1, g_{i+1} 0, g_{i+1}\,1, \ldots\}.$$

In other words the general pattern is now $a0, a1, b0, b1$ where $a$ and $b$ are two successive elements of $G_n$. Taking every other node of $\hat{G}_{n+1}$, we get precisely the subsequence of $G_{n+1}$ terminating with a zero, i.e.

$$\hat{G}_n = \{g_0\,0, g_1\,0, g_2\,0, \ldots g_i\,0 \ldots g_{2^n-1}\,0\},$$

which apart from the last bit is the $n$-bit Gray code. Therefore, if we send the data (i.e the residual) in processor $g_i 1$ for $i$ odd, to its neighbor $g_i 0$ before we visit a coarser grid, all neighboring grid points of the next coarser grid will then reside in neighboring processors. Finally, to show that this property is still true at even coarser levels, observe that the mesh-node assignment represented by $\hat{G}_n$ can be considered as the binary reflected Gray code $G_n$ applied to an $n-$dimensional subcube — the half of the $(n+1)-$cube whose node labels have 0 in the last bit of its binary representation. Therefore, by removing the last zero bit of $\hat{G}_n$ the argument used for $G_{n+1}$ can be used recursively to show that, by similar exchanges of data prior to visiting coarser meshes, all neighboring grid points on all levels are mapped to neighboring processor nodes.

Note that a similar exchange also occurs in the reverse direction when coming back from a coarse grid to the next finer grid. This time the correction is sent instead of the residual.

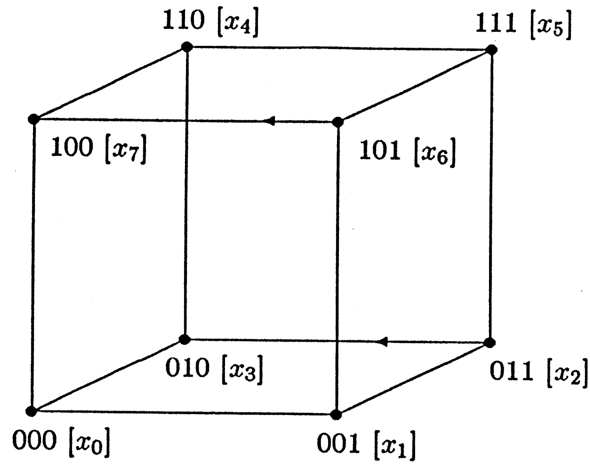
Numbering the levels by $l$, $l = 0, 1, \ldots$ starting from the finest to coarsest level, we can summarize the property of our algorithm in the following proposition.

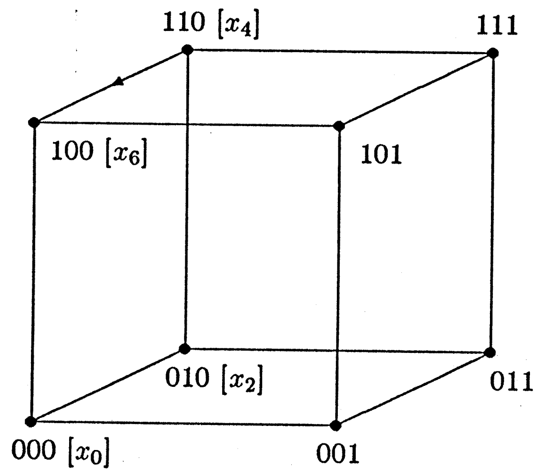
**Proposition 5.1.** *By transferring the residual from the nodes $g_i 10^l$, $i$ odd, of level $l$ to the node $g_i 0^{l+1}$, and the correction in the reverse direction, relaxation sweeps only require communication between neighboring nodes.*

This exchange method is illustrated in Figure 5.

**5.5. One dimensional hierarchical Gray codes for concurrent algorithms**

As was mentioned earlier, the previous parallel implementation of the standard multigrid algorithm leaves many nodes inactive during coarse grid relaxations. In fact, as is shown in [3], the *efficiency* of the standard algorithm decreases like $(\log_2 m)^{-1}$ as the grid size $m$ increases. An intuitive reason for this is that while the finer grids get a higher proportion of the available processors, they are not proportionally active more often than the coarser grids. Therefore, a natural strategy is to assign the mesh points of different levels to different nodes and have the



