An Efficient Algorithm for Row Minima Computations on Basic Reconfigurable Meshes

Koji Nakano, *Member*, *IEEE*, and Stephan Olariu, *Member*, *IEEE*

Abstract—A matrix A of size $m \times n$ containing items from a totally ordered universe is termed *monotone* if, for every i, j, $1 \leq i < j \leq m$, the minimum value in row j lies below or to the right of the minimum in row i. Monotone matrices, and variations thereof, are known to have many important applications. In particular, the problem of computing the row minima of a monotone matrix is of import in image processing, pattern recognition, text editing, facility location, optimization, and VLSI. Our first main contribution is to exhibit a number of nontrivial lower bounds for matrix search problems. These lower bound results hold for arbitrary, infinite, two-dimensional reconfigurable meshes as long as the input is pretiled onto a contiguous $n \times n$ submesh thereof. Specifically, in this context, we show that every algorithm that solves the problem of computing the minimum of an $n \times n$ matrix must take $\Omega(\log \log n)$ time. The same lower bound is shown to hold for the problem of computing the minimum in each row of an arbitrary $n \times n$ matrix. As a byproduct, we obtain an $\Omega(\log \log n)$ time lower bound for the problem of selecting the kth smallest item in a *monotone* matrix, thus extending the best previously known lower bound for selection on the reconfigurable mesh. Finally, we show an $\Omega\left(\sqrt{\log \log n}\right)$ time lower bound for the task of computing the row minima of a *monotone* $n \times n$ matrix. Our second main contribution is to provide a nearly optimal algorithm for the row-minima problem: With a monotone matrix of size $m \times n$ with $m \leq n$ pretiled, one item per processor, onto a basic reconfigurable mesh of the same size, our row-minima algorithm runs in $O(\log n)$ time if $1 \leq m \leq 2$ and in $O\left(\frac{\log n}{\log m} \log \log m\right)$ time if $m > 2$. In case $m = n^{\epsilon}$ for some constant ϵ, $(0 < \epsilon \leq 1)$, our algorithm runs in $O(\log \log n)$ time.

Index Terms—Monotone matrices, totally monotone matrices, search problems, row minima, reconfigurable meshes, basic reconfigurable meshes, VLSI design, facility location problems, cellular system design.

1 Introduction

Recently, in an attempt to reduce its large computational diameter, the mesh-connected architecture has been enhanced with various broadcasting capabilities. Some of these involve endowing the mesh with *static* buses, that is, buses whose configuration is fixed and cannot change; more recently, researches have proposed augmenting the mesh architecture with *reconfigurable* broadcasting buses: These are high-speed buses whose configuration can be dynamically changed in response to specific processing needs. Examples include the *bus automaton* [25], [26], the *reconfigurable mesh* [21], the *mesh with bypass capability* [12], the *content addressable array processor* [31], the *reconfigurable network* [7], the *polymorphic processor array* [16], [20], the *reconfigurable bus with shift switching* [15], the *gated-connection network* [27], [28], the PARBS [30], and the *polymorphic torus network* [13], [17]. We refer the interested reader to the comprehensive survey paper of Nakano [22].

Among these architectures, the reconfigurable mesh and its variants have turned out to be valuable theoretical models that allowed researchers to fathom the power of reconfiguration and its relationship with the PRAM. From a practical standpoint, however, the reconfigurable mesh and its variants [21], [30] omit important properties of physical architectures and, consequently, do not provide a complete and precise characterization of real systems. Moreover, these models are so flexible and powerful that it has turned out to be impossible to derive from them high-level programming models that reflect their flexibility and intrinsic power [16], [20]. Worse yet, it has recently been shown that the reconfigurable mesh and the PARBS do not scale and, as a consequence, do not immediately support virtual parallelism [18], [19].

Motivated by the goal of developing algorithms in a scalable model of computation, we adopt a restricted version of the reconfigurable mesh that we call the *basic reconfigurable mesh* (BRM, for short). Our model is derived from the Polymorphic Processor Array (PPA) proposed in [16], [20]: The BRM shares with the PPA all the restrictions on the reconfigurability and the directionality of the bus system. The BRM differs from the PPA in that we do not allow torus connections. As a result, the BRM is potentially weaker than the PPA. It is very important to stress that the programming model developed in [16], [20] for the PPA also applies to the BRM. In particular, all the broadcast primitives developed in [16], [20], with the exception of those using torus connections, can be inherited by the BRM. In fact, all the algorithms developed in this paper could have been just as easily written using the extended C language

• *K. Nakano is with the Department of Electrical and Computer Engineering, Nagoya Institute of Technology, Showa-ku, Nagoya 466, Japan. E-mail: nakano@elcom.nitech.ac.jp.*
• *S. Olariu is with the Department of Computer Science, Old Dominion University, Norfolk, VA 23529. E-mail: olariu@cs.odu.edu.*

Manuscript received 13 Sept. 1996.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number 100297.

TABLE 1
ILLUSTRATING A MONOTONE MATRIX

2	13	6	10	11
5	**3**	10	9	5
10	**6**	9	8	22
20	9	**4**	8	17
16	21	17	**9**	19

primitives of [16], [20]. We opted for specifying our algorithm in a more conventional fashion only to make the presentation easier to follow.

Consider a two-dimensional array (i.e., a matrix) A of size $m \times n$ with items from a totally ordered universe. Matrix A is termed *monotone* if, for every i, j, $1 \le i < j \le m$, the smallest value in row j lies below or to the right of the smallest value in row i, as illustrated in Table 1, where the row minima are highlighted. A matrix A is said to be *totally monotone* if every submatrix of A is monotone. The concepts of monotone and totally monotone matrices may seem artificial and contrived at first. Rather surprisingly, however, these concepts have found dozens of applications to problems in optimization, VLSI design, facility location problems, string editing, pattern recognition, computational geometry, and cellular system design, among many others. The reader is referred to [1], [2], [3], [4], [5], [6], where many of these applications are discussed in detail.

One of the recurring problems involving matrix searching is referred to as *row-minima computation* [6]. In particular, Aggarwal et al. [2] showed that the task of computing the row-minima of an $m \times n$ monotone matrix has a sequential lower bound of $\Omega(n \log m)$. They also showed that this lower bound is tight by exhibiting a sequential algorithm for the row-minima problem running in $O(n \log m)$ time. In the case where the matrix is totally monotone, the sequential complexity is reduced to $\Theta(m + n)$.

