An Effective Structure for Algorithmic Design and a Parallel Prefix Algorithm on Metacube

Yamin Li and Shietung Peng
Department of Computer Science
Hosei University
Tokyo 184-8584 Japan
{yamin, speng}@k.hosei.ac.jp

Wanming Chu
Department of Computer Hardware
University of Aizu
Aizu-Wakamatsu 965-8580 Japan
w-chu@u-aizu.ac.jp

Abstract

Metacube is an attractive, hypercube-like interconnection network that can connect an extremely large number of nodes with a small node degree while keeping a relatively short diamemter. A Metacube MC(k, m) connects $2^{2^k m+k}$ nodes with only $k + m$ links per node. Metacube can be used to build parallel computing systems of very large scale with a small number of links per node. In this paper, we propose a new presentation of Metacube for algorithmic design on Metacube. Based on the new presentation, we give an efficient algorithm for parallel prefix computation on Metacubes that runs in $2^k m(k + 1) + k$ communication steps and $2^{k+1}m + 2k$ computation steps.

1. Introduction

The modern high-performance supercomputers consist of hundreds of thousands of CPUs. In the near future, the number of CPUs in supercomputers will reach to several millions. The interconnection networks (INs) in such supercomputers play an important role for achieving high performance of the supercomputers. A "good" interconnection network should use a small number of links and meanwhile keep the diameter as shorter as possible. The symmetric structure and efficient routing should also be considered.

The hypercube is one of the most versatile and efficient networks for parallel computation [1, 3, 4]. It can efficiently simulate any other network of the same size. Hence, the hypercube is an excellent and popular choice for the architecture of a multi-purpose parallel computer. However, one drawback to the hypercube is that the number of connections to each processor grows logarithmically with the size of the network, This will be a problem for very large parallel computers that contain tens of thousands or more processors.

In order to overcome this problem, several bounded-degree derivatives of hypercube have been proposed and analyzed in the literatures [1, 4, 9]. Among them are the butterfly, shuffle-exchange graph, de Bruijn graph, Benes network, and cube-connected-cycles (CCC). However, these networks have their own drawbacks that prevent them to be used as a practical architecture for very large parallel computers. Li and Peng proposed a new hypercube-like architecture, called dual-cube [5, 8]. A dual-cube uses hypercubes as basic components. A dual-cube with $m + 1$ links per node has 2^{m+1} m-cubes, called clusters. Each node has a link (cross-link) to connect that node to a node in another cluster. Two nodes in a dual-cube are connected via a link only if their addresses differ in one bit. A dual-cube cuts about half the number of links in the hypercube of the same size. A dual-cube retains most of the desirable properties of the hypercube, such as symmetrical and recursive structures.

Metacube [6, 7] (MC for short) is an extension of dual-cube that can connect extremely large number of nodes with small number of links, meanwhile it holds a short diameter and keeps the simplicity of routing algorithms. An MC(k, m) network can connect 2^{m2^k+k} nodes with $m + k$ links per node, where k is the dimension of the high-level cubes (class-cubes) and m is the dimension of the low-level cubes (clusters). For example, an MC(3,3) with 6 links per node can connect 2^{27}, or 134,217,728, nodes. When $k = 1$, Metacube becomes a dual-cube.

The prefix is an important primitive for data-parallel computing [2]. It takes an ordered set $[a_0, a_1, \ldots, a_{n-1}]$ of n elements and an associated binary operator $\otimes$ and returns the ordered set $[a_0, (a_0 \otimes a_1), \ldots, (a_0 \otimes a_1 \otimes \ldots \otimes a_{n-1})]$. The associative operation can be addition, multiplication, and maximum, *etc*. The parallel prefix computation is used by many applications. Some of them are radix sorting, quick sorting, carry-look-ahead circuits, evaluation of polynomials, and solution of linear recurrence equations.

In this paper, we propose a new presentation of Metacube for algorithmic design on Metacube. Based on this structure, we give an efficient algorithm for parallel prefix computation on Metacubes.

The rest of this paper is organized as follows. Section 2 describes the Metacube networks. Section 3 reviews parallel prefix computation and broadcasting on hypercube. Section 4 gives a new presentation of Metacube. Section 5 gives an efficient prefix algorithm on Metacube based on the new presentation of Metacube. Section 6 concludes the paper and presents some open questions.

2. Metacube Interconnection Networks

The MC network has a 2-level cube structure: a high-level cube represented by the leftmost k bits of the binary node address of $m2^k + k$ bits (these k bits serve as a class indicator); and low-level cubes, called clusters that forms the basic components in the network, represented by the m bits of the remaining $m2^k$ bits, which occupy the different portions of the $m2^k$ bits for the different classes.

