A Comprehensive Study of In-Memory Computing on Large HPC Systems

Dan Huang, Zhenlu Qin, Qing Liu, Norbert Podhorszki, Scott Klasky

Sun Yat-sen University

New Jersey Institute of Technology

Oak Ridge National Lab

Why need in-memory computing in HPC

+ Compute is evolving from peta-scale to exa-scale era.

+ In particular compute is powered by customized GPUs.

- Persistent storage I/O has been a performance bottleneck on data-intensive HPC simulations and analytics for last decades.

- SSD based burst buffer can alleviate I/O bottleneck, can not solve it.

- Drive HPC communities to look for more efficient solutions.
- In-memory computing aims to address this challenge at memory layer.

Two models of HPC in-memory computing

- In-transit: Simulation data is staged at dedicated staging area.
  + Simulation can run asynchronously.
  - Extra data movement and data staging area.

  E.g. DataSpaces, DIMES, Flexpath and Decaf.
- In-situ: Analytics can directly access the simulation memory.
  + Limited or no data copy.
  - Simulation is slowed down due to time-sharing.

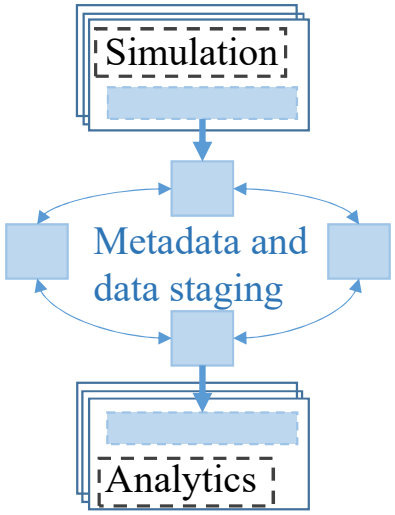
  E.g. SENSEI for HPC visualization.
- Lack of complete evaluations and understanding of in-memory computing on large HPC systems.

Contributions

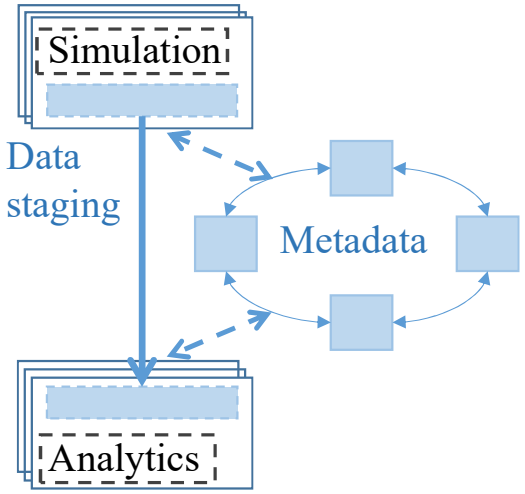
- One of the most comprehensive studies on HPC in-memory computing.
  - The end-to-end performance of scientific workflow.
  - Software usability, portability and robustness.
  - In-depth qualitative and quantitative analyses of the behavior of in-memory computing.
  - Summarize a number of key findings shedding a light on the weaknesses and possible areas for future research.
  - Show the insights of how to tune the performance of in-memory libraries on the two distinct HPC systems

Background

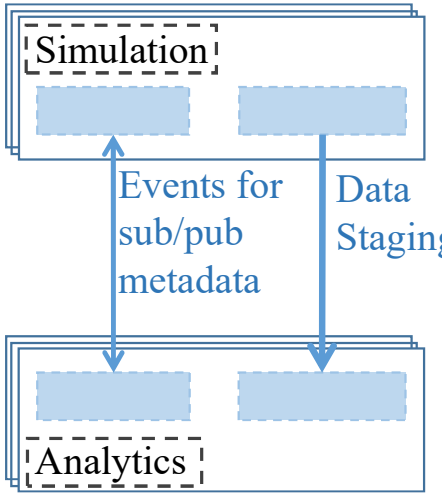
- HPC in-memory computing frameworks:



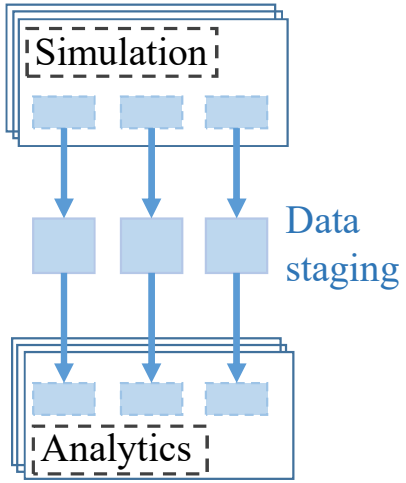
DataSpaces, Rutgers University



DIMES, Rutgers University



Flexpath, GaTech/ORNL



Decaf, ANL

Comparison of usability

- Usability matters for its broad adoption due to need to be in the hands of domain scientists.
- The engineering efforts using existing frameworks can be substantial:
  - Depict global data via configuration file (XML).
  - Almost access servers via either the communication APIs or third-party IO interfaces e.g. ADIOS.
  - Decaf adopts python or C++ to wrap components into one MPI communicator.
  - Configurations for build options.

Lines of code for configuration and API invocation.

