NATIVE MODE PROGRAMMING

Fiona Reid

Overview

- What is native mode?
- What codes are suitable for native mode?
- MPI and OpenMP in native mode
- MPI performance in native mode
- OpenMP thread placement
- How to run over multiple Xeon Phi cards
- Symmetric mode using both host & Xeon Phi

Native mode: introduction

- Range of different methods to access the Xeon Phi
  - native mode
  - offload mode
  - symmetric mode
- This lecture will concentrate mostly on *native* mode
- In native mode:
  - `ssh` directly into the card, running own Linux OS
  - Run applications on the command line
  - Use any of the supported parallel programming models to make use of the 240 virtual threads available
- Can be a quick way to get a code running on the Xeon Phi
- Not all applications are suitable for native execution

Steps for running in native mode

- Determine if your application is suitable (see next slide)
- Compile application for native execution
  - Essentially just add the `-mmic` flag
- Build any libraries for native execution
- Depending on your system you may also need to:
  - Copy binaries, dependencies, input files locally to Xeon Phi card
  - If Xeon Phi and host are cross-mounted you won't need to do this
- Log in to Xeon Phi, set up environment, run application

Suitability for native mode

- Remember native mode gives you access to up to 240 virtual cores
  - You want to use as many of these as possible
- Your application should have the following characteristics:
  - A small memory footprint using less than the memory on the card
  - Be highly parallel
  - Very little serial code – this will be even slower on the Xeon Phi
  - Minimal I/O – NFS allows external I/O but limited bandwidth
  - Complex code with no well defined hotspots

Compiling for native execution

- Compile on the host using the `-mmic` flag e.g.

```
ifort -mmic helloworld.f90 -o helloworld
```

- NB: You must compile on a machine with a Xeon Phi card attached as you need access to the MPSS libraries etc at compile time
- Any libraries your code uses have to be built with `-mmic`
- If you use libraries such as LAPACK, BLAS, FFTW etc then you can link to the Xeon Phi version of MKL

Compiling for native execution

- MPI and OpenMP compilation are identical to host, just add the `-mmic` flag e.g.

MPI

```
mpiicc -mmic helloworld_mpi.c -o helloworld_mpi
```

OpenMP

```
icc -openmp -mmic helloworld_omp.c -o helloworld_omp
```

Running a native application

- Login to the Xeon Phi card
  - Copy any files across locally if required
- Set up your environment
- Run the application

Running a native application – MPI

```
[host src]$ ssh mic0
[mic0 ~]$ cd /home-hydra/h012/fiona/src
[mic0 src]$ source /opt/intel/composer_xe_2015/mkl/bin/mklvars.sh mic
[mic0 src]$ source /opt/intel/impi/5.0.3.048/mic/bin/mpivars.sh

[mic0 src]$ mpirun -n 4 ./helloworld_mpi
Hello world from process 1 of 4
Hello world from process 2 of 4
Hello world from process 3 of 4
Hello world from process 0 of 4
```

Running a native application – OpenMP

```
[host src]$ ssh mic0
[mic0 ~]$ cd /home-hydra/h012/fiona/src
[mic0 src]$ export OMP_NUM_THREADS=8
[mic0 src]$ source /opt/intel/composer_xe_2015/mkl/bin/mklvars.sh mic

[mic0 src]$ ./helloworld_omp
Maths computation on thread 1 = 0.000003
Maths computation on thread 0 = 0.000000
Maths computation on thread 2 = -0.000005
Maths computation on thread 3 = 0.000008
Maths computation on thread 5 = 0.000013
Maths computation on thread 4 = -0.000011
Maths computation on thread 7 = 0.000019
Maths computation on thread 6 = -0.000016
```

Running a native application – MPI/OpenMP

```
[host src]$ ssh mic0
[mic0 ~]$ cd /home-hydra/h012/fiona/src
[mic0 src]$ export OMP_NUM_THREADS=4
[mic0 src]$ source /opt/intel/composer_xe_2015/mkl/bin/mklvars.sh mic
[mic0 src]$ source /opt/intel/impi/5.0.3.048/mic/bin/mpivars.sh
[mic0 src]$ mpirun -n 2 ./helloworld_mixedmode_mic
```

