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

January 27, 2017

Data Movement at Scale

- Computer simulation: climate simulation, heart or brain modelling, cosmology, etc.
- Large needs in terms of I/O: high resolution, high fidelity

Table: Example of large simulations I/O coming from diverse papers

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**

- Growth of supercomputers to meet the performance needs but with increasing gaps

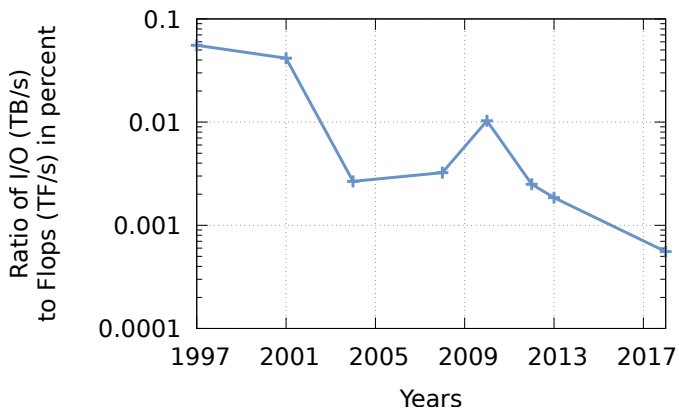
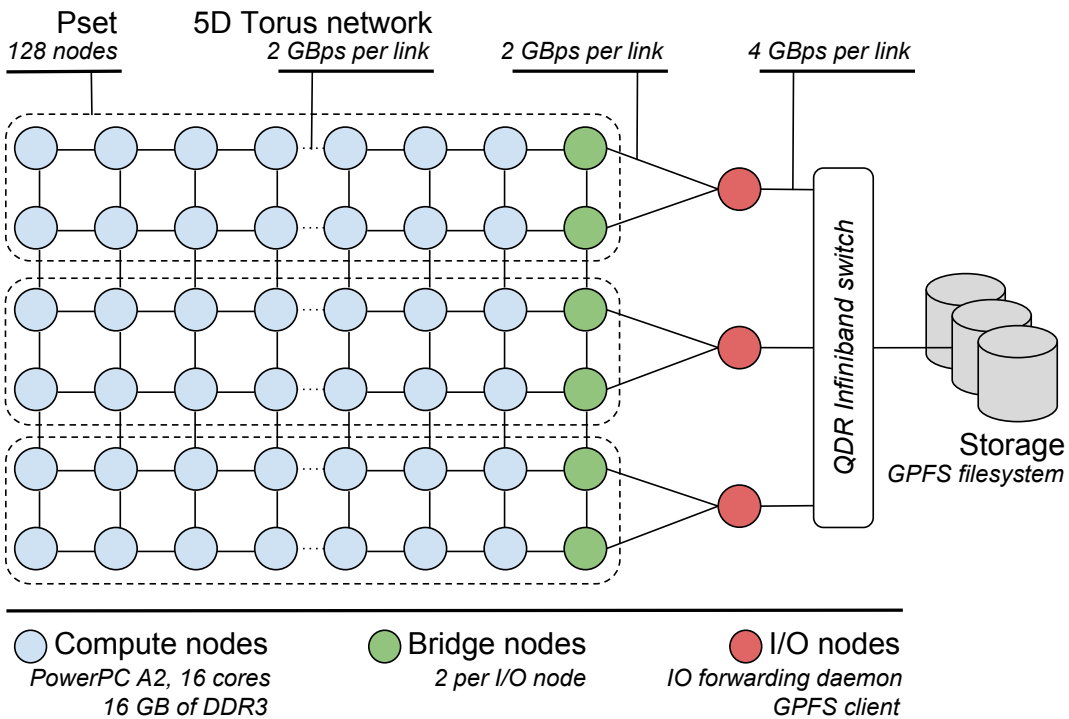


Figure: Ratio IOPS/FLOPS of the #1 Top 500 for the past 20 years. Computing capability has grown at a faster rate than the I/O performance of supercomputers.

Complex Architectures

- Complex network topologies tending to reduce the distance between the data and the storage
  - Multidimensional tori, dragonfly, ...
- Partitioning of the architecture to avoid 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

- Present in MPI I/O implementations like ROMIO
- Optimize collective I/O performance by reducing network contention and increasing I/O bandwidth
- Chose a subset of processes to aggregate data before writing it to the storage system

- Better for large messages (from experiments)
- No real efficient aggregator placement policy
- Informations about upcoming data movement could help

