arm

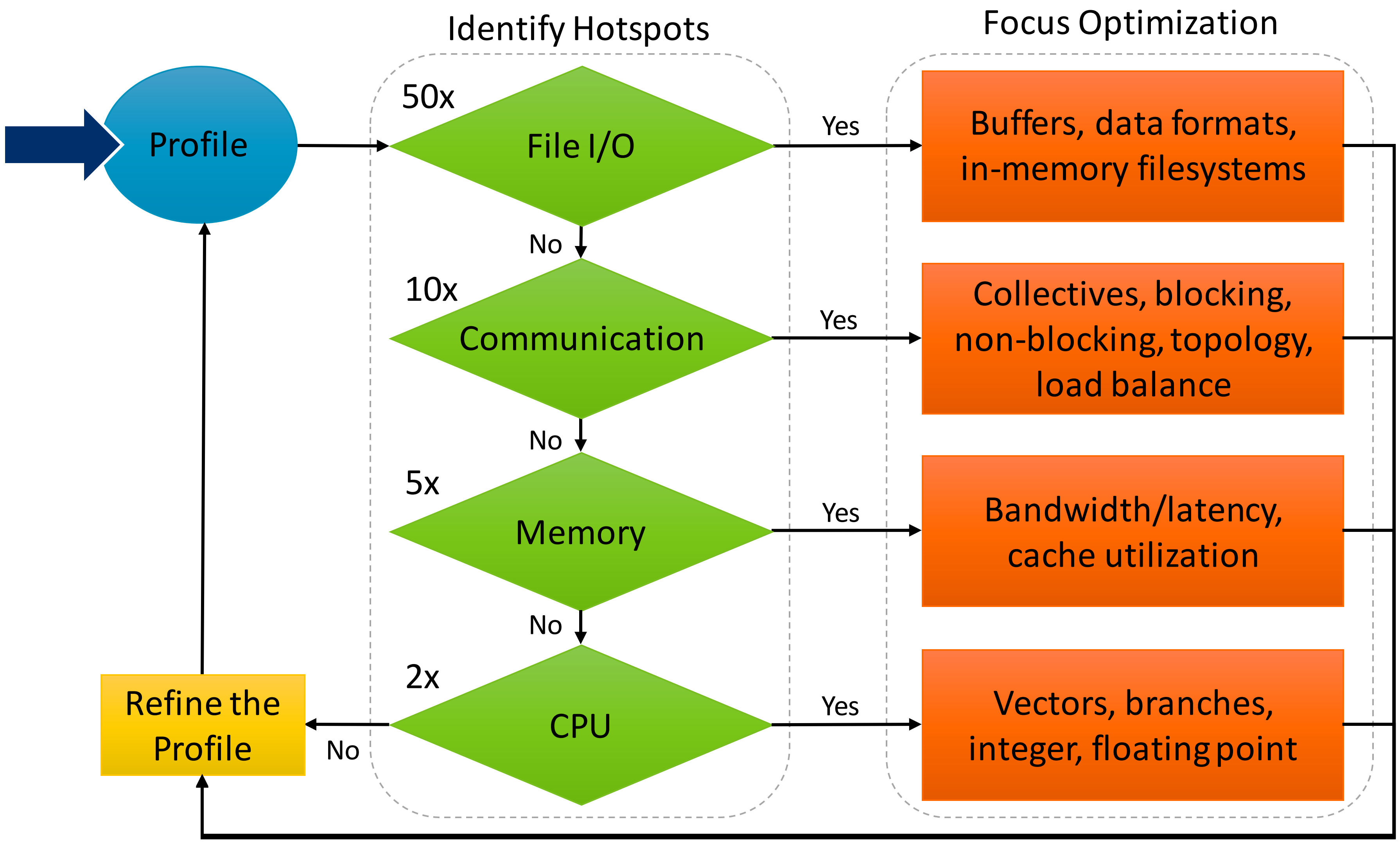
Resolving Inefficiencies in Complex I/O

12 July 2018

John C. Linford, Florent Lebeau,
Keeran Brabazon, Olly Perks, et al.

Iteratively identify and resolve performance issues

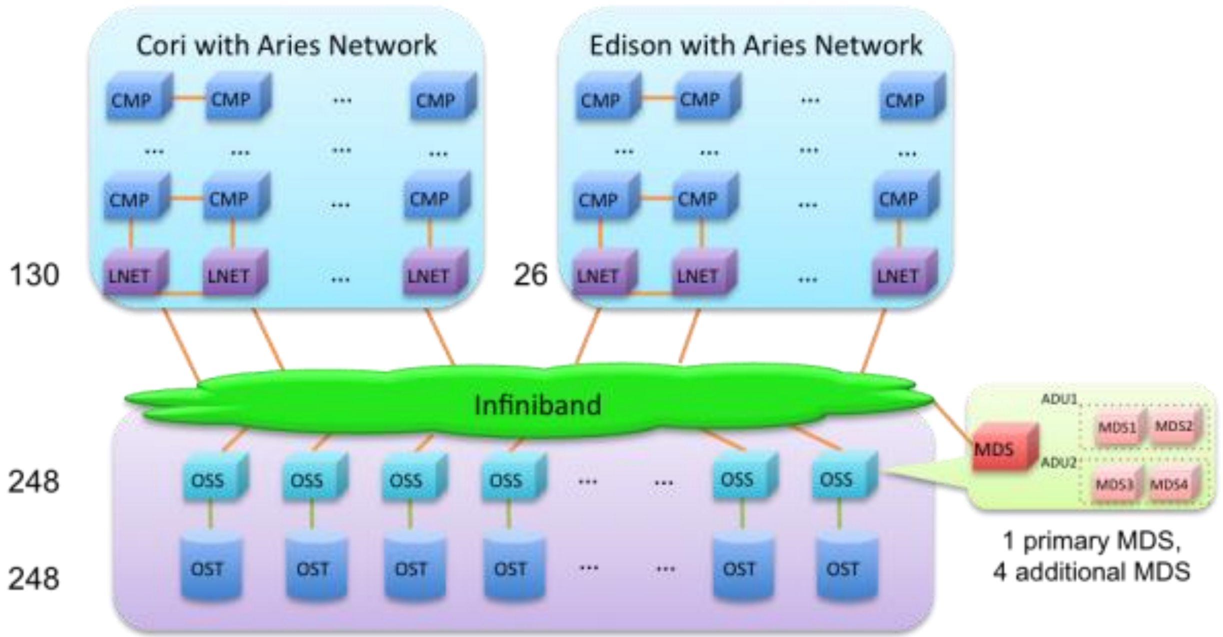
Profiling is central to understanding and improving application performance.



What is high performance I/O?

Complex data movement system optimized for parallel computing.

- Looks like a normal filesystem, but data are distributed over thousands of drives.
- **Latency**: moving data requires multiple network hops.
  - Avoid small sequential operations.
- **Bandwidth**: can perform many I/O operations in parallel.
  - Prefer parallel block-sized operations.
- **Complexity**: performance may depend on many non-obvious factors.
  - Use portable tools to investigate I/O performance.



Each OSS controls one OST. The Infiniband connects the MDS, ADUs and OSSs to the LNET routers on the Cray XC System. The OSTs are configured with GridRAID, similar to RAID6, (8+2), but can restore failure 3.5 times faster than traditional RAID6. Each OST consists of 41 disks, and can deliver 240TB capacity.

Credit: NERSC

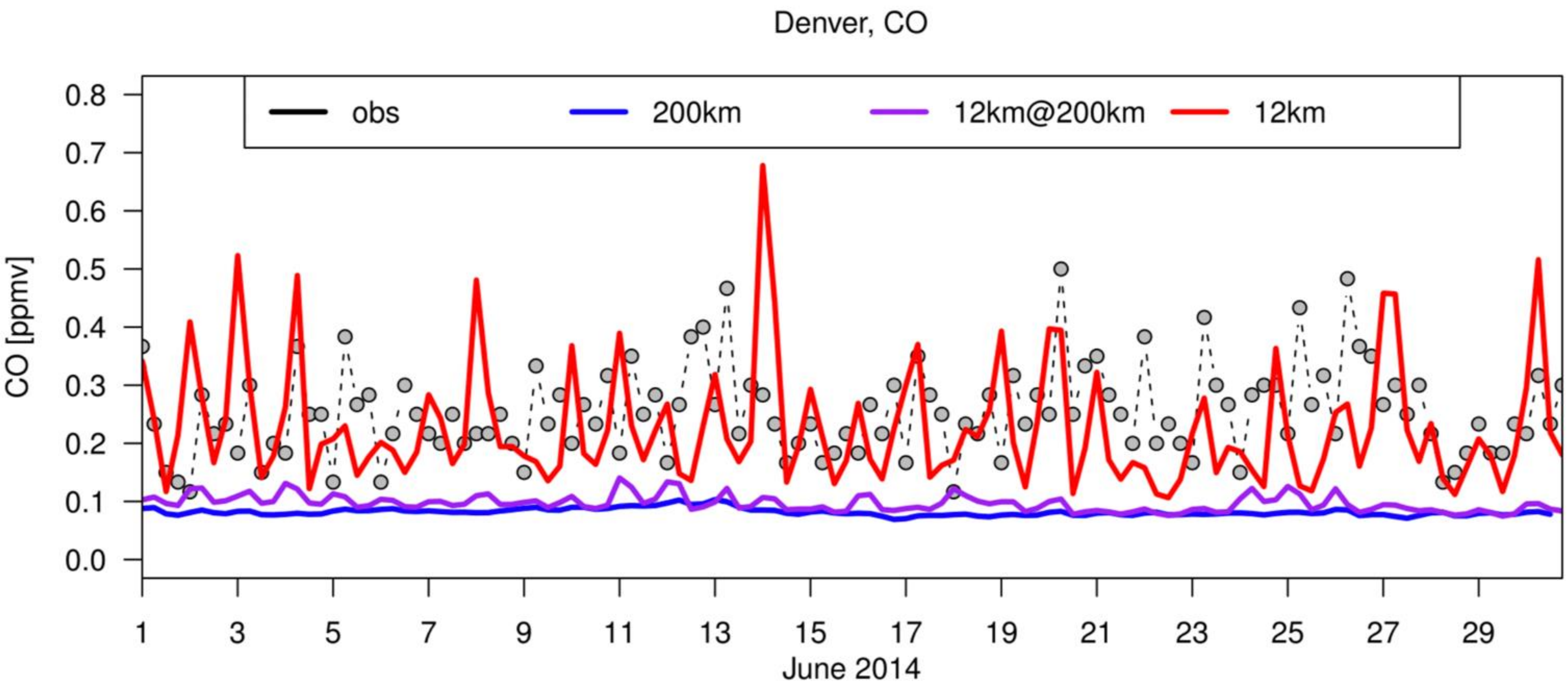
Why does I/O have such a huge impact on performance?

