CMT106: High Performance Computing

HPC Lab 3: Programming with CUDA

Where: C/2.08, Queen’s Buildings (second floor, central building)
When: 1:30 -3:00pm, 26 November 2019

Description of problem

You will be given a sequential program, laplaceSEQ1D.c, which solves the two-dimensional Laplace equation on a rectangle. The aim is to determine the electric field around a conducting object held at a fixed electrical potential inside a box at a fixed electrical potential of 1, with the electrical potential on the boundary being fixed at 0.

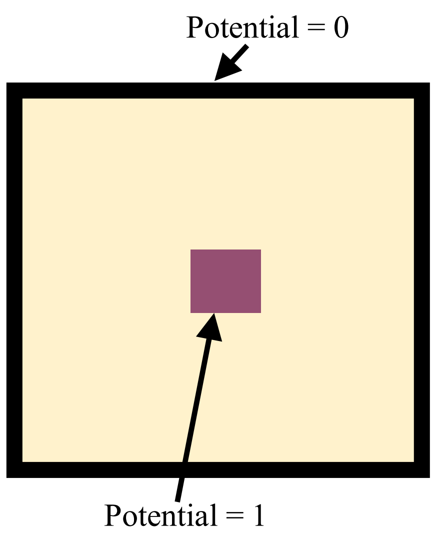

Figure 1: The conducting object is shown in purple. We want to find the electrical potential in the yellow shaded region.

The code uses an iterative algorithm to find the approximate solution at a rectangular grid of points within the yellow region in Fig. 1. The main computational work is done in the following lines of code:

```
for(k=1;k<=nsteps;k++){
    for(j=0;j<nptsy;j++)
      for(i=0;i<nptsx;i++){
        x = j*nptsx+i;
        if (mask[x]) phi[x] = 0.25*(oldphi[x+1] +
          oldphi[x-1] + oldphi[x+nptsx] + oldphi[x-nptsx]);
          }
```

The outer loop over k is the iterative loop. This contains two doubly-nested for loops. The first updates the solution by replacing phi at each point by the average of the neighbouring values from the previous iteration. The mask array is set to 0 on the boundary and at the 4 central points (we don’t want to change these values of phi), and is 1 at all other points. The second doubly-nested loop copies the solution from the current iteration (phi) to the oldphi array.

What you must do

1. Parallelise the code for a GPU using CUDA, following the code in the PowerPoint slides. You should include the cuda.h file at the start of the code, and change the name of the output file in the output_array routine from “outSEQ.ps” to “outGPU.ps”. In the performUpdates routine you must allocate memory on the GPU for the phi, oldphi and mask arrays, and copy the phi and mask arrays to the GPU. After performing the updates you need to copy the phi array from the device to the host and free the arrays in the GPU memory. The parallel code should be called laplaceGPU.cu

Optional Tasks

You probably won’t have time for the following tasks in the lab:

1. After the parallel code is working correctly you should run it for thread blocks of different size and record the times taken. For example, 4x4, 8x8, 8x16, 16x16, and 16x32.
2. Plot a histogram of the time taken for the different thread block sizes, and compare these with the time taken by the sequential code, laplaceSEQ1D.c. Include the problem size and number of iterations on the plot (these are the values nx, ny, and ns mentioned below).

What to do if you need help

If you need help, ask me or the lab assistant. If I’m not in the lab you can email me at WalkerDW@cardiff.ac.uk.

How to compile and run the code

To compile the sequential code:

```
gcc -o laplaceSEQ1D laplaceSEQ1D.c -lm
```

To compile the CUDA code:

```
nvcc -o laplaceGPU laplaceGPU.cu -lm
```

To run the sequential code:

```
laplaceSEQ1D -nx 1000 -ny 1000 -ns 2000
```

This will run the sequential code on a 1000x1000 grid for 2000 iterations.

To run the CUDA code:

```
laplaceGPU -nx 1000 -ny 1000 -ns 2000
```

The sequential and CUDA codes should give exactly the same output files. You can check this by doing:

```
diff outSEQ1D.ps outGPU.ps
```

If this produces no output then outSEQ.ps and outGPU.ps are the same.

You can visualise the output using any PostScript viewing tool. For example:

```
gs outSEQ1D.ps
```