P-Bandwidth Priority Queues on Reconfigurable Tree of Meshes

ALAN A. BERTOSSI AND ALESSANDRO MEI

Department of Mathematics, University of Trento, 38050 Povo, Trento, Italy

This paper shows a parallel implementation of a priority queue with bandwidth \( P \) and maximum size \( nP \) by means of a network with reconfigurable buses. The proposed solution is based on a tree of meshes architecture of \( O(nP^2) \) processors and \( O(P \log n) \) maximum subbus length. The computational times required by the operations of a priority queue with bandwidth \( P \) are \( O(1) \) for all the operations, using the unit-time delay model for broadcasting, while they are \( O(1) \) for MIN and \( O(\log P + \log \log n) \) for both DELETEMIN and INSERT, using the log-time delay model. The proposed network can be laid out in a classical H-shaped manner to occupy \( O(nP^2) \) area in the VLSI model. When \( P = O(1) \), the required area is optimal and, using the unit-time delay model, the resulting \( AT^2 \) is also optimal. The paper presents also a very simple and efficient way of merging two sorted sequences on a reconfigurable mesh, which is used in the implementation of the priority queue operations.

1. INTRODUCTION

Reconfigurable networks have received much attention in the last few years, due to technological developments that allowed some experimental and commercial reconﬁgurable chips with thousands of processors to be built [2, 9].

In a reconfigurable network, each node can dynamically connect and disconnect its adjacent edges in various patterns. Specifically, each node of the network consists of a processing unit, a small local memory, and a switch, while each edge is viewed as a building block for a larger bus. Each switch has some I/O ports and each port is directly connected to at most one edge. While the edges outside the switch are ﬁxed, the internal connections between the I/O ports of each switch can be locally conﬁgured by the processor itself into any combination of pairs and singletons. In this way, during the execution of an algorithm, the edges of the network are dynamically partitioned into edge-disjoint paths. Every such path forms a subbus, and allows (only) one processor of the subbus to broadcast at any given time a message to all the other processors sharing the same subbus.

Efficient parallel algorithms that use reconfigurable networks have been devised for many problems, such as sorting [3, 6, 8, 10, 16], matrix multiplication [12], finding the connected components of a graph [9] and image processing [7, 9]. Most such algorithms achieve an \( O(1) \) time complexity by considering the so called unit-time delay model, in which it is assumed that each broadcast takes constant time to reach all the processors in a subbus, regardless of its length (i.e., number of edges). However, the log-time delay model, where each broadcast takes a time which is logarithmic in the length of the subbus, is also considered. Since the implications of the subbus lengths for the actual performance of the network cannot be ignored, it is of paramount importance to ﬁnd algorithms which reduce the maximum subbus length as much as possible.

This paper presents a parallel implementation of a P-bandwidth priority queue, which is capable of storing at most \( nP \) keys. A priority queue with bandwidth \( P \) is a generalization of a classical priority queue, in which the usual operations of the data structure handle a set of \( P \) keys instead of only one key (e.g., see [13]). Specifically, the MIN operation returns the \( P \) smallest keys in the queue, the DELETEMIN operation removes such keys, while the INSERT(\( X \)) one adds \( P \) new keys into the queue, namely, those belonging to the set \( X \). Therefore, when \( P = 1 \) the usual priority queue results.

The proposed implementation is based on a reconfigurable tree of meshes architecture of \( O(nP^2) \) processors, and assumes that each switch has four I/O ports. The network has an \( O(P \log n) \) maximum subbus length, and the computational times required by the priority queue operations are \( O(1) \) for all the operations, using the unit-time delay model, while they are \( O(1) \) for MIN and \( O(\log P + \log \log n) \) for both DELETEMIN and INSERT(\( X \)), using the log-time delay model. For the latter model, the computational times are as good as the best times known for the EREW PRAM model ([13, 17]).

