Accelerating CFD simulations with DCPMM

Michèle Weiland
m.weiland@epcc.ed.ac.uk

Prototype specification

- System built by Fujitsu using bespoke motherboard
- 34 compute nodes - node configuration
  - Dual socket, 2 24-core Intel Xeon Platinum 8260M CPUs
  - 192GB DDR4 DRAM (12 x 16GB)
  - 3TB DCPMM (12 x 256GB)
- Omni-Path interconnect
  - Dual rail
- 270TB Lustre file system
- Total memory capacity
  - 6.5TB DRAM
  - 100TB NVRAM

DCPMM App Direct mode

- NVRAM has to be addressed directly
  - PMDK, system software, direct loads/stores, filesystem
- Different namespaces
  - fsdax: “filesystem DAX” - default, block device
  - devdax: “device DAX” - character device file rather than block device
  - raw: memory disk, no DAX support

```
[mweiland@nextgenio-cn06 ~]$ df -h /mnt/pmem_fsdax*
Filesystem      Size  Used Avail Use% Mounted on
/dev/pmem12     1.5T   44G  1.4T   4% /mnt/pmem_fsdax0
/dev/pmem13     1.5T   18G  1.4T   2% /mnt/pmem_fsdax1
```

OpenFOAM example

- v1812 built with Intel 19.0.3.199 compilers & MPI library
- Test case: open wheel race car geometry – 90 million cells

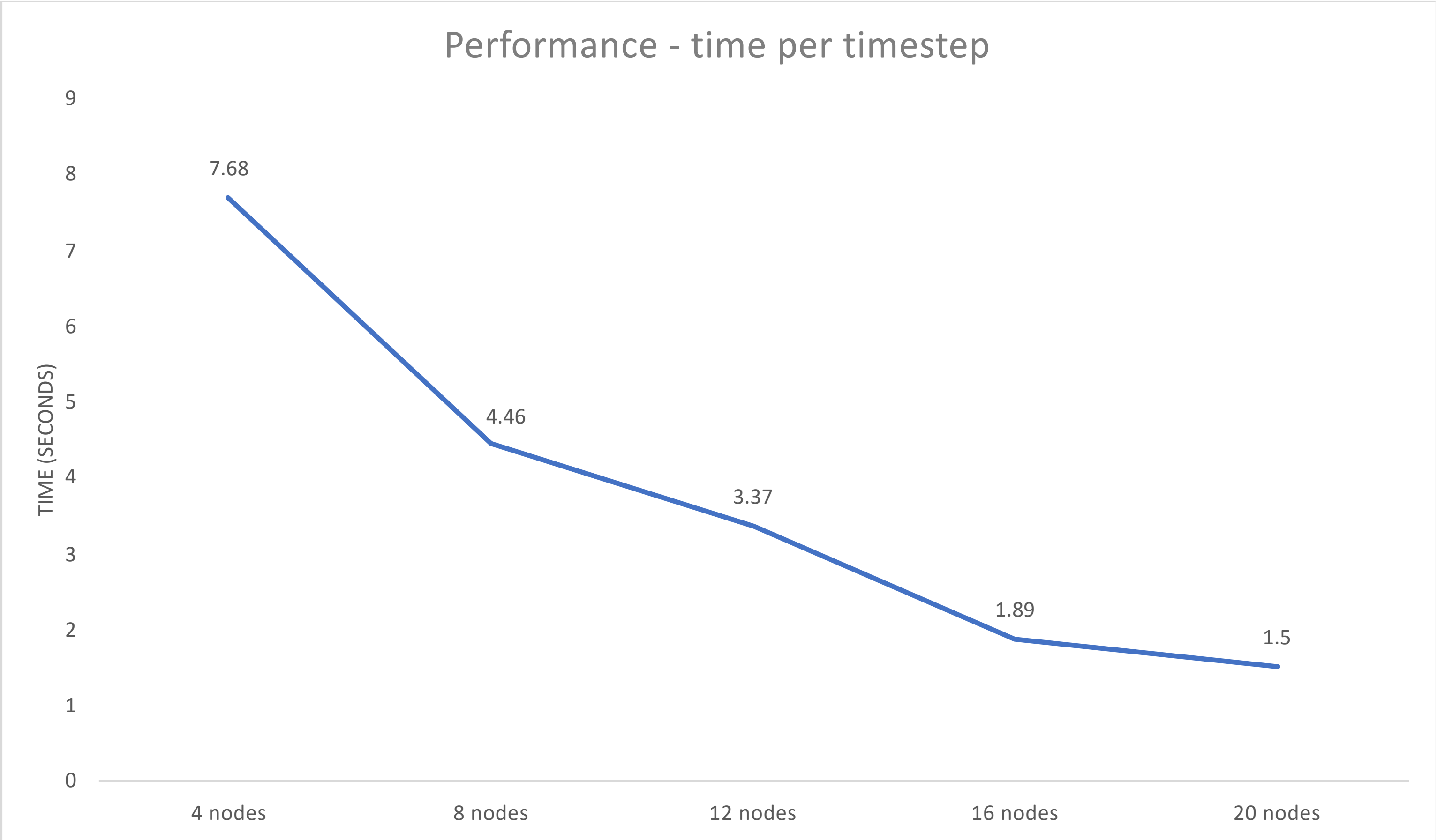
decomposePar →
- serial
- writes out decomposed mesh

