Bit Summation on the Reconfigurable Mesh

Martin Middendorf *

Institut für Angewandte Informatik
und Formale Beschreibungsverfahren, Universität Karlsruhe,
D-76128 Karlsruhe, Germany
mmi@aifb.uni-karlsruhe.de

Abstract. The bit summation problem on reconfigurable meshes is studied for the case where the number of bits is as large as the number of processsors. It is shown that kn binary values can be computed on a reconfigurable mesh of size $k \times n$ in time $O(\log^* k + (\log n / \sqrt{k \log k}))$.

1 Introduction

Processor arrays with a dynamically reconfigurable bussytem promise interesting applications because they allow to solve several problems much faster than on most other parallel architectures. The reconfigurability of the bussytem allows to transfer computations from time to space, because the formation of a new bus structure may depend on local data and can therefore be used as part of the computation. Clearly, there is a tradeoff between the size of the mesh and the time that can be achieved to solve a problem. Therefore, many of the fast algorithms for reconfigurable meshes work only for instances that are small compared to the size of the mesh (e.g. constant time summation of n numbers on a mesh of size $n \cdot \log^3 n$ [10]) or for sparse instances (e.g. matrix multiplication where time depends on the degree of sparseness of the matrices [5]).

One of the basic problems on reconfigurable meshes studied by several authors is the summation of binary values where each processor stores at most one bit. Algorithms for this problem are used as building blocks to solve several other problems (e.g. summation of integers [1], matrix multiplication [10]).

There are two models of reconfigurable meshes that differ by the buswidth and the maximal length of the operands for the arithmetic and logic operations that can be performed by the processors. In the bit-model the width of the buses is only one bit and processors can perform only bit operations. Whereas in the word-model the width of buses is at least the logarithm of the number of processors and the processors can perform operations on operands of that length. The bit summation problem has been studied for both models.

For the bit-model Miller et al. [6] and Wang et al. [11] have shown that n binary values can be added in time $O(1)$ on an $n \times n$ bit-model reconfigurable mesh. This was improved by Jang et al. [4] who gave a $O(1)$ algorithm to compute

* This work was done during the authors stay at the Faculty of Computer Science, University of Dortmund, Germany

the sum of n^2 bits (one per PE) on a bit-model $n \log n \times n \log n$ reconfigurable mesh. Note, that in both cases the bits of the result value has to be stored in different processors.

For the word-model reconfigurable mesh the following results are known. Jang and Prasanna [2] showed that n bits can be summed in time $O(t)$ on an $2n^{1/t} \times n$ RG for $1 \leq t \leq \log n$. Olariu et al. [9] showed that prefix sums of n binary values can be computed in time $O((\log n / \log m) + 1)$ on an $m \times n$ reconfigurable mesh. Miyashita et al. (cited in [8]) showed how to solve the prefix-sum problem for n binary values in $O(\log \log n)$ time on an $(\log^2 n/(\log \log n)^2) \times n$ reconfigurable mesh. Nakano [8] has shown that the prefix-sums of n binary values can be computed in time $O(\log n/\sqrt{k \log k}) + \log \log n)$ on an $k \times n$ reconfigurable mesh. This result implies that the problem can be solved in time $O(\log \log n)$ on an $(\log^2 n/(\log \log n)^3) \times n$ mesh which improves the result of Miyashita et al.. Nakano [7] also showed that n bits can be summed up in time $O(\log n/\sqrt{k \log k} + 1)$ on an $k \times n$ reconfigurable mesh. The addition of binary values on reconfigurable meshes where the number of processors equals the size of the problem instance was studied by Jang et al. [3]. They showed that an array of $n \times n$ bits can be summed in time $O(\log^* n)$ on an $n \times n$ word-model reconfigurable mesh. If the size of the mesh is allowed to be $\log^2 n \times n^2$ then n^2 bits can be added in constant time on the word-model reconfigurable mesh (Jang et al. [3], Park et al. [10]).

