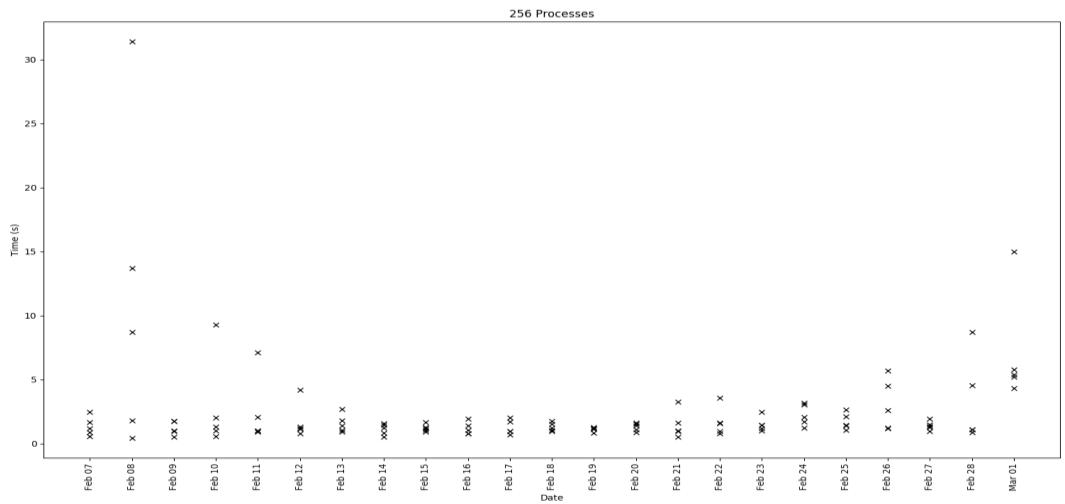
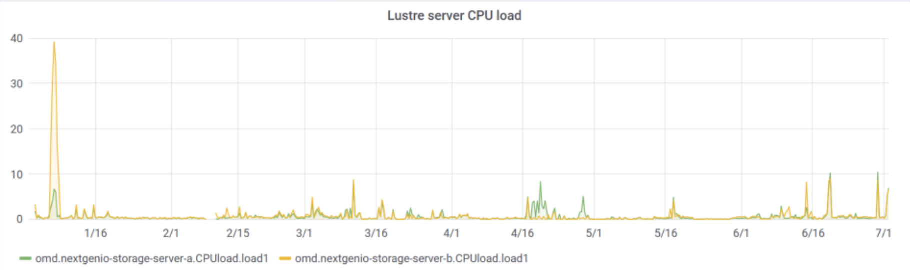
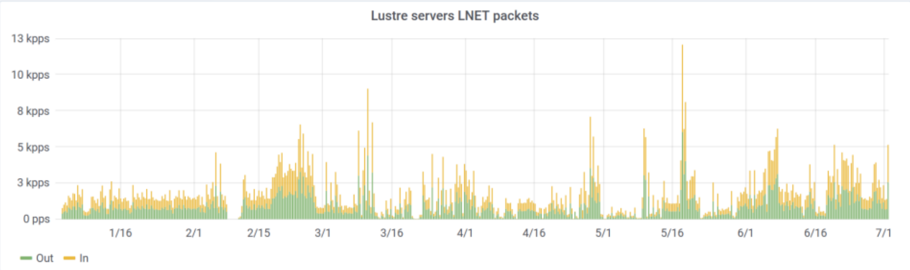
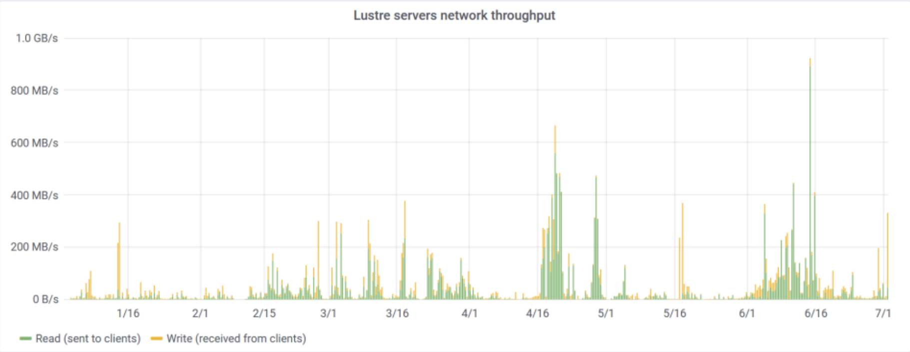
Optimising Performance Through Data Localisation

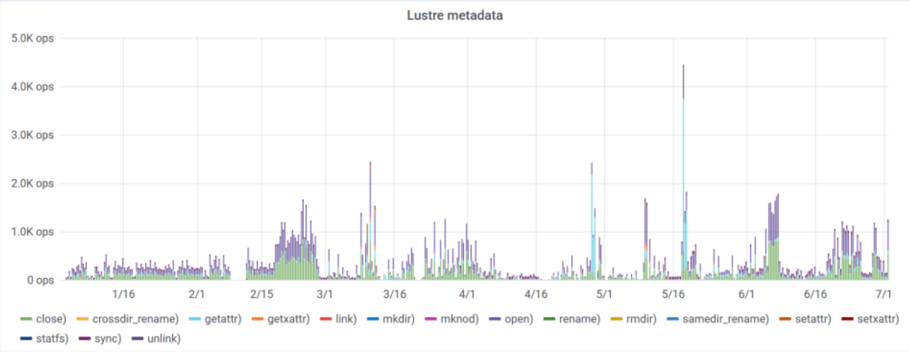
Adrian Jackson
EPCC, The University of Edinburgh
a.jackson@epcc.ed.ac.uk
@adrianjhpc


256 Processes
Time (s)
Date
Feb 07
Feb 08
Feb 09
Feb 10
Feb 11
Feb 12
Feb 13
Feb 14
Feb 15
Feb 16
Feb 17
Feb 18
Feb 19
Feb 20
Feb 21
Feb 22
Feb 23
Feb 24
Feb 25
Feb 26
Feb 27
Feb 28
Mar 01

