アルゴリズム 53－3
（1996．9．13）

基本再構成メッシュ上の行最小値計算のための効率よいアルゴリズム

中野浩嗣　名古屋工業大学 電気情報工学科
ステファン・オラリウ　オールドドミニオン大学 計算機科学科

大きさ $m \times n$ の行列 A の任意の行 i と行 j $(1 \leq i < j \leq n)$ について、行 j の最小値が行 i 最小値の右側にあるとき、行列 A を単調であると呼ぶ。本論文では、大きさ $m \times n$ $(1 \leq m \leq n)$ の行列が同じ大きさの基本再構成メッシュに与えられたときに、その行ごとの最小値を求める $O(\log n)$ 時間 $(m = 1, 2$ のとき$)$ と $O(\frac{\log n}{\log m} \log\log m)$ 時間 $(m > 2$ のとき$)$ のアルゴリズムを示す。さらに、この問題と関連する行列探索問題に対する自明でない計算時間の下界を示す。

An Efficient Algorithm for Row Minima Computations on Basic Reconfigurable Meshes[†]

Koji Nakano
Dept. of Electrical and Computer Engineering
Nagoya Institute of Technology
Showa-ku, Nagoya 466, Japan
nakano@elcom.nitech.ac.jp

Stephan Olariu
Department of Computer Science
Old Dominion University
Norfolk, Virginia 23529, USA
olariu@cs.odu.edu

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. Our first main contribution is to show that the task of computing the row minima of an $m \times n$ monotone matrix, $1 \leq m \leq n$, pretiled onto a Basic Reconfigurable Mesh of the same size can be performed in $O(\log n)$ time if $m = 1, 2$ and in $O(\frac{\log n}{\log m} \log\log m)$ time if $m > 2$. Our second contribution is to exhibit a number of non-trivial lower bounds for matrix search problems.

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 in response to specific processing needs; more recently, researches have proposed augmenting the mesh architecture with reconfigurable broadcasting buses: these are high-speed buses whose configuration can be dynamically changed to accommodate various computational requirements. Among these, 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 [11, 16] 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 [8]. Worse yet, it has been recently shown that the reconfigurable mesh and the PARBS do not scale and, as a consequence, do not immediately support virtual parallelism [9, 10].

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 [8]: 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 [8] for the PPA also applies to the BRM. In particular, all the broadcast primitives developed in [8], 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 primitives of [8]. We have 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, $\leq i < j \leq m$, the smallest value in row j lies below or to the right of the smallest value in row i, as illustrated in the figure below, where the row minima are highlighted. A matrix is said to be *totally monotone* if every 2×2 minor thereof 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

[†]Work supported in part by NSF grant CCR-8909996, and by ONR grant N00014-95-1-0779

recognition, and computational morphology, among many others. The reader is referred to [1, 2, 3, 4, 5, 6] where many of these applications are discussed in detail.

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

One of the recurring problem involving matrix searching is referred to as *row-minima computation* [6]. In particular, Aggarwal *et al.* [2] have shown 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 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. One of the main contributions of this paper is to propose a non-trivial time lower bound for the row-minima problem in the BRM model. Specifically, we show that every algorithm that solves the row minima problem of a monotone matrix of size $n \times n$ must take $\Omega(\sqrt{\log \log n})$ time on a BRM of size $n \times n$. At present, we do not know whether this lower bound is tight. 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 $m \leq n$ pretiled, one item per processor, onto a BRM of the same size, our row-minima algorithm runs in $O(\frac{\log n}{\log m} \log \log m)$ time. In case $m = n^\epsilon$ for some constant ϵ, $(0 < \epsilon \leq 1)$, our algorithm runs in $O(\log \log n)$ time.

The remainder of this work is organized as follows: Section 2 introduces the model of computations adopted in this paper; Section 3 presents basic algorithms that will be key in our subsequent row-minima algorithm; Section 4 discusses a number of relevant lower-bound results; 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 4 ports N, S, E, and W, as illustrated in Figure 1. Local connections between these ports can be established, subject to the following restrictions. Specifically, in every time unit:

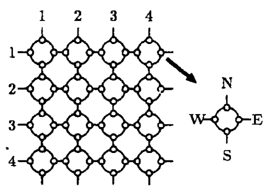

Figure 1: *A basic reconfigurable mesh of size 4 × 4*

1. 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*. In other words, 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.

