Highly Scalable Parallel Sorting

Edgar Solomonik
University of Illinois at Urbana-Champaign
April 29, 2010

8th Annual Workshop on Charm++ and its Applications

PARALLEL PROGRAMMING LAB
DEPT. OF COMPUTER SCIENCE, UNIVERSITY OF ILLINOIS
PPL UIUC

Outline

- Parallel sorting background
- Histogram Sort overview
- Histogram Sort optimizations
- Charm++ implementation
- Results
- Limitations of work
- Contributions
- Future work

Parallel Sorting

- Input
  - There are ***n*** unsorted keys, distributed evenly over ***p*** processors
  - The distribution of keys in the range is unknown and possibly skewed
- Goal
  - Sort the data globally according to keys
  - Ensure no processor has more than ***(n/p)+threshold*** keys

Scaling Challenges

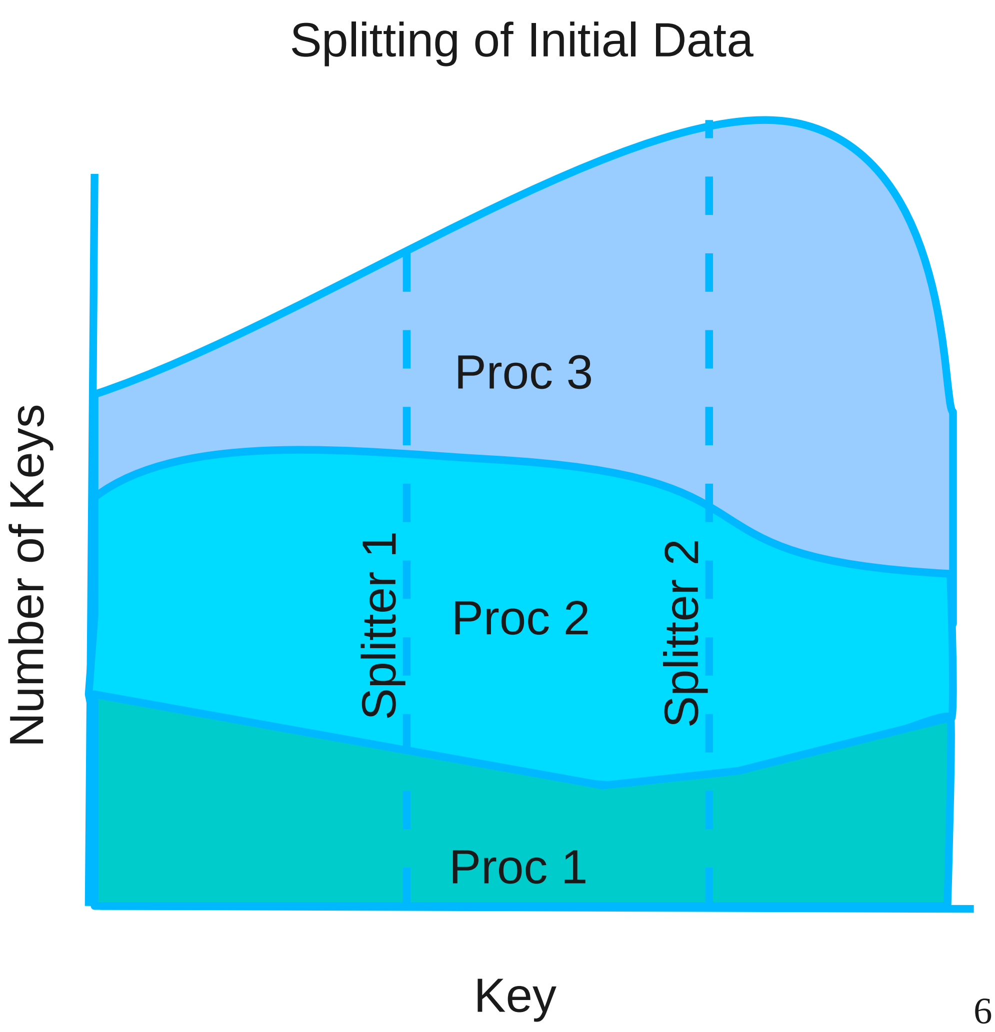
- Load balance
  - Main objective of most parallel sorting algorithms
  - Each processor needs a continuous chunk of data
- Data exchange communication
  - Can require complete communication graph
  - All-to-all contains $\boldsymbol{n}$ elements in $\boldsymbol{p}^2$ messages

Parallel Sorting Algorithms

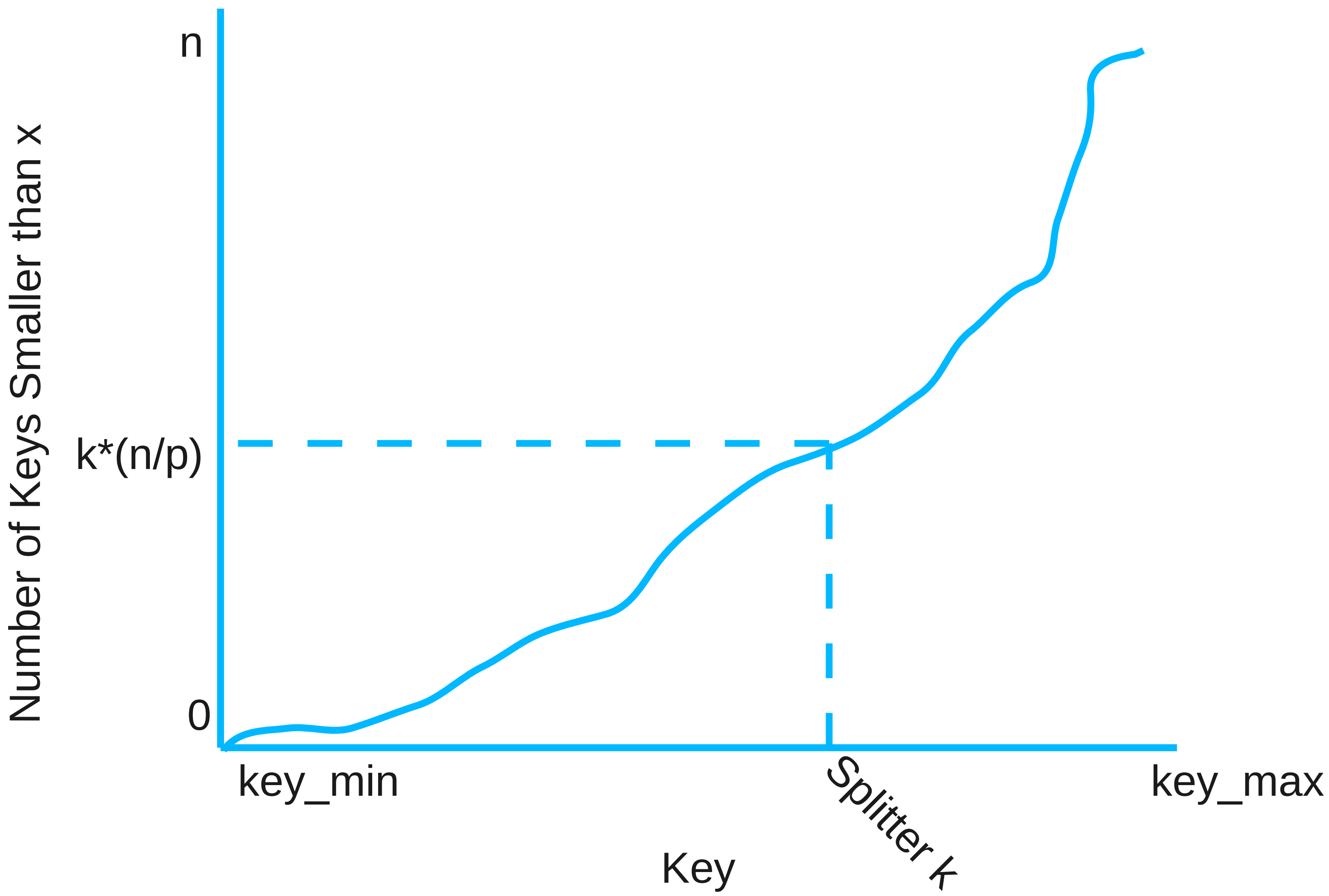
Type	Data movement
• Merge-based	
– Bitonic Sort	***$\frac{1}{2}*n*log^2(p)$***
– Cole's Merge Sort	***O(n*log(p))***
• **Splitter-based**	
– Sample Sort	***n***
– **Histogram Sort**	***n***
• Other	
– Parallel Quicksort	***O(n*log(p))***
– Radix Sort	***O(n)~4*n***

