

# JLESC Project: Toward taming large and complex data flows in data-centric supercomputing

François Tessier\*, Venkatram Vishwanath\*, Emmanuel Jeannot†

\*Argonne National Laboratory, USA

†Inria Bordeaux Sud-Ouest, France

November 29, 2016



## Project's status

### **Toward taming large and complex data flows in data-centric supercomputing**

- ▶ Summary
  - Collaboration between Argonne National Lab. and Inria Bordeaux
  - Started in the beginning of 2015
  - Understand, characterize, and transform application data flows over a range of architectures
- ▶ Visits and meetings
  - E. Jeannot visited ANL on March 2015 and June 2016
  - I stayed 10 days at ANL on March 2015
- ▶ Impact and publications
  - I moved from Inria to ANL in February 2016 for a postdoc position
  - Our first results have been published and presented at the COM-HPC Workshop few weeks ago (SC'16)

### **Topology and Affinity Aware Hierarchical and Distributed Load-balancing in Charm++**

- ▶ JLPC Inria-PPL project completed two years ago
- ▶ Recent publication at the COM-HPC Workshop (SC'16) thanks to the JLESC for Blue Waters allocations

# Topology-Aware Data Aggregation for Intensive I/O on Large-Scale Supercomputers

François Tessier\*, Preeti Malakar\*, Venkatram Vishwanath\*,  
Emmanuel Jeannot†, Florin Isaila‡

\*Argonne National Laboratory, USA

†Inria Bordeaux Sud-Ouest, France

‡University Carlos III, Spain

November 29, 2016



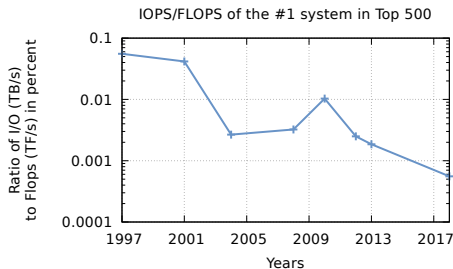
# Data Movement at Scale

- ▶ Computational science simulation such as climate, heart and brain modelling or cosmology have large I/O needs
  - Typically around 10% to 20% of the wall time is spent in I/O

**Table:** Example of I/O from large simulations

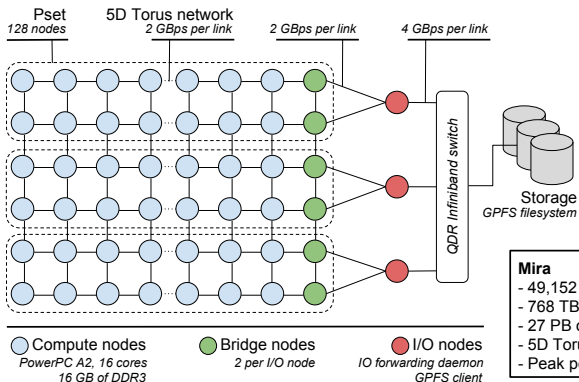
| Scientific domain   | Simulation  | Data size           |
|---------------------|-------------|---------------------|
| Cosmology           | Q Continuum | 2 PB / simulation   |
| High-Energy Physics | Higgs Boson | 10 PB / year        |
| Climate / Weather   | Hurricane   | 240 TB / simulation |

- ▶ Increasing disparity between computing power and I/O performance in the largest supercomputers



# Complex Architectures

- ▶ Complex network topologies: multidimensional tori, dragonfly, ...
- ▶ Partitioning of the architecture to reduce I/O interference
  - IBM BG/Q with I/O nodes (Figure), Cray with LNET nodes
- ▶ New tiers of storage/memory for data staging
  - MCDRAM in KNL, NVRAM, Burst buffer nodes



## Mira

- 49,152 nodes / 786,432 cores
- 768 TB of memory
- 27 PB of storage, 330 GB/s (GPFS)
- 5D Torus network
- Peak performance: 10 PetaFLOPS

## Two-phase I/O

- ▶ Available in MPI I/O implementations such as ROMIO
- ▶ Improves I/O performance by writing larger data chunks
- ▶ Selects a subset of processes to aggregate data before writing it to the storage system

