Abstract

We present an efficient mapping of a min-max-pair heap of size $N$ on a hypercube multicomputer of $p$ processors in such a way the load on each processor’s local memory is balanced and no additional communication overhead is incurred for implementation of the single insertion, deletemin and deletemax operations. Our novel approach is based on an optimal mapping of the paths of a binary heap into a hypercube such that in $O(\log N) + \log p$ time we can compute the Hamiltonian-Suffix, which is defined as a pipelined suffix-minima computation on an $O(\log N)$-length heap path embedded into the Hamiltonian path of the hypercube according to the binary reflected Gray codes. However, the binary tree underlying the heap data structure is not altered by the mapping process.

1. Introduction

Efficient data structures are crucial for designing fast and efficient algorithms, be them sequential or parallel.

Although parallel heaps or priority queues for the shared memory machines have been well studied [1, 6], relatively less is known about implementing such data structures on the synchronous, distributed memory multicomputers [2].

In this paper, we propose a simple technique for efficient implementation of heap operations on the hypercube architecture, using a complete binary tree as the underlying representation of the heap. Without any slow-down, we have been able to simulate the heap management algorithms due to Pinotti and Pucci [6] designed for the PRAM model, thus leading to the cost-optimal hypercube algorithms for a min-max-pair heap which allows insertion as well as deletion of both the largest and the smallest items. Precisely, our implementation of a single insertion, deletemin and deletemax operations operations) on a min-max-pair heap of $N$ items and height $h$, embedded in a hypercube of size $p$, requires an optimal time of $O(\frac{h}{\log h}) = O(\log \log N)$ employing $p = O(\frac{h}{\log h}) = O(\frac{\log N}{\log \log N})$ processors.

The well-known notions of the cost and speed-up complexity for PRAM models have been extended in this paper to the synchronous distributed systems. The sublogarithmic time complexity is attained with the help of what we call the Hamiltonian-Suffix computation, which efficiently performs a suffix operation on a heap path, of length $O(\log N)$, embedded on the Hamiltonian path (constructed using binary reflected Gray codes) of a hypercube. In our opinion, the Hamiltonian-Suffix operation is of interest on its own and may have applications in other problems. The proposed mapping technique also exemplifies the problems involved in the embedding of a dynamic data structure.

In this context, let us mention that recently Das et. al. [2] have addressed the problem of how to map the nodes of a binary heap to a given set of memory modules such that the load is well balanced among the modules and different portions of the data structure can be accessed simultaneously by multiple processors without memory contentions. There, we have introduced a new data structure, called the slope-tree, for efficiently storing a heap on the hypercube. A slope-tree is a special kind of binary tree whose nodes in the upper $n$ levels are reorganized in a chain of length $2^n$, it has a larger height compared to a complete binary tree storing the same number of items. Thus, the actual cost of managing the heap operations is higher although the order of complexity may remain the same. Additionally, the use of a new data structure yields a less straightforward mapping, which motivates the present work.

This paper is organized as follows. Section 2 proposes a dynamic embedding of a complete binary tree into the hypercube network, with the help of Gray codes based Hamiltonian path. Section 3 deals with the Hamiltonian-Suffix computation, while Section 4 designs cost-optimal algorithms for the min-max-heap operations.

*This work is partially supported by Texas Advanced Technology Program grant under Award No. TATP-003594031.
2. Optimal Mapping of Paths of a Binary Tree into Hypercube

The proposed embedding is not aimed for arbitrary applications whose underlying structures are bounded trees, but is oriented towards the efficient implementation of heap data structures on a synchronous distributed system. Our mapping is only defined for dynamic complete binary trees, which grows or shrinks level by level.

The sought embedding of a heap \( H \) of height \( h \) into a hypercube \( Q_n \) with \( p = 2^n \) processors should satisfy the following three constraints.

1. Provide a one-to-one embedding of any subpath of \( p \) consecutive nodes of a path \( \pi \) in such a way that their images span all the \( p \) processors of \( Q_n \), and that two adjacent nodes in \( \pi \) are mapped to processors at a distance \( O(1) \). This mapping allows the shifting of \( p \) nodes on a path in constant parallel time.

