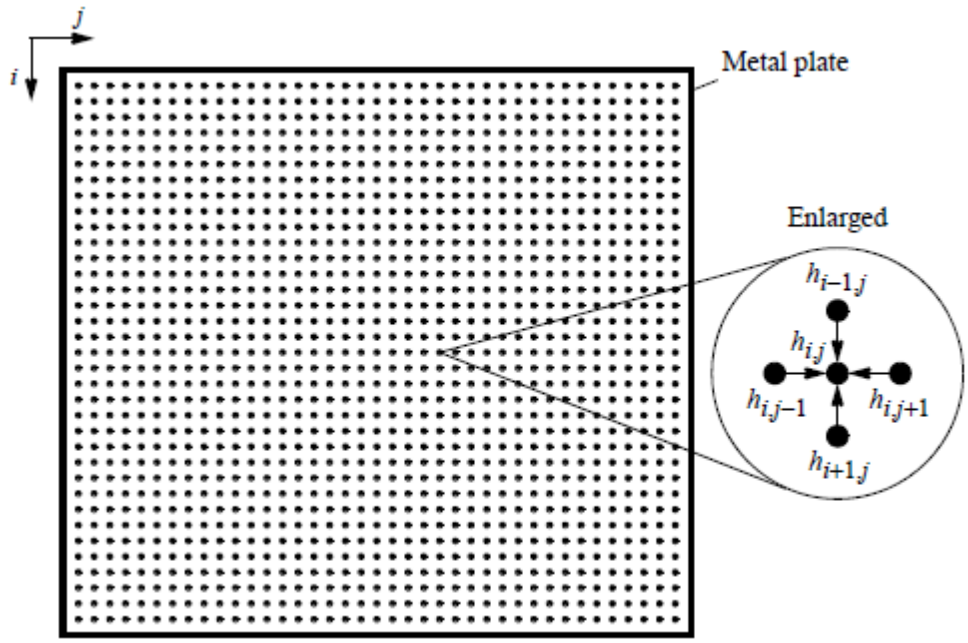
CSCI-GA.3033-004
Graphics Processing Units (GPUs): Architecture and Programming
Programming Assignment 1

In your first programming assignment, you will implement a GPU friendly application and see yourself when a GPU friendly application will yield better performance than sequential version and when not.
You will write CUDA program to determine the heat distribution in a space using synchronous iteration on a GPU. We have 2-dimensional square space and simple boundary conditions (edge points have fixed temperatures). The objective is to find the temperature distribution within. The temperature of the interior depends upon the temperatures around it. We find the temperature distribution by dividing the area into a fine mesh of points, $h_{i,j}$. The temperature at an inside point is calculated as the average of the temperatures of the four neighboring points, as shown in the following figure.



The edge points are when i = 0, i = n-1, j = 0, or j = n-1, and have fixed values corresponding to the fixed temperatures of the edges. The temperature at each interior point is calculated as:

$$h_{i,j} = \frac{h_{i-1,j} + h_{i+1,j} + h_{i,j-1} + h_{i,j+1}}{4}$$

(**0 < i < n** ; **0 < j < n**, remember that edge points have fixed values)

Stopping condition: You can have a fixed number of iterations or when the difference between two consecutive iterations (calculated as average among all points) is less than or equal to some number *e*. For this assignment, we will use fixed number of iterations.

Assume we have (n x n) points (including edge points), the initial situation is as follows:

- The edge points are fixed at 80F, except:
- Points (0, 10) to (0, 30) inclusive have temperature of 150F.
- All internal points are initialized to zero.

What you have to do:

1. [40 points] Write CUDA version with all optimizations you can think of, in the provided file heatdist.cu. The simplest way to compile your code is:
   nvcc -o heatdist heatdist.cu
   The generated binary will have the name: *heatdist.*
   Type: **./heatdist** and press enter to see the usage. Then start implementing your GPU version in the designated place in the *.cu file. Read the comments on that file before starting to code.
2. Experiment 1: Number of iterations is fixed to 50. Starting with n = 100 and then doubling in each experiment (i.e. 100, 200, 400, 800, 1600, and 3200), record the time taken by the CPU and that taken by the GPU. The provided file already calculates the time for you and print it on the screen,
3. Experiment 2: Fix n to 1600 and vary the number of iterations starting from 50 and doubling (50, 100, 200, 400, 800, and 1600).
4. [20 points, 10 per graph)] Draw a bar-graph showing n (x-axis) and the time (y-axis) for the 2 versions of the program (CPU and GPU) (2 bars per n on the same graph) for each experiment.
5. [40 points, 10 per bullet point] What are your conclusions regarding:
   a) When is GPU usage more beneficial (at which n)? and why?
   b) When is the speedup (i.e. time of CPU version / time of CUDA version) at its lowest? And why?
   c) When is the speedup at its highest? And why?
   d) Which has more effect: number of iterations or the problem size? and why?

   Note: In answering the above questions, you need to **provide numbers** to explain your conclusions, not just stating answers like "communication cost is high"**.** Make of nvprof to get some useful profiling info. You can simply type:
   nvprof ./heatdist <arguments of headdist >

What to submit:

- The source code heatdist.cu
- A report that contains the graphs and answers to the questions mentioned above.

Put all the above in a zip file named by your lastname-firstname.zip and upload them to NYU classes.

Important: Do this lab on either cuda2 or cuda5

I hope that after implementing this programming assignment, **you learned the following items** first-hand:

- How to allocate/free memory space at the device.
- How to move data from host to device, and back, if needed.
- How to launch a kernel from host to device.
- Get a feel of when the problem size is enough to get better performance on device when the kernel is GPU friendly.

So …..

Have Fun!