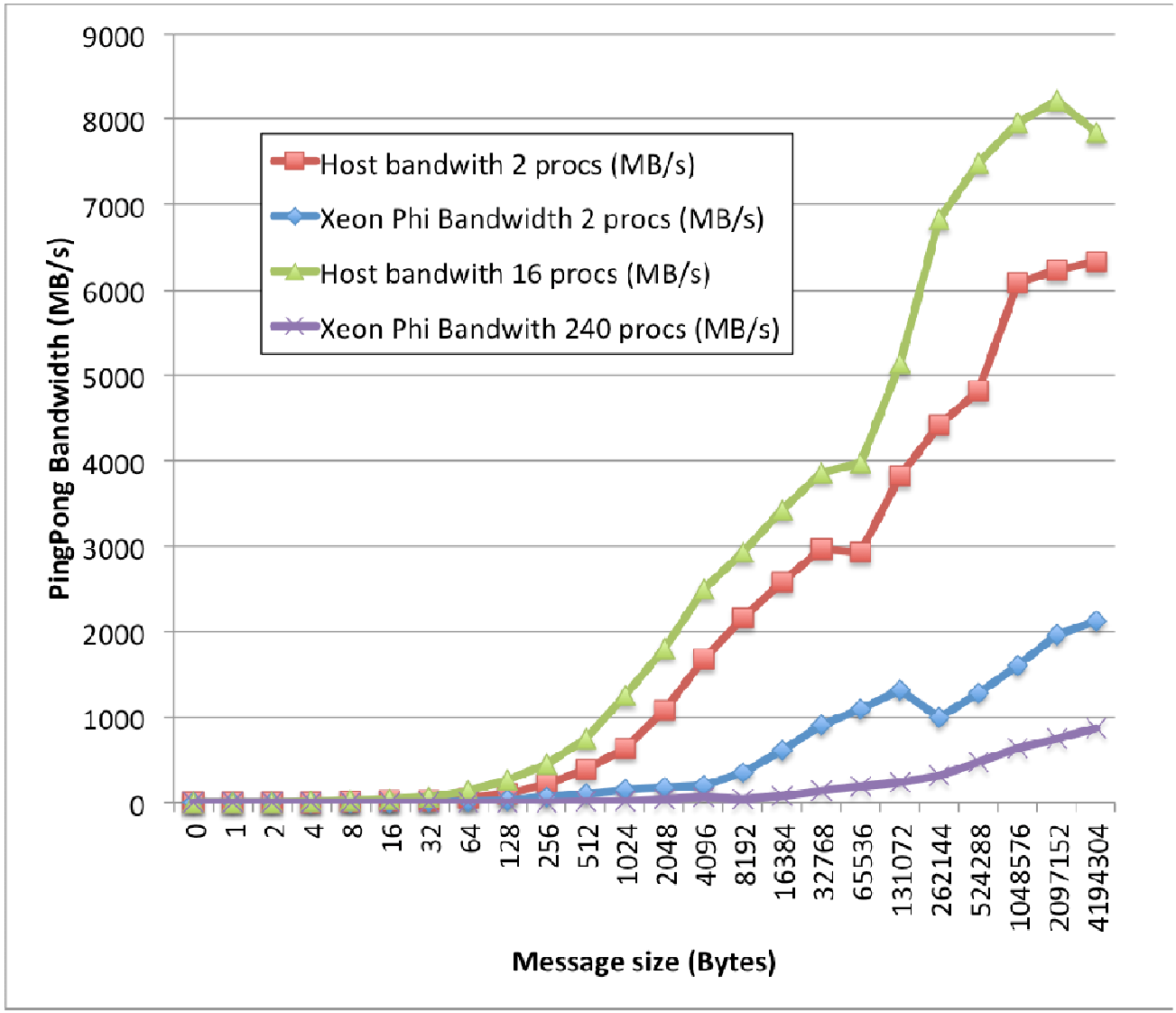
```
Hello from thread 0 out of 4 from process 0 out of 2 on phi-mic0.hydra
Hello from thread 2 out of 4 from process 0 out of 2 on phi-mic0.hydra
Hello from thread 0 out of 4 from process 1 out of 2 on phi-mic0.hydra
Hello from thread 3 out of 4 from process 0 out of 2 on phi-mic0.hydra
Hello from thread 1 out of 4 from process 0 out of 2 on phi-mic0.hydra
Hello from thread 1 out of 4 from process 1 out of 2 on phi-mic0.hydra
Hello from thread 2 out of 4 from process 1 out of 2 on phi-mic0.hydra
Hello from thread 3 out of 4 from process 1 out of 2 on phi-mic0.hydra
```

