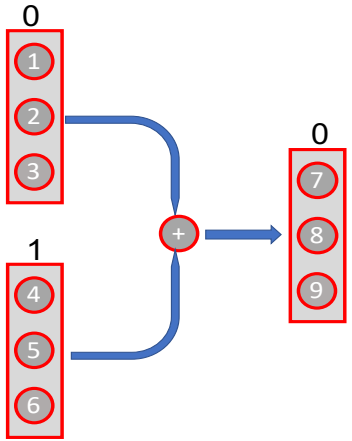
Improving MPI Reduction Performance for Manycore Architectures with OpenMP and Data Compression

Hongzhang Shan, Samuel Williams
Performance and Algorithms Research Group
Computational Research Division
Lawrence Berkeley National Laboratory

Calvin W. Johnson
Department of Physics
San Diego State University

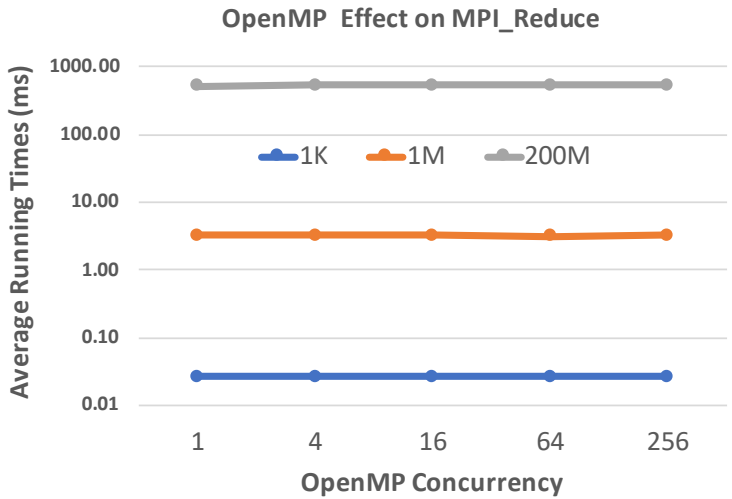
Background

- When parallelized, many numerical methods require dot or norm calculations, result in global reduction on scalar values
- In multidimensional parallelization, lead to global broadcast and reduction on large vectors of size $N/\sqrt{P}$ (could be megabytes or gigabytes)
- MPI_Reduce succinctly express such requisite functionality and are widely used in many applications
  - int MPI_Reduce(
    - const void *sendbuf,
    - void *recvbuf,
    - int count,
    - MPI_Datatype datatype,
    - MPI_Op op,
    - int root,
    - MPI_Comm comm)



Motivation

- MPI_Reduce on large vectors often become the performance scaling bottleneck
- Most optimizations focus on latency and bandwidth



- **Problem:**
  - Low single core performance
  - Idle threads
  - No performance improvement

Approach

- Two new perspectives to improve reduce performance:
  - Using OpenMP threads to accelerate local reduction
  - Using data compression to reduce communication volume
- Accomplishments:
  - Could perform up to 4X better than the Cray implementation on large vector size
  - Improve application BIGSTICK performance up to 2.6X

Outline

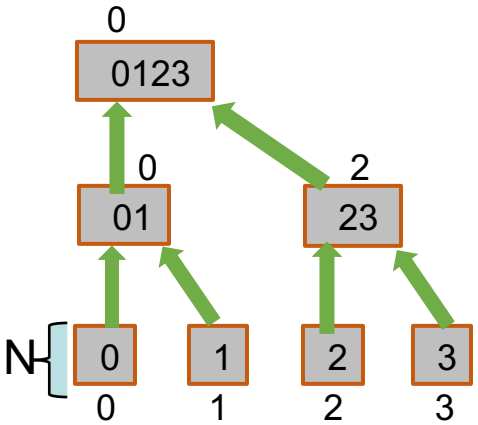
- Background and Motivation
- Algorithms for MPI_Reduce and their performance characteristics
  - Binomial, Reduce-Scatter, Ring
- Performance analysis for MPI_Reduce
- Performance analysis for application BIGSTICK
- Summary and future work

Algorithms

- MPI_Reduce:
  - Binomial for short messages
  - Reduce-Scatter for long messages
  - Ring
- Simple Cost Model ($\alpha + N\beta + N\gamma$)
  - N: vector size
  - α: latency
  - β: transfer time per byte
  - γ: compute time per byte (aggregate)

