**CME213, Spring 2022**
**Parallel Computing**
**Julian Cooper**

## Homework 3
## Due Sunday, May 1st via GradeScope

**Problem 1: Recurrence** Implement a simple CUDA program for a recurrence relation (inspired by the Mandelbrot Set) for many different starting points.

- **1.1 Allocate GPU memory** Idea: Use `cudaMalloc` to allocate memory on the GPU device for both the input and output arrays we will need for the recurrence implementation. Free memory with `cudaFree` at end of `main()`.

```
        // Allocate num_bytes of memory to the device arrays
        cudaMalloc(&device_input_array, num_bytes);
        cudaMalloc(&device_output_array, num_bytes);
        ...
        ...

        // Deallocate memory from both device arrays
        cudaFree(device_input_array);
        cudaFree(device_output_array);
```

- **1.2 Initialize array of random floats** Idea: Use in-built rand() functor from the standard cpp library and scale it between -1 and 1 as required.

```
        // Initialize an array of size arr_size in input_array with
        //random floats between -1 and 1
        void initialize_array(vec &input_array, size_t arr_size) {
          input_array.resize(arr_size);
          std::generate(input_array.begin(), input_array.end(), rand);

          for(int i=0; i<arr_size; i++){
            input_array[i] = static_cast<float>(input_array[i])/
                                (RAND_MAX/2)-1;
          }
        }
```

- **1.3 Implement recurrence kernel** Idea: Recurrence operations themselves are not independent and therefore not parallelizable, however, we can parallelize doing many of these recurrence loops for different starting points (constants). So, we want to parallelize over our 1-dim input array of constants.

  Since our kernel needs to handle cases where the number of threads is less than the number of entries in our input array, we need to use a grid-stride loop.

```
    /**
    * Implement the kernel recurrence.
```

```
3       * The CPU implementation is in host_recurrence() in main_q1.cu.
4       */
5       __global__ void recurrence(const elem_type* input_array,
6                                  elem_type* output_array,
7                                  size_t num_iter,
8                                  size_t array_length) {
9
10          for (int xid = blockIdx.x * blockDim.x + threadIdx.x;
11               xid < array_length;
12               xid += blockDim.x * gridDim.x) {
13
14              elem_type z = 0;
15              elem_type constant = input_array[xid];
16
17              int it=0;
18              while(it<num_iter) {
19              z = z * z + constant;
20              it++;
21              }
22              output_array[xid] = z;
23          }
24      }
```

Console logs.

```
Starting at Fri Apr 29 00:28:45 UTC 2022

nvcc -O3 -std=c++11 -arch=compute_75 -code=sm_75 -o main_q1 main_q1.cu

Output from main_q1
----------------
Largest error found at pos: 15 error 7.81565e-08
    expected 1.52526 and got 1.52526

Largest error found at pos: 0 error 0
    expected 0.680375 and got 0.680375

Largest error found at pos: 439038 error 1.19193e-07
    expected 1.00014 and got 1.00014

Largest error found at pos: 142710 error 2.38333e-07
    expected 2.00072 and got 2.00072

Largest error found at pos: 482709 error 5.61004e-07
    expected 16.9994 and got 16.9994

Largest error found at pos: 482709 error 1.15797e-06
    expected 289.897 and got 289.898
```

```
Largest error found at pos: 482709 error 2.324e-06
    expected 84041.4 and got 84041.6

Largest error found at pos: 482709 error 4.63941e-06
    expected 7.06296e+09 and got 7.063e+09

Largest error found at pos: 482709 error 9.25711e-06
    expected 4.98854e+19 and got 4.98859e+19

Largest error found at pos: 138972 error 1.79297e-05
    expected 8.03779e+21 and got 8.03765e+21

Largest error found at pos: 905817 error 2.59306e-05
    expected 1.66519e+35 and got 1.66523e+35

Questions 1.1-1.3: your code passed all the tests!
```

- **1.4 Vary number of threads per block** When we vary number of threads per block while holding number of iterations (40,000), array size (1e6) and grid size (72) constant, we get the plot shown below.

  Performance improves roughly linearly with threads per block for the first 400 threads, after which we get asymptotic behaviour and a saw-tooth pattern. This is because we reach the point where the overhead incurred from additional threads starts to dominate performance benefit of computing incremental recurrence relations in parallel.

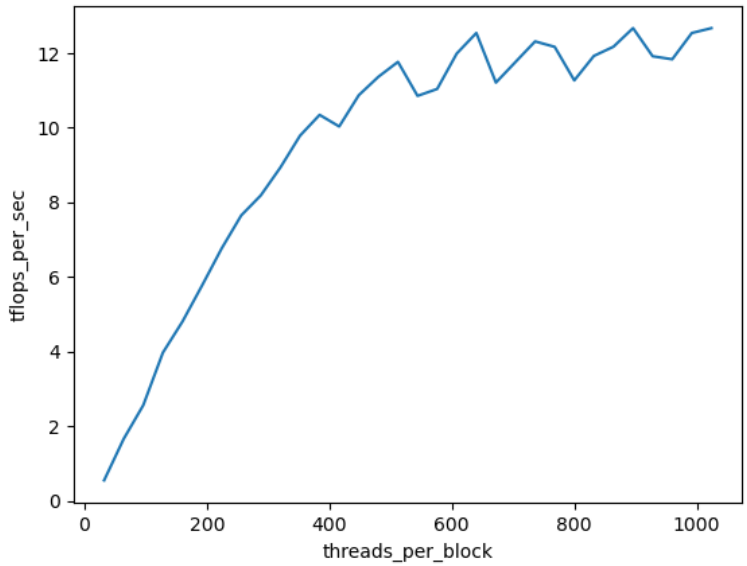


Figure 1: Performance as we vary number of threads per block

- **1.5 Vary number of blocks** When we vary number of blocks (grid size) while holding number of iterations (40,000), array size (1e6) and threads per block (128) constant, we get the plot shown below.

