CS 5220 – Peer review – Group 24

Junteng Jia – jj585
Eric Gao – emg222

Common

There are a few optimizations we did in common:

1. **Block Multiplication**: It is a common technique everyone used.They split the matrices into 128 * 128 blocks while we split the matrices into several blocks. We chose the blocks to be size 32, which means that each block takes up 8KB ($32^2 * 8$).
2. **Copy Optimization**: Their code allocate a piece of memory every time they call the function "DO_BLOCK" to make sure the piece of memory in the block is continuous. While we copied and rearrange matrix A, B and C at the begining once, so we don't need to allocate a piece of memory every time we call "DO_BLOCK" function, we just pass the pointer pointing at the begining of that block.
3. **Rearrange i, j, k**: They did a few experiemnt to decide which order to use with i, j, k, while we analysis base on computer architecture without much test.

Worth Learning

Group 24 did some optimizations we need to implement:

1. **Parameter Optimization**: They tested against all possible permutations of orders. They tested against different block sizes, which can help them find the best code suitable for the arhcitecture of totient clusters.
2. **Compiler Optimization**: The used the icc flag `+OPTFLAGS = -O3 -fast -xCORE-AVX2 -unroll-aggressive -fp-model fast`, which would probably help us speed up our own code.

Suggestion

I do have some ideas that our group used and I believe would help them speed up their code:

1. **Data Alignment**: They can try to use `_mm_malloc(size, alignment)` instead of `malloc(size)` to allocate a piece of memory in heap that is convinent for cache to get.

2. **Vectorization**: Although they had went through all possible permutations of i, j, k to get the best performance, there are a certain point where they can not get Vectorization due to data reliance. For example:

```
1  void basic_dgemm_copied(const int M, const int N, const int K,
       const double *restrict A, const double *restrict B, double *
       restrict C)
2  {
3    int i, j, k;
4    for (k = 0; k < K; ++k) {
5      for (j = 0; j < N; ++j) {
6        for (i = 0; i < M; ++i) {
7          double cij = C[j*M+i];
8          cij += A[k*M+i] * B[j*K+k];
9          C[j*M+i] = cij;
10       }
11     }
12   }
13 }
```

   In line 8, this operation can not be vectorized because updating cij relies on previous cij. This problem can be fixed by updating several element in C matrix at the same time.

Others

There are a few things that I believe could be implemented that neither of our group has done:

1. **Further Blocking**: I believe we can use blocking in different level to suit into different level of cache.
2. **Roofline Analysis**: We can use Roofline analysis to figure out what should we do next.