SAND2022-6355C

CCGrid 2022: The 22nd IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing

Assembling Portable In-Situ Workflow from Heterogeneous Components using Data Reorganization

Bo Zhang*, Pradeep Subedi†, Philip E Davis*, Francesco Rizzi‡, Keita Teranishi§ and Manish Parashar*
*Scientific Computing and Imaging Institute, University of Utah, Salt Lake City, Utah 84112
†Samsung Semiconductor Inc., San Jose, California, 95134
‡NexGen Analytics, Sheridan, Wyoming 82801
§Sandia National Laboratories, Livermore, California 94551

Sandia National Laboratories

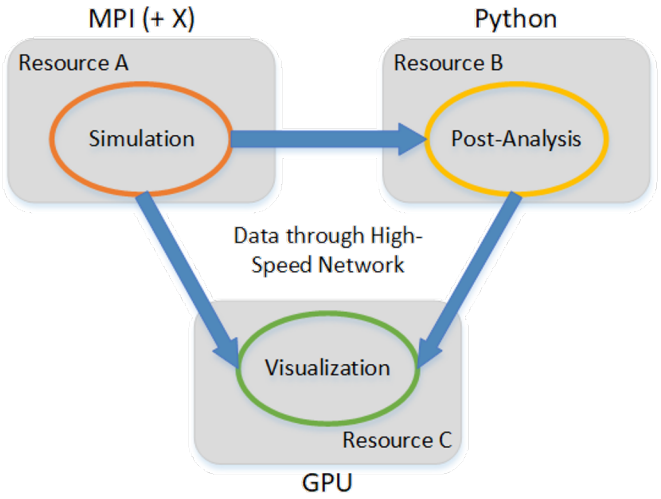
Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Outline

- Introduction
- Background and Related Work
- Heterogeneous Data Reorganization Methods
- Implementation
- Evaluation
- Conclusion and Future Work

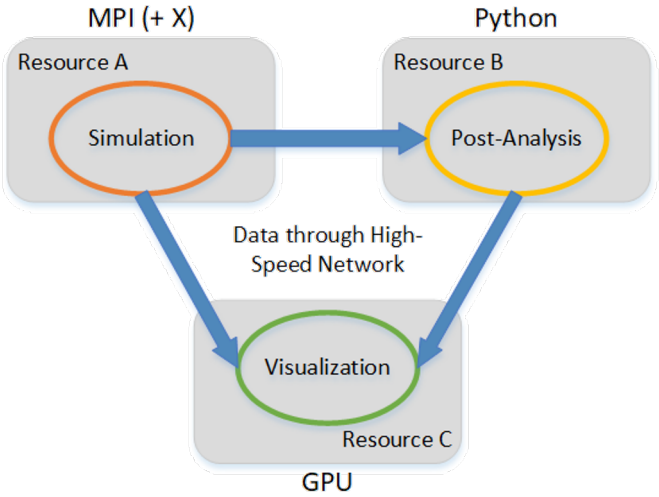
Introduction

- Key Concepts
  - **Loosely-coupled in-situ workflow**: Tasks are running on several components with flexible resources configuration. Data is shared through high-speed network. The memory used to temporarily store the transferred data is called staging area.
  - **Heterogeneity**: Individual components use different programming models (languages) due to performance considerations or development cost.
  - **Data Layout**: The mapping from logical data representation to physical memory location, which is usually constrained by Heterogeneity.
  - **Portability:**
    - **Performance**: Better performance after porting.
    - **Development Cost**: Easy to port.



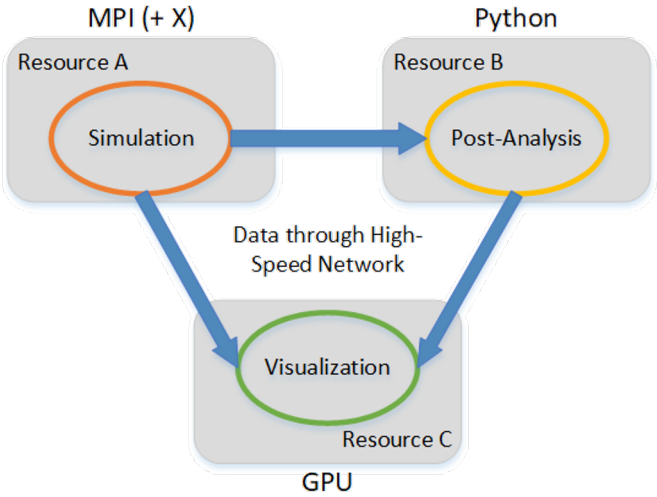
Introduction

- Principles for Building Portable In-Situ Workflow with Heterogeneous Components
  - Individual components:
    - Maximize the performance of each components.
    - Maintain the flexibility of being ported to new model.
  - Data Movement:
    - Optimize the data movement path.
    - **Hide data reorganization when layouts mismatch.**
  - **Coupling Semantics:**
    - **Add heterogeneity into abstraction.**
    - **Automated data reorganization when layouts mismatch.**



Introduction

- Abstraction for Portable In-Situ Workflows with Heterogeneous Components
  - How to maintain the flexibility of the workflow while each component can be easily switched to another programming model?
  - Where should data reorganization be placed to best hide the overhead of layout transformation with different workflow features and resource configurations?