renumberMesh →
- parallel
- reads in decomposed mesh
- applies renumbering by overwriting existing mesh

potentialFoam →
- parallel
- reads in field data
- Applied boundary conditions

simpleFoam
- parallel
- reads in mesh, fields and boundary conditions
- writes output at user defined intervals

Performance of default setup



I/O settings in OpenFOAM

Option	Description	Our setting(s)
`fileHandler`	uncollated = 1 directory per process, and per timestep - many, many (small) files collated = 1 directory for all processes, data in a single file - fewer files, master I/O	uncollated
`writeInterval`	dictates how often is data written	600/100/10/1
`purgeWrite`	controls how many time directories are kept. Purging the time directories increases meta-data operations, not purging increases amount of data that is kept	1
`runTimeModifiable`	controls whether dictionaries are re-read during a simulation at the beginning of each time step - on or off	off
`writeFormat`	binary or ASCII	binary
`writeCompression`	on or off	off

I/O characteristics for this case

```
-- processorN
   |-- constant
   |-- 0
   |-- timestepT
       |-- k
       |-- nut
       |-- omega
       |-- p
       |-- phi
       |-- U
       |-- uniform
           |-- time
           |-- functionObjects
               |-- functionObjectProperties
```

"constant" and "0" are input – 31GB

timestep folder and its contents are output

Data volume for 192 processes:
~40MB per timestep on each process
→ total: 7.5GB per timestep

→ writing every step for 500 steps
7.5GB * 500 = ~3.7TB

Total number of files created: N * T * 8
Total number of directories created: N * T * 3

Example: 960 processors, 100 iterations, write interval 1
→ 960 * 100 * 8 = 768,000 files
→ 960 * 100 * 3 = 288,00 directories

Using fsdax directly

- Taking advantage of OpenFOAM's I/O strategy
  - Write once, read never
  - Write locally (if uncollated)

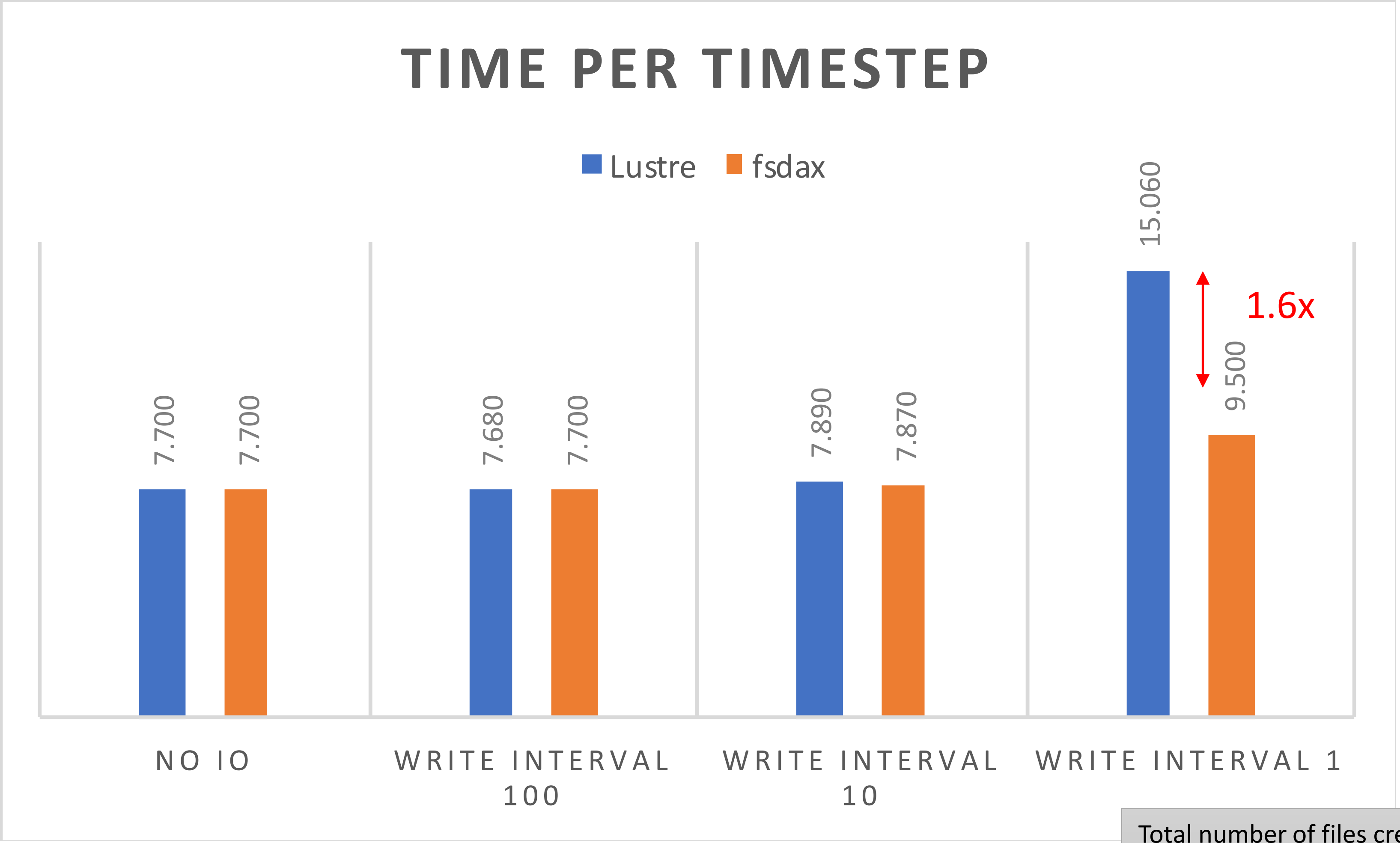
- Copy test case data to DCPMM
  - *Use fsdax on both sockets!*

```
# Copying data to 4 nodes
cd /mnt/pmem_fsdax0
time srun -n 4 -N 4 cp -fr /home/nx01/nx01/mweiland/caseDir .
cd /mnt/pmem_fsdax1
time srun -n 4 -N 4 cp -fr /home/nx01/nx01/mweiland/caseDir .
```