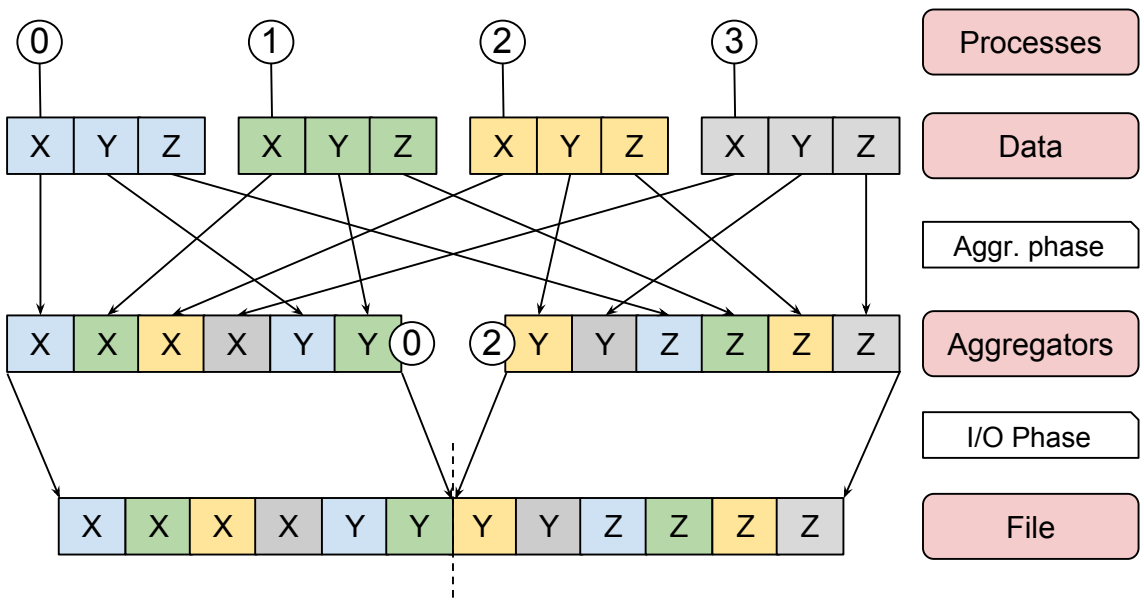


Figure: Two-phase I/O mechanism

Outline

Approach

- Relevant aggregator placement while taking into account:
  - The topology of the architecture
  - The data pattern
- Efficient implementation of the two-phase I/O scheme
  - I/O scheduling with the help of information about the upcomming readings and writings
  - Pipelining aggergation and I/O phase to optimize data movements
  - One-sided communications and non-blocking operations to reduce synchronizations

Aggregator Placement

- **Goal**: find a compromise between aggregation and I/O costs
- 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 (can be compared to a MPICH strategy)
  - **Topology-aware**

How to take the topology into account in aggregators mapping?

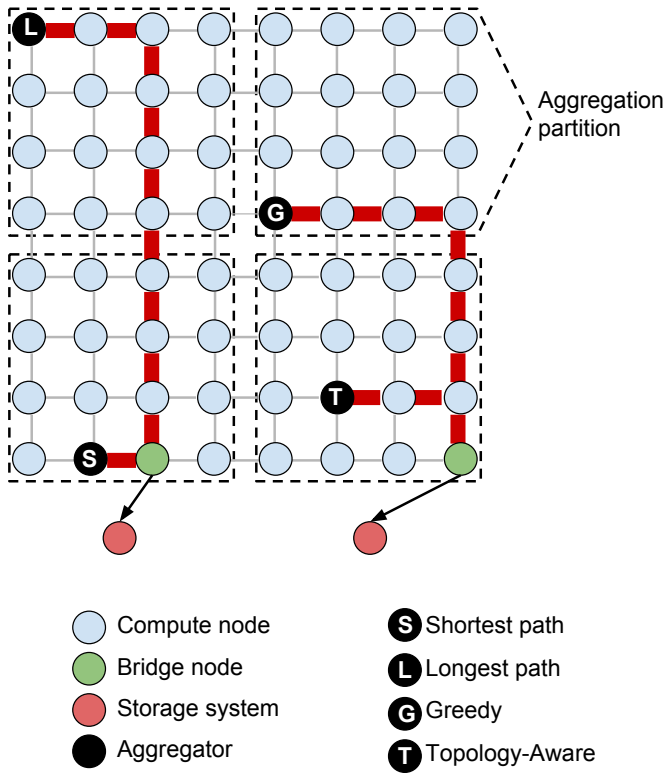
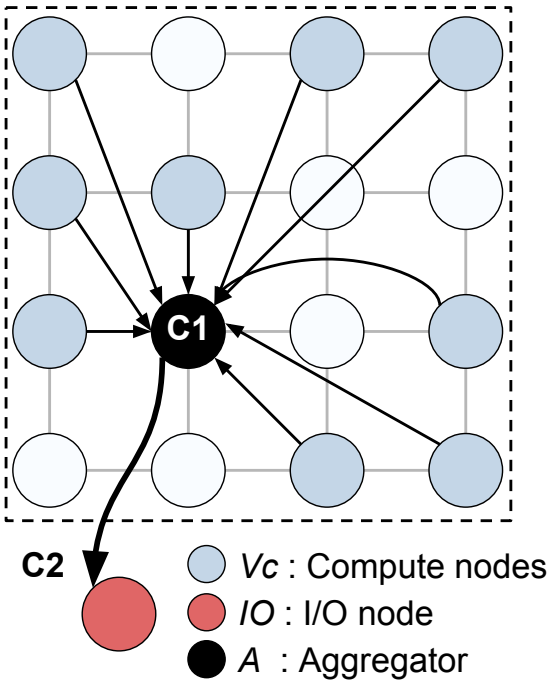


Figure: Data aggregation for I/O: simple partitioning and aggregator election on a grid.

