Parallel Prefix Computation in the Recursive Dual-Net

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. In this paper, we propose an efficient algorithm for parallel prefix computation in recursive dual-net, a newly proposed network. The recursive dual-net $RDN^k(B)$ for $k > 0$ has $(2n_0)^{2^k}/2$ nodes and $d_0 + k$ links per node, where n_0 and d_0 are the number of nodes 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 in $RDN^k(B), k > 0$, 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 in the base network B, respectively.

Keywords. Interconnection networks, algorithm, parallel prefix computation

1 Introduction

In massively parallel processor (MPP), the interconnection network plays a crucial role in the issues such as communication performance, hardware cost, computational complexity, fault-tolerance. Much research has been reported in the literature on interconnection networks that can be used to connect parallel computers of large scale (see [1–3] for the review of the early work).

The following two categories have attracted a great research attention. One is the networks of hypercube-like family that have the advantage of short diameters for high-performance computing and efficient communication [4–8]. The other is the networks of 2D/3D meshes or tori that have the advantage of small and fixed node-degrees and easy implementations. Traditionally, most MPPs in the history including those built by NASA, CRAY, FGPS, IBM, use 2D/3D meshes or tori or their variations with extra diagonal links.

Recursive networks also have been proposed as effective interconnection networks for parallel computers of large scale. 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, due to advances in computer technology and competition among computer makers, computers containing hundreds of thousands of nodes have been built [11]. It was predicted that the MPPs of the next decade will contain 10 to 100 millions of nodes [12]. For such a parallel computer of very-large scale, the traditional interconnection networks may no longer satisfy the requirements for the high-performance computing or efficient communication.

For future generations of MPPs 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 *Recursive Dual-Net* (RDN). The RDN is based on recursive dual-construction of a regular base-network. The dual-construction extends a regular 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 the 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 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. The major contribution of this paper is to design efficient algorithm for parallel prefix computation in RDN.

The prefix computation is fundamental to most of 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 $c_0 \oplus c_1 \ldots \oplus c_{n-1}$.

The rest of this paper is organized as follows. Section 2 describes the recursive dual-net in details. Section 3 describes the proposed algorithm for parallel prefix computation in RDN. Section 4 concludes the paper and presents some future research directions.

2 Recursive Dual-Net

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 $D(G) = \max\{L(s, t)|s, t \in G\}$.

For any two nodes s and t in G, if there is a path connecting s and t, we say G is a connected graph. Suppose we have a symmetric connected 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 connected 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 Figure 1).

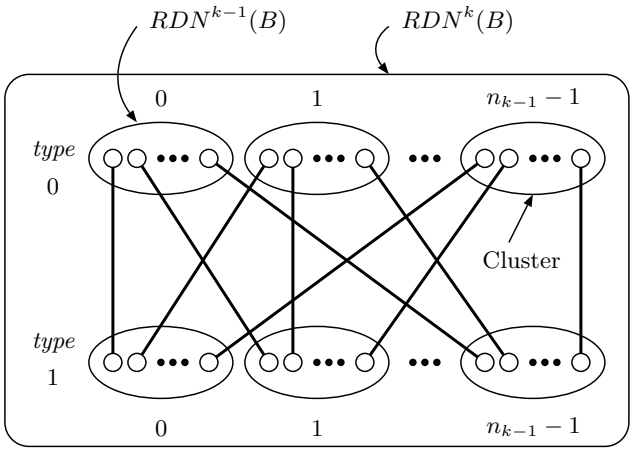


Fig. 1. Build an $RDN^k(B)$ from $RDN^{k-1}(B)$

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 $0 \leq i \leq n_{k-1} - 1$ is the cluster ID. Each cluster in the other set is of *type* 1, denoted as C_j^1, where $0 \leq j \leq n_{k-1} - 1$ 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_i^0 and a node in C_j^1, $0 \leq i, j \leq n_{k-1} - 1$. In Figure 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 Figure 2 and Figure 3, respectively. Figure 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, or 18. The node degree is 3 and the diameter is 4.