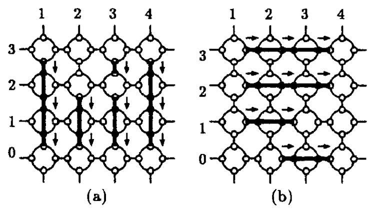

Figure 2: *Examples of unidirectional subbuses*

We refer the reader to Figure 2(a)-(b) 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 [8, 9, 10] Maresca *et al.* have demonstrated that the PPA architecture and the corresponding programming 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 workers 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 [8, 9].

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 [8], we assume ideal communications along buses (no delay). Although inexact, a series of recent experiments with the PPA [8] and the GCN [13, 14] seem to indicate that this is a reasonable working hypothesis.

3 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, $(1 \leq j \leq n)$, by setting $z_j = \max\{a_1, a_2, \ldots, a_j\}$. Recently Olariu *et al.* [12] showed that the task of computing the prefix maxima of a sequence of n numbers stored in the first row of a reconfigurable mesh of size $m \times n$ can be solved in $O(\log n)$ time if $m = 1$, and in $O(\frac{\log n}{\log m})$ time if $2 \leq m \leq n$. 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 [12] for basic reconfigurable meshes.

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, $(1 \leq j \leq n)$, 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 Figure 3(a)–(f) where the algorithm is illustrated on the input sequence 7, 3, 8, 6.

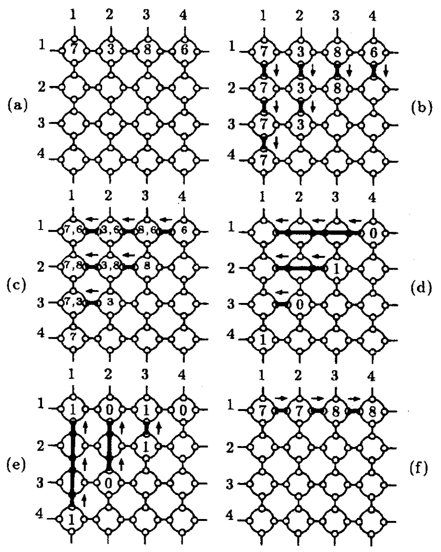

Figure 3: *Illustrating algorithm Prefix-Maxima-1*

Algorithm Prefix-Maxima-1;

Step 1. Establish a vertical bus in every column j, $(1 \leq j \leq n-1)$, from $P(1, j)$ to $P(n+1-j, j)$; every processor $P(1, j)$, $(1 \leq j \leq n-1)$, broadcasts the item a_j southbound along the vertical bus in column j, as illustrated in Figure 3(b);

Step 2. Establish a horizontal bus in every row i, $(1 \leq i \leq n-1)$, from $P(i, n+1-i)$ to $P(i, 1)$; every processor $P(i, n+1-i)$, $(1 \leq i \leq n-1)$, broadcasts the item a_{n+1-i} westbound along the horizontal bus in row i (see Figure 3(c));

Step 3. At the end of Step 2, every processor $P(i, j)$, $(i + j \leq n)$, stores the items a_{n+1-i} and a_j; every processor $P(i, j)$, $(i + j \leq n + 1)$, sets a local variable b_{ij} as follows:

$$b_{ij} = \begin{cases} 0 & \text{if } i + j \leq n \text{ and } a_{n+1-i} < a_j \\ 1 & \text{otherwise;} \end{cases}$$

Step 4. Every processor $P(i, j)$, $(i + j \leq n)$, with $b_{ij} = 1$ connects its ports E and W; every processor $P(i, j)$, $(i + j \leq n)$, with $b_{ij} = 0$ broadcasts a 0 eastbound, as illustrated in Figure 3(d); every processor $P(i, n+1-i)$, $(1 \leq i \leq n-1)$, that receives a 0 from its W port sets b_{ii} to 0, as illustrated in Figure 3(d);

Step 5. Every processor $P(i, j)$, $(i + j \leq n)$, connects its ports N and S; every processor $P(n+1-i, i)$, $(1 \leq i \leq n-1)$, broadcasts b_{ii} northbound on the bus in column i; every processor $P(1, i)$, $(1 \leq i \leq n-1)$, copies the value received into b_{1i}, as shown in Figure 3(e);

