Zhaohua Gao cs61c-iz
Pei Jun Chen cs61c-ja

Proj3 Part 2 Write-up

A brief description of any changes you made to your code from part 1 in order to get it to run well on a range of matrix sizes (max 150 words)

In the cache blocking strategy, we tried several ways to do the cache blocking matrix multiplication.

The first way we tried was to use a fixed block size of 64x64. We made this as the first because we believed that this way should be the slowest among the 3 ways we were going to used. We believed it should give the worst performance because in the worst case, it remains 63 rows/columns which cannot be block. Obviously, it is the most time consuming to solve the fringes of matrices.

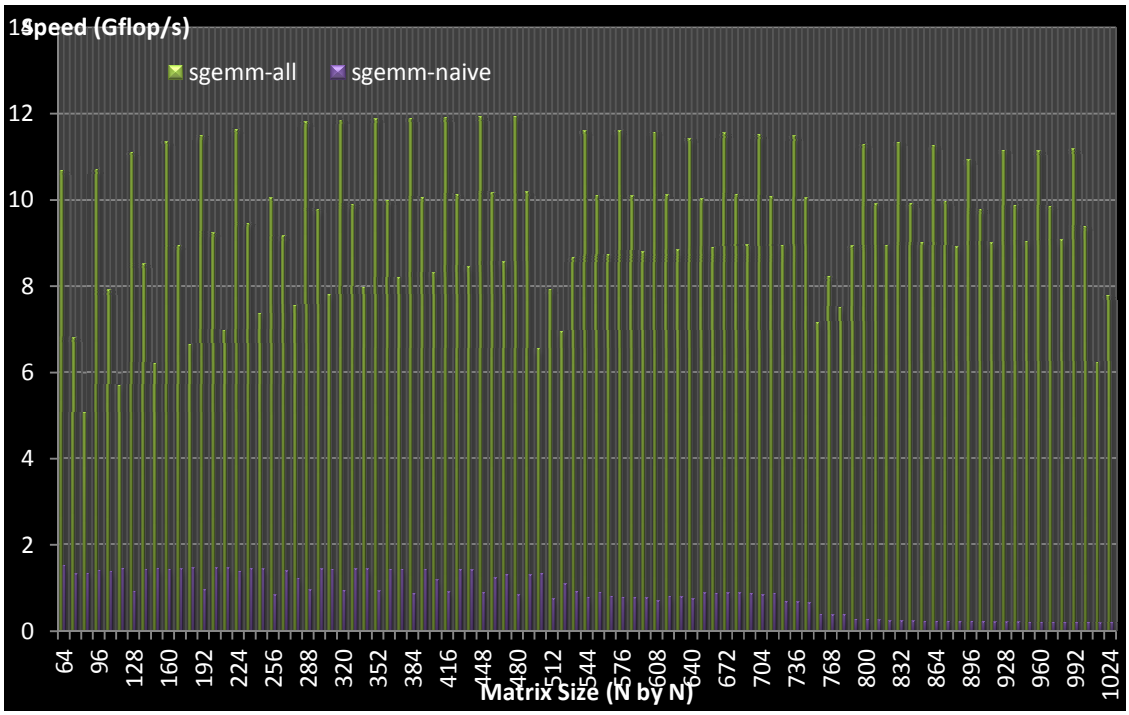
The second way was to reduce the block size to 32x32. Although it requires more calculations on the block multiplication, however, the time requires to solve the fringes will be significantly reduced due to the size of the remained rows/columns are only half of those from block size equal to 64.

After repeating all different block sizes, we surprisingly observed that they are almost at the same speed. The 32x32 block size ran slightly faster than others on average.

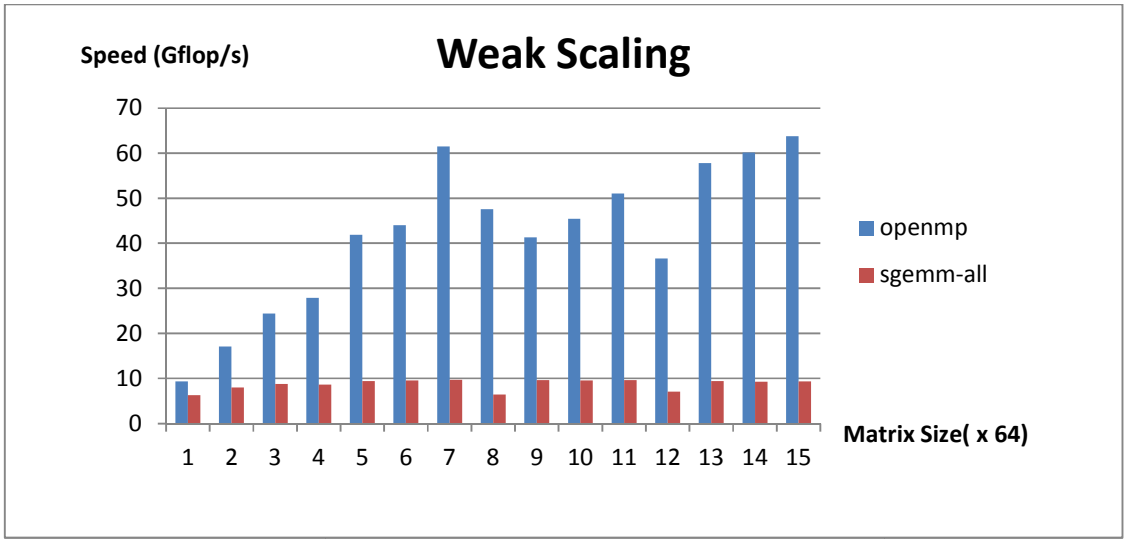
A brief description of how you used OpenMP pragmas to parallelize your code in sgemm-openmp.c (max 150 words).

In parallelization strategy, we put parallel computing on block calculations (calculate different blocks simultaneously). We tested different number of threads, 4, 8, 16 and 32. It results the one with 8 threads gives the fastest speed.

A plot showing the speedup of sgemm-all.c over sgemm-naive.c for values of N between 64 and 1024



A weak scaling plot of the performance of your sgemm-openmp.c code (use your sgemm-all.c code as the baseline for the single threaded case)



A strong scaling plot of the performance of your sgemm-openmp.c code