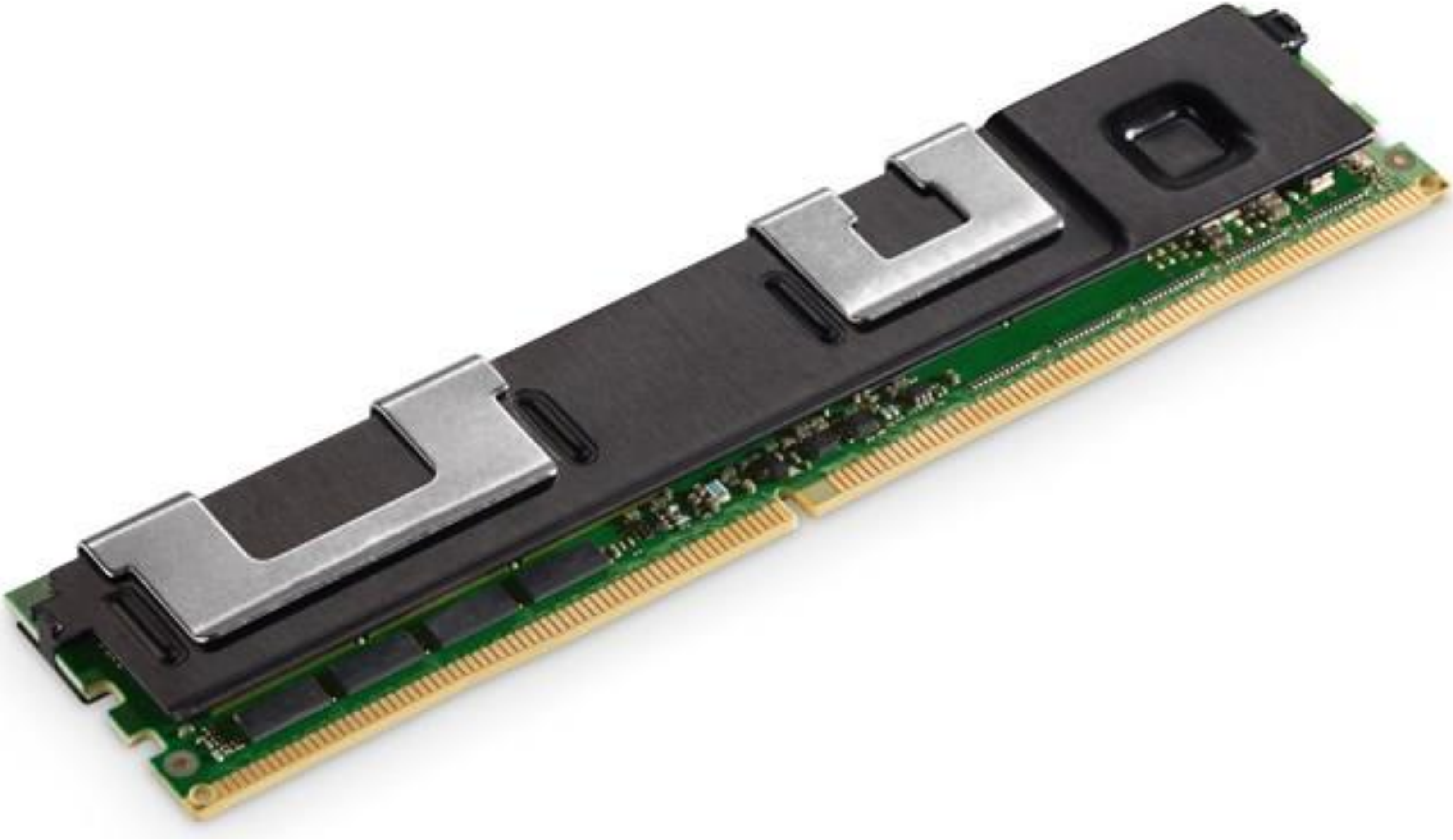

Lustre server CPU load
omd.nextgenio-storage-server-a.CPUload.load1
omd.nextgenio-storage-server-b.CPUload.load1


Lustre servers LNET packets
Out
In

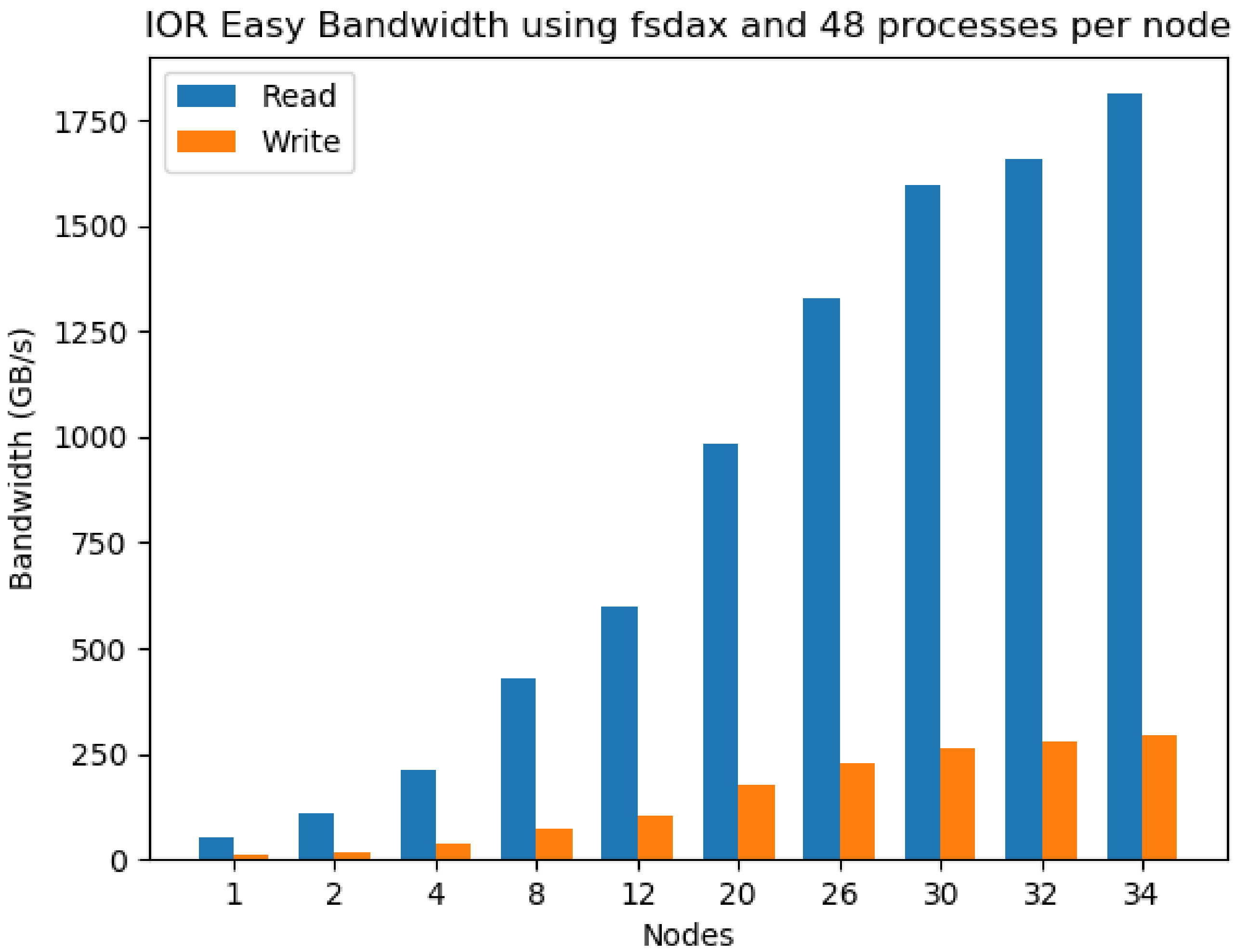

Lustre servers network throughput
Read (sent to clients)
Write (received from clients)


Lustre metadata
close)
crossdir_rename)
getattr)
getxattr)
link)
mkdir)
mknod)
open)
rename)
rmdir)
samedir_rename)
setattr)
setxattr)
statfs)
sync)
unlink)

