MEMORY ON THE KNL

Adrian Jackson

adrianj@epcc.ed.ac.uk

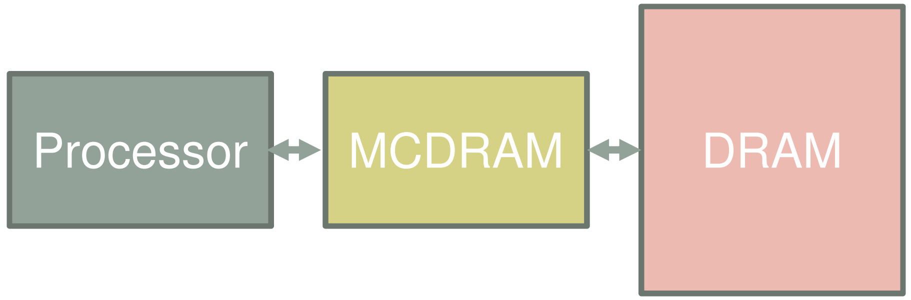
@adrianjhpc

Some slides from Intel Presentations

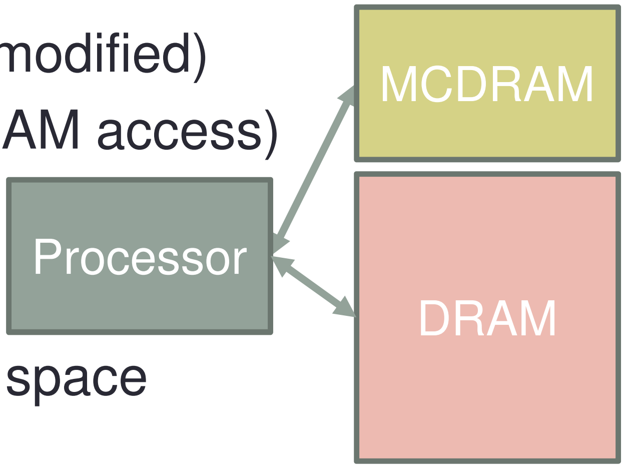
Memory

- Two levels of memory for KNL
  - Main memory
    - KNL has direct access to all of main memory
    - Similar latency/bandwidth as you'd see from a standard processors
    - 6 DDR channels
  - MCDRAM
    - High bandwidth memory on chip: 16 GB
    - Slightly higher latency than main memory (~10% slower)
    - 8 MCDRAM channels
  - Cluster modes have impact on memory performance
    - In fact, this is all cluster modes do

Memory Modes



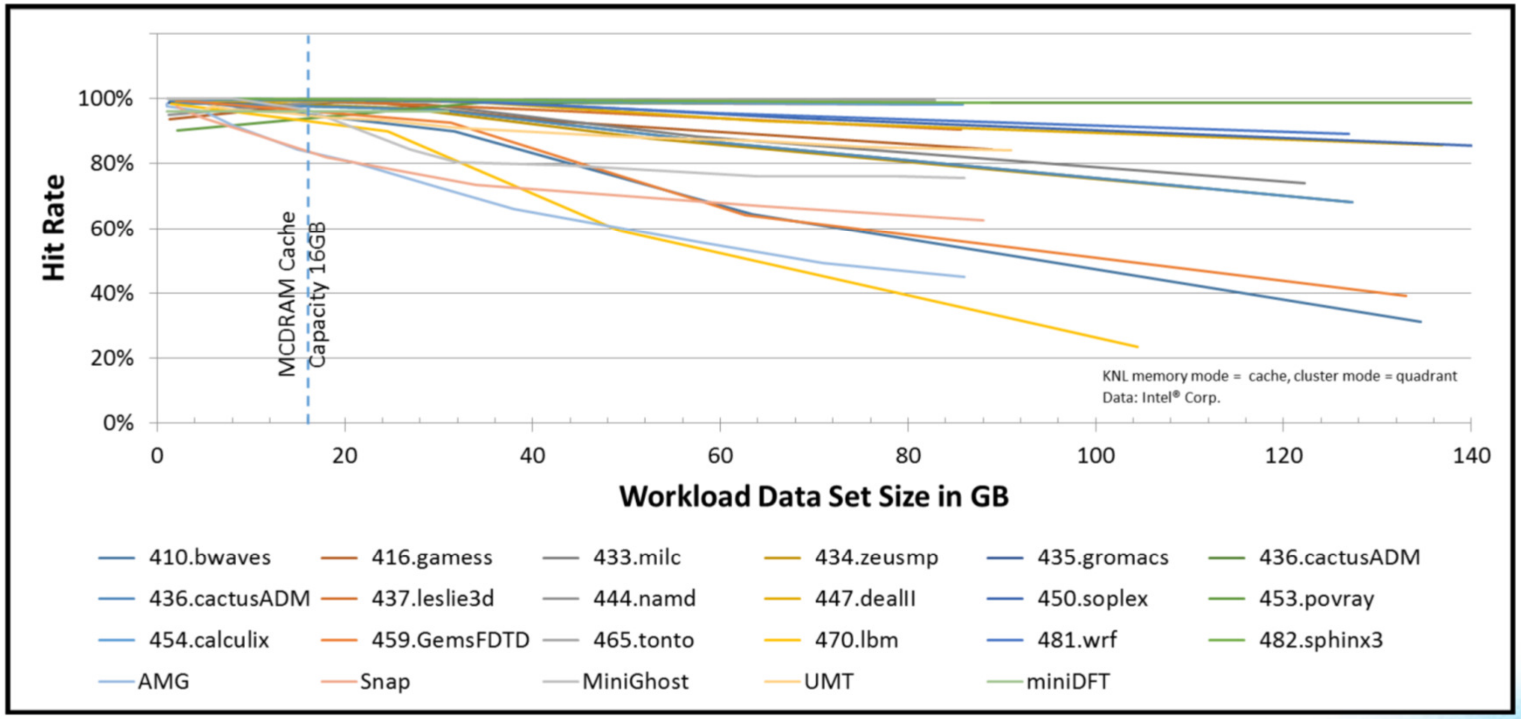
- Cache mode
  - MCDRAM cache for DRAM
  - Only DRAM address space
  - Done in hardware (applications don't need modified)
  - Misses more expensive (DRAM and MCDRAM access)



