I. Implementation:

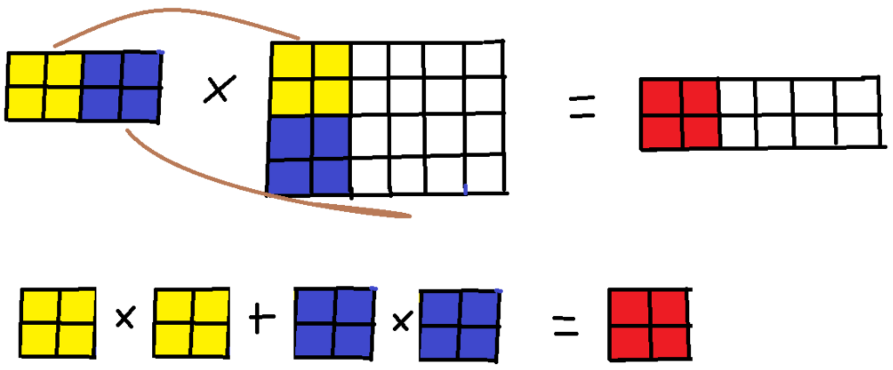
A. How do you implement the matrix multiplication (both version)?

- Standard_matrix_multiplacation:跟用手算的方法一樣，A 矩陣乘以 B 矩陣=C 矩陣，A 的列數去乘以 B 的行數則對應到 C 矩陣的(列,行)座標，而列數乘以行數是每一列的數字對應到每一行的元素來相乘。
- Blocked_matrix_multiplication:之前的線性代數有教過，運用矩陣分割，6 個 FOR 迴圈簡單來說就是外層 3 個 FOR 迴圈是一個矩陣乘法，內層 3 個 FOR 也是一個矩陣乘法，也就是矩陣中的矩陣乘法，至於 BLOCK 就是由看他的 BLOCKSIZE 來切 A、B、C 矩陣，整體來說就是做兩次矩陣乘法。

B. How do you verify the correctness of your program?

- 除了用網路上做好的網站做好二次驗證之外，將矩陣的資料減少到 10x10 內，多次測試各種資料，並將每個 C(i,j)的列乘以行的算式都 print 出來做確認，如果列跟行的數值都確認是對應上的數字，那算出來的結果便不會錯誤。

