Histogram Sort with Sampling (HSS)

Vipul Harsh  Laxmikant Kale  Edgar Solomonik

University of Illinois at Urbana-Champaign
June 23, 2019. SPAA 2019
Parallel sorting: Problem

Input

- $p$ processors
- $N/p$ keys per processor
- Arbitrary distribution of keys

Goal

- Shuffle data to ensure global sorted order
  - keys on processor $i <$ keys on processor $(i+1)$
Parallel sorting requirements

- Load balance across processors after sorting
  - No one gets more than \((1+\epsilon)N/p\) keys

- Minimal data movement

- Robust guarantees (independent of distribution)

- Scalability, performance
BSP cost model (Valiant)

- Execution split across supersteps
- In a superstep, a processor can exchange messages and perform local computation
- Complexity: Work done across all supersteps
  \[ T = O(S \alpha + M \beta + T_{\text{comp}} \gamma) \]
- S: number of supersteps, M: size of messages exchanged, 
  \( T_{\text{comp}} \): Local computation time
Parallel sorting: Past work

- **Merge based**
  - Bitonic sort [Batcher, AFIPS ‘68]
  - AKS network [Ajtai, STOC’83]
  - Cole’s Merge tree [Cole, J. Comp.’88]

- **Partition based**
  - Sample sort [Frazer, JACM’70]
  - Histogram sort [Kale, ICPP’93]
  - HykSort [Sundar ICS’13]
  - AMS sort [Axtmann, SPAA’15]
Parallel sorting: Past work

- **Merge based**
  - Bitonic sort [Batcher, AFIPS ‘68]
  - AKS network [Ajtai, STOC’83]
  - Cole’s Merge tree [Cole, J. Comp.’88]

- **Partition based**
  - Sample sort [Frazer, JACM’70]
  - Histogram sort [Kale, ICPP’93]
  - HykSort [Sundar ICS’13]
  - AMS sort [Axtmann, SPAA’15]

Suboptimal $\Omega(N \log p)$ data movement
Parallel sorting: Past work

- **Merge based**
  - Bitonic sort [Batcher, AFIPS ‘68]
  - AKS network [Ajtai, STOC’83]
  - Cole’s Merge tree [Cole, J. Comp.’88]

- **Partition based**
  - Sample sort [Frazer, JACM’70]
  - Histogram sort [Kale, ICPP’93]
  - HykSort [Sundar ICS’13]
  - AMS sort [Axtmann, SPAA’15]

Suboptimal $\Omega(N \log p)$ data movement

- Optimal $\Theta(N)$ data movement
- High partitioning cost
Partition based sorting: A basic template
Partition based sorting: A basic template

1. Local sorting: processors sort local data
Partition based sorting: A basic template

1. Local sorting: processors sort local data

2. Data partition: determine \((p-1)\) keys to partition key range into \(p\) buckets
Partition based sorting: A basic template

1. Local sorting: processors sort local data

2. Data partition: determine \((p-1)\) keys to partition key range into \(p\) buckets

3. Data exchange: send keys to their destination processors
1. Local sorting: processors sort local data

2. Data partition: determine \((p-1)\) keys to partition key range into \(p\) buckets

3. Data exchange: send keys to their destination processors

Focus of our paper; Crucial for load balance, performance
Sample sort
Sample sort

P₁  P₂  Pₚ  Processors
Sample sort

Processors

Local samples

P₁

P₂

Pₚ

Processors

15
Sample sort

Combined sorted sample

Local samples

Processors

\( P_1 \)

\( P_2 \)

\( P_p \)
Sample sort

Equally spaced \((p-1)\) splitter keys

\[\begin{align*}
S_1 & \quad S_2 & \quad S_{p-1}
\end{align*}\]

Combined sorted sample

\[\begin{align*}
P_1 & \quad P_2 & \quad \cdots & \quad P_p
\end{align*}\]

