Optimising CFD I/O through on-node non-volatile memory

Adrian Jackson

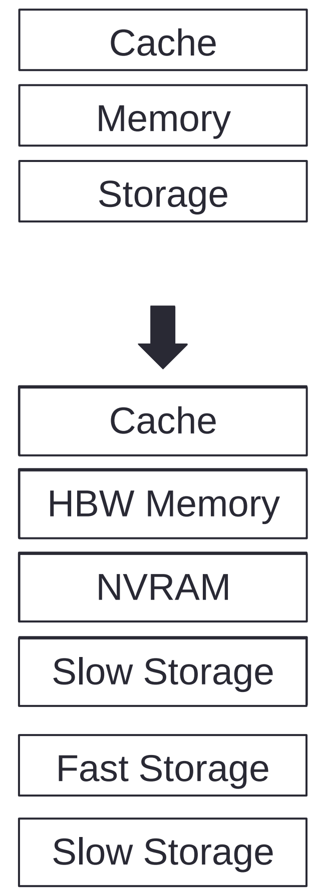
a.jackson@epcc.ed.ac.uk

THE UNIVERSITY *of* EDINBURGH

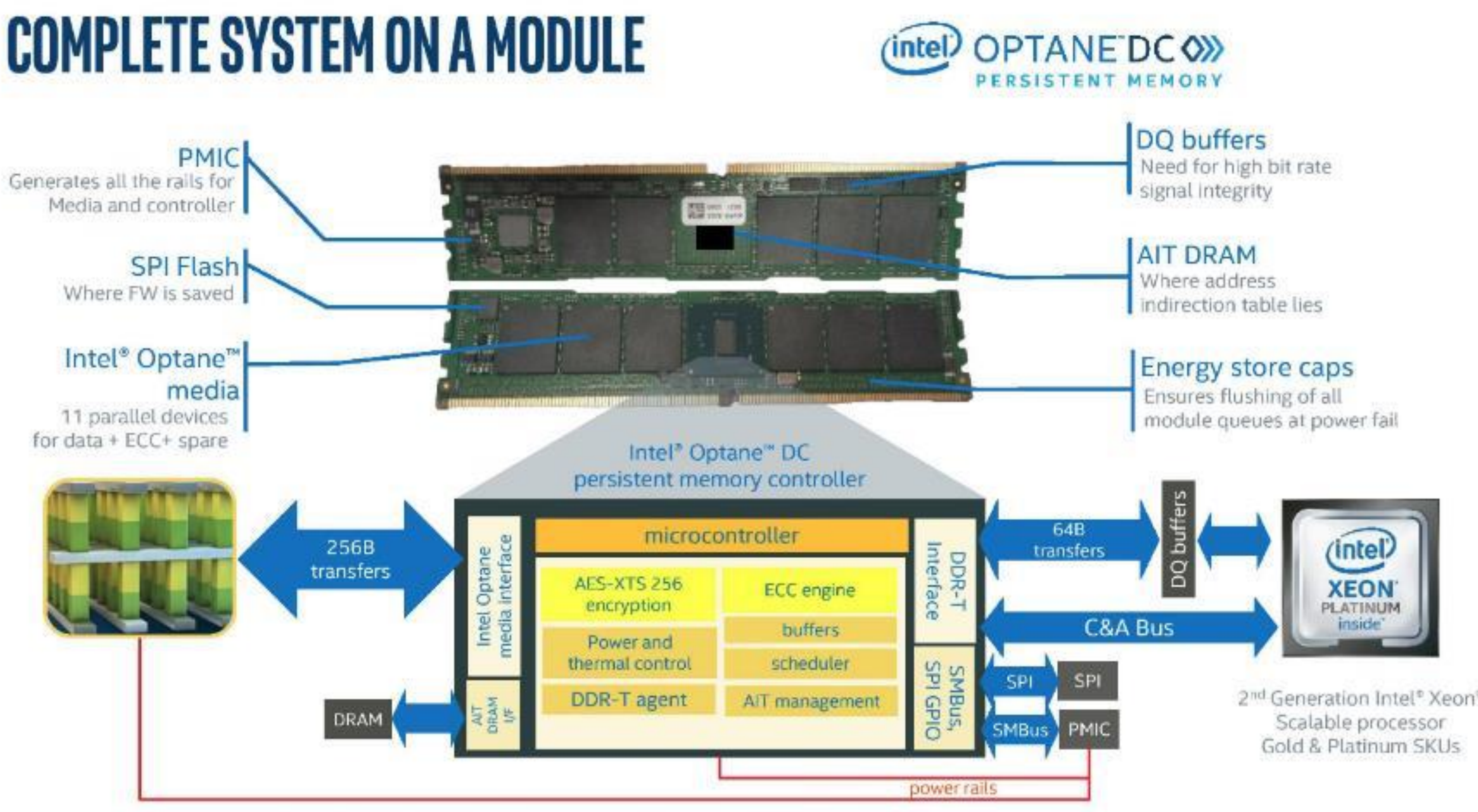
epcc

New Memory Hierarchies

- High bandwidth, on processor memory
  - Large, high bandwidth cache
  - Latency cost for individual access may be an issue
- Main memory
  - DRAM
  - Costly in terms of energy, potential for lower latencies than high bandwidth memory
- Byte-addressable Persistent Memory (B-APM)
  - High capacity, ultra fast storage
  - Low energy (when at rest) but still slower than DRAM
  - Available through same memory controller as main memory, programs have access to memory address space



