18.337 Parallel Scientific Computing February 23, 1995

Lecture 5a

Topic: Hypercubes and Communication *1995 Scribe: Greg McLaren*

Overview

- The Gray Code and Other Embeddings of the Hypercube
- Going Beyond Embedded Cycles
- Hypercube theory: Unique Edge-Disjoint Hamiltonian Cycles in a Hypercube
- An Application: Matrix Multiplication on a Grid
- The Direct N-Body problem on a Hypercube
- The Hamming Sum
- Extending Gray Code-directed Body Movement to Multiple Sets of Bodies
- Matrix Multiplication using all Hypercube Wires
- Matrix Multiplication using the Cartesian Product

Interprocessor Communication and the Hypercube Graph

All parallel supercomputers allow message passing between nodes as a means of communication. This programming model differs from programming in a data-parallel language such as CM-FORTRAN or C*, which has been done previously in this course. The CM-5 was designed primarily for data parallel programming, but may also be used for message passing. In the past, message passing did not give users access to the vector units which make the CM-5 so fast. The latest nodal model allows message passing to take advantage of the vector units.

As research in parallel computation has come to the forefront of computer science in recent years, parallel algorithm designers have had to struggle with the complications imposed by a large network of processors working to solve a single problem. As the number of processors increases, experimental evidence indicates that *communication bandwith* (throughput), rather than latency, becomes the bottleneck limiting the scalability of parallel algorithms. Thus, choosing the correct communication subsubstrate for one's parallel computer is of paramount importance to its performance. Because of the versatility of an interconnection graph known as the *hypercube*, it looked very attractive to machine designers in the recent past, and was used in the CM-2, Cosmic Cube, and N-Cube (one is located here at MIT, in the Earth Resources Laboratory).

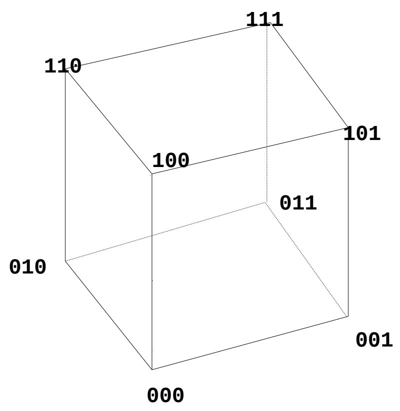


Figure 1: The Hypercube for $d = 3$

A Hypercube graph consists of:

1. d Dimensions

2. 2^d Vertices

3. $(b_0 ... b_{d-1})$ and $(b'_0 ... b'_{d-1})$ are connected by an edge if *and only if they* differ in exactly one bit.

There are $d2^{d-1}$ edges in a hypercube of dimension d.

A picture of the three-dimensional hypercube may not be as helpful as it would seem, but is included as Figure 1.

On a sad note, the hypercube design for the most part has been abandoned. Network routing today is often randomized, taking away the users' control over network communication. Fat trees, consisting of interconnection trees in which the processors are nodes and the links near the root have a proportionately higher bandwith, are now enjoying acclaim. They have a lower edge count than hypercubes (not counting bandwith per edge) and win under randomized communication schemes.

The Gray Code and Other Embeddings of the Hypercube

A hypercube is an amazingly versatile medium on top of which familiar and useful topologies can be easily created. Perhaps the most basic such embedding is the *Hamiltonian Path*[1], a topology in which the nodes appear to be in a ring, so that each node has two neighbors and

[1] A Hamiltonian Path is one which touches each node of a graph exactly once. Contrast with an Eulerian Path.

a message traveling along the ring encounters each node exactly once. Finding such a path in a d-dimensional hypercube is easy because of the existence of a d-bit *Gray code*.

The Gray Code is named after Frank Gray, who formalized it in the 1940's while working on radar-related engineering problems. Given a rapidly changing binary output from a physical measuring device, sampling accuracy can be improved by using the Gray Code to encode adjacent numbers rather than conventional binary. The code was originally invented by Emile Baudet (1845-1903) for telegraph communication optimization—proper equipment can send several telegraph signals down the same wire to maximize bandwith under fixed physical constraints.

