Welcome

Virtual tutorial starts at 15.00 BST

archer

epcc

THE UNIVERSITY OF EDINBURGH

Using KNL on ARCHER

Adrian Jackson

adrianj@epcc.ed.ac.uk

@adrianjhpc

With thanks to:

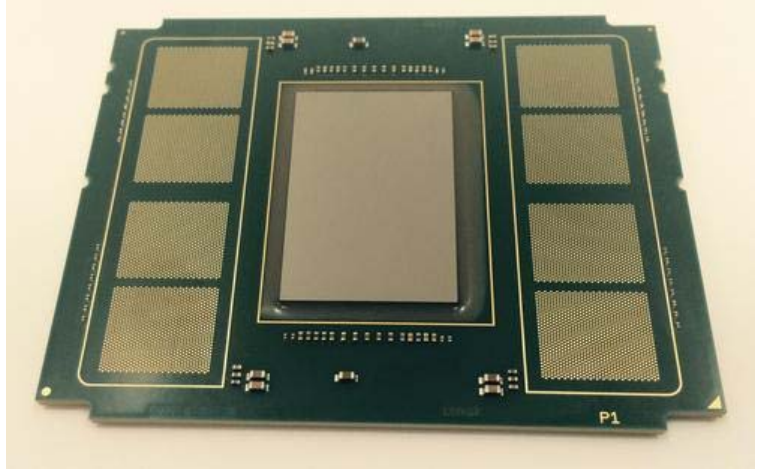
Harvey Richardson

from Cray

Slides from Intel

Xeon Phi – Knights Landing (KNL)

- Intel's latest many-core processor
  - Knights Landing
  - 2nd generation Xeon Phi
- Successor to the Knights Corner
  - 1st generation Xeon Phi
- New operation modes
- New processor architecture
- New memory systems
- New cores



archer

epcc

THE UNIVERSITY OF EDINBURGH

ARCHER KNL

- 12 KNLs in test system
- Should be available mid-October
  - ARCHER users get access
  - Non-ARCHER users can get access through driving test
- Initial access will be unrestricted
  - After first month usage will be charged
- Each node has
  - 1 x Intel(R) Xeon Phi(TM) CPU 7210 @ 1.30GHz
    - 64 core/4 hyperthreads
    - 16GB MCDRAM
  - 96GB DDR4@2133 MT/s

Running applications on the XC40

- 256 threads per KNL processor
  - Numbering wraps, i.e. 0-63 the hardware cores, 64-127 wraps onto the cores again, etc…
  - Meaning core 0 has threads 0,64,128,192, core 1 has threads 1,64,129,193, etc…
- Use PBS and `aprun` as in ARCHER
  - Standard PBS script, with one extra for selecting memory/communication setup (more later)
  - Standard `aprun`, run 64 MPI processes on the 64 KNL cores:

```
aprun –n 64 ./my_app
```

Running applications on the XC40

- For hyperthreading (using more than 64 cores):

```
OMP_NUM_THREADS=4
aprun -n 64 -d 4 -cc depth -j 4 ./my_app
```

- Should also be possible to control thread placement with `OMP_PROC_BIND`:

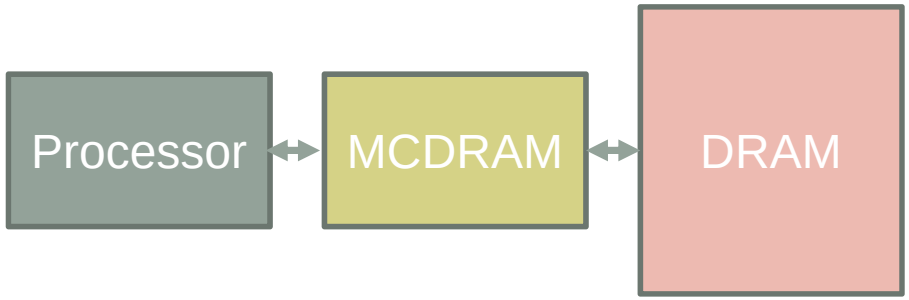
```
OMP_PROC_BIND=true
OMP_NUM_THREADS=4
aprun -n 64 -cc none -j 4 ./my_app
```

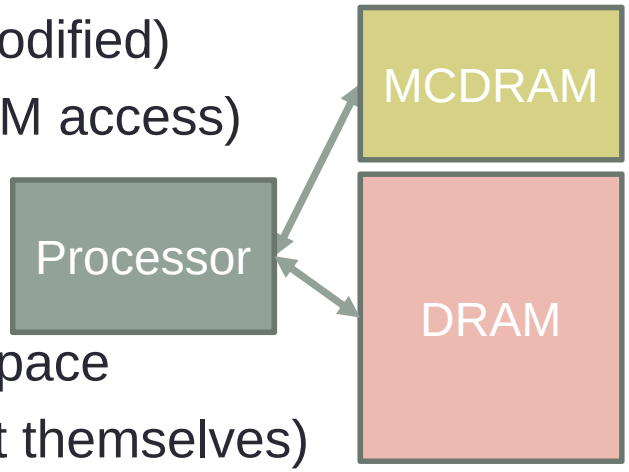
Memory

- 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 (~10-15% slower)
    - 8 MCDRAM channels

