DV2520 — Assignment 1 — Matrix Mult

Deadline	2011-09-21 09:00
Submission	tarball including source, Makefile and report (in pdf) uploaded to It's Learning

1 Description

You will implement matrix multiplication in OpenCL. The only host transfers allowed are once at the beginning to copy initial matrices into the device memory, and once at the end to retrieve results. Your code must handle arbitrary sized matrices, although you can assume that all matrices used will be square.

For the purposes of timing your code you will calculating powers of matrices. You should not use anything sophisticated for calculating the power (such as a square-multiply algorithm) as you will be calculating the performance of the multiply routine, not the wrapper around it.

1.1 Terminology

You will use two separate transformations of the kernel in your profiling experiments.

Splitting will convert the execution of a single kernel into a set of kernel executions. This corrosponds to splitting part of the problem into smaller pieces. The result calculated by each thread in the original unsplit version will be calculated by threads in several kernels. You will need a way to combine the set of results into the single result, taking synchronisation into account.

Tiling will merge several threads within a single kernel into thread. This reduces the parallelism, but increases the CGMA ratio. When you convert a kernel into x by y sized tiles, a set of xy threads in the original will be replaced by a single thread that outputs $x \times y$ results.

2 Grading

Pass - 3 Write a program that caclulates matrix multiplication in OpenCL. Your program should accept the size of the matrix and the exponent as parameters (either on the command line or as preprocessor defines) and should compute M^e as $M \cdot M \cdot \ldots M$. The result should be verified against a CPU implementation and the time per multiplication should be output to the console.

Pass - 4 Implement a simple splitting of the multiplication kernel, that is, split the inner product in every cell $(r, c) : \sum_{i=0..N} x_{r,i} \cdot y_{i,c}$ into T separate pieces and accumulate the T partial results into the solution. Your kernels will need to be scheduled using wait events to guarantee they are executed in the correct order. As part of your submission generate a graph of the tiling performance for a particular matrix size (ensure the graph is in `.pdf` format).

Pass - 5 Implement a tiling of the multiplication across two dimensions. As well as splitting the sum of partial products into pieces, you will combine the evaluation of several cells into a single kernel (i.e computing a square of (r, c) elements inside each kernel). Evaluate the tiling across a range of parameters and find the optimal splitting and tiling for $N = 256$ and $N = 512$.

3 Submission

You must submit your source-code in a tarball through It's Learning. For a grade 4 include a (maximum) one-page report that includes a graph of the performance vs matrix size. For a grade 5 include a (maximum) two-page report that include the single graph above and further graphs to show the trade-off between the tiling and splitting transformations.

In your reports include descriptions of:

- How you selected parameterisation within the ranges for tests.
- How you performed the measurements.
- How you would quantify your confidence in your result.

4 Notes

Do not try and implement a sophisticated matrix multiplication algorithm: you are being evaluated on how you can apply tiling as a technique for improving performance. The simple naive matrix mutliplication with $O(n^3)$ is the one you should implement. This is the straight-forward way to calculate:

$$z_{r,c} = \sum_{i}^{N} x_{r,i} \cdot y_{i,c}$$

For timing use the RDTSC macros that are supplied in class.

For graphing you may use whichever software that you want (as long as the graphs are readable and in `.pdf` format). However, even if you have not used it before installing and using `gnuplot` will save you a lot of time and effort.