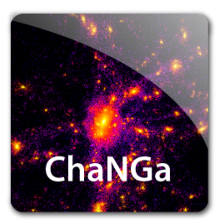
Histogram Sort with Sampling (HSS)

ILLINOIS
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Vipul Harsh, Laxmikant Kale {vharsh2, kale}@illinois.edu

PPL UIUC PARALLEL PROGRAMMING LABORATORY

Massively parallel applications use parallel sorting



- Charm N-body GrAvity solver
- Cosmological N-body simulations
- Parallel sorting in every iteration

- Cosmology code based on Chombo
- Global sorting every step for load balancing and locality

CHARM

Parallel sorting: Goals

- Load balance after sorting
- Optimal data movement
- Generality: should work for any inputs
- Scalability and performance for massive clusters

Parallel sorting : Skeleton for many algorithms

- p processors, N/p keys in each processor
- Determine (p-1) splitter keys to partition the keys into p buckets, one for each processor. Make these splitters available to all processors.
- Send all keys to destination processor depending on the which bucket if falls in.
- E.g. sample sort, histogram sort

Problems with existing algorithms

- **Sample sort**: Pick a sample from the input, choose (p-1) splitters from the sample. Variants: Regular sampling, random sampling.
  Problem: Required sample size is too large for good splitting
- **Histogram sort:** Maintain a set of candidate splitters, obtain their rank in the local input by performing histogramming rounds. Refine candidate keys every round till a given threshold.
  Problem: Required number of histogramming rounds could be large, depending upon the input distribution

Our approach: Histogram sort with sampling (HSS)

- **Key Idea:** Sample keys before every histogramming round. Use information from previous rounds to sample intelligently.
- By performing histogramming on the sample, HSS reduces the sample size requirements of sample sort by multiple orders of magnitude.

Overview: HSS maintains "splitter intervals" for all splitter keys. A splitter interval marks the nearest key seen so far, both smaller (left) and greater (right) than the ideal splitter. ε is the maximum load imbalance permitted by the application.

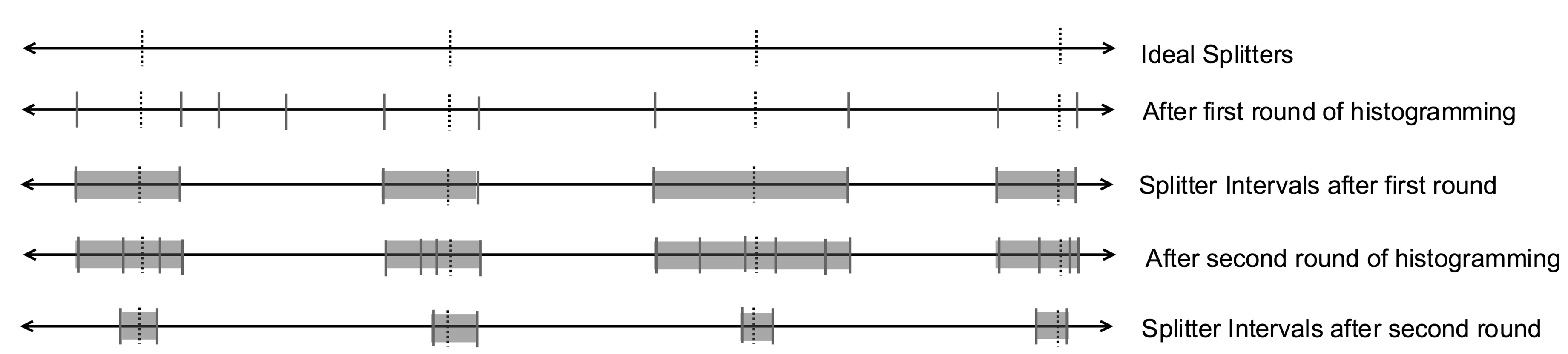


Fig 1: Figure illustrating HSS with multiple rounds. After first round, samples are picked only from the splitter intervals. Notice how the splitter intervals shrink as the algorithm progresses.

