Constant Time Priority Queues
Constant Time Priority Queues

Priority Queues on Parallel Machines

Gerth S. Brodal

BRICS

Aarhus University
The Problem

Sequential Priority Queues

Parallel Machines

**Problem:** How can parallelism be exploited when performing single priority queue operations?
Operations

The common sequential priority queue operations

- MAKE QUEUE
- INSERT (Q,e)
- MELD (Q₁,Q₂)
- FIND MIN (Q)
- EXTRACT MIN (Q)
- DELETE (Q,e)
- DECREASE KEY (Q,e,e')
- BUILD (e₁,...,eₙ)
- MULTI INSERT (Q,e₁,...,eₙ)

Assume that a pointer to e is given

- creates an empty priority queue
- deletes the minimum element from Q.
- replaces e by a smaller element
- creates a priority queue containing e₁,...,eₙ
Model

- Comparison model
- Parallel machines: EREW or CREW PRAM
- $n$ denotes the maximum size of a priority queue
- $k$ number of elements involved in MultiINSERT
# Previous & New Results

<table>
<thead>
<tr>
<th>Model</th>
<th>Pinotti, Pucci 92 EREW</th>
<th>Pinotti, Pucci 91 CREW</th>
<th>Chen, Hu 94 EREW</th>
<th>Ranade, et al. 94 Array</th>
<th>This Talk CREW</th>
</tr>
</thead>
<tbody>
<tr>
<td>FindMin</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Insert</td>
<td>(\log \log n)</td>
<td>-</td>
<td>-</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ExtractMin</td>
<td>(\log \log n)</td>
<td>-</td>
<td>-</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Meld</td>
<td>-</td>
<td>(\log \frac{n}{k} + \log \log k)</td>
<td>(\log \log \frac{n}{k} + \log k)</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>Delete</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>Decrease Key</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>Build</td>
<td>(\log n)</td>
<td>(\frac{n}{k} \log k)</td>
<td>(\log \frac{n}{k} \log k)</td>
<td>-</td>
<td>(\log n)</td>
</tr>
<tr>
<td>Multi Insert</td>
<td>-</td>
<td>(\log \frac{n}{k} + \log k)</td>
<td>(\log \log \frac{n}{k} + \log k)</td>
<td>-</td>
<td>(\log k)</td>
</tr>
<tr>
<td>Multi Delete</td>
<td>-</td>
<td>(\log \frac{n}{k} + \log \Delta k)</td>
<td>(\log \log \frac{n}{k} + \log k)</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

*Delete the \(k\) smallest elements*

*Assume that \(k\) is fixed*

*Only one priority \(\Delta\) queue*
A Simple Priority Queue

Basic Idea: Represent a priority queue by a forest of heap ordered binomial trees
- A rank 0 tree is a single node
- A rank $r$ tree is obtained from two trees of rank $r-1$.

Primitive operations:
A Simple Priority Queue

The Invariant:
- There are 1, 2 or 3 trees of each rank
- The minimum root of rank \( i \) is smaller than all roots of rank \( \geq i \).

Example:

\[
3 \leq 7 \leq 8 \leq 11 \leq 15 \leq 22
\]
Parallel Linking and Unlinking

Notice: ParUnlink followed by ParLink is not the identity!
The Priority Queue Operations

**FindMin** (Q): return \( \min (Q.L[0]) \)

**Insert** (Q, e): \( Q.L[0] := Q.L[0] \cup \{e\} \)
\[ \text{ParLink} (Q) \]

**Meld** (Q₁, Q₂): for \( p := 0 \) to \( \log n \) pardo \( Q.L[p] := Q.L[p] \cup \{e\} \)
do 3 times \[ \text{ParLink} (Q) \]

**Extract Min** (Q): \( e := \min (Q.L[0]) \)
\( Q.L[0] := Q.L[0] \setminus \{e\} \)
\[ \text{ParUnlink} (Q) \]
\[ \text{ParLink} (Q) \]
return e

The above operations can all be implemented in constant time with log \( n \) processors on a CREW PRAM.
Building a Priority Queue

Step I: Build a normal Binomial Queue

Time $O(\log n)$ with $O(\frac{n}{\log n})$ processors on a EREW PRAM

Step II: for $i := 1$ to $\log n$ do
PARUNLINK,
PARLINK.

Time $O(\log n)$ with $O(\log n)$ processors on a EREW PRAM
Building a Priority Queue

Step I: Build a normal Binomial Queue

Time $O(\log n)$ with $O(\frac{n}{\log n})$ processors on a CREW PRAM

Step II: for $i = 1$ to $\log n$ do

PARUNLINK,
PARLINK.