C. Why the blocked version is correct? Answer by explaining the mathematics or program's calculation steps.



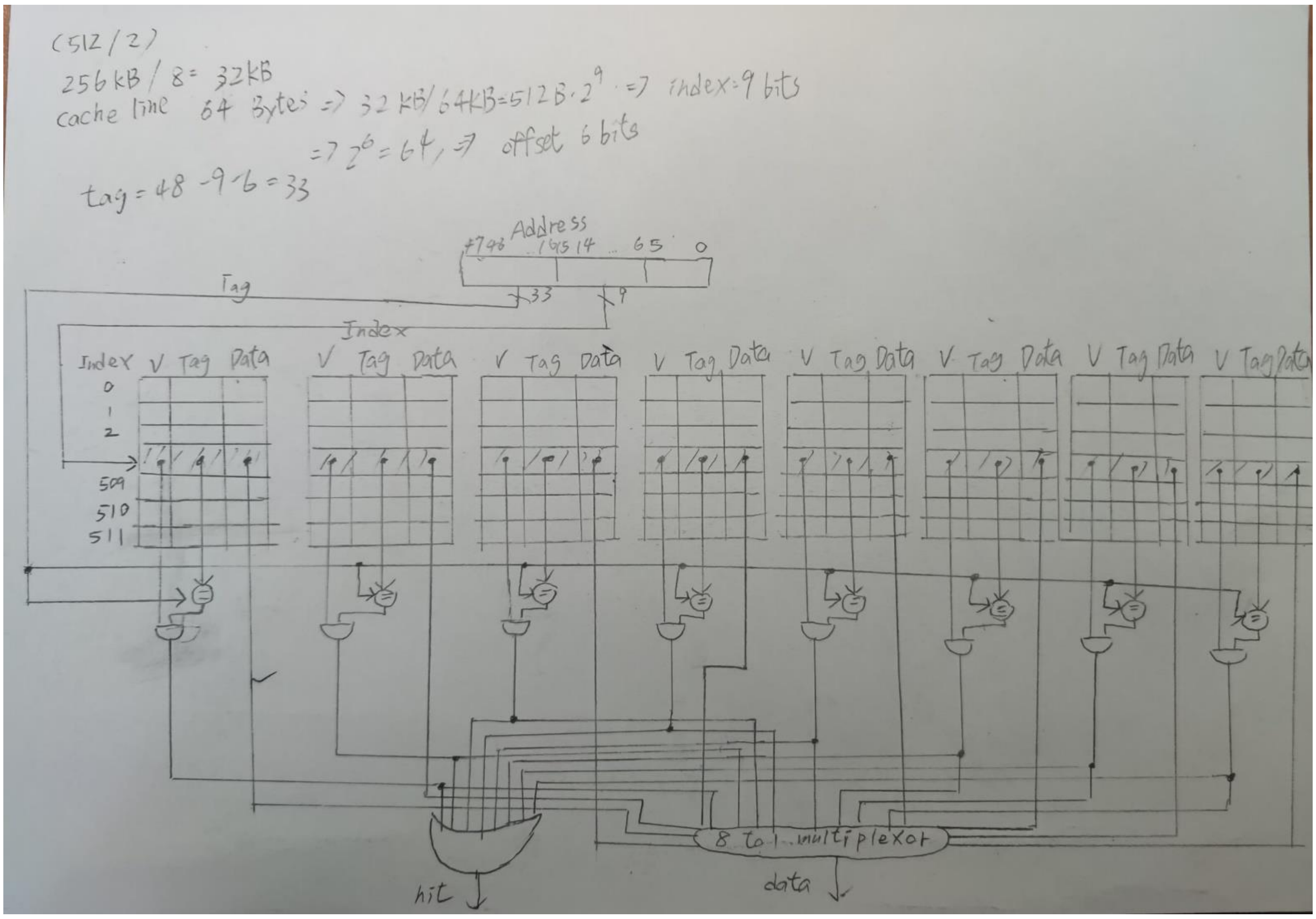
➢ 以題目給的範例來說，就是先把它變成 A 矩陣 2X4=>1X2 B 矩陣 4X6=>2X3 C 矩陣 2X6 的變成 1X3，矩陣內的再做矩陣乘法，因此只要確認 block 內的數值都是對的，剩下來的 block 都是對的。

II. Experiment:

Hardware		Intel(R)Core(TM)i7-1065G7 CPU@1.30GHZ
		SODIMM 2667MHz
		L1:320KB L2:2MB L3:8MB
Software		Embarcadero Dev-C++ 6.3
		g++ -c 檔名.cpp
		Window 10

A. Given that the size of your CPU's data cache is half of the L1 cache; 48 bits address in total; 8-way associative; and 64 bytes per cache line, draw the data cache address design based on Figure 5.18.. You should specify and explain how many bits the byte offset, index and tag takes. Note that Figure 5.18. has the cache line size of 32 bits (4 bytes), so that the number of byte offset bits is 2. If your L1 cache is not a power of 2, take the smallest number that is greater than your cache with a power of two. For instance, the cache you draw in your report will be 256 KiB if the actual data cache size is 160 KiB.

➢



B. Let's restrict n, m, k to the same value. Conduct at least 5 experiments based on different input size for your both versions of program. Explain how you produce the test data and what is the input size of those test data. The input sizes that you select should produce significantly different GFOPS values, as explained in the next paragraph.

