UNIVERSITÄT KARLSRUHE
FAKULTÄT FÜR INFORMATIK

Randomized Priority Queues for Fast Parallel Access

Peter Sanders

Overview

- Model
- Approaches to Parallelization
- The Basic Algorithm
- Parallel Selection
- Refinements
- Implementation
- Conclusions and Future Work

The Problem

Scheduling of prioritized load units

Applications:

- Parallel branch-and-bound
- Parallel discrete event simulation
- Approximate sequential evaluation order
- ...

Algorithm Scheme (synchronous):

Initialize PQ
repeat
  delete "best" load units
  evaluate one load unit per PE
  insert new load units
until PQ empty

Notation

Machine Model

n**:** Number of PEs (processing elements); message passing

operations: Routing, broadcast, reduction, prefix sum and sorting of $\mathrm{O}(\sqrt{n})$ elements

d**:** Network diameter. Time $\mathrm{O}(d)$ Suffices for above operations on many networks.

Parallel Priority Queues

`deleteMin*` Delete n out of m *elements* with smallest *key*

`insert*` Insert ($\mathrm{O}(n)$) elements.

T_{access}**:=** $\max(T_{\texttt{deleteMin*}}, T_{\texttt{insert*}})$

Approaches to parallelization

Centralized Server

$T_{\text{access}} \in \Theta(n \log m)$ (e.g. using binary heap)

Pipelined Access

- $T_{\text{access}} \in \Theta(n)$ using $\log m$ PEs for queue

– Worsened network contention

n-Bandwidth Heap

[Deo/Prasad 92, Das/Pinotti/et.al., . . .]

- replace compare and exchange by parallel sorting and merging
- $T_{\text{access}} \in \Theta(\log m)$ for EREW-PRAMS

– slower on weaker models

– constant factors?

Using Random Placement

Approximated Priority Queues:

[Karp/Zhang/Ranade] (for parallel branch-and-bound)

- Send inserted elements to random PEs
- Delete from local Queues

– Different semantics ($\Rightarrow$ constant factor more node expansions for branch-and-bound)

Key Idea

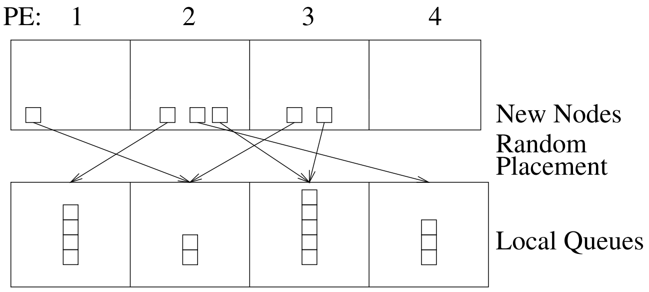
- Locate **globally** best elements for `deleteMin*`

$\rightarrow$ Parallel selection on $\tilde{O}\left(n\frac{\log n}{\log\log n}\right)$ elements suffices

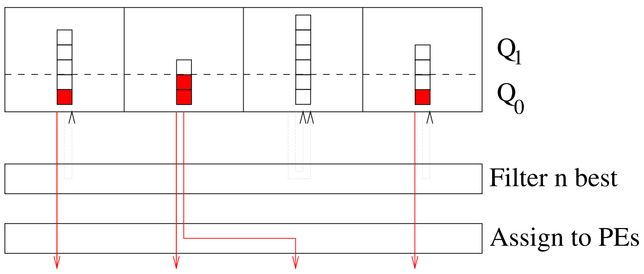
- Simple variants are already optimal for meshes [Sanders 9/94, Ranade et. al. 10/94]

The Basic Approach

`insert*`



`delMin*`



`insert*`

var i: Integer (* call counter *)
var Q_0: Sorted Array (* buffer *)
var Q_1: Heap (* main queue *)

Procedure `insert*`(e: Array of Element)
  send each e_i to a randomly selected PE
  insert received elements into Q_0
  $i := i + 1$; **if** $i \equiv 0 \bmod \log n$ **then** $Q_1 := Q_0 \cup Q_1$; $Q_0 := \emptyset$ **fi**

`deleteMin*`

var Q_0: Sorted Array (* buffer *)
var Q_1: Heap (* main queue *)

Function `deleteMin*`() : Element
  while $|\{e \in \breve{Q}_0 : e < \min \breve{Q}_1\}| < n$ **do** $Q_0 := Q_0 \cup \{\mathrm{deleteMin}(Q_1)\}$
  extract the n smallest elements in $\breve{Q}_0$
  enumerate the smallest elements $e_0, \ldots, e_{n-1}$
  send e_i to PE i and return it

