1st exaFOAM Workshop
Parallel I/O

Exploitation of Exascale Systems for Open-Source Computational Fluid Dynamics by Mainstream Industry

Funding Body: EuroHPC-03-2019
Call: H2020-JTI-EuroHPC-2019-1
Project Number: 9564167
Project Duration: 1st April 2021 to 31st March 2024

Presenter: R. Gregor Weiß
Sergey Lesnik, Flavio C. C. Galeazzo, Henrik Rusche, Andreas Ruopp

EuroHPC
Joint Undertaking

June 12th, 2023

Agenda

- Introduction to Parallel I/O
- OpenFOAM I/O Formats
- Towards Parallel I/O for OpenFOAM
- Conclusion

Introduction – Parallel File Systems

- Space and bandwidth scale with number of object storage hardware.

  → Up to O(TB/s).

- Storage metadata stored as **inode per file**.

  → minimize number of files.

- Lookup suffers contention if overutilized by many **requests**.

  → minimize I/O operations.

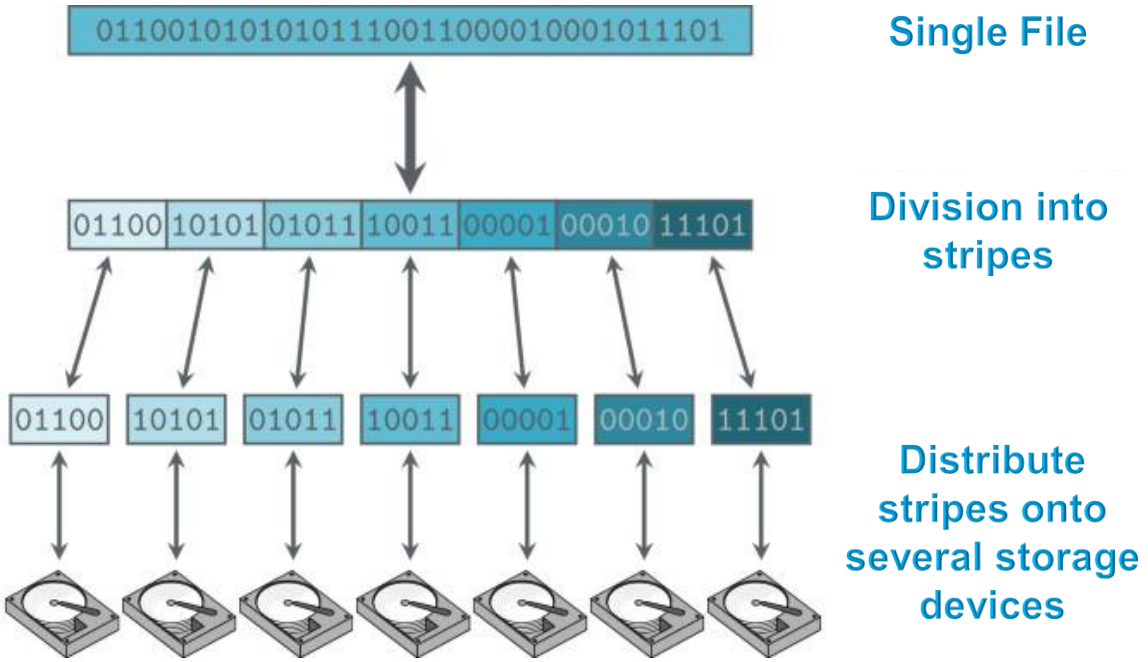


Figure modified and used under CC-BY 4.0 from [1]

Optimize number of files, I/O operations and access size.

EuroHPC Joint Undertaking

[1] M. Seiz et al. Lustre I/O Performance Investigations on Hazel Hen: Experiments and Heuristics. J. Supercomut 77, 12508-12536 (2021)

Introduction – Parallel File Systems

- Space and bandwidth scale with number of object storage hardware.

  → Up to O(TB/s).

- Storage metadata stored as **inode per file**.

  → minimize number of files.

- Lookup suffers contention if overutilized by many **requests**.

  → minimize I/O operations.