Category	LOC	Functionality
DataSpace and DIMES (ADIOS)		
Build options	13	Enable RDMA, socket and etc.
Runtime config.	8	Define staging area: dimensions, size, offset and etc.
ADIOS XML config.	18	Data description in ADIOS: dimensions, size, offset and etc.
ADIOS data staging API	30	Server and client init, put/get data, and finalize
DataSpace and DIMES (native)		
Build options	13	Enable RDMA, socket and etc.
Runtime config.	8	Define staging area: dimensions, size, offset and etc.
Data staging API	81	Server and client init, lock/unlock, put/get data, and finalize
Flexpath		
Build options	5	RDMA API options, compiler and flags.
ADIOS XML config.	18	Data description in ADIOS: dimensions, size, offset and etc.
Data staging API	30	Init, put/get data and finalize
Decaf		
Build options	8	Enable transport layers, e.g. MPI
Bootstrap script	21	Define and link producer, consumer and staging processes
Data staging API	32	Init, dynamical load libs, data transformation, staging and finalize

Comparison of usability

- Usability matters for its broad adoption due to need to be in the hands of domain scientists.
- The engineering efforts using existing frameworks can be substantial:
  - Depict global data via configuration file (XML).
  - Almost access servers via either the communication APIs or third-party IO interfaces e.g. ADIOS.
  - Decaf adopts python or C++ to wrap components into one MPI communicator.
  - Configurations for build options.

Finding 1.In terms of usability, in-memory libraries are still far from being plug-and-play for domain scientists, and most of them require substantial support from HPC administrators or library developers, e.g. choosing the optimal build options and runtime I/O configurations

Testbed configurations

Build and runtime configurations.

Method	Version	Build options	Runtime configurations
DataSpaces/ADIOS and DIMES/ADIOS	DataSpaces 1.7.2, ADIOS 1.13	–with-dataspaces, –with-dimes, –with-mxml, –with-flexpath, –enable-dimes, –with-dimes-rdma-buffer-size=1024, –enable-drc	lock_type=2, hash_version=2, max_versions=1
DataSpaces/native and DIMES/native	DataSpaces 1.7.2, ADIOS 1.13	–enable-dimes, –enable-drc, –with-dimes-rdma-buffer-size=2048	lock_type=2, hash_version=2, max_versions=1
MPI-IO/ADIOS	ADIOS 1.13	–with-mxml	lfs setstripe –stripe-size 1m –stripe-count -1, ADIOS XML: stats=off
Flexpath/ADIOS	ADIOS 1.13, EVPath for ADIOS 1.13	–with-flexpath	CMTransport=nnti, ADIOS XML: queue_size=1
Decaf	version as of 06/20/2018	transport_mpi=on, build_bredala=on, build_manala=on	prod_dflow_redist=‘count’, dflow_con_redist=‘count’

Workflow configurations

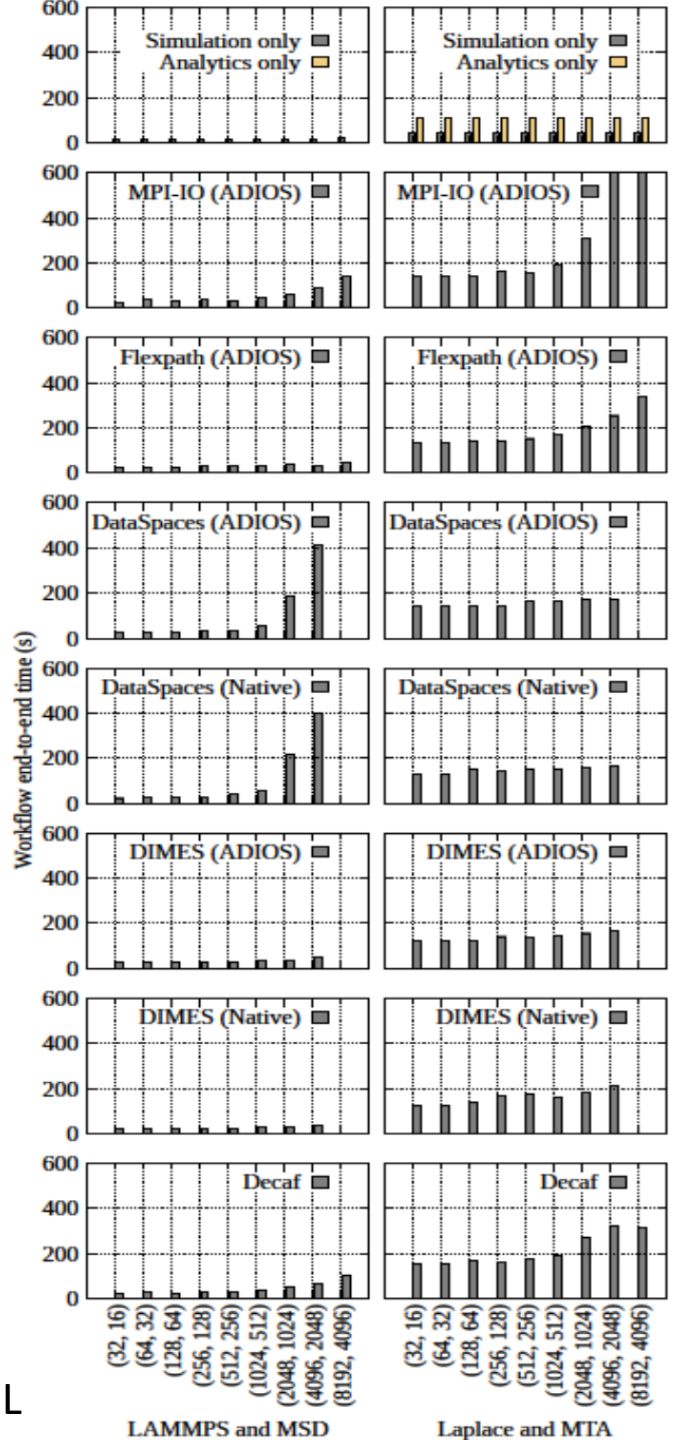
Workflow description. Note that $nprocs$ is the number of MPI processors used in the simulation.

