CS 5220: Project 1 Initial Report

Group 018: Guantian Zheng (gz94), Stephen McDowell (sjm324)

1 Introduction

Our initial findings for this assignment are somewhat discouraging. We initially had a bit of trouble figuring out how all of the parts of this assignment fit together. At this time, we feel that we have a good understanding of how we are supposed to be performing the tasks expected of us, but not much performance has been achieved at this point.

1. The first stage of diagnostics was to begin examining how the different permutations of `i`, `j`, and `k` can have an impact on the speed of the basic matrix multiply.
2. The next natural piece to examine *at a high level* was comparing how `icc` and `gcc` compare with one another. During this phase we used the basic permutations to help us understand which was superior, but have not gone as far as changing the compilation flags – yet!
3. We then began examining how these permutations and best-suited compiler can affect the blocked strategy provided in the initial framework, in conjunction with adjusting the `BLOCK_SIZE` variable to see its effect on the blocking efficiency.
4. Where we are currently is trying to query the hardware to programmatically determine the optimal `BLOCK_SIZE`, as well as toying with the idea of transposing the `A` matrix in memory.

Part of the reason why we have not made as much progress as we would have liked to is because we only have two group members. This was not really made apparent to us until tonight.

2 Permutations of Loop Variables

We experimented with different permutations of the three loop variables `i`, `j` and `k`. Apparently, `j, k, i` (starting from the outermost loop) gains the most advantage by reading in continuous blocks of matrix `C` and `A`. The loops are as follows:

```
for(j = 0; j < M; ++j) {
    for(k = 0; k < M; ++k) {
        double b_kj = B(k, j);
        for(i = 0; i < M; ++i) {
            C(i, j) += A(i, k) * b_kj;
        }
    }
}
```

3 Tuning the Block Size

Building on previous results, we decided to adopt the `j-k-i` loop for per-block computation (`basic_dgemm`), while searching for an appropriate block size.

Looks like reordered `blocked` with block size 256 takes the lead. Let's try with larger sizes:

The Gflops of `blocked_perm` keeps rising until size 2048, which means `j-k-i` looping blocked implementation with size 1024 is by far the optimal solution for our test cases.