### Limitations:

- ▶ Poor for small messages (from experiments)
- ▶ Inefficient aggregator placement policy
- ▶ Fails to take advantage of data model, data layout and memory hierarchy

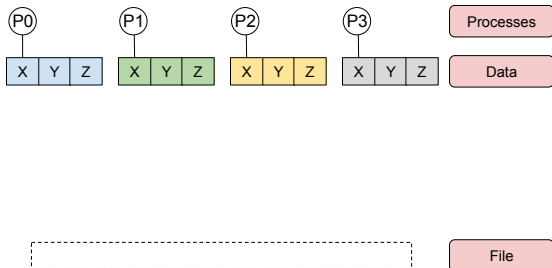


Figure: Two-phase I/O mechanism

## Two-phase I/O

- ▶ Available in MPI I/O implementations such as ROMIO
- ▶ Improves I/O performance by writing larger data chunks
- ▶ Selects a subset of processes to aggregate data before writing it to the storage system

### Limitations:

- ▶ Poor for small messages (from experiments)
- ▶ Inefficient aggregator placement policy
- ▶ Fails to take advantage of data model, data layout and memory hierarchy

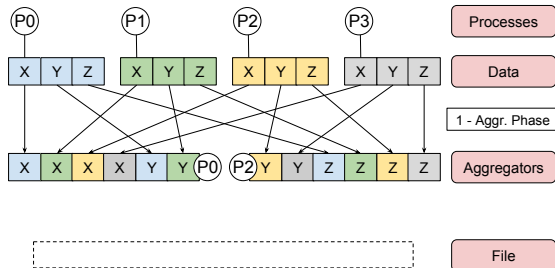


Figure: Two-phase I/O mechanism

## Two-phase I/O

- ▶ Available in MPI I/O implementations such as ROMIO
- ▶ Improves I/O performance by writing larger data chunks
- ▶ Selects a subset of processes to aggregate data before writing it to the storage system

### Limitations:

- ▶ Poor for small messages (from experiments)
- ▶ Inefficient aggregator placement policy
- ▶ Fails to take advantage of data model, data layout and memory hierarchy

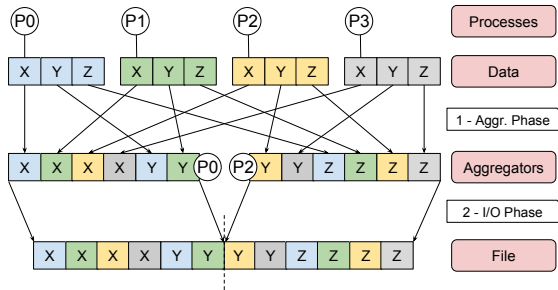


Figure: Two-phase I/O mechanism



# Outline

- 1 Context
- 2 Approach**
- 3 Evaluation
- 4 Conclusion and Perspectives

# Approach

## Improved aggregator placement while taking into account:

- ▶ The topology of the architecture
- ▶ The data access pattern

## Efficient implementation of the two-phase I/O scheme

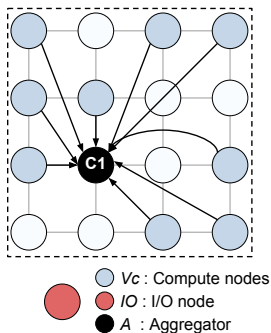
- ▶ Captures the data model and the data layout to optimize the I/O scheduling
- ▶ Pipelining of aggregation phase and I/O phase to optimize data movement
- ▶ Leverage one-sided communication
- ▶ Uses non-blocking operation to reduce synchronization

# Aggregator Placement - Topology-aware strategy

- ▶  $\omega(u, v)$ : Amount of data exchanged between nodes  $u$  and  $v$
- ▶  $d(u, v)$ : Number of hops from nodes  $u$  to  $v$
- ▶  $l$ : The interconnect latency
- ▶  $B_{i \rightarrow j}$ : The bandwidth from node  $i$  to node  $j$ .
- ▶  $C_1 = \max \left( l \times d(i, A) + \frac{\omega(i, A)}{B_{i \rightarrow A}} \right), i \in V_C$
- ▶  $C_2 = l \times d(A, IO) + \frac{\omega(A, IO)}{|V_C| \times B_{A \rightarrow IO}}$