- Flat mode
  - MCDRAM and DRAM are both available
  - MCDRAM is just memory, in same address space
  - More overall memory available
  - Software managed (applications need to do it themselves)
- Hybrid – Part cache/part memory
  - 25%, 50%, or 75% cache, the rest flat

Cache mode

MCDRAM Cache Hit Rate



MCDRAM performs well as cache for many workloads
Enables good out-of-box performance without memory tuning

Using flat mode

- Two memory spaces are available
  - By default no MCDRAM is used

- Using MCDRAM without changing an application
  - Set bulk memory policy
    - Preferred or enforced memory for application
    - MCDRAM exposed as NUMA node
    - Use `numactl` program

numactl in flat mode

- Example code:
  - Check available memory

```
[adrianj@esknl1 ~]$ aprun -n 1 numactl --hardware
available: 2 nodes (0-1)
node 0 cpus: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26
27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54
55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82
83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107
108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128
129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149
150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170
171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191
192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212
213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233
234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254
255
node 0 size: 98178 MB
node 0 free: 81935 MB
node 1 cpus:
node 1 size: 16384 MB
node 1 free: 15924 MB
node distances:
node   0   1
  0:  10  31
  1:  31  10
```

numactl in flat mode

- Specifying memory policy
  - Mandating the memory to use
    - -m or --membind=
    - Fails if exhausts memory

```
aprun -n 64  numactl -m 1 ./castep.mpi forsterite
```

  - Setting the preferred memory
    - -p or --preferred=
    - Tries to used preferred memory, falls back if exhausts memory

```
aprun -n 64  numactl -p 1 ./castep.mpi forsterite
```

- With `aprun` the executable needs to be on `/work` if using `numactl`
  - `aprun` will copy the executable from `/home` if run normally, but with `numactl` `aprun` is actually running `numactl` not your executable so won't copy it automatically for you

Memkind and hbwmalloc

- Open source software available that can manage MCDRAM
  - memkind and hbwmalloc
  - https://github.com/memkind
- memkind
  - built on jemalloc
  - heap manager
  - allows defining memory “kind”s
- hbwmalloc
  - Implements memory model for knl using memkind
  - Predefines kind options for easy knl usage
  - Can have selective fallback behaviour
  - Can target different page sizes (2MB – 1GB)

hbwmalloc for C and C++

- Allocate arrays in C:

```
#include<hbwmalloc.h>
…
double *a = (double *) malloc(sizeof(double)*2048);
double *b = (double *) hbw_malloc(sizeof(double)*2048);
```

