Optimizing Data Aggregation by Leveraging the Deep Memory Hierarchy on Large-scale Systems

François Tessier, Paul Gressier, Venkatram Vishwanath

Argonne National Laboratory, USA

Thursday 14th June, 2018

Argonne NATIONAL LABORATORY

Context

- Computational science simulation in scientific domains such as in materials, high energy physics, engineering, have large performance needs
  - In computation: the Human Brain Project, for instance, goes after at least 1 ExaFLOPS
  - In I/O: 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**

- New workloads with specific needs of data movement
  - Big data, machine learning, checkpointing, in-situ, co-located processes, ...
  - Multiple data access pattern (model, layout, data size, frequency)

Context

- Massively parallel supercomputers supplying an increasing processing capacity
  - The first 10 machines listed in the top500 ranking are able to provide more than 10 PFlops
  - Aurora, the first Exascale system in the US (ANL!), will likely feature millions of cores
- However, the memory per core or TFlop is decreasing...

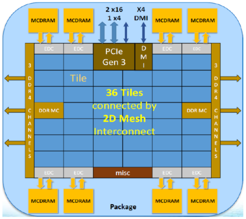
Criteria	2007	2017	Relative Inc./Dec.
Name, Location	BlueGene/L, USA	Sunway TaihuLight, China	N/A
Theoretical perf.	596 TFlops	125,436 TFlops	×210
#Cores	212,992	10,649,600	×50
Memory	73,728 GB	1,310,720 GB	×17.7
Memory/core	346 MB	123 MB	÷2.8
Memory/TFlop	124 MB	10 MB	÷12.4
I/O bw	128 GBps	288 GBps	×2.25
I/O bw/core	600 kBps	27 kBps	÷22.2
I/O bw/TFlop	214 MBps	2.30 MBps	÷93.0

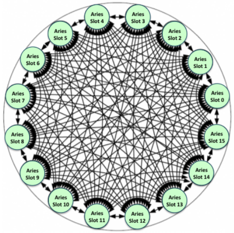
Table: Comparison between the first ranked supercomputer in 2007 and in 2017.

Growing importance of movements of data on current and upcoming large-scale systems

Context

- Mitigating this bottleneck from an hardware perspective leads to an increasing complexity and a diversity of the architectures
  - Deep memory and storage hierarchy
    - Blurring boundary between memory and storage
    - New tiers: MCDRAM, node-local storage, network-attached memory, NVRAM, Burst buffers
    - Various performance characteristics: latency, bandwidth, capacity
  - Complexity of interconnection network
    - Topologies: 5D-Torus, Dragon-fly, fat trees
    - Partitioning: network dedicated to I/O
    - Routing policies: static, adaptive





Credits: LLNL / LBNL

Data Aggregation

- Selects a subset of processes to aggregate data before writing it to the storage system
- Improves I/O performance by writing larger data chunks
- Reduces the number of clients concurrently communicating with the filesystem
- Available in MPI I/O implementations such as ROMIO

Limitations:

- Inefficient aggregator placement policy
- Cannot leverage the deep memory hierarchy
- Inability to use staging data

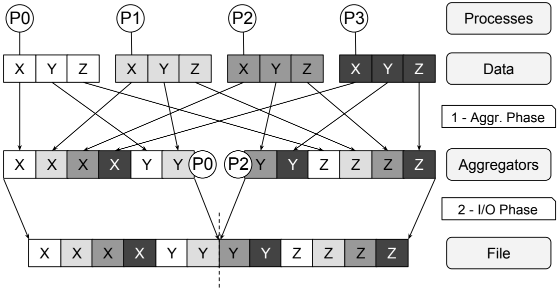
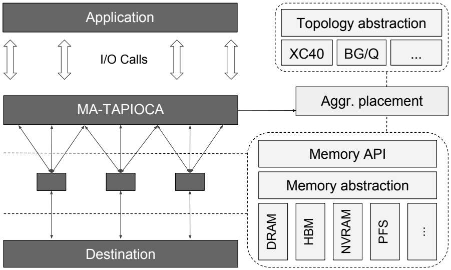


