Visualization of Embedded Binary Trees in the Hypercube

James Slack
University of British Columbia
Department of Computer Science
jslack@cs.ubc.ca

April 30, 2003

Abstract

In the realm of parallel multiprocessor systems, it is convenient to represent the structure of a system as a regular, replicated structure such as a standard hypercube to aid in communication routing. A typical parallel program, however, makes use of simple communication structures often represented as a tree. I investigate the difficulty of visualizing the embeddings of arbitrary balanced binary trees to aid in the embedding process. The visualization system presented is assessed for usefulness in developing new algorithms to take advantage of the representation. Various other visualization techniques are also introduced to take advantage of this system.

1 Introduction

A hypercube multiprocessor system uses a regular and replicated structure to its advantage; rings, meshes, and efficient routing makes such a system layout attractive to system designers. In such a system, the nodes of the hypercube correspond to the processors while the edges of the hypercube correspond to the communication links between adjacent hypercubes. Also, some hypercube systems are not complete in that not all edges which appear in a fully connected hypercube exist. Furthermore, for many reasons, the links between hypercube nodes are not identical in length, communication latency, or other communication performance metrics. In this paper, I will assume that the hypercube graph is fully connected in that if the binary representation of hypercube nodes is considered, each adjacent node pair will have a Hamming distance of 1. I will also assume communication links are homogeneous.

To make use of multiprocessor systems, it is often natural to represent the communication between processors as a tree; such communication structures occur frequently as client-server or processor farm models. The usefulness of binary trees in communication is not clear, but since the branching factor of binary trees is small, the amount of communication work forced onto a single node is negligible. Communication bottlenecks in a binary tree are not an issue since the maximum degree

of any node is 3. The tradeoff in branching factor is simple: with a large branching factor, many trees become congested but the diameter is potentially small while a small branching factor has converse properties and the distribution of work is often improved at a cost of moving data long distances. Again, the nodes of a tree will represent processors in a multiprocessor system, much like the hypercube model of a multiprocessor system. These facts lead into a general problem: can the nodes and edge links in a tree be given a static assignment to nodes and edge links in a hypercube?

It is not trivial to embed trees in a hypercube [10], however, the complexity of embedding arbitrary binary trees is not known. Many attempts to embed special types of binary trees are known; heuristics exist for classes of trees known as caterpillars [2], quasi-stars [5], and straight-line trees by Hamiltonian cycles. Some attempts to embed arbitrary trees where *average edge dilation*, the average number of hypercube edges that are used in a particular hypercube embedding for all edges that are used, is minimized are also common [8]. Due to the nature of hypercubes, it is a straightforward observation by the parity of the binary representation of adjacent nodes of a hypercube that any *perfect*, all nodes embedded with no extra edge dilation, embedding requires the embedded graph to be *colour balanced*. In a colour balanced graph, in this case a tree, the graph is bipartite.

Havel's conjecture [3] introduces the idea that if a binary tree is colour balanced then it is embeddable in a hypercube of optimal size with no extra edge dilation. This hypothesis has not been proven nor has there been a counterexample found to disprove the claim. Exhaustive proofs for up to H_5 have been completed, but exhaustive proofs for H_6 and larger have not due to the explosive complexity of the problem and the long running time of embedding algorithms for a problem of that size.

The embedding problem of a colour balanced tree with 64 nodes into an optimal hypercube (H_6) was attempted by Helmer and Eisenberg [4] in the context of parallel systems research. The problem space was divided into domains for distributing the computational load characteristic of the simple backtracking algorithm that was used. An in depth formal definition of the problem can be found here [1] where Aderhold and Slack embedded colour balanced trees in optimal hypercubes with stochastic search methods. Their algorithms resulted in embeddings of all 32 node *strongly balanced*[1] trees as well as embeddings of 64 node trees that Helmer and Eisenberg could not embed. The embedding results for 64 node colour balanced trees from Aderhold and Slack will be visualized in this paper.