The Gray Code for $d = 3$ embeds a Hamiltonian path in the three-dimensional hypercube shown above:

GRAY CODE	TRANSITION SEQUENCE
0 0 0	
0 0 1	0
0 1 1	1
0 1 0	0
1 1 0	2
1 1 1	0
1 0 1	1
1 0 0	0

Each time we advance from one element of the Gray Code to the next, we move along a single dimension. The *transition sequence* is merely the sequence of such change directions mapped out as the Gray Code sweeps along its Hamiltonian path. (As a single bit must change at every step, only the dimension of the changed bit, 0, 1, or 2, in this case, must be described.)

To build a Gray Code for d dimensions, one need only take the Gray Code for $d - 1$ dimensions, reflect it top to bottom across a horizontal line just below the last element, and add a leading one to each new element below the line of reflection. Given that, you can see how the Gray Code above was derived easily from the Gray Code for $d = 2$:

GRAY CODE (LOCATION OF VERTEX)
0 0
0 1
1 1
1 0

Going Beyond Embedded Cycles

We are not limited to cycles; now that the Hamiltonian cycle has been described, we can use it to inbed a *grid* inside a hypercube. For example, suppose we have a $d = 5$ hypercube. We

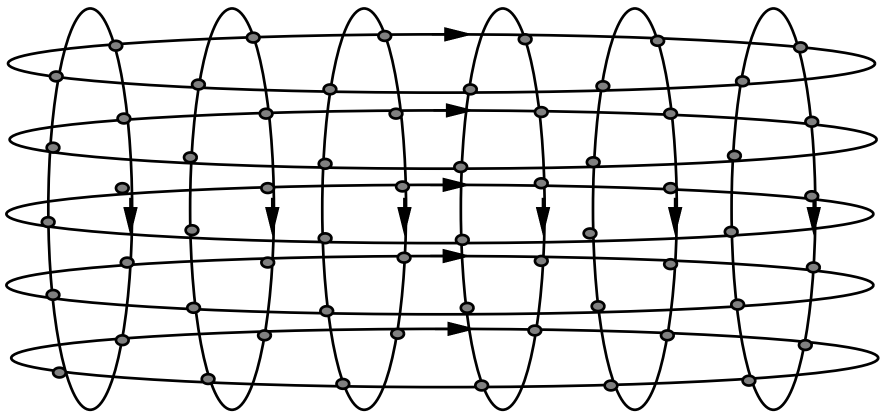


Figure 2: Embedding a Two-dimensional Grid in a Hypercube by bit-field partitioning.

can group the bits together like $d_4d_3|d_2d_1d_0$, where the left group is given two bits, and the right group three. By using *two indepedent Gray Codes* for the two sets of bits, we can index into the total set of nodes using *two numbers*. Each number determines a sequential position in one of the Gray Codes (or, equivalently, along one of the Hamiltonian cycles mapped out by them), so that together they completely specify a unique node. By sweeping through the Gray Code mapped to the top two bits, the position moves up and down a "column," and by sweeping through the code mapped to the lower three, the position moves back and forth in a "row". The result is a grid like that shown in Figure 2. Three dimensional (or even higher dimensional) grids could be generated using this bit-partitioning technique—we are not limited only to two dimensions.

Finally, note that in a hypercube of high dimensionality, several distinct Hamiltonian paths (spanning the entire set of nodes) can exist which have no (directed) edges in common. A set of nodes can thus be spanned by two completely separate loops around which information can be passed without worrying about collisions. An example of this is shown in Figure 3.

Hypercube theory: Unique Edge-Disjoint Hamiltonian Cycles in a Hypercube

Consider in more detail the question of how many edge-disjoint Hamiltonian cycles exist for a hypercube in d dimensions. (If two Hamiltonian cycles traverse an edge in opposite directions, they will be considered edge-disjoint for our purposes.)

For hypercubes of even dimension, there are exactly d edge-disjoint Hamiltonian paths. This is not surprising, since from above we know that a hypercube of dimension d has exactly $d2^{d-1}$ edges. Doubling this number to account for both directions of an edge and dividing it by the number of edges in a Hamiltonian path (2^d, since that's the total number of nodes) *also yields d*. The match tells us that *all edges* can be part of the set of edge-disjoint Hamiltonian paths in a hypercube of even dimensionality.