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
- $B_{i \to j}$: The bandwidth from node i to node j.
- $C_1 = max\left(l \times d(i, A) + \frac{\omega(i,A)}{B_{i \to A}}\right), i \in V_C$
- $C_2 = l \times d(A, IO) + \frac{\omega(A,IO)}{|V_C| \times B_{A \to IO}}$



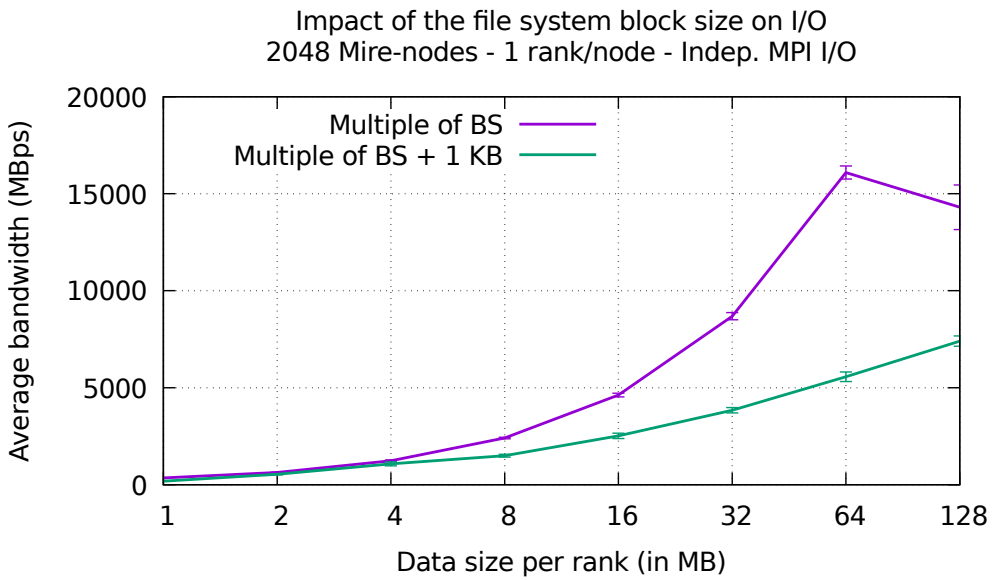
Objective function:

$$TopoAware(A) = min\,(C_1 + C_2)$$

- Computed by each process independently in $O(n), n = |V_C|$

Optimized Buffering

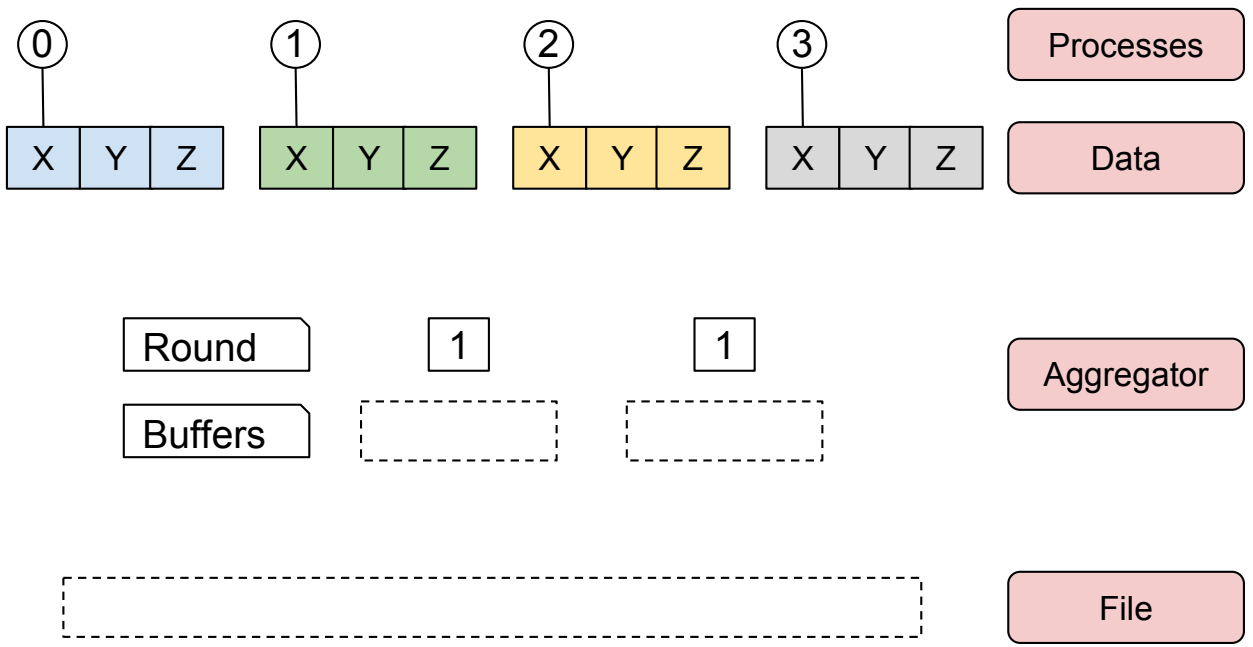
- Block of a filesystem: indivisible block of memory on disk requested for each I/O
- Lock contention avoided in our implementation (in MPI I/O as well)
- Simple benchmark with processes writing $n \times BS$ (purple) or $n \times BS + 1024$ (green)



- Two pipelined buffers per aggregator (communication overlaped)
  - Aggergation phase: RMA operations (one-sided communications)
  - I/O phase: non-blocking independent write