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

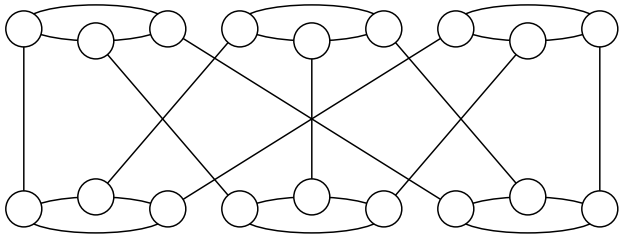


Fig. 2. A Recursive Dual-Net $RDN^1(B(3))$

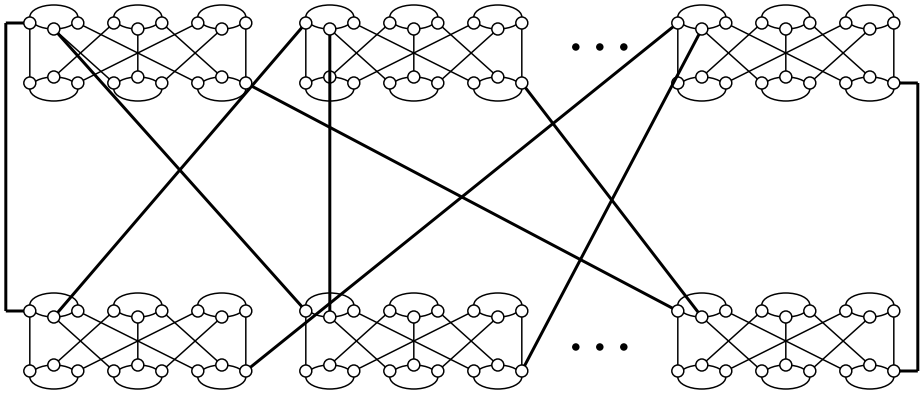


Fig. 3. A Recursive Dual-Net $RDN^2(B(3))$

Similarly, we can construct an $RDN^3(B(3))$ containing 2×648^2, or 839,808 nodes with node degree of 5 and diameter of 22. In contrast, the 839,808-node 3D torus machine (adopt by IBM Blue Gene/L [13]) configured as $108 \times 108 \times 72$ nodes, the 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 regular network with node-degree $d_0 + k$ if the base network is a symmetric regular network with node-degree d_0. The following theorem is from [14].

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

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 [14]. 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

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

The cost ratio of an n-cube is 2 regardless of its size. The CR for some $RDN^k(B)$ is 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 size around 1K, we set $k = 1$, while for INs of size larger 1M, 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.

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

Network	n	d	D	CR
10-cube	1,024	10	10	2.00
$RDN^1(B(25))$	1,250	5	10	1.46
$RDN^1(B(27))$	1,458	7	8	1.43
3D-Tori(10)	1,000	6	15	2.11
22-cube	4,194,304	22	22	2.00
$RDN^2(B(25))$	3,125,000	6	22	1.30
$RDN^2(B(27))$	4,251,528	8	18	1.18
3D-Tori(160)	4,096,000	6	240	11.20

A presentation for $RDN^k(B)$ that provides an 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, $0 \leq i \leq n_0-1$. 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, $1 \leq i \leq k$, 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 Figure 4.