Local samples

Processors
Sample sort

Equally spaced \((p-1)\) splitter keys

\[\begin{align*}
S_1 & \quad S_2 & \quad S_{p-1}
\end{align*}\]

Combined sorted sample

Requires \(\Theta(p(\log p)/\epsilon^2)\) samples. Too costly for large scale

\[\begin{align*}
P_1 & \quad P_2 & \quad P_p
\end{align*}\]

Local samples

Processors
HSS: Key idea

1. Sampling: Sample a small number of keys

2. Histogramming: Determine rank of all sampled keys
   - via a global reduction
   - same complexity as sampling with pipelined reduction

3. If sample contains satisfactory splitters, return
   - Else, sample next set of keys and repeat (2)
HSS: Sampling methodology

- Maintain “splitter” intervals around $\frac{N}{p}, \frac{2N}{p}, \ldots, \frac{iN}{p}, \ldots$
  - Use closest key (left and right) in sample so far
  - For splitter $i$: $[L_i, R_i]$
HSS: Sampling methodology

- Maintain “splitter” intervals around $\frac{N}{p}, \frac{2N}{p}, \ldots, \frac{iN}{p}, \ldots$
  - Use closest key (left and right) in sample so far
  - For splitter $i$: $[L_i, R_i]$

- Update splitter intervals after every histogramming round
  - $L_i = \max_{\text{rank}(k) \leq Ni/p} k$
  - $R_i = \min_{\text{rank}(k) \geq Ni/p} k$

Sample only within splitter intervals, proportional to interval size
HSS: Splitter intervals over time

N/p  2N/p  3N/p  4N/p

Ideal splitters
HSS: Splitter intervals over time

- N/p
- 2N/p
- 3N/p
- 4N/p

Ideal splitters

After first round of histogramming
HSS: Splitter intervals over time

- N/p
- 2N/p
- 3N/p
- 4N/p

Ideal splitters

After first round of histogramming

Splitter intervals after first round
HSS: Splitter intervals over time

- **Ideal splitters**
- After first round of histogramming
- Splitter intervals after first round
- After second round of histogramming

- N/p
- 2N/p
- 3N/p
- 4N/p
HSS: Splitter intervals over time

Ideal splitters

After first round of histogramming

Splitter intervals after first round

After second round of histogramming

Splitter intervals after second round
Analysis

- Size of union of splitter intervals decreases exponentially w.h.p.
- $O(\log((\log p)/\epsilon))$ rounds suffice for high probability bounds
- $k$: total rounds, $\epsilon$: Desired $(1\pm \epsilon)$ load balance
HSS vs other partitioning schemes

- Sample sort
  - Sample size: $O(p(\log p)\varepsilon^{-2})$ vs $O(p \log(\varepsilon^{-1} \log p))$ in HSS
HSS vs other partitioning schemes

- Sample sort
  - Sample size: $O(p(\log p)\epsilon^{-2})$ vs $O(p \log(\epsilon^{-1} \log p))$ in HSS

- Histogram sort
  - Suboptimal worst case guarantees that are loose
  - Not comparison based
HSS vs other partitioning schemes

- Sample sort
  - Sample size: $O(p(\log p)\epsilon^{-2})$ vs $O(p \log(\epsilon^{-1} \log p))$ in HSS

- Histogram sort
  - Suboptimal worst case guarantees that are loose
  - Not comparison based

- AMS sort: Larger sample size due to non-iterative partitioning
  - Sample size: $O(p(\log p + \epsilon^{-1}))$ vs $O(p \log(\epsilon^{-1} \log p))$ in HSS
HSS vs other partitioning schemes

- Sample sort
  - Sample size: $O(p(\log p)\epsilon^{-2})$ vs $O(p \log(\epsilon^{-1} \log p))$ in HSS

- Histogram sort
  - Suboptimal worst case guarantees that are loose
  - Not comparison based

