Parallel Prefix Computation and Sorting on a Recursive Dual-Net

Yamin Li*, Shietung Peng* and Wanming Chu**

Abstract—In this paper, we propose efficient algorithms for parallel prefix computation and sorting on a recursive dual-net. The recursive dual-net $RDN^k(B)$ for $k > 0$ has $(2n_0)^{2K}/2$ nodes and $d_0 + k$ links per node, where n_0 and d_0 are the number of nod es and the node-degree of the base-network B, respectively. Assume that each node holds one data item, the communication and computation time complexities of the algorithm for parallel prefix computation on $RDN^k(B)$, $k > 0$, are 2^{k+1}-2+2$^k T_{comm}(0)$ and 2 $^{k+1}$-2+2$^k T_{comp}(0)$, respectively, where $T_{comm}(0)$ and $T_{comp}(0)$ are the communication and computation time complexities of the algorithm for parallel prefix computation on the base-network B, respectively. The algorithm for parallel sorting on $RDN^k(B)$ is restricted on $B = Q_m$ where Q_m is an m-cube. Assume that each node holds a single data item, the sorting algorithm runs in $O((m2^k)^2)$ computation steps and $O((km2^k)^2)$ communication steps, respectively.

Keywords— Interconnection Networks, Algorithm, Parallel Prefix Computation, Sorting

1. INTRODUCTION

The purpose of the interconnection networks (INs) is to connect processor/memory boards together to form a parallel or distributed system. In massively parallel computer systems, the interconnection networks play a crucial role in issues such as communication performance, hardware cost, computational complexity, and fault-tolerance. Much research has been reported in the literature on interconnection networks, which can be used to construct parallel computers of large scale [1,2,3].

The following two categories have attracted great research attention. One is the hypercube-like family that has the advantage of short diameters for high-performance computing and efficient communication [4,5,6,7,8]. The other is the family of 2D/3D meshes or tori that has the advantage of small and fixed node-degrees and easy implementation. Traditionally, most parallel systems including those built by CRAY, IBM, SGI, and Intel use 3D tori or hypercubes.

Recursive networks have also been proposed as effective interconnection networks for large-scale parallel computers. For example, the WK-recursive network [9,10] is a class of recursive scalable networks. It offers a high-degree of regularity, scalability, and symmetry and has a compact VLSI implementation.

Recently, because of the advance in computer technology and competition among computer makers, supercomputers containing hundreds of thousands of nodes have been constructed [11]. It was

Manuscript received September 27, 2010; accepted February 13, 2011.
Corresponding Author: Yamin Li
* Dept. of Computer Science, Hosei University, Tokyo 184-8584 Japan ({yamin, speng}@hosei.ac.jp)
** Dept. of Computer Hardware, University of Aizu, Aizu-Wakamatsu 965-8580 Japan (w-chu@u-aizu.ac.jp)

predicted that the parallel systems of the next decade will contain 10 to 100 millions of nodes [12].

An interconnection network consists of switches with multiple communication ports and cables that connect the ports by following some topologies. For a parallel computer of a very-large scale, the traditional interconnection networks may no longer satisfy the requirements for high-performance computing or efficient communication. For future generations of supercomputers with millions of nodes, the node-degree and the diameter will be the critical measures for the effectiveness of the interconnection networks. The node-degree is limited by the hardware technologies and the diameter affects all kinds of communication schemes directly. Other important measures include bisection bandwidth, scalability, and efficient routing algorithms.

In this paper, we first describe a newly proposed network, called the *Recursive Dual-Net* (RDN). The RDN is based on the recursive dual-construction of a symmetric base-network. The dual-construction extends a symmetric network with n nodes and node-degree d to a network with $2n^2$ nodes and node-degree $d + 1$. The RDN is especially suitable for the interconnection network of parallel computers with millions of nodes. It can connect a huge number of nodes with just a small number of links per node and very short diameters. For example, a 2-level RDN with a 5-ary, 2-cube as the base-network can connect more than 3-million nodes with only 6 links per node and its diameter equals to 22.

RDN has been proven to have excellent topological properties including small node-degree, short diameter, efficient routing algorithms, and efficient communication schemes for collective communication. However, to be an effective, high-performance interconnection network of parallel computers, it is important that efficient algorithms that can perform some basic computational tasks in computer science do exist. The significant contribution of this paper is to develop efficient algorithms for parallel prefix computation and parallel sorting on RDN. We also demonstrate certain techniques for algorithmic design on RDN that might be useful while developing efficient algorithms for other important computational problems on RDN.

The prefix computation is fundamental to most numerical algorithms. Let $\oplus$ be an associative binary operation. Given n numbers $c_0,c_1,\ldots,c_{n-1}$, prefix computation is to compute all of the prefixes of the expression $c_0 \oplus c_1 \oplus \ldots \oplus c_{n-1}$. The parallel sorting on networks is an important problem for many applications using parallel computer systems. For an n-cube, the best known deterministic sorting algorithm can sort in $O(n\log n)$ time in the worst case [13]. However, the algorithm is not practical due to the hidden large constant. The most popular algorithm for parallel sorting on networks is Batcher's bitonic sorting algorithm. The communication and computation time complexities of the proposed algorithm for parallel prefix computation on $RDN^k(B)$, $k > 0$, are $2^{k+1}\text{-}2+2^kT_{comm}(0)$ and $2^{k+1}\text{-}2+2^kT_{comp}(0)$, respectively, where $T_{comm}(0)$ and $T_{comp}(0)$ are the communication and computation time complexities of the algorithm for parallel prefix computation on the base-network B, respectively. The proposed algorithm for sorting on an RDN with an m-cube as its base-network is based on the bitonic sorting. In $RDN^k(Q_m)$, assume that each node holds a single data item, the sorting algorithm runs in $O((m2^k)^2)$ computation steps and $O((km2^k)^2)$ communication steps.

The rest of this paper is organized as follows: Section 2 describes the recursive dual-net in detail. Section 3 describes the proposed algorithm for parallel prefix computation on an RDN. Section 4 describes the presentation of an RDN with an m-cube as its base-network. Section 5 describes the proposed sorting algorithm on $RDN^k(Q_m)$. Section 6 concludes the paper and presents some future research directions.

2. RECURSIVE DUAL-NETS

Let G be an undirected graph. The size of G, denoted as $|G|$, is the number of vertices. A path from node s to node t in G is denoted by $s \rightarrow t$. The length of the path is the number of edges in the path. For any two nodes s and t in G, we denote $L(s,t)$ as the length of a shortest path connecting s and t. The diameter of G is defined as $D(G) = \max\{L(s,t) \mid s,t \in G\}$.

For any two nodes s and t in G, if there is a path connecting s and t, we say that G is a connected graph. A graph is symmetric if it is connected and every node in the graph looks alike. Suppose that we have a symmetric graph B and there are n_0 nodes in B and the node degree is d_0. A k-level Recursive Dual-Net $RDN^k(B)$, also denoted as $RDN^k(B(n_0))$, can be recursively defined as follows:

1. $RDN^0(B) = B$ is a symmetric graph with n_0 nodes, called *base-network*.
2. For $k > 0$, an $RDN^k(B)$ is constructed from $RDN^{k-1}(B)$ by a *dual-construction* as explained below (also see Fig.1).