To the best of our knowledge, no time lower bound for the row-minima problem has been obtained in parallel models of computation, in spite of the importance of this problem. The first main contribution of this paper is to propose a number of nontrivial time lower bounds for matrix search problems. These lower bounds hold for general two-dimensional reconfigurable meshes of infinite size, as long as the input is pretiled onto a contiguous submesh of size $n \times n$. Specifically, in this context, we show that every algorithm that solves the problem of computing the smallest item of an $n \times n$ matrix must take $\Omega(\log \log n)$ time. The same lower bound is shown to hold for the problem of computing the minima in each row of an arbitrary $n \times n$ matrix. As a byproduct, we obtain an $\Omega(\log \log n)$ time lower bound for the problem of selecting the kth smallest item in a *monotone* matrix. Previously, Hao et al. [10] have obtained an $\Omega(\log \log n)$ lower bound for selection in arbitrary matrices on *finite* reconfigurable meshes. Thus, our lower bound extends the result of [10] in two directions: We show that the same lower bound applies to selection on monotone matrices and on a reconfigurable mesh of an infinite size. Finally, we show an almost tight $\Omega\left(\sqrt{\log \log n}\right)$ time lower bound for the task of computing the row minima of a *monotone* $n \times n$ matrix. Our second main contribution is to provide an efficient algorithm for the row-minima problem: With a monotone matrix of size $m \times n$ with $m \le n$ pretiled, one item per processor, onto a BRM of the same size, our row-minima algorithm runs in $O(\log n)$ time if $1 \le m \le 2$ and in $O\left(\frac{\log n}{\log m} \log \log m\right)$ time if $m > 2$. In case $m = n^{\epsilon}$ for some constant ϵ, $(0 < \epsilon \le 1)$, our algorithm runs in $O(\log \log n)$ time.

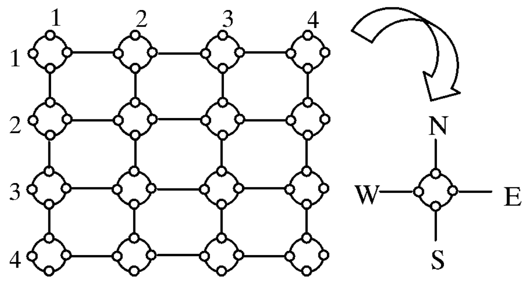


Fig. 1. A basic reconfigurable mesh of size 4×4.

The remainder of this work is organized as follows: Section 2 introduces the model of computations adopted in this paper; Section 3 discusses a number of relevant lower-bound results; Section 4 presents basic algorithms that will be key in our subsequent row-minima algorithm; Section 5 gives the details of our row-minima algorithm; finally, Section 6 offers concluding remarks and poses open problems.

2 THE BASIC RECONFIGURABLE MESH

A basic reconfigurable mesh (BRM, for short) of size $m \times n$ consists of mn identical SIMD processors positioned on a rectangular array with m rows and n columns. As usual, it is assumed that every processor knows its own coordinates within the mesh: We let $P(i, j)$ denote the processor placed in row i and column j, with $P(1, 1)$ in the north-west corner of the mesh.

Each processor $P(i, j)$ is connected to its four neighbors $P(i - 1, j)$, $P(i + 1, j)$, $P(i, j - 1)$, and $P(i, j + 1)$, provided they exist, and has four ports N, S, E, and W, as illustrated in Fig. 1. Local connections between these ports can be established, subject to the following restrictions:

1) In each time unit, at most one of the pairs of ports (N, S) or (E, W) can be set; moreover,
2) All the processors that connect a pair of ports must connect the same pair;
3) Broadcasting on the resulting subbuses is unidirectional. For example, if the processors set the (E, W) connection, then, on the resulting horizontal buses, all broadcasting is done either "eastbound" or else "westbound," but not both.

We refer the reader to Figs. 2a and 2b for an illustration of several possible unidirectional subbuses. The BRM is very much like the recently proposed PPA multiprocessor array, except that the BRM does not have the torus connections present in the PPA. In a series of papers [16], [18], [19], [20], Maresca and his coworkers demonstrated that the PPA architecture and the corresponding programming

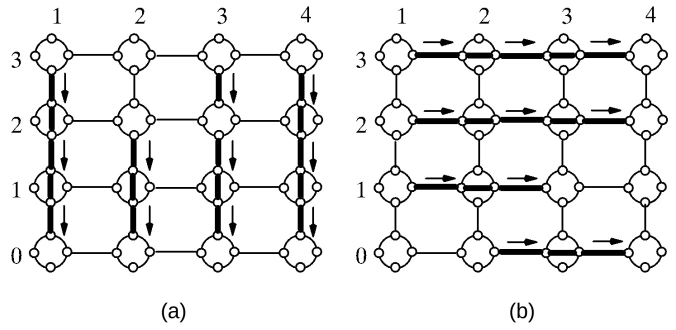


Fig. 2. Examples of unidirectional horizontal subbuses.

environment is not only feasible and cost effective to implement, it also enjoys additional features that set it apart from the standard reconfigurable mesh and the PARBS. Specifically, these researchers have argued convincingly that the reconfigurable mesh is too powerful and unrestricted to support virtual parallelism under present-day technology. By contrast, the PPA architecture has been shown to scale and, thus, to support virtual parallelism [16], [18].

The BRM is easily shown to inherit all these attractive characteristics of the PPA, including the support of virtual parallelism and the C-based programming environment, making it eminently practical. As in [16], we assume ideal communications along buses (no delay). Although inexact, a series of recent experiments with the PPA [16] and the GCN [27], [28] seem to indicate that this is a reasonable working hypothesis.

3 Lower Bounds

The main goal of this section is to demonstrate nontrivial lower bounds for several matrix search problems. Our lower bound arguments do not use the restrictions of the BRM, holding for more powerful reconfigurable meshes that allow any local connections. In fact, our arguments hold for arbitrary two-dimensional, reconfigurable meshes of an infinite size, provided that the input is placed into a contiguous $n \times n$ submesh thereof.

Formally, this section deals with the following problems:

Problem 1. Given an $n \times n$ matrix pretiled one item per processor onto an $n \times n$ submesh of an $\infty \times \infty$ reconfigurable mesh, find the minimum item in the matrix.

Problem 2. Given an $n \times n$ matrix pretiled one item per processor onto an $n \times n$ submesh of an $\infty \times \infty$ reconfigurable mesh, find the minimum item of each row.

Problem 3. Given an $n \times n$ *monotone* matrix pretiled one item per processor onto an $n \times n$ submesh of an $\infty \times \infty$ reconfigurable mesh, find the minimum item of each row.

Problem 4. Given an $n \times n$ *totally monotone* matrix pretiled one item per processor onto an $n \times n$ submesh of an $\infty \times \infty$ reconfigurable mesh, find the minimum item of each row.

We propose to show that Problems 1 and 2 have an $\Omega(\log\log n)$-time lower bound and that Problem 3 has an $\Omega\left(\sqrt{\log\log n}\right)$-time lower bound. The lower bound for Problem 4 is still open.

The proofs are based on a technique detailed in [11], [29] that uses the following graph-theoretic result of Turán [8]. Recall that an *independent set* in a graph is a set of pairwise nonadjacent vertices.

Lemma 3.1. *Let $G = (V, E)$ be an arbitrary graph. G has an independent set U such that*

$$|U| \geq \frac{|V|^2}{2|E| + |V|}.$$

This lemma is used in an implicit adversary argument to bound from below the number of items in the matrix that are possible choices for the minimum. Let V be the set of candidates for the minimum at the beginning of the current iteration and let E stand for the set of pairs of candidates that are compared within the current iteration. The situation benefits by being modeled by a graph $G = (V, E)$ with V and E representing, respectively, the vertices and the edges of the graph. It is intuitively obvious that an adversary can choose the outcome of the comparisons in such a way that the next set of candidates is no larger than the size of an independent set U in G. In other words, for a set V of candidates and for a set E of pairs that are compared by a minimum finding algorithm, items in the independent set U have the potential of becoming the minimum. Consequently, all items in U are still candidates for the minimum after comparing all pairs in E.

