Prefix Computation and Sorting in Dual-Cube

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

In this paper, we describe two algorithmic techniques for the design of efficient algorithms in dual-cube. The first uses cluster structure of dual-cube, and the second uses recursive structure of the dual-cube. We propose efficient algorithms for parallel prefix computation and sorting in dual-cube based on the two techniques, respectively. For a dual-cube D_n with 2^{2n-1} nodes and n links per node, the communication and computation times of the algorithm for parallel prefix computation are at most $2n+1$ and $4n-2$, respectively; and those of the algorithm for sorting are at most $6n^2$ and $2n^2$, respectively.

1. Introduction

The hypercube is one of the most versatile and efficient networks for parallel computation [2, 4, 5]. 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. Among them are the butterfly, shuffle-exchange graph, de Bruijn graph, Benes network, and cube-connected-cycles (CCC) [5, 9]. However, these networks have their own drawbacks that prevent them to be used as a practical architecture for very large parallel computers.

The dual-cube [6, 7] was proposed to overcome the problem mentioned above that keeps most of the interesting properties of the hypercube architecture. Dual-cube can be viewed as an improvement over CCC networks. Dual-cube holds more hypercube-like properties than others, such as recursive construction, binary presentation with two nodes connected by an edge only if they differ in exactly one bit, etc. The communications in dual-cube are very efficient, almost as efficient as in hypercube. The diameter of dual-cube is that of hypercube of the same size plus one. However, the number of edges per node in dual-cube is about half of that in the hypercube of the same size. Therefore, parallel computers with tens of thousands of processors can be constructed by dual-cube practically with up to eight connections each processor.

A dual-cube uses binary hypercubes as basic components. Each such hypercube component is referred to as a *cluster*. In an n-dual-cube D_n, the number of nodes in a cluster is 2^{n-1}. In D_n, there are two *classes* with each class consisting of 2^{n-1} clusters. The total number of nodes is 2^{2n-1}. Therefore, the node address has $2n-1$ bits. The leftmost bit is used to indicate the type of the class (class 0 and class 1). For the class 0, the rightmost $n-1$ bits are used as the node ID within the cluster and the middle $n-1$ bits are used as the cluster ID. For the class 1, the rightmost m bits are used as the cluster ID and the middle m bits are used as the node ID within the cluster. Each node in a cluster of class 0 has one and only one extra connection to a node in a cluster of class 1. These two node addresses differ only in the leftmost bit position.

In an n-connected dual-cube, $n-1$ edges are used within cluster to construct an $(n-1)$-cube and a single edge is used to connect a node in a cluster of another class. There is no edge between the clusters of the same class. If two nodes are in one cluster, or in two clusters of distinct classes, the distance between the two nodes is equal to its Hamming distance, the number of bits where the two nodes have distinct values. Otherwise, it is equal to the Hamming distance plus two: one for entering a cluster of another class and one for leaving.

In this paper, we propose two algorithmic techniques for the design of efficient algorithms in dual-cube. The

first uses the cluster structure and the second uses recursive structure of the dual-cube. We develop algorithms for parallel prefix computation and sorting in dual-cube using the first and the second techniques, respectively. For a dual-cube D_n with n links per node, the algorithm for parallel prefix computation runs in $2n + 1$ time, and the algorithm for sorting runs in $3n^2$ time.

The rest of this paper is organized as follows. Section 2 describes the dual-cube structure in details. Section 3 shows an optimal algorithm for parallel prefix computation in dual-cube. Section 4 describes a recursive presentation of dual-cube. Section 5 reviews sorting algorithms in hypercube. Section 6 gives an efficient sorting algorithm in dual-cube based on bitonic sort. Section 7 concludes the paper and presents some future research directions.

2. Dual-Cube Networks

An n-connected dual-cube D_n is a undirected graph on the node set $\{0,1\}^{2n-1}$ such that there is an edge between two nodes $u = (u_{2n-1} \dots u_1)$ and $v = (v_{2n-1} \dots v_1)$ in D_n if and only if the following conditions are satisfied:

(1) u and v differ exactly in one bit position i.

(2) if $1 \le i \le n-1$ then $u_{2n-1} = v_{2n-1} = 0$.

(3) if $n \le i \le 2n-2$ then $u_{2n-1} = v_{2n-1} = 1$.

Intuitively, the set of the nodes u of form $(0u_{2n-2} \dots u_n * \dots *)$, where $*$ means "don't care", constitutes a $(n-1)$-dimensional hypercube. We call these hypercubes *clusters* of class 0. Similarly, the set of the nodes u of form $(1 * \dots * u_{n-1} \dots u_1)$ constitutes a $(n-1)$-dimensional hypercube, and we call them clusters of class 1. The edge connects two nodes in two clusters of distinct class is called *cross-edge*. In other word, $\langle u, v \rangle$ is a cross-edge if and only if u and v differ at the leftmost bit position only.

We divide the binary representation of a node into three parts: Part I is the rightmost $r-1$ bits, part II is the next $r-1$ bits, and part III is the leftmost bit. For the nodes in a cluster of class 0 (class 1), part I (part II) is called *node ID* and part II (part I) is called *cluster ID*. Part III is a *class indicator*. The cluster contains node u is denoted as C_u. For any two nodes u and v in D_n, $C_u = C_v$ if and only if u and v are in the same cluster.