Memory Modes

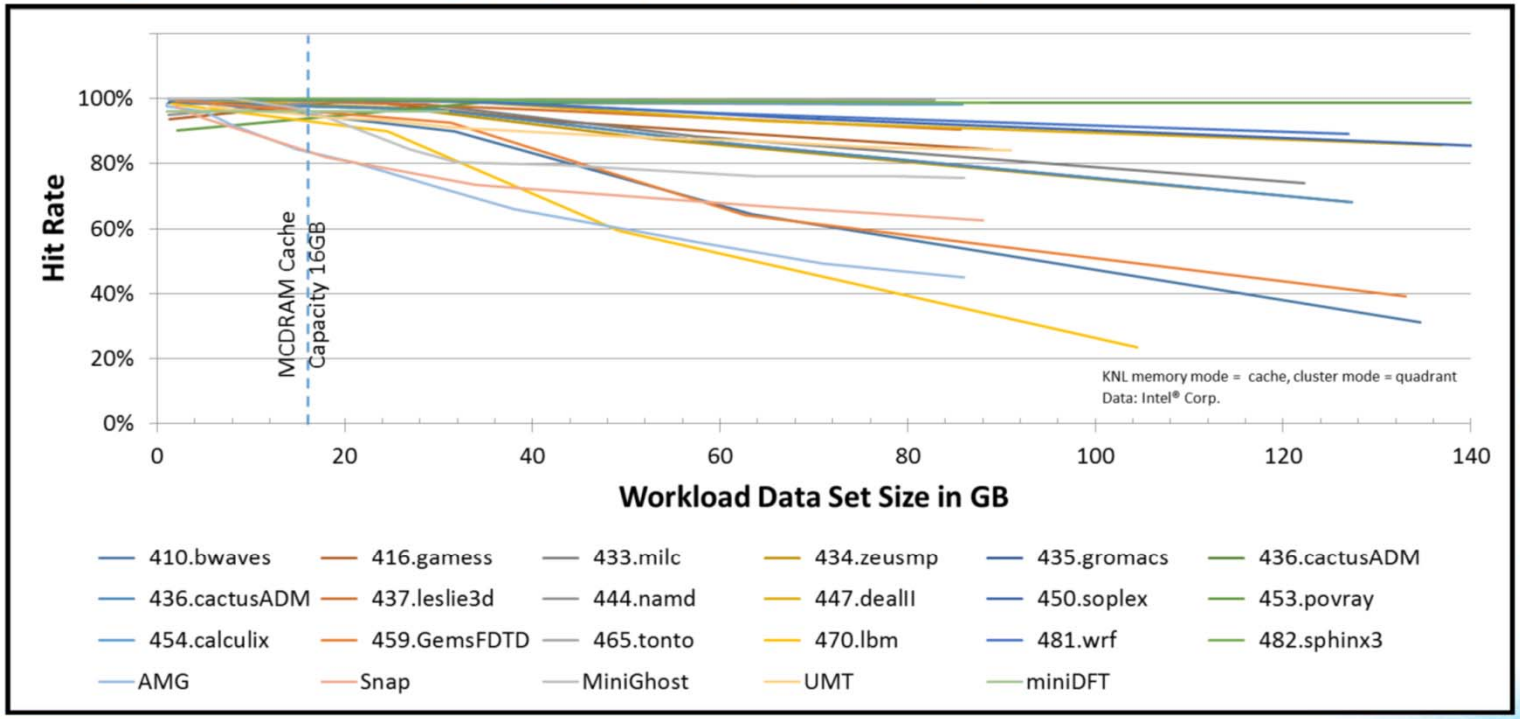
- Cache mode
  - MCDRAM cache for DRAM
  - Only DRAM address space
  - Done in hardware (applications don't need modified)
  - Misses more expensive (DRAM and MCDRAM access)
- Flat mode
  - MCDRAM and DRAM are both available
  - MCDRAM is just memory, in same address space
  - Software managed (applications need to do it themselves)
- Hybrid – Part cache/part memory
  - 25% or 50% cache
  - Possible cost of additional address lookup





Cache mode

MCDRAM Cache Hit Rate



Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you purchases, including the performance of that product when combined with other products. KNL results measured on pre-production parts. Any difference in system hardware or software design or configuration may affect actual performance. For more information go to http://www.intel.com/performance *Other names and brands may be claimed as the property of others

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

Slide from Intel

Discovering memory modes

- `apstat -M` (KNL configuration)
- MCDRAM exposed as separate NUMA node
  - Use `numactl` program
- 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
```

Using flat mode

- `numactl` can also set bulk memory policy
  - Preferred or enforced memory for application

- Example code:
  - Force to use MCDRAM, fails if exhausts memory

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

  - Tries to used preferred memory, falls back if exhausts memory

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

Allocating MCDRAM

A Heterogeneous Memory Management Framework

MEMKIND

- Defines a plug-in architecture.
- Each plug-in is called a “kind” of memory.
- Built on top of jemalloc: the FreeBSD OS default heap manager.
- Partition is defined by functions that provide inputs for operating system calls.
- High level memory management functions can be over-ridden as well.
- https://github.com/memkind

HBWMALLOC

- Implements easy model for KNL.
- Implemented using memkind; simplifies plug-in (kind) selection.
- Provides support for 2MB and 1GB pages.
- Select fallback behavior when on package memory does not exist or is exhausted.
- Check for existence of on package memory.

Jeff Hammond
Intel Parallel Computing Lab

Allocating MCDRAM

End Goal Usage: Code Snippets

Heap allocation in C

```
float * fv1 = malloc(sizeof(float) * 1000);
float * fv2 = hbw_malloc(sizeof(float) * 1000);
```

Allocatable arrays in Fortran