  Performance improves roughly linearly with number of blocks for the first 200 blocks, after which we get asymptotic behaviour and a saw-tooth pattern. Similar to q1.4, this is because we reach the point where the overhead incurred from additional blocks starts to dominate performance benefit of computing incremental recurrence relations in parallel.

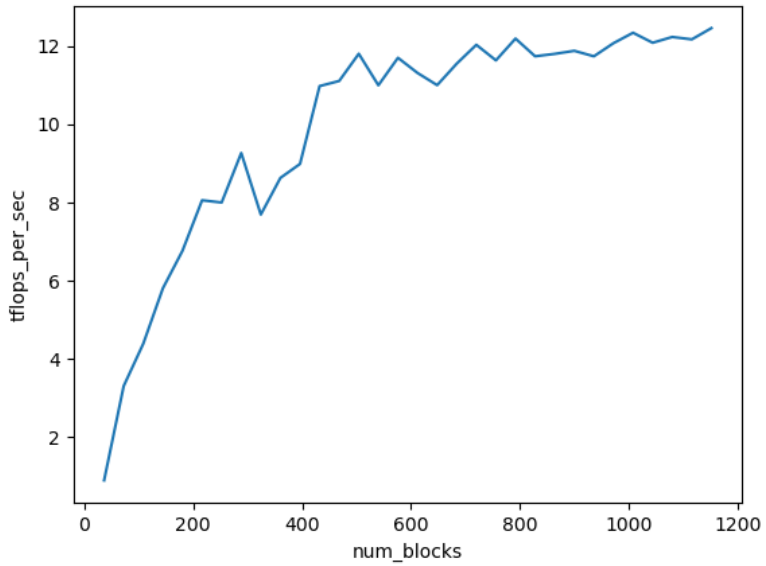


Figure 2: Performance as we vary number of blocks

- **1.6 Vary number of iterations** When we vary number of iterations while holding number of threads per block (256), number of blocks (576) and array size (1e6) constant, we get the plot shown below.

  Performance jumps quickly to 7 Tflops per second as we increase number of iterations from 20-120, after which we get asymptotic behaviour and a saw-tooth pattern. Iteratiions are done sequentially and so a higher number of iterations means that each thread takes longer to complete its task before reassigned. This initially means that we reduce overhead of assigning threads per iterations, but eventually we are working with tasks that are too big to spread evenly across threads (i.e. last thread running has a large portion of its task still to complete).

**Problem 2: PageRank** ...

- **2.1: ...** ...

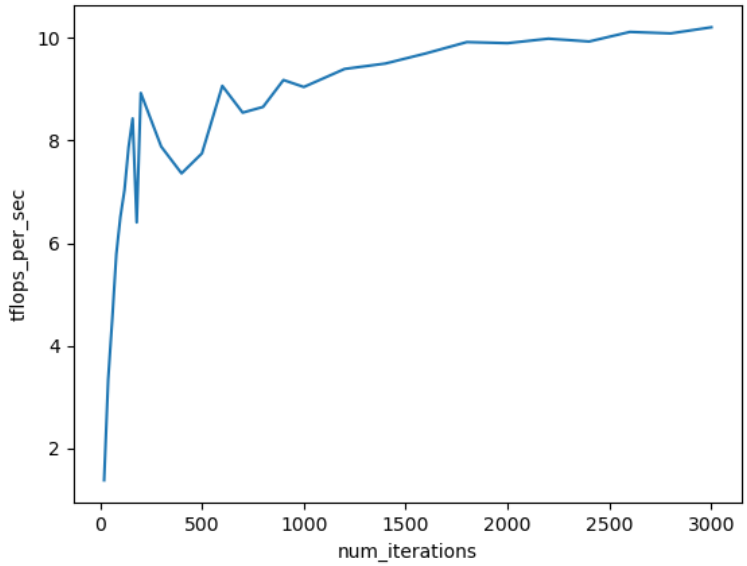


Figure 3: Performance as we vary number of iterations

Submission information logs.

```
jelc@cardinal1:~$ /afs/ir.stanford.edu/class/cme213/script/submit.py hw2 private/cme213-
Submission for assignment 'hw2' as user 'jelc'
Attempt 1/10
Time stamp: 2022-04-13 20:09
List of files being copied:
    private/cme213-jelc53/hw2/sum.h  [768 bytes]
    private/cme213-jelc53/hw2/parallel_radix_sort.h  [7625 bytes]

Your files were copied successfully.
Directory where files were copied: /afs/ir.stanford.edu/class/cme213/submissions/hw2/jel
List of files in this directory:
    sum.h  [768 bytes]
    parallel_radix_sort.h  [7625 bytes]
    metadata  [137 bytes]

This completes the submission process. Thank you!

jelc@cardinal1:~$ ls /afs/ir.stanford.edu/class/cme213/submissions/hw2/jelc/1
metadata  parallel_radix_sort.h  sum.h
```