CUG 2015 Technical Talk: Applications

MP-sort:
Sorting at Scale on BlueWaters in BlueTides Simulation

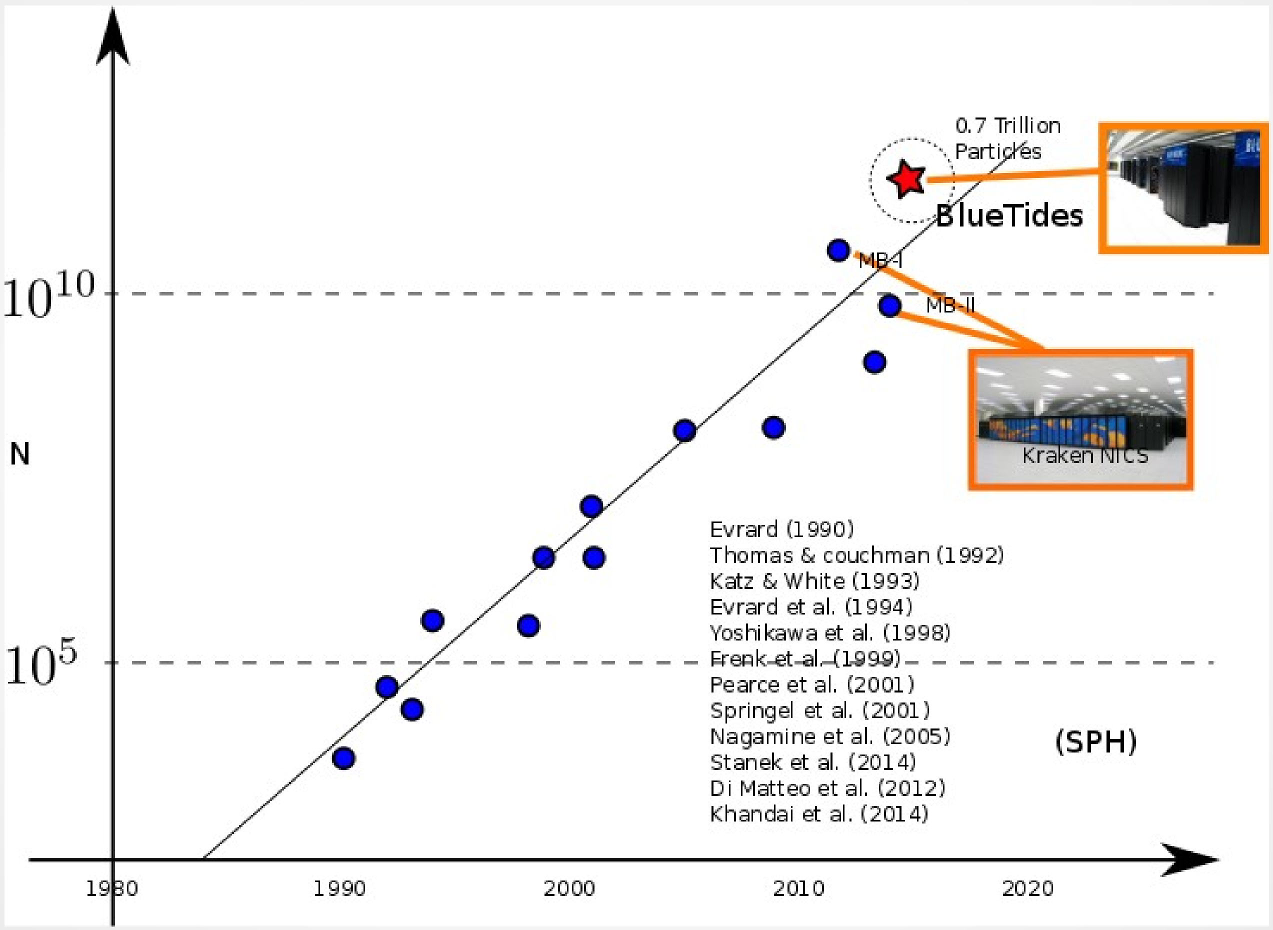
Yu Feng (UCB), Mark Straka (NCSA),
Tiziana Di Matteo (CMU), Rupert Croft (CMU)