Figure: Two-phase I/O mechanism

MA-TAPIOCA - Memory-Aware TAPIOCA

- Based on TAPIOCA, a library implementing the two-phase I/O scheme for topology-aware data aggregation at scale[1] and featuring:
  - Optimized implementation of the two-phase I/O scheme (I/O scheduling)
  - Network interconnect abstraction for I/O performance portability
  - Aggregator placement taking into account the network interconnect and the data access pattern
- Augmented to include:
  - Abstraction including the topology and the deep memory hierarchy
  - Architecture-aware aggregators placement
  - Memory-aware data aggregation algorithm



[1] F. Tessier, V. Vishwanath, and E. Jeannot. "TAPIOCA: An I/O Library for Optimized Topology-Aware Data Aggregation on Large-Scale Supercomputers". In: *2017 IEEE International Conference on Cluster Computing (CLUSTER)*. Sept. 2017.

MA-TAPIOCA - Abstraction for Interconnect Topology

- Topology characteristics include:
  - Spatial coordinates
  - Distance between nodes: number of hops, routing policy
  - I/O nodes location, depending on the filesystem (bridge nodes, LNET, ...)
  - Network performance: latency, bandwidth
- Need to model some unknowns such as routing in the future

Listing 1: Function prototypes for network interconnect

```
int networkBandwidth            (int level);
int networkLatency              ();
int networkDistanceToIONode     (int rank, int IONode);
int networkDistanceBetweenRanks (int srcRank, int destRank);
```

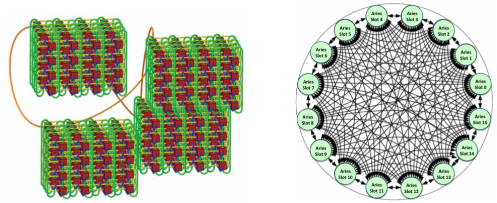
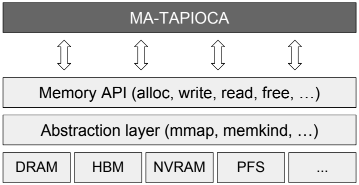


Figure: 5D-Torus on BG/Q and intra-chassis Dragonfly Network on Cray XC30 (Credit: LLNL / LBNL)

MA-TAPIOCA - Abstraction for Memory and Storage

- Memory management API
- Topology characteristics including spatial location, distance
- Performance characteristics: bandwidth, latency, capacity, persistency
- Scope of memory/storage tiers (PFS vs node-local SSD)
  - On those cases, a process has to be involved at destination

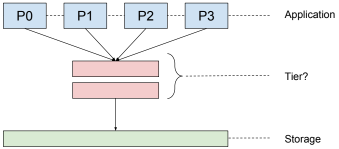

MA-TAPIOCA
Memory API (alloc, write, read, free, ...)
Abstraction layer (mmap, memkind, ...)
DRAM
HBM
NVRAM
PFS
...

Listing 2: Function prototypes for memory/storage data movements

```
buff_t* memAlloc        (mem_t mem, int buffSize, bool masterRank,
                         char* fileName, MPI_Comm comm);
void    memFree         (buff_t *buff);
int     memWrite        (buff_t *buff, void* srcBuffer,
                         int srcSize, int offset, int destRank);
int     memRead         (buff_t *buff, void* srcBuffer,
                         int srcSize, int offset, int srcRank);
void    memFlush        (buff_t *buff);
int     memLatency      (mem_t mem);
int     memBandwidth    (mem_t mem);
int     memCapacity     (mem_t mem);
int     memPersistency  (mem_t mem);
```