Strangely, for hypercubes of odd dimension, *there are only* $d-1$ *edge-disjoint Hamiltonian*

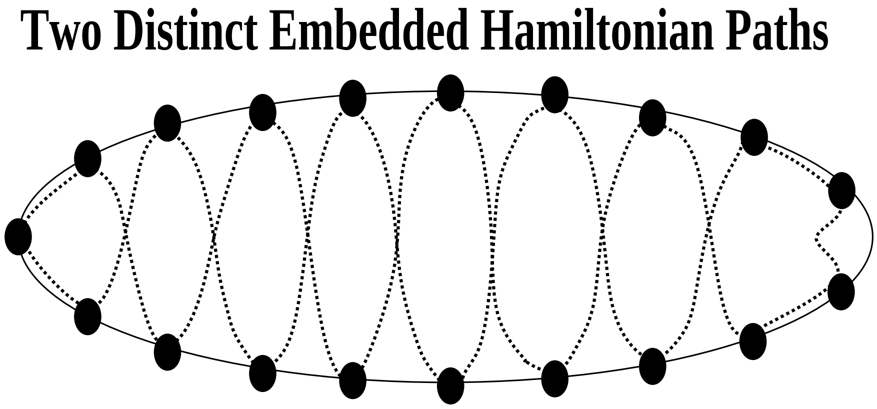


Figure 3: Two edge-disjoint Hamiltonian paths spanning a set of nodes.

paths.[2] Since the same edge-use analysis applies as did in the case of hypercubes of even dimension, we conclude some edges must not be used.

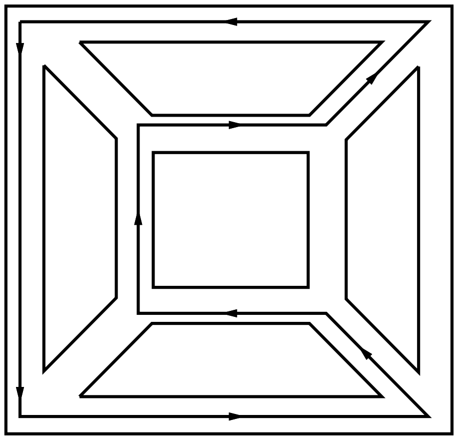
We can prove the assertion in the case of $d = 3$. Let's consider a single Hamiltonian cycle embedded in the hypercube of dimension three, as depicted in flattened form in Figure 4(a). It should be clear that in three dimensions, any Hamiltonian path can be rotated onto the one shown, so that there is no loss of generality in assuming the presence of the particular path shown. To see this, simply note that each two-dimensional slice of the $d = 3$ cycle must be a cycle in two dimensions (lacking one edge). The cycle in two dimensions is unique except for direction. Furthermore, it can be combined with another such cycle only in one way to form the three-dimensional Hamiltonian cycle. Given this pre-existing cycle in a hypercube of dimension 3, we assume that two additional edge-disjoint Hamiltonian cycles exist in the same hypercube, and produce a contradiction.

The proof is quite simple. Consider in turn the lower-left and upper-left corners of Figure 4(a). In the lower-left corner, there are two possibilities for edges entering the corner, and two possibilities for edges leaving. These can be combined to form segments of two paths in a unique way[3], so we have determined the behavior at that point of the two additional edge-disjoint Hamiltonian cycles which we assumed existed. See Figure 4(b). A similar analysis can be made for the upper-left corner; we again are led to a unique pair of path segments which must belong to the two new edge-disjoint Hamiltonian cycles, as shown in Figure 4(c). The only logical way to connect the segments found in Figures 4(b) and 4(c) is shown in Figure 4(d).

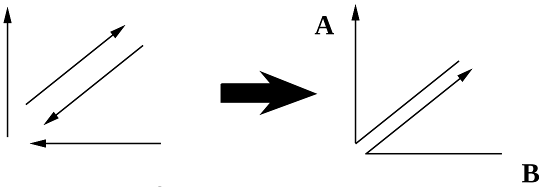
We are almost done. Look at the point marked X on Figure 4(d). When the path we have just connected gets to this point, it cannot go to the right, for that edge has already been used for rightward travel by the original Hamiltonian cycle. It certainly can't go back

[2] I am reminded of the famous vector-field theorem which states that a vector field can exist on the surface of a ball in **n** dimensions without a singular point iff n is even. Could these two results be related?

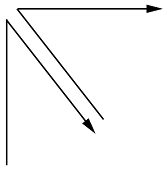
[3] A Hamiltonian cycle in no case immediately backtraces an edge it has just traversed; this provides the uniqueness.