Splitter-Based Parallel Sorting

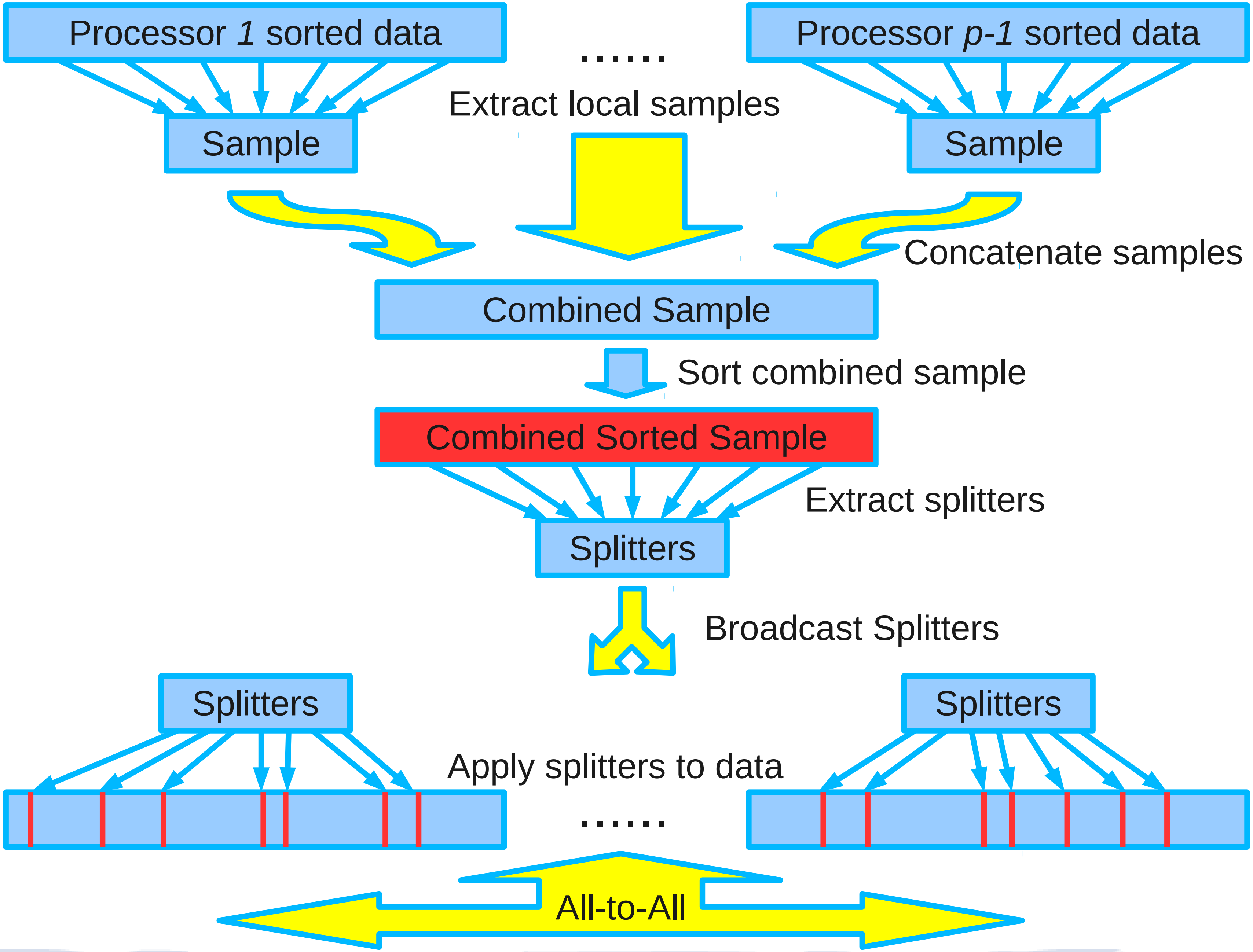
- A ***splitter*** is a key that partitions the global data at a desired location
- ***p-1*** global splitters needed to subdivide the data into ***p*** continuous chunks
- Each processor can send out its local data based on the splitters
  - ***Data moves only once***
- Each processor merges the data chunks as it receives them



Splitter on Key Density Function



Sample Sort



Processor *1* sorted data
......
Processor *p-1* sorted data
Extract local samples
Sample
Sample
Concatenate samples
Combined Sample
Sort combined sample
Combined Sorted Sample
Extract splitters
Splitters
Broadcast Splitters
Splitters
Splitters
Apply splitters to data
......
All-to-All

