An Efficient List-Ranking Algorithm on a Reconfigurable Mesh with Shift Switching

Young-Hak Kim

Kumoh National Institute of Technology, Gumi, Korea

Summary
The parallel list-ranking is a difficult problem because of its extreme sequential property and irregularity. Given a linked list of n nodes, the proposed algorithm runs in $O(1)$ time on the reconfigurable mesh of size $n \times n$ with shift switching. We improve the time complexity by a factor of $O(logn/loglogn)$ as compared to the previous result using the same architecture. Our result also is improved over a logarithm factor in AT^2 as compared to the previous algorithms using two dimensional processor arrays with reconfigurable buses.

Key words:
List ranking, Reconfigurable mesh/bus, Shift switching

1. Introduction

The list-ranking problem has emerged as one of the fundamental techniques in designing parallel algorithms [1]. The list-ranking problem is defined as follows. We assume that a linked list of n nodes is given as an array *next[]* such that *next(i)* points to the node next to the node i in the list, and *next(i)* is *nil* if the node i is the last node in the list. Let $rank(j)$ $(0 \leq j \leq n-1)$ be the number of nodes that precede a node j in the list. The list-ranking problem then is to find, in parallel, the $rank(j)$ of every node j. For example, $rank(j)$ $(0 \leq j \leq 6)$ is $\{4, 3, 1, 5, 2, 0, 6\}$ when a linked list of seven nodes is given as an array *next[]* = $\{3, 0, 4, 6, 1, 2, nil\}$.

Different architectures equipped with reconfigurable buses have been considered in an attempt to reduce the communication diameter of a conventional mesh of processors. These include RMESH [2], RN [3], PARBS [4], REBSIS [5], CRAP [6], etc, and also reconfigurable processors for general application have been studied and developed [7-9]. These architectures have been used as an attractive and powerful computational model. Efficient parallel algorithms for a variety of problems such as fundamental data movements, sorting, selection, image processing, graph problems, and computational geometry have been proposed using this enhanced capability.

Manuscript received June 5, 2007
Manuscript revised June 25, 2007

The list-ranking problem also has been widely studied on a variety of architectures. The number of nodes in a linked list is noted as n throughout this paper. Horng has shown how to solve the problem in $O(1)$ time on a three dimensional $n \times n \times n$ PARBS [10]. Olariu et al. proposed algorithms that run in $O(1)$ and $O(logn/logm)$ time on a two dimensional $(n^e(n+1)+2) \times 3n$ RMESH, $0<e \leq 1$, and $nm \times n$ RMESH, respectively [11]. Lin et al. proposed an algorithm that runs in $O(logn/loglogn)$ time on an $n \times n$ REBSIS, which has a local cross-bridge switch within each processor [5]. Lee proposed a constant time algorithm on an $n(logn)^e \times n(logn)^e$ RMESH [12]. Hayashi et al. developed an efficient algorithm that runs in $O(log^*n)$ time on an $n \times n$ RMESH [13]. Kim et al. showed that the algorithm of Hayashi et al. can be performed at the same time on a reconfigurable mesh of high dimensions with $O(n^{1+e})$ processors [14].

It is difficult to design an efficient parallel algorithm for the list-ranking problem because of its extreme sequential property and irregularity. Even though processor arrays with reconfigurable buses are very powerful, the list-ranking problem has not yet been solved in constant time on a reconfigurable mesh of size $n \times n$ with the bus of width *logn* bits. In this paper, we consider an $n \times n$ reconfigurable mesh on which the bus is of width *logn* bits and processors have the shift switching capability at the bit level. This computation model is the same as REBSIS and CRAP proposed by Lin et al. [5] and Kao et al. [6] respectively.

This paper presents an efficient parallel algorithm for the list-ranking problem that runs in $O(1)$ time on the reconfigurable mesh of size $n \times n$ with shift switching. In this paper, we improve the time complexity by a factor of $O(logn/loglogn)$ as compared to the REBSIS algorithm of Lin et al. [5] using the same computational model. The proposed algorithm takes $O(n^2)$ in AT^2 which is a complexity measure of VLSI algorithms [15], where A is the area of its layout and T is the execution time of the algorithm. We also improve the VLSI complexity measure over a logarithm factor as compared to the previous algorithms developed on two dimensional processor arrays with reconfigurable buses.

