An Efficient Parallel Sorting Algorithm on Metacube Multiprocessors

Yamin Li[1], Shietung Peng[1], and Wanming Chu[2]

[1] Department of Computer Science
Hosei University
Tokyo 184-8584 Japan
{yamin, speng}@k.hosei.ac.jp
[2] Department of Computer Hardware
University of Aizu
Aizu-Wakamatsu 965-8580 Japan
w-chu@u-aizu.ac.jp

Abstract. Parallel sorting algorithms in hypercubes have been studied extensively. One of the practical parallel sorting algorithms is Bitonic Sort, which is implemented in $O(n^2)$ time for sorting $N = 2^n$ numbers in an n-cube. A versatile family of interconnection networks alternative to hypercube, called metacube, was proposed for building extremely large scale multiprocessor systems with a small number of links per node. A metacube MC(k, m) connects $2^{2^k m+k}$ nodes with only $k + m$ links per node. In this paper, we present an efficient sorting algorithm on metacube multiprocessors. The proposed sorting algorithm is based on the Batcher's bitonic sorting algorithm. In order to perform the parallel sorting efficiently in metacube, we give a new presentation of the metacube such that the communications required by the algorithm can be done efficiently with gather and scatter operations. The parallel bitonic sort algorithm implemented in metacubes with the new presentation runs in $O(2^k m + k)^2$ computation steps and $O(2^k m(2k + 1) + k)^2$ communication steps.

Keywords. Parallel algorithm, sort algorithm, hypercube, metacube

1 Introduction

The supercomputers consisting of hundreds of thousands nodes have been built recently. In near future, the number of nodes in a parallel system will reach to several millions. How to connect these large number of nodes is an important issue 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.

Hypercube is an interest interconnection network and is used in many supercomputers [3, 12, 13]. An n-dimensional hypercube, or n-cube, connects $N = 2^n$ nodes with n links per node: There is a link between two nodes whose n-bit binary addresses differ in a single bit position. However, the number of links increases logarithmically as the number of nodes in the system increases. A system with several millions of nodes requires each node to have more than 20 links if hypercube is applied as the interconnection network.

In order to reduce the required number of links per node while keeping the hypercube properties in a very large supercomputer, the metacube (MC) has been introduced [8, 9]. An MC(k, m) network can connect $2^{2^k m+k}$ nodes with $m + k$ links per node, where k is the dimension of the high-level cubes (classes) 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. A metacube can be implemented to a dual-cube ($k = 1$), a quad-cube ($k = 2$), an oct-cube ($k = 3$), or a hex-cube ($k = 4$). Rao and Chalamaiah gave routing and broadcasting algorithms for metacubes [11]; Jiang and Wu presented fault-tolerant routing in dual-cubes [4]; Wu and Wu showed the self-similarity and hamiltonicity of dual-cubes and gave the VLSI layout of dual-cubes [14]; Laia and Tsai embedded cycles into faulty dual-cubes [6]; Chen and Kao extended dual-cube idea to a dual-cube extensive network [2].

Sorting is an important component of many applications. The parallel sorting algorithms have been studied extensively. Batcher's $O(n^2)$-time bitonic sorting algorithm [1] for sorting $N = 2^n$ numbers is presently the practical deterministic sorting algorithm designed specially for parallel machines. Deterministic means that the sequence of comparisons is not data-dependent. More complicated $O(n \log n)$-time algorithms are not competitive to bitonic algorithm for $n < 20$ and are complex for the implementation on parallel machines [7, 10].

In this paper, we present a new parallel sorting algorithm on metacube multiprocessors which may consist of a very large number of nodes. The new sorting algorithm is based on bitonic sort on hypercubes. In order to perform the parallel sorting efficiently, we give a new presentation of the metacube that is class-based so that the communications between node pairs can be done more efficiently than the original cluster-based presentation. We also show how to use gather and scatter operations to speedup the communications in metacubes that are required by the algorithm.

2 Bitonic Sorting on Hypercube

Bitonic sort is based on repeatedly merging two *bitonic sequences* to form a larger bitonic sequence. A bitonic sequence is a sequence of values $(a_0, a_1, \ldots, a_{n-1})$ with the property that either (1) there exists an index i, where $0 \leq i \leq n-1$, such that $(a_0 \ldots, a_i)$ is monotonically increasing and $(a_{i+1}, \ldots, a_{n-1})$ is monotonically decreasing, or (2) there exists a cyclic shift of indices so that (1) is satisfied. For example, $(2, 3, 8, 13, 15, 14, 7, 0)$ is a bitonic sequence because it first increases and then decreases.