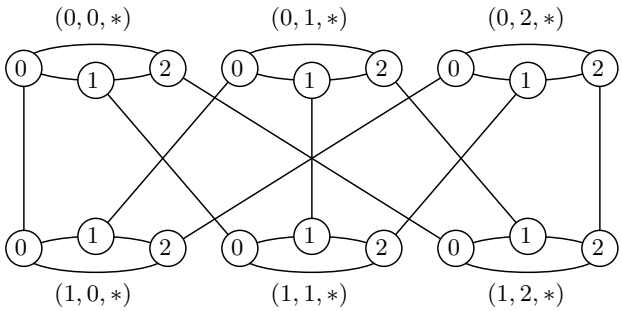


Fig. 4. $RDN^1(B(3))$ presentation

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

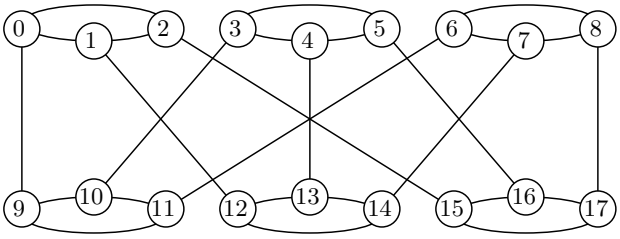


Fig. 5. $RDN^1(B(3))$ with integer node ID

3 Parallel Prefix Computation in Recursive Dual-net

Let $\oplus$ be an associative binary operation. Given n numbers $c_0, c_1, \ldots, c_{n-1}$, parallel prefix computation [15, 16] is defined as simultaneously evaluating all of the prefixes of the expression $c_0 \oplus c_1 \ldots \oplus c_{n-1}$. The ith prefix is $s_i = c_0 \oplus c_1 \ldots \oplus c_{i-1}$.

The parallel prefix computation can be done efficiently in recursive dual-net. Assume that each node $i,\ 0 \leq i \leq n_k - 1$, 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 parallel prefix (or diminished prefix which excludes c_i in s_i) computation in $RDN^k(B)$ is a recursive algorithm on k. We assume that the algorithm RDN_prefix(B, c, b) for prefix and diminished prefix computation in the base network ($b = 1$ for prefix and $b = 0$ for diminished prefix) is available. We describe it 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.

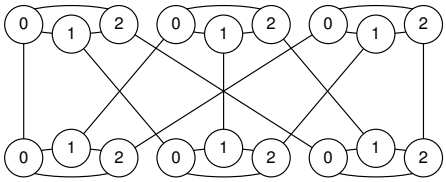
The details are specified in Algorithm 1. Examples of prefix_sum in $RDN^1(B)$ and $RDN^2(B)$ are shown in Figure 6 and Figure 7, respectively.

Algorithm 1: RDN_prefix($RDN^k(B),\ c,\ b$)
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** RDN_prefix($B,\ c,\ b$)
  /* 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. */
      RDN_prefix($RDN_j^{k-1}(B),\ c,\ b$);
      /* 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**
      RDN_prefix($RDN_j^{k-1}(B),\ temp,\ 0$);
      /* 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

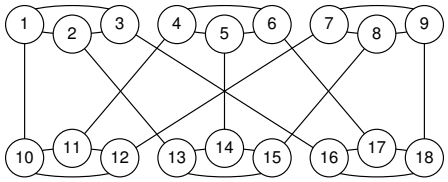
Theorem 2. Assume 1-port, bidirectional-channel communication model. Assume also that each node holds a single data item. Parallel prefix computation in recursive dual-net $RDN^k(B), k > 0$, can be done in $2^{k+1} - 2 + 2^k * T_{comm}(0)$ communication steps and $2^{k+1} - 2 + 2^k * T_{comp}(0)$ computation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

Proof. At Step 1, the local prefix in each cluster of level k is computed. At Steps 2 - 4, The part of the prefix located in other clusters of the same type is

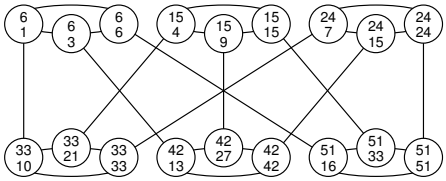
```
prefix_sum (1,2,3, 4, 5, 6, 7, 8, 9,10,11,12,13, 14, 15, 16, 17, 18)
         = (1,3,6,10,15,21,28,36,45,55,66,78,91,105,120,136,153,171)
```



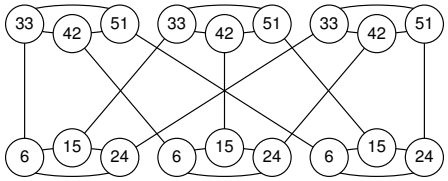
(a) Presentation of $RDN^1(B(3))$



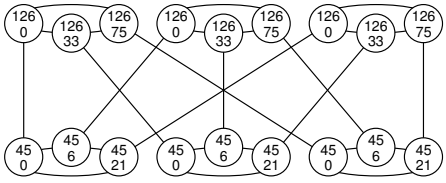
(b) Data distribution



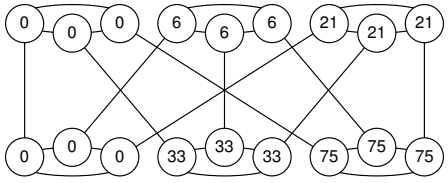
(c) $t[u]$ (top) and $s[u]$ (bottom)