The remainder of this paper is organized as follows. In section 2, we review the architecture of a reconfigurable mesh with shift switching. In section 3, we describe our algorithm in detail, and compare our result with the previous results. Finally, we conclude with some final remarks.

2. The computational model

2.1 The reconfigurable mesh

We first review the basic features of a reconfigurable mesh that is an attractive and powerful computational platform among the architectures with reconfigurable buses. The details on the reconfigurable mesh are shown in [2]. A reconfigurable mesh of size $n \times n$ consists of n^2 identical processors positioned on a rectangular array with n rows and n columns. An example of the reconfigurable mesh with size 4×4 is shown in Figure 1. In the figure, every processor has four ports denoted by N, S, E, and W. $PE(i,j)$ denotes the processor in row i and column j, and $PE(0,0)$ denotes the processor located at the upper leftmost corner. As usual, all the $PE(i,j)$s know their coordinate (i,j) within the mesh. Each processor includes fixed amount storages and can execute basic arithmetic and logic operations.

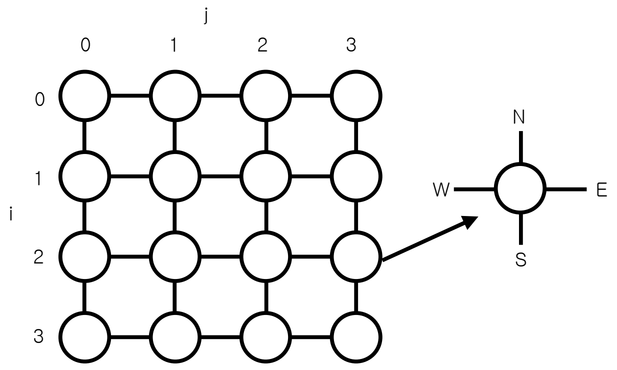

Fig. 1. An example of a reconfigurable mesh

In the architecture, the internal connections among four ports within each processor can be reconfigured during the execution of an algorithm. For example, {WE, NS} represents the configuration in which W port is connected to E port and N port is connected to S port. If every processor establishes its internal connection {WE, N, S}, all the processors in each row are connected by a single bus. Buses are capable of carrying the data of $\Theta(\log n)$ bits long. Each processor communicates with others by broadcasting data on a bus and multiple processors are allowed to broadcast data on the different buses simultaneously at any time if no collision occurs. The communication among processors on the same bus takes $O(1)$ time.

2.2 The reconfigurable mesh with shift switching

Lin et al. [5] introduced the REBSIS(reconfigurable buses with shift switching) architecture that adds shift switching within each processor on a reconfigurable mesh, so that the processors on the same bus can shift data by one bit in left or right. Kao et al. [6] also introduced the CRAP (cross-bridge reconfigurable array of processors) model similar to the REBSIS. The local cross-bridge switch in the CRAP is the same as shift switching. Kao et al. and Lin et al. also referred that the local cross-bridge switch within each processor can be implemented with a little bit of overhead [5, 6].

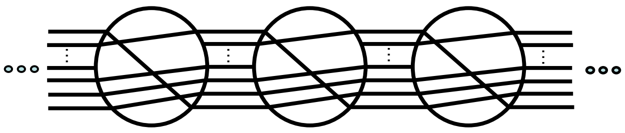

Fig. 2. An example of shift switching

Now, we briefly explain shift switching to be used in our computation. The details can be found in [5, 6]. Let W_i, E_i, S_i, N_i be the i-th bit of ports W, E, S, N within each processor on the mesh, respectively, where $0 \le i \le \log n - 1$. Figure 2 shows an example of the shift switching regarding the configuration {$W_iE_{(i+1) \bullet mod \bullet \log n}$, N, S}. In the figure, the i-th bit of W is connected to the $((i+1) \bullet mod \bullet \log n)$-th bit of E, while N is disconnected to S.