## Objective function:

- ▶  $\text{TopoAware}(A) = \min (C_1 + C_2)$
- ▶ Computed by each process independently in  $O(n)$ ,  $n = |V_C|$

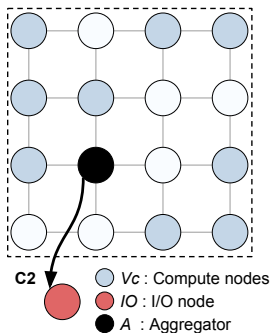


# Aggregator Placement - Topology-aware strategy

- ▶  $\omega(u, v)$ : Amount of data exchanged between nodes  $u$  and  $v$
- ▶  $d(u, v)$ : Number of hops from nodes  $u$  to  $v$
- ▶  $l$ : The interconnect latency
- ▶  $B_{i \rightarrow j}$ : The bandwidth from node  $i$  to node  $j$ .
- ▶  $C_1 = \max \left( l \times d(i, A) + \frac{\omega(i, A)}{B_{i \rightarrow A}} \right), i \in V_C$
- ▶  $C_2 = l \times d(A, IO) + \frac{\omega(A, IO)}{|V_C| \times B_{A \rightarrow IO}}$

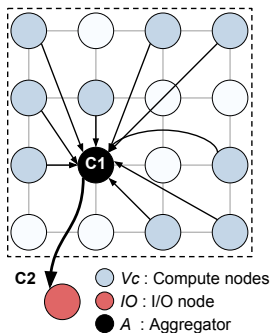
## Objective function:

- ▶  $\text{TopoAware}(A) = \min(C_1 + C_2)$
- ▶ Computed by each process independently in  $O(n)$ ,  $n = |V_C|$



# Aggregator Placement - Topology-aware strategy

- ▶  $\omega(u, v)$ : Amount of data exchanged between nodes  $u$  and  $v$
- ▶  $d(u, v)$ : Number of hops from nodes  $u$  to  $v$
- ▶  $l$ : The interconnect latency
- ▶  $B_{i \rightarrow j}$ : The bandwidth from node  $i$  to node  $j$ .
- ▶  $C_1 = \max \left( l \times d(i, A) + \frac{\omega(i, A)}{B_{i \rightarrow A}} \right), i \in V_C$
- ▶  $C_2 = l \times d(A, IO) + \frac{\omega(A, IO)}{|V_C| \times B_{A \rightarrow IO}}$

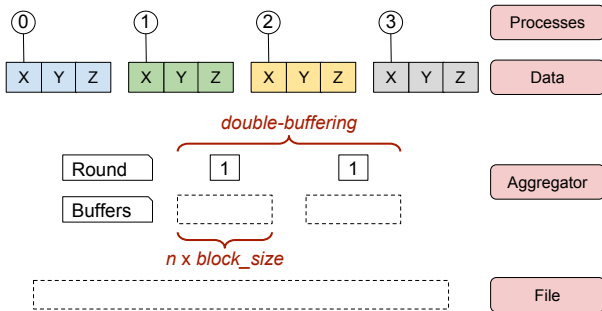


## Objective function:

- ▶  $\text{TopoAware}(A) = \min(C_1 + C_2)$
- ▶ Computed by each process independently in  $O(n)$ ,  $n = |V_C|$