Introduction

- Limited Existing Work
  - Performance portable programming frameworks only focus on the portability of single applications.
  - Workflow coupling middleware do not consider the heterogeneity between components and the requirement for the same data but in different memory layouts associated with these components.
  - The question “how to efficiently solve the data layout mismatch between heterogeneous components” needs to be explored within a breadth of workflow configurations.

Introduction

- Major Contributions
  - An exploration of the trade-offs between three data reorganization methods with respect to the available resources and features of the workflow.
  - A self-adaptive data reorganization method that reduces resource consumption by collecting data access pattern information.
  - A portable application coupling framework prototype that extends Kokkos abstraction to heterogeneous workflow level.
  - An evaluation of the portable application coupling framework prototype compared to current file-based solutions in a synthetic workflow configuration.

Background and Related Work

- Heterogeneity for high performance, but imposes constraints on data layouts.
- Heterogeneity comes from:
  - New Hardware: e.g. CUDA Memory Coalescing[1][2].
  - Legacy Software: e.g. Math Library, Fortran Code.
- Porting Individual applications:
  - Ad-Hoc Design: e.g. CUDA expert porting OpenMP code to CUDA.
  - Portable Programming Abstraction: e.g. Kokkos[3].

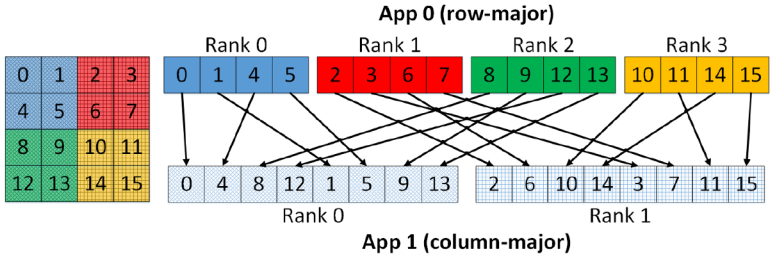


Fig. 1. Data layout mismatch between heterogeneous applications. Left: Application 0 partitions a 4x4 2D array into 4 processors. Application 1 partitions it into 2 processors. Above: Row-major data layout in each processors memory. Below: Column-major data layout in each processor's memory. Arrows show required data movement.

- Assembling heterogeneous applications to a workflow:
  - Data movements between components whose data layout mismatch
    - Unify the data layouts? -> Lose Performance.
    - Ad-Hoc design? -> Development Cost, Overhead of Transformation.

Background and Related Work

- Individual Application Portability
  - Kokkos
  - RAJA[4]
  - SYCL[5]
  - MPI + X[6][7] Practice
- Reorganization Mechanism
  - Parallel I/O Systems[8] - [10]
  - Cloud Environment (Apache Arrow[11])
- Workflow Coupling Framework
  - ADIOS[12]
  - DataSpaces[13]

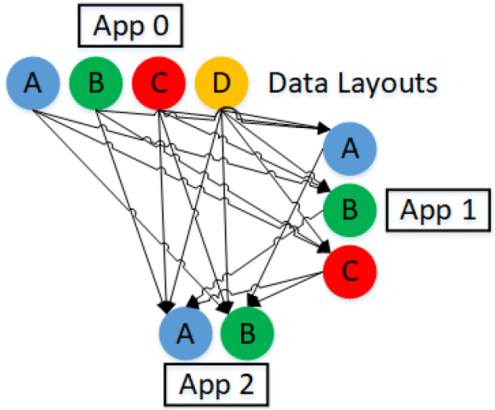


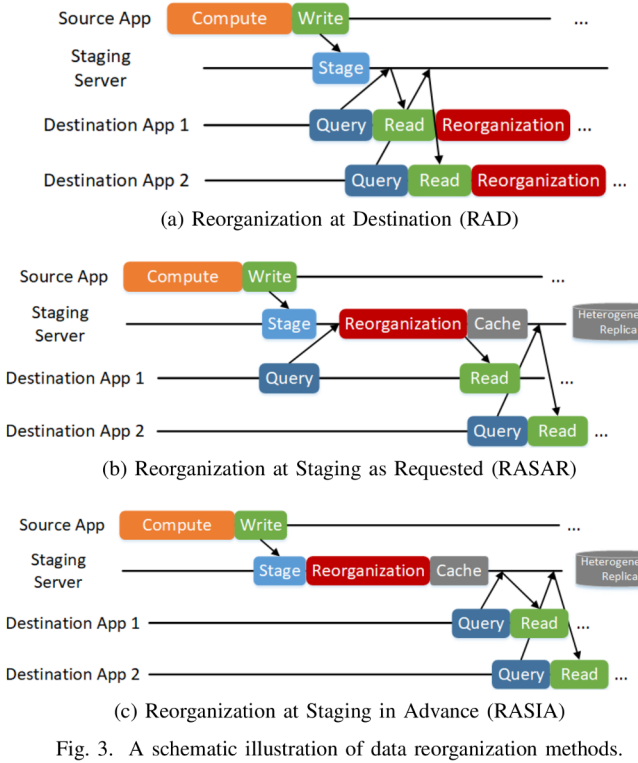
Fig. 2. Complex workflow requires combinatorial numbers of ad-hoc data layout transformations for polymorphic applications.