Optane DCPMM



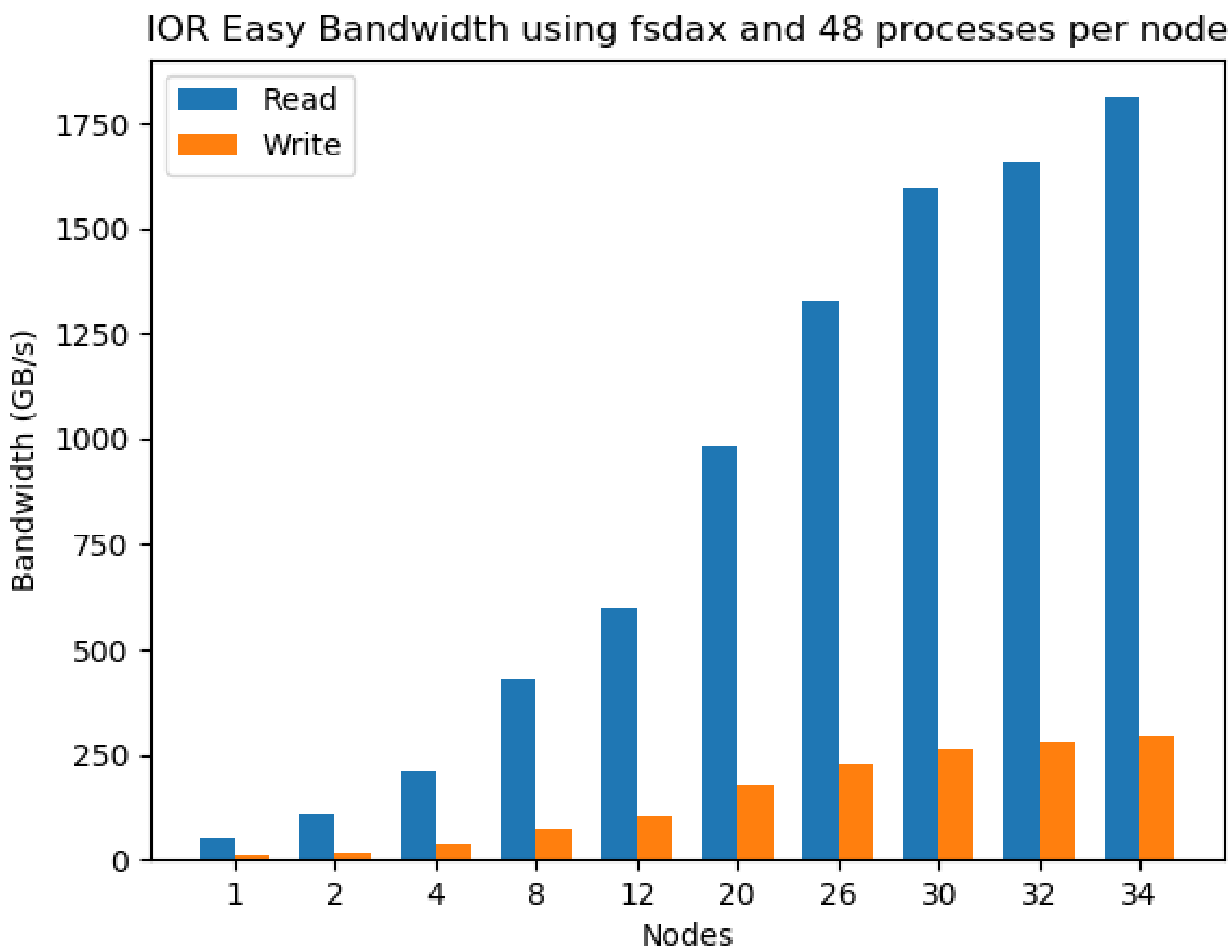
Capacity, performance, and persistence

- New memory technologies offer differing options for future memory hierarchies
  - DRAM for average volume, average bandwidth, average latency, high energy
  - HBM for lower volume, high bandwidth, average latency, very high energy
  - B-APM for very high volume, low bandwidth, high latency, low energy
- B-APM also offers persistence as a by-product of it’s underlying hardware
- B-APM also presents asymmetric performance
  - Higher bandwidth for reads

Hierarchical solutions

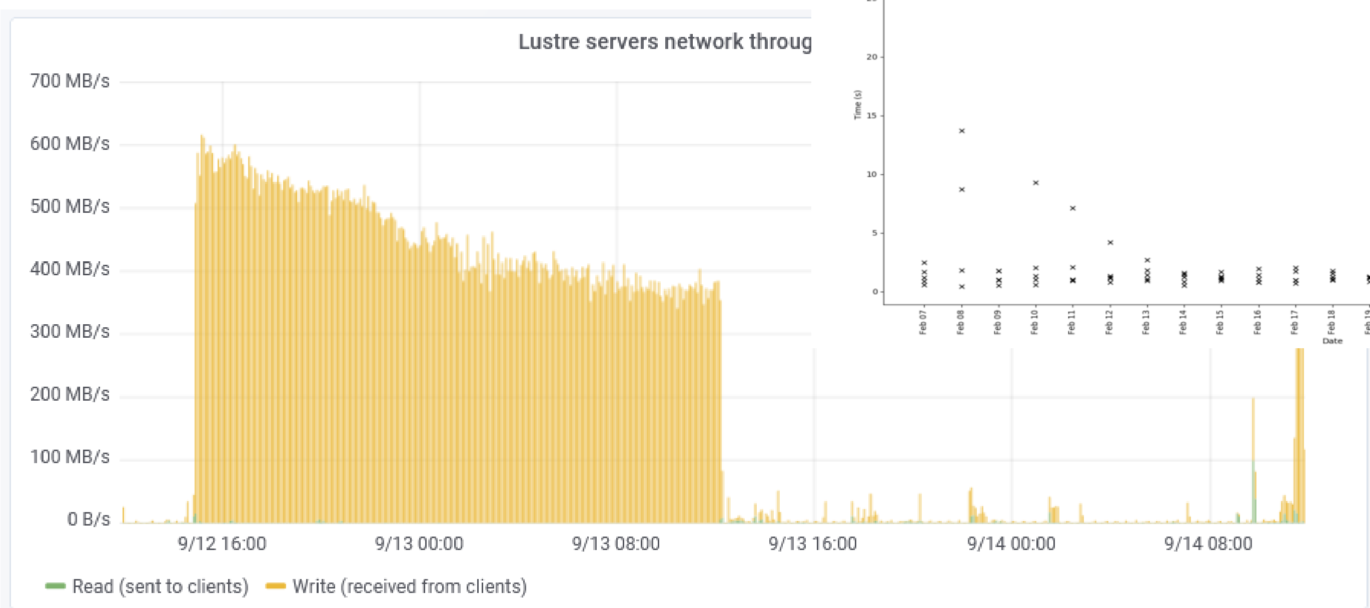
- Memory hierarchies offer automatic solutions for managing different types of memory with different performance characteristics
  - i.e. Intel Memory mode – 2 level-memory
  - Memory controller knows of the two levels of external memory
    - Fast and small memory is used as cache for slow and large memory
- This ignores the persistent functionality available in B-APM
  - Volatile in practise, even though the storage medium is persistent

