Integer summing algorithms on reconfigurable meshes

Koji Nakano*, Koichi Wada

* Advanced Research Lab., Hitachi Ltd, Hatoyama, Saitama 350-03, Japan
b Department of Electrical and Computer Engineering, Nagoya Institute of Technology, Showa-ku, Nagoya 466, Japan

Received December 1994; revised October 1996
Communicated by S. Miyano

Abstract

This paper presents the following algorithms to compute the sum of \( n \) \( d \)-bit integers on reconfigurable parallel computation models: (1) a constant-time algorithm on a reconfigurable mesh of the bit model of size \( \sqrt{n \log^{O(1)} n} \times d \sqrt{n} \), (2) an \( O(\log^* n) \)-time algorithm on a reconfigurable mesh of the bit model of size \( \sqrt{n/\log^* n} \times d \sqrt{n/\log^* n} \), (3) an \( O(\log d + \log^* n) \)-time algorithm on a reconfigurable mesh of the word model of size \( \sqrt{n/(\log d + \log^* n)} \times \sqrt{n/(\log d + \log^* n)} \), and (4) an \( O(\log^* n) \)-time algorithm on a VLSI reconfigurable circuit of area \( O(dn/\log^* n) \).

1. Introduction

A reconfigurable mesh (RM) is a processor array that consists of processors arranged in a two-dimensional grid and a dynamically reconfigurable bus system (Fig. 1). There is a link between the ports of each two adjacent processors and the four ports of each processor can be connected or disconnected locally during execution of an algorithm. Each connected component formed by links and internal connections constitutes a subbus. On a RM of the word model, each processor can execute one of the word operations and a word of data can be transferred through a subbus in one unit of time. On the bit model, a bit operation and the transfer of a bit of data requires one unit of time. RMs have recently attracted considerable attention as theoretical models of parallel computation, and many studies have been devoted to developing efficient parallel algorithms on RMs. For example, they efficiently solve problems such as sorting [25, 31], selection [8, 10], arithmetic operations [2, 18, 6], graph problems [20, 15, 33], geometric problems [24, 28], image processing [1, 16, 17, 3].

* Corresponding author. Present address: Department of Electrical and Computer Engineering, Nagoya Institute of Technology, Showa-ku, Nagoya 466, Japan. E-mail: nakano@elcom.nitech.ac.jp.
This paper deals with the problems of computing the sum of \( n \) binary values and the sum of \( n \) \( d \)-bit integers and it describes efficient algorithms on RMs. These problems are quite important because they are fundamental procedures used as subroutines in many algorithms, sorting, selection, geometric algorithms, graph algorithms, arithmetic operations, matrix computations, and so on. Furthermore, these summing problems have no highly parallelized algorithms on traditional parallel machine models without a dynamically reconfigurable bus system: even a PRAM requires \( \Omega(\log n/\log \log n) \) time to solve these problems [4]. It is, therefore, interesting to find highly parallelized algorithms for these problems on RMs.

Tables 1 and 2 list relevant results known previously as well as results presented in this paper. We will first show that for \( n \) binary values given to processors on every \( \sqrt{m} \) rows, their sum can be computed in \( O(\log^* n - \log^* m) \) \((1 \leq m \leq \log n)\) time on a \( \sqrt{nm} \times \sqrt{n} \) RM of the bit model, where \( \log^{(k+1)} n = \log(\log^{(k)} n) \) for all \( k \), \( \log^{(1)} n = \log n \), and \( \log^* n \) is the minimum integer \( k \) such that \( \log^{(k)} n \leq 1 \). The key idea of the algorithm for this is implementation of Nakano's summing algorithm [22] in the \( \sqrt{nm} \times \sqrt{n} \) RM. From this algorithm, we can get an \( O(\log^* n) \)-time algorithm on a \( \sqrt{n} \times \sqrt{n} \) RM and an \( O(1) \)-time algorithm on a \( \sqrt{n} \log^{(O(1))} n \times \sqrt{n} \) RM. These algorithms are improvements of the Jang's algorithms [11, 13] that have been the best known algorithms, in the sense that our algorithms use fewer processors.

We will also present an algorithm that computes the sum of \( n \) \( d \)-bit integers in \( O(\log^* n - \log^* m) \) \((1 \leq m \leq \log n)\) time on a \( \sqrt{nm} \times d \sqrt{n} \) RM of the bit model. This
Table 2
Algorithms for summing $n$ $d$-bit integers

<table>
<thead>
<tr>
<th>Model</th>
<th>Size of RM</th>
<th>Computing time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wang et al. [6]</td>
<td>bit $dn \times dn$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Ben-Asher et al. [5]</td>
<td>bit $n \times n \times n$ for $n$ $n$-bit integers</td>
<td>$O(\log^* n)$</td>
</tr>
<tr>
<td>Jang et al. [14]</td>
<td>bit $n \times dn$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Jang et al. [13]</td>
<td>bit $\min(dn \times n, dn \times d \log^2 n)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Olariu et al. [27]</td>
<td>word $n \times n$ for $n \log n$-bit integers</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Chen et al. [7]</td>
<td>word $\sqrt{n} \times \sqrt{n}$</td>
<td>$O(d \log \log n)$</td>
</tr>
<tr>
<td>Fragopoulou [9]</td>
<td>word $\sqrt{nm} \times d \sqrt{n}$ (1 $\leq m \leq \log n)$</td>
<td>$O(d \log \log n - \log^* m)$</td>
</tr>
<tr>
<td>This paper</td>
<td>bit $\sqrt{n} \log^*(O(1)) n \times d \sqrt{n}$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>This paper</td>
<td>bit $\sqrt{n} / \log^<em>(n) \times d \sqrt{n / \log^</em> n}$</td>
<td>$O(\log^* n)$</td>
</tr>
<tr>
<td>This paper</td>
<td>word $\sqrt{n / (\log d + \log^* n)} \times \sqrt{n / (\log d + \log^* n)}$</td>
<td>$O(\log d + \log^* n)$</td>
</tr>
</tbody>
</table>

This paper is based on the two-stage summing method (Fig. 7), in which the sum of each digit of given integers are computed and then the sum of these sums is computed. In this algorithm, the sum of each digit is computed by the $O(\log^* n - \log^* m)$-time algorithm for binary values above, and the sum of these sums is computed by using Jang’s algorithm [14]. Thus, a $\sqrt{n} \log^*(O(1)) n \times d \sqrt{n}$ RM is sufficient for constant-time summing, and if the number of processors is the same as the number of input bits, the sum can be computed in $O(\log^* n)$ time. We will then consider the case where $n$ $d$-bit integers are given to a $\sqrt{n} \times \sqrt{n}$ RM of the word model and present an $O(d \log \log n)$-time algorithm. We will also show that the size of the RM can be reduced to $\sqrt{n / (\log d + \log^* n)} \times \sqrt{n / (\log d + \log^* n)}$ without asymptotically increasing the computing time. As a previously known best algorithm, Fragopoulou [9] presented an $O(d \log \log n)$-time algorithm. Thus our algorithm is an improvement of this algorithm.

