再構成メッシュ上の加算アルゴリズム

1 P − 5

中野浩嗣 [†] 和田幸一 [‡]
（株）日立製作所基礎研究所 名古屋工業大学電気情報工学科

1 Introduction

An $n \times m$ *reconfigurable mesh* ($n \times m$-RM for short) is a processor array that consists of nm processors arranged in a 2-dimensional grid of size $n \times m$. Any two adjacent processors are connected with a link, and the internal connections between the four ports of each processor can be configured locally during execution of the algorithm. The connected components formed by links and internal connections form *subbuses*. The processors can communicate through subbuses. We assume that all broadcasts through a subbus are completed in a unit of time, and a simultaneous broadcast to the same subbus is allowed only if the sending values are the same. Recently, reconfigurable meshes have attract considerable attention.

This paper deals with the problem to compute the sum of n binary values and shows an efficient algorithm on the reconfigurable meshes. This problem is very important because it is a fundamental procedure that is used as a subroutine in lots of algorithms, for example, sorting, arithmetic operations, and so on. Several algorithms for summing n binary values have been obtained. Olariu *et al.* [5] showed an $O(\log n/\log m)$ time algorithm on an $n \times m$-RM for all m $(1 \leq m \leq n)$. One of the authors improved this result and showed an $O(\log n/\sqrt{m \log m})$ time algorithm on an $n \times m$-RM for all m $(1 \leq m \leq \log^2 n/\log\log n)$ [4]. Chen *et al.* [1] showed an $O(\log\log n)$ time algorithm on a $\sqrt{n} \times \sqrt{n}$-RM. Jang *et al.* [2] showed an $O(\log^* n)$ time algorithm on a $\sqrt{n} \times \sqrt{n}$-RM. Jang *et al.* [3] also showed a constant time algorithm on a $\sqrt{n}\log n \times \sqrt{n}\log n$-RM.

We will show that for n binary values given to processors on every $\sqrt{m}$ rows, one binary value for each processor, the sum of them can be computed in $O(\log^* n - \log^* m)$ $(1 \leq m \leq \log n)$ time on a $\sqrt{nm} \times \sqrt{n}$ RM. From this algorithm, a $\sqrt{n}\log^{(O(1))} n \times \sqrt{n}$ RM is sufficient for constant time summing, and if the number of processors is the same as that of the input bits, the sum can be computed in $O(\log^* n)$ time. Therefore, our algorithm is an improvement of the previously known algorithms above.

A Summing Algorithm on a Reconfigurable Mesh

[†]Koji Nakano, Advanced Research Lab., Hitachi Ltd., Hatoyama, Saitama 350-03, Japan

[‡]Koichi Wada, Department of Electrical and Computer Engineering, Nagoya Institute of Technology, Showa-ku, Nagoya 466, Japan

2 Basic summing algorithm

This section reviews a basic summing algorithm for binary values [4], which is used by the summing algorithm in this paper.

For the first q prime numbers, $p_1(=2), p_2(=3), \ldots, p_q$, let $p_q^{\oplus} = p_1 + p_2 + \cdots + p_q$ and $p_q^{\otimes} = p_1 \cdot p_2 \cdots p_q$. For given n binary values $a_0, a_1, \ldots, a_{n-1}$, let the remainder of them be $x \bmod p_q^{\otimes}$, where $x = a_0 + a_1 + \cdots + a_{n-1}$. The remainder can be computed in $O(1)$ time on a $2n \times (p_q^{\oplus} + q)$-RM as follows.

Imagine that the RM is partitioned into q subRMs of sizes $2n \times (p_1 + 1)$, $2n \times (p_2 + 1)$, $\ldots$, $2n \times (p_q + 1)$. In each ith subRM, the value of $x \bmod p_i$ is computed using the prefix remainder algorithm [4] and it is sent to the same subRM. In each $2j$th column of ith subRM, it is checked whether $j = x \bmod p_i$. Then, in each $2j$th column, it is checked whether $j \bmod p_i = x \bmod p_i$ holds for all i. After that, the minimum j such that $j = x \bmod p_i$ for all i, is computed. For such j, $j = x \bmod p_q^{\otimes}$ holds. Therefore, the value of $x \bmod p_q^{\otimes}$ can be computed in $O(1)$ time.