3. The list-ranking algorithm

3.1 The basic operations

This section reviews a couple of the basic results that will be used in designing our list-ranking algorithm. Let a binary sequence be $b_0, b_1, b_2, \cdots, b_{n-1}$, and the prefix sums of a binary sequence be b_0, $b_0 + b_1$, $b_0 + b_1 + b_2$, $\cdots$, $b_0 + b_1 + b_2 + \cdots + b_{n-1}$. The prefix sum problem has been proved to be one of the fundamental techniques in designing parallel algorithm. The following lemma shows a result of the problem on a reconfigurable mesh.

Lemma 1. *The prefix sums of n binary sequences stored in one row or column of a reconfigurable mesh of size $n \times n$ can be computed in O(1) time [16].*

As mentioned in section 1, some parallel algorithms for the list-ranking problem have been proposed on processor arrays with reconfigurable buses. We use the result of Lee [12] as a basic tool, and state as follows.

Lemma 2. *The list-ranking problem with n nodes can be solved in O(1) time on a reconfigurable mesh of size* $n(logn)^e \times n(logn)^e$, $(0<e\leq 1)$ *[12].*

Next, let us consider a list $\{a_0, a_1, a_2, \cdots, a_{n-1}\}$ where a_i is either an integer between *0* and *n-1* or *nil*. Our purpose rearranges a given list so that all the elements except *nil* are moved in the left side, preserving the same order as the list has before. For example, given a list {*nil, 2, nil, 5, 3, nil, 7*} of seven elements, the rearranged list consists of {*2, 5, 3, 7, nil, nil, nil*}. We start by proving this in the following lemma.

Lemma 3. *Given a list of n elements consisting of an integer or nil stored in one row of a reconfigurable mesh of size* $n \times n$*, the problem of rearranging all the elements except nil in consecutive positions can be solved in O(1) time.*

Proof. Initially, it is assumed that each element a_i in the list $\{a_0, a_1, a_2, \cdots, a_{n-1}\}$ is stored in the processor *PE(0,i)* of the first row on the mesh. We copy the values stored in the first row to all rows by using a column bus. Every processor *PE(i,j)* $(0\leq i,j\leq n-1)$ establishes its local connection {*WE, N, S*} if the value copied is *nil*, and {*WN, SE*} otherwise. Now, using the bus configuration, each processor *PE(0,j)* $(0\leq j\leq n-1)$ in the first row broadcasts its value through port *N*, and *PE(i,0)* $(0\leq i\leq n-1)$ in the first column reads the value received from port *W* and stores it itself. At this time, every processor *PE(i,0)* without receiving any value sets its value to *nil*. Then, all the values other than *nil* are moved into consecutive positions from the uppermost processor of the first column. The values in the first column can be easily routed to the first row. It is easy to see that these steps can be performed in *O(1)* time. □

3.2 The algorithm

We first outline the basic ideas of our algorithm that rely on prefix sums, and compute the rank of a single node in the list through the divide and conquer technique. The outline of our algorithm is shown below. We later will show how to implement each step in *O(1)* time on the reconfigurable mesh of size $n \times n$ with shift switching.

Step 1. Divide the given linked list into $\lceil n/logn \rceil$ sublists of length *logn* and compute separately the rank values of all the nodes in each sublist.

Step 2. Construct the new linked list of length $\lceil n/logn \rceil$ consisting of the first node of each sublist. Here, the first node means the node at the leftmost position in each sublist divided by step 1.

Step 3. Compute the rank value of each node in the new linked list and then multiply each rank value by *logn*.

Step 4. Compute the rank values of all the nodes in each sublist by using the rank values of the new linked list.

For clarity, we assume that *logn* is an integer that is divisible by *n*. Step 1 divides the given linked list into $\lceil n/logn \rceil$ sublists, each containing *logn* nodes. Hence, the rank of each node in sublists has a value between *0* and *logn-1*. Step 2 constructs the new linked list of length $\lceil n/logn \rceil$ consisting of the first node of each sublist without violating an order in the given linked list. Step 3 computes the rank values of all the nodes included in the new linked list by using Lemma 2 and then multiplies the rank value of each node by *logn*. Once step 3 is done, the rank values of all the nodes in the linked list can be computed. In step 4, we broadcast the rank value of the first node in each sublist, which is already computed in step 3, to all the nodes within the sublist including itself, and then add this value to the rank value computed in step 1. Finally, the rank value of each node in the linked list exactly equals to the result of step 4.