  - Need to link to memkind library
  - Automatic variables/arrays are in main memory
    - May need to restructure data to deal with this
  - Also need to free with:
    - `hbw_free`
  - Also `hbw_calloc`
- Fallback
  - If not enough memory available it will fallback
  - Three different options
    - HBW_POLICY_BIND
      - Fail
    - HBW_POLICY_PREFERRED (default)
      - Fallback to main memory
    - HBW_POLICY_INTERLEAVE
      - Interleave pages between MCDRAM and DDR
  - `hbw_set_policy()` sets the current fallback policy
  - `hbw_get_policy()` gets the current fallback policy

hbwmalloc for C and C++

- Can check for hbw from code

```
hbw_check_available()
```

- Can specify page size

```
hbw_posix_memalign_psize()
HBW_PAGESIZE_4KB
HBW_PAGESIZE_2MB
HBW_PAGESIZE_1GB
```

- Allocating in C++
  - STL allocator

```
#include<hbw_allocator.h>
…
std::vector<double, hbw::allocator<double> > a(100);
```

- Need to link to memkind for this too

MCDRAM from Fortran

- Intel directive for allocatable arrays

```
!DIR$ ATTRIBUTES FASTMEM :: …
```

- i.e.

```
real, allocatable(:) :: a, b
 …
!dir$ attributes fastmem :: a
…
allocate(a(1000))
allocate(b(20))
```

- Need to link with memkind otherwise will still use main memory and **not tell you**
- Only works for allocatable variables
- Also some restrictions on where it can be used (this may change/may have changed)
  - Not on components of derived types (might now be available in v17)
  - Not in common blocks

MCDRAM from Fortran

- Intel
  - `FASTMEM` is Intel directive
  - Only works for allocatable arrays
- Cray CCE:

```
!dir$ memory(attributes)
#pragma memory(attributes)
```

- Placed before allocation and deallocation statements
  - Fortran: allocate, deallocate (optional)
  - C/C++: malloc, calloc, realloc, posix_memalign, free
  - C++: new, delete, new[], delete[]
- Directive on deallocation must match (C/C++ only)
- Converts normal allocation to high-bandwidth memory
- The `bandwidth` attribute maps to MCDRAM

Cray CCE MCDRAM Support

```
integer, dimension(:,:), allocatable :: A
!dir$ memory(bandwidth) B
integer :: B(N)

!dir$ memory(bandwidth)
allocate(A(N,N))
```

- Allocation will fail (in above examples) if MCDRAM is unavailable/exhausted
- More general form can deal with this:

  `!dir$ memory([fallback,] attributes)`

  - i.e. `!dir$ memory(fallback,bandwidth)` will fall back to DDR if MCDRAM isn't available

hbw_malloc from Fortran

- If don’t have access to Cray compiler and want to do more than Intel compiler can do
  - Or if your Fortran code already uses malloc
- Wrapped hbw_malloc
  - Call malloc directly in Fortran
  - https://github.com/jeffhammond/myhbwmalloc

```
use fortran_hbwmalloc
include 'mpif.h'
integer offset_kind
parameter(offset_kind=MPI_OFFSET_KIND)
integer(kind=offset_kind) ptr
INTEGER(C_SIZE_T) param
type(C_PTR) localptr

real (kind=8) r8
pointer (pr8, r8)
if (type.eq.'r8') then
      param = 8*dim
      localptr = hbw_malloc(param)
else if (type.eq.'i4') then
      param = 4*dim
      localptr = hbw_malloc(param)
end if
ptr = transfer(localptr,ptr)
if (type.eq.'r8') then
      call c_f_pointer(localptr, pr8)
      call zeroall(dim,r8)
end if
```

autohbw

- Automatic allocator, part of memkind
  - Intercepts heap allocates and performs them on MCDRAM
  - Can be tuned for different size limits
  - Link to the library and set environment variables:

```
export AUTO_HBW_SIZE=1K:5K
export AUTO_HBW_SIZE=10K
```

  - Needs to be linked before system libraries, i.e. LD_PRELOAD or `equivalent`

mkl

- mkl 2017 will automatically try to use MCDRAM
  - Unlimited access to MCDRAM

- Restricting how much MCDRAM mkl uses is possible
  - Environment variable (in MB):

```
MKL_FAST_MEMORY_LIMIT=40
```