The proposed network can be laid out in a classical H-shaped manner to occupy \( O(nP^2) \) area in the VLSI model. In particular, when \( P = O(1) \), an optimal \( O(n) \) area results, since any priority queue storing \( O(n) \) keys has an obvious \( \Omega(n) \) lower bound on the area. Moreover, when \( P = O(1) \) and the unit-time delay model is used, an optimal \( O(n) \) \( AT^2 \) also results. Although several VLSI implementations
of the related dictionary data structure have been proposed in the literature (e.g., see [11, 14]), to our knowledge a VLSI implementation of a priority queue is given for the first time in the present paper.

Briefly, the remainder of the paper is organized as follows. In Section 2 the reconﬁgurable mesh architecture is reviewed and a very simple and efﬁcient way of merging two sorted sequences on a reconﬁgurable mesh is presented. Such a merging algorithm will be used in the next sections for the implementation of the priority queue operations. In Section 3 the reconﬁgurable tree of meshes architecture is deﬁned and its H-shaped layout in the VLSI grid model is shown. Section 4 shows how to simulate a heap implementation for the P-bandwidth priority queue by the processors themselves. It is worth noting that, under the assumptions that processors, switches, and edges occupy O(1) space, the reconﬁgurable mesh can be laid out on a rectangle of O(pq) area in the VLSI grid model [9].

The remainder of this section shows how a reconﬁgurable mesh can be used to efﬁciently solve a merging problem: namely, given two sorted sequences X = x0 ... xp−1 and Y = y0 ... yq−1, ﬁnd a sorted sequence Z = z0 ... zp+q−1 containing exactly every element of X and every element of Y. It is assumed that the elements to be merged are drawn from an ordered set of ﬁnite size and that each element is represented by O(1) bits. For the sake of simplicity, it is assumed that three additional elements xp, yq, and zp+q are appended, respectively, to the ends of the sequences X, Y, and Z. These elements all have a bigger value, say +∞, than any other element in X and Y.

To solve the merging problem, a (p + 1) × (q + 1) mesh is employed such that the generic processor Pij at the ith row and the jth column of the mesh holds xi and yj, 0 ≤ i ≤ p and 0 ≤ j ≤ q.

DEFINITION 2.1. Given a (p + 1) × (q + 1) mesh, the kth antidiagonal is the set of processors

\[ A_k = \{P_{ij} : i + j = k\}, \]

where 0 ≤ k ≤ p + q.

DEFINITION 2.2. Let X = x0 ... xp−1 and Y = y0 ... yq−1 be two sorted sequences to be merged. A processor Pij of a (p + 1) × (q + 1) mesh is called active when:

\[ \forall 0 \leq j < i \quad x_i > y_k, \quad \text{and} \quad (1) \]

\[ \forall 0 \leq h < i \quad x_k \leq y_j. \quad (2) \]

See, for example, Fig. 2.

LEMMA 2.3. Every antidiagonal Ak, 0 ≤ k ≤ p + q, of a (p + 1) × (q + 1) mesh contains exactly one active processor Pμv, and z0 = min{[x, y]}.

Proof. The proof is by induction on p + q. When p + q = 0, the basis of the induction holds true. Indeed, x0 = y0 = +∞, a 1 × 1 mesh is used, P00 is obviously active, and z0 = min{[+∞, +∞]}.

Assume the lemma is true for |X| + |Y| < p + q, and let |X| = p and |Y| = q. Without loss of generality, assume that x0 > y0 (the case x0 ≤ y0 can be proved in a similar way, by swapping rows and columns). In this case, the processors of the (p + 1) × (q + 1) mesh can be partitioned into three submeshes P00, v, and S as shown in Fig. 3 (note that the submesh v can be empty).

By deﬁnition, P00 is active and the processors in the submesh v are inactive. Since x0 > y0, the merged sequence Z is such that z0 = y0 and Z′ = z1 ... zp+q−1 is given by merging X = x0 ... xp−1 and Y = y1 ... yq−1. Consider now
FIG. 4. Switch settings to create a subbus per antidiagonal.