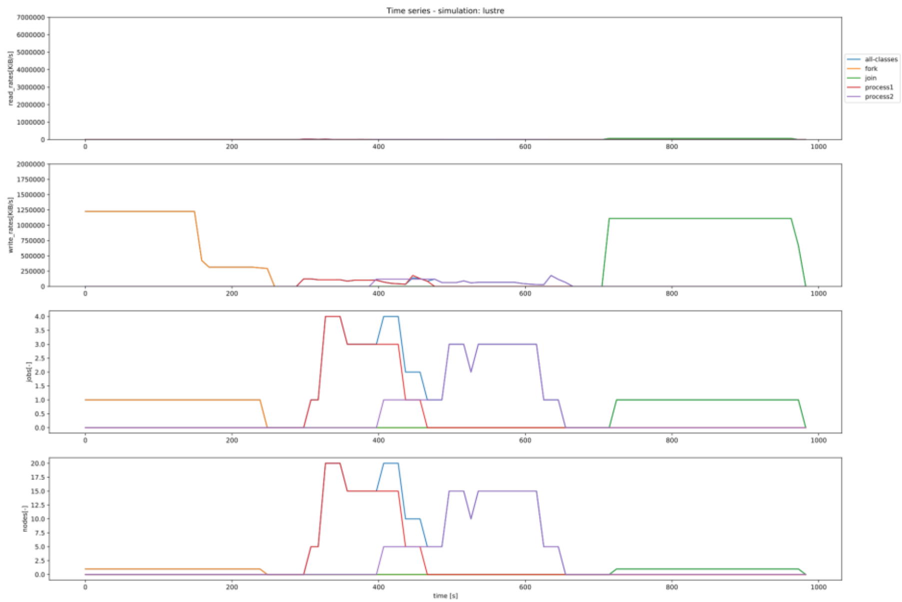
NVRAM / B-APM

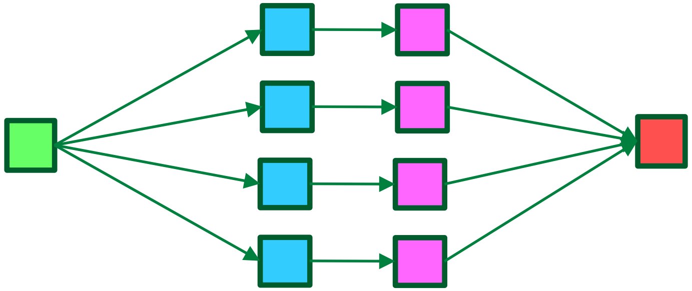


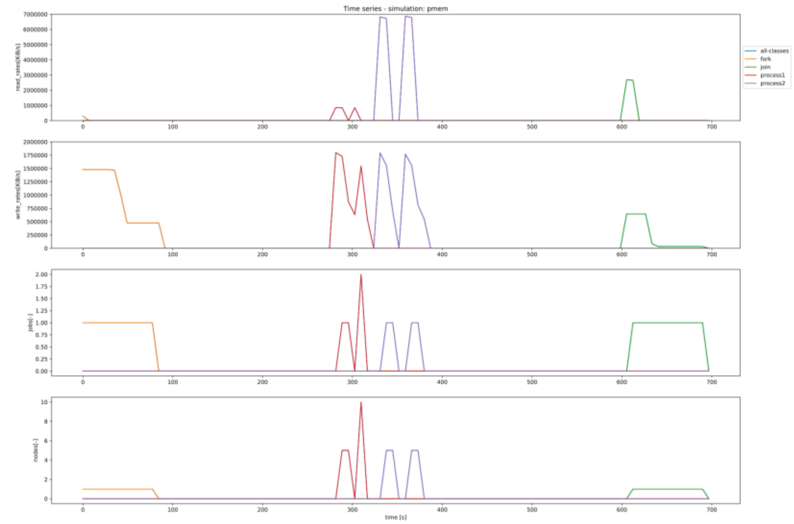
Persistent memory performance



Workflows







1 node
4 processes
4 files

20 nodes
80 processes
80 files

1 node
4 processes
80 files

I/O Optimisation with persistent memory

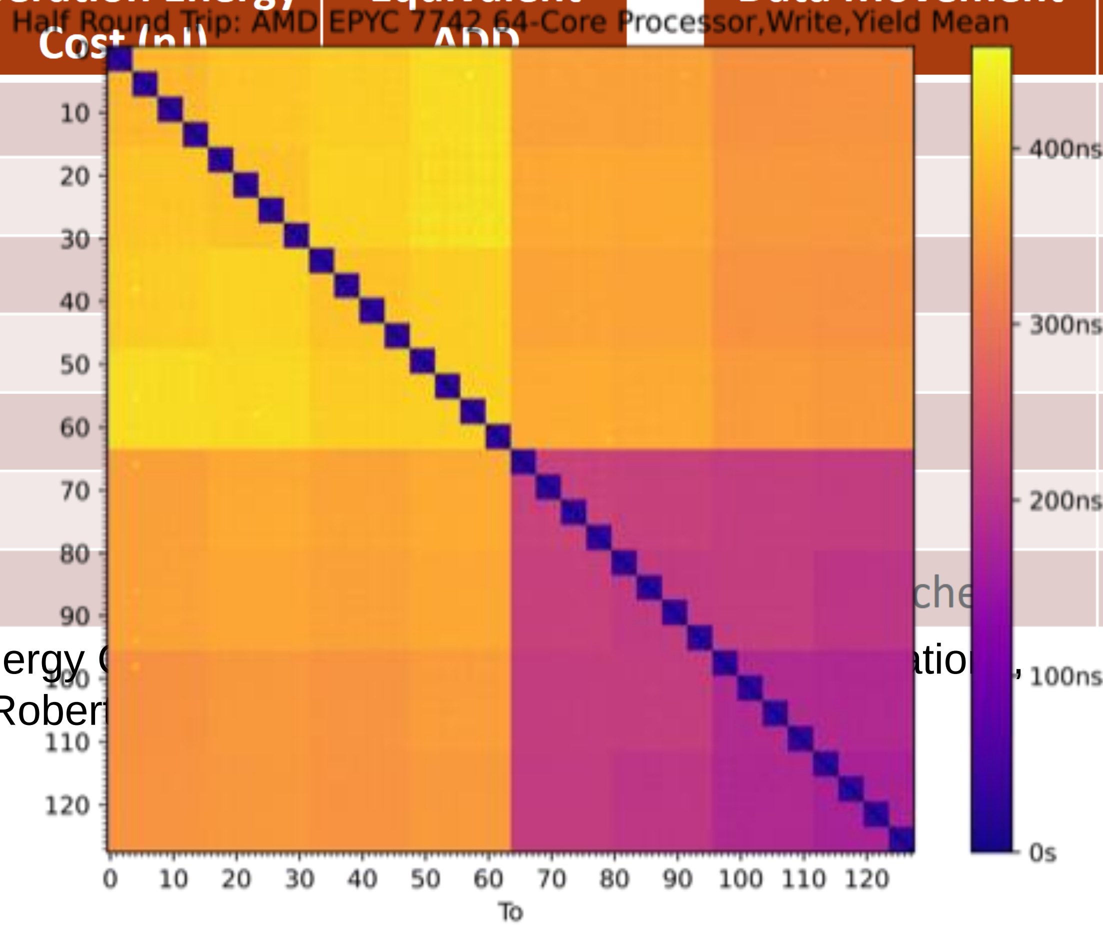
- n3d CFD application that uses combined forward/adjoint method
  - DNS used for Navier Stokes forward approach
  - Adjoint method requires full DNS output
  - DNS state is very large
- Medium simulation
  - 72 processes maximum
  - DNS state requires 4TB for storage
- Large simulation
  - 512 processes maximum
  - DNS state requires 40TB for storage
- Filesystem used to store data for the transition between phases

