Tim McMullen - 159.735 parallel

Assignment 5

Mandelbrot Set

MPI version

The MPI version of the Mandelbrot set works similarly to the sequential version. To implement this code for MPI we use the working pool technique in which the data is sent to each node as it is requested. The MPI_Recv's in this program are what allows the working pool to function, by allowing the master to accept an incoming request from any processor, this is important as with the Mandelbrot set some lines take substantially longer to process and would hold up the system if we were unable to use MPI_ANY_SOURCE.

The Program starts by firstly having the master node sending each processor a starting line number to work from, once they have completed calculating the pixels for that line they then send the master node their ID and what line they have just completed, along with the the line information itself. the master node then sends them the next line to work on before adding the line information to the image, once the image has been completed it is then written to a fits file.

Cuda version

To implement this program using cuda was very simple as we use streaming processors removing the need to implement a working pool system. To start with, we create a kernel which is then given the image sizes and other required information. On the graphics card we then assign each processor a different section to work on. Each processor then works with the data passed to the kernel by putting it through a loop which is able then able to add the data directly to the image buffer once the correct conditions have been meet.

By using Nvidia's Compute Visual Profiler it is visible to see that the majority of the work required for the graphics card to perform this task is with the time taken to transfer the completed data from the device to the host, taking 76.98% of the total time.

MPI vs CUDA

Results

MPI (iimscluster)

Number of Proccessors - Number of Nodes	Time taken
4 - 2	1.278628
3-2	1.686247
2-1	2.798115

Cuda

using ./time to time the program the result was that using the cuda implementation it took .345 s

Cuda(Using Compute Visual Profiler)

Method	#Calls	GPU usec	%GPU time
Calc_pixals	1	4458.08	23.01
memcpyDtoH	1	14916.3	76.98

The above results show that by using Cuda to create our solution we are able to dramatically reduce time, and even by spreading the load over 4 normal processor, the cuda implementation was still faster. The advantage of using GPUs over a traditional cluster is that the GPU is local, it’s a streaming processor, is faster at accessing system data, and each processor is able to access the same data at once. While the GPU does come with its disadvantages such as that it is not nearly as scalable, and that in creating a GPU processor some shortcuts are taken, slowing down the GPUs decision making ability, and that it is possible for each processor to be working on a different iteration, but still editing the same memory address, which could cause problems in finding the correct solution

By using a Cluster we increase the communication time as the data has to travel from one node to another which is considerably slower than moving data from a processor to a graphics card, but we are able to hold all the data at once centralized location, and by using standard processors we are able to use IF statements and loops without considerably impacting on the overall performance of the system.

This leads to the conclusion that if more nodes were available the MPI solution could be faster, but a balance would need to be found between the number of nodes working on the problem, and the time taken to pass data between each of those node. But as shown by the results a single GPU was able to out perform 4 processors by roughly 0.9s meaning that for this problem and with these resources a GPU based solution is by far the best option.