Summary

Your goal here is to take a naïve, serial code that runs a percolation model, and make this run correctly and efficiently using a single GPU on Cirrus (i.e. one NVIDIA V100).

You must also prepare a brief report (maximum 1 page) explaining, with reference to the algorithm and hardware, how the changes you have made to the code achieve this performance.

As part of marking, your code will be compiled and run on Cirrus using the `gpu-cascade` partition, so please use that for any tuning and profiling you do.

There are two versions of the software available, written in C and Fortran. **Please choose one version to work with as you prefer - there is no difference in marks available.**

Expected time to complete: 5 hours

Problem description

So called "percolation models" are used as simplified models of various systems, including forest fires, disease propagation, and flow in porous media. Here we focus on the latter, asking: given a random lattice 2D material with a porosity `p` (i.e. the material is approximated as a grid of equal sized squares, each being empty with independent probability `p`), do the pore spaces connect?

This can be done by first labelling each non-solid cell with a unique number, then iteratively updating this value to the maximum of the labels at that cell and its four immediate neighbours, i.e.

```
new_label[i,j] = max(
                 label[  i,j+1],
label[i-1,  j], label[  i,  j], label[i+1, j]
                 label[  i,j-1]
)
```

To know when to finish, the algorithm needs to know when this process has converged, i.e. when the total number of changes across the whole grid is zero.

The supplied code applies this, in serial, on the CPU. The driver code runs this once on the CPU to compute a reference solution, then a number of times using the GPU implementation (in the initial code, this is simply a copy of the CPU implementation with an altered function name), before validating the solution, printing timing statistics, and writing the a image of the output.

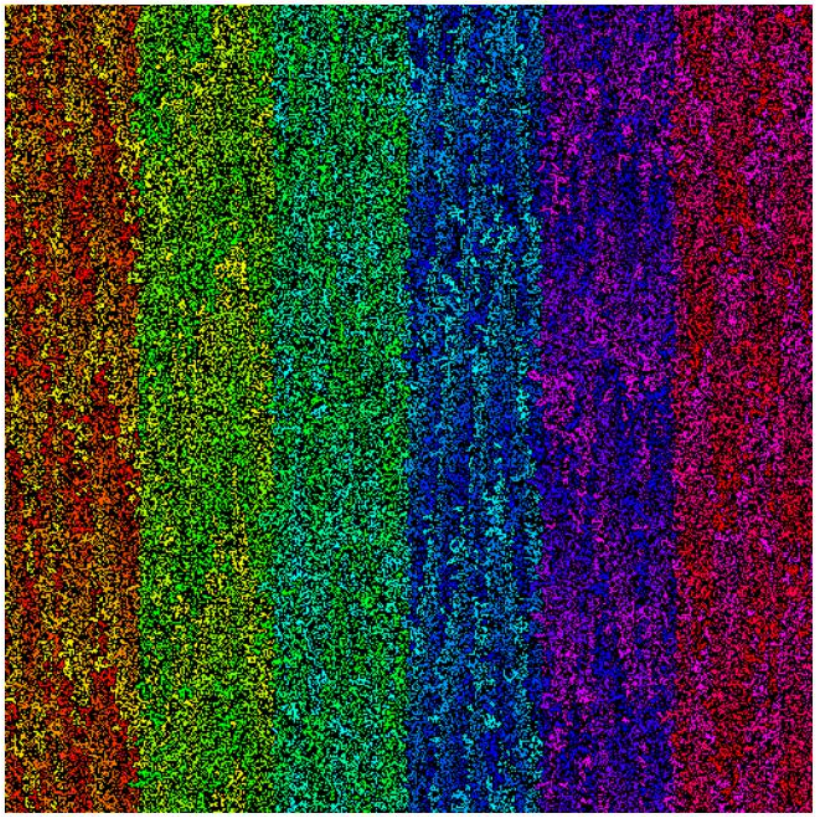

Figure 1: Sample output image

Figure 1: Sample output image

It accepts the following flags:

`-M <integer>` - horizontal size of grid (default 512)

`-N <integer>` - vertical size of grid (default 512)

`-S <integer>` - random seed (default 1234)

`-r <integer>` - number of repeats for benchmarking (default 3)

`-p <float>` - target porosity between 0 and 1 (default 0.4)

`-o <path>` - file name to write output PNG image (default `test.png`)

Set up

1. Clone the code repository into your home directory `git clone /lustre/home/shared/apt-cuda-cw`
2. Choose C (`cd apt-cuda-cw/c`)or Fortran (`cd apt-cuda-cw/fortran`) versions of the assessment.
3. Load the required NVIDIA module. For C use `module load nvidia/cuda-11.2` and for Fortran use `module load nvidia/nvhpc/21.2`.
4. Compile the code with `make`. This will produce an executable `test`.
5. Running the *unmodified* code is possible on the login nodes. If you run with no options (just `./test`) the produced image should be identical to the `default_output.png` in the root of the repository.
6. I have provided several template batch scripts, one (`run.sh`) simply runs the executable while the other two profile it.

Requirements for your code

You must adapt the code so that it runs on one GPU, correctly and with good performance.

Your code will form part of the submission and is worth 50% of the marks for this assignment. You may only change either `c/perc_gpu.cu` or `fortran/perc_gpu.cuf`. This submitted file will be added to a copy of the supplied repository for benchmarking.

Your code must compile (with only the modules given above loaded) simply by running `make` and run with the existing command line options. Note that a code that does not compile is, by definition incorrect, and that a code that produces incorrect results scores no points for performance.

You may use functions and types from the standard library and base CUDA library, but from no other sources.

To be sure your code works, please test it on Cirrus and submit that version!

Clarity: your modifications will be marked for usual good practice in programming. Are variables/functions sensibly named? Code well formatted? Are comments present, where necessary, to explain what is not obvious?

Correctness: the benchmark code compares the GPU results to those from the (unmodified) serial CPU version. I will compile and run this against a range of problem sizes and porosities.

Performance: I will use several problems of different sizes and porosities, taking the best performance run in each case. Sizes will be between 128 and 16,384 inclusive along each dimension.

Requirements for report

This must be a PDF with a maximum of one A4 page and text at 12pt or greater.

You must give a clear statement of whether you are using C or Fortran for your solution and then answer the following (please number your answer).

1. Why is this problem more complex to solve on a GPU than a single core? (2-3 sentences) [10 %]
2. Give a brief outline of the approach you have chosen. Which features of the hardware/software does it use to achieve good performance, despite the points raised in (1)? You might choose to refer to your code (e.g. "see comments on lines 90-95 of perc_gpu.cuf for full details"). What level of performance constitutes good and why? [30%]
3. A brief discussion of your process. You may wish to include: approaches considered/tried but discounted; description of any debugging/profiling/tuning; potential improvements you might make. [10%]