This paper also deals with a VLSI reconfigurable circuit (VLSI RC) under the assumption that a switch with two terminals and a single control terminal is available. By the signal sent to the control terminal, the switch connects or disconnects the two terminals. Other assumptions of this treatment of the VLSI RC are the same as those of the usual VLSI model [30]. Under the usual VLSI model, Muller et al. [21] presented a VLSI circuit with $O(n)$ gates and $O(\log n)$ depth to compute the sum of $n$ binary values. Based on this circuit, Wada et al. [32] showed a VLSI circuit of area $O(n / \log n)$ to compute the sum of $n$ binary values in $O(\log n)$ time. This VLSI algorithm attains a trivial $AT$ lower bound $\Omega(n)$. For summing $n$ $d$-bit integers, Wada et al. [32] also presented a VLSI circuit constructed with area $O(dn + (d \log n)^2)$ and time $O(\log n + \log d)$. In this paper, we will show that the sum of $n$ binary values can be computed in $O(\log^* n)$ time using a VLSI RC of area $O(n / \log^* n)$ and that the sum of $n$ $d$-bit integers can be computed in $O(\log^* n)$ time using a VLSI RC of area $O(dn / \log^* n)$. These VLSI RCs are optimal for trivial $AT$ lower bounds $\Omega(n)$ and $\Omega(dn)$, respectively. Furthermore, simulation of the VLSI RCs by a RM of the bit model
enables us to reduce the size of RM for summing and get the following algorithms: an $O(\log^* n)$-time algorithm for summing $n$ binary values on a $\sqrt{n/\log^* n} \times \sqrt{n/\log^* n}$ RM of the bit model, and an $O(\log^* n)$-time algorithm for summing $n$ $d$-bit integers on a $\sqrt{n/\log^* n} \times d \sqrt{n/\log^* n}$ RM of the bit model. These algorithms are cost optimal in the sense that the product of the time and the number of processors is equal to the number of bits of the input.

This paper is organized as follows. Section 2 defines RMs formally, and Section 3 briefly explains the basic algorithms and techniques used in this paper. Section 4 presents a summing algorithm for the bit model, Section 5 presents a summing algorithm for the word model, and Section 6 presents an $AT^*$ optimal VLSI RC for the summing problems.

2. Reconfigurable mesh

This section defines a RM. A reconfigurable mesh (RM) consists of processors arranged in a grid. An $n$ RM is a RM with $n$ processors $PE(0), PE(1), \ldots, PE(n-1)$, in which for each $i$, $PE(i)$ and $PE(i+1)$ are connected with a link (Fig. 1). An $n \times m$ RM is a RM in which $n \times m$ processors are arranged in a two-dimensional grid in which any two adjacent processors are connected with a link (Fig. 1).

The control mechanism of the RM is based on the SIMD principle. A single control unit dispatches instructions to each processor. Although all processors execute the same instructions, their behaviors may differ, because they work on different input and different coordinates. Each processor has locally controllable switches that can configure the connection patterns of its four ports denoted by $N(North)$, $E(East)$, $W(West)$, and $S(South)$. The computing power of RMs depends on the connection patterns that are allowed. For example, if the cross-over pattern (i.e. connection of $N$ and $S$, and that of $E$ and $W$ are configured independently), is not allowed, a $\sqrt{n} \times \sqrt{n}$ RM requires $\Omega(\log^* n)$ time to compute the parity of $n$ binary values, whereas if all patterns are allowed it can compute the parity in constant time [19]. If the branch patterns, (i.e. three or four ports are connected internally) is allowed, the connected components of an $n$-node graph can be labeled in constant time on an $n \times n$ RM [34], whereas we have not found an algorithm that can do this labeling in constant time when branch patterns are not allowed. In this paper, we assume that processors in a RM can configure any pattern except branch patterns.

The connected components formed by links and internal connections constitute subbuses, through which the processors can communicate. We assume that any broadcast through subbus takes a unit of time, and for each subbus only one processor can send a piece of data to it. We deal with two models of RMs, the bit model and the word model. On the RM of the bit model, each processor has a constant storage size and the bandwidth of each subbus is 1; that is, either 0 or 1 can be transferred through the subbus. On the RM of the word model, each processor has an arbitrarily large
storage size, can perform basic arithmetic operations (addition, multiplication, division, log, square root and so on), and can transfer arbitrarily large value through a subbus in a unit of time.

A VLSI recongurable circuit (VLSI RC), which is an extension of the usual VLSI model [30], has a switch device with two terminals and one control terminal. By the signal sent to the control terminal, the switch connects or disconnects the two terminals. If two terminals are connected, a signal sent to a terminal passes to the other terminal without delay. A 4-terminal switch that can configure every internal connection patterns of a processor of the bit-model RM can be constructed by six switches as shown in Fig. 2. We assume that a switch occupies constant area, hence a 4-terminal switch also occupies constant area. We will use 4-terminal switches to implement an algorithm executed on RM s in a VLSI RC.

Since we are interested in the computational power of reconfiguration in terms of the capacity for summing, we assume that the computing time of an algorithm excludes the time necessary to give an input data to each processor.

To represent an integer $x$ on a RM, we use the following formats:

VALUE This format is available only to the word model. A processor knows the value of $x$. For example, the value of $x$ is stored to the local storage of PE(0).

UNARY A processor PE($x$) knows that the integer is equal to its own index, and the other processors know that the integer is not equal to their own index.