I/O has the potential to make or break the performance of the whole system.

- A shared resource on practically all HPC systems.
  - Bandwidth to disk is shared between processes.
  - Bandwidth to network is shared between nodes.
- Has the potential to affect the performance of other users' jobs.
  - Data are physically located outside the compute node.
  - Using shared I/O outside the compute node has an impact on the performance of other users' jobs.
  - Even if other users are not using the shared filesystem, communicating with the filesystem over the network can affect other user’s inter-node communications (e.g. MPI).
- The slowest tier of the memory hierarchy.
  - Small mistakes in I/O will cost you more than huge mistakes at higher tiers, e.g. cache.
  - Simple, low effort optimizations in filesystem I/O will pay out more than high effort optimizations at higher tiers.

Reduction isn't an option: have to optimize I/O

Models require high resolutions to accurately describe physical conditions.

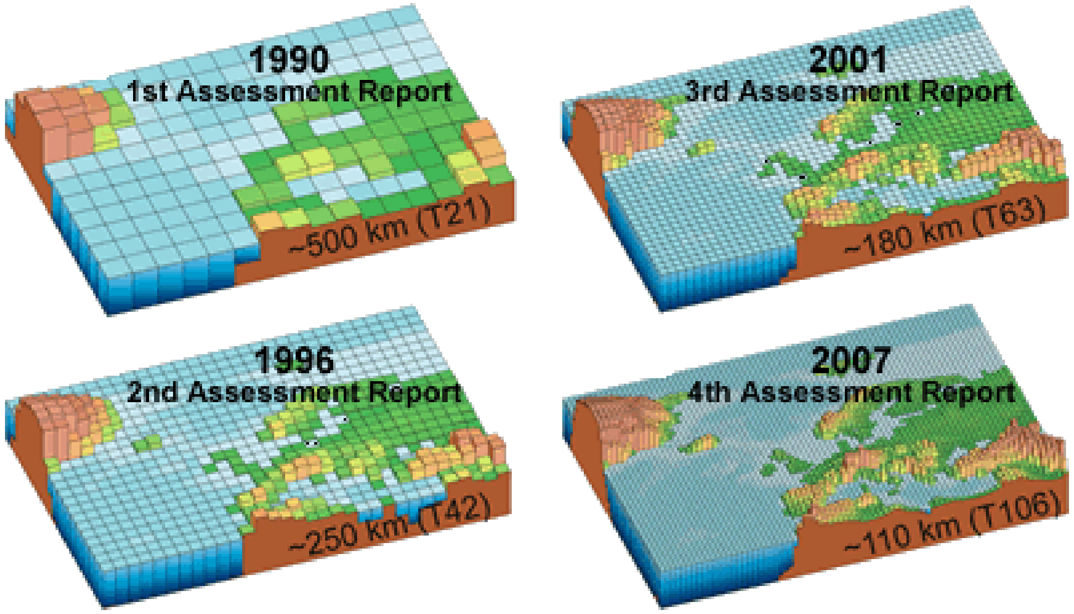


Credit: [NASA GMAO, Christoph Keller.]()

Data drives high performance computing

Many HPC applications are dominated by I/O, and I/O requirements are growing.

- Applications driven by datasets
  - High resolution models, and getting higher.
  - Applications often have low FLOPS/byte.
- Checkpoint restarts
  - Periodic dumps of state to filesystem.
  - Resilience, reproducibility, history, etc.
- Visualizations
  - Classical pre-process / process / post-process workflow is still prevalent.
  - Snapshots of in-situ post-processing.



Typical spatial resolution used in state-of-the-art climate models around the times of each of the four IPCC Assessment Reports. Credit: UCAR and the IPCC.

Understand your I/O system

Use portable, cross-platform tools and libraries.

- Storage systems host filesystems
  - Lustre, GPFS, BeeGFS: POSIX-compliant block storage designed for scalability.
  - Ceph: Object storage, block storage, and POSIX-compliant filesystem.
- Infrastructure hosts storage systems
  - The network fabric connects all compute nodes in a predefined (physically hard wired) topology.
  - I/O nodes serve multiple compute nodes (potential bottleneck)
- Infrastructure can be optimized for HPC
  - Small local (i.e. non-shared) filesystems, possibly in memory (e.g. /dev/shm)
  - Burst buffers
  - NVDIMMS.

Understand how your application uses the I/O system

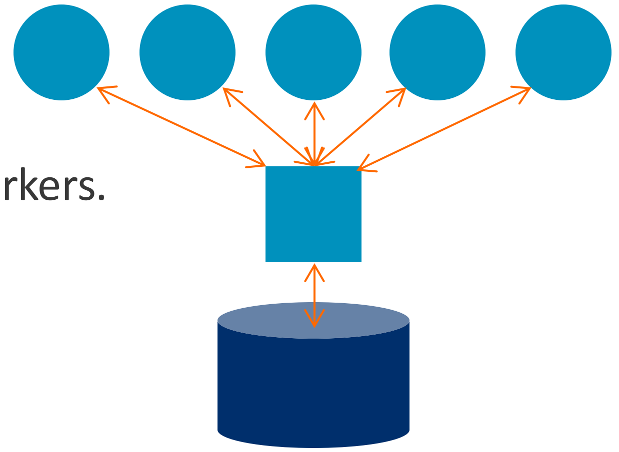
You have the greatest control over your application's behavior.

- I/O Characteristics
  - How many reads vs. how many writes?
  - Data access pattern: sequential, aligned, random?
  - I/O in bursts? Streaming I/O?
- I/O Operations
  - Standard library calls: fopen, fread, fwrite
  - MPI-IO calls: MPI_File_open, MPI_File_write, MPI_File_close
  - I/O library: HDF5, NetCDF, ADIOS, …
- Non-I/O communication that may influence I/O performance
  - Communication-heavy application phases.
  - Inter-node data movement to prepare for I/O.

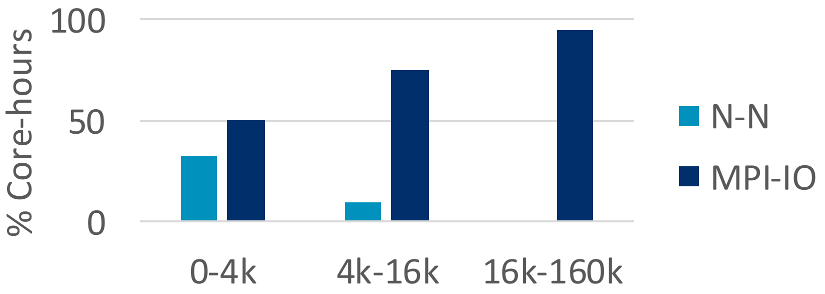
Simple approaches to parallel I/O

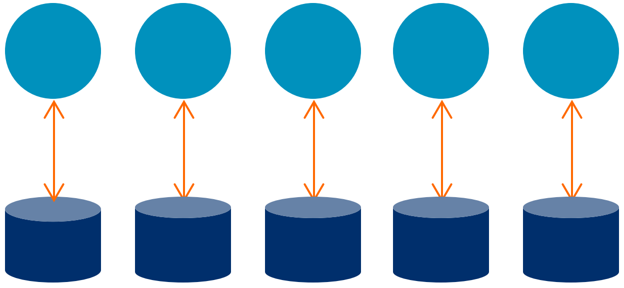
Simple approaches work for small applications, but typically don't scale.

- 1 – 1: Master and workers
  - A master process performs I/O on behalf of many workers.
  - Collective operations (e.g. MPI_Gather, MPI_Scatter) move data to/from workers.
  - Performance bottleneck at the master.



- N – N: Every process for itself
  - Each process reads/writes it's own data in a uniquely named file.
  - Large number of open files can quickly degrade performance.