Supported by NCSA and NSF OCI-0725070, OCI-0749212 and AST-1009781

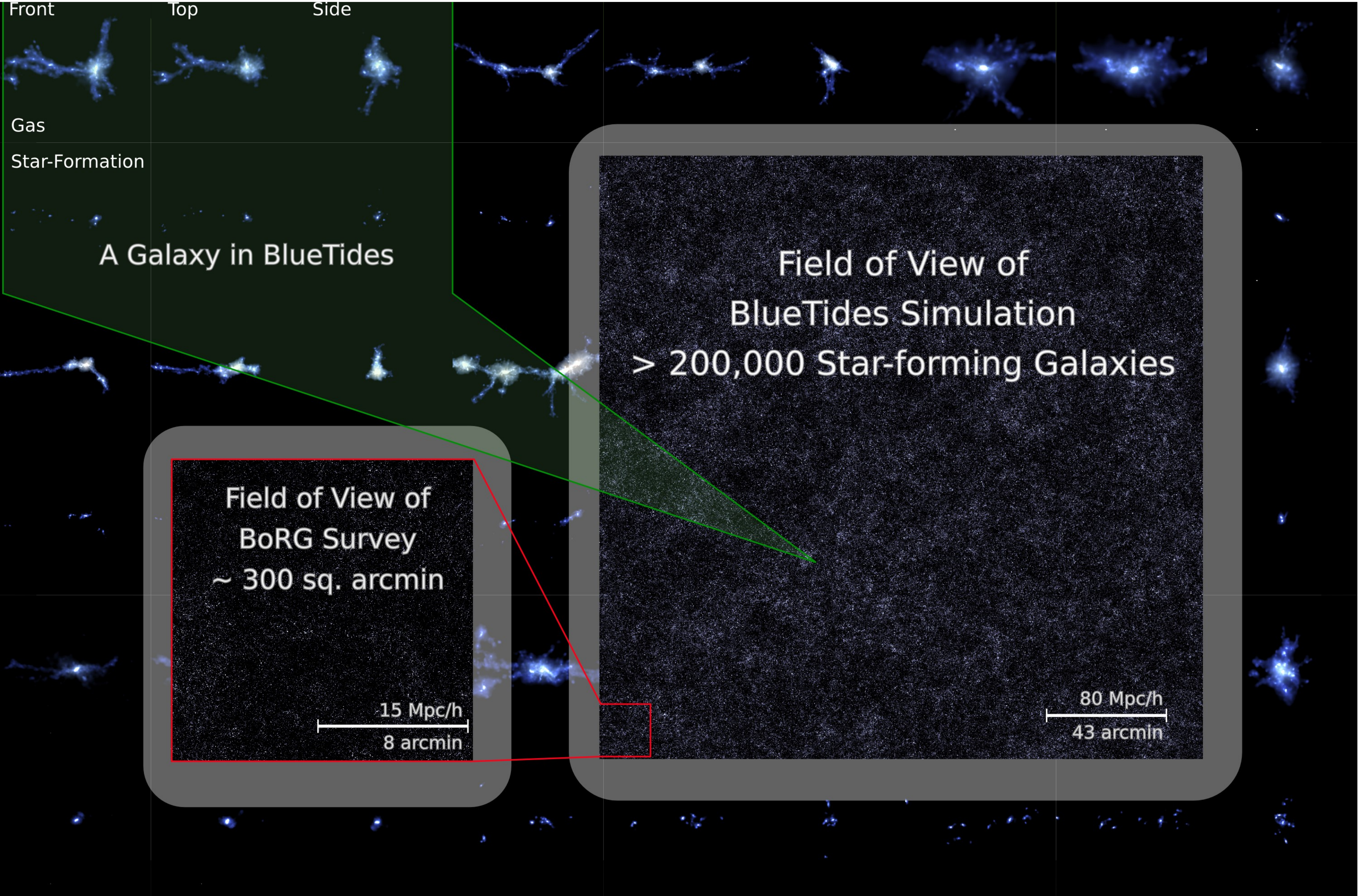
BlueTides Simulation

- Largest hydro-dynamical simulation of the universe;
- 700 Billion Particles;
- 20250 of Cray XE nodes in BlueWaters; (90% utilization)
- 81000 MPI ranks, 8 OpenMP Threads each;
- MP-Gadget:
  - Substantially improved scaling for BlueTides