- AMS sort: Larger sample size due to non-iterative partitioning
  - Sample size: $O(p(\log p + \epsilon^{-1}))$ vs $O(p \log(\epsilon^{-1} \log p))$ in HSS

- HykSort: Suboptimal sampling methodology
  - requires $\Omega(\log p / \log \log p)$ rounds, compared to $O(\log \log p)$ in HSS
Cost analysis

- Cost of sampling (and sorting) S keys: $O(S \log N)$

- Cost of computing rank of S keys: $O(S \log N)$
  - via global pipelined reductions [Thakur, PVM/MPI’03]
### Results summary

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Cost Complexity</th>
<th>BSP Supersteps</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sample Sort</td>
<td>$O(p \log p \log N /\varepsilon^2)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>AMS Sort</td>
<td>$O(p (\log p + \varepsilon^{-1}) \log N)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>HSS with $k$ rounds</td>
<td>$O(pk^{k/\sqrt{\log p / \varepsilon \log N}})$</td>
<td>$O(k)$</td>
</tr>
<tr>
<td>HSS optimal cost</td>
<td>$O(p \log(\varepsilon^{-1} \log p) \log N)$</td>
<td>$O(\log(\varepsilon^{-1} \log p)$</td>
</tr>
</tbody>
</table>
Experimental results
Number of histogramming rounds

Gradual increase in number of rounds for HSS; $O(\log \log p)$
Overall performance vs Hyksort

Margin of improvement increases with scale
Multi-stage sorting

For very large scale, data exchange can be inefficient, $O(p^2)$ messages
Multi-stage sorting

For very large scale, data exchange can be inefficient, $O(p^2)$ messages

- Split data exchange into $k$ (typically $\leq 2$) phases
- In phase 1, exchange data across $r$ processor groups
  - 1 processor group = $p/r$ processors
- In phase 2, recursively sort data within each processor group
Multi-stage sorting

Modest improvement in overall running time
Real world application: ChaNGa

~25% improvement in sorting time
Robust to input distributions

Running time invariant to input distributions
Conclusion

- Scalable, efficient large scale parallel sorting
  - Near linear time (in $p$) data-partitioning
- Guarantees carry over for multi-stage sorting
- Experimental results show benefit over other algorithms
  - Over Hyksort for single-staged and multi-staged settings
  - Over Histogram sort in a real application
Open questions

- Tight lower bound on cost of partitioning
- Trade-off between supersteps and cost
  - Can we achieve similar cost with fewer rounds
Acknowledgements

- Omkar Thakoor
- Nitin Bhat
- Harshita Prabha
- Umang Mathur
- Members of PPL@UIUC
- Anonymous SPAA reviewers
- ALCF and TACC supercomputing centers
### Results summary

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Cost Complexity</th>
<th>BSP Supersteps</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sample Sort</td>
<td>$O(p \log p \log N /\epsilon^2)$</td>
<td>$0(1)$</td>
</tr>
<tr>
<td>AMS Sort</td>
<td>$O(p (\log p + \epsilon^{-1}) \log N)$</td>
<td>$0(1)$</td>
</tr>
<tr>
<td>HSS with $k$ rounds</td>
<td>$O(pk^{\frac{k}{\sqrt{\log p / \epsilon \log N}}})$</td>
<td>$0(k)$</td>
</tr>
<tr>
<td>HSS optimal cost</td>
<td>$O(p \log(\epsilon^{-1} \log p) \log N)$</td>
<td>$0(\log(\epsilon^{-1} \log p))$</td>
</tr>
</tbody>
</table>
Backup slides
HSS: Main result

Theorem 4.7: With $k$ rounds of histogramming and a sample size $O\left(p^k \sqrt{\frac{\log p}{\epsilon}}\right)$ per round, HSS achieves load $(1 + \epsilon)$ balance w.h.p. for large enough $p$.