We now explain how each step of the above algorithm can be implemented in *O(1)* time. Initially, it is assumed that *next(j)* $(0\leq j\leq n-1)$ is stored in the processor *PE(0,j)* of the first row on the reconfigurable mesh of size $n \times n$ with shift switching. Actually, the remaining steps except for step 1 are solved in *O(1)* time on the reconfigurable mesh without shift switching. We omit the details on how to build the bus configurations if they can be obtained by simple port connections.

Lemma 4. *Step 1 in the algorithm can be solved in O(1) time on the reconfigurable mesh of size* $n \times n$ *with shift switching.*

Proof. This step divides the given linked list to $\lceil n/logn \rceil$ sublists of length *logn* and computes separately the rank values of all the nodes within each sublist. We first construct the bus configuration connecting main diagonal processors using *next(j)* held in each *PE(0,j)* $(0\leq j\leq n-1)$. Every diagonal processor *PE(i,i)* should be connected to *PE(j,j)* via *PE(j,i)*, as *next(i)* equals to *j*. In order to establish this bus configuration, *PE(0,j)* $(0\leq j\leq n-1)$ in the first row broadcasts *next(j)* to all the processors in column *j*, and then every processor *PE(i,j)* $(0 \leq i,j \leq n-1)$ marks a '*' if its index *i* equals to *next(j)*. There exists at most one processor having '*' mark on each row because *next(j)* $(0 \leq j \leq n-1)$ has a unique value. We can easily establish the bus configuration as shown in Figure 3 by using '*' mark in each row and column. The figure shows an example of the bus configuration for the given linked list {*3, 0, 4, 6, 1, 2, nil*}, starting at node *5* and ending at node *6*. The processor holding a starting node on the mesh can be determined as *PE(k,k)* without '*' mark in any row *k*, and similarly the processor holding an ending node can be founded as *PE(r,r)* without '*' mark in any column *r*. These steps take *O(1)* time because it requires only simple data movements and port configurations.

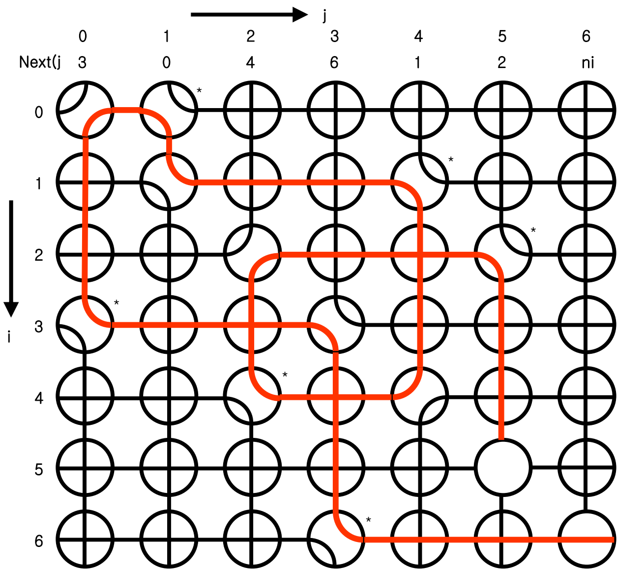

Fig. 3. The bus configuration of a linked list with 7 nodes

Once the bus configuration is established, diagonal processors can be connected as a linear array with *n* processors so that the starting and ending processors are placed on the leftmost and rightmost position, respectively. In order to divide the given linked list into $\lceil n/logn \rceil$ sublists of length *logn* and compute *(rank(j)) mod (logn)* of node *j* within each sublist, all the processors on the linear array establish their port connections as shift switching with left(incoming port) to right(outgoing port) pattern in bit level. Through the above local connection, a signal that is sent from any bit of the starting processor can be passed to the other processors by shifting it cyclically.

