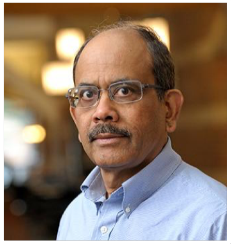
Histogram Sort with Sampling (HSS)


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

$$\mathrm{T} = \mathrm{O}(S\,\alpha + \mathrm{M}\,\beta + \mathrm{T}_{\mathrm{comp}}\gamma)$$

- S: number of supersteps, M: size of messages exchanged, $\mathrm{T}_{\mathrm{comp}}$: 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]

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

- 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]

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

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

- Optimal $O(N)$ 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

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

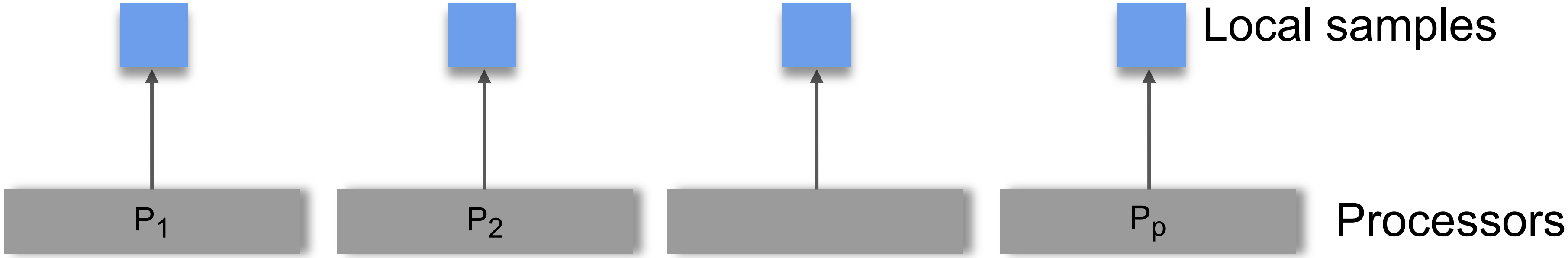
3. Data exc
processo

Focus of our paper;
Crucial for load balance, performance

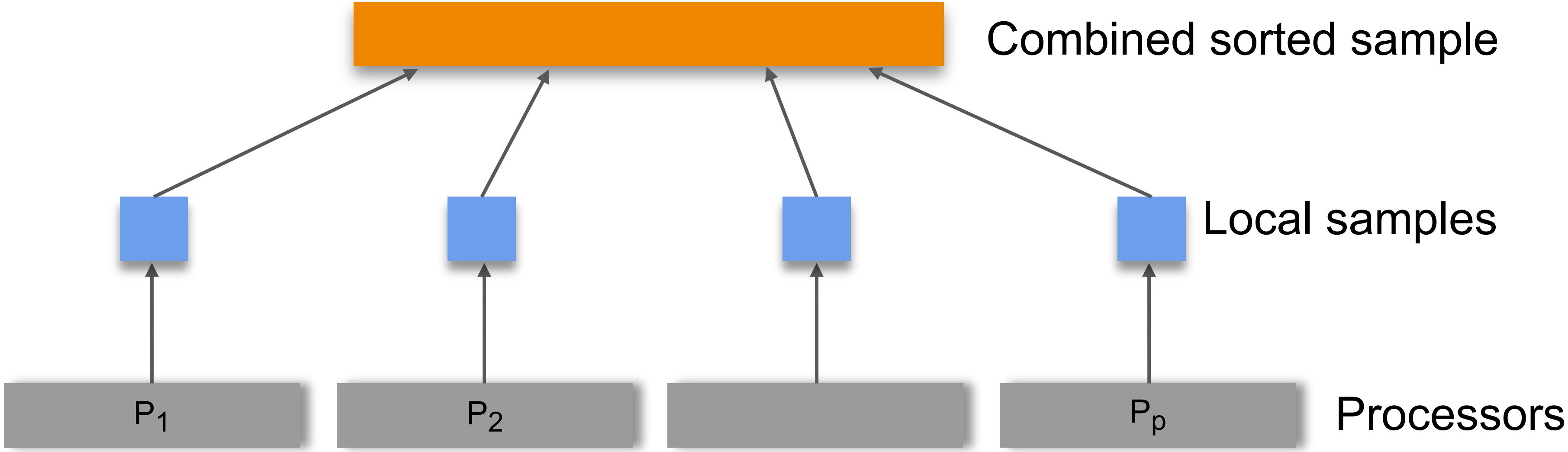
Sample sort

Sample sort

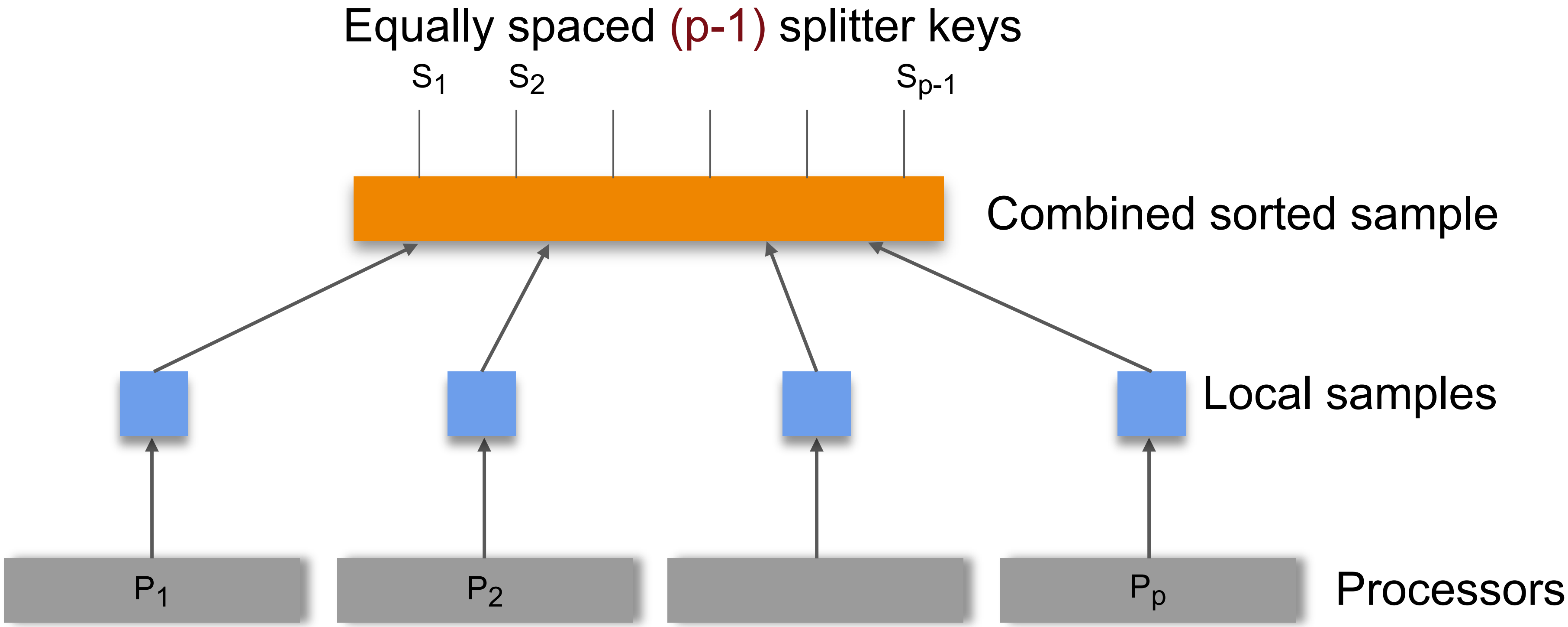
Sample sort



Sample sort



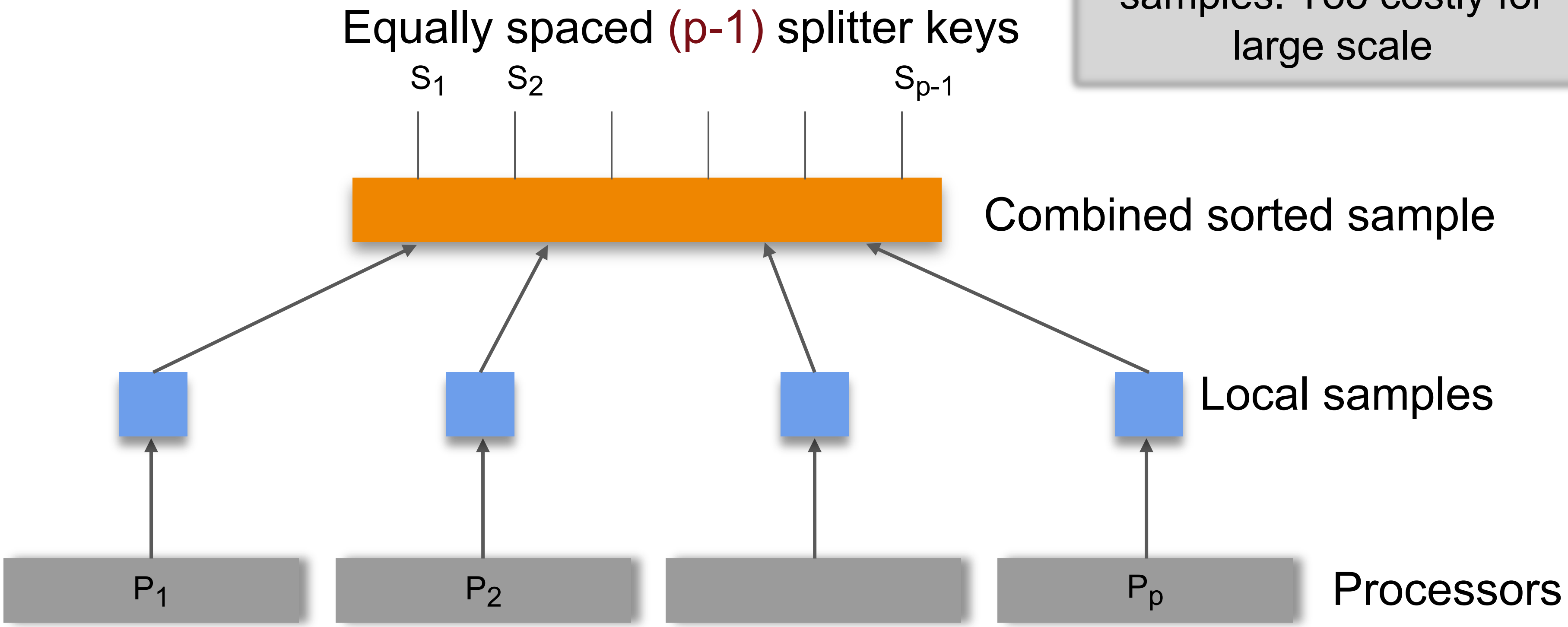
Sample sort



Equally spaced (p-1) splitter keys
S_1
S_2
S_{p-1}
Combined sorted sample
