Practical Massively Parallel Sorting

*Michael Axtmann, Timo Bingmann, Peter Sanders, Christian Schulz*

19th January 2016 @ Lecture Parallel Algorithms
Motivation

Example

- Space-filling curves for load balancing in supercomputers
- Relatively small input

Development over time: cores of the #1 supercomputer

Data source: TOP500 November 2014
\(p\)-way Sample Sort

- Input: large \(n\)
- Many processing elements (PE) \(p\)
- Delivering data once

G. E. Blelloch et al. 3rd SPAA
BSP Model

- Bulk synchronous
- Data exchange: $p$ startups in practice

<table>
<thead>
<tr>
<th>PE 0</th>
<th>PE 1</th>
<th>PE 2</th>
<th>PE 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>![Red]</td>
<td>![Green]</td>
<td>![Blue]</td>
<td>![Yellow]</td>
</tr>
</tbody>
</table>

Computation of elements

Communication

Synchronization
Massively Parallel Sorting Algorithms

- Very small data volume
- Logarithmic number of exchange phases
- Multi-level algorithms in the BSP model [2, 3]
- p-way parallel sample sort [4]

Merge sort, quick sort [1]

Worst case: receive $\Theta(p)$ messages

References:
BSP model generalization

- Data exchange function: \( \text{Exch}(p, h, r) \)
  
  - Involved PEs
  - Max send/receive volume per PE
  - Send/recv messages per PE

Diagram:

- Computation of elements
- Communication
- Synchronization

PE 0

PE 1

PE 2

PE 3

\( p \)  \( h \)  \( r \)
Comparison

Assumptions

- Number of levels $k \in \mathcal{O}(1)$
- Single-ported message passing
  - Sending of $\ell$ machine words: $\alpha + \beta \ell$

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Isoefficiency function</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p$-way parallel sample sort [1]</td>
<td>$\mathcal{O}(p^2 \cdot \frac{1}{\log p})$</td>
</tr>
<tr>
<td>Multi-level BSP-based [2,3]</td>
<td>$\Omega(p^2 \cdot \frac{1}{\log p})$ in our model</td>
</tr>
</tbody>
</table>

Comparison

Assumptions

- Number of levels $k \in \mathcal{O}(1)$
- Single-ported message passing
- Sending of $\ell$ machine words: $\alpha + \beta \ell$

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Isoefficiency function</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p$-way parallel sample sort [1]</td>
<td>$\mathcal{O}(p^2 \cdot \frac{1}{\log p})$</td>
</tr>
<tr>
<td>Multi-level BSP-based [2,3]</td>
<td>$\Omega(p^2 \cdot \frac{1}{\log p})$ in our model</td>
</tr>
<tr>
<td>Multi-level merge sort</td>
<td>$\mathcal{O}(p^{1+\frac{1}{k}} \cdot \log p)$</td>
</tr>
<tr>
<td>Multi-level sample sort</td>
<td>$\mathcal{O}(p^{1+\frac{1}{k}} \cdot \frac{1}{\log p})$</td>
</tr>
</tbody>
</table>

Multi-Level Sorting Approach

- Subdivide PEs into groups
- Move data to suitable group
- $k$ levels of recursion
  - Groups $r \approx \sqrt{p}$
Adaptive Multi-Level Sample Sort

Requirements

- Fast parallel sorting of samples
- Sample reduction by overpartitioning
- Reduce startup overheads to $O(k \sqrt{p})$
Adaptive Multi-Level Sample Sort

Requirements

- Fast parallel sorting of samples
- Sample reduction by overpartitioning
- Reduce startup overheads to $O(k \sqrt[p]{p})$
Adaptive Multi-Level Sample Sort

Requirements

- Fast parallel sorting of samples
- Sample reduction by overpartitioning
- Reduce startup overheads to $O(k \sqrt{p})$
Open submodules

1. Fast sample sorting
   - Oversampling
2. Optimal overpartitioning
3. Group-based data delivery
Fast Parallel Sample Sorting

- **Parallel sorting** of $s$ samples
- Rectangular $a \times b$ array of PEs

<table>
<thead>
<tr>
<th>PE 0</th>
<th>PE 1</th>
<th>PE 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>v x</td>
<td>d t</td>
<td>h g</td>
</tr>
<tr>
<td>PE 3</td>
<td>PE 4</td>
<td>PE 5</td>
</tr>
<tr>
<td>o q</td>
<td>r i</td>
<td>l w</td>
</tr>
<tr>
<td>PE 6</td>
<td>PE 7</td>
<td>PE 8</td>
</tr>
<tr>
<td>f m</td>
<td>p c</td>
<td>b k</td>
</tr>
</tbody>
</table>
Fast Parallel Sample Sorting