Let $s = (a_0, a_1, \ldots, a_{n-1})$ be a bitonic sequence such that $a_0 \leq a_1 \leq \ldots \leq a_{n/2-1}$ and $a_{n/2} \geq a_{n/2+1} \geq \ldots \geq a_{n-1}$. The bitonic sequence s can be sorted with *bitonic split* operation which halves the sequence into two bitonic sequences s_1 and s_2 such that all the values of s_1 are smaller than or equal to all the values of s_2 [5]. That is, the bitonic split operation performs:

$$s_1 = (\min\{a_0, a_{n/2}\}, \ldots, \min\{a_{n/2-1}, a_{n-1}\})$$
$$s_2 = (\max\{a_0, a_{n/2}\}, \ldots, \max\{a_{n/2-1}, a_{n-1}\})$$

For example, the bitonic sequence mentioned above $s = (2, 3, 8, 13, 15, 14, 7, 0)$ will be divided to two bitonic sequences $s_1 = (2, 3, 7, 0)$ and $s_2 = (15, 14, 8, 13)$. Note

that both the s_1 and s_2 are bitonic sequences. Thus, given a bitonic sequence, we can use bitonic splits recursively to obtain short bitonic sequences until we obtain sequences of size one, at which point the input bitonic sequence is sorted. This procedure of sorting a bitonic sequence using bitonic splits is called *bitonic merge* (BM).

Given a set of elements, we must transform them into a bitonic sequence. This can be done recursively doubling the size of bitonic sequence. The *bitonic sorting network* for sorting N numbers consists of $\log N$ bitonic sorting stages, where the ith stages is composed of $N/2^i$ alternating increasing and decreasing bitonic merges of size 2^i.

Figure 1 shows the block structure of a bitonic sorting network of size $N = 16$. $\oplus$BM[k] and $\ominus$BM[k] denote increasing and decreasing bitonic merging networks of size k, respectively. The last merging network ($\oplus$BM[16]) sorts the input.

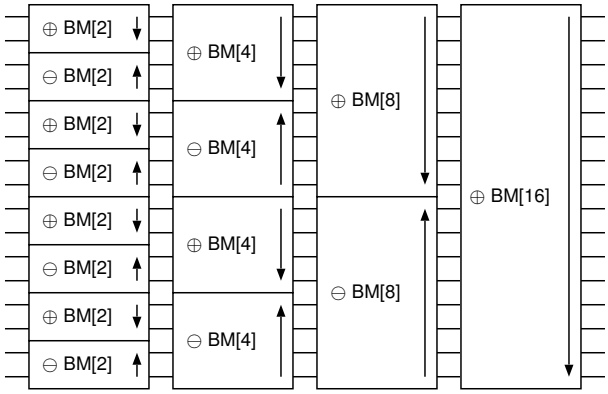

Fig. 1. Bitonic sorting network of size 16

```
Bitonic_sort_hypercube (my_id, my_number, n, result)
begin
  result ← my_number;
  for i ← 0 to n − 1 do
    for j ← i downto 0 do
      partner ← my_id XOR 2^j;
      send result to partner;
      receive number from partner;
      if (my_id AND 2^(i+1) ≠ my_id AND 2^j)                    /* max */
        if (number > result)
          result ← number;
      else                                                      /* min */
        if (number < result)
          result ← number;
end
```

Fig. 2. Bitonic sorting algorithm on n-cube

Figure 2 gives a bitonic sorting algorithm on an n-cube. The algorithm executes on every node in the n-cube in parallel. There are four parameters in the algorithm:

my_id is the binary node address; *my_number* is the number residing in the node; *n* is dimension of the hypercube; and *result* is the sorted number. The sorted numbers are in the same order as the node addresses. The outer **for** loop generates bitonic sequences in the dimension order of $i = 0$ to $n - 1$. Each iteration of the loop doubles the size of bitonic sequences. The bitonic merge is done by the inner **for** loop, which takes the order of $j = i$ to 0. In the current step j, nodes u and $u^{(j)}$ exchange their numbers each other through the link of the dimension j and compare the two numbers, where the addresses of nodes u and $u^{(j)}$ differs only in the dimension j. After the comparison, node u keeps the number as *result* based on the following rule: If the value of jth bit of the address differs from the value of $i + 1$st bit of the address, the node keeps the maximum of the two numbers; minimum otherwise.

