A SEQUENCED HYPERCUBE TOPOLOGY FOR A MASSIVELY-PARALLEL DATABASE COMPUTER

Naphtali Rishe, Doron Tal, Qiang Li

School of Computer Science
Florida International University–
The State University of Florida at Miami
University Park, Miami, Florida 33199

Abstract

The architecture of a massively parallel multi-processor and multi-disk database computer is presented. The interprocessor communication network has a hypercube topology. The architecture requires selection of linear ordering of the nodes of a network of processors. A method is developed and presented here which can arrange the nodes in sequences efficient for management of data. Among the features of the produced sequences is that the size of the sequence can grow as the size of the hypercube grows, without changing the existing sub-sequences.

Keywords: Massively parallel architecture, Interprocessor network, Hypercube, Sequencing, Database computer.

1. Introduction

The massively parallel database machines offer a way, perhaps the only way, to meet the ever increasing demands of information processing. Most of the contemporary approaches seek to achieve this goal by increasing the size of the processor, memory and disk, and also by employing a large number of processors. The primary purpose of doing so is to increase the parallel processing power and the parallel secondary storage accessing power. However, in the overall structures of the typical systems, the processing units are one group, and the secondary storage units, possibly with some preprocessing power, are another. The processing unit group and the secondary storage group are connected by communication channels. The system's formation is still "Processors–I/O Channels–Secondary Storage". The traditional I/O channel bottleneck is still present. This problem remains no matter how many processors we add to the machine, because the throughput is dominated by the relatively poor performance of the disk and channel. One can see the analogy between the processor/disk split and the processor/memory split that leads to the von Neumann bottleneck.

This research has been supported in part by a grant from the Florida High Technology and Industry Council.

These shortcomings have motivated the proposal of a new architectural concept called Linear-throughput Semantic Database Machine (LSDM). We discuss here only those concepts of LSDM relevant to this paper, and more details can be found in [7]. LSDM consists of thousands processors coupled with disks. Each processor-disk unit consists of one or a few fairly powerful processors, a dedicated memory module and a small capacity disk, e.g., a 20 megabyte disk. The processor-disk units are linked into a tightly coupled network.

The processor-disk units comprising the machine work simultaneously on different segments of the same query and on concurrent queries as well, offering two levels of parallelism. The hosts receive streams of users' requests. Each request is dispatched to the processor having the best control of the relevant fragments of the database. The processor then decomposes the request into smaller operations and communicates them to subcontracting processors. The processors related to a request can communicate with each other to get the data necessary to carry out the operations concurrently. With the completion of a request, the results are sent to the host nodes and eventually back to the users.

Our database machine uses the Semantic Binary Database Model [5]. The Semantic Binary Database Model represents information of an application's world as a collection of elementary facts of two types: unary facts categorizing objects of the real world and binary facts establishing relationships of various kinds between pairs of objects. The purpose of the model is to provide a simple natural data-independent flexible and non-redundant specification of information.

In order to fully utilize the parallel processing power of the proposed architecture, it is essential to have an appropriate data structure so that the processor-disk units have balanced load. In our implementation of the Semantic Binary Database Model, we store the entire database, including all indexing information, in one logical file. (The file is organized in a B-tree like structure.) The file is partitioned into segments of the size that can be stored in small disks. There is no logical difference between one segment and another. This important property allows us to

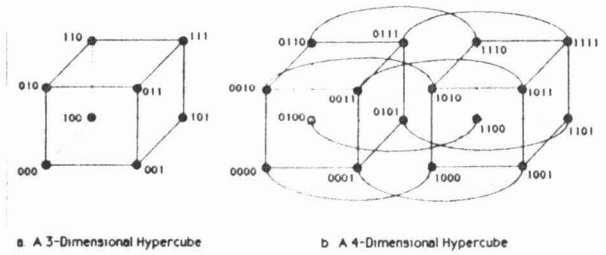

Figure 1: Hypercubes

partition the database based solely on the consideration of maximum parallelism. In addition, the data is stored with some small but intelligent duplications which greatly increase the locality of data accessing[6]. This implies a much lower data traffic between the processor-disk units, resulting in a higher degree of parallelism.