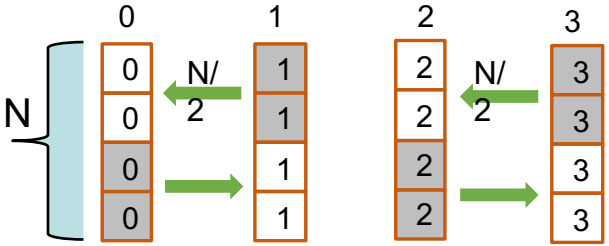
Binomial

- **Cost:**
  - $log(P)\times(\alpha + N\beta + N\gamma)$
- **Case: P = 4**
  - The numbers in the block represent the process ranks whose data has already been reduced.
  - The shaded blocks represents the communication data.

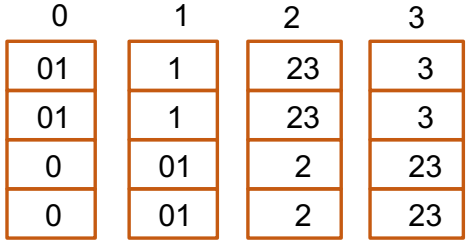


Algorithm: Reduce-Scatter

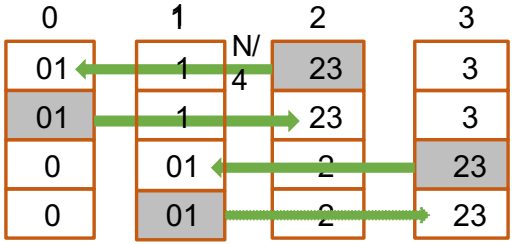
- **Cost Model:** $2\alpha log(P) + 2\frac{P-1}{P}N\beta + \frac{P-1}{P}N\gamma$



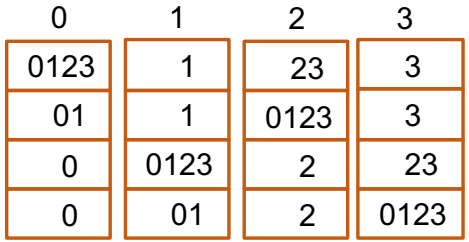
a) Initial status and communication in 1st step



b) After 1st step status



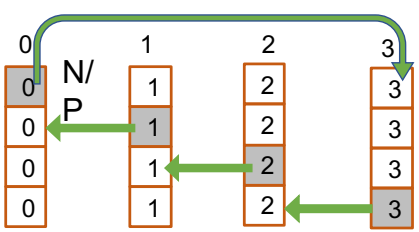
c) 2rd step communication



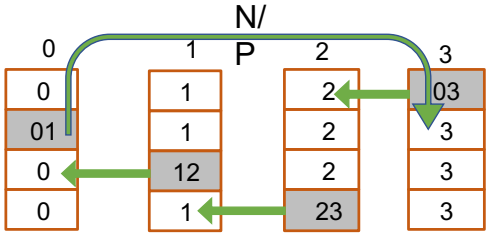
d) After 2rd step status

Algorithm: Ring

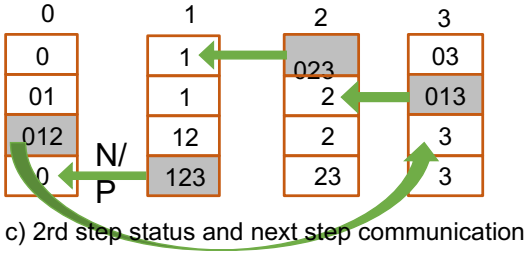
- **Cost Model:** $(\log(P) + P)\alpha + 2\frac{P-1}{P}N\beta + \frac{P-1}{P}N\gamma$