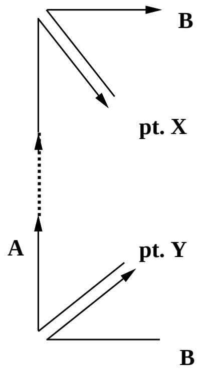
(a) A single Hamiltonian cycle in the d = 3 hypercube



(b) The set of legal incoming and outgoing directed edges at the lower left corner for new Hamiltonian paths



(c) The set of legal Hamiltonian cycle segments at the upper-left corner



(d) Path segment A cannot go right at pt. X because of the original Ham. cycle. But if it goes down it forms a closed loop of 4 nodes!

Figure 4: Proof that for the $d = 3$ hypercube there are only 2, not 3, edge-disjoint Hamiltonian cycles.

out the way it came in. Thus, it must go down, to the point marked Y. *However, this would form a closed loop consisting of four nodes!* Such a loop might be part of larger loop, but then it wouldn't be *Hamiltonian*, since it would pass through two nodes twice.

Since we have reached a contradiction, we conclude there are not two additional edge-disjoint Hamiltonian paths in the three dimensional hypercube, and in fact only one additional such path. Just reverse the direction of the original Hamiltonian path to find it.

An Application: Matrix Multiplication on a Grid

The first algorithm for matrix multiplication on a grid is Canon's. This algorithm is designed for two dimensional grids. It still can run very fast on the CM-5 using CSHIFTS, which will be shown later.

The goal is to find $\mathbf{C}$ where $\mathbf{C} = \mathbf{A} * \mathbf{B}$ and $\mathbf{A},\ \mathbf{B}\ and\ \mathbf{C}$ are all matrices. Canon's algorithm involves two steps: a skewing step and a systolic step. In the skewing step, the rows of $\mathbf{A}$ are slid to the left (with wraparound), such that each row is slid to the left a number of slots equal to its row number, with the first row not moving at all. The *columns* of $\mathbf{B}$ are, analogously, each slid upwards a number of slots equal to their column number, with the first column not moving at all. These movements are an effort to achieve a special configuration of the data which will be used by the following systolic step. In the systolic step, we peform k iterations, moving $\mathbf{A}$ horizontally one slot (as a whole) and $\mathbf{B}$ vertically one slot (also as a whole) in each iteration. As the iterations go by, the products of the elements of $\mathbf{A}$ and $\mathbf{B}$ that pass by a location are accumulated into that location, and at the end the product $\mathbf{C}$ has been computed in each row and column position.

1. Skewing Step

$$A_{i,j} \leftarrow A_{i,i+j} \qquad (modulo\ n)$$

$$B_{i,j} \leftarrow B_{i+j,j} \qquad (modulo\ n)$$

2. Systolic Step

$$C_{(i,j)} = \sum_{k=0}^{n-1} A_{i,j+k} B_{i+k,j}$$

This algorithm is depicted symbolically in Figure 5.

Another method for matrix multiplication involves broadcasts on a grid. Thinking about this algorithm on a grid of n^2 processors reduces a matrix multiply to O(n) time (Ref. Bertsekas 77). Imagine a $n * n$ processor machine with a block of matrix $\mathbf{A}$ and matrix $\mathbf{B}$ on each node (you can imagine one element on each node, just don't tell Prof. Edelman).

1. Each processor sends $a_{i,j}$ to all the other processors in the i^{th} row. (a_{ij} is the (row i, column j) element of matrix $\mathbf{A}$). On a graph this operation looks like Figure 6.

