Greedy Partitioning Strategy for Banyan-Hypercube Networks

Abdelghani Bellaachia
Abdou Youssef

Department of Electrical Engineering and Computer Science
The George Washington University
Washington, D.C. 20052

Abstract

We have developed a partitioning strategy (FP) for banyan-hypercubes (BH's) networks, similar to the best fit strategy of memory allocation. In this paper, we will design and study a greedy partitioning (GP) strategy for BH's which to some extent resembles the first fit strategy of memory allocation. We simulated both FP and GP strategies. The simulation results show that the two strategies yield the same system performance. However, the FP strategy runs faster than the GP strategy and is preferable for practical considerations.

1. Introduction

Interconnections networks are among the most critical components in multiprocessor systems. Their efficiency depends on several factors. Among these factors is the partitionability of the network [2], [5], [7]. It provides an increase in system throughput as well as in the speedup of individual jobs [5]. Examples of partitionable systems include the Ncube [6], PASM [7], and the Connection Machine[4].

Partitioning consists of splitting the network into several subnetworks, called partitions, of different sizes. Each partition must have the same structure as the original network in order to use the same routing and mapping algorithms. Network partitioning software consists of three main components: (1) a data structure to keep track of available partitions in the system; (2) a partition allocation process that partitions the network and allocates partitions to requests; and (3) a deallocation (recombining) process that merges freed partitions with other free partitions in the system to form larger partitions and minimize fragmentation.

Banyan-Hypercube (BH) networks were recently introduced as fixed interconnection networks in [8]. They have many hypercube features such as self-routing and efficient embedding of rings and meshes. In addition, they have advantages over hypercubes in extendability, diameter, average distance, and embedding of trees, pyramids and multiple pyramids.

This paper will study a partitioning strategy of BH's, called greedy partitioning (GP), and compare it with another strategy, called first partitioning (FP), which we developed elsewhere [1]. It was shown that FP strategy has a better internal fragmentation than that of the buddy system on the hypercubes and both strategies have similar total fragmentation for large request sizes. The new greedy partitioning (GP) strategy of BH's always searches the smallest available partition to satisfy an incoming request. The recombining of a released partition is always done on its largest available buddy. The GP strategy uses the same data structure used for the FP strategy [1]. Algorithms for both allocation and recombining processes will be described, simulation results of both FP and GP strategies will be compared and discussed.

The paper is organized as follows. The next section gives some definitions and notations. Section 3 reviews the data structures used for partitioning the BH. In section 4, algorithms for the GP strategy are presented. Simulation results are discussed in Section 5. The last section presents conclusions and future directions.

2. Preliminaries and Definitions

Banyan-hypercubes are a synthesis of banyans [3] and hypercubes. A banyan-hypercube, denoted BH(0,h,k,s), where $h \leq k+1$ and s is power of 2, is the first h levels (from the base) of a (k+1)-level rectangular banyan of spread s, such that the nodes in each level are interconnected by a hypercube (dashed lines in Figure 1). The levels of the banyan are numbered from 0 to h-1 and the nodes in each level are labeled in binary from 0 to s^k-1. Therefore, each node is labeled by a pair (L,X) where $0 \leq L \leq h$ -1 and X is a cube address of the form $x_{k-1}...x_1x_0$ in the number system of base s. The total number of nodes in the network is hs^k. For a complete definition of the banyan-hypercube, the reader is referred to [10]. If h =k+1, the network is called a full banyan-hypercube. Figure 1 shows a full banyan-hypercube network.

Definition 1: A subbanyan-hypercube BH(b_i,l_i,k_i,s) of BH(0,h,k,s), where $0 \leq b_i \leq h$-1, $1 \leq l_i \leq h$, and $0 \leq k_i \leq k$, is a banyan-hypercube of l_i levels starting at level b_i such that the nodes in each level are interconnected in a k_i-cube.