Gory Details

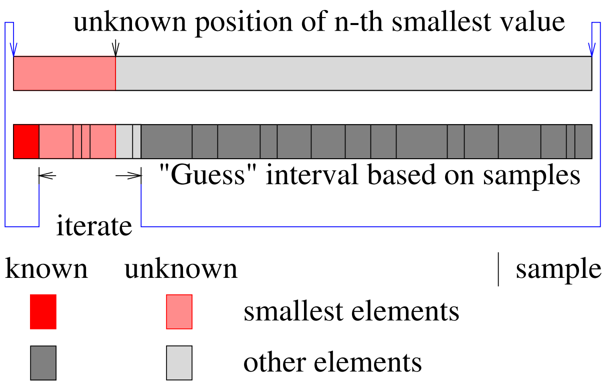
Q_0: Buffer for best elements

Q_1: Main store

- Move nodes to Q_0 (synchronously) until (**globally**) $\geq n$ elements are smaller than the minimum of the (**globally**) remaining elements
- Empty Q_0 every $\log n$ calls of `insert*`
- Only $\tilde{O}(1)$ accesses to Q_1 are required per call (amortized over $\log n$ calls)
- $|Q_0| \in \tilde{O}(\log n)$
- Assignment of results using prefix sums

A Special Selection Problem

Given $\tilde{O}(n \log n)$ randomly placed values.
Find the n-th smallest one.



- Sample size $O(\sqrt{n})$
- After $\tilde{O}(1)$ iterations full sort is easy
- No data redistribution necessary ($\neq$[Raj90])
- Beats worst case lower bound for selection [Pla89]

The Selection Algorithm

forall PEs **dopar** synchronously
  $Q' := Q_0$ (* solution candidates *)
  $m' := |\breve{Q}'|$ (* number of remaining candidates *)
  $Q_{\mathrm{out}} := \emptyset$ (* elements already found *)
  while $n' > 0 \wedge m' > n^{\frac{1}{2}} \wedge m' > n'$ **do**
    randomly choose $n^{\frac{1}{2}}$ samples from $\breve{Q}'$ with replacement
    let s_i denote the the i-th smallest sample (* sort them *)
      ($s_i = -\infty$ for $i \leq 0$, $s_i = \infty$ for $i > n^{\frac{1}{2}}$)
    $u := s_{\frac{n'}{m'} n^{\frac{1}{2}} + n^{\frac{1}{4}+\epsilon}}$ (* upper pivot *)
    $l := s_{\frac{n'}{m'} n^{\frac{1}{2}} - n^{\frac{1}{4}+\epsilon}}$ (* lower pivot *)
    if $|\{v \in \breve{Q}' : v < u\}| \geq n'$ **then**
      $Q' := Q' \setminus \{v \in Q' : v \geq u\}$
    if $|\{v \in \breve{Q}' : v \leq l\}| \leq n'$ **then**
      $Q' := Q' \setminus \{v \in Q' : v \leq l\}$
      $Q_{\mathrm{out}} := Q_{\mathrm{out}} \cup \{v \in Q' : v \leq l\}$
      $n' := n' - |\{v \in \breve{Q}' : v \leq l\}|$
    $m' := |\breve{Q}'|$
  if $m' \leq n'$ **then** $Q_{\mathrm{out}} := Q_{\mathrm{out}} \cup Q'$
  else if $n' > 0$ **then** (* $m' \leq n^{\frac{1}{2}}$; sorting is easy now *)
    sort $\breve{Q}'$ and insert the n' smallest elements into Q_{out}

Result

$$T_{\text{access}} \in \tilde{\mathrm{O}}\left(d + \log\frac{m}{n}\right)$$

for meshes, butterflies, ...

$$|Q_1| \le \frac{m}{n} + \tilde{\mathrm{O}}\left(\sqrt{\frac{m}{n}\log n} + \log n\right)$$

Simple Refinements

- Choose clever representation of local PQ.
- Only move references to nodes.