Although the embedding of a tree in a hypercube is easily verified by checking the injective invariant of embedding, it is not so easy to visually interpret the embedding. To visualize embeddings in hypercubes H_3 and smaller, it is fast and simple to sketch the result. For H_4, it is still possible to sketch the embedding as a simple stretch of H_3; the "4^{th} dimension" edges connect two H_3 structures. H_6 embeddings are much more difficult than H_5 since the two dimensional stretch of H_5 from H_3 stays in a plane while three dimensional stretch of H_6 occludes the "rear" H_5. See Figure 1 for an example of H_6, noting that it is indeed difficult to visualize without a graphics package capable of freely rotating the hypercube in 3 dimensions.

[1] *Strongly balanced* trees are binary trees that can be formed by a simple process of adding pairs of connected nodes to a strongly balanced tree; these trees have special *folding* [9] properties and are proven to be perfectly embeddable in optimal hypercubes.

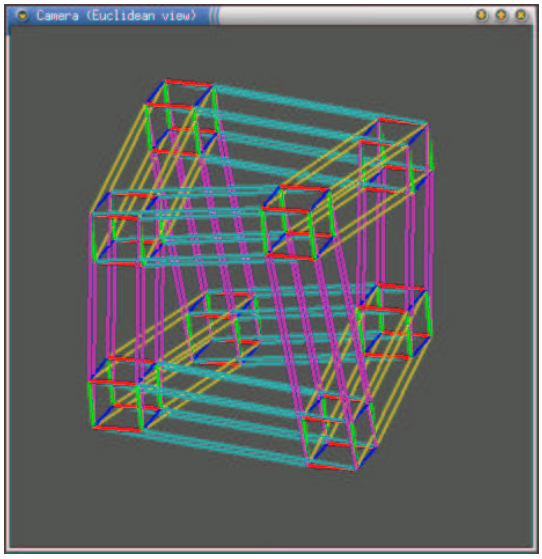

Figure 1: A full hypercube (H_6)

The paper is organized as follows: First in Section 2 I describe the programs used to visualize embeddings. In Section 3 the visualization of embedding binary trees in hypercube H_6 is introduced. Then, in Section 4 several reductions are introduced to remove edges that are common to two binary tree embeddings; the embeddings are usually taken from the same binary tree instance, but this is not absolutely necessary. Finally, Section 5 goes over some of the results of this project and Section 6 concludes.

2 The Program

For this project I have chosen to use a combination of `Geomview` [6], a software package for visualization, and Java. For the visualization component, `Geomview` provides some fundamental 3 dimensional transformations such as rotations and translations and was very convenient for visualizing hypercubes as complicated as H_6 with all edges shown simultaneously. For the user interface component, Java provided an easy to use GUI layout engine and many other nice tools that are found in libraries such as hash tables, parsing, multiple threads, and basic file I/O. Further high-level implementation details will be addressed in Section 5.

3 Visualization of Embeddings

This section describes methods of visualizing an embedding of a single tree in a hypercube. The methods use an already embedded tree as input and either visualize the embedding as a tree (Section 3.1) or as a hypercube structure (Section 3.2).