More specifically, there are two parameters in an MC network, k and m. An MC(k, m) contains 2^k *classes*. Each class contains $2^{m(2^k-1)}$ *clusters*, and each cluster contains 2^m *nodes*. Therefore, an MC(k, m) uses $m2^k + k$ bits to identify a node and the total number of nodes is 2^n where $n = m2^k + k$. The value of k affects strongly the growth rate of the size of the network. An MC$(1, m)$ containing 2^{2m+1}nodes is called a *dual-cube*. Similarly, an MC$(2, m)$, an MC$(3, m)$, and an MC$(4, m)$ containing 2^{4m+2} nodes, 2^{8m+3} nodes, and 2^{16m+4} nodes, are called *quad-cube*, *oct-cube*, and *hex-cube*, respectively. Since an MC$(3, 3)$ contains 2^{27} nodes, the oct-cube is sufficient to construct practically parallel computers of very large size. The hex-cube is of theoretical interest only. Note that an MC$(0, m)$ is a hypercube.

The $(m2^k + k)$-bit node address in MC(k, m) is divided into three parts: a k-bit class ID, an $m(2^k - 1)$-bit cluster ID, and an m-bit node ID. The leftmost k-bit number defines a class of clusters (class ID). There are 2^k classes. Of each class, there are $2^{m(2^k-1)}$ clusters, an $m(2^k - 1)$-bit number identifies a cluster of the class (cluster ID). An m-bit number, located in a special portion of the $m2^k$-bit (will be explained in the next paragraph) identifies a node within the cluster (node ID).

In the following discussion, we use the format of $(c, m_{h-1}, \ldots, m_1, m_0)$ to denote the node address where $h = 2^k$, c is the k-bit class ID, m_c is the m-bit node ID, and $(m_{h-1}, \ldots, m_{c+1}, m_{c-1}, \ldots, m_0)$ is the $m(2^k - 1)$-bit cluster ID. Figure 1 shows the format of the address.

An MC(k, m) is constructed with the following method. Within a cluster, the m-bit node ID forms an m-cube with

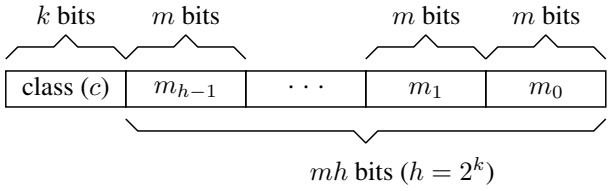

Figure 1. Format of a node address for an MC($\boldsymbol{k}, \boldsymbol{m}$)

m links or *cube-edges*. For a cluster of class i ($i = 0, 1, \ldots, h-1$), the node ID within the cluster is m_i, while $(m_{h-1}, \ldots, m_{i+1}, m_{i-1}, \ldots, m_0)$ is the cluster ID. The links that connect nodes in different clusters are defined as follows. For any two nodes whose addresses differ only in a bit position in the class field, there is a link, a *cross-edge*, connecting these two nodes. This is, the k-bit in the class field defines a high-level k-cube which connects those nodes whose addresses except class field are the same. A cluster then can be considered as a low-level m-cube.

The addresses of two nodes connected by a cross-edge differ only in one bit within the k-bit class field and there is no direct connection among the clusters of the same class. Therefore, a node in an MC(k, m) has $m + k$ links: m links construct an m-cube cluster and k links construct a k-cube. For example, the three neighbors within the cluster of the node with address $(01, 111, 101, 110, 000)$ in an MC$(2, 3)$ have addresses $(01, 111, 101, 11\underline{\mathbf{1}}, 000)$, $(01, 111, 101, 1\underline{\mathbf{0}}0, 000)$, and $(01, 111, 101, \underline{\mathbf{0}}10, 000)$. The underlined bits are those that differ from the corresponding bits in the address of the referenced node. The two neighbors in the high-level cube are $(0\underline{\mathbf{0}}, 111, 101, 110, 000)$ and $(\underline{\mathbf{1}}1, 111, 101, 110, 000)$.

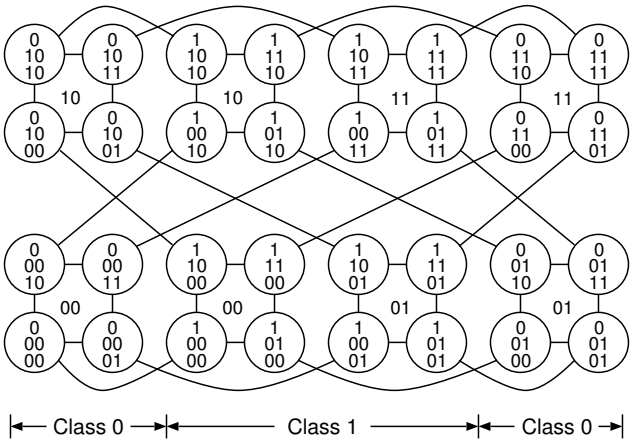

Figure 2. The Metacube MC(1,2)