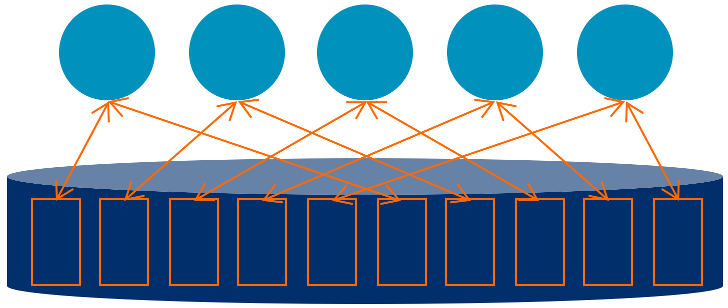


Credit: Argonne National Lab

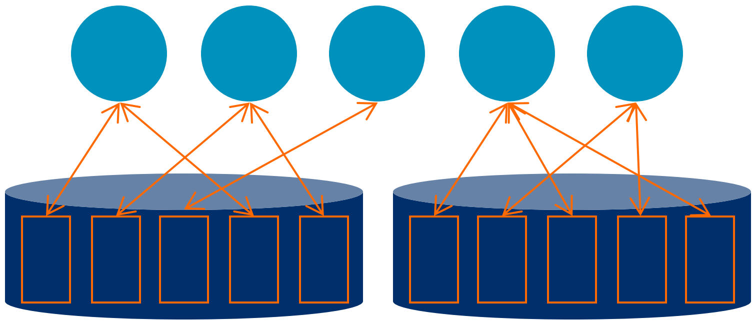
Treating parallel I/O like shared memory

Use a library like MPI-IO or HDF5 for optimal portability and performance.

- N – 1: Multiple writers to same resource
  - Many processes read/write to the same resource, e.g. a file.
  - Files broken up in to lock units; boundaries determined by system.
  - Clients must obtain locks before performing I/O.
  - Enables caching: as long as client holds the lock the cache is valid.



- N – M: Cooperating gangs
  - Groups of processes combine to operate on shared resources.
  - Mirroring physical hardware infrastructure can improve performance.
  - Implementation best left to the libraries.
  - Balance gang size against available bandwidth.



Arm Forge

An interoperable toolkit for debugging and profiling

Commercially supported by Arm

Fully Scalable

Very user-friendly

The de-facto standard for HPC development

- Available on the vast majority of the world's top supercomputers
- Fully supported by Arm on x86, IBM Power, Nvidia GPUs, etc.

State-of-the art debugging and profiling capabilities

- Powerful and in-depth error detection mechanisms (including memory debugging)
- Sampling-based profiler to identify and understand bottlenecks
- Available at any scale (from serial to petaflopic applications)

Easy to use by everyone

- Unique capabilities to simplify remote interactive sessions
- Innovative approach to present quintessential information to users

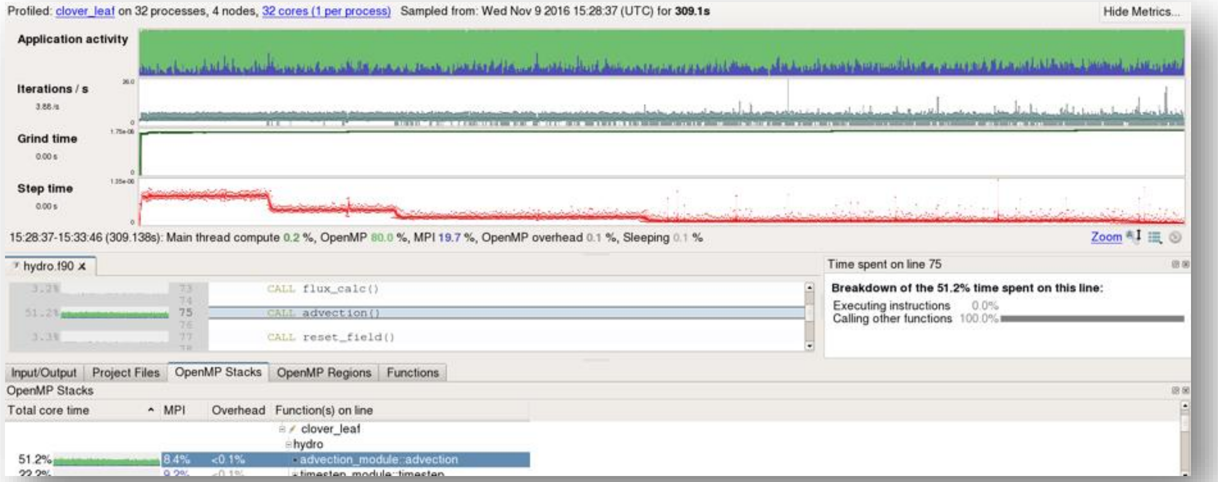
Optimize the application

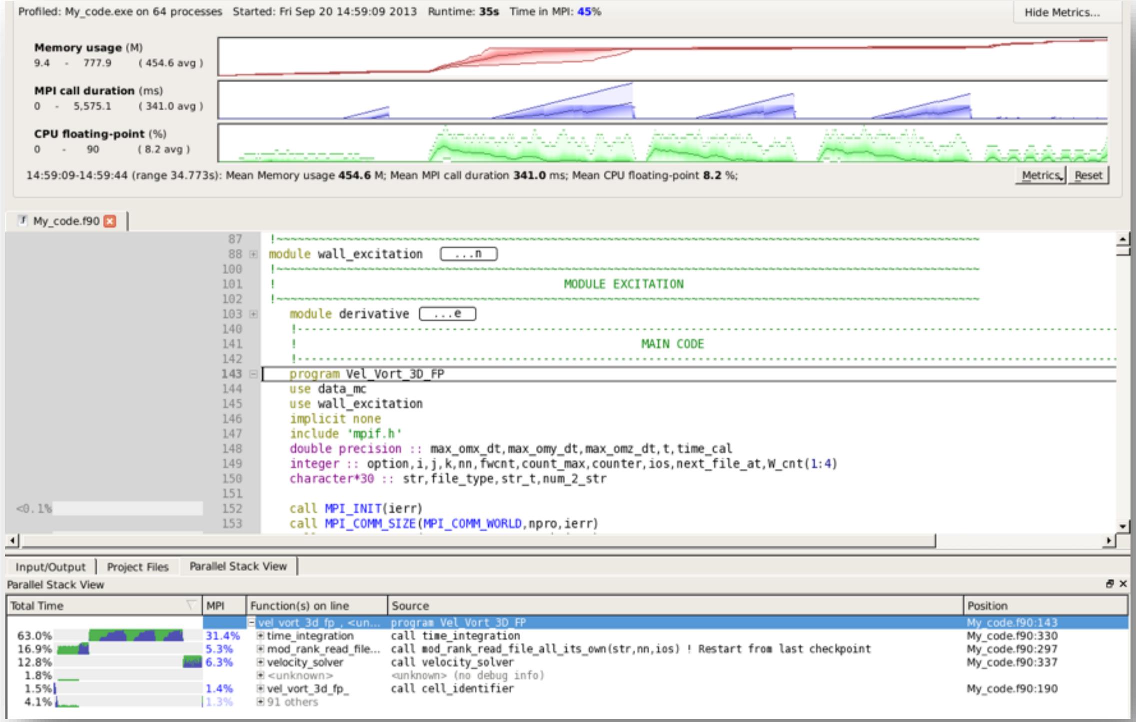
Identify bottlenecks and rewrite some code for better performance

- Run with the representative workload you started with
- Measure all performance aspects with **Arm Forge Professional**

Examples:

```
$> map -profile mpirun -n 48 ./example
```





arm

Arm Performance Reports

Characterize and understand the performance of HPC application runs

Commercially supported by Arm

Accurate and astute insight

Relevant advice to avoid pitfalls

Gathers a rich set of data

- Analyzes metrics around CPU, memory, IO, hardware counters, etc.
- Possibility for users to add their own metrics

Build a culture of application performance & efficiency awareness

- Analyzes data and reports the information that matters to users
- Provides simple guidance to help improve workloads' efficiency