```
    REAL, ALLOCATABLE :: A(:), B(:), C(:)
!DIR$ ATTRIBUTES FASTMEM :: A
    NSIZE=1000
!   allocate array 'A' from MCDRAM
    ALLOCATE (A(1:NSIZE))
!   Allocate arrays that will come from DDR
    ALLOCATE  (B(1:NSIZE), C(1:NSIZE))
```

Automatic variables will be allocated in DDR in flat mode.

This means you may need to convert from automatic to heap arrays or use hybrid mode if such data is used in a bandwidth-intensive way.

Standard containers in C++ (not documented upstream yet)

```
std::vector<float, hbwmalloc::hbwmalloc_allocator<float> > vec;
```

Jeff Hammond
Intel Parallel Computing Lab

memkind on XC40 and from Fortran

- Will be available as a module on the syste
  - module load cray-memkind
  - man memkind/hbwmalloc for details of APIs
- 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
```

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

CCE MCDRAM Support

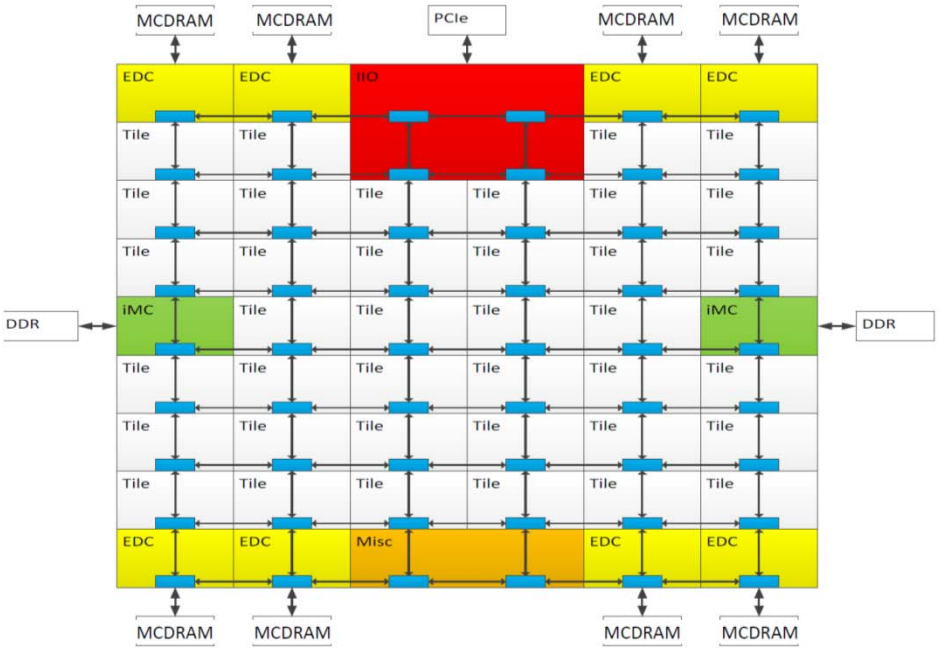
```
integer, dimension(:,:), allocatable :: A
!dir$ memory(bandwidth) B
integer :: B(N)

!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

KNL

KNL Mesh Interconnect



Mesh of Rings

- Every row and column is a (half) ring
- YX routing: Go in Y → Turn → Go in X
- Messages arbitrate at injection and on turn

Cache Coherent Interconnect

- MESIF protocol (F = Forward)
- Distributed directory to filter snoops

Three Cluster Modes

hemisphere

(1) All-to-All (2) Quadrant (3) Sub-NUMA Clustering

Avinash Sodani CGO PPoPP HPCA Keynote 2016

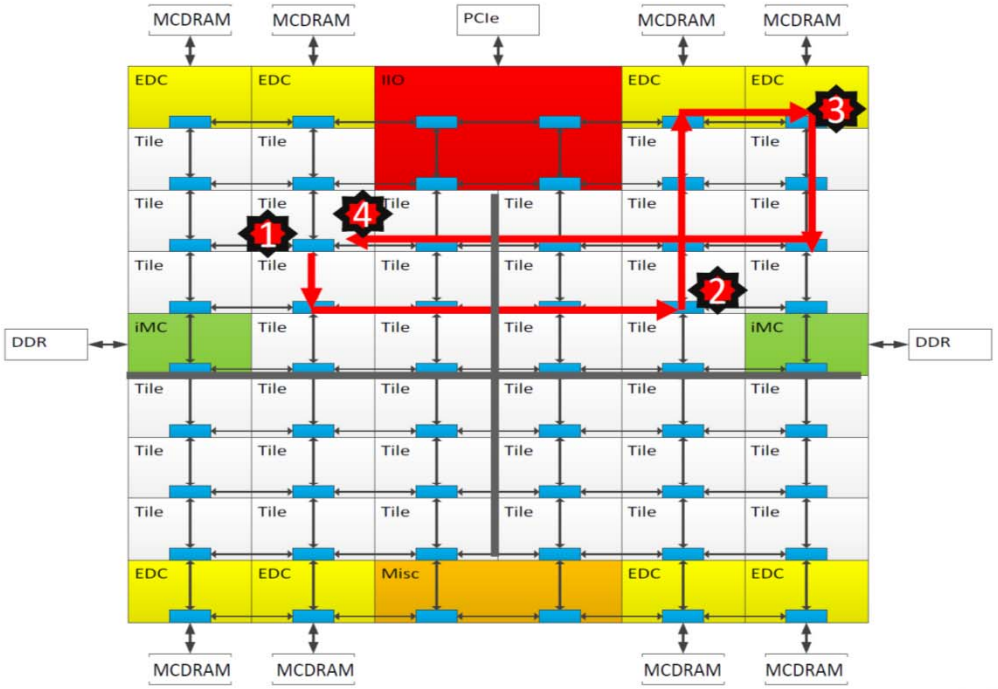
archer

epcc

THE UNIVERSITY OF EDINBURGH

Hemisphere is like quadrant but only uses 2 virtual halves

KNL
Cluster Mode: Quadrant



Chip divided into four virtual Quadrants

Address hashed to a Directory in the same quadrant as the Memory

Affinity between the Directory and Memory

Lower latency and higher BW than all-to-all. SW Transparent.

1) L2 miss, 2) Directory access, 3) Memory access, 4) Data return

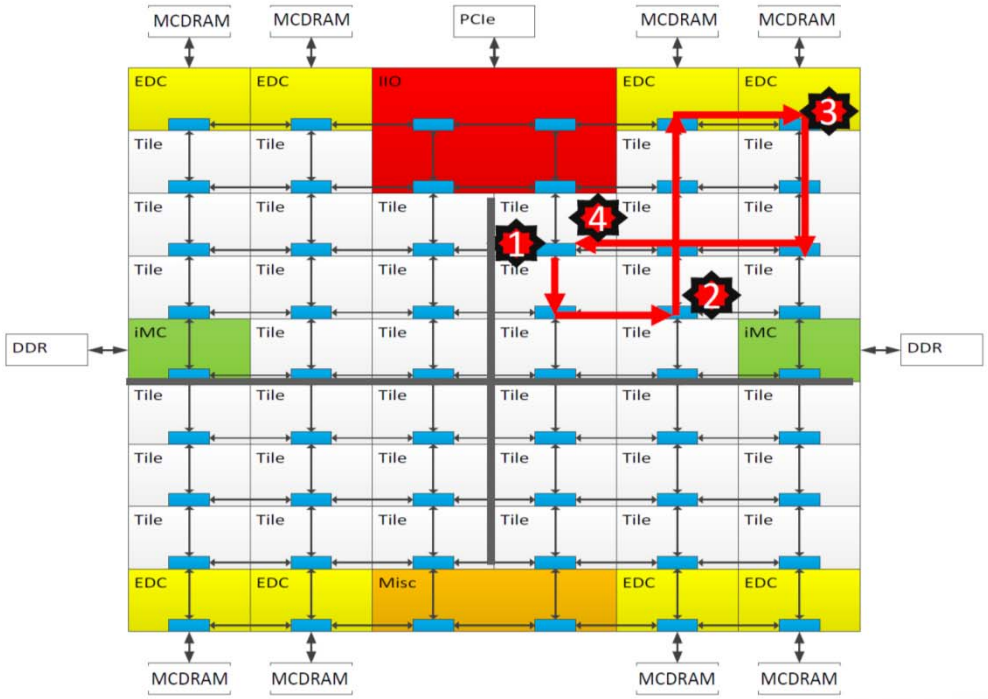
Avinash Sodani CGO PPoPP HPCA Keynote 2016

KNL

If using only 1 MPI rank and OpenMP to fill up cores
If also using SNC have to enable all memory access