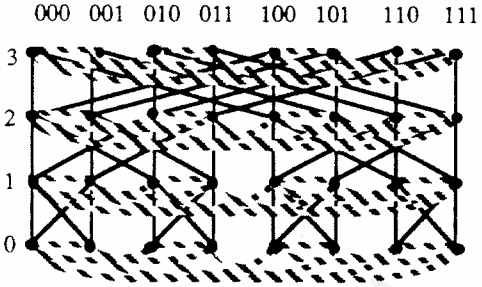
TH0363-2/91/0000/0487$01.00 © 1991 IEEE

The base level b_i must satisfy the following conditions:

(1) $0 \le b_i \le k_i+1-l_i$ if $1<l_i \le h$ $(l_i \ne 1)$

(2) $0 \le b_i \le h-1$ if $l_i=1$

Note that the labels of the levels of BH(b_i,l_i,k_i,s) are b_i, b_i+1, b_i+2,..., b_i+l_i-1. Throughout this paper b_i is the *base* of the subbanyan-hypercube, k_i is the *logarithmic width*, s^{ki} is the *width*, and l_i is the *height*.



(a) BH(0,4,3,2)

Figure 1

Definition 2: An (l_i,k_i)-partition is any subbanyan-hypercube whose height is l_i and logarithmic width is k_i, where $l_i \le k_i+1$. The size of an (l_i,k_i)-partition is $l_i\, s^{ki}$.

For practical considerations, the value of s is preferred to be 2 or 4. Throughout this paper only banyan-hypercubes with spread s=2 are treated. In this case each node in the network is identified by its level and its cube-address $x_{k-1}...x_1x_0$, where x_i is either 0 or 1.

Each partition BH_i is identified by a triplet (b_i,l_i,cube address), where cube address is a k-digit address and each digit is either 0, 1, * where * is "don't care". The cube address has k_i *'s in the k_i least significant bit position, i.e. $a_{k-1}a_{k-2}...a_{ki}*^{k_i}$ where a_i = 0 or 1 for $k_i<a_i<k$.

Theorem 1: The total number of (l_i,k_i)-partitions in a BH(0,h,k,2) is:

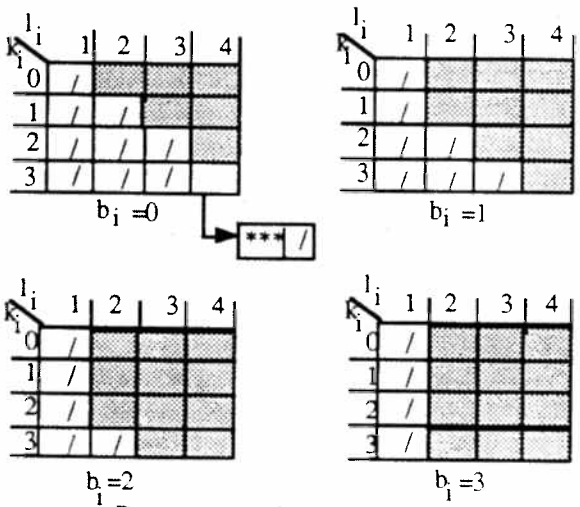
$$P_{h,k}(l_i,k_i)=\begin{cases}(k_i-l_i+2)2^{k-k_i} & \text{if } l_i \ne 1\\ h2^{k-k_i} & \text{if } l_i=1\end{cases}$$

Proof: The proof is given in [1].

3. Network Data Structures

During partitioning, some partitions may become free, either when a given task completes its execution or when splitting a partition into smaller partitions to accommodate an incoming task with a small request. In either case, a data structure is needed to record available partitions. Two data structures were presented for the partitioning of the banyan-hypercube [1]. They are reviewed in this section.

Since each partition in the network is characterized by its base level, its number of levels, its logarithmic width, and its cube address, a three-dimensional array, **P**, is first considered, where each entry points to a linked list of free partitions. The nodes of the linked list pointed to by the entry (b_i,k_i,l_i) contain the cube addresses of those partitions that have the base level b_i, the number of levels l_i, and the logarithmic width k_i. Initially, all entries point to null, except for the entry **P**[0,k,l], corresponding to the original network. Figure 2 shows the data structure of the BH(0,4,3,2).