Local samples
P_1
P_2
P_p
Processors

Sample sort

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



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

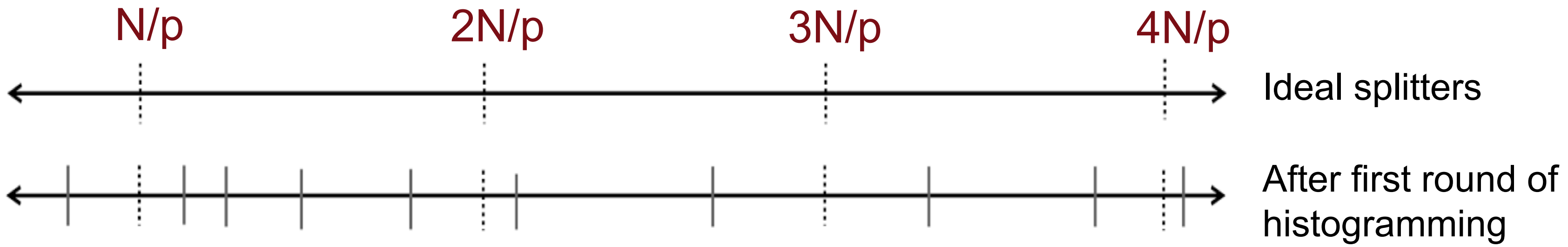
- Maintain "splitter" intervals around $\frac{N}{p}, \frac{2N}{p} \ldots \frac{iN}{p} \ldots$
  - Use closest key (left and right) in sample so far
  - For splitter i: $[L_i, R_i]$

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

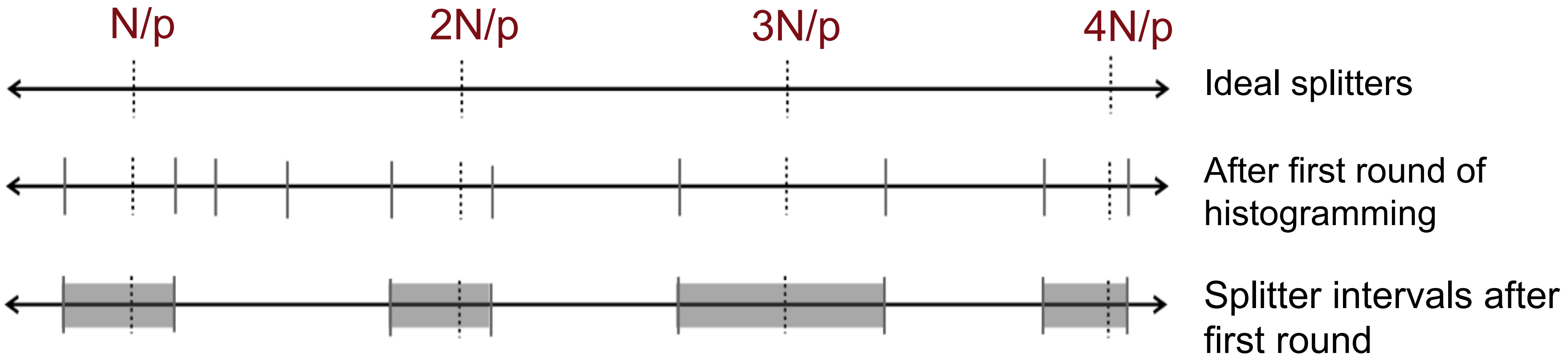
Sample only within splitter intervals, proportional to interval size

