Programming Massively Parallel Hardware

Assignment 3

Viktor Hansen

Task 1 - Matrix Transposition

The solutions are implemented in the `code/task1` directory. To compile and run the tests, invoke `make` followed by `make run` from the `code/task1` directory. The same is the case for the rest of the implemented exercises in this assignment.

Task 2 - Usefulness of Matrix Transposition

2.a - Loop level parallelism

The outer loop is not parallel as the the `accum` is declared outside the loop and needs to be privatized to preserve the semantics. Rewriting the code yields:

```
1. for i from 0 to N-1 // outer loop
2.   accum[i] = A[i,0] * A[i,0];
3.   B[i,0] = accum;
4.   for j from 1 to 63 // inner loop
5.     tmpA   = A[i, j];
6.     accum  = sqrt(accum) + tmpA*tmpA;
7.     B[i,j] = accum;
```

Listing 1: Code with `accum` privatized.

There are no loop-carried dependencies in the inner loops, and thus it can be parallelized. Expressing it in terms of parallel operators yields `scanInc (\a e -> sqrt(a) + e*e) accum`. Rewriting line 6 as `accum = accum + tmpA*tmpA` means that the inner loop can be rewritten as a map/scan (or segmented scan) composition. The Haskell code for the inner loop would look like `scanInc (+) accum $ map (\x->x*x)`.

Task 3 - Matrix Multiplication

3.a - Sequential Implementation

The sequential implementation is to be found in `code/matrix.cu`.

3.b - OMP Implementation

Not implemented.

3.c - Naive CUDA Implementation

Implemented in `code/matrix.cu`.

3.d - Tiled CUDA Implementation

Attempted implementation of kernel in `code/matrix_kernels.cu.h`, but I did not succeed. I attached code where I attempted to solve this about three weeks ago, attached in `code/matrixMult`. Unfortunately it only yields the correct results when the size of the matrix is a multiple of the block size. This can be run by invoking `make run` and running the generated binary.