Performance analysis of estOLS application

Saksham Phul

Computing ordinary least square is crucial to model a single response variable which has been recorded on at least an interval scale. The equation is as follows:

$$\hat{\beta} = \underbrace{(X^T X)^{-1}}_{A} \underbrace{X^T Y}_{B} \quad \text{……. Eq. 1}$$

Where X is a n x n matrix, Y is a m x 1 vector Given n >> m and m > 500.

For simplicity, we break this problem of computation into 2 parts, Matrix A of size m x m and vector B of size m x 1 as shown in Eq. 1. Based on the performance, it was found that this application spent more than 95% of time in computing matrix A for n = 50,000 and m>500. Therefore, this report ignores the analysis on matrix B and focuses on the major floating-point operations and memory latency required to compute matrix A. The matrix A can be further broken down into 2 parts: matrix multiplication (X^TX) and inverse of a resultant matrix $(X^TX)^{-1}$. Due to this dependency, it is difficult to compute multiplication and inverse concurrently. Thus, we took serial approach using GNU scientific library to build this application.

Calculating matrix A is simple and inexpensive for small numbers of n and m. However, for large matrices, this computation grows exponentially due to large number of floating-point operations and limitations on computer architecture (mainly memory paradigm). To be specific, we observe that matrix multiplication (X^TX) for large number of rows is limited due to memory access (from main memory) despite of a large number of floating-point operations with inexpensive addition and multiplication. The inverse for large floating-point matrix is also usually limited by memory access and large number of mathematical operations. Since n>> m, size of the inverse matrix (m x m) is relatively small compared to matrix X (n x m) (see Figures 2 and 3). Therefore, this application spent only around 20% of total time in computing inverse. Rest 75% of the time was spent in computation of X^TX (see Figure 1) which is a major limiting factor in the performance of this application. This is attributed to limited memory on L3 cache and cache misses leading to random memory access (X^T). Size of L3 cache is usually 8MB which limits the number of columns of matrix X that can sit in cache memory at a single call. For size of 8 MB, a row size (n) less than 1441 can be store in the cache at a single call. Any size greater than 1441 would force memory access directly from main memory which is 100x slower than that of cache. Another issue that impacts the perform of multiplication is random memory access due to transpose of X. Since X is

stored continuously in the memory, just interchanging rows to columns will result into random memory access leading to cache misses.

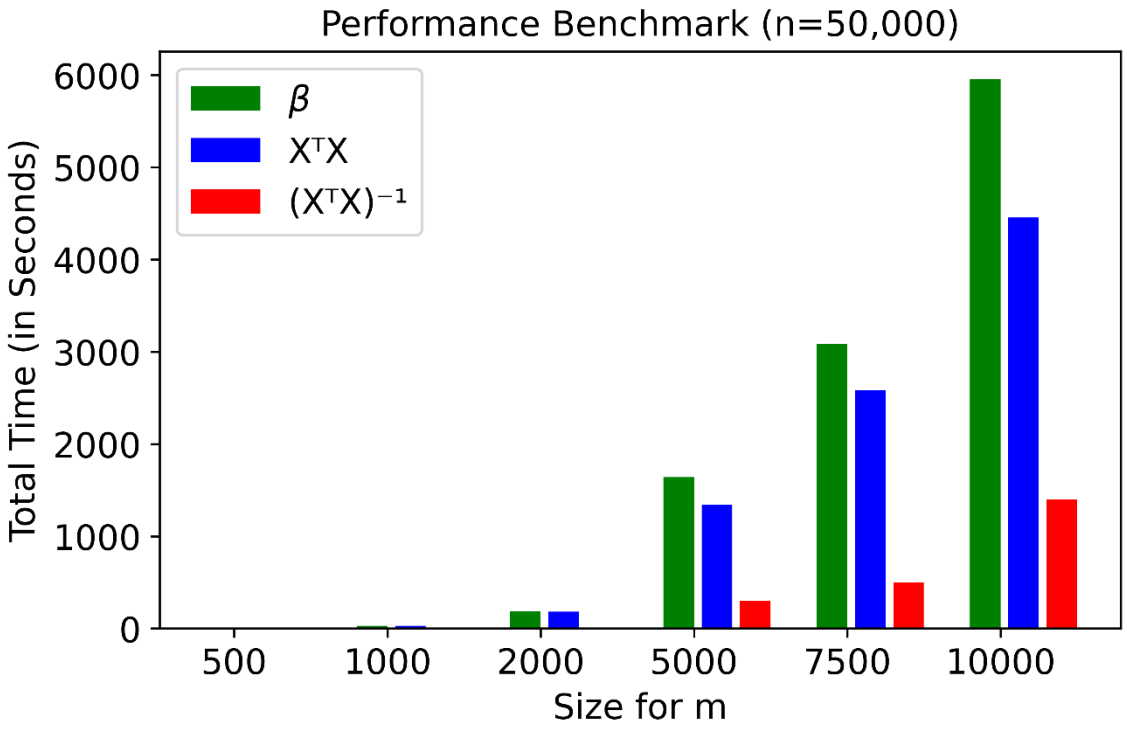

Figure 1: Time spent by the application in different regions of computations

<u>Mathematical analysis to find limiting factor in matrix multiplication</u>

Let's analyze computing time for n=50,000 and m=2,000.