HSS: Splitter intervals over time

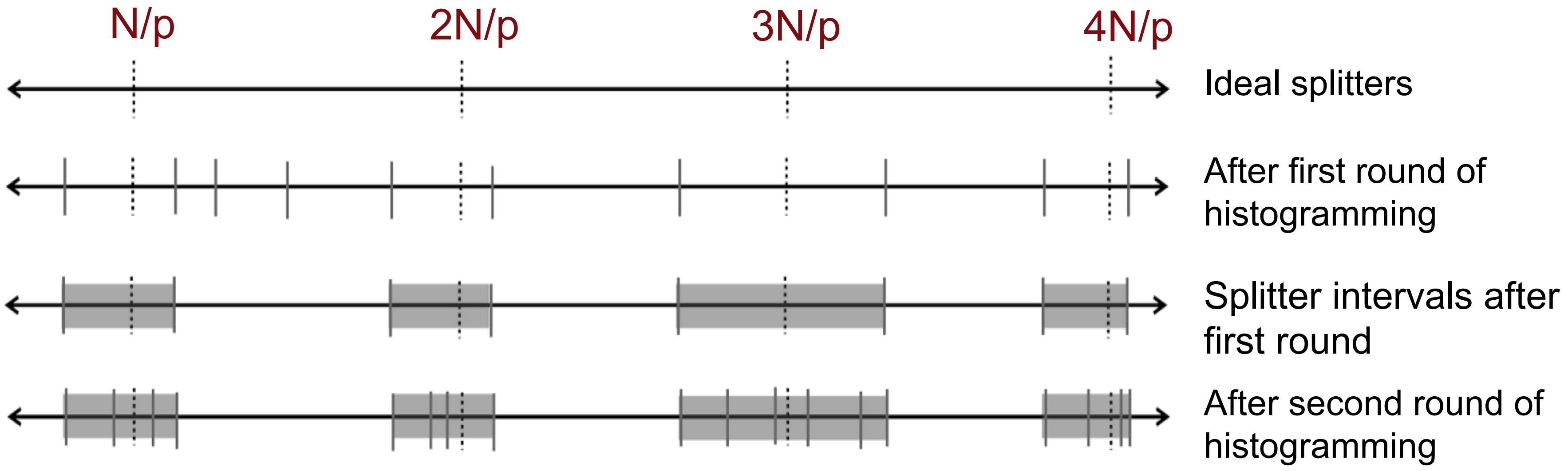
HSS: Splitter intervals over time



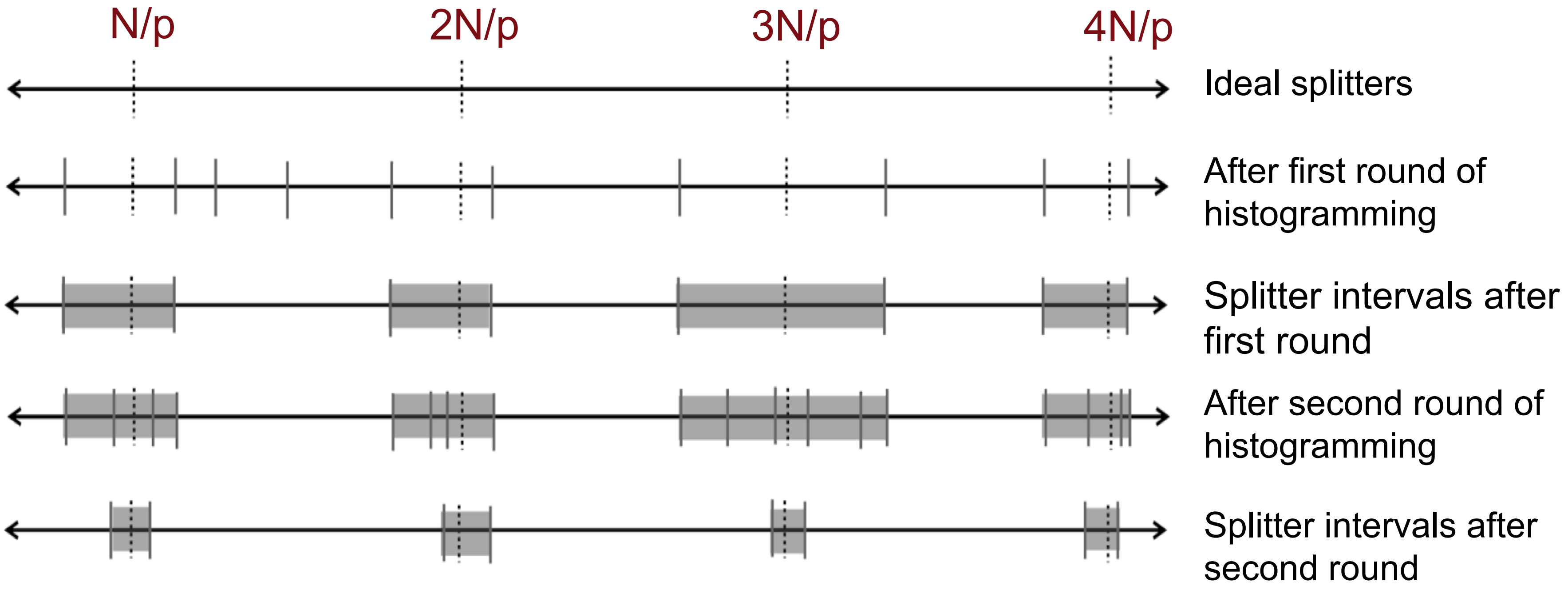
HSS: Splitter intervals over time



HSS: Splitter intervals over time



HSS: Splitter intervals over time



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, ϵ: Desired $(1 \pm \epsilon)$ load balance

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

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: $O(S \log N)$