Dual-construction: Let $RDN^{k-1}(B)$ be referred to as a *cluster* of level k and $n_{k-1} = |RDN^{k-1}(B)|$ for $k > 0$. An $RDN^k(B)$ is a graph that contains $2n_{k-1}$ clusters of level k as subgraphs. These clusters are divided into two sets with each set containing n_{k-1} clusters. Each cluster in one set is said to be of *type* 0, denoted as C_i^0 where $0 \le i \le n_{k-1}$-1 is the cluster ID. Each cluster in the other set is of *type* 1, denoted as C_j^1, where $0 \le j \le n_{k-1}$-1 is the cluster ID. At level k, each node in a cluster has a new link to a node in a distinct cluster of the other type. We call this link *cross-edge* of level k. By following this rule, for each pair of clusters C_i^0 and C_j^1, there is a unique edge connecting a node in C_j^1 and a node in C_j^1, $0 \le i,j \le n_{k-1}$-1. In Fig. 1, there are n_{k-1} nodes within each cluster $RDN^{k-1}(B)$.

We give two simple examples of recursive dual-nets with k = 1 and 2, in which the base network is a ring with 3 nodes, in Fig. 2 and Fig. 3, respectively. Fig. 2 depicts an $RDN^1(B(3))$ network. There are 3 nodes in the base-network. Therefore, the number of nodes in $RDN^1(B(3))$ is $2*3^2 = 18$. The node-degree is 3 and the diameter is 4.

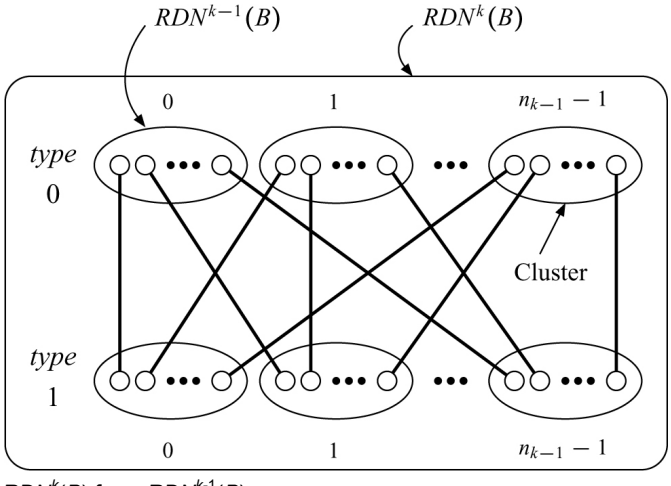


Fig. 1. Build an $RDN^k(B)$ from $RDN^{k-1}(B)$

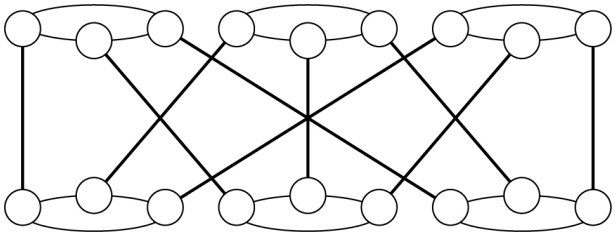


Fig. 2. An $RDN^1(B(3))$ with B as a ring

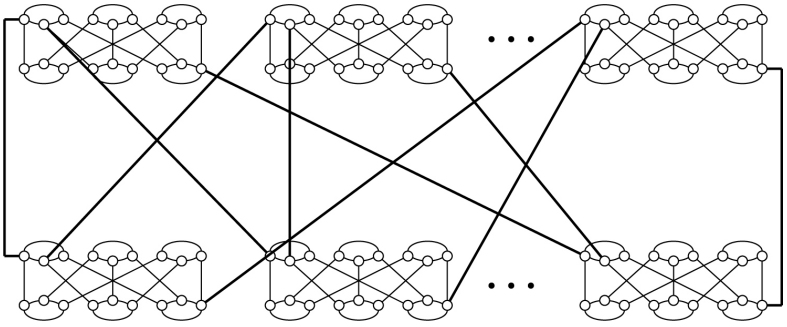


Fig. 3. An $RDN^2(B(3))$ with B as a ring

Fig. 3 shows the $RDN^2(B(3))$ constructed from the $RDN^1(B(3))$ in Fig. 2. We did not show all of the nodes in the figure. The number of nodes in $RDN^2(B(3))$ is $2*18^2 = 648$. The node degree is 4 and the diameter is 10.

Similarly, we can construct an $RDN^3(B(3))$ containing $2*648^2 = 839{,}808$ nodes with a node degree of 5 and a diameter of 22. In contrast, the 839,808-node 3D torus machine (adopted by IBM Blue Gene/L [14]) is configured as 108*108*72 nodes. Its diameter is equal to 54 + 54 + 36 = 144 with a node degree of 6.

We can see from the recursive dual-construction described above that an $RDN^k(B)$ is a symmetric network with the node-degree d_0+k if the base-network is a symmetric network with the node-degree d_0. The following theorem is from [15].

Theorem 1. Assume that the base-network B is a symmetric graph with the size n_0, the node-degree d_0, and the diameter D_0. Then, the size of $RDN^k(B)$ is $(2n_0)^{2k}/2$, the node-degree is d_0+k, the diameter is $2^kD_0+2^{k+1}-2$, and the bisection bandwidth is $[(2n_0)^{2k}/8]$.

The *cost ratio* $CR(G)$ for measuring the combined effects of the hardware cost and the software efficiency of an interconnection network was also proposed in [15]. Let $|(G)|$, $d(G)$, and $D(G)$ be the number of nodes, the node-degree, and the diameter of G, respectively. We define $CR(G)$ as

$$CR(G) = (d(G)+D(G)) / \log_2|(G)|$$

The cost ratio of an n-cube is 2 regardless of its size. The *CRs* for some $RDN^k(B)$ are shown in Table 1. Two small networks including 3-ary 3-cube and 5-ary 2-cube are selected as practical base networks. For INs of a size around 1K, we set $k = 1$, while for INs of a size larger than 1M,

Table 1. *CR*s for some $RDN^k(B)$

Network	*n*	*d*	*D*	*CR*
10-cube	1,024	10	10	2.00
RDN1(B(25))	1,250	5	10	1.46
RDN1(B(27))	1,458	7	8	1.43
3D-Tori(10)	1,000	6	15	2.11
22-cube	4,194,304	22	22	2.00
RDN2(B(25))	3,125,000	6	22	1.30
RDN2(B(27))	4,251,528	8	18	1.18
3D-Tori(160)	4,096,000	6	240	11.20

we set $k = 2$. The results show that the cost ratios of $RDN^k(B)$ are better than hypercubes and 3D-tori in all cases.

A presentation for $RDN^k(B)$ that provides a unique ID to each node in $RDN^k(B)$ is described as follows. Let the IDs of nodes in B, denoted as ID_0, be i, $0 \le i \le n_0$-1. The ID_k of node u in $RDN^k(B)$ for $k > 0$ is a triple (u_0,u_1,u_2), where u_0 is a 0 or 1, u_1 and u_2 belong to ID_{k-1}. We call u_0, u_1, and u_2 typeID, clusterID, and nodeID of u, respectively. With this ID presentation, (u,v) is a cross-edge of level k in $RDN^k(B)$ iff $u_0 \neq v_0$, $u_1 = v_2$, and $u_2 = v_1$. In general, ID_i, $1 \le i \le k$, can be defined recursively as follows: $ID_i = (b,ID_{i-1},ID_{i-1})$, where $b = 0$ or 1. A presentation example is shown in Fig. 4.