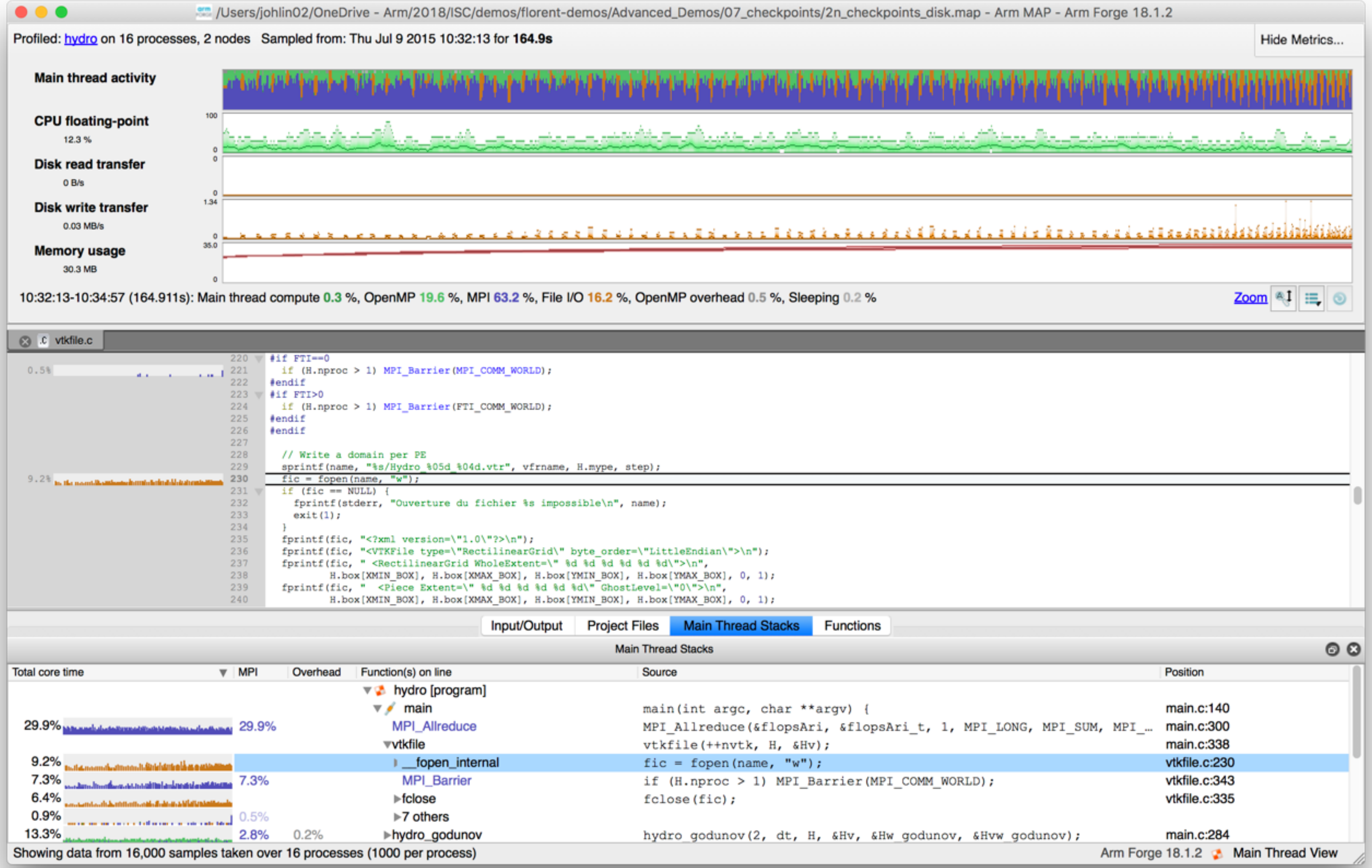
Adds value to typical users' workflows

- Define application behaviour and performance expectations
- Integrate outputs to various systems for validation (e.g. continuous integration)
- Can be automated completely (no user intervention)

arm

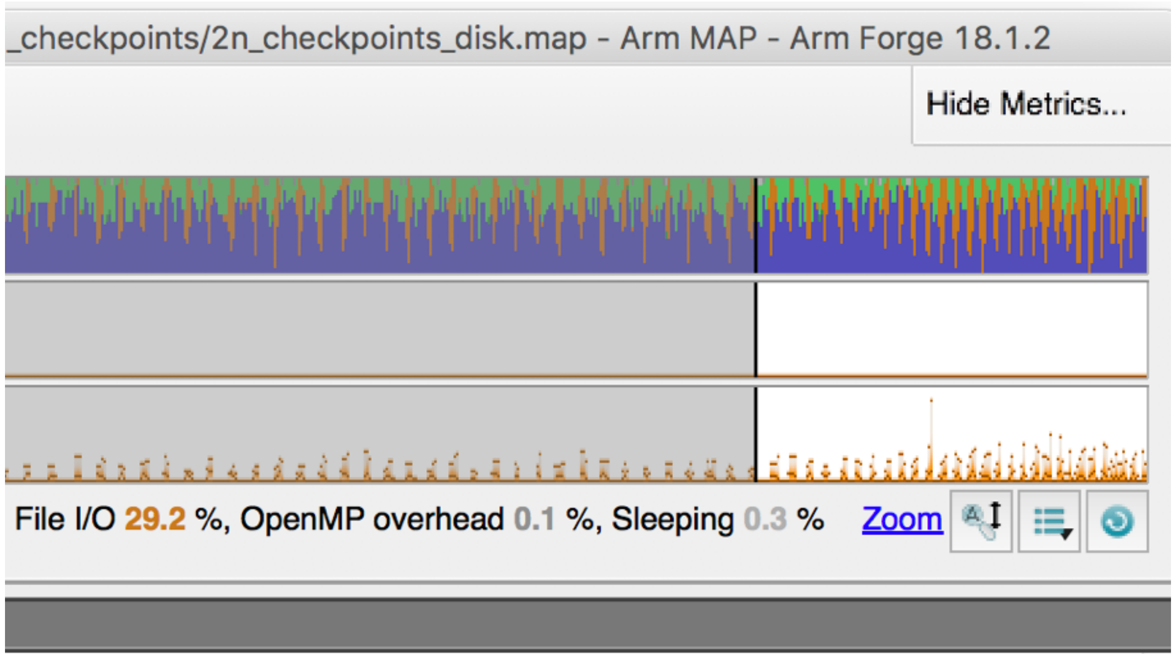
Initial profile shows 9.2% of runtime spent just opening files

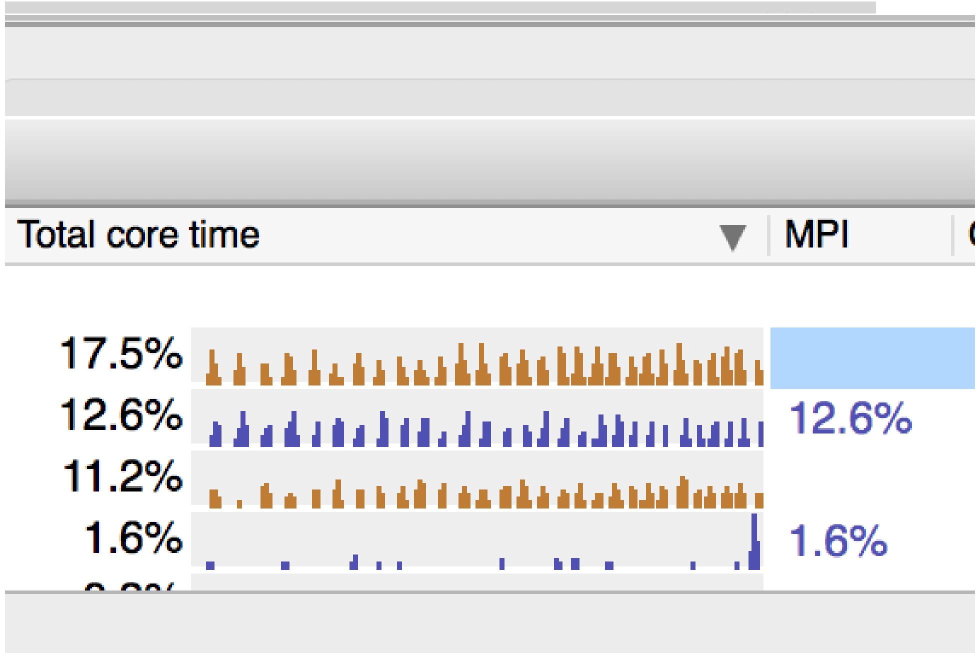
16.2% of runtime is I/O, but only 5% is spent in read/write operations.



Focusing on hotspot shows almost 30% of runtime in I/O

File open and close operations are very expensive on this filesystem.