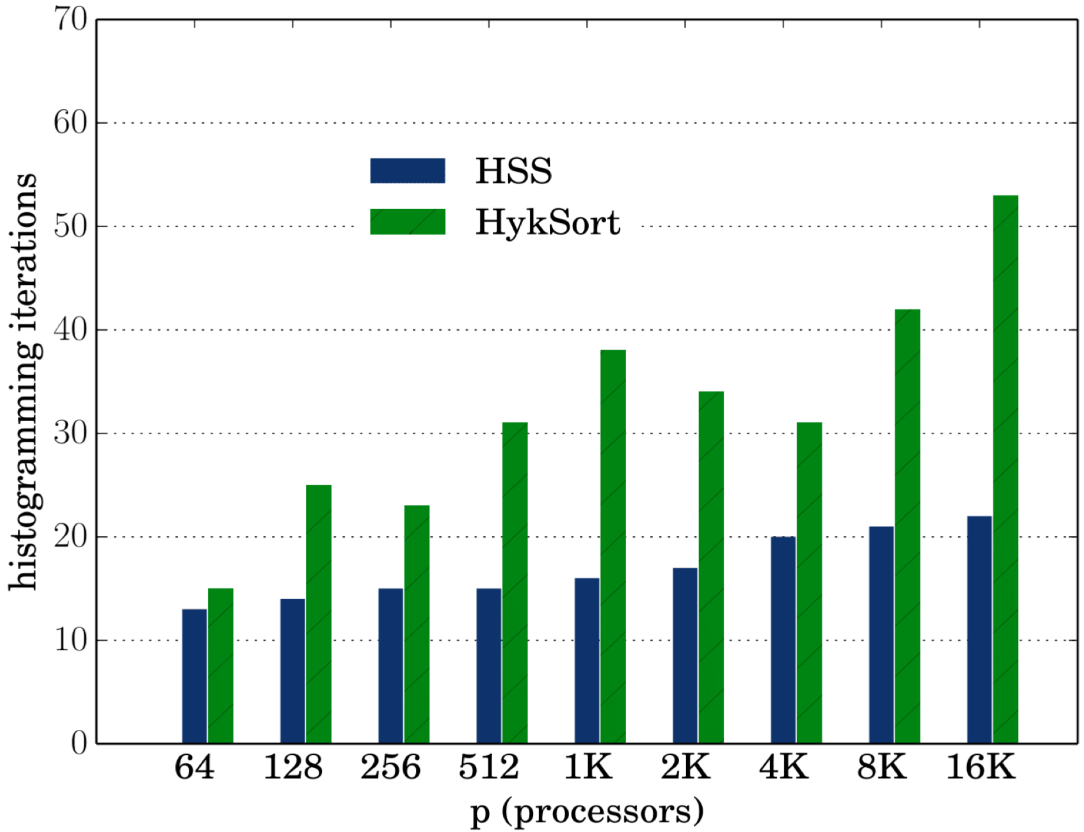
- Cost of computing rank of S keys: $O(S \log N)$
  - via global pipelined reductions [Thakur, PVM/MPI'03]

Results summary

Algorithm	Cost Complexity	BSP Supersteps
Sample Sort	$O(p \log p \ \log N / \epsilon^2)$	$O(1)$
AMS Sort	$O(p (\log p + \epsilon^{-1}) \log N)$	$O(1)$
HSS with k rounds	$O(pk \sqrt[k]{\log p / \epsilon} \log N)$	$O(k)$
HSS optimal cost	$O(p \log(\epsilon^{-1} \log p) \log N)$	$O(\log(\epsilon^{-1} \log p))$