2. Perform a suffix computation on any path \( \pi \) of the heap \( H \) in \( O\left(\frac{h}{p} + \log p\right) \) time.

3. Maintain a balanced load on the local memory of each processor in \( Q_n \).

As explained in [3], the existing results [4] on dynamic tree embedding cannot be applied in our application. For example, in the trivial level-by-level embedding of a complete binary tree into a hypercube such that one hypercube processor is assigned per tree-level, it is possible to shift each path in the desired time. However, the load on local memory will be highly imbalanced since the number of nodes doubles at subsequent levels.

In order to discuss our approach, let \( T \) be a complete binary tree of height \( h \) to be embedded in the hypercube \( Q_n \) of \( p = 2^n \) processors. Let each node of \( Q_n \) be specified by the binary address \( (b_{n-1}, b_{n-2}, \ldots, b_0) \), where the bit \( b_i \) corresponds to the \( i \)-th dimension of \( Q_n \). Moreover, let \( G(n) = \{G_0(n), G_1(n), \ldots, G_{2^n-1}(n)\} \) denote the set of all \( n \)-digit code words of the binary reflected Gray codes, where \( G(1) = \{0, 1\} \) and recursively \( G(n + 1) = \{0G_0(n), 0G_1(n), \ldots, 0G_{2^n-1}(n), 1G_{2^n-1}(n), 1G_{2^n-2}(n), \ldots, 1G_0(n)\} \).

Letting \( G_i(n) \) also denote the binary address of a processor in \( Q_n \).

**Definition 1**: The Hamiltonian path \( \Pi \) visits the nodes of \( Q_n \) in the order \( G_0(n), G_1(n), \ldots, G_{2^n-1}(n) \). The \( x \)-th node, \( G_x(n) \), of \( \Pi \) will be referred to as \( \Pi(x) \) which is at Hamming distance from \( x \) at unit. As an example, \( \Pi = (0, 1, 3, 2, 6, 7, 5, 4) \) is a Hamiltonian path in \( Q_3 \), and \( \Pi(3) = 2 \).

Let \( (i, j) \) denote the \( j \)-th node at level \( i \) of the tree \( T \), assuming that the root is at level 0 and the nodes at each level are counted from left to right, starting from 0. Moreover, let the upper-part \( U \) of \( T \) consists of its top \( n \) levels numbered from 0 to \( n - 1 \). All the remaining \( h - n \) levels of \( T \) form the lower-part \( L \) of \( T \). (Note that \( h \) can assume any value.) These upper- and lower-parts will be mapped onto \( Q_n \) according to two different strategies, guaranteeing that between a tree-node \( (i, j) \) and its parent, the dilation is at most 2 if \( (i, j) \in U \) and the dilation is 1 if \( (i, j) \in L \).

To map the lower part \( L \), we group the nodes in each level \( i \geq n - 1 \) into \( p \) clusters as follows. At level \( n \), each of the \( p \) clusters has only one node. At level \( n + 1 \), the \( i \)-th cluster contains the two children of the node in the \( i \)-th cluster at level \( n \). In general,

**Definition 2** [2]: The \( j \)-th cluster \( C_{i,j} \) in level \( i \), where \( i \geq n \) and \( 0 \leq j \leq n - 1 \), consists of the nodes \( (i, k) \) such that \( j2^{i-n} \leq k \leq (j + 1)2^{i-n} - 1 \).

We now make use of Latin squares for distributing the clusters to the hypercube processors.

**Definition 3**: A \( p \times p \) matrix, \( M \), is called a Latin square of order \( p \) if every integer from the set \( \{0, 1, 2, \ldots, p - 1\} \) occurs exactly once in each row and in each column of \( M \).

The first row of \( M \) contains the processor indices in the order of appearance in \( \Pi \). Every other row is formed by cyclically shifting the previous row one position to the left. The \( j \)-th column of \( M \) contains the processors in \( \Pi \) rotated cyclically \( j \) times to the left.