The n-cube topology has been chosen to connect the network. The n-cube [2,4,1,9] is defined as follows: Each of the 2^n nodes is labeled from 0 to 2^{n-1} by a unique binary string of length n. Two nodes are connected iff they differ in exactly one bit position. Figure 1 shows a 3-dimensional and a 4-dimensional hypercube.

2. The sequencing problem

2.1 Database machine sequencing problem

An essential problem is how to map a linear file onto the hypercube topology which is non-linear. Practical considerations, such as heavy data flow between consecutive file fragments, dictate that any two nodes of the network containing consecutive file fragments should be directly connected in the network. Therefore, we need a Hamiltonian path or loop (i.e. a path going through every node exactly once) in the hypercube so that the i-th partition of the file can be associated with the i-th node of the path. This is our *Requirement 1*. In addition, when more than one database is stored in the hypercube architecture in an overlapped fashion (i.e., each database is partitioned among all the nodes), the databases should be stored in non-coinciding paths to avoid unbalanced data traffic load. Therefore, many paths are needed.

There are many sequences that satisfy *Requirement 1*. At least one such sequence is guaranteed to exist because every hypercube has a Hamiltonian path since the hypercube satisfies the known criterion for Hamiltonian cycle: the graph does not contain the so-called theta-subgraph. As we shall show later, there are very many sequences which satisfy *Requirement 1*. We shall describe another requirement in term of sub-hypercubes.

Let H be a hypercube of dimension d. Let f be a part of a bit-string (binary number) of length d, i.e. for some positions in the bit-string it assigns bit values. For example, f gives the second and the fifth bits of an 8-dimensional hypercube node as follows: **?0??1???**, where **?** stands for **0** or 1. The sub-hypercube defined by f is the set of all nodes of H having the pattern f. There are as many nodes in the sub-hypercube as there are the possibilities to fill the question marks. Notice, that the nodes can be relabeled so that the sub-hypercube would itself be a hypercube. Its dimension is the number of question marks in f. For example, the nodes

00000 00001 00100 00101 01000 01001 01100 01101

form a 3-subcube for f=**0??0?** in the 5-cube; for f=**00???** the sub-hypercube is:

00000 00001 00010 00011 00100 00101 00110 00111.

Now we shall define a hierarchy of sub-hypercubes. Consider, for example the hierarchy of sub-hypercubes of the 3-cube as defined by the tree of f-patterns in Figure 2.

In the above example, we first varied bit #2, then #1, then #3, and received a hierarchy of sub-hypercubes consisting of 1 cube, 2 square, 4 segments, and 8 points. There are as many such hierarchical families of sub-hypercubes as there are orders in which to vary the bits. A more formal definition follows.

Let p be a permutation of position numbers from 1 to d. E.g., $p = (2,1,3)$ in the above example. The hierarchical family specified by p is the set of the following

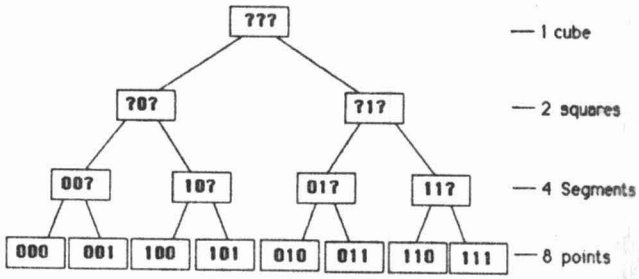

Figure 2: A Sub-hypercube Hierarchy

sub-hypercubes of dimensions 1 to d. For every D, for every f-pattern assigning constant bit values to positions $p[1], p[2], ..., p[d-D]$, the sub-hypercube specified by f is in the family. E.g., for the above example, the family consists of:

- the 2-dimensional sub-hypercubes specified by the f-patterns assigning constant bits to position $p[1] = 2$, i.e. to patterns **?0?** and **?1?**;
- the 1-dimensional sub-hypercubes specified by the f-patterns assigning constant bits to positions $p[1] = 2$ and $p[2] = 1$, i.e. to patterns **00?**, **10?**, **01?**, and **10?**;

- and so forth.