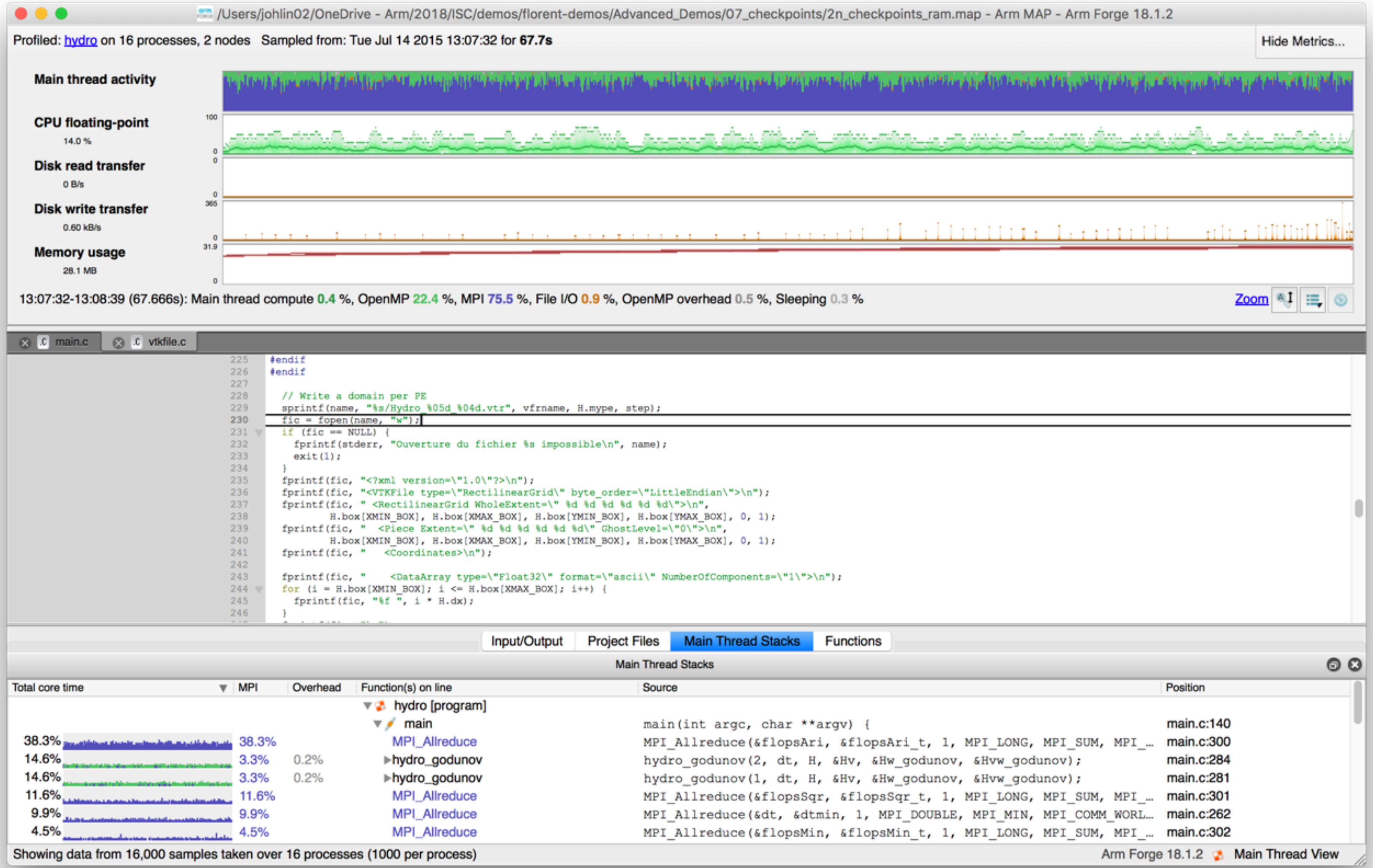




- Intermediate files for visualization are being written to disk.
- Fix: write intermediate files to an in-memory filesystem, e.g. /dev/shm.

Easy fix: write intermediate files to /dev/shm

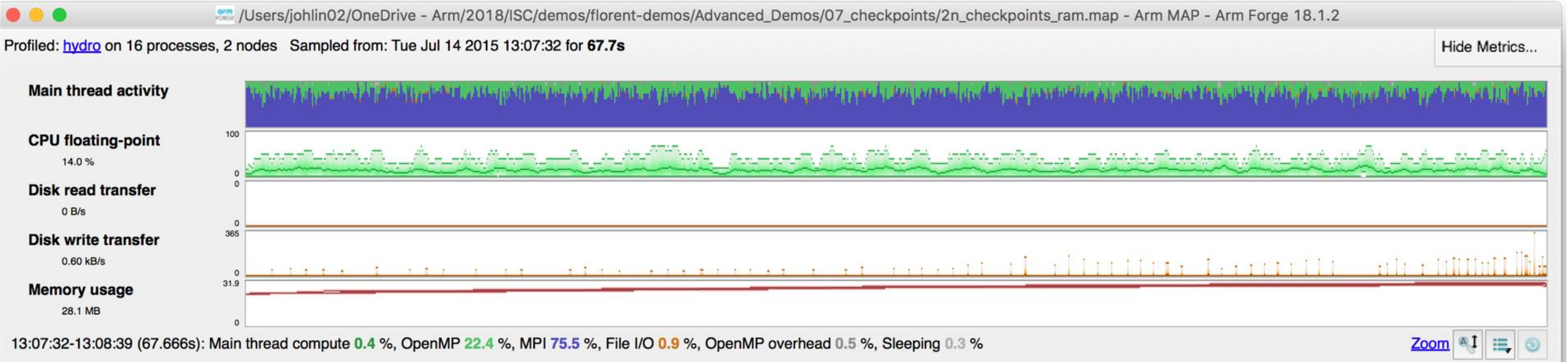
Writing temporary files to in-memory filesystem can dramatically improve performance.



arm

After fix, only 0.9% of runtime spent in I/O

Writing temporary files to in-memory filesystem can dramatically improve performance.



Total core time	MPI	Overhead	Function(s) on line
			hydro [program]
			main
38.3%	38.3%		MPI_Allreduce
14.6%	3.3%	0.2%	hydro_godunov
14.6%	3.3%	0.2%	hydro_godunov
11.6%	11.6%		MPI_Allreduce
9.9%	9.9%		MPI_Allreduce
4.5%	4.5%		MPI_Allreduce

Showing data from 16,000 samples taken over 16 processes (1000 per process)

Arm Performance Reports

High-level view of application performance shows low write rate.

arm PERFORMANCE REPORTS

Command:	mpiexec.hydra -host node-1,node-2 -map-by socket -n 16 -ppn 8 ./Bin/low_freq/../../Src//hydro -i ./Bin/low_freq/../../../../Input/input_250x125_corner.nml
Resources:	2 nodes (8 physical, 8 logical cores per node)
Memory:	15 GiB per node
Tasks:	16 processes, OMP_NUM_THREADS was 1
Machine:	node-1
Start time:	Thu Jul 9 2015 10:32:13
Total time:	165 seconds (about 3 minutes)
Full path:	Bin/../Src

Summary: hydro is MPI-bound in this configuration

Compute	20.6%	Time spent running application code. High values are usually good. This is **very low**; focus on improving MPI or I/O performance first
MPI	63.2%	Time spent in MPI calls. High values are usually bad. This is **high**; check the MPI breakdown for advice on reducing it
I/O	16.2%	Time spent in filesystem I/O. High values are usually bad. This is **average**; check the I/O breakdown section for optimization advice

I/O

A breakdown of the 16.2% I/O time:

Time in reads	0.0%
Time in writes	100.0%
Effective process read rate	0.00 bytes/s
Effective process write rate	1.38 MB/s

Most of the time is spent in write operations with a very low effective transfer rate. This may be caused by contention for the filesystem or inefficient access patterns. Use an I/O profiler to investigate which write calls are affected.

After the fix, write rate has improved 41.6x

Eliminating file open/close bottleneck has dramatically improved I/O performance.

arm PERFORMANCE REPORTS

Command:	mpiexec.hydra -host node-1,node-2 -map-by socket -n 16 -ppn 8 ./Bin/../Src//hydro -i ./Bin/../../../Input/input_250x125_corner.nml
Resources:	2 nodes (8 physical, 8 logical cores per node)
Memory:	15 GiB per node
Tasks:	16 processes, OMP_NUM_THREADS was 1
Machine:	node-1
Start time:	Tue Jul 14 2015 13:07:32
Total time:	68 seconds (about 1 minutes)
Full path:	Src

Summary: hydro is MPI-bound in this configuration

Compute	23.5%	Time spent running application code. High values are usually good. This is **very low**; focus on improving MPI or I/O performance first
MPI	75.5%	Time spent in MPI calls. High values are usually bad. This is **very high**; check the MPI breakdown for advice on reducing it
I/O	0.9%	Time spent in filesystem I/O. High values are usually bad. This is **very low**; however single-process I/O may cause MPI wait times

I/O

A breakdown of the 0.9% I/O time:

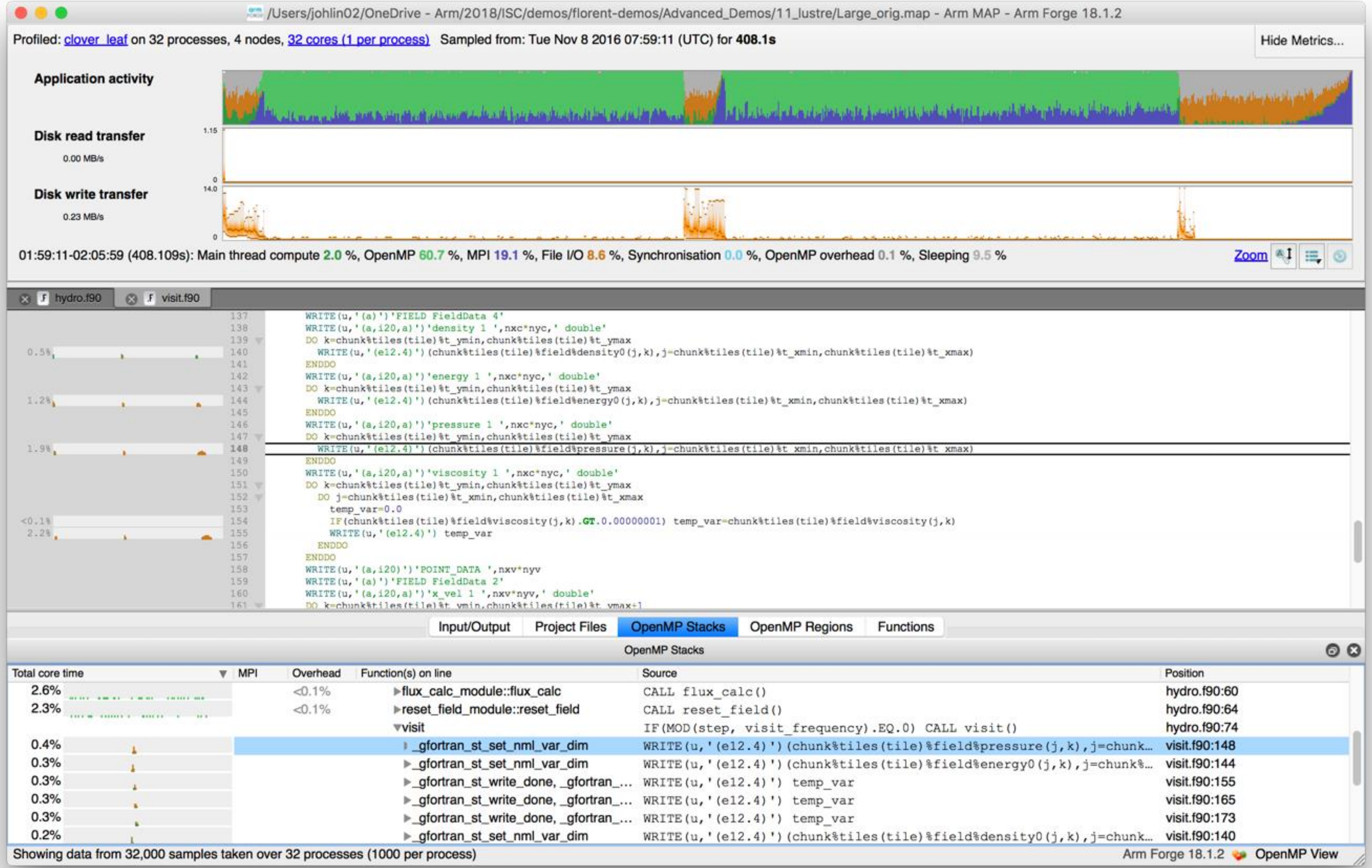
Time in reads	0.0%
Time in writes	100.0%
Effective process read rate	0.00 bytes/s
Effective process write rate	57.5 MB/s