MA-TAPIOCA - Memory and topology aware aggregator placement

- Initial conditions: memory capacity for aggregation and destination.
- $\omega(u, v)$: Amount of data to move from memory bank u to v
- $d(u, v)$: distance between memory bank u and v
- l: The latency such as $l = max\left(l_{network}, l_{memory}\right)$;
- $B_{u \to v}$: The bandwidth from memory bank u to u, such as $B_{u \to v} = min\left(Bw_{network}, Bw_{memory}\right)$.
- A: Aggregator, T: Target



$$\mathbf{Cost_A} = \sum_{i \in V_C, i \neq A} \left(l \times d(i, A) + \frac{\omega(i, A)}{B_{i \to A}} \right)$$

$$\mathbf{Cost_T} = l \times d(A, T) + \frac{\omega(A, T)}{B_{A \to T}}$$

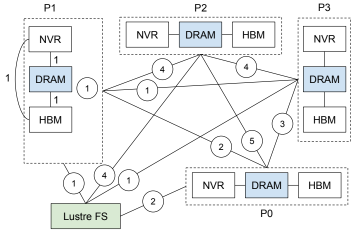
$$\mathbf{MemAware(A)} = min\left(Cost_A + Cost_T\right)$$

MA-TAPIOCA - Memory and topology aware aggregator placement

$$\mathbf{Cost_A} = \sum_{i \in V_C, i \neq A} \left(l \times d(i,A) + \frac{\omega(i,A)}{B_{i \to A}} \right)$$

$$\mathbf{Cost_T} = l \times d(A,T) + \frac{\omega(A,T)}{B_{A \to T}}$$

$$\mathbf{MemAware(A)} = min\left(Cost_A + Cost_T \right)$$



Value#	HBM	DRAM	NVR	Network
Latency (ms)	10	20	100	30
Bandwidth (GBps)	180	90	0.15	12.5
Capacity (GB)	16	192	128	N/A
Persistency	No	No	job lifetime	N/A

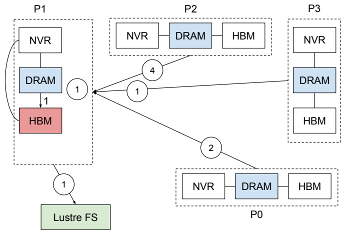
Table: Memory and network capabilities based on vendors information

MA-TAPIOCA - Memory and topology aware aggregator placement

$$\mathbf{Cost_A} = \sum_{i \in V_C, i \neq A} \left(l \times d(i,A) + \frac{\omega(i,A)}{B_{i \to A}} \right)$$

$$\mathbf{Cost_T} = l \times d(A,T) + \frac{\omega(A,T)}{B_{A \to T}}$$

$$\mathbf{MemAware(A)} = min\left(Cost_A + Cost_T \right)$$



Value#	HBM	DRAM	NVR	Network
Latency (ms)	10	20	100	30
Bandwidth (GBps)	180	90	0.15	12.5
Capacity (GB)	16	192	128	N/A
Persistency	No	No	job lifetime	N/A

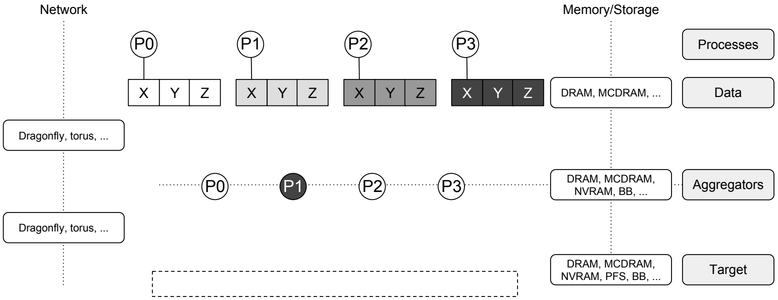
Table: Memory and network capabilities based on vendors information