2. Each processor sends b_{ij} to the processors in the jth column. This operation is shown in Figure 7.

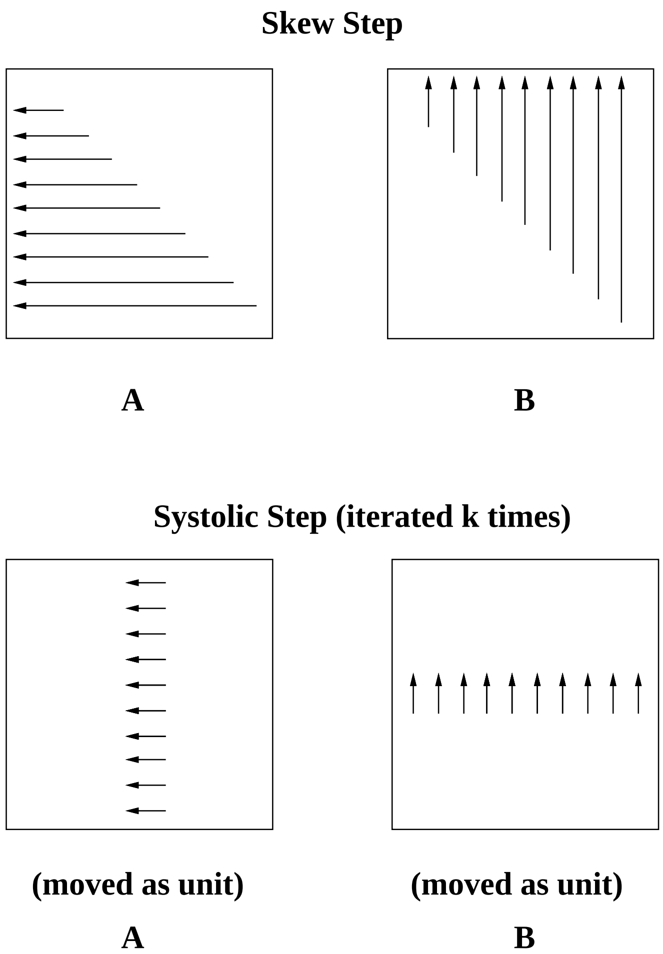


Figure 5: Geometry of Canon's Algorithm

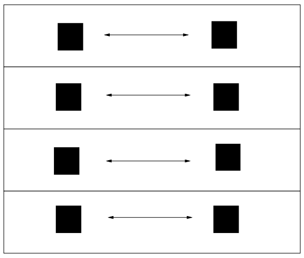


Figure 6: Multiple Row Sends

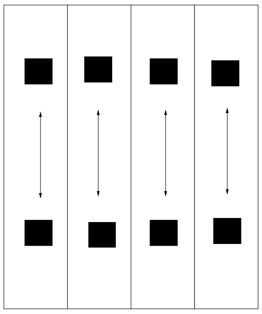


Figure 7: Multiple Column Sends

3. The matrix **C** is formed on each node by

$$\sum_{1}^{n} a_{i,j} b_{i,j}$$

This algorithm can also be implemented using C-SHIFTS. This can be easily pictured in Figure 6 and Figure 7.

Another way of improving Canon's algorithm (systolic step) can be easily pictured. In the original implementation, each element of a block in **A** was sent to the left (to its western neighbor), and each element of a block in **B** was sent up (to its northern neighbor). Thus, in each dimension, the ring of connections was only used unidirectionally. The improvement consists in allowing each ring to be used *bidirectionally*. Each block of the **A** matrix can be divided into left and right halves while each block of the **B** matrix can be divided into top and bottom halves. Matrix **A** can simultaneously send information east and west while **B**

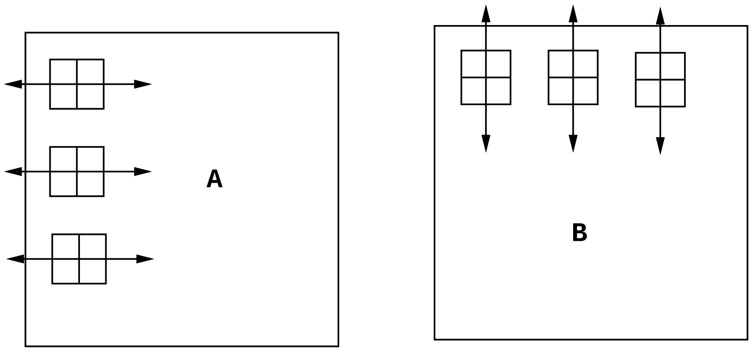


Figure 8: Two Direction Matrix Multiply

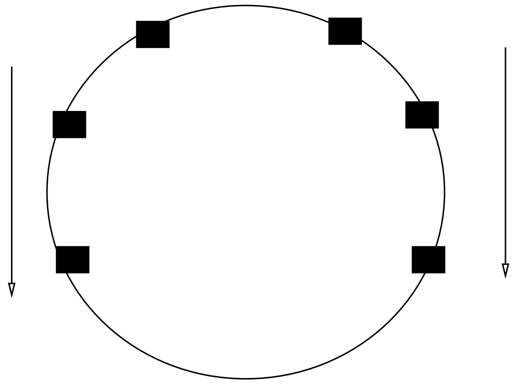


Figure 9: Two Direction Ring

can simultaneously send information north and south. In such an operation, the elements of a block in **A** in the block's right half are sent east, and the elements of a block in **A** in the block's left half are sent west. Similarly, the elements of a block in **B** in the block's top half are sent north, while the elements of a block in **B** in the block's bottom half are sent south. This does not change the geometry of the information storage in each matrix, and thus the multiplication occurs after much less the communication time. Figure 8 describes this operation pictorially.