- Parallel sorting of $s$ samples
- Rectangular $a \times b$ array of PEs

Local sort

<table>
<thead>
<tr>
<th>PE 0</th>
<th>PE 1</th>
<th>PE 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
<td>d</td>
<td>g</td>
</tr>
<tr>
<td>x</td>
<td>t</td>
<td>h</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>PE 4</td>
<td>PE 5</td>
</tr>
<tr>
<td>o</td>
<td>i</td>
<td>l</td>
</tr>
<tr>
<td>q</td>
<td>r</td>
<td>w</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>PE 7</td>
<td>PE 8</td>
</tr>
<tr>
<td>f</td>
<td>c</td>
<td>b</td>
</tr>
<tr>
<td>m</td>
<td>p</td>
<td>k</td>
</tr>
</tbody>
</table>
Fast Parallel Sample Sorting

- **Parallel sorting** of $s$ samples
- Rectangular $a \times b$ array of PEs

Column-wise exchange merge

<table>
<thead>
<tr>
<th>PE 0</th>
<th>PE 1</th>
<th>PE 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f$ $m$ $o$ $q$ $v$ $x$</td>
<td>$c$ $d$ $i$ $r$ $p$ $t$</td>
<td>$b$ $g$ $h$ $k$ $l$ $w$</td>
</tr>
<tr>
<td>$x$ $v$</td>
<td>$d$ $t$</td>
<td>$g$ $h$</td>
</tr>
<tr>
<td>PE 3</td>
<td>PE 4</td>
<td>PE 5</td>
</tr>
<tr>
<td>$f$ $m$ $o$ $q$ $v$ $x$</td>
<td>$c$ $d$ $i$ $r$ $p$ $t$</td>
<td>$b$ $g$ $h$ $k$ $l$ $w$</td>
</tr>
<tr>
<td>$o$ $q$</td>
<td>$i$ $r$</td>
<td>$l$ $w$</td>
</tr>
<tr>
<td>PE 6</td>
<td>PE 7</td>
<td>PE 8</td>
</tr>
<tr>
<td>$f$ $m$ $o$ $q$ $v$ $x$</td>
<td>$c$ $d$ $i$ $r$ $p$ $t$</td>
<td>$b$ $g$ $h$ $k$ $l$ $w$</td>
</tr>
<tr>
<td>$m$ $f$</td>
<td>$p$ $c$</td>
<td>$k$ $b$</td>
</tr>
</tbody>
</table>
Fast Parallel Sample Sorting

- Parallel sorting of \( s \) samples
- Rectangular \( a \times b \) array of PEs

Row-wise exchange merge

<table>
<thead>
<tr>
<th>PE 0</th>
<th>PE 1</th>
<th>PE 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>f m o q v x</td>
<td>c d i r p t</td>
<td>b g h k l w</td>
</tr>
<tr>
<td>d g h t v x</td>
<td>d g h t v x</td>
<td>d g h t v x</td>
</tr>
<tr>
<td>PE 3</td>
<td>PE 4</td>
<td>PE 5</td>
</tr>
<tr>
<td>f m o q v x</td>
<td>c d i r p t</td>
<td>b g h k l w</td>
</tr>
<tr>
<td>i l o q r w</td>
<td>i l o q r w</td>
<td>i l o q r w</td>
</tr>
<tr>
<td>PE 6</td>
<td>PE 7</td>
<td>PE 8</td>
</tr>
<tr>
<td>f m o q v x</td>
<td>c d i r p t</td>
<td>b g h k l w</td>
</tr>
<tr>
<td>b c f k m p</td>
<td>b c f k m p</td>
<td>b c f k m p</td>
</tr>
</tbody>
</table>
**Fast Parallel Sample Sorting**

- **Parallel sorting of** $s$ **samples**
- **Rectangular** $a \times b$ **array of PEs**