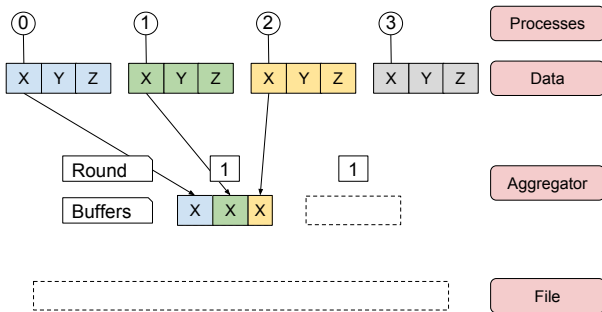
# Algorithm

- ▶ Initialization: allocate buffers, create MPI windows, compute tuples  $\{\text{round}, \text{aggregator}, \text{buffer}\}$  for each process  $P$ 
  - Let's say  $P_1$  is the aggregator
- ▶  $P_0, P_1$  and  $P_2$  put data in buffer 1 (round 1) of  $P_1$ .  $P_3$  waits (fence)
- ▶  $P_1$  writes buffer 1 in file and aggregates data from all the ranks in buffer 2
- ▶ 2<sup>nd</sup> round.  $P_1$  writes buffer 2 and aggregates data from  $P_1, P_2$  and  $P_3$
- ▶ and so on...
- ▶ Limitations: `MPI_Comm_split`, one aggr./node at most



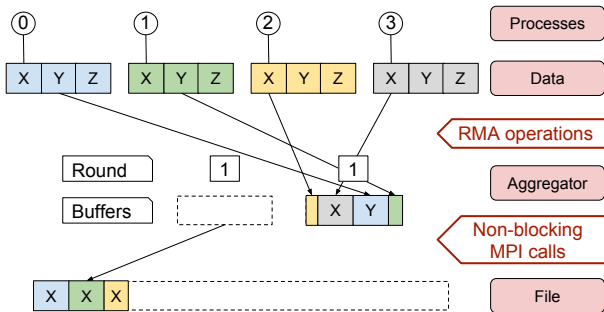
# Algorithm

- ▶ Initialization: allocate buffers, create MPI windows, compute tuples {round, aggregator, buffer} for each process P
  - Let's say P1 is the aggregator
- ▶ P0, P1 and P2 put data in buffer 1 (round 1) of P1. P3 waits (fence)
- ▶ P1 writes buffer 1 in file and aggregates data from all the ranks in buffer 2
- ▶ 2<sup>nd</sup> round. P1 writes buffer 2 and aggregates data from P1, P2 and P3
- ▶ and so on...
- ▶ Limitations: MPI\_Comm\_split, one aggr./node at most



# Algorithm

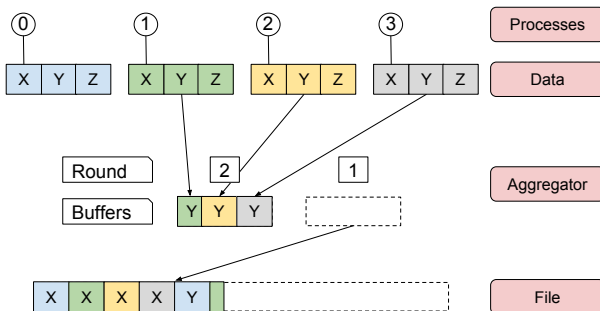
- ▶ Initialization: allocate buffers, create MPI windows, compute tuples {round, aggregator, buffer} for each process P
  - Let's say P1 is the aggregator
- ▶ P0, P1 and P2 put data in buffer 1 (round 1) of P1. P3 waits (fence)
- ▶ P1 writes buffer 1 in file and aggregates data from all the ranks in buffer 2
- ▶ 2<sup>nd</sup> round. P1 writes buffer 2 and aggregates data from P1, P2 and P3
- ▶ and so on...
- ▶ Limitations: MPI\_Comm\_split, one aggr./node at most





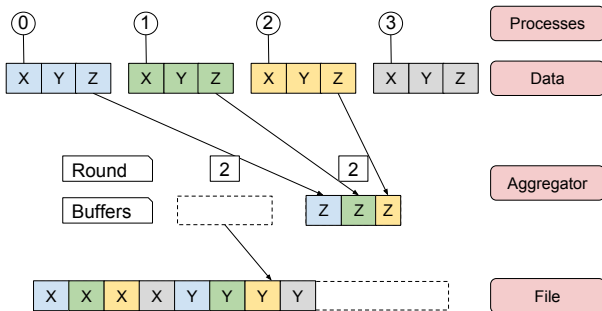
# Algorithm

- ▶ Initialization: allocate buffers, create MPI windows, compute tuples {round, aggregator, buffer} for each process P
  - Let's say P1 is the aggregator
- ▶ P0, P1 and P2 put data in buffer 1 (round 1) of P1. P3 waits (fence)
- ▶ P1 writes buffer 1 in file and aggregates data from all the ranks in buffer 2
- ▶ 2<sup>nd</sup> round. P1 writes buffer 2 and aggregates data from P1, P2 and P3
- ▶ and so on...
- ▶ Limitations: MPI\_Comm\_split, one aggr./node at most