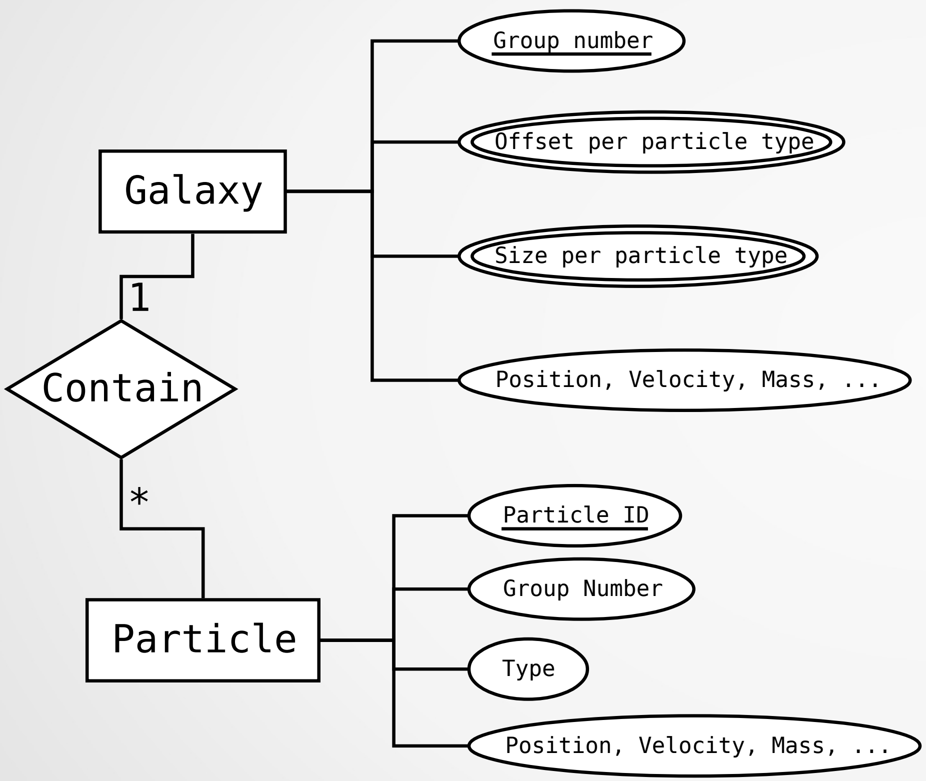
BlueTides on the Chart



BlueTides: How did first galaxies rise from a uniform universe?