- Exploit MPMD for good data locality

```
time mpirun -ppn 48 -n 24 simpleFoam -case /mnt/pmem_fsdax0/caseDir -parallel : \
                    -n 24 simpleFoam -case /mnt/pmem_fsdax1/caseDir -parallel : \
                    -n 24 simpleFoam -case /mnt/pmem_fsdax0/caseDir -parallel : \
                    -n 24 simpleFoam -case /mnt/pmem_fsdax1/caseDir -parallel : \
                    -n 24 simpleFoam -case /mnt/pmem_fsdax0/caseDir -parallel : \
                    -n 24 simpleFoam -case /mnt/pmem_fsdax1/caseDir -parallel : \
                    -n 24 simpleFoam -case /mnt/pmem_fsdax0/caseDir -parallel : \
                    -n 24 simpleFoam -case /mnt/pmem_fsdax1/caseDir -parallel
```

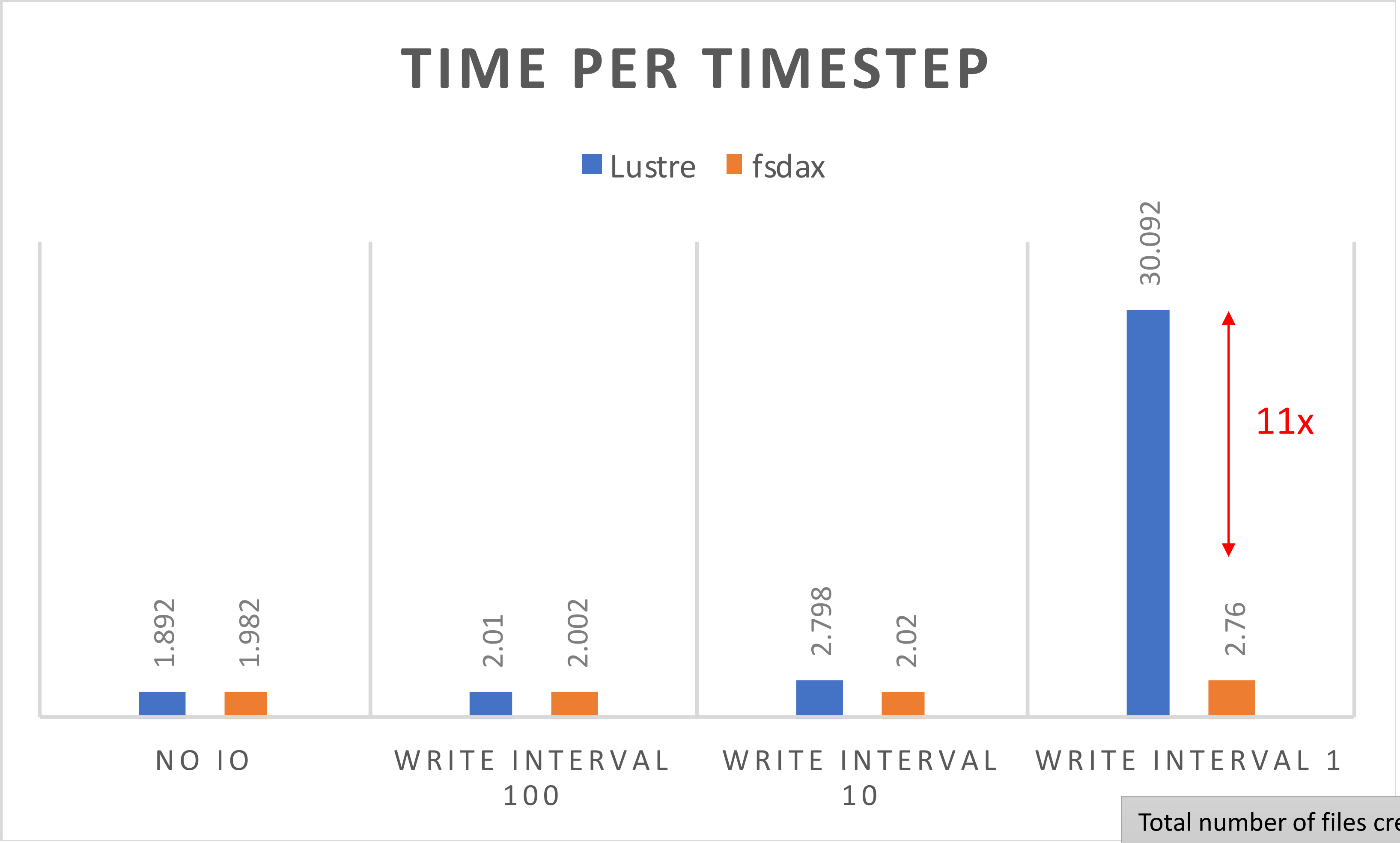
4 node experiments



Total number of files created:
Write interval 1: 192 * 100 * 8 = 153,600

Total number of directories created: N * T * 3
Write interval 1: 192 * 100 * 3 = 57,600