Heterogeneous Data Reorganization Methods

A. Reorganization at Destination (RAD)
   - Straightforward; consistent performance.
   - No transformation time overlap; no replica reuse.

B. Reorganization at Staging as Requested (RASAR)
   - Replica at server for reuse.
   - No transformation time overlap when no replica is available.

C. Reorganization at Staging in Advance (RASIA)
   - Overlapped transformation time.
   - Unnecessary replica; waste computing and high memory consumption.



(a) Reorganization at Destination (RAD)

(b) Reorganization at Staging as Requested (RASAR)

(c) Reorganization at Staging in Advance (RASIA)

Fig. 3. A schematic illustration of data reorganization methods.

Heterogeneous Data Reorganization Methods

D. Self-Adaptive Hybrid Reorganization (SAHR)

- Assumptions:
  - A particular numerical application is interested in a fixed set of data objects with iterative values.
  - The particular numerical application only requests one specific layout for each data object.
- Access Pattern Collection module:
  - An access pattern is defined by: variable name, layout, domain index descriptor(bounding box), and frequency.
  - Clients (individual applications) keeps their own patterns and update to the staging server; Server keeps all patterns from every application.
  - Always calculate the superset for intersections to avoid duplicated region.

Algorithm 1 Get Pattern Update

```
qodsc ←MetaData for the queried data object {Contains varname, bounding box descriptor, version, layout, src_layout, etc..}
query ← Query(qodsc) {Request for a specified data object}
pattern ← ExtractPattern(query)
record_list ←SearchGetPattern(pattern.varname, pattern.layout)
if record_list != NULL then
  for all record in record_list do
    if CheckGridIntersection(pattern.bbox, record.bbox) then
      pattern ←CalculateSuperSet(pattern, record)
    end if
  end for
end if
record_list ←UpdateGetPattern(pattern)
```

Heterogeneous Data Reorganization Methods

D. Self-Adaptive Hybrid Reorganization (SAHR)

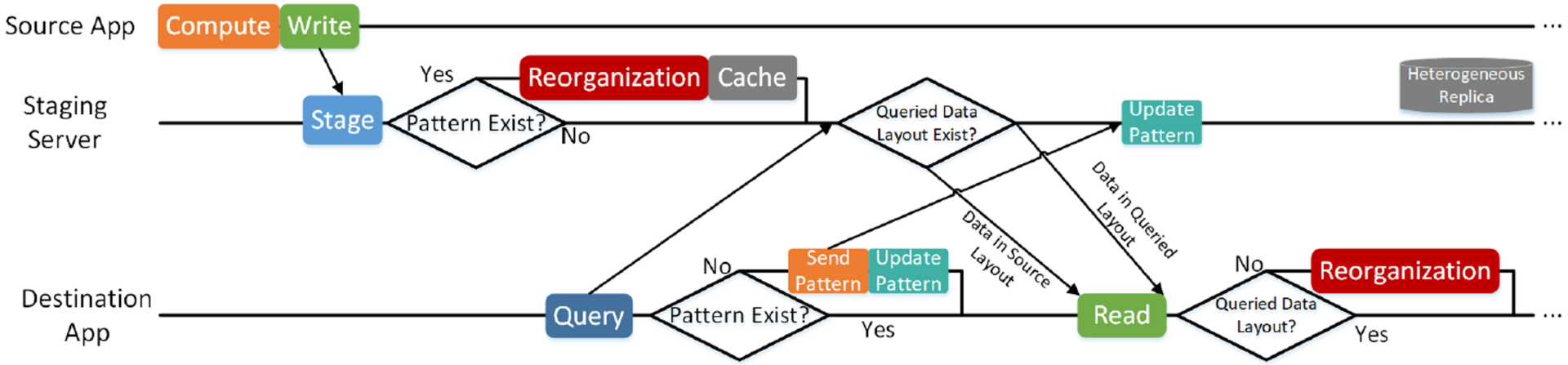


Fig. 4. A schematic illustration of Self-Adaptive Hybrid Reorganization (SAHR) method.