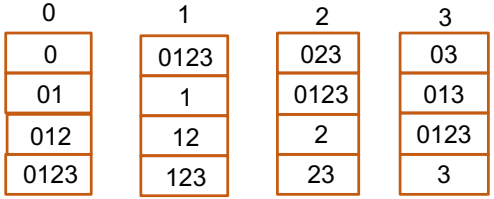
a) Initial status and communication in 1st step



b) 1st step status and next step communication

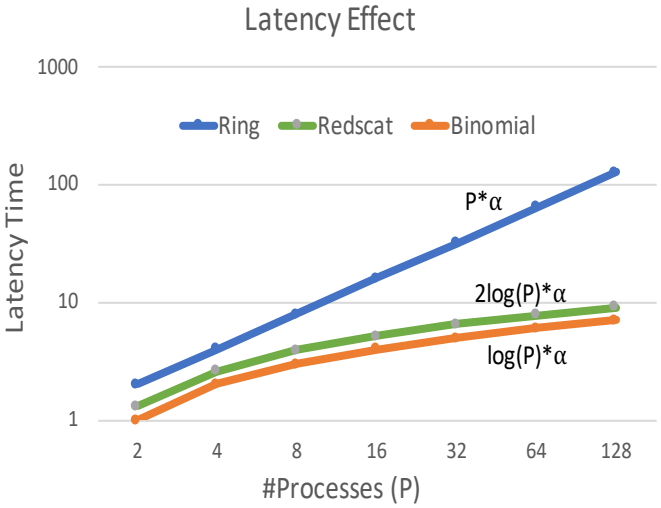


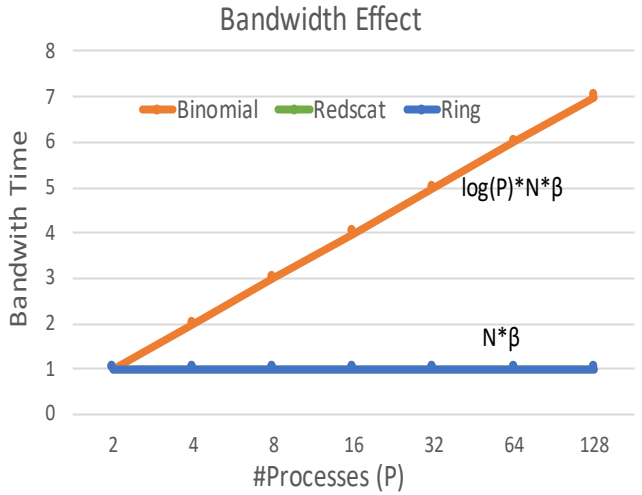
c) 2rd step status and next step communication



d) Final status

Performance Characteristics





- **Binomial:** $log(P)\alpha + lop(P)N\beta + lop(P)N\gamma$
- **RedScat:** $2log(P)\alpha + 2\frac{P-1}{P}N\beta + \frac{P-1}{P}N\gamma$
- **Ring:** $(\log(P) + P)\alpha + 2\frac{P-1}{P}N\beta + \frac{P-1}{P}N\gamma$

OpenMP Implementation

- Each thread has its own independent buffers
  - Float *rbuffer[NTHREADS]
- Using OpenMP parallel Region to implement workload partition:
  - #pragma omp parallel
  - {
    - tid = omp_get_thread_num()
    - nthreads = omp_get_num_threads()
    - partition the work manually
    - perform the assigned work
  - }