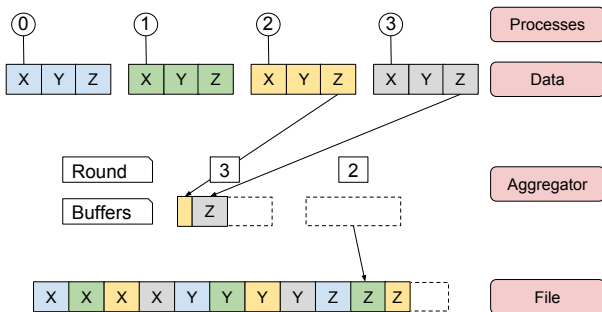
# Algorithm

- ▶ Initialization: allocate buffers, create MPI windows, compute tuples {round, aggregator, buffer} for each process P
  - Let's say P1 is the aggregator
- ▶ P0, P1 and P2 put data in buffer 1 (round 1) of P1. P3 waits (fence)
- ▶ P1 writes buffer 1 in file and aggregates data from all the ranks in buffer 2
- ▶ 2<sup>nd</sup> round. P1 writes buffer 2 and aggregates data from P1, P2 and P3
- ▶ and so on...
- ▶ Limitations: `MPI_Comm_split`, one aggr./node at most



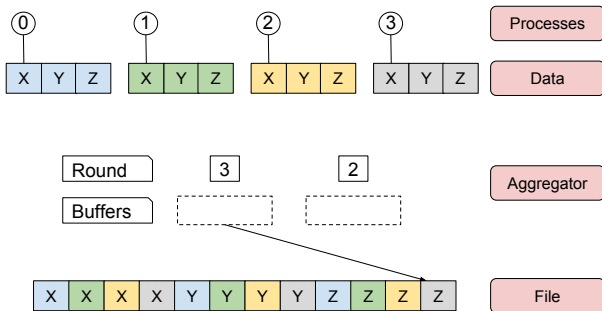
# Algorithm

- ▶ Initialization: allocate buffers, create MPI windows, compute tuples {round, aggregator, buffer} for each process P
  - Let's say P1 is the aggregator
- ▶ P0, P1 and P2 put data in buffer 1 (round 1) of P1. P3 waits (fence)
- ▶ P1 writes buffer 1 in file and aggregates data from all the ranks in buffer 2
- ▶ 2<sup>nd</sup> round. P1 writes buffer 2 and aggregates data from P1, P2 and P3
- ▶ and so on...
- ▶ Limitations: `MPI_Comm_split`, one aggr./node at most



# Algorithm

- ▶ Initialization: allocate buffers, create MPI windows, compute tuples {round, aggregator, buffer} for each process P
  - Let's say P1 is the aggregator
- ▶ P0, P1 and P2 put data in buffer 1 (round 1) of P1. P3 waits (fence)
- ▶ P1 writes buffer 1 in file and aggregates data from all the ranks in buffer 2
- ▶ 2<sup>nd</sup> round. P1 writes buffer 2 and aggregates data from P1, P2 and P3
- ▶ and so on...
- ▶ Limitations: MPI\_Comm\_split, one aggr./node at most