Implementation

A. Heterogeneous DataSpaces

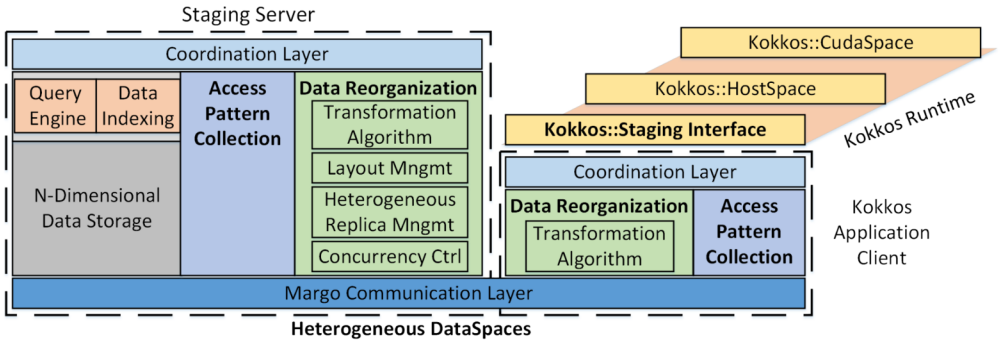


Fig. 5. Architecture of Kokkos Staging Space. The data reorganization module and Kokkos::Staging API were implemented on top of the DataSpaces and Kokkos framework respectively.

Access Pattern Collection module:
Identify and record new data access pattern

Data Reorganization module:

- **Transformation Algorithm**: Define how to transform from Layout A to B.
- **Layout Management**: Manage all layouts.
- **Heterogeneous Replica Management**: Keep a superset of domain data for specific layouts; remove redundancy.
- **Concurrency Control**: avoid repetitive transformation incurred by concurrent data requests.

Implementation

B. Kokkos::Staging Interface

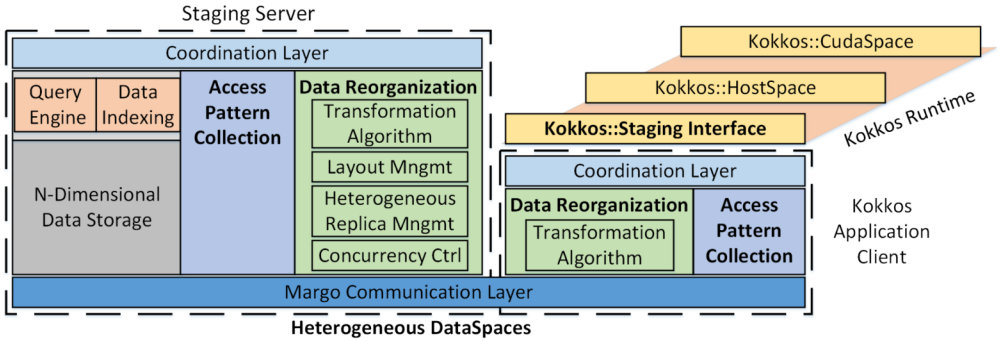


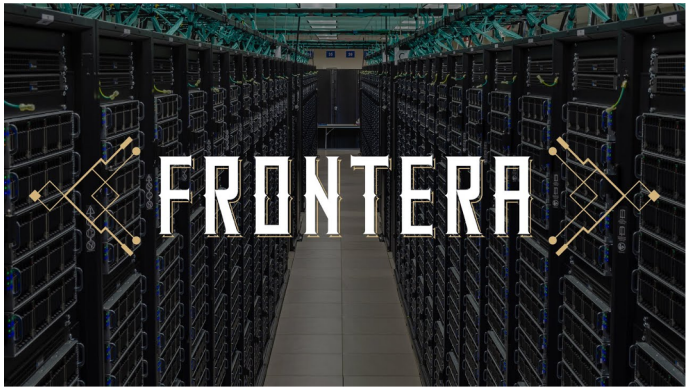
Fig. 5. Architecture of Kokkos Staging Space. The data reorganization module and Kokkos::Staging API were implemented on top of the DataSpaces and Kokkos framework respectively.

```
1  Kokkos::Staging::initialize();
2  {
3  using ViewHost_lr_t     = Kokkos::View<Data_t**,
4        Kokkos::LayoutRight, Kokkos::HostSpace>;
5  using ViewHost_ll_t     = Kokkos::View<Data_t**,
6        Kokkos::LayoutLeft, Kokkos::HostSpace>;
7  using ViewStaging_lr_t = Kokkos::View<Data_t**,
8        Kokkos::LayoutRight, Kokkos::StagingSpace>;
9  using ViewStaging_ll_t = Kokkos::View<Data_t**,
10       Kokkos::LayoutLeft, Kokkos::StagingSpace>;
11 ViewHost_lr_t v_P("PutView", i0, i1);
12 ViewStaging_lr_t v_S_lr("StagingView_LayoutRight",
13                          i0, i1);
14 ViewStaging_ll_t v_S_ll("StagingView_LayoutLeft",
15                          i0, i1);
16 ViewHost_ll_t v_G("GetView", i0, i1);
17 // global domain geometric descriptor
18 Kokkos::Staging::set_lower_bound(v_S_lr, lb0, lb1);
19 Kokkos::Staging::set_upper_bound(v_S_lr, lb0, lb1);
20 Kokkos::Staging::set_lower_bound(v_S_ll, lb0, lb1);
21 Kokkos::Staging::set_upper_bound(v_S_ll, lb0, lb1);
22 // global iteration
23 Kokkos::Staging::set_version(v_S_lr, version);
24 Kokkos::Staging::set_version(v_S_ll, version);
25 // bind two staging views in different layout
26 Kokkos::Staging::view_bind_layout(v_S_ll, v_S_lr);
27 // from host to staging
28 Kokkos::deep_copy(v_S_lr, v_P);
29 // from staging to host
30 Kokkos::deep_copy(v_G, v_S_ll);
31 }
32 Kokkos::Staging::finalize();
```