**Definition 4**: All the nodes in a given cluster \( C_{i,j} \), where \( i \geq n \), are assigned to the processor whose index is equal to the matrix element \( M_{i \text{ mod } p,j} \).

Now to explain the mapping of the upper-part \( U \), let us consider an extended-upper-part, \( E \), of \( T \) consisting of the top \( n + 1 \) levels. Since the leaves of \( E \) are part of \( L \), their mapping is already defined. The mapping of the remaining nodes of \( E \) can be described by looking at its leaves. Let \( E_{i,j} \) be the subtree of \( E \) rooted at the node \( (i, j) \), where \( 0 \leq i \leq n - 1 \) and \( 0 \leq j \leq 2^i - 1 \). The node \( (i, j) \) is assigned to the same processor to which the rightmost leaf of \( E_{i,j} \) is assigned if \( j \) is even, that is if \( j \) is the left child of its parent. On the other hand, \( (i, j) \) and the leftmost leaf of \( E_{i,j} \) are assigned to the same processor if \( j \) is odd, i.e. the right child of its parent. See Figure 2 for an illustration. Formally,
Thus, any path \( \tau \) of the lower-part of the tree \( T \), originating at the node \((n, j)\), where \( 0 \leq j \leq 2^n - 1 \), is a folded path since every \( 2^n \) consecutive levels of \( \tau \), starting from level \( n + j - 1 \), are assigned to \( \Pi \).

The objective of this section is to show how to compute efficiently a suffix-minima on a folded path. After such a suffix computation, it is required that the value stored at the node at level \( z \) of \( \pi \) is the minimum among all the items stored from levels \( z \) to the leaf level \( h \) of \( \pi \).

**Definition 6**: Let the Hamiltonian-Suffix on \( \pi \) be defined as the suffix-minima on a folded path \( \pi \).

To the best of our knowledge, none of the existing algorithms for suffix computation on a hypercube considers the processors of the hypercube in the order induced by the Hamiltonian path based on the Gray codes. They assume that the \( i \)-th input data item is stored in the processor \( P_i \), for \( 0 \leq i \leq p - 1 \), of the hypercube. Therefore, by the existing methods, the Hamiltonian-Suffix can only be computed using a preprocessing step to route all the data of the path into the hypercube. This step requires \( O(\frac{2^n}{p} \log p) \) time [4], thus preventing us to reach the claimed time bound.

The time complexity of \( O(\frac{2^n}{p} \log p) \) for the Hamiltonian-Suffix is achieved in three logical steps, the detailed implementations being described in the following subsections.

**Step 1**: Describe how to compute the In-Order-Suffix on a complete binary tree \( T \) of size \( p \). That is, numbering the nodes of \( T \) from 0 according to the in-order traversal, compute for each node \( x \) the minimum among the items stored in the nodes \( x, \ldots, p - 1 \).

**Step 2**: Define a hang binary tree, \( G(B) \), as an augmented full binary tree embedded in \( Q_n \), whose in-order traversal corresponds to the Hamiltonian path \( \Pi \) of \( Q_n \).

**Step 3**: Explain the pipelining on the folded path \( \pi \). At the leaves of \( G(B) \), at any time, start a new suffix-minima computation with the \( p \) consecutive nodes of \( \pi \). Each suffix-minima is performed by applying the In-Order-Suffix on the hang binary tree \( G(B) \) whose traversal corresponds to \( p \) subsequent nodes of \( \pi \).