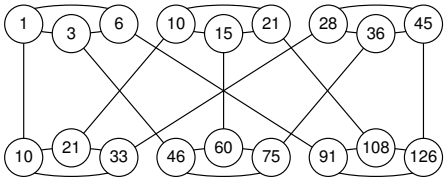
(d) Send $t[u]$ and receive $temp[u]$



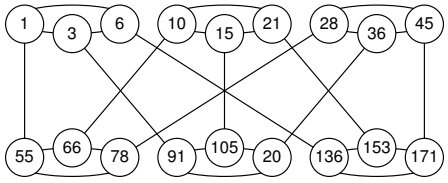
(e) $t'[u]$ (top) and $s'[u]$ (bottom)



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



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

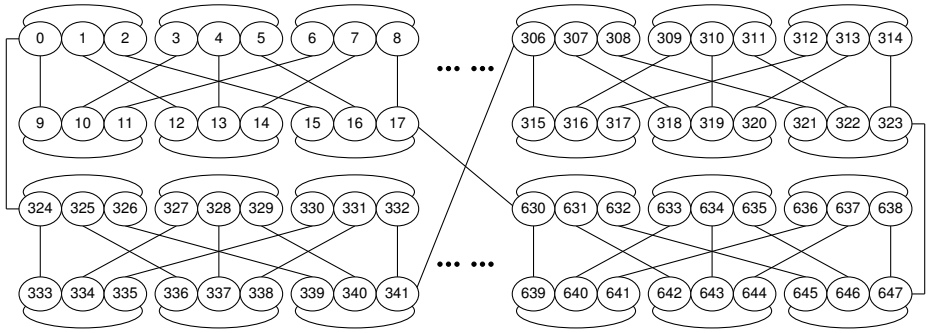


(h) In type 1, $s[u] \leftarrow s[u] \oplus t'[u]$ (final result)

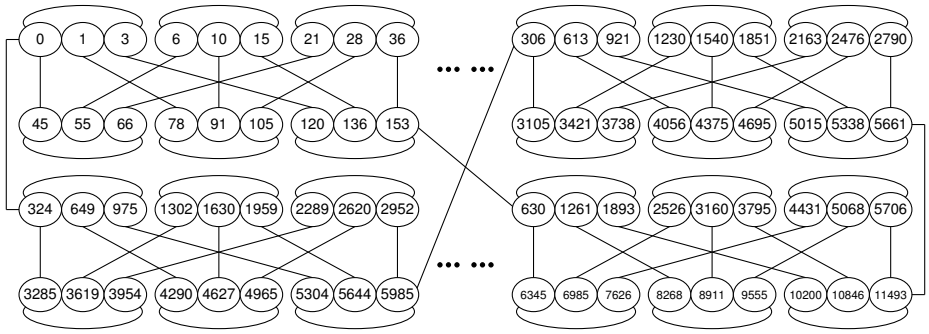
Fig. 6. Example of prefix_sum in $RDN^1(B(3))$

computed. Finally, at 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.

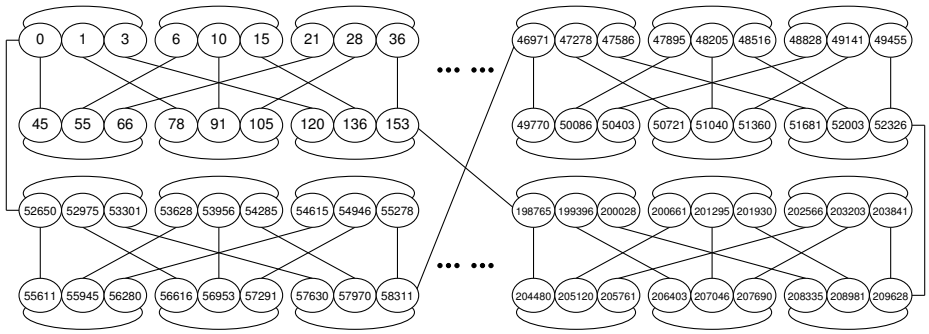
Next, we assume that the edges in $RDN^k(B)$ are bidirectional channels, and at each clock cycle, each node in D_n can send or get at most one message. In



