Introduction to HPC

Lecture 21

Lennart Johnsson
Dept of Computer Science

Sample Sort

- Randomly select a set of splitters (at least N-1 for N processors) locally
- Sort the splitters globally
- Assign elements to buckets locally
- Permute
- Local sort
Bucket expansion for sample sorting 10^9 keys on 1024 nodes as a function of oversampling ratio s. The two dashed curves show bucket expansion not to be exceeded by a probability of 0.999 and 0.999999 respectively. The solid curves show the maximum and average observed expansion over 1000 trials.

Sample sort time as a function of the oversampling ratio for 16384 keys per node of a 1024 node CM-2.
Sample sort execution time on a 1024 node CM-2 for 64-bit keys. Note broadcast time for the splitters is independent of the input size. For 4k keys per processors and beyond the oversampling ratio is increased from 32 to 64 in order to reduce the time for the local sort, which improves with reduced bucket expansion. The per key times for send and binary search remain constant.

Comparing 64-bit key sorting times on a 1024 node CM-2. Memory denotes the memory used by the algorithm relative to the original data. Rank denotes the time for rank relative to the time for sort.
Bitonic merge

Compare

Min

Max

Min

Max

UNIVERSITY of HOUSTON
Bitonic Merge

```
<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a1</td>
<td>A</td>
<td>L</td>
<td>c1</td>
</tr>
<tr>
<td>a2</td>
<td>B</td>
<td>H</td>
<td>c2</td>
</tr>
<tr>
<td>b1</td>
<td>A</td>
<td>L</td>
<td>c3</td>
</tr>
<tr>
<td>b2</td>
<td>B</td>
<td>H</td>
<td>c4</td>
</tr>
</tbody>
</table>
```

Complexity: $N/2\log_2 N$

---

Bitonic recursive merge

```
<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a1</td>
<td>A</td>
<td>L</td>
<td>c1</td>
</tr>
<tr>
<td>a2</td>
<td>B</td>
<td>H</td>
<td>c2</td>
</tr>
<tr>
<td>a3</td>
<td>A</td>
<td>L</td>
<td>c3</td>
</tr>
<tr>
<td>a4</td>
<td>B</td>
<td>H</td>
<td>c4</td>
</tr>
<tr>
<td>b1</td>
<td>A</td>
<td>L</td>
<td>c5</td>
</tr>
<tr>
<td>b2</td>
<td>B</td>
<td>H</td>
<td>c6</td>
</tr>
<tr>
<td>b3</td>
<td>A</td>
<td>L</td>
<td>c7</td>
</tr>
<tr>
<td>b4</td>
<td>B</td>
<td>H</td>
<td>c8</td>
</tr>
</tbody>
</table>
```

Complexity: $N/2\log_2 N$
Bitonic recursive merge

Complexity: $N/2 \log_2 N$

Bitonic Sort

Complexity:

Bitonic merge: $N \log_2 N$

Bitonic Sort: $N (\log_2 N)^2$
Bitonic Sort
Multiple elements per core

Cyclic allocation

First $k$ steps local: $2^{k-1}$ comparisons per core per step.
Last $n$ steps local: $2^k$ bitonic sequences each with one element per core.

Time: $k2^{k-1} + 2^n$ (or $\sim P/N \log_2(P/N) + P/N \log_2 N = P/N \log_2 P$).
Sequential Merge

Complexity: $P$ for $P$ elements

Bitonic Merge: $P \log_2 P$

Inefficient by a factor of $\log_2 P$
Valiant’s Parallel Merge Sort

- Merge two sorted sequences P and R in parallel by splitting each sequence evenly in sqrt size chunks.
- Sequence P: chunk size $\sqrt{P}$, $\sqrt{P}$ chunks
- Sequence R: chunk size $\sqrt{R}$, $\sqrt{R}$ chunks
- Merge the $\sqrt{P}$ and $\sqrt{R}$ splitters
- Insert the $\sqrt{P}$ splitters into the proper chunk of R
- Repeat the process recursively for each sublist created by the insertion

Valiant’s Parallel Merge sort

- Merge the $\sqrt{P} + \sqrt{R}$ splitters requires $\sqrt{P} \times \sqrt{R}$ comparisons which can be carried out on $\sqrt{P} \times \sqrt{R}$ cores in unit time.
- Second step the same …
- And the third ….. see notes
References


References (cont’d)