To make the presentation easier to follow, we assume that each time unit is partitioned into the following three stages:

Phase 1: Bus reconfiguration. The processors set local connections;

Phase 2: Broadcasting. The processors send at most one data item to each port, and receive one data item from each port;

Phase 3: Local computation. Every processor selects two elements stored in its local memory, compares them and changes its internal status.

We begin by proving the following lemma.

Lemma 3.2. *Every algorithm that solves Problem 1 requires $\Omega(\log\log n)$ time.*

Proof. Let us evaluate the number of pairs that can be compared by an algorithm in Phase 3 of time unit t. Notice that in Phase 2 of a time unit, at most $4n$ items can be sent to the outside of the submesh. Hence, altogether, at most $4nt$ items can be sent before the execution of Phase 3 of time unit t. Therefore, the outside of the submesh can compare at most $\binom{4nt}{2} \leq 16n^2t^2$ pairs of items. The inside of the submesh can compare at most n^2 pairs in each Phase 3. Consequently, in Phase 3 of time unit t, at most $16n^2t^2 + n^2 \leq 17n^2t^2$ pairs can be compared by the $\infty \times \infty$ reconfigurable mesh. Let c_t be the number of candidates that can be the minimum after Phase 3 of time unit t. Then, by virtue of Lemma 3.1, we have,

$$\begin{aligned} c_t &\ge c_{t-1}^{\,2} / (2 \cdot 17n^2t^2 + c_{t-1}) \\ &\ge c_{t-1}^{\,2} / (35n^2t^2). \qquad (\text{since } c_{t-1} \le n^2) \end{aligned}$$

By applying the logarithm, we obtain

$$\begin{aligned} \log c_t &\ge 2 \log c_{t-1} - \left(\log 35 + 2\log n + 2\log t\right) \\ &\ge 2\log n - \sum_{i=1}^{t}\left(2^{i-1}\log 35 + 2^{t-i}\log i\right) \\ &\ge 2\log n - 2^t\left(\log 35 + \log t\right). \end{aligned}$$

To complete the algorithm at the end of T time units, c_T must be less than or equal to one. Therefore, $2 \log n \le 2^T(\log 35 + \log T)$ must hold. In turn, this implies that $T \in \Omega(\log \log n)$, as claimed. □

It is worth mentioning that Lemma 3.2 implies a similar lower bound for the task of selection in *monotone* matrices. To see this, note that given an arbitrary matrix A of size $n \times n$ we can construct a monotone matrix A' of size $n \times (n + 1)$ by simply adjoining to A a column vector of all of whose entries are $-\infty$. It is now clear that the minimum item in A is precisely the $(n + 1)$th smallest item in A'. Thus, we have the following result.

LEMMA 3.3. *Every algorithm that selects the* kth *smallest item in a monotone matrix of size* $n \times n$ *requires* $\Omega(\log \log n)$ *time.*

Previously, Hao et al. [10] have obtained an $\Omega(\log \log n)$ lower bound for selection in arbitrary matrices on *finite* reconfigurable meshes. Thus, Lemma 3.3 extends the result of [10] in two directions: First it shows that $\Omega(\log \log n)$ remains the lower bound for selection on monotone matrices and, second, it shows that the lower bound must hold even for *infinite* reconfigurable meshes.

LEMMA 3.4. *Every algorithm that solves Problem 2 requires* $\Omega(\log \log n)$ *time.*

PROOF. Suppose, to the contrary, that Problem 2 requires $o(\log \log n)$ time However, by using the algorithm of Proposition 4.1 in Section 4, the minimum in the matrix can be computed in O(1) further time. This contradicts Lemma 3.2. □

LEMMA 3.5. *Every algorithm that solves Problem 3 requires* $\Omega\left(\sqrt{\log \log n}\right)$ *time.*

PROOF. Since there is an algorithm that solves Problem 3 in $O(\log \log n)$ time (see Section 5), we can assume that the upper bound for Problem 3 is $O(\log \log n)$. Assume that a row-minima algorithm spent $t - 1$ time and has found no row-minima so far and, now, it is about to execute Phase 3 of time unit t, where $t < \epsilon \log \log n$ for some small fixed $\epsilon > 0$.

Proceeding as in the proof of Lemma 3.2, we see that at most $17n^2t^2$ pairs can be compared in Phase 3 of time unit t. Now a simple counting argument guarantees that at most $n^{1-1/4^t}$ rows have been assigned at least $17n^2i^2 \big/ \left(n^{1-1/4^i}\right) = 17n^{1+1/4^t}i^2$ comparisons each in time unit i $(1 \le i \le t)$. Hence, at time i, at least $n - in^{1-1/4^t}$ rows have been assigned at most $17n^{1+1/4^t}i^2$ comparisons. Assume that the topmost row was assigned at most $17n^{1+1/4^t}i^2$ comparisons in each time unit i $(1 \le i \le t)$, and let c_i be the number of candidates in the top row at the end of Phase 3 of time unit t.

$$\begin{aligned} c_i &\ge c_{i-1}^{\,2} \Big/ \left(2 \cdot 17n^{1+1/4^t}i^2 + c_{i-1}\right) \\ &\ge c_{i-1}^{\,2} \Big/ \left(35t^2n^{1+1/4^t}i^2\right) \qquad \left(\text{from } c_{i-1} \le n\right). \end{aligned}$$

By applying the logarithm, we have

$$\begin{aligned} \log c_i &\ge 2\log c_{i-1} - \left(\log 35 + 2\log i + \left(1 + 1/4^t\right)\log n\right) \\ &\ge \left(1 - 2^t/4^t\right)\log n - \sum_{j=1}^{i}\left(2^{j-1}\log 35 + 2^{i-j}\log j\right) \\ &\ge \left(1 - 1/2^t\right)\log n - 2^i\left(\log 35 + \log i\right). \end{aligned}$$

Hence, for some small fixed $\epsilon > 0$, $c_{\epsilon \log \log n} > 1$ for large n. Therefore, at least $n - tn^{1-1/4^t}$ rows, including the topmost row, cannot find the row-minima in Phase 3 of time unit t. Consequently, at most $tn^{1-1/4^t}$ rows can find the row-minima in Phase 3 of time unit t. In turn, this implies that there exist $n \Big/ \left(tn^{1-1/4^t}\right) = n^{1/4^t}\big/t$ consecutive rows that cannot find the row-minima in Phase 3 of time t. Therefore, we can find a submatrix of size $n^{1/4^t}\big/t \times n^{1/4^t}\big/t$ such that all of the $n^{1/4^t}$ row-minima are in it but no row-minima is found. Let $d_t \times d_t$ be the size of submatrix such that all d_t row-minima are in it but no row-minima is found at time t. Then, $d_t \ge d_{t-1}^{\,1/4^t}\big/t$. In addition, for large t, $d_t \ge d_{t-1}^{\,1/4^t}\big/t \ge d_{t-1}^{\,1/8^t}$ holds. Thus, for large t, we have: $d_t \ge d_{t-1}^{\,1/8^t}$. By applying the logarithm twice, we can write

$$\begin{aligned} \log\log d_t &\ge \log\log d_{t-1} - 3t \\ &\ge \log\log n - \sum_{i=1}^{t} 3i \\ &\ge \log\log n - 3t^2. \end{aligned}$$

Hence, in order to have $d_T \le 1$, it must be the case that $T \in \Omega\left(\sqrt{\log \log n}\right)$ and the proof is complete. □

4 PRELIMINARIES

Data movement operations are central to many efficient algorithms for parallel machines constructed as interconnection networks of processors. The purpose of this section is to review a number of basic data movement techniques for basic reconfigurable meshes.

