# Introduction to High Performance Computing – Assignment 1

Michael Rollins - mr16338

## 1 Introduction

In this assignment we were tasked with optimising a piece of code which applied a stencil over an image. To do this incremental changes were made to the code to: reduce the amount of arithmetic operations the computer had to perform; minimise the number of times the program needed to access memory caches; alter data types and layouts; and use vectorising to allow multiple operations in a for loop to be made simultaneously. Throughout the process different compilers were used, using different optimisation flags in order to find which produced the fastest running executable. The latest versions (installed on Blue Crystal) of GCC (7.1.0) and ICC (16-u2) were the fastest so they were used throughout. Using different levels of optimisation, via the command line flags, changed the running time throughout the experimention.

The speeds quoted in this report are the mean of the run times of the code being tan ten times.

## 2 Optimisation

When first running the original, unaltered, code using the GCC compiler on the grid of size 1024 the run time for 100 iterations was 8.31 seconds. When operating on a grid of size 4096 it took 381 seconds to run, running the code on an 8000 x 8000 grid took too long to be worthwhile measuring before some optimisations had been performed. By experimenting with the GCC optimisation compiler flags the -O3 flag was found to speed the code up to 6.79 seconds on the 1024 grid and the -O1 flag sped the 4096 grid up to 119.94 seconds.

The first code change made to speed the code up was to remove the costly division operations within the conditionals, this was simple as they were dividing constants which could easily be done by hand. This sped the code up significantly on the 1024 grid, the run time being decreased to 2.15 seconds using ICC with the –O1 flag. There was less of a significant improvement on the 4096 grid, only decreasing to 93.7 seconds with the same compiler and flag. At this point the grid of size 8000 was running quickly enough to be measured, and over 10 runs it had an average run time of 105.62 with the -O2 flag. The second attempt made to optimise was to remove the index calculations within each of the four conditionals as they are used four times, by declaring a variable n=j+i*ny at the start of each iteration of the inner loop, so that the calculation would only be made once. This did not make any difference in the run times however, as this may have been done automatically by even the lowest level of compiler optimisation.

The next improvement made was to remove the conditional statements nested within the inner for loop. This is because they force the processor to guess whether or not the conditional will be true as it builds its pipeline, if it guesses incorrectly it will have to do the costly operation of computing the pipeline again. The for loops act the same way on all cells of the grid other than the edges, adding together portions of the cells above, below and beside the cell, as well as the cell itself. The conditionals were there to control how the program acted on the edges which do not have all four surrounding cells. To remove the conditionals the for loops were changed to only act from 1 to n-2 in each direction, and the edges and corners calculated separately. This change alone actually slowed down the run time very slightly, however as it allowed for further optimisation to be made later on, it was left in.

The way the original code was written, the processing of the stencil over the grid was done in a column major order, as in it worked down each column and then along the rows which meant that the array that the data was stored in, was being accessed out of order. This is less efficient as the order the cells are accessed is different to how they are stored, so the data will take longer to access. When the code was changed so that the for loops accessed along each row first so that data array was accessed in order, the run time of a 1024 x 1024 grid was dramatically reduced from around 2.5 seconds, depending on whether using GCC or ICC and the flags used, down to a minimum of 0.560 seconds when compiled with GCC -Ofast. With grids of size 4096 and 8000

the run times were reduced from 91.6 seconds to 12.9 seconds and 110.4 seconds to 42.15 seconds respectively. According to the GCC flag -fopt-info-vec-optimized this allowed the compiler to start vectorising the for loop which iterates over the majority of the grid, where as previously the flag indicated no vectorisation had taken place.

When the if statements were removed from the for loops their conditional blocks were left in place, and they had the form:

```
tmp_image[x] += image[y]
```

repeated five times with different values for y, this slowed the code down as it required the compiler to fetch and update the value of tmp_image for each statement. By combining them into one longer statement it meant the value only needed to be fetched and updated once per iteration of the loop. This reduced the run times with GCC -Ofast to 0.236, 5.11 and 18.3 seconds.