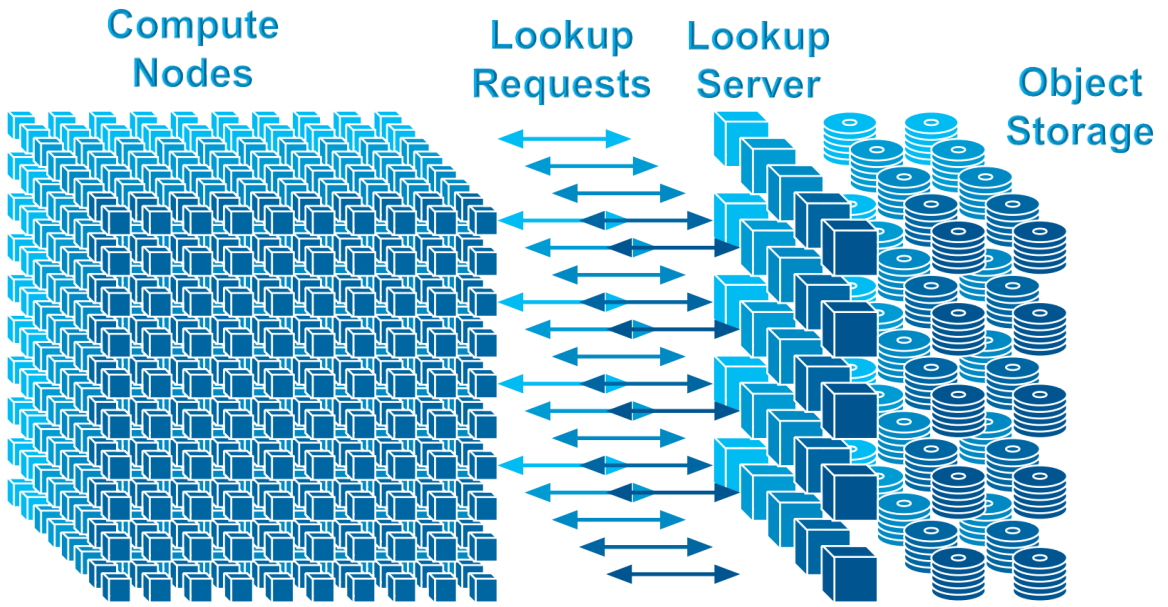


Figure modified from [2]

Optimize number of files, I/O operations and access size.

Introduction – Parallel I/O Libraries

MPI-IO

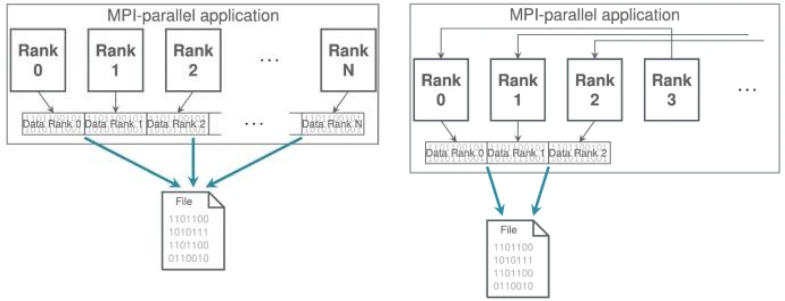


Figure modified and used under CC-BY 4.0 from [1]

- N:1 model or M:1 model.
- Plain binary data files.
- Flexible choice of # of writers at runtime.
- Can handle irregular access pattern.

ADIOS2

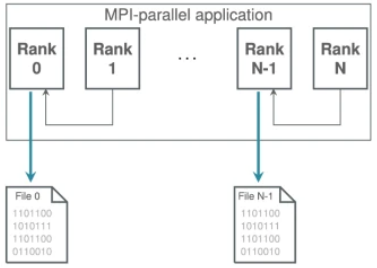


Figure modified and used under CC-BY 4.0 from [1]

```
.
`-- adios2.bp
    |-- data.0
    |-- data.1
    |-- data.*
    ...
    |-- md.0
    |-- md.idx
    |-- mmd.0
    `-- profiling.json
```

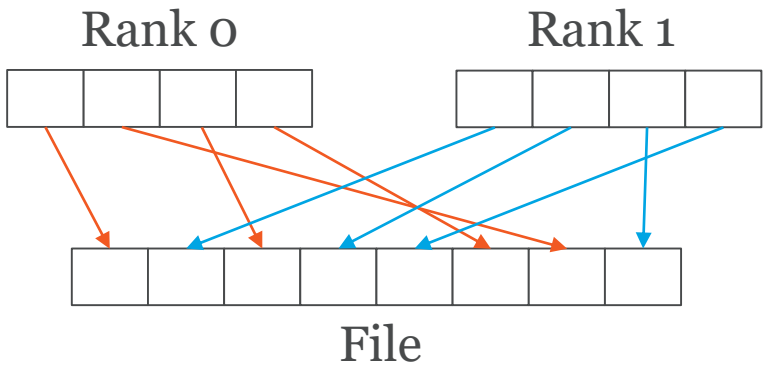
- N:M model.
- Structure of binary files by metadata.
- Flexible choice of # of files at runtime.
- Requires contiguous access pattern.

EuroHPC Joint Undertaking

[1] M. Seiz et al. Lustre I/O Performance Investigations on Hazel Hen: Experiments and Heuristics. J. Supercomut 77, 12508-12536 (2021)

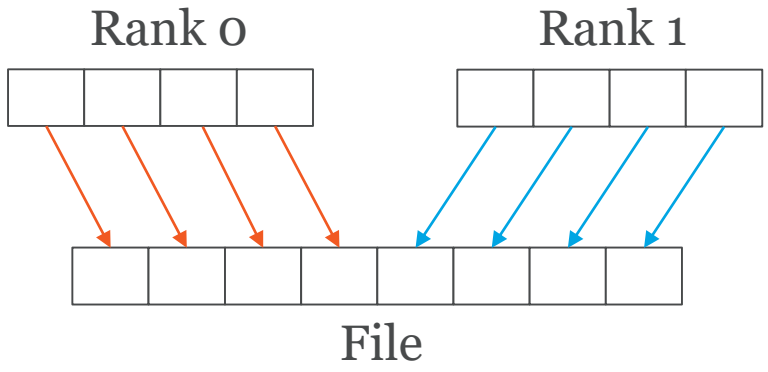
Introduction – I/O Access Patterns

Irregular Access Pattern



- Irregular mapping required.
- Assembly overhead during I/O.

Contiguous Access Pattern



- Start and size required.
- No reorganization.

