Exercise: Running a Simple Parallel Program

David Henty

1 Introduction

The basic aim of this exercise is to familiarise you with compiling and running parallel programs on `HECToR`. The program does a simple form of image processing to try and sharpen up a fuzzy picture. You will be able to measure the time taken by the code on different numbers of CPUs to check that the execution time decreases with processor count as expected.

2 Compiling the code

All the source files are stored in the tar file `sharpen.tar`. This should be copied to your work directory:

```
cd /work/d26/d26/${USER}
cp /work/d26/d26/guestadm/Cray/sharpen.tar .
```

To unpack the tar file:

```
tar xvf sharpen.tar
```

You will see four directories containing versions of the code in C and Fortran, parallelised with MPI and OpenMP. You can opt to work with whichever of these you are most familiar with. If you have time, you may want to compare the results you obtain from both the MPI and OpenMP versions.

The following text examples assume you are using the Fortran MPI version of the code. You will see similar (but not identical) output if you are using one of the other versions.

A Makefile is supplied for compilation – just change directory and use `make`:

```
cd sharpen
cd F-MPI
make
```

which creates the executable program `sharpen`.

You can look at the input file using the `display` program:

```
export PATH="$PATH:/work/d26/d26/guestadm/bin/"
display fuzzy.pgm
```

and type `q` anywhere in the display window to quit `display`.

3 Running the Code

To run the code on the main system you need to submit a script to the PBS Pro batch system. You are provided with a template script `batch.pbs` which is appropriate for submitting any code parallelised using the Message Passing Interface or OpenMP,

To run the code on 32 processors:

```
qsub -l mppwidth=32,mppnppn=32 batch.pbs
```

or for the OpenMP version:

```
qsub -l mppwidth=1,mppnppn=1,mppdepth=32 batch.pbs
```

`mppwidth` describes the total number of MPI ranks or processes to create. `mppnppn` describes the number of MPI ranks to place on an individual node, on a Cray XE6 this can be between 1 and 24. `mppdepth` describes the number of threads each MPI rank will spawn and spreads the MPI ranks appropriately over the node. This should be the same value as the `OMP_NUM_THREADS` variable.

When the code has completed it will produce an output image `sharp.pgm` and a log file which contains, among other things, information about the execution time. The log file will have a name of the form `batch.pbs.number`, where `number` is the identifier of the job you submitted. You should look at the sharpened image and compare it to the original fuzzy image; you should also run on one processor and check that the output is still correct.

4 Parallel Performance

If you examine the log file you will see that it contains two timings: the total time taken by the entire program (including IO) and the time taken solely by the calculation. The image input and output is not parallelised so this is a serial overhead, performed by a single processor. The calculation part is, in theory, perfectly parallel (each processor operates on different parts of the image) so this should get faster on more processors.

You should do a number of runs and fill in Table 1: the IO time is the difference between the calculation time and the overall run time; the total CPU time is the calculation time multiplied by the number of processors.

Look at your results – do they make sense?

# Processors	Overall run time	Calculation time	IO time	Total CPU time
1				
2				
4				
7				
10				

Table 1: Time taken by parallel image processing code

Given the structure of the code, you would expect the IO time to be roughly constant, and the performance of the calculation to increase linearly with the number of processors: this would give a roughly constant figure for the total CPU time. Is this what you observe?