# ECE/CS 466 Project 2

We have discussed the blocking technique to improve the performance of dense matrix multiplication in the class. In this part of project assignment, you are asked to implement the original and blocking codes of dense matrix multiplication, run the following experiments on your computer (real machine, not SimpleScalar) and report the results. Assume the array size is N x N.

1. Implement the original code of matrix multiplication and set the value of N to 1024. Compile the code using optimization option "-O0" (no compiler optimization) and "-O3" (standard compiler optimization) with gcc, respectively. What is the speedup using "-O3" compared with using "-O0"? If you are using another compiler other than gcc, use two different optimization options supported by the compiler to run the experiments, one without any compiler optimization and another with standard compiler optimization.

2. In the experiments thereafter, using "-O3" option (or standard compiler optimization with compiler other than gcc) when compiling your code. Implement the original code of matrix multiplication, vary the array size from N = 16 to 4096 (doubling the value of N each time). When does the program execution time increase sharply? What's the relationship between the cache size of your computer and the array size around the jump point?

3. Implement the blocking method, set the blocking factor B to 8, and vary the array size as above. Compare the execution time of matrix multiplication with and without blocking. Based on the results, what's your suggestion on applying the blocking method?

4. Use the blocking method, set the array size N to 2048, and vary the blocking factor B from 4 to 512 (in power of two only). What is the optimal blocking factor for your program and what is your suggestion on choosing the blocking factor?

5. What did you learn from this project?

Note:
(1) You can use timing function gettimeofday() in C before and after the matrix multiplication to get the accurate execution time.
(2) You can initialize array elements with fixed value for a small value of N (e.g. 16) and compare the outcome with and without blocking implementations to verify your code.
(3) The expected execution time will increase with N doubled even without the cache miss effect.
(4) When the array size is large, allocating the array as static one may cause segmentation error. You can use malloc() to allocate array elements in stead.