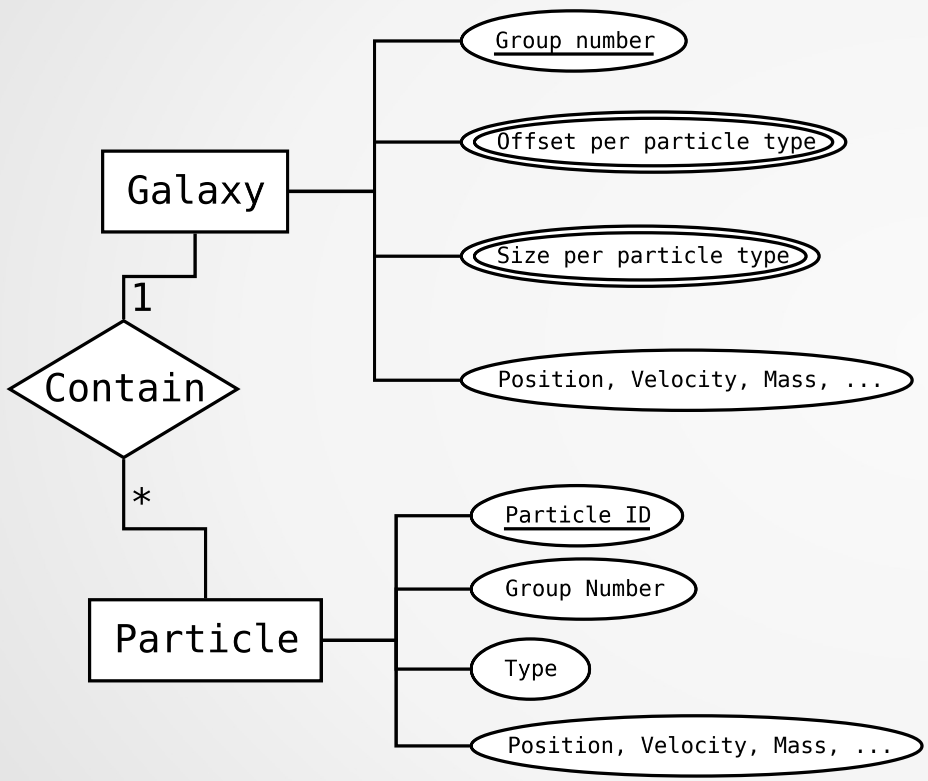


Galaxy Catalog



- A physicist's database:
  - **Sort** particles by their Group Number
  - Store a jump-table for the offset of the first particle in a galaxy
  - More complicated in reality, because particles have different types
  - Google: **bigfile github**

Galaxy Catalog



- A physicist's database:
  - **Sort** particles by their Group Number
  - Store a jump-table for the offset of the first particle in a galaxy
  - More complicated in reality, because particles have different types
  - Google: **bigfile github**

Introducing MP-sort

- At BlueTides scale (81,000 ranks, choice of sorting algorithm matters.
  - Comparison based, parallel Merge-sort scales badly.
- MP-sort is the new sorting module in BlueTides Simulation
  - Partition-based sorting
  - Performs reasonably well
  - A standalone library
    - Simple API, via C and Python
    - Small code footprint (< 2,000 lines)
  - http://github.com/rainwoodman/MP-sort

MP-Sort: Partition-Based Sorting

- Many names:

  Partition-sort, histogram-sort, bucket-sort;
- Distributed data
- Naive algorithms
- “Plan & Deliver”
- Need numerical keys for items
  - galaxy number

- Algorithm
  1. Local Sorting
  2. Find Splitters: edges of the histogram bins;
  3. Solve for Shuffling Matrix (P x P);
  4. Shuffle Items: moving from initial ranks to the final ranks
  5. Local Sorting

Partition-Based Sorting Illustrated

`[8 6 4 2 0] [9 7 4 3 1]`

1. Local Sorting

[0 2 **4** 6 8] [1 3 **4** 7 9]

2. Find Splitters

`(0, 4, 10)`

3. Calculating Shuffling Matrix

`[0 2 4 6 8] [1 3 4 7 9]`

4. Shuffle with `MPI_Alltoallv`

`[0 2 4 1 3] [6 8 4 7 9]`

5. Local Sorting

`[0 1 2 3 4] [4 6 7 8 9]`

Partition-based Sorting: Remarks

- Simplest Implementation
  - Local sorting: `qsort_r`
  - Splitter finding: binary search
  - Shuffle: `MPI_Alltoallv`
- Plan & Deliver:
  - Any item is on the network at most once.
- Only non-trivial step is to solve for Shuffling Matrix.