- In general, for HSS with k rounds, sample* required per round is $O(p\sqrt[k]{\log p/\varepsilon})$
- For HSS with O(p) sample every round, #rounds required is O(log ((log p)/ ε))
- Sample sort requires $O(p \log N/\varepsilon^2)$ samples

** with high probability*

HSS is theoretically more efficient than sample sort

Algorithm	Overall sample size	Overall sample size for $p=10^5, \epsilon=5\%$	Computation complexity	Communication complexity
Sample sort with regular sampling	$\mathcal{O}(\frac{p^2}{\epsilon})$	1600 GB	$\mathcal{O}\left(\frac{N}{p}\log\frac{N}{p}+\frac{p^2}{\epsilon}\log p+\frac{N}{p}\log p\right)$	$\mathcal{O}\left(\frac{p^2}{\epsilon}+p+\frac{N}{p}\right)$
Sample sort with random sampling	$\mathcal{O}(\frac{p\log N}{\epsilon^2})$	8.1 GB	$\mathcal{O}\left(\frac{N}{p}\log\frac{N}{p}+\frac{p\log N\log p}{\epsilon^2}+\frac{N}{p}\log p\right)$	$\mathcal{O}\left(\frac{p\log N}{\epsilon^2}+p+\frac{N}{p}\right)$
HSS with one round	$\mathcal{O}(\frac{p\log p}{\epsilon})$	184 MB	$\mathcal{O}\left(\frac{N}{p}\log\frac{N}{p}+\frac{p\log p}{\epsilon}\log N+\frac{N}{p}\log p\right)$	$\mathcal{O}\left(\frac{p\log p}{\epsilon}+p+\frac{N}{p}\right)$
HSS with two rounds	$\mathcal{O}(p\sqrt{\frac{\log p}{\epsilon}})$	24 MB	$\mathcal{O}\left(\frac{N}{p}\log\frac{N}{p}+p\sqrt{\frac{\log p}{\epsilon}}\log N+\frac{N}{p}\log p\right)$	$\mathcal{O}\left(p\sqrt{\frac{\log p}{\epsilon}}+p+\frac{N}{p}\right)$
HSS with k rounds	$\mathcal{O}(kp\sqrt[k]{\frac{\log p}{\epsilon}})$	-	$\mathcal{O}\left(\frac{N}{p}\log\frac{N}{p}+kp\sqrt[k]{\frac{\log p}{\epsilon}}\log N+\frac{N}{p}\log p\right)$	$\mathcal{O}\left(kp\sqrt[k]{\frac{\log p}{\epsilon}}+p+\frac{N}{p}\right)$
HSS with $\mathcal{O}(\log\frac{\log p}{\epsilon})$ rounds	$\mathcal{O}(p\log\frac{\log p}{\epsilon})$	10 MB	$\mathcal{O}\left(\frac{N}{p}\log\frac{N}{p}+p\log\frac{\log p}{\epsilon}\log N+\frac{N}{p}\log p\right)$	$\mathcal{O}\left(p\log\frac{\log p}{\epsilon}+p+\frac{N}{p}\right)$

Table 1: Running time complexity and comparison of required sample sizes of HSS and sample sort.

HSS in practice

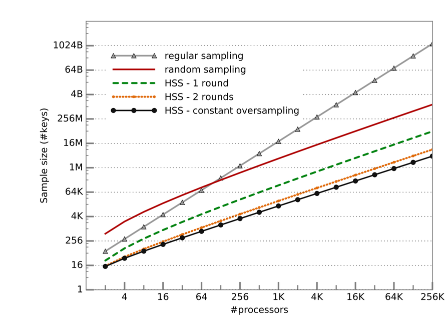


Fig 2: HSS vs Sample sort: sample size required for splitting, ε = 5%

Histogramming takes very little fraction of the running time, which is dominated by local sorting and data exchange, both of which are inherent costs!

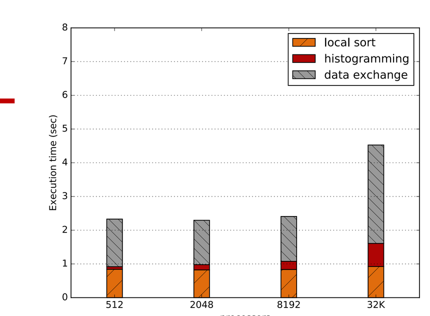


Fig 3: HSS on Mira(ALCF) BG/Q. Each processor had 1 million 8 byte keys with a 4 byte payload with each key

p (x 1000)	sample size/round (x p)	Number of rounds	Bound on number of rounds
4	5	4	8
8	5	4	8
16	5	4	8
32	5	4	8

Table 2: Number of rounds required for determining all splitters (experimentally)

Conclusion

- HSS combines histogramming and sampling to achieve fast splitter determination
- HSS is extremely practical for massively parallel applications

- **Future Work**
  → Integration in real applications; ChaNGa
- **Details**:vharsh2.web.engr.illinois.edu/projects/parsort/prelimreport.pdf
- **Acknowledgements:** Edgar Solomnik, Omkar Thakoor, Umang Mathur