The antidiagonal $A_k$, $0 \leq k \leq p + q$. If $k = 0$, then $A_0$ contains the active processor $P_{00}$ only and, since $z_0 = y_0$, the proof follows. If $k > 0$, then $A_k$ contains the inactive processor $P_{k0}$ of the submesh $v$, and all the processors in the $(k - 1)$th antidiagonal of the submesh $S$. The status (active/inactive) of the processors in $S$ is that obtained by merging $X$ and $Y'$ on the $(p + 1) \times q$ submesh $S$. By inductive hypothesis, since $|X| + |Y'| < p + q$, the $(k - 1)$th antidiagonal of $S$ has exactly one active processor and the minimum between the two elements in such a processor is equal to the $(k - 1)$th element of $Z'$. Therefore such a minimum is also the $k$th element of $Z$, and the lemma is proved.

The next lemma refers to a well-known technique for finding the leftmost or rightmost 1 (or 0) in a sequence of bits. This has been used before in Ref. [9] and others in the context of finding the OR, and will be useful in the remainder of this paper, too. Obviously, the sequence of bits could be replaced by a sequence of boolean values that can be locally calculated by each processor in constant time.

**Lemma 2.4.** Given a sequence of bits stored one per processor in a subbus, the leftmost or rightmost 1 (or 0) can be found in $O(1)$ time, using the unit-time delay model, and in $O(\log m)$ time, where $m$ is the length of the subbus, using the log-time delay model.

**Theorem 2.5.** A $(p + 1) \times (q + 1)$ reconfigurable mesh can merge two sorted sequences $X = x_0 \ldots x_{p-1}$ and $Y = y_0 \ldots y_{q-1}$ in $O(1)$ time, using the unit-time delay model, and in $O(\log p + \log q)$ time, using the log-time delay model.

**Proof.** Assume that the $(p + 1) (q + 1)$ values $x_0 \ldots x_p$ $(y_0 \ldots y_q)$ are stored one per processor in column (row) zero. Configure all the switches $EW$, $NS$ so as to have one subbus per column and one subbus per row. Then every processor $P_{i0}$ ($P_{0j}$) broadcasts its value $x_i$ ($y_j$) on its subbus, after which every processor $P_{ij}$ knows the values $x_i$ and $y_j$.

Successively, the status (active/inactive) of each processor $P_{ij}$ has to be set. By Definition 2.2, $P_{ij}$ is inactive if and only if at least one of the following two conditions is true: (1) there is $0 \leq k < j$ such that $x_k \leq y_k$, or (2) there is $0 \leq h < i$ such that $x_i > y_j$. Consider row $i$ of the mesh, and let $h_i = \min\{j : x_i \leq y_j\}$. Note that $h_i$ is well-defined, since $y_q = +\infty$. Processors $P_{i0}, \ldots, P_{ih_i}$ do not satisfy condition (1), and hence can be either active or inactive, whereas processors $P_{ih_i+1}, \ldots, P_{iq}$ are surely inactive. To set such processors inactive, $h_i$ has to be found; this can be done applying Lemma 2.4 to row $i$ of the mesh. By performing the above procedure in parallel on all the rows, it is possible to set all the processors which are inactive because of condition (1). In a similar way, considering column $j$ and letting $h_j = \min\{i : x_i > y_j, x_i = +\infty\}$ (so that $h_j$ is well defined), it is possible to set all the processors which are inactive because of condition (2). In this way all the inactive processors are detected, and the remaining processors can thus be set active.

Finally, configure all the switches $EW$, $NS$ to have a subbus per antidiagonal, as shown in Fig. 4. Each active processor, say $P_{ij}$, of each antidiagonal $A_k$, broadcasts $z_k = \min\{x_i, y_j\}$ on the subbus. Thus the merged sequence $Z = z_0 \ldots z_{q+p-1}$ is available on the borderline processors, one element per processor in column 0 and row $p$ (or, equivalently, in row 0 and column $q$).