Step 3: Solving for Shuffling Matrix

```
(0, 4, 10)  [0 2 4 6 8] [1 3 4 7 9]
```

- Shuffling Matrix L[q, p]:
  - **SendDispl:** Rank p sends items L[q – 1, p] : L[q, p] to Rank q;
  - **Bounded** by $C_1[q, p] \le L[q, p] \le C_2[q, p]$
  - **Constrained** by total number of items to be received per rank
- Lower and Upper Bounds:
  - $C_1[q, p]$ is the total number of items **less than** splitter E[q];
  - $C_2[q, p]$ is the total number of items **less than or equal to** splitter E[q].
  - C_1 = [(0, 2, 5), (0, 2, 5)], C_2=[(0, 3, 5), (0, 3, 5)]

Parallel Solver

(0, 4, 10) [0 2 4 6 8] [1 3 4 7 9]

- For every column in L
  - Initialize with lower bound C_1
  - Increase the items in L from lower to high row, limited by the upper bound C_2
  - Until the column sum equals to the expected (cumulative) sum.
- Parallel in columns

C_1=[(0, 2, 5),
(0, 2, 5)]
C_2=[(0, 3, 5),
(0, 3, 5)]
L=[(0, 2, 5),
(0, 2, 5)]
✘ Sum of L: 0, 4, 10
L=[(0, 3, 5),
(0, 2, 5)]
✓ Sum of L: 0, 5, 10

Shuffling Matrix Solver: Remarks

- With parallelism:
  - Time complexity is O(P);
  - memory requirement per rank is O(P);
- Without parallelism both becomes $O(P^2)$
  - 100,000 Ranks => 10G elements in Shuffling Matrix!
- Communication overhead is small
  - 3 AlltoAll communication to transpose C_1, C_2, and L.
- Stable
  - Maintaining relative ordering of non-unique items
  - Items from lower ranks are sent to lower ranks