### Rank column $i$ in row $j$

<table>
<thead>
<tr>
<th>PE 0</th>
<th>PE 1</th>
<th>PE 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 3 3 3 4 5</td>
<td>0 0 3 3 3 3</td>
<td>0 1 2 3 3 5</td>
</tr>
<tr>
<td>f m o q v x</td>
<td>c d i r p t</td>
<td>b g h k l w</td>
</tr>
<tr>
<td>d g h t v x</td>
<td>d g h t v x</td>
<td>d g h t v x</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PE 3</th>
<th>PE 4</th>
<th>PE 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 2 2 3 5 6</td>
<td>0 0 0 3 4 5</td>
<td>0 0 0 1 1 5</td>
</tr>
<tr>
<td>f m o q v x</td>
<td>c d i r p t</td>
<td>b g h k l w</td>
</tr>
<tr>
<td>i l o q r w</td>
<td>i l o q r w</td>
<td>i l o q r w</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PE 6</th>
<th>PE 7</th>
<th>PE 8</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 4 5 6 6 6</td>
<td>1 2 3 5 6 6</td>
<td>0 3 3 3 4 6</td>
</tr>
<tr>
<td>f m o q v x</td>
<td>c d i r p t</td>
<td>b g h k l w</td>
</tr>
<tr>
<td>b c f k m p</td>
<td>b c f k m p</td>
<td>b c f k m p</td>
</tr>
</tbody>
</table>

Row data column data local rank
Fast Parallel Sample Sorting

- Parallel sorting of $s$ samples
- Rectangular $a \times b$ array of PEs

Sum rank over column

<table>
<thead>
<tr>
<th>PE 0</th>
<th>PE 1</th>
<th>PE 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 9 10 12 15 17</td>
<td>1 2 6 11 13 14</td>
<td>0 4 5 7 8 16</td>
</tr>
<tr>
<td>f m o q v x</td>
<td>c d i r p t</td>
<td>b g h k l w</td>
</tr>
<tr>
<td>d g h t v x</td>
<td>d g h t v x</td>
<td>d g h t v x</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PE 3</th>
<th>PE 4</th>
<th>PE 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 9 10 12 15 17</td>
<td>1 2 6 11 13 14</td>
<td>0 4 5 7 8 16</td>
</tr>
<tr>
<td>f m o q v x</td>
<td>c d i r p t</td>
<td>b g h k l w</td>
</tr>
<tr>
<td>i l o q r w</td>
<td>i l o q r w</td>
<td>i l o q r w</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PE 6</th>
<th>PE 7</th>
<th>PE 8</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 9 10 12 15 17</td>
<td>1 2 6 11 13 14</td>
<td>0 4 5 7 8 16</td>
</tr>
<tr>
<td>f m o q v x</td>
<td>c d i r p t</td>
<td>b g h k l w</td>
</tr>
<tr>
<td>b c f k m p</td>
<td>b c f k m p</td>
<td>b c f k m p</td>
</tr>
</tbody>
</table>
Fast Parallel Sample Sorting

Single-ported message passing
- Sending of $\ell$ machine words: $\alpha + \beta \ell$

- Local sort
- Column-wise allgather merge
- Row-wise allgather merge
- Rank column $i$ in row $j$
- Sum rank over column

$O(\alpha \log p + \beta \frac{s}{\sqrt{p}} + \frac{s}{p} \log \frac{s}{p})$
Adaptive Multi-Level Sample Sort

Open submodules

1. Fast sample sorting
   - Oversampling
2. Optimal overpartitioning
3. Group-based data delivery

Diagram showing partitioning by sampling.
Optimal Overpartitioning

- Requirement: $L_{max} = (1 + \epsilon) \frac{n}{r}$ with high probability
- Oversampling: $a$
- Overpartitioning: $b \in \Theta(\frac{1}{\epsilon})$
Optimal Overpartitioning

- Requirement: $L_{\text{max}} = (1 + \epsilon) \frac{n}{r}$ with high probability
- Oversampling: $a$
- Overpartitioning: $b \in \Theta\left(\frac{1}{\epsilon}\right)$
Optimal Overpartitioning

- Requirement: \( L_{\text{max}} = (1 + \epsilon) \frac{n}{r} \) with high probability
- Oversampling: \( a \)
- Overpartitioning: \( b \in \Theta\left(\frac{1}{\epsilon}\right) \)

![Diagram showing global partitions and greedy assignment.](image)
Optimal Overpartitioning

- Requirement: $L_{\text{max}} = (1 + \epsilon) \frac{n}{r}$ with high probability
- Oversampling: $a$
- Overpartitioning: $b \in \Theta(\frac{1}{\epsilon})$
Optimal Overpartitioning

- Requirement: $L_{\text{max}} = (1 + \epsilon) \frac{n}{r}$ with high probability
- Oversampling: $a$
- Overpartitioning: $b \in \Theta\left(\frac{1}{\epsilon}\right)$
- Fewer samples $abr \in \Theta\left(r \log r\right)$

\[ \mathcal{O}(br + \alpha \log p) \]
Open submodules

1. Fast sample sorting
   - Oversampling
2. Optimal overpartitioning
3. Group-based data delivery
Group-Based Data Delivery