Since the above steps require broadcasts on subbuses which are either $O(p)$ or $O(q)$ long, the overall computa-
3. THE RECONFIGURABLE TREE OF MESHES ARCHITECTURE

The proposed parallel implementation of a P-bandwidth priority queue is based on a reconfigurable tree of meshes architecture. Such an architecture consists of a complete binary tree of $n = 2^L - 1$ nodes, each of which is a reconfigurable mesh of size $2(P + 1) \times (P + 1)$.

As shown in Fig. 5, the generic node of the tree consists of two submeshes. The upper submesh (from row 0 to row $P$) is a reconfigurable mesh of $(P + 1)^2$ processors, has the purpose of storing and processing keys, and is connected to its parent node. In particular, processors at row 0 of the root can communicate with the external world. The lower submesh (from row $P + 1$ to row $2P + 1$), instead, has only the $P + 1$ processors on the main diagonal, which allow to communicate either with the left or with the right son, depending on the configurations of its switches. More precisely, when the $P + 1$ switches are all configured NW (EN), then the node is connected to its left (right) son node, while when they are all configured EW, then the two sons nodes are directly connected between them. The above configurations, along with all switches configured O, are the only feasible configurations for the lower submesh and will be denoted, respectively, PL, PR, LR, and O. Thus, for the sake of conciseness, we shall talk about “nodes configured PL, PR, LR, or O,” instead of “all the $P + 1$ switches of the lower submesh configured NW, EN, EW, or O.” Moreover, for ease of reference, a generic node of the tree will be denoted in the paper by $N_{i,h}$, where $l$ is its level ($0 \leq l \leq L - 1$) and $h$ its heap number ($1 \leq h \leq n$). In this way, the root of the tree is $N_{0,1}$, the left (right) son of node $N_{i,l}$ is $N_{i+1,2h}$ ($N_{i+1,2h+1}$), and the rightmost leaf is $N_{L-1,n}$. However, the usual mesh notation reviewed in Section 2 will be used for processors within a node of the tree. It is worth noting that no processor needs to know the level or the heap number of the tree node to which it belongs. It is only assumed that a processor knows whether it belongs to the root, and whether it is in the last row of the upper submesh or in the lower submesh, by means of properly set bits.

The proposed reconfigurable tree of meshes architecture can be laid out in the VLSI grid model. Indeed, it is well known that a complete binary tree of $n$ nodes can be laid out in an H-shaped manner to occupy $O(n)$ area, provided that nodes have $O(1)$ area and links are $O(1)$ wide. The proposed tree of meshes can be seen as a complete binary tree in which nodes have $O(P^2)$ area and links can carry $O(P)$ keys. Therefore, it is easy to see that an H-shaped layout of the reconfigurable tree of meshes requires $O(nP^2)$ area.

4. HEAP IMPLEMENTATION ON A RECONFIGURABLE TREE OF MESHES

This section gives a heap implementation of a P-bandwidth priority queue on a reconfigurable tree of meshes (e.g. see [4] for a sequential heap implementation when $P = 1$ and [13] for a PRAM implementation when $P > 1$). Each node of the tree either is empty or stores $P$ keys, in nonincreasing order, in the first column of the root. The keys are stored in the nodes so as to maintain the following heap properties:

(a) if the node $N_{i,l}$ is empty, then every node $N_{k,h}$ with $k \geq l$ and $h > i$ is also empty;
(b) all the keys stored in a node are greater than or equal to the keys stored in its parent node.

Clearly, the $P$ smallest keys are stored in the root, and an error message must be provided when a DELETEMIN (INSERT(X)) operation is required on an empty (full) heap. For the sake of simplicity, the operations MIN, DELETEMIN, and INSERT(X) are first implemented by means of three distinct networks, each capable of performing only one operation. Successively, it is shown how to perform all the operations on the same network.

**DEFINITION 4.1.** The insertion node $N_{INS}$ is either the empty node with minimum heap number or the last node of the heap, if there is no empty node.