epcc | THE UNIVERSITY OF EDINBURGH

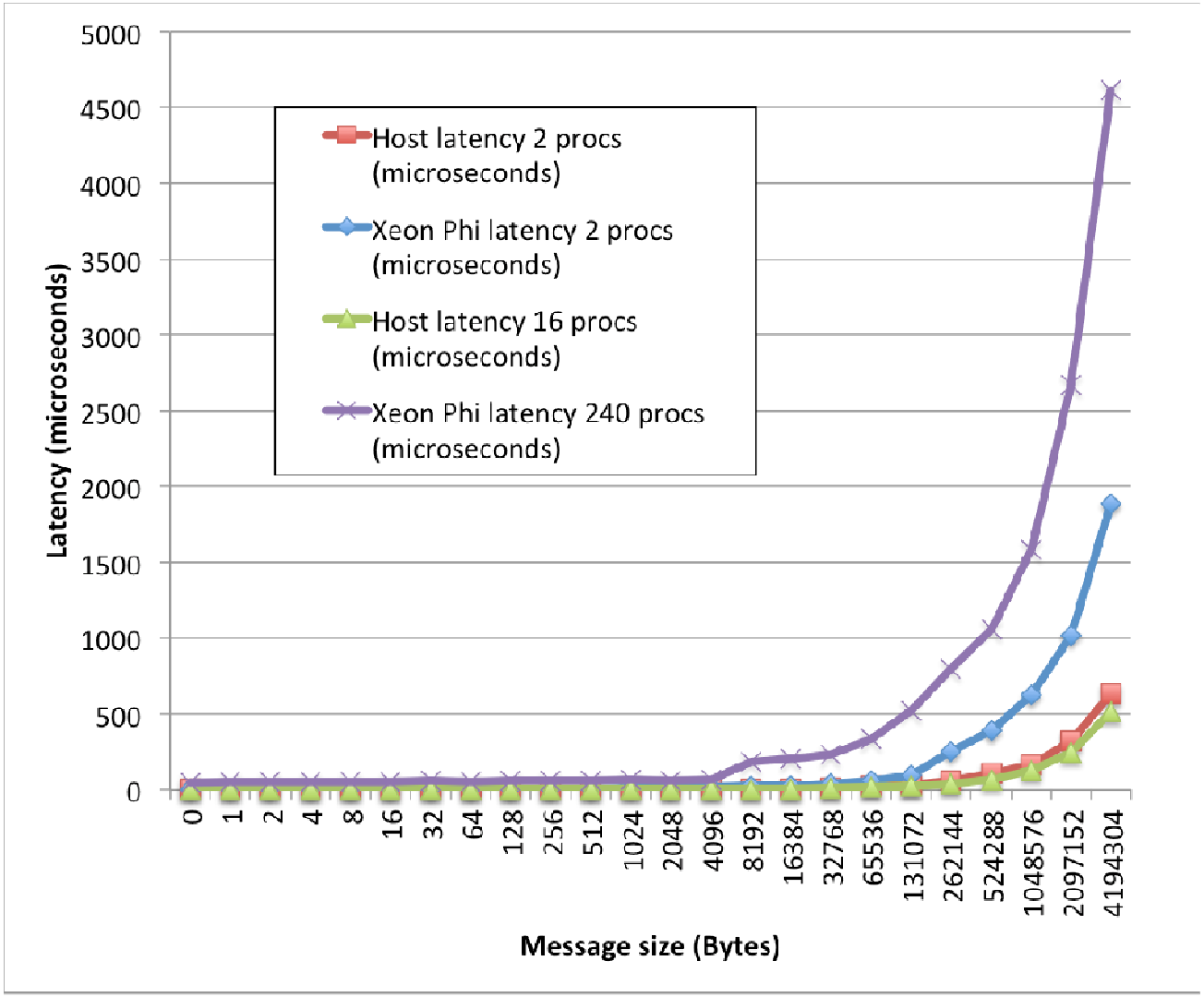
MPI performance in native mode

- The MPI performance on the Xeon Phi is generally much slower than you will get on the host
- Used the Intel MPI benchmarks to measure the MPI performance on the host and Xeon Phi
- https://software.intel.com/en-us/articles/intel-mpi-benchmarks
- Compared point-to-point via PingPong and collectives via MPI_Allreduce