The ID of a node u in $RDN^k(B)$ can also be presented by a unique integer i, $0 \le i \le (2n_0)^{2k}/2-1$, where i is the lexicographical order of the triple (u_0,u_1,u_2). For examples, the ID of node (1,1,2) in $RDN^1(B(3))$ is $1*3^2 + 1*3 + 2 = 14$ (see Fig. 5); the ID of node (1,(0,2,2),(1,0,1)) in $RDN^2(B(3))$ is $1*18^2 + 8*18 + 10 = 324 + 144 + 10 = 478$.

A high-performance supercomputer based on the RDN can be implemented easily. We can use the Gigabit Ethernet or Infiniband products, or we can design a switch chip with multiple ports, as the RDN switch or router. Then we can connect ports with high-speed cables just by following the RDN topology.

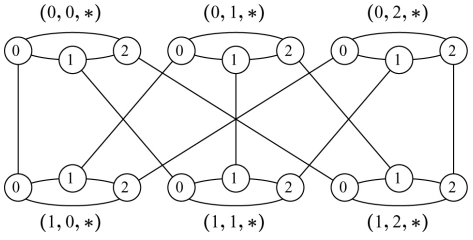


Fig. 4. A presentation of $RDN^1(B(3))$ with B as a ring

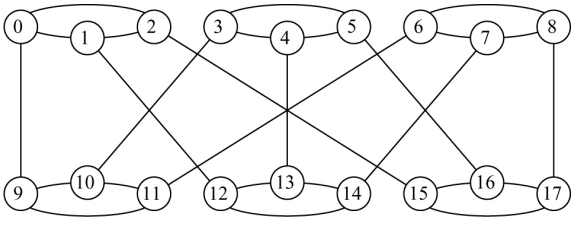


Fig. 5. An $RDN^1(B(3))$ with integer node ID

3. PARALLEL PREFIX COMPUTATION ON RECURSIVE DUAL-NETS

Let $\oplus$ be an associative binary operation. Given n numbers $c_0,c_1,\ldots,c_{n-1}$, parallel prefix computation [16,17] is defined as simultaneously evaluating all of the prefixes of the expression $c_0 \oplus c_1 \oplus \ldots \oplus c_{n-1}$. The ith prefix is $s_i = c_0 \oplus c_1 \oplus \ldots \oplus c_{i-1}$.

The parallel prefix computation can be done efficiently on a recursive dual-net. Assume that each node i, $0 \le i \le n_k-1$, in an $RDN^k(B)$ holds a number c_i. Let x_i and y_i are local variables in node i that will hold prefixes and *total_sum* at the end of the algorithm. The algorithm for a parallel prefix (or diminished prefix which excludes c_i in s_i) computation on $RDN^k(B)$ is a recursive algorithm on k. We assume that the algorithm RDN_prefix(B, c, b) for prefix and diminished prefix computation on the base network (b = 1 for prefix and b = 0 for diminished prefix) is available. We describe the algorithm briefly below.

First, through a recursive call for every cluster of level k, we calculate the local prefix x_i and the local sum y_i in node i, where local prefix and local sum are the prefixes and the sum on the data items in each cluster of level k. To get the prefix of the data items in other clusters, we calculate the diminished prefix of all local sums of the clusters of the same type. This can be done by transferring the local sum to its neighbor via the cross-edge of level k, and then the prefix x'_i and the sum y'_i of all local sums of the same type can be computed by the nodes in every cluster of the other type via a recursive call.

After the second recursive call, the missing parts of the prefixes are ready for the nodes in clusters of type 0. Then, these values are transferred back to the nodes in the cluster of the original type via the cross-edge of level k and are added to its own local prefix. Finally, the algorithm adds the sum y'_i of data items in the nodes in clusters of type 0 to the current prefix of every node j in cluster of type 1. Notice that the value y'_i exists in every node j in the clusters of type 1 when the second recursive call is done.

The formal algorithm for parallel prefix computation on an RDN is specified in Algorithm 1. Examples of prefix_sum on $RDN^1(B)$ and $RDN^2(B)$ are shown in Fig. 6 and Fig. 7, respectively.

Theorem 2. Assume bidirectional-channel communication model. Assume also that each node holds a single data item. Parallel prefix computation on $RDN^k(B)$, $k > 0$, can be done in 2^{k+1}-2+$2^k T_{comm}(0)$ communication steps and 2^{k+1}-2+$2^k T_{comp}(0)$ computation steps, where $T_{comm}(0)$ and $T_{comp}(0)$ are communication and computation steps for prefix computation on the base-network, respectively.

Proof. In Step 1, the local prefix in each cluster of level k is computed. In Steps 2-4, the part of the prefix located in other clusters of the same type is computed. Finally, in Step 5, for clusters of type 1, part of the prefix located in the clusters of type 0 is added to the nodes in the cluster of type 1. It is easy to see the correctness of the algorithm.

Next, we assume that the edges in $RDN^k(B)$ are bidirectional channels, and that at each clock cycle, each node can send or receive one message at most. In Algorithm 1, Step 1 and Step 3 are recursive calls and Step 2 and Step 4 involve one communication step each. Therefore, the complexity for communication satisfies recurrence $T_{comm}(k) = 2T_{comm}(k-1)+2$. Solving the recurrences, we get 2^{k+1}-2+$2^k T_{comm}(0)$. Similarly, Steps 4 and 5 involve one computation step each. The recurrence for computation time satisfies the same concurrence.

Therefore, we conclude that the prefix computation on $RDN^k(B)$ for $k > 0$ can be done in 2^{k+1}

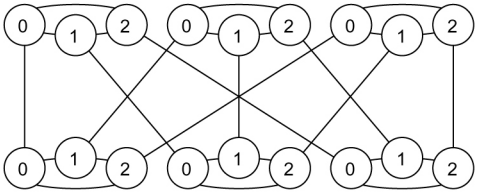
Algorithm 1: Prefix_RDN($RDN^k(B),\ c,\ b$)
Input: Recursive dual-net $RDN^k(B)$, an array of keys c with $|c| = n_k$, and a boolean variable b. Assume that node i holds c_i.
Output: node i holds $x_i = c_0 \oplus c_1 \ldots \oplus c[i]$ if $b = 1$, $c_0 \oplus c_1 \ldots \oplus c_{i-1}$ otherwise
begin
  if $k = 0$ **then** Prefix_RDN($B,\ c,\ b$) /* Assume that RDN_prefix($B,\ c,\ b$) is available. */
  else
    for $RDN_j^{k-1}(B),\ 0 \le j \le n_{k-1} - 1$, **parallel do** /* j is the cluster ID. */
      Prefix_RDN($RDN_j^{k-1}(B),\ c,\ b$);
      /* The values x_i and y_i at node i are the local prefix and
      the local sum in the clusters of level k. */
    for node $i,\ 0 \le i \le n_k - 1$, **parallel do**
      send y_i to node i' via cross-edge of level k;
      $temp_i \leftarrow y_{i'}$;
    for $RDN_j^{k-1}(B),\ 0 \le j \le n_{k-1} - 1$, **parallel do**
      Prefix_RDN($RDN_j^{k-1}(B),\ temp,\ 0$);
      /* Compute the diminished prefix of $temp$ */
      /* The results are denoted as x'_i and y'_i. */
    for node $i,\ 0 \le i \le n_k - 1$, **parallel do**
      send x'_i to node i' via cross-edge of level k;
      $temp_i \leftarrow x'_{i'}$;
      $s_i \leftarrow s_i \oplus temp_i$;
    for node $i,\ n_k/2 \le i \le n_k - 1$, **parallel do**
      $s_i \leftarrow s_i \oplus y'_i$;
  endif
end