A bitonic sorting example on a 4-cube is shown in Figure 3 where the sorted *result* is the same value as *my_id*.

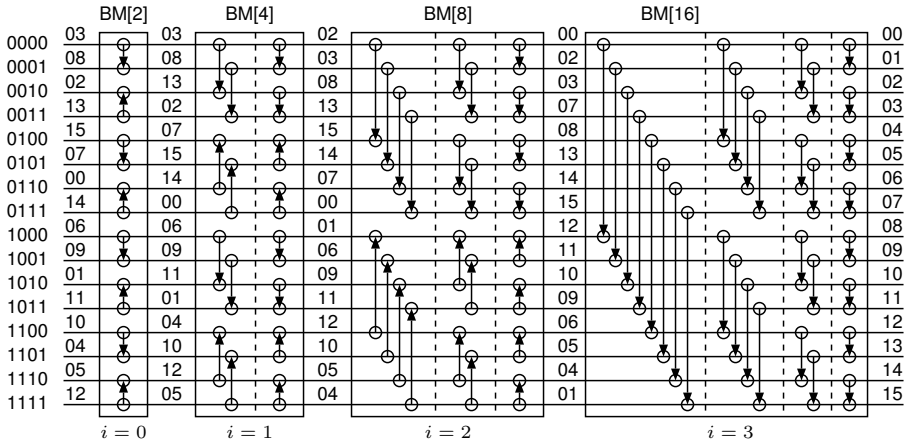

Fig. 3. Bitonic sorting on a 4-cube

The computational complexity for sorting $N = 2^n$ numbers in an n-cube is $O(n \times (n + 1)/2) = O(n^2)$. Similarly, the communications take $O(n^2)$ steps.

3 Metacube and Its New Presentation

Metacube (MC for short) is a versatile family of interconnection networks. It includes hypercube as a special case. In this section, we first introduce the original presentation of the MC architecture. Then, we give a new presentation of the MC that makes the efficient parallel sorting on MC possible.

The address format of an MC is shown in Figure 4. There are two parameters in an MC network, k and m. The value c of the most significant k bits defines the *class ID* of a node. The rest of the address consists of 2^k fields e_i, for $0 \leq i \leq 2^k - 1$, and each field has m bits. Therefore, an MC(k, m) uses $2^k m + k$ bits to identify a node and the total number of nodes is 2^n where $n = 2^k m + k$.

Referring to Figure 4, the nodes in an MC(k, m) are connected with the following method. There are $k+m$ links per node. For any two nodes whose addresses differ only in a bit position in the class ID, there is a link connecting these two nodes. This is, the k-bit class ID defines a *high-level* k-cube. The links in the k-cube is called *cross-edge*. Among the 2^k fields e_i, for $0 \leq i \leq 2^k - 1$, only the field e_c forms a *low-level* m-cube with m links, where c is the class ID of the node. The links in an m-cube is called *cube-edges*.

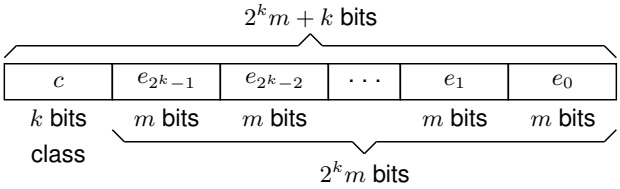

Fig. 4. Address format of metacube

For example, the three neighbors within the low-level m-cube of the node with address $(01, 111, 101, 110, 000)$ in an MC$(2, 3)$ have addresses $(01, 111, 101, 11\underline{1}, 000)$, $(01, 111, 101, 1\underline{0}0, 000)$, and $(01, 111, 101, \underline{0}10, 000)$. 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 k-cube are $(0\underline{0}, 111, 101, 110, 000)$ and $(\underline{1}1, 111, 101, 110, 000)$. An MC$(2, 1)$ is shown in Figure 5 in which we drew only the cross-edges of nodes $(xx, 0, 0, 0, 0)$ in addition to the cube-edges.

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 supercomputers of very large size. The hex-cube is of theoretical interest only. Note that an MC$(0, m)$ is a hypercube.