Step 6. Every processor $P(1, i)$, $(1 \leq i \leq n)$, with $b_{1i} = 1$ sets z_i to a_i; every processor $P(1, i)$, $(1 \leq i \leq n-1)$, with $b_{1i} = 0$ connects its ports E and W; every processor $P(1, i)$, $(1 \leq i \leq n-1)$, with $b_{1i} = 1$ broadcasts a_i eastbound; every processor $P(1, i)$, $(1 \leq i \leq n-1)$, with $b_{1i} = 0$ 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 3.1 *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 [12], 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 $2 \leq m \leq n$. 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 continue until the original mesh is obtained. 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 [12].

To summarize the above discussion we state the following result.

Proposition 3.2 *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 $2 \leq m \leq n$ can be computed in $O(\frac{\log n}{\log m})$ time.* □

Proposition 3.2 has the following important consequence that will be used again and again in the remainder of this work.

Proposition 3.3 *Let ϵ be an arbitrary constant in the range $0 < \epsilon \leq 1$. 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.* □

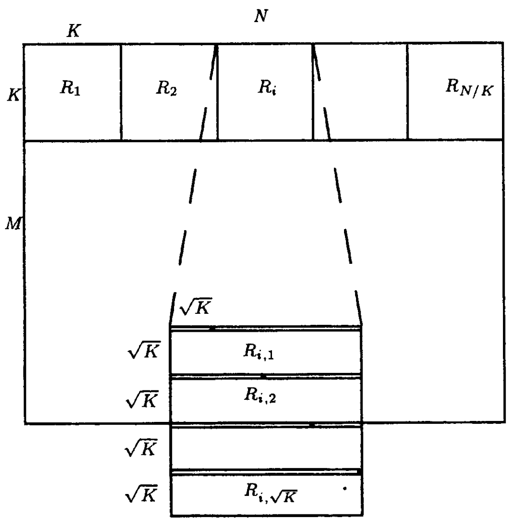

Figure 4: *Illustrating algorithm Selective-Row-Minima*

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 rows of a BRM of size $M \times N$. For simplicity of exposition we assume that A is stored in the first K rows of the platform, but this is non-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 $R_1, R_2, \ldots, R_{N/K}$ each of size $K \times K$; further partition each submesh R_i, $(1 \leq i \leq N/K)$, into submeshes $R_{i,1}, R_{i,2}, \ldots, R_{i,\sqrt{k}}$ each of size $\sqrt{K} \times K$, as aillustrated in Figure 4;
Step 2. Compute the minimum in the first row of each submesh $R_{i.j}$ in O(1) time using Proposition 3.3; 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 $a_{i,j}$, $(1 \leq j \leq \sqrt{K})$, 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 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, $(1 \leq i \leq \sqrt{K})$, 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 $(i-1)\sqrt{K}+1$, $(2 \leq i \leq \sqrt{K})$, of A can be broadcast southbound to the first row of T_i;
Step 4. Using the algorithm of Proposition 3.2 compute the minimum in the first row of each T_i, $(1 \leq i \leq \sqrt{K})$ in $O\left(\frac{\log N - \log K}{\log M - \log \sqrt{K}}\right)$ time.

Thus, we have proved the following result.

Lemma 3.4 *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 $O\left(\frac{\log N - \log K}{\log M - \log \sqrt{K}}\right)$ time.* □

If $K = M^\delta$ for some fixed constant δ, $(0 \leq \delta \leq 1)$, Lemma 3.4 implies the following result.

Corollary 3.5 *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 $M^\delta \times N$ stored one item per processor in M^δ rows of a BRM of size $M \times N$ can be performed in $O\left(\frac{\log N}{\log M}\right)$ time.* □

4 Lower bounds for matrix minima problems

This section shows non-trivial lower bounds for the several matrix minima problems. Since the proofs for the lower bounds do not use the restriction of a BRM, they hold for more powerful reconfigurable meshes that allow any local connections. Furthermore, the lower bounds hold for an $\infty \times \infty$ *reconfigurable mesh*, which has infinitely many processors in two dimensions. Formally, this section deals with the following problems:

Problem 1 Given an $n \times n$ matrix to an $n \times n$ submesh of an $\infty \times \infty$ reconfigurable mesh, find the minimum item of the matrix,

Problem 2 Given an $n \times n$ matrix to 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 to 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 to an $n \times n$ submesh of an $\infty \times \infty$ reconfigurable mesh, find the minimum item of each row.