Most of the time is spent in write operations with a low effective transfer rate. This may be caused by contention for the filesystem or inefficient access patterns. Use an I/O profiler to investigate which write calls are affected.

arm

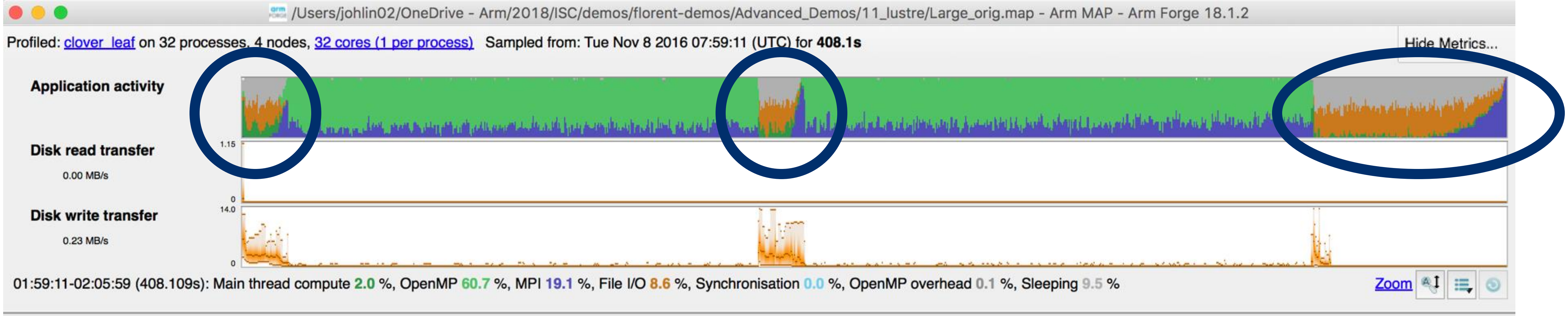
Initial profile of CloverLeaf shows surprisingly unequal I/O

Each I/O operation should take about the same time, but it’s not the case.



Symptoms and causes of the I/O issues

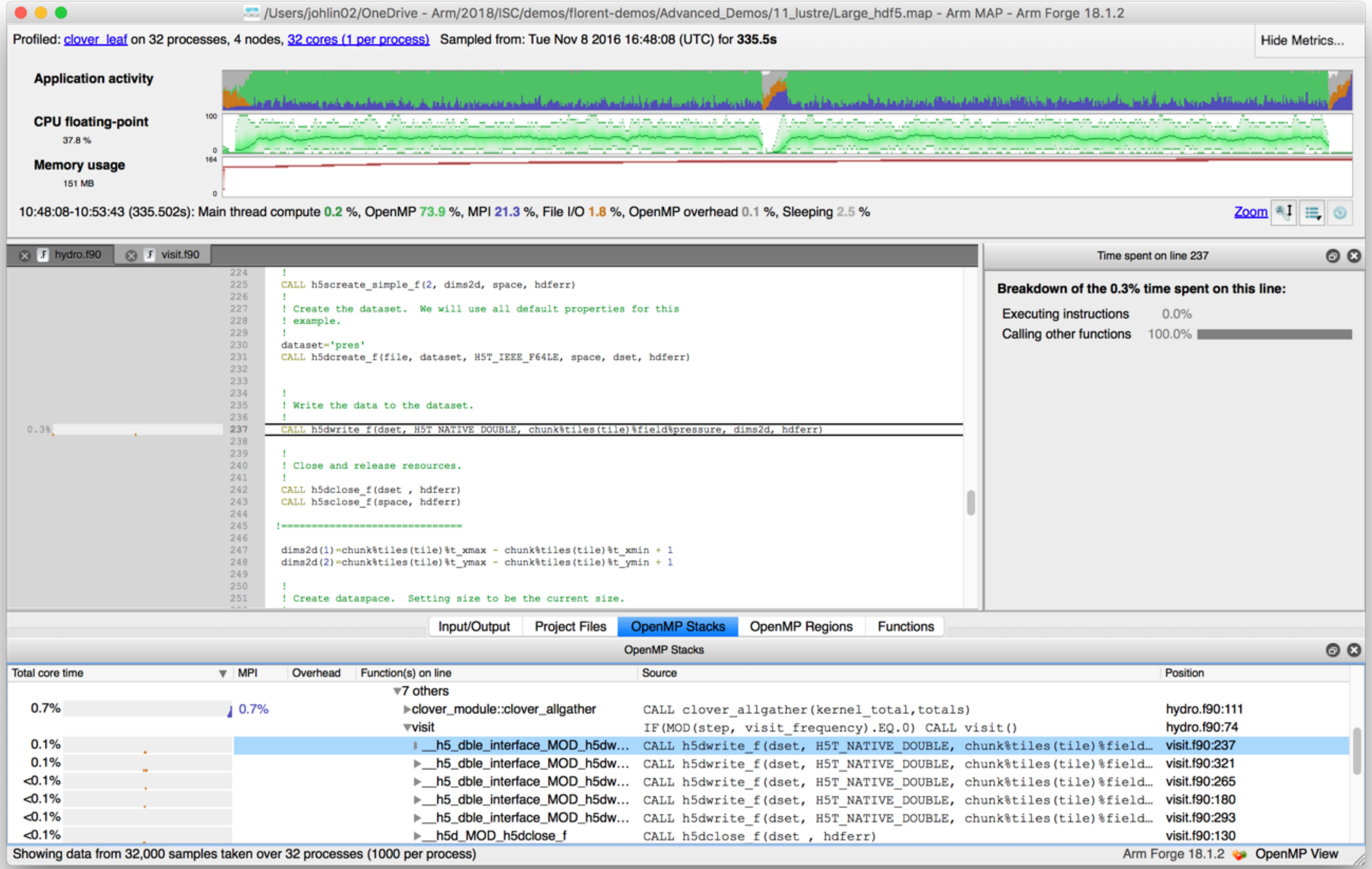
Sub-optimal file format and surprise buffering.



- Write rate is less than 14MB/s.
- Writing an ASCII output file.
- Writes not being flushed until buffer is full.
  - Some ranks have much less buffered data than others.
  - Ranks with small buffers wait in barrier for other ranks to finish flushing their buffers.

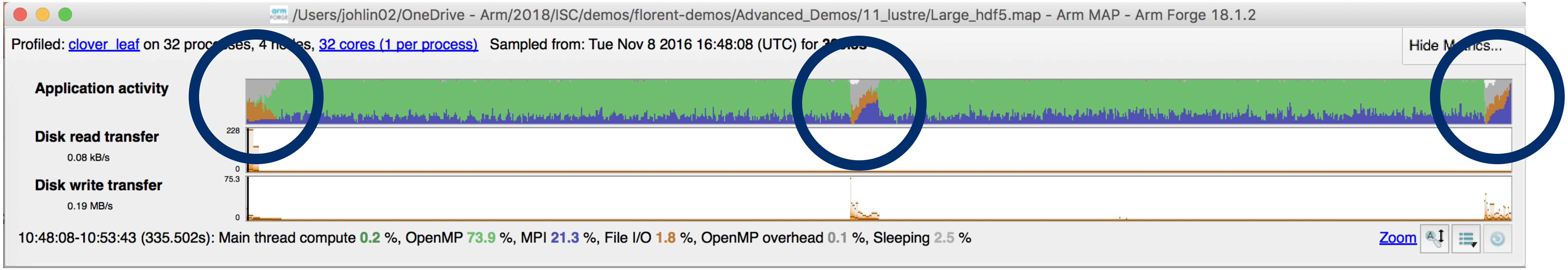
Solution: use HDF5 to write binary files

Using a library optimized for HPC I/O improves performance and portability.



Solution: use HDF5 to write binary files

Using a library optimized for HPC I/O improves performance and portability.