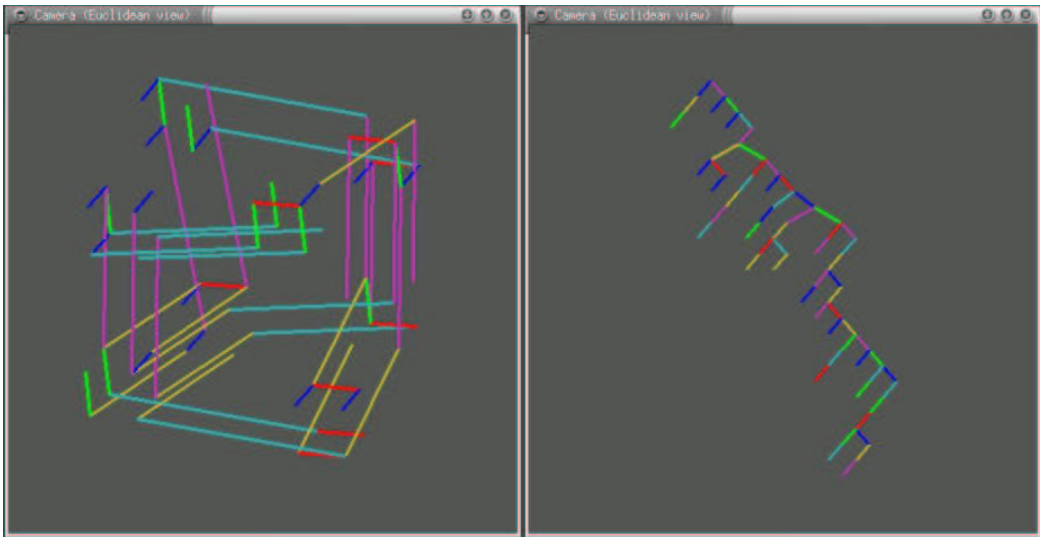

Figure 2: A hypercube with an embedded tree (left) and the tree embedded in the hypercube (right)

3.1 Colouring Trees

Upon embedding a 64 node tree, the most natural way of looking at a tree without putting it into a 3 dimensional structure was colouring the edges of the tree. This way, the structure of the tree is maintained as a 2 dimensional layout while still giving information about the dimensions used in the hypercube. Two trees shown with this method provided an easy comparison for simple isomorphic embeddings of the same tree instance but if two edges from the same tree in the same position were coloured the same, it *did not* mean that those edges occupied the same edge in the hypercube. No information as to where individual edges lie in the hypercube is given, so of course other techniques are required. See the right side of Figure 2 for a tree drawn with colours representing dimensions.

Of course since a hypercube is symmetric in any direction, the colours are completely interchangeable. See Section 5.1 for an explanation of why this is important to the results of this visualization.

3.2 Trees in Hypercubes

This method provides much more detail as to where an edge is embedded relative to other edges. Since all embeddings input by the program are perfectly embedded in the hypercube, no edges need to be drawn differently to represent dilation. However, the number of edges required to draw an embedding (63) is approximately $\frac{1}{3}$ of the number of edges in H_6 (192) and the hypercube looks very cluttered. See the left side of Figure 2 for an example of a tree embedded in H_6. The section following (Section 4) describes methods to reduce the clutter by using comparison techniques for any two embeddings, either by drawing the edge similarities between two embeddings or by drawing the edge differences between two embeddings.

4 Reducing Embeddings

Since embeddings can become very cluttered, I have chosen to remove edges when possible instead of further additions of edges. If I had chosen to show different edges by adding slightly different coloured edges to an embedding of a tree in the hypercube from a second embedding, the hypercube would surely be too cumbersome to be useful for analysis. Now I will present several techniques I explored empirically during my evaluation of these reduction techniques mentioned. The program allows for both showing only similarities or showing only differences between embeddings. If similarities are to be shown, then edges will only appear in the hypercube if both embeddings include that exact edge. Similar for differences, edges will only appear in the hypercube when an edge only appears in the first tree embedding, but not the second embedding. Results from these reductions will be given in their respective sections as well.

4.1 Uninformed Reduction

The first, naive approach I attempted was a very simple matching technique: if two embeddings had edges that overlapped without rotation of any edges, then the edges matched. When edges matched, then the edge either was drawn in the similarity hypercube, or was removed from the first tree embedding in the differences hypercube. This approach, although very simple and not as successful as other techniques, was definitely the most efficient since each tree needed only to be parsed in order and therefore zero *dimension swaps*[2] were required.

This technique was the worst at finding similar edges, as was expected. Since this is a baseline for further reduction techniques that attempt to change the number of matching dimensions, this is far from a negative result.

4.2 Reduction by Dimension Count

A second approach attempted was a slightly more complicated attempt to match edges. For this evaluation, I reordered the dimensions with dimension swaps until the most frequently occurring dimensions were assigned the lowest dimension-numbers possible. The dimension-number is an arbitrary number that corresponds to a colour/dimension of the hypercube, while the frequency of a dimension is a count of the number of times a dimension is used in an embedding.