This method can be best thought of as a ring going in two directions, as is shown in Figure 9.

The Direct N-Body problem on a Hypercube

The simplest way to solve the direct N-body problem on a hypercube is two create two copies of the set of bodies, one stationary and one moving, allocated to the nodes spanned by a single large Hamiltonian cycle. In each step, the moving bodies advance along the fixed Hamiltonian path one unit, and interaction computations occur in parallel at each node adding the effects of the current pair of bodies to the accumulated sum. To increase efficiency, two edge-disjoint Hamiltonian cycles spanning the nodes of a hypercube can be used instead of a single cycle, as was shown in Figure 3. The advantage of this approach is that two sets of moving bodies can be present (each set moving along one of the Hamiltonian cycles) so that the effects of two pairs of bodies are accounted for per systolic step.

A faster method for a direct N-Body solution can take advantage of the hypercube graph and the transition sequence (of a Gray Code) described earlier. A dynamic and static copy of each body can be made at every node. The dynamic body will move around the hypercube changing along the dimension dictated by the transition sequence of a fixed Gray Code. Every body will follow the transition sequence, but not necessarily the Hamiltonian Cycle. The body at (0,0,0) will follow the Hamiltonian cycle exactly, while others will follow in a similar sequence, but not right behind on the same path. Each body will be on a different path, but no body will cross another or "step on another body" at the same moment in time.

In order to define how the bodies other than the one at the zeroth node move, we must introduce the concept of the Hamming sum. (The zeroth node moves exactly in a Hamiltonian path, using the transition sequence derived from that path.)

The Hamming Sum

The Hamming sum will help create an optimal matrix multiply algorithm for the hypercube.

The Hamming Sum is defined as

$$b_{d-1}...b_0 \bigoplus b'_{d-1}...b'_1 b'_0 = b_{d-1} \bigoplus b'_{d-1}, ..., b_1 \bigoplus b'_1, b_0 \bigoplus b'_0$$

where $\bigoplus$ is an *exclusive or* operation. It is essentially just a bitwise exclusive-or operation. (We can now define the **transition sequence** conveniently as the bit position for which $g_i \bigoplus g_{i+1}$ is 1—the dimension along which the Gray Code changes in that step.)

Let's pick a fixed Gray Code G_0. Then the direct N-Body algorithm puts the body that starts at vertex v into the vertex $v \bigoplus g_k$ at the k^{th} step, where g_k is the k^{th} element of G_0.

Extending Gray Code-directed Body Movement to Multiple Sets of Bodies

Heretofore, we have been discussing a *single set of moving bodies.* The one at the zeroth node moves in a Hamiltonian cycle, and the others move in similar paths derived from the original Hamiltonian cycle using the node's position and the Hamming sum. Now, however, we consider *multiple sets of moving objects.* For the sake of argument, let's assume we have a three-dimensional hypercube and three sets of moving objects–a red set, a blue set, and a green set. We want to move each colored set according to the rules just defined for a single set. Obviously, we can't assign G_0 to be the Gray Code for each of the three colors, since that would cause the objects on the zeroth node to travel the exact same Hamiltonian path at the same time and collide. (Assume only a single object can be sent along a wire at one time.) What can be done to use "all of the wires all of the time"?

The solution is to note that there are many Gray Codes for a given bit vector length. New Gray Codes can be created from existing ones by *shifting the columns of bits one or more positions to the left, wrapping around when columns move off the left side.* For example, using our previous table, we could make the transformation:

OLD GRAY CODE	OLD TRAN. SEQ.	NEW GRAY CODE	NEW TRAN. SEQ.
LOC. OF VERTEX		LOC. OF VERTEX	
0 0 0		0 0 0	
0 0 1	0	0 1 0	1
0 1 1	1	1 1 0	2
0 1 0	0	1 0 0	1
1 1 0	2	1 0 1	0
1 1 1	0	1 1 1	1
1 0 1	1	0 1 1	2
1 0 0	0	0 0 1	1

Note that the transition sequences of these two Gray Codes are related in a special way. The second can be produced from the first by adding a 1 (modulo 3) to each element of the first's transition sequence. In fact, there is a third unique Gray Code, which can be produced from the first by adding a *2* (modulo 3) to each element of the first's transition sequence. In general, given a hypercube in $d = k$ dimensions, there are k unique Gray Codes. (Gray codes which represent the same path but traversed in an opposite direction are not counted as unique.)