Consider a sequence of n items $a_1, a_2, \ldots, a_n$. We are interested in computing the prefix maxima $z_1, z_2, \ldots, z_n$, defined for every j, $(1 \le j \le n)$, by setting $z_j = \max\{a_1, a_2, \ldots, a_j\}$. Recently, Olariu et al. [23] showed that the task of computing the prefix maxima of a sequence of n numbers stored in the

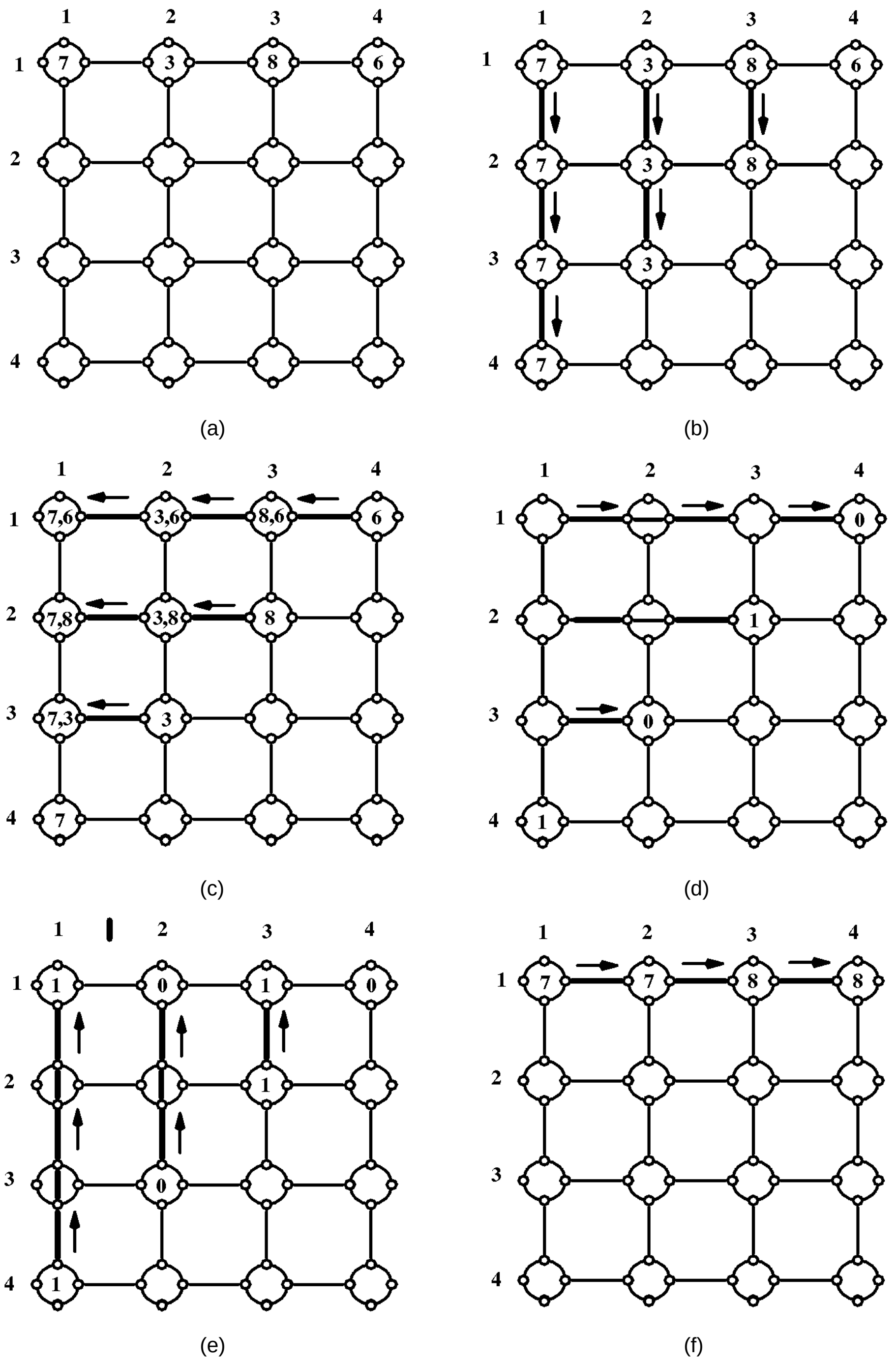


Fig. 3. Illustrating algorithm Prefix-Maxima-1.

first row of a reconfigurable mesh of size $m \times n$ can be solved in O(log n) time if $m = 1$, and in $O\left(\frac{\log n}{\log m}\right)$ time if $2 \le m \le n$. Since their algorithm is crucial for understanding our algorithm for computing the row minima of a monotone matrix, we now present an adaptation of the algorithm in [23] for the BRM.

To begin, we exhibit an O(1) time algorithm for computing the prefix maxima of n items on a BRM of size $n \times n$. The idea of this first algorithm involves checking, for all j ($1 \le j \le n$), whether a_j is the maximum of $a_1, a_2, \ldots, a_j$. The details are spelled out by the following sequence of steps. The reader is referred to Figs. 3a, 3b, 3c, 3d, 3e, 3f, where the algorithm is illustrated on the input sequence 7, 3, 8, 6.

Algorithm Prefix-Maxima-1;

STEP 1. Establish a vertical bus in every column j ($1 \le j \le n - 1$) from $P(1, j)$ to $P(n + 1 - j, j)$; every processor $P(1, j)$ ($1 \le j \le n - 1$) broadcasts the item a_j southbound along the vertical bus in column j;

STEP 2. Establish a horizontal bus in every row i ($1 \le i \le n - 1$) from $P(i, n + 1 - i)$ to $P(i, 1)$; every processor $P(i, n + 1 - i)$ ($1 \le i \le n - 1$) broadcasts the item a_{n+1-i} westbound along the horizontal bus in row i;

STEP 3. At the end of Step 2, every processor $P(i, j)$ ($i + j \le n + 1$) stores the items a_{n+1-i} and a_j; every processor $P(i, j)$

$(i + j \le n + 1)$ sets a local variable $b_{i,j}$ as follows:

$$b_{i,j} = \begin{cases} 0 & \text{if } a_{n+1-i} < a_j \\ 1 & \text{otherwise.} \end{cases}$$

