Name : Zhaohua Gao   Pei Jun Chen
Login: cs61c-iz   cs61c-ja

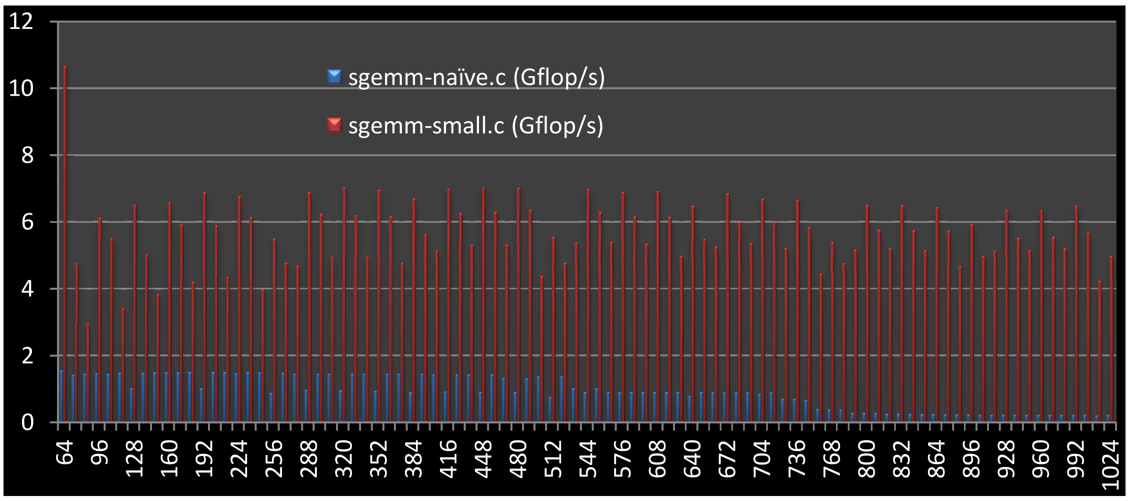
Describe how you use SSE registers in the innermost loop of your code

In 64x64 optimization, we store the entire column of matrix B into 16 xmm registers because each float data type is 32 bits, one __m128 register can hold 4 floats, and 16 registers are just the exact size to store 64 floats which is a column of data in 64x64 matrix. In the innermost loop, we did add, mul, loadu in a single instruction. At this point, we have some choices to do the calculation for the result. We can either keep repeating using one register to sum the result, or we can use several registers to do the calculation in different segments and sum them up at the end. After tried using 1, 2, 4, 8 registers, we found that 2 registers gave the fastest speed. We guess the slower speed on more registers was because of the time consuming required for storing data to memory before each register was being used.

Describe how you deal with the fringes of matrices when N isn’t evenly divisible by the SSE register width (max 100 words)

For total 16 xmm registers, the worst case would be 63 values left (N%64 = 63) if we use the data block size =64. We need to use a for loop to do the rest 63 values' calculation. However, we chose the block size to be 32 because we thought that the chance to have <=31 values left is greater than the chance to have <=63 values left. After some tests, we found that the speed running on block size = 64 makes no big difference with that running on block size = 32. The big speed difference occurs on handling the rest of the values because they are not using SSE instructions. Therefore, we chose block size = 32 to balance the values handled by SSE instructions and those by non SSE instructions.

Include a plot showing the performance of your code as compared to the code in sgemm-naive.c



How many XMM registers does your code use?

In N= 64 optimization, all 16 xmm registers were used.

In other matrices calculations, 6 xmm registers were used.

Are any values being spilled to the stack during the innermost loop?

In N= 64 optimization, found 3 %rbp pointers which means three values were spilled.

In other matrices calculations, found only 1 %rbp.

How many scalar floating point instructions does your code use and why?

There were 8 scalar floating point instructions in our code used. There were all used in the instruction _mm_store_ss which severely increases the speed of around 1.1 Gflop/s.