P#	$\omega(i, A)$	HBM	DRAM	NVR
0	10	0.593	0.603	2.350
1	50	**0.470**	0.480	2.020
2	20	0.742	0.752	2.710
3	5	0.503	0.513	2.120

Table: For each process, MemAware(A)

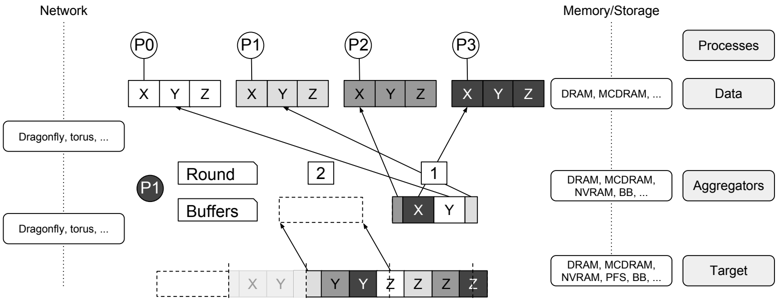
MA-TAPIOCA - Two-phase I/O algorithm

- Aggregator(s) selection according to the cost model described previously
- Overlapping of I/O and aggregation phases based on recent MPI features such as RMA and non-blocking operations
- The aggregation can be either defined by the user or chosen with our placement model
  - `MA-TAPIOCA_AGGTIER` environment variable: topology-aware placement only
  - `MA-TAPIOCA_PERSISTENCY` environment variable to set the level of persistency required in case of a memory and topology aware placement



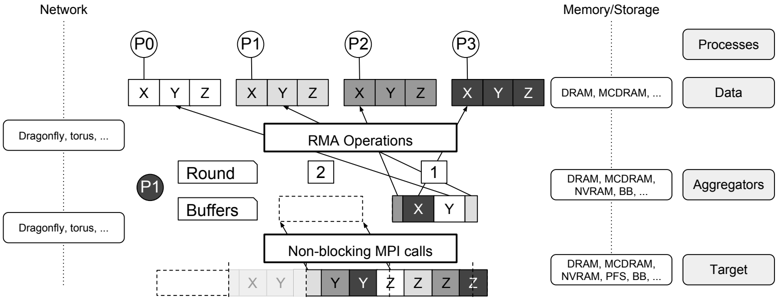
MA-TAPIOCA - Two-phase I/O algorithm

- Aggregator(s) selection according to the cost model described previously
- Overlapping of I/O and aggregation phases based on recent MPI features such as RMA and non-blocking operations
- The aggregation can be either defined by the user or chosen with our placement model
  - `MA-TAPIOCA_AGGTIER` environment variable: topology-aware placement only
  - `MA-TAPIOCA_PERSISTENCY` environment variable to set the level of persistency required in case of a memory and topology aware placement



MA-TAPIOCA - Two-phase I/O algorithm

- Aggregator(s) selection according to the cost model described previously
- Overlapping of I/O and aggregation phases based on recent MPI features such as RMA and non-blocking operations
- The aggregation can be either defined by the user or chosen with our placement model
  - `MA-TAPIOCA_AGGTIER` environment variable: topology-aware placement only
  - `MA-TAPIOCA_PERSISTENCY` environment variable to set the level of persistency required in case of a memory and topology aware placement



MA-TAPIOCA - Two-phase I/O algorithm

- Aggregator(s) selection according to the cost model described previously
- Overlapping of I/O and aggregation phases based on recent MPI features such as RMA and non-blocking operations
- The aggregation can be either defined by the user or chosen with our placement model
  - `MA-TAPIOCA_AGGTIER` environment variable: topology-aware placement only
  - `MA-TAPIOCA_PERSISTENCY` environment variable to set the level of persistency required in case of a memory and topology aware placement