BINARY Let $x_n-1x_{n-2} \cdots x_0$ be the binary representation of $x$. Each PE($i$) ($0 \leq i \leq n - 1$) knows the value of $x_i$.

The conversions between these formats will be shown in the following section.

3. Basic algorithms and techniques

This section briefly explains the basic algorithms and techniques used in this paper.
3.1. Basic algorithms on one-dimensional RM

First, we will note basic algorithms on a one-dimensional RM.

Lemma 3.1 (Nakano, Masuzawa and Tokura [23]). For \( n \) binary values given to an \( n \) RM, the rightmost element whose value is \( 1 \) can be determined in \( O(1) \).

The idea of the proof is as follows. Each processor connects its two ports if the input is \( 0 \), otherwise disconnects them. By using the subbuses thus configured, the rightmost elements can be determined. Similarly, for given \( n \) Boolean values, their logical OR can be determined in \( O(1) \) time.

By transferring carries through the subbus, we have

Lemma 3.2 (Thangavel and Muthuswamy [29]). The sum of two \( n \)-bit integers can be computed in constant time on an \( n \) RM of the bit model.

By applying the divide-and-conquer technique, we have the following lemma.

Lemma 3.3 (Nakano [22]). The sum of \( n \) integers given to an \( n \) RM of the word model can be computed in \( O(\log n) \) time.

3.2. Integer summing by using lookahead carry generators

This subsection briefly explains an integer summing algorithm by Jang et al. [11, 14]. The algorithm computes the sum of \( n \) \( d \)-bit integers in \( O(1) \) time on a \( 2n \times 2dn \) RM of the bit model. See [14] for details.

Fig. 3 shows a lookahead carry generator that sums four binary values and a carry in the right part divides the result by two in the left part. The carry enters from the rightmost column and the sum is represented by the position of the topmost processor that the signal goes through. In Fig. 3, the carry is 3 and \( 0 + 1 + 0 + 1 \) is added to it. Then, the position of the topmost processor that the signal goes through in the leftmost column corresponds to the carry to the next digit. For example, in the figure, the carry to the next digit is \( [(3 + 0 + 1 + 0 + 1)/2] = 2 \).

By using \( n \) lookahead carry generators as shown in the figure, we can compute the sum of \( n \) \( d \)-bit integers in constant time on the \( 2n \times 2dn \) RM. Furthermore, since the product of two \( n \)-bit integers corresponds to the sum of \( n \) \((2n - 1)\)-bit integers, it can be computed in constant time on a \( 2n \times 4n^2 \) RM of the bit model. Consequently, we have

Lemma 3.4. The sum of \( n \) \( d \)-bit integers can be computed in constant time on a \( 2n \times 2dn \) RM of the bit model, and the product of two \( n \)-bit integers can be computed in constant time on a \( 2n \times 4n^2 \) RM of the bit model.

A more efficient multiplication algorithm that computes the product of two \( n \)-bit integers in constant time on an \( n \times n \) RM has been developed [12], but although our
summing algorithms use the multiplication algorithm as a subroutine, they do not require such high performance.

3.3. Conversions between VALUE, UNARY, and BINARY formats

We will show how to convert an integer $x$ ($0 \leq x \leq n - 1$) represented in one of the three formats to the other formats in constant time, if sufficiently many processors are available. We will start with the conversions on the bit model, where it is easy to convert from the UNARY format to the BINARY format. Assume that PE($x, 0$) knows that $x$ is equal to its own index. PE($x, 0$) sends 1 to the processors in the same row, and every processor tries to receive it. Each PE($x, i$) that succeeds in receiving 1 learns that the $i$th digit of the binary representation of $x$ is $x_i$. So PE($x, i$) sends $x_i$ to PE($i, 0$). Note that for each PE($x, i$) the value of $x_i$ is a constant that does not depend on the input. Therefore, the UNARY format can be converted to the BINARY format in constant time on an $n \times \log n$ RM of the bit model.

Conversely, the conversion from the BINARY format to the UNARY format can also be done in constant time on an $n \times n$ RM of the bit model. Assume that the BINARY of $x$ is given to the top row. Each PE($0, i$) ($0 \leq i \leq \log n - 1$) first broadcasts $x_i$ to $2^i$ processors PE($2^i, 0$), PE($2^i + 1, 0$), ..., PE($2^{i+1} - 1, 0$), and each of them makes a copy of $x_i$ in its local storage. Hence, $2^i$ copies of each $x_i$ are made in the leftmost column. By using the right part of the lookahead carry generator for Lemma 3.4, the UNARY of the number of copies can be computed in constant time, which corresponds to the UNARY of $x$. 

Fig. 3. Integer summing using lookahead carry generators.
Now consider the conversions on the word model. The conversions between the UNARY and BINARY formats obviously can be done in the same way as the bit model, so we will consider the conversions to and from the VALUE. It is quite easy to convert from the VALUE format to the UNARY and BINARY formats: PE(0, 0) broadcast $x$ to all processors, and each PE($i$, 0) can determine the corresponding digit of the UNARY or BINARY of $x$ by local computation in constant time. The conversion from the UNARY format to the VALUE format is as follows: PE($x$, 0) sends $x$ to PE(0, 0), and PE(0, 0) can then get the VALUE of $x$. The conversion from the BINARY format to the VALUE format can be done by two conversions: after the BINARY format is converted to the UNARY format by the algorithm on the bit model, the UNARY format is converted to the VALUE format by the method just described. The conversion from the BINARY format to the VALUE format can thus be done in constant time on an $n \times n$ RM of the word model. By repeating this procedure, the $O(\log n)$-bit integer represented in the BINARY format, whose value is at most $n^{O(1)}$, can also be converted in constant time on an $n \times n$ RM.

3.4. Summing binary values using prefix remainder computation

This subsection describes an algorithm to compute the sum of binary values using prefix-remainders computation. The algorithm computes the sum of $n$ binary values given to an $m \times n$ RM of the bit model in $O(\log n/\sqrt{m})$. See [22] for details.