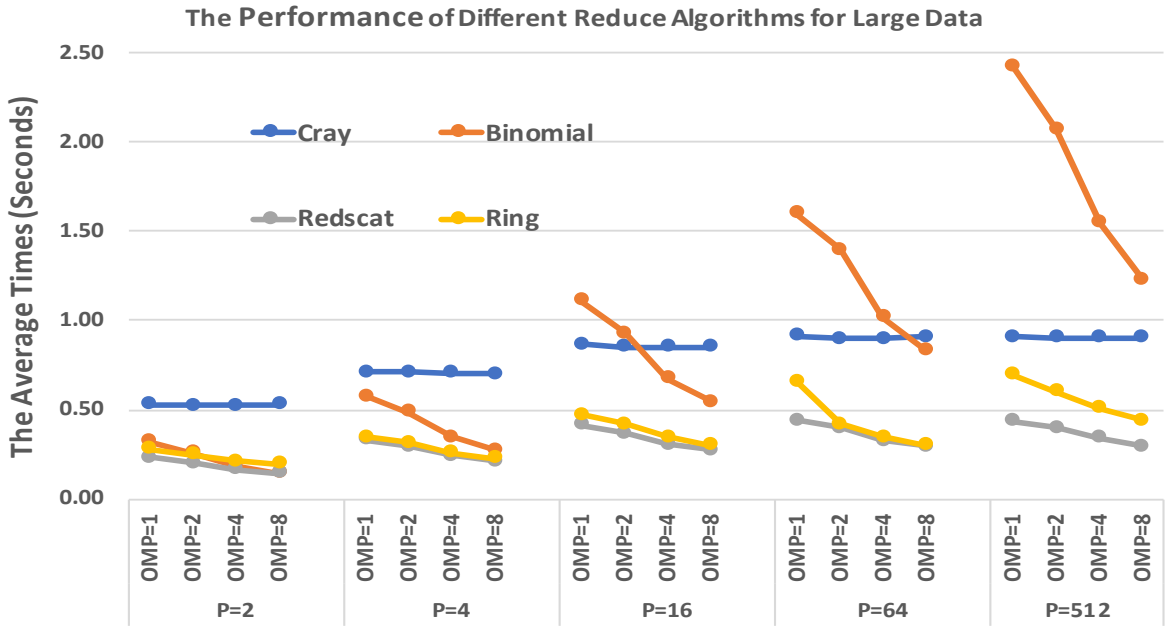
Outline

- Background and Motivation
- Algorithms for MPI_Reduce and their performance characteristics
  - Binomial, Reduce-Scatter, Ring
- **Performance analysis for MPI_Reduce**
- **Performance analysis for application BIGSTICK**
- **Summary and future work**

Platform

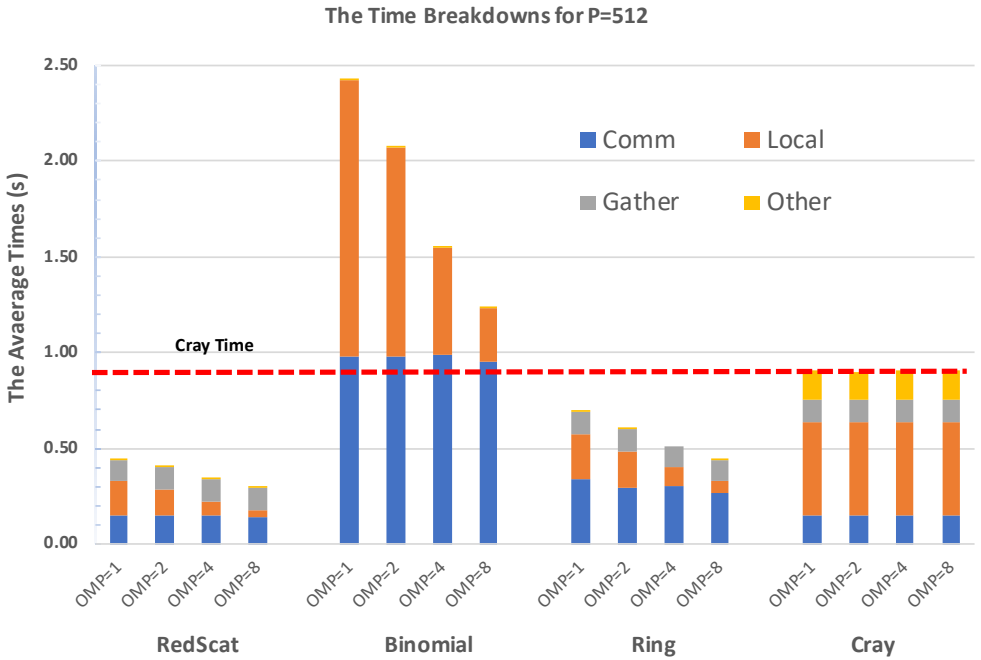
- Cori-KNL, located at NERSC
- Programming Environment:
  - PrgEnv-intel/6.0.4
  - MPI Version: cray-mpich/7.7.0
  - Compiler: Intel/18.0.1.163
- Baseline: DMAPP enabled Cray implementation
  - MPICH_NEMESIS_ASYNC_PROGRESS=MC
  - MPICH_MAX_THREAD_SAFETY=multiple
  - MPICH_USE_DMAPP_COLL=1
- Using OMP_NUM_THREADS to set the number of OpenMP threads