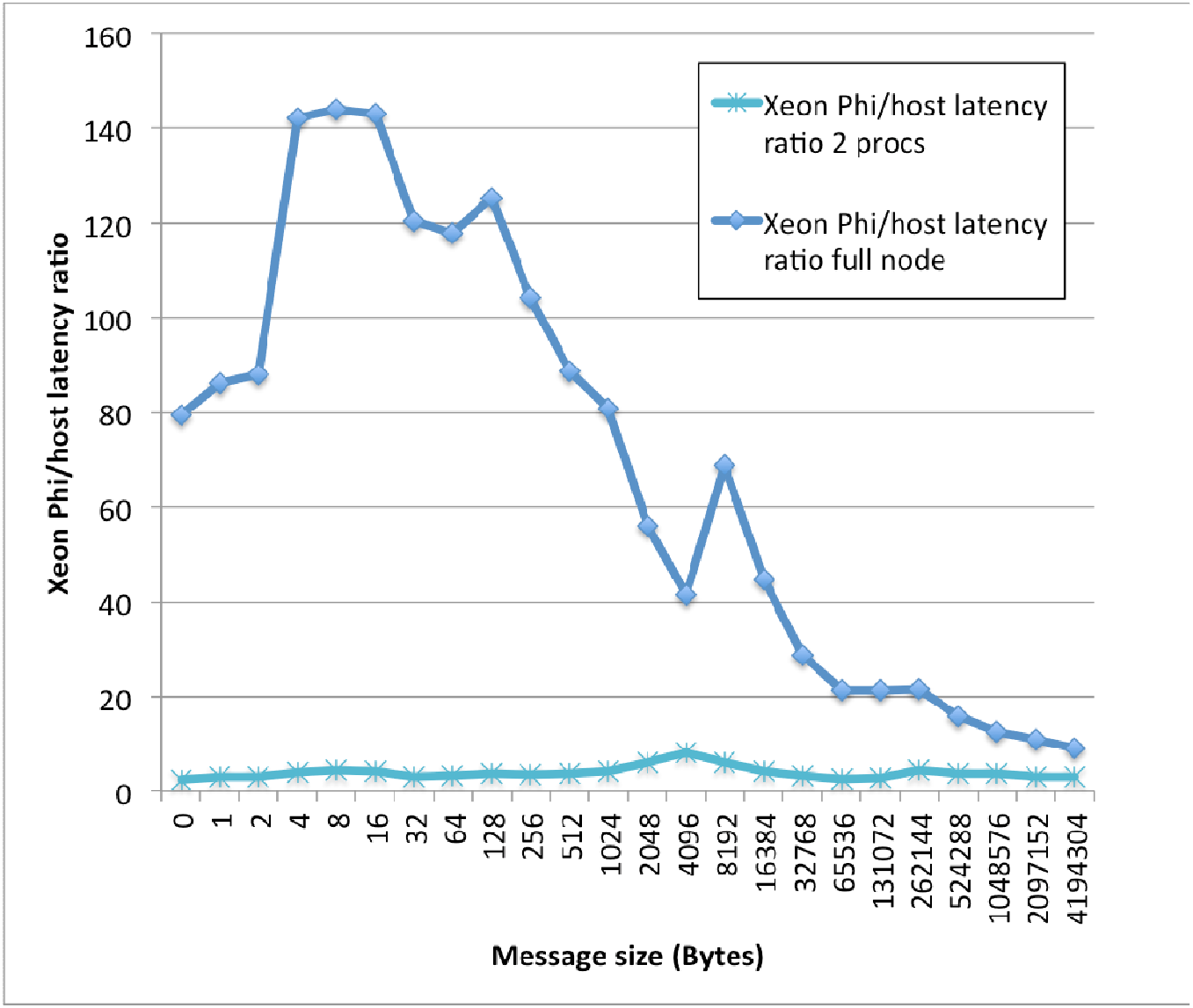
PingPong Bandwidth



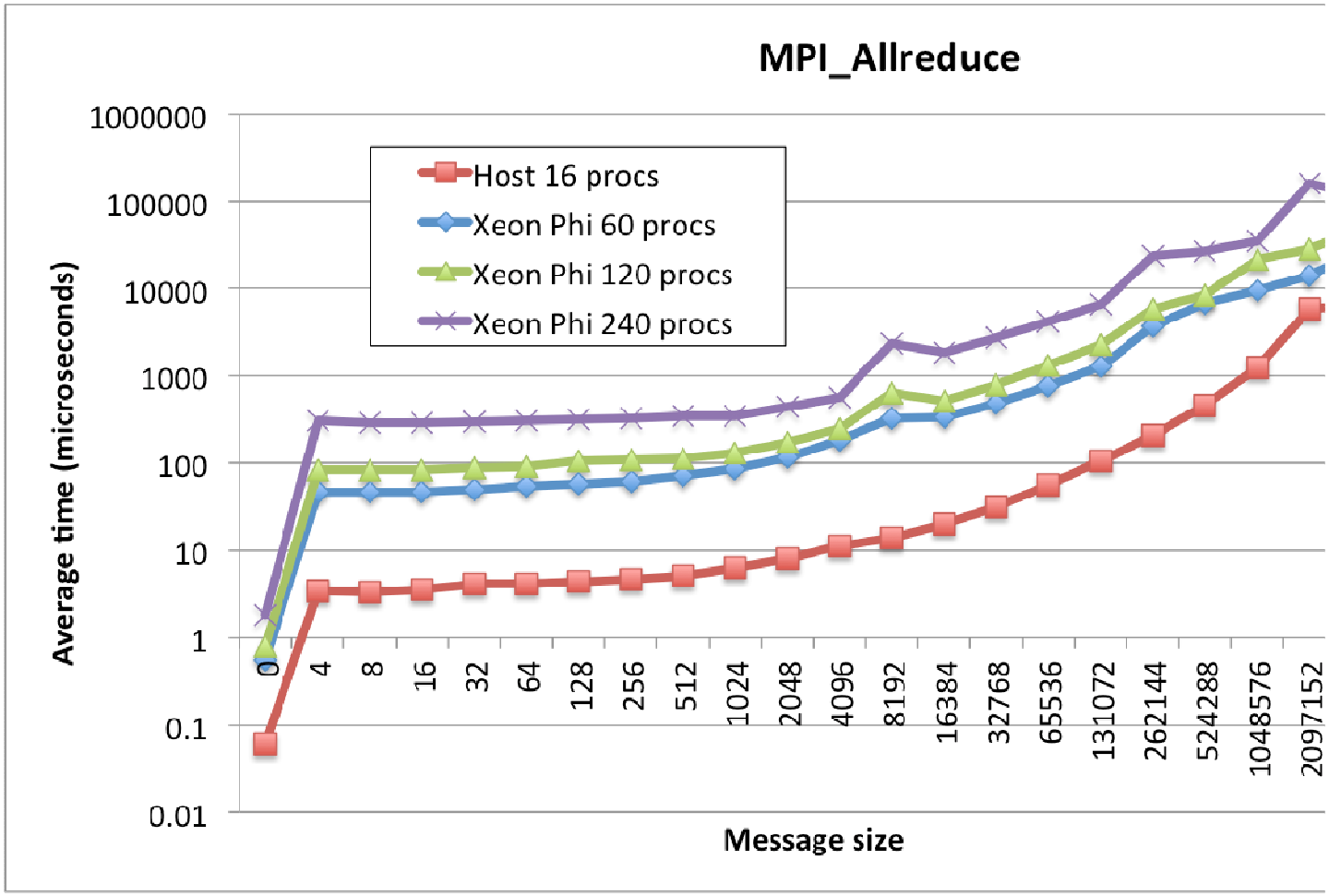
PingPong Latency



PingPong Latency



MPI_Allreduce



epcc

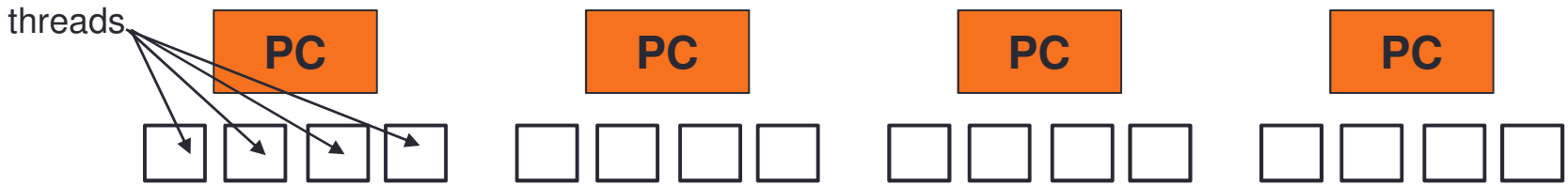
THE UNIVERSITY OF EDINBURGH