In this paper we study the bit summation problem on word-model reconfigurable meshes for case that the number of bits is as large the number of processors. We generalise the result of Jang et al. [3] by considering $k \times n$ meshes. Moreover, our result partially improves the result of Nakano [7]. We show that kn binary values can be computed in time $O(\log^* k + (\log n/\sqrt{k \log k}))$ on a reconfigurable mesh of size $k \times n$ if $n \geq \log^3 k$. Compared to the result of Nakano we can handle larger problem instances while using the same number of processors and the same time if $(\log n/\sqrt{k \log k}) \geq \log^* k$. The basic idea of our algorithm is to use the method of Nakano first in small submeshes to obtain prefix-remainders and a small number of remaining bits in every submesh. Then, several submeshes are combined to a larger submesh. Again the method of Nakano is applied to obtain prefix-remainders now with respect to a larger modulus and even fewer remaining one bits in every submesh. This process is repeated until we end up with at most n remaining bits in the whole mesh. Then the algorithm of Nakano is applied directly on the whole mesh.

2 Model of Computation

The model of computation is an SIMD $k \times n$-array of processing elements (PE's) with dynamically reconfigurable buses as depicted in Figure 1. We assume that only linear buses can be formed, i.e. every processore can connect only pairs of its ports. Every PE can read from every bus it is connected to, but only one PE at a time can write the value of one of its registers on a bus, i.e. we have CREW-buses. If no PE writes on a bus then its value is 0.

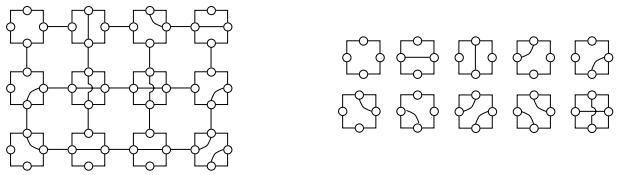

Fig. 1. Reconfigurable mesh (left) and possible connections of ports within a PE (right)

Every PE has a constant number of registers of length $\log n + \log k$. Every PE knows its row and column indices. Within one time step every PE can locally configure the bus, write to and/or read from one of the buses it is connected to, and perform some local computation. Signal propagation on buses is assumed to take constant time regardless of the number of switches on the bus — a standard assumption for this model of computation (e.g. [6]).

3 Nakano's Method

We first describe the algorithm of Nakano for computing the sum x of n binary values $a_1, a_2, \ldots, a_n$ on a reconfigurable mesh of size $k \times n$ ($k \geq 16$). For $k \leq 16$ his result holds trivially. The algorithm makes use of the result that prefix-remainders modulo m of n binary values can be computed in time $O(1)$ on an $(m+1) \times n$ reconfigurable mesh (e.g. Nakano et al. cited in [8]). The idea is to recursively apply the prefix-remainders algorithm using the fact that

$$a_1 + a_2 + \ldots + a_n = (a_1 + a_2 + \ldots + a_n) \bmod z + z \cdot (b_1 + b_2 + \ldots + b_n)$$

where $b_j = 1$ iff $a_j = 1 \wedge (a_1 + a_2 + \ldots + a_j) \bmod z = 0$. Instead of computing prefix-remainders with respect to some number z directly it is more space efficient to compute the prefix-remainders with respect to the prime factors of z. This is done in parallel for all prime factors of z using a submesh of size at least $(p+1) \times n$ for each prime factor p. Number z is chosen as the product of the first q prime numbers $p_1, p_2, \ldots, p_q$ for an integer q defined as follows: q is the maximal value such that $q \leq \lfloor \sqrt{k / \log k} \rfloor$ and $p_q \leq \lfloor \sqrt{k \log k} \rfloor - 1$. By the prime number theorem it can be shown that this definition of q allows to use for every prime factor p_i, $1 \leq i \leq q$ a submesh of size $\lfloor \sqrt{k \log k} \rfloor \times n$ and all this can be done on a mesh of size $k \times n$. By the Chinese Remainder theorem there exists

exactly one $y \in [1:z]$ with $(a_1+a_2+\ldots+a_n) \bmod p_i = (a_1+a_2+\ldots+a_n) \bmod y$ for all $i \in [1:q]$. For this y it holds that $(a_1+a_2+\ldots+a_n) \bmod z = y$.