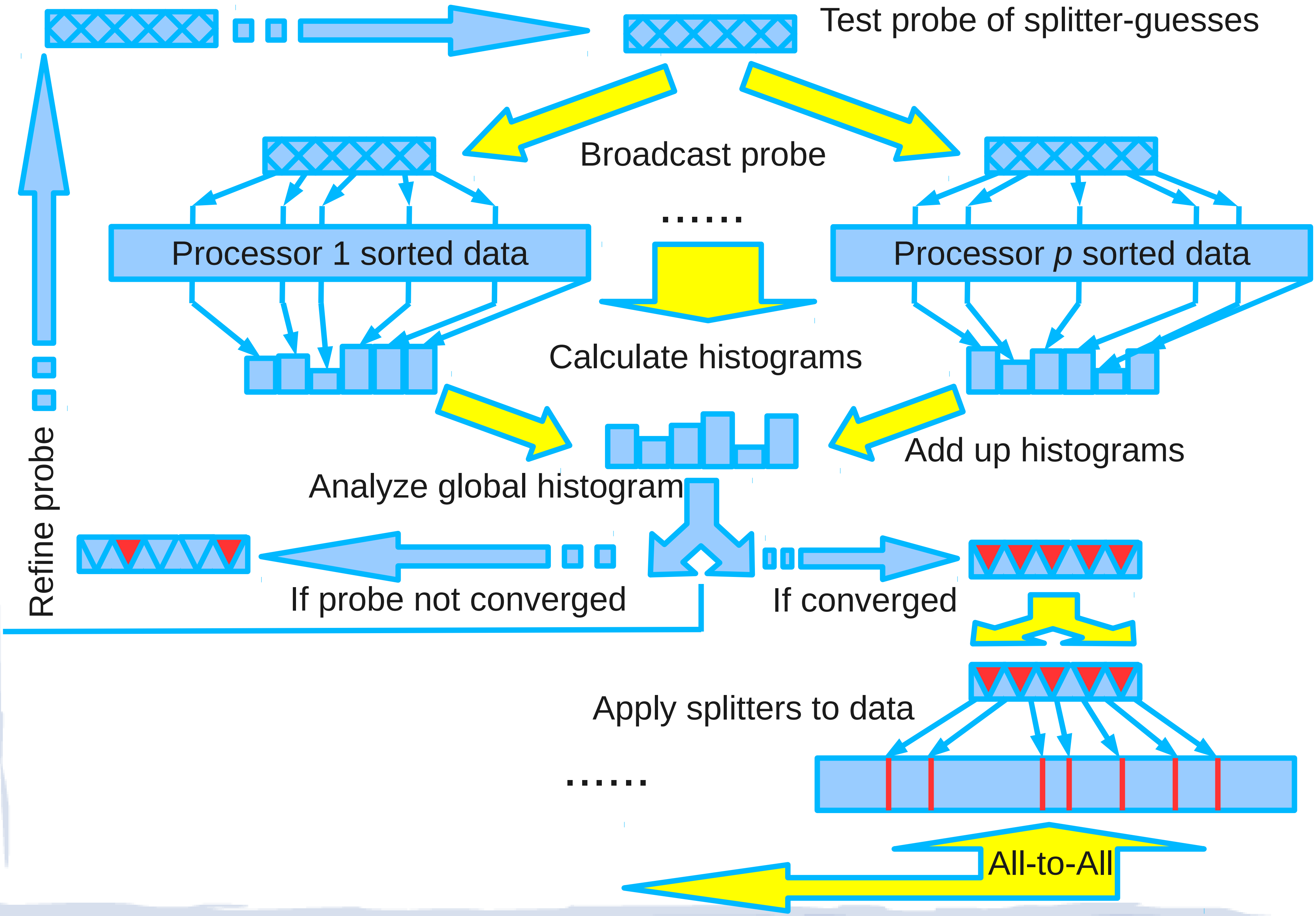
Sample Sort

- The sample is typically regularly spaced in the local sorted data ***s=p-1***
  - Worst case final load imbalance is ***2*(n/p)*** keys
  - In practice, load imbalance is typically very small
- Combined sample becomes bottleneck since $(s*p) \sim p^2$
  - With *64*-bit keys, if ***p = 8192***, sample is **16 GB**!

Basic Histogram Sort

- Splitter-based
- Uses iterative guessing to find splitters
  - ***O(p)*** probe rather than ***O(p²)*** combined sample
  - Probe refinement based on global histogram
    - Histogram calculated by applying splitters to data
- Kale and Krishnan, ICPP 1993
- Basis for this work

Basic Histogram Sort



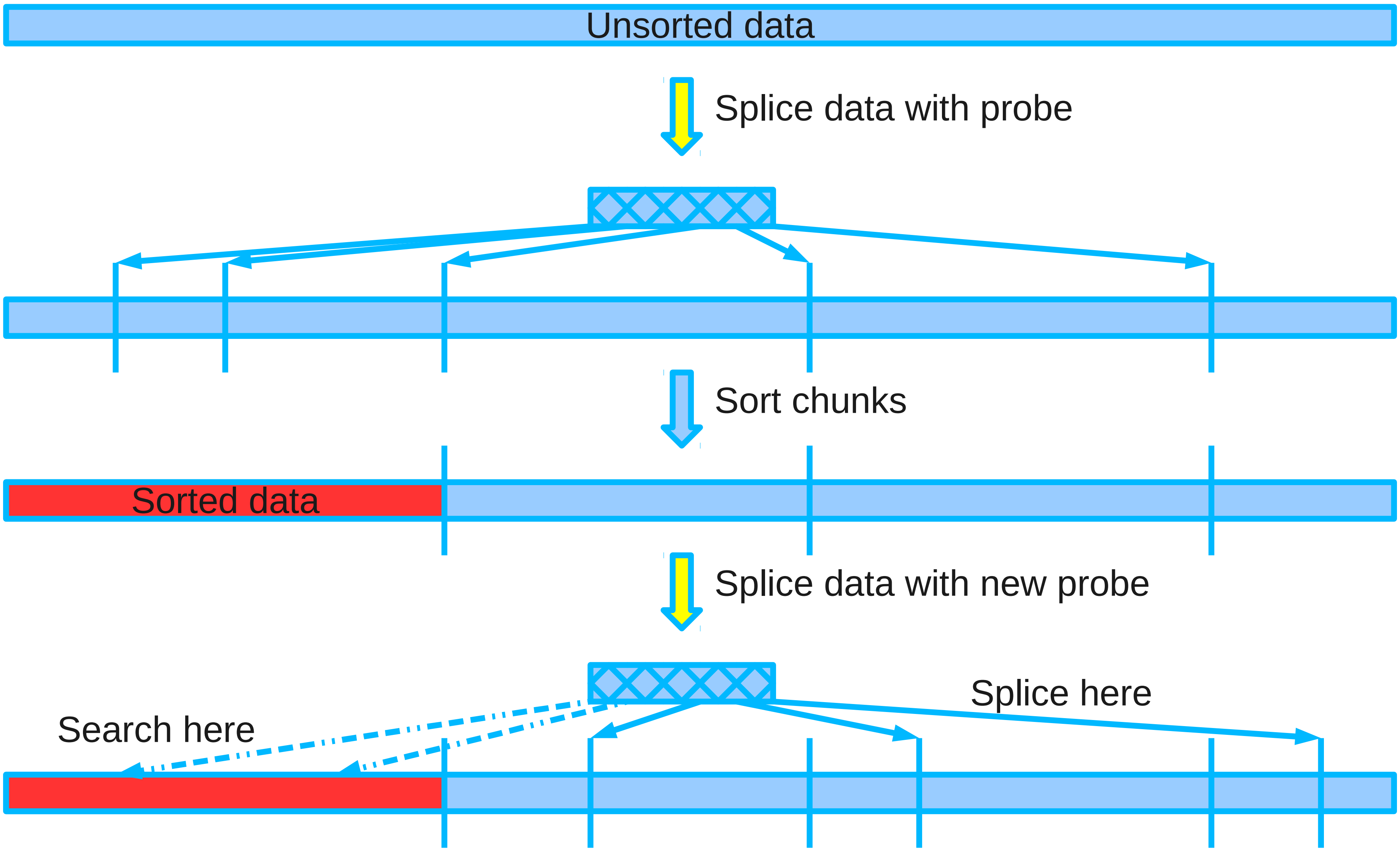
Basic Histogram Sort

- Positives
  - Splitter-based: single all-to-all data transpose
  - Can achieve arbitrarily small ***threshold***
  - Probing technique is scalable compared to sample sort, $O(p)$ vs $O(p^2)$
  - Allows good overlap between communication and computation (to be shown)
- Negatives
  - Harder to implement
  - Running time dependent on data distribution

Sorting and Histogramming Overlap

- Don't actually need to sort local data first
- ***Splice data*** instead
  - Use splitter-guesses as Quicksort pivots
  - Each splice determines location of a guess and partitions data
- Sort chunks of data while histogramming happens