Workflow name	**Simulation**	**Analytics**	**Output data**
LAMMPS	LAMMPS (version as of 08/22/2018), a molecular dynamics simulator	mean squared displacement (MSD)	$5 \times nprocs \times 512000$ double-precision data
Laplace	Solving Laplace's equation in a rectangle region	moment turbulence data analysis (MTA)	$4096 \times nprocs \times 4096$ double-precision data
Synthetic	A parallel benchmark that outputs data to the staging servers	A reader that retrieves data from the staging servers	Configurable

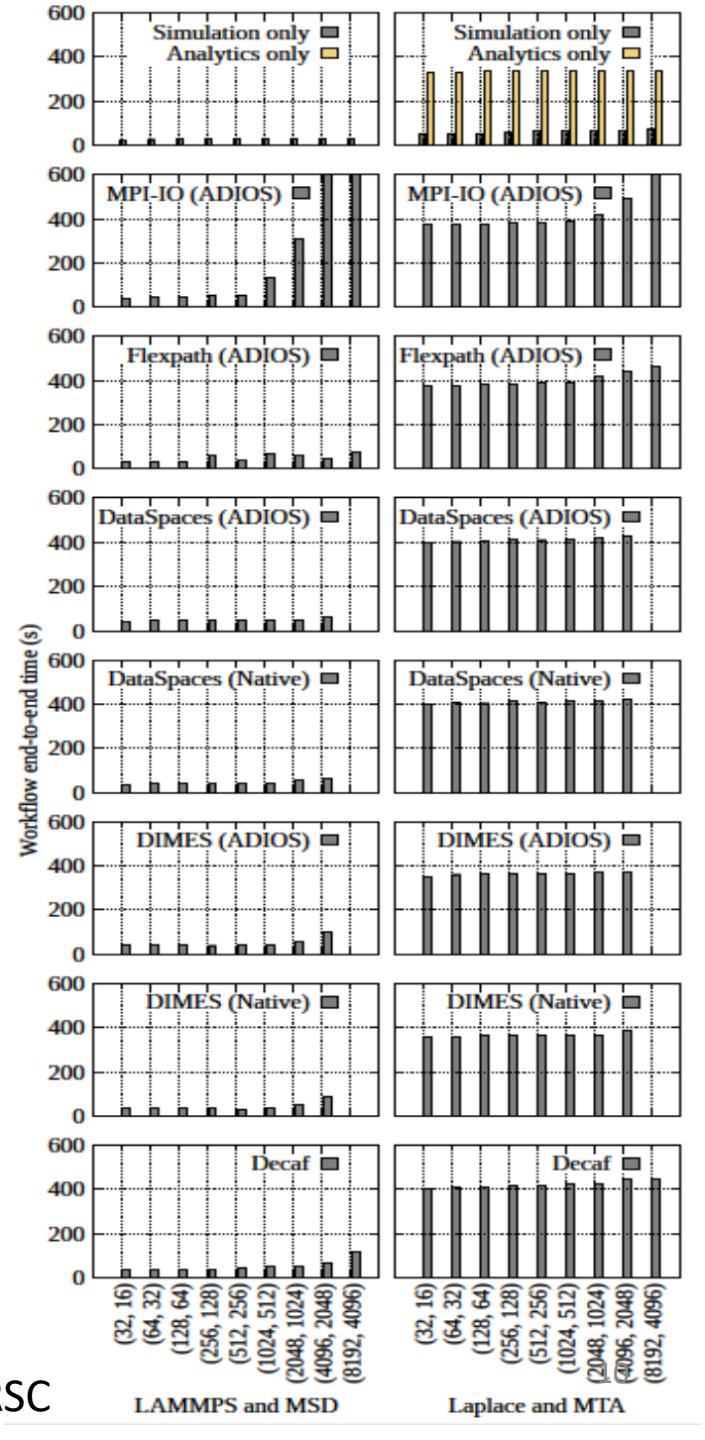
Weak scaling results

Observations:

1. Overall, in-memory libraries show better performance than MPI-IO, except DataSpaces on Titan.
2. Fixed number of Lustre OSTs and MDS limits the scalability of MPI-IO.
3. The scalability of DataSpaces is worse than others due to the mismatch of global data decomposition and layout.
4. DataSpaces and DIMES fail to serve workflows with scale of (8192, 4096).