Let *mrank(j)* be the rank value of node *j* to be computed separately in each sublist. On the linear array we have just described, the starting processor sends a signal '1' through the left most(first) bit of its left port assuming as if it received the signal from the preceding processor, and then each processor *PE(j,j)* sets *mrank(j)* to the bit position of its left port passing the signal. The *mrank(j)* of each node *j* has an integer between *0* and *logn-1* after completing the steps described above. Thus, the value of *mrank(j)* will result in the same as *(rank(j)) mod (logn)* because all the processors on this array are connected to the bus corresponding to the given linked list and the bus width is of length *logn* bits. Finally, if all the processor *PE(j,j)*s with *mrank(j)* equal to *0* establish its port connection to be disconnected, the bus can be divided into $\lceil n/logn \rceil$ subbuses. In case of an $n \times n$ reconfigurable mesh with shift switching, we can achieve the same result by setting the port connection of the corresponding diagonal processors as {*W, E, N, S*}. These steps clearly take *O(1)* time. □

Lemma 5. *Step 2 in the algorithm can be solved in O(1) time on the reconfigurable mesh of size* $n \times n$.

Proof. This step constructs the new linked list of length $\lceil n/logn \rceil$ consisting of the first node of each sublist. We define $\lceil n/logn \rceil$ subbuses divided in the previous step as $S_0, S_1, S_2, \cdots, S_{\lceil n/logn \rceil -1}$. The new linked list can be composed of the first processors on each subbuses that mean all the *PE(j,j)*s with *mrank(j)* equal to *0*. Let *mnext(j)* be the node following node *j* in the new linked list. The first processor *PE(j,j)* on the subbus S_i broadcasts its index *j* to the processors on the subbus S_{i-1} in parallel, and then the first processor *PE(k,k)* on the subbus S_{i-1} sets *mnext(k)* to the received value. At the same time, the other processors *PE(l,l)*s except the first processor on each subbus set *mnext(l)* to *nil*.

Each *PE(j,j)* $(0 \leq j \leq n-1)$ broadcasts *mrank(j)* and *mnext(j)* to each *PE(0,j)* in the first row, and then each *PE(0,j)* $(0 \leq j \leq n-1)$ in the first row has the values as follows: *next(j), mrank(j), mnext(j)*. Since *mnext(j)* has either an integer between *0* and *n-1* or *nil*, the list of *mnext(j)*s $(0 \leq j \leq n-1)$ in the first row is the same as the list defined in Lemma 3. Also, there will exist at most $\lceil n/logn \rceil$ elements without *nil* on the list. We have to compact the values except *nil* in the first row in order to construct the new linked list consisting of $\lceil n/logn \rceil$ nodes. For compaction, if *mrank(j)* is equal to *0*, each *PE(0,j)* $(0 \leq j \leq n-1)$ in the first row stores 1 in one of its register, and otherwise it stores *0*. After this is done, the prefix sum for a sequence of binary numbers in the first row is computed by applying Lemma 1.

Let *pre(j)* be the sum of -1 and the prefix sum held in each *PE(0,j)* and *mnext_t(j)* be the value that will be used in order to adjust the position of *mnext(j)* according to the result of the prefix sum. Now each processor *PE(0,j)* in the first row whose *mrank(j)* is equal to *0* sets *mnext_t(pre(j))* to the value of *pre(mnext(j))* held in *PE(0,mnext(j))*. Each processor *PE(0,j)* can receive the value of *pre(mnext(j))* held in *PE(0,mnext(j))* via *PE(mnext(j),j)* and P*E(mnext(j), mnext(j))*. No conflicts occur since *mnext(j)* has a unique value, and these steps are performed in parallel in *O(1)* time. Finally, *mnext_t()*s in the first row can be compacted consecutively from left in the first row by using Lemma 3. We also must route *mnext_t(pre(j))* and *j* together as Lemma 3 is applied, and later the index *j* will be used to route the rank value of each node in the new linked list to the original position. Consequently, step 2 takes *O(1)* time. □

Lemma 6. *Step 3 in the algorithm can be solved in O(1) time on the reconfigurable mesh of size n x n.*

