Xeon Phi Native Mode - Sharpen Exercise

Fiona Reid, Andrew Turner, Dominic Sloan-Murphy, David Henty, Adrian Jackson

June 22, 2016

Contents

1 Aims

The aim of this exercise is to get you compiling and running native mode applications on the Xeon Phi.

2 Introduction

In this exercise you will run a simple program, in serial and parallel on both the host machine and the Xeon Phi, to sharpen the provided image file.

2.1 Download and extract the exercise files

Once you have logged into the host machine you need to obtain a copy of the source code. You can obtain this using *wget* as follows:

```
wget https://www.archer.ac.uk/training/course-material/2016/06/
xeonphi_soton/exercises/exercise2/sharpen_xeonphi.tar.gz
```

This program takes a fuzzy image and uses a simple algorithm to sharpen the image. A very basic parallel version of the algorithm has been implemented which we will use in this exercise. There are a number of versions of the sharpen program available:

C-SER Serial C version

F-SER Serial Fortran version

C-MPI Parallel C version using MPI

F-MPI Parallel Fortran version using MPI

C-OMP Parallel C version using OpenMP

F-OMP Parallel Fortran version using OpenMP

C-HYB Parallel C version using MPI and OpenMP

F-HYB Parallel Fortran version using MPI and OpenMP

We won't use the hybrid MPI and OpenMP version in this practical but you may wish to experiment with it in future.

To unpack the archive use:

```
[host]$ tar -xzvf sharpen_xeonphi.tar.gz
sharpen/C-SER/
sharpen/C-SER/filter.c
...
--snip--
...
sharpen/F-OMP/dosharpen.f90
sharpen/F-OMP/Makefile
sharpen/F-OMP/fuzzy.pgm
sharpen/setup.sh
sharpen/setup_mic.sh
```

If you are interested in the C examples move to the *C-SER* subdirectory; for Fortran, move to *F-SER*. For example:

```
[host]$ cd sharpen/C-SER
```

2.2 Compile the source code to produce an executable file for the host

We will first compile the serial version of the code for the host machine. The source files for this can be found in the *C-SER* or *F-SER* directory depending on your choice of programming language. Each directory contains a separate makefile for the host, *Makefile* and for the Xeon Phi, *Makefile.mic*. To build the host version you can use *make* as follows:

```
[host]$ cd sharpen
source setup.sh

[host]$ cd sharpen/C-SER
[host]$ ls
cio.c        filter.c   Makefile      sharpen.c  submit.sh    utilities.h
dosharpen.c  fuzzy.pgm  Makefile.mic  sharpen.h  utilities.c

[host]$ make
icc -openmp -g -DC_SERIAL_PRACTICAL -c sharpen.c
icc -openmp -g -DC_SERIAL_PRACTICAL -c dosharpen.c
icc -openmp -g -DC_SERIAL_PRACTICAL -c filter.c
icc -openmp -g -DC_SERIAL_PRACTICAL -c cio.c
icc -openmp -g -DC_SERIAL_PRACTICAL -c utilities.c
icc -openmp -g -DC_SERIAL_PRACTICAL -o sharpen sharpen.o dosharpen.o
    filter.o cio.o utilities.o -lm
[host]$
```

The command *source setup.sh* ensures that the Intel compiler versions and MPI libraries, wrappers etc are to your path. After building the code you should have an executable file called *sharpen* which we will run on the host.

For the Fortran version, the process is exactly the same as above, except you should move to the *F-SER* subdirectory and build the program there:

```
[host]$ cd sharpen/F-SER
[host]$ make
...
```

As before, this should produce a *sharpen* executable.

Don't worry about the C file `utilities.c` – it is just providing an easy method for printing out various information about the program at run time, and it is most easily implemented in C.

2.3 Running the serial version on the host

You can run this serial program directly, e.g.:

```
[host]$ ulimit -s unlimited
[host]$ cd sharpen/C-SER
[host]$ ./sharpen

Image sharpening code running in serial

Input file is: fuzzy.pgm
Image size is 564 x 770

Using a filter of size 17 x 17

Reading image file: fuzzy.pgm
... done

Starting calculation ...
Program on core 8-15
... finished

Writing output file: sharpened.pgm

... done

Calculation time was 7.549171 seconds
Overall run time was 7.714736 seconds
```

Remember to unlimit the stacksize with *ulimit -s unlimited* before running the code on the host otherwise you'll most likely get a segmentation fault when you run out of memory.

2.4 Viewing the images

To see the effect of the sharpening algorithm, you can view the images using the `display` program.

```
[host]$ display fuzzy.pgm
[host]$ display sharpened.pgm
```

Type "control + q" in the image window to close the program or use the menus.