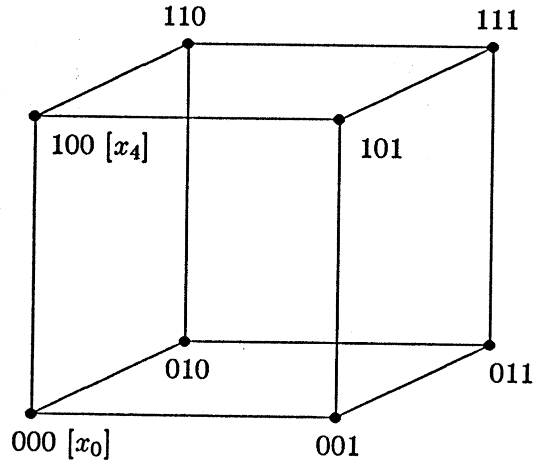




**Figure 5:** The exchange algorithm. The picture shows the transfers from fine to coarse grids only.

relaxation sweeps proceed at all levels in parallel. There are two types of communication for such a concurrent algorithm. The first is as before between the nodes containing the mesh points of the same level. Since there must be interaction between the mesh points of different levels, another type of communication is between the nodes holding data of different levels. We would like the two types of communication to be fast if possible.

Consider the example of Figure 2 where we have a total a 14 points, 8 for the finest (third) level, 4 for the second level and, 2 for the coarsest level, to assign to the 16 nodes of a 4-cube. Note that more generally the total number of mesh points at all levels is of the form $(2^{n+1}-1)m_1$ where $m_1$ is the number of points of the coarsest level and $n$ is the number of refinements. We would like to assign these points to the $2^{n+1}$ nodes of an $(n+1)-$cube. The 4-cube is split in two subcubes of lower dimensions, as represented in Figure 6. The first subcube, consisting of all the nodes whose labels have the bit one as their last bit, will contain the 8 points of the finest mesh. For the purpose of having neighboring mesh points assigned to neighboring nodes, we will assign the points

$$x_0 \leq x_1 \leq \ldots \leq x_{2^n-1}$$

successively to the nodes numbered $g_i$ 1 where $g_i, i = 0, \ldots 2^n - 1$, is the sequence of the binary reflected $n$-bit Gray code.

Let the $x'_{2i}, i = 0, 1, \ldots 2^{n-1} - 1$ denote the grid points of the next coarser grid. Note that we use the same subscript but the prime indicates that the same physical point is now represented by two different points on the pyramid, namely $x_{2i}$ in the fine level and $x'_{2i}$ in the coarse level. The problem at this point is to assign the coarse mesh points $x'_{2i}, i = 0, 1, \ldots 2^{n-1} - 1$, so that

- the inter-level communication is inexpensive
- the grid-points of the same level are held in neighboring nodes.

Note that the remaining points, i.e. the grid points of all the coarser levels, will now be assigned to the nodes of the form $g_i$ 0 where again $g_i, i = 0, 1 \ldots 2^n - 1$, is the $n$-bit binary reflected Gray code. We assign the points of the next coarse grid successively to the nodes $g'_i$ 10 where $g'_i, i = 0, 1, \ldots, 2^{n-1} - 1$, is the sequence of $(n-1)$-bit binary reflected Gray code. This is illustrated in Figure 6 for $n = 4$ where we have assigned the 14 points of the example of Figure 2.

Consider the sequence of every other node in the finest grid:

$$g_0 1 , g_2 1 , g_4 1 , \ldots$$

The points $x_0, x_2, x_4 \ldots$ held by these nodes correspond physically to the same points as the coarse points $x'_0, x'_2, x'_4 \ldots$ that are held by the nodes

$$g'_0 10 , g'_1 10 , g'_2 10 \ldots.$$

Therefore, we want the distance between the nodes $g'_i 10$ and $g_{2i} 1$ to be small. In fact as we now show this distance alternates between one and two. To prove this, we need to use the second algorithm for generating binary reflected Gray codes, described in Section 5.1. Appending the bit one to the sequence $g_i$ of the $n$-bit Gray code obtained by that algorithm from the $(n-1)$-bit Gray code sequence $g'_i$, we find the sequence:

$$g'_0 01 , g'_0 11 , g'_1 11 , g'_1 01 , g'_2 01 , g'_2 11 , g'_3 11 , g'_3 01 , \ldots , g'_i 01 , g'_i 11 , g'_{i+1} 11 , g'_{i+1} 01 , \ldots$$