**DEFINITION 4.2.** The insertion path is the (unique) simple path between the root $N_{0,1}$ and the insertion node $N_{INS}$.

In the implementation, the insertion node knows its status by means of a bit properly set in each processor of the node. Similarly, each node in the insertion path knows its status and the switch configuration to connect itself to the next node in the path. In this way, a subbus can be created along the insertion path in $O(1)$ time. The property of

---

**FIG. 5.** A node of the tree of meshes ($P = 4$ in this example).
empty (full) heap is checked after each \textsc{DeleteMin}(\textsc{Insert}(X)) operation and the resulting information is maintained by the root. Moreover, each node knows whether it stores keys or not and, in the former case, is configured \textit{O}. When the computation starts, the heap is empty, and the root is the insertion node.

**Proposition 4.3.** The reconfigurable tree of meshes performs the \textsc{Min} operation in \(O(1)\) time.

**Proof.** The root transmits to the external world either an error message (if it is the insertion node) or the \(P\) keys stored in its first row (if it is not the insertion node). The time required is obviously \(O(1)\).

**Definition 4.4.** Given a node \(N_{l,i}\), the next node next\((N_{l,i})\) is the node following \(N_{l,i}\) in the heap numbering. Formally,

\[
\text{next}(N_{l,i}) = \begin{cases} 
N_{l+1,i} & \text{if } 0 \leq l < L \text{ and } i < 2^{l+1} - 1 \\
N_{l,i+1} & \text{if } 0 \leq l < L - 1 \text{ and } i = 2^{l+1} - 1 \\
N_{l,i} & \text{if } l = L - 1 \text{ and } i = n.
\end{cases}
\]

**Definition 4.5.** A node \(N\) is updated if:

1. \(N\) and only \(N\) is flagged as the insertion node; and
2. all nodes in the path from the root to \(N\), and only these nodes, are flagged as being in the insertion path.

**Lemma 4.6.** If \(N_{\text{INS}}\) is updated, then the reconfigurable tree of meshes can update next\((N_{\text{INS}})\) in \(O(1)\) time, using the unit-time delay model, and in \(O(\log P + \log \log n)\) time, using the log-time delay model.

**Proof.** Let \(N_{Y}\) be the least common ancestor of \(N_{\text{INS}}\) and next\((N_{\text{INS}})\), namely the node of maximal level belonging to both the simple paths \(N_{0,1} - N_{\text{INS}}\) and \(N_{0,1} - \text{next}(N_{\text{INS}})\), respectively. Configure the switches in the nodes of \(N_{0,1} - N_{\text{INS}}\) to create a subbus along such a path.

Assume \(N_{\text{INS}} = N_{l,i}\) with \(i < 2^{l+1} - 1\). As shown in Fig. 6, \(N_{\text{INS}}\) and next\((N_{\text{INS}})\) are both at the same level, but \(N_{\text{INS}}\) is the rightmost node in the left subtree of \(N_{Y}\), while next\((N_{\text{INS}})\) is the leftmost node in the right subtree of \(N_{Y}\). This implies that \(N_{Y}\) is the node of maximal level in \(N_{0,1} - N_{\text{INS}}\) configured \(PL\) and the node of maximal level in \(N_{0,1} - \text{next}(N_{\text{INS}})\) configured \(PR\). The above property allows \(N_{Y}\) to be detected along the insertion path subbus using the technique of Lemma 2.4. Then \(N_{Y}\) configures itself \(PR\), while all the nodes storing keys which do not belong to the insertion path configure themselves \(PL\). The root sends a signal along the subbus. By construction, all the nodes which receive the signal belong to the new insertion path and, in particular, the only node which receives the signal and is configured \(O\) (since it stores no key) is next\((N_{\text{INS}})\).