Often several dimensions may be used an equal number of times and therefore have identical frequencies; this may have resulted in poor performance for this reduction attempt. Since this reduction attempts to increase the number of similar edges, I would have to say that this reduction, although interesting, is about as effective as an uninformed reduction. Visually, it does not appear that more edges are similar when dimensions are ordered in this respect[3]. This method seemed promising at first, but since the number of edges wasn't reduced substantially from the

[2] *Dimension swaps* are a simple exchange between 2 dimensions. When 2 dimensions swap, all edges that lie in one dimension will move to the second dimension and vice versa. This operation requires a traversal through each node of the tree.

[3] I have not evaluated numerical results for this method nor have I evaluated any of the other results numerically.

uninformed result, this method is not recommended for evaluating embeddings to come up with better embedding algorithms.

A small positive: this algorithm performed well with respect to run time performance since it only required at most 5 dimension swaps per tree. Nevertheless, since the result of this method produces nothing of use, this positive is not good enough to give this method any advantage over much slower methods.

4.3 Greedy Minimization Reduction

This was an attempt to minimize the number of edges that are similar in two embeddings. I used the slow, but complete, search technique that checked the embedding of a stationary, non-swapping tree with a tree that permuted the dimension assignments through every possible assignment. This means that this method performed at most 5 dimension swaps $6!$ times. When the method found a lower number of matching edge assignments than the lowest so far, it saved the assignment. Although this reduction was very slow compared to the previously mentioned reductions, it is still reasonably fast enough so the wait for the check to complete on a computer with decent performance[4] is not too long.

The results for this method are not very useful for the intended problem of finding better algorithms for embedding trees. However, a new problem comes to mind when observing how few edges are in the similarities hypercube: embed two trees in a hypercube such that the tree has minimal average dilation. This might be a useful solution in the context of running *multiple* simultaneous communication trees on a hypercube multiprocessor to reduce the load on the edges in both trees, or perhaps the second tree could act as a reserve communications network in case the primary system fails. This may be a problem worth considering as a future project.

4.4 Greedy Maximization Reduction

Similar to the Minimization Reduction, this reduction tries to find the most similarity between two embeddings of a tree. This is undoubtedly the best reduction possible with any two embeddings and has the same performance characteristics as the Minimization Reduction.

Again, the results for this method, although the best possible for finding the most similar dimension assignments, are still disappointingly not useful for analysis. Since the reduction does not have success assigning many edges to the same dimension, it appears that the reduction approach taken for this project is not successful. The success of the reductions depends on the similarities of the embeddings and finding two *very* similar tree embeddings in a large domain[5] is practically hopeless. Locating two similar trees would be by extreme luck alone: it may be practical to find embeddings by permuting existing embeddings. Because the best stochastic method found [1] uses population based methods in a similar manner, it might be more practical to use existing em-

[4] Such as on a 1 GHz *Pentium* III.

[5] The domain of embeddings in H_6 may have size on the order of magnitude of $64!$, but this rough size overestimation does not take advantage of the Hamming distance properties of an embedding so it may be much smaller.

beddings as a strong influence on a new population based method than to rely on visualization for this problem. However, one must not discount a visualization system for the problem of finding similar tree embeddings quite yet; finding the most similar pair of trees could perhaps be done much faster on a different visualization system for a human perceptual system than a $\binom{n}{2}$ number crunching algorithm.

5 Results

This section provides further results somehow unrelated to the stated problem to find more techniques for finding perfect mappings of tree nodes to hypercube nodes. Results pertaining to reduction methods are described in their appropriate subsection in Section 4.

5.1 Dimension Swapping

Dimension swapping is not only useful in visualizing isomorphic representations of the same embedding, but it can be used to reduce the number of edges shown in an embedding comparison. In Section 4, several different techniques used dimension swapping in an attempt to find better isomorphic embeddings to either match two trees or make two trees as different as possible. The inflexibility of the system does not allow a user to choose which dimensions he wishes to swap; this process involves a large number ($6!$ in H_6) of permutations and therefore would make the system more complex than it should be. It happens that dimension swapping does not help solve problems of finding where the same edge is embedded; it can help perturb an embedding to determine dimensional permutations that result in a minimal number of shared edges for two embeddings in the same hypercube.