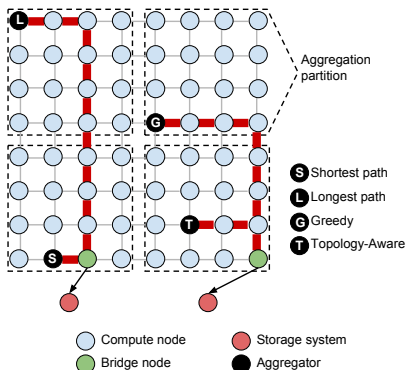


# Outline

- 1 Context
- 2 Approach
- 3 Evaluation**
- 4 Conclusion and Perspectives

## Micro-benchmark - Placement strategies

- ▶ Evaluation on Mira (BG/Q), 512 nodes, 16 ranks/node
  - ▶ Each rank sends a data buffer chosen randomly between 0 and 2 MB
  - ▶ Writes to /dev/null of the I/O node (aggregation and I/O phases only)
  - ▶ Aggregation settings: 16 aggregators, 16 MB buffer size
- 
- ▶ Four tested strategies
    - **Shortest path:** smallest distance to the I/O node
    - **Longest path:** longest distance to the I/O node
    - **Greedy:** lowest rank in partition (similar to the default MPICH strategy)
    - **Topology-aware**



## Micro-benchmark - Placement strategies

- ▶ Evaluation on Mira (BG/Q), 512 nodes, 16 ranks/node
- ▶ Each rank sends a data buffer chosen randomly between 0 and 2 MB
- ▶ Writes to /dev/null of the I/O node (aggregation and I/O phases only)
- ▶ Aggregation settings: 16 aggregators, 16 MB buffer size

Table: Impact of aggregators placement strategy

| Strategy       | I/O Bandwidth (MBps) | Aggr. Time/round (ms) |
|----------------|----------------------|-----------------------|
| Greedy         | 1927.45              | 421.33                |
| Longest path   | 2202.91              | 370.40                |
| Shortest path  | 2484.39              | 327.08                |
| Topology-Aware | 2638.40              | 310.46                |

- ▶ I/O bandwidth increased by 37% in comparison to the Greedy strategy and 6% over the Shortest Path approach

## HACC-IO

- ▶ I/O part of a large-scale cosmological application simulating the mass evolution of the universe with particle-mesh techniques
- ▶ Each process manage particles defined by 9 variables (38 bytes)
  - $XX, YY, ZZ, VX, VY, VZ, phi, pid$  and  $mask$
- ▶ One file per  $Pset$  (128 nodes) vs. one single shared file
- ▶ Aggregation settings: 16 aggregators per  $Pset$ , 16 MB buffer size (MPICH)
- ▶ Average and standard deviation on 10 runs

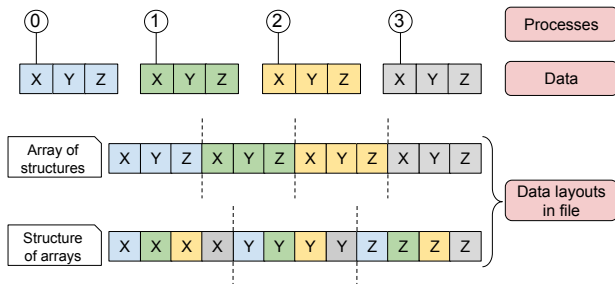
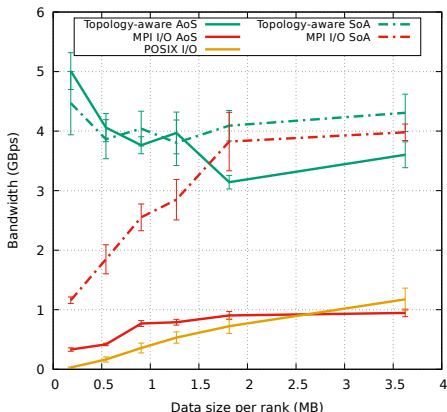


Figure: Data layouts in HACC-IO



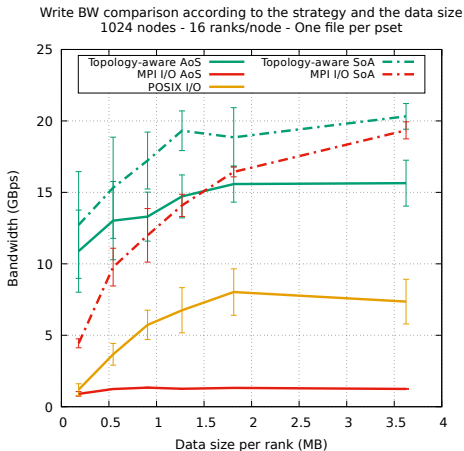
# HACC-IO - 1024 nodes - 16K ranks - Single shared file