Figure 2 shows the structure of an MC$(1, 2)$, where the

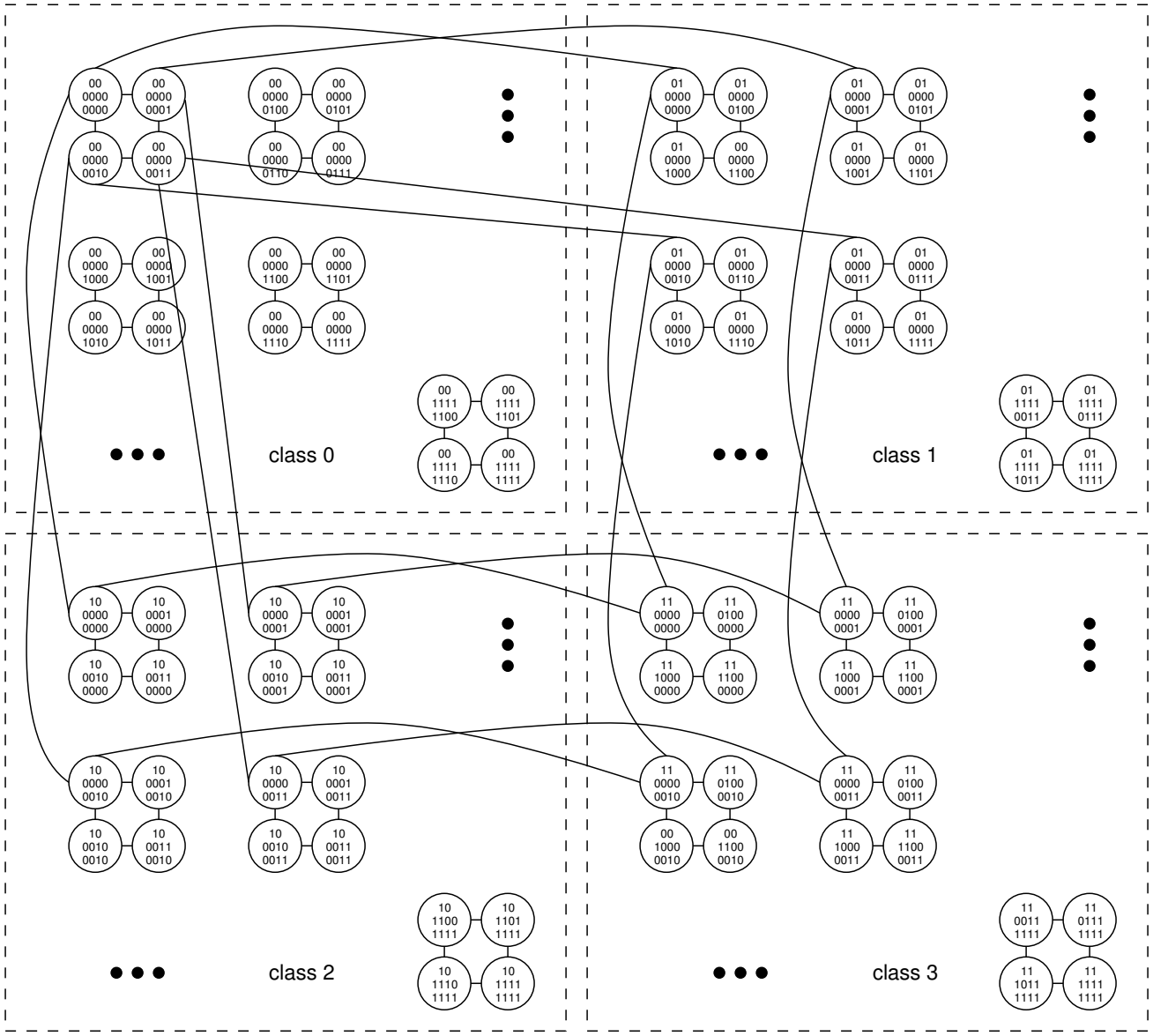

Figure 3. The Metacube MC(2,2)

cluster is a 2-cube and there are two classes. Each node has a cross-edge attached to a node of the other class. The binary number shown in the center of a cluster is the cluster ID. Figure 3 shows the structure of an MC$(2, 2)$, where the clusters in the same square are of the same class. In Figure 3, there are $2^{2(2^2-1)} = 64$ clusters in each square, and each cluster is a 2-cube. The figure shows only 4 high-level cubes which contain 4 nodes in the cluster 0 of the class 0, respectively.

The ratio of the total number of links in a hypercube to the total number of links in an MC network is equal to $n/(m+k)$, where $n = m2^k+k$. For example, for $k = 2$ and $m = 3$ ($n = 14$), each of the two networks contains 16384 nodes: a hypercube contains $2^{14} \times 14/2 = 114688$ links, whereas an MC network contains $2^{14} \times (3+2)/2 = 40960$ links. The reduction in the total number of links for this example is 73728 links or about 64%.

Table 1 lists the number of nodes a Metacube can connect. We can see that an MC$(2, 3)$ which requires 5 links per node is sufficient for building a large parallel computer system. To build a system of the same size with a hypercube, each node requires 14 links. The diameter of an MC$(2, 3)$ is 16, only 2 large than that of the hypercube. An MC(3,3) with 6 links per node can connect 2^{27}, or 134,217,728, nodes. It is extremely large. The topological properties of the Metacube are given in [6].

Table 1. Total number of nodes

Links/node	3	4	5	6	7	8
Hypercube	8	16	32	64	128	265
MC$(1, m)$	32	128	512	2,048	8,192	32,768
MC$(2, m)$	64	1,024	16,384	2^{18}	2^{22}	2^{26}
MC$(3, m)$	–	2,048	2^{19}	2^{27}	2^{35}	2^{43}
MC$(4, m)$	–	–	2^{20}	2^{36}	2^{52}	2^{68}

3. A New Presentation of Metacube

The presentation of Metacube given in Section 2 (cluster-based presentation) is elegant and can be used for developing efficient algorithms for collective communication in Metacube [7]. However, for design of basic algorithms such as prefix computation and sorting on Metacube, such a presentation is not proper due to its large number of clusters and lack of links among the clusters of different classes.