Since this project aims to satisfy routing in *real* systems, dimension swapping may have a practical application as well. Since most multiprocessor systems have tighter coupling between local nodes, such as an 8 processor system, than between remote nodes, such as between many 8 processor systems, a user may wish to favour the use of certain node links. This may have a weak correspondence to the reduction using dimension frequency in Section 4.2.

5.2 Edge Selection

The edge selection feature is perhaps the strongest simple feature of this visualization system. With a tree embedded in a hypercube, edges selected on the tree will also be selected on the hypercube and vice versa; edges selected in the hypercube when similarities are being shown will appear in both trees[6]. Edges will stay selected until the user pushes the `Change` button[7] or toggles the edge back to its unselected state. Since the program receives information from `Geomview` in a straight

[6] However, edges selected in trees when similarities are shown will not select edges in the other tree and will *toggle* the selection of the edge in the hypercube.

[7] This is kind of a bug, but not really a totally bad thing.

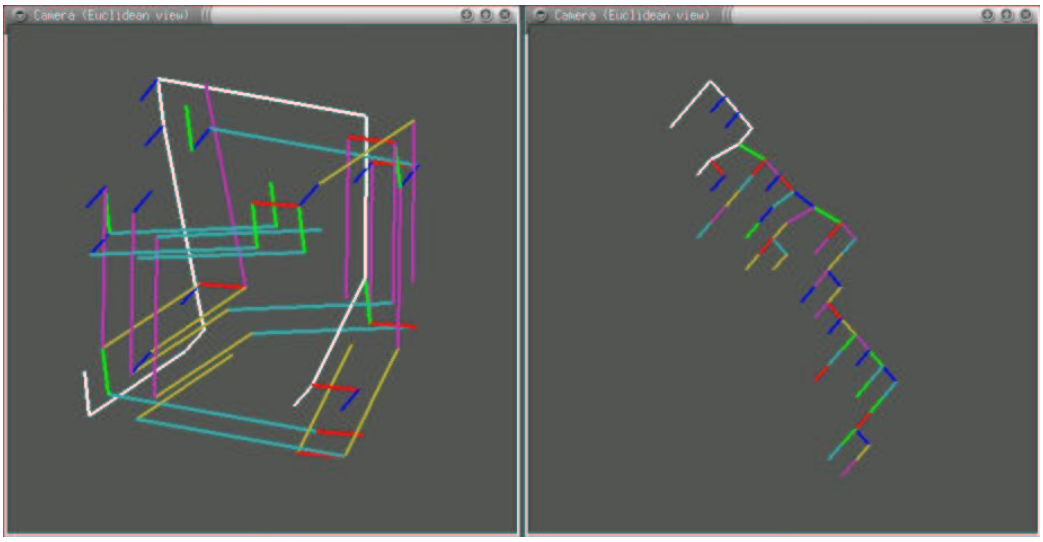

Figure 3: A path selected in a hypercube with an embedded tree (left) and the tree embedded in the hypercube (right)

text file[8] a restart of the GUI without deleting the input files and restarting `Geomview` will result in a replay of the selections since those files were created. This can be interesting sometimes and can always be undone by pressing the `Change` button, but it is annoying if the file gets too big.

Selecting edges allows for several useful features. Edge selection showing where similarities occur in a similarity comparison between two trees shows some interesting results. For some trees that look quite similar, it is hard to see how far apart similar edges in two embeddings actually are without using edge selection. Edge selection also allows for path traversal, explained in the next subsection.

5.3 Following Paths

Following from edge selection, a user may wish to follow a path in a tree and see the corresponding path appear in the embedded hypercube. See Figure 3 for an example. This is possible since each edge of the tree maintains its own state, which includes whether it is currently selected. Selecting edges to form paths is handy especially for non-comparative embedding tasks. As was stated previously, in Section 5.1, applications to efficient use of real networks of processors require less communication between more separated processors. By minimizing the communication between processors that have the poorest communication metrics, even the most efficient implementation of a complex system may still require a visual understanding of the system bottlenecks.