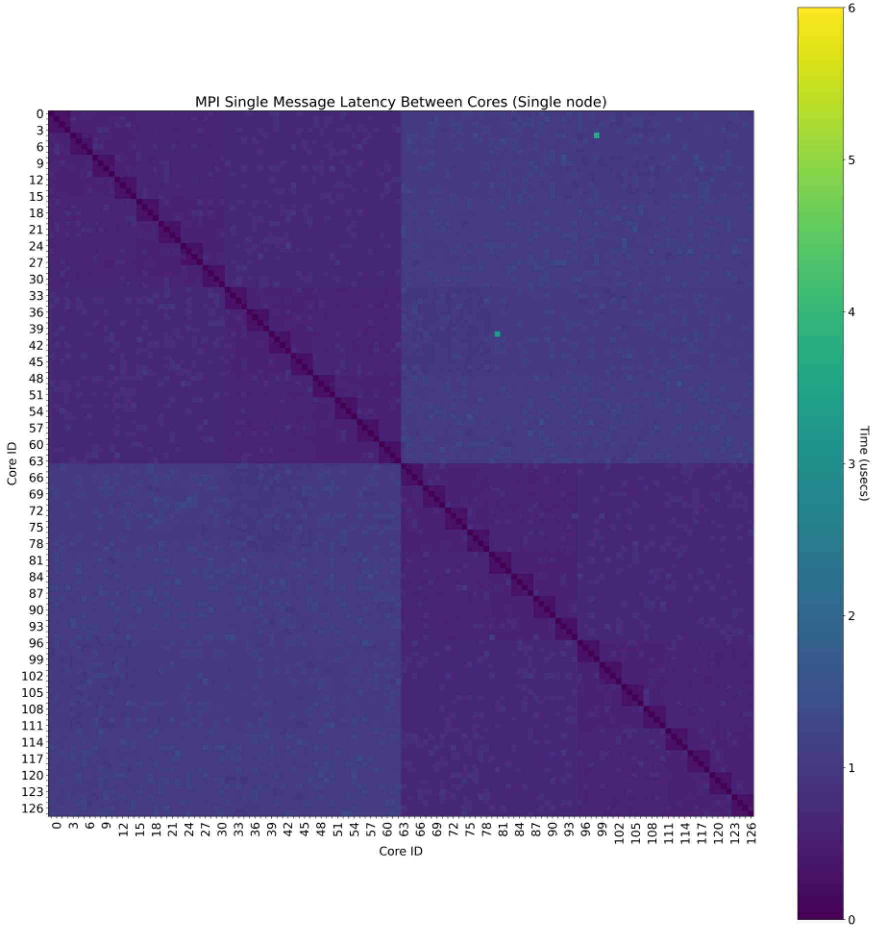
I/O Optimisation with persistent memory

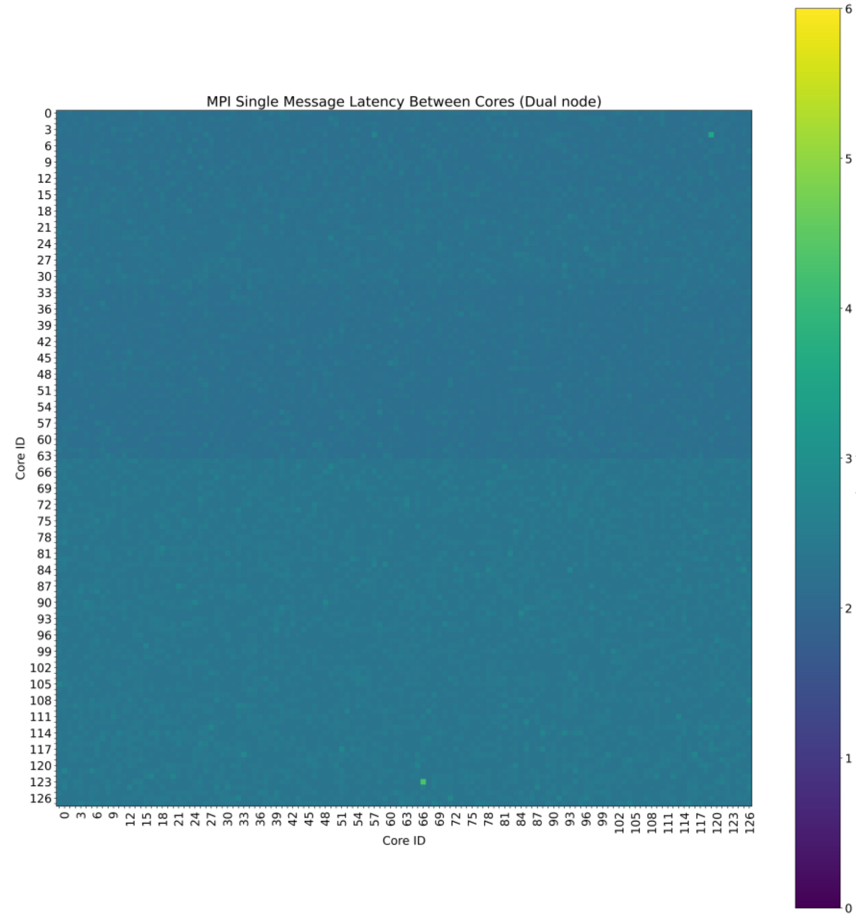
- Assuming compute nodes with 256GB DRAM, to fit in DRAM
  - Medium case would require a minimum of 16 nodes
  - Large scale would require a minimum of 160 nodes
- Using filesystem (Lustre) takes:
  - Medium case using 3 nodes: ~9800 seconds
  - Large case using 22 nodes: ~80000 seconds
- Using persistent memory for I/O on the nodes
  - Medium case using 3 nodes: ~8500 seconds (~15% faster)
  - Large case using 22 nodes: ~9200 seconds (~90% faster)
- Using persistent memory as memory on the nodes
  - Medium case using 3 nodes: ~8300 seconds
  - Large case using 22 nodes: ~9000 seconds

Operation	Operation Energy Cost (nJ)	Equivalent ADD	Data Movement	Data movement Energy (nJ)
ADD				-
L1->REG				1.11
L2->REG				1.10
L3->REG				7.59
MEM->REG				53.84
Stall				-
Prefetching			...che	65.08

Analyzing the Energy ... ation ,
Gokcen Kestor, Robert ...

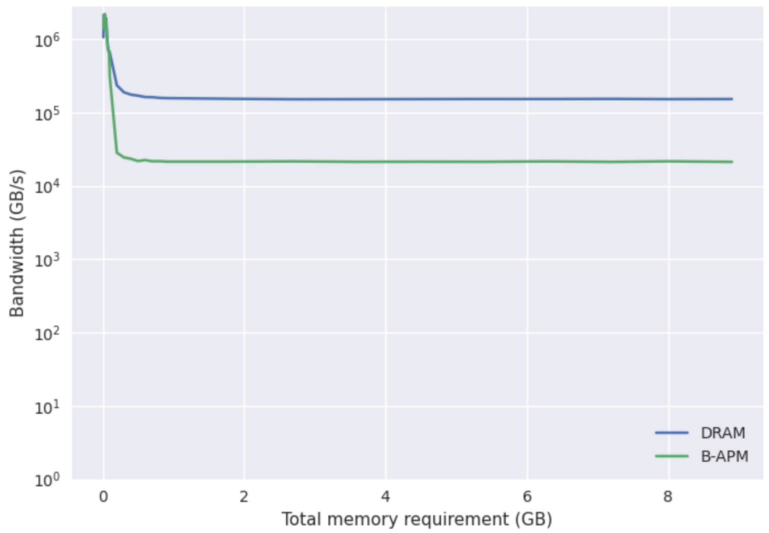



MPI Single Message Latency Between Cores (Single node)
Core ID
Core ID
Time (usecs)
0
1
2
3
4
5
6

