Understanding Software Dynamics Ch 4 Lab Report

This is difficult. I barely understand the given code.

4.0) Run matrix.cc

```
ubuntu@ubuntu2:~/Documents/github/KUtrace-experiments$ ./matrix_ku
a  sum= 1598331.289600000
aa sum= 1598331.289599999
c  sum= 1598331.289600000
Equal
Remap Misses L1/L2/L3          0          0          0
b  sum= 1598331.289600000
bb sum= 1598331.289600001
c  sum= 1598331.289600000
Equal
Transpose Misses L1/L2/L3          0          0          0
b  sum= 1598331.289600000
bb sum= 1598331.289600001
c  sum= 1598331.289600000
Equal
BlockTranspose Misses L1/L2/L3          0          0          0
SimpleMultiply                  3.259 seconds, sum=2494884076.030955315
Misses L1/L2/L3          0          0          0
SimpleMultiplyColumnwise        4.451 seconds, sum=2494884076.030955315
Misses L1/L2/L3          0          0          0
SimpleMultiplyTranspose         1.073 seconds, sum=2494884076.030955315
Misses L1/L2/L3          0          0          0
SimpleMultiplyTransposeFast     0.480 seconds, sum=2494884076.030954838
Misses L1/L2/L3          0          0          0
BlockMultiplyRemap              0.351 seconds, sum=2494884076.030955315
Misses L1/L2/L3          0          0          0
IGNORE SimpleMultiplyOne        0.254 seconds, sum=    1024.003072004
Misses L1/L2/L3          0          0          0
```

4.1) matrix_remap_openmp.cc
I use the OpenMP API because it is convenient. Alternatively, I may consider parallelizing myself in the future.

```
339   #pragma omp parallel for collapse(2) schedule(dynamic)
```

Explain: for parallelizing nested loops and using dynamic scheduling to control how loop iterations are divided by threads.

I extract the code about Faster Matrix Multiply – Subblock method from matrix.cc and create matrix_ku.cc. Using the OpenMP method is 5 times faster than the original one.