We will start with the prefix-remainders computation. The prefix $w$-remainders are $\langle x'_0, x'_1, \ldots, x'_{n-1} \rangle$ where $x'_i = (a_0 + a_1 + \cdots + a_i) \mod w$ for given $n$ binary values $A = \langle a_0, a_1, \ldots, a_{n-1} \rangle$. Each $a_i$ ($0 \leq i \leq n-1$) is given to the $2i$th column. After executing the algorithm, each $2i$th column knows the UNARY of $x'_i$. In the algorithm each column functions as a cyclic shift register of size $w$. In the rightmost column, only the bottom element of the cyclic shift register is 1. In each column, if the element given to the column is 1, then the processors in the column shift the cyclic shift register, otherwise, they hold it. Then each prefix-remainder is equal to the position where 1 is found on the cyclic shift register. See Fig. 4 for an example of the bus configuration. Hence, the UNARY of prefix $w$-remainders can be computed in constant time on a $(w + 1) \times 2n$ RM.

![Fig. 4. Prefix 4-remainders.](image-url)
We can compute the larger remainder of the sum by computing remainders for different divisors. For any integer \( q \geq 2 \), let \( \text{lcm}(q) \) be the least common multiple of 2, 3, 4, \ldots, \( q \). For \( n \) given binary values \( a_0, a_1, \ldots, a_{n-1} \), let their \( \text{lcm}(q) \) remainder be \( x \mod \text{lcm}(q) \), where \( x = a_0 + a_1 + \cdots + a_{n-1} \). The remainder of \( n \) binary values can be computed in constant time on a \(((q-1)(q+4)/2) \times 2n\) RM as follows. First imagine that the RM is partitioned into \( q-1 \) subRMs of sizes \( 3 \times 2n, 4 \times 2n, \ldots, (q+1) \times 2n \). In each subRM of size \((i+1) \times 2n\) \((2 \leq i \leq q)\) the UNARY of \( x \mod i \) is computed using the prefix-remainders algorithm and is sent to all processors in the subRM. In each \( 2j \)th column of the subRM of size \((i+1) \times 2n\), it is checked whether \( j \mod i = x \mod i \). Then in each \( 2j \)th column, the logical OR technique shown in Lemma 3.1 is used to check whether \( j \mod i = x \mod i \) holds for all \( i \) \((2 \leq i \leq q)\). After that, the minimum \( j \) such that \( j \mod i = x \mod i \) for all \( i \), is computed using the rightmost determining. For such \( j \), the relation \( j = x \mod \text{lcm}(q) \) holds. Therefore, the UNARY of \( x \mod \text{lcm}(q) \) can be computed in constant time. Furthermore, the BINARY of \( x \mod \text{lcm}(q) \) can be obtained in constant time by converting the UNARY to the BINARY.

Using this remainder algorithm above, we can compute the sum efficiently. Let \( b_j \) be a binary value such that \( b_j = 1 \) iff \( x_j \mod \text{lcm}(q) = 0 \) and \( a_j = 1 \), where \( x_j = a_0 + a_1 + \cdots + a_j \). Then \( x = (x \mod \text{lcm}(q)) + (b_0 + b_1 + \cdots + b_{n-1}) \text{lcm}(q) \) holds. The BINARY of \( x \) is, therefore, computed by recursively computing the sum of \( \langle b_0, b_1, \ldots, b_{n-1} \rangle \): after computing the sum of \( b_0, b_1, \ldots, b_{n-1} \), the multiplication by \( \text{lcm}(q) \) is computed in constant time on a \( 2 \log n \times 4 \log^2 n \) RM, because this can be done by the multiplication of two \( \log n \)-bit integers. The addition to \( x \mod \text{lcm}(q) \) can be done in constant time on a \( \log n \) RM from Lemma 3.2. Hence, each recursion takes constant time.

Now let us estimate the depth \( t \) of the recursion. For the estimation of \( t \), we have to evaluate the value of \( \text{lcm}(q) \). From the prime number theorem, there are approximately \( q/\log q \) prime numbers less than or equal to \( q \). More precisely, for any small \( \varepsilon > 0 \), there exists \( q' \) such that for all \( q \geq q' \), the number of prime numbers less than or equal to \( q \) is more than \((1 - \varepsilon)q/\log q \) and less than \((1 + \varepsilon)q/\log q \). Hence, there are \( \Theta(q \log q) \) prime numbers in the range of \([q/2, q]\) and \( \text{lcm}(q) = \Theta(q)^{\Theta(q/\log q)} = 2^{\Theta(q)} \) holds. (See [22] for the detail of the estimation.) Since the recursion is terminated when all the \( b \)'s are zero, \( n \leq (\text{lcm}(q))^t \) is sufficient for the termination. Thus \( t \leq \log n/\log(\text{lcm}(q)) = O(\log n/\log q) \), which corresponds to the computing time. If an \( m \times 2n \) RM is available, the sum of \( n \) binary values can be computed in \( O(\log n/\sqrt{m}) \) time as follows: To execute the above algorithm on an \( m \times 2n \) RM, let \( q \) be the maximum integer satisfying \((q - 1)(q + 4)/2 \leq m \). Since \( q \geq \sqrt{m} \), the computing time is \( O(\log n/\sqrt{m}) \). Furthermore, if \( n \) binary value are given to an \( m \times n \) RM (one binary value for each processor in the bottom row), the sum can be computed without an asymptotic increase in the computing time: the sum for even columns and the sum for odd columns are computed, and then, their sum is computed by the addition algorithm for Lemma 3.2. Consequently we have

\textbf{Lemma 3.5} (Nakano [22]). \textit{The sum of \( n \) binary values can be computed in \( O(\log n/\sqrt{m}) \) time on an \( m \times n \) RM of the bit model.}
By choosing the prime numbers and computing the prefix remainders for them, the computing time of the above algorithm can be reduced to $O(\log n/\sqrt{m} \log m)$ [22]. However, the $O(\log n/\sqrt{m})$-time algorithm is sufficient for our algorithms to use as a subroutine.

4. Integer summing on the bit model

This section first gives an algorithm that sums $n$ binary values on a $\sqrt{nm} \times \sqrt{n}$ RM and then uses this algorithm in an algorithm that computes the sum of $n$ $d$-bit integers on a $\sqrt{nm} \times d \sqrt{n}$ RM.