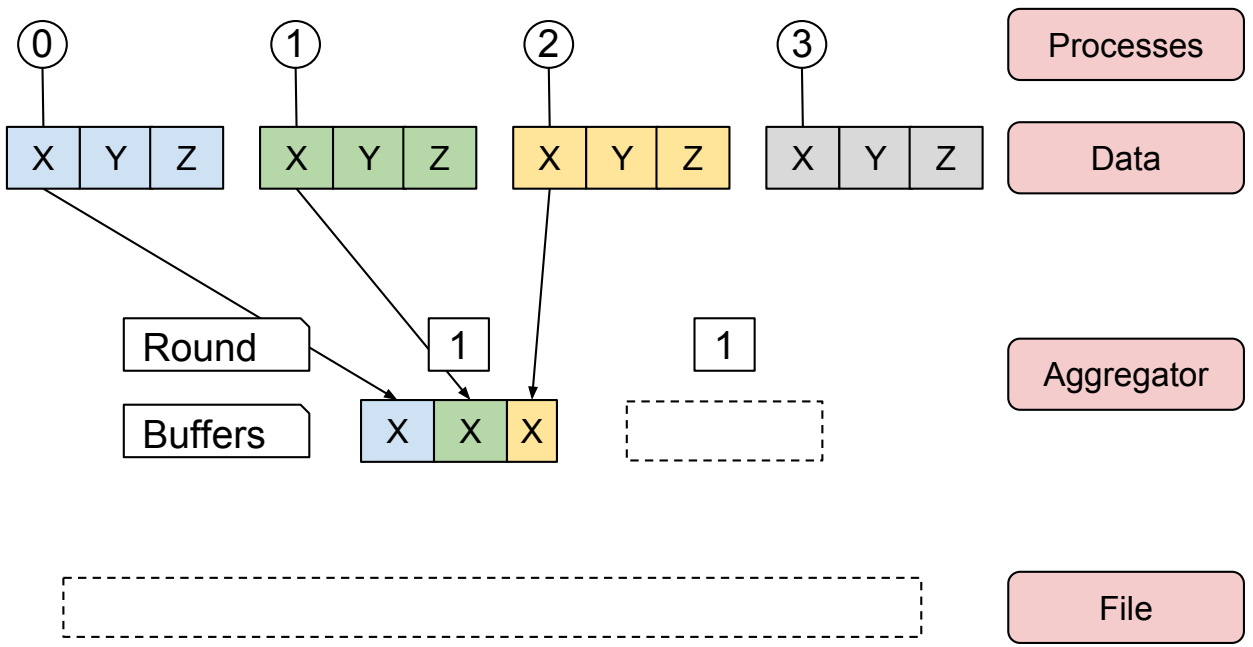
Algorithm

- 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
- 2nd 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



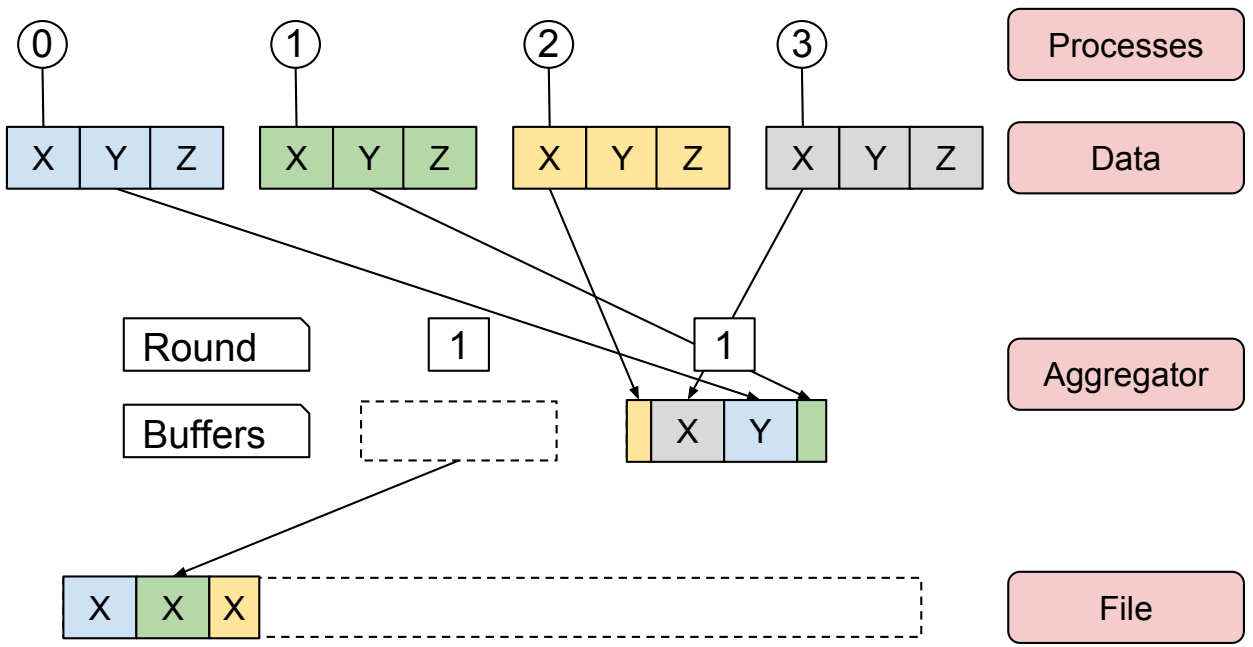
Algorithm

- 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
- 2nd 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