```
numactl –m 4,5,6,7
```

Cluster Mode: Sub-NUMA Clustering (SNC)



Each Quadrant (Cluster) exposed as a separate NUMA domain to OS.

Looks analogous to 4-Socket Xeon

Affinity between Tile, Directory and Memory

Local communication. Lowest latency of all modes.

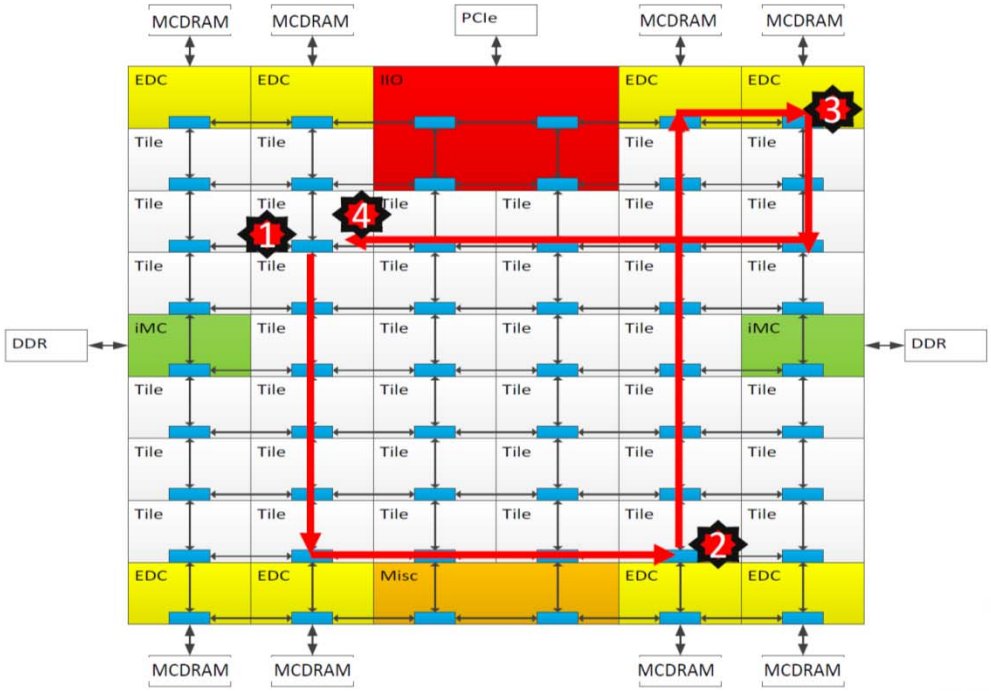
SW needs to NUMA optimize to get benefit.

1) L2 miss, 2) Directory access, 3) Memory access, 4) Data return

Avinash Sodani CGO PPoPP HPCA Keynote 2016

KNL

Cluster Mode: All-to-All



Address uniformly hashed across all distributed directories

No affinity between Tile, Directory and Memory

Most general mode. Lower performance than other modes.

Typical Read L2 miss

1. L2 miss encountered
2. Send request to the distributed directory
3. Miss in the directory. Forward to memory
4. Memory sends the data to the requestor

Programming the KNL

- Standard HPC - parallelism
  - MPI
  - OpenMP
    - Default `OMP_NUM_THREADS` may be 256
  - mkl
- Standard HPC – compilers
  - module load craype-mic-knl
  - Intel compilers
    `-xMIC-AVX512` (without the module)
  - Cray compilers
    `-hcpu=mic-knl` (without the module)
  - GNU compilers
    `-march=knl` or `-mavx512f -mavx512cd -mavx512er -mavx512pf`
    (without the module)

Compiling for the KNL

- Standard KNL compilation targets the KNL vector instruction set
  - This **won't** run on standard processor
  - Binaries that run on standard processors **will** run on the KNL
- If your build process executes programs this may be an issue
  - Can build a fat binary using Intel compilers

    `-ax MIC-AVX-512,AVX`
  - For other compilers can do initial compile with KNL instruction set
    - Then re-compile specific executables with KNL instruction set
    - i.e. `–aAVX` for Intel, `-hcpu=…` for Cray, `-march=…` for GNU

Configuring KNL

- Different memory modes and cluster options
  - Configured at boot time
    - Switching between cache and flat mode
    - Switching cluster modes
- For ARCHER XC40 Cluster configuration is done through batch system (PBS)
- Modes can be requested as a resource:

```
#PBS -l select=4:aoe=quad_100
#PBS -l select=4:aoe=snc2_50
```

  - This is in the form :`aoe=numa_cfg_hbm_cache_pct`
- Available modes are:
  - For the NUMA configuration (`numa_cfg`): a2a, snc2, snc4, hemi, quad
  - For the MCDRAM cache configuration (`hbm_cache_pct`): 0, 25, 50, 100
- So for quadrant mode and flat memory (MCDRAM and DRAM separate) this would be:

```
#PBS -l select=4:aoe=quad_0
```

- If assigned nodes are not in the correct state re-boot will automatically occur
  - This may delay the job starting

Warning: Still under configuration

epcc

Suggestions

- Pure MPI is fine for some codes, hybrid better for others
- If hybrid need a minimum number of MPI processes for memory and network bandwidth
  - i.e. quadrant mode would want 1 MPI process per quadrant
- If hybrid then need good second level parallelism
  - Serial code is still slower than standard multi-core
  - Need to make sure as much code is OpenMP'd or equivalent
- Cache mode is a good place to start
  - Compare flat mode with cache mode to see performance impact of MCDRAM
- Quadrant mode is a good place to start
- Vectorisation is key for performance (but not as hard as KNC)
- Hyperthreads can be beneficial
  - Not required by can give performance boost

Getting access

- There will be a button **KNL access** in the **Login account details page** on SAFE
  - Page for the account you get to from the **Login accounts** menu
  - Click this button to apply for accounts
- This will join your existing account to the k01 project
  - give them access to a personal budget **k01-<username>** with 30 kAU
- `/home` will be cross-mounted from ARCHER
- `/work` is a new disk system
  - You will get a user quota on /fs5
  - Set up as a scratch disk.

System setup

- KNL system will have it's own login nodes
  - Not accessible from the outside world
  - Have to login in to the ARCHER login nodes first
- SSH into the KNL login nodes
  - Name still under consideration
- Compile jobs there
