Central Washington University

High Performance Computing

Spring 2019

Project 1 Report

Author:
Hermann Yepdjio

Supervisor:
Dr. Szilard Vajda

April 13, 2019

Contents

1 Introduction

For this project, we've been asked to implement the power of a square matrix. The goal was to benchmark 2 different implementations (classical using 2d arrays and a more complex version using double linked lists) of the square matrix by comparing the time it would take to raise similar instances of both of them to a given large power. Therefore, after implementing the two required versions of the square matrix, we started the experimentation by creating two instances (one for each version) of the same size, then while keeping their sizes fixed (500 * 500), we incremented their powers by 100 up to 1000 and recorded the computing time after each increment. The second part of the experiment consisted in keeping the power of both instances fixed (200) and increment their sizes by 100 up to 1000 and the time for computing the power after each increment was recorded. The results of the experimentation are displayed and discussed in the next section.

2 Result of the Experimentation

2.1 Benchmark Using the C language

For the experimentation using the C language, we recorded the running times using 3 different functions : time(), clock() and gettimeofday().

2.1.1 Keeping the Dimensionality constant

Observation As we can see from Figure 1 below,

- increasing the power of the matrix increases the time complexity linearly,
- the classical implementation takes less time to compute the power than the double linked list implementation,
- time(), clock() and gettimeofday() return about the same value.

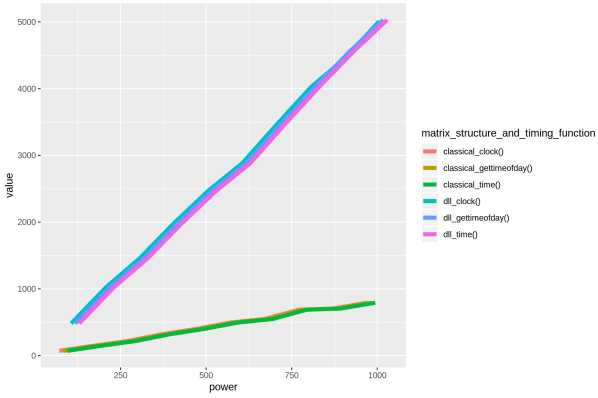

Figure 1: C-implementation(Fixed Dimensionality and Variable Power)

2.1.2 Keeping the Power constant

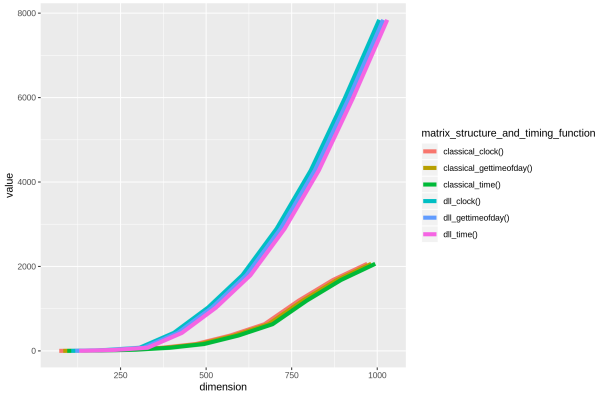

Figure 2: C-implementation(Fixed Power and Variable Dimensionality)

Observation As we can see from Figure 2 above,

- increasing the dimensionality of the matrix increases the time complexity exponentially,
- the classical implementation takes less time to compute the power than the double linked list implementation,
- time(), clock() and gettimeofday() return about the same value.

2.2 Benchmark Using the R language

For the experimentation using the R language, we considered only the classical implementation of the square matrix but used two different methods to compute its power:

- using the R "%*%" (without the quotation marks) operator to multiply two matrices,
- using 3 "for loops" with the regular "*" operator to multiply two matrices.

Hence, the two methods above are what we are benchmarking in this section.

We recorded the running times using the system.time() function which takes as input the operation or function to be timed and returns 3 different values:

- the user CPU time: which gives the CPU time spent by the current process,
- the system CPU time: which gives the CPU time spent by the kernel on behalf of the current process for doing things such as opening files, looking at the system clock or starting other processes etc...,
- the elapsed time: which is the total elapsed time perceived by the user.

2.2.1 Keeping the Dimensionality constant

Observation As we can see from Figure 3 below,

- increasing the power of the matrix increases the time complexity linearly,
- the classical implementation (i.e using the R "%*%" operator) takes less time to compute the power than the pure classic implementation (i.e using the "*" operator),
- the user CPU time is slightly better than the total elapsed time for the pure classical implementation.