Proof. This step computes the rank value of each node in the new linked list, and then multiplies its value by *logn*. The list-ranking problem with *n* nodes can be solved in *O(1)* time on an *nlogn x nlogn* reconfigurable mesh by Lemma 2. After step 2 is completed in the algorithm, the new linked list is represented consecutively from leftmost processor in the first row, and its length has at most $\lceil n/logn \rceil$. Therefore, Lemma 2 can be directly applied on an *n x n* reconfigurable mesh since *(n/logn)log(n/logn) x (n/logn)log(n/logn) < n x n*. Let *lrank(k)* be the rank value of node *k* in the new linked list. Finally, each *PE(0,k)* $(0 \leq k \leq \lceil n/logn \rceil -1)$ in the first row sets the value of *lrank(k) to lrank(k) x logn*. This step takes *O(1)* time by Lemma 2. □

Lemma 7. *Step 4 in the algorithm can be solved in O(1) time on the reconfigurable mesh of size n x n.*

Proof. This step computes the rank values of all the nodes in each sublist by using the rank value in the new linked list. First, each *PE(0,k)* $(0 \leq k \leq n-1)$ in the first row routes the value of *lrank(k)* computed in step 3 to *PE(j,j)*, using the index *j* held in step 2, and then each *PE(j,j)* whose *mrank(j)* is equal to *0* sets *rank(j)* to the received value. Next, the first processor *PE(j,j)*, which has *mrank(j)* equal to *0*, on the subbus S_i broadcasts *rank(j)* to all the processors on the subbus including itself in parallel. Every processor *PE(k,k)* except the first processor on each subbus sets *rank(k)* to the sum of *mrank(k)* and the received value, and then each *PE(j,j)* $(0 \leq j \leq n-1)$ routes *rank(j)* to *PE(0,j)*. Finally, each *PE(0,j)* $(0 \leq j \leq n-1)$ in the first row has *rank(j)* for node *j*. These steps clearly take *O(1)* time. □

As shown in the above lemmas, the main diagonal processors need the shift switching in order to solve step 1 in *O(1)* time, and the remaining steps can be solved on the reconfigurable mesh without shift switching. Hence, we have the following result.

Theorem 8. *The list-ranking problem with n nodes can be solved in O(1) time on the reconfigurable mesh of size n x n with shift switching.*

Proof. It is obvious that this problem can be solved in *O(1)* time using Lemma 4, 5, 6, 7. □

3.3 The comparison

Now let us compare our result to the previous ones. First, the time complexity is improved by a factor of *O(logn/loglogn)* when we compare our result with the one of Lin et al. [5] used the same computation model.

Table 1. The comparison in the VLSI complexity measure

Model	Size / Time	AT^2
REBSIS [5]	*n x n* *O(logn/loglogn)*	$O(n^2(logn/logn)^2)$
RMESH [11]	n^{1+e} *x n* *O(1)*	$O(n^{2+e})$
RMESH [12]	$n(logn)^e$ *x* $n(logn)^e$ *O(1)*	$O(n^2(logn)^{2e})$
RMESH [13]	*n x n* $O(log^*n)$	$O(n^2(log^*n)^2)$
Ours	*n x n* *O(1)*	$O(n^2)$

Second, we consider AT^2, a complexity measure of VLSI algorithms [15], where *A* is the area of its layout and *T* is the execution time of the algorithm. We also consider only the results using two dimensional processor arrays with reconfigurable bus because the VLSI complexity in higher dimensions is increased. The result of comparison is summarized in Table 1, and it is assumed that $0<e \leq 1$. Our result takes $O(n^2)$ in AT^2. However, the results of Lee [12] and Hayashi et al. [13] have $O(n^2(logn)^{2e})$ and $O(n^2(log^*n)^2)$ in AT^2, respectively. Therefore, our result is improved by factors of $O((logn)^{2e})$ and $O((log^*n)^2)$ in AT^2, respectively.

4. Conclusions

The list-ranking is one of the fundamental techniques in designing parallel algorithms. Efficient parallel algorithms for the list-ranking problem have been proposed using processor arrays with reconfigurable buses. However, to the best of our knowledge, the list-ranking problem has not yet been solved in constant time on a reconfigurable mesh with two dimensions. In this paper, we proposed an efficient algorithm for this problem that runs in constant time on a reconfigurable mesh with shift switching in two dimensions. The time complexity of our algorithm has been improved as compared to the previous result using the same architecture. We also compared our result with the previous algorithms using two dimensional processor arrays with reconfigurable buses in the area of the VLSI complexity.