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

The proofs are based on the techniques in [7, 15] that uses the following lemma to prove the lower bounds for computing the minimum.

Lemma 4.1 *For a graph $G = (V, E)$, $U \subseteq V$ is an independent set if no two vertices of U are joined with an edge. There exists an independent set U such that $|U| \geq |V|^2/(2|E| + |V|)$.*

This lemma is used to evaluate the number of candidates for the minimum. In other words, for a set V of candidates and a set E of pairs that are compared by a minimum finding algorithm, items in the independent set U have possibility to become the minimum. So, all items in U are still candidates for the minimum after comparing all pairs in E.

For simplicity we assume processors on a reconfigurable mesh execute the following three phases in a unit of time:

Phase 1 reconfigure bus,

Phase 2 send at most a piece of data to each port, and receive a piece of data from each port,

Phase 3 select two elements stored in the local memory, compare them and change internal status.

We first prove the following lemma as a preliminary.

Lemma 4.2 *Problem 1 requires* $\Omega(\log\log n)$ *time.*

Proof. We will evaluate the number of pairs that can be compared by an algorithm at Phase 3 of time t. In each unit of time, at most $4n$ item can be sent to the outside of the submesh at Phase 2. Hence, at most $4nt$ items can be sent before the execution of Phase 3 at time 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. Totally, at Phase 3 of time 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 t. Then, from Lemma 4.1 we have,

$$\begin{aligned} c_t &\geq {c_{t-1}}^2/(2\cdot 17n^2t^2 + c_{t-1}) \\ &\geq {c_{t-1}}^2/(35n^2t^2). \qquad (\text{from } c_{t-1} \leq n^2) \end{aligned}$$

By applying logarithmic, we have

$$\begin{aligned} \log c_t &\geq 2\log c_{t-1} - (\log 35 + 2\log n + 2\log t) \\ &\geq 2\log n - \sum_{i=1}^{t}(2^{i-1}\log 35 + 2^{t-i}\log i) \\ &\qquad (\text{from } c_0 = n^2) \\ &\geq 2\log n - 2^t(\log 35 + \log t) \end{aligned}$$

To complete the algorithm at time T, c_T must be less than or equal to 1. Therefore, $2\log n \leq 2^T(\log 35 + \log T)$ must hold. Consequently, $T = \Omega(\log\log n)$ holds. □

Lemma 4.3 *Problem 2 requires* $\Omega(\log\log n)$ *time.*

Proof. We assume that Problem 2 can be solved in $o(\log\log n)$ time, and show a contradiction. After solving Problem 2, the minima of all rows are computed. Then, the minimum of them can be computed in $O(1)$ time. Therefore, Problem 1 can be solved in $o(\log\log n)$ time, a contradiction. □

Lemma 4.4 *Problem 3 requires* $\Omega(\sqrt{\log\log n})$ *time.*

Proof. Since there is an algorithm that solves Problem 3 in $O(\log\log n)$ time, we can assume that the lower bound for the Problem 3 is $O(\log\log n)$. Assume that a row-minima algorithm spent $t-1$ time and it have found no row-minima so far, and now try to execute the Phase 3 of time t where $t < \epsilon\log\log n$ for small fixed $\epsilon > 0$. At most $n^{1-1/4^t}$ rows have been assigned at least $17n^2i^2/(n^{1-1/4^t}) = 17n^{1+1/4^t}i^2$ comparisons each at time i $(1 \leq i \leq t)$. Hence, at time i, at least $n - in^{1-1/4^t}$ rows have been assigned at least $17n^{1+1/4^t}i^2$ comparisons. Assume that the topmost row assigned at most $17n^{1+1/4^t}i^2$ at every time i $(1 \leq i \leq t)$ and let c_i be the number of candidate in the top row after Phase 3 of time t.