This is precisely the Gray code for the finest mesh. Therefore the subsequence of every other node, i.e. the sequence of nodes of the finest level which hold the points $x_{2i}, i = 0, 2, \ldots 2^{n-1}$, is

$$g'_0 01 , g'_1 11 , g'_2 01 , g'_3 11 , \ldots , g'_i 01 , g'_{i+1} 11 , \ldots$$

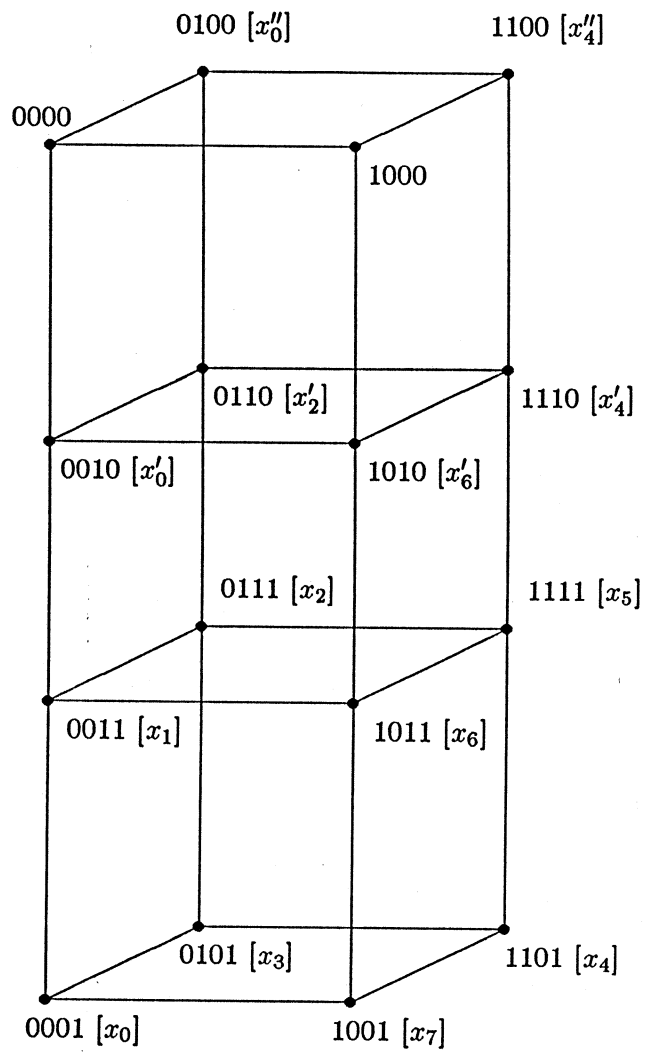



**Figure 6:** Mapping of three levels of grid points into a 4-cube. The links between the nodes of the lower and upper horizontal planes are omitted.

The nodes $g_i'11$ are directly connected to their counterparts $g_i'10$ containing the points $x_{2i}'$ of the coarser grid, i.e. the distance between them is one. However, the nodes of the form $g_i'01$ are at distance 2 from their counterparts $g_i'10$ containing the points $x_{2i}'$ of the coarser grid, the path of length 2 being $g_i'01 \rightarrow g_i'11 \rightarrow g_i'10$.

To assign the grid points $x_{4i}'', i = 0, 1, \ldots, 2^{n-2} - 1$, of the third level, observe that by removing the ending zeroes from the sequence $g_i'10$ we are back to the previous situation with $n$ replaced by $n - 1$. Hence, we assign the points of the third grid successively to the nodes $g_i''100$, where $g_i'', i = 0, 1, \ldots 2^{n-2} - 1$, is the $(n-2)$-bit binary reflected Gray code. More generally, numbering the levels from finest to coarsest by $l = 0, 1, \ldots$, we assign the grid points of level number $l$ successively to the nodes $g^{(l)}10^l$ where the power refers to concatenation and where $g_i^{(l)}, i = 0, 1, \ldots 2^{n-l} - 1$, is the $(n - l)$-bit binary reflected Gray code. The proof that the distance between interlevel grid points are at distance one or two is straightforward.

**Proposition 5.2.** *The mapping*

$$x_{2^l i}^{(l)} \longrightarrow g_i^{(l)} 10^l$$

*is such that the distance between neighboring points on the same level is one and the distance between the grid points $x_{2^l i}^{(l)}$ of level number $l$ and $x_{2^l i}^{(l-1)}$ of level number $(l-1)$ is at most two.*

**5.6. Higher Dimensional Problems**

As a result of Theorem 4.1, the mappings that we have introduced so far extend straightforwardly to higher dimensional problems. For mapping the grid of these problems, one uses cross products of one-dimensional binary reflected Gray code mappings. A higher dimensional submesh required by the multigrid algorithm is precisely the cross product of the corresponding one dimensional submeshes. Therefore, the proximity preserving property of one dimensional binary reflected Gray codes ensures that the same property holds for higher dimensional meshes.

