Code: https://github.com/hvl5451/CSE-557-MMM

Question 1:

The performance evaluation was done in the following environment:

MacBook Pro with a Dual-Core Intel i-5 processor with a processor speed of 2.3Ghz. Hyperthreading was enabled. Total hardware threads available: 2 + 2 = 4

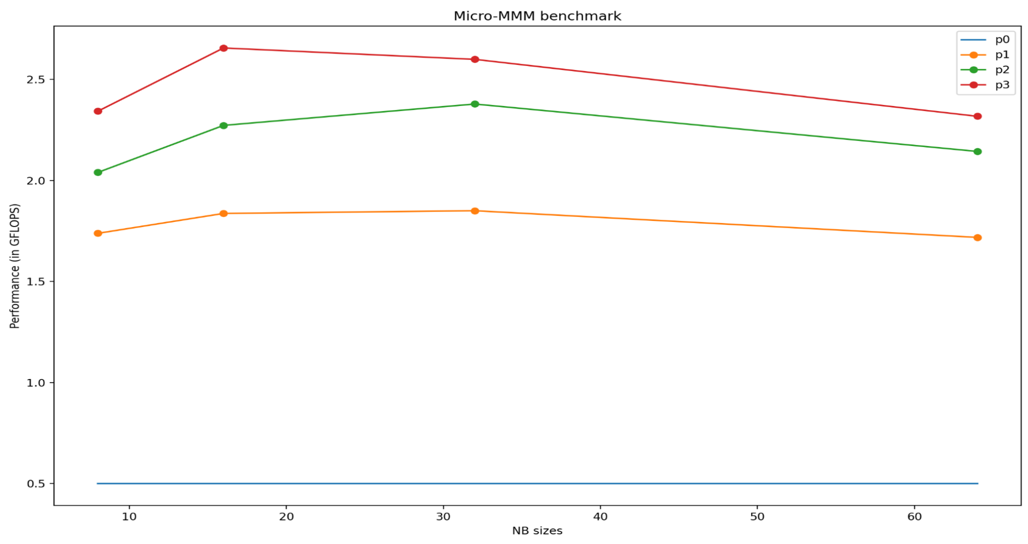
OS: macOS 11.5.2, Darwin 20.6.0

Compiler version: gcc-11.2.0

Compiler flags used: `-Ofast -march=native`

L2 cache size: 256KB; sqrt(C) = 126.49

All the four subroutines (as explained in the problem) were implemented and the best block size was found using parameter-based performance tuning. Performance (in Gflop/s) was measured for all the 2 powers of NB between 16 and 80. (i.e. NB = [16, 32, 64]) for all the 4 subroutines. The matrix of dimension 2048 was used in this experiment.



The best performance is achieved by Program 4 (Register + cache level blocking) with a value of block size as 16. This is because of the register level and cache level blocking implemented in Program 4 along with scalar replacement.

It's also interesting to note that block size 32 is the optimal block size for Program 1 and Program 2.

Program 4 Performance: 2.655 GFLOPS

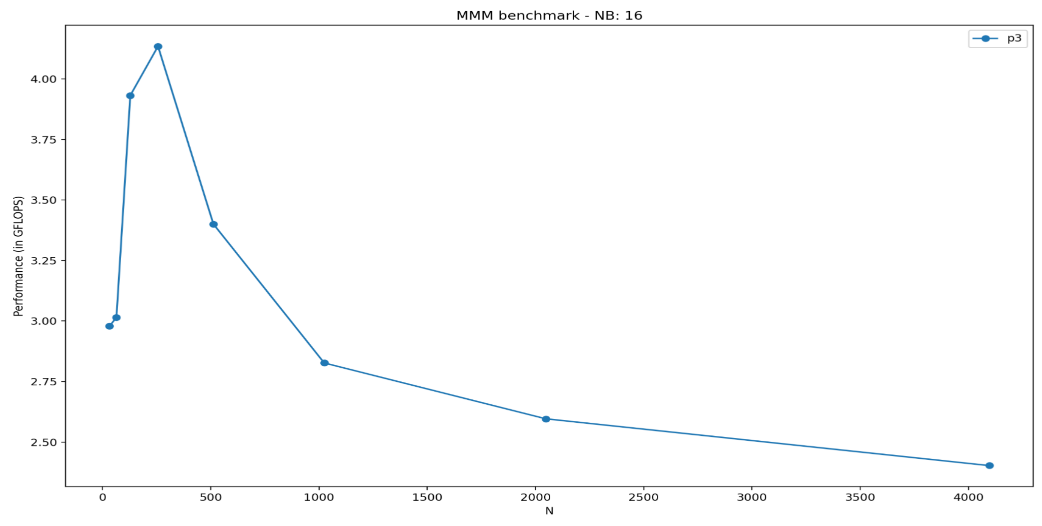
Theoretical Peak Performance: 73.6 GFLOPS

Percentage of Peak Performance: 3.6%

The program doesn't improve when the order of the outer loop is switched from ijk to jik. The performance is similar or degrades in some of the runs.

Question 2

The environment specified above was used in this experiment. Program 3 subroutine and block size of 16 was used to profile the matrix multiplication. Matrix sizes from N=32 to N=4096, in increasing powers of 2 were analyzed. The performance was measured in Glop/s with operation counts as 2N^3.



It’s interesting to note that the performance of MMM peaks when the dimension size is 256 and decreases after that. This is caused due to the fact that the majority of the 3 matrices can be entirely fit into the L2 cache and the blocks being aligned.

Decreasing performance with increasing input size can be reasoned due to cache misses as the number of blocks increases.