-2+2$^k T_{comm}(0)$ communication steps and 2^{k+1}-2+2$^k T_{comp}(0)$ computation steps, where $T_{comm}(0)$ and $T_{comp}(0)$ are communication and computation steps for prefix computation on the base-network, respectively. ■

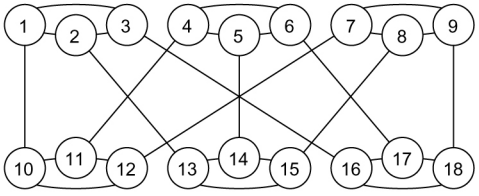
The extension of the parallel prefix algorithm to the general case where each node initially holds more than one data item is straightforward. Let the size of array c be $m > n$. The algorithm consists of three stages. In the first stage, each node does a prefix computation on its own data set of size m/n sequentially. In the second stage, the algorithm performs a diminished parallel computation on the RDN as described in Algorithm 1 with $b = 0$ and c_i equals the local sum. In the third stage, for each node, the algorithm combines the result from this last computation with the locally computed prefixes to get the final result. We show the parallel prefix computation for the general case in theorem 3.

Theorem 3. Assume the bidirectional-channel communication model. Assume also that the size of the input array is m, and that each node holds m/n_k numbers. Parallel prefix computation on $RDN^k(B)$, $k > 0$, can be done in 2^{k+1}-2 + 2$^k T_{comm}(0)$ communication steps and $2m/n_k + 2^{k+1}$-3 + 2$^k T_{comp}(0)$ computation steps, where $T_{comm}(0)$ and $T_{comp}(0)$ are communication and computation steps for prefix computation on the base-network with each node holds one single number, respectively.

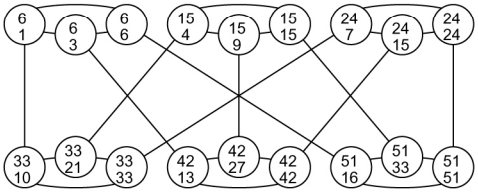
```
prefix_sum (1,2,3, 4, 5, 6, 7, 8, 9,10,11,12,13, 14, 15, 16, 17, 18)
         = (1,3,6,10,15,21,28,36,45,55,66,78,91,105,120,136,153,171)
```



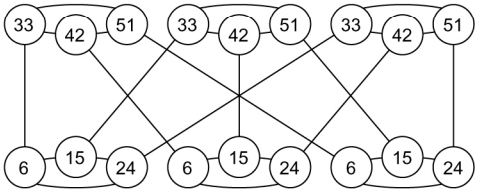
(a) Presentation of $RDN^1(B(3))$



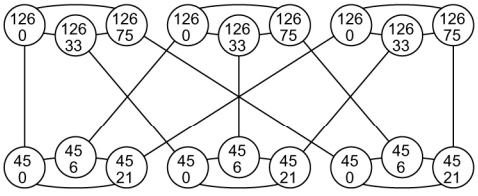
(b) Data distribution



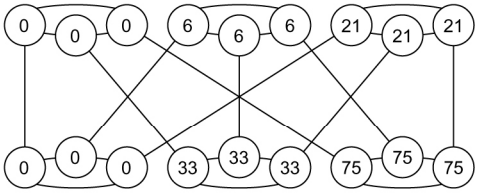
(c) $t[u]$ (top) and $s[u]$ (bottom)



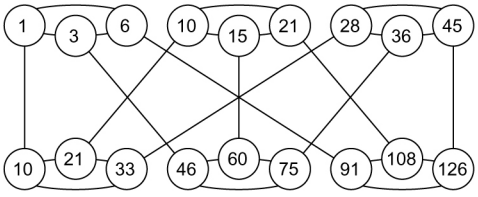
(d) Send $t[u]$ and receive $temp[u]$



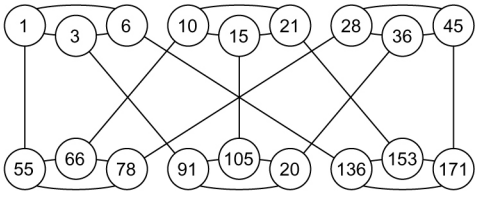
(e) $t'[u]$ (top) and $s'[u]$ (bottom)



(f) Send $s'[u]$ and receive $temp[u]$



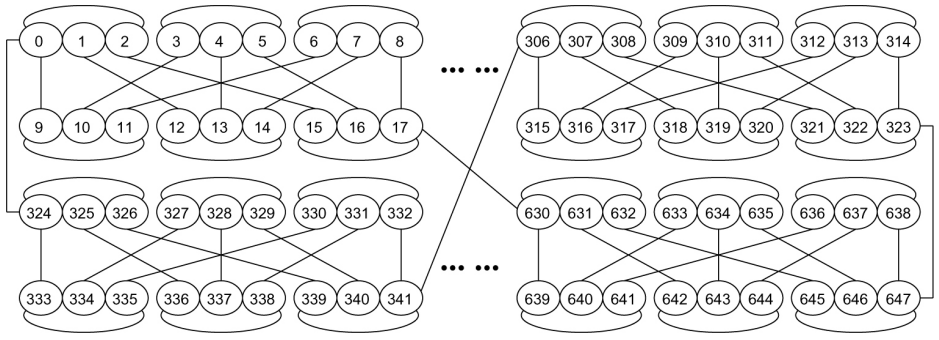
(g) $s[u] \leftarrow s[u] \oplus temp[u]$



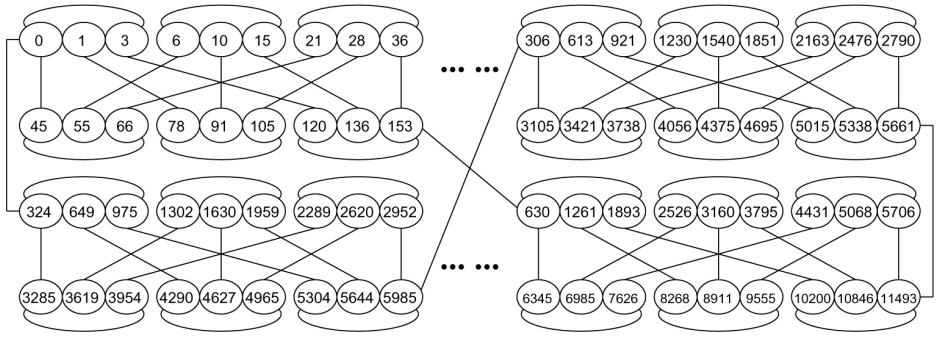
(h) In type 1, $s[u] \leftarrow s[u] \oplus t'[u]$ (final result)