Number of flops performed to compute (n+n)(mxm) matrix multiplication operations is 2 * 50,000*2000*2000 = 4 x 10^{11} Flops

Time taken by arithmetic operating unit (assuming 30 Gflops/sec as peak performance) = 4 x 10^{11} / 30x10^{9} =40/3 = 13.33 seconds

Similarly, amount of float bytes (4*2n*m*m) transferred from main memory to cache is 8*50,000*2,000*2,000 = 160*10^{10} B = 1,600 GB

Due to limitation on bus speed(assuming 10 GB/S), we can transfer 1,600 GB of data in 160 seconds which is close to observed time(=182 seconds) as shown in figure 1. 182 sec is greater than 13.33 sec proves that matrix multiplication is limited by memory access for n=50,000 and m>=2,000.

Another validation that matrix multiplication that this scenario is limited by memory latency highlighted when we tried to use openMP to split nested for loops within different threads. There was no observed speed up but increase in computation time due to pragma overhead cost,

suggesting that all the concurrent threads are waiting for the data as they share same data pipe to access memory.

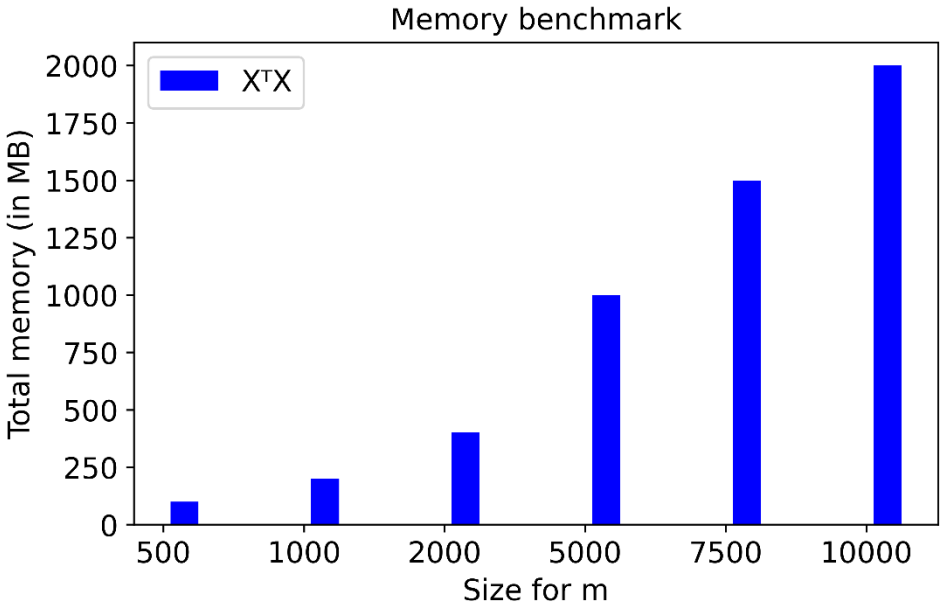

Figure 2: Memory occupied during the computation of X^TX

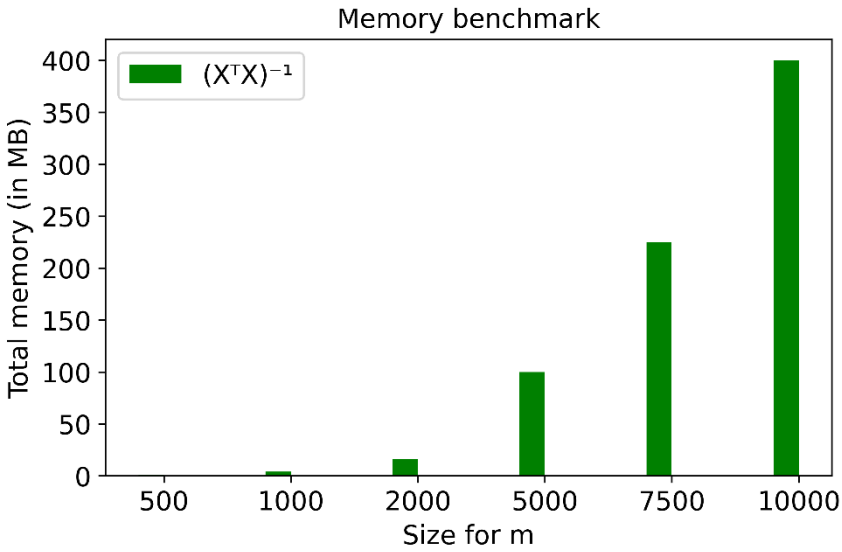

Figure 3: Memory occupied during the computation of inverse(X^TX)