Using the remainder algorithm above, we can compute the sum efficiently. Let b_i be the integer such that $b_i = 1$ if the following two conditions hold: *i)* $x \bmod p_j = 0$, for all j, i.e., $x \bmod p_q^{\otimes} = 0$, and *ii)* $a_i = 1$, otherwise, $b_i = 0$. Then, $x = (x \bmod p_q^{\otimes}) + (b_0 + b_1 + \cdots + b_{n-1})p_q^{\otimes}$. Therefore, by recursively computing the sum of $b_0, b_1, \cdots, b_{n-1}$, the value of x is computed: after computing the sum of $b_0, b_1, \ldots, b_{n-1}$, the multiplication by $p_q^{\otimes}$ is computed in constant time on a $\log n \times \log^2 n$-RM because this can be done by the multiplication of two $\log n$-bit integers [2]. The addition to $x \bmod p_q^{\otimes}$ can be done in constant time on a $\log n \times 1$-RM. Hence, each recursion can be completed in constant time. Therefore, the depth of the recursion, $O(\log n/\log p_q^{\otimes})$ corresponds to the computing time.

3 Summing algorithm

We will show a more efficient summing algorithm based on the basic summing algorithm.

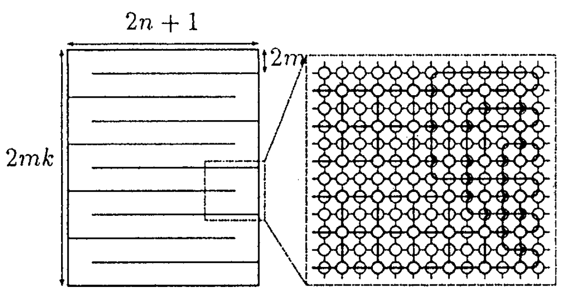

Figure 1: Snake-like embedding

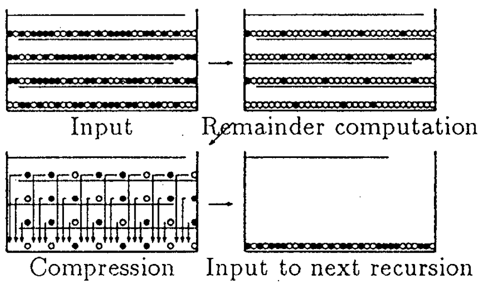

Figure 2: Summing algorithm

Figure 1 illustrates the *snake-like embedding* of an $nk \times m$-RM in a $(2n+1) \times 2mk$-RM. In the snake-like embedding, a processor on the $nk \times m$-RM corresponds to a processor whose row is even and column is odd on the $(2n+1) \times 2mk$-RM. In the figure, these processors are represented by thick circles, and the connections for the embedding are also represented by thick lines. The $nk \times m$-RM is bent $k-1$ times, and is partitioned into k segments, each of which corresponds to consecutive $2m$ rows in the $(2n+1) \times 2mk$-RM.

Suppose that the remainder of nk $(k \leq n)$ binary values $a_0, a_1, \ldots, a_{nk-1}$ is computed on the snake-like embedding using the algorithm in the previous section. The nk binary values is given to the RM such that n binary values are given to each segment, one binary value for every two column (Fig. 1). That is, the input is given to every $2m$ rows on the $(2n+1) \times 2mk$-RM. Let q be the maximum integer such that $p_q^{\oplus} + q \leq m$. Using the remainder algorithm, it is easy to compute $x \bmod p_q^{\otimes}$, and $b_0, b_1, \ldots b_{nk-1}$, where $x = a_0 + a_1 + \cdots + a_{nk-1}$, and each b_i is defined in the same way as the algorithm in the previous section. If the sum of $b_0, b_1, \ldots, b_{nk-1}$ is computed recursively, the sum x can be computed in $O(\log(nk)/\log p_q^{\otimes}) = O(\log n/\sqrt{m \log m})$ time as shown in the previous section. However, we can reduce the computing time by compressing $b_0, b_1, \ldots, b_{nk-1}$.