  - Different versions of the system software from the standard ARCHER nodes
- Submit jobs using PBS from those nodes

Training and documentation

- Upcoming 1 day course about Using KNL on ARCHER
  - https://www.archer.ac.uk/training/
- Documentation
  - http://archer.ac.uk/documentation/knl-guide/
- ARCHER Helpdesk

Performance

- Initial performance experiences with a single KNL

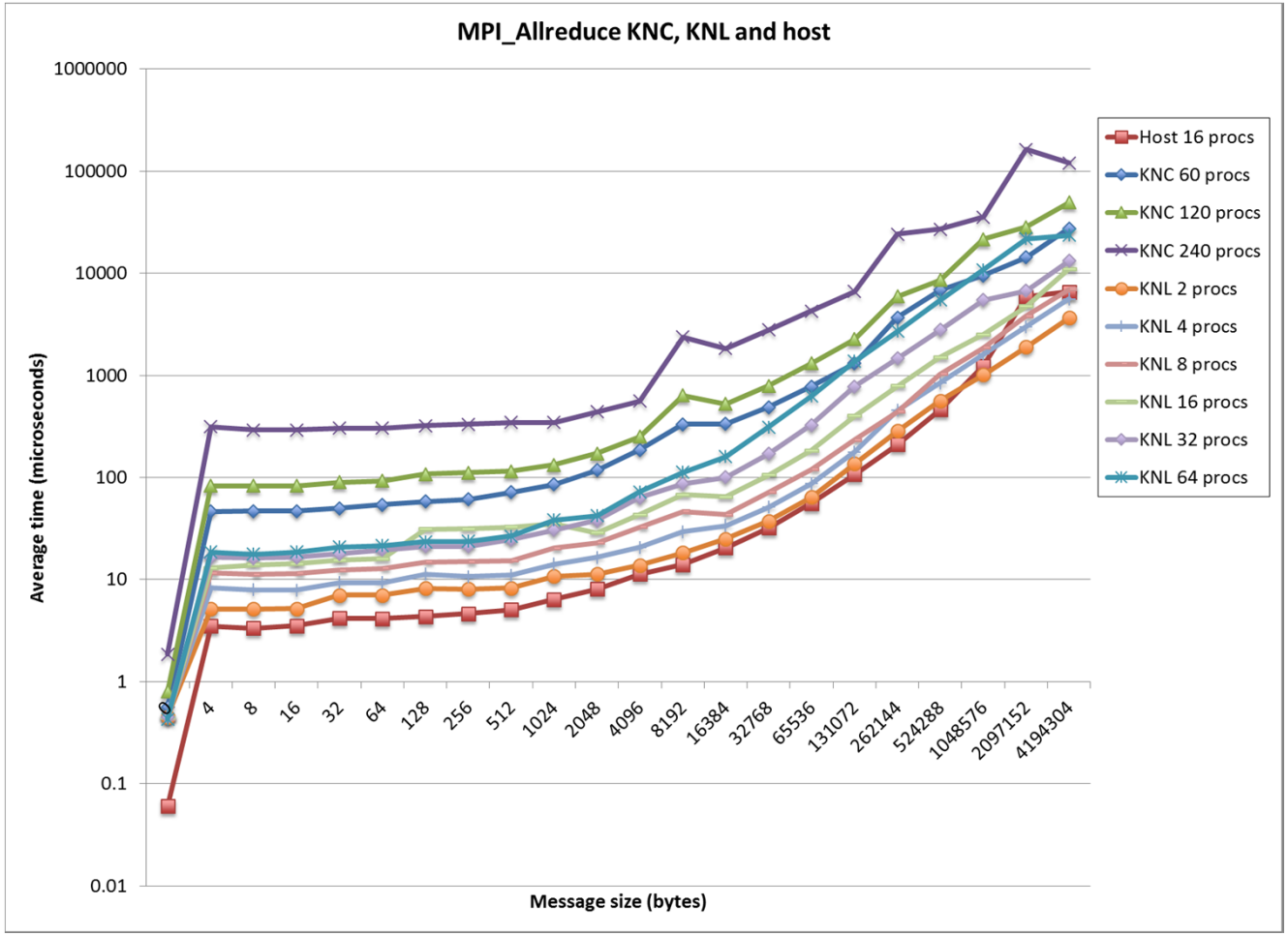
Application	KNC	KNL	KNL HB	IvyBridge	Broadwell
COSA		561	450	497	349
GS2	400	184.2	103.8	126.6	83.4
CASTEP		149	146	102	38

Performance multi-node

- COSA - Fluid dynamics code
  - Harmonic balance (frequency domain approach)
  - Unsteady navier-stokes solver
  - Optimise performance of turbo-machinery like problems
  - Multi-grid, multi-level, multi-block code

ARCHER	Broadwell	KNL	KNL HB	KNL 2 Node HB
497	349	561	450	197

MPI Performance - Allreduce

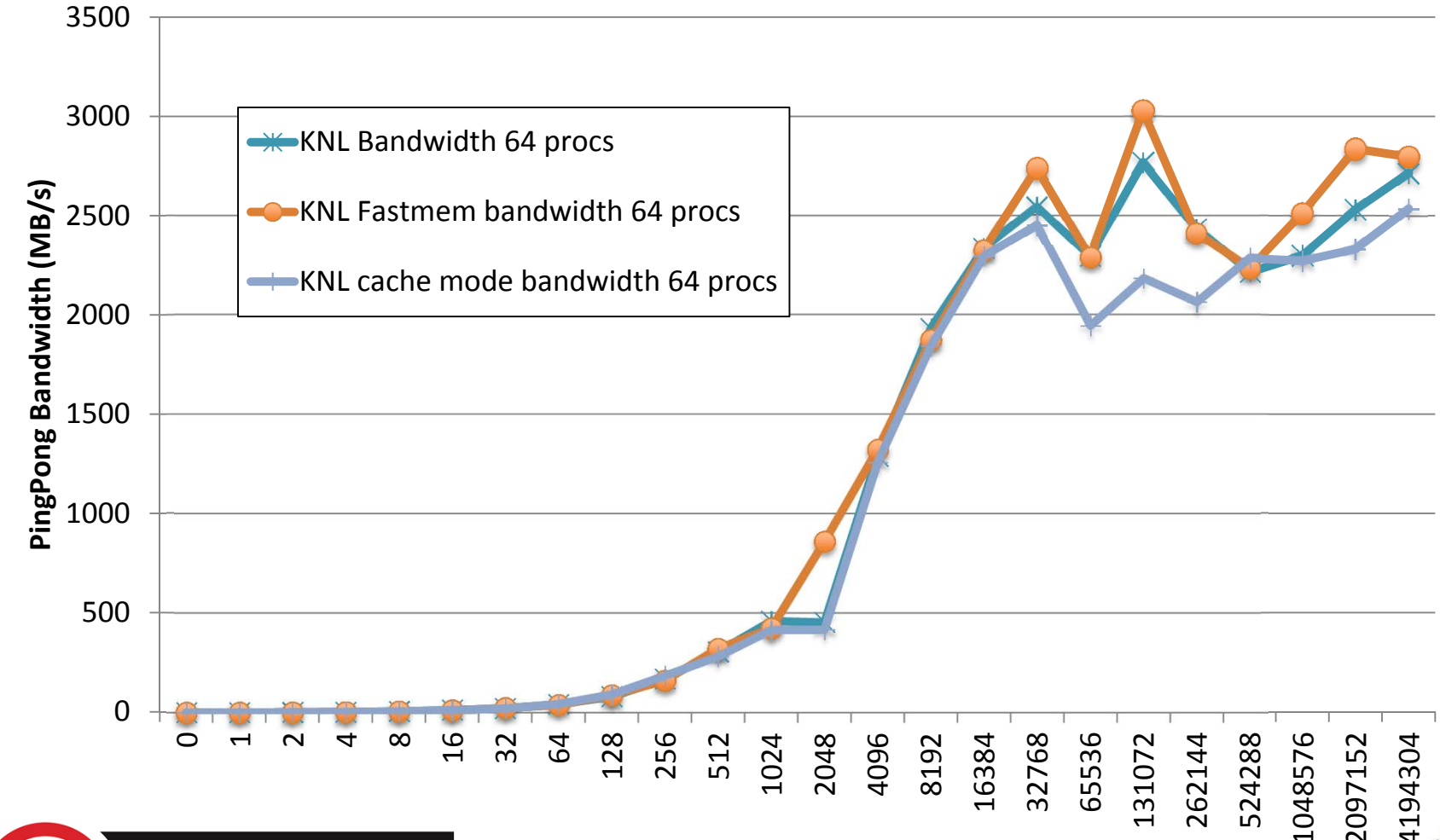


archer

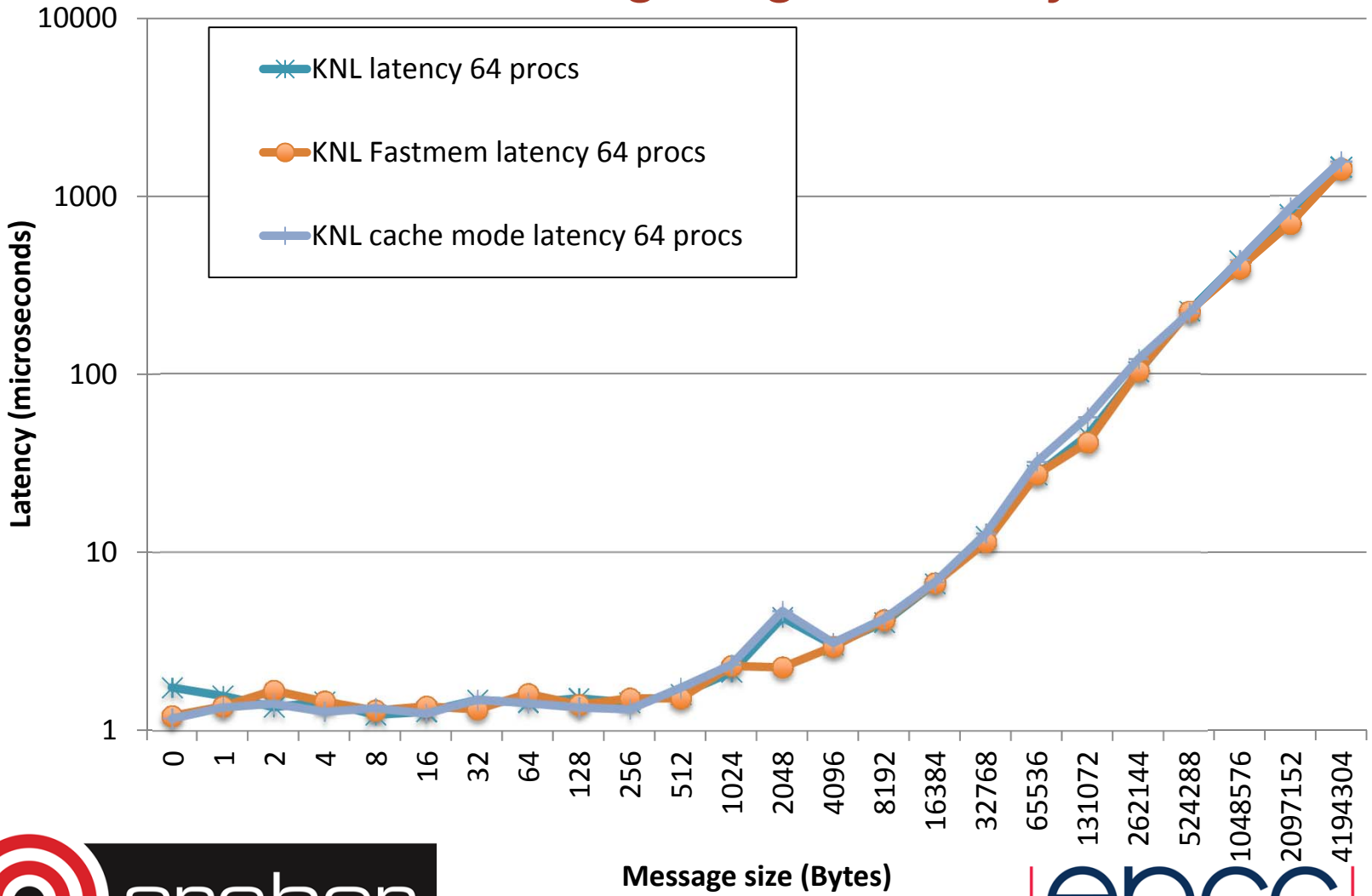
epcc

THE UNIVERSITY OF EDINBURGH