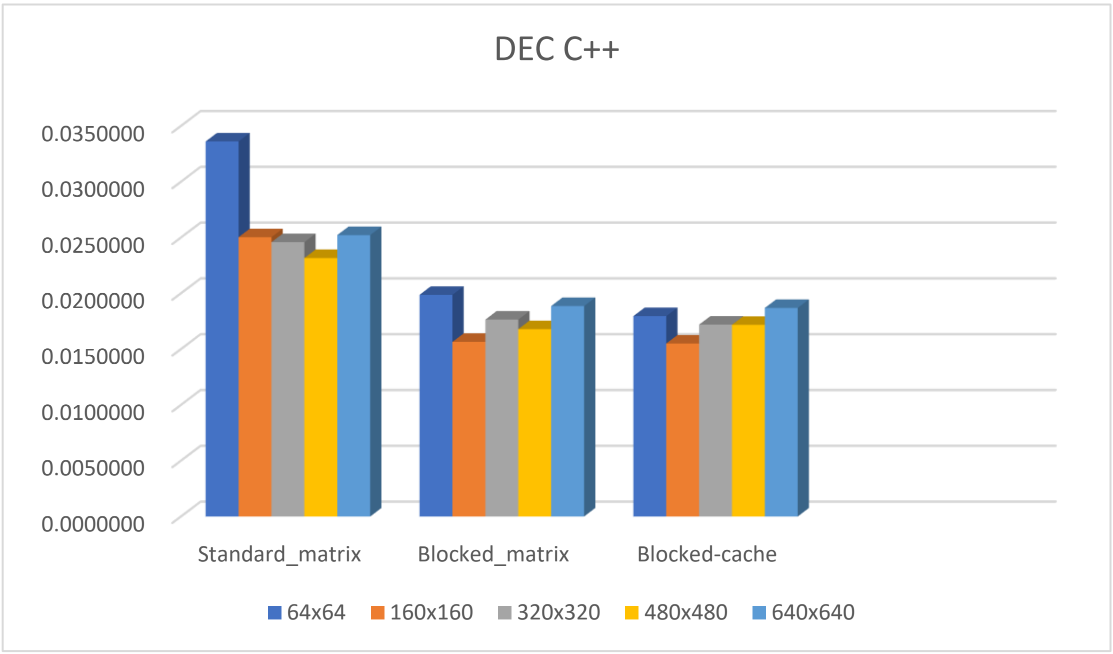
➢ 1. Test data 是由使用者輸入 n、m、k。

2. MAX=1000，MIN=-1000，用 RAND()產生再 MIN~MAX 的數字，生成 nxm 個數據和 mxk 的數據。

3. 將 n、m、k 和 nxm 個數據及 mxk 的數據輸出到 input.txt。

4. 方便各個矩陣乘法讀取及做運算。

C. Measure the performance of your code by the 5 test cases you have created, and draw the result figure similar to Figure 5.23 (Note. This figure appears in the fifth edition of the textbook and in our class handout, but not in the sixth edition of the textbook). You should measure the performance as gigaFLOPS(GFLOPS). GFLOPS of this problem can be calculated by n·k·m / execution time (ns.) . Explain why can we measure GFLOPS in such calculation.



- 從程式碼來判斷，兩個矩陣 A 矩陣 nxk 和 B 矩陣 kxm 兩個矩陣，在 standard_matrix_multiplacation 程式碼中，先看外層 2 個 for 迴圈我們在每一列(n)x 每一行(m)矩陣乘法中需要內層 for 迴圈中的 m 次加法運算，因此最後的運算量為 n*k*m 次，而因為程式碼第 38 行 sum = sum +(A[i][l] * B[l][j])，有一次加法，所以總的來說應該是 2*n*k*m 而再除以時間為 2*n*k*m /execution time (ns)，「*2」可以忽略，總結為 n*k*m /execution time (ns)。

D. Explain how you measure the execution time of the program. Is it appropriate to measure the total time of this program?