Coarser Grained Communication

- Assume that the n best elements are among the $2\frac{\log n}{\log\log n} + 10$ top elements of Q_0
- $\mathbf{P}$ [Fallback routine required] $\le 2 \cdot 10^{-9}$
- Selection using a single reduction of size $\mathrm{O}\left(\sqrt{n\log n}\right)$:
  - Merge sequences and throw away extreme elements
  - Works with high probability

Selection by Pruned Merging

Function select(k, m, k', s)
  (s, x, Δ, y, n') := globalReduce(prunedMerge$(k, m, k', \ldots)$,
    prune$(k, m, k', (s, -\infty, 0, \infty, 1)))$
  if $0 \le k - \Delta \le 2k' + 1 \wedge x \le s[i] \le y$ **then return** $s[k - \Delta]$
  else return slowSelect(k, m, s)

Function prunedMerge$(k, m, k', (s_1, x_1, \Delta_1, y_1, n_1'), (s_2, x_2, \Delta_2, y_2, n_2'))$
  return prune$(k, m, k'$,
    (merge(s_1, s_2), $\max(x_1, x_2)$, $\Delta_1 + \Delta_2$, $\min(y_1, y_2)$, $n_1' + n_2'$))

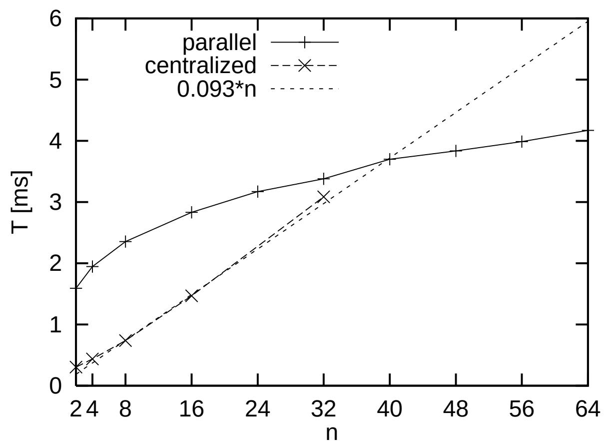
Function prune$(k, m, k', (s, x, \Delta, y, n'))$
  i := $\mathrm{round}\left(k\frac{n'}{n}\right) - \Delta$
  ... **(*** omitted cases where not both sides of the sequence need pruning ***)** ...
  return $(s[i - k', \ldots, i + k'], \max(x, s[i - k' - 1]), \Delta + i - k'$,
    $\min(y, s[i + k' + 1]), n')$

Asynchronous Operation

- Independent access ⇒ better load balancing
- Collective `deleteMin*` (in background) fills distributed FIFO
- Insertions invalidate FIFO content
  (but with some delay)
- Approximate selection suffices
- Waiting times can be overlapped

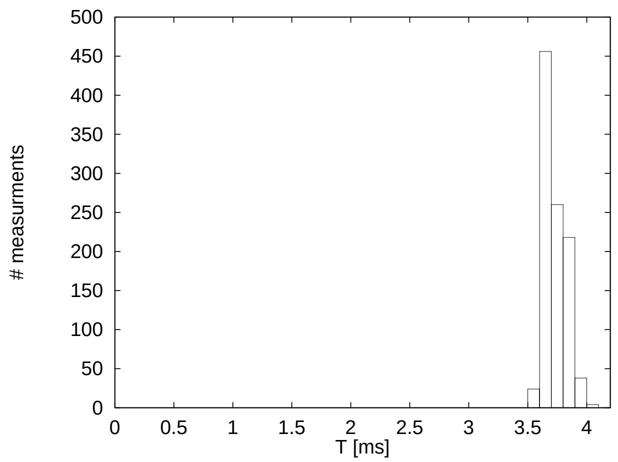
Implementation

- Portable C-code using MPI
- Optimizations for coarse grained operation
- Results on IBM SP:
  - $m = 2^{24}$
  - D-heap access time: $7\ \mu$s
  - Centralized Server: about $93\ \mu$s per access



Results on Cray T3D

- Same programs
- Centralized Server: about $110\ \mu$s per access
- $n = 256$
- Parallel algorithm ≈ 7.5 times faster



Conclusions

Parallel Priority Queues

- $T_{\text{access}} \in \tilde{\mathrm{O}}\left(d + \log \frac{m}{n}\right)$ (amortized)
- Very fast on constant degree log. diameter networks
- Memory efficient
- Machine independent
- Relatively simple
- Practical on mainstream parallel machines

Other results

- Fast parallel selection on randomly placed data
- Parallel selection using a single reduction
- Yields best known parallel algorithm for a class of branch-and-bound problems

Future Work

- Interaction with applications
  - Discrete event simulation
  - When is batched access useful?
  - Other choices for local queues
  - Additional operations
  - Integration with applications
- Implementation
  - Asynchronous algorithm
  - Larger machines
  - Faster machine interfaces
- More general results for parallel selection?