To determine the first q prime numbers each PE P_{ij}, $i<j$, $i \leq (k \log k)^{1/4}$, $j \leq \sqrt{k \log k}$ tests whether i divides j and sets a flag to 1 if it does. If for all $i<j$ i does not divide j then j is a prime number. It is easy to determine this in time $O(1)$ for all $j \leq n$ in parallel by ORing over the flags in every column. For identifying which prime number is the ith one, $1 \leq i \leq q$ a simple counting technique that works in time $O(1)$ can be applied in a $q \times p_q$ submesh.

To determine the number y for which $(a_1+a_2+\ldots+a_n) \bmod p_i = (a_1+a_2+\ldots+a_n) \bmod y$ holds for all $i \in [1:q]$ the prefix-remainders are computed for all p_i with respect to the sequence consisting of n one bits. Then it is easy to determine in time $O(1)$ in every column j, $j \in [1:z]$ whether $j \bmod p_i = (a_1+a_2+\ldots+a_n) \bmod p_i$ for every $i \in [1:q]$.

The computation of z as the product of the first q prime numbers is done by computing in every column j, $j \leq n$ of the mesh whether $j \bmod p_i = 0$ holds for every $i \in [1:q]$. The smallest such j equals z. Clearly this can be done in time $O(1)$. The whole algorithm of Nakano works as follows.

Algorithm Sum-1 (Nakano [7]):

1. Determine the first $q(n)$ prime numbers $p_1, p_2, \ldots, p_q$
2. Compute $z = p_1 \cdot p_2 \cdot \ldots \cdot p_q$
3. Initialise variables $i := 0$ and $x := 0$.

WHILE OR$(a_1, a_2, \ldots, a_n)$ DO

4. Set $i := i+1$ and compute $x_i := (a_1+a_2+\ldots+a_n) \bmod z$.
5. For $j \in [1:n]$ set $b_j := 1$ if $a_j = 1$, and $a_1+\ldots+a_j \bmod z = 0$. Otherwise, set $b_j := 0$.
6. Set $a_j := b_j$ for $j \in [1:n]$.
7. $x := x + z^{i-1} \cdot x_i$

To analyse the running time of algorithm Sum-1 Nakano has shown the following lemma by using the prime number theorem.

Lemma 1. *[Nakano [7]] There exists a constant $c>0$ and an integer k_0 such that for all $k \geq k_0$ $q/2 \geq c \cdot \sqrt{k/\log k}$ and $p_{q/2} \geq c \cdot \sqrt{k \log k}$.*

Since every of the steps (1) to (7) works in time $O(1)$ it remains to determine how often steps (4) to (7) are executed. Using Lemma 1 it can be shown that there exists a constant $c>1$ such steps (4) to (7) are executed at most t times where t is the minimal integer with $(c^{\sqrt{k \log k}})^t \geq n$. Hence the running time is $O((\log n/\sqrt{k \log k})+1)$.

4 The Algorithm

In this section we describe our algorithm to compute the sum of kn bits $a_1, a_2, \ldots, a_{kn}$ on a $k \times n$ reconfigurable mesh. We assume that every PE stores exactly one of

the given bits. We need the following definition: For an integer $m \geq 16$ let $q(m)$ be the maximal value such that $q(m) \leq \lfloor \sqrt{m/\log m} \rfloor$ and $p_{q(m)} \leq \lfloor \sqrt{m \log m} \rfloor - 1$.