Figure 1 depicts a D_3 network. The class indicator is shown at the top position in the node address. For the nodes of class 0 (class 1), the node ID (cluster ID) is shown at the bottom, and the cluster ID (node ID) is shown at the middle. Figure 2 depicts a D_4 network. Notice that only those edges connecting to cluster 0 of class 1 are shown in the figure. Dual-cube has very nice topological properties, similar to that of hypercube such that node and edge symmetricity, and recursive construction. The routing algorithm in dual-cube is also very simple and the diameter of a D_n (of size 2^{2n-1}) is only $2n$.

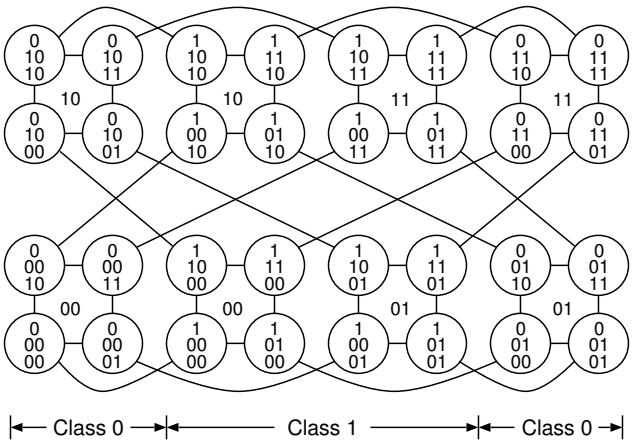


Figure 1. The dual-cube D_3

3. Parallel Prefix Computation in Dual-Cube

Let $\oplus$ be an associative binary operation. Given n numbers $c_0, c_1, \dots, c_{n-1}$, parallel prefix computation [2, 3] is defined as simultaneously evaluating all of the prefixes of the expression $c_0 \oplus c_1 \dots \oplus c_{n-1}$. The ith prefix is $s_i = c_0 \oplus c_1 \dots \oplus c_{i-1}$. The parallel prefix computation can be done optimally in hypercube: only involve n communication steps for computing prefixes in n-cube. Assume that each node u, $0 \le u \le 2^n - 1$, in an n-cube holds a number $c[u]$. The algorithm for parallel prefix (or diminished prefix which excludes $c[u]$ in $s[u]$) computation in n-cube is an ascend algorithm in which each node successively communicates with its neighbors in dimension order from 0 to $q-1$. Each node deals with two variables: a subcube "total" t in the current subcube and a subcube parallel prefix result s that gives the result of parallel prefix computation in the same subcube. Eventually, s becomes the required prefix (or diminished prefix) within the entire n-cube. The algorithm is shown in Algorithm 1.

For parallel prefix computation in D_n, we first assume that each node in D_n hold a key value in $c[0..2^{2n-1}-1]$. However, different with that in hypercube, we arrange the input data in a way such that the indices of c kept in the nodes of a cluster of class 1 will be in consecutive order. Notice that the IDs of nodes in a cluster of class 1 is not consecutive. That is, node u holds value $c[u]$ if $ID_s(u) = 0$; otherwise, value $c[u']$, where u' obtained from u by swap$[(u_{2n-2} \dots u_n), (u_{n-1} \dots u_1)]$.

In order to get the values of the total in other clusters of the same class for prefix computation, we need additional