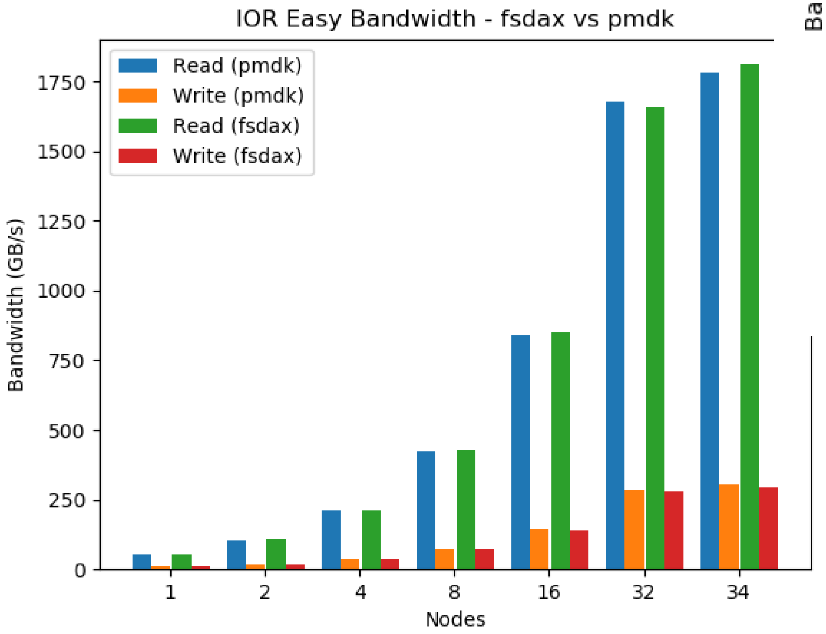

MPI Single Message Latency Between Cores (Dual node)
Core ID
Core ID
Time (usecs)
0
1
2
3
4
5
6

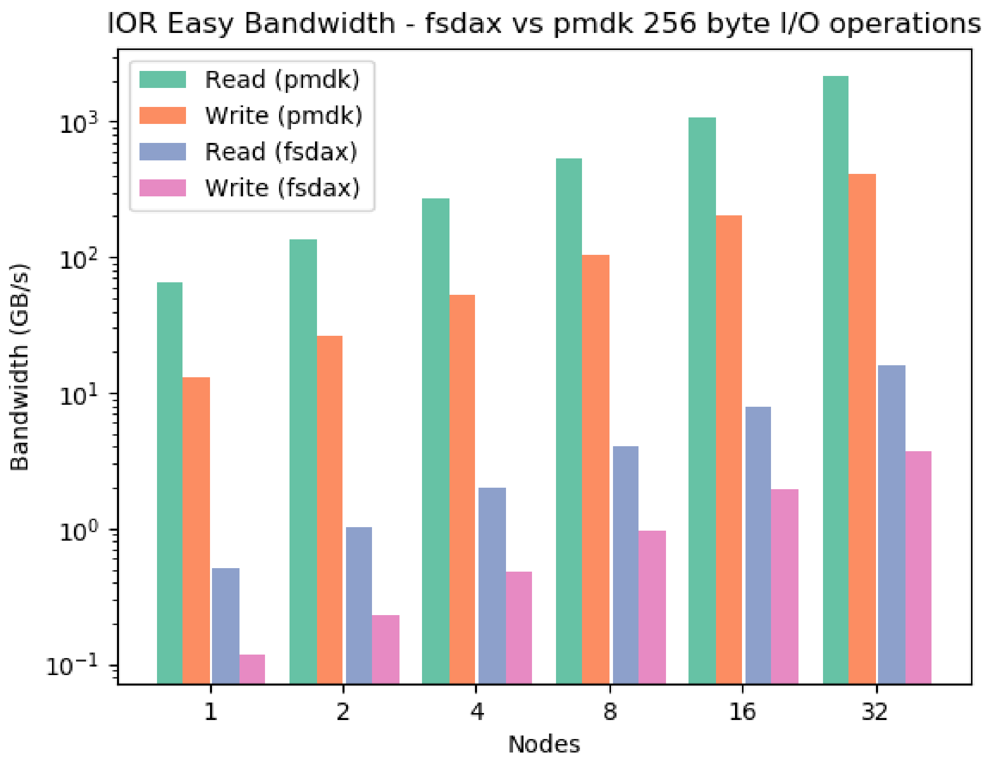
https://github.com/adrianjhpc/DistributedStream.git

Mode	Min BW (GB/s)	Median BW (GB/s)	Max BW (GB/s)
App Direct (DRAM)	142	150	155
App Direct (DCPMM)	32	32	32
Memory mode	144	146	147
Memory mode (large)	12	12	12