4.1. Algorithm for summing $n$ binary values

This subsection gives a summing algorithm that is based on the algorithm for Lemma 3.5 but that is more efficient. The key idea is as follows. We first assume that the $nk$ ($k \leq n$) binary values are given, and show an efficient summing algorithm on an $2mk \times (2n + 1)$ RM. The algorithm for Lemma 3.5 can compute the sum of $nk$ binary values can be computed in $O(\log(nk)/\sqrt{m})$ on an $m \times nk$ RM. By embedding this algorithm to an $2mk \times (2n + 1)$ RM based on the snake like embedding, the sum can be computed faster.

Fig. 5 illustrates the snake-like embedding of an $m \times nk$ RM in a $2mk \times (2n + 1)$ RM. In snake-like embedding, a processor in the $m \times nk$ RM corresponds to a processor whose position is in an even row and odd column of the $2mk \times (2n + 1)$ RM. In the figure these processors are represented by thick circles and the connections for the embedding are represented by thick lines. The $m \times nk$ RM is bent $k - 1$ times and is partitioned into $k$ segments, each of which corresponds to consecutive $2m$ rows in the $2mk \times (2n + 1)$ RM. To complete the embedding, the connections between any two adjacent processors in the $m \times nk$ RM should be embedded in the $2mk \times (2n + 1)$ RM. Embedding of the connection between two adjacent processors in the same segment is trivial. To embed inter-segment connections, $m$ processors in the rightmost column of each segment should be connected one-to-one to $m$ processors in the next segment. For one-to-one connections between the $2m$ processors, $m$ odd rows in the $2mk \times (2n + 1)$ RM are used as shown in the figure. Similarly, $m$ processors in the leftmost column can be connected to the corresponding processors in the previous segment.

Suppose that the algorithm for Lemma 3.5 is used to compute the remainder of $nk$ ($k \leq n$) binary values $\langle a_0, a_1, \ldots, a_{nk-1} \rangle$ on the snake-like embedding. The $nk$ binary values are given to the RM such that $n$ binary values are given to each segment, one binary value for every two column (Fig. 5). That is, the input is given to every $2m$ rows in the $2mk \times (2n + 1)$ RM. Let $q$ be the maximum integer such that $(q-1)(q+4)/2 \leq m$. Using the algorithm for Lemma 3.5, it is easy to compute $x \mod \text{lcm}(q)$, 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 in the summing algorithm for Lemma 3.5. If the sum of $b_0, b_1, \ldots, b_{nk-1}$ is computed recursively,
the sum $x$ can be computed in $O(\log(nk)\sqrt{m})$ time as shown in Lemma 3.5. We can reduce the computing time, however, by compressing $b_0, b_1, \ldots, b_{nk-1}$.

Let $c_j = b_j \cdot \text{lcm}(q) + b_j \cdot \text{lcm}(q) + 1 + \cdots + b_{(j+1) \cdot \text{lcm}(q) - 1}$ for $0 \leq j \leq \frac{nk}{\text{lcm}(q)} - 1$. Note that $b_i = 1$ iff $(a_0 + a_1 + \cdots + a_i) \mod \text{lcm}(q) = 0$ and $a_i = 1$. Hence, if $b_i = 1$ then $b_{i+1} = b_{i+2} = \cdots = b_{i+\text{lcm}(q)} = 0$. Since at most one of $b_j \cdot \text{lcm}(q), b_{j+1} \cdot \text{lcm}(q), \ldots, b_{(j+1) \cdot \text{lcm}(q) - 1}$ is 1 for each $j$, the value of each $c_j$ is either 0 or 1 and can be determined by the logical OR of $b_j \cdot \text{lcm}(q), \ldots, b_{(j+1) \cdot \text{lcm}(q) - 1}$. See Fig. 6 for an example (in which a black circle denotes 1 and a white circle 0) where $n = 32$ and $\text{lcm}(q) = 4$. Although 4 is an impossible value for $\text{lcm}(q)$, the figure shows the essence of the algorithm. For $c_j$'s thus obtained, $x = (x \mod \text{lcm}(q)) + (c_0 + c_1 + \cdots + c_{nk/\text{lcm}(q)-1}) \cdot \text{lcm}(q)$ holds because each $c_j$ with value 1 corresponds to $\text{lcm}(q)$ 1's in the input. The value of $x$ can therefore also be obtained by the recursive computation of the sum of $(c_0, c_1, \ldots, c_{nk/\text{lcm}(q)-1})$. To speed the execution of the recursion, for each $j$, the values of $c_{j0}, c_{j1}, \ldots, c_{j(nk/\text{lcm}(q)-1)}$ are transferred to a row, one for each processor. For example, $c_0, c_1, \ldots, c_{n-1}$ are transferred to a row as shown in Fig. 6. Then all of the $c$'s are distributed to every $2m \cdot \text{lcm}(q)$ rows and the sum of $c_0, c_1, \ldots, c_{nk/\text{lcm}(q)-1}$ is computed recursively. Therefore, in the following recursion, the computation of the sum of $nk/\text{lcm}(q)$ binary values is based on the embedding of an $m \cdot \text{lcm}(q) \times nk/\text{lcm}(q)$ RM in the $m \times (2n+1)$ RM.

In each recursion, the binary of $(x \mod \text{lcm}(q)) + (c_0 + c_1 + \cdots + c_{nk/\text{lcm}(q)-1}) \cdot \text{lcm}(q)$ can be computed by using the algorithm for Lemma 3.4 in the same manner as the algorithm for Lemma 3.5. Therefore each recursion can be done in constant time. Finally, we will evaluate the depth of the recursion. Let $q_i$ be the value of $q$ at the $i$th recursion. For each $i$, integer $q_i$ is the maximum integer such that $(q_i - 1)(q_i + 4)/2 \leq m \cdot \text{lcm}(q_1) \cdot \text{lcm}(q_2) \cdots \text{lcm}(q_{i-1})$. Obviously, $q_i \geq \sqrt{m}$ holds. Furthermore, from $\text{lcm}(q_i) = 2^\theta(q_i)$, $q_i = 2^{\theta(q_{i-1})}$ holds. Let $t$ be the depth of the recursion. Then, the
condition \( \text{lcm}(q_1) \geq n \) is sufficient for the termination of the recursion. By applying \( \log^{(t-1)} \) to the inequation \( \text{lcm}(q_1) \geq n \), we have \( \Theta(q_1) \geq \log^{(t-1)} n \). Furthermore, by applying \( \log^{(\log^{*} n - t)} \) again, \( \log^{(\log^{*} n - t)}(\Theta(q_1)) \geq \log^{(\log^{*} n - 1)} n \geq 1 \) holds. Thus, from \( q_1 \geq \sqrt{m} \), the inequation \( t > \log^* n - \log^* m + O(1) \) is sufficient for the termination of the recursion. Therefore, the depth of the recursion is \( t = O(\log^* n - \log^* m) \). As a result, we can see that the sum of \( nk \) binary values can be computed in \( O(\log^* n - \log^* m) \) time on a \( 2^{mk} \times (2^n + 1) \) RM. Replacing \( n, m, \) and \( k \) by \( \sqrt{n}, \sqrt{m}, \) and \( \sqrt{n} \), we can compute the sum of \( n \) binary values in \( O(\log^* n - \log^* m) \) time on a \( 2^{\sqrt{nm}} \times (2^{\sqrt{n}} + 1) \) RM of the bit model.