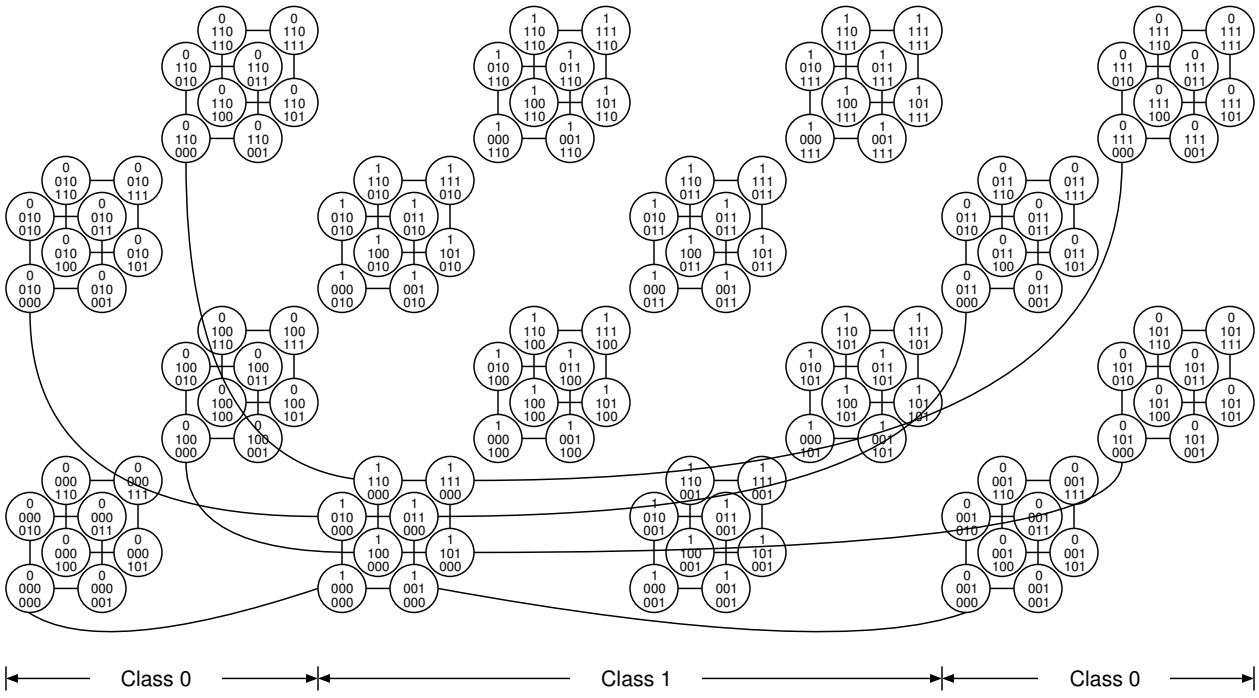


Figure 2. The dual-cube D_4

Algorithm 1: Cube_prefix(Q_n, c, tag)
Input: n-cube Q_n with each node u holds $c[u]$ and a boolean tag
Output: (t, s), sum $t[u]$ and prefix $s[u] = c[0] \oplus c[1] \ldots \oplus c[u]$ if $tag = 1$, diminished prefix $s[u] = c[0] \oplus c[1] \ldots \oplus c[u-1]$ otherwise, at node u
begin
  for node u, $0 \le u \le 2^n - 1$ **parallel do**
    $t[u] \leftarrow c[u]$;
    if $(tag = 1)$ $s[u] \leftarrow c[u]$
      else $s[u] \leftarrow 0$; /* Initialize t and s */
    for $i \leftarrow 0$ to $n-1$ **do**
      /* u and $\bar{u}^i$ differ at the ith bit only. */
      send $t[u]$ to $\bar{u}^i$;
      get $temp[u] \leftarrow t[\bar{u}^i]$;
      $t[u] \leftarrow t[u] \oplus temp[u]$;
      if $u > \bar{u}^i$ $s[u] \leftarrow s[u] \oplus temp[u]$;
end

---

two communication steps via cross-edges, and one more parallel prefix computation in each cluster on the values got through the cross-edges. Each node has four variables: Variables t and s are the same as that in hypercube. Additional two variables t' and s' are for the calculation of the prefixes of the sub-totals in the clusters of the same class.

The details are specified in Algorithm 2. An example of prefix_sum on D_3 is shown in Figure 3. Figure 3(a) shows the input sequence c. Figure 3(b) to Figure 3(f) show the results of step 1 - 5 of the algorithm, respectively.

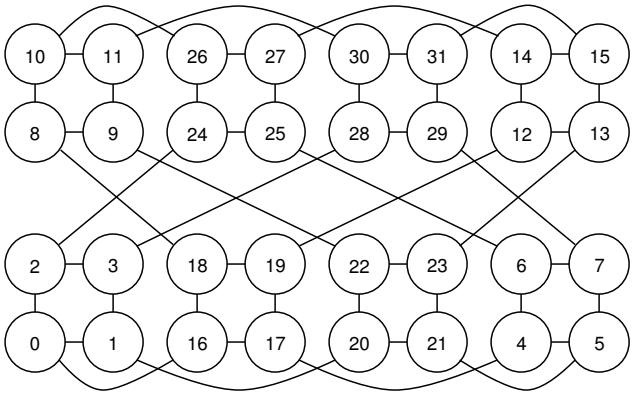
Theorem 1 Assume 1-port, bidirectional-channel communication model. Parallel prefix computation in n-dual-cube D_n with $N = 2^{2n-1}$ nodes can be done in $2n+1$ communication steps and $4n-1$ computation steps.

Proof: From the arrangement of the initial value that each node holds, the indices of $c[u']$ held in node u are consecutive inside every cluster. From the algorithms for parallel prefix computation in hypercube and dual-cube, it is easy to check that, after algorithm D_prefix(D_n) is done, the resulting values held in the nodes of D_n are prefixes of $c[0..2^{2n-1}-1]$. Next, we assume that the edges in D_n are bidirectional channels, and at each clock cycle, each node in D_n can send or get at most one message. In Algorithm 2, the first and the third steps that call Cube_prefix for $(n-1)$-cube require $n-1$ communication steps and at most $2(n-1)$ computation steps. Therefore, the communication and computation times of Algorithm 2 are $T_{comm}(n) = 2(n-1)+3 = 2n+1$ and $T_{comp}(n) = 4(n-1)+2 = 4n-2$, respectively. $\diamondsuit$

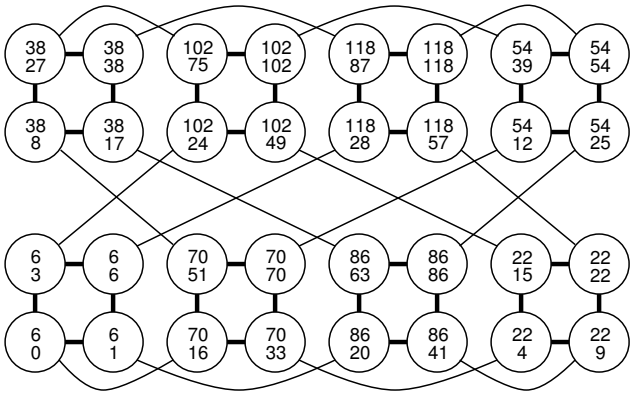
4. Recursive Presentation Of Dual-Cube

In this section, we introduce an alternative presentation of a dual-cube. This representation of a dual-cube