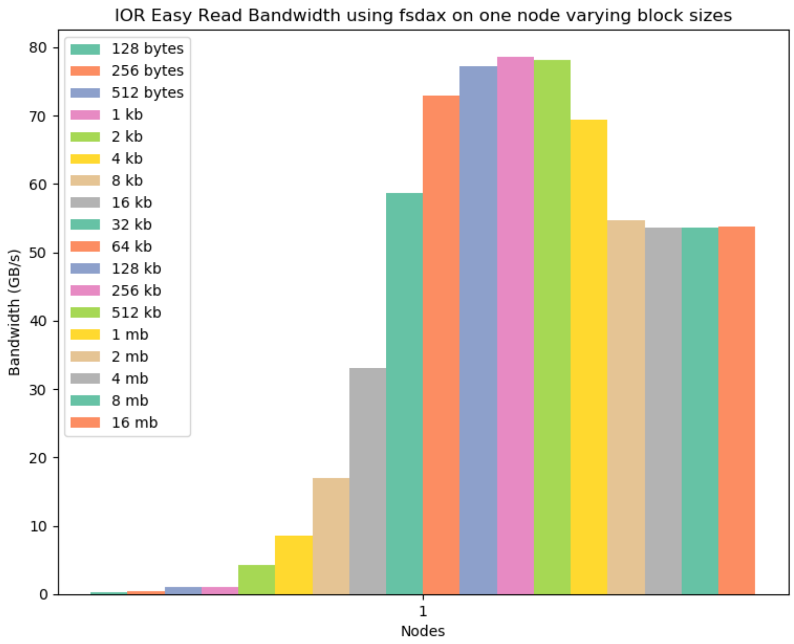


Data access sizes

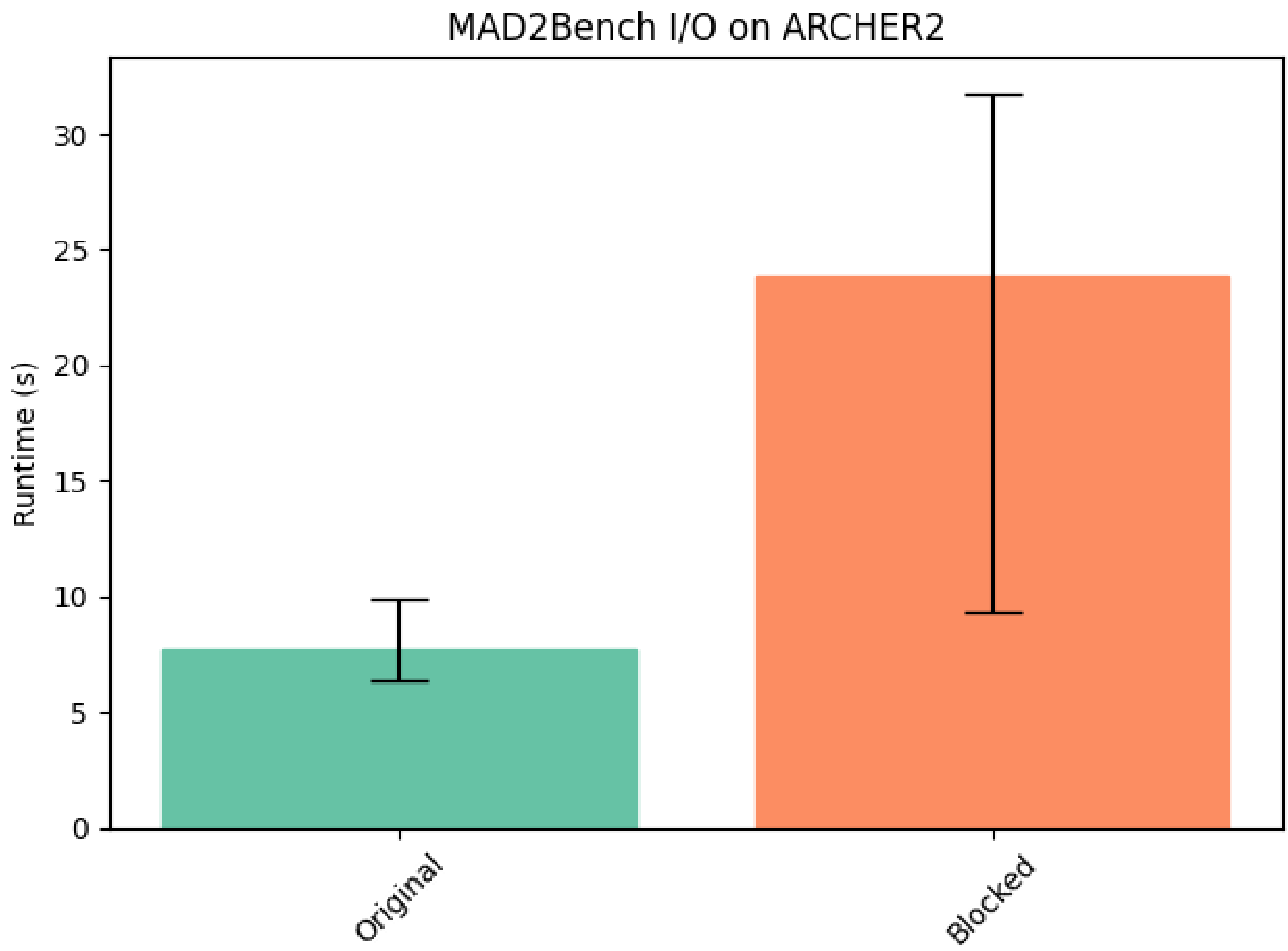




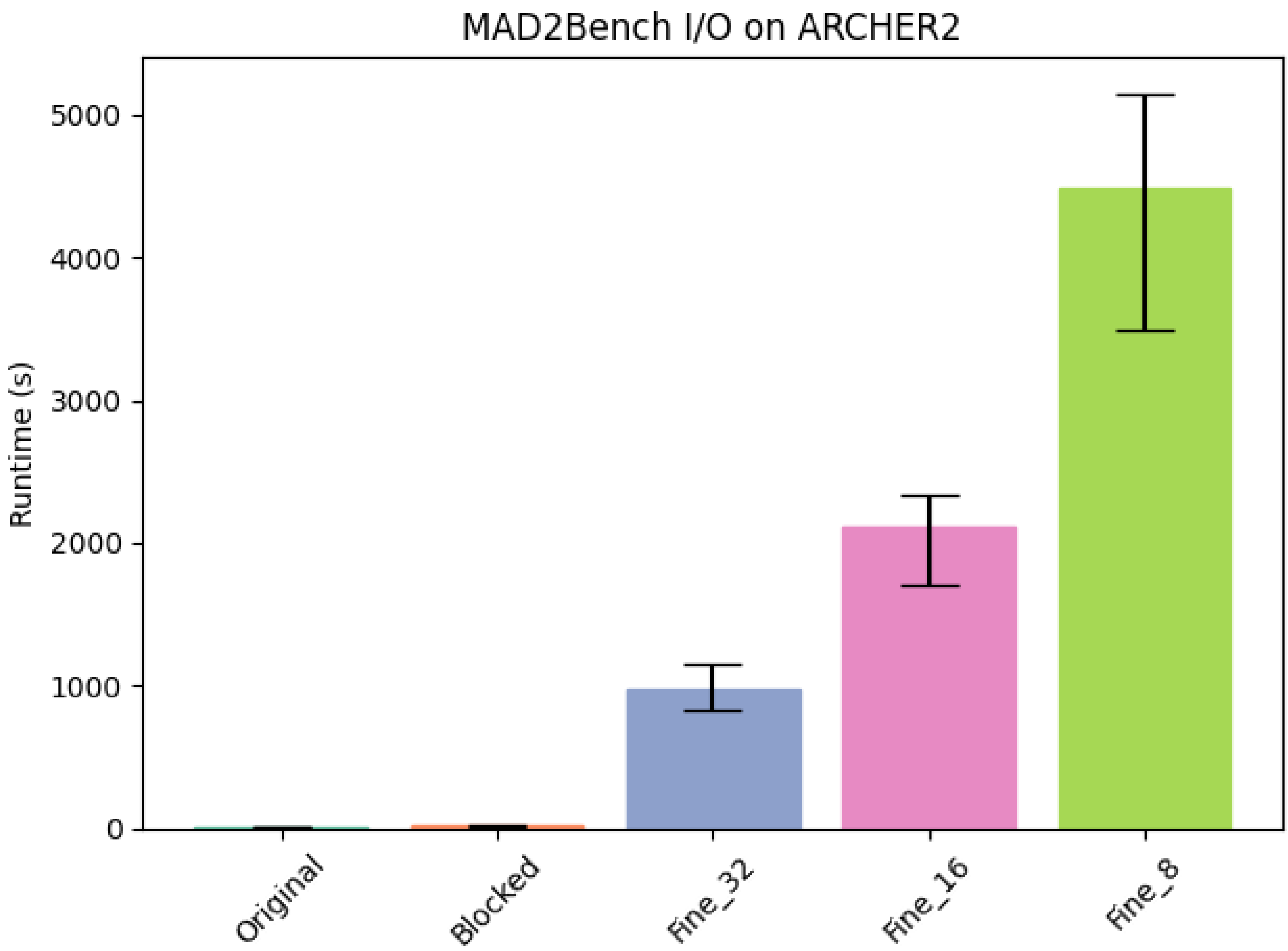
IOR - Data block sizes



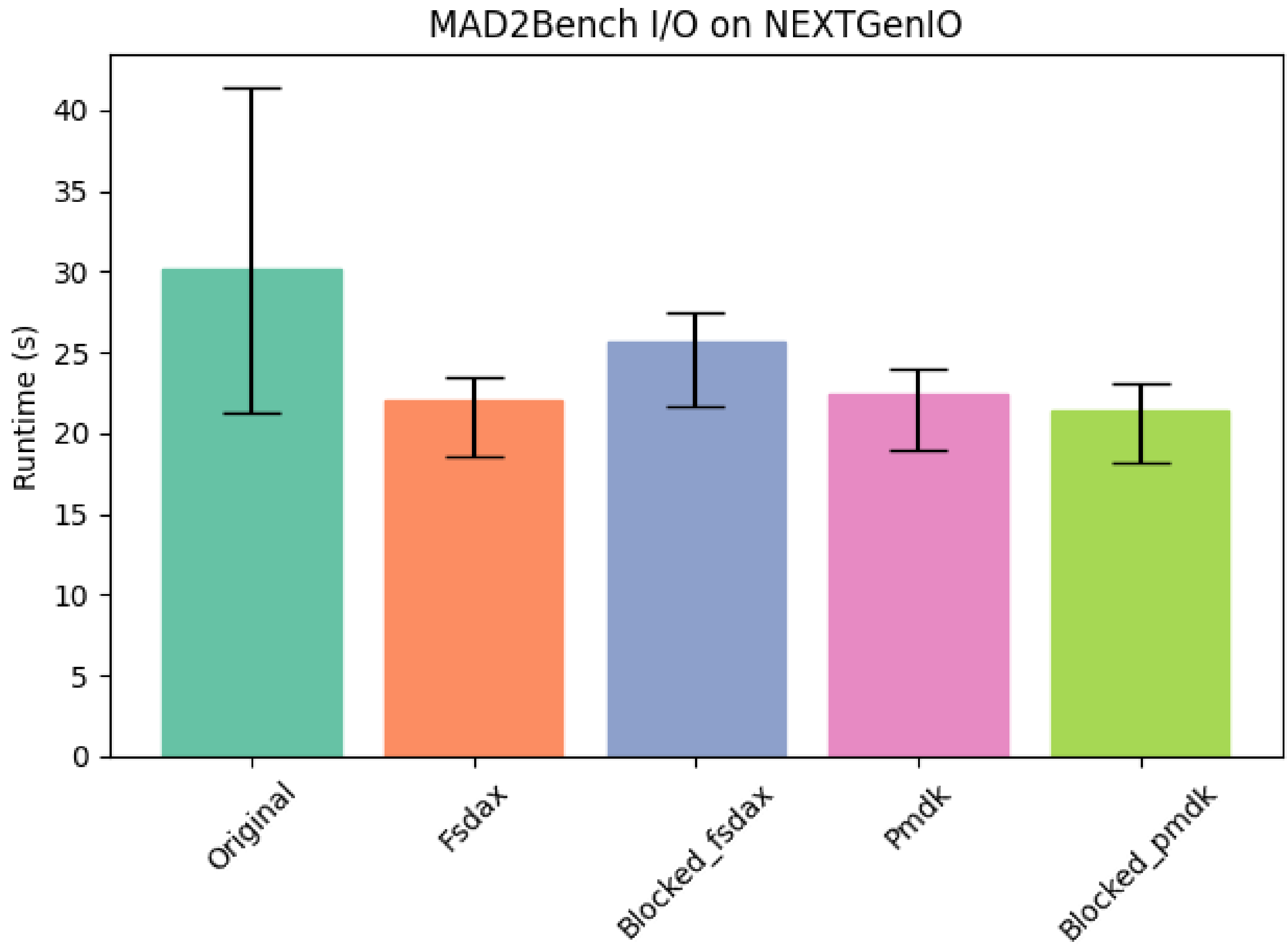
MAD2Bench