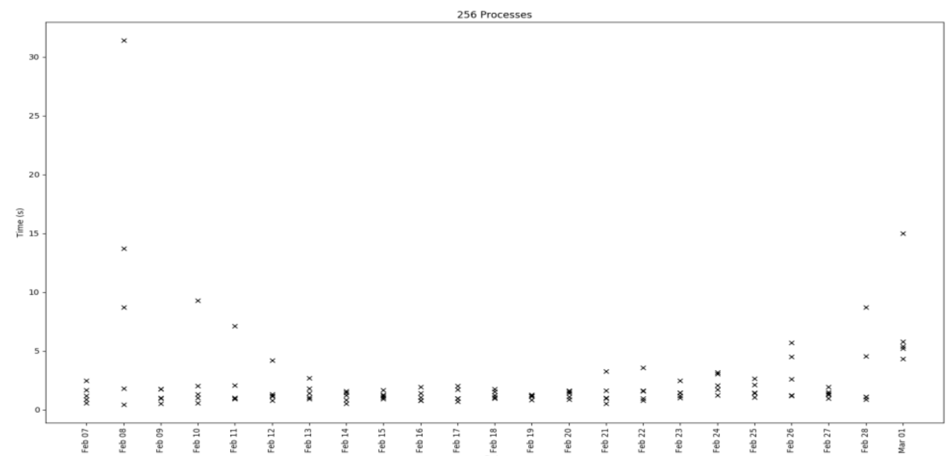
I/O Performance



B-APM potential

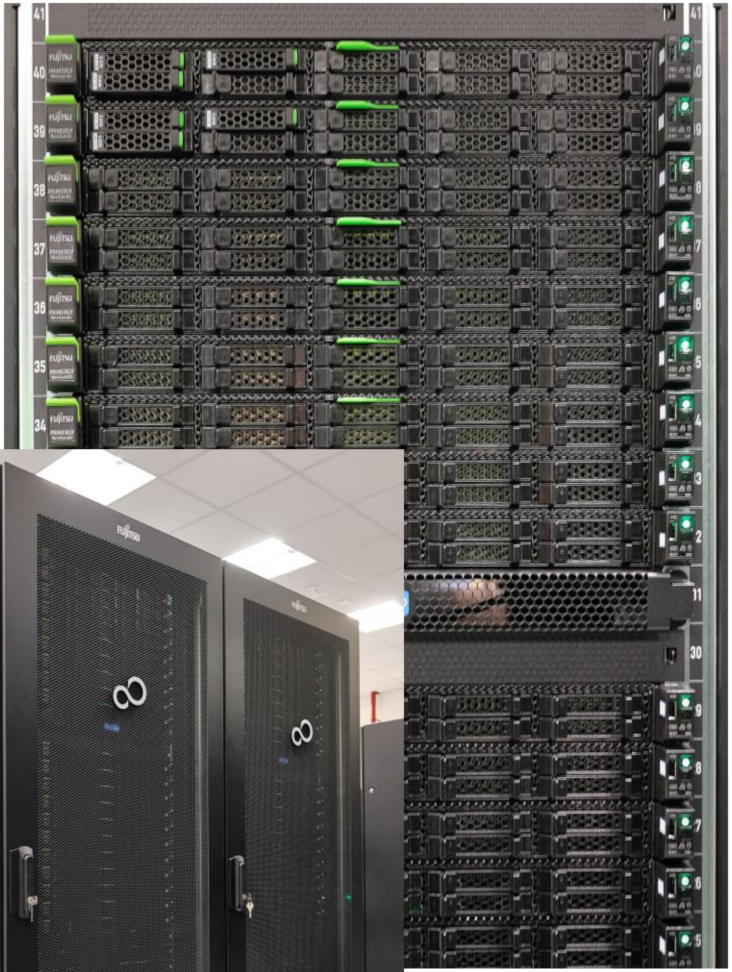
- Provide scalable storage hardware with compute nodes
- Localise performance variation to assigned nodes
- Challenges:
  - Data movement
  - Interfaces





NGIO Prototype

- 34 node cluster with 3TB of Intel DCPMM per node
  - 2 CPUS per node, each with 1.5TB of DCPMM and 96GB of DRAM
- External Lustre filesystem
- The EPCC NGIO system was funded by the European Union's Horizon 2020 Research and Innovation programme under Grant Agreement no. 671951.