Assume now that \(i = 2^{l+1} - 1\) and \(l < L - 1\). In this case, \(N_{Y}\) is \(N_{0,1}\) and can be detected as before. There is no node configured \(O\) in \(N_{0,1} - N_{\text{INS}}\), therefore, the root changes its configuration from \(PR\) (or \(O\), if \(N_{\text{INS}} = N_{0,1}\)) to \(PL\), and the computation proceeds as in the previous case.

Finally, when \(i = 2^{l+1} - 1\) and \(l < L - 1\) are performed. In the last step, however, there is no node configured \(O\) which receives the signal broadcast by the root, since the heap is full. This situation is detected by the root, and the insertion path is not changed.

The time complexity of the above algorithm is due to broadcasts on subbuses which are \(O(P \log n)\) long. Therefore it is \(O(1)\), using the unit-time delay model, and \(O(\log P + \log \log n)\), using the log-time delay model.

**Lemma 4.7.** A reconconfigurable mesh of size \(m \times m\) can sort \(m\) values, stored one per processor in the first row, in \(O(1)\) time, using the unit-delay model, and in \(O(\log m)\) time, using the log-time delay model.

**Proof.** See, for example, the algorithm shown in [10]. Note that the algorithm of [10] works on the bit-model, too, with minor and straightforward changes.

**Theorem 4.8.** The reconconfigurable tree of meshes performs the \textsc{Insert}(\textit{X}) operation in \(O(1)\) time, using the unit-time delay model, and in \(O(\log P + \log \log n)\) time, using the log-time delay model.

**Proof.** The \(P\) keys in \(X\) are read by the processors at row 0 of the root. Such keys are then sorted by the upper mesh of the root in \(O(1)\) time, using the unit-time delay model, or in \(O(\log P + \log \log n)\) time, using the log-time delay model (Lemma 4.7).

Configure the switches to build the insertion path. Let \(l\) be the level of the insertion node. It is possible to consider a big \((l + 1)P \times (P + 1)\) submesh by selecting the processors in the first \(P\) rows belonging to the upper submeshes of the nodes in the insertion path. Note that this “big submesh” is distributed over the tree of meshes; i.e., it does not consist of adjacent processors. In this way, merging can be performed between the \(P\) keys at row 0 of the root, and the \(IP\) keys at column 0 of the big submesh. An example is provided in Fig. 7, where active and inactive processors of the big submesh are white and shaded, respectively. By Theorem 2.5, this requires \(O(1)\) time, using the unit-time delay model, and \(O(\log P + \log \log n)\) time, using the log-time delay model, since the submesh has size
A path of minima is rearranged if all the keys stored in $M_1$ are less than or equal to all the keys stored in $M_2$, for all brother nodes $M_1$ and $M_2$ such that $M_1$ belongs to the path.

Observe that a path of minima always exists but is not unique. In the following implementation of the DELETEMIN operation, it is sufficient to detect any path of minima.

**Lemma 4.12.** The reconﬁgurable tree of meshes can conﬁgure a subbus along a path of minima and rearrange such a path in $O(1)$ time, using the unit-time delay model, and in $O(\log P)$ time, using the log-time delay model.

**Proof.** Each node either transmits to its parent the maximum key it stores or communicates to its parent the fact that it is empty. Such a procedure can be performed in two phases, one for the left sons, and the other for the right sons. The parents keep track of the son containing the maximum key, merge the $2P$ keys coming from its sons, and rearrange these keys into the sons. Finally, each parent conﬁgures its switch to connect itself to the proper son. The overall time complexity is dominated by that of merging $O(P)$ keys and therefore is $O(1)$, using the unit-time delay model, and $O(\log P)$, using the log-time delay model. ■

**Theorem 4.13.** The reconﬁgurable tree of meshes performs the DELETEMIN operation in $O(1)$ time, using the unit-time delay model, and in $O(\log P + \log \log n)$ time, using the log-time delay model.