Fig. 6. An example of prefix_sum on $RDN^1(B(3))$

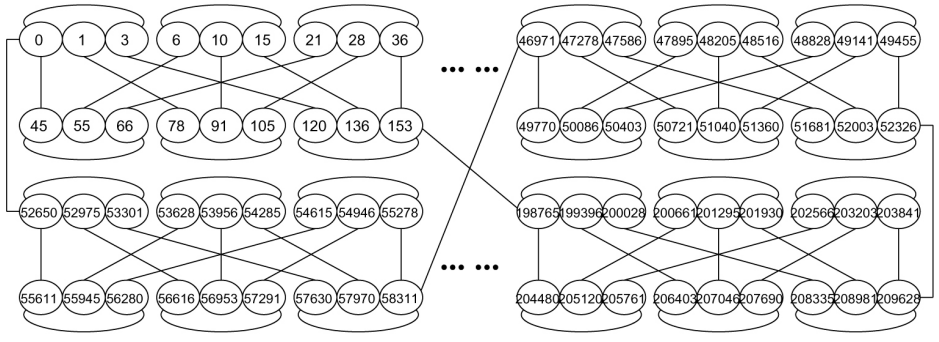
Proof. The first and the third stages of the algorithm contains only local computations inside each node and the total number of computations are (m/n_k) -1 and m/n_k, respectively. In the second stage, the algorithm performs parallel prefix computation on an RDN with each node holding a single number. Following Theorem 2, it requires 2^{k+1}-2+$2^k T_{comm}(0)$ communication steps and 2^{k+1}-2+$2^k T_{comp}(0)$ computation steps. Therefore, we conclude that the parallel prefix computation of array of size $m > n_k$ on $RDN^k(B)$ requires 2^{k+1}-2+$2^k T_{comm}(0)$ communication steps and $(2m/n_k+2^{k+1}-3)+2^k T_{comp}(0)$ computation steps. ■



(a) Data distribution



(b) s for $k = 1$



(c) s for $k = 2$ (final prefix result)

Fig. 7. An example of prefix_sum on $RDN^2(B(3))$

4. A RECURSIVE DUAL-NET WITH A HYPERCUBE AS ITS BASE-NETWORK

In this paper, we focus our design of sorting algorithm on $RDN^k(Q_m)$, where the base-network is an m-cube Q_m. First, we will describe a presentation for $RDN^2(Q_m)$.

A presentation for $RDN^k(Q_m)$ that provides a unique ID to each node in $RDN^k(Q_m)$ is described as follows. Let the ID of a node in Q_m, denoted as ID_0, be an m-bit number $b_{m-1}\ldots b_1b_0$. The ID_k of a node u in $RDN^k(Q_m)$ for $k > 0$ is a triple (u_0,u_1,u_2), where u_0 is a 0 or 1, u_1 and u_2 belong to ID_{k-1}. We call u_0, u_1, and u_2 typeID, clusterID, and nodeID of u, respectively. With this ID pres-

entation, (u,v) is a cross-edge of level k in $RDN^k(Q_m)$ iff $u_0 \neq v_0$, $u_1 = v_2$, and $u_2 = v_1$. In general, ID_i, $1 \leq i \leq k$, can be defined recursively as follows: $ID_i = (c, ID_{i-1}, ID_{i-1})$, where $c = 0$ or 1.

In this paper, we present a parallel sorting algorithm on $RDN^2(Q_m)$ ($k = 2$). The format of the node ID is given in Fig. 8.

In Fig. 8, $b^0_{m-1} \cdots b^0_0$ is the node ID in Q_m; $(c_0, b^1_{m-1} \cdots b^1_0, b^0_{m-1} \cdots b^0_0)$ is the node ID in $RDN^1(Q_m)$; and $(c_1, b^3_{m-1} \cdots b^3_0, b^2_{m-1} \cdots b^2_0)$ is the clusterID of a node in $RDN^2(Q_m)$.

Each node in an $RDN^2(Q_m)$ has m+2 links. Because $b^0_{m-1} \cdots b^0_0$ is the ID of a node in the m-cube, there is a link connecting two nodes if the IDs of the two nodes differ only in one bit position. There are another two links (cross-edges) for each node in an $RDN^2(Q_m)$. For $k = 1$, there is a link between nodes $(0, b^1_{m-1} \cdots b^1_0, b^0_{m-1} \cdots b^0_0)$ and $(1, b^0_{m-1} \cdots b^0_0, b^1_{m-1} \cdots b^1_0)$. For $k = 2$, there is a link between nodes $(0, c_1, b^3_{m-1} \cdots b^3_0, b^2_{m-1} \cdots b^2_0, c_0, b^1_{m-1} \cdots b^1_0, b^0_{m-1} \cdots b^0_0)$ and $(1, c_0, b^1_{m-1} \cdots b^1_0, b^0_{m-1} \cdots b^0_0, c_1, b^3_{m-1} \cdots b^3_0, b^2_{m-1} \cdots b^2_0)$.

Two presentation examples of $RDN^1(Q_m)$ and $RDN^2(Q_m)$ with $m = 2$ are shown in Fig. 9 and Fig. 10, respectively.

The node ID in an $RDN^2(Q_m)$ has $4m$+3 bits. Because the sorting algorithm presented in this paper requires only the communication between nodes u and $u^{(i)}$ where the IDs of u and $u^{(i)}$ differ in a bit position i for $i = 4m+2,\ldots,1,0$. We give a simple routing algorithm for those node pairs. The routing algorithm for a general case was given in [15].

If we use $b_{4m+2} \ldots b_0$ to denote the $(4m+3)$-bit ID of a node in an $RDN^2(Q_m)$, then

- $b_{m-1} \ldots b_0 = b^0_{m-1} \cdots b^0_0$;
- $b_{2m-1} \ldots b_m = b^1_{m-1} \cdots b^1_0$;
- $b_{2m} = c_0$;
- $b_{3m} \ldots b_{2m+1} = b^2_{m-1} \cdots b^2_0$;
- $b_{4m} \ldots b_{3m+1} = b^3_{m-1} \cdots b^3_0$;
- $b_{4m+1} = c_1$; and
- $b_{4m+2} = c_2$.

The routing algorithm between nodes u and $u^{(i)}$ for $i = 0, 1, \ldots, 4m+2$ is simply described as below:

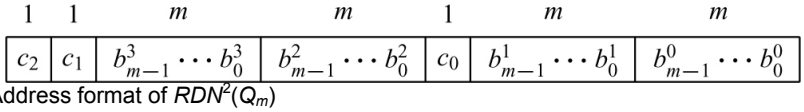


Fig. 8. Address format of $RDN^2(Q_m)$

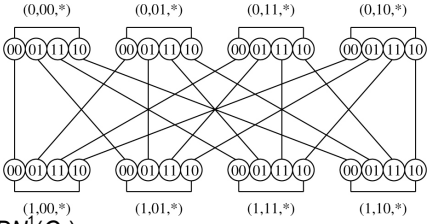


Fig. 9. A presentation of $RDN^1(Q_2)$

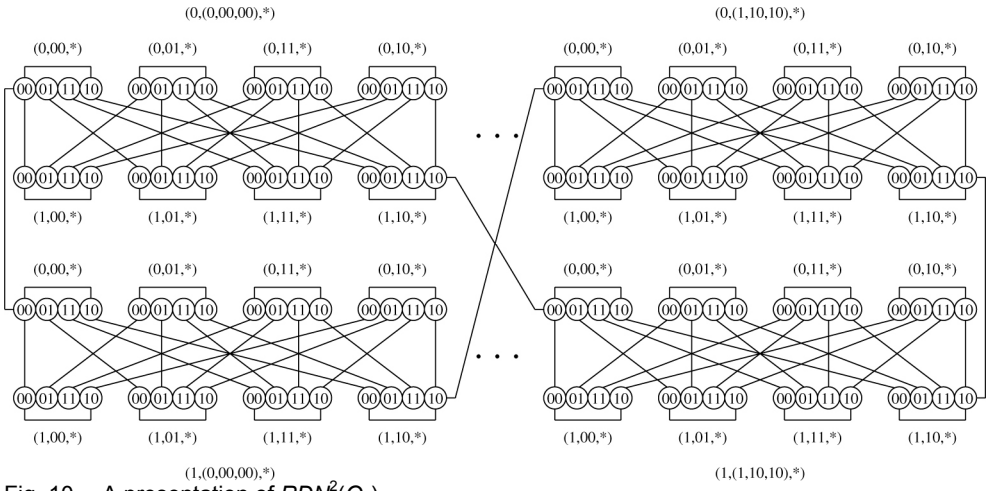


Fig. 10. A presentation of $RDN^2(Q_2)$

- **Case 1:** If m-1$\geq i\geq$ 0, $i = 2m$, or $i = 4m$+2, nodes u and $u^{(i)}$ can send and receive data each other directly because there is a link connecting the two nodes.
- **Case 2:** If $2m$-1$\geq i\geq m$, nodes u and $u^{(i)}$ can communicate along with the $2m$th dimension first such that the bits $b_{2m-1}\ldots b_m$ will be exchanged to the positions $b_{m-1}\ldots b_0$. Then it becomes the same as Case 1. The final step is to route along with the $2m$th dimension again (exchange back).
- **Case 3:** If $4m\geq i\geq 2m$+1, nodes u and $u^{(i)}$ can communicate along with the $4m$+2nd dimension first such that the bits $b_{4m}\ldots b_{2m+1}$ will be exchanged to the positions $b_{2m-1}\ldots b_0$. Then it becomes the same as Case 2 or Case 1. The final step is to route along with the $4m$+2nd dimension again (exchange back).

The following example shows the routing path between nodes u = 0 0 00 00 0 00 00 and $u^{(i)}$ in an $RDN^2(Q_2)$ for i = 0, 1, …, 10.

0) 0 0 00 00 0 00 00 → 0 0 00 00 0 00 0<u>1</u>
1) 0 0 00 00 0 00 00 → 0 0 00 00 0 00 <u>1</u>0
2) 0 0 00 00 0 00 00 → 0 0 00 00 1 00 00 → 0 0 00 00 1 00 01 → 0 0 00 00 0 0<u>1</u> 00
3) 0 0 00 00 0 00 00 → 0 0 00 00 1 00 00 → 0 0 00 00 1 00 10 → 0 0 00 00 0 <u>1</u>0 00
4) 0 0 00 00 0 00 00 → 0 0 00 00 <u>1</u> 00 00
5) 0 0 00 00 0 00 00 → 1 0 00 00 0 00 00 → 1 0 00 00 0 00 01 → 1 0 00 0<u>1</u> 0 00 00
6) 0 0 00 00 0 00 00 → 1 0 00 00 0 00 00 → 1 0 00 00 0 00 10 → 1 0 00 <u>1</u>0 0 00 00
7) 0 0 00 00 0 00 00 → 1 0 00 00 0 00 00 → 1 0 00 00 1 00 00 → 1 0 00 00 1 00 01 → 1 0 00 00 0 01 00 → 0 0 0<u>1</u> 00 0 00 00
8) 0 0 00 00 0 00 00 → 1 0 00 00 0 00 00 → 1 0 00 00 1 00 00 → 1 0 00 00 1 00 10 → 1 0 00 00 0 10 00 → 0 0 <u>1</u>0 00 0 00 00
9) 0 0 00 00 0 00 00 → 1 0 00 00 0 00 00 → 1 0 00 00 1 00 00 → 0 <u>1</u> 00 00 0 00 00
10) 0 0 00 00 0 00 00 → <u>1</u> 0 00 00 0 00 00

Theorem 4. In the bidirectional channel communication model, the communication between

nodes u and $u^{(i)}$ in $RDN^k(Q_m)$, where the addresses of u and $u^{(i)}$ differ in ith bit position for $0 \leq i < 2^k m+2^k-1$ takes at most $t_k = 2k + 1$ steps.

Proof. For $k = 0$, an RDN is an m-cube, since there is a direct link in every dimension, the communication takes only one step.

For $k = 1$, a node address has $2m+1$ bits. The routing in each bit of the clusterID (m bits) takes two more steps: one for going to another type and one for coming back. Therefore, it takes three steps.

Assume it is true for k-1. For k, the clusterID has $2^{k-1}m + 2^{k-1}-1$ bits. Routing in each bit of this part takes two more steps. Therefore, the communication time is $t_k = t_{k-1}+ 2 = 2(k-1) + 3 = 2k + 1$. Therefore, the theorem is correct. ■

5. PARALLEL SORTING ON RECURSIVE DUAL-NETS

In this section, we present a new sorting algorithm on $RDN^k(Q_m)$ based on parallel bitonic sorting. Bitonic sorting repeatedly merges 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 by a *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 [18]. 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 into 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 a *bitonic merge* (BM).

Given a set of elements, we must transform them into a bitonic sequence. This can be done recursively by doubling the size of the bitonic sequence. The *bitonic sorting network* for sorting N numbers consists of $\log_2 N$ bitonic sorting stages, where the ith stage is composed of $N/2^i$ alternating increasing and decreasing bitonic merges of size 2^i.

Fig. 11 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, k = 2, 4, 8, 16, respectively. The last merging network ($\oplus$BM[16]) sorts the input.

A bitonic sorting example on a 4-cube is shown in Fig. 12. The computational complexity for sorting $N = 2^n$ numbers in an n-cube is $O(n(n+1)/2) = O(n^2)$. Similarly, the communications take $O(n^2)$ steps.

We assume that each node in $RDN^k(Q_m)$ holds a single element (number). The sorting algo-

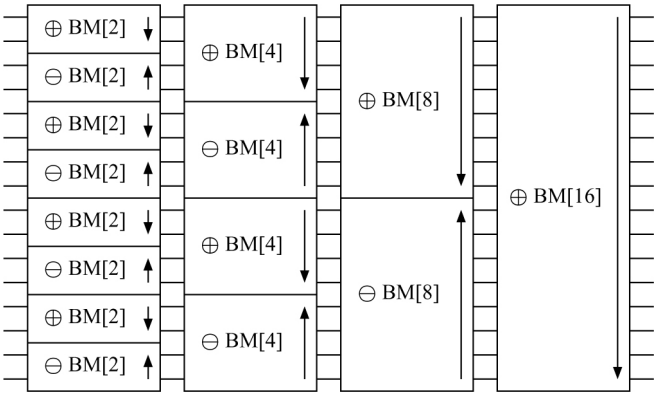


Fig. 11. A bitonic sorting network of size 16

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

The node address has $2^k m+2^k-1$ bits (dimensions) and there are only $k+m$ links per node in $RDN^k(Q_m)$. For $k = 2$, there are four m-bit fields and three single-bit fields. We build a *path* between nodes u and $u^{(i)}$ for $0 \leq i < 2^k m+2^k-1$ in seven cases (one for each field). The sorting algorithm on $RDN^2(Q_m)$ is formally given in Algorithm 2.