Note that in this algorithm \( n \) binary values are given to a \( 2^{\sqrt{mn}} \times (2^{\sqrt{n}} + 1) \) RM, one value for each \( 2^{\sqrt{m}} \times 2 \) subRM. Therefore, by executing this algorithm four times we can compute the sum of \( 4n \) binary values (given one for each \( \sqrt{m} \times 1 \) subRM). Using this technique generally enables the size of a RM to be reduced by a constant factor without asymptotically increasing the computing time. As a result, we have

**Theorem 4.1.** For \( n \) binary values given to every \( \sqrt{m} \) rows of a \( \sqrt{m} \times \sqrt{n} \) RM, the sum of these values can be computed in \( O(\log^* n - \log^* m) \) time.

If \( m = \log^{(k)} n \), then \( \log^* n - \log^* m = k \). Thus, as a corollary to Theorem 4.1, we have

**Corollary 4.2.** 1. The sum of \( n \) binary values given one for each processor on a \( \sqrt{n} \times \sqrt{n} \) RM can be computed in \( O(\log^* n) \) time, and
2. The sum of \( n \) binary values given to every \( \log^{(O(1))} n \) rows of a \( \sqrt{n} \log^{(O(1))} n \times \sqrt{n} \) RM can be computed in constant time.

If the pipeline method is used, the size of the RM implementing the \( O(\log^* n) \)-time algorithm of Corollary 4.2 can be reduced to \( \sqrt{n/\log^* n} \times \sqrt{n/\log^* n} \). This reduction will be shown in Section 6.
4.2. Algorithm for summing n d-bit integers

This subsection shows an algorithm that computes the sum of n d-bit integers in $O(\log^* n - \log^* m)$ ($1 \leq m \leq \log n$) time on a $\sqrt{nm} \times d \sqrt{n}$ RM.

The algorithm is based on the two-stage summing method (Fig. 7). For each $i$ ($0 \leq i \leq n - 1$) let $A_i$ be a d-bit integer and $a_{i,d-1}a_{i,d-2} \cdots a_{i,0}$ its binary representation. The two-stage summing method computes the sum $x = A_0 + A_1 + \cdots + A_{n-1}$ as follows: In the first stage, for each $j$ ($0 < j < d - 1$) the value of $B_j = a_{0,j} + a_{1,j} + \cdots + a_{i,j}$ is computed. In other words, the sum of each column of square $A$ is computed and each $B_j$ corresponds to each row of parallelogram $B$ in the figure. For $B_j$'s thus obtained, computation of $B_0 \cdot 2^0 + B_1 \cdot 2^1 + \cdots + B_{d-1} \cdot 2^{d-1}$ in the second stage yields the sum $x$.

Fig. 8 shows a RM for computing the sum of n d-bit integers in a way based on the two-stage summing method. For each $j$, binary values $a_{0,j}, a_{1,j}, \ldots, a_{n-1,j}$ are given to a $\sqrt{nm} \times \sqrt{n}$ subRM corresponding to the $j$th square $A$ from the right in the figure. In each subRM $A$, the BINARYY of $B_j = a_{0,j} | a_{1,j} | \cdots | a_{n-1,j}$ is computed in $O(\log^* n - \log^* m)$ time by the summing algorithm for Theorem 4.1. Each sum thus obtained has log $n$ bits, and each bit is transferred to a subRM $B$ as follows: Since $B_j = b_{j,\log n}b_{j,\log n-1} \cdots b_{j,1}$ is computed in the $j$th square $A$, log $n$ bits are transferred to the $j$th subRM $B$ through the $(j + \log n - 1)$st subRM $B$, one bit for each subRM. Hence, 2 log $n$ rows arc enough to transfer every bit simultaneously. In the figure, these transfers are illustrated by arrows. After that, bits transferred to each subRM $B$ in Fig. 8 are correspond to a column of the parallelogram $B$ in Fig. 7. Therefore, to compute the sum of the parallelogram $B$ in Fig. 7, a $2 \log n \times 2 \log n$ lookahead carry generator is implemented in each subRM $B$. Each lookahead carry generator receives a carry from
the previous carry generator, adds it to the sum of a column of the parallelogram $B$ in
Fig. 7, and sends the carry to the following carry generator. Hence, in a way similar

Fig. 8. Integer summing algorithm for the bit model.

Theorem 4.3. The sum of $n$ $d$-bit integers can be computed in $O(\log^* n - \log^* m)$
$(1 < m \leq \log n)$ time on a $\sqrt{nm} \times d\sqrt{n}$ RM. Furthermore, big-‘O’ notation in
the size of the RM can be removed in the same way as Theorem 4.1. As a consequence, we have

Theorem 4.3. The sum of $n$ $d$-bit integers can be computed in $O(\log^* n - \log^* m)$
$(1 < m \leq \log n)$ time on a $\sqrt{nm} \times d\sqrt{n}$ RM.

Similar to Corollary 4.2, as a corollary to Theorem 4.3, we have

Corollary 4.4. The sum of $n$ $d$-bit integers can be computed
1. in $O(\log^* n)$ time on a $\sqrt{n} \times d\sqrt{n}$ RM, and
2. in constant time on a $\sqrt{n}\log^{O(1)} n \times d\sqrt{n}$ RM.