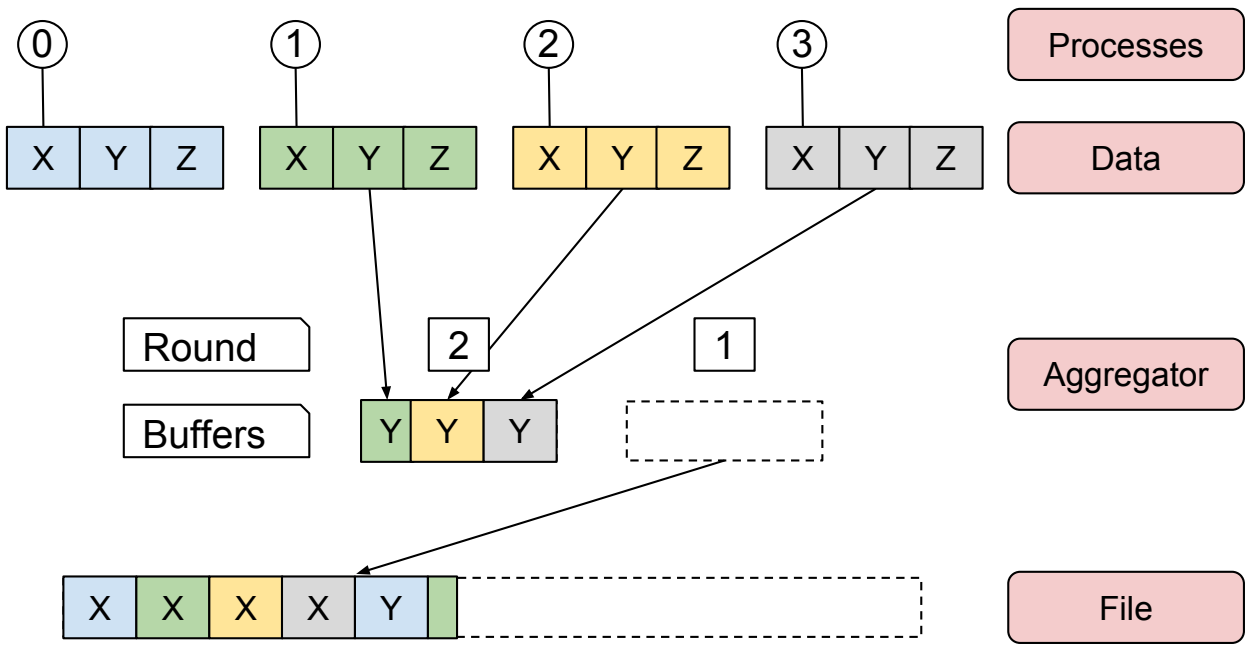
Algorithm

- 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
- 2nd 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



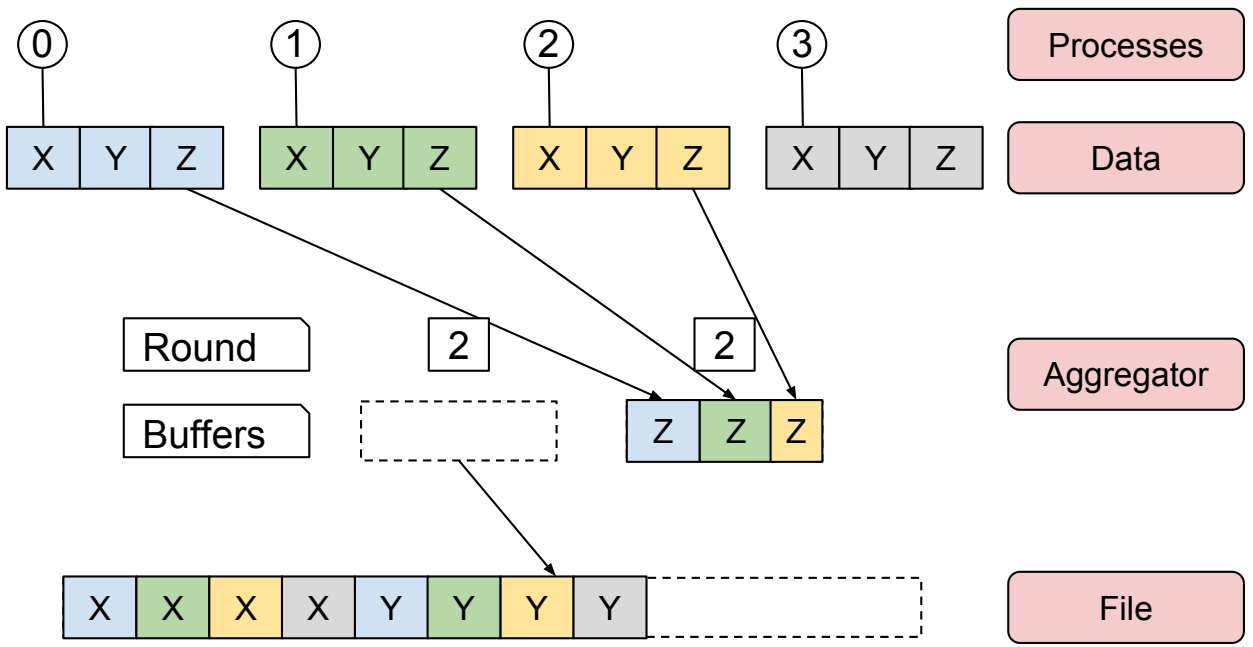
Algorithm

- 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
- 2nd 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