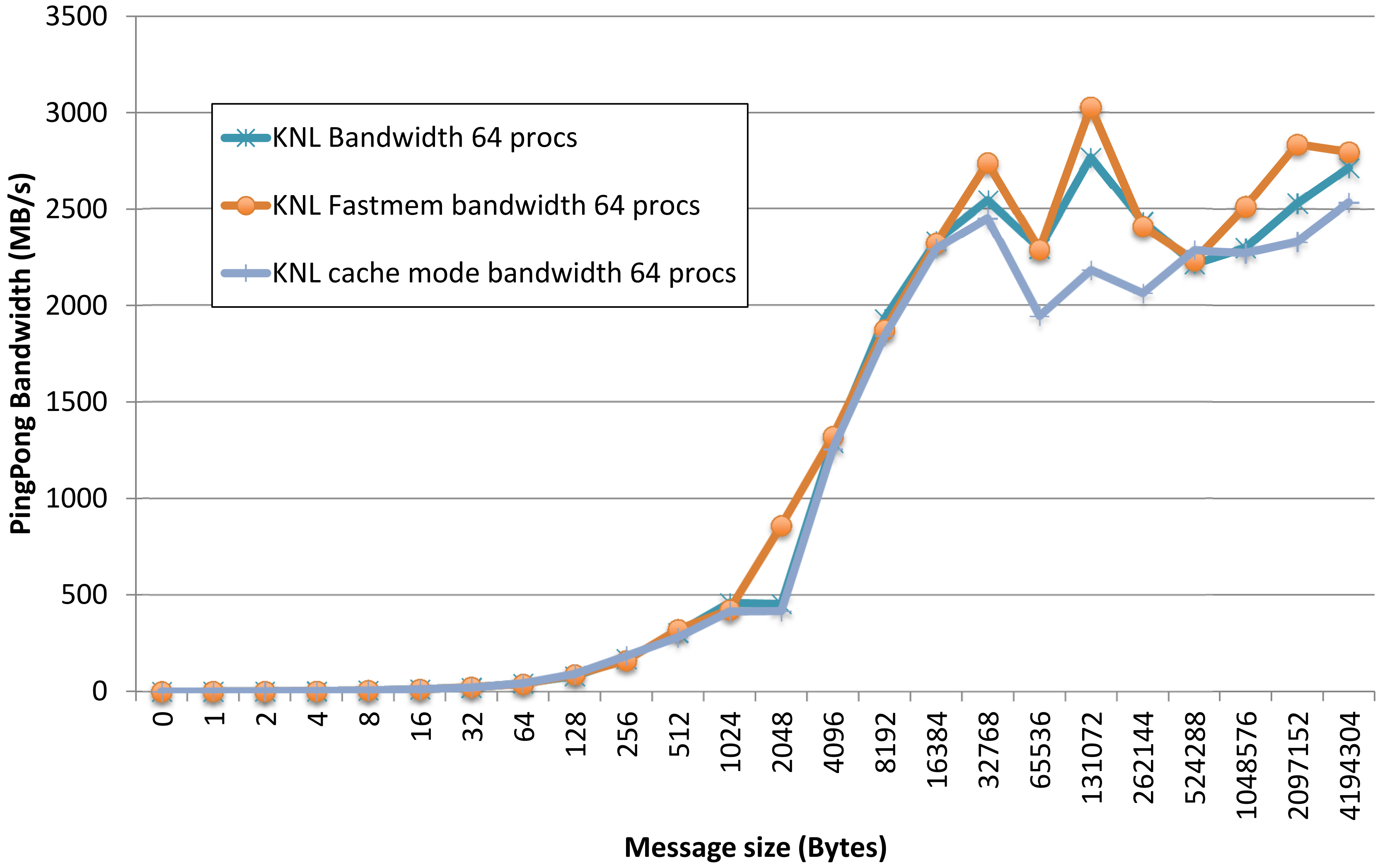
  - Function call (in MB):

```
mkl_set_memory_limit(MKL_MEM_MCDRAM, 40)
```