An MC(k, m) has much less links than the corresponding n-cube for $n = 2^k m + k$, and hence, the point-to-point communication in metacubes is little bit complex than that in hypercubes. In a metacube, communication between two nodes that are not in a same m-cube must go through the k-cube so that the different address bits of the two nodes in all the fields e_i, for $0 \leq i \leq 2^k - 1$, can be routed. The following example shows the routing path between nodes $s = 01, 111, 101, 110, 000$ and $t = 10, 110, 001, 100, 100$:

$$01, 111, 101, 110, 000 - 01, 111, 101, 1\underline{0}0, 000 - 0\underline{0}, 111, 101, 100, 000 -$$
$$00, 111, 101, 100, \underline{1}00 - \underline{1}0, 111, 101, 100, 100 - 10, 111, \underline{0}01, 100, 100 -$$
$$1\underline{1}, 111, 001, 100, 100 - 11, 11\underline{0}, 001, 100, 100 - 1\underline{0}, 110, 001, 100, 100$$

This address presentation is *cluster-based*. It is efficient for the collective communications. In order to perform the bitonic sorting in metacube networks efficiently, we use a new presentation, called *class-based*, as shown as in Figure 6.

The value c of the least significant k bits (bits $k-1, \ldots, 1, 0$) defines the class ID of a node. These k bits form a k-cube, which is the same as the high-level k-cube in

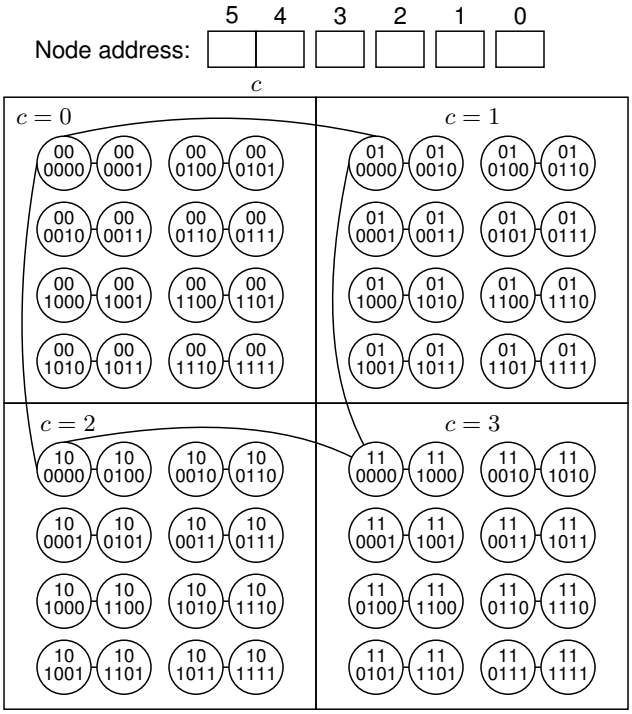

Fig. 5. The Metacube MC(2,1)

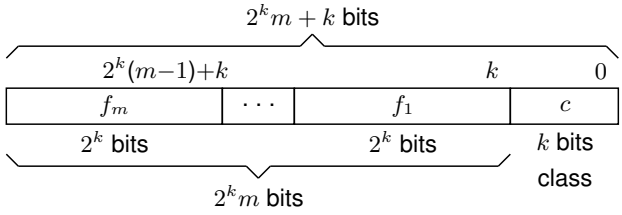

Fig. 6. The new presentation of metacube

the original address definition. Next to the class ID, there m fields f_i, for $1 \leq i \leq m$. Each field has 2^k bits. Among these 2^k bits in field f_i, for $1 \leq i \leq m$, there is only one bit position $j = 2^k(i-1) + c + k$ at which a link connects node u to another node $u^{(j)}$ whose address differs from u at the bit position j. We call this link *bridge*. The m bridges (each field contributes one bridge) form an m-cube which is equivalent to the low-level m-cube in the original address definition.

Figure 7 shows an MC(k, m) structure with $k = 2$ and $m = 1$ using the new address presentation. A node address has $2^k m + k = 2^2 \times 1 + 2 = 6$ bits shown in two rows: The 4-bit value in upper row is f_1 and the 2-bit value in lower row is the class ID c. Each node has $k + m$, or 3, links: Two links are used to form a k-cube and one link (bridge) connects two nodes whose addresses differ in $c+k$th bit position. For example,

nodes $0000, 11$ and $1000, 11$ of class 3 are connected with a bridge because these two nodes' addresses differ in $3 + 2$, or 5th (left-most) bit position.