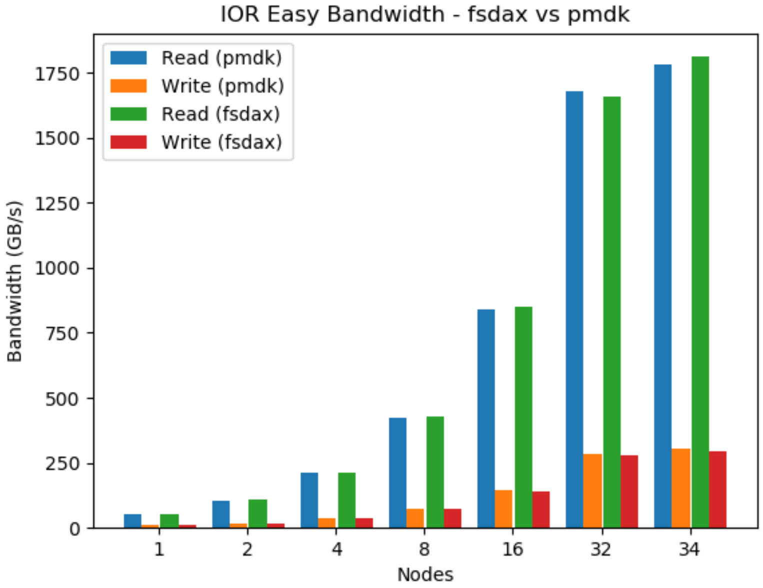


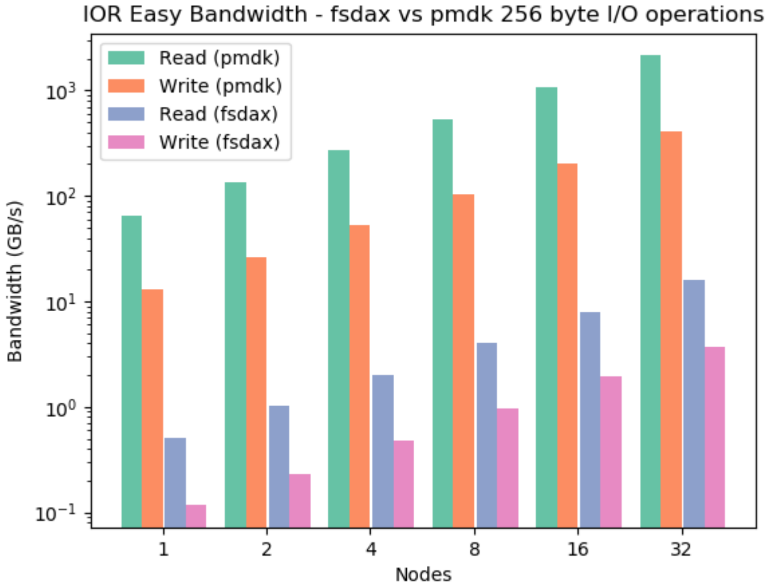
THE UNIVERSITY *of* EDINBURGH

epcc

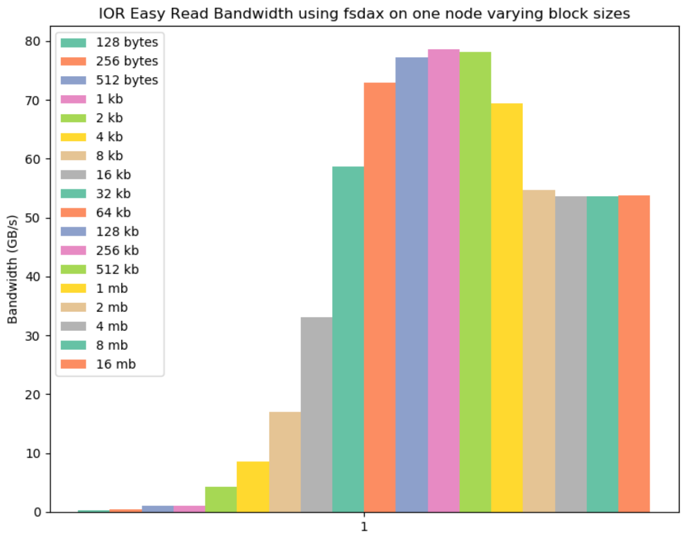
Move from I/O to Data

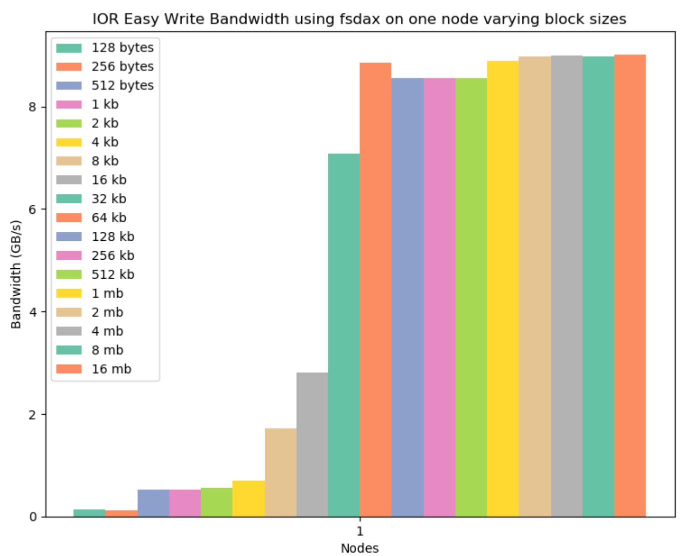
- Biggest potential for B-APM is removing the I/O interface
  - moving from I/O and application memory operations model, to just application operations
- Removing file (and block) operations