Write BW comparison according to the strategy and the data size  
1024 nodes - 16 ranks/node - Single shared file

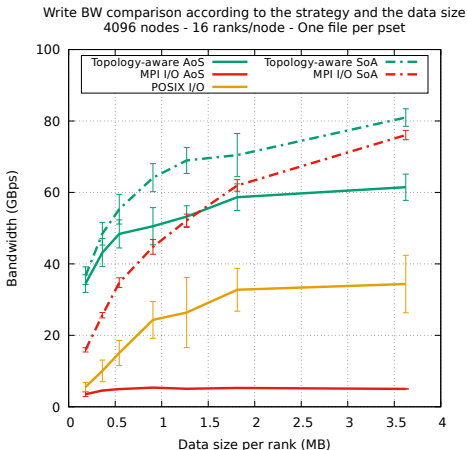


- ▶ Peak is estimated to 22.4 GBps (theoretical: 28.8 GBps)
- ▶ Our approach achieves higher performance than the default strategies
  - 5K particles (190 KB) and AoS data layout: **15×** faster than MPI I/O

# HACC-IO - 1024 nodes - 16K ranks - Sub-filing (One file per $P$ set)



- ▶ Sub-filing is an efficient approach for improved I/O performance
- ▶ Our topology-aware strategy achieves **90%** of the peak I/O bandwidth (22.4 GBps)
  - Significant improvement particularly for small messages

HACC-IO - 4096 nodes - 65K ranks - Sub-filing (One file per  $Pset$ )

- ▶ Peak is estimated to 89.6 GBps (theoretical: 115.2 GBps)
- ▶ **90%** of the peak I/O bandwidth achieved as on 1024 nodes
- ▶ Improved I/O performance for both AoS and SoA layouts and significant improvement on smaller messages for the SoA case (up to 43%)

# Outline

- 1 Context
- 2 Approach
- 3 Evaluation
- 4 Conclusion and Perspectives**

# Conclusion and Perspectives

## Conclusion

- ▶ Optimized two-phase I/O library incorporating
  - Topology-aware aggregator placement
  - Optimized data movement and buffering (double-buffering, one-sided communication, block size awareness)
- ▶ Very good performance at scale, outperforming standard approaches
- ▶ On the I/O part of a cosmological application, up to 12× improvement on 65K ranks
- ▶ Architecture characteristics are critical for performance at scale

## Next steps

- ▶ Take the routing policy into account
- ▶ Incorporate additional data models and layouts (2D, 3D-arrays)
- ▶ Hierarchical approach to tackle different tiers of storage

# Conclusion

## Acknowledgments

- ▶ Argonne Leadership Computing Facility at Argonne National Laboratory
- ▶ DOE Office of Science, ASCR
- ▶ NCSA-Inria-ANL-BSC-JSC-Riken Joint-Laboratory on Extreme Scale Computing
- ▶ European Union Seventh Framework Program

# Conclusion

Thank you for your attention!

[ftessier@anl.gov](mailto:ftessier@anl.gov)

## Micro-benchmark - #Aggr and buffer size

- ▶ Evaluation on Mira (BG/Q), 1024 nodes, 16 ranks/node
- ▶ Each rank writes 1 MB
- ▶ Write to /dev/null of the I/O node (performance of just aggregation and I/O phases)

**Table:** I/O Bandwidth (in MBps) achieved on a simple benchmark with a topology-aware aggregator placement while varying the number of aggregators and the buffer size.

| #Aggr/Pset | Buffer size |         |         |
|------------|-------------|---------|---------|
|            | 8 MB        | 16 MB   | 32 MB   |
| 8          | 7652.49     | 8848.28 | 9050.71 |
| 16         | 7318.15     | 8774.58 | 9331.84 |
| 32         | 6329.95     | 7797.12 | 8134.41 |