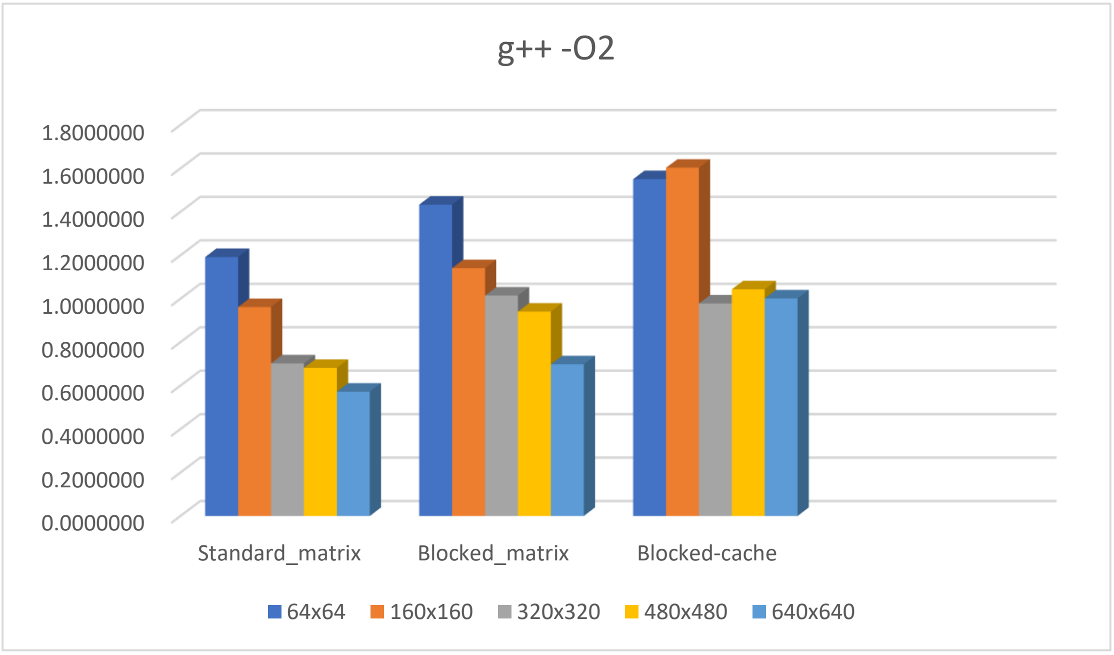
- Include<chrono>，用來紀錄時間，提供高精準的時間日期庫，使用 steady_clock 無法去更改時鐘的數值，只能讓它以不變的比率增加，之所以不用 system_clock 是因為使用者可能改了系統事件，時間就可能會受到影響在。乘法開始之前存到變數 now 和實驗結束存到 end 變數，(end-now)這時候用 duration_cast 來進行單位的顯示轉換。