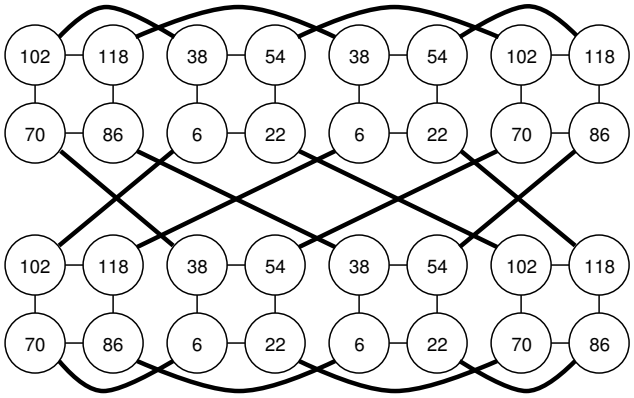
Prefix_sum([0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31]) =
[0,1,3,6,10,15,21,28,36,45,55,66,78,91,105,120,136,153,171,190,210,231,253,276,300,325,351,378,406,435,465,496]



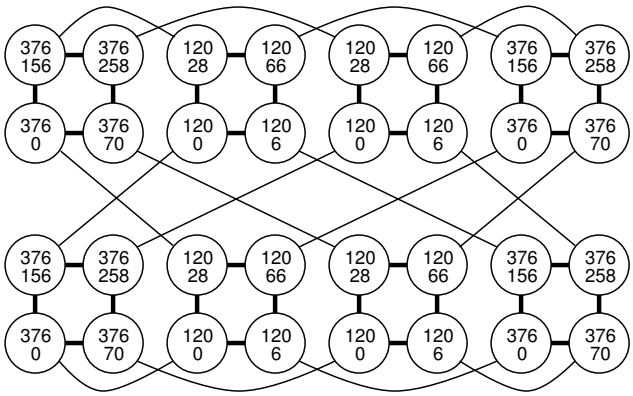
(a) Original data distribution



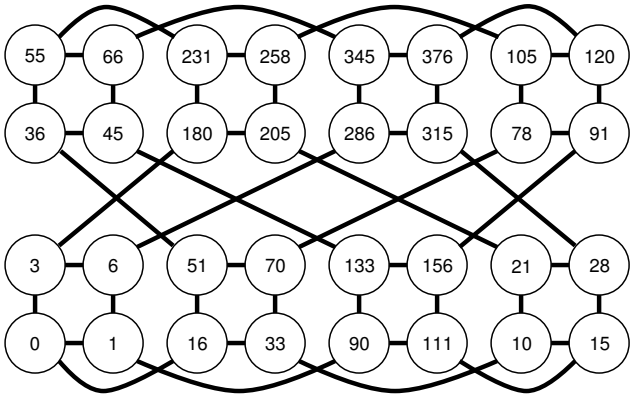
(b) Prefix inside cluster ($c0 = c$, generate t and s)



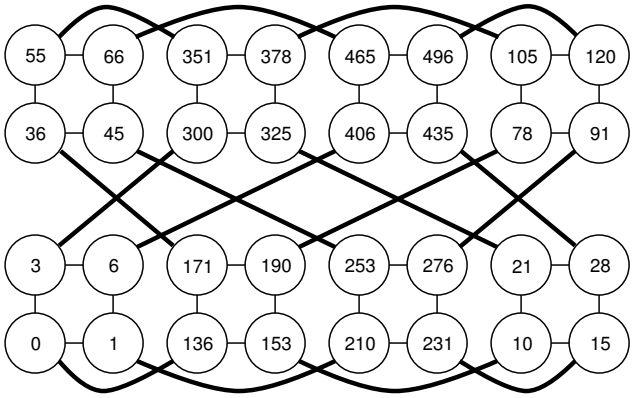
(c) Exchange t via cross-edge



(d) Prefix inside cluster ($c0 = 0$, generate t' and s')



(e) Get s' and prefix one time



(f) Final result (Class 1 + 120)

Figure 3. An example of prefix_sum on dual-cube

D_n is motivated by the recursive construction of D_n from D_{n-1}. Since the number of nodes in D_n is four times of the number of nodes in D_{n-1}, we need four D_{n-1} in order to construct a D_n. In this presentation, each