To view the image you will need an X window client installed. Linux or Mac systems will generally have such a program available, but Windows does not provide X windows functionality by default. There are many X window systems available to install on Windows; we recommend Xming available at:

- http://sourceforge.net/projects/xming/

2.5 Compling the parallel versions of the code for the host

OpenMP and MPI versions of the the code can be found in the *C-OMP* and *C-MPI* directories respectively. As with the serial version, these can be compiled with *make*, e.g. to compile the MPI version of the code you would use:

```
[host]$ cd sharpen/C-MPI
[host]$ make
mpiicc -openmp -g -DC_MPI_PRACTICAL -c sharpen.c
mpiicc -openmp -g -DC_MPI_PRACTICAL -c dosharpen.c
mpiicc -openmp -g -DC_MPI_PRACTICAL -c filter.c
mpiicc -openmp -g -DC_MPI_PRACTICAL -c cio.c
mpiicc -openmp -g -DC_MPI_PRACTICAL -c utilities.c
mpiicc -openmp -g -DC_MPI_PRACTICAL -o sharpen sharpen.o dosharpen.o
    filter.o cio.o utilities.o -lm
```

2.6 Running the parallel versions on the host

Normally parallel codes would be run via a batch system but for simplicity we will just run these interatively on the host. To run the OpenMP version of the code you'll need to set the number of threads via *OMP_NUM_THREADS* and then run the executable as follows:

```
[host]$ cd sharpen/C-OMP
[host]$ export OMP_NUM_THREADS=4
[host]$ ./sharpen

Image sharpening code running on 4 thread(s)

Input file is: fuzzy.pgm
Image size is 564 x 770

Using a filter of size 17 x 17

Reading image file: fuzzy.pgm
... done

Starting calculation ...
Thread 0 on core 0-47
Thread 1 on core 0-47
Thread 3 on core 0-47
Thread 2 on core 0-47
... finished

Writing output file: sharpened.pgm

... done
```

```
Calculation time was 1.576143 seconds
Overall run time was 1.715887 seconds
```

To run the MPI version of the code on the host you need to use *mpirun* to launch the job as follows:

```
  [host]$ cd sharpen/C-MPI
  [host]$ mpirun -np 8 ./sharpen

Image sharpening code running on 8 processor(s)

Input file is: fuzzy.pgm
Image size is 564 x 770

Using a filter of size 17 x 17

Reading image file: fuzzy.pgm
... done

Starting calculation ...
Rank 0 on core 0-2,24-26 of node <crb103>
Rank 4 on core 12-14,36-38 of node <crb103>
Rank 6 on core 18-20,42-44 of node <crb103>
Rank 2 on core 6-8,30-32 of node <crb103>
Rank 3 on core 9-11,33-35 of node <crb103>
Rank 1 on core 3-5,27-29 of node <crb103>
Rank 7 on core 21-23,45-47 of node <crb103>
Rank 5 on core 15-17,39-41 of node <crb103>
... finished

Writing output file: sharpened.pgm

... done

Calculation time was 0.776558 seconds
Overall run time was 0.921541 seconds
```

2.7 Compiling native mode binaries for the Xeon Phi

To compile code which will run in native mode on the Xeon Phi you just need to add the *-mmic* flag to your compile/link flags. In each of the source directories you will find a *Makefile.mic* file which can be used to generate a native mode binary for the Xeon Phi. E.g. to compile a native mode OpenMP binary do the following:

```
  [host]$ cd sharpen/C-OMP
  [host]$ make -f Makefile.mic clean
  [host]$ make -f Makefile.mic
  icc -mmic -openmp -g -DC_OPENMP_PRACTICAL -c sharpen.c
  icc -mmic -openmp -g -DC_OPENMP_PRACTICAL -c dosharpen.c
  icc -mmic -openmp -g -DC_OPENMP_PRACTICAL -c filter.c
  icc -mmic -openmp -g -DC_OPENMP_PRACTICAL -c cio.c
  icc -mmic -openmp -g -DC_OPENMP_PRACTICAL -c utilities.c
  icc -mmic -openmp -g -DC_OPENMP_PRACTICAL -o sharpen.mic sharpen.o dosharpen.o
      filter.o cio.o utilities.o -lm
```

Note: if you've already built the host version then you'll need to do a clean build (with *make -f Makefile.mic clean*) to remove the host specific object files. If you don't remove these files then the compiler will attempt to use the host specific object files which cannot be built into a Xeon Phi native mode compatable binary. You'll get an error message like:

```
[host]$ make -f Makefile.mic
icc -mmic -openmp -g -DC_OPENMP_PRACTICAL -o sharpen.mic sharpen.o
    dosharpen.o filter.o cio.o utilities.o -lm
x86_64-k1om-linux-ld: i386:x86-64 architecture of input file
    'sharpen.o' is incompatible with k1om output
... <SNIP>
```

The MPI native binary can be compiled in exactly the same way. You should ensure you've compiled both the OpenMP and MPI native binaries before moving on to the next section.