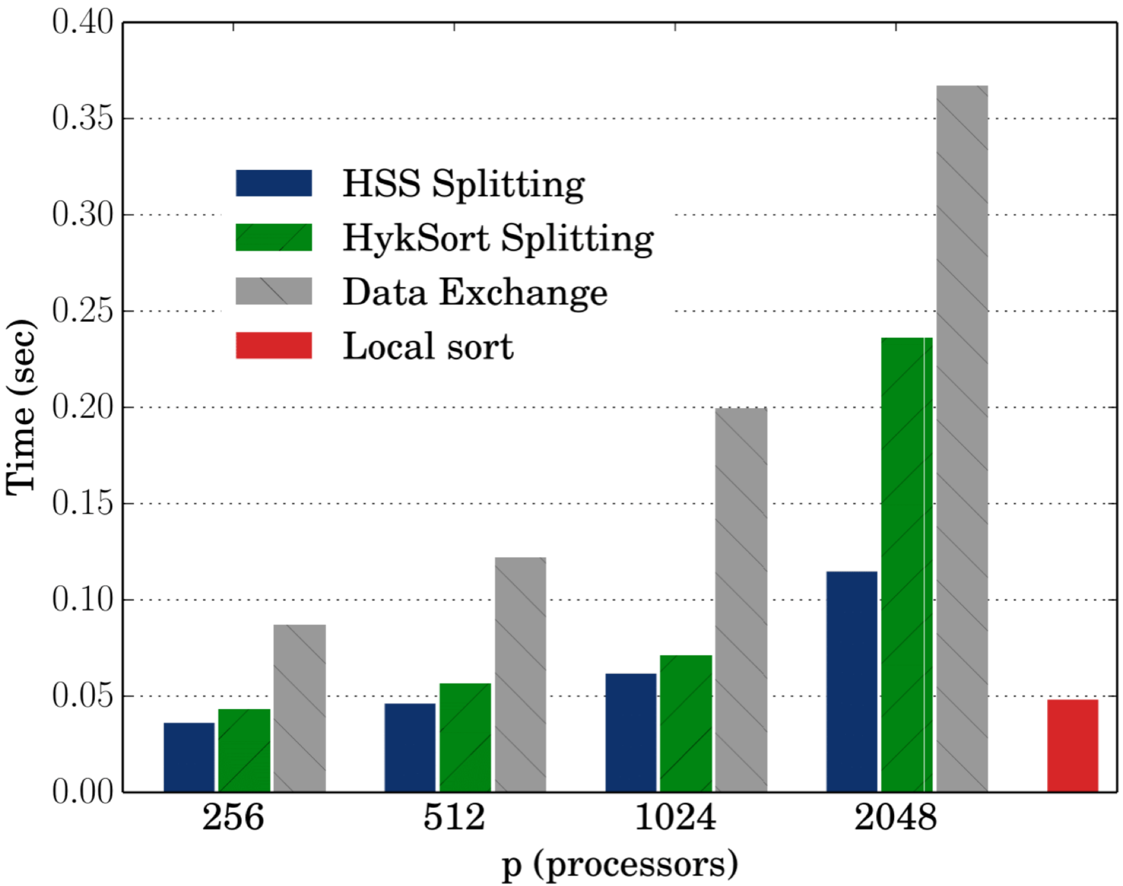
Experimental results

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

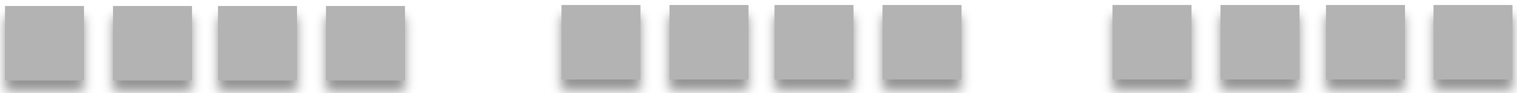
Multi-stage sorting

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

Multi-stage sorting

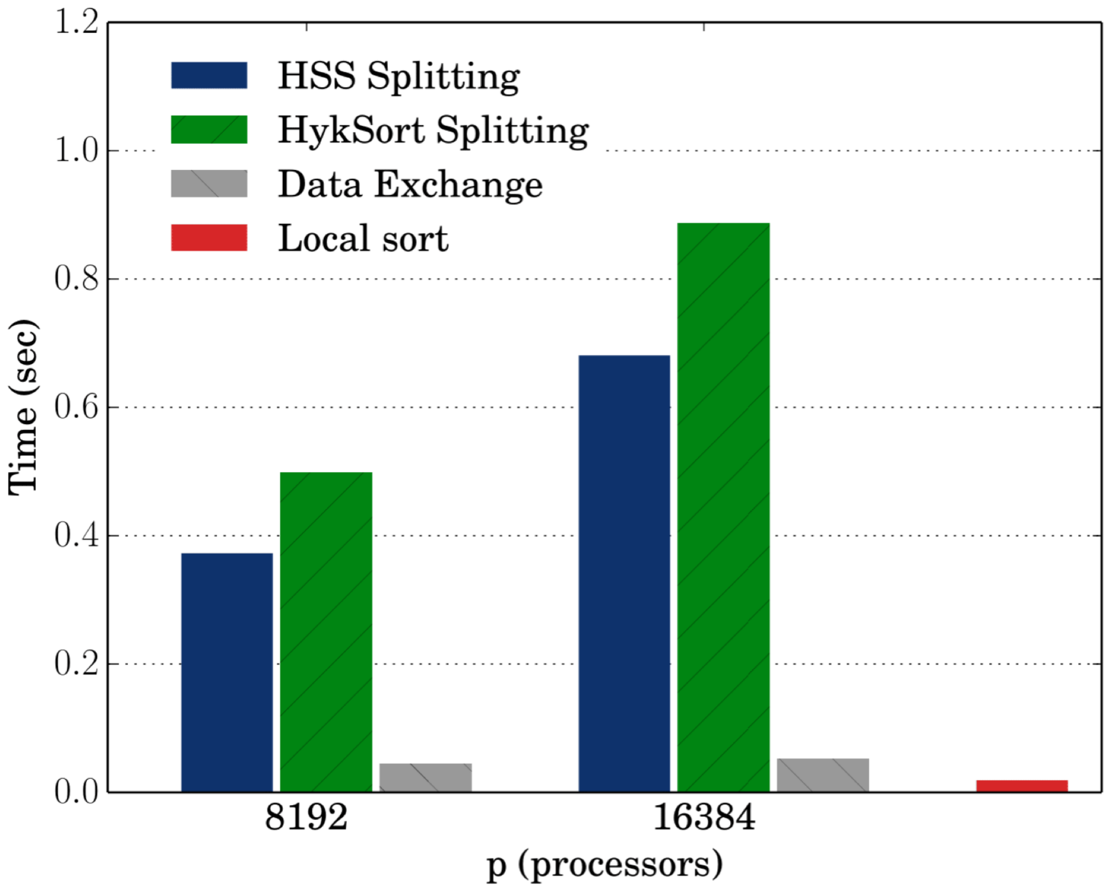