Simple I/O access patterns optimize time spend in reorganization overheads.

OpenFOAM I/O Formats

OpenFOAM I/O Formats

Uncollated Format

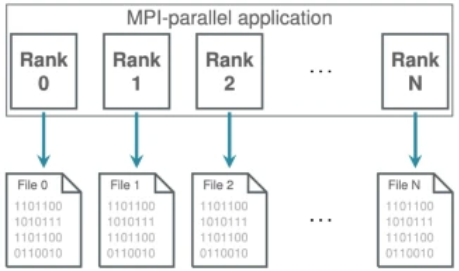


Figure modified and used under CC-BY 4.0 from [1]

- N:N model → # of inodes scales with # of processors
- Once the case is decomposed:
  - Only restarts on the same decomposition.
  - Modification in boundary conditions are tedious.
- Decomposition, reconstruction, and redistribution are costly for the time-to-solution.

Case structure parallel run

```
.
|-- processor0
|   |-- 0
|   |   |-- p
|   |   `-- U
|   `-- constant
|       `-- polyMesh
|           |-- boundary
|           |-- boundaryProcAddressing
|           |-- cellProcAddressing
|           |-- faceProcAddressing
|           |-- faces
|           |-- neighbour
|           |-- owner
|           |-- pointProcAddressing
|           `-- points
|-- processor1
|   |-- 0
|   |   |-- p
|   |   `-- U
```

Showstopper at scale.

EuroHPC Joint Undertaking

[1] M. Seiz et al. Lustre I/O Performance Investigations on Hazel Hen: Experiments and Heuristics. J. Supercomut 77, 12508-12536 (2021)

Collated Format

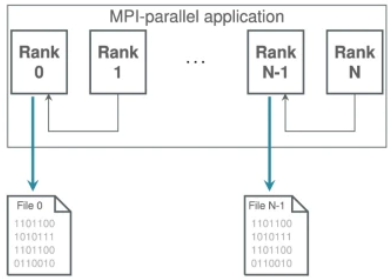
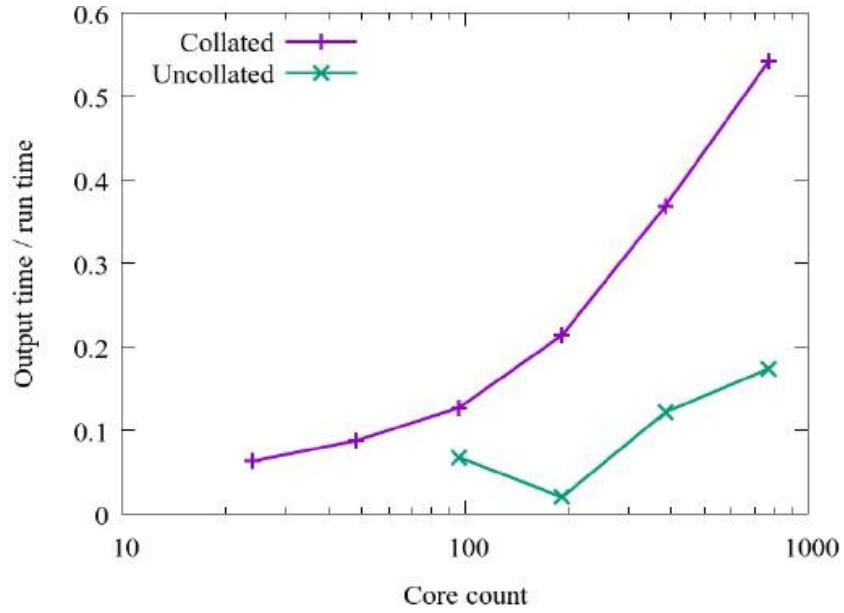


Figure modified and used under CC-BY 4.0 from [1]

- N:M model → Reduces the number of inodes.
- Once the case is decomposed:
  - Only restarts on the same decomposition.
- Adds considerable runtime overhead.
- Decomposition, reconstruction, and redistribution are costly for the time-to-solution.

More than 50% of the runtime spend in I/O.

Figure modified from [3]



[1] M. Seiz et al. Lustre I/O Performance Investigations on Hazel Hen: Experiments and Heuristics. J. Supercomput 77, 12508-12536 (2021)
[3] https://www.archer2.ac.uk/training/courses/200624-openfoam-io/ - 14th Oct 2022 10AM