Algorithm 2: D_prefix(D_n)
Input: n-dual-cube D_n and an array of keys $c[0, 2^{2n-1}-1]$
Assume node u holds $c[u']$, where $u' = u$ if $ID_s(u) = 0$,
$= (1, cluster\_ID(u), node\_ID(u))$, otherwise.
Output: prefix $s[u'] = c[0] \oplus c[1] \ldots \oplus c[u']$ at node u
begin
1. **for** cluster $C_i,\ 0 \le i \le 2^n - 1$, **parallel do**
   $(t, s) \leftarrow$ Cube_prefix($C_i,\ c,\ 1$);
2. **for** node $u,\ 0 \le u \le 2^{2n-1} - 1$, **parallel do**
   send $t[u]$ to $\bar{u}^{2n-1}$;
   get $temp[u] \leftarrow t[\bar{u}^{2n-1}]$;
3. **for** cluster $C_i,\ 0 \le i \le 2^n - 1$, **parallel do**
   $(t', s') \leftarrow$ Cube_prefix($C_i,\ temp,\ 0$);
4. **for** node $u,\ 0 \le u \le 2^{2n-1} - 1$, **parallel do**
   send $s'[u]$ to $\bar{u}^{2n-1}$;
   get $temp[u] \leftarrow s'[\bar{u}^{2n-1}]$;
   $s[u] \leftarrow s[u] \oplus temp[u]$;
5. **for** node $u,\ 0 \le u \le 2^{2n-1} - 1$, **parallel do**
   if $ID_s(u) = 1$
   send $t'[u]$ to $\bar{u}^{2n-1}$;
   get $temp[u] \leftarrow t'[\bar{u}^{2n-1}]$;
   $s[u] \leftarrow s[u] \oplus temp[u]$;
end

node in D_n is assigned an ID, $(a_{2n-1} \ldots a_1)$, where a_1 is class ID, $(a_{2n-1}, a_{2n-3} \ldots a_3$ is a cluster (node) ID and $(a_{2n-2} \ldots a_2)$ is a node (cluster) ID for $a_1 = 0$ ($a_1 = 1$). From this new presentation, a D_n can be viewed as a graph including four copies of D_{n-1} (the leftmost two bits serve as ID of each copy of D_{n-1}).

The D_n can be constructed recursively as follows:

- Base: $D_1 = Q_1$;
- Recursive step: $n > 1$

  1. The four subsets $\{(00u) |\ u \in \{0,1\}^{2n-3}\}$, $\{(00u) |\ u \in \{0,1\}^{2n-3}\}$, $\{(00u) |\ u \in \{0,1\}^{2n-3}\}$, and $\{(00u) |\ u \in \{0,1\}^{2n-3}\}$, form four D_{n-1};
  2. Create a link for each pair of nodes $(x0u0)$ and $(x1u0)$, and each pair of nodes $(0xu1)$ and $(1xu1)$, where $(u1) \in Dn-1$, where $x \in \{0,1\}$ and $u \in \{0,1\}^{2n-4}$.

Figure 4 shows the recursive construction of D_2 and D_3 from D_1 and D_2, respectively. The recursive connections are marked with bold lines and curves. F_1 in Figure 4(a) is a K_2 that has only two nodes, one for each class. Figure 4(c) is the same as Figure 4(b), and Figure 4(d) is the same as Figure 1. The b_2b_1 is written in the left side in Figure 4(b) and Figure 4(d).

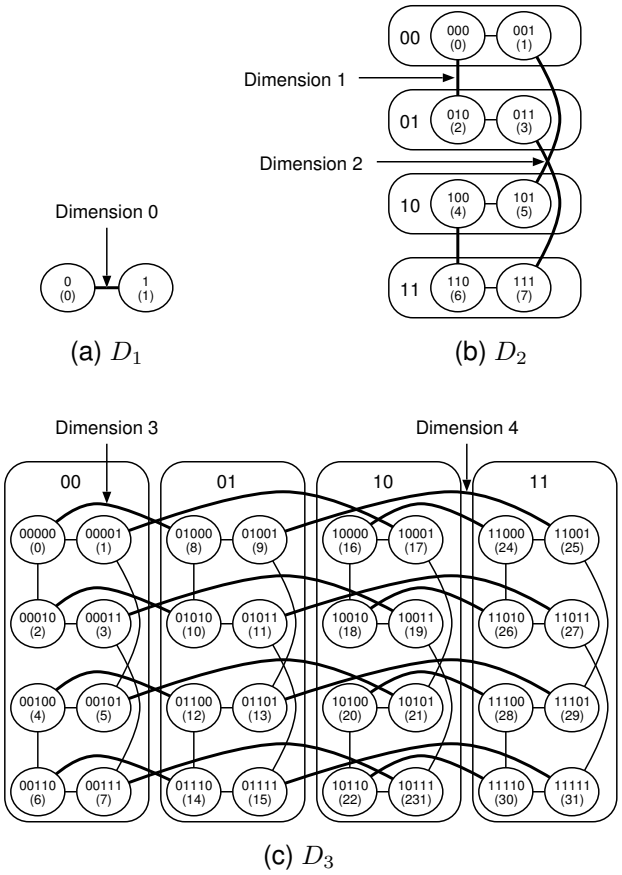


Figure 4. The construction of D_2 and D_3 from D_1

5. Sorting in Hypercube

The problem of sorting on hypercube networks has been intensively studied. Batcher's $O(n^2)$-time bitonic and odd-even merge sorting algorithms [1] for sorting $N = 2^n$ numbers on an n-cube are presently the fastest practical deterministic sorting algorithms available. The more complicated $O(n \log n)$-time algorithms are not competitive for $n < 20$. Randomized algorithms can sort in $O(n)$ time. However, they do not provide guaranteed speedup [5, 8].