(a) Data distribution



(b) s for $k = 1$



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

Fig. 7. Example of prefix_sum in $RDN^2(B(3))$

Algorithm 1, Step 1 and Step 3 are recursive calls, 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 $T_{comm}(k) = 2^{k+1} - 2 + 2^k * T_{comm}(0)$. 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 in $RDN^k(B)$ for $k > 0$ can be done in $2^{k+1} - 2 + 2^k * T_{comm}(0)$ communication steps and $2^{k+1} + 2^k * T_{comp}(0)$ computation steps, where $T_{comm}(0)$ and $T_{comp}(0)$ are communication and computation steps for prefix computation in the base network, respectively. □

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 do 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 describe in Algorithm 1 with $b = 0$ and c_i equals to the local sum; In 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 1-port, bidirectional-channel communication model. Assume also that the size of the input array is m, and each node holds m/n_k numbers. Parallel prefix computation in recursive dual-net $RDN^k(B), k > 0$, 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 in the base network with each node holds one single number, respectively.

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 in RDN with each node holding a single number. Following Theorem 1, 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 $m > n_k$ in $RDN^k(B)$ 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. □

4 Concluding Remarks

In this paper, we showed an efficient algorithm for parallel prefix computation in recursive dual-net. Recursive dual-net is a potential candidate for the supercomputers of next generations. It has many interesting properties that are very attractive as an interconnection network of massively parallel computer. Design efficient algorithms for basic computational problems in an interconnection network is an important issue. The further research will include design of efficient searching and sorting algorithms for recursive dual-net.

References

1. Aki, S.G.: Parallel Computation: Models and Methods. Prentice-Hall (1997)
2. Leighton, F.T.: Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes. Morgan Kaufmann (1992)
3. Varma, A., Raghavendra, C.S.: Interconnection Networks for Multiprocessors and Multicomputers: Theory and Practice. IEEE Computer Society Press (1994)
4. Ghose, K., Desai, K.R.: Hierarchical cubic networks. IEEE Transactions on Parallel and Distributed Systems **6** (1995) 427–435
5. Li, Y., Peng, S.: Dual-cubes: a new interconnection network for high-performance computer clusters. In: Proceedings of the 2000 International Computer Symposium, Workshop on Computer Architecture, ChiaYi, Taiwan (2000) 51–57
6. Li, Y., Peng, S., Chu, W.: Efficient collective communications in dual-cube. The Journal of Supercomputing **28** (2004) 71–90
7. Preparata, F.P., Vuillemin, J.: The cube-connected cycles: a versatile network for parallel computation. Commun. ACM **24** (1981) 300–309
8. Saad, Y., Schultz, M.H.: Topological properties of hypercubes. IEEE Transactions on Computers **37** (1988) 867–872
9. Chen, G.H., Duh, D.R.: Topological properties, communication, and computation on wk-recursive networks. Networks **24** (1994) 303–317
10. Vicchia, G., Sanges, C.: A recursively scalable network vlsi implementation. Future Generation Computer Systems **4** (1988) 235–243
11. TOP500: Supercomputer Sites. http://top500.org/ (2008)
12. Beckman, P.: 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 (2008)
13. Adiga, N.R., Blumrich, M.A., Chen, D., Coteus, P., Gara, A., Giampapa, M.E., Heidelberger, P., Singh, S., Steinmacher-Burow, B.D., Takken, T., Tsao, M., Vranas, P.: Blue gene/l torus interconnection network. IBM Journal of Research and Development, http://www.research.ibm.com/journal/rd/492/tocpdf.html **49** (2005) 265–276
14. Li, Y., Peng, S., Chu, W.: Recursive dual-net: A new universal network for supercomputers of the next generation. In: Proceedings of the 9th International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP'09), Taipei, Taiwan, Springer, Lecture Notes in Computer Science (LNCS) (2009) 809–820
15. Grama, A., Gupta, A., Karypis, G., Kumar, V.: Introduction to Parallel Computing. Addison-Wesley (2003)
16. Hillis, W.D., Jr, G.L.S.: Data parallel algorithms. Communications of the ACM **29** (1986) 1170–1183