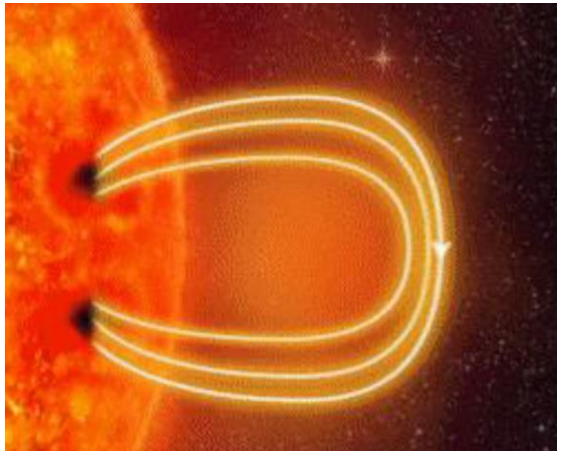


- Replace Fortran write statements with HDF5 library calls.
  - Binary format reduces write volume and can improve data precision.
  - Maximum transfer rate now 75.3 MB/s, over 5x faster.
- Note MPI costs (blue) in the I/O region, so room for improvement.

Advanced I/O investigation of Lustre on Archer

Simultaneously view system-level and application-level performance.

- Show data from Lustre client logs along with application data
- iPIC3D: kinetic simulation of plasma
  - Fully 3D implicit particle-in-cell (PIC)
  - C++ and MPI
  - Intermediate simulation results saved in VTK binary files, single file per quantity
  - Checkpointing done through HDF5 to individual files per process
  - Field values saved using collective MPI-IO to single file



Available performance data

Use MAP's ability to measure filesystem performance at the system and application levels

System level performance data

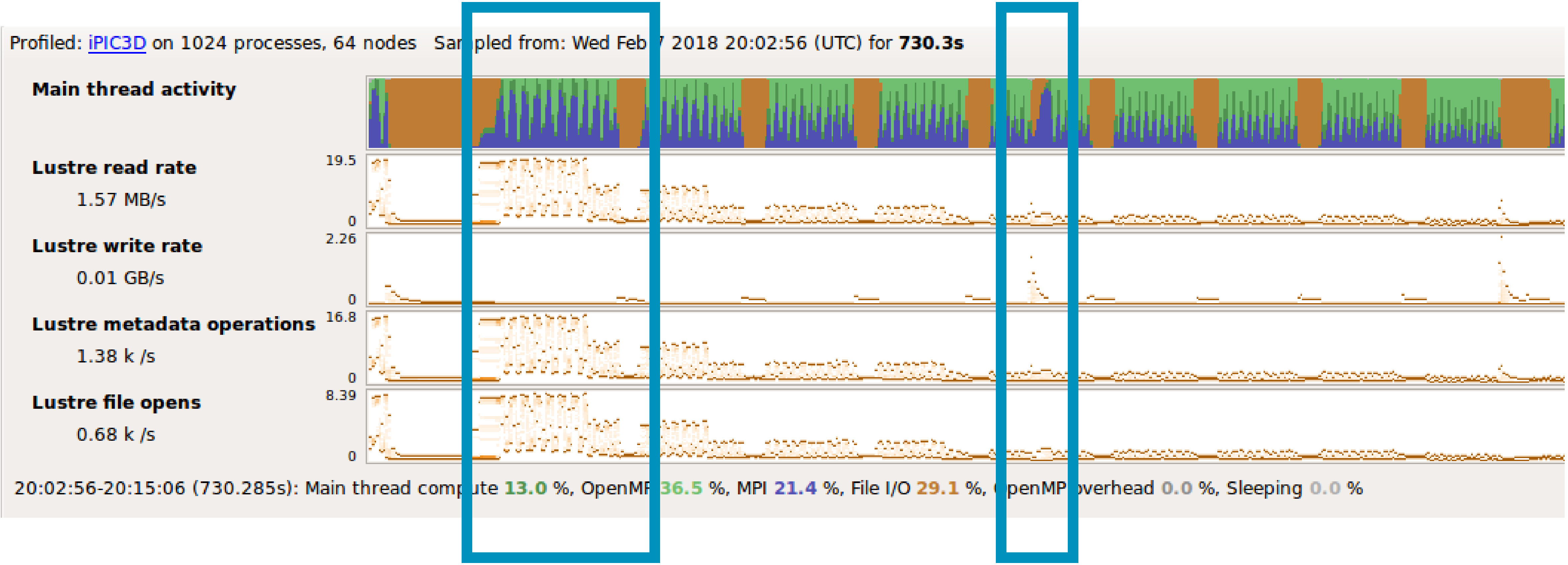
- Lustre logs: each read, write, or metadata operation recorded from each Lustre client.
- Aggregate I/O data for precise bandwidth figures for read/write at any moment in time.
- Max/min/mean bandwidth.
- Scheduler logs: application run start and end time and assigned nodes.

Application level performance data

- Approximate I/O bandwidth in a timeline.
- Approximate classification of I/O instructions (methods).
- In block-synchronous approach, it is possible to identify different I/O phases.

MAP aligns the system timeline with the application timeline

Lustre data is read from the lustre client's log files, while application data is read directly.



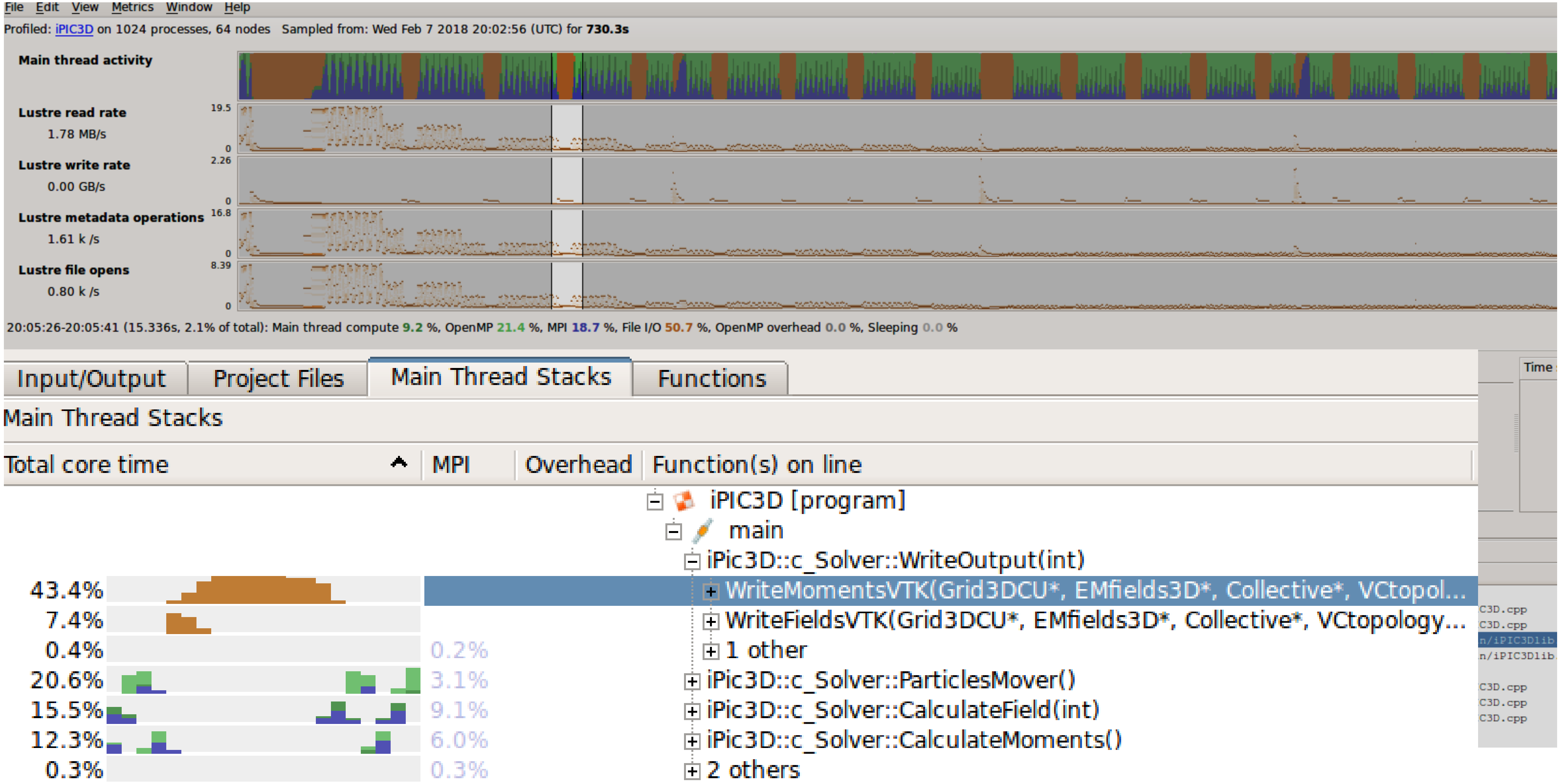
N-N file read shows spike in file open/read operations.