Collated Format

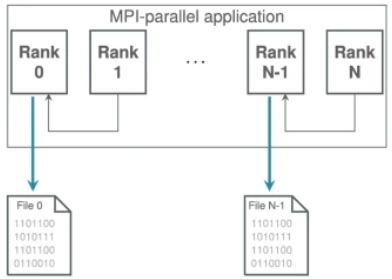
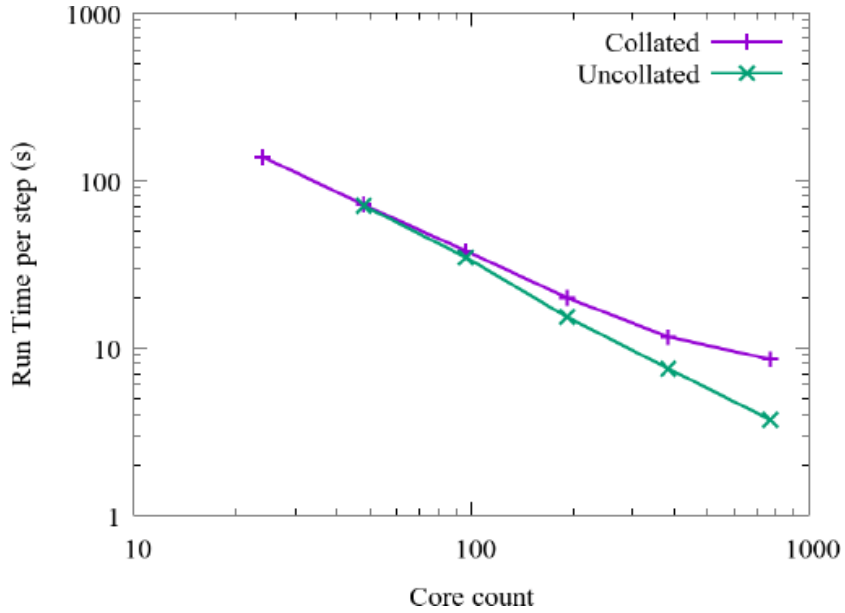


Figure modified and used under CC-BY 4.0 from [1]

- N:M model → Reduces the number of inodes.
- Once the case is decomposed:
  - Only restarts on the same decomposition.
- Adds considerable runtime overhead.
- Decomposition, reconstruction, and redistribution are costly for the time-to-solution.

More than 100% increase in runtime.

Figure modified from [3]



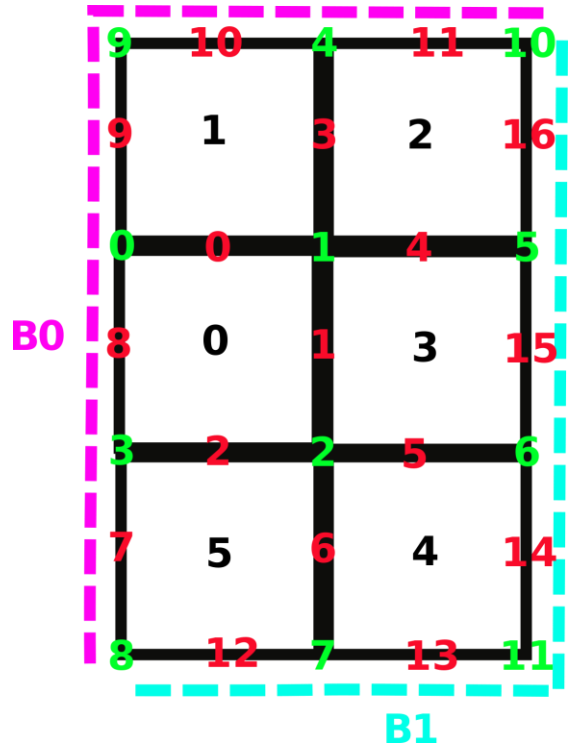
Limited scalability.

[1] M. Seiz et al. Lustre I/O Performance Investigations on Hazel Hen: Experiments and Heuristics. J. Supercomut 77, 12508-12536 (2021)
[3] https://www.archer2.ac.uk/training/courses/200624-openfoam-io/ - 14th Oct 2022 10AM

Data layout of OpenFOAM

- Storage of mesh connectivity by owner and neighbour cell IDs of faces.
- Boundary patch faces follow internal faces.
- Decomposition renumbers and fragments the mesh.
- Fragments would pose an irregular I/O access pattern.

Global			Partition 0			Partition 1		
face	owner	neighbour	face	owner	neighbour	face	owner	neighbour
0	0	1	0	0	1			
1	0	3	7	0		8	0	
2	0	5	8	0		9	2	
3	1	2	1	1	2			
4	2	3	9	2		7	0	
5	3	4				0	0	1
6	4	5				1	1	2
7	5					6	2	
8	0		2	0				
9	1		3	1				
10	1		4	1				
11	2		5	2				
12	5					5	2	
13	4					4	1	
14	4					3	1	
15	3					2	0	
16	2		6	2				