Fig. 6. Code example of data exchange between Kokkos views in different layouts

Evaluation - Platform

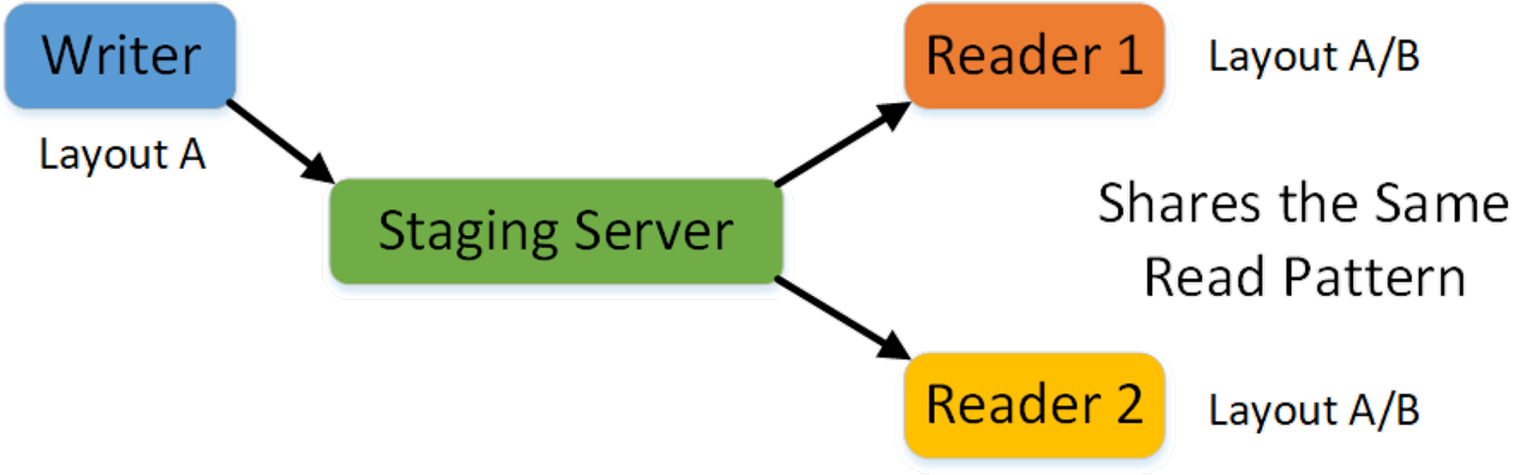
- Frontera at TACC



Processor	Clock Rate	Physical Cores /Node	Threads/ Core	Sockets/ Node	RAM/ Node	Network
Intel Xeon Platinum 8280	2.7 GHz	56	1	2	192 GB DDR4	Mellanox HDR

Evaluation - Workflow

- Synthetic Staging-Based In-Situ Workflow



Evaluation

A. Exploring the task placement of data reorganization

i. Metric 1 - Cycle time of writer and reader

- writers and readers ran with 0, 5, 10, 20 seconds of sleep after each computation time step.

TABLE I
EXPERIMENTAL SETUP CONFIGURATIONS FOR SYNTHETIC EXPERIMENTS

Data Domain	1024 × 1024 × 1024
No. of Parallel Writer Cores (Nodes)	512(16)
No. of Parallel Reader Cores (Nodes)	64(4)
No. of 2nd Parallel Reader Cores (Nodes)	64(4)
No. of Staging Cores (Nodes)	32(8)
Total Staged Data Size (15 Time-steps)	120 GB