Histogramming by Splicing Data



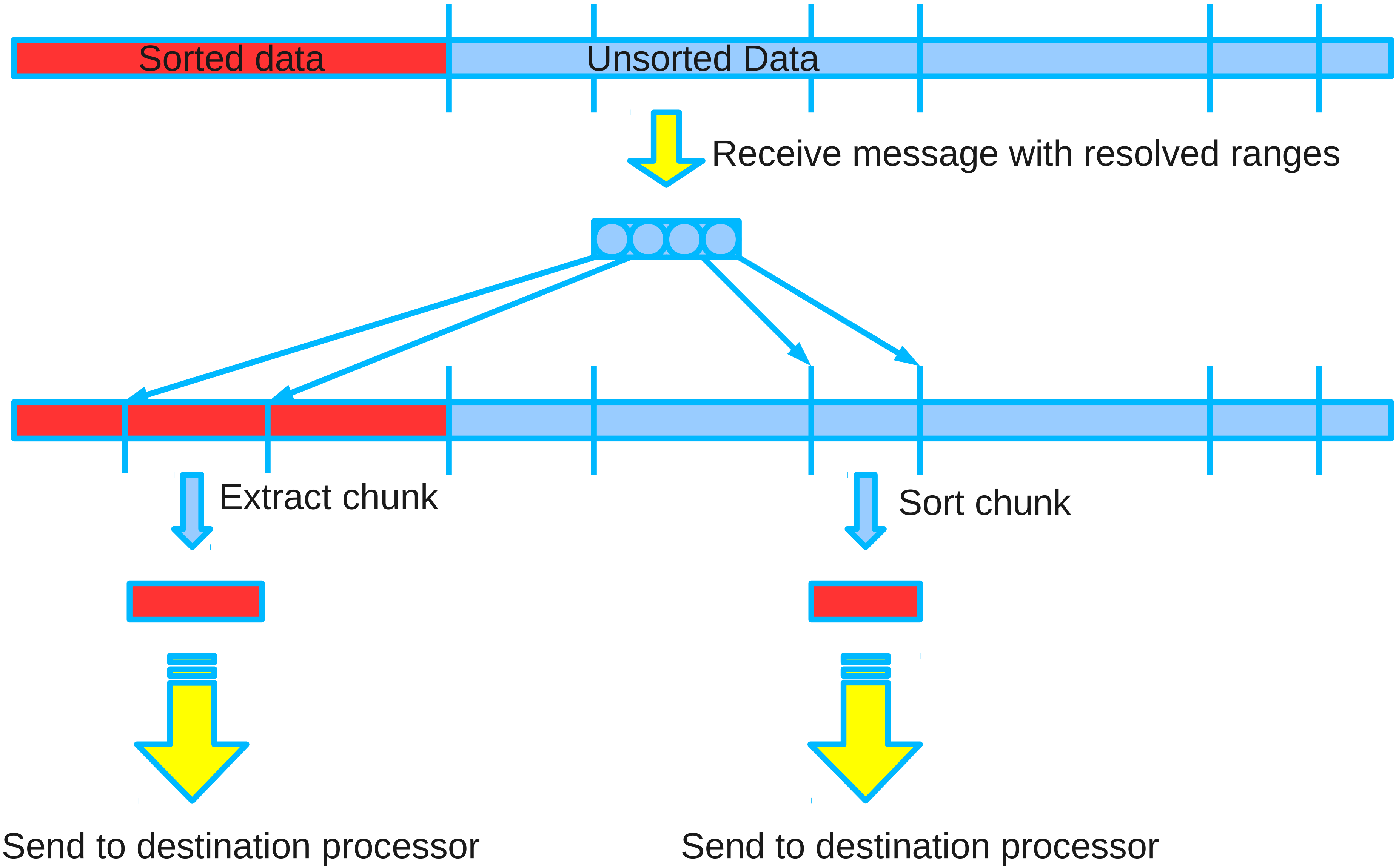
Histogram Overlap Analysis

- Probe generation work should be offloaded to one processor
  - Reduces critical path
- Splicing is somewhat expensive
  - ***O((n/p)*log(p))*** for first iteration
    - ***log(p)*** approaches ***log(n/p)*** in weak scaling
  - Small theoretical overhead (limited pivot selection)
  - Slight implementation overhead (libraries faster)
  - Some optimizations/code necessary

Sorting and All-to-All Overlap

- Histogram and local sort overlap is good but the all-to-all is the worst scaling bottleneck
- Fortunately, much all-to-all overlap available
- All-to-all can initially overlap with local sorting
  - Some splitters converge every histogram iteration
    - This is also prior to completion of local sorting
    - Can begin sending to any defined ranges

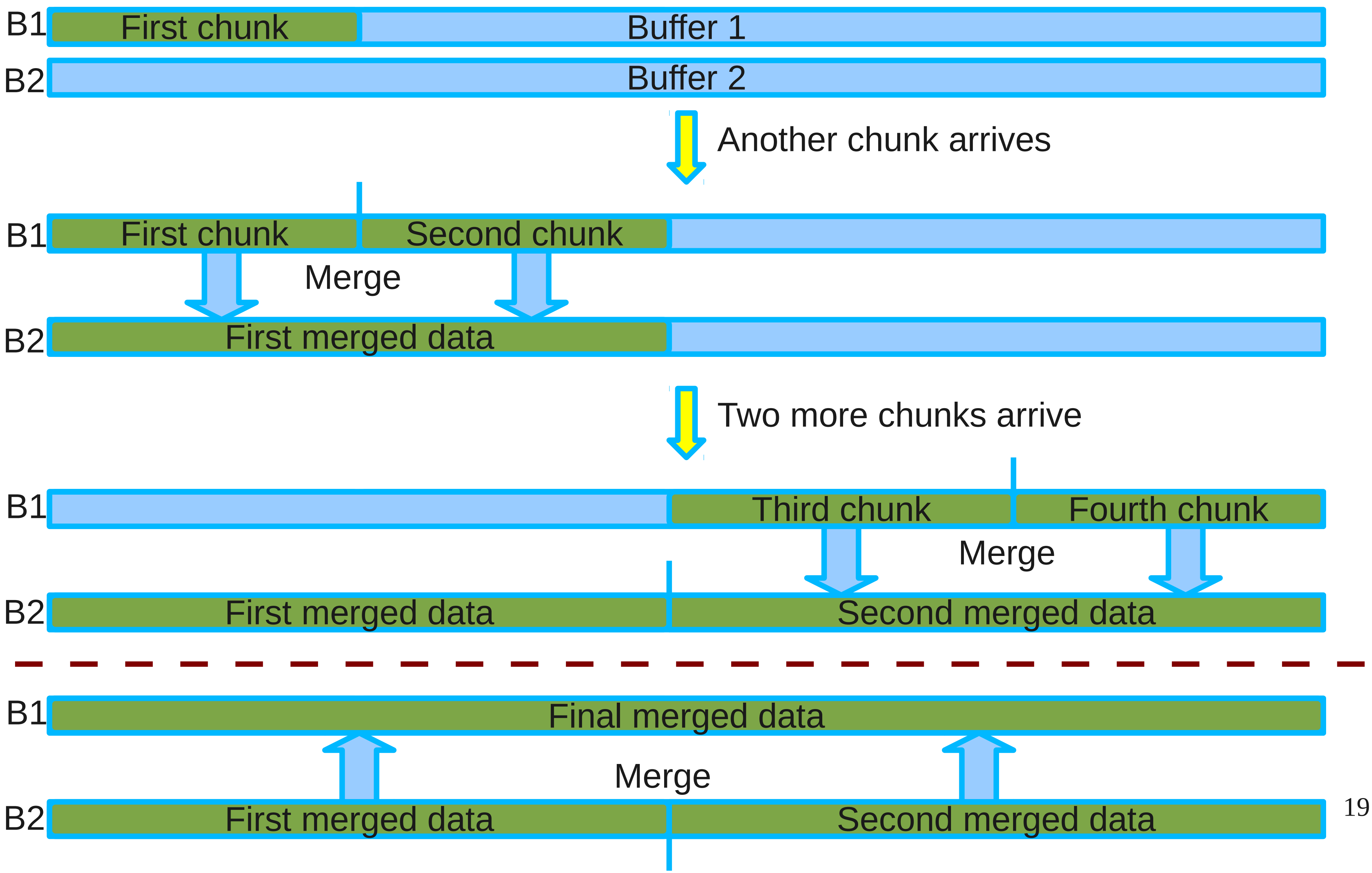
Eager Data Movement



All-to-All and Merge Overlap

- The *k*-way merge done when the data arrives should be implemented as a tree merge
  - A *k*-way heap merge requires all *k* arrays
  - A tree merge can start with just two arrays
- Some data arrives much earlier than the rest
  - Tree merge allows overlap