Since the proposed sorting algorithm in dual-cube is based on bitonic sorting, we will show below the bitonic sorting in hypercube. A sequence $\{a_1, \ldots, a_n\}$ is a bitonic sequence if it "rises then falls" ($a_1 \le a_2 \le \ldots \le a_i \ge a_{i+1} \ge \ldots \ge a_n\}$), "falls then rises" ($a_1 \ge a_2 \ge \ldots \ge a_i \le a_{i+1} \le \ldots \le a_n\}$), or is obtained from the above two types of sequences through cyclic shifts or rotations. When input sequence is a bitonic sequence, sorting an n-cube can be done by a descend algorithm: Each node successively communicates with its neighbors in dimension order from n to 1 (left-to-right). In a single communication step, the

Algorithm 3: D_sort($D_n,\ tag$)
Input: Dual-cube D_n with recursive presentation and a boolean variable tag /* each node $u \in D_n$ holds a key $c[u]$. */
Output: Sorted sequence of keys in ascending/descending order when the value of tag is 0/1
begin
/* Let four disjoint D_{n-1} in D_n be $D^i_{n-1},\ 0 \le i \le 3$, where the leftmost two bits of node $u \in D^i_{n-1}$ equal i. */
if $n = 1$ **then**
if $tag = 0$ **then**
if $c[1] < c[0]$ **then** swap($c[0], c[1]$)
else
if $c[1] > c[0]$ **then** swap($c[0], c[1]$)
else /* $n > 1$:/
D_sort($D^0_{n-1},\ 0$);
D_sort($D^1_{n-1},\ 1$);
D_sort($D^2_{n-1},\ 0$);
D_sort($D^3_{n-1},\ 1$);
/* Sort keys in D^i_{n-1} in ascending/descending order for i is even/odd. */
for $j \leftarrow 2n-2$ **downto** 1 **do**
/* For node u, let $\bar{u}^j$ be the node differs with u at the jth bit only. */
for each node u **parallel do**
if $(j = 1)$ or ($u_1 = 0$ and $j > 1$ is even) or ($u_1 = 1$ and $j > 1$ is odd)
then send $c[u]$ to and receive $d[u] = c[\bar{u}^j]$ from node $\bar{u}^j$ via $(u, \bar{u}^j) \in E(D_n)$
else send $c[u]$ to and receive $d[u] = c[\bar{u}^j]$ from node $\bar{u}^j$ via $(u, v = \bar{u}^1)$, $(v, w = \bar{v}^j)$, and $(w, \bar{w}^1 = \bar{u}^j)$;
if ($u_{2n-1} = 0$ and $(u_j = 0)$ or ($u_{2n-1} = 1$ and $(u_j = 1)$
then $c[u] \leftarrow \min\{c[u], d[u]\}$
else $c_u \leftarrow \max\{c[u], d[u]\}$;
for $j \leftarrow 2n-1$ **downto** 1 **do**
for each node u **parallel do**
if $(j = 1)$ or ($u_1 = 0$ and $j > 1$ is even) or ($u_1 = 1$ and $j > 1$ is odd)
then send $c[u]$ to and receive $d[u] = c[\bar{u}^j]$ from node $\bar{u}^j$ via $(u, \bar{u}^j) \in E(D_n)$
else send $c[u]$ to and receive $d[u] = c[\bar{u}^j]$ from node $\bar{u}^j$ via $(u, v = \bar{u}^1)$, $(v, w = \bar{v}^j)$, and $(w, \bar{w}^1 = \bar{u}^j)$;
if ($tag = 0$ and $(u_j = 0)$ or ($tag = 1$ and $(u_j = 1)$
then $c[u] \leftarrow \min\{c[u], d[u]\}$
else $c_u \leftarrow \max\{c[u], d[u]\}$;
end

pair of nodes compare and then exchange the two numbers if necessary. For sorting in increasing (decreasing) order, the smaller one stays at the lower-labeled (higher-labeled) node. The bitonic sorting on an n-cube is a recursive one: First, divide an n-cube Q_n into two $(n-1)$-subcubes, Q^0_{n-1} and Q^1_{n-1}, where the leftmost bit of a node in Q^0_{n-1} (Q^1_{n-1}) is 0 (1). Second, sort the numbers in Q^0_{n-1} and Q^1_{n-1} recursively in opposite direction such that the resulting sequence in Q_n is bitonic. Third, call the descend algorithm described above to perform parallel compare-and-exchange operations n times. It is easy to see that the running time of the recursive algorithm is $O(n^2)$.

6. Sorting in Dual-Cube

In this section, D_n will be presented by the format described in Section 4. The proposed sorting algorithm in D_n is similar to the bitonic sorting in hypercube. We first explain the compare-and-exchange operation in D_n. Suppose that the compare-and-exchange operation should be performed by the pair of nodes, u and v in D_n that differ only at the ith bit for $i > 1$. Let $u = (u_{2n-1} \dots u_i \dots u_1)$ and $v = (v_{2n-1} \dots v_i \dots v_1)$, where $u_j = v_j$ for $j \neq i$, $u_i = 0$, $v_i = 1$, and $u_1 = v_1 = 0$. Since $u_1 = v_1 = 0$, there is a link between u and v if and only if i is an even number. For an even i, the compare-and-exchange operation can be done by a single data transfer plus a comparison. However, for an odd i, the data transfer requires three hops ($u \rightarrow (u_{2n-1} \dots u_i \dots 1 \rightarrow (u_{2n-1} \dots v_i \dots 1 \rightarrow (v_{2n-1} \dots v_i \dots 0)$. Therefore, in D_n, a parallel compare-and-exchange operation for all pairs of nodes at the ith dimension takes three time-units.