### 3.1. In-Order-Suffix

Let us first describe the In-Order-Suffix computation on a balanced binary tree \( T \) of height \( n \). It is performed by two traversals – a climbing up followed by a climbing down. While climbing up, each node \( x \) holds the minimum \( L_x \) of the items stored in the subtree rooted at its left child, the minimum \( R_x \) of the items stored in the subtree rooted at its right child, and its local item, \( I_x \). For a leaf \( l \), we assign \( L_l = R_l = \infty \). The node \( x \) computes the local minima \( L_x \) and \( R_x \) by finding the minimum among the values computed at the previous step in its left child and right child. Then, climbing down, each node \( x \) computes its suffix value \( Min(x) = \min_{i=0}^{n-1} \{ I_i \} = \min \{ F_{p(x)}[x], R_x, L_x \} \), where \( F_{p(x)} \) is the value received from the parent \( p(x) \). We
assume that the root receives the value \( +\infty \) from its void parent. Then, the node \( x \) sends the values \( M_{\text{in}}(x) \) and \( F_{\text{P}(x)} \) to its left and right children, respectively.

By induction, the correctness of the In-Order-Suffix procedure, requiring \( O(n) \) sequential time, can be proved.

3.2. Hang Tree

A hang-tree \( B(2^n) \) is a binary tree of size \( 2^n \) and height \( n+1 \) whose root has only the left child, which in turn is the root of a full binary tree of height \( n \).

Let us introduce some notations for \( B(2^n) \), as illustrated in Figure 3. Let \((i, j)\) denote the \( j \)-th node at level \( i \), where \( 0 \leq i \leq n+1 \), of \( B(2^n) \) counting the nodes (starting from 0) at each level from left to right, and counting the levels bottom-up. Let \( LC(i, j) \) and \( RC(i, j) \) be, respectively, the left and the right child (if any) of the node \((i, j)\).

**Definition 7**: The route-leaf \( S(i, j) \) is the leftmost leaf of the subtree rooted at the node \( RC(LC(i, j)) \). When \( i = 2 \), \( S(i, j) \) is simply the right child of the left child of \((i, j)\) since the subtree rooted at the node \( RC(LC(i, j)) \) is degenerated.

Finally, let us number the node \((i, j)\) with \( \eta(i, j) \) if it is the \( \eta(i, j) \)-th node visited during the in-order traversal of \( B(2^n) \). It is easy to show that \( \eta(i, j) = 2^{i+1} + 2^j - 1 \). Observing that \( LC(i, j) = (i - 1, 2j) \), \( RC(i, j) = (i - 1, 2j + 1) \), and \( S(i, j) \) is the successor of \( LC(i, j) \) during the in-order traversal, it follows that

**Lemma 1**: According to the in-order traversal numbering of the nodes in the hang-tree \( B(2^n) \), \( LC(i, j) \) and \( RC(i, j) \) of a node \((i, j)\), where \( i \geq 1 \), are numbered as

\[
\eta(LC(i, j)) = \eta(i, j) - 2^{i-1} = j2^{i+1} + 2^j - 1, \quad \text{and} \\
\eta(RC(i, j)) = \eta(i, j) + 2^{i-1} = j2^{i+1} + 2^j + 2^{i-1} - 1.
\]

Moreover, the route-leaf of \((i, j)\), where \( i \geq 2 \), is numbered as

\[
\eta(S(i, j)) = \eta(LC(i, j)) + 1 = j2^{i+1} + 2^{i-1} + 1.
\]

Let us now embed the \( B(2^n) \) onto the hypercube \( Q_n \).

**Definition 8**: Using the binary reflected Gray codes, the node \((i, j)\) of \( B(2^n) \) is assigned to the processor \( G_{\eta(i,j)}(n) \) of \( Q_n \). The embedded \( B(2^n) \) in \( Q_n \) will be referred to as \( G(B(2^n)) \), or simply “\( G(B) \) in \( Q_n \)”, where \( G \) stands for the Gray codes.

By Definition 8, the in-order traversal of \( G(B) \) visits the processors of \( Q_n \), in the same order as in the Hamiltonian path \( \Pi = (G_0(n), G_1(n), \ldots, G_{2^n-1}(n)) \) of \( Q_n \).

**Property 2**: The embedding of the hang-tree yields:

1. unit dilation between a node at level 1 and its children;
2. a dilation of 2 between any node at level \( i \), where \( i \geq 2 \), and its children.