STEP 4. Every processor $P(i, j)$ $(i + j \le n)$ with $b_{i,j} = 1$ connects its ports E and W; every processor $P(i, j)$ $(i + j \le n)$ with $b_{i,j} = 0$ broadcasts a 0 eastbound; every processor $P(i, n + 1 - i)$ $(1 \le i \le n - 1)$ that receives a 0 from its W port sets $b_{i,n+1-i}$ to 0;

STEP 5. Every processor $P(i, j)$ $(i + j \le n)$ connects its ports N and S; every processor $P(n + 1 - i, i)$ $(1 \le i \le n - 1)$ broadcasts $b_{n+1-i,i}$ northbound on the bus in column i; every processor $P(1, i)$ $(1 \le i \le n - 1)$ copies the value received into $b_{1,i}$;

STEP 6. Every processor $P(1, i)$,$(1 \le i \le n)$ with $b_{1,i} = 1$ sets z_i to a_i; every processor $P(1, i)$ $(1 \le i \le n - 1)$ with $b_{1,i} = 0$ connects its ports E and W; every processor $P(1, i)$ $(1 \le i \le n - 1)$ with $b_{1,i} = 1$ broadcasts a_i eastbound; every processor $P(1, i)$ $(1 \le i \le n - 1)$ with $b_{1,i} = 0$ sets z_i to the value received from its port W.

The correctness of the algorithm above is easily seen. Thus, we have the following result.

PROPOSITION 4.1. *The prefix maxima of n items from a totally ordered universe stored one item per processor in the first row of a basic reconfigurable mesh of size $n \times n$ can be computed in $O(1)$ time.*

Next, following [23], we briefly sketch the idea involved in computing the prefix maxima of n items $a_1, a_2, \ldots, a_n$ on a BRM of size $m \times n$ with $2 \le m \le n$. Begin by partitioning the original mesh into submeshes of size $m \times m$, and apply Prefix-Maxima-1 to each such submesh of size $m \times m$.

We further combine groups of m *consecutive submeshes of size* $m \times m$ into a submesh of size $m \times m^2$, then combine groups of m consecutive submeshes of size $m \times m^2$ into a submesh of size $m \times m^3$, and so on. Note that if the prefix maxima of a group of m consecutive submeshes are known, then the prefix maxima of their combination can be computed essentially as in Prefix-Maxima-1. For details, we refer the reader to [23].

To summarize the above discussion, we state the following result:

PROPOSITION 4.2. *The prefix maxima of n items from a totally ordered universe stored in one row of a basic reconfigurable mesh of size $m \times n$ with $2 \le m \le n$ can be computed in $O\left(\frac{\log n}{\log m}\right)$ time.*

Proposition 4.2 has the following important consequence.

PROPOSITION 4.3. *Let ϵ be an arbitrary constant in the range $0 < \epsilon \le 1$. The prefix maxima of n items from a totally ordered universe stored one item per processor in the first row of a basic reconfigurable mesh of size $n^\epsilon \times n$ can be computed in* O(1) *time.*

For later reference we now solve a particular instance of the row-minima problem, that we call the selective row minima problem. Consider an arbitrary matrix A of size $K \times N$ stored, one item per processor, in K consecutive rows of a BRM of size $M \times N$. For simplicity of exposition, we assume

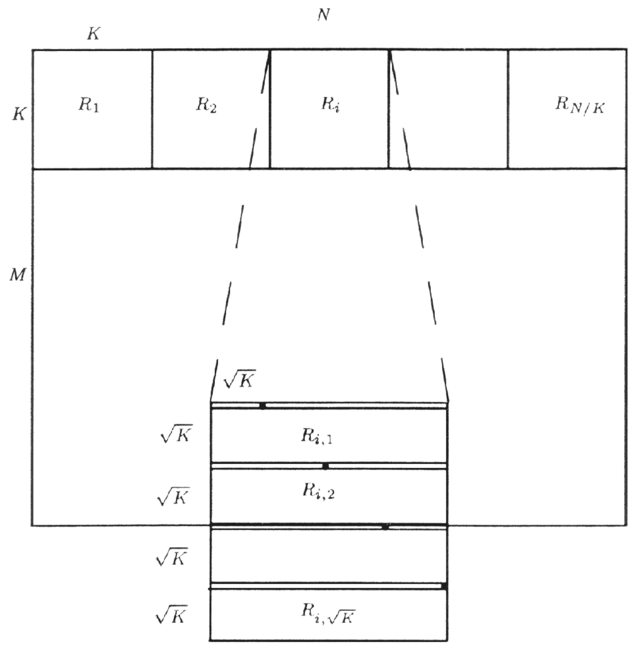


Fig. 4. Illustrating algorithm Selective-Row-Minima.

that A is stored in the first K rows of the platform, but this is not essential. The goal is to compute the minima in rows $1, \sqrt{K}+1, 2\sqrt{K}+1, \ldots, K-\sqrt{K}+1$ of A. We proceed as follows:

Algorithm Selective-Row-Minima;

STEP 1. Partition the BMR into N/K submeshes R_1, R_2, …, $R_{N/K}$ each of size $K \times K$ as illustrated in Fig. 4; further partition each submesh R_i $(1 \le i \le N/K)$ into submeshes $R_{i,1}, R_{i,2}, \ldots, R_{i,\sqrt{k}}$ each of size $\sqrt{K} \times K$;

STEP 2. Compute the minimum in the first row of each submesh $R_{i,j}$ in O(1) time using Proposition 4.3; let $a_{i,1}, a_{i,2}, \ldots, a_{i,\sqrt{K}}$ be the minima in the first row of $R_{i,1}, R_{i,2}, \ldots, R_{i,\sqrt{K}}$, respectively; by using appropriately established horizontal buses, we arrange for every $a_{i,j}$ $\left(1 \le j \le \sqrt{K}\right)$ to be moved to the processor in the first row and $j\sqrt{K}$th column of $R_{i,j}$;

STEP 3. We now perceive the original BRM of size $M \times N$ as consisting of $\sqrt{K}$ submeshes $T_1, T_2, \ldots, T_{\sqrt{K}}$ each of size $\frac{M}{\sqrt{K}} \times N$; the goal now becomes to compute for every i, $\left(1 \le i \le \sqrt{K}\right)$, the minimum of row $(i-1)\sqrt{K}+1$ of A in T_i; it is easy to see that, after having established vertical buses in all columns of the BRM, all the partial minima in row $(i-1)\sqrt{K}+1$ $\left(2 \le i \le \sqrt{K}\right)$ of A can be broadcast southbound to the first row of T_i;

STEP 4. Using the algorithm of Proposition 4.2, compute the minimum in the first row of each T_i $\left(1 \le i \le \sqrt{K}\right)$ in $O\left(\frac{\log N - \log K}{\log M - \log\sqrt{K}}\right) = O\left(\frac{\log N}{\log M}\right)$ time.