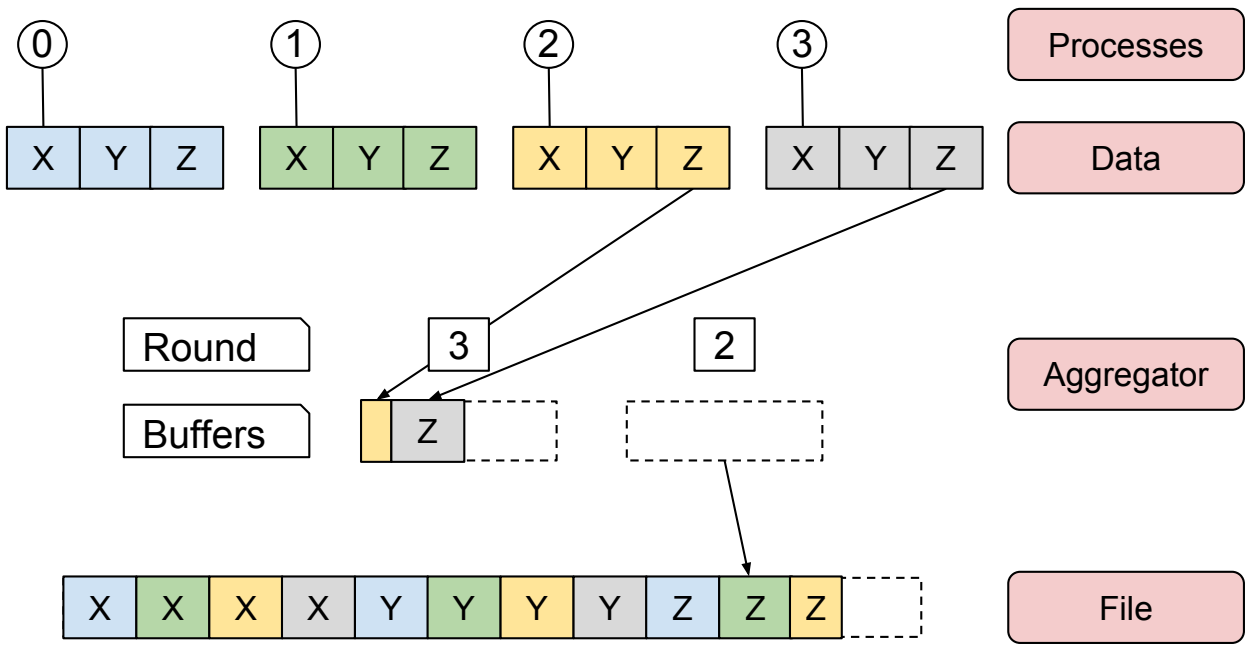
Algorithm

- 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
- 2nd 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



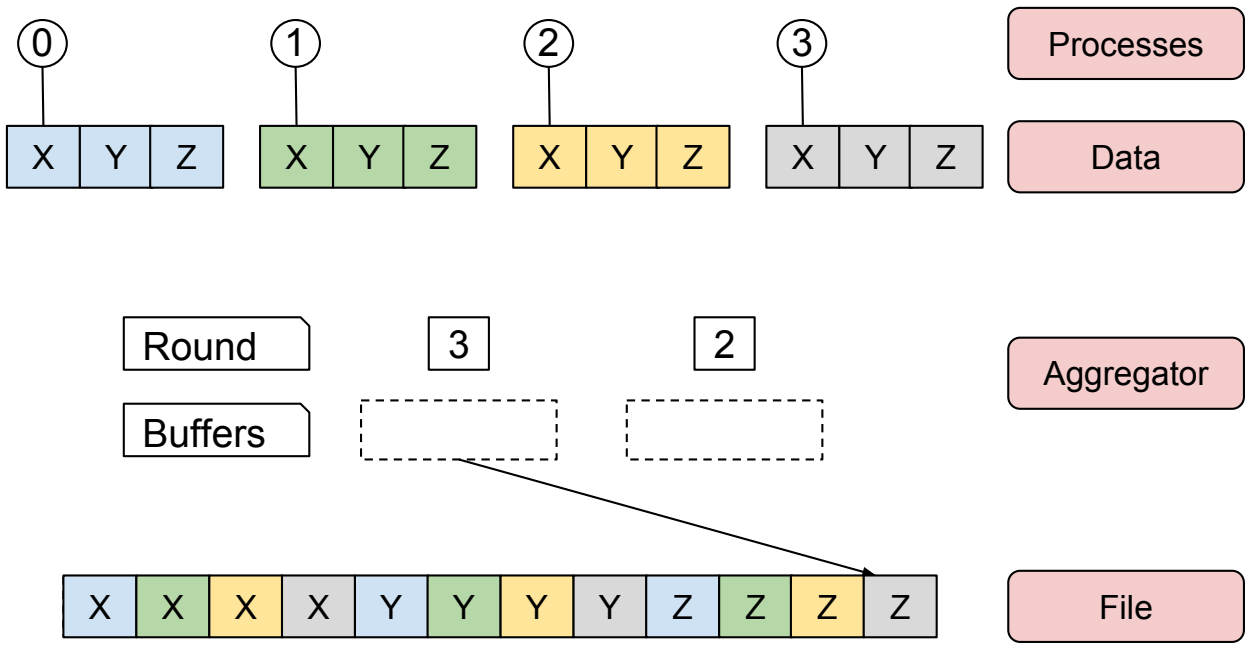
Algorithm

- 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
- 2nd 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
- 2nd 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