Requirement 2 of the sequence is that when a hierarchical family of sub-hypercubes is given, every sub-hypercube of the family should comprise a contiguous sub-sequence. This requirement has several purposes. First, the hypercube can be expanded without changing the existing logical sequence. Second, since a sub-hypercube can be easily identified in the sequence as a consecutive sub-sequence, data backup, trouble shooting and module replacement are much easier. Third, because of the simple mapping between the sub-hypercubes and the sequence, the connections between the positions in the sequence are expected to follow a systematic pattern, which will facilitate the analysis and simulation of the system.

There can be many sequences, depending on which sub-hypercubes are required to be consecutive in the sequence. For example, for the hierarchical family with $p = (1,2,3)$, the sequence is:

000 001 011 010 110 111 101 100;

for the family with $p = (2,3,1)$, the sequence is:

000 001 101 100 110 111 011 010

In the first sequence, the nodes of the square 000 001 011 010 are consecutive, and in the second they are not. It is the opposite for the square 000 001 101 100. (A square is a 2-dimensional sub-hypercube.)

2.2 Generalization of the sequencing problem

The sequencing problem in a hypercube has more general applications. Many applications need to map sequential data structures into a hypercube. Some applications, involve sequential operations between nodes of a hypercube, e.g., scanning. Sometimes, several logical sequences are needed at the same time.

The following is a general definition of the requirements. Let H be a d-dimensional hypercube. Let p be a specification of a hierarchical family of sub-hypercubes. Let $L = 2^d - 1$. A sequence $N(0), N(1), ..., N(L)$ of all the nodes of H is sought, satisfying:

1. For all i in $0..L-1$, $N(i)$ and $N(i+1)$ are adjacent in H. Also, $N(L)$ is adjacent to $N(0)$.
2. For every sub-hypercube S of dimension D in the family specified by p, for some i, the sub-sequence $N(i), N(i+1), ..., N(i+2^{D-1})$ is the sub-hypercube S.

3. The sequencing method

A problem equivalent to a subset of our problem has been solved in the Control Theory. That solution is known as the Gray Code [3] for sequencing of binary numbers. If it is adapted to our hypercube problem, we would have *Requirement 1* satisfied, as well as a portion of *Requirement 2*. The Gray Code would be consecutive for only one hierarchical family of sub-hypercubes, while we need to be able to have an arbitrary family as a parameter to sequencing. We call the family which happens to be consecutive in the Gray Code, "The natural family". For 5-dimensional hypercube, the natural family is specified by $p = (1,2,3,4,5)$. As another example, the followings are two of the sequences generated for a 4-cube with $p = (1,2,3,4)$ and $p = (3,1,4,2)$:

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

In the above sequences, every 4 nodes form a 2-cube, and every 8 nodes form a 3-cube.

We have solved the general case of the problem. Our sequencing algorithm follows. The proof of its correctness is in [8].

A parameter to the algorithm is an array p of position numbers which is a specification of a hierarchical family of sub-hypercubes.

We shall describe our algorithm in terms of a binary tree T. The tree T has the following properties. It is a full binary tree; each node of T has a label. The root has label "1". For any node, its label is greater by 1 than that

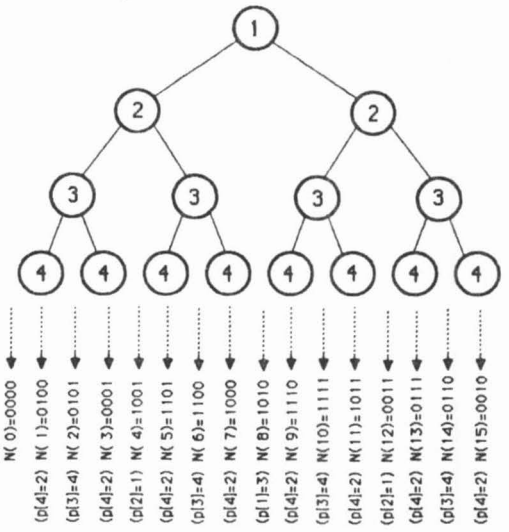

Figure 3a. The Tree For a 4-Cube

Figure 3b. The Sequence Generated For p=(3,1,4,2)

of its parent. All the leaves have the label "d", where d is the dimension of the hypercube. All the nodes at the same level have equal labels. Figure 3a shows an example of the tree T for $d = 4$.