Goal

- **Partition** \( i \) to group \( i \)
- Each PE in group receives **same amount** of data
- \((1 + o(1)) \text{Exch}(p, \frac{n}{p}, O(r))\)

- Send/recv messages per PE
- Max send/receive volume per PE
- Involved PEs
- **Reduce startup overheads** to \( O(\sqrt{p}) \)

Diagram showing partitioning by sampling between Group 0 and Group 1.
Group-Based Data Delivery

Group 0

[Diagram showing groups divided into segments with different colors]

Group 1

[Diagram showing groups divided into segments with different colors]
Group-Based Data Delivery

Trivial approach

Group 0

Group 1
Group-Based Data Delivery

Our approach

- Distribution of small pieces $|\cdot| \leq \frac{n}{2pr}$ // round-robin
Group-Based Data Delivery

Our approach

Distribution of small pieces $| \cdot | \leq \frac{n}{2pr}$ // round-robin
Group-Based Data Delivery

Our approach

- Distribution of small pieces \( \cdot \leq \frac{n}{2pr} \) // round-robin
- Distribution of large pieces // prefix-sum and merging
Group-Based Data Delivery

Our approach

- Distribution of small pieces $| \cdot | \leq \frac{n}{2pr}$ // round-robin
- Distribution of large pieces // prefix-sum and merging
Group-Based Data Delivery

Our approach

- Distribution of small pieces $\mid \cdot \mid \leq \frac{n}{2pr}$ // round-robin
- Distribution of large pieces // prefix-sum and merging

Reduces startup overheads to $O(\sqrt[4]{p})$
Recurse Last Multiway Mergesort

Highlights

- Multisequence selection
- Perfect load balance
- Reduces startup overheads to $O(k \sqrt[4]{p})$

Group 0

PE 0

PE 1

partitioning by sampling

Splitter

recurse on $\frac{n}{r}$ items with $\frac{n}{r}$ PEs

Group 1

PE 2

PE 3

recurse on $\frac{n}{r}$ items with $\frac{n}{r}$ PEs
Experiments
SuperMUC in Munich

2 Intel Xeon E5-2680 8-core

pruned tree (4 : 1 bandwidth ratio)

Cores used: 32 768 (4 islands)
## Experiments

### Sample sort median wall-times in seconds

<table>
<thead>
<tr>
<th>$n/p$</th>
<th>$512$</th>
<th>$2 048$</th>
<th>$8 192$</th>
<th>$32 768$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$10^5$</td>
<td>0.0228</td>
<td>0.0277</td>
<td>0.0359</td>
<td>0.0707</td>
</tr>
<tr>
<td>$10^6$</td>
<td>0.2212</td>
<td>0.2589</td>
<td>0.2687</td>
<td>0.9171</td>
</tr>
<tr>
<td>$10^7$</td>
<td>2.6523</td>
<td>2.9797</td>
<td>4.0625</td>
<td>6.0932</td>
</tr>
</tbody>
</table>

### Speedup of sample sort compared to sequential sort

<table>
<thead>
<tr>
<th>$n/p$</th>
<th>$512$</th>
<th>$2 048$</th>
<th>$8 192$</th>
<th>$32 768$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$10^5$</td>
<td>273</td>
<td>956</td>
<td>3 208</td>
<td>6 929</td>
</tr>
<tr>
<td>$10^6$</td>
<td>321</td>
<td>1 146</td>
<td>4 747</td>
<td>6 164</td>
</tr>
<tr>
<td>$10^7$</td>
<td>295</td>
<td>1 124</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>

- **Level 1**
- **Level 2**
- **Level 3**
Comparison to Literature

Solomonik and Kale [1]: CrayXT 4
- Slower processors, higher bandwidth
- \( n = 8 \cdot 10^6 p \), up to \( p = 2^{15} \)
  - Similar performance

MP-sort [2]: Cray XE6
- \( n = 10^5 p \) and \( p = 2^{14} \)
  - 289 times faster
- 6 times faster for large inputs

// vs. \( n_{\text{ref}} = 10^7 p \)
// vs. \( p_{\text{ref}} = 2^{15} \)

Conclusion

Result
- Scalable in theory and practice
- Improved wall-time: large $p$ and moderate $n$
- Competitive: large $p$ and large $n$

Future work
- Perform experiments with more PEs
- Shared memory on node-local level
- Better exchange algorithms
- Fault tolerance
Acknowledgement

The authors gratefully acknowledge the Gauss Centre for Supercomputing e.V. (www.gauss-centre.eu) for funding this project by providing computing time on the GCS Supercomputer SuperMUC at Leibniz Supercomputing Centre (LRZ, www.lrz.de).