[8] I had troubles using pipes for this communication, but sending to `Geomview` uses pipes.

5.4 Linking Views

The views of two trees and a hypercube are effectively linked together. Edge selection allows what happens in one view to be mirrored to the appropriate edge in a second, or a third, view. Since each action is its own inverse, there is no need for an undo feature since it is built into the system. Also, when a user selects a different reduction technique, the request is sent to all views at the same time. Due to the limited nature of the interaction possible with this system, it may be useful to allow users to manually swap edge dimensions. Although this may complicate the system, it would provide flexibility to more interactively modify the similarities/differences hypercubes that appear from edge reductions.

5.5 Colour Keyed Dimensions

With a small enough number of dimensions, I was able to encode dimensions as unique colours: red, blue, green, cyan, magenta, and yellow were used for unselected edges in respective dimensions while white was used as a colour to represent selected edges in any dimension. I found it difficult to find selected white edges if I didn't know where to look, but I also found it convenient to rapidly select an edge to make it appear to flash. This of course makes the selected/unselected edge pop out and drags my attention to the region of interest. It was also much easier to find white paths that were longer than one edge in the hypercube since a continuous white line, not necessarily straight, also appears with a low amount of perceptual effort since it clearly contrasts with the other edges which were shorter and only occupying single dimensions. The problem of course becomes more difficult to find a selected edge when many edges are selected or when many edges are in the reduction/embedding.

5.6 Tree Layout

My previous attempts at tree layouts were not quite successful. Tree layout in a "nice" way was a difficult problem; bushy trees often had edges that meet where they are not supposed to or edges that cross. My tree layout concerns were put to rest after making use of the horizontal layout courtesy of Reingold and Tilford [7]. I will not go into detail on how this algorithm works, but will say that this algorithm provides a much improved look to the binary trees and allows edge selection in trees where my previous attempts would have proven difficult.

6 Conclusions

This paper presented a project that was capable of visualizing a hypercube of up to H_6. The project performed well in displaying the embedding of binary trees in the hypercube as well as several other tasks such as edge picking and displaying trees with a nice layout. Although the purpose of the project was intended to evaluate embeddings of similar trees to search visually for better embedding techniques, there was no evidence that modifying the dimensions with simple swap moves would help. The visualization reduction methods used only work on non-isomorphic but

very similarly structured embeddings of the same tree, although no such embeddings have been found. A different visualization system may be helpful in this respect.

The major success of this project has to be how selected binary tree paths can be visualized within a hypercube structure. A result such as this provides some insight to interesting paths in single embedded trees. Another interesting result was the potential interesting problem that involved embedding two binary trees in the same hypercube.

References

[1] M. Aderhold and J. Slack. Embedding balanced binary trees in the hypercube. University of British Columbia, April 2003.

[2] I. Havel and P. Liebl. One-legged caterpillars span hypercubes. *Journal of Graph Theory*, 10:69–77, 1969.

[3] I. Havel and J. Morávek. B-valuations of graphs. *Czechoslvak Math. J.*, 22:338–351, 1972.

[4] S. Helmer and A. Eisenberg. Exploring issues of embedding color balanced binary trees into 64 node hypercubes. University of British Columbia, December 2002.

[5] L. Nebesky. On quasistars in n-cubes. *Casopis pro Pestovani Matematiky*, 109:153–156, 1984.

[6] M. Phillips. Geomview/oogl release 1.8.0. Geometry Technologies, Inc., November 2000.

[7] E. M. Reingold and J. S. Tilford. Tidier drawings of trees. *IEEE Transactions on Software Engineering*, 7:223–228, 1981.

[8] G. Smedley. Algorithms for embedding binary trees into hypercubes. Master's thesis, University of British Columbia, 1989.

[9] A. S. Wagner. Embedding all binary trees in the hypercube. *Journal of Parallel and Distributed Computing*, 18:33–43, 1993.

[10] A. S. Wagner and D. Corneil. Embedding trees in the hypercube is NP-complete. *SIAM Journal on Computing*, 19(4):570–590, June 1990.