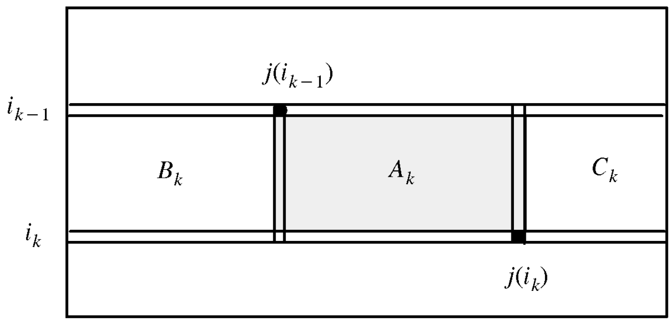


Fig. 5. Illustrating the proof of Lemma 5.1.

Thus, we have proven the following result.

LEMMA 4.4. *The task of computing the minima in rows* $1, \sqrt{K}+1, 2\sqrt{K}+1, \ldots, K-\sqrt{K}+1$ *of an arbitrary matrix of size* $K \times N$ *stored one item per processor in K rows of a BRM of size* $M \times N$ *can be performed in* $O\left(\frac{\log N}{\log M}\right)$ *time.*

5 THE ALGORITHM

The goal of this section is to present the details of an efficient algorithm for computing the row-minima of an $m \times n$ monotone matrix A. The matrix is assumed pretiled one item per processor onto a BRM $\mathcal{R}$ of the same size, such that for every i, j $(1 \le i \le m;\ 1 \le j \le n)$ processor $P(i, j)$ stores $A(i, j)$.

We begin by stating a few technical results that will come in handy later on. To begin, consider a subset $i_1, i_2, \ldots, i_p$ of the rows of A and let $j(i_1), j(i_2), \ldots, j(i_p)$ be such that, for all k $(1 \le k \le p)$ $A(i_k, j(i_k))$ is the minimum in row r_k. Since the matrix A is monotone, we must have

$$j(i_1) \le j(i_2) \le \ldots \le j(i_p).$$

Let $A_1, A_2, \ldots, A_p$ be the submatrices of A defined as follows:

- A_1 consists of the intersection of the first $i_1 - 1$ rows with the first $j(i_1)$ columns of A;
- for every k $(2 \le k \le p-1)$ A_k consists of the intersection of rows $i_{k-1}+1$ through $i_k - 1$ with the columns $j(i_{k\cdot 1})$ through $j(i_k)$;
- A_p consists of the intersection of rows $i_p + 1$ through m with the columns $j(i_p)$ through n.

The following result will be used again and again in the remainder of this section.

LEMMA 5.1. *Every matrix* A_k $(1 \le k \le p)$ *is monotone.*

PROOF. First, let k be an arbitrary subscript with $2 \le k \le p$. and refer to Fig. 5. Let B_k consist of the submatrix of A consisting of the intersection of rows $i_{k-1} + 1$ through $i_k - 1$ with columns $j(i_{k-1})$ through $j(i_k)$. Similarly, let C_k be the submatrix of A consisting of the intersection of rows $i_{k-1}+1$ through $i_k - 1$ with columns $j(i_{k-1})$ through $j(i_k)$.

Since the matrix A is monotone and since $A(i_{k-1}, j(i_{k-1}))$ is the minimum in row i_{k-1}, it follows that none of the minima in rows $i_{k-1} + 1$ through $i_k - 1$ can occur in the submatrix B_k. Similarly, since $A(i_k, j(i_k))$ is the minimum in row i_k, no minima in rows $i_{k-1} + 1$ through $i_k - 1$ can occur in the submatrix C_k. It follows that the minima in rows $i_{k-1} + 1$ through $i_k - 1$ must occur in the submatrix A_k. Consequently, if A_k is not monotone, then we violate the monotonicity of A.

A perfectly similar argument shows that A_1 and A_p are also monotone, completing the proof of the lemma. □

The matrices A_k $(1 \le k \le p)$ defined above pairwise share a column. The following technical result shows that one can always transform these matrices such that they involve distinct columns. For this purpose, consider the matrix A'_k obtained from A_k by replacing for every i, $(i_{k-1} + 1 \le i \le i_k - 1)$, entry $A_k(i, j(i_{k-1}))$ of A_k with $\min\{A_k(i, j(i_{k-1})), A_k(i, j(i_{k-1}) + 1)\}$ and by dropping column $j(i_k)$. In other words, A'_k is obtained from A_k by retaining the minimum values in its first and next column and then removing the last column. The last matrix A'_p is taken to be A_p. The following result, whose proof is omitted, will be used implicitly in our algorithm.

LEMMA 5.2. *Every matrix* A'_k $(1 \le k \le p)$ is monotone.

In outline, our algorithm for computing the row-minima of a monotone matrix proceeds as follows. First, we solve an instance of the selective row minima whose result is used to partition the original matrix into a number of monotone matrices, as described in Lemmas 5.1 and 5.2. This process is continued until the row minima in each of the resulting matrices can be solved directly. If $m = 1$, then the problem has a trivial solution running in $\Theta(\log n)$ time, which is also best possible even on the more powerful reconfigurable mesh [23].

We shall, therefore, assume that $m \ge 2$.

Algorithm Row-Minima(A);

STEP 1. Partition $\mathcal{R}$ into $\sqrt{m}$ submeshes $T_1, T_2, \ldots, T_{\sqrt{m}}$ each of size $\sqrt{m} \times n$ such that, for every i, $\left(1 \le i \le \sqrt{m}\right)$, T_i involves rows $(i-1)\sqrt{m}+1$ through $i\sqrt{m}$ of $\mathcal{R}$, as illustrated in Fig. 6;

STEP 2. Using the algorithm of Lemma 4.4 compute the minima of the items in the first row of every submesh T_i $\left(1 \le i \le \sqrt{m}\right)$ in $O\left(\frac{\log n}{\log m}\right)$ time;

STEP 3. Let $c_1, c_2, \ldots, c_{\sqrt{m}}$ be the columns of $\mathcal{R}$ containing the minima in $T_1, T_2, \ldots, T_{\sqrt{m}}$, respectively, computed in Step 2. The monotonicity of A guarantees that $c_1 \le c_2 \le \ldots \le c_{\sqrt{m}}$. Let R_i $\left(1 \le i \le \sqrt{m}\right)$ be the submesh of $\mathcal{R}$ consisting of all the processors $P(r, c)$ such that $(i-1)\sqrt{m}+2 \le r \le i\sqrt{m}$ and $c_i \le c \le c_{i+1}$. In other words, R_i consists of the intersection of rows $(i-1)\sqrt{m}+2, (i-1)\sqrt{m}+3, \ldots, i\sqrt{m}$ with columns c_i, $c_i + 1$, $c_i + 2$, $\ldots$, c_{i+1} as illustrated in Fig. 6;

STEP 4. Partition the mesh $\mathcal{R}$ into submeshes $S_1, S_2, \ldots, S_{\sqrt{m}}$ with S_i of size $m \times (c_{i+1} - c_i)$, as illustrated in Fig. 7; for