We also show in Section 6 that the size of a RM for the $O(\log^* n)$-time algorithm

5. Integer summing on the word model

This section describes an algorithm that computes the sum of $n$ $d$-bit integers, given
one for each processor, in $O(\log d + \log^* n)$ time on a $\sqrt{n} \times \sqrt{n}$ RM of the word model.
In this algorithm each $d$-bit integer is given in the VALUE format, and the VALUE
format of the sum is computed.

If $d > n^{1/4}$ the sum can be computed in $O(\log n) = O(\log d)$ time from Lemma 3.3.
Hence, we consider only the case where $d \leq n^{1/4}$.

First, let us imagine that the RM is partitioned into $\sqrt{n}/d \times \sqrt{n}/d$ subRMs each of
size $d \times d$ as shown in Fig. 9. The sum of each group can be computed in $O(\log d)$
time from Lemma 3.3. Since each sum has at most $d + \log d$ bits, we have to compute
the sum of $n/d^2 (d + \log d)$-bit integers. To do this, we apply the two-stage summing
method (Fig. 7).
To apply the first stage of the two-stage summing method, the sum of each bit of $n/d^2$ $(d + \log d)$-bit integers is computed independently: Every processor that is not in the diagonal of each $d \times d$ subRM configures the cross-over pattern as shown in Fig. 9. Then we can regard the RM as $d \sqrt{n}/d \times \sqrt{n}/d$ subRMs. Using each $\sqrt{n}/d \times \sqrt{n}/d$ subRM, the algorithm for Corollary 4.2 can compute the sum of $n/d^2$ binary values can be computed in $O(\log^* n)$ time. Note that the sum of each bit has at most $\log(n/d^2)$ bits.

We have next to compute the sum of the parallelogram $B$ in Fig. 7. To do this, we use the method used in Lemma 3.4 that implements lookahead carry generators. In this case, the size of each lookahead carry generator is $2 \log(n/d^2) \times 2 \log(n/d^2)$ and the number of generators is $d + \log d + \log(n/d^2) - 1 \leq d + \log n$. Therefore, $2 \log(n/d^2)(d + \log n) \times 2 \log(n/d^2) \leq 2n^{1/4} \log n \times 2 \log n$ is sufficient to compute all the bits in constant time. Note that the sum thus obtained in the BINARY format.

Finally, we have to convert the BINARY of the sum into the VALUE format. The BINARY has $\log n + \log d \leq 2 \log n$ bits, because $d \leq n^{1/4}$. The conversion can therefore be done in constant time. Since computing the partial sum of each $d \times d$ requires $O(\log d)$ time and the two-stage summing method requires $O(\log^* n)$ time, we have

**Lemma 5.1.** The sum of $n$ $d$-bit integers can be computed in $O(\log d + \log^* n)$ time on a $\sqrt{n} \times \sqrt{n}$ RM of the word model.

Furthermore, we can reduce the size of the RM to $\sqrt{n/(\log d + \log^* n)} \times \sqrt{n/(\log d + \log^* n)}$. In this case $(\log d + \log^* n)$ $d$-bit integers are given to each processor. Each processor first independently computes the sum of $\log^* n + \log d$ $d$-bit integers in $O(\log d + \log^* n)$ time. The sum thus obtained has at most $d + \log(\log d + \log^* n)$ bits. Then, the sum of $n/(\log d + \log^* n)$ $(d + \log(\log d + \log^* n))$-bit integers is computed in $O(\log(d + \log(\log d + \log^* n)) + \log^*(n/(\log d + \log^* n))) = O(\log d + \log^* n)$ time on the $\sqrt{n/(\log d + \log^* n)} \times \sqrt{n/(\log d + \log^* n)}$ RM by the algorithm for Lemma 5.1. Hence, the computing time is still $O(\log d + \log^* n)$. Therefore, we have
Theorem 5.2. The sum of \( n \) \( d \)-bit integers can be computed in \( O(\log d + \log^* n) \) time on a \( \sqrt{n/(\log d + \log^* n)} \times \sqrt{n/(\log d + \log^* n)} \) RM of the word model.

6. VLSI RC for summing integers

This section describes an \( AT \) optimal VLSI RC for summing integers. First, though, let us implement the algorithm for Theorem 4.1 to the VLSI RC, and estimate its \( AT \). Since each processor executes the algorithm in \( O(\log^* n - \log^* m) \) time and has a constant storage size, the processor can be embedded in \( O(\log^* n - \log^* m) \) area on the VLSI RC as follows. Each processor consists of \( O(\log^* n - \log^* m) \) registers, a local storage, a decoder, and a 4-terminal switch as shown in Fig. 10. The \( i \)th register stores an instruction that should be executed at time \( i \). Each instruction includes a code for control its 4-terminal switch and bit operations between local storage. The instruction stored in the \( i \)th register can be determined by an instruction dispatched by the single control unit of the original RM at time \( i \) and the coordinates of the processor. For each \( i \), an instruction stored in the \( i \)th register is moved to the \( (i - 1) \)th register during a single step execution of the algorithm. The decoder receives an instruction, a value of the local storage, and signals transferred through terminals of the 4-terminal switch, and then outputs a new value of local storage, signals to control the switch, and data to be sent from the terminals. Since the decoder receives and outputs a constant number of bits, the size and the depth of the circuit necessary to implement the decoder is constant. Therefore, the implementation shown in Fig. 10 has an area of \( O(\log^* n - \log^* m) \) and performs one step execution of a bit-model processor in constant machine cycles.

By using the implementation above, the total area is \( A = O(n\sqrt{m}(\log^* n - \log^* m)) \) and the computing time is \( T = O(\log^* n - \log^* m) \). Hence, \( AT = O(n\sqrt{m}(\log^* n - \log^* m)^2) \) holds and this \( AT \) upper bound does not match the trivial lower bound \( AT = \Omega(n) \). If the program has loops, a processor may be embedded in smaller area by reusing a subsequence of instructions. However, even if every processor can be implemented in constant area, the upper bound is \( AT = O(n\sqrt{m}(\log^* n - \log^* m)) \), which does not match the lower bound.