exaFOAM

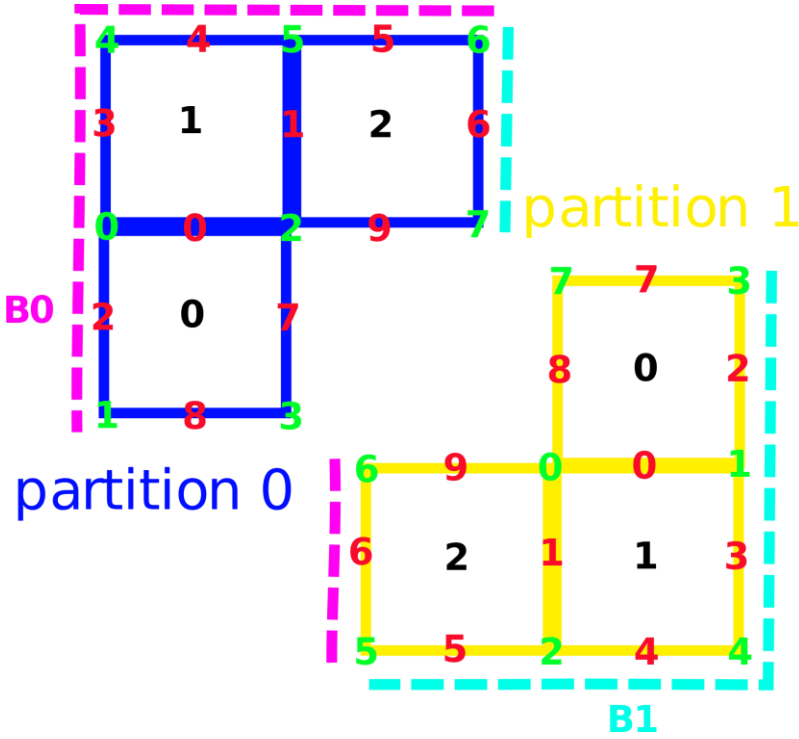
HLRS

WIKKI

Data layout of OpenFOAM

- Storage of mesh connectivity by owner and neighbour cell IDs of faces.
- Boundary patch faces follow internal faces.
- Decomposition renumbers and fragments the mesh.
- Fragments would pose an irregular I/O access pattern.

Global			Partition 0			Partition 1		
face	owner	neighbour	face	owner	neighbour	face	owner	neighbour
0	0	1	0	0	1			
1	0	3	7	0		8	0	
2	0	5	8	0		9	2	
3	1	2	1	1	2			
4	2	3	9	2		7	0	
5	3	4				0	0	1
6	4	5				1	1	2
7	5					6	2	
8	0		2	0				
9	1		3	1				
10	1		4	1				
11	2		5	2				
12	5					5	2	
13	4					4	1	
14	4					3	1	
15	3					2	0	
16	2		6	2				



EuroHPC Joint Undertaking

Towards Parallel I/O for OpenFOAM

Parallel I/O for exascale ready OpenFOAM.

Objectives

1. Reduce number of files and optimize I/O performance at scale. → Current showstopper for scalability.
2. Decrease time-to-solution by improved I/O access pattern. → “Performance optimization” sits here.

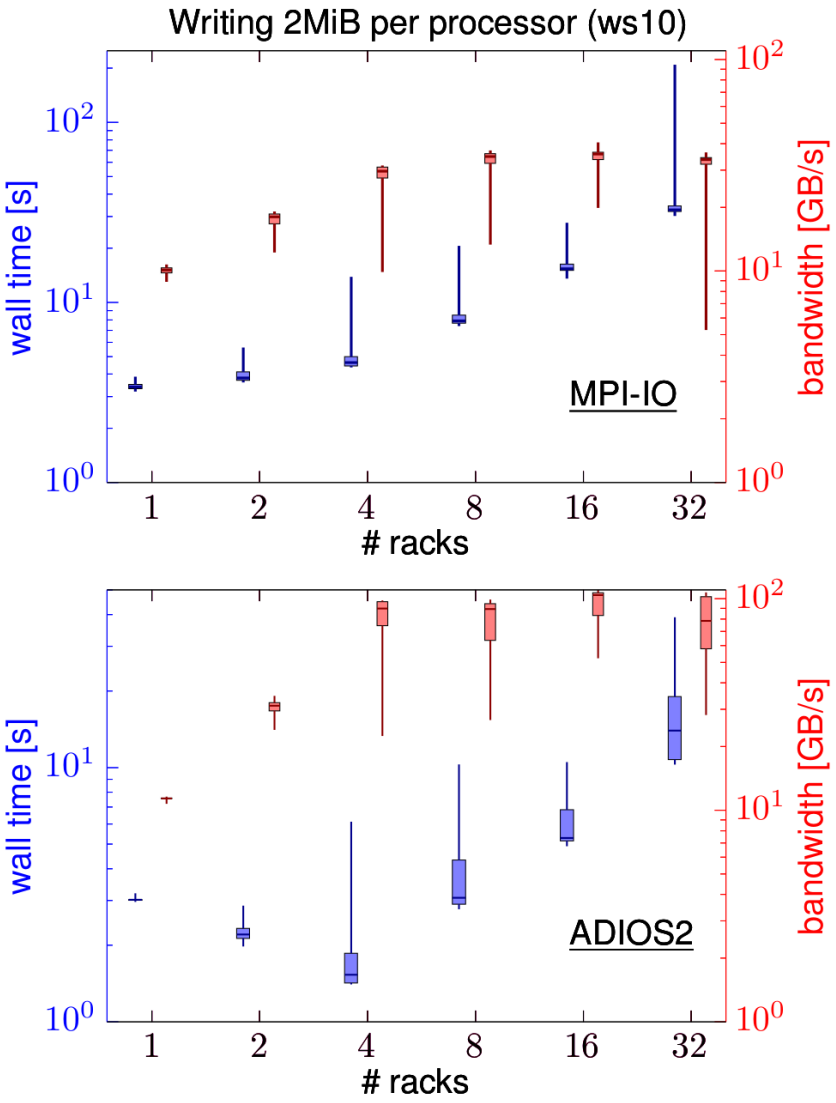
Approach

- Development of an I/O proxy application to compare I/O schemes and libraries.
- Design and implement contiguous I/O access pattern for FOAM-native parallel I/O.

I/O Proxy Application

- Simple heat transfer solver with contiguous data layout.
- Parallel I/O at scale with ADIOS2 and MPI-IO.
- Wall time of opening, writing, and closing.
- Weak scaling benchmark of writing 2MiB per processor.
- Blocking semantics in MPI-IO render ‘slower’ I/O ops.
- Non-blocking semantics of ADIOS2 facilitate ‘faster’ I/O ops.

