Optimising Performance Through Data Localisation

Adrian Jackson
EPCC, The University of Edinburgh
a.jackson@epcc.ed.ac.uk
@adrianjhpc
NVRAM / B-APM
Persistent memory performance

IOR Easy Bandwidth using fsdax and 48 processes per node

Bandwidth (GB/s)

Nodes

1  2  4  8  12  20  26  30  32  34

Read
Write
Workflows

1 node  20 nodes  1 node
4 processes  80 processes  4 processes
4 files  80 files  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
Analyzing the Energy Cost of Data Movement in Scientific Applications

<table>
<thead>
<tr>
<th>Operation</th>
<th>Operation Energy Cost (nJ)</th>
<th>Equivalent ADD</th>
<th>Data Movement Energy (nJ)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>L1-&gt;REG</td>
<td></td>
<td></td>
<td>1.11</td>
</tr>
<tr>
<td>L2-&gt;REG</td>
<td></td>
<td></td>
<td>1.10</td>
</tr>
<tr>
<td>L3-&gt;REG</td>
<td></td>
<td></td>
<td>7.59</td>
</tr>
<tr>
<td>MEM-&gt;REG</td>
<td></td>
<td></td>
<td>53.84</td>
</tr>
<tr>
<td>Stall</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Prefetching</td>
<td></td>
<td></td>
<td>65.08</td>
</tr>
</tbody>
</table>
https://github.com/adrianjhpc/DistributedStream.git

<table>
<thead>
<tr>
<th>Mode</th>
<th>Min BW (GB/s)</th>
<th>Median BW (GB/s)</th>
<th>Max BW (GB/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>App Direct (DRAM)</td>
<td>142</td>
<td>150</td>
<td>155</td>
</tr>
<tr>
<td>App Direct (DCPMM)</td>
<td>32</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td>Memory mode</td>
<td>144</td>
<td>146</td>
<td>147</td>
</tr>
<tr>
<td>Memory mode (large)</td>
<td>12</td>
<td>12</td>
<td>12</td>
</tr>
</tbody>
</table>

![Bandwidth Chart](chart.png)
Data access sizes
MAD2Bench I/O on ARCHER2

Runtime (s)

Original

Blocked
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 $\nabla^2$
- 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);

pmmaddr1 = pmem_map_file(filename, array_size, PMEM_FILE_CREATE|PMEM_FILE_EXCL, 0666, &mapped_len1, &is_pmem);
fuzzy = int2D(nx, ny, pmmaddr1);

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

\[ \text{psinew}[i][j] = 0.25*(\text{psi}[i+1][j] + \text{psi}[i-1][j] + \text{psi}[i][j+1] + \text{psi}[i][j-1]) \]
Multi-level memory exploitation

```c
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;
}
```

<table>
<thead>
<tr>
<th>No persist:</th>
<th>DRAM: 7.95 seconds</th>
<th>B-APM: 9.64 seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>DRAM required:</td>
<td>40GB</td>
<td></td>
</tr>
<tr>
<td>Partial persist:</td>
<td>DRAM: 7.95 seconds</td>
<td>B-APM: 10.67 seconds</td>
</tr>
<tr>
<td>DRAM required:</td>
<td>25GB</td>
<td></td>
</tr>
<tr>
<td>Full persist:</td>
<td>DRAM: 7.95 seconds</td>
<td>B-APM: 41.84 seconds</td>
</tr>
<tr>
<td>DRAM required:</td>
<td>2GB</td>
<td></td>
</tr>
</tbody>
</table>
Architectural 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