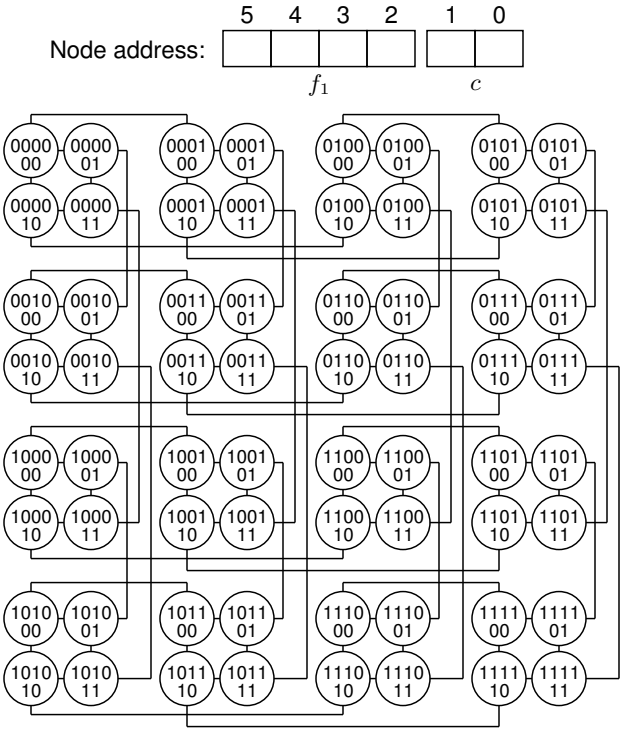

Fig. 7. The MC(2,1) with new presentation

Next, we explain the reasons of why we use this new presentation for the design of efficient algorithms such as sorting algorithm on metacube. Using the new presentation, if we map a k-cube to a single *super-node* then an MC(k, m) will be mapped onto a $2^k m$-cube; a bridge that connects nodes in distinct super-nodes in MC(k, m) becomes a hypercube edge. Mapping each 2-cube in Figure 7 into a single super-node, we get a 4-cube. Therefore, communications *among* k-cubes in MC(k, m) can be treated exactly as that in hypercube. For communications *inside* a k-cube, we can use collective communication procedures such as broadcast, gather, scatter etc., to collect or distribute data from a specific node to all other nodes in the k-cube.

For sorting on a metacube, we use the new address presentation of the metacubes and bitonic sorting algorithm. The communications between the node pairs in the bitonic sorting on a metacube can be done with gather and scatter operations through bridges efficiently which we describe in the next section.

4 Sorting on Metacube

In this section, we present a new sorting algorithm on metacube based on the parallel bitonic sorting. Given an MC(k, m), we assume that each node in MC(k, m) holds a

single element (number). The sorting algorithm compares and exchanges elements so that, at the end, all the elements are in the ascending order arranged by their addresses.

The parallel sorting on metacubes is based on the bitonic sorting on hypercubes. The basic operation is compare-and-exchange: Nodes u and $u^{(j)}$ whose addresses differ in jth bit position for $0 \leq j \leq 2^k m + k - 1$ send their elements to each other. Nodes u and $u^{(j)}$ retain the smaller number and bigger number, respectively, if $u < u^{(j)}$. However, there may be no direct links in some dimensions between nodes u and $u^{(j)}$ in a metacube.

As we described in the previous section, the node address has $2^k m + k$ bits (dimensions) and there are only $k + m$ links per node in an MC(k, m). For a dimension j, if $0 \leq j \leq k - 1$, there is a link between nodes u and $u^{(j)}$, otherwise, there is a link (bridge) only in the dimension j that satisfies $(j - k)$ MOD $2^k = c$ where c is the class ID of nodes u and $u^{(j)}$. Figure 8 shows the four bridges of the 2-cube in which the field f_1 of the node addresses is 0000 in an MC$(2, 1)$.