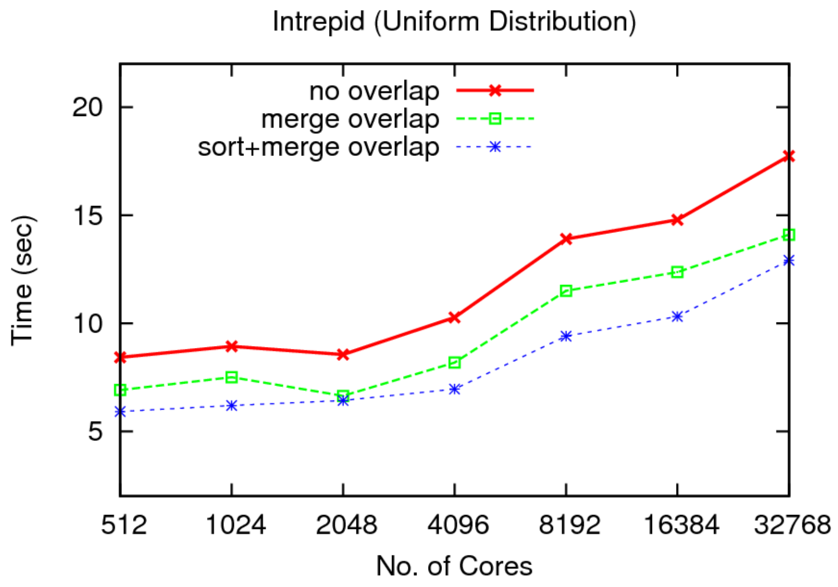
Tree k-way Merging



Charm++ Implementation

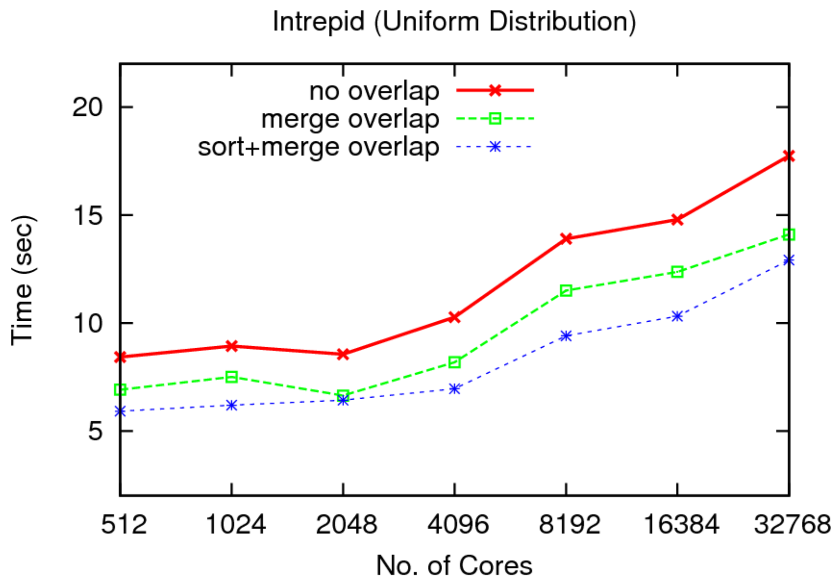
- Why?
  - Sort is compatible with Charm++ applications
  - Division between histogramming analysis work and data containers
    - More natural
    - Flexible
  - Charm++ scheduler used to automatically overlap executing stages and push probes through
- MPI implementation possible, but more difficult

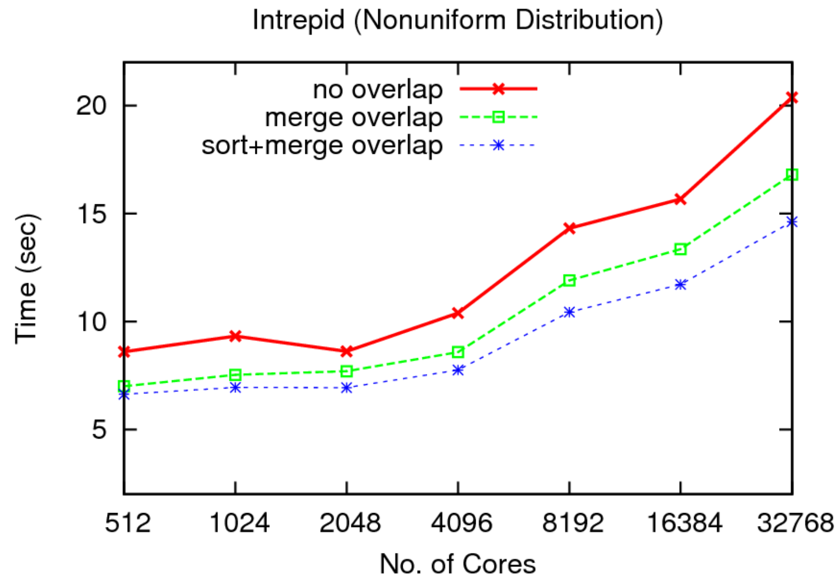
Overlap Benefit (Weak Scaling)



Tests done on Intrepid (BG/P) and Jaguar (XT4) with 8 million 64-bit keys per core.