32 racks are 524’288 cores and MPI ranks on the HAWK supercomputer at HLRS.



Reduced foam-extend 4.1 for design of parallel I/O

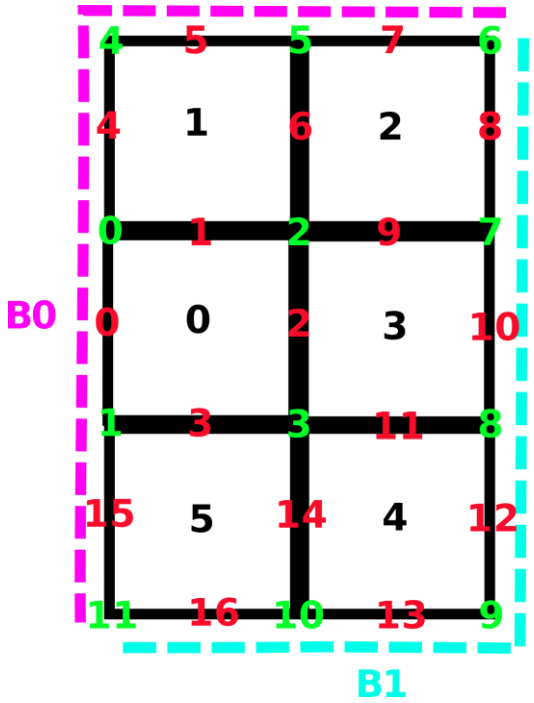
- Native implementation adds ADIOS2 in IOstream family of classes used by registered IOobjects.
- I/O serial in design of insertion (<<) and extraction (>>) operators.
  - Information for ADIOS2 is transferred and buffered during serial call tree of << and >>.
- Refactoring of I/O for polyMesh.
  - Coherent data layout in mesh I/O.
- Refactoring of I/O for GeometricField.
  - Adaptation of coherent data layout.
  - Different offsets of volume, surface, and point fields.

```
bool Foam::GeometricField::writeData(Ostream& os)
{
   os << *this;
   return os.good();
}


Foam::Ostream& Foam::operator<<
(
    Ostream& os,
    const GeometricField gf
)
{
    // Logic for serial uncollated I/O
    // Refactored for additions with parallel I/O
}

/* Somewhere in Ostream derived source */
…
    adiosStream.open(file);
    adiosStream.write(identifier, start, size, data);
    adiosStream.close();
…
```

ADIOS2 implementation in core (libfoam) library.

Coherent and sliceable data layout of the mesh.

- Upper triangular order/sorted by ownership incorporating boundary faces.
- Face-ordered ‘owner’ list → ‘ownerStart’ as cell-ordered list as entry point for coherence.
- Partitioning into slices of **cell-ordered lists**.
- Owners determine slices in **face-ordered lists**.
- Slices of **point-ordered lists** known by maximum ID in faces.

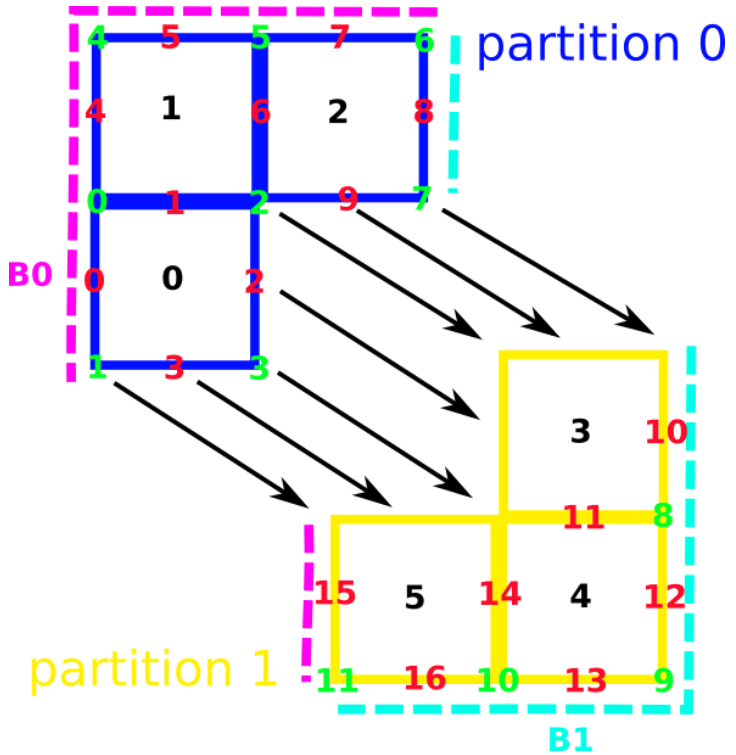


Decomposition		Ownership		Neighbourhood		Mesh Entities		
partition	partitionStart	cell	ownerStart	face	neighbour	faces	point	points
0	0	0	0	0	-1	(1 0)	0	(0.0 0.0)
1	3	1	4	1	1	(0 2)	1	(0.0 -0.1)
end	6	2	7	2	3	(2 3)	2	(0.1 0.0)
		3	10	3	5	(3 1)	3	(0.1 -0.1)
		4	12	4	-1	(0 4)	4	(0.0 0.1)
		5	15	5	-1	(4 5)	5	(0.1 0.1)
		end	17	6	2	(5 2)	6	(0.2 0.1)
				7	-1	(5 6)	7	(0.2 0.0)
				8	-2	(6 7)	8	(0.2 -0.1)
				9	3	(7 2)	9	(0.2 -0.2)
				10	-2	(7 8)	10	(0.1 -0.2)
				11	4	(8 3)	11	(0.0 -0.2)
				12	-2	(8 9)		
				13	-2	(9 10)		
				14	5	(10 3)		
				15	-1	(11 1)		
				16	-2	(10 11)		

max. ID

Coherent and sliceable data layout of the mesh.