For design of efficient algorithms on Metacube, we introduce a new presentation of Metacube. The new presentation of Metacube is class-cube-based. That is, for the $m2^k + k$ bits of a node address of an MC(k, m), the rightmost k bits presents the k-cube (class-cube) and the rest are divided into m parts with each part containing 2^k bits. Each of the 2^k bits represents one of the 2^k classes in MC(k, m).

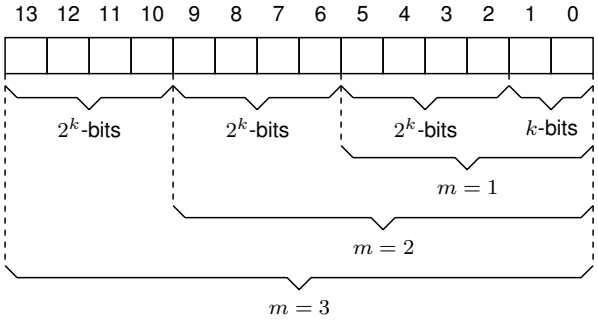

Figure 4. A new presentation of Metacube

Referring to Figure 4, we use $b_{n-1}\ldots b_1 b_0$ to denote the node address where $n = k+2^k m$. Bits $b_{k-1}\ldots b_1 b_0$ present the node class ID. The rest (cluster ID and node ID) are divided to m parts and each part contains 2^k bits. For a class $c = b_{n-1}\ldots b_1 b_0$, the m-bit node ID within the low-level m-cube is $b_{2^k(m-1)+c+k}, \ldots, b_{2^k+c+k}, b_{c+k}$ and the rest bits serve as the cluster ID. Note that Figure 4 actually shows the address format of the MC(2,3).

There are k links per node within a k-cube and there are m links per node within a cluster (m-cube). For example, the three neighbors within the cluster of the node with address $(1100, 0110, 1011, 10)$ in an MC$(2, 3)$ have addresses $(1100, 0110, 1\underline{1}11, 10)$, $(1100, 0\underline{0}10, 1011, 10)$, and $(1\underline{0}00, 0110, 1011, 10)$. The underlined bits are those that differ from the corresponding bits in the address of the referenced node. The two neighbors in the high-level cube are $(1100, 0110, 1011, 1\underline{1})$ and $(1100, 0110, 1011, \underline{0}0)$. Using this presentation, the Metacubes MC(1,2) and MC(2,1) are shown in Figures 5 and 6, respectively.

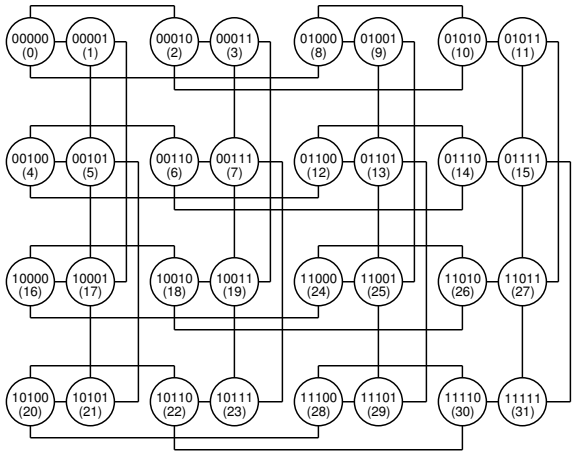

Figure 5. The Metacube MC(1,2)

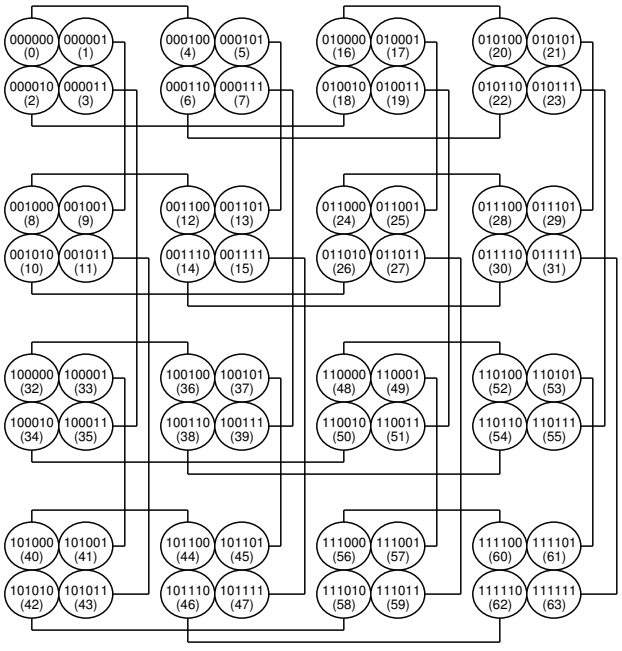

Figure 6. The Metacube MC(2,1)

Intuitively, if we view a k-cube as a "supernode" the MC(k, m) can be considered as a "virtual" $m2^k$-cube in

which a supernode has a link to other supernode at each of the $m2^k$ dimensions.

