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 parallism be exploited when performing single priority queue operations?

Operations

The common sequential priority queue operations

MakeQueue — creates an empty priority queue

Insert(Q, e)

Meld(Q_1, Q_2)

FindMin(Q)

ExtractMin(Q) — deletes the minimum element from Q.

Assume that a pointer to e is given

Delete(Q, e)

DecreaseKey(Q, e, e') — replaces e by a smaller element

Build$(e_1, \dots, e_n)$ — creates a priority queue containing $e_1, \dots, e_n$

MultiInsert$(Q, e_1, \dots, e_k)$

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

Model	Pinotti, Pucci '92 EREW	Pinotti, Pucci '91 CREW	Chen, Hu '94 EREW	Ranade, et al. '94 Array	This Talk CREW
FindMin	1	1	1	1	1
Insert	$\log\log n$	-	-	1	1
ExtractMin	$\log\log n$	-	-	1	1
Meld	-	$\log \frac{n}{k} + \log\log k$	$\log\log \frac{n}{k} + \log k$	-	1
Delete	-	-	-	-	1
DecreaseKey	-	-	-	-	1
Build	$\log n$	$\frac{n}{k} \log k$	$\log \frac{n}{k} \log k$	-	$\log n$
MultiInsert	-	$\log \frac{n}{k} + \log k$	$\log\log \frac{n}{k} + \log k$	-	$\log k$
MultiDelete	-	$\log \frac{n}{k} + \log\log k$	$\log\log \frac{n}{k} + \log k$	-	-

Delete the k smallest elements

Assume that k is fixed

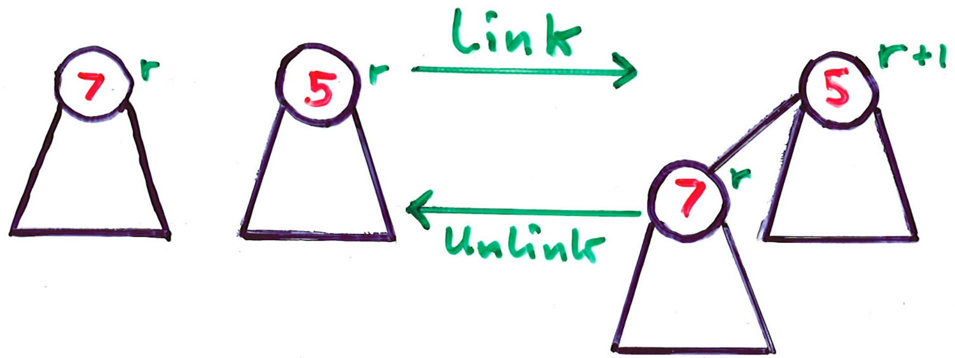
Only one priority queue

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:

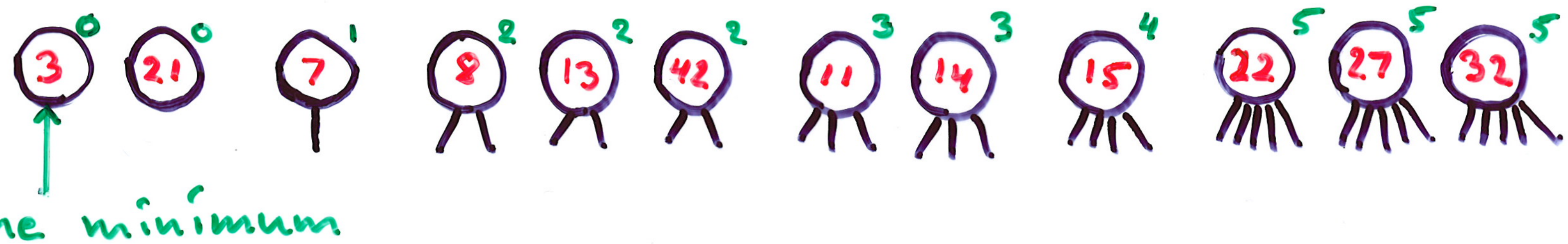


A Simple Priority Queue

The Invariant:

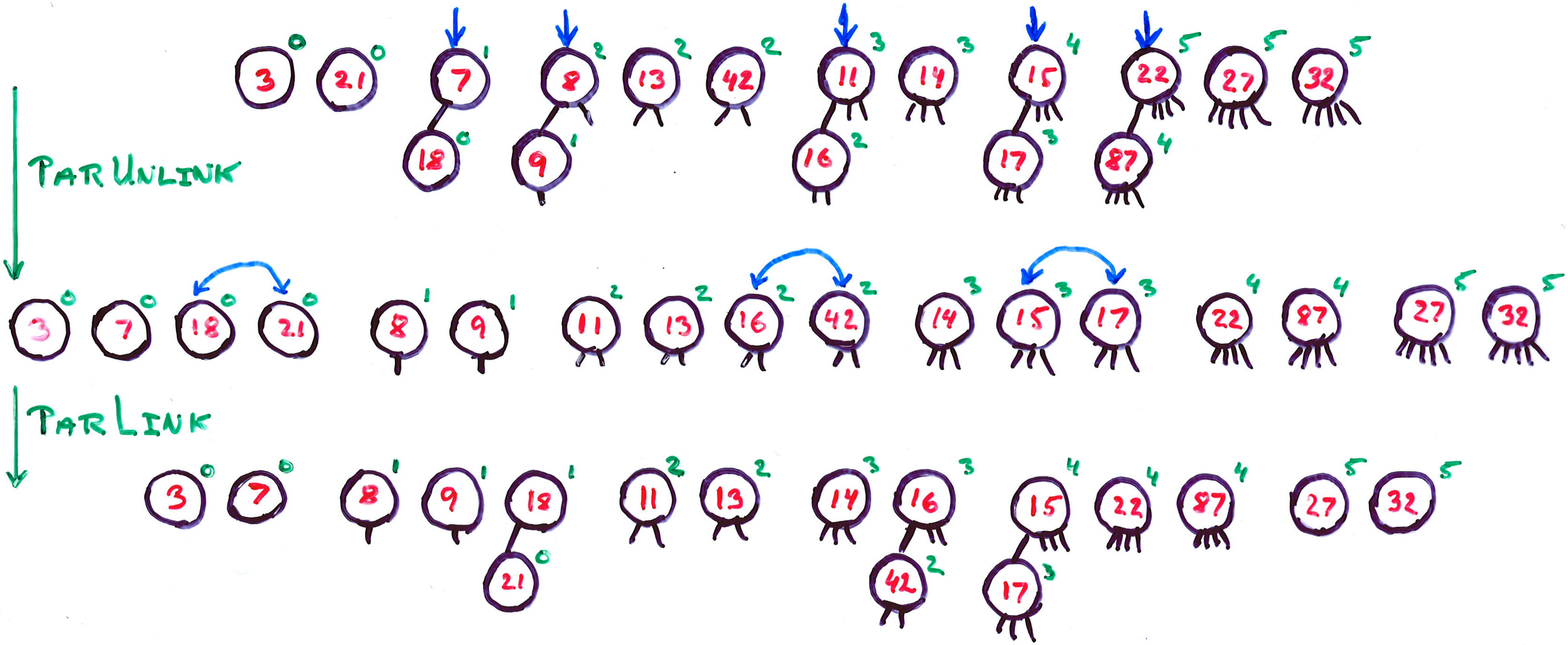
- There are 1, 2 or 3 trees of each rank
- The minimum root of rank i is smaller than all roots of rank $\geq i$.

Example:



$$3 \leq 7 \leq 8 \leq 11 \leq 15 \leq 22$$

Parallel Linking and Unlinking



Notice: PARUNLINK followed by PARLINK is <u>not</u> the identity!

The Priority Queue Operations

FindMin (Q):	return $\min(Q.L[0])$
Insert (Q, e):	$Q.L[0] := Q.L[0] \cup \{e\}$ ParLink (Q)
Meld (Q_1, Q_2):	for $p := 0$ to $\log n$ pardo $Q_1.L[p] := Q_1.L[p] \cup Q_2.L[p]$ do 3 times ParLink (Q_1)
ExtractMin (Q):	$e := \min(Q.L[0])$ $Q.L[0] := Q.L[0] \setminus \{e\}$ ParUnlink (Q) ParLink (Q) return e

The above operations can all be implemented in <u>constant time with $\log n$ processors</u> 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 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

MULTIINSERT

$$\text{MULTIINSERT}(Q, e_1, \ldots, e_k) \equiv \text{MELD}(Q, \text{BUILD}(e_1, \ldots, e_k))$$

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

Summary
(simple priority queues)

Model	CREW PRAM*	EREW PRAM
FindMin	1	1
Insert, ExtractMin, Meld	1	$\log\log n$
Build	$\log n$	$\log n$
MultiInsert	$\log k$	$\log k + \log\log n$

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

Summary
(simple priority queues)

Model	CREW PRAM*	EREW PRAM
FindMin	1	1
Insert, ExtractMin, Meld	1	$\log\log n$
Build	$\log n$	$\log n$
MultiInsert	$\log k$	$\log k + \log\log n$

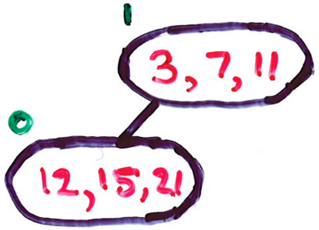
*The bounds also hold for the EREW PRAM, provided the processors know which priority queues are involved.

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: $k=3$.