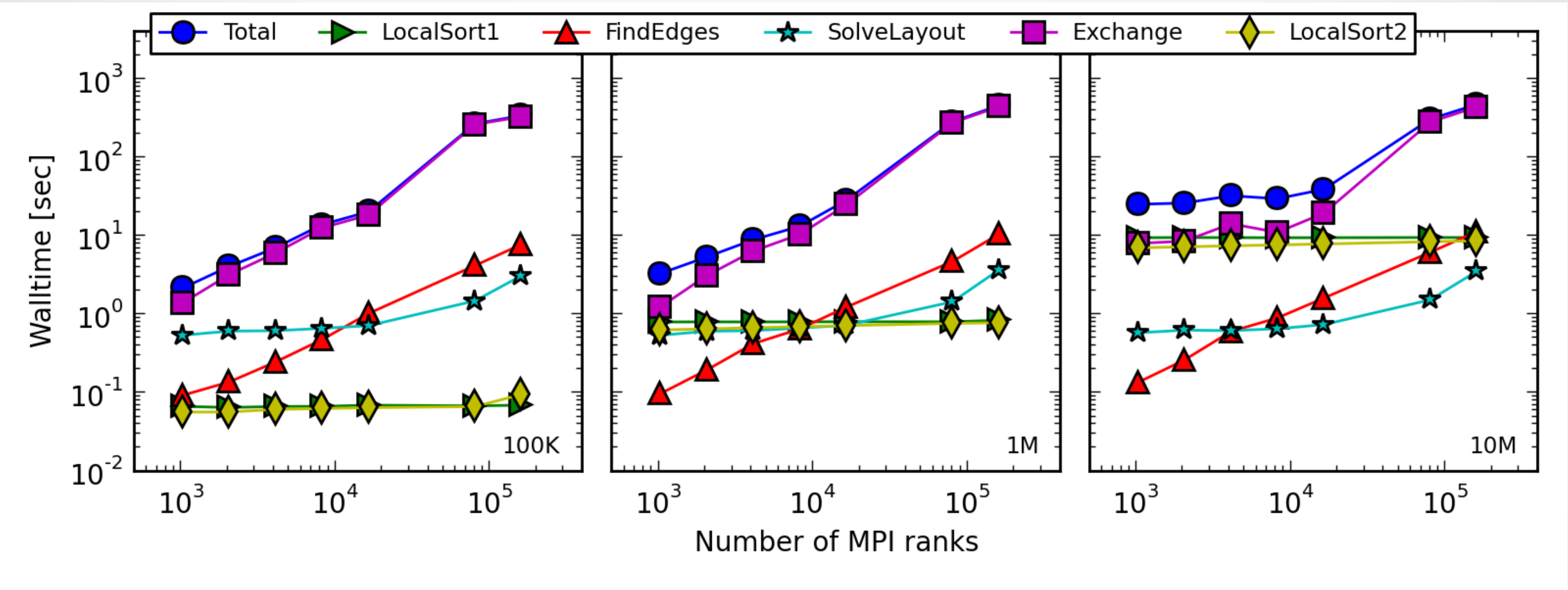
MP-Sort: Algorithm Summary

- Intuitive algorithm:
  - Massively parallel sorting in 5 steps
- Standard routines:
  - `qsort_r`, bindary search and `MPI_Alltoallv`
- No local optimization was done
- “Plan & Deliver”
  - A single call to `MPI_Alltoallv`
  - Items are on the network at most once
  - **Optimal Communication**

Benchmarks

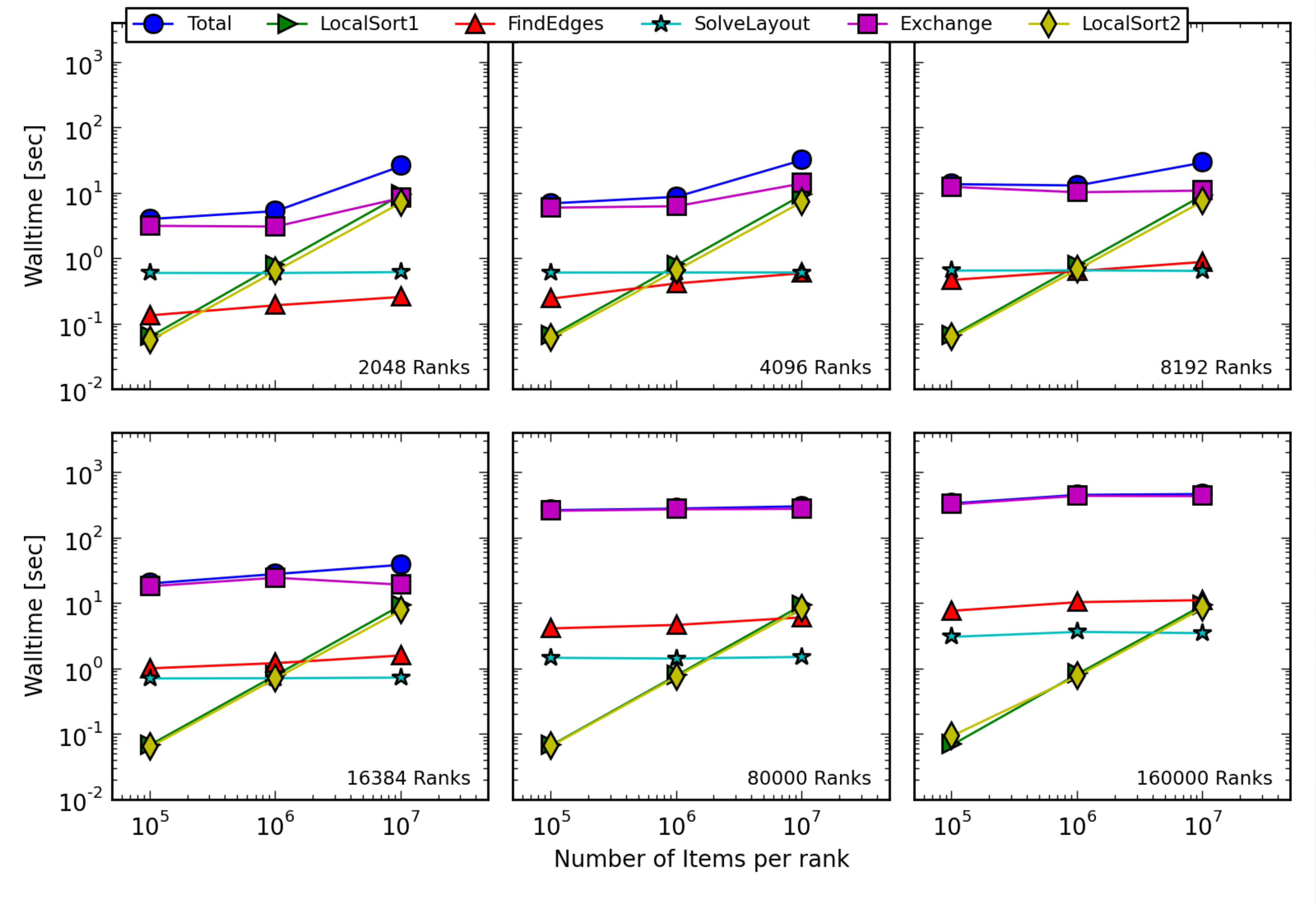
How does MP-sort perform?