This presentation is suitable for design algorithms on Metacube that use the incremental method as the ascend/descend algorithms in hypercube. Since the k-cube is presented by the rightmost k bits, the operations at each of the 2^k bits can be handled through a broadcast or a hamiltonian walk in k-cube. In this paper, we used the described methodology to design an efficient algorithm for parallel prefix computation on Metacube. We describe this algorithm in the next section.

4. Parallel Prefix on Metacube

The prefix algorithm is a basic component of a large class of algorithms, in other words, it has many applications.

Prefix takes an ordered set $[a_0, a_1, \ldots, a_{n-1}]$ of n elements and an associated binary operator $\otimes$ and returns the ordered set

$$[a_0, (a_0 \otimes a_1), \ldots, (a_0 \otimes a_1 \otimes \ldots \otimes a_{n-1})]$$

For example, if $\otimes$ is $+$ (addition),

$$\text{prefix\_sum}([1, 2, 3, 4, 5, 6, 7, 8]) = [1, 3, 6, 10, 15, 21, 28, 36]$$

4.1 Parallel Prefix on Hypercube

The parallel prefix computation can be done efficiently in hypercube [1]. Assume that each node u_i, $0 \leq i \leq 2^d - 1$, in a d-cube holds a number a_i. The algorithm for parallel prefix computation in d-cube is an ascend algorithm in which each node successively communicates with its neighbors in dimension order from 0 to $d-1$. Each node deals with two variables: $total$ that gives the *total* computation result in the current subcube and p that gives the result of parallel prefix computation in the same subcube. $total$ is needed for the computation in the next dimension and is sent to a neighbor of the node. Finally, p becomes the required prefix within the entire d-cube. For completeness, we include it as Algorithm 1.

From Algorithm 1, we conclude that parallel-prefix computation in d-cube can be done in d communication steps and $2d$ computation steps assuming that, for each node, sending and receiving a message takes one communication step.

Next, we review broadcasting in hypercube that will be used as a subroutine for the proposed prefix algorithm on Metacube. It is well known that broadcasting from a node v in a d-cube can be done in d communication steps using the binomial tree of the d-cube rooted at v. We use procedure broadcast_HC(Q_d, $source\_id$, $message$) to present the algorithm that broadcasts *message* from source node to other nodes in d-cube Q_d.

Algorithm 1: prefix_HC (Q_d, my_id, my_number, p)
begin
  $p \leftarrow my\_number$;
  $total \leftarrow p$;
  for $i \leftarrow 0$ to $d-1$ **do**
    $partner \leftarrow my\_id$ XOR 2^i;
    send $total$ to $partner$;
    receive $number$ from $partner$;
    $total \leftarrow total \otimes number$;
    if ($partner < my\_id$)
      $p \leftarrow p \otimes number$;
    endif
  endfor
end

4.2 Proposed Prefix Algorithm on Metacube

Now, we are ready to give an efficient parallel prefix algorithm on Metacube. Similar to the parallel prefix algorithm on hypercube, we develop our algorithm based on the incremental method on dimension: start from dimension 0 and then move to the left one-bit per iteration. The first (right-most) k-bit of the address is the class ID of a node. These k bits can be dealt with by the parallel prefix algorithm in hypercube.

Then we deal with the rest $2^k m$ bits one by one from right to left. For each bit i, $i = k, k+1, \ldots, 2^k m + k - 1$, only the node u_i that satisfies $(((i-k) \text{ MOD } m) == c)$ has a link to node u'_i where c is the class ID of node u_i and u'_i differs only in ith bit with u_i. That is, only one node $u_{(i-k) \text{ MOD } m}$ in a k-cube can receive the data from a node $u'_{(i-k) \text{ MOD } m}$ outside the k-cube. From Subsection 4.1, we know that the data sent by node u_i to node u'_i is the variable $total$, and all the data within the k-cube should have the same value. Therefore, we just need an one-to-all broadcast within the k-cube after a node received the $total$. The parallel prefix algorithm on Metacube is shown in Algorithm 2 in which we use the variable j to count each of m, $(0 \leq j \leq m-1)$ and i to count 2^k bits, $(0 \leq i \leq 2^k - 1)$.