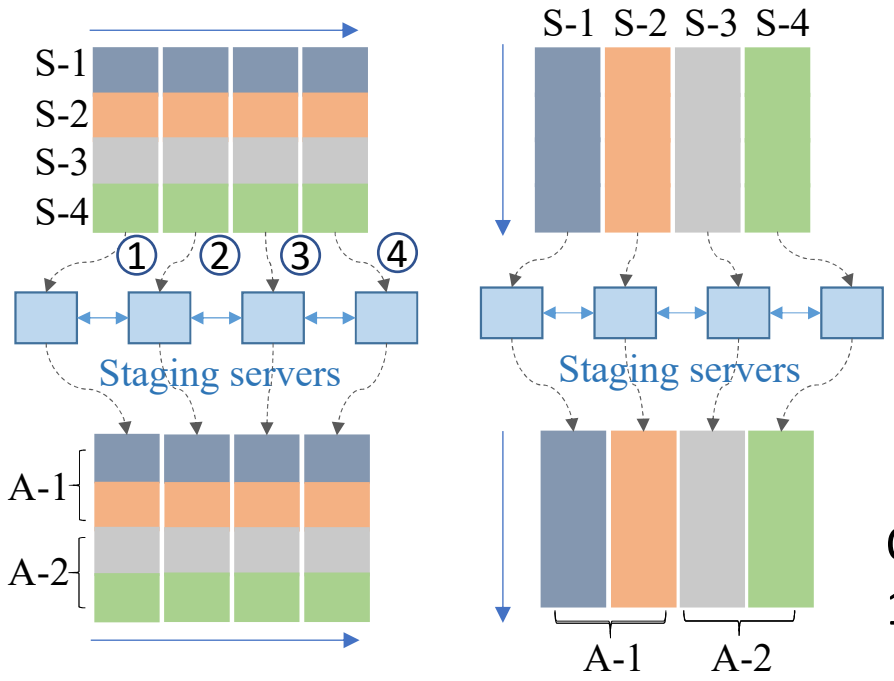


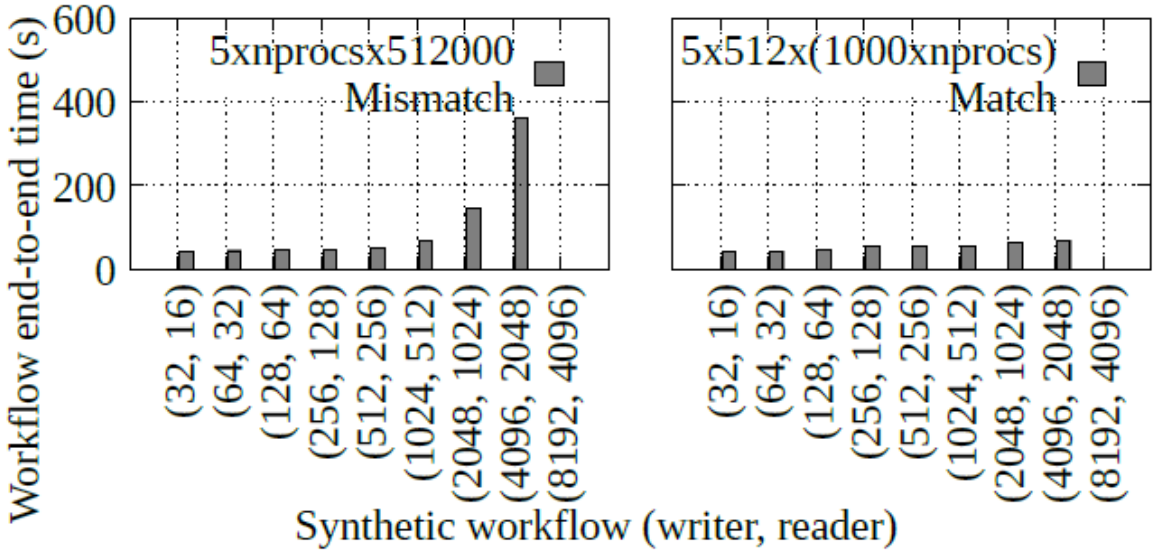
Titan, ORNL



Cori, NERSC

Global data decomposition

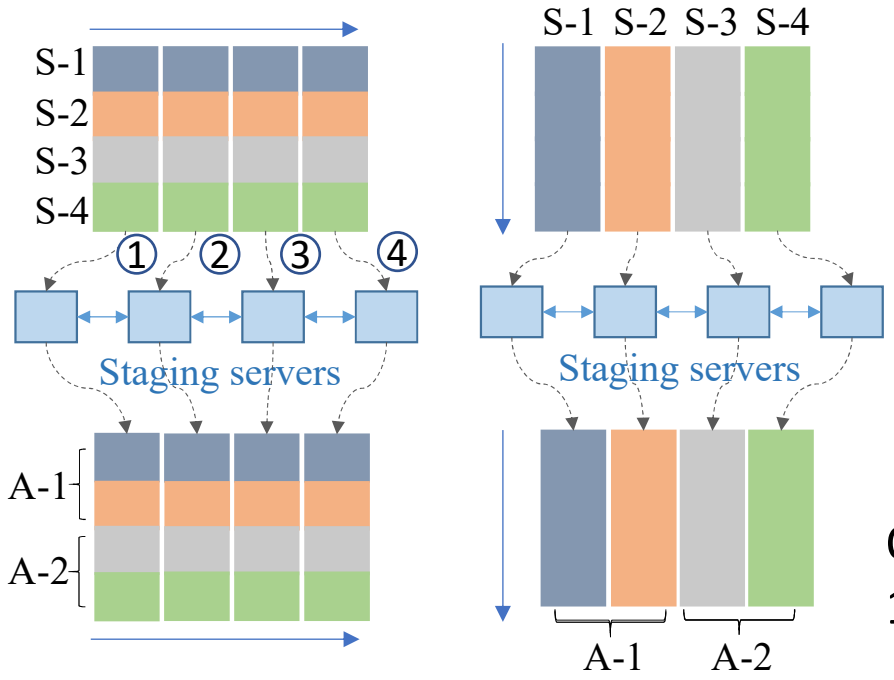




Observations and analyses:

1. Global data partition and layout on staging servers significantly impact the performance of data movement.
2. Avoid decomposed vector fragments to multiple staging nodes.
3. Design a parallel N-to-N accessing at in-memory computing framework.