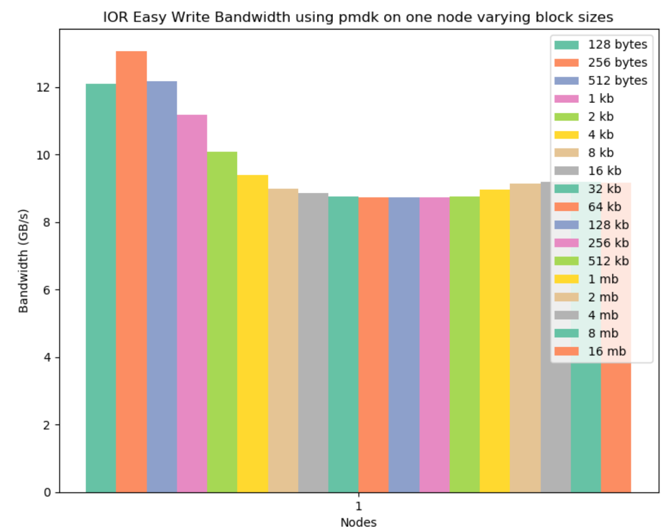




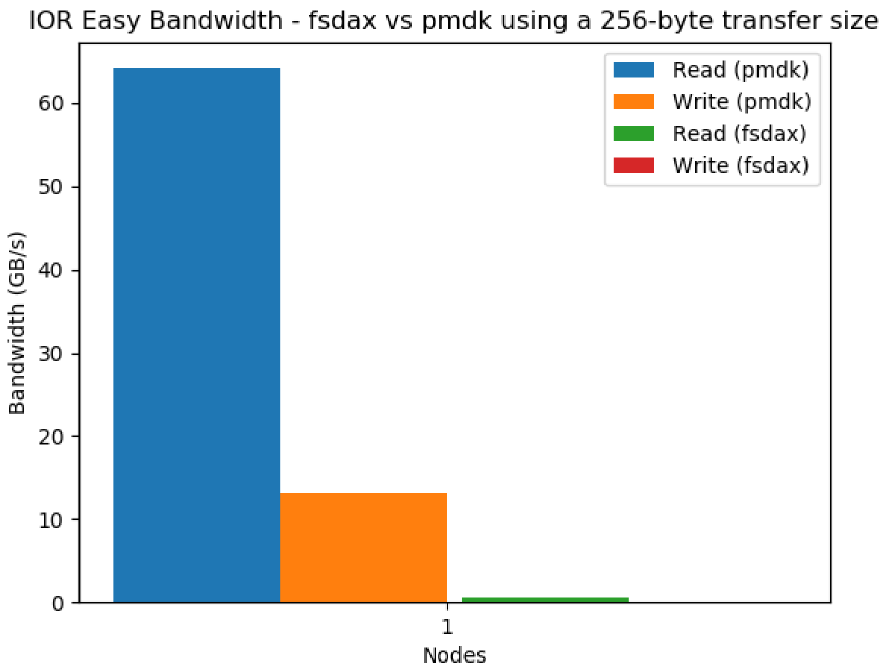
IOR - Data block sizes

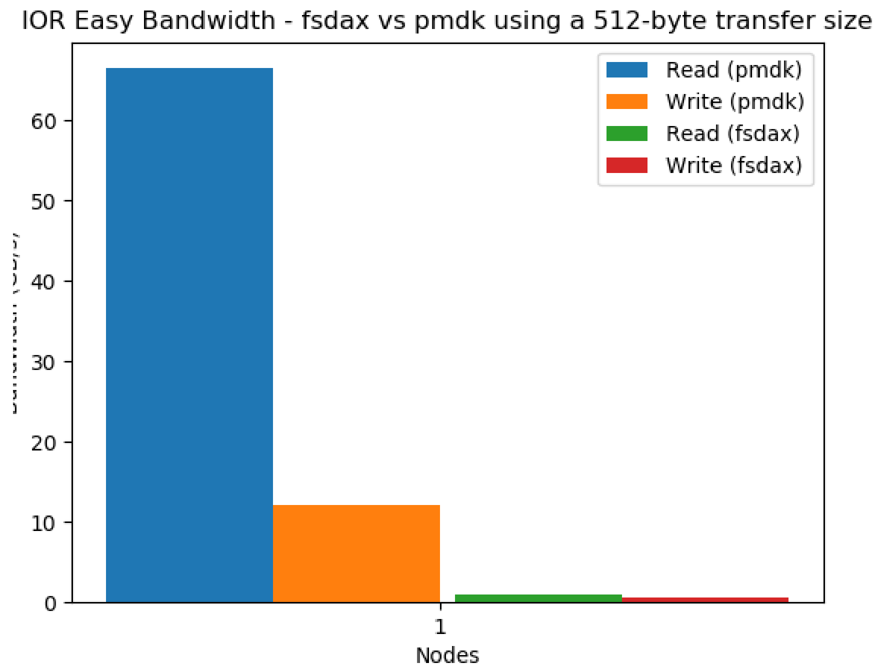






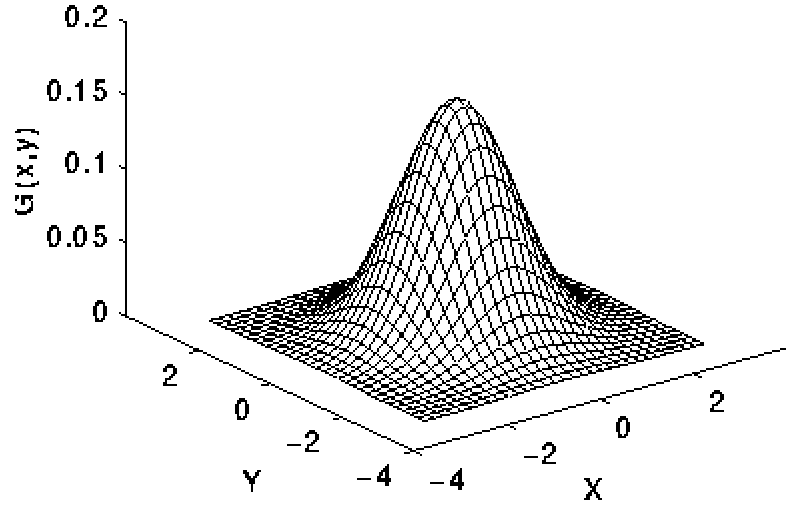
Data access sizes

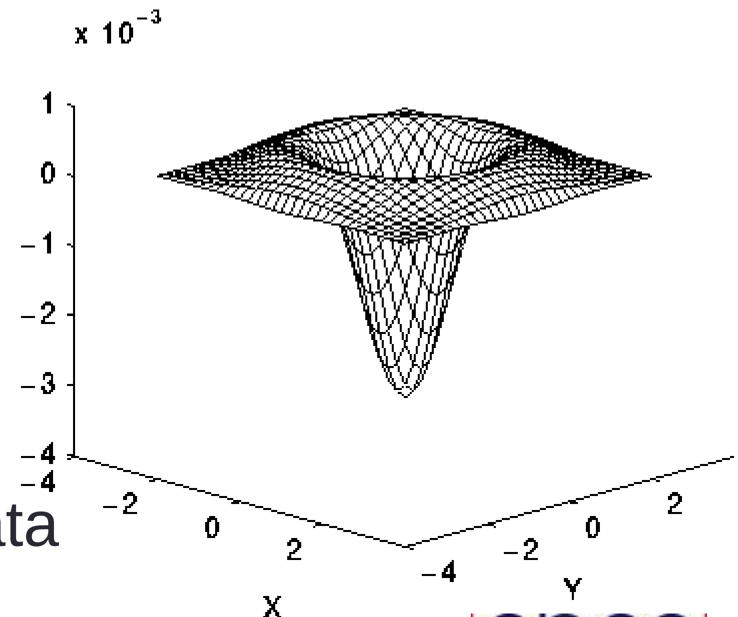




Multi-level memory exploitation

- 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
```

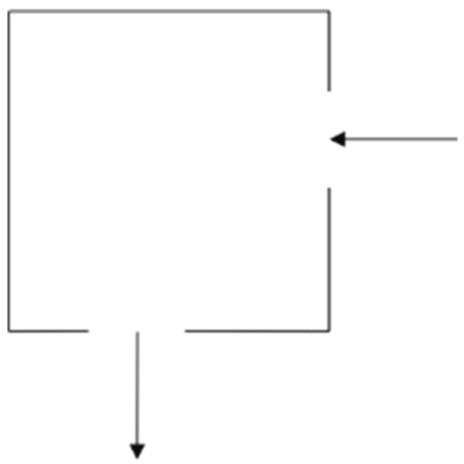
- Read-only data in B-APM

```
Calculation time was 53.992465 seconds
Overall run time was 56.385472 seconds
```

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

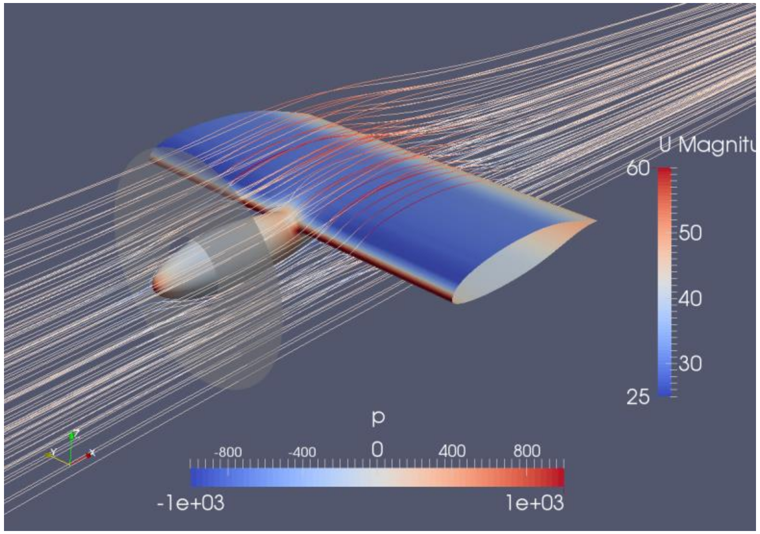
```
 totalfilename = (char *)malloc(1000*sizeof(char));