Data structure for BH(0,4,3,2)

Figure 2

The unused entries (shaded area) in the array correspond to the subnetworks which do not satisfy either conditions (1) or (2) defined in the previous section. As can be seen in the example given in Figure 2, the array is very sparse.

The second data structure is a mapping of the 3-dimensional array **P** into a one-dimensional array, called **PART**. The size of **PART** is equal to the number of the used entries in **P**. The mapping function of an element of **P** into an element of **PART** is given in [1]. The space complexity of both data structures is given in [1].

4. The Greedy Partitioning Strategy

To partition the banyan-hypercube, a greedy partitioning (GP) is proposed. In [1], the banyan-hypercube was shown to be partitioned along two directions: horizontally and vertically. The vertical partitioning uses the binary buddy system, and it is possible only if the partition, BHi(b_i,l_i,k_i,2), is not a full banyan-hypercube: $b_i<k_i+1-l_i$ (condition (1) of definition 1). The horizontal partitioning can be done at any level of the network.

Some partitions can be split or combined horizontally and vertically, others can only be split (or combined) horizontally. For example if the network is full, it cannot be split vertically. During the process of splitting and combining partitions, we are interested in valid partitions. Formally, a partition $BH_i(b_i,l_i,k_i,2)$ in a $BH(0,h,k,2)$ is a valid partition if its cube address is of the form $a_{k-1}a_{k-2}...a_{k-ki+1}*^{k_i}$ and if b_i, l_i, and k_i satisfy conditions (1) and (2) in definition 1.

A valid partition $BH_i(b_i,l_i,k_i,2)$ with cube address $a_{k-1}a_{k-2}...a_{ki}*^{k_i}$ can be split vertically into two valid partitions of the form $BH_i(b_i,l_i,k_i-1,2)$ with cube addresses $a_{k-1}a_{k-2}...a_{ki}0*^{k_i-1}$ and $a_{k-1}a_{k-2}...a_{ki}1*^{k_i-1}$. The two partitions are called cube-buddies.

For every l in the range between 1 and l_i-1, $BH_i(b_i,l_i,k_i,2)$ can be split horizontally into two partitions $BH_l(b_i,l,k_i,2)$ and $BH_u(b_i+l,l_i-l,k_i,2)$, both of cube address $a_{k-1}a_{k-2}...a_{ki}*^{k_i}$. BH_l and BH_u are called level-buddies; BH_l is the lower level-buddy and BH_u is the upper level-buddy .

Combining, the opposite process of splitting, can be done horizontally and vertically as follows. If $BH_i(b_i,l_i,k_i,2)$ and $BH_j(b_j,l_j,k_j,2)$ are valid partitions with cube addresses $A_i=a_{k-1}a_{k-2}...a_{ki}*^{k_i}$ and $A_j=c_{k-1}c_{k-2}...c_{kj}*^{k_j}$, then:

- if $b_i=b_j$ and $l_i=l_j$ and $k_i=k_j$ and A_i and A_j agree in all bits except for a_{ki} and c_{ki} then the two partitions (cube-buddies) can be combined into a single partition $BH(b_i,l_i,k_i+1,2)$ with cube address $a_{k-1}a_{k-2}...a_{ki+1}*^{k_i+1}$.
- if $k_i=k_j$ and $b_j=b_i+l_i$ and the two cube addresses are identical, then the two partitions arelevel-buddies and can be combined into a single partition $BH(b_i,l_i+l_j,k_i,2)$ with cube address $a_{k-1}a_{k-2}...a_{ki}*^{k_i}$.

4.1 Allocation algorithm

The GP strategy uses a greedy allocation procedure which always allocates the smallest available partition that can accomodate the incoming request. To illustrate how the allocation procedure works, we will to use the first data structure described in section 3.