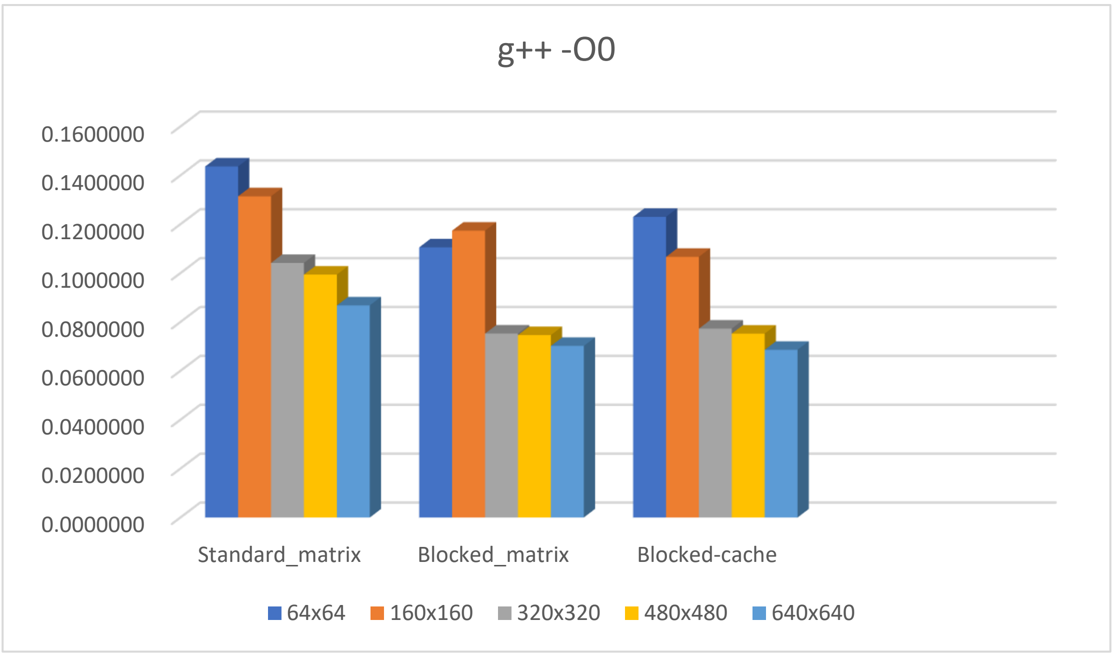
E. Based on the result, explain the differences between the standard version and the blocked version. In addition, why different input size will have different performance in both versions?

- 計算的效率相當重要，在 stand_matrix 中，需要用到三個 for 迴圈，在一列乘以一行當中，一行(column-major)的部分會因為要讀取這些元素會從 cache 頻繁進出，因而拖慢計算效率，導致時間較久，而 blocked-matrix 中則是以 blocksize 大小為單位來做運算，在不同的 input size 當中，重要的是如何把 cache 傳到 main memory 的速度提升，也因為如此，blocked-version 的 performance 才會比較高
- 我寫的時候遇到一個問題，當我 input size 設為 480 以

上時，用 array 宣告的會回傳亂數，但當我用同樣程式碼在同學更好的電腦設備當中卻可以跑出來，我猜測在面對大型矩陣的乘法當中，矩陣越大越難放進 cache 當中，因此讓我 cache 裝滿而放不下去，因此我的解決辦法是改用 vector 來做，因為在 c++裡，它是動態宣告陣列，不需要用到資料的就會刪掉。

F. Conduct the experiments (b) again, but using the compile parameters g++ -O2 matrix.cpp -o matrix2.exe and g++ -O0 matrix.cpp -o matrix0.exe. (Assume your program name is matrix.cpp). Plot the result similar to the result figure in (c) and explain the differences between the flag -O2 and -O0.





➢ Flag 02 跟 00(沒有優化)相比 ，數字越高代表最佳化的程度越多，而 03 最為優化級別當是是最高的，因此 02 自然比 00 最佳化程度還高，很多 Project 在 Compile 的時候都會又-o2 來進行最佳化

III. Advanced questions

A. Cache line optimization: Conventionally, array data are stored in the main memory in a row-major fashion. Data in a row are stored consecutively, and therefore they have the property of spacial locality in the cache block. As a result, accessing the array data in the row-major fasion has higher cache hit rate than accessing the array data in the column-major fasion. Modify the code so that the B matrix is read in row major (transpose it to BT), and compare the performance to the original code. Explain the differences.

➢ 雖然我們宣告陣列的時候是用二維的方式，但讀取 data 的時候，我們仍用一維去讀，一般的算法是列乘以行，那個「行」讀進去一直換會很花費時間，所以我們將 B 矩陣做轉置，變成兩個都是 row-major，等於是 row x row，

而 row-major cache hit 的機會又比 column-major 高，

進而就減少了 cache miss 後還要再去找的時間。