Performance: Micro Large Messages



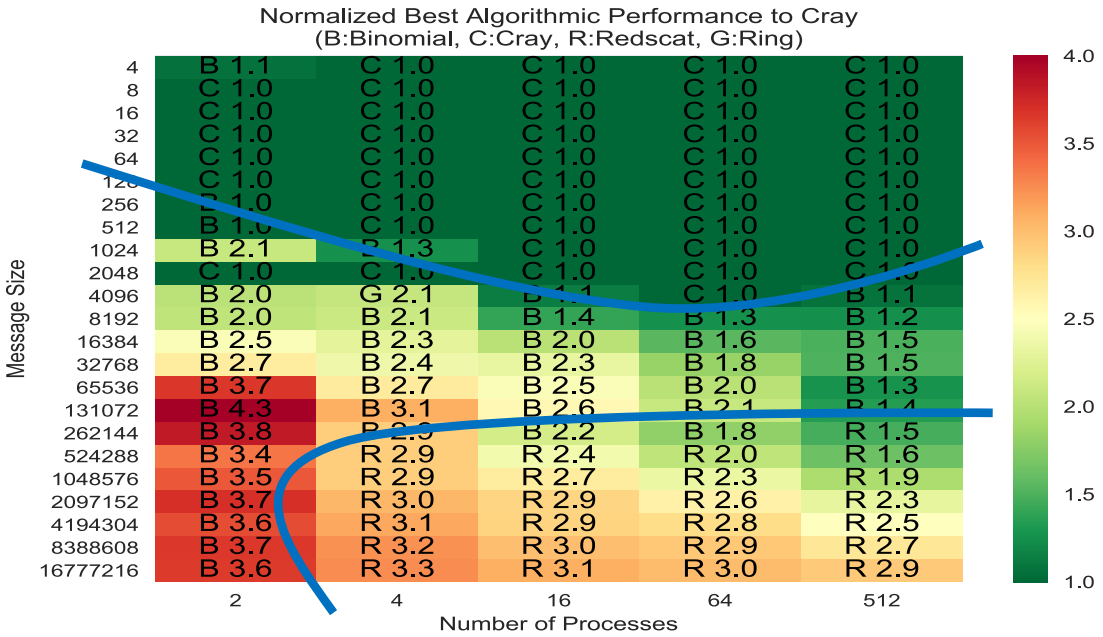
- N=200M floats, one MPI process/node
- Redscat performs best, significant better performance than Cray
- Threading is important

Time Breakdown: Micro Large Messages



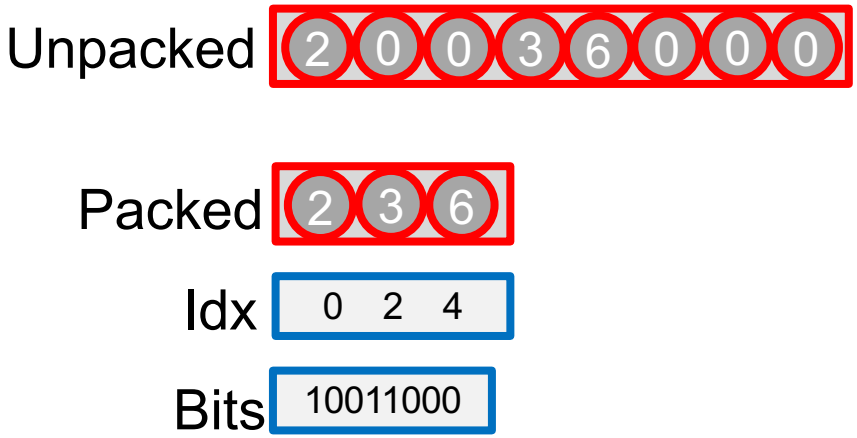
- **Cray: local time dominate, threading does not help**
- **Other three: threading reduce the local time significantly; the "other" time has been eliminated**