For the nodes at levels \( i \geq 2 \), let us prove that all the left (resp., right) nodes of \( G(B) \) can simultaneously send a message to their parents. This allows that the suffix computation on a folded path can be pipelined.

**Property 3**: For each node \((i, j)\), where \( i \geq 2 \), on the embedded tree \( G(B) \), the message from \((i, j)\) to one of its children is routed through the route-leaf \( S(i, j) \) such that all the nodes at levels \( i \) can simultaneously receive (resp., send) a message from (resp., to) their left children. Similarly, all the nodes at levels \( i \) can simultaneously receive (resp., send) a message from (resp., to) their right children.

3.3. Hamiltonian-Suffix

Applying the In-Order-Suffix procedure to the hang-tree \( G(B) \), we compute the Hamiltonian-Suffix on a subpath of size \( p \) since the in-order traversal of \( G(B) \) corresponds to the Hamiltonian Path \( \Pi \) of \( Q_n \). Then, working in a pipelined fashion, the Hamiltonian-Suffix on the folded path \( \pi \) can be computed in \( O(\frac{p}{2} + \log p) \) time.

The In-Order-Suffix procedure starts working on the items stored at the levels \( \{h - 2^n + 1, \ldots, h\} \) of \( \pi \). After the leaves (at level 0) of \( G(B) \) have sent their partial results to their parents at level 1, the leaves of \( G(B) \) are fed by the items at levels \( \{h - 2^n + 1, \ldots, h - 2^n\} \). Since the messages from the nodes of \( G(B) \) at level 1 to those at level 2 pass through the leaves, three steps are required to move the computations from levels 0 and 1, respectively, to the levels 1 and 2. More precisely, first the left nodes at level 1 and then the right nodes at level 1 send their values to level 2. Finally, messages are sent from the leaves to the level 1.

However, starting from this moment the computations at level \( i > 1 \) of the embedded hang-tree \( G(B) \) move to level \( i+1 \) simultaneously, as proved in Property 3. Hence, again after three steps, a new computation can start at the leaves of \( G(B) \). This process continues.

**Theorem 2**: The suffix-minima on a path \( \pi \) from the root to the leaf \((h, j)\) of the binary tree \( T \) can be computed in \( O(\frac{2^n}{2} + n) \) parallel time on the hypercube \( Q_n \).

Note here that this implementation exploits the large number of links available in the hypercube architecture.

4. Hypercube Implementation of Min-Max-Pair Heap

A min-max pair heap [5] is a complete binary tree filled in all levels except possibly the lowest, which is filled from left to right. Each node, except possibly the more recent inserted node at the lowest level, contains a pair of elements. The first element is called the min-element while the second is the max-element. The min-max-ordering at

![Figure 3: The embedding of the hang-tree B(16) into Q_4.](image-url)
a node is specified as: (i) no min-element is greater than the max-element (if there is one), and (ii) the min-element (max-element) is the smallest (largest) element in the subtree rooted at the node. Clearly, the root contains the smallest and the largest among all the elements from the tree. The four operations that manipulate a min-max-pair heap \( H \) are [5]: MakeMinMax\( (H) \), Insert \((H, L, x)\), Deletemin \((H)\), and Deletemax \((H)\).

Let \((r, k)\)min and \((r, k)\)max, respectively, denote the min-element and the max-element of a node \((r, k)\) of \( H \). Let the left-heap\((H)\) be the min-heap formed by all the min-elements of the nodes of \( H \). Similarly, the right-heap\((H)\) is defined as the max-heap. Let the path \( \pi[R, L] \) be the path of the min-max heap from its root \( R \) to the leaf \( L \), and let the left-path (resp., right-path) of \( \pi[R, L] \) be the path with only the min-elements (resp., the max-elements) of \( \pi[R, L] \). Observe that the elements stored in the left-path of \( \pi[R, L] \) followed by the ones stored in the right-path in reverse order (i.e., the nodes of the right-path visited from the leaf to the root) of \( \pi[R, L] \) form a sorted sequence in the non-decreasing order.