**Proof.** Configure the switches so as to build the insertion path and move the $P$ keys from the insertion node to the processors at row 0 and column 0 of the root. In this way, $N_{\text{INS}}$ becomes empty and the $P$ smallest keys are lost. According to Lemma 4.10, update $\text{prev}(N_{\text{INS}})$. If $N_{\text{INS}} = N_{i,j}$, then the root keeps track that the heap is now empty. Otherwise, in order to maintain property (b) of the heap, conﬁgure a subbus along a path of minima and rearrange the keys as described in the proof of Lemma 4.12. In this way, it is possible to move upwards the keys contained in the path of minima without violating property (b) of the heap. Specifically, each node in the path stores the $P$ keys previously contained in its son. Then the root contains in column 0 the $P$ keys coming from its son, and in row 0 the $P$ keys of the old insertion node. Therefore, merging can be performed between the $P$ keys at row 0 of the root, and the keys at column 0 of the nodes in the path of minima, as described in Theorem 4.8. Since all the steps of the above procedure require $O(1)$ time, using the unit-time delay model, and $O(\log P + \log \log n)$ time, using the log-time delay model, the proof follows. ■

For the sake of simplicity, the operations MIN, DELETEMIN, and INSERT($X$) were implemented by means of three distinct networks, each capable of performing only one operation. We now show how to perform all the operations on the same network. To do this, proper commands

$O(P \log n) \times O(P)$. Observe that the keys in the merged sequence are available one per processor at column 0 of the big submesh and, in particular, the $P$ largest keys correctly go into the first $P$ processors at column 0 of the insertion node.

It is readily seen that properties (a) and (b) of the heap are maintained. Indeed, only the insertion node is no more empty among the previously empty nodes, while the keys stored in each node of the insertion path are smaller than or equal to the previously stored keys. The procedure terminates by updating the insertion node, as described in the proof of Lemma 4.6, during which the root can detect whether the heap is full or not. Since this requires $O(1)$ time, using the unit-time delay model, and $O(\log P + \log \log n)$ time, using the log-time delay model, the proof follows. ■

**Definition 4.9.** Given a node $N_{i,j}$, the previous node $\text{prev}(N_{i,j})$ is the node preceding $N_{i,j}$ in the heap numbering.

**Lemma 4.10.** If $N_{\text{INS}}$ is updated, then the reconﬁgurable tree of meshes can update $\text{prev}(N_{\text{INS}})$ in $O(1)$ time, using the unit-time delay model, and in $O(\log P + \log \log n)$ time, using the log-time delay model.

**Proof.** The proof is analogous to that of Lemma 4.6. ■

**Definition 4.11.** A path of minima is a simple path $N_{i,1} - N_{i,j}$ which satisﬁes the following two properties:

(i) the node $N_{i,j}$ is not empty, while both its sons (if any) are empty;

(ii) if $M_1$ and $M_2$ are two brother nodes such that $M_1$ belongs to the path, then $M_2$ holds the maximum key among the $2P$ keys stored in $M_1$ and $M_2$.
have to be received by the processors without increasing the overall time complexity. Observe that commands cannot be broadcast by the root to all the remaining nodes, since otherwise the same $O(\log n)$ time complexity of the sequential implementation would result.

**Theorem 4.14.** A reconfigurable tree of meshes of $O(nP^2)$ processors and $O(P\log n)$ maximum subbus length can implement a $P$-bandwidth priority queue of maximum size $nP$. The root is capable of receiving an operation code denoting \textsc{MIN}, \textsc{DELETEMIN}, or \textsc{INSERT}, and eventually receiving or furnishing $P$ keys. The time required by the network to perform an operation is $O(1)$ for all the operations, using the unit-time delay model for broadcasting, while it is $O(1)$ for \textsc{MIN} and $O(\log P + \log \log n)$ for both \textsc{DELETEMIN} and \textsc{INSERT}, using the log-time delay model. When the execution of an operation is terminated, the network is ready to perform another operation.