To describe the general idea of the algorithm let us assume first that we have to sum only kn/k_0 bits which are stored in the PEs of every k_0th row for some integer $k_0 \geq 16$. The algorithm starts by partitioning the mesh into small horizontal slices each of size $k_0 \times n$. The bits are stored in the PEs of the first rows of the slices. Let $a_{m1}, a_{m2}, \ldots, a_{mn}$ be the bits in the mth slice, $m \in [1 : k/k_0]$. In every slice m the prefix remainders modulo $z_1 := p_1 \cdot \ldots \cdot p_{q(k_0)}$ of the bits in the first row of the slice are computed using the method of Nakano. After setting all those one bits a_{mj} to zero for which $(a_{m1} + a_{m2} + \ldots + a_{mj}) \bmod z_1 \neq 0$ holds there are at most n/z_1 remaining one bits in the first row of each slice. These bits will be added in the next steps. Now the mesh is partitioned into larger slices each containing $k_1 := z_1 \cdot k_0$ rows. All the at most n one bits in each new slice are sent to disjoint PEs of the first row of the slice. Again, prefix-remainders of the bits in the first row are computed in each slice, but this time modulo $z_2 := p_1 \cdot \ldots \cdot p_{q(k_1)}$. The algorithm proceeds recursively in the same manner each time using broader slices until there is only one slice left containing at most n one bits in the first row. In this situation the algorithm of Nakano can be applied directly.

One problem remains, namely, how to compute the (weighted) sum of all remainders which are obtained during each iteration for every slice. It is not clear how to do this using a larger mesh or more time (with respect to $Big-O$). The idea is to circumvent the problem by glueing together all slices used in one step of the algorithm so that they form one long band. In every iteration only one remainder is computed for the whole band.

Now we give a more detailed description of the algorithm called Sum-2. We first assume that we have a $k \times 3n$ mesh and kn bits that have to be summed. The bits are stored in the PEs of the n middle columns, i.e. in PEs P_{ij} with $1 \leq i \leq k$, $n+1 \leq j \leq 2n$. We call the submesh containing the bits the *centre* and the submesh consisting of the first (last) n columns the *left* (*right*) *side*.

In the following we use the operation "the centre is configured into a band of size $m \times (kn/m)$". This operation is realized by partitioning the centre into slices of size $m \times n$ and connecting the last (first) PE of row i of slice s, for s odd (even), to the last (first) PE of row i of slice $s+1$, $i \in [1 : m]$, $s \in [1 : k/m - 1]$ (for ease of description we assume that m divides k). The sides are used to realize the connections. How this can be done is depicted in Figure 2.

The algorithm starts by configuring the centre into a band of size $(k_0 + 1) \times (kn/(k_0 + 1))$ for some constant $k_0 \geq 16$ (for ease of description we assume $k_0 + 1$ divides kn). For every row of the band the prefix-remainders of the bits in the row modulo k_0 are computed using the method of Nakano. Afterwards all bits are set to zero with the exception of those bits for which the prefix-sum modulo k_0 in the row up to the bit itself is zero. Then we compute the sum of the remainders modulo k_0 of the bits in a row that were obtained one for each row using an ordinary $O(\log k_0)$ time tree structured algorithm. The result is sent to PE P_{11} and stored there in variable x which will later hold the final result.

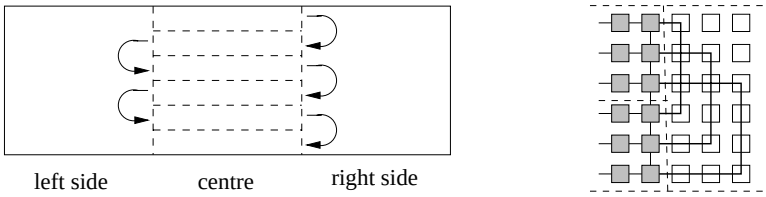

Fig. 2. Forming a band by connecting slices. Right: detail showing connections between PEs (dark boxes) of two slices using the right side PEs (light boxes)

Observe, that there are at most n/k_0 remaining one bits in every row. The centre is partitioned into slices of size $k_0 \times n$ and the at most $k_0 \times (n/k_0) = n$ one bits in a slice are sent to disjoint PEs of the first row of that slice.