Global data decomposition

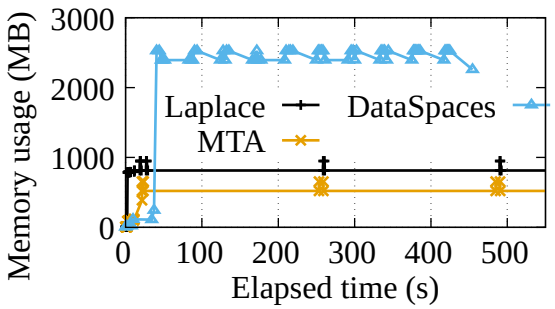


Finding 2. The mismatch between staging data layout and the decomposition strategy can result in the unexpected N-to-1 access to data staging area. This can introduce a significant performance penalty at scale (5.3X performance degradation in our experiments).

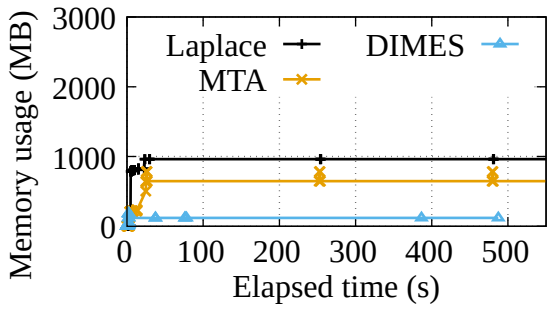
Observations and analyses:

1. Global data partition and layout on staging servers significantly impact the performance of data movement.
2. Avoid decomposed vector fragments to multiple staging nodes.
3. Design a parallel N-to-N accessing at in-memory computing framework.

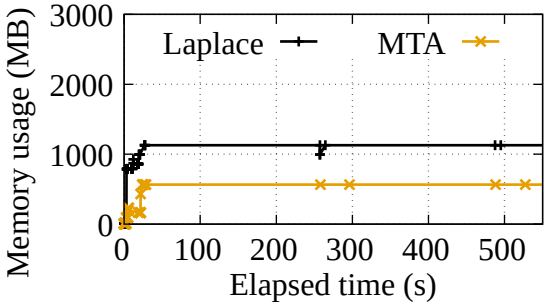
Memory usage and breakdown



DataSpaces, Laplace



DIMES, Laplace



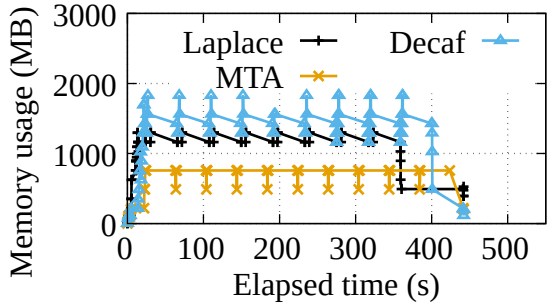
Flexpath, Laplace

Specifications:

1. Each Laplace processor outputs 128 MB.
2. Each DataSpaces server serves 16 Laplace processors.
3. Each Decaf server servers 2 Laplace processors.
4. Flexpath server is integrated into simulation processor.

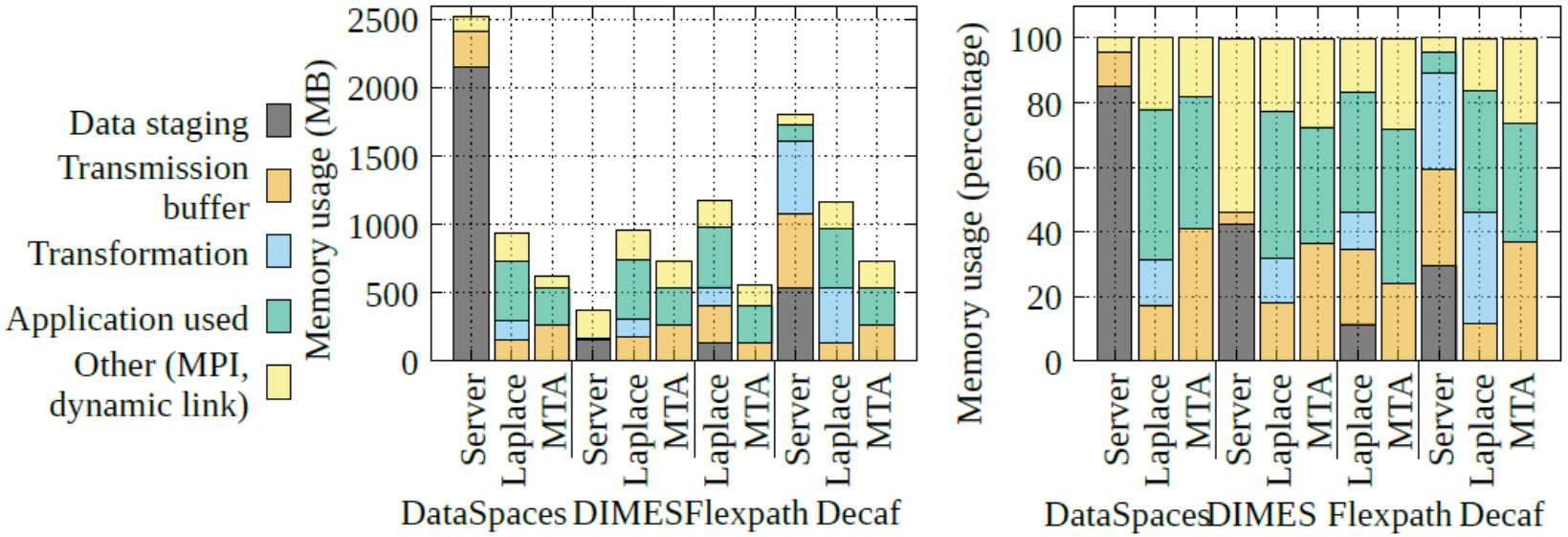
Observations:

1. DIMES can save main memory usage via staging data at RDMA.
2. Decaf uses 2 GB memory, significantly more than we expect (256MB).



Decaf, Laplace

Memory usage breakdown (cont'd)



Observations and analyses:

1. Data transformation and data buffer contribute largely to the memory usage of Decaf (69 %).
2. The transformation and buffer are not only in the server side, but also in simulation and analytics, which are wrapped by Decaf clients.
3. Extra data transformation between raw data and the internal object with rich semantic information, the total memory consumption of Decaf is 7 times that of the raw data size.

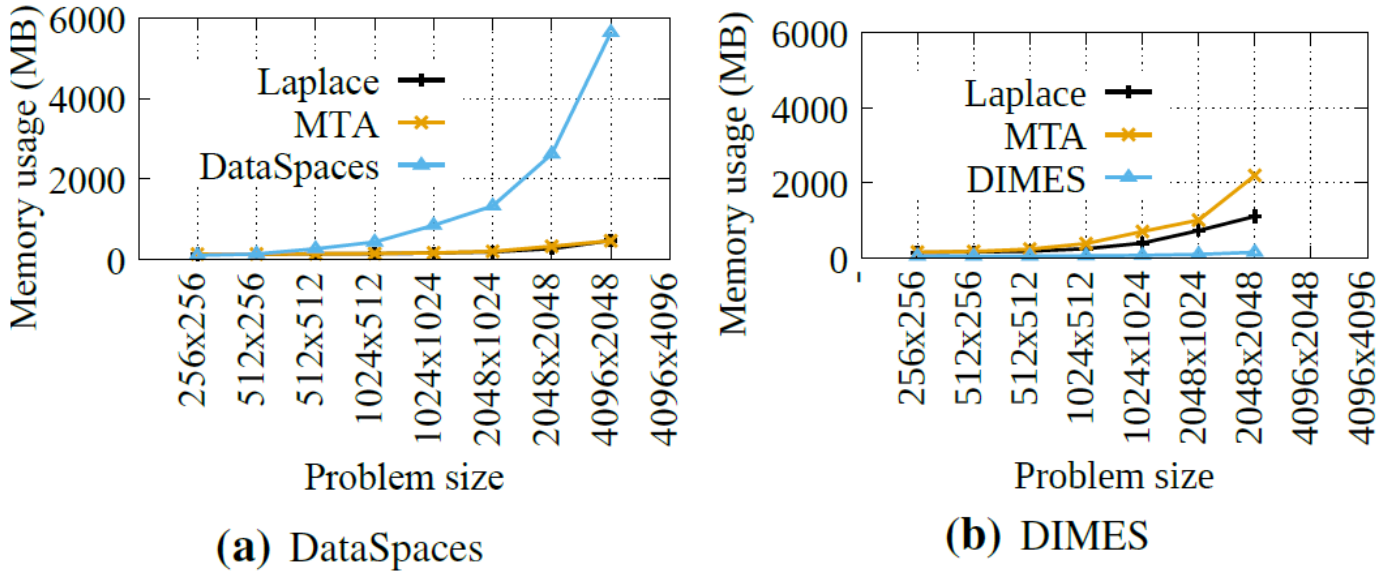
Memory usage breakdown (cont'd)

Finding 3. The raw data transformation to high-level data abstraction with rich metadata and semantics can be overly expensive with regard to the memory consumption, and therefore needs to be carefully managed.

Observations and analyses:

1. Data transformation and data buffer contribute largely to the memory usage of Decaf (69 %).
2. The transformation and buffer are not only in the server side, but also in simulation and analytics, which are wrapped by Decaf clients.
3. Extra data transformation between raw data and the internal object with rich semantic information, the total memory consumption of Decaf is 7 times that of the raw data size.