$k = O(\log((\log p)/\epsilon))$ gives optimal cost
On the size of splitter intervals

- Size of union of splitter intervals decreases exponentially w.h.p.
  - In round $j$ as a fraction of input size, $f_j = O\left(\left(\frac{2 \log p}{\epsilon}\right)^{-\frac{(j-1)}{k}}\right)$

- $O(\log((\log p)/\epsilon))$ rounds suffice for high probability bounds

- $k$: total rounds, $\epsilon$: Desired $(1\pm\epsilon)$ load balance
Bounding number of rounds

- Union of sample intervals $f_j$ decreases exponentially
- Sampling density increases
HSS: Main result

Theorem 4.7: With $k$ rounds of histogramming and a sample size $O\left(p^k \sqrt{\frac{\log p}{\epsilon}}\right)$ per round, HSS achieves load $(1 + \epsilon)$ balance w.h.p. for large enough $p$.

- Substituting $k = O\left(\log\left(\frac{\log p}{\epsilon}\right)\right)$ has optimal cost

Theorem 4.8: With $O\left(\log\frac{\log p}{\epsilon}\right)$ rounds of histogramming and a sample size $O(p)$ per round ($O(1)$ per processor), HSS achieves load $(1 + \epsilon)$ balance for large enough $p$. 
## HSS: Sample size

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Overall sample size for $p = 10^5, \epsilon = 5%$</th>
<th>Overall sample size</th>
<th>Computation complexity</th>
<th>Communication complexity</th>
<th>Supersteps</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regular sampling</td>
<td>$O\left(\frac{p^2}{\epsilon}\right)$</td>
<td>1600 GB</td>
<td>$O\left(\frac{p^2}{\epsilon} \log p \log p\right)$</td>
<td>$O\left(\frac{p^2}{\epsilon}\right)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Random sampling</td>
<td>$O\left(\frac{p \log N}{\epsilon^2}\right)$</td>
<td>8.1 GB</td>
<td>$O\left(\frac{p \log N \log p}{\epsilon^2}\right)$</td>
<td>$O\left(\frac{p \log N}{\epsilon^2}\right)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Single stage AMS sort</td>
<td>$O\left(p (\log p + \frac{1}{\epsilon})\right)$</td>
<td>32 MB</td>
<td>$O\left(p (\log p + \frac{1}{\epsilon}) \log N\right)$</td>
<td>$O\left(p (\log p + \frac{1}{\epsilon})\right)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>HSS with one round</td>
<td>$O\left(\frac{p \log p}{\epsilon}\right)$</td>
<td>184 MB</td>
<td>$O\left(\frac{p \log p}{\epsilon} \log N\right)$</td>
<td>$O\left(\frac{p \log p}{\epsilon}\right)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>HSS with two rounds</td>
<td>$O\left(p \sqrt{\frac{\log p}{\epsilon}}\right)$</td>
<td>24 MB</td>
<td>$O\left(p \sqrt{\frac{\log p}{\epsilon}} \log N\right)$</td>
<td>$O\left(p \sqrt{\frac{\log p}{\epsilon}}\right)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>HSS with $k$ rounds</td>
<td>$O\left(kp \sqrt{\frac{\log p}{\epsilon}}\right)$</td>
<td>-</td>
<td>$O\left(kp \sqrt{\frac{\log p}{\epsilon}} \log N\right)$</td>
<td>$O\left(kp \sqrt{\frac{\log p}{\epsilon}}\right)$</td>
<td>$O(k)$</td>
</tr>
<tr>
<td>HSS with $O(1)$ samples/proc/round</td>
<td>$O\left(p \log \frac{\log p}{\epsilon}\right)$</td>
<td>10 MB</td>
<td>$O\left(p \log \frac{\log p}{\epsilon} \log N\right)$</td>
<td>$O\left(p \log \frac{\log p}{\epsilon}\right)$</td>
<td>$O(\log \frac{\log p}{\epsilon})$</td>
</tr>
</tbody>
</table>