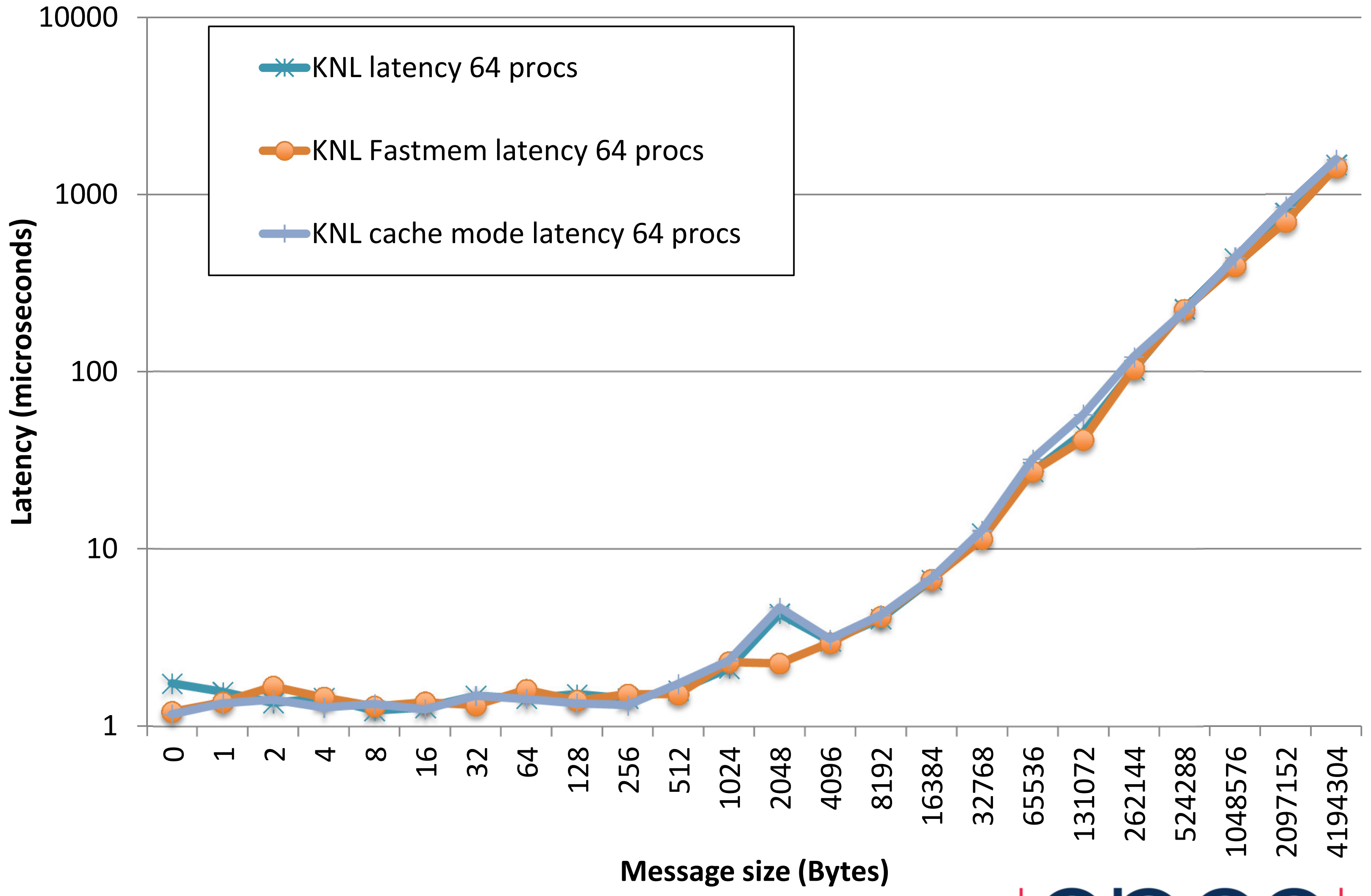
MCDRAM and the MPI library

- In cache mode MPI calls will be using MCDRAM
- In flat mode, it depends…
  - Can impact performance
- Intel MPI (not on ARCHER) starting to take this into account (2017 version)

```
I_MPI_HBW_POLICY
```



KNL Bandwidth 64 procs
KNL Fastmem bandwidth 64 procs
KNL cache mode bandwidth 64 procs
PingPong Bandwidth (MB/s)
3500
3000
2500
2000
1500
1000
500
0
0
1
2
4
8
16
32
64
128
256
512
1024
2048
4096
8192
16384
32768
65536
131072
262144
524288
1048576
2097152
4194304
Message size (Bytes)



10000
1000
100
10
1
Latency (microseconds)
KNL latency 64 procs
KNL Fastmem latency 64 procs
KNL cache mode latency 64 procs
0
1
2
4
8
16
32
64
128
256
512
1024
2048
4096
8192
16384
32768
65536
131072
262144
524288
1048576
2097152
4194304
Message size (Bytes)

Emulating MCDRAM

- Using multiprocessor node can emulate MCDRAM affect on application:
  - `export MEMKIND_HBW_NODES=0`
  - `mpirun -n 64  numactl -m 1 -N 0 ./my_application`
- Force all application memory to be allocated on the memory of the other processor
  - HBW memory will be allocated on the local memory
  - This will have lower latency, which isn't accurate
  - But will give an idea of the impact of higher bandwidth

Memory modes

- Basic KNL memory setup is simple
  - For a lot of applications cache mode is sufficient
- If you want to see MCDRAM impact numactl can easily enable MCDRAM interaction
  - Need flat mode
- Finer grained programing possible
  - New malloc routine (hbw_malloc)
  - Both Intel and Cray compilers have Fortran compatibility
    - Cray is a bit more developed than the Intel stuff so far
  - Can call malloc from Fortran if required, i.e. if compiling with gfortran...