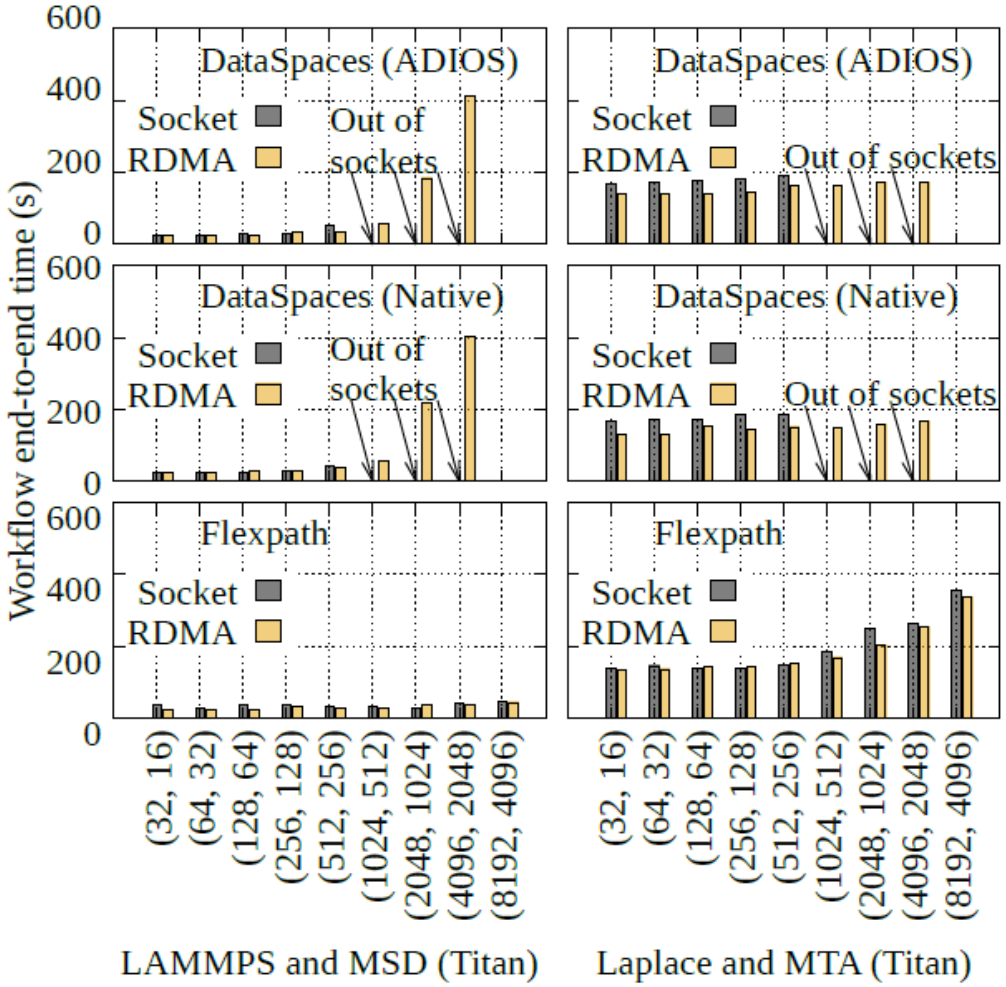
Memory usage (Cost of indexing)



Observations and analyses:

1. DataSpaces (higher indexing cost)
   - SFC will construct a mapping between the 2D data and index space, which is then evenly mapped to DataSpaces servers.
2. DIMES (smaller indexing cost)
   - It stores the index at the simulation processors, rather than the metadata servers.

Using socket for data movement



Observations and analyses:

1. Socket is a system-wide resource and limited by a maximum number.
2. When doing large-scale parallel data movements, DataSpaces and Dimes fail to handle the max socket connections, due to lack of resource control mechanism on sockets.
3. Although, RDMA resource is handled by Cray uGNI. It still suffers “out of memory” without an abstraction layer for resource control.