For very large scale, data exchange can be inefficient, $O(p^2)$ messages

- Split data exchange into k (typically <=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



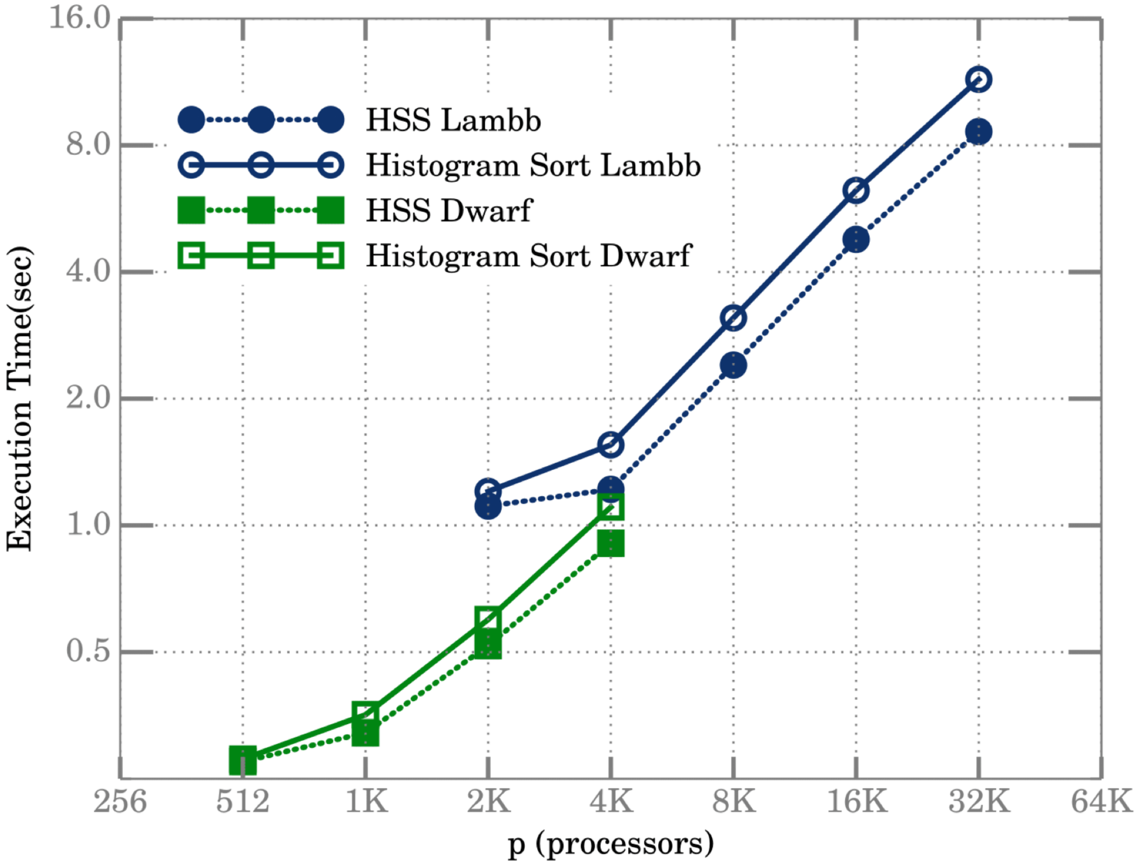
Processor Group 1 Processor Group 2 Processor Group 3

