Highly Scalable Parallel Sorting

Edgar Solomonik and Laxmikant Kale
University of Illinois at Urbana-Champaign
April 20, 2010
Outline

- Parallel sorting background
- Histogram Sort overview
- Histogram Sort optimizations
- 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 $(n/p)+\text{threshold}$ 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 $n$ elements in $p^2$ messages
## Parallel Sorting Algorithms

<table>
<thead>
<tr>
<th>Type</th>
<th>Data movement</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Merge-based</strong></td>
<td></td>
</tr>
<tr>
<td>- Bitonic Sort</td>
<td>$\frac{1}{2}n \log^2(p)$</td>
</tr>
<tr>
<td>- Cole's Merge Sort</td>
<td>$O(n \log(p))$</td>
</tr>
<tr>
<td><strong>Splitter-based</strong></td>
<td></td>
</tr>
<tr>
<td>- Sample Sort</td>
<td>$n$</td>
</tr>
<tr>
<td>- Histogram Sort</td>
<td>$n$</td>
</tr>
<tr>
<td><strong>Other</strong></td>
<td></td>
</tr>
<tr>
<td>- Parallel Quicksort</td>
<td>$O(n \log(p))$</td>
</tr>
<tr>
<td>- Radix Sort</td>
<td>$O(n) \sim 4n$</td>
</tr>
</tbody>
</table>
Splitter-Based Parallel Sorting

- A **splitter** is a key that partitions the global set of keys 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

Number of Keys Smaller than x

0  k*(n/p)  n

key_min  Key  Splitter k  key_max
Sample Sort

- Processor 1 sorted data
- Processor p-1 sorted data
- Combine sorted data
- Extract local samples
- Sort combined sample
- Concatenate samples
- Extract splitters
- Broadcast 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 $O(p^2)$ 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

- Process or 1
- Processor \( p \) sorted data
- Test probe of splitter-guesses
- Broadcast probe
- Calculate histograms
- Add up histograms
- Analyze global histogram
- Apply splitters to data
- If probe not converged
- If converged
- All-to-All
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

1. Unsorted data
2. Splice data with probe
3. Sort chunks
4. Sorted data
5. Splice data with new probe
6. Search here
7. Splice here
Histogram Overlap Analysis

- Probe generation work should be offloaded to one processor
  - Reduces critical path
- Splicing is somewhat expensive
  - $O((n/p) \times \log(p))$ 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

Receive message with resolved ranges

Send to destination processor

Send to destination processor

Extract chunk

Sort chunk

Unsorted Data

Sorted data
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

First chunk
Buffer 1

First chunk
Buffer 2

Two more chunks arrive

First chunk
Second chunk

Merge

First merged data

First chunk

Merge

First merged data

Another chunk arrives

Third chunk

Fourth chunk

Merge

First merged data

Second merged data

Two more chunks arrive

Third chunk

Fourth chunk

Merge

First merged data

Second merged data

Final merged data

First merged data

Second merged data

Merge

Final merged data
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
Tests done on 4096 cores of Intrepid (BG/P) with 8 million 64-bit keys per core.
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 $p \rightarrow n/p$, 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.