Semestral Project Announcement

Jiří Filipovič

Fall 2024

Savings Accounting

The goal of the project is to implement an GPU-accelerated code that accounts savings of c customers for p discreete periods

- the input is a 2D array, which contains money sent to a savings account for each custommer
  - a column of the array contains info about one customer
  - a row of the array contains money added in one time period
- the output is a 2D array containing accounts ballance and 1D array summing all money per period

```
void solveCPU(int *changes, int *account, int *sum,
              int clients, int periods) {
  for (int i = 0; i < clients; i++)
    account[i] = changes[i]; // the first change is copied
  for (int j = 1; j < periods; j++) {
    for (int i = 0; i < clients; i++) {
      account[j*clients + i] = account[(j-1)*clients + i]
        + changes[j*clients + i];
    }
  }

  for (int j = 0; j < periods; j++) {
      int s = 0;
    for (int i = 0; i < clients; i++) {
      s += account[j*clients + i];
    }
    sum[j] = s;
  }
}
```

Implementation

You get a framework, which does all the boring stuff:

- creates input, copies it into GPU memory
- check result of CUDA implementation against non-optimized CPU code
- benchmarks your code

Your work

- you are expected to write CUDA code (kernel and code calling the kernel in file `kernel.cu`)
- you can get inspiration (and precise specification) from unoptimized code in `kernel_CPU.C`
- compilation: `nvcc -o framework framework.cu`, you don't need to use Makefile

Project Rules

What will be tested?

- the input size predefined in `framework.cu` can be changed (can be rectangular)
- the size will be divisible by 128 in each dimension
- the code should run on computing capability 3.0 and newer
- your code can expect that output arrays are zeroized

What is forbidden?

- collaboration (discuss general questions, not your code)

Project Stages

The project has three stages:

- running parallel implementation (till Nov 4th): 25p
- efficient implementation (till Dec 2nd, required performance discussed on the next slide): 25p
- the final competititon (till Dec 12th): up to 20p for above average implementations

Submit all stages via IS.

The First Stage

Write a correct implementation in C for CUDA.

- the performance is not relevant
- but must be efficiently parallelized (computation in multiple blocks, each having multiple threads, all doing something useful :-))
- till November 4th (any daytime)
- the points will be assigned according to the functionality of your code (check different input sizes!), if delayed, -2 points for each day of delay
- I highly recommend to start with optimization immediatelly after you have a functional code

The Second Stage

Write an efficent implementation in C for CUDA.

- tested on input size 8192×8192
- performance on airacuda (GeForce GTX 1070): 200 megavalues/s
- performance on barracuda (GeForce RTX 2080 Ti): 400 megavalues/s
- you have to deliver fast **and** correct code (also for different input sizes than 8192×8192)
- I will compile the code with AIRACUDA/BARRACUDA macro, so you can use it if you want different optimizations for {aira,barra}cuda
- till December 2nd (any daytime)
- the points will be assigned according to the speed of your code (you have to match performance of both machines), if delayed, -2 points for each day of delay

The Third Stage

Submit your best code.

- tested on input size 8192×8192 and 512×512
- the score will be computed as a sum of performances of all combinations of inputs and machines
- the code has to be correct, otherwise zero score is assigned
- the students with above average score will get from 1 to 20 points according to their position
- till December 12th (any daytime), there is no possibility to submit your code after the deadline (fair play)