$$\begin{aligned} c_i &\geq {c_{i-1}}^2/(2\cdot 17n^{1+1/4^t}i^2 + c_{i-1}) \\ &\geq {c_{i-1}}^2/(35t^2n^{1+1/4^t}i^2) \qquad (\text{from } c_{i-1} \leq n) \end{aligned}$$

By applying logarithmic, we have

$$\begin{aligned} &\log c_i \\ &\geq 2\log c_{i-1} - (\log 35 + 2\log i + (1 + 1/4^t)\log n) \\ &\geq (1 - 2^t/4^t)\log n - \sum_{j=1}^{i}(2^{j-1}\log 35 + 2^{i-j}\log j) \\ &\qquad (\text{from } c_0 = n) \\ &\geq (1 - 1/2^t)\log n - 2^i(\log 35 + \log i). \end{aligned}$$

Hence, for 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 at Phase 3 of time t. That is, at most $tn^{1-1/4^t}$ rows can find the row-minima at Phase 3 of time t. There exist $n/(tn^{1-1/4^t}) = n^{1/4^t}/t$ consecutive rows that cannot find the row-minima at Phase 3 of time t. From this fact, we can find $n^{1/4^t}/t \times n^{1/4^t}/t$ sub-matrix 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 sub-matrix such that all d_t row-minima are in it but no row-minima is found at time t. Then, $d_t \geq {d_{t-1}}^{1/4^t}/t$ $(\geq {d_{t-1}}^{1/8^t}$ for large $t)$ holds. By applying logarithmic twice,

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

Hence, $T = \Omega(\sqrt{\log\log n})$ holds to satisfy $d_T \leq 1$. □

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, $(1 \leq i \leq m;\ 1 \leq j \leq n)$, 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 $j(i_{k-1})$ 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 Figure 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 $j(i_k)$.

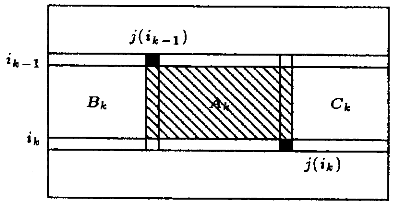

Figure 5: *Illustrating the proof of Lemma 5.1*

Since the matrix A is monotone and $A(i_{k-1}, j(i_{k-1}))$ 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 j, $(1 \le j \le i_k - i_{i-1} - 1)$, entry $A_k(i_{k-1}+j, j(i_{k-1}))$ of A_k with $\min\{A_k(i_{k-1}+j, j(i_{k-1})), A_k(i_{k-1}+j, j(i_k))\}$ 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 last 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 reflemma;disitnct. This process is continued untils 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 [12].

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

---

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, $(1 \le i \le \sqrt{m})$, T_i involves rows $(i-1)\sqrt{m}+1$ through $i\sqrt{m}$ of $\mathcal{R}$, as illustrated in Figure 6;

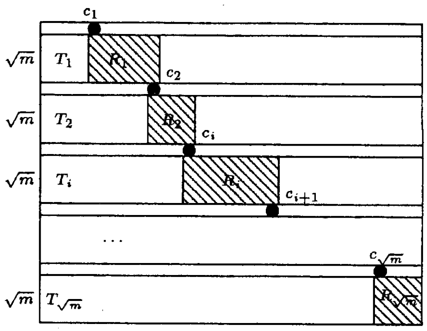

Figure 6: *Illustrating the partition into submeshes T_i and R_i*

Step 2. Using the algorithm of Lemma 3.4 compute the minima of the items in the first row of every submesh T_i, $(1 \le i \le \sqrt{m})$, in $O(\frac{\log n}{\log m})$ 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, $(1 \le i \le \sqrt{m})$, be the submesh of 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 $(i-1)\sqrt{m}+2, (i-1)\sqrt{m}+3, \ldots, i\sqrt{m}$

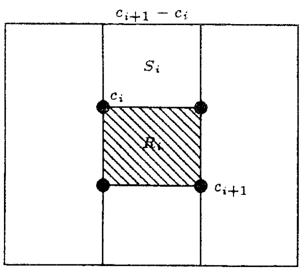

Figure 7: *Illustrating the submeshes S_i*

with columns $c_i, c_i + 1, c_i + 2, \ldots, c_{i+1}$, as illustrated in Figure 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 Figure 7; for $\log\log m$ iterations, repeat Steps 1–3 above in each submesh S_i.