There are four parameters in the algorithm: *my_id* is the binary node address; *my_number* is the number residing in the node; m is the 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 $4m+2$. Each iteration of the loop doubles the size of the bitonic sequences. The bitonic merge is done by the inner **for** loop, which

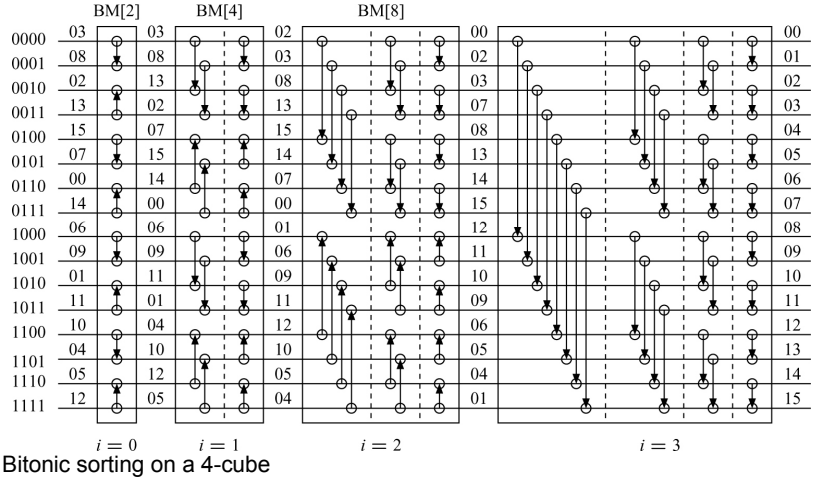


Fig. 12. Bitonic sorting on a 4-cube

Algorithm 2 Bitonic_Sort_RDN (*my_id, my_number, m, result*)
begin
  $result \leftarrow my\_number$;
  for $i \leftarrow 0$ **to** $4m+2$ **do**
    for $j \leftarrow i$ **downto** 0 **do**
      if $m-1 \geq j \geq 0$ **then** $t = j$; $path =$
        $(c_2, c_1, b^3_{m-1}\cdots b^3_0, b^2_{m-1}\cdots b^2_0, c_0, b^1_{m-1}\cdots b^1_0, b^0_{m-1}\cdots b^0_t\cdots b^0_0) \rightarrow$
        $(c_2, c_1, b^3_{m-1}\cdots b^3_0, b^2_{m-1}\cdots b^2_0, c_0, b^1_{m-1}\cdots b^1_0, b^0_{m-1}\cdots \overline{b^0_t}\cdots b^0_0)$;
      else if $2m-1 \geq j \geq m$ **then** $t = j-m$; $path =$
        $(c_2, c_1, b^3_{m-1}\cdots \overline{b^3_0}, b^2_{m-1}\cdots b^2_0, c_0, b^1_{m-1}\cdots b^1_t\cdots b^1_0, b^0_{m-1}\cdots b^0_0) \rightarrow$
        $(c_2, c_1, b^3_{m-1}\cdots b^3_0, b^2_{m-1}\cdots b^2_0, \overline{c_0}, b^0_{m-1}\cdots b^0_0, b^1_{m-1}\cdots b^1_t\cdots b^1_0) \rightarrow$
        $(c_2, c_1, b^3_{m-1}\cdots b^3_0, b^2_{m-1}\cdots b^2_0, \overline{c_0}, b^0_{m-1}\cdots b^0_0, b^1_{m-1}\cdots \overline{b^1_t}\cdots b^1_0) \rightarrow$
        $(c_2, c_1, b^3_{m-1}\cdots b^3_0, b^2_{m-1}\cdots b^2_0, c_0, b^1_{m-1}\cdots \overline{b^1_t}\cdots b^1_0, b^0_{m-1}\cdots b^0_0)$;
      else if $j = 2m$ **then** $path =$
        $(c_2, c_1, b^3_{m-1}\cdots b^3_0, b^2_{m-1}\cdots b^2_0, c_0, b^1_{m-1}\cdots b^1_0, b^0_{m-1}\cdots b^0_0) \rightarrow$
        $(c_2, c_1, b^3_{m-1}\cdots b^3_0, b^2_{m-1}\cdots b^2_0, \overline{c_0}, b^1_{m-1}\cdots b^1_0, b^0_{m-1}\cdots b^0_0)$;
      else if $3m \geq j \geq 2m+1$ **then** $t = j-(2m+1)$; $path =$
        $(c_2, c_1, b^3_{m-1}\cdots b^3_0, b^2_{m-1}\cdots b^2_t\cdots b^2_0, c_0, b^1_{m-1}\cdots b^1_0, b^0_{m-1}\cdots b^0_0) \rightarrow$
        $(\overline{c_2}, c_0, b^1_{m-1}\cdots b^1_0, b^0_{m-1}\cdots b^0_0, c_1, b^3_{m-1}\cdots b^3_0, b^2_{m-1}\cdots b^2_t\cdots b^2_0) \rightarrow$
        $(\overline{c_2}, c_0, b^1_{m-1}\cdots b^1_0, b^0_{m-1}\cdots b^0_0, c_1, b^3_{m-1}\cdots b^3_0, b^2_{m-1}\cdots \overline{b^2_t}\cdots b^2_0) \rightarrow$
        $(c_2, c_1, b^3_{m-1}\cdots b^3_0, b^2_{m-1}\cdots \overline{b^2_t}\cdots b^2_0, c_0, b^1_{m-1}\cdots b^1_0, b^0_{m-1}\cdots b^0_0)$;
      else if $4m \geq j \geq 3m+1$ **then** $t = j-(3m+1)$; $path =$
        $(c_2, c_1, b^3_{m-1}\cdots b^2_t\cdots b^3_0, b^2_{m-1}\cdots b^2_0, c_0, b^1_{m-1}\cdots b^1_0, b^0_{m-1}\cdots b^0_0) \rightarrow$
        $(\overline{c_2}, c_0, b^1_{m-1}\cdots b^1_0, b^0_{m-1}\cdots b^0_0, c_1, b^3_{m-1}\cdots b^2_t\cdots b^3_0, b^2_{m-1}\cdots b^2_0) \rightarrow$
        $(\overline{c_2}, c_0, b^1_{m-1}\cdots b^1_0, b^0_{m-1}\cdots b^0_0, \overline{c_1}, b^2_{m-1}\cdots b^2_0, b^3_{m-1}\cdots b^2_t\cdots b^3_0) \rightarrow$
        $(\overline{c_2}, c_0, b^1_{m-1}\cdots b^1_0, b^0_{m-1}\cdots b^0_0, \overline{c_1}, b^2_{m-1}\cdots \overline{b^2_0}, b^3_{m-1}\cdots \overline{b^2_t}\cdots b^3_0) \rightarrow$
        $(\overline{c_2}, c_0, b^1_{m-1}\cdots b^1_0, b^0_{m-1}\cdots b^0_0, c_1, b^3_{m-1}\cdots \overline{b^2_t}\cdots b^3_0, b^2_{m-1}\cdots b^2_0) \rightarrow$
        $(c_2, c_1, b^3_{m-1}\cdots \overline{b^2_t}\cdots b^3_0, b^2_{m-1}\cdots b^2_0, c_0, b^1_{m-1}\cdots b^1_0, b^0_{m-1}\cdots b^0_0)$;
      else if $j = 4m+1$ **then** $path =$
        $(c_2, c_1, b^3_{m-1}\cdots b^3_0, b^2_{m-1}\cdots b^2_0, c_0, b^1_{m-1}\cdots b^1_0, b^0_{m-1}\cdots b^0_0) \rightarrow$
        $(c_2, \overline{c_1}, b^3_{m-1}\cdots b^3_0, b^2_{m-1}\cdots b^2_0, c_0, b^1_{m-1}\cdots b^1_0, b^0_{m-1}\cdots b^0_0)$;
      else if $j = 4m+2$ **then** $path =$
        $(c_2, c_1, b^3_{m-1}\cdots b^3_0, b^2_{m-1}\cdots b^2_0, c_0, b^1_{m-1}\cdots b^1_0, b^0_{m-1}\cdots b^0_0) \rightarrow$
        $(\overline{c_2}, c_1, b^3_{m-1}\cdots b^3_0, b^2_{m-1}\cdots b^2_0, c_0, b^1_{m-1}\cdots b^1_0, b^0_{m-1}\cdots b^0_0)$;
      send *result* along with *path*;
      receive *number* along with *path*;
      if (*my_id* AND $2^{i+1} \neq$ *my_id* AND 2^i) /* max */
        if (*number* $>$ *result*) $result \leftarrow number$;
      else /* min */
        if (*number* $<$ *result*) $result \leftarrow number$;
end