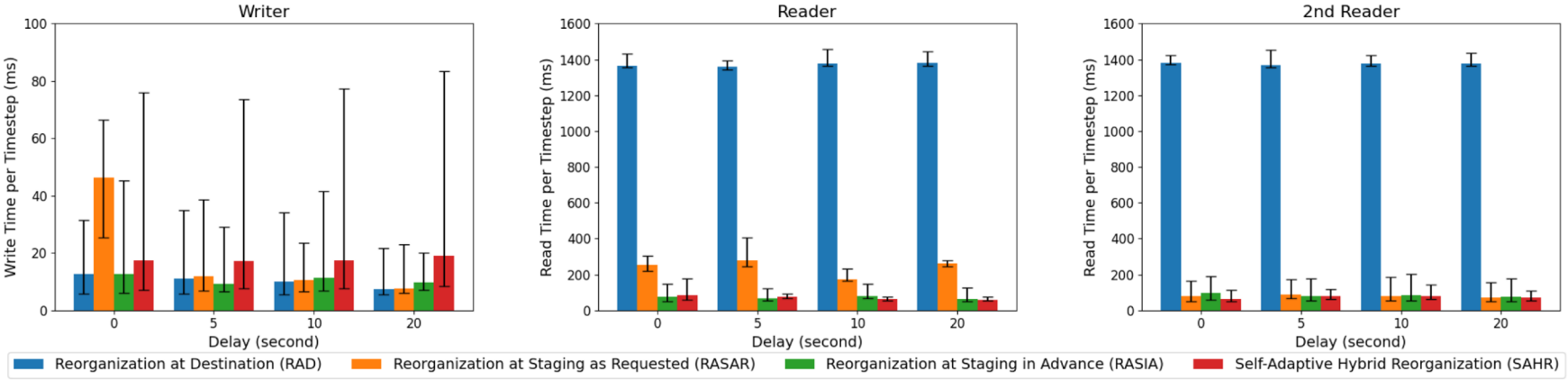


Fig. 7. Comparison of I/O time per time step among four data reorganization methods with varying cycle time of applications

Evaluation

ii. Metric 2 - Staging server scale

- Fixed writer scale: 512 cores (16 nodes); reader scale: 64 cores (4 nodes).
- Vary staging server scale: 8 cores (2 nodes), 16 cores (4 nodes), 32 cores (8 nodes).

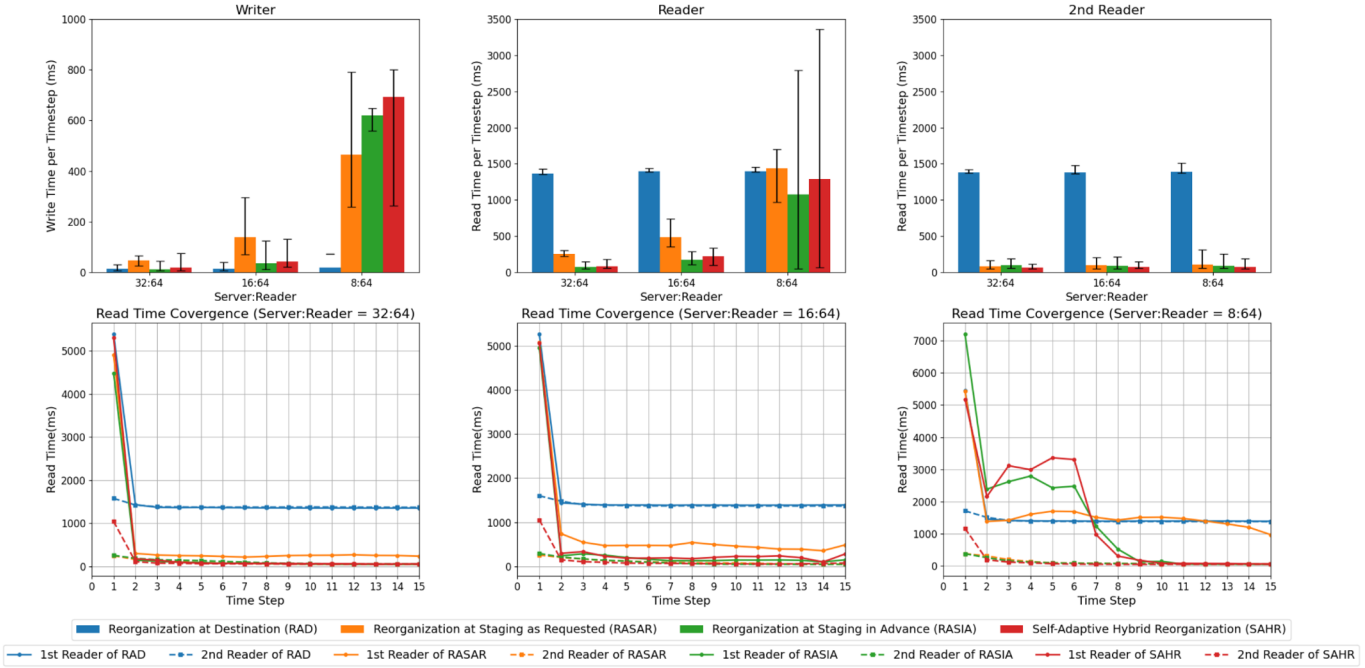


Fig. 8. Comparison of I/O time per time step among four data reorganization methods with different staging server scale

Evaluation

iii. Metric 3 - Data size of reading subset domain

- Read a geometric core, whose coordinates are {*(1024-d)/2, (1024+d)/2*} in each dimension.
- *d*=128, 256, 512; only read a d^3 cube from the geometric core of the entire data domain.