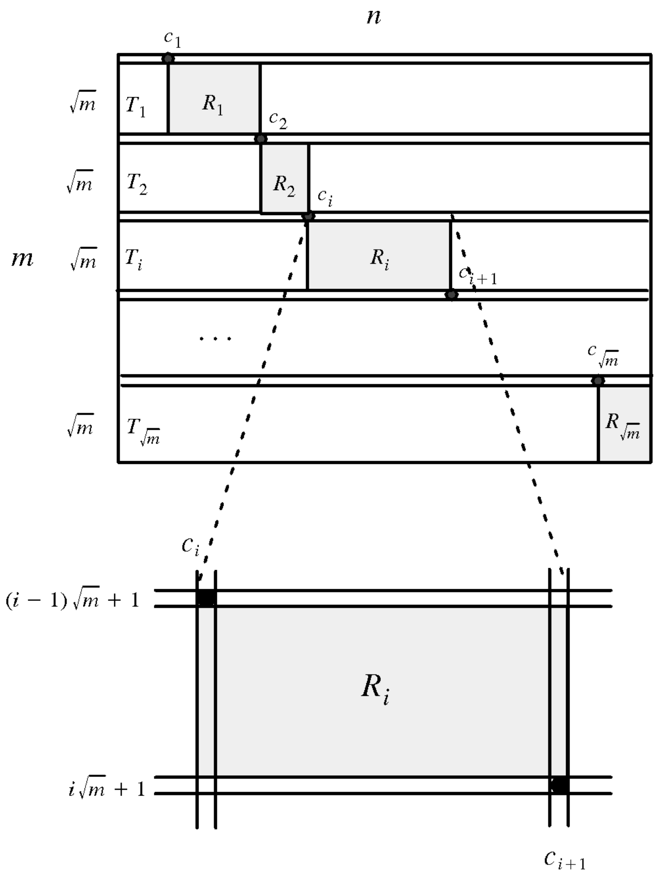


Fig. 6. Illustrating the partition into submeshes T_i and R_i.

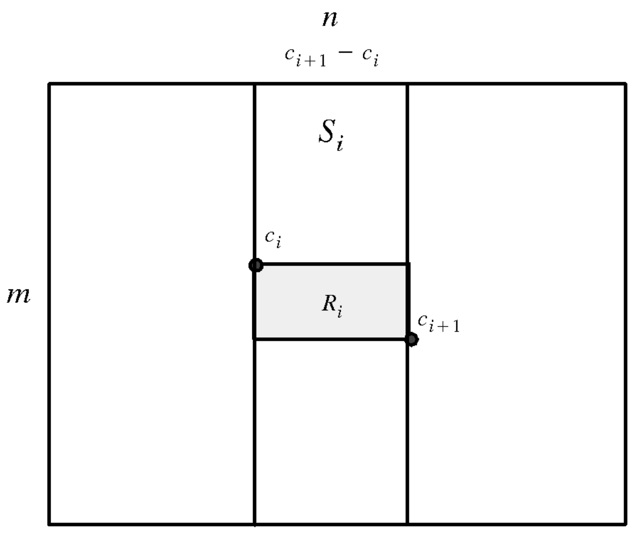


Fig. 7. Illustrating the submeshes S_i.

log log m iterations, repeat Steps 1-3 above in each submesh S_i.

The correctness of the algorithm being easy to see, we now turn to the complexity. Steps 1-3 have a combined complexity of $O\left(\frac{\log n}{\log m}\right)$. In Step 4, $c_{i+1} - c_i \leq n$ and so, by Lemma 4.4, each iteration of Step 4 also runs in $O\left(\frac{\log n}{\log m}\right)$ time. Since there are, essentially, log log m such iterations, the overall complexity of the algorithm is $O\left(\frac{\log n}{\log m}\log\log m\right)$. To summarize our findings, we state the following result:

THEOREM 5.3. *The task of computing the row-minima of a monotone matrix of size $m \times n$ with $1 \leq m \leq n$ pretiled one item per processor in a BRM of the same size can be solved in* $O(\log n)$ *time if* $m = 1, 2$, *and in* $O\left(\frac{\log n}{\log m}\log\log m\right)$ *time if* $m > 2$.

By writing $m = n^{\epsilon}$ for some constant ϵ, $0 < \epsilon \leq 1$, Theorem 5.3 yields the following result.

COROLLARY 5.4. *For every constant* $0 < \epsilon \leq 1$, *the task of computing the row-minima of a monotone matrix of size $m \times n$ with $m = n^{\epsilon}$, pretiled one item per processor in a BRM of the same size, can be solved in* $O(\log\log n)$ *time.*

6 Conclusions and Open Problems

We have shown that the problem of computing the row-minima of a monotone matrix can be solved efficiently on the basic reconfigurable mesh (BRM)—a weaker variant of the recently proposed Polymorphic Processor Array [16].

Specifically, we have exhibited an algorithm that, with a monotone matrix A of size $m \times n$ ($1 \leq m \leq n$) stored in a BRM of the same size, as input solves the row-minima problem in O(log n) time in case $m \in O(1)$, and in $O\left(\frac{\log n}{\log m}\log\log m\right)$ time, otherwise. In particular, if $m = n^{\epsilon}$ for some fixed constant ϵ, ($0 < \epsilon \leq 1$), our algorithm runs in O(log log n) time.

One of our main contributions was to propose a number of nontrivial time lower bounds for matrix search problems. These lower bounds hold for general two-dimensional reconfigurable meshes of infinite size, as long as the input is pretiled onto an $n \times n$ submesh thereof. Specifically, in this context we show that every algorithm that solves the problem of computing the smallest item of an $n \times n$ matrix, or the smallest item in each row of an $n \times n$ matrix, must take $\Omega(\log\log n)$ time. This result implies an $\Omega(\log\log n)$ time lower bound for the problem of selecting the kth smallest item in a *monotone* matrix, extending the result of [10] in two directions: We show that the same lower bound applies to selection on monotone matrices and on a reconfigurable mesh of an infinite size. Finally, we showed an $\Omega\left(\sqrt{\log\log n}\right)$ time lower bound for the task of computing the row minima of a *monotone* $n \times n$ matrix. These are the first nontrivial lower bounds of this kind known to the authors.

A number of problems remain open. First, there is a discrepancy between the time lower bound we obtained for the task of computing the row-minima of a monotone matrix and the upper bound provided by our algorithm. Narrowing this gap will be a hard problem that we leave for future research. Second, no nontrivial lower bounds for the problem of computing the row-minima of a totally monotone matrix are known to us. This promises to be an exciting area for future research. Yet another problem of interest would be to solve the row-minima problem for the special case of totally monotone matrices. Trivially, our algorithm for monotone matrices also works for totally monotone ones. Unfortunately, to date, we have not been able to find a nontrivial lower bound for this problem.

ACKNOWLEDGMENT

The authors wish to thank Mike Atallah for many useful comments and for pointing out a number of relevant references. This work was supported in part by U.S. National Science Foundation Grants CCR-9407180 and CCR-9522093, and by ONR Grant N00014-97-1-0526.

REFERENCES

[1] A. Aggarwal and M.M. Klawe," Applications of Generalized Matrix Searching to Geometric Problems," *Discrete Applied Mathematics*, vol. 27, pp. 3–23, 1990.