We will use the $2^d - 1$ nodes of the tree, plus one additional node, to generate the 2^d nodes of the hypercube sequence. In the algorithm, a variable N is used. The

initial value of N is the first binary number, $N(0)$, in the sequence to be generated. The rest of the binary numbers of the hypercube are generated while traversing T in the inorder order. When a node of T is visited, one bit of the contents of N is flipped. If the label of the tree node is a number i, then the bit position to be flipped is $p[i]$. Every change in the value of N generates the next binary number in the output sequence. The bits are numbered from left to right. For example, if $N = 00011, p = (1,3,2,4,5), i = 3$, then $p[i] = 2$; we flip bit $\#2$ to receive the next $N = 01011$. Since the output sequence will be a cycle, the first binary number $N(0)$ can be an arbitrary node of the hypercube.

1. $N := 0$ (Let 0, i.e. a string of d zero-bits, be the first number in the sequence, without loss of generality.)
2. repeat
   (a) Get the next node from T according to the inorder traversal. Let i be the label of the node.
   (b) $N := N$ XOR $2^{d-p[i]}$ (flip the $p[i]$-th bit of N to get the next hypercube node number in the sequence).

   until all the nodes of the tree have been traversed.

Example. Using the tree of Figure 3a, the algorithm generates the sequence shown in Figure 3b for the sub-hypercube family defined by $p = (3,1,4,2)$.

Our program implementing the algorithm does not physically create a tree, but rather performs analytical calculation. The algorithm is linear in the number of nodes of the hypercube.

4. Discussion

The proposed system can lend itself to a stand alone database computer as well as a backend connected to hosts. Several processors can be identified as the hosts or interfaces to user hosts.

One of the primary goals of our architecture is to achieve a high throughput and approximate linearity of the throughput in the degree of parallelism. The degree of parallelism is the number of processor-disk units. The throughput is measured as the average number of transactions per time unit. The linearity is to be achieved for a typical transaction load comprised of a large number of relatively small, localizable queries and transactions. Under the current design (several thousands of processor-disk units), a conservative estimation of the throughput is more than 3000 simple queries per second per thousand processor-disk units, provided that the host interfaces have the same or higher throughput.

Currently, we are implementing our architecture on a network of 32-bit INMOS transputers and 20 megabyte Winchester disks.

Acknowledgement

The authors gratefully acknowledge the advice of David Barton, Nagarajan Prabhakaran, Istvan Ereny.

References

[1] S. Heller. *Directed Cube Network: A Practical Investigation.* Technical Report 253, MIT, 1985.

[2] K. Hwang and F. Briggs. *Computer Architecture and Parallel Processing.* McGraw-Hill, 1984.

[3] R.M. Klein. *Digital Computer Design*, pages 33-34. Prentice-Hall, Inc., Englewood Cliffs, NJ., 1977.

[4] F.P. Preparata and J. Vuillemin. The cube-connected cycles: a versatile network for parallel computation. *Communications of the ACM*, 24(5):300–309, May 1981.

[5] N. Rishe. *Database Design Fundamentals: A Structured Introduction to Database and a Structured Database Design Methodology.* Prentice-Hall, Englewood Cliffs, NJ, 1988.

[6] N. Rishe. *A File Structure for Semantic Database.* Technical Report SCS TR 88-001, Florida International University, 1988.

[7] N. Rishe, D. Tal, and Q. Li. The architecture for a massively parallel database machine. *The Euromicro Journal*, Aug 1988. In press.

[8] Naphtali Rishe, Doron Tal, and Qiang Li. *A Sequenced Hypercube Topology for a Multi-disk Multi-processor Database Computer.* Technical Report 88-006, Florida International University, Miami, FL., 1988.

[9] L.G. Valiant and G.J. Brebner. Universal schemes for parallel communication. In *STOC, ACM Conference Proceedings*, Milwaukee, 1981.