To attain \( AT = O(n) \) we will reduce the number of processors by using the Muller–Preparata’s VLSI circuit on the usual VLSI model. As a preliminary, we start with explaining the Muller–Preparata’s VLSI circuit \([21,32]\), which is an \( AT \) optimal VLSI circuit for summing \( n \) binary values. Fig. 11 illustrates the Muller–Preparata’s circuit \( W(k) \), which outputs the \( k \)-bit integer of the sum of \( 2^k \) binary values. It has a recursive and H-shaped structure such that \( W(k) \) can be constructed by two \( W(k - 1) \)s and \( k - 1 \) FAs (Full Adders). The sum of \( 2^{k-1} \) integers in each \( W(k - 1) \) is computed and they are added by the \( k - 1 \) FAs. The circuit can be laid out in \( O(\sqrt{2^k}) \times O(\sqrt{2^k}) \) area and has a depth of \( O(k) \). Hence, the sum of \( n = 2^k \) binary values can be computed in \( O(k) = O(\log n) \) time in the area of \( O(\sqrt{n}) \times O(\sqrt{n}) \).
Furthermore, by applying the pipeline scheme to $W(k)$, the sum of $k2^k$ binary values can be computed without asymptotically increasing the computing time. Let $S(k)$ be a $W(k)$ with $(k + \log k)$ FAs and $(k + \log k)$-bit register as shown in Fig. 11. The input is partitioned into $k$ groups, each of $2^k$ binary values. Each of groups is given to the $W(k)$ in sequence, and the sum of each group is outputted from the $W(k)$. The FAs add the sum of each group to the register. After adding all sums, the register holds the sum of $k2^k$ integers. Therefore, the sum of $n = k2^k$ binary values can be computed in $O(k) = O(\log n)$ time in $O(\sqrt{n}/\log n) \times O(\sqrt{n}/\log n)$ area and the circuit attains $AT = O(n)$.

By using the Muller–Preparata’s VLSI circuit $S(k)$ as a sub-circuit, the sum of $n$ binary values can be computed in $O(\log^* n)$ time on a VLSI RC of area $O(\sqrt{n}/\log^* n) \times$
Fig. 12 illustrates the VLSI RC, which is partitioned into \( \sqrt{n/(2^{\log^* n} \log^* n)} \times \sqrt{n/(2^{\log^* n} \log^* n)} \) blocks each of which occupies an area of \( O(\sqrt{2^{\log^* n}}) \times O(\sqrt{2^{\log^* n}}) \). Each block consists of \( S(\log^* n) \) and a \( 2^{\log^* n} \times 2^{\log^* n} \) subRM of the bit model. Since each processor has a program of size \( O(\log^* n) \), the area for each subRM is \( O((\log^* n)^3) \) and does not dominate the area in its block. Each subRM is connected to the subRMs in four adjacent blocks. Thus, the VLSI RC has a \( 2^{\sqrt{n \log^* n}} \times 2^{\sqrt{n \log^* n}} \) RM overall. In each block the sum of \( 2^{\log^* n} \log^* n \) binary values is computed in \( O(\log^* n) \) time by using \( S(\log^* n) \), and it is transferred to the subRM in the same block. The sum thus obtained has at most \( \log(2^{\log^* n} \log^* n) \leq 2^{\log^* n} \) bits. Then the sum of \( n/(2^{\log^* n} \log^* n) \) \( 2^{\log^* n} \)-bit integers is computed in \( O(\log^* n) \) time by the two-stage summing method in the same way as in the summing algorithm for the word model in Section 5. That is, in the first stage the sum of each digit of the \( n/(2^{\log^* n} \log^* n) \) \( 2^{\log^* n} \)-bit integers is computed independently by the RM. This can be done in \( O(\log^* n) \) time because the RM can be partitioned into \( 2^{\log^* n} \) subRMs each of size \( \sqrt{n/(2^{\log^* n} \log^* n)} \times \sqrt{n/(2^{\log^* n} \log^* n)} \). Each sum thus obtained has at most \( \log n \) bits. In the second stage \( 2^{\log^* n} \) lookahead carry generators, each of size \( 2^{\log n} \times 2^{\log n} \), are implemented in the RM, and the sum of \( 2^{\log^* n} \log^* n \)-bit integers is computed in constant time. The RM in the VLSI RC is large enough to implement the lookahead carry generators. Therefore, we have

**Theorem 6.1.** The sum of \( n \) binary values can be computed in \( O(\log^* n) \) time on a VLSI RC of area \( O(\sqrt{n/\log^* n}) \times O(\sqrt{n/\log^* n}) \).

The sum of integers can be computed on a VLSI RC by a method similar to that given in Subsection 4.2, which uses the RM as shown in Fig. 8. The RC for summing integers has the same layout as shown in Fig. 8, in that the RC for summing binary values is implemented in each \( A \) and a lookahead carry generator is implemented in each \( B \). Since each \( A \) occupies area of \( O(\sqrt{n/\log^* n}) \times O(\sqrt{n/\log^* n}) \), the total area is
The sum of \( n \) \( d \)-bit integers can be computed in \( O(\log^* n) \) time using a VLSI RC of area \( O(\sqrt{n/\log^* n}) \times O(d\sqrt{n/\log^* n}) \).

Furthermore, in general, a VLSI RC of area \( m \times n \) can be simulated by an \( m \times n \) RM of the bit model within a constant factor overhead in the computing time, because a processor can simulate a logic gate, a bit of a register, and a subbus can simulate a fixed link and bus in constant time. Therefore, from Theorems 6.1 and 6.2, we have

**Corollary 6.3.**
1. The sum of \( n \) binary values can be computed in \( O(\log^* n) \) time on a \( \sqrt{n/\log^* n} \times \sqrt{n/\log^* n} \) RM of the bit model.
2. The sum of \( n \) \( d \)-bit integers can be computed in \( O(\log^* n) \) time on a \( d\sqrt{n/\log^* n} \) RM of the bit model.

Note that this corollary assumes that the input is given to the RM by the pipeline scheme. If the pipeline scheme is not available, \( n \) processors to store the input are required.

### 7. Conclusions

This paper has presented efficient summing algorithms for binary values and integers on reconfigurable meshes of the bit and the word models. It has also presented \( AT \)-optimal VLSI reconfigurable circuits for summing problems. An interesting open problem is whether the sum of \( n \) binary values can be computed in constant time on a reconfigurable mesh of size \( \sqrt{n} \times \sqrt{n} \).

### Acknowledgements

The authors wish to thank the anonymous referees for many useful comments to improve this paper.

### References