OpenMP performance/ thread affinity

- In native mode, we have 60 physical cores each running 4 hardware threads, so 240 threads in total
- To obtain good performance we need at least 2 threads running on each core
- Often running 3 or 4 threads per core is best
- Where/how we place these threads is very important
- `KMP_AFFINITY` can be used to find out and control thread distribution

Thread/process affinity

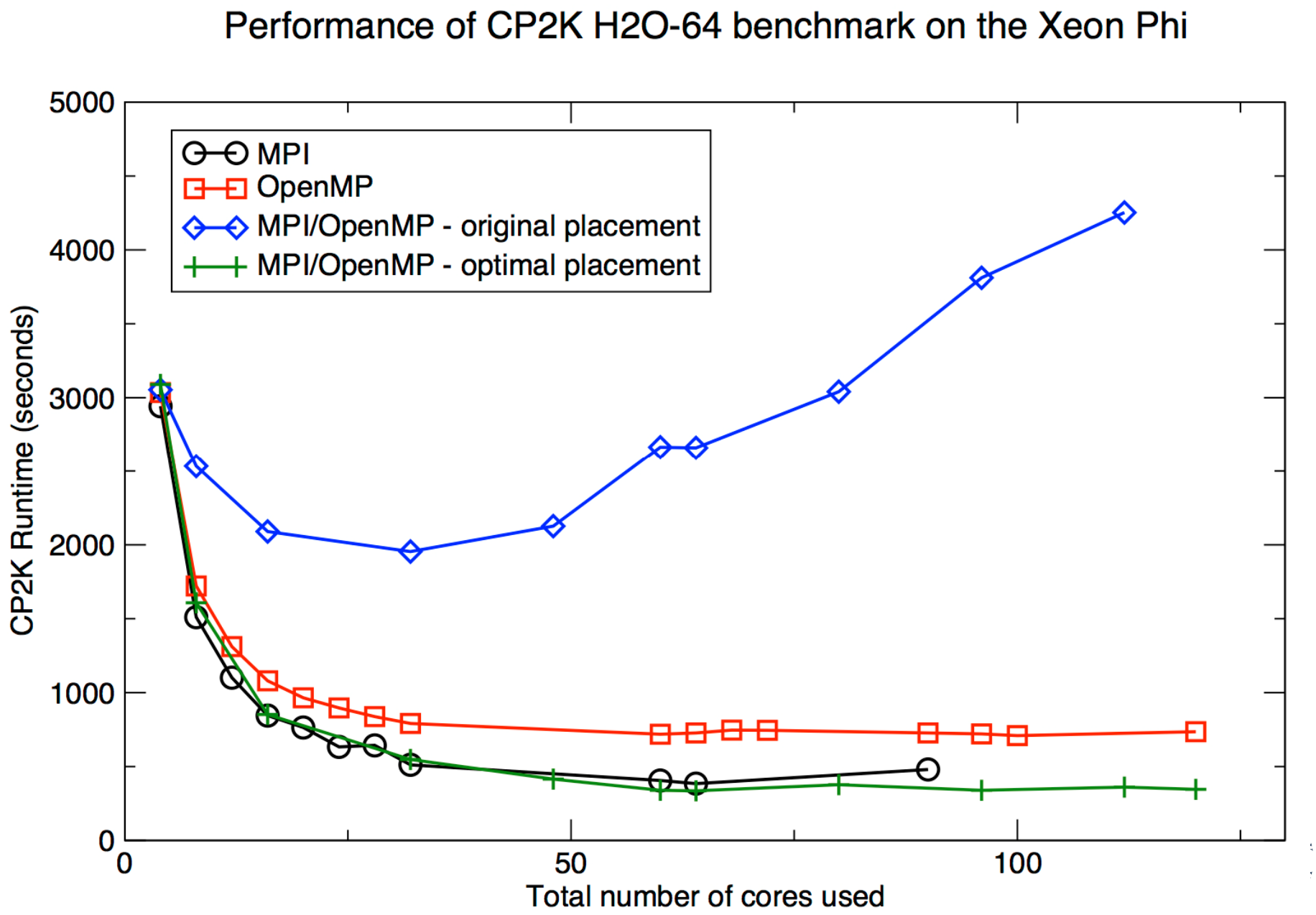
- We have 60 physical cores (PC), each running 4 virtual threads



