A Comparison of Sorting Algorithms for the Connection Machine CM-2

presented by Benjamin Black
Optimizing sorting on CM-2

- CM-2: hypercube network connect
- Algorithms to optimize
  - Bitonic sort
  - Sample sort
  - Radix sort
Primitives

- Arithmetic (Map)
- Send across network
- Scan (Cumulative sum)
- Cube swap (send along each dimension of hypercube)

Table 1. The time required for operations on a 32K Connection Machine CM-2. *

<table>
<thead>
<tr>
<th>Operation</th>
<th>Symbolic time</th>
<th>Actual time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arithmetic</td>
<td>$A \cdot (n/p)$</td>
<td>$1 \cdot (n/1024)$</td>
</tr>
<tr>
<td>Cube Swap</td>
<td>$Q \cdot (n/p)$</td>
<td>$40 \cdot (n/1024)$</td>
</tr>
<tr>
<td>Send (routing)</td>
<td>$R \cdot (n/p)$</td>
<td>$130 \cdot (n/1024)$</td>
</tr>
<tr>
<td>Scan (parallel prefix)</td>
<td>$3A \cdot (n/p) + S$</td>
<td>$3 \cdot (n/1024) + 50$</td>
</tr>
</tbody>
</table>

*The value $p$ is the number of processors (Sprint nodes), and $n$ is the total number of elements being operated on. All operations are on 64-bit words, except for scans which are on 32-bit words. All times are in microseconds.
Bitonic-sort

- Similar to merge-sort except every other sub-sequence is sorted in reverse order
- Key operation-bitonic merge
  - Naturally organized like a hypercube
  - Most efficient for small numbers of keys

Blelloch et. al. 1991
Bitonic sort optimizations

- Optimization pipelined-bitonic sort
  - Multiple keys per processor
  - Exchange all keys before they are needed
Radix sort

- Iterated bucket sort
- In place sort
- Need to know not only which bucket each key is in, but which rank the key is within each bucket
- Counting rank for each bucket be done with a Scan operation
Optimizing radix sort

- More efficient to calculate all ranks internally for each processor, combine across processors
- Choosing parameter $r$

\[
T_{\text{simple-rank}} = 2^r \cdot (3A \cdot (n/p) + S) + 2^r (2A)(n/p)
= A \cdot ((2 + 3)2^r(n/p)) + S \cdot 2^r,
\]

\[
T_{\text{rank}} = A \cdot (2 \cdot 2^r + 10(n/p)) + S \cdot 2^r,
\]

![Chart showing time per key per processor vs. bits per pass](chart.png)
Sample Sort

- Divide and conquer algorithm--Similar to quicksort
- Quicksort:
  - find 1 pivot value using a random strategy
  - Divide input below and above pivot point
  - Sort each half independently.
- Sample sort
  - Sample p-1 pivot values using a random strategy (p is the number of processors)
  - Sort pivot values
  - Send all pivot values to all processors
  - Divide input on each processors into the p buckets in parallel
    - binary search
  - Sort each bucket on each processor
Sample sort optimizations

- **Oversampling:**
  - To make evenly sized buckets, $s(p-1)$ samples are selected, where $p$ is the number of processors and $s$ is the oversampling ratio.
  - Pivots are the samples in the $s, 2s, 3s, \ldots (p-1)s$ ranks in the sample.
  - Chose value 64 for $s$ empirically for 16384 keys per processor.
Final results