Figures 7-12 show each step of the parallel prefix sum computation on a MC(2,1) based on Algorithm 3. The MC(2,1) has 2^{2^2+2}, or 64 nodes. For simplicity, we assign each node a value as the same as the node address. We expect to get prefix_sum([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63]) = [0, 1, 3, 6, 10, 15, 21, 28, 36, 45, 55, 66, 78, 91, 105, 120, 136, 153, 171, 190, 210, 231, 253, 276, 300, 325, 351, 378, 406, 435, 465, 496, 528, 561, 595, 630, 666,

Algorithm 2: prefix_MC(MC$(k, m), my_id, my_number, p$)
begin
/* my_k_cube is the leftmost $m2^k$ bits of my_id */
/* my_class is the rightmost k bits of my_id */
$a \leftarrow my\_k\_cube$;
$c \leftarrow my\_class$;
prefix_HC(Q_k^a, c, my_number, p);
for $j \leftarrow 0$ to $m-1$ **do**
for $i \leftarrow 0$ to $2^k - 1$ **do**
if $(i = c)$
$b \leftarrow 2^k j + i + k$;
$partner \leftarrow my\_id$ XOR 2^b;
send $total$ to $partner$;
receive $number$ from $partner$;
endif
broadcast_HC($Q_k^a, c, number$);
$total \leftarrow total \otimes number$;
if $(partner < my\_id)$
$p \leftarrow p \otimes number$;
endif
endfor
endfor
end

703, 741, 780, 820, 861, 903, 946, 990, 1035, 1081, 1128, 1176, 1225, 1275, 1326, 1378, 1431, 1485, 1540, 1596, 1653, 1711, 1770, 1830, 1891, 1953, 2016].

Figure 7 gives the distribution of the input data. Figure 8 shows the values of $total$ and prefix p after the parallel prefix computation finished in k-cube. We can see that the $total$ values of all the nodes within a k-cube are the same. To this point, the bits 0 and 1 were dealt with. The value of $total$ is then sent by a node u to a node u' and broadcasted by the node u' to all the other nodes within the k-cube. The prefix on bit 2 is computed. The result is shown in Figure 9. Note that only a node of class 0 (left-up) in a k-cube can get the values of $total$ and broadcasts it inside the k-cube.

Figure 10 shows the case of the prefix computation on bit 3. First, a node of class 1 (right-up) in a k-cube gets the values of $total$ and broadcasts it inside the k-cube. Then the values of $total$ and prefix p are computed. Similarly, Figure 11 shows the case of the prefix computation on bit 4. First, a node of class 2 (left-down) in a k-cube gets the values of $total$ and broadcasts it inside the k-cube. Then the values of $total$ and prefix p are computed. Finally, Figure 12 shows the case of the prefix computation on bit 5. First, a node of class 3 (right-down) in a k-cube gets the values of $total$ and broadcasts it inside the k-cube. Then the values of $total$ and prefix p are computed. The p in each node is the final prefix result. We got what we want.

Theorem 1 Parallel prefix computation in MC(k, m) with $N = 2^{2^k m+k}$ nodes can be done in $2^k m(k+1) + k$ communication steps and $2^{k+1}m + 2k$ computation steps.

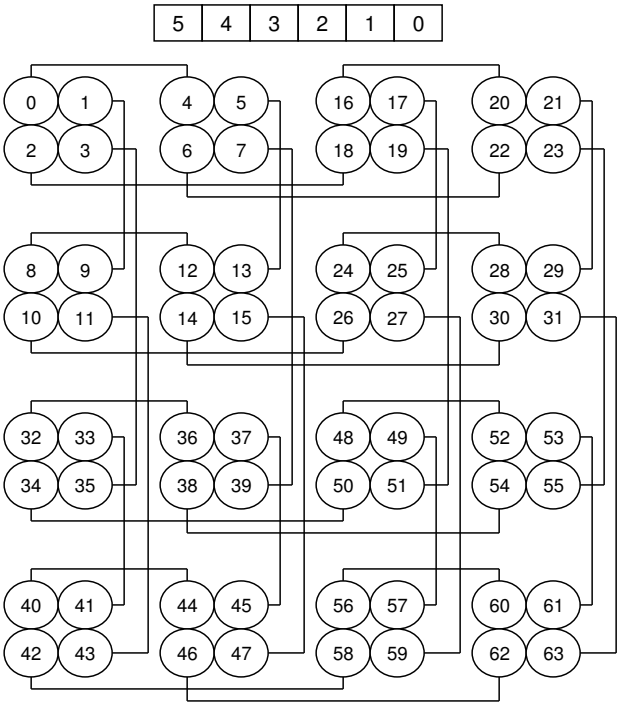

Figure 7. Original data distribution

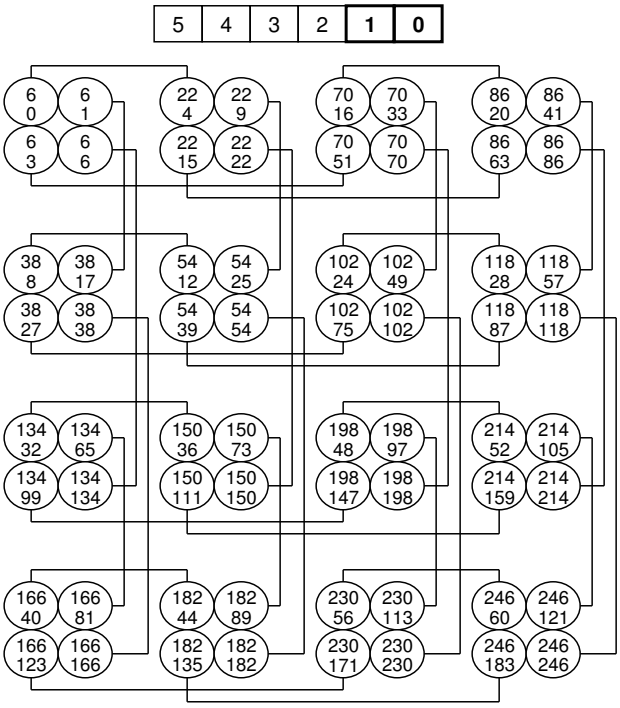

Figure 8. Prefix k-cube (bits 1 and 0)