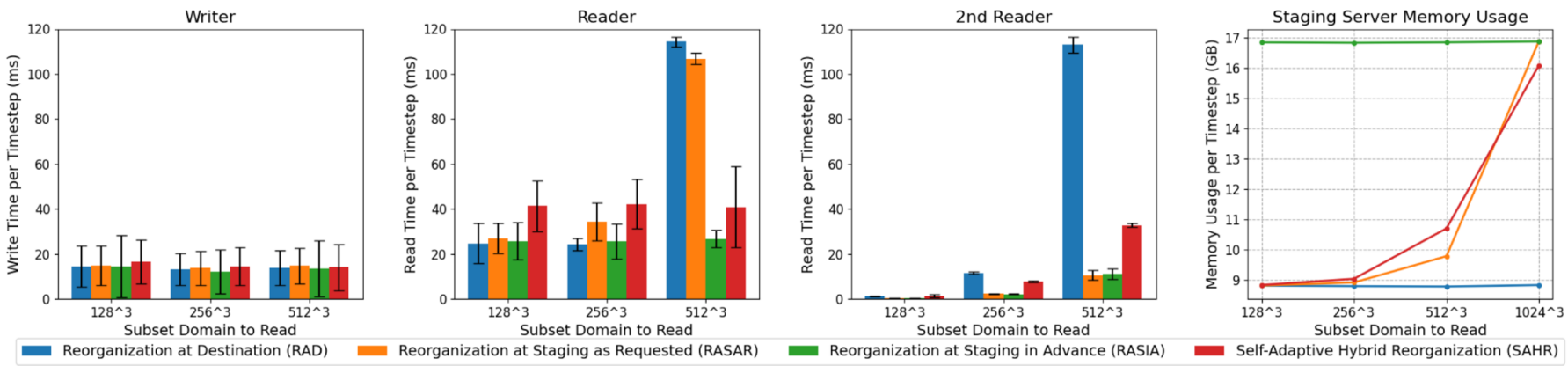


Fig. 9. Comparison of I/O time & staging server memory usage per time step among four data reorganization methods with different size of subset domain to read

Evaluation

B. Strong scaling comparison to existing Kokkos backends

- Same Layout: StdIO vs HDF5 vs DataSpaces.
- Different Layout: StdIO + Ad-Hoc Re-org vs HDF5 + Ad-Hoc Re-org vs DataSpaces +SAHR.

TABLE II

EXPERIMENTAL SETUP CONFIGURATIONS OF DATA DOMAIN, CORE-ALLOCATIONS AND SIZE OF THE STAGED DATA FOR STRONG SCALING TESTS

Data Domain	$1024 \times 1024 \times 1024$				
No. of Parallel Writer Cores (Nodes)	256(8)	512(16)	1024(32)	2048(64)	4096(128)
No. of Parallel Reader Cores (Nodes)	32(2)	64(4)	128(8)	256(16)	512(32)
No. of 2nd Parallel Reader Cores (Nodes)	32(2)	64(4)	128(8)	256(16)	512(32)
No. of Staging Cores (Nodes)	16(4)	32(8)	64(16)	128(32)	256(64)
Total Staged Data Size (15 Time-steps)	120 GB				
Cycle Time	20 second				

Evaluation

B. Strong scaling comparison to existing Kokkos backends

- Same Layout: StdIO vs HDF5 vs DataSpaces.
- Different Layout: StdIO + Ad-Hoc Re-org vs HDF5 + Ad-Hoc Re-org vs DataSpaces +SAHR.