The values stored in the grid are always far below the limit for not only doubles as they were originally stored, but also for floating point values. Floats are stored in half as many bytes as doubles which means they can be stored in lower level caches closer to the processor which allows for quicker access. By converting the doubles to floats the run times were once again decreased, the GCC 7.1.0 compiler with the -Ofast flag was still the fastest compiler. The run times on the grids of size 1024, 4096 and 8000 were reduced to 0.154, 2.745 and 10.75 seconds respectively.

By adding the keyword restrict to the declaration of the image pointers, it tells the compiler that only the pointer will be used to access the object to which it points, meaning that there is no overlap between the input and output images. This allows the compiler to perform additional vectorisation and optimisation to further increase the speed the code runs at. With this keyword in place the ICC compiler became the fastest to run the code, using the -fast optimisation flag the 1024 x 1024 grid was able to run at 0.098 seconds and using the -O3 flag the 4096 x 4096 grid ran at 2.468 seconds and the 8000 x 8000 grid at 8.610 seconds.

The final improvement made to the code was simply reducing the number of arithmetic operations performed. This was possible as each of the cells that surround the one being operated on have their values multiplied by 0.1 and then added to the output image, so by adding all the surrounding cells' values first and then multiplying by 0.1, the number of multiplication operations are reduced. This change, when compiled using ICC -O3, produced run times for the three grids of 0.095, 2.465, 8.555 seconds. However if compiled with GCC-7.1.0 -Ofast and the -march=native flag, which tells the compiler to optimise the code for the processor on which it is compiled, the run times are reduced to 0.088, 2.424, and 8.531 seconds. These were the fastest times achieved.

# 3 Conclusion

The optimisations which caused the most improvement in the run time of the code were when they improved the vectorisation of the main for loops. The first improvement which did this was when the order in which the stencil was applied was changed into row major order, this is when the loop was first vectorised and it sped the smallest grid up by over four times, the 4096 grid by seven times and the 8000 grid by three times. The second time vectorisation was affected was when the restrict keyword was applied to the declarations of the images. This sped all sizes of grid by about two times. This was because vectorisation allows for operations to be performed on multiple cells of the grid simultaneously, rather than some of the other optimisations which just reduced the number of operations or the amount of memory being loaded and stored by a small percentage.

| | | | | |
|---|---|---|---|---|
| ICC 16.0.3 -fast | Removed division | 2.1578229 | 93.919032 | 106.7014983 |
| ICC 16.0.2 -fast | Remove conditionals from loops | 2.5889 | 91.500536 | 111.2027329 |
| GCC 7.1.0 -Ofast | Remove conditionals from loops | 2.45759633 | 97.7539779 | 109.2641595 |
| ICC 16.0.2 -fast | Reordered to be row major | 0.7815351 | 12.8914647 | 43.1529387 |
| GCC 7.1.0 -Ofast | Reordered to be row major | 0.5595526 | 9.2561674 | 33.982348 |
| ICC 16.0.2 -fast | Tried to remove some mem reqs | 0.3019891 | 6.1135916 | 22.7378788 |
| GCC 7.1.0 -Ofast | Tried to remove some mem reqs | 0.236144 | 5.1122622 | 18.21469 |
| GCC 7.1.0 -Ofast | Changed doubles to floats | 0.1539761 | 2.7453929 | 10.751992 |
| ICC 16.0.2 -O3 | Changed doubles to floats | 0.2984827 | 5.0025737 | 19.5041971 |
| GCC 7.1.0 -Ofast | Added restrict keyword | 0.1330138 | 2.6342547 | 9.6754745 |
| ICC 16.0.2 -fast | Added restrict keyword | 0.0988891 | 2.5847829 | 8.7861546 |
| ICC 16.0.2 -O3 | Added restrict keyword | 0.1010225 | 2.4681371 | 8.6107392 |
| ICC 16.0.2 -fast | Added parens to reduce mults | 0.933067 | 2.5389406 | 8.6901117 |
| ICC 16.0.2 -O3 | Added parens to reduce mults | 0.0945428 | 2.4664149 | 8.5553821 |
| GCC 7.1.0 -Ofast -march | Added march=native flag | 0.0884445 | 2.4240147 | 8.5312949 |