Micro-benchmark - Placement strategies

- Evaluation on Mira (BG/Q), 512 nodes, 16 ranks/node
- Each rank sends an amount of data distributed randomly between 0 and 2 MB
- Write to /dev/null of the I/O node (performance of just aggregation and I/O phases)
- Aggregation settings: 16 aggregators, 16 MB buffer size

Table: Impact of aggregators placement strategy

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

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 (*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)

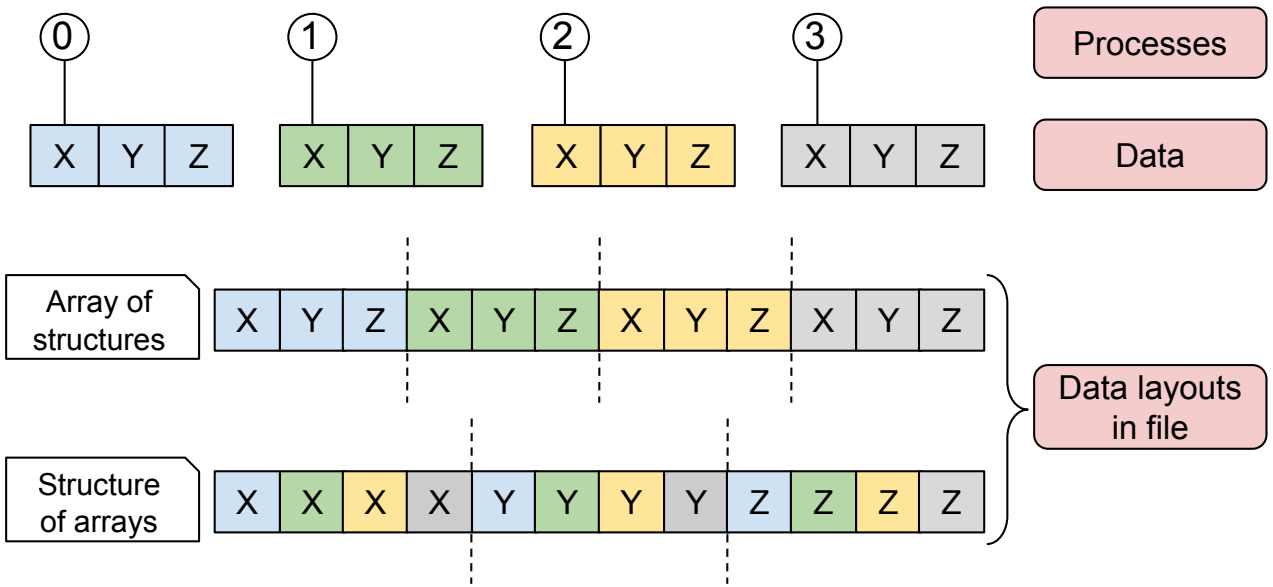
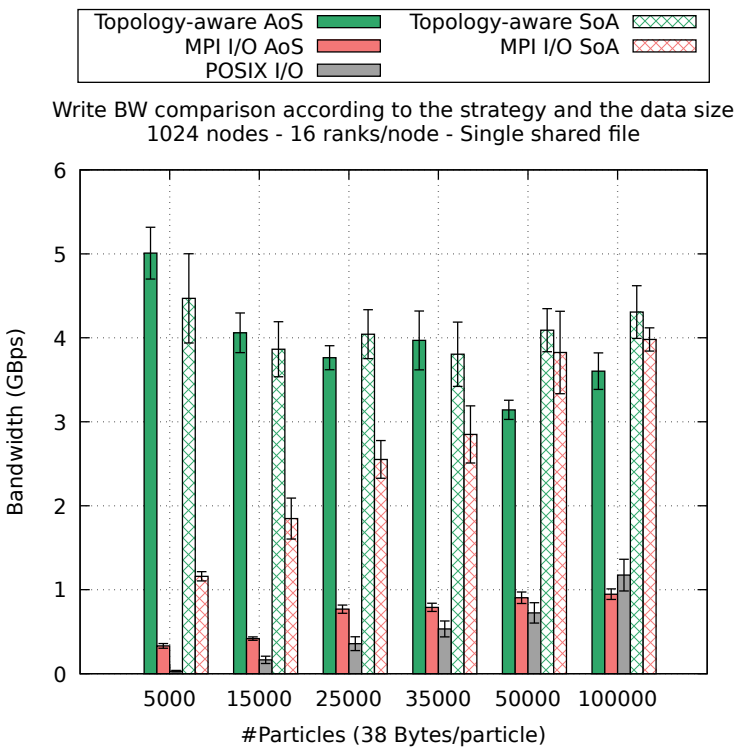