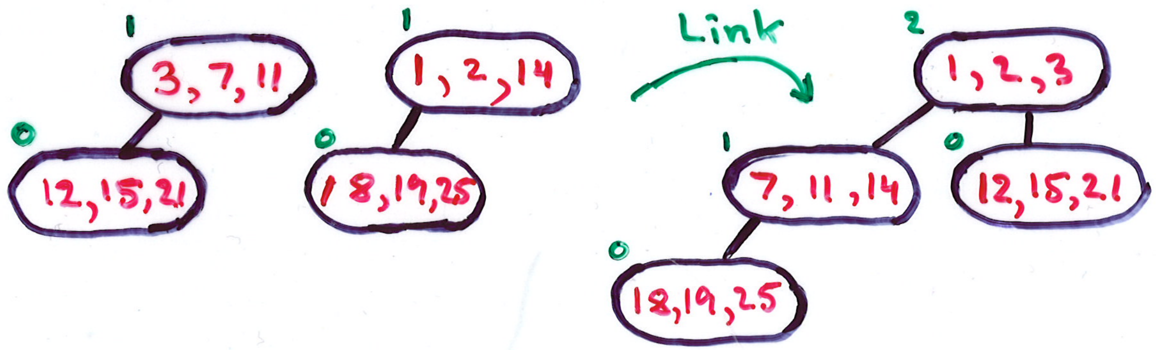


MultiDelete

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.



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

Summary
(MultiDelete)

- Model: CREW PRAM
- MultiInsert: $O(T_{SORT}(k)) = O(\log k)$ – Cole 88
- MultiDelete, 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

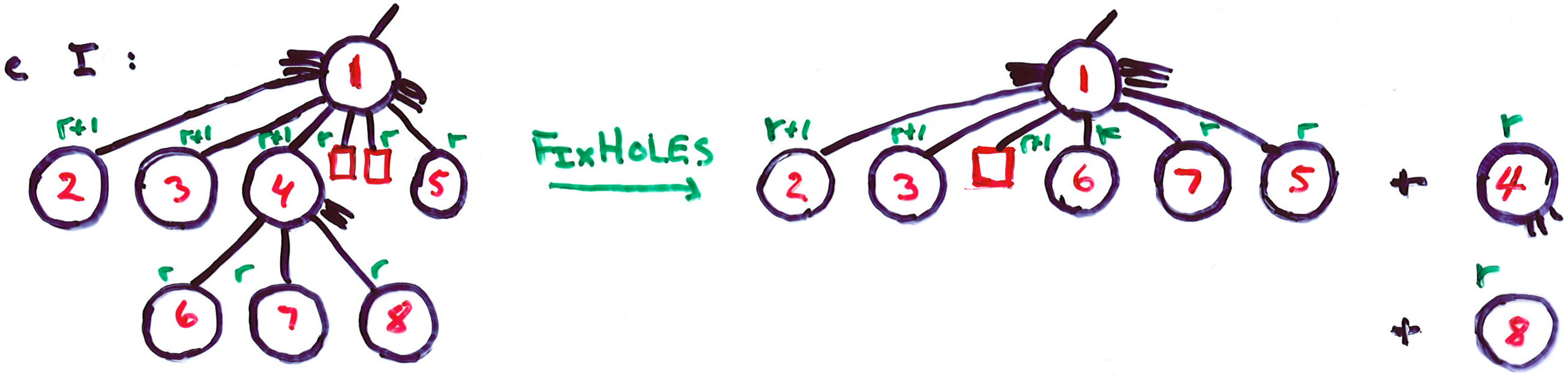
Basic ideas:
- A node of rang r has 3 sons of each rank 0,…, r-1
- Subtrees can be missing - holes
- At most two holes have equal rank in a priority queue

Reducing The Number of Holes

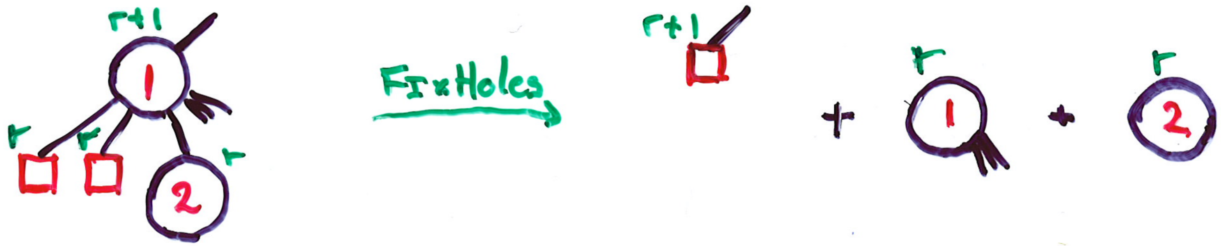
Two holes of rank r can be replaced by a rank r+1 hole

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

Case I :



Case II :



- Performing the transformations in parallel for each possible r $\Rightarrow$ # holes of each rank is ≤ 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)

Model	CREW PRAM*	EREW PRAM
FindMin	1	1
Insert, ExtractMin, Meld	1	$\log \log n$
Build	$\log n$	$\log n$
MultiInsert	$\log k$	$\log k + \log \log n$
Delete, DecreaseKey	1 *	$\log \log n$

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

* On a EREW this bound becomes $\log \log n$

Conclusion

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

Open Problems

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