Let $c_j = b_{j \cdot p_q^{\otimes}} + b_{j \cdot p_q^{\otimes}+1} + \cdots + b_{(j+1) \cdot p_q^{\otimes}-1}$ for $0 \leq j \leq nk/p_q^{\otimes} - 1$. Since at most one of $b_{j \cdot p_q^{\otimes}}, b_{j \cdot p_q^{\otimes}+1}, \ldots, b_{(j+1) \cdot p_q^{\otimes}-1}$ is 1 for each j, the value of each c_j is either 0 or 1, and its value can be determined in constant time by the simultaneous sending. See Fig. 2 for an example in which a black circle denotes 1 and a white circle 0, where $n = 32$, and $p_q^{\otimes} = 4$. Although 4 is an impossible value for $p_q^{\otimes}$, the figure shows the essence of the algorithm. For c_j's thus obtained, $x = (x \bmod p_q^{\otimes}) + (c_0 + c_1 + \cdots + c_{nk/p_q^{\otimes}-1}) p_q^{\otimes}$ holds, because each c_j with value 1 corresponds to $p_q^{\otimes}$ 1's in the input. Therefore, the value of x can also be obtained by the recursive computation of the sum of $c_0, c_1, \cdots, c_{nk/p_q^{\otimes}-1}$. In order to reduce the depth of the recursion, for each j, $c_{jn}, \ldots, c_{(j+1)n-1}$ are transferred to a row, one for each processor. For example, $c_0, c_1, \cdots, c_{n-1}$ is transferred to a row as shown in Fig. 2. After that, all of the c's are distributed to every $2mp_q^{\otimes}$ rows and the sum of $c_0, c_1, \ldots, c_{nk/p_q^{\otimes}-1}$ is computed recursively. Therefore, in the following recursion, the sum of $nk/p_q^{\otimes}$ binary values is computed based on the embedding of an $nk/p_q^{\otimes} \times mp_q^{\otimes}$-RM in the $(2n+1) \times 2mk$-RM. Using this technique, the depth of the recursion can be reduced to $O(\log^* n - \log^* m)$. Replacing n, m, and k by $\sqrt{n}$, $\sqrt{m}$ and $\sqrt{n}$, we have

Theorem 1 *For n binary values given to every $\sqrt{m}$ rows of a $\sqrt{nm} \times \sqrt{n}$-RM, the sum of them can be computed in $O(\log^* n - \log^* m)$ $(1 \leq m \leq \log n)$ time.*

References

[1] Y. -C. Chen and W. -T. Chen. Reconfigurable mesh algorithms for summing up binary values and its applications. In *Proceedings of 4th Symposium on Frontiers of Massively Parallel Computation*, pages 427–433. IEEE, October 1992.

[2] J. -W. Jang, H. Park, and V. K. Prasanna. A fast algorithm for computing histograms on a reconfigurable mesh. In *Proceedings of 4th Symposium on Frontiers of Massively Parallel Computation*, pages 244–251. IEEE, October 1992.

[3] J. -W. Jang, H. Park, and V. K. Prasanna. A bit model of reconfigurable mesh. In *Proceedings of Reconfigurable Architecture Workshop*. IEEE, April 1994.

[4] K. Nakano. An efficient algorithm for summing up binary values on a reconfigurable mesh. *IEICE Transactions (Japan)*, E77-A(4):652–657, April 1994.

[5] S. Olariu, J. L. Schwing, and J. Zhang. Fundamental algorithms on reconfigurable meshes. In *Proceedings of 29th Allerton Conference on Communications, Control, and Computing*, pages 811–820, 1991.