Checkpoint I/O corresponds to spike in Lustre write rate

arm

We can focus on each I/O operation individually

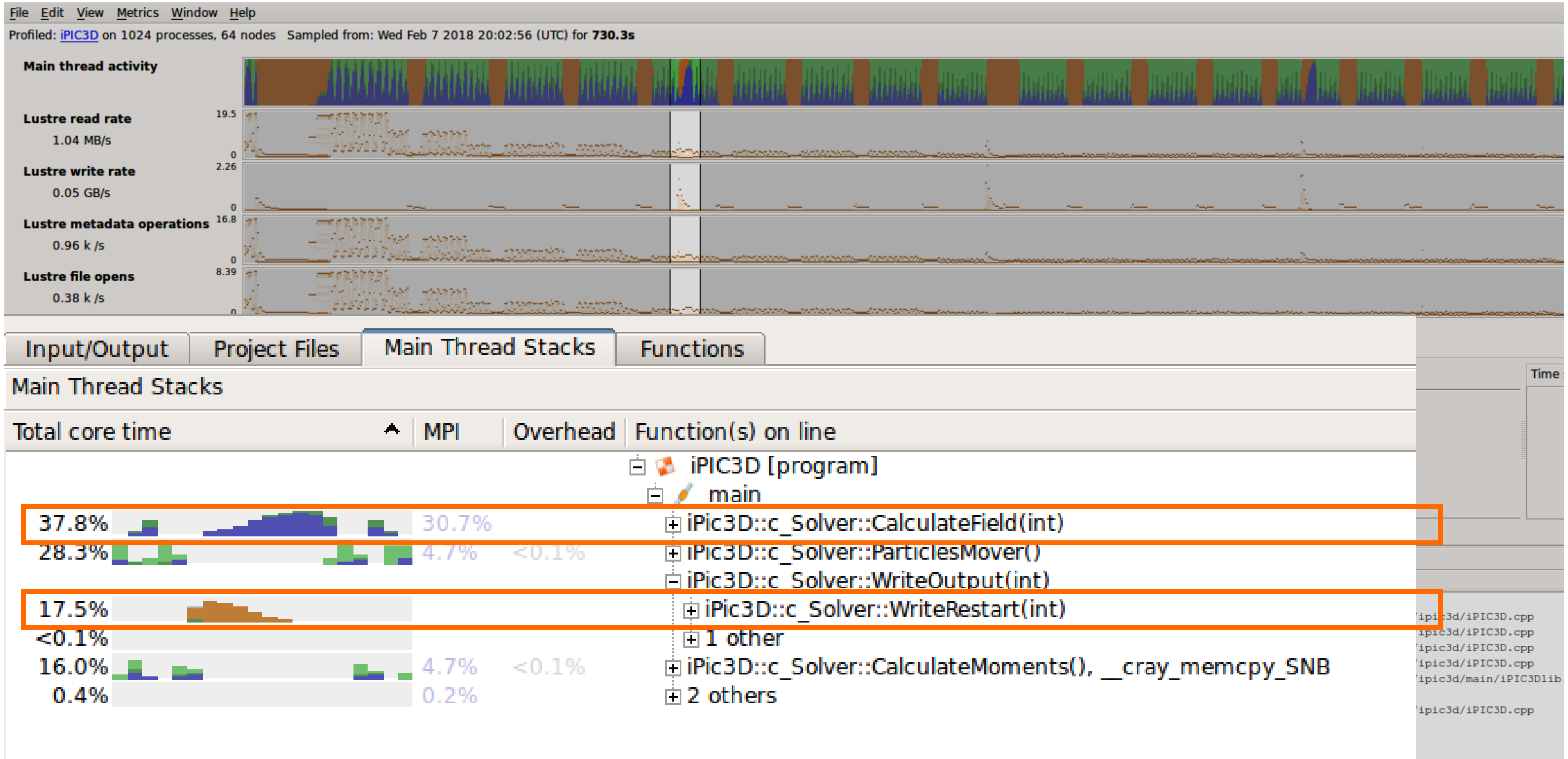
Select a portion of the application timeline to view the source code performing I/O.



arm

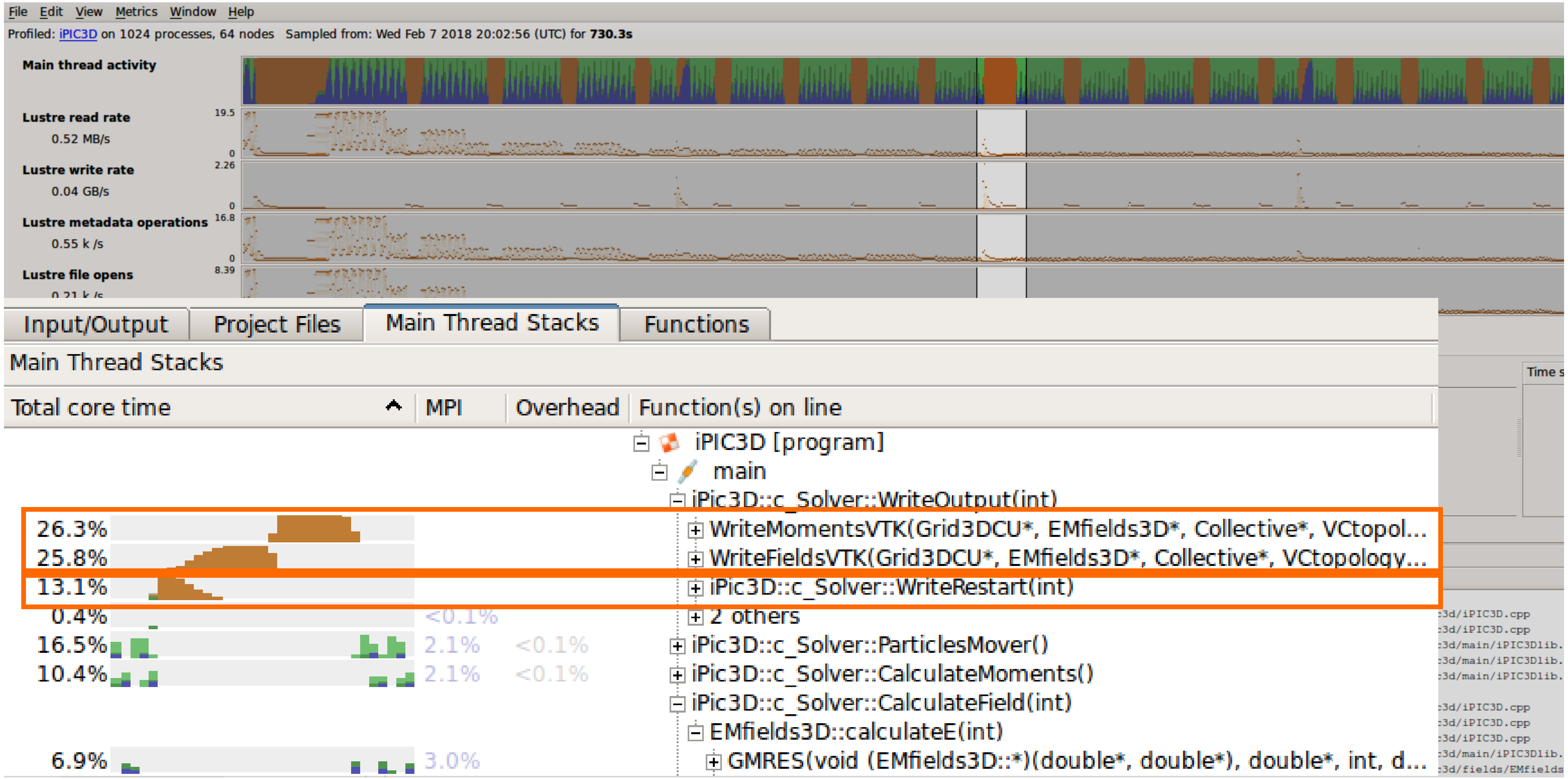
MAP’s timeline shows I/O overlapping with communication

We see elevated Lustre write rate when writing checkpoint restart files in HDF5.



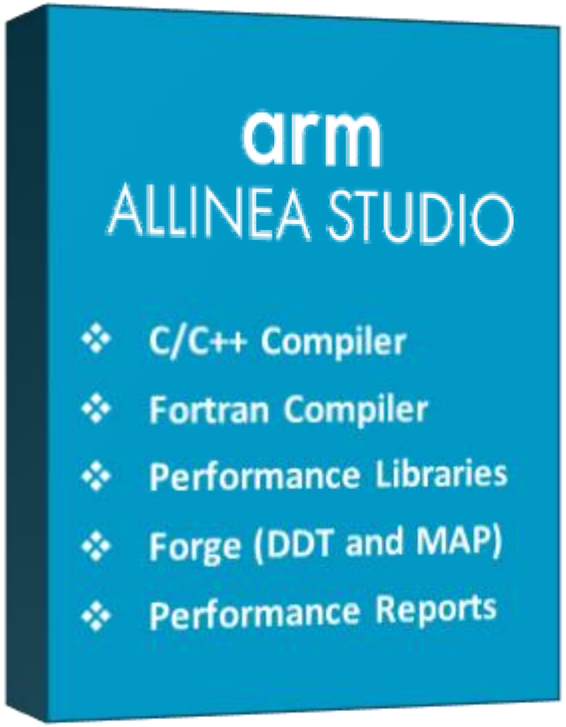
It’s possible to overlap different I/O approaches

HDF5 and VTK I/O operations occur at the same time on different ranks.



Wrap Up

Visit arm.com/hpc to learn more about Arm Forge and download a free trial.



- Use a profiler like MAP to drive performance engineering.
- Be aware of common I/O patterns and when to use them.
- Be aware of the filesystems available on your HPC system.

Download a free trial of Arm Forge

Thank You!

Danke!

Merci!

谢谢!

ありがとう!

Gracias!

Kiitos!

감사합니다

धन्यवाद