For an incoming task of size n, the computation of the size of the required partition is such that the internal fragmentation is minimized. The logarithmic width k_t and the height l_t of the required partition are such that $n \le l_t 2^{k_t}$[1].

Allocation of partitions always starts from the top available partitions in the network (i.e. high base levels). This is because all the top partitions with a higher base can be allocated at smaller base level but not vice-versa. For example in Figure 1 we can always allocate a (2,3)-partition either at base 0 or at base 2, but a (2,1)-partition can only be allocated at base 0.

Figure 3 depicts one entry of the data structure given in Figure 2, b_i=0, $0 \le k_i \le k$, and $1 \le l_i \le h$. In this figure, it is clear that if there is no available partition at the entry b_i=0, k_i=1, and l_i=1, then there are two possible ways to choose a larger partition : either a partition with larger logarithmic width k_j such that $k_i < k_j \le k$, or a partition with a larger number of levels l_j such that $l_i < l_j \le k_i+1-b_i$ if the network is full, otherwise $l_i < l_j \le h$.

The allocation procedure consists of three steps. The first step computes the set B of all possible values of the base of the (l_t,k_t)-partition. Those values are in the range from 0 to k_t-l_t+1 if $l_t \ne 1$, and in the range from 0 to h-1 if $l_t=1$ (definition 1). The search of a free partition is done in the second and third steps. The second step starts looking for a free (l_i,k_i)-partition, for all possible values of B, starting from the largest value, where initially $l_i=l_t$ and $k_i=k_t$. If there is a free partition

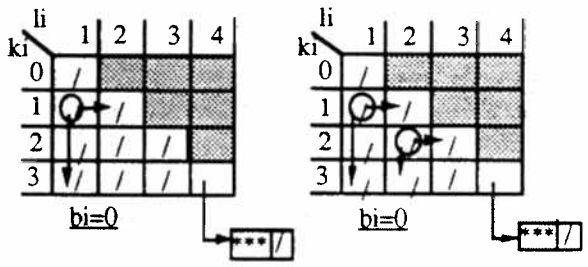


Figure 3

then it is allocated; otherwise, a larger partition should be searched in the third step. In this step, both (l_i,k_j)-partition and a (l_j,k_i)-partition are searched, for every possible value b_i in B for $k_i < k_j \le k$ and for $l_i < l_j \le k_i+1-b_i$ ($l_i < l_j \le h$ if the network is not full). If both (l_i,k_j)-partition and (l_j,k_i)-partition are found, the GP strategy allocates the smaller one; otherwise, if one of them is found then it is allocated. If there is no free partition at any base b_i, both l_i *and* k_i are incremented by one and the cycle is repeated, as shown in Figure 3 (b), until a partition is found or $k_i=k$ or $l_i=h$. If there is no free partition, then the request is queued for later scheduling.

Clearly, the time complexity is $O(h^2k)$.

4.2 Recombining algorithm

The last step of our partitioning system consists of recombining released partitions with available ones in order to reduce the total fragmentation in the system. Since each partition may have a cube-buddy freed partition or a level-buddy freed partition, the recombining is also a greedy procedure; it always chooses the buddy which, when combined with the released partition, results in a larger partition. Figure 4 (a) shows the case when the cube-buddy yields larger partition whereas Figure 4 (b) shows the case when the level-buddy yields a larger partition.

It is clear that the complexity of the recombining algorithm is $O(h)$ which is the time to search either an upper level-buddy or a lower level-buddy.