  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 persistence: DRAM: 7.95 seconds   B-APM: 9.64 seconds

Persistence: DRAM: 7.95 seconds   B-APM: 10.67 seconds

Performance – workflows

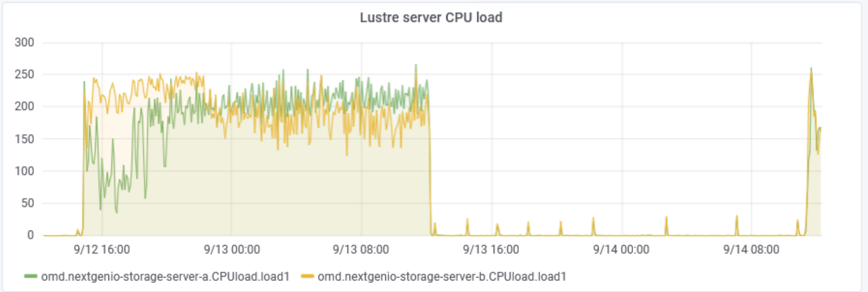
OpenFOAM simulation: *low-Reynolds number laminar turbulent transition modeling*
Input: mesh with ≈43M points
Stages: linear decomposition, parallel solver
768 MPI processes, 16 nodes
2 configurations:
① read/write to Lustre
② stage in, read/write on NVM, stage out

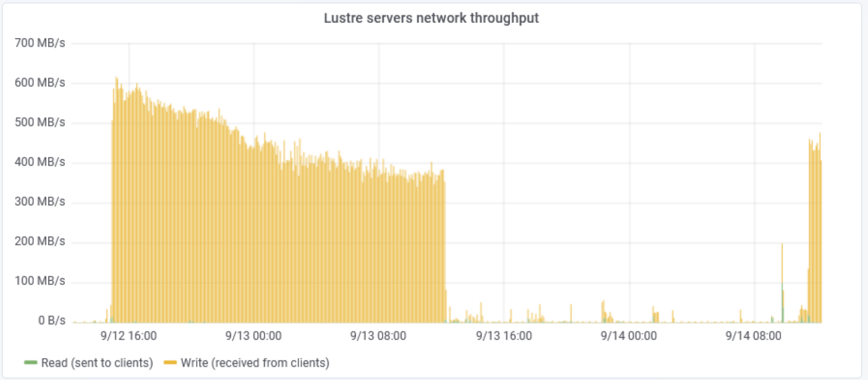


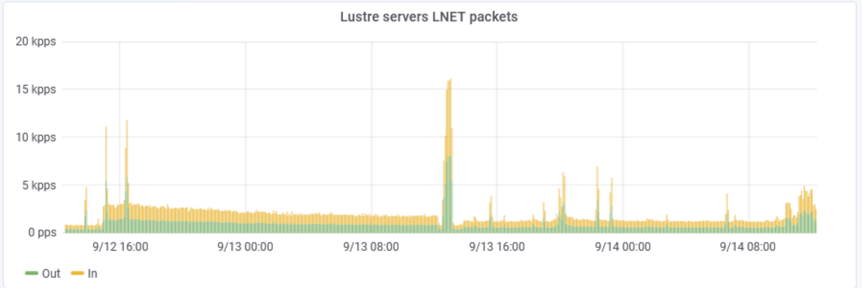
Performance benefits of data staging on OpenFOAM workflow						
	16 nodes, 768 MPI procs			20 nodes, 960 MPI procs		
Stage	Lustre	B-APM	Benefit	Lustre	B-APM	Benefit
decomposition	1191 secs	1105 secs	–	1841 secs	1453 secs	–
data staging	–	32 secs	–	–	330 secs	–
solver	123 secs	66 secs	46% faster	664 secs	78 secs	88% faster
Total	1314 secs	1203 secs	8% faster	2505 secs	1861 secs	25% faster

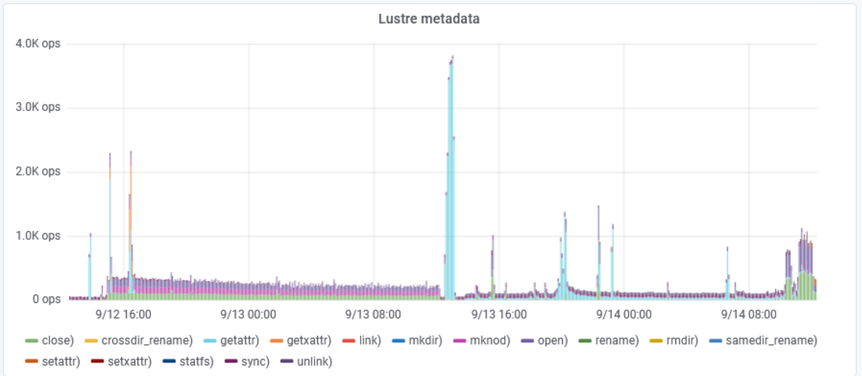
N3D/SEMTEX

- Small test case:
  - 72 processes
    - 900,000 files, 4.5 TBs produced
- Larger test case:
  - 512 processes
    - 6,400,000 files, 30 TBs produced
- Files required to transfer data from the forward phase to the adjoint phase
  - Velocity on each process at each time step









THE UNIVERSITY of EDINBURGH

N3D/SEMTEX

- Optimise by moving these temporary files to the B-APM
  - Use as files initially
  - Small case single iteration runtime:
    - Lustre: 8403 seconds
    - B-APM: 7365 seconds
  - Larger scale case single iteration runtime:
    - Lustre: 76872 seconds
    - B-APM: 36354 seconds
  - Next step to remove the files and use as memory only
    - Lots of small access should benefit from this optimisation