Multi-stage sorting



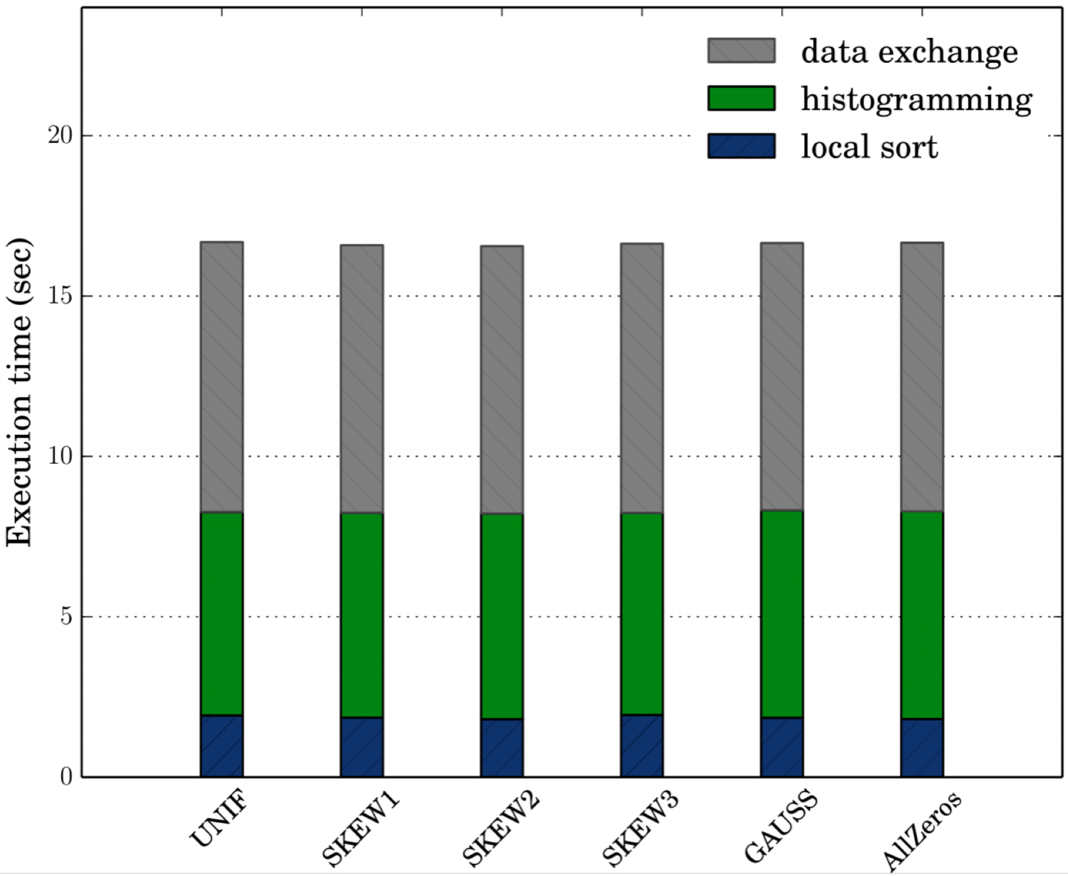
Modest improvement in overall running time

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

Algorithm	Cost Complexity	BSP Supersteps
Sample Sort	$O(p \log p \ \log N / \epsilon^2)$	$O(1)$
AMS Sort	$O(p (\log p + \epsilon^{-1}) \log N)$	$O(1)$
HSS with k rounds	$O(pk \sqrt[k]{\log p / \epsilon} \log N)$	$O(k)$
HSS optimal cost	$O(p \log(\epsilon^{-1} \log p) \log N)$	$O(\log(\epsilon^{-1} \log p))$

Thank You!

Backup slides

HSS: Main result

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

- Substituting $k = \mathrm{O}(\log((\log p)/\epsilon))$ 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

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