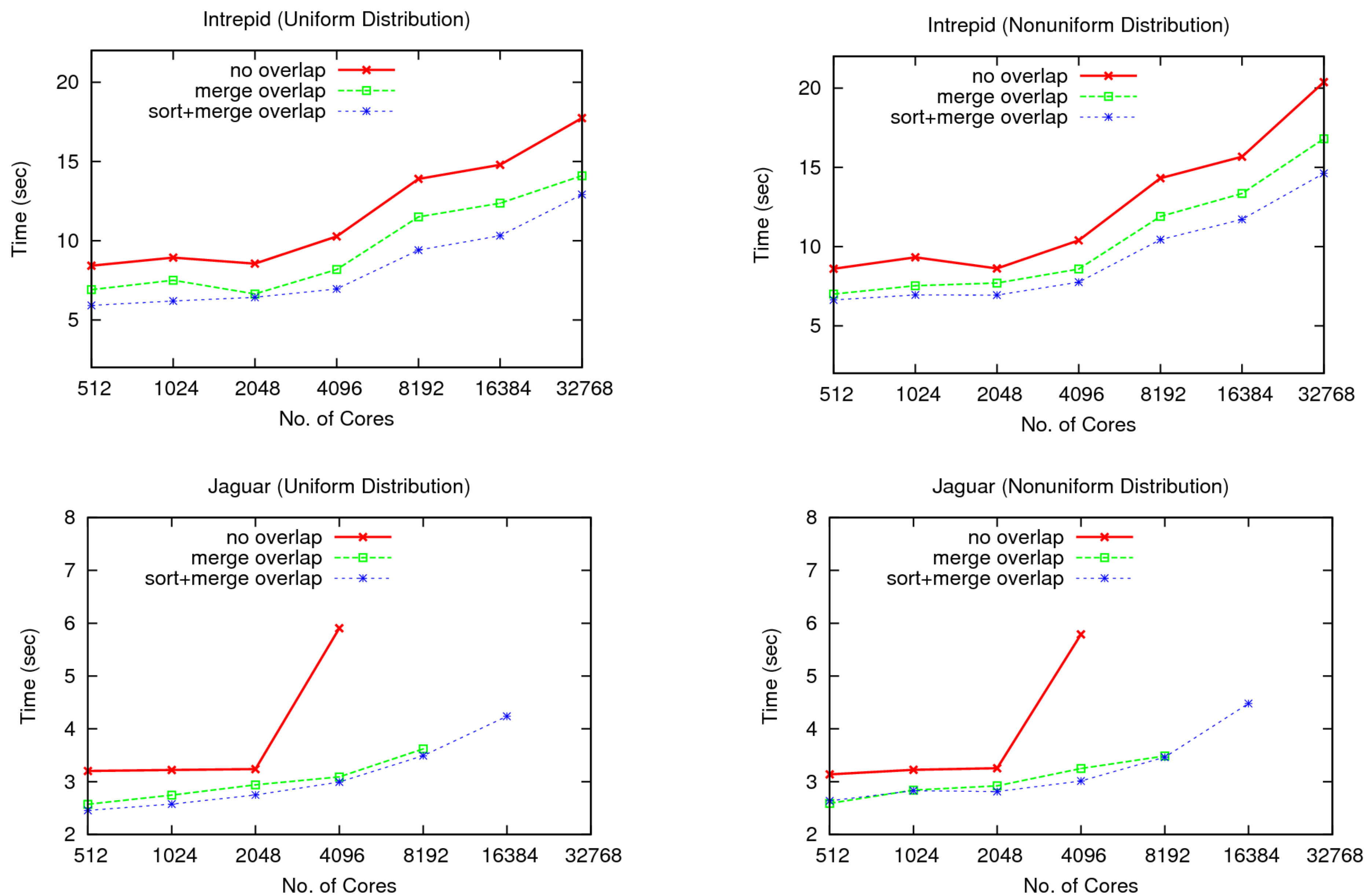
Overlap Benefit (Weak Scaling)





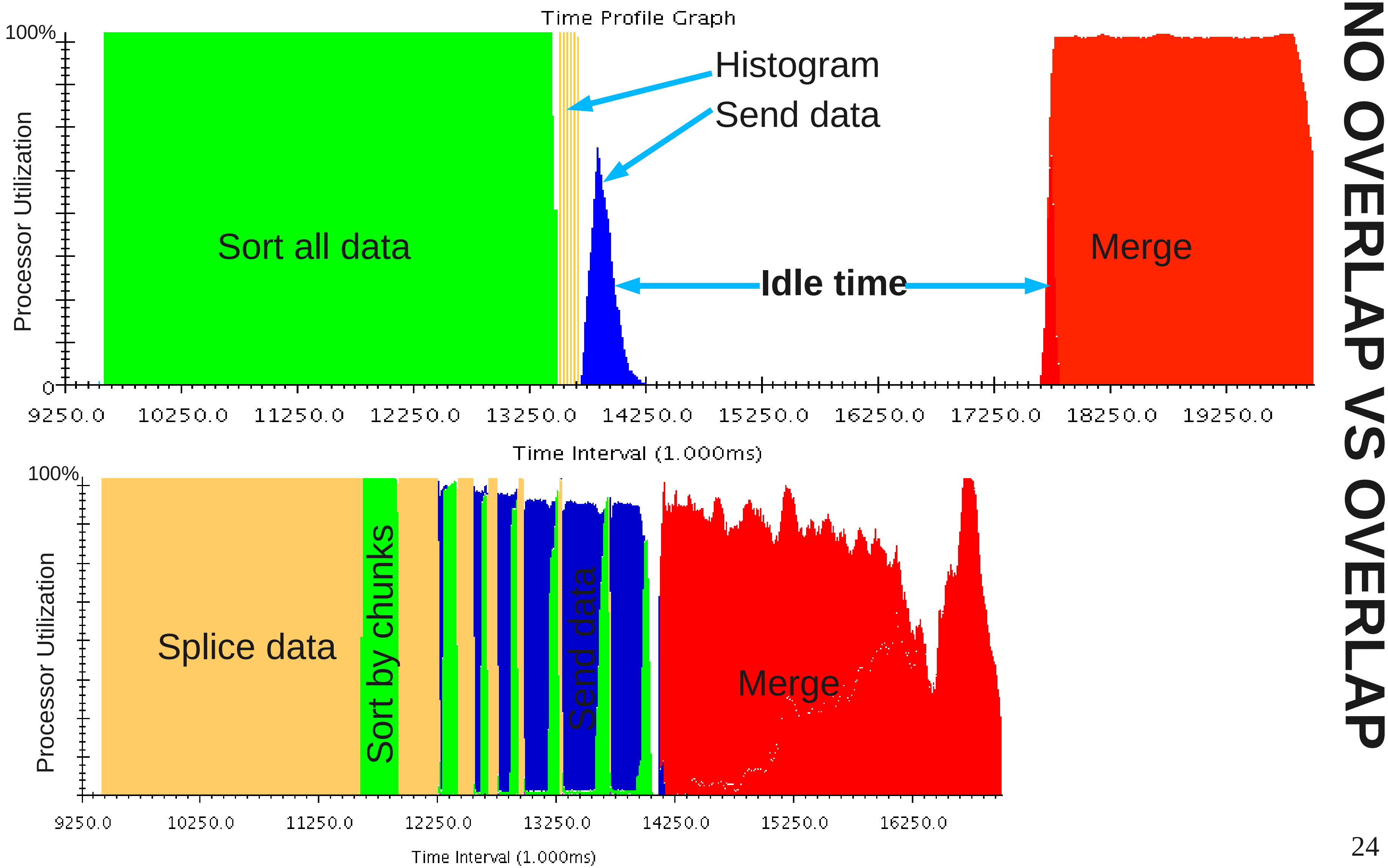
Tests done on Intrepid (BG/P) and Jaguar (XT4) with 8 million 64-bit keys per core.

Overlap Benefit (Weak Scaling)



Tests done on Intrepid (BG/P) and Jaguar (XT4) with 8 million 64-bit keys per core.