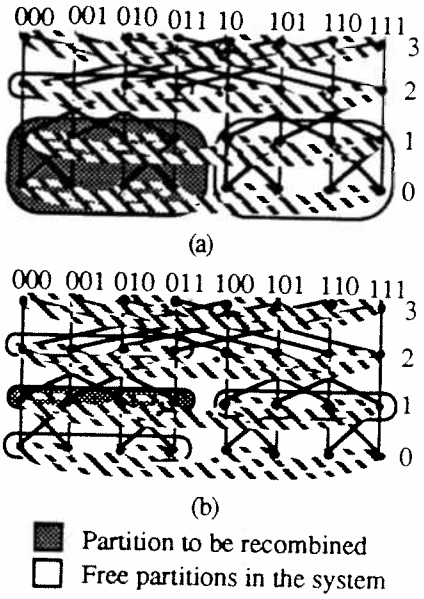


Figure 4

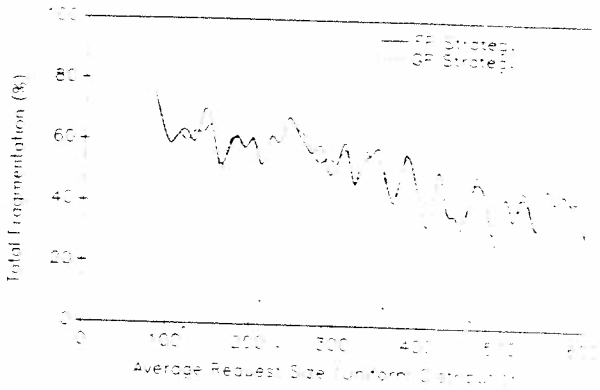


Figure 5

5. Simulations

We have simulated both GP and FP strategies on a BH of 1 K nodes, using uniform and exponential distributions for request sizes, and uniform distribution for task lifetime.

The fragmentation analysis of FP was presented in [1]. Similar fragmentation analysis was done for GP strategy. Internal, external, and total fragmentation of both strategies turns out to be very similar as shown in Figure 5. Other measures were computed for both strategies. These include the total number of finishing requests per unit of time, the turnaround time, and the total number of allocated nodes for incoming requests. All these measures were also very close for both strategies.

6. Conclusions

A new GP strategy for partitioning banyan-hypercubes is presented. It resembles the Best Fit memory allocation strategy. It always allocates the smallest available partition for an incoming request and recombines the released partition with its largest available buddy. The implementation of both allocation algorithm and recombining algorithm and their time complexities were discussed. We simulated GP and compared it to another partitioning strategy (FP) that we implemented elsewhere [1].

It is concluded that the BH yields the same performance under both partitioning strategies. However, the GP strategy has a bigger execution time than the FP strategy. Therefore, FP is preferred for practical considerations.

Future work includes the extension of this strategy to BH's with spread larger than two. In addition, this strategy will be rendered distributed to reduce partitioning overhead.

References

[1] A. Bellaachia and A. Youssef, "Partitioning on the Banyan-Hypercube Networks," *to appear in the Third Symposium on the Frontiers of Massively Parallel Computation '90.*

[2] M. Chen and K.G. Shin, "Processor Allocation in an N-Cube Multiprocessor Using Gray Codes," *IEEE Transactions on Computers*, Vol.C-36, No.12, pp. 1396-1407, December 1987.

[3] R. L. Goke, "Banyan Networks for Partitioning Multiprocessor Systems", Doctoral Dissertation, University of Florida, 1976.

[4] W. D. Hillis, *The connection Machine*, MIT Press, 1985.

[5] M. Jeng and H.J Siegel, "A Distributed Management Scheme for Partitionable Parallel Computers," *IEEE Transactions on parallel and Distributed Processing*, vol. 1, pp. 120-126, Jan. 1990.

[6] NCUBE Corp., *NCUBE/ten: An Overview*, Beaverton, OR, Nov. 1985.

[7] H.J. Siegel, T. Schwederski, J. T. Kuehn, and N. J. Davis IV, "An overview of the PASM parallel processing system," in *Computer Architecture*, D. D. Gajski, V. M. Milutinovic, H. J. Siegel, and B. P. Furth. Eds. Washigton, DC: IEEE Computer Society Press, 1987, pp. 387-407.

[8] A. Youssef, B. Narahari, "Banyan-Hypercube Networks," *IEEE Transactions on Parallel and Distributed* Systems, pp. 160-169, April 1990.