Compact	0	1	2	3	4	5									
Scatter	0	4			1	5			2				3		
Balanced	0	1			2	3			4				5		

- Various placement strategies possible
  - Compact – preserves locality but some physical cores end up with lots of work and some end up with none
  - Scatter – destroys locality but if < 60 virtual threads used is fine
  - Balanced – preserves locality and works for all thread counts

epcc | THE UNIVERSITY OF EDINBURGH


Performance of CP2K H2O-64 benchmark on the Xeon Phi
MPI
OpenMP
MPI/OpenMP - original placement
MPI/OpenMP - optimal placement
CP2K Runtime (seconds)
Total number of cores used
0
1000
2000
3000
4000
5000
0
50
100

Affinity example with MPI/OpenMP

For 2 MPI processes each running 2 OpenMP threads:

```
export OMP_NUM_THREADS=2
mpirun -prepend-rank -genv LD_LIBRARY_PATH path_to_the_mic_libs \
-np 1 -env KMP_AFFINITY verbose,granularity=fine,proclist=[1,5],explicit \
-env OMP_NUM_THREADS ${OMP_NUM_THREADS} $CP2K_BIN/cp2k.psmp H2O-64.inp : \
-np 1 -env KMP_AFFINITY verbose,granularity=fine,proclist=[9,13],explicit \
-env OMP_NUM_THREADS ${OMP_NUM_THREADS} $CP2K_BIN/cp2k.psmp H2O-64.inp &> x
```

- For *every* MPI process you say where its threads will be placed
- With large numbers of processes this gets quite messy!
- The default placement is often ok
- Use `export KMP_AFFINITY=verbose` to check

Native mode: 2 Xeon Phi cards

- You can run your native code using several Xeon Phi cards
- Here you compile a native binary and then launch the job on multiple cards from the host e.g.

```
[host ~]$ export I_MPI_MIC=enable
[host ~]$ export DAPL_DBG_TYPE=0
[host ~]$ mpiexec.hydra -host mic0 -np 2 /path_on_mic/test.mic : \
              -host mic1 -np 2 /path_on_mic/test.mic
Hello from process 2 out off 4 on phi-mic1.hydra
Hello from process 3 out off 4 on phi-mic1.hydra
Hello from process 0 out off 4 on phi-mic0.hydra
Hello from process 1 out off 4 on phi-mic0.hydra
```

- MPI ranks are assigned in the order that cards are specified
- For an MPI/OpenMP code you'll need to use `-env` to set the number of threads on each card and `LD_LIBRARY_PATH`

Symmetric mode: host & Xeon Phi(s)

- You can also use a combination of the host and Xeon Phi
- Build two binaries, one for the host and one for the Xeon Phi
- The MPI ranks are across host (0:nhost-1) and Xeon Phi (nhost:total number of procs-1)

```
[host src]$ mpiicc helloworld_symmetric.c -o hello_sym.host
[host src]$ mpiicc -mmic helloworld_symmetric.c -o hello_sym.mic
[host ~]$ export I_MPI_MIC=enable
[host ~]$ export DAPL_DBG_TYPE=0
[host src]$ mpiexec.hydra -host localhost -np 2 ./hello_sym.host : \
        -host mic0 -np 4 /home-hydra/h012/fiona/src/hello_sym.mic
Hello from process 0 out off 6 on phi.hydra
Hello from process 1 out off 6 on phi.hydra
Hello from process 2 out off 6 on phi-mic0.hydra
Hello from process 3 out off 6 on phi-mic0.hydra
Hello from process 4 out off 6 on phi-mic0.hydra
Hello from process 5 out off 6 on phi-mic0.hydra
```

epcc | THE UNIVERSITY OF EDINBURGH

Summary

- Native mode provides an easy way to get code running on Xeon Phi – just add `-mmic`
- Not all codes are suitable
- You should now be able to compile + run in native mode
- Thread/task/process placement is important
- Have also discussed running on multiple Xeon Phi's