**Proof.** All the processors in the tree configure the switches according to the following cyclic algorithm:

- **Step 1.** The nodes in the insertion path create a subbus along the path itself, while the remaining nodes configure them $PR$ to perform the actions described in Lemma 4.10.

- **Step 2.** The nodes in the tree create a subbus along the path of minima.

- **Step 3.** The nodes in the insertion path create a subbus along the path itself, while the remaining nodes configure them $PL$ so as to perform the actions described in Lemma 4.6.

Assume that each step of the above algorithm is slow enough to allow a constant number of broadcasts along subbuses of $O(P\log n)$ length to be performed, as well as a merging on an insertion path or on a path of minima.

If the root receives a \textsc{MIN} operation code, the root itself performs the operation, as shown in Proposition 4.3, without involving any other node.

If the root receives an \textsc{INSERT} operation code, along with the $P$ keys to be inserted, two cases arise. If the queue is full, the root itself outputs an error message. Otherwise, when Step 1 begins, the root broadcasts to all the processors in the insertion path a proper code, so that such processors perform the merging algorithm of Theorem 4.8. Then, when Step 3 begins, the root broadcasts to all the processors in the insertion path another code, so that the insertion path is changed as described in Lemma 4.6.

Finally, two cases arise also when the root receives a \textsc{DELETEMIN} operation code. If the queue is empty, the root itself outputs an empty message. Otherwise, when Step 1 begins, the root broadcasts to all the processors in the insertion path a proper code, in order to update the insertion path as described in Lemma 4.10 and move the $P$ keys from the insertion node to the root. Finally, during Step 2, when the path of minima is created, the merging procedure described in Theorem 4.13 is performed.

It is easy to see that the above actions can be performed so as to take $O(1)$ time for all the priority queue operations, using the unit-time delay model, or $O(1)$ for \textsc{MIN} and $O(\log P + \log \log n)$ for both \textsc{DELETEMIN} and \textsc{INSERT}, using the log-time delay model.

**5. Conclusions**

A parallel implementation of a $P$-bandwidth priority queue of size $nP$ by a tree of meshes network with reconfigurable busses of $O(nP^2)$ processors and $O(P\log n)$ maximum subbus length has been presented. It is worth noting that the proposed tree of meshes network can be embedded on an $O(Pn^{1/2}) \times O(Pn^{1/2})$ reconfigurable mesh. However, the maximum subbus length becomes $O(Pn^{1/2})$, thus increasing the computational time in the log-time delay model. Since reconfigurable meshes are scalable, i.e., can run algorithms for large problem instances on small machines (see Theorem 4.1 of [1] for the \textit{LRN} model), the parallel implementation of the priority queues proposed in the present paper is scalable as well. In particular, an $m \times m$ reconfigurable mesh can simulate the $O(Pn^{1/2}) \times O(Pn^{1/2})$ one with $O(Pn^{1/2})$ slowdown and $O(Pn^{1/2})$ extra space at each processor of the $m \times m$ mesh.

Several questions, however, remain open. In particular, it would be interesting either to find implementations with smaller subbus length and/or to prove $AT^2$ optimality using the log-time delay model. Moreover, it would be interesting as well to find further applications of the reconfigurable tree of meshes proposed in this paper.

**References**


Received October 13, 1995; revised October 7, 1996; accepted October 11, 1996

ALAN ALBERT BERTOSSI was born in London, England, in 1956. He received the Laurea degree in computer science from the University of Pisa, Italy, in 1979. Afterward, he worked as a system programmer and designer. From 1983 through 1994 he was with the Department of Computer Science of the University of Pisa, as a research associate and as an associate professor. Since 1995 he has been with the Department of Mathematics of the University of Trento, as a professor of computer science. His main research interest is the design and analysis of algorithms for parallel and distributed systems.

ALESSANDRO MEI received the Laurea degree in Computer Science from the University of Pisa, Italy, in 1994. Currently, he is a Ph.D. student in the Department of Mathematics of the University of Trento. His research interest is the design and analysis of algorithms for parallel and distributed systems.