Algorithm 1: Collective MPI I/O

```
$n \leftarrow 5$;
$x[n]$, $y[n]$, $z[n]$;
$ofst \leftarrow rank \times 3 \times n$;

MPI_File_read_at_all ($f$, $ofst$, $x$, $n$, $type$, $status$);
$ofst \leftarrow ofst + n$ ;

MPI_File_read_at_all ($f$, $ofst$, $y$, $n$, $type$, $status$);
$ofst \leftarrow ofst + n$;

MPI_File_read_at_all ($f$, $ofst$, $z$, $n$, $type$, $status$);
```

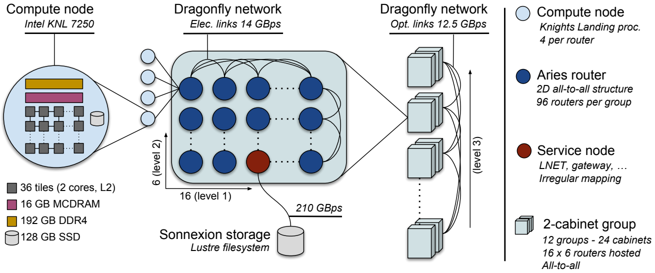
Algorithm 2: MA-TAPIOCA

```
$n \leftarrow 5$;
$x[n]$, $y[n]$, $z[n]$;
$ofst \leftarrow rank \times 3 \times n$;

for $i \leftarrow 0$, $i < 3$, $i \leftarrow i + 1$ do
    $count[i] \leftarrow n$;
    $type[i] \leftarrow$ sizeof ($type$);
    $ofst[i] \leftarrow ofst + i \times n$;

MA-TAPIOCA_Init ($count$, $type$, $ofst$, 3);

MA-TAPIOCA_Read ($f$, $ofst$, $x$, $n$, $type$, $status$);
$ofst \leftarrow ofst + n$ ;

MA-TAPIOCA_Read ($f$, $ofst$, $y$, $n$, $type$, $status$);
$ofst \leftarrow ofst + n$;

MA-TAPIOCA_Read ($f$, $ofst$, $z$, $n$, $type$, $status$);
```

Experiments - Test-beds

Theta

- Cray CX40 11.69 PFlops supercomputer at Argonne
  - 4,392 Intel KNL nodes with 64 cores
  - 16 GB of HBM, 192 GB of DRAM and 128 GB on-node SSD
- 10 PB parallel file system managed by Lustre
- Cray Aries dragonfly network interconnect



Cooley

- Intel Haswell-based visualization and analysis cluster at Argonne
  - 126 nodes with 12 cores and a NVIDIA Tesla K80
  - 384 GB of DRAM and a local hard drive (345 GB)
- 27 PB of storage managed by GPFS
- FDR Infiniband interconnect

Experiments - S3D-IO

S3D-IO

- I/O kernel of direct numerical simulation code in the field of computational fluid dynamics focusing on turbulence-chemistry interactions in combustion.
- 3D domain decomposition
- The state of each element is stored in an array of structure data layout
- The files as output are used for checkpointing and data analysis

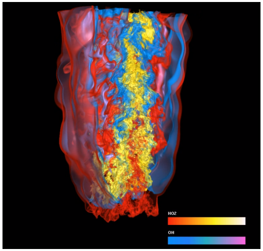


Figure: Credits: C.S. Yoo et Al., Ulsan NIST, Republic of Korea

Experimental setup

- Theta, a 11 PFlops **Cray XC40** supercomputer with a Lustre filesystem
  - Single shared file collectively written every *n* timesteps, stripped among OST.
  - Available tiers of memory: DRAM, HBM, on-node SSD
  - 96 aggregators for 256 nodes and 384 for 1024 nodes for both MPI-IO and MA-TAPIOCA
  - Lustre: 48 OST, 16MB stripe size, 4 aggr. per OST, 16MB buffer size
- Average and standard deviation on 10 runs

S3D-IO on Cray XC40 + Lustre

- Typical use-case with 134 and 537 millions grid points respectively distributed on 256 and 1024 nodes on Theta (16 ranks per node)
- Aggregation performed on HBM with MA-TAPIOCA
- I/O bandwidth increased by a factor of **3x** on 1024 nodes.