The algorithm can be described as follows. First, we sort

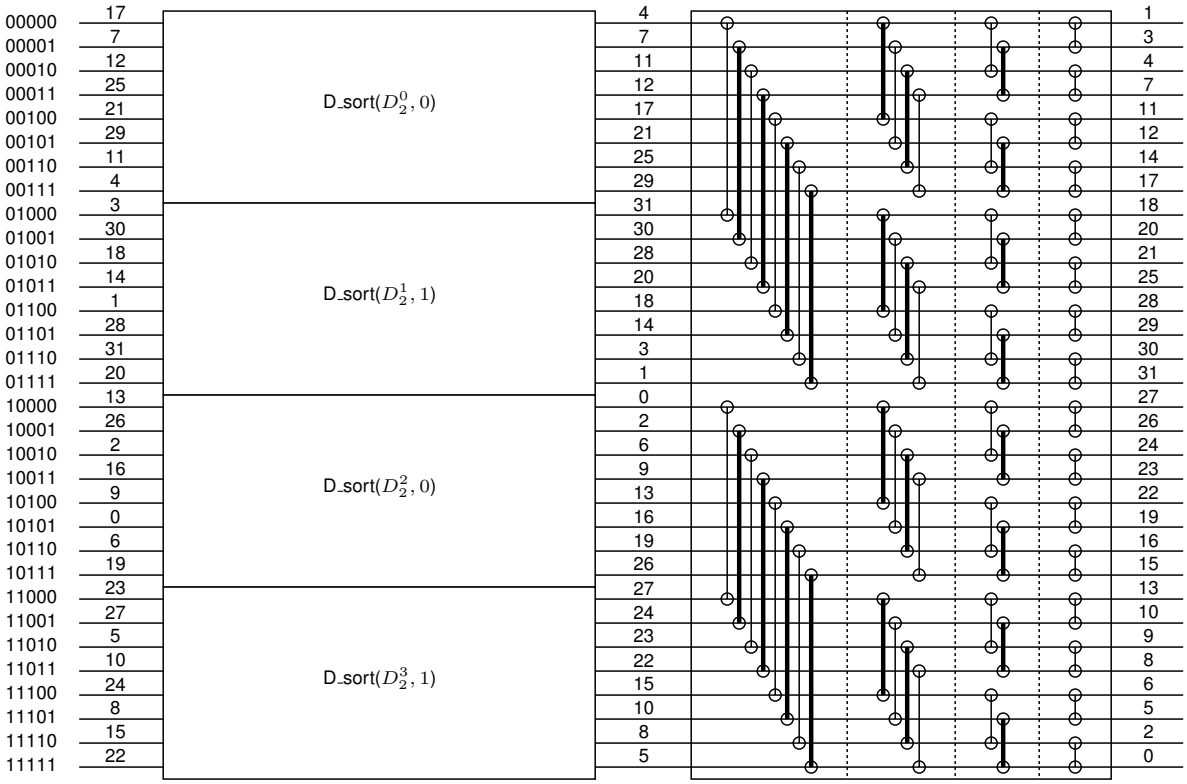


Figure 5. Generate bitonic sequence in D_3

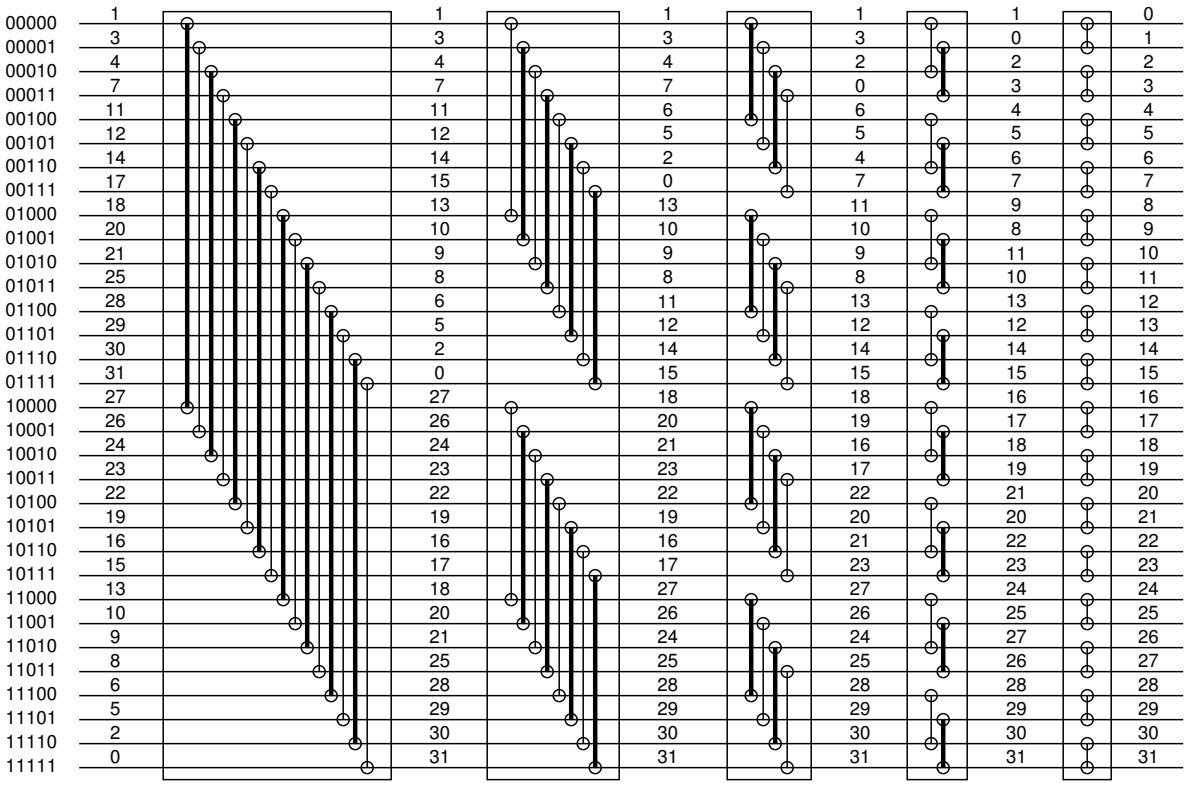


Figure 6. Sort bitonic sequence in D_3

four D_{n-1} sub-dual-cubes (D_{n-1}^{xy}, where $x,\ y$ are 0 or 1) recursively such that the sequences in $D_{n-1}^{00} \cup D_{n-1}^{01}$ and $D_{n-1}^{10} \cup D_{n-1}^{11}$ form bitonic sequences. Second, we perform $2n-2$ steps of dimensional-order (left-to-right) compare-and-exchange operations for the right $2n-2$ dimensions in D_n to form a bitonic sequence in D_n. Notice that the output sequences in $D_{n-1}^{00} \cup D_{n-1}^{01}$ and $D_{n-1}^{10} \cup D_{n-1}^{11}$ are arranged in ascendant and descendant orders, respectively. Third, we perform $2n-1$ steps of dimensional-order (left-to-right) compare-and-exchange operations for the $2n-1$ dimensions in D_n to get the output sequence in sorted order. The algorithm is formally presented in Algorithm 3. An example that show how the algorithm 3 works for D_sort($D_3,\ 0$) is showed in Figures 5 and 6. The thin lines between u and $\bar{u}^j$ are edges in D_3, while the thick lines between u and $\bar{u}^j$ are paths of length 3 in D_3. In Figure 5, the input sequence in D_3 is transformed into a bitonic sequence. In Figure 6, the bitonic sequence is sorted in ascending order.

Theorem 2 Assume bidirectional channel, 1-port communication model. Sorting in dual-cube D_n with $N = 2^{2n-1}$ nodes can be done in at most $6n^2$ communication steps and at most $2n^2$ comparison steps.

Proof: The Algorithm 3 emulates the bitonic sorting in dual-cube. Therefore, the correctness of Algorithm 3 can be verified easily. Next, we assume that the edges in D_n are bidirectional and each node in D_n can send and receive at most one message in one clock cycle. In Algorithm 3, at each step, all pairs of nodes in D_n, $(u, \bar{u}^j)$ for a specific dimension j, perform compare-and-exchange operation. However, only half of the pairs have direct links for $j > 1$. The pairs without a direct link need three hops (use cross-edges twice) for a single compare-and-exchange operation. Therefore, the communication time $T_{comm}(n)$ and computation time T_{comp} of Algorithm 3 can be presented by the recurrences $T_{comm}(n) = T_{comm}(n-1) + 3((2n-2)+(2n-1)) < T(n-1)+12n$ and $T_{comp}(n) = T_{comp}(n-1) + ((2n-2)+(2n-1)) < T(n-1)+4n$, respectively. By solving the recurrences, we conclude that the communication and computation times for sorting in D_n are at most $6n^2$ and $2n^2$, respectively. $\diamondsuit$

7. Concluding Remarks

In this paper, we showed two algorithmic techniques for the design of efficient algorithms in dual-cube. The first one using cluster structure and then takes care the inter-cluster communication via cross-edge. If inter-cluster communication can be designed to satisfy the requirement of the algorithm then it is very likely to find an optimal or near optimal algorithm in dual-cube using this technique, such as the one for parallel prefix computation. The second one uses recursive structure. Since most of the algorithms in hypercube are recursive, the algorithms that emulate these hypercube algorithms can be developed using the second technique. However, the overhead for the emulation will be 3 times of the corresponding hypercube algorithm in the worst-case due to the lack of edges.

The future work include the following:

1. Generalize the proposed algorithms to include the cases that input sequences are larger than the size of the dual-cube.
2. Do some simulations and empirical analysis for the proposed algorithms.
3. Investigate and develop more application algorithms in dual-cube using the proposed techniques.

References

[1] K. E. Batcher. Sorting networks and their applications. In *Proceedings of AFIPS Spring Joint Computer Conference*, pages 307–314, Apr. 1968.

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

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

[4] 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.

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

[6] 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.

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

[8] Behrooz Parhami. *Introduction to parallel processing: algorithm and architecture*. Kluwer Academic Publishers, Norwell, MA, USA, 1999.

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