Scaling with fixed load



Single call to `MPI_Alltoallv`
Optimal communication
99% of wall-time

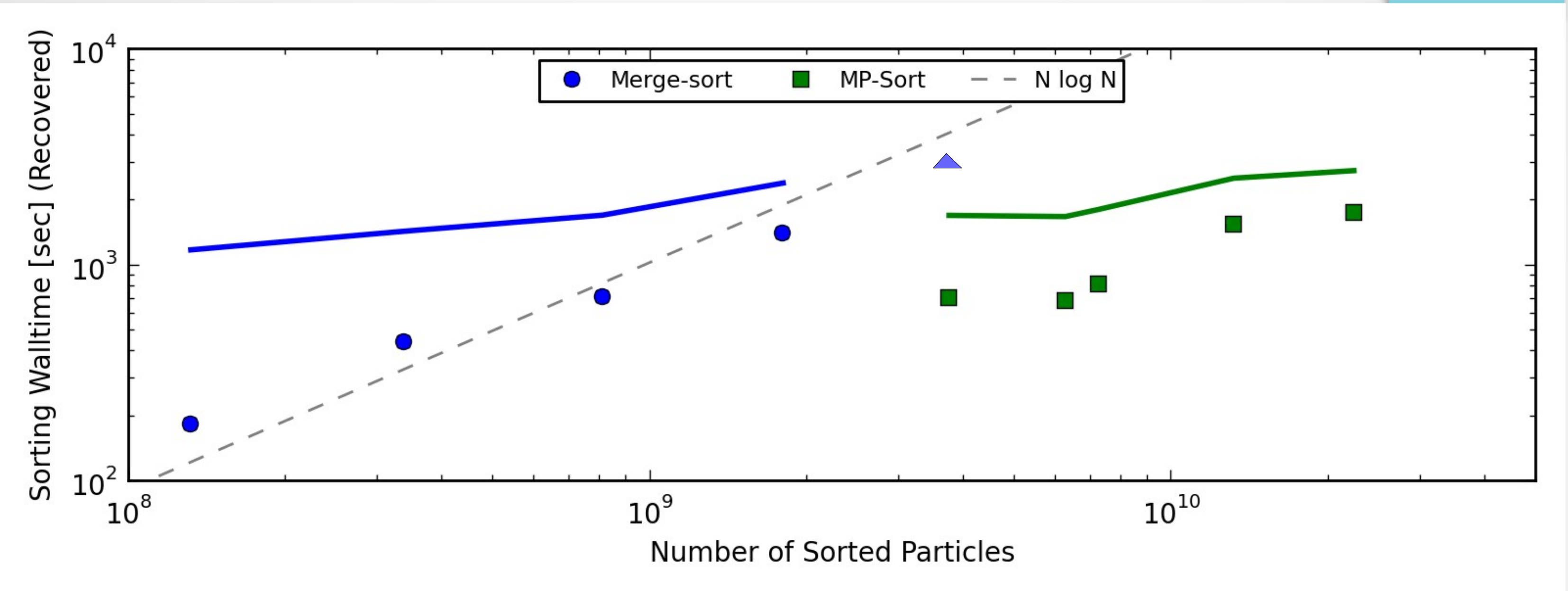
Scaling with fixed ranks



Insights from Benchmarks

- In large scale parallel applications (~100,000 MPI ranks)
  - Effectiveness of local optimization can be marginalized
  - Because, communication eventually takes over (99% of walltime)
- What does not help:
  - Overlapping communication with local sorting
  - Merge instead of sorting in final step
  - Requiring unique keys
- What really helps:
  - Faster inter-connection network, lower latency and higher bandwidth;
  - And maybe, a smarter `MPI_Alltoallv`