Now the algorithm determines all prime numbers $p_1, p_2 \ldots, p_{q(k)}$ as in the algorithm of Nakano. The centre is configured into a band of size $k_1 \times (kn/k_1)$ with $k_1 := k_0$. Then prefix-remainders modulo $z_1 := p_1 \cdot \ldots \cdot p_{q(k_1)}$ are computed in the band. Afterwards all one bits are set to zero with the exception of those one bits for which the prefix-sum in the band up to the bit itself equals 0 mod z_1. At most $kn/(z_1 \cdot k_0)$ one bits are remaining in the first row of the band. The computed remainder of the bits in the band modulo z_1 is sent to PE P_{11} where it is multiplied by k_0 and added to x.

Now, the centre is partitioned into broader slices containing $k_2 := z_1 \cdot k_0$ rows. All one bits in a slice are sent to disjoint PEs in the first row of the slice. The algorithm proceeds similar as above, i.e. the centre is configured into a band of size $k_2 \times (kn/k_2)$ with, the prefix remainders modulo $z_2 := p_1 \cdot \ldots \cdot p_{q(k_2)}$ are computed and so forth. This procedure is applied recursively on bands of increasing width and decreasing length. The recursion stops when the band has width n. Then the mesh contains at most n remaining one bits which are stored in different PEs of the first row and the algorithm of Nakano can be applied directly to compute the sum of the remaining bits. The obtained result is multiplied by $z_{r-1} \cdot z_{r-2} \cdot \ldots \cdot z_1 \cdot k_0$ and added to variable x in PE P_{11} which stores now the final result.

It remains to describe how the algorithm Sum-2 works on a mesh of size $k \times n$. The idea is to fold the sides that were used to realize the connections into the centre. To make this possible we perform the procedure — called Sum-

Submesh — described above four times. The first time we sum only bits stored in PEs of the odd rows and odd columns reserving the other PEs for realizing the connections (see Figure 3). The other three times only bits in PEs of odd rows and even columns (respectively even rows and odd columns, even rows and even columns) are summed. As can be seen from Figure 3 we also need a "free" column before and behind the PEs and a "free" row below the PEs containing the bits that are summed in the procedure described above. Therefore algorithm Sum-2 starts by applying the algorithm of Nakano to sum the bits in the first and last columns and the last row so that they become free (Procedure Sum-Submesh is given later).

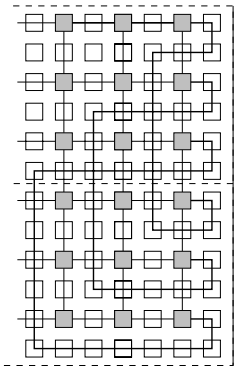

Fig. 3. Connections between two slices

Algorithm: Sum-2:

1. Compute the first prime numbers $p_1, p_2, \ldots, p_{q(k)}$ using a submesh of size $(k \log k)^{1/4} \times \sqrt{k \log k}$.
2. Compute the sum of the bits in the first and last columns, and the last row using algorithm Sum-1 three times.
3. If n and k are both even apply procedure Sum-Submesh four times to the following submeshes (the cases that n or k are odd are similar and omitted here): the submesh consisting of columns 1 to $n-1$ and rows 1 to k, the submesh consisting of columns 1 to n and rows 1 to k, the submesh consisting of columns 1 to $n-1$ and rows 2 to $k-1$, the submesh consisting of columns 1 to $n-1$ and rows 2 to $k-1$.

In the following procedure Sum-Submesh we assume that we have a mesh called centre of size $k \times n$ which contains the bits to be added and also "hidden" columns and rows of PEs that allow to configure the centre as a band. Recall that the prime numbers $p_1, \ldots, p_{q(k)}$ have already been computed by Sum-2 before procedure Sum-Submesh is invoked.