Effect of All-to-All Overlap

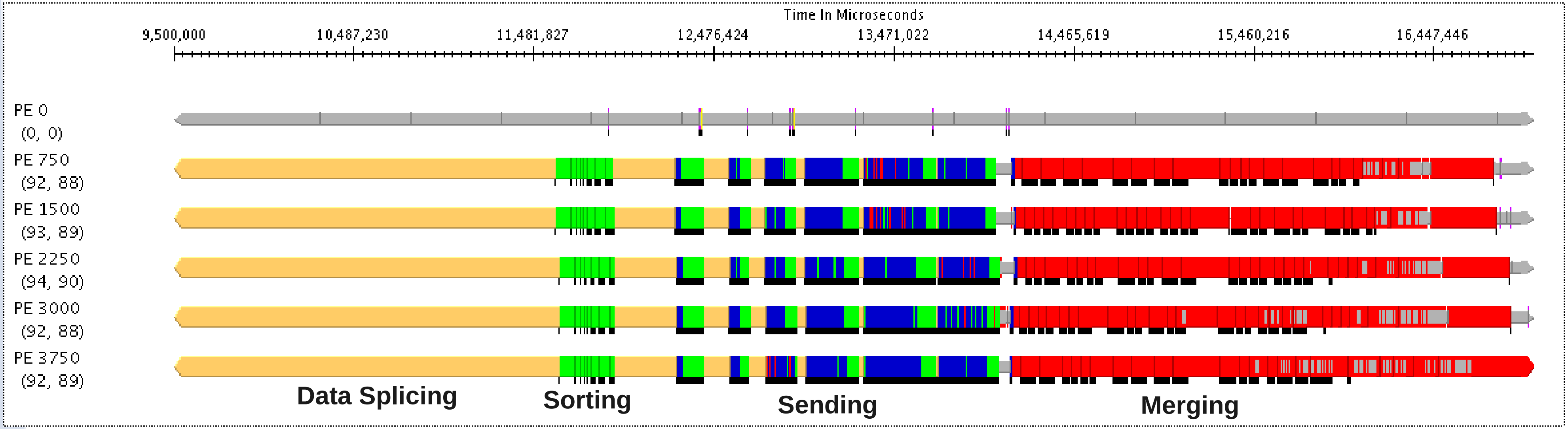


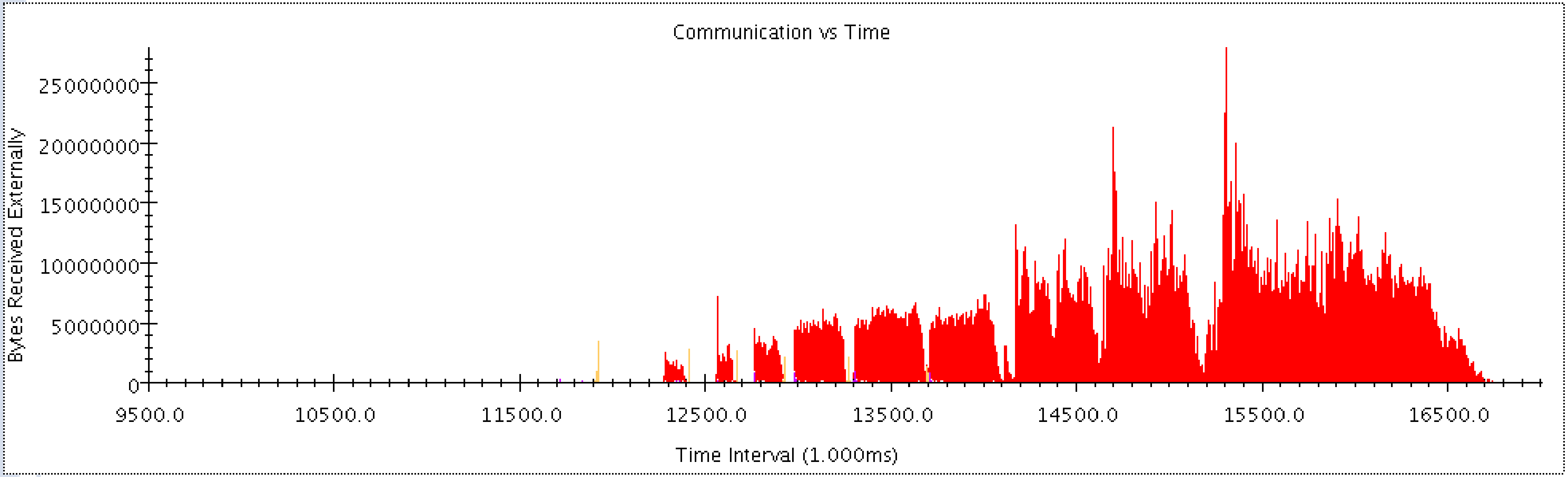
Tests done on 4096 cores of Intrepid (BG/P) with 8 million 64-bit keys per core.

All-to-All Spread and Staging

- Personalized all-to-all collective communication strategies important
  - All-to-all eventually dominates execution time
- Some basic optimizations easily applied
  - Varying order sends
    - Minimizes network contention
  - Only a subset of processors should send data to one destination at a time
    - Prevents network overload