Table: Maximum write bandwidth (GBps).

	Points	**Size**	**256 nodes**	**1024 nodes**
MPI-IO	134M	160 GB	3.02 GBps	4.42 GBps
MA-TAPIOCA	537M	640 GB	4.86 GBps	13.75 GBps
Perf. Improvement	N/A	N/A	+60.93%	+210.91%

- Experiments on 256 nodes (134 millions grid points) while artificially reducing the memory capacity.
- The capacity requirement not being fulfilled, our placement algorithm selects another aggregation layer (gray boxes)

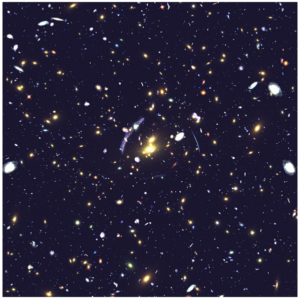
Table: Maximum write bandwidth (GBps).

Run	**HBM**	**DDR**	**NVRAM**	**Bandwidth**	**Std dev.**
1	16 GB	192 GB	128 GB	4.86 GBps	0.39 GBps
2	↓ 32 MB	192 GB	128 GB	4.90 GBps	0.43 GBps
3	↓ 32 MB	↓ 32 MB	128 GB	2.98 GBps	0.15 GBps

Experiments - HACC-IO

HACC-IO

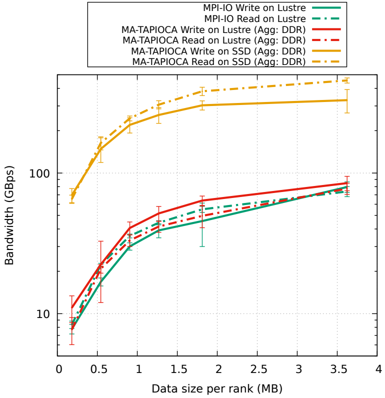
- I/O part of a large-scale cosmological application simulating the mass evolution of the universe with particle-mesh techniques
- Each process manages particles defined by 9 variables (38 bytes)
  - *XX, YY, ZZ, VX, VY, VZ, phi, pid* and *mask*
- Checkpointing files with data in an array of structure data layout



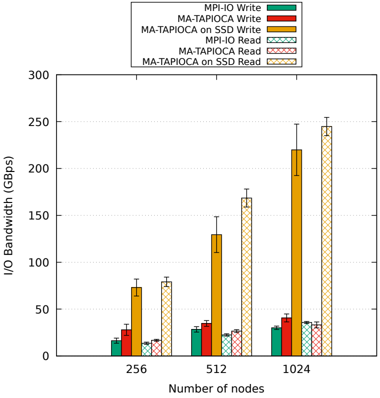
Experimental setup

- Theta, a 11 PFlops **Cray XC40** supercomputer with a Lustre filesystem
  - Available tiers of memory: DRAM, HBM, on-node SSD
  - Lustre: 48 OST, 16MB stripe size, 4 aggr. per OST, 16MB buffer size
- Cooley, an **Haswell-based** visualization and analysis cluster with GPFS
  - Available tiers of memory: DRAM, on-node HDD
- Average and standard deviation on 10 runs

HACC-IO on Cray XC40 + Lustre



(a) One file per node on 1024 nodes while varying the data size per rank.



(b) One file per node, 1MB/rank, while varying the number of nodes.