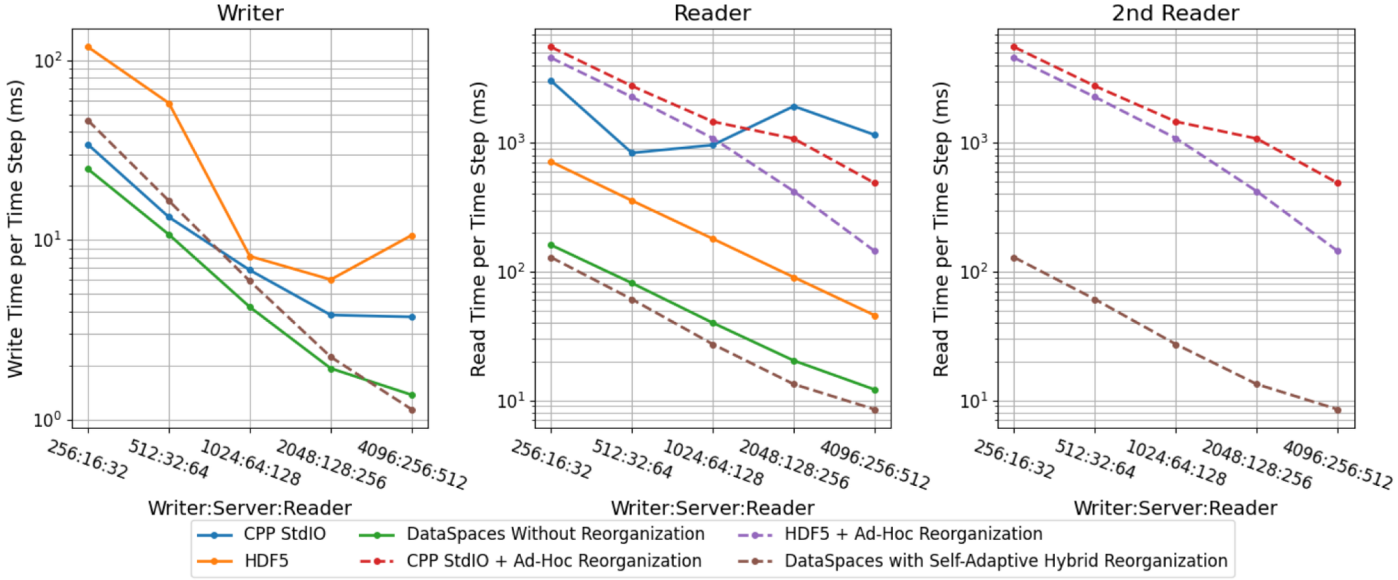


Fig. 10. Strong scaling comparison of I/O time per time step among C++ standard I/O, HDF5 and DataSpaces

Conclusion and Future Work

- We explored the trade-offs between different data reorganization methods within various experimental configurations, and then propose a Self-Adaptive Hybrid Reorganization (SAHR) method which reduces resource consumption by collecting data access pattern information.
- By integrating asynchronous data layout conversions, we implemented Kokkos Staging Space as an extension of Kokkos to workflow level, and demonstrated its effectiveness in terms of both time-to-solution and scalability for inter-application data exchange.
- In future work, we plan to support more data reorganization types, such as the transformation between Array of Structs(AoS) and Struct of Arrays(SoA), and to evaluate these methods using a production scientific workflow of heterogeneous components.

Reference

1. Bell, Nathan, and Michael Garland. *Efficient sparse matrix-vector multiplication on CUDA*. Vol. 2. No. 5. Nvidia Technical Report NVR-2008-004, Nvidia Corporation, 2008.
2. (2021) Cuda c++ programming guide - v11.5.1. [Online]. Available: https://docs.nvidia.com/cuda/pdf/CUDA C Programming Guide.pdf
3. Trott, Christian R., et al. "Kokkos 3: Programming model extensions for the exascale era." IEEE Transactions on Parallel and Distributed Systems 33.4 (2021): 805-817.
4. Beckingsale, David A., et al. "Raja: Portable performance for large-scale scientific applications." *2019 ieee/acm international workshop on performance, portability and productivity in hpc (p3hpc)*. IEEE, 2019.
5. Reyes, Ruyman, et al. "SYCL 2020: more than meets the eye." *Proceedings of the International Workshop on OpenCL*. 2020.
6. Khuvis, Samuel, et al. "Exploring Hybrid MPI+ Kokkos Tasks Programming Model." *2020 IEEE/ACM 3rd Annual Parallel Applications Workshop: Alternatives To MPI+ X (PAW-ATM)*. IEEE, 2020.
7. Deakin, Tom, and Simon McIntosh-Smith. "Evaluating the performance of HPC-style SYCL applications." *Proceedings of the International Workshop on OpenCL*. 2020.
8. Wan, Lipeng, et al. "Improving I/O Performance for Exascale Applications through Online Data Layout Reorganization." *IEEE Transactions on Parallel and Distributed Systems* 33.4 (2021): 878-890.
9. He, Shuibing, et al. "Optimizing parallel I/O accesses through pattern-directed and layout-aware replication." *IEEE Transactions on Computers* 69.2 (2019): 212-225.
10. Tang, Houjun, et al. "Usage pattern-driven dynamic data layout reorganization." *2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*. IEEE, 2016.
11. (2021) Apache arrow: A cross-language development platform for in-memory data. [Online]. Available: https://arrow.apache.org/
12. Godoy, William F., et al. "Adios 2: The adaptable input output system. a framework for high-performance data management." *SoftwareX* 12 (2020): 100561.
13. Docan, Ciprian, Manish Parashar, and Scott Klasky. "Dataspaces: an interaction and coordination framework for coupled simulation workflows." *Cluster Computing* 15.2 (2012): 163-181.

Thanks for your time !

bozhang@sci.utah.edu