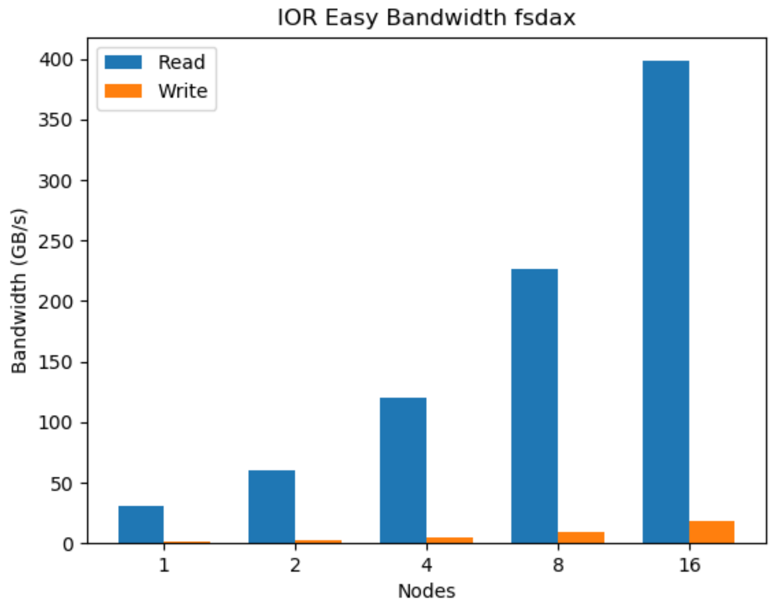
Performance - STREAM

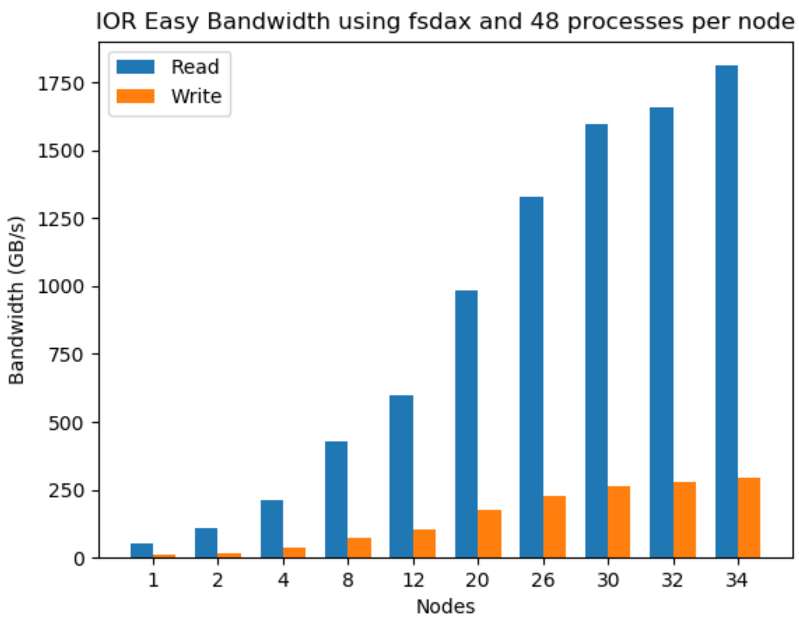
https://github.com/adrianjhpc/DistributedStream.git

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

```
STREAM_TYPE     *a, *b, *c;
pmemaddr = pmem_map_file(path, array_length,
                         PMEM_FILE_CREATE|PMEM_FILE_EXCL,
                         0666, &mapped_len, &is_pmem)
a = pmemaddr;
b = pmemaddr + (*array_size+OFFSET)*BytesPerWord;
c = pmemaddr + (*array_size+OFFSET)*BytesPerWord*2;

#pragma omp parallel for
for (j=0; j<*array_size; j++){
   a[j] = b[j]+scalar*c[j];
}
pmem_persist(a, *array_size*BytesPerWord);
```