MPI Performance – PingPong – Memory modes



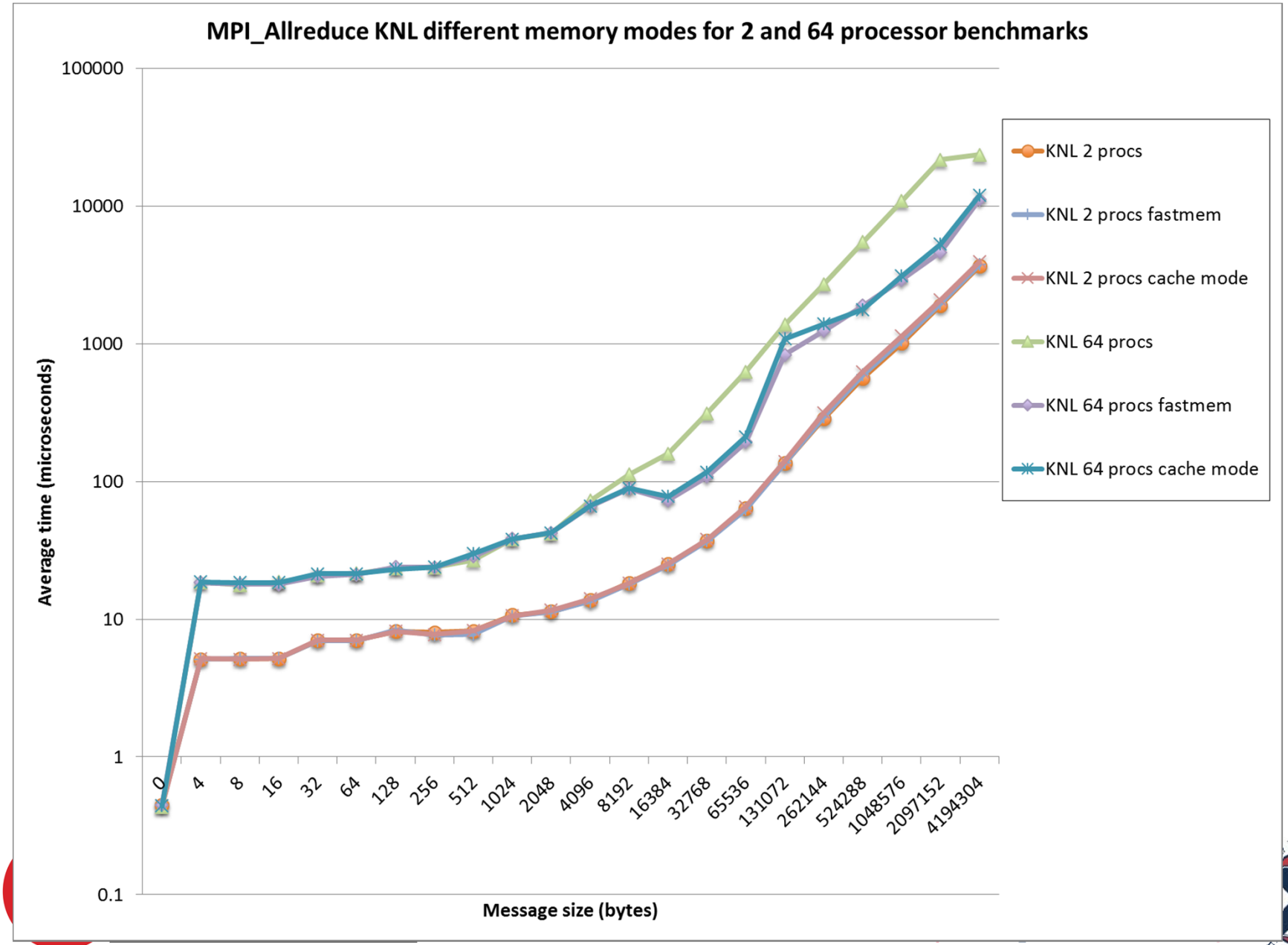
MPI Performance – PingPong – Memory modes



archer

epcc

THE UNIVERSITY OF EDINBURGH


MPI_Allreduce KNL different memory modes for 2 and 64 processor benchmarks
Average time (microseconds)
100000
10000
1000
100
10
1
0.1
0
4
8
16
32
64
128
256
512
1024
2048
4096
8192
16384
32768
65536
131072
262144
524288
1048576
2097152
4194304
Message size (bytes)
KNL 2 procs
KNL 2 procs fastmem
KNL 2 procs cache mode
KNL 64 procs
KNL 64 procs fastmem
KNL 64 procs cache mode

EPCC IPCC

- EPCC has IPCC collaboration with Intel
- Working on porting and optimising codes on Xeon Phi
- Training and support of Xeon Phi
- Get in touch if you've got any questions, or something you'd like to collaborate on

Goodbye

Virtual tutorial has finished

Please check here for future tutorials and training

http://www.archer.ac.uk/training

http://www.archer.ac.uk/training/virtual/

archer

epcc

THE UNIVERSITY OF EDINBURGH