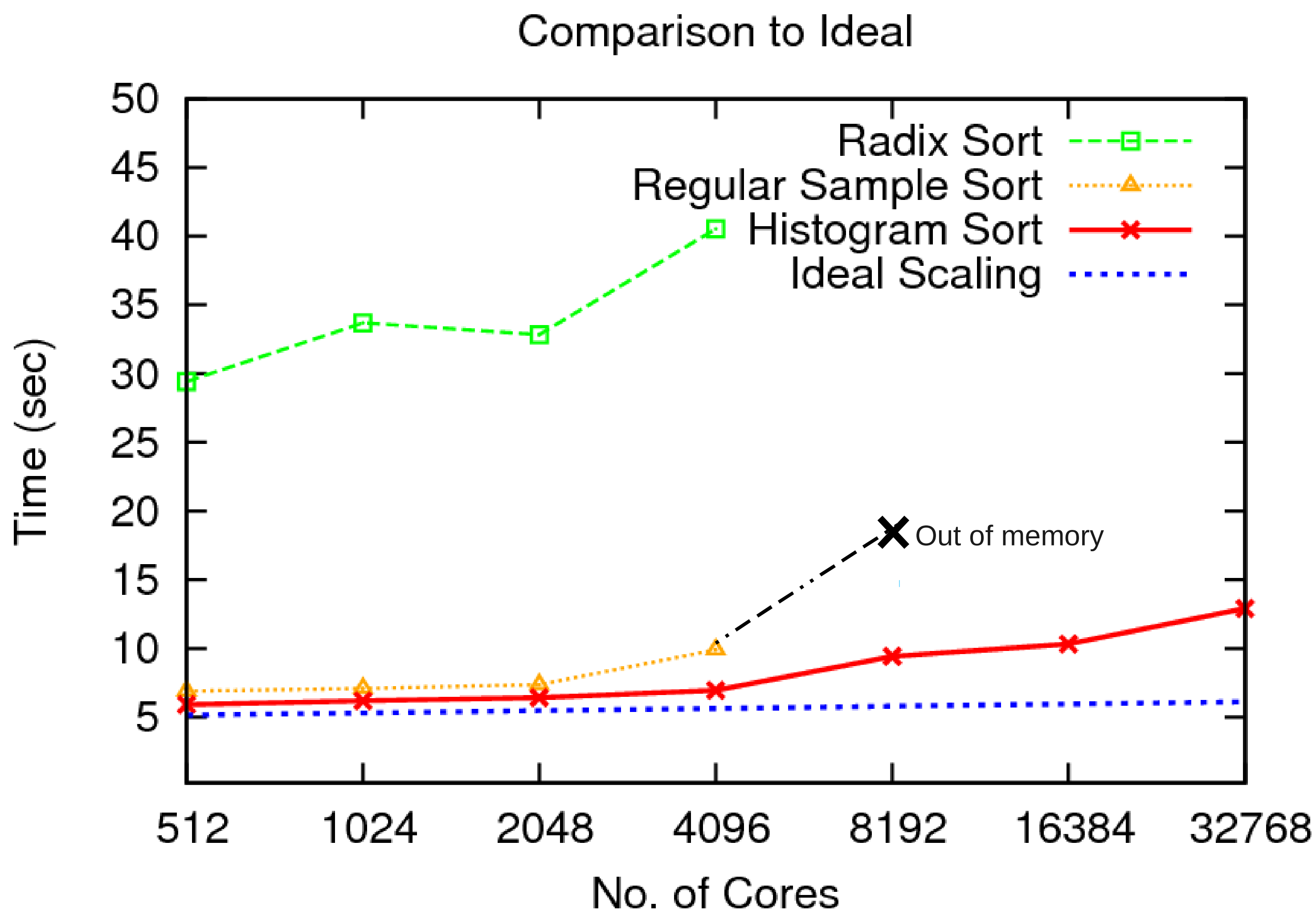
Communication Spread





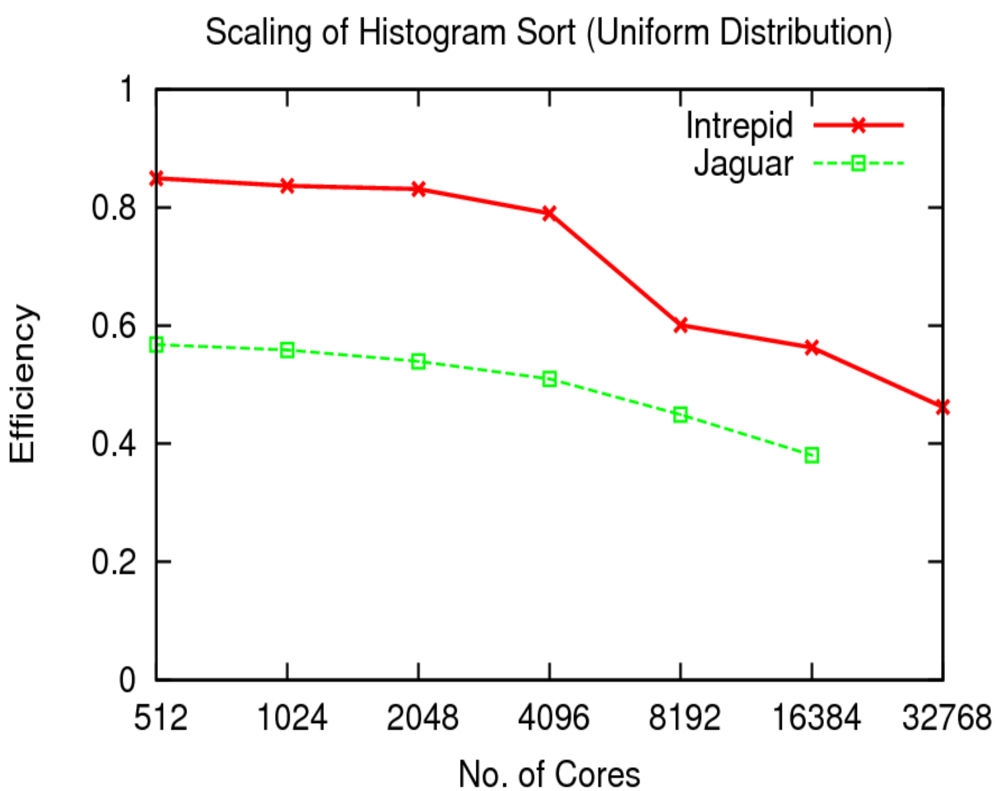
Tests done on 4096 cores of Intrepid (BG/P) with 8 million 64-bit keys per core.

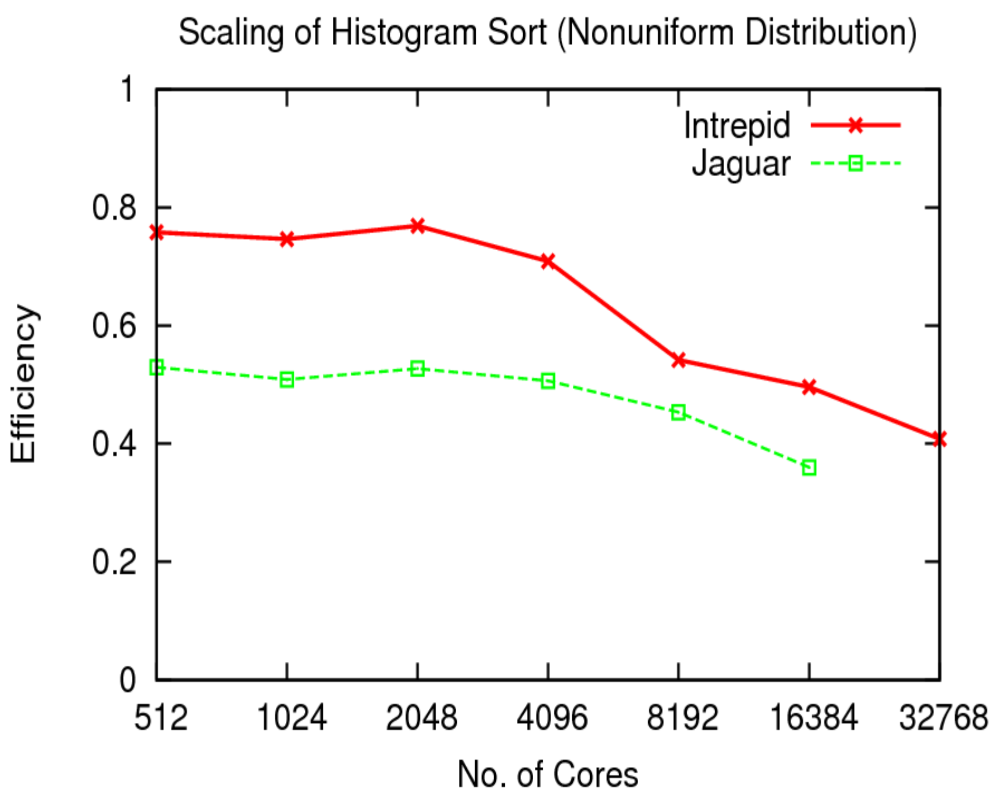
Algorithm Scaling Comparison



Tests done on Intrepid (BG/P) with 8 million 64-bit keys per core.

Histogram Sort Parallel Efficiency





Tests done on Intrepid (BG/P) and Jaguar (XT4) with 8 million 64-bit keys per core.

Some Limitations of this Work

- Benchmarking done with 64-bit keys rather than key-value pairs
- Optimizations presented are only beneficial for certain parallel sorting problems
  - Generally, we assumed $n > p^2$
    - Splicing useless unless $n/p > p$
    - Different all-to-all optimizations required if n/p is small (combine messages)
  - Communication usually cheap until $p > 512$
- Complex implementation another issue

Future/Ongoing Work

- Write a further optimized library implementation of Histogram Sort
  - Sort key-value pairs
  - Almost completed, code to be released
- To scale past 32k cores, histogramming needs to be better optimized
  - As $\boldsymbol{p \rightarrow n/p}$, probe creation cost matches the cost of local sorting and merging
  - One promising solution is to parallelize probing
    - Can use early determined splitters to divide probing

Contributions

- Improvements on original Histogram Sort algorithm
  - Overlap between computation and communication
  - Interleaved algorithm stages
- Efficient and well-optimized implementation
- Scalability up to tens of thousands of cores
- Ground work for further parallel scaling of sorting algorithms

Acknowledgements

- Everyone in PPL for various and generous help
- IPDPS reviewers for excellent feedback
- Funding and Machine Grants
  - DOE Grant DEFG05-08OR23332 through ORNL LCF
  - Blue Gene/P at Argonne National Laboratory, which is supported by DOE under contract DE-AC02-06CH11357
  - Jaguar at Oak Ridge National Laboratory, which is supported by the DOE under contract DE-AC05-00OR22725
  - Accounts on Jaguar were made available via the Performance Evaluation and Analysis Consortium End Station, a DOE INCITE project.