takes the order of $j = i$ to 0. In the current step j, nodes u and $u^{(j)}$ exchange their numbers with each other through the *path* and compare the two numbers, where the addresses of nodes u and $u^{(j)}$ differ only in the dimension j. After the comparison, node u keeps the number as *result* based on the following rule: If the value of the jth bit of the address differs from the value of i+1st bit of the address, the node keeps the maximum of the two numbers; minimum otherwise.

Theorem 5. In the bidirectional channel communication model, bitonic sorting on $RDN^k(Q_m)$ with $N = 2^{2^k m+2^k-1}$ nodes can be done in $O((m2^k)^2)$ computation steps and $O((mk2^k)^2)$ com-

munication steps, respectively.

Proof. The Algorithm 2 performs bitonic sorting on $RDN^2(Q_m)$. The outer **for** loop generates bitonic sequences in the dimension order of $i = 0$ to $4m+2$. Each iteration of the loop doubles the size of the bitonic sequences. The bitonic merge is done by the inner **for** loop. At each iteration of the loop, a compare-and-exchange operation is executed between nodes u and $u^{(j)}$ through a path of length at most five (from Theorem 4). There are seven cases since there are four m-bit fields and three single-bit fields in the node address for $k = 2$. It is not difficult to extend Algorithm 2 to $RDN^k(Q_m)$ for $k > 2$. Since bitonic sorting is used, the computation time is $O((m2^k)^2)$ as that on an n-cube, where $n = 2^k m+2^k-1$. From Theorem 4, the worst-case for communication time between the pair u and $u^{(j)}$ is $2k+1$. Therefore, the upper bound of the communication time is $O((mk2^k)^2)$. ■

6. CONCLUDING REMARKS

In this paper, we presented efficient algorithms for parallel prefix computation and parallel sorting on $RDN^2(B)$. The algorithm for parallel sorting is restricted on the case $B = Q_m$. One of the further research topics on RDN is to extend the algorithm for parallel sorting to a general $RDN^k(B)$ assuming that parallel sorting on B can be done efficiently.

The recursive dual-net is a potential candidate for the supercomputers of future generations. It has many interesting properties that are very attractive as an interconnection network of massively parallel computers. To design efficient algorithms for basic computational problems on an interconnection network is an important issue. The other research topics may include the design of efficient algorithms for numerical computations and the fault tolerant routing on recursive dual-nets.

REFERENCE

[1] S. G. Aki, Parallel Computation, Models and Methods, Prentice-Hall, 1997.
[2] F. T. Leighton, Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes, Morgan Kaufmann, 1992.
[3] A. Varma and C. S. Raghavendra, Interconnection Networks for Multiprocessors and Multicomputers: Theory and Practice, IEEE Computer Society Press, 1994.
[4] K. Ghose and K. R. Desai, "Hierarchical cubic networks," IEEE Transactions on Parallel and Distributed Systems, Vol.6, No.4, pp.427–435, April 1995.
[5] Y. Li and S. Peng, "Dual-cubes: a new interconnection network for high-performance computer clusters," Proceedings of the 2000 International Computer Symposium, Workshop on Computer Architecture, ChiaYi, Taiwan, December 2000, pp.51–57.
[6] Y. Li, S. Peng, and W. Chu, "Efficient collective communications in dual-cube," The Journal of Supercomputing, Vol.28, No.1, pp.71–90, April 2004.
[7] F. P. Preparata and J. Vuillemin, "The cube-connected cycles: a versatile network for parallel computation," Commun. ACM, Vol.24, pp.300–309, May 1981.
[8] Y. Saad and M. H. Schultz, "Topological properties of hypercubes," IEEE Transactions on Computers, Vol.37, No.7, pp.867–872, July 1988.
[9] G. H. Chen and D. R. Duh, "Topological properties, communication, and computation on wk-recursive networks," Networks, Vol.24, No.6, pp.303–317, 1994.
[10] G. Vicchia and C. Sanges, "A recursively scalable network vlsi implementation," Future Generation Computer Systems, Vol.4, No.3, pp.235–243, 1988.
[11] TOP500, Supercomputer Sites, http://www.top500.org/, Nov. 2010.
[12] P. Beckman, "Looking toward exascale computing, keynote speaker," in International Conference on

Parallel and Distributed Computing, Applications and Technologies (PDCAT'08), University of Otago, Dunedin, New Zealand, December 2008.

[13] R. Cypher and C. G. Plaxton, "Deterministic sorting in nearly logarithmic time on the hypercube and related computers," Proceedings of the 22dn Annual ACM Symposium on Theory of Computing, 1990, pp.193–203.

[14] N. R. Adiga, M. A. Blumrich, D. Chen, P. Coteus, A. Gara, M. E. Giampapa, P. Heidelberger, S. Singh, B. D. Steinmacher-Burow, T. Takken, M. Tsao, and P. Vranas, "Blue gene/l torus interconnection network," IBM Journal of Research and Development, http://www.research.ibm.com/journal/rd/492/tocpdf.html, Vol.49, No.2/3, pp.265–276, 2005.

[15] Y. Li, S. Peng, and W. Chu, "Recursive dual-net: A new universal network for supercomputers of the next generation," Proceedings of the 9th International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP'09), Taipei, Taiwan, Springer, Lecture Notes in Computer Science (LNCS), June 2009, pp.809–820.

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

[17] W. D. Hillis and G. L. S. Jr, "Data parallel algorithms," Communications of the ACM, Vol.29, No.12, pp.1170–1183, Dec. 1986.

[18] V. Kumar, A. Grama, A. Gupta, and G. Karypis, Introduction to parallel computing: design and analysis of algorithms, Benjamin/Cummings Press, 1994.

Yamin Li

He received his Ph.D in computer science from Tsinghua University in 1989. He currently is a professor in the Department of Computer Science at Hosei University. His research interests include computer arithmetic algorithms, computer architecture, CPU design, parallel and distributed computing, interconnection networks, and fault tolerant computing. Dr. Li is a senior member of the IEEE and a member of the IEEE Computer Society.

Shietung Peng

He received Ph.D (1986) in computer science from the University of Texas in Dallas, Texas. He was a faculty member of the University of Maryland in Baltimore, Maryland, and the University of Aizu in Japan. He joined the Faculty of Computer and Information Science at Hosei University as a full professor in 2000. His research interests are parallel and distributed processing, interconnection networks, fault-tolerant and optical routing.

Wanming Chu

She is a faculty member of the Department of Computer Hardware, the University of Aizu in Japan. Her research interests include computer arithmetic algorithm and hardware implementation, multithreaded computer architecture, interconnection networks, fault tolerant computing, and performance evaluation.