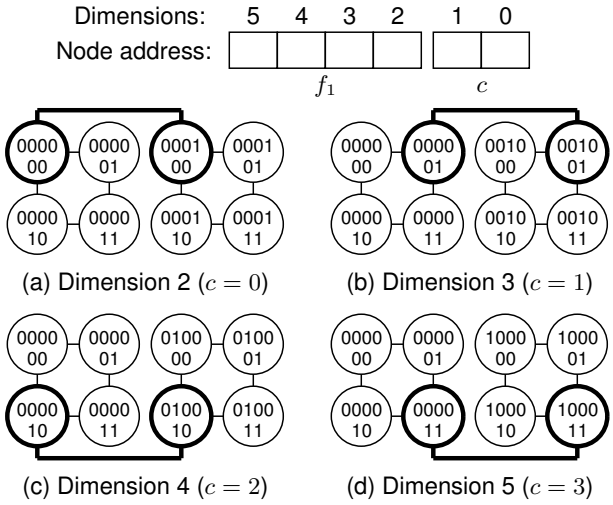

Fig. 8. Bridges in dimensions 2, 3, 4, and 5

In the case of Figure 8(b), the two node in every pair $(0, 8)$, $(1, 9)$, $(2, a)$, and $(3, b)$ communicate simultaneously. Nodes 1 and 9 can send their elements to each other directly because there is a link (bridge) between the two nodes. The other three pairs cannot do it directly due to the lack of the direct links: They must go through the bridge that links nodes 1 and 9.

Because nodes 0, 1, 2, and 3 are in the same 2-cube, we can *gather* elements of the four nodes to node 1, send the gathered elements to node 9 via the bridge, and then *scatter* the received elements to nodes 8, 9, a, and b. We can do the similar operations in the opposite direction.

Generally, in the gather operation, a single node collects a unique message from each node (also called concatenation). In the scatter operation, a single node sends a unique message to every other node (also called a one-to-all personalized communi-

cation). The scatter operation is exactly the inverse of the gather operation. Figure 9 shows the communication steps for the gather and scatter operations on the two 2-cubes of Figure 8(b).

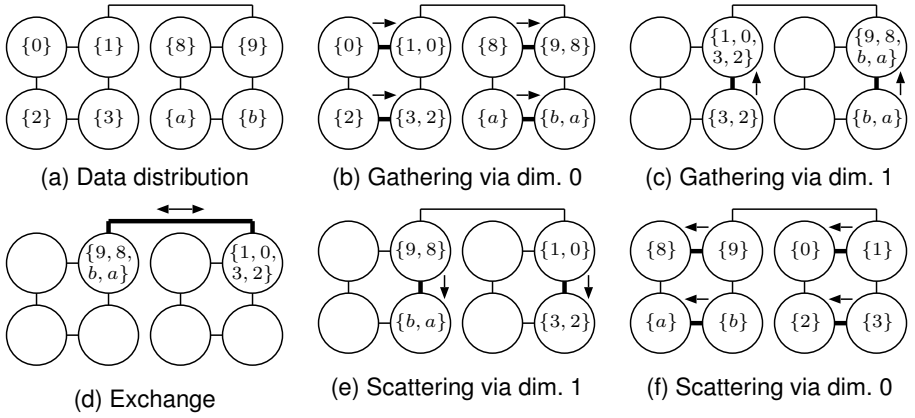

Fig. 9. Gather and scatter in dimension 3

In Figure 9(a), node i for $i = 0, 1, 2, 3, 8, 9, a, b$, contains its element $\{i\}$. In Figure 9(b), the arrowed lines denote the communication direction along with the dimension 0, that is, four nodes 0, 2, 8, and a send their elements to nodes 1, 3, 9, and b, respectively. Nodes 1, 3, 9, and b concatenate the received elements with their own elements. In Figure 9(c), the communications take place in dimension 1. Note that only two nodes 3 and b send their own elements gathered in the first step, and nodes 1 and 9 concatenate the received elements with their own elements. At this step, nodes 1 and 9 have gathered the entire elements within their 2-cube, respectively. Then, nodes 1 and 9 send the gathered elements to each other through the bridge as shown as in Figure 9(d). Figures 9(e) and 9(f) show the steps of the scatter operation which is the reverse of the gather operation. As the final result, every node contains the element of its partner in addition to its original element. There are $2k + 1$ communication steps in total: Each of the gather and scatter operations takes k steps and the exchange through the bridge requires one step.

We formally give the gather and scatter algorithm in Figure 10. In an MC(k, m), node $u =$ *my_id* sends the element *number_in* to node $u^{(j)}$ and receives the element *number_out* from node $u^{(j)}$ where the addresses of nodes u and $u^{(j)}$ differ in the j bit position and $0 \leq j \leq 2^k m + k - 1$.