However, there is a difficulty with the implementation of the *concurrent* algorithm. This stems from the fact that, for higher dimensional problems, the total number of points on the pyramid is not close to a power of 2. This misfit with the number of nodes in a cube leaves many nodes unassigned. To be specific, consider a $d$-dimensional problem whose finest grid has $2^n$ points to be assigned to the nodes of an $(n + 1)$−cube. The total number of points $N_p$ on the pyramid is given by

$$N_p = 2^n + 2^{n-d} + \cdots + 1 = 2^n \left( \frac{1 - 2^{-nd}}{1 - 2^{-d}} \right) \approx \frac{2^n}{1 - 2^{-d}}.$$

Therefore, the fraction of the total number of nodes that are idle is

$$\frac{2^{n+1} - N_p}{2^{n+1}} \approx \frac{1 - 2^{-d+1}}{2 - 2^{-d+1}}.$$

Thus, for $d = 2$, a third of the nodes of the $(n+1)$−cube is never used while for $d = 3$, this fraction becomes $\frac{3}{7}$. This negates to some extent the advantage of being able to perform the concurrent iterations on different grid levels simultaneously.

The above discussions assumes that there are enough processors to accomodate the number of grid points. For higher dimensional problems, this may require more processors than are available. In such cases, it is natural to consider assigning each processor to more than just one grid point. For example, in a two dimensional problem, each processor can hold a line of grid points in one coordinate direction. For the other coordinate direction, one can use the one dimensional binary refected Gray code for the mapping. The proximity preserving property obviously holds. Since the fine grids require fewer processors, such implementations can be more efficient in their usage of the

available processors [3]. This is just a special case of the more general idea of *domain decomposition.* The idea is to decompose the finest grid into a collection of smaller domains, each assigned to a different processor. In other words, a particular level of the pyramid, not necessarily the finest, will get assigned one node per grid point. Processors are idle only when iterating on levels coarser than this selected level. This has the effect of increasing the efficiency of processor utilization [8].

## 6. Complexity

In order to compare the performance of the three different algorithms described earlier, in this section we analyze their arithmetic and communication complexity. In this discussion we assume that each grid point is assigned to a different processor.

Let $W_a$ denote the arithmetic complexity for the work performed per cycle in one single grid before a transfer to a different grid. Then $W_a$ is given approximately by:

$$W_a = (st_s + t_r + t_i),$$

where $s$ denotes the total number of relaxation sweeps performed on that grid (before and after transferring to another grid), $t_s$ denotes the time for performing one relaxation sweep, $t_r$ denotes the time for computing and projecting the residual and $t_i$ denotes the time for performing the interpolation and correction. Note that $W_a$ is the same for all three algorithms. For a general variable coefficient problem in $d$ dimensions with a nearest neighbor stencil discretization, injection of residuals and linear interpolation, we can estimate these times as follows:

$$\begin{aligned} t_s &= time\ of[(2d+1)\ multiplies + (2d-1)\ additions] \\ t_r &= time\ of[(2d+1)\ multiplies + 2d\ additions] \\ t_i &= time\ of[(2d-1)\ additions]. \end{aligned}$$

Next we consider the communication complexity of the algorithms. For simplicity, we assume that each node of the hypercube can send and receive data on all its data paths simultaneously. Let $t_c$ denote the time it takes to send one floating point number from one node to its neighbor. For the standard algorithm with the binary reflected Gray code, we can estimate the communication complexity $W_c^{st}$ on any grid as

$$W_c^{st} = (s+2)2t_c,$$

where the $(s+2)$ term denotes the number of data transfers needed to perform the $s$ relaxation sweeps and the computation of the residual and the interpolation; and the term $2t_c$ is a bound for the time it takes for each transfer. Similarly, the communication complexity $W_c^{ex}$ for the standard algorithm with the exchange of data before transfer is estimated by:

$$W_c^{ex} = (s+2)t_c + 2t_c,$$

because the exchange takes two extra data transfers, one before and one after transferring to a different grid, but afterwards each transfer takes only time $t_c$. Finally, the communication complexity $W_c^{co}$ for the concurrent algorithm is estimated by:

$$W_c^{co} = (s+2)t_c + 4t_c,$$

because the transfer between grids takes two data transfers each way.