The correctness of 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, $c_{i+1} - c_i, n$ and so, by Lemma 3.4 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 $m > 2$.* □

If $m = n^\epsilon$ for some fixed $\epsilon \leq 1$ then Theorem 5.3 has the following consequence.

Corollary 5.4 *The task of computing the row-minima of a monotone matrix of size $m \times n$ with $m = n^\epsilon$, $\epsilon \leq 1$, 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 [8].

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 ϵ, $(0 < \epsilon \leq 1)$, then our algorithm runs in $O(\log\log n)$ time.

Our second main contribution was to exhibit a number of non-trivial lower bounds for the task of matrix searching on reconfigurable meshes. In particular, we obtained an $\Omega(\sqrt{\log\log n})$ time lower bound for the problem of computing the row minima of a monotone matrix of size $n \times n$ stored in n contiguous rows and columns of an infinite platform. This is the first non-trivial lower bound of this kind known to the authors.

There is a discrepancy between the lower bound obtained and the upper bound provided by our algorithm. narrowing this gap will be a hard problem that we leave 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 this date we have not been able to find a non-trivial lower bound for this problem.

Acknowledgement: The second author wishes to thank Mike Atallah for many useful comments and for pointing out a number of relevant references.

References

[1] A. Aggarwal and M. M. Klawe, Applications of generalized matrix searching to geometric problems, *Discrete Applied Mathematics*, 27, (1990), 3–23.

[2] A. Aggarwal, M. M. Klawe, S. Moran, P. Shor, and R. Wilber, geometric applications of a matrix-searching algorithm, *Algorithmica*, 2, (1987), 195–208.

[3] A. Aggarwal and J. Park, Notes on searching in multidimensional monotone arrays, *Proc. 29th Annual Symposium on Foundations of Computer Science*, October 1988, 497–512.

[4] A. Apostolico, M. J. Atallah, L. L. Larmore, and S. McFaddin, Efficient parallel algorithms for string editing and related problems, *SIAM Journal on Computing*, 19, (1990), 968–988.

[5] M. J. Atallah, A faster parallel algorithm for a matrix searching problem, *Algorithmica*, 9, (1993), 156–167.

[6] M. J. Atallah and S. R. Kosaraju, An efficient parallel algorithm for the row minima of a totally monotone matrix, *Journal of Algorithms*, 13, (1992), 394–413.

[7] J. JáJ'a, *An Introduction to Parallel Algorithms*, Addison Wesley, pp.188-189, 1992.

[8] M. Maresca, Polymorphic processor arrays, *IEEE Transactions on Parallel and Distributed Systems*, 4, (1993), 490–506.

[9] M. Maresca and H. Li, Virtual parallelism support in reconfigurable processor arrays, University of California at Berkeley, UCB–ICSI Tech. Report-91–041, July 1991.

[10] M. Maresca and H. Li, Hierarchical node clustering in polymorphic processor arrays, University of California at Berkeley, UCB–ICSI Tech. Report-91–042, July 1991.

[11] R. Miller, V. K. P. Kumar, D. Reisis, and Q. F. Stout, Parallel Computations on Reconfigurable Meshes, *IEEE Transactions on Computers*, 42, (1993), 678–692.

[12] S. Olariu, J. L. Schwing, and J. Zhang, Fundamental Data Movement on Reconfigurable Meshes, *International Journal of High Speed Computing*, 6, (1994), 311–323.

[13] D. B. Shu, L. W. Chow, and J. G. Nash, A content addressable, bit serial associate processor, *Proceedings of the IEEE Workshop on VLSI Signal Processing*, Monterey CA, November 1988.

[14] D. B. Shu and J. G. Nash, The gated interconnection network for dynamic programming, S. K. Tewsburg *et al.* (Eds.), Concurrent Computations, Plenum Publishing, 1988.

[15] L. G. Valiant, Parallelism in comparison problems, *SIAM Journal on Computing*, 4, (1975), 348-355.

[16] B. F. Wang and G. H. Chen, Constant time algorithms for the transitive closure problem and its applications *IEEE Transactions on Parallel and Distributed Systems* 1, (1990), 500–507.