Acknowledgements

The work was partly supported by the Kumoh National Institute of Technology in 2006.

References

[1] J. JaJa, An introduction to parallel algorithms, Addison-Wesley Publishing Company, 1992.

[2] R. Miller, V. K. Prasanna-Kumar, D. Reisis, and Q. F. Stout, “Parallel computations on reconfigurable meshes,” IEEE Trans. Comput., vol. 42, no. 6, pp. 678-692, Jun. 1994.

[3] Y. Ben-Asher, D. Peleg, R. Ramaswami, and A. Schuster, “The power of reconfiguration,” J. Parallel Distribut. Comput., vol. 13, no. 2, pp. 139-153, 1991.

[4] B. F. Wang and G. H. Chen, “Constant time algorithms for the transitive closure and some related graph problems on processor array with reconfigurable bus systems,” IEEE Trans. Parallel Distribut. Systems, vol. 1, no. 4, pp. 500-507, Oct. 1990.

[5] R. Lin and S. Olariu, “Reconfigurable buses with shift switching: concepts and applications,” IEEE Trans. Parallel Distribut. Systems, vol. 6, no. 1, pp. 93-104, Jan. 1995.

[6] T. W. Kao, S. J. Horng, Y. L. Wang, and H. R. Tsai, “Designing efficient parallel algorithms on a CRAP,” IEEE Trans. Parallel Distribut. Systems, vol. 6, no. 5, pp. 554-560, May. 1995.

[7] A. Niyonkuru and H. Zeidler, “Designing a runtime reconfigurable processor for general purpose applications,” 18th International Parallel and Distributed Processing Symposium, pp. 143b, 2004.

[8] M. Okada, T. Hiramatsu, H. Nakajima, M. Ozone, K. Hirase, and S. Kimura, “A reconfigurable processor based on ALU array architecture with limitation on the interconnection,” 19th International Parallel and Distributed Processing Symposium, pp. 152a, 2005.

[9] B. Veale, J. Antonio, and M. Tull, “Configuration steering for a reconfigurable superscalar processor,” 19th International Parallel and Distributed Processing Symposium, pp. 152b, 2006.

[10] S. J. Horng, “Constant time algorithms for summation and graph problems on a reconfigurable array of processors,” Technical Report, ATT Bell Lab., 1991.

[11] S. Olariu, J. L. Schwing, and J. Zhang, “Fundamental algorithms on reconfigurable meshes,” Proc. 29th Annual Allerton Conf. on Comm., Control and Comput., pp. 811-820, 1991.

[12] K. U. Lee, A study on constant time algorithms for evaluating set manipulation operation sequences and list-ranking on a reconfigurable mesh, Ph.D dissertation, Sogang University, 1997.

[13] T. Hayashi, K. Nakano, and S. Olariu, “Efficient list ranking on the reconfigurable mesh with applications,” Proc. 7th International Symposium on Algorithms and Computation, pp. 326-335, Dec. 1996.

[14] S. Kim and K. Park, “Efficient list ranking algorithms on reconfigurable mesh,” 6th International Computing and Combinatorics Conference, Lecture Notes in Computer Science Vol.1858, Springer, pp. 262-271, 2000.

[15] J. D. Ullman, Computational aspects of VLSI, Computer Science Press, 1984.

[16] S. Olariu, J. L. Schwing, and J. Zhang, “Applications of reconfigurable meshes to constant-time computations,” Parallel Computing 19, pp. 229-237, 1993.

Young-Hak Kim received the M.S. and Ph.D. degrees in Computer Engineering from Sogang University in 1989 and 1997, respectively. He is currently an associate professor in the school of computer and software engineering at Kumoh National Institute of Technology, Gumi, Korea. He was a visiting scholar in the school of electrical and computer engineering at Georgia Institute of Technology. His research interests include parallel algorithm, parallel processing, and embedded system.