- Upper triangular order/sorted by ownership incorporating boundary faces.
- Face-ordered ‘owner’ list → ‘ownerStart’ as cell-ordered list as entry point for coherence.
- Partitioning into slices of **cell-ordered lists**.
- Owners determine slices in **face-ordered lists**.
- Slices of **point-ordered lists** known by maximum ID in faces.

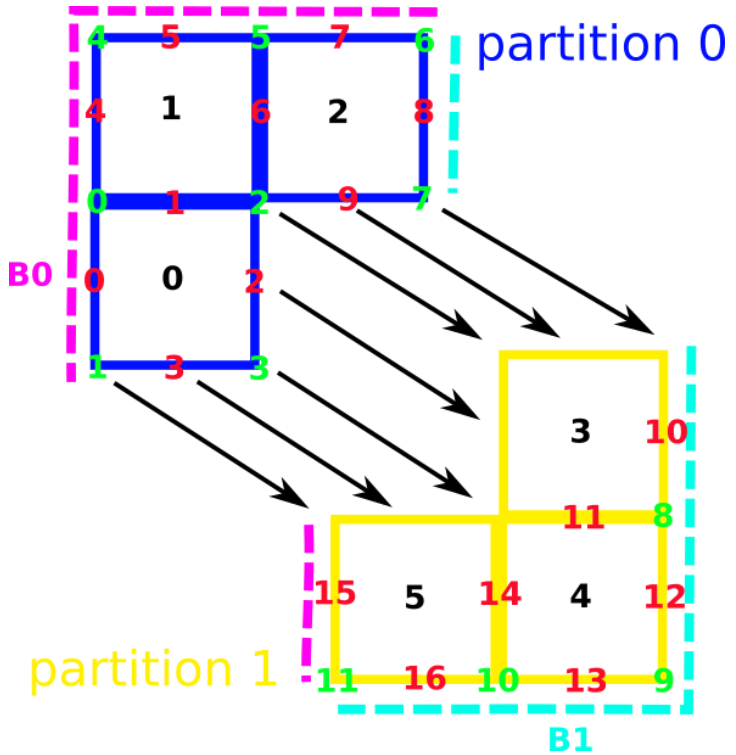


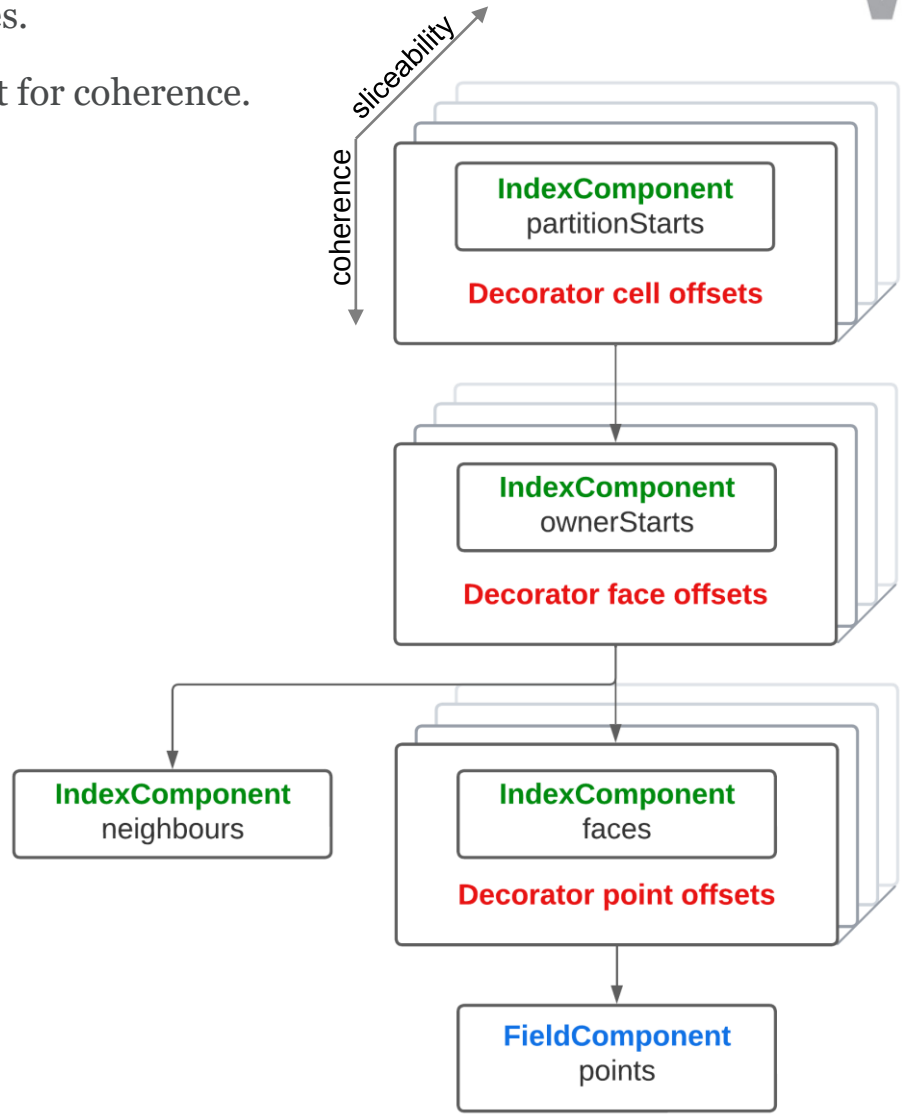
Decomposition		Ownership		Neighbourhood		Mesh Entities		
partition	partitionStart	cell	ownerStart	face	neighbour	faces	point	points
0	0	0	0	0	-1	(1 0)	0	(0.0 0.0)
1	3	1	4	1	1	(0 2)	1	(0.0 -0.1)
end	6	2	7	2	3	(2 3)	2	(0.1 0.0)
		3	10	3	5	(3 1)	3	(0.1 -0.1)
		4	12	4	-1	(0 4)	4	(0.0 0.1)
		5	15	5	-1	(4 5)	5	(0.1 0.1)
		end	17	6	2	(5 2)	6	(0.2 0.1)
				7	-1	(5 6)	7	(0.2 0.0)
				8	-2	(6 7)	8	(0.2 -0.1)
				9	3	(7 2)	9	(0.2 -0.2)
				10	-2	(7 8)	10	(0.1 -0.2)
				11	4	(8 3)	11	(0.0 -0.2)
				12	-2	(8 9)		
				13	-2	(9 10)		
				14	5	(10 3)		
				15	-1	(11 1)		
				16	-2	(10 11)		

