High Performance Computing

Assignment 1

The goal of this assignment is to understand how memory access patterns affect performance, using matrix multiplication as an example. We won't worry about parallelism yet: your program will run on a single core. In later assignments, we will add different types of parallelism to the matrix multiplication example.

Given a matrix A of dimensions $m \times q$ and a matrix B of dimensions $q \times n$, the product $C = AB$ has dimensions $m \times n$ with the entries defined as

$$C_{ij} = \sum_{k=1}^{q} A_{ik} B_{kj}$$

(a) **10 marks.** Write a program called `matmult` that can be run using command line parameters as follows:

```
matmult m q n filenameA filenameB filenameC
```

where `filenameA` and `filenameB` are the names of two matrix input files, and `filenameC` is the name of a matrix output file to be created. The positive integers `m`, `q`, and `n` specify the dimensions of the input matrices. The format of the matrix files is a list of all the entries of the matrix (see the test examples below), in column-major order. Your program should compute the product of the two input matrices and write the result to the output file. Use double precision for all your matrices and computations.

Your code should also be able to operate by calling it with no files, i.e.,

```
matmult m q n
```

In this case, your code should generate random entries for the two input matrices, and then perform the multiplication. This is useful when you are only interested in the performance of matrix multiplication for large matrices and you don't want to bother with reading and writing files.

Check that your code is correct by testing it with the files `matrixA`, `matrixB`, and `matrixC`, corresponding to $AB = C$. For this test data, $m = 3$, $q = 4$, $n = 5$. A set of larger test matrices and a test matrix generator are also provided.

(b) **5 marks.** Once you know that your code is correct, you can start investigating its performance. Measure the execution time of matrix multiplication (not including reading and writing files) for different sizes of matrices. For simplicity, choose $m = k = n$ and $n = 500, 1000, 2000, \ldots$ to as large a problem your computer can handle. Plot the timings and the Gflop rate as a function of n. This is your *baseline* performance.

(c) **10 marks.** Try to improve the performance of matrix multiplication by storing matrix A in row-major order. This involves transposing the data structure for A. Separately report the timings for transposition and matrix multiplication. For matrix multiplication, again plot your timings and Gflop rates like the above.

(d) **10 marks.** Try to improve the performance of matrix multiplication by using "two-dimensional blocking." (Sections 3.5.2 and 3.5.3 of the text should be helpful.) Try different values of the block size and plot your results in a meaingful way.

(e) **5 marks.** Submit a short report of your results as a pdf file. Marks will be given for the clarity of your report, labeling your graph axes, etc. Briefly explain why you think your results are reasonable. Include a listing of your code in the pdf file. (This makes sure we are reading your code in the way you intended.) Also specify the peak Gflop rate and the single threaded stream bandwidth of the computer used in your performance tests. These numbers should help in your explanation of your results (e.g., is the transposition step bandwidth bound?).