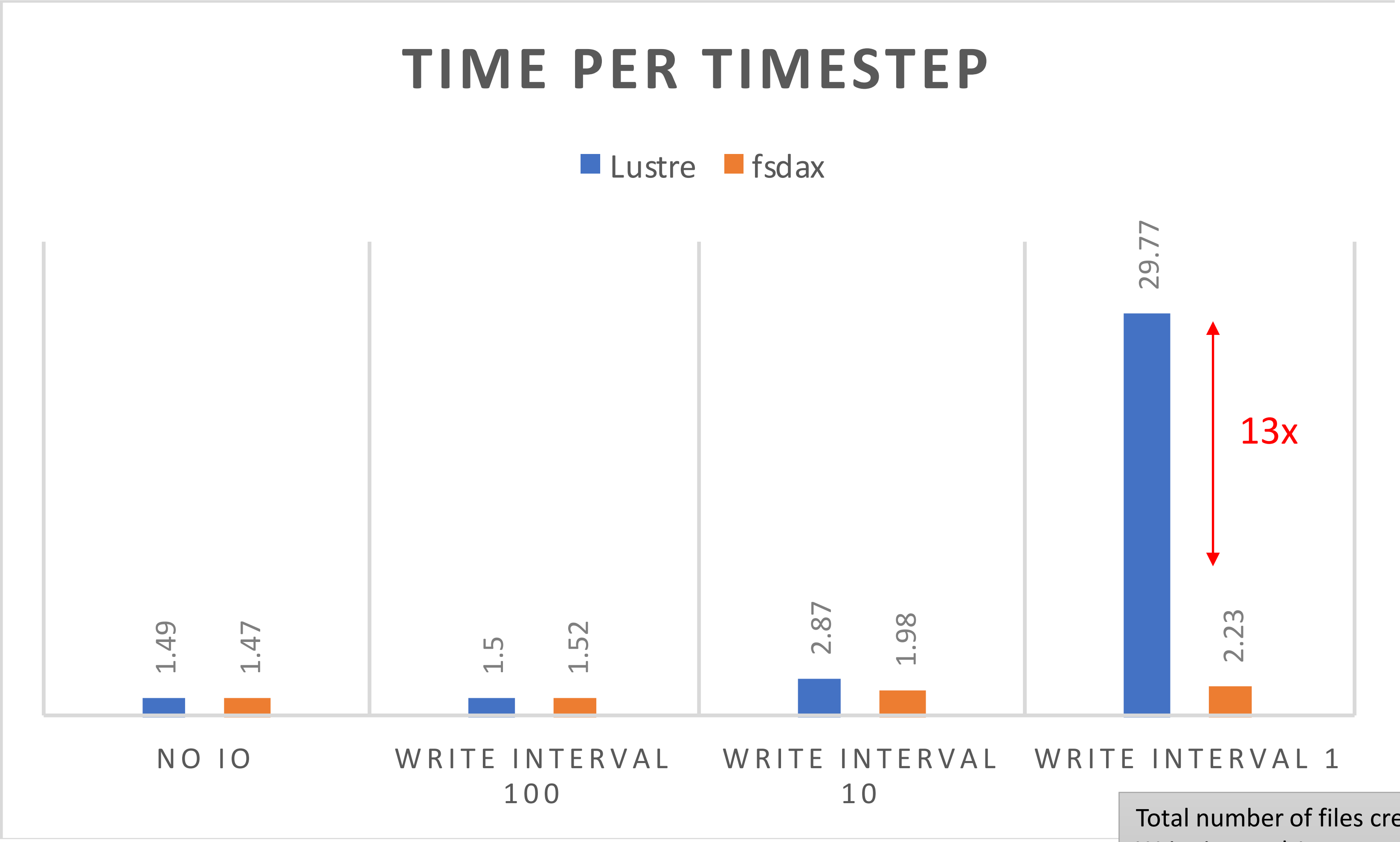
16 node experiment



Total number of files created:
Write interval 1: 768 * 100 * 8 = 614,400

Total number of directories created: N * T * 3
Write interval 1: 768 * 100 * 3 = 230,400