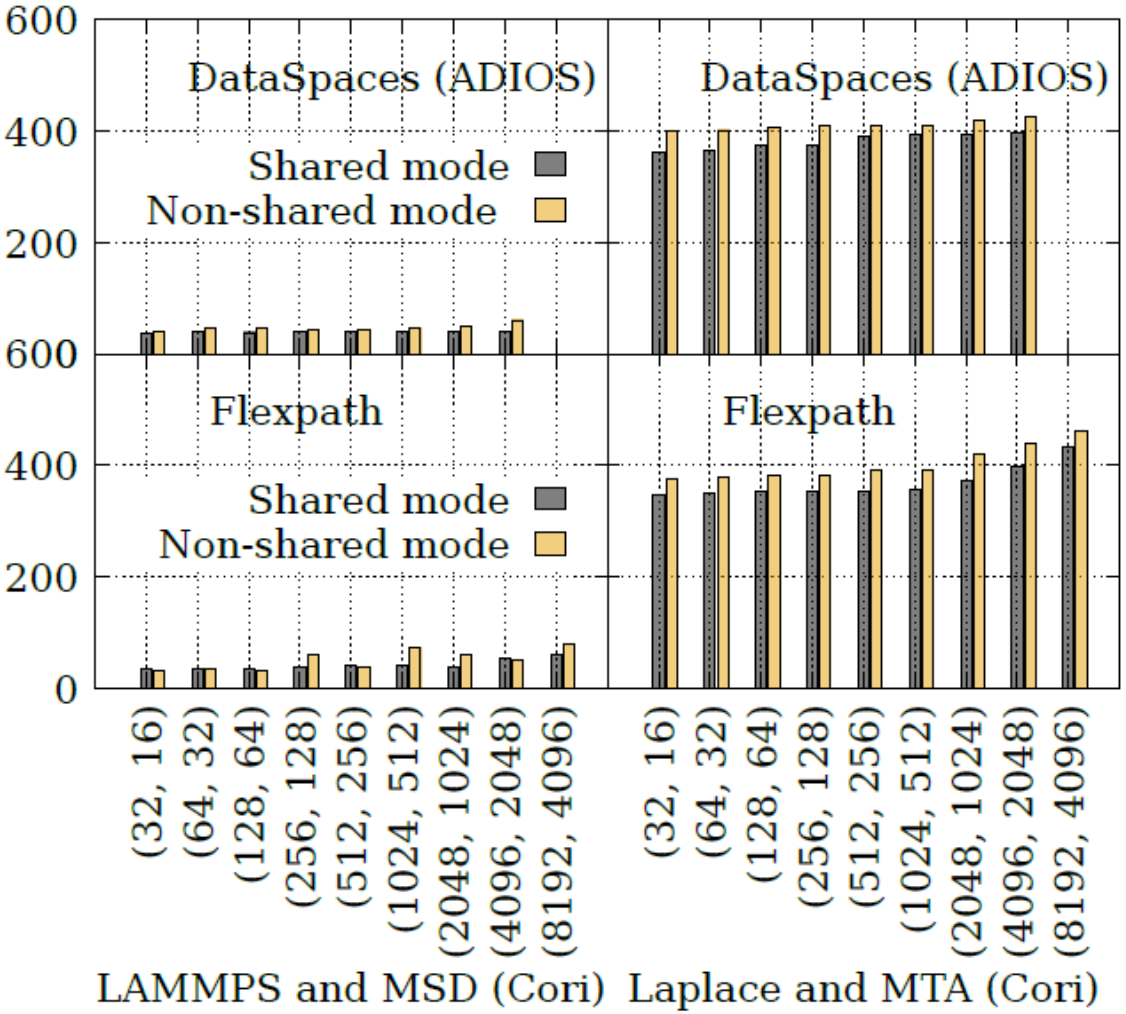
Using socket for data movement

Finding 4. While using high-level protocols is more convenient and portable, proprietary low-level RDMA implementations, yield substantial performance. The accompanied challenge is that the non-trivial engineering effort on adapting the low level implementations to various application scenarios.

Observations and analyses:

1. Socket is a system-wide resource and limited by a maximum number.
2. When doing large-scale parallel data movements, DataSpaces and Dimes fail to handle the max socket connections, due to lack of resource control mechanism on sockets.
3. Although, RDMA resource is handled by Cray uGNI. It still suffers “out of memory” without an abstraction layer for resource control.

Using shared mode



Observations and analyses:

1. The shared mode (co-running simulation, in-memory framework and analysis on one node) can improve about 10 % of performance.
2. Not all HPC systems support shared mode well. E.g. Titan does not allow two MPI instances run on one node and Cori not allow *heterogeneous running.*
3. Cori uses the dynamic RDMA credentials (DRC) to allow for the shared mode. DRC as a centralized service, may be overwhelmed by large-scale parallel data movement.

Using shared mode

Finding 5. Despite the substantial performance improvement (about 10%), shared memory is a restricted running mode on some leadership HPC systems due to security.

Observations and analyses:

1. The shared mode (co-running simulation, in-memory framework and analysis on one node) can improve about 10 % of performance.
2. Not all HPC systems support shared mode well. E.g. Titan does not allow two MPI instances run on one node and Cori not allow *heterogeneous running.*
3. Cori uses the dynamic RDMA credentials (DRC) to allow for the shared mode. DRC as a centralized service, may be overwhelmed by large-scale parallel data movement.

Comparison of Portability

Observations and analyses:

1. Hardware level: GPU is mostly not supported by the current in-memory libraries and GPU-enabled workflows are required to take care of the movement between GPU and CPU memory.
2. Transport layer: DataSpaces, DIMES and Flexpath support both TCP sockets as well as high performance protocols. In contrast, Decaf wraps the workflow components into an MPI communicator.
3. Cori uses the dynamic RDMA credentials (DRC) to allow for the shared mode. DRC as a centralized service, may be overwhelmed by large-scale parallel data movement.

Finding 6. To achieve high performance and portability for expert users, most of in-memory libraries can be configured to reduce the layers of I/O stack and ported to low-level APIs. For non-expert users without the knowledge of performance tuning, these libraries can be ported to high-level abstraction API.

Comparison of Robustness

Issues of running in-memory workflows:

1. Out of RDMA
   - Data movement can deplete the shared RDMA resources on a compute node.
2. Data dimension overflow
   - The dimension size can be overflown, if it is set to 32-bit unsigned integer.
3. Out of main memory
   - In-memory libraries might incur huge memory footprint, resulting in unexpected out of memory aborts.
4. Out of sockets
   - The socket descriptors can be depleted on a compute node.
5. Out of DRC
   - A large scientific workflow may overwhelm the DRC.

Finding 7. Using sophisticate high-level abstractions does not always improve usability and robustness. In an extreme run, available resources might be overwhelmed by high abstraction overhead and lead to crash, particularly while running those data intensive workflows.

Essentials of HPC in-memory computing

- Global in-memory object store (e.g. Dataspaces and DIMEs).
  - + Data organized and staged at memory, can be accessed by various analytics via common APIs, e.g. read, write.
  - - HPC data with high dimension, difficult to be organized and indexed.
  - - High dimension data complex mapping between in-memory storage and user applications.
  - - Lack of supporting in-memory N-to-N access.
- Data flow system.
  - + No need to index data in other storage.
  - + Low complexity on data organization and layout.
  - - Predefine data flow path and data operations.
- Primitive data operations.
  - Split, merge, reduce, transform and etc.

Thank you.