One interesting point to note is that

$$W_c^{ex} < W_c^{st}.$$

In other words, the standard algorithm with exchange is always better than the standard algorithm without exchange, in terms of communication cost. Moreover, this advantage increases with increasing $s$, the number of relaxation sweeps. For this reason, it seems that the exchange algorithm is to be always preferred. Grosch [6] considers a similar strategy for his implemetation of multigrid algorithms on the perfectly shuffled nearest neighbor array. It is more difficult to compare the concurrent algorithm with the standard ones due to the difference in convergence rates. We only note that for large values of $s$, $W_c^{co} \approx W_c^{ex}$.

Finally, for small values of $d$ and $s$, the number of arithmetic operations and data transfers are about the same order of magnitude and therefore communication should not become a bottleneck if the time it takes to send one floating point number is not much greater than the time it takes for one arithmetic operation.

## 7. Concluding Remarks

The hybercube seems to be ideally suited for implementing the standard multigrid algorithm. A communication-efficient mesh-to-node mapping is possible via the binary reflected Gray codes and extends to higher dimensional meshes in a straightforward way. The concurrent algorithm, on the other hand, does not seem as well-suited to the hypercube for higher dimensional problems. The main problem is that in order to have each point on the pyramid mapped into distinct nodes, one has to use a hypercube of one higher dimension than that needed for just the fine mesh. As was shown in Section 4.6 almost half of all the nodes are *never used.* For the concurrent algorithm to be cost-effective as compared to the standard algorithm, its convergence rate must be at least twice as fast. Unfortunately, very little is known about the convergence rate of the concurrent algorithm and experiemental evidence indicates that the convergence rate *decreases* with increasing $m$ [4]. Therefore, being able to perform the concurrent iterations on different grid levels simultaneously does not pay. The only way in which we can still achieve maximal use of the available processors for higher dimensional problems is by using each node of the one dimensional pyramid to hold a line or a plane of unknowns, as discussed in Section 4.6.

When there are more than one problem to be solved on the same grid structure, the standard algorithm on the hypercube can be made more effective by pipelining these different problems. When the nodes assigned to a particular grid are inactive for one of the problems, they can be working on the other problems. The programming of the individual nodes may be more complicated, however.

It is interesting to note that the potential for a concurrent algorithm is there with the mapping described in Section 4.3 for the standard algorithm. This is so because when the nodes assigned to a coarse grid are active, the other nodes, while not completely representing the inactive grids, can still be used to perform some sort of concurrent relaxation sweeps.

Finally, the mappings described in this paper, being dependent mainly on the grid structure, are useful for other variants of the standard multigrid algorithms, such as the nested iteration and the FAS algorithms [13].

## 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] A. Brandt, Multigrid solvers on parallel computers, M.H. Schultz ed., *Elliptic Problem Solvers,* Academic Press, 1981, pp. 39–83.

[3] T.F. Chan, R. Schreiber, *Parallel Networks for Multigrid algorithms: Architecture and complexity,* SIAM J. Scient. Stat. Comp., 6 (July 1985), pp. 698–711.

[4] D. Gannon, J. van Rosendale, *Highly Parallel Multigrid Solvers for Elliptic PDE's,* Technical Report 82-36, ICASE, 1984.

[5] C.E. Grosch, Poisson Solvers on Large array computers, B.L. Buzbee and J.F. Morisson ed., *Proceedings 1978 LASL Workshop on Vector and Parallel Computers,* 1978, pp. 98–132.

[6] ———, *Performance analysis of Poisson solvers on array computers,* Technical Report TR 79-3, Old Dominium University, 1979.

[7] Johnsson, S.L., *Odd-Even Cyclic Reduction on Ensemble Architectures.,* SIAM J. Sci. Stat. Comp, (1986).

[8] Ortega, J.M. and Voigt, R.G., *Solution of Partial Differential Equations on Vector and Parallel Computers,* SIAM Review, 27 (1985), pp. 149–240.

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

[10] Y. Saad and M.H. Schultz, *Data Communication in Hypercubes,* Research Report 428, Dept Computer Science, Yale University, 1985.

[11] ———, *Topological Properties of Hypercubes,* Research Report 389, Dept Computer Science, Yale University, 1985.

[12] C.L. Seitz, *The Cosmic Cube,* CACM, 28 (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.