Performance: Micro
Small Messages



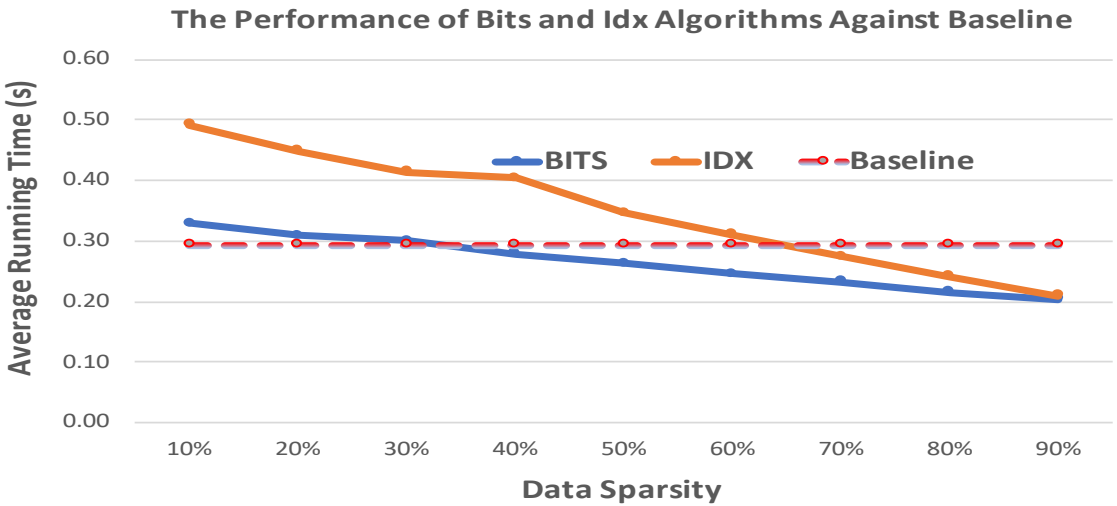
- **Msg sizes < 2K floats, Cray performs best**
- **Msg sizes > 512K floats, Redscat performs best**
- **Msg sizes between 2K – 512K, binomial performs best**

Data Sparsity



- **Algorithm IDX: using an extra index array**
- **Algorithm BITS: using one bit to indicate whether the element is zero or not**

Performance: Data Sparsity

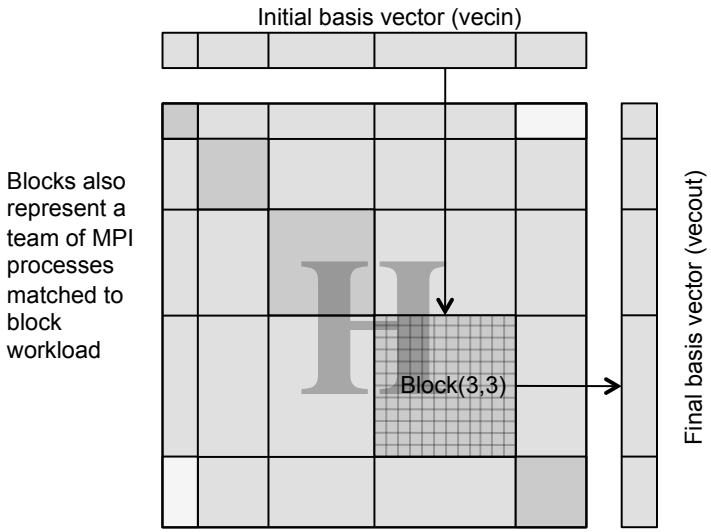


- Implemented on top of threading and the packing/unpacking is vectorized efficiently
- The break-even point is around 35% for BITS and 65% for IDX

Outline

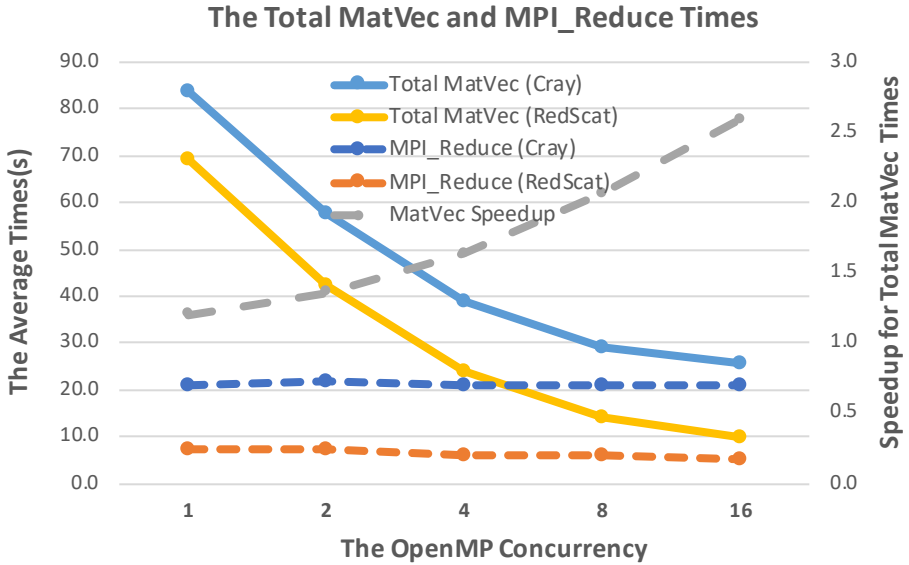
- Background and Motivation
- Algorithms for MPI_Reduce and their performance characteristics
  - Binomial, Reduce-Scatter, Ring
- Performance analysis for MPI|Reduce
- **Performance analysis for application BIGSTICK**
- **Summary and future work**