Time $O(\log n)$ with $O(\log n)$ processors on a CREW PRAM

**MULTI INSERT**

**MULTI INSERT**($Q, e_1, ..., e_k$) = **MELD**($Q, \text{BUILD}(e_1, ..., e_k)$)

Time $O(\log k)$ with $O(\frac{\log n + k}{\log k})$ processors on a CREW PRAM
## Summary  
(simple priority queues)

<table>
<thead>
<tr>
<th>Model</th>
<th>CREW PRAM*</th>
<th>EREW PRAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIND MIN</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>INSERT, EXTRACT MIN, MELD</td>
<td>1</td>
<td>\log \log n</td>
</tr>
<tr>
<td>BUILD</td>
<td>\log n</td>
<td>\log n</td>
</tr>
<tr>
<td>MULTI INSERT</td>
<td>\log k</td>
<td>\log k + \log \log n</td>
</tr>
</tbody>
</table>

*The bounds also hold for the EREW PRAM, provided the processors know which priority queues are involved.*
**Summary**

(simple priority queues)

<table>
<thead>
<tr>
<th>Model</th>
<th>CREW PRAM*</th>
<th>EREW PRAM</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FindMin</strong></td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td><strong>Insert, ExtractMin, Meld</strong></td>
<td>1</td>
<td>log log n</td>
</tr>
<tr>
<td><strong>Build</strong></td>
<td>log n</td>
<td>log n</td>
</tr>
<tr>
<td><strong>MultiInsert</strong></td>
<td>log k</td>
<td>log k + log log n</td>
</tr>
</tbody>
</table>

*The bounds also hold for the EREW PRAM, provided the processors know which priority queues are involved.*
**MultiDelete**

**MultiDelete** was first considered by Pinotti, Pucci 91.

Basic idea:
- **Fix** $k$
- **Store** $k$ elements (in sorted order) in each node (instead of only one)
- All elements in a node are smaller than the elements at the sons.

Example: $k = 3$.

3, 7, 11

12, 15, 21
**Multi Delete**

Multi Delete was first considered by Pinotti, Pucci 91.

Basic idea:
- **Fix** $k$
- **Store** $k$ elements (in sorted order) in each node (instead of only one)
- All elements in a node are smaller than the elements at the sons.

Example: $k = 3$.

The merging of the two root sets can be done in time $O(\log \log k)$ on a CREW PRAM — Kruskal 83
Summary (Multi-Delete)

- Model: CREW PRAM

- Multi-Insert: $O(T_{sort}(k)) = O(\log k)$ - Cole 88

- Multi-Delete, Meld: $O(T_{merge}(k)) = O(\log \log k)$ - Kruskal 83

- Build: $O(T_{sort}(k) + \log \frac{n}{k} T_{merge}(k)) = O(\log k + \log \frac{n}{k} \log \log k)$
DELETE

It is not obvious how the simple priority queues can support DELETE operations in constant time!

Idea: Combine ideas from
- Fibonacci Heaps, Fredman, Tarjan 84
- Relaxed Heaps Driscoll et al. 88

Basic ideas:
- A node of rank $r$ has 3 sons of each rank 0, ..., $r-1$
- Subtrees can be missing - holes
- At most two holes have equal rank in a priority queue
Reducing The Number of Holes

Two holes of rank $r$ can be replaced by a rank $r+1$ hole.

Assume w.l.o.g. the two holes are brothers:

**Case I:**

$\begin{array}{c}
\text{1} \\
\text{2} \\
\text{3} \\
\text{4} \\
\text{5} \\
\text{6} \\
\text{7} \\
\text{8}
\end{array}$

$\begin{array}{c}
\text{1} \\
\text{2} \\
\text{3} \\
\text{4} \\
\text{5} \\
\text{6} \\
\text{7} \\
\text{8}
\end{array}$

**Case II:**

$\begin{array}{c}
\text{1} \\
\text{2}
\end{array}$

$\begin{array}{c}
\text{1} \\
\text{2}
\end{array}$

Performing the transformations in parallel for each possible $r \Rightarrow \#$holes of each rank is $\leq 2$. 
The DELETE operation

Step I: Cut off the subtree at the node to be deleted
- Creates a hole.

Step II: Perform once parallel FixHoles.
- Bounds the number of holes by two of each rank.

Step III: Perform PARLINK O(1) times
- Bounds the number of roots of each rank by O(1).

- DELETE can be performed in constant time with O(log n) processors on a CREW PRAM
# Summary

(simple priority queues)

<table>
<thead>
<tr>
<th>Model</th>
<th>CREW PRAM</th>
<th>EREW PRAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIND MIN</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>INSERT, EXTRACTMIN, MELD</td>
<td>1</td>
<td>log log n</td>
</tr>
<tr>
<td>BUILD</td>
<td>log n</td>
<td>log n</td>
</tr>
<tr>
<td>MULTI INSERT</td>
<td>log k</td>
<td>log k + log log n</td>
</tr>
<tr>
<td>DELETE, DECREASE KEY</td>
<td>1 *</td>
<td>(log log n)</td>
</tr>
</tbody>
</table>

*The bounds also hold for the EREW PRAM, provided the processors know which priority queues are involved.

*On a EREW this bound becomes log log log n
Conclusion

- Constant time priority queues
- Without DELETE the data structure is simple

Open Problems

- Multi DELETE faster / less work?
- — for non fixed k?
- DECREASE KEY (and MELD, INSERT) with O(1) work?
- Applications...
- Can the ideas related to DELETE be used to simplify Relaxed Heaps?