Thus, if in three dimensions we call these different Gray Codes G_0, G_1, and G_2, we know by how their transition sequences can be derived that, *at any given node of the hypercube*, the transition sequence elements of G_0, G_1, G_2 are all different. Because of this, if we allow one set of dynamic bodies to be controlled by Gray code G_0, another by Gray Code G_1, and the last by Gray Code G_2, we may safely conclude that at any instant in time the red, blue, and green objects that were at the zeroth node at time $t = 0$ follow the Hamiltonian paths implied by G_0, G_1, and G_2 *without colliding.*

You may be asking, *"You proved earlier that you can't have 3 edge-disjoint Hamiltonian paths on a hypercube in three dimensions!? What's going on?"* The question is good one to ask, but is not difficult to answer. Before, we were talking about *edge-disjoint cycles*—paths which never pass down the same edge going the same direction. Here, we're talking about *time-edge-disjoint* cycles. We don't care whether the red, green, and blue objects that were at the zeroth node at $t = 0$ ever travel down the same wire in the same direction, we only

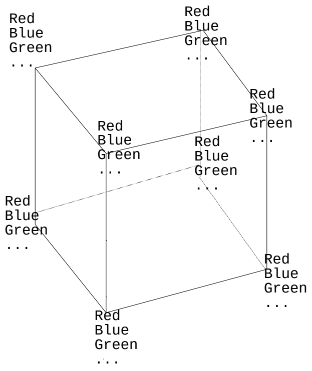


Figure 10: N-Body Breakup on a Hypercube

care whether they might do so *at the same time.* As the transition sequences are related by modulo arithmetic in our scheme, a collision is impossible.

Thus, by taking different colored sets of bodies, assigning Gray Codes related by column shifts to those sets, and moving each set around using its assigned Gray Code and the Hamming sum as described above, we can *use all the wires all the time.* In fact, this set up can be used to extend the grid multiplication routine (with the skew and systolic phases) presented earlier so that instead of using only a two-dimensional grid embedded in the hypercube, it uses the hypercube's full connectivity.

Matrix Multiplication using all Hypercube Wires

We can use every connection of the hypercube on every iteration by using the Hamming sum in the calculation of the matrix product. To do so, we require only that the matrices to be multiplied are square, and that the hypercube we are given is of an even dimension. We first split the complete bit-field specifying location in the hypercube into a left and right half, each of which is subsequently addressed as a bit-field of length d/2. Using a Gray Code in d/2 dimensions, g_k, and the hamming sum on d/2-length bit fields, we can perform a matrix multiply in two phases:

1. Skewing Step
$$\mathbf{A}_{ij} \leftarrow \mathbf{A}_{i,i \bigoplus j}$$
$$\mathbf{B}_{ij} \leftarrow \mathbf{B}_{i \bigoplus j,j}$$

2. Systolic Step
$$\mathbf{C} = \sum_k \mathbf{A}_{i,j \bigoplus g_k} \mathbf{B}_{i \bigoplus g_k,j}$$

In each case, when a variable is subscripted by two expressions separated by a comma, the left one refers to the left d/2-bit field, and the right one refers to the right d/2-bit field.

Matrix Multiplication using the Cartesian Product

If **A** is any graph with the cartesian product that allows **k** timewise edge-disjoint cycles, then the cartesian product of the graph with itself can be used for matrix multiplication. This is essentially a generalization of the idea we used above when we split the original d-bit field of the hypercube given to us into two d/2-bit fields, and formed a cartesian product of those fields in order to carry out the matrix multiplication in matrix equations immediately above.

References

[1] Bertsekas, Dimitri, et al. *Parallel and Distributed Computation.* Prentice hall, New Jersey. 1989.

[2] Leighton, F. Thomson. *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes.* Morgan Kaufmann, San Mateo, California, 1992.

Parts of these notes were taken from last year's notes written by Scott Wallace.