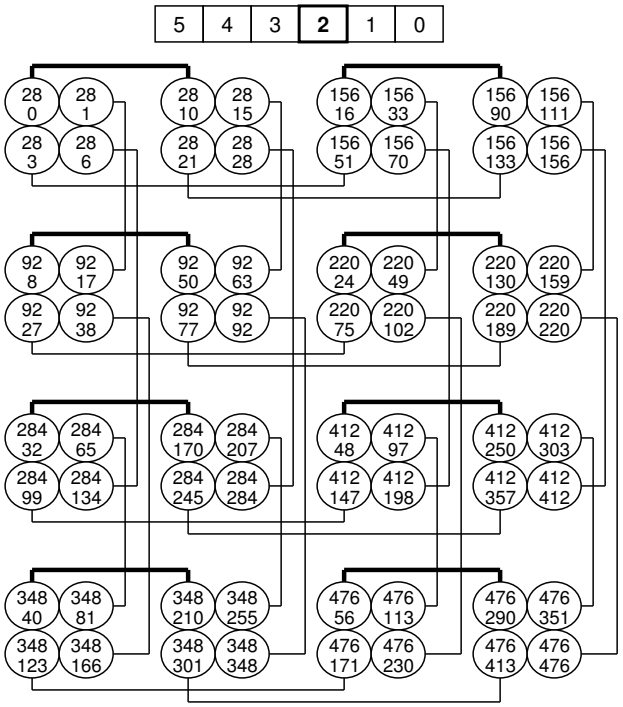

Figure 9. Prefix m-cube (bit 2)

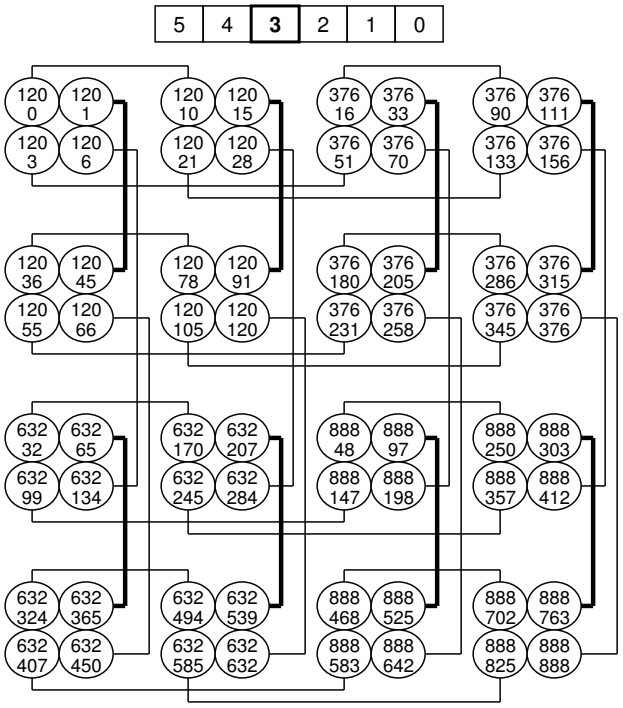

Figure 10. Prefix m-cube (bit 3)

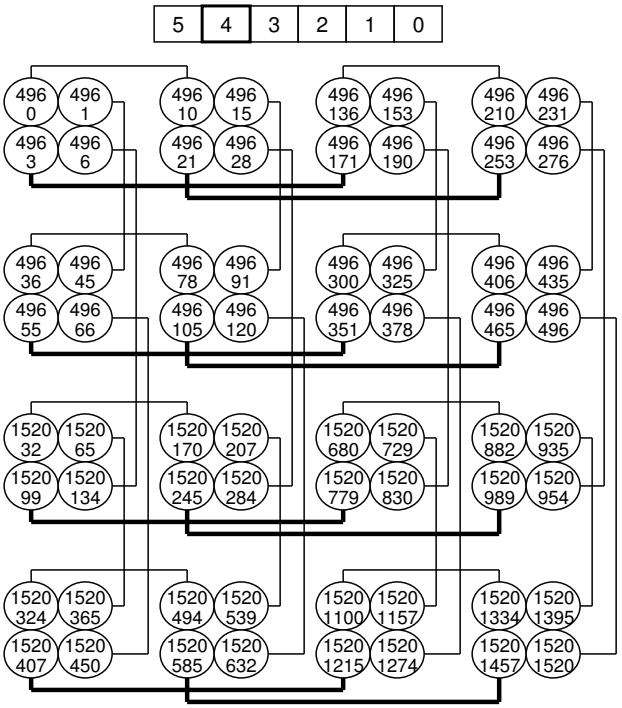

Figure 11. Prefix m-cube (bit 4)