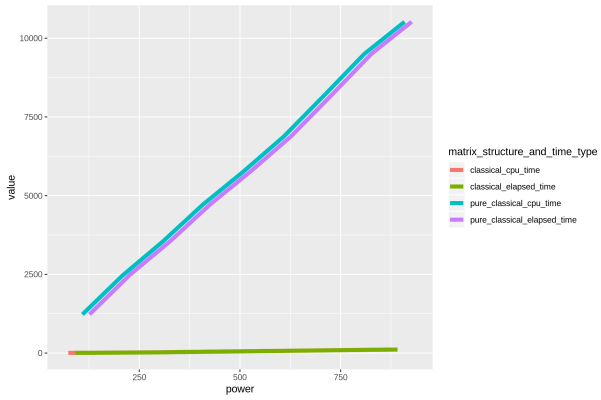

Figure 3: R-implementation(Fixed Dimensionality and Variable Power)

2.2.2 Keeping the Power constant

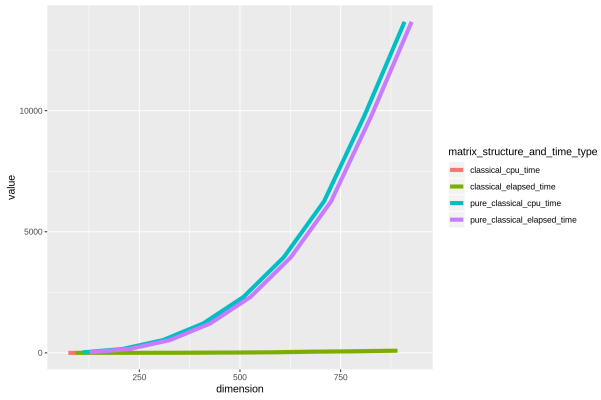

Figure 4: R-implementation(Fixed Power and Variable dimensionality)

Observation As we can see from Figure 4 above,

- increasing the power of the matrix increases the time complexity exponentially,
- the classical implementation (i.e using the R "%*%" operator) takes less time to compute the power than the pure classic implementation (i.e using the "*" operator),

- the user CPU time is slightly better than the total elapsed time for the pure classical implementation.

2.3 C-classical vs R-Pure Classical

For this experimentation, we considered using the clock() function to record the time for the C-implementation and compared it with the "user CPU" time returned by the system.time() function on the R implementation.

2.3.1 Keeping the Dimensionality constant

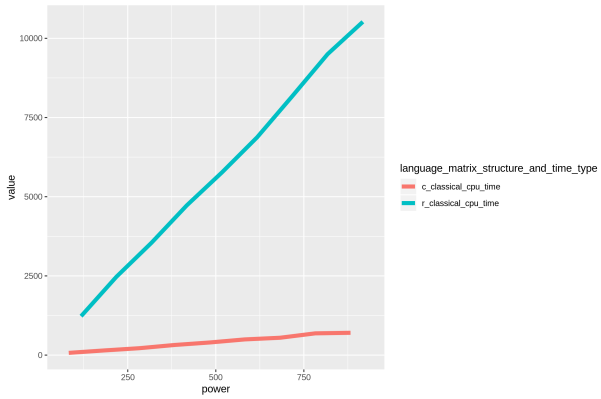

Figure 5: C Vs R-implementations(Fixed Dimensionality and Variable Power)

Observation As we can see from Figure 5 above,

- increasing the power of the matrix increases the time complexity linearly,
- the C implementation takes less time to compute the power than the R one,

2.3.2 Keeping the Power constant

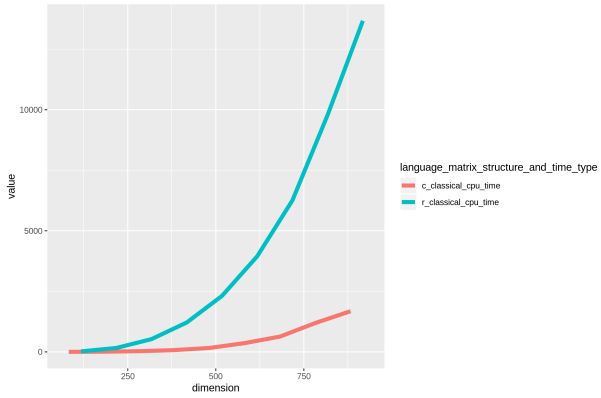

Figure 6: C Vs R-implementations(Fixed Power and Variable Dimensionality)

Observation As we can see from Figure 6 above,

- increasing the power of the matrix increases the time complexity exponentially,
- the C implementation takes less time to compute the power than the R one,

3 Conclusion

After experimenting with the power of square matrices (using 2 different implementations and 2 different programming languages), we can conclude that in C, the classical implementation is faster than the double linked list one, in R, the classical implementation is faster than the pure classical one, the classical implementation in C is faster than the pure classical one in R and the classical implementation in R is better than the the classical one in C. Therefore, we recommend using the R classical implementation to compute the power of a large matrix.