BIGSTICK



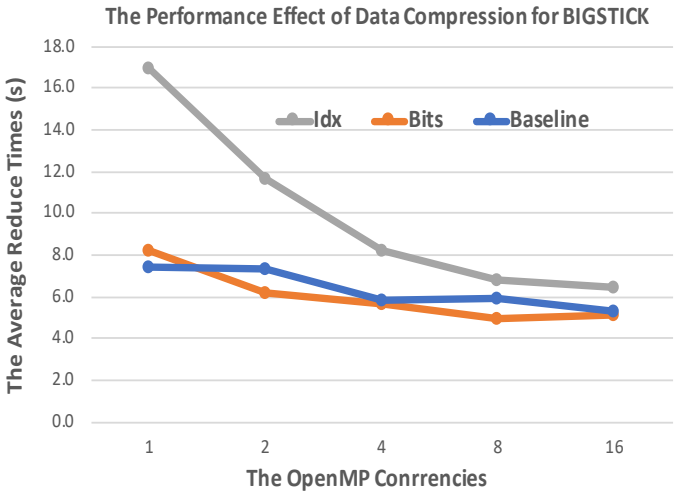
- Dominated by sparse matrix vector multiplication
- Vector size is big, around 10.6 billion elements for ^{130}Cs
- Communication: MPI_Reduce in row direction; MPI_Bcast in col direction

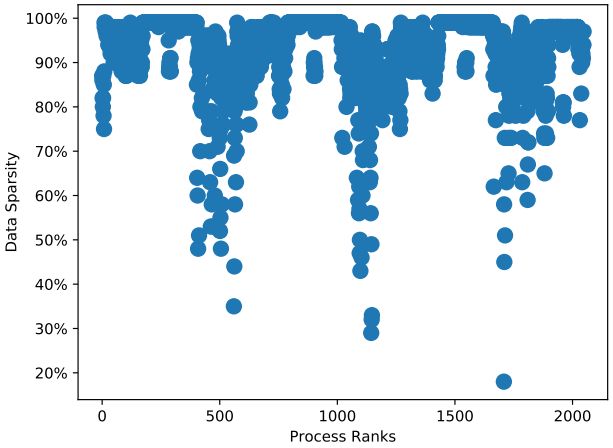
Performance: BIGSTICK Threading



- BIGSTICK scaling performance is highly related with MPI_Reduce
- Using threading could improve the performance up to 2.5X

Performance: BIGSTICK Data Compression





- Data compression does not help performance though the average data sparsity is over 90%
- The density imbalance destroys the performance advantage of data compression

Conclusion

- Making effective use of the idling threads on the manycore architectures could significantly improve large MPI_Reduce performance
- Our threading algorithms performs up to 4x better than the Cray implementation and improve the BIGSTICK performance up to 2.6x.
- Compressing the sparse data could also help the performance, but may suffer from density imbalance.

Future Work

- Extend out algorithms to more applications and other HPC platforms
- Using threads to increase network injection bandwidth
- Develop smart algorithms to address the density imbalance problem
- Develop hierarchical algorithms to combine advantages of different ones
- Develop auto-tuning framework for MPI persistent reduce functions

Acknoledgements

- This material is based upon work supported by the Advanced Scientific Computing Research Program in the U.S. Department of Energy, Office of Science, under Award Number DE-AC02-05CH11231.
- This research used resources of the National Energy Research Scientific Computing Center (NERSC), which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.