[2] A. Aggarwal, M.M. Klawe, S. Moran, P. Shor, and R. Wilber, "Geometric Applications of a Matrix-Searching Algorithm," *Algorithmica*, vol. 2, pp. 195–208, 1987.

[3] A. Aggarwal and J. Park, "Notes on Searching in Multidimensional Monotone Arrays," *Proc. 29th Ann. Symp. Foundations of Computer Science*, pp. 497–512, Oct. 1988.

[4] A. Apostolico, M.J. Atallah, L.L. Larmore, and S. McFaddin, "Efficient Parallel Algorithms for String Editing and Related Problems," *SIAM J. Computing*, vol. 19, pp. 968–988, 1990.

[5] M.J. Atallah, "A Faster Parallel Algorithm for a Matrix Searching Problem," *Algorithmica*, vol. 9, pp. 156–167, 1993.

[6] M.J. Atallah and S.R. Kosaraju, "An Efficient Parallel Algorithm for the Row Minima of a Totally Monotone Matrix," *J. Algorithms*, vol. 13, pp. 394–413, 1992.

[7] Y. Ben-Asher, D. Peleg, R. Ramaswani, and A. Schuster, "The Power of Reconfiguration," *J. Parallel and Distributed Computing*, vol. 13, pp. 139–153, 1991.

[8] C. Berge, *Graphs and Hypergraphs*, pp. 280–282. Amsterdam: North-Holland, 1973.

[9] R.O. Duda and P.E. Hart, *Pattern Classification and Scene Analysis*. New York: Wiley and Sons, 1973.

[10] E. Hao, P.D. Mackenzie, and Q.F. Stout, "Selection on the Reconfigurable Mesh," *Proc. Fourth Symp. Frontiers of Massively Parallel Computation*, pp. 38–45, McLean, Va., Oct. 1992.

[11] J. JáJá, *An Introduction to Parallel Algorithms*, pp. 188–189. Reading, Mass.: Addison Wesley, 1992.

[12] D. Kim and K. Hwang, "Mesh-Connected Array Processors with Bypass Capability for Signal/Image Processing," *Proc. Hawaii Conf. System Sciences*, Maui, Hawaii, Jan. 1988.

[13] H. Li and M. Maresca, "Polymorphic-Torus Network," *IEEE Trans. Computers*, vol. 38, pp. 1,345–1,351, 1989.

[14] H. Li and M. Maresca, "Polymorphic-Torus Architecture for Computer Vision," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 11, pp. 233–243, 1989.

[15] R. Lin and S. Olariu, "Reconfigurable Buses with Shift Switching—Concepts and Applications," *IEEE Trans. Parallel and Distributed Systems*, vol. 6, pp. 93–102, 1995.

[16] M. Maresca, "Polymorphic Processor Arrays," *IEEE Trans. Parallel and Distributed Systems*, vol. 4, pp. 490–506, 1993.

[17] M. Maresca and H. Li, "Connection Autonomy in SIMD Computers: A VLSI Implementation," *J. Parallel and Distributed Computing*, vol. 7, pp. 302–320, 1989.

[18] M. Maresca and H. Li, "Virtual Parallelism Support in Reconfigurable Processor Arrays," UCB–ICSI Technical Report 91–041, Univ. of California at Berkeley, July 1991.

[19] M. Maresca and H. Li, "Hierarchical Node Clustering in Polymorphic Processor Arrays," UCB–ICSI Technical Report 91–042, Univ. of California at Berkeley, July 1991.

[20] M. Maresca, H. Li, and P. Baglietto, "Hardware Support for Fast Reconfigurability in Processor Arrays," *Proc. Int'l Conf. Parallel Processing*, vol. I, pp. 282–289, St. Charles, Ill., 1993.

[21] R. Miller, V.K.P. Kumar, D. Reisis, and Q.F. Stout, "Parallel Computations on Reconfigurable Meshes," *IEEE Trans. Computers*, vol. 42, pp. 678–692, 1993.

[22] K. Nakano, "A Bibliography of Published Papers on Dynamically Reconfigurable Architectures," *Parallel Processing Letters*, vol. 5, pp. 111-124, 1995.

[23] S. Olariu, J.L. Schwing, and J. Zhang, "Fundamental Data Movement on Reconfigurable Meshes," *Int'l J. High Speed Computing*, vol. 6, pp. 311–323, 1994.

[24] S. Olariu, J.L. Schwing, and J. Zhang, "Fundamental Algorithms on Reconfigurable Meshes," *Proc. 29th Ann. Allerton Conf. Comm., Control, and Computing*, pp. 811–820, 1991.

[25] J. Rothstein, "On the Ultimate Limitations of Parallel Processing," *Proc. Int'l Conf. Parallel Processing*, pp. 206–212, 1976.

[26] J. Rothstein, "Bus Automata, Brains, and Mental Models," *IEEE Trans. Systems, Man, and Cybernetics*, vol. 18, pp. 522–531, 1988.

[27] D.B. Shu, L.W. Chow, and J.G. Nash, "A Content Addressable, Bit Serial Associate Processor," *Proc. IEEE Workshop VLSI Signal Processing*, Monterey, Calif., Nov. 1988.

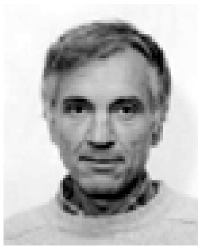
[28] D.B. Shu and J.G. Nash, "The Gated Interconnection Network for Dynamic Programming," *Concurrent Computations*, S.K. Tewsburg et al., eds. Plenum Publishing, 1988.

[29] L.G. Valiant, "Parallelism in Comparison Problems," *SIAM J. Computing*, vol. 4, pp. 348-355, 1975.

[30] B.F. Wang and G.H. Chen, "Constant Time Algorithms for the Transitive Closure Problem and Its Applications," *IEEE Trans. Parallel and Distributed Systems*, vol. 1, pp. 500–507, 1990.

[31] C.C. Weems, S.P. Levitan, A.R. Hanson, E.M. Riseman, J.G. Nash, and D.B. Sheu, "The Image Understanding Architecture," *Int'l J. Computer Vision*, vol. 2, pp. 251–282, 1989.

Koji Nakano received the BE, ME, and PhD degrees from Osaka University, Japan, in 1987, 1989, and 1992, respectively. In 1992-1995, he was a research scientist at Advanced Research Laboratory, Hitachi Ltd. Since 1995, he has worked at Nagoya Institute of Technology, Japan. He is currently an associate professor with the Department of Electrical and Computer Engineering. His research interests include parallel algorithms and architectures, computational complexity, and graph theory.



Stephan Olariu received the MSc and PhD degrees in computer science from McGill University, Montreal, in 1983 and 1986, respectively. In 1986, he joined Old Dominion University, where he is now a professor of Computer Science.

Dr. Olariu has published extensively in various journals, book chapters, and conference proceedings. His research interests include image processing and machine vision, parallel architectures, design and analysis of parallel algorithms, computational graph theory, computational geometry, and mobile computing.

Dr. Olariu is an associate editor of the *International Journal of Computer Mathematics* and serves on the editorial boards of the *Journal of Parallel and Distributed Computing*, *VLSI Design*, and *Parallel Algorithms and Applications*.