To speed up the parallel implementation of the deleteMin and deleteMax operations, let us introduce the concepts of min-path and max-path.

**Definition 9** [6]: For each node \((r, k)\) of the min-max-pair heap \( H \), let the min-path \( \mu(r, k) \) be defined by the following two conditions:

- The node \((r, k)\)min belongs to \( \mu(r, k) \).
- Let a non-leaf node \((s, t)\)min belong to \( \mu(r, k) \). If \((s, t)\)min has only one child \((s + 1, u)\)min, then \((s + 1, u)\)min belongs to \( \mu(r, k) \). Otherwise, if \((s + 1, u)\)min and \((s + 1, v)\)min are the two children of \((s, t)\) such that \((s + 1, u)\)min \( < \) \((s + 1, v)\)min, then \((s + 1, u)\)min belongs to the path \( \mu(r, k) \).

Similarly, the max-path \( \nu(r, k) \) starting from each node \((r, k)\)max of \( H \) can also be defined [3].

Finally, a min-max-pair-heap contains for each node \((r, k)\) two additional information – the min-target-leaf, \( \lambda(r, k) \), and the max-target-leaf, \( \Lambda(r, k) \), which are respectively the addresses of the leaves at which the min-path \( \mu(r, k) \) and the max-path \( \nu(r, k) \) terminate.

**Property 4**: For the path \( \pi[X, I] \) from a node \( X \) to a leaf \( I \), the min-max-ordering remains preserved if the left-path (resp., the right-path) of \( \pi[X, I] \) is shifted one level down, i.e., if any node on the left-path (resp., right-path) \( \pi[X, I] \) is replaced with the smaller (resp., larger) value which was earlier one level up on the same path.

**Property 5**: For any node \((r, k)\) of \( H \), shifting the left-path (resp., the right-path) of \( \mu(r, k) \) (resp., \( \nu(r, k) \)) one level up preserves the min-max-ordering.

Extending the PRAM algorithms in [6], the hypercube procedures are outlined below. (The deletion procedure being analogous to the deleteMax procedure, is omitted.)

**Procedure Insert \((H, L, x)\)**

\/* A new item \( x \) is inserted in the first vacant leaf \( L \) of the min-max-pair heap \( H \) of height \( h \) and with root \( R \) */

1. Locate the nodes \( P \) and \( Q \) at levels \( i \) and \( (i + 1) \) respectively, where \( 0 \leq i \leq h \), of the path \( \pi[R, L] \) such that either \( P\)min \( \leq x < Q\)min or \( P\)max \( \leq x < P\)max.
2. Shift one level down the left-path of \( \pi[Q, I] \) if \( P\)min \( \leq x < Q\)min. Or, shift one level down the right-path of \( \pi[P, I] \) if \( P\)max \( \leq x < P\)max.
3. Insert \( x \) in the empty position created at Step 2.
4. Recompute the min- and max-target-leaf information on \( \pi[R, L] \) by a suitable suffix-minima computation.

**Procedure Deletemax\((H)\)**

\/* The largest item on the min-max-pair heap \( H \) of height \( h \) and with root \( R \) is returned. */

1. Return \( R\)max.
2. Move one level up all the nodes on the right-path of the max-path \( \nu(R) \).
3. Extract the last item \( k \) from the rightmost non-empty leaf \( E \) of \( H \). If \( E \) is a max-element (resp., min-element), recompute the max-target-leaf (resp., the min-target-leaf) information on the path \( \pi[R, E] \).
4. Insert \((H, \Lambda(0, 0), k)\).

**Theorem 3** The insertion, deleteMin and deleteMax operations on a min-max-pair heap of height \( h \) and containing \( N \) items require \( O(\frac{h}{\log h} + \log p) \) parallel time on a hypercube. When \( p = O(\frac{\log h}{\log N}) = O(\frac{\log N}{\log \log N}) \), these operations take \( O(\log \log N) \) time and are cost-optimal.

**References**