The exchange of the elements is done with the gather and scatter operations through the bridge. In Figure 10, *my_class* $=$ *my_id* AND $(2^k - 1)$ is the class ID of node u; *bridge_class* $= (j - k)$ MOD 2^k is the class ID of two nodes which are connected with the bridge; and *class_diff* $=$ *my_class* XOR *bridge_class* is the dimension difference between the classes of *my_class* and *bridge_class* that is used to control the sending and receiving operations. The algorithm is composed of three parts: 1) gathering elements to a node of *bridge_class*; 2) exchanging elements through the bridge; and 3) scattering elements to nodes in k-cube. The gather operation collects elements from all the nodes

```
Gather_scatter (number_in, my_id, k, j, number_out)
begin
  my_class ← my_id AND 2^k − 1;
  bridge_class ← (j − k) MOD 2^k;
  class_diff ← my_class XOR bridge_class;
  /* 1. Gathering elements to a node of bridge_class */
  R ← {number_in};
  mask ← 0;
  for i ← 0 to k − 1 do
    /* Select nodes whose lower i bits are 0 */
    if (class_diff AND mask = 0)
      if (class_diff AND 2^i ≠ 0)
        msg_destination ← my_id XOR 2^i;
        send R to msg_destination;
      else
        msg_source ← my_id XOR 2^i;
        receive S from msg_source;
        R ← R ∪ S;
    mask ← mask XOR 2^i;                          /* Set bit i of mask to 1 */
  /* 2. Exchanging elements through the bridge */
  if (class_diff = 0)                             /* my_class = bridge_class */
    partner ← my_id XOR 2^j;
    send R to partner;
    receive S from partner;
  /* 3. Scattering elements to nodes in k-cube */
  mask ← 2^k − 1;
  for i ← k − 1 downto 0 do
    mask ← mask XOR 2^i;                          /* Set bit i of mask to 0 */
    if (class_diff AND mask = 0)
      if (class_diff AND 2^i = 0)
        msg_destination ← my_id XOR 2^i;
        send second half of S to msg_destination;
        S ← first half of S;
      else
        msg_source ← my_id XOR 2^i;
        receive S from msg_source;
  number_out ← S;
end
```

Fig. 10. Gather and scatter on MC(***k,m***)

in the k-cube to a node of *bridge_class*. This is done by a **for** loop with $i = 0, 1, \ldots, k-1$. Because the communication patterns of the nodes in the k-cube are not same, we must control the sending and receiving operations of the nodes based on *class_diff*, i and *mask*. The *mask* makes the number of the nodes that participate sending and receiving operation to be half for the next loop iteration. R is the collected element set to which the received element S is concatenated.

After finishing the gather operation, R contains all the elements of the nodes in the k-cube and is sent to another node through the bridge. The scatter operation distributes each element of the received element set S to all the nodes in the k-cube. It is the reverse of the gather operation. As the output of the algorithm, each node u gets an element *number_out* of the node $u^{(j)}$. The sorting algorithm on metacubes is given in Figure 11. When the dimension is less than k, we perform **send** and **receive** directly between the

two nodes of a node pair; otherwise, we perform **Gather_scatter** operations to exchange the elements.

```
Bitonic_sort_metacube (my_id, my_number, k, m, result)
begin
  n ← 2^k m + k;
  result ← my_number;
  for i ← 0 to n − 1 do
    for j ← i downto 0 do
      if (j < k)                                   /* direct communication in k-cube */
        partner ← my_id XOR 2^j;
        send result to partner;
        receive number from partner;
      else                                         /* gather and scatter via k-cube */
        Gather_scatter (result, my_id, k, j, number);
      if (my_id AND 2^(i+1) ≠ my_id AND 2^j)                                  /* max */
        if (number > result)
          result ← number;
      else                                                                    /* min */
        if (number < result)
          result ← number;
end
```

Fig. 11. Bitonic sorting algorithm on MC(k, m)

Theorem 1. In the bidirectional channel and 1-port communication model, bitonic sorting in an MC(k, m) with $N = 2^{2^k m+k}$ nodes can be done in $(2^k m + k)^2$ computation steps and $(2^k m(2k + 1) + k)^2$ communication steps, respectively.