max. ID

Coherent and sliceable data layout of the mesh.

- Upper triangular order/sorted by ownership incorporating boundary faces.
- Face-ordered ‘owner’ list → ‘ownerStart’ as cell-ordered list as entry point for coherence.
- Partitioning into slices of **cell-ordered lists**.
- Owners determine slices in **face-ordered lists**.
- Slices of **point-ordered lists** known by maximum ID in faces.





Coherent Data Layout for Field I/O

- Field data layout **adopts coherence** of mesh.
- Geometric fields consist of internal field and boundary patches.
  - Enables complex boundary fields.
  - Processor patches are either not present (volume fields) or integrated into the internal field (surface fields).
- **Plain time folders** for field I/O in serial and parallel runs (no processor folders)
  - Binary field data is stored in ADIOS2 files, i.e. data.bp
  - Field metadata stored in ASCII files, e.g. p, phi, U

Case structure serial/parallel run

```
.
|-- 0
|   |-- p
|   `-- U
|-- 0.05
|   |-- data.bp
|   |-- p
|   |-- phi
|   `-- U
|-- 0.1
|   |-- data.bp
|   |-- p
|   |-- phi
|   `-- U
|-- constant
|   |-- polyMesh.bp
|   `-- transportProperties
...
```

Significantly reduced time to solution.

Benefits

1. Reduction of inodes:
   - Exascale ready I/O strategy.
   - Flexible aggregation pattern at runtime.
2. Simplicity from coherent data layout:
   - Start a parallel run (without decomposition) on any number of cores.
   - No reconstruction for post-processing.
   - Edit ASCII metadata like boundary conditions in one place.

Work on your parallel case as if it was a serial case.

Demo.

Case structure serial/parallel run

```
.
|-- 0
|   |-- p
|   `-- U
|-- 0.05
|   |-- data.bp
|   |-- p
|   |-- phi
|   `-- U
|-- 0.1
|   |-- data.bp
|   |-- p
|   |-- phi
|   `-- U
|-- constant
|   |-- polyMesh.bp
|   `-- transportProperties
...
```

Conclusion

- Developed a new design for high performance parallel I/O.
- Implementation in the core (libfoam) of foam-extend 4.1.

1. Reduction of inodes.
    - Lower load on parallel file systems.
2. Coherent data layout simplifies pre- and post-processing of parallel cases.
    - Simplified user experience at all scales.
    - Significant reduction of time-to-solution.

Outlook

- Detailed benchmarks.
- Release into production code.

Thank you.

exafoam-project

exafoamproject

https://exafoam.eu/

Questions?

Presenter: R. Gregor Weiß
gregor.weiss@hlrs.de

SPONSORED BY THE

Federal Ministry of Education and Research

anr agence nationale de la recherche

Ministero dello Sviluppo Economico

REPUBLIKA HRVATSKA Ministarstvo znanosti i obrazovanja

GSRI GENERAL SECRETARIAT FOR RESEARCH AND INNOVATION

FCT Fundação para a Ciência e a Tecnologia

EuroHPC Joint Undertaking

This project has received funding from the European High-Performance Computing Joint Undertaking Joint Undertaking (JU) under grant agreement No 956416. The JU receives support from the European Union's Horizon 2020 research and innovation programme and France, United Kingdom, Germany, Italy, Croatia, Spain, Greece, and Portugal.