Procedure: Sum-Submesh:

1. Configure the centre as a band of size $(k_0+1) \times kn/(k_0+1)$.
2. FOR $i = 1$ TO k_0+1 DO

   Compute the prefix-remainders modulo k_0 of the bits in row i of the band. Let x_i be the remainder modulo k_0 of the bits in row i. Let $a_{i,1} \ldots, a_{i,kn/(k_0+1)}$ be the bits in row i of the band. Set $b_{ij} := 1$, $j \in [1 : kn/(k_0+1)]$ if $a_{ij} = 1$ and $a_{i1} + \ldots + a_{ij} \bmod k_0 = 0$ and, otherwise, set $b_{ij} := 0$.
3. Determine the sum y of all x_i that were computed in Step (2) by a simple $O(\log k_0)$ tree structured algorithm. Send the result to PE P_{11} and set $x := y$.
4. Partition the centre into slices of size $k_0 \times n$ and send all b_{ij}'s with value one into disjoint PEs of the first row of the slice as follows: In row m, $m \in [1 : k_0]$ there is at most one 1-bit b_{ij} in columns $1 + (l-1) \cdot k_0$ to $l \cdot k_0$ for each $l \in [1 : n/k_0]$. If there is such a 1-bit send it first in its row to the PE in column $m + (l-1) \cdot k_0$ and then along the column to the PE in the first row of the slice.
5. Set $r = 1$ and $k_1 = k_0$.

WHILE $k_r < k$ DO

6. Determine $z_r := p_1 \cdot p_2 \cdot \ldots \cdot p_{q(k_r)}$ and send it to PE P_{11}.
7. Configure a $k_r \times kn/k_r$ band. Compute the prefix remainders modulo z_r of the bits in the first row. Let $a_1, a_2, \ldots, a_{kn/k_r}$ be the bits in the first row of the band. For $i \in [1 : kn/k_r]$ set $b_i := 1$ if $a_i = 1$ and $a_1 + \ldots + a_i \bmod z_r = 0$ and otherwise, set $b_i := 0$. Send the obtained remainder modulo z_r off all bits in the first row to PE P_{11}, multiply it by $z_{r-1} \cdot \ldots \cdot z_1 \cdot k_1$ and add it to variable x.
8. Partition the centre into slices of size $k_{r+1} \times n$ where $k_{r+1} := k_r \cdot z_r$. Similar as in Step (4) send all b_i's which are one into disjoint PEs of the first row of it's slice.
9. Set $r := r + 1$

10. Compute the sum of the remaining n one bits using algorithm Sum-1. Send the result to PE P_{11}, multiply it by $z_r \cdot \ldots \cdot z_1 k_1$ and add it to variable x.

The correctness of algorithm Sum-2 is easy to show. Let us analyse the running time. Step (1) of Sum-2 takes time $O(1)$. Step (2) applies the algorithm Sum-1 and therefore takes time $O(\log n/\sqrt{k \log k} + 1)$. The final Step (3) invokes procedure Sum-Submesh four times. It remains to analyse the running time of Sum-Submesh. Step (1) of Sum-Submesh takes constant time. Step (2) takes time $O(k_0 + 1)$ which is $O(1)$ since k_0 is constant. Step (3) is done in time $O(\log k_0)$ which is constant. Each of the steps (4) - (9) can be done in time $O(1)$. Step (10) applies the algorithm Sum-1 and therefore needs time $O(\log n/\sqrt{k \log k} + 1)$. It remains to analyse how often the steps (6) - (9) are repeated. By Lemma 1 there exists a constant $c > 0$ and an integer $\hat{k}$ such that for all $k \geq \hat{k}$ we have $q/2 \geq c \cdot \sqrt{k/\log k}$ and $p_{q/2} \geq c \cdot \sqrt{k \log k}$. Thus, for $k_0 = k_1 \geq \hat{k}$ we have that $\log(p_1 \cdot p_2 \cdot \ldots \cdot p_{q(k_1)}) \geq c \cdot (q(k_1)/2) \cdot \log p_{q(k_1)/2} \geq$

