Exercise 04 - Computation of a variance-covariance matrix

In this exercise, we are to utilise what we have learned so far. The task will be to compute in parallel a variance-covariance matrix utilising the power of a many-core GPU.

Assume that we have a table of raw historical data

$$X = [X_1 \quad X_2 \quad \cdots \quad X_n]$$

capturing the historical values in terms of N scores X_i for each of n assets.

The variability or spred in the data can be measured by the variance computed as

$$Var(X) = \frac{1}{N}\sum_{i=1}^{N}(X_i - \bar{X})^2 = \frac{1}{N}\sum_{i=1}^{N}x_i^2$$

where the mean value is computed as

$$\bar{X} = \frac{1}{N}\sum_{i=1}^{N}X_i$$

and $x_i = X_i - \bar{X}$ represents the deviation from the mean for the i'th record.

The covariance measures to which extend corresponding elements from two sets of ordered data move in the same direction and can be computed using

$$Cov(X,Y) = \frac{1}{N}\sum_{i=1}^{N}(X_i - \bar{X})(Y_i - \bar{Y}) = \frac{1}{N}\sum_{i=1}^{N}x_i y_i$$

The variance-covariance matrix is defined as

$$C = \begin{bmatrix} Var(X_1) & Cov(X_1,X_2) & \cdots & Cov(X_1,X_N) \\ Cov(X_2,X_1) & Var(X_2,X_2) & \cdots & Cov(X_2,X_N) \\ \vdots & \vdots & \vdots & \vdots \\ Cov(X_N,X_1) & Cov(X_N,X_2) & \cdots & Var(X_N,X_N) \end{bmatrix}$$

Concepts covered

The following concepts are covered in this exercise

- How to write GPU kernels.
- How to use shared memory to improve naive implementations.
- Performance profiling of incremental improvements to kernel code.
- How different reduction schemes can affect branch divergence and bank conflicts.
- How to optimize by unrolling loops and using that threads in a warp execute in full synchronization.

Files needed

The following files will be needed for the exercise

- `VarCovar.cu` (needs to be updated)
- `VarCovar.h` (requires no updating)
- `VarCovar_kernel.cu` (needs to be updated)

The code will not pass until all work steps described below has been successfully completed.

Work steps

You will need to modify the C code supplied in the files. You will copy-paste the CPU and naive kernel from the Lab03_Sum exercise.

The device code in `VarCovar_kernel.cu` needs to be updated through the steps

1. Copy-paste the CPU and naive kernel from the Lab03_Sum exercise (`CoVar_gold()` and `CoVar_kernel_v1()`).
2. Produce a naive kernel (i.e. one with no optimizations of performance) that calculates a variance-covariance for a data set that is loaded from a file.
3. Utilise the learnings from the past exercises to produce a faster kernel and benchmark the kernel for each technique you try (e.g. use shared memory, memory coalescing, etc.).
4. Experiment with the number of `threadsPerBlock` and `blocksPerGrid`.

If time permits, try to use a library to perform a PCA (e.g. see tutorial with basic example here `http://www.cs.otago.ac.nz/cosc453/student_tutorials/principal_components.pdf`) or SVD decomposition of the Variance-Covariance matrix and measure the performance.

6. Test the performance of a PCA or SVD decomposition into eigenvalue and eigenvectors of a Variance-Covariance matrix using a library (to involved to write such routines yourself in short times).

Compilation

Compile the final code using the included `Makefile` which includes the common parameters for compilation in `../../common.mk`. To compile successfully on your laptops you probably need to modify `common.mk`, where it is assumed that CUDA has been installed in `/usr/local/cuda`. In case of compilation errors you will need to debug the code until it compiles successfully.

Execution

Execute your compiled program. If you program executes successfully the output should look like this

```
Computation of a Variance-Covariance matrix.
  Usage: ./VarDovar <N:default=1000000> <threadsPerBlock:default=64> <blocksPerGrid:default=64>
   <reps:default=10> <file:default=data.txt>

Device 0: "Tesla C2050".
  Maximum number of threads per block: 1024.

Threads per block = 64.
Number of blocks [gridDim.x] = 64.
Total number of threads = 4096.
N = 1000000.

  CPU time            : ? (ms)
  CPU rel. error      : ?

  GPU time compute : ? (ms)
  GPU time transfer: ? (ms)
  GPU time total   : ? (ms) , speedup ?x
  CPU rel. error      : ?
  PASSED
```