Production in BlueTides



- 10x faster than the old merge sort module
- Sorting is no longer the bottleneck
- ~ 2000 seconds per catalog
- ~ 20 catalogues produced, and actively used in scientific publications

Further Insights

- MP-sort is a key part enabling the scientific discovery in BlueTides
- Building “relational” scientific simulation data
  - (somewhat) Big Data in a traditional HPC environment
  - Database perspective, without database management systems
  - Efficiently; as fast as the BlueWaters allows
- Parallel non-numerical algorithms alike have a place in large scale numerical applications

Conclusion

- MP-sort: A Library for Massively Parallel Sorting
  - Optimal in communication
  - Performed at scale on BlueWaters for BlueTides simulation
  - Scaling Tests up to 160,000 cores
    - `MPI_Alltoallv` is the key
  - A tool for Big Data analysis on traditional HPC infrastructure
- http://github.com/rainwoodman/MP-sort
  - C Interface
  - Python Interface
  - Like MP-sort on Github!

Building the galaxy catalog

1. Assign global index to particles;
2. **Sort** global index of particles by galaxy/group number;
3. Assign ranks to particles
4. **Sort** ranks of particles by global index;
5. Exchange particles to the ranks with particle-exchange module

Sorting is used **twice!**

Weak Scaling Summary

- At scale (large load and large number of ranks), communication dominates the total time
- Hardware and software implementation of MPI_Alltoall seems to treat large number of ranks differently, as seen by the sudden jump at 80000 ranks.
- Matrix solver scales worse than linear:
  - A large fraction of time is in Alltoall of C_1, C_2 and L;
  - Still small fraction of time than the Alltoall of data items.
- Local sorting always a small fraction of wall-time.

Galaxy catalogue

- Galaxy catalog (PIG)
  - Less than 5% of all particles; or 1.5 TB in size;
  - Contains all galaxies;
  - Particles are indexed by galaxies;
- Full snapshot:
  - 40 TB per snapshot
  - Hard to transfer and analyze
  - challenging for offline analysis

Step 3: Parallel Solver Example

C_1

0	0	0	0	0
2	1	1	2	6
3	2	2	4	11
4	4	4	6	18
6	6	5	7	24

C_2

Recv->

0	0	0	0	0
2	1	1	3	7
3	2	3	5	13
5	4	4	6	19
6	6	5	7	24

Sender -> Sum

L

0	**0**	**0**	**0**	**0**
2	1	1	2	6
3	2	2	4	11
4	4	4	6	18
6	**6**	**5**	**7**	**24**

L

0	**0**	**0**	**0**	**0**
2	**1**	**1**	**2**	**6**
3	2	2	4	11
4	4	4	6	18
6	**6**	**5**	**7**	**24**

L

0	**0**	**0**	**0**	**0**
2	**1**	**1**	**2**	**6**
3	**2**	**3**	**4**	**12**
4	4	4	6	18
6	**6**	**5**	**7**	**24**

L

0	**0**	**0**	**0**	**0**
2	**1**	**1**	**2**	**6**
3	**2**	**3**	**4**	**12**
4	**4**	**4**	**6**	**18**
6	**6**	**5**	**7**	**24**

Strong Scaling Summary

- Small number of ranks
  - The single AlltoAll operation uses a small fraction of walltime (~ 30%)
  - Increasing number of items increases walltime; due to increased Local sorting time
- Large number of ranks
  - The single AlltoAll operation uses a large fraction of walltime (~ 90%)
  - Increasing number of items does not increase walltime;
  - Walltime of local sorting is negligible;
  - Walltime of Split and Matrix solver is stable and negligible.