MAD2Bench I/O on ARCHER2
Runtime (s)
0
5
10
15
20
25
30
Original
Blocked

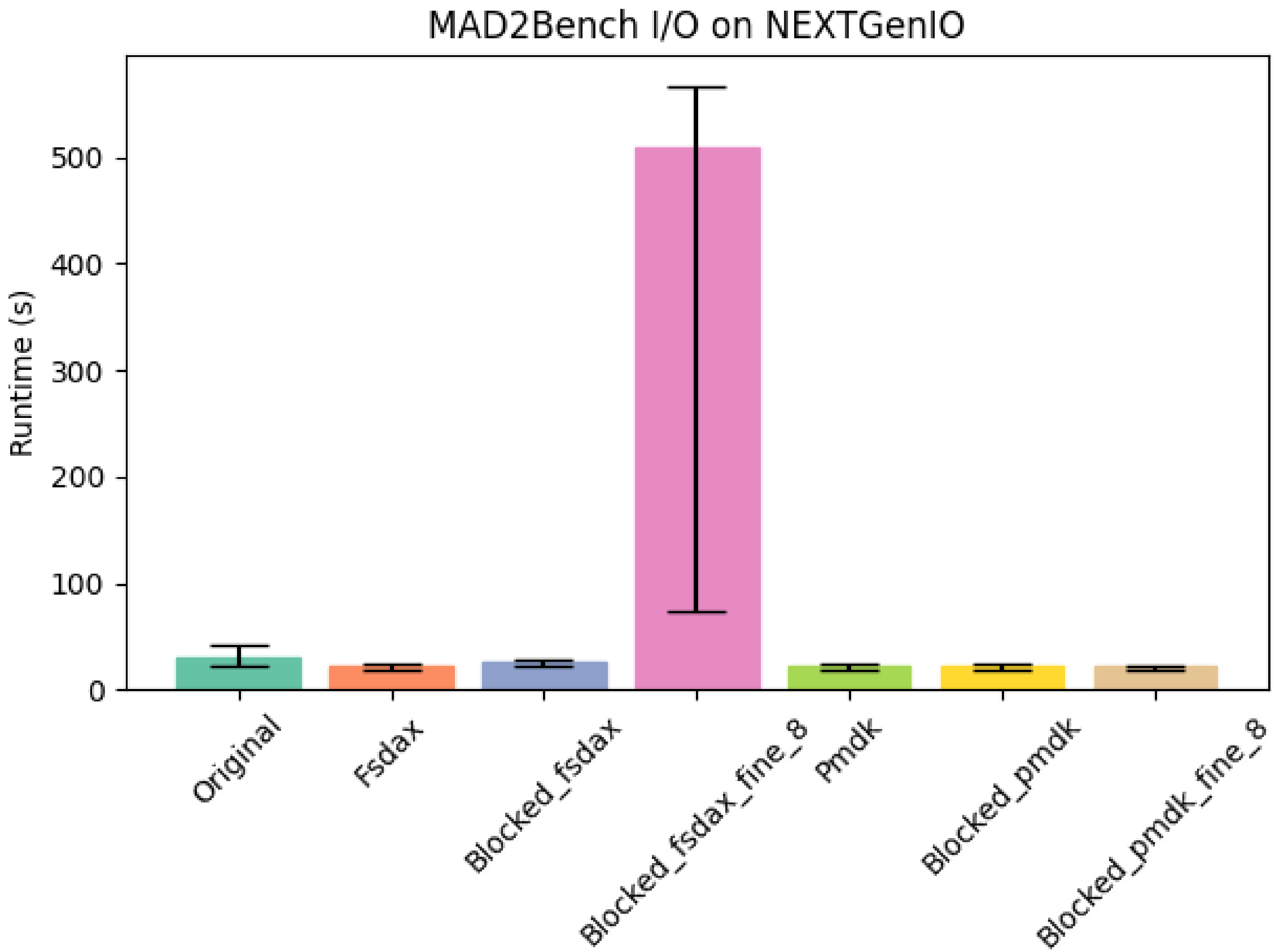
MAD2Bench


MAD2Bench I/O on ARCHER2
Runtime (s)
0
1000
2000
3000
4000
5000
Original
Blocked
Fine_32
Fine_16
Fine_8