NUMA issues



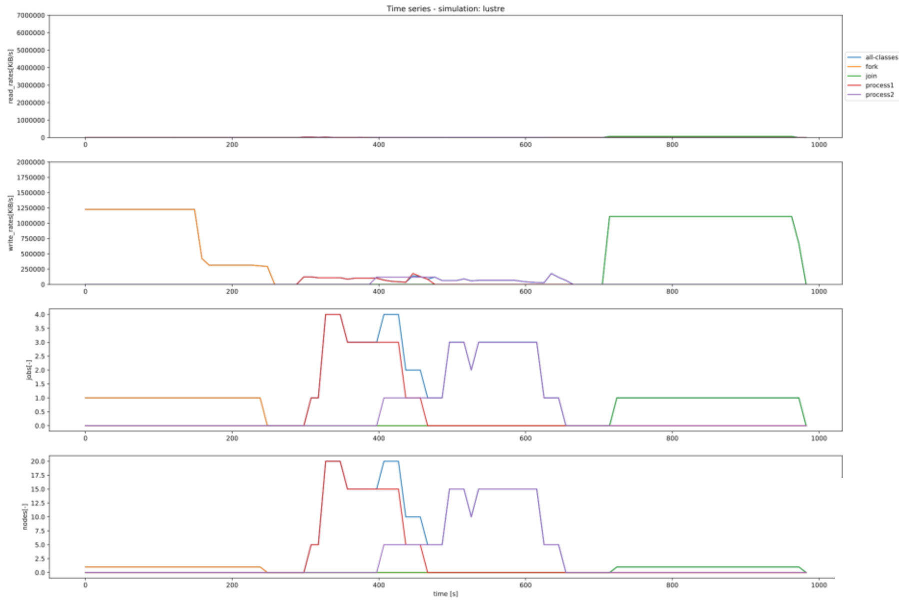


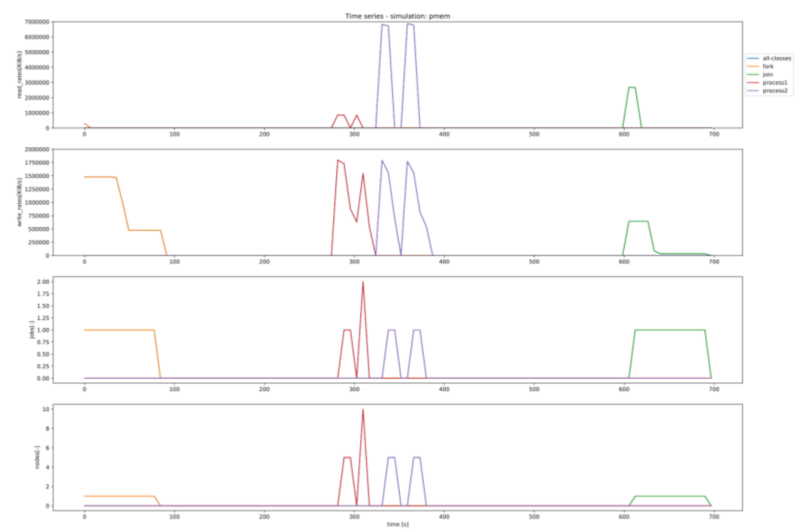
THE UNIVERSITY of EDINBURGH

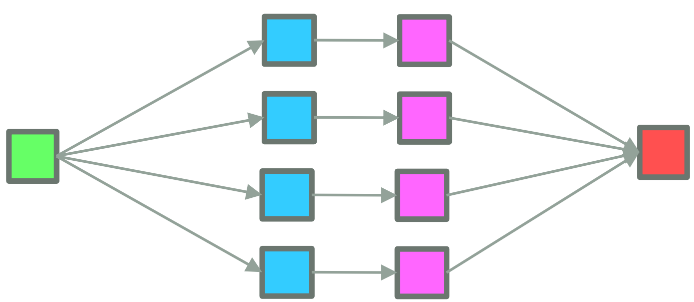
NUMA issues

```
unsigned long get_processor_and_core(int *socket, int *core){
  unsigned long a,d,c;
  __asm__ volatile("rdtscp" : "=a" (a), "=d" (d), "=c" (c));
  *socket = (c & 0xFFF000)>>12;
  *core = c & 0xFFF;
   return ((unsigned long)a) | (((unsigned long)d) << 32);;
}


strcpy(path,"/mnt/pmem_fsdax");
sprintf(path+strlen(path), "%d", socket/2);
sprintf(path+strlen(path), "/");
```

Performance - workflows







1 node
4 processes
4 files

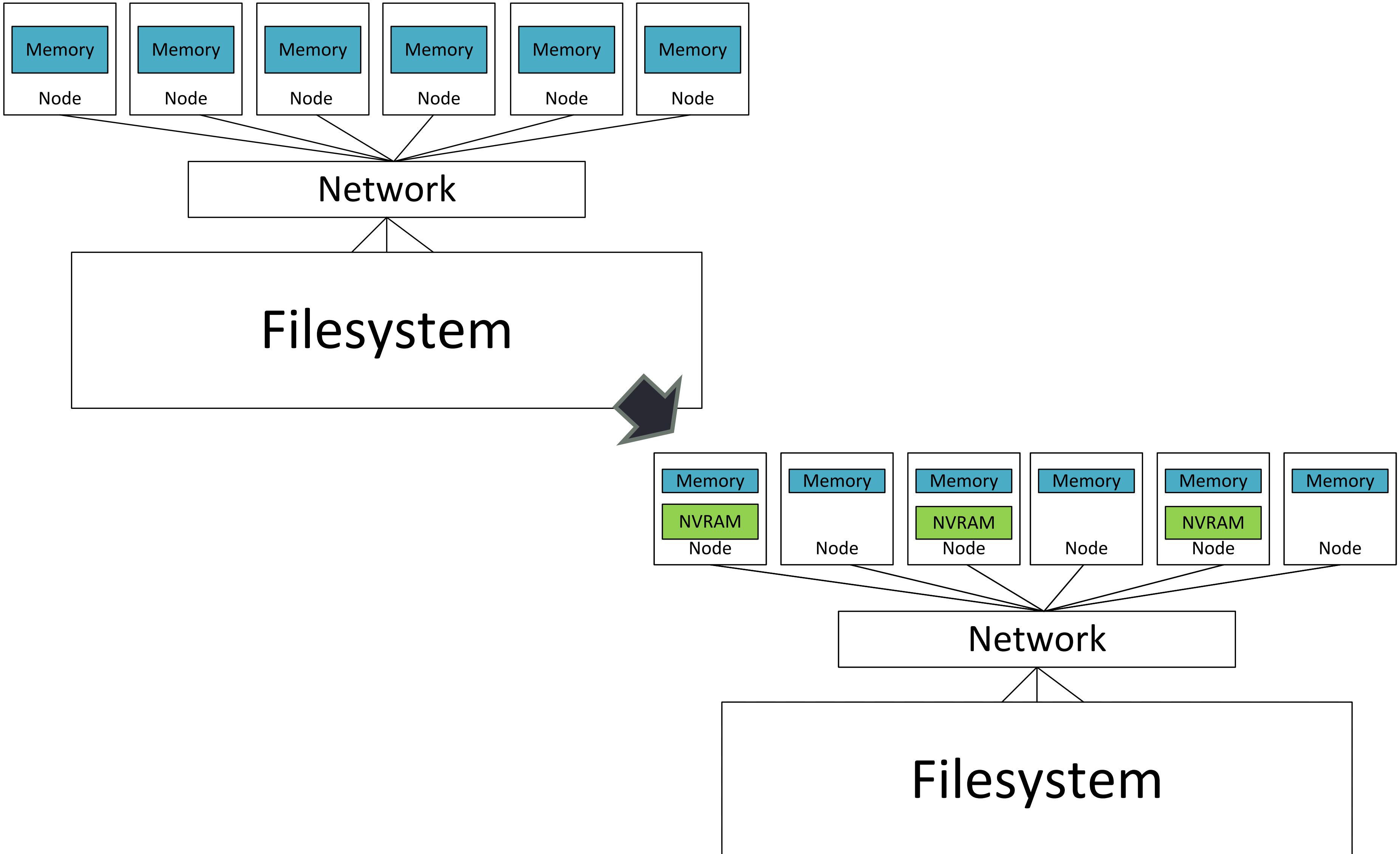
20 nodes
80 processes
80 files

1 node
4 processes
80 files

Persistent B-APM usage

- Strategy needed to recover data on failure
- Transactional approach
  - Use higher level pmem library functions
- Application logic
  - Using low level pmem functions
- Main focus is hardware failure
  - i.e. reboot but memory still intact
- Data resiliency another issue
  - What if an NVDIMM fails
  - Using low level pmem functionality there is no automatic redundancy
  - No RAIDing

Exploiting distributed storage



Optimising data usage

- Reducing data movement
  - Time and associated energy cost for moving data too and from external parallel filesystems
  - Move compute to data
- Considering full scientific workflow
  - Data pre-/post-processing
  - Multi-physics/multi-application simulations
  - Combined simulation and analytics
- Enable scaling I/O performance with compute nodes

Summary

- Multi-level memory offers the potential for
  - Merging I/O and memory operations into a single space
  - Reducing volatile memory requirements for system architectures
  - Removing I/O overheads and localising performance
- Enabling new technologies with HPC systems requires systemware support
- Transparently handling data for applications requires integration with job schedulers and data storage targets
- In-node B-APM is potentially very powerful for performance, but will require some changes to use efficiently (either at the systemware level or the application level)