$c \cdot \sqrt{k_1/\log k_1} \cdot \log(c \cdot \sqrt{k_1 \log k_1}) \geq \frac{1}{2} c \cdot \sqrt{k_1 \log k_1}$. This implies that $z_1 = p_1 \cdot \ldots \cdot p_{k_1} \geq \hat{c}^{\sqrt{k_1 \log k_1}}$ for some constant $\hat{c}$. Hence $k_2 \geq k_1 \cdot \hat{c}^{\sqrt{k_1 \log k_1}}$. Iteratively one can show that $k_j \geq k_{j-1} \cdot \hat{c}^{\sqrt{k_{j-1} \log k_{j-1}}}$, $j \in [2:r]$. Thus for k_1 large enough, but still constant, we have $k_j \geq k_{j-1} \hat{c}^{\sqrt{k_{j-1} \log k_{j-1}}} \geq 2^{\sqrt{k_{j-1}}}$. Thus $k_j \geq 2^{\sqrt{2\sqrt{k_{j-2}}}}$. For k_{j-2} large enough we obtain $k_j \geq 2^{k_{j-2}}$. Thus for k_0 large enough we have $r/2 \leq \log^* k = O(\log^* k)$. Altogether we obtain that the running time of algorithm Sum-2 is $O(\log^* k + \log n/\sqrt{k \log k})$ and the following theorem is shown.

Theorem 1. *The sum of kn bits can be computed in $O(\log^* k + \log n/\sqrt{k \log k})$ time on a reconfigurable mesh of size $k \times n$ using only linear buses.*

5 Conclusion

We have shown that kn binary values can be computed on a reconfigurable mesh of size $k \times n$ in time $O(\log^* n + (\log n/\sqrt{k \log k}))$. This result partially improves a result of Nakano who has shown for a reconfigurable mesh of the same size as ours that the sum of n can be computed in time $O(\log n/\sqrt{k \log k}) + 1)$. That is we can add more bits on a reconfigurable mesh of the same size in the same time if $(\log n/\sqrt{k \log k}) \geq \log^* k$.

References

1. P. Fragopoulou, On the efficient summation of n numbers on an n-processor reconfigurable mesh. *Par. Proc. Lett.* 3: 71-78, 1993.
2. J. Jang, V.K. Prasanna. An optimal sorting algorithm on reconfigurable mesh. In: *Proc. Int. Parallel processing Symp.*, 130-137, 1992.
3. J.-W. Jang, H. Park, V.K. Prasanna. A fast algorithm for computing histogram on reconfigurable mesh. In: *Proc. of Frontiers of Massively parallel Computation '92*, 244-251, 1992.
4. J.-W. Jang, H. Park, V.K. Prasanna. A bit model of reconfigurable mesh. In :*Proc. Workshop on Reconfigurable Architectures*, Cancun, Mexico, 1994.
5. M. Middendorf, H. Schmeck, H. Schröder, G. Turner. Multiplication of matrices with different sparseness properties. to appear in: *VLSI Design.*
6. R. Miller, V.K. Prasanna-Kumar, D.I. Reisis, Q.F. Stout. Parallel Computations on Reconfigurable Meshes. *IEEE Trans. Comput.* 42: 678-692, 1993.
7. K. Nakano. Efficient summing algorithms for a reconfigurable mesh. In: *Proc. Workshop on Reconfigurable Architectures*, Cancun, Mexico, 1994.
8. K. Nakano. Prefix-sum algorithms on reconfigurable meshes. *Par. Process. Lett.*, 5: 23-35, 1995.
9. S. Olariu, J. L. Schwing, J. Zhang. Fundamental algorithms on reconfigurable meshes. Proc. 29th Allerton Conference on Communications, Control, and Computing, 811-820, 1991.
10. H. Park, H.J. Kim, V.K. Prasanna. An $O(1)$ time optimal algorithm for multiplying matrices on reconfigurable mesh. *Inf. Process. Lett.*, 47: 109-113, 1993.
11. B.F. Wang, G.H. Chen, H. Li. Configurational computation: A new computation method on processor arrays with reconfigurable bus systems. In: *Proc. 1st Workshop on Parallel Processing*, Tsing-Hua University, Taiwan, 1990.