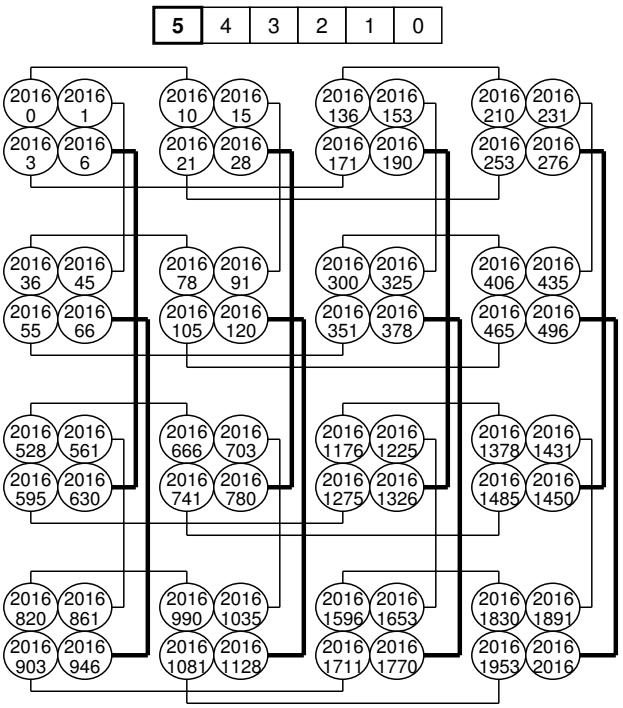

Figure 12. Prefix m-cube (bit 5)

Proof: We first show that the correctness of Algorithm 2. We define a 2^k-to-1 mapping from the set of vertices of MC(k, m) onto the set of vertices of Q_{m2^k} as follows: $f : b_{n-1} \dots b_1 b_0 \rightarrow b_{n-1} \dots b_k$, where

$n = m2^k + k$. From the definition of Metacube, the edges of MC(k, m) is mapped onto the edges of Q_{m2^k}. That is, MC(k, m) is mapped onto a $(m2^k)$-cube. After mapping, the prefix_HC(Q_k^a, c, my_number, p) becomes a local prefix computation and the broadcast_HC(Q_k^a, c, my_number, p) becomes local memory accesses. It is easy to see that the algorithm computes the prefixes of $N = 2^{m2^k+k}$ numbers in the mapped $(m2^k)$-cube with each node holding 2^k numbers. Therefore, from the correctness of prefix_HC(Q_k^a, c, my_number, p) and broadcast_HC$(Q_k^a, c, number)$, we conclude that Algorithm 2 computes prefixes correctly.

Next, we assume that at each communication step, each node can perform *send* and *receive* a single message. In the algorithm 2, the prefix within k-cube require k communication steps and $2k$ computation steps. For each bit of the rest $2^k m$ bits, in addition to the communication via an inter-cube link, a broadcast inside the k-cube is required. A broadcasting in k-cube takes k communication steps. Therefore, the algorithm runs in $2^k m(k + 1) + k$ communication steps and $2^{k+1}m + 2k$ computation steps. $\diamondsuit$

5. Concluding Remarks

In this paper, we presented a new presentation of Metacube and showed a parallel prefix algorithm on Metacube based on the new presentation. The algorithm is efficient in the sense that its running time is only k times of that in the hypercube of the same size ($k \leq 3$ in all practical cases). As an open question, it is worth to discover a parallel prefix algorithm on Metacube that runs faster than the proposed algorithm. Another direction for the future work is to investigate the possibility of using the new presentation for design efficient algorithms for basic computational problems such as sorting on Metacube.

References

[1] A. Grama, A. Gupta, G. Karypis, and V. Kumar. *Introduction to Parallel Computing*. Addison-Wesley, 2003.

[2] W. D. Hillis and G. L. Steele Jr. Data parallel algorithms. *Communications of the ACM*, 29(12):1170–1183, Dec. 1986.

[3] T. C. Lee and J. P. Hayes. A fault-tolerant communication scheme for hypercube computers. *IEEE Transactions on Computers*, 10(41):1242–1256, October 1992.

[4] F. T. Leighton. *Introduction to Parallel Algorithms and Architectures*. Morgan Kaufmann Publishers, 1992.

[5] Y. Li and S. Peng. Dual-cubes: a new interconnection network for high-performance computer clusters. In *Proceedings of the 2000 International Computer Symposium, Workshop on Computer Architecture*, pages 51–57, ChiaYi, Taiwan, December 2000.

[6] Y. Li, S. Peng, and W. Chu. Metacube - a new interconnection network for large scale parallel systems. *Australian Computer Science Communications*, 24(3):29–36, Jan. 2002.

[7] Y. Li, S. Peng, and W. Chu. Efficient communication in metacube: A new interconnection network. In *Proceedings of the International Symposium on Parallel Architectures, Algorithms and Networks (I-SPAN 2002)*, pages 165–170, Manila, Philippines, May 2002. IEEE Computer Society Press.

[8] Y. Li, S. Peng, and W. Chu. Efficient collective communications in dual-cube. *The Journal of Supercomputing*, 28(1):71–90, April 2004.

[9] F. P. Preparata and J. Vuillemin. The cube-connected cycles: a versatile network for parallel computation. *Commun. ACM*, 24:300–309, May 1981.