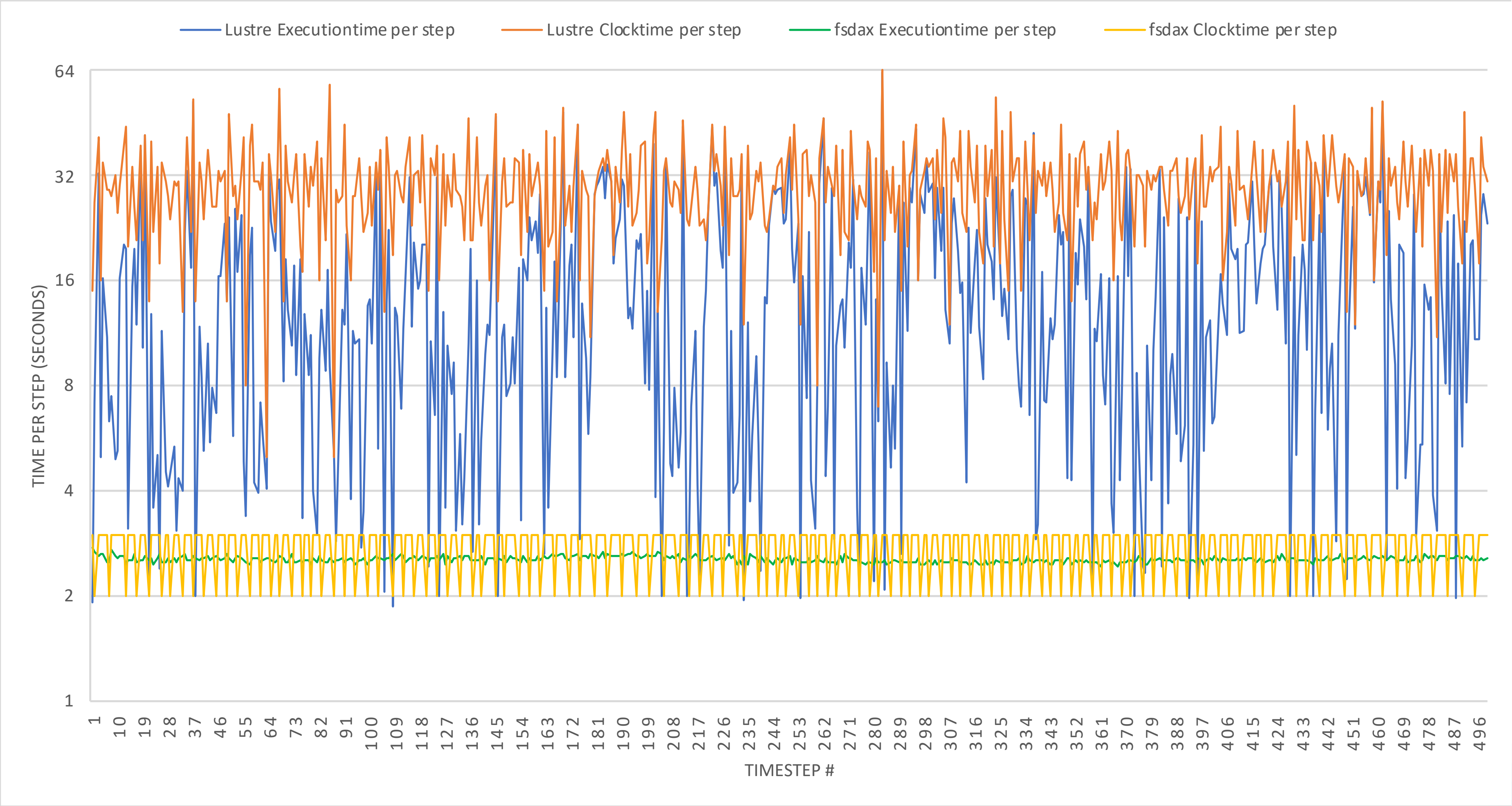
20 node experiment

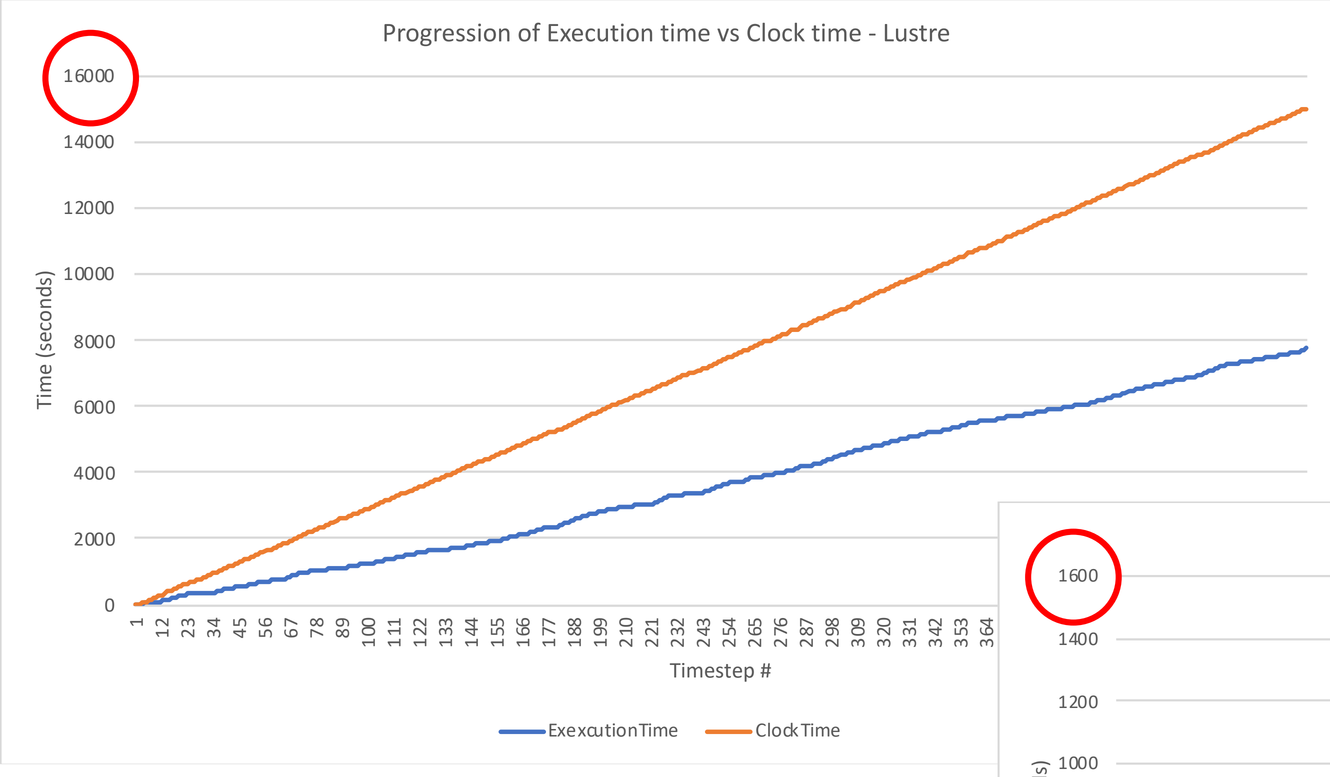


Total number of files created:
Write interval 1: 960 * 100 * 8 = 768,000

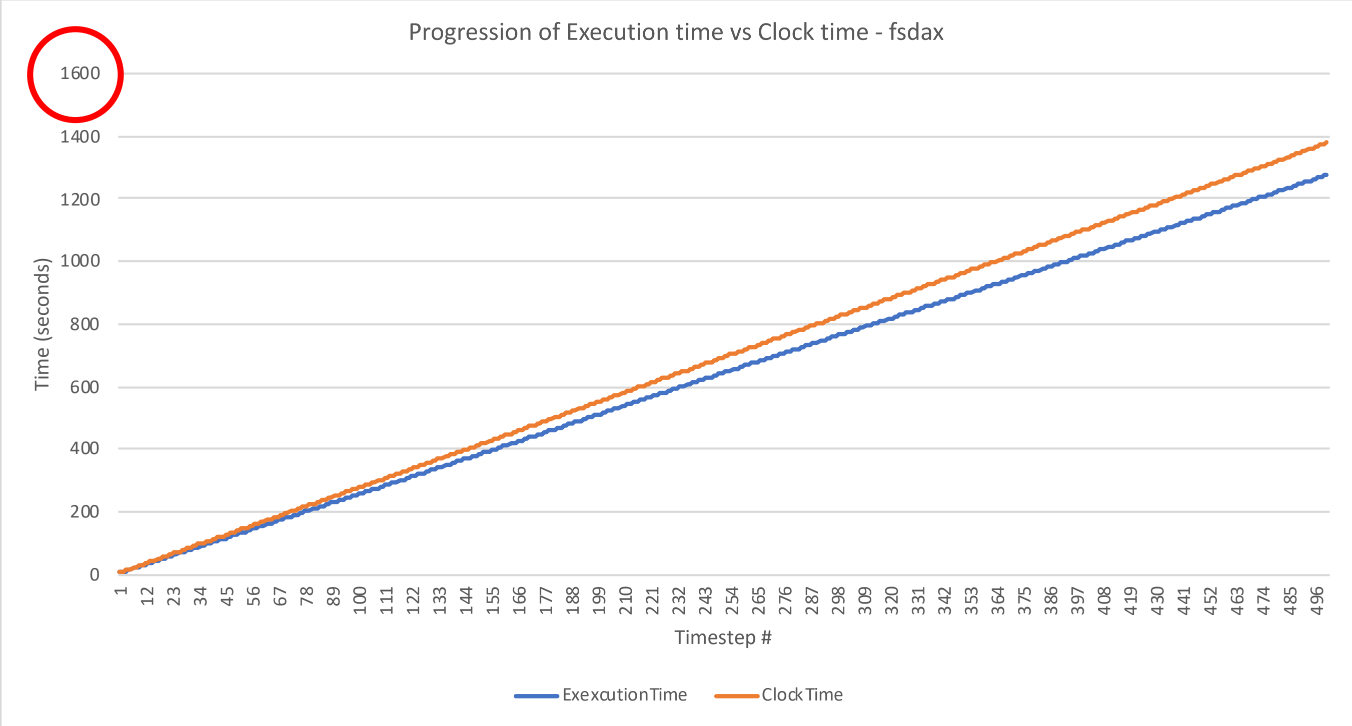
Total number of directories created: N * T * 3
Write interval 1: 960 * 100 * 3 = 256,000

Performance stability





Progression of Execution time vs Clock time - Lustre
16000
14000
12000
10000
8000
6000
4000
2000
0
Time (seconds)
Timestep #
ExecutionTime
ClockTime



Progression of Execution time vs Clock time - fsdax
1600
1400
1200
1000
800
600
400
200
0
Time (seconds)
Timestep #
ExecutionTime
ClockTime

Achieved I/O performance

- Difference in time per timestep between “no IO” and “write interval X” can be entirely attributed to data writes and associated metadata operations

- On 20 nodes – for “write interval 1”:

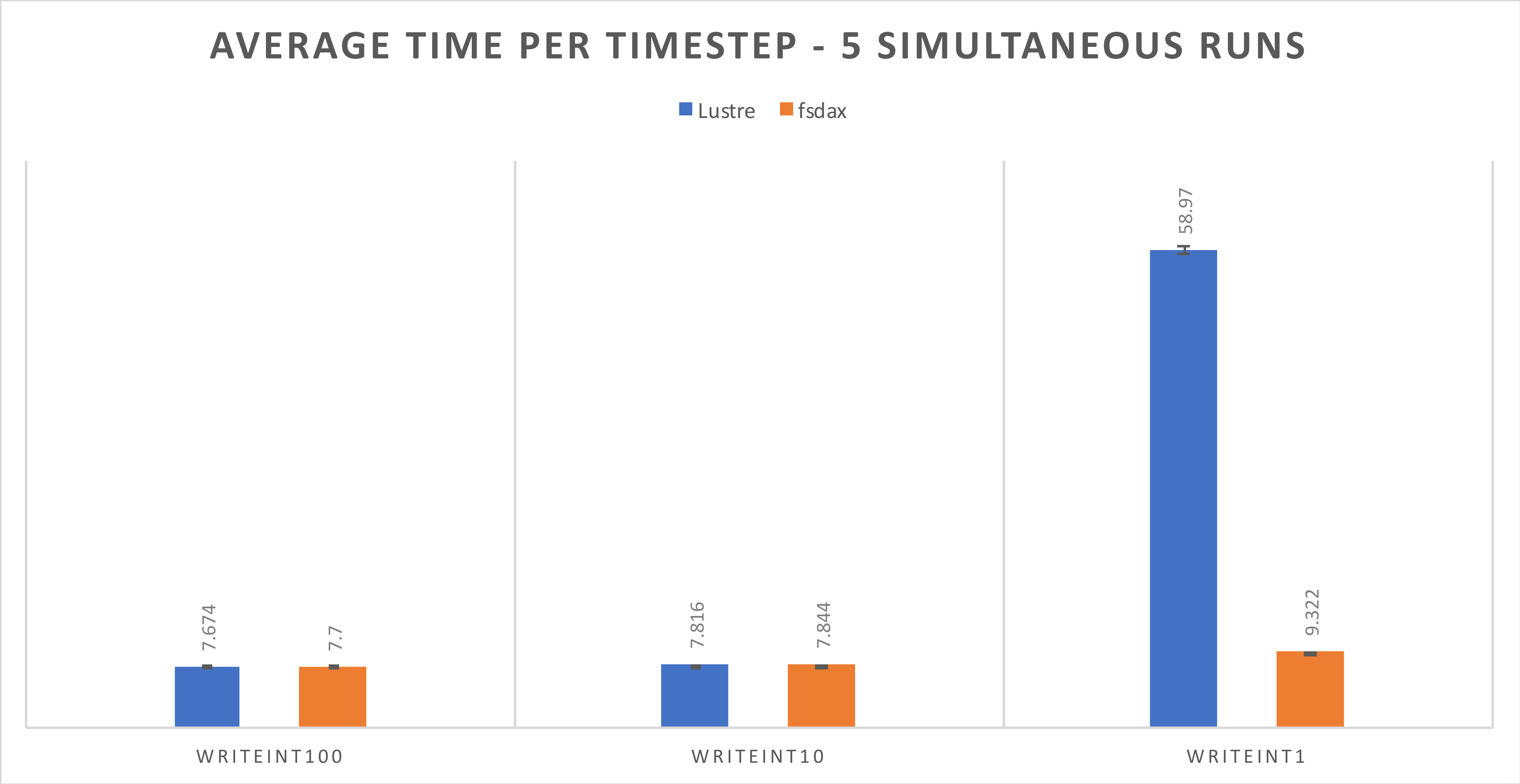
Lustre	*29.77s - 1.47s = 28.28s*
	7.5GB / 28.28s = 0.256GB/s
fsdax	*2.23s – 1.47s = 0.76s*
	7.5GB / 0.76s = 9.87GB/s

Ensemble experiments

- Nobody runs on empty systems
- Single isolated experiments are also rare

- For a more realistic view, we compare performance of five 4-node experiments running concurrently
  - Using a total of 20 nodes, but independent simulations
  - More representative of real, busy environments
  - Plus: same number of metadata operations, but 5 times the amount of data written
    - 7.5GB per timestep *per simulation*

Ensemble performance



Time per timestep: single run vs ensemble

	Lustre		fsdax	
	Single run (4 nodes)	Ensemble average (5 x 4 nodes)	Single run (4 nodes)	Ensemble average (5 x 4 nodes)
write interval 100	7.680	7.674	7.700	7.700
write interval 10	7.890	7.816	7.870	7.844
write interval 1	15.060	**58.97**	9.500	9.322

7.5GB per step (Lustre single run, fsdax single run)

37.5GB per step (Lustre ensemble average, fsdax ensemble average)

Lustre
7.5GB / (15.06-7.68) = 1.01GB/s
37.5GB / (58.97-7.67) = 0.73GB/s

fsdax
7.5GB / (9.5-7.7) = 4.17GB/s
37.5GB/ (9.3-7.7) = 23.12GB/s

Cost of moving data

- Copying the data to DCPMM is not free – on 20 nodes time taken is

```
fsdax0                  fsdax1
real  3m28.461s         real  1m32.576s
user  0m0.032s          user  0m0.032s
sys   0m0.043s          sys   0m0.047s
```

- However, currently simply copying *everything* to *everywhere* – it would not be difficult to only copy the relevant data
  - On 20 nodes this would mean copying 24 processor directories per socket, not all 960!

Future work

- Full system runs and increasing the I/O per time step without adding more metadata operations
  - How far can we push fsdax?
- Decomposition still done on Lustre
  - Will look into moving this to DCPMM and distributing data from there
- Meshing
  - Example used here does not include meshing, but it is an area of interest
- Explore devdax?
  - OpenFOAM is a nightmare to modify, but it's "only" the output stream that needs to be changed (write once, read never)

Conclusions

- OpenFOAM’s I/O strategy is inadvertently an extremely good fit for fsdax on DCPMM
  - Trivial to exploit, really good gains when pushing scale and I/O frequency in particular
- Lustre is fine up to a point, but easy to push over the edge
  - DCPMM on the other hand does not struggle even with high demand on write and metadata operations
- Only beginning the scratch the surface of the opportunities for real applications