2.8 Running native mode binaries on the Xeon Phi

To run in native mode on the Xeon Phi recall that you need to login to the co-processor, set up your environment and run the native mode binary. Thus to run the OpenMP version of the image sharpening code on 4 threads you would do:

```
[host]$ ssh mic0
[mic0]$ source /gpfs/stfc/local/apps/intel/intel_cs/
2016.0.047/mkl/bin/mklvars.sh mic
[mic0]$ ulimit -s unlimited
[mic0]$ export OMP_NUM_THREADS=4
[mic0]$ cd /path_to_source_on_mic/sharpen/C-OMP
[mic0]$ ./sharpen.mic

Image sharpening code running on 4 thread(s)

Input file is: fuzzy.pgm
Image size is 564 x 770

Using a filter of size 17 x 17

Reading image file: fuzzy.pgm
... done

Starting calculation ...
Thread 0 on core 1
Thread 1 on core 5
Thread 3 on core 13
Thread 2 on core 9
... finished

Writing output file: sharpened.pgm

... done

Calculation time was 26.026030 seconds
Overall run time was 27.794996 seconds
```

To run the MPI version of the code on 4 MPI processes you would do:

```
[mic0]$ source /gpfs/stfc/local/apps/intel/intel_cs/
```

```
2016.0.047/mkl/bin/mklvars.sh mic
[mic0]$ source /gpfs/stfc/local/apps/intel/intel_cs/
2016.2.062/impi/5.1.3.181/mic/bin/mpivars.sh
[mic0]$ ulimit -s unlimited
[mic0]$ cd /path_to_source_on_mic/sharpen/C-MPI
[mic0]$ mpirun -np 4 ./sharpen.mic

Image sharpening code running on 4 processor(s)

Input file is: fuzzy.pgm
Image size is 564 x 770

Using a filter of size 17 x 17

Reading image file: fuzzy.pgm
... done

Starting calculation ...
Rank 0 on core 1-60 of node <idb1c18-mic0>
Rank 2 on core 121-180 of node <idb1c18-mic0>
Rank 1 on core 61-120 of node <idb1c18-mic0>
Rank 3 on core 0,181-239 of node <idb1c18-mic0>
... finished

Writing output file: sharpened.pgm

... done

Calculation time was 24.504778 seconds
Overall run time was 26.343592 seconds
```

If you've already setup the environment and unlimited the stacksize when running the OpenMP binary then you don't need to re-run the first two commands. Alternatively you can use the *setup_mic.sh* script to setup the paths and unlimit the stacksize by running the command:

```
[mic0]$ cd sharpen
[mic0]$ source setup_mic.sh
```

You may have noticed that the MPI version of the code is compiled with the *-openmp* flag. This is because we use *omp_get_wtime* timer function to time the image sharpening code. This means you need to ensure the OpenMP libraries are in your path on the Xeon Phi, i.e. you need to make sure you have sourced the *mklvars.sh mic* script.

3 Comparing the performance of the host and Xeon Phi

You should now investigate how the performance of the Xeon Phi compares to that of the host. Ideally you want to compare a fully populated (i.e. 16 OpenMP threads or 16 MPI processes) host node with a fully populated Xeon Phi card (240 OpenMP threads or 240 MPI processes).

The tables at the end of this document may be be helpful in deciding what thread and processor counts to run. The IO time can be obtained by taking the difference of the overall run time and the calculation time. The total CPU time is the calculation time multiplied by the number of cores.

How does the IO time vary with thread or process count on the host and Xeon Phi? How does the 16 thread or process host version compare with using all 240 virtual cores on the Xeon Phi?

# Threads	Overall run time	Calculation time	IO time	Total CPU time
1				
2				
4				
8				
16				

Table 1: Time taken by parallel image processing code on the host: OpenMP version

# Processes	Overall run time	Calculation time	IO time	Total CPU time
1				
2				
4				
8				
16				

Table 2: Time taken by parallel image processing code on the host: MPI version

# Threads	Overall run time	Calculation time	IO time	Total CPU time
4				
16				
30				
60				
120				
180				
240				

Table 3: Time taken by parallel image processing code on the Xeon Phi: OpenMP version

# Processes	Overall run time	Calculation time	IO time	Total CPU time
4				
16				
30				
60				
120				
180				
240				

Table 4: Time taken by parallel image processing code on the Xeon Phi: MPI version