Figure: Data layouts implemented in HACC

HACC-IO

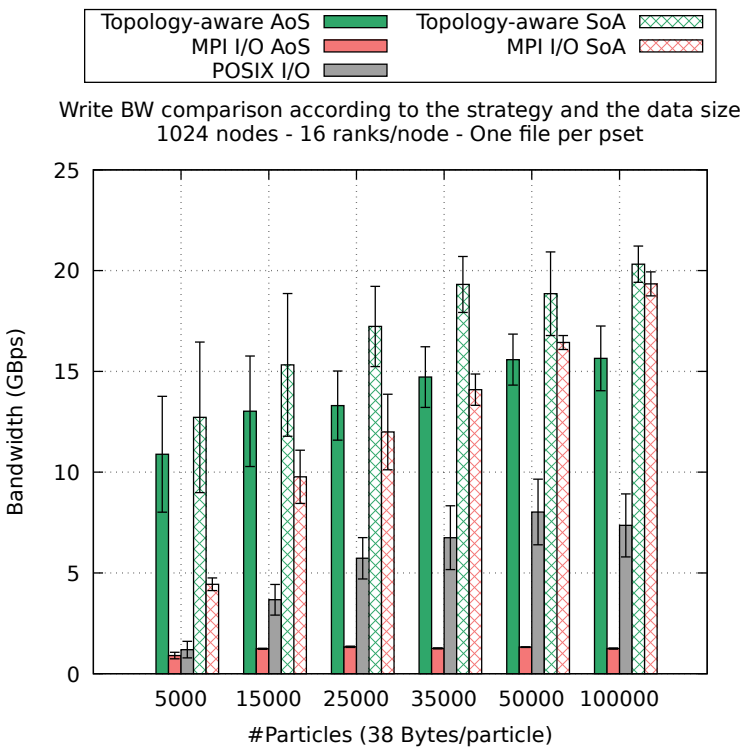
- I/O part of a large-scale cosmological application simulating the mass evolution of the universe with particle-mesh techniques
- Each process manage particles defined by 9 variables (*XX*, *YY*, *ZZ*, *VX*, *VY*, *VZ*, *phi*, *pid* and *mask*)
- One file per *Pset* (128 nodes) vs one single shared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



- Poor performance in general
- Peak estimated to 22.4 GBps (theoretical: 28.8 GBps)
- Our strategy works better than the standard ones
  - 5K particles and AoS data layout: 15× faster than MPI I/O
  - Very poor performance from MPI I/O on AoS

HACC-IO

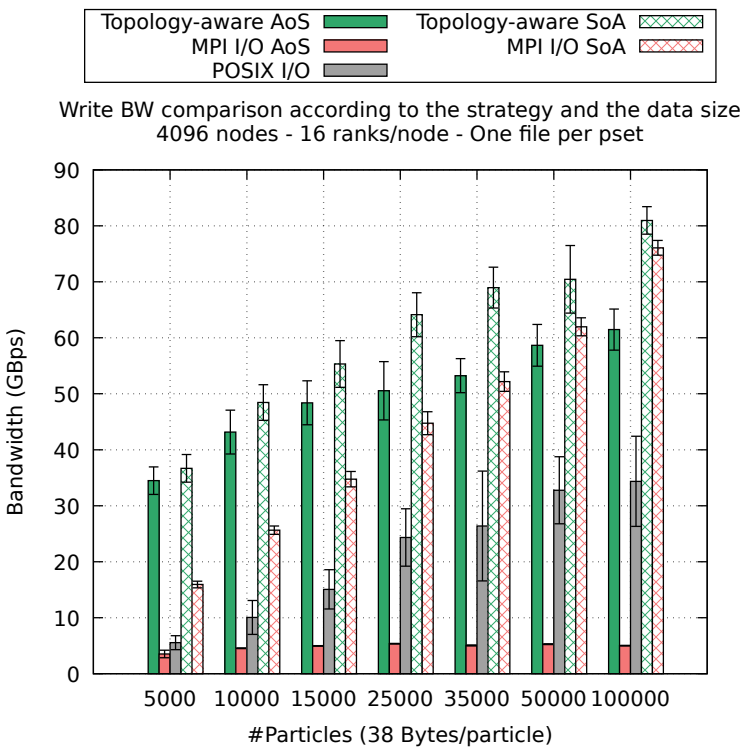
- I/O part of a large-scale cosmological application simulating the mass evolution of the universe with particle-mesh techniques
- Each process manage particles defined by 9 variables (*XX*, *YY*, *ZZ*, *VX*, *VY*, *VZ*, *phi*, *pid* and *mask*)
- One file per *Pset* (128 nodes) vs one single shared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



- Sub-filing is a efficient solution
- I/O bandwidth achieved close to the peak value
- Our topology-aware solution performs better
  - Particularly on small messages
  - Gap with MPI I/O descresing as the data size increases

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 (*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)



- Good scalability on 4K nodes
- Similar behavior as on 1024 nodes
- I/O bandwidth improved no matter the data layout and particularly on messages smaller than 2 MB

Conclusion and Perspectives

Conclusion

- I/O library based on the two-phase scheme developed to optimize data movements
  - Topology-aware aggregator placement
  - Optimized buffering (two pipelined buffers, one-sided communications, block size awareness)
- Very good performance at scale, outperforming standard approaches
- On the I/O part of a cosmological application, up to 15× improvement
- Need to fully exploit the architecture characteristics to perform at scale

Next steps

- Take the routing policy into account
- Larger variety of data patterns (2D, 3D-arrays)
- Hierarchical approach to tackle different tiers of storage

Conclusion

Thank you for your attention!

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