GPU Programming
Prac 4

Instructions
Due date: noon on Friday 17 May 2024

By submitting your assignment via the RUConnected link, you declare that the assignment submitted is entirely your own work, unless noted otherwise.

Aim
Create a naive CUDA version of the sequential code provided for the Laplace solver.

Laplace solver (30 marks)

Rewrite the sequential code given for a Laplace solver in CUDA-C. Note you do not need to optimise this code. A running version that has all the necessary synchronization and a sensible launch configuration (i.e. grid / block sizes) is all that is necessary. A 1D grid is acceptable, however, it is likely that a 2D block is the most sensible working option.

Note: you may make use of any CUDA concepts covered in any part of this course, including the use of Unified Memory.

There are two versions of the Laplace_serial code uploaded: one for running sequentially on Windows (*Laplace_serial_MS.c*) and the other for Linux (*Laplace_serial_linux.c*). You may use either version.

Submit your CUDA C/C++ code as well as the overall timing of this code as calculated on the CPU from start to finish of main().

Allocation of marks will be as follows:
Execution model (i.e. launch config, etc.) 5
Actual laplace kernel code 10
Boiler plate code 5
Dealing with boundary conditions 5
Memory usage and synchronisation (if needed) 5

Execution should produce something like:
---------- Iteration number: 100 ------------
[995,995]: 63.33 [996,996]: 72.67 [997,997]: 81.40 [998,998]: 88.97 [999,999]: 94.86
[1000,1000]: 98.67

Max error at iteration 100 was 0.355752
Total time was 0.770932 seconds

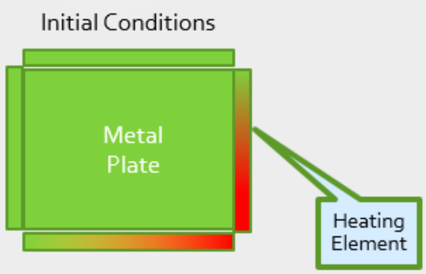
Additional information on the Laplace algorithm is given below.

Laplace solver

- This is a great simulation problem.
- In this most basic form, it solves the Laplace equation:

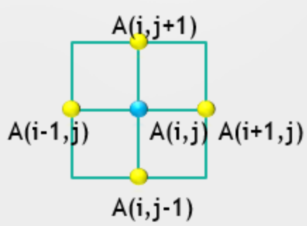
$$\nabla^2 f(x, y) = 0$$

- The Laplace Equation applies to many physical problems, including: electrostatics, fluid flow, and temperature
- For temperature, it is the Steady State Heat Equation:



Jacobi iteration

- The Laplace equation on a grid states that each grid point is the average of its neighbors.
- We can iteratively converge to that state by repeatedly computing new values at each point from the average of neighboring points (e.g. temperature).
- We just keep doing this until the difference from one pass to the next is small enough for us to tolerate.



$$A_{k+1}(i,j) = \frac{A_k(i-1,j) + A_k(i+1,j) + A_k(i,j-1) + A_k(i,j+1)}{4}$$

Basic implementation

```
for(i = 1; i <= ROWS; i++)
 {for(j = 1; j <= COLUMNS; j++)
    {Temp[i][j] = 0.25 *
     (Temp_last[i+1][j] + Temp_last[i-1][j] +
     Temp_last[i][j+1] + Temp_last[i][j-1]);
    }
 }
```

Serial C code

```
while ( dt > MAX_TEMP_ERROR && iteration <= max_iterations ) {

     for(i = 1; i <= ROWS; i++) {
          for(j = 1; j <= COLUMNS; j++) {
                Temperature[i][j] = 0.25 *
                 (Temperature_last[i+1][j] + Temperature_last[i-1][j] +
                  Temperature_last[i][j+1] + Temperature_last[i][j-1]);
          }
     }

     dt = 0.0;

     for(i = 1; i <= ROWS; i++){
        for(j = 1; j <= COLUMNS; j++){
             dt = fmax( fabs(Temperature[i][j]
                                  -Temperature_last[i][j]), dt);
             Temperature_last[i][j] = Temperature[i][j];
          }
     }
     if((iteration % 100) == 0) {
          track_progress(iteration);
     }
     iteration++;
}
```

Done?

Calculate

Update temp array and find max change

Output