- Experiments on 1024 nodes on Theta
- Aggregation layer set with the `MA-TAPIOCA_AGGTIER` environment variable
- Regardless of the subfiling granularity, MA-TAPIOCA can use the local SSD as a shared file destination (*mmap* + `MPI_Win`)

HACC-IO on Cray XC40 + Lustre

- Experiments on 1024 nodes on Theta, one file per node
- Comparison between aggregation on DRAM and HBM when writing on the local SSD
- I/O performance achieved comparable
- Predicted by our model

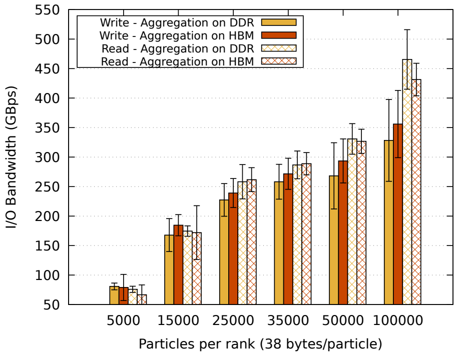


Figure: One file per node written on the local SSD. Aggregation on DRAM and HBM.

HACC-IO on Cray XC40 + Lustre

- Typical workflow that can be seamlessly implemented with MA-TAPIOCA
- Experiments on 256 nodes on Theta
- Write time counter-balanced by the read time from the local storage
- Total I/O time reduced by more than **26%**

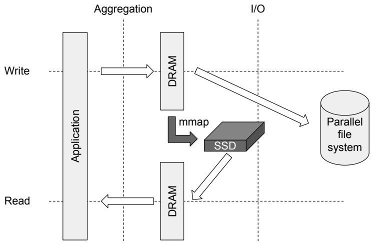


Table: Max. Write and Read bandwidth (GBps) and total I/O time achieved with and without aggregation on SSD

	Agg. Tier	Write	Read	I/O time
MA-TAPIOCA	DDR	47.50	38.92	693.88 ms
MPI-IO	DDR	32.95	37.74	843.73 ms
MA-TAPIOCA	SSD	26.88	227.22	617.46 ms
Variation		-36.10%	+446.94%	-26.82%

HACC-IO on Cooley + GPFS

- Code and performance portability thanks to our abstraction layer
- Experiments on 64 nodes on Cooley (Haswell-based cluster)
- Same application code, same optimization algorithm using our memory and network interconnect abstraction
- Total I/O time reduced by **12%**

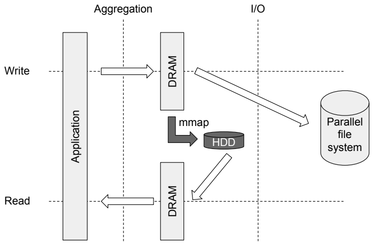


Table: Max. Write and Read bandwidth (GBps) and total I/O time achieved with and without aggregation on local HDD

	Agg. Tier	Write	Read	I/O Time
MA-TAPIOCA	DDR	6.60	38.80	123.41 ms
MPI-IO	DDR	6.02	17.46	155.40 ms
MA-TAPIOCA	HDD	5.97	35.86	135.86 ms
Variation		-0.83%	+105.38%	-12.57%

Conclusion and Future Work

- MA-TAPIOCA, a data aggregation library able to take advantage of the network interconnect and the deep memory hierarchy for improved performance
  - Architecture abstraction making possible to perform data aggregation on any type of memory or storage
  - Memory and topology aware aggregators placement
  - Efficient data aggregation algorithm
- Good performance at scale, outperforming MPI I/O
  - On a typical workflow, up to **26%** improvement on a Cray XC40 supercomputer with Lustre and up to **12%** on a visualization cluster
- Code and performance portability on large-scale supercomputers
  - Same application code running on various platforms
  - Same optimization algorithms using our interconnect abstraction

Future Work

- As the memory hierarchy tends to be deeper and deeper, multi-level data aggregation is of interest
- Intervene at a lower level to capture any kind of data types
- Transfer to widely used I/O libraries

Conclusion

Acknowledgments

- Argonne Leadership Computing Facility at Argonne National Laboratory
- DOE Office of Science, ASCR
- Proactive Data Containers (PDC) project

Conclusion

Thank you for your attention!

ftessier@anl.gov