Proof: We first show that the correctness of the algorithm Bitonic_sort_metacube. We define a 2^k-to-1 mapping from the set of vertices of MC(k, m) onto the set of vertices of $2^k m$-cube as follows: $f : b_{n-1} \dots b_1 b_0 \rightarrow b_{n-1} \dots b_k$, where $n = 2^k m + k$. From the new presentation of metacube, the bridges of MC(k, m) are mapped onto the edges of $2^k m$-cube. That is, MC(k, m) is mapped onto a $2^k m$-cube. After mapping, the operations in step 1 and step 3 in algorithm Gather_scatter become local memory accesses. It is easy to see that the algorithm Bitonic_sort_metacube emulates the algorithm Bitonic_sort_hypercube in the mapped $2^k m$-cube with each node holding 2^k numbers. Therefore, from the algorithm Gather_scatter which gathers 2^k elements in a k-cube and scatters them to another k-cube via a bridge (an edge in the mapped $2^k m$-cube), we conclude that the algorithm emulates the algorithm Bitonic_sort_hypercube correctly.

Next, we assume that the edges in MC(k, m) are bidirectional and each node in MC(k, m) can receive a message, concatenate it with local data and send the concatenated message to its neighbor in one time unit. In the proposed algorithm, at each iteration, all pairs of nodes in MC(k, m), $(u, u^{(j)})$ for a specific dimension j, should perform the compare-and-exchange operation. This is done by performing Gather_scatter for every pair of k-cubes that requires a total of $2k + 1$ communication steps for a sin-

gle compare-and-exchange operation. Therefore, the communication time $T_{comm}(n)$ and computation time T_{comp} of the proposed algorithm are $(2^k m(2k+1)+k)^2$ and $(2^k m + k)^2$, respectively. $\diamondsuit$

5 Concluding Remarks

In this paper, we showed an efficient sorting algorithm on MC(k, m) that uses a new presentation of the metacube. Based on the new presentation, the hypercube algorithms can be emulated effectively. The overhead for the emulation is mainly due to communicate between the two matched k-cubes through a bridge. Since k is small ($k \leq 3$ for any possible practical parallel computers), the overhead for the communication is relatively small. The future work includes the following: (1) Generalize the proposed algorithm for sorting input sequences of any size on metacube and perform simulations and empirical analysis for the proposed algorithm. (2) Investigate and develop more application algorithms on metacube based on the new presentation.

References

1. K. E. Batcher. Sorting networks and their applications. In *Proceedings of AFIPS Spring Joint Computer Conference*, pages 307–314, Apr. 1968.
2. Shih-Yan Chen and Shin-Shin Kao. The edge-pancyclicity of dual-cube extensive networks. In *Proceedings of the 2nd WSEAS International Conference on Computer Engineering and Applications*, pages 233–236, Acapulco, Mexico, January 2008.
3. J. P. Hayes and T. N. Mudge. Hypercube supercomputers. *Proc. IEEE*, 17(12):1829–1841, Dec. 1989.
4. Z. Jiang and J. Wu. Fault-tolerant routing in dual-cube networks. In *Proc. of the 7th Joint Conference on Information Sciences*, Sept. 389-392.
5. V. Kumar, A. Grama, A. Gupta, and G. Karypis. *Introduction to parallel computing: design and analysis of algorithms*. Benjamin/Cummings Press, 1994.
6. Chia-Jui Laia and Chang-Hsiung Tsai. On embedding cycles into faulty dual-cubes. *Information Processing Letters*, 109(2):147–150, December 2008.
7. F. T. Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays, Trees and Hypercubes*. Morgan Kaufmann Pub, 1992.
8. 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.
9. Y. Li, S. Peng, and W. Chu. Metacube – a new interconnection network for large scale parallel systems. In *Australian Computer Science Communications*, volume 24, pages 29–36, 2002.
10. Behrooz Parhami. *Introduction to parallel processing, algorithm and architecture*. Plenum Press, 1999.
11. M. Venkata Rao and N. Chalamaiah. Routing and broadcasting algorithms for a metacube interconnection topology for large scale parallel systems. In *Proceedings of Asia Pacific Confeence on Parallel and Distributed Computing Technologies*, pages 1037–1049, December 2004.
12. SGI. *Origin2000 Rackmount Owner's Guide, 007-3456-003*. http://techpubs.sgi.com/, 1997.
13. L. W. Tucker and G. G. Robertson. Architecture and applications of the connection machine. *IEEE Computer*, 21:26–38, August 1988.
14. C. Wu and J. Wu. On self-similarity and hamiltonicity of dual-cubes. In *Proc. of Workshop on Massively Parallel Processing (in conjunction with IPDPS)*, April 2003.