MAD2Bench

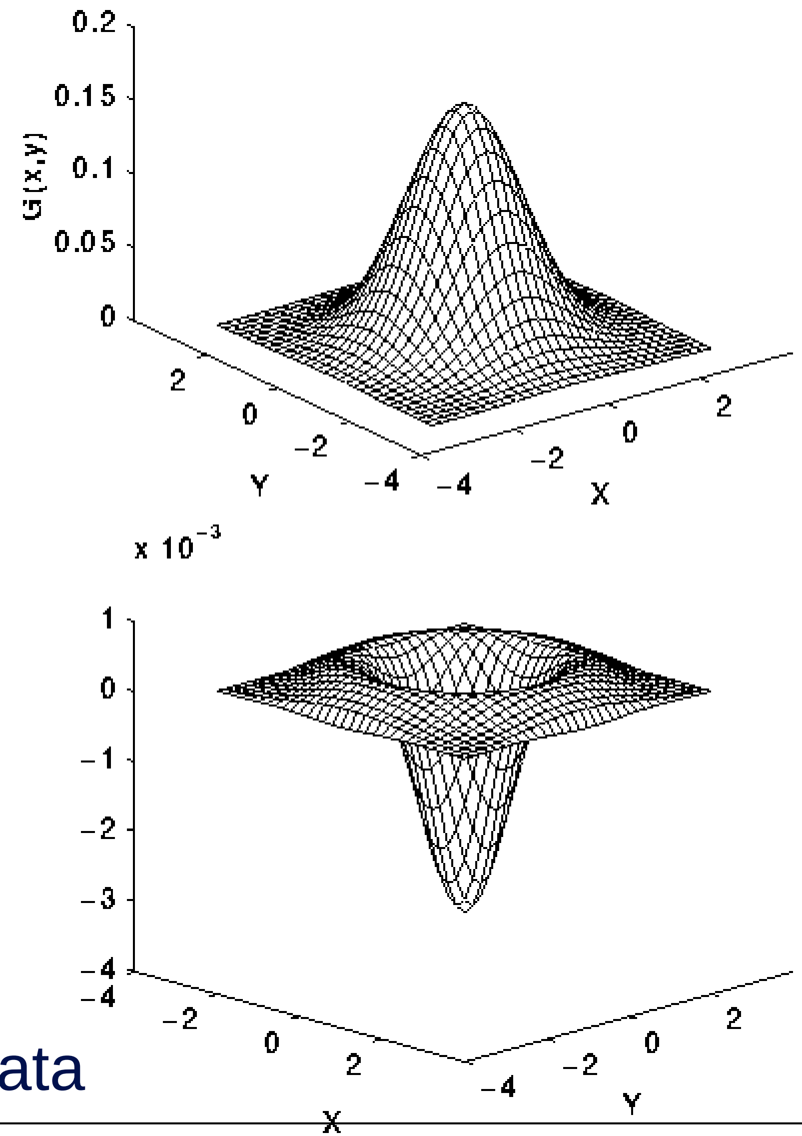


MAD2Bench I/O on NEXTGenIO
Runtime (s)
0
5
10
15
20
25
30
35
40
Original
Fsdax
Blocked_fsdax
Pmdk
Blocked_pmdk


MAD2Bench I/O on NEXTGenIO
Runtime (s)
0
100
200
300
400
500
Original
Fsdax
Blocked_fsdax
Blocked_fsdax_fine_8
Pmdk
Blocked_pmdk
Blocked_pmdk_fine_8

Multi-level memory exploitation

- Simple image sharpening stencil
  - Each pixel replaced by a weighted average of its neighbours
  - weighted by a 2D Gaussian
  - averaged over a square region
  - we will use:
    - Gaussian width of 1.4
    - a large square region
  - then apply a Laplacian
    - this detects edges
    - a 2D second-derivative ∇^2
- Combine both operations
  - produces a single convolution filter
- 4 similar sized arrays, two that are updated and two that are source data



Multi-level memory exploitation

```
address = (int **) malloc(nx*sizeof(int *) + nx*ny*sizeof(int));
fuzzy = int2D(nx, ny, address);
```

```
pmemaddr1 = pmem_map_file(filename, array_size,PMEM_FILE_CREATE|PMEM_FILE_EXCL,
                          0666, &mapped_len1, &is_pmem)
fuzzy =  int2D(nx, ny, pmemaddr1);


int **int2D(int nx, int ny, int **idata){
  int i;
  idata[0] = (int *) (idata + nx);

  for(i=1; i < nx; i++){
      idata[i] = idata[i-1] + ny;
    }

  return idata;
}
```

- Read-only data in DRAM

```
Calculation time was 56.175083 seconds
Overall run time was 58.261385 seconds
DRAM required 285GB
```

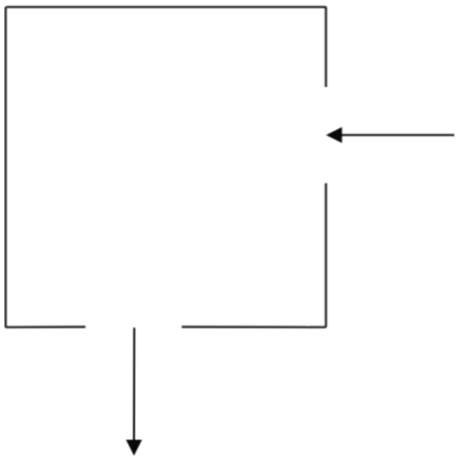
- Read-only data in Persistent Memory

```
Calculation time was 53.992465 seconds
Overall run time was 56.385472 seconds
DRAM required 170GB
```

Multi-level memory exploitation

- 2D CFD Stream function kernel

$$\nabla^2\Psi = \frac{\partial^2\Psi}{\partial x^2} + \frac{\partial^2\Psi}{\partial y^2} = 0$$



$$\Psi_{i-1,j} + \Psi_{i+1,j} + \Psi_{i,j-1} + \Psi_{i,j+1} - 4\Psi_{i,j} = 0$$

- Jacobi kernel updates the grid
  - Swap update and data arrays at each iterator

```
psinew[i][j] = 0.25*(psi[i+1][j] + psi[i-1][j] +
                                     psi[i][j+1] + psi[i][j-1])
```

Multi-level memory exploitation

```
strcpy(totalfilename,"/mnt/pmem_fsdax");
sprintf(totalfilename+strlen(totalfilename), "%d/", socket);
strncat(totalfilename, filename, strlen(filename));
sprintf(totalfilename+strlen(totalfilename), "%d", rank);

// total memory requirements including pointers
mallocsize = nx*sizeof(void *) + nx*ny*typesize;

if ((array2d = pmem_map_file(totalfilename, mallocsize,
                              PMEM_FILE_CREATE|PMEM_FILE_EXCL,
                              0666, mapped_len, &is_pmem)) == NULL) {
  perror("pmem_map_file");
  fprintf(stderr, "Failed to pmem_map_file for filename: %s\n",totalfilename);
  exit(-100);
}

void swap_pointers(double*** pa, double*** pb) {
    double** temp = *pa;
    *pa = *pb;
    *pb = temp;
}
```

```
No persist:         DRAM: 7.95 seconds        B-APM: 9.64 seconds
DRAM required:      40GB
Partial persist:    DRAM: 7.95 seconds        B-APM: 10.67 seconds
DRAM required:      25GB
Full persist:       DRAM: 7.95 seconds        B-APM: 41.84 seconds
DRAM required:      2GB
```

Architectual optimisation

- Single application performance key to users and developers
  - Very few systems are application specific
- Multi-purpose, multi-user systems require hardware choices
  - Processor, memory, accelerator, storage
  - Optimising for a range of applications hard
- A64FX one end of the spectrum
  - Small memory footprint for high performance/energy balance
- SGI UV2000 the other end of the spectrum
  - Very large memory footprint for shared memory/non-scaling applications
- Persist memory provides scope to optimise DRAM usage and I/O performance
  - Support low volume high performance memory
  - Support very high performance I/O
  - Enable application specialisation for memory performance
- Multi-tiered memory configurations
  - 3 tier memory structures to be investigated
    - HBM – DRAM – B-APM