Matrix Efficiencies in R

Introduction

Indicative report comparing R matrix efficiencies. The initial investigation is on R native solutions including techniques and support libraries for matrix operations. A test on elementwise function change to 2 dimiensional matrix of 1000 x 1000 matrix with:

- stardard looping through matrix
- sapply
- parrallel (n clusters)

To measure perfomance proc.time function was applied to determines how much real and CPU time (in seconds) the currently running R process has already taken. The user time is the CPU time charged for the execution of user instructions of the calling process. The system time is the CPU time charged for execution by the system on behalf of the calling process. 1

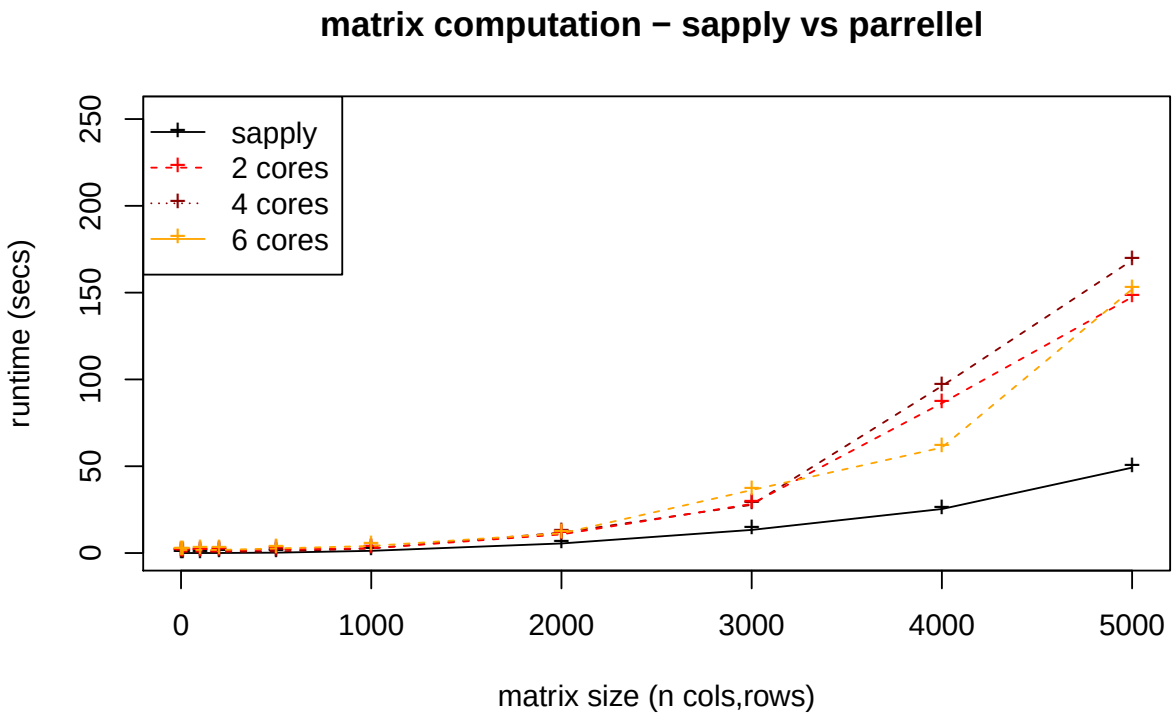
	for_loop	sapply	parallel_2	parallel_4	parallel_6
user.self	4.63	1.28	0.84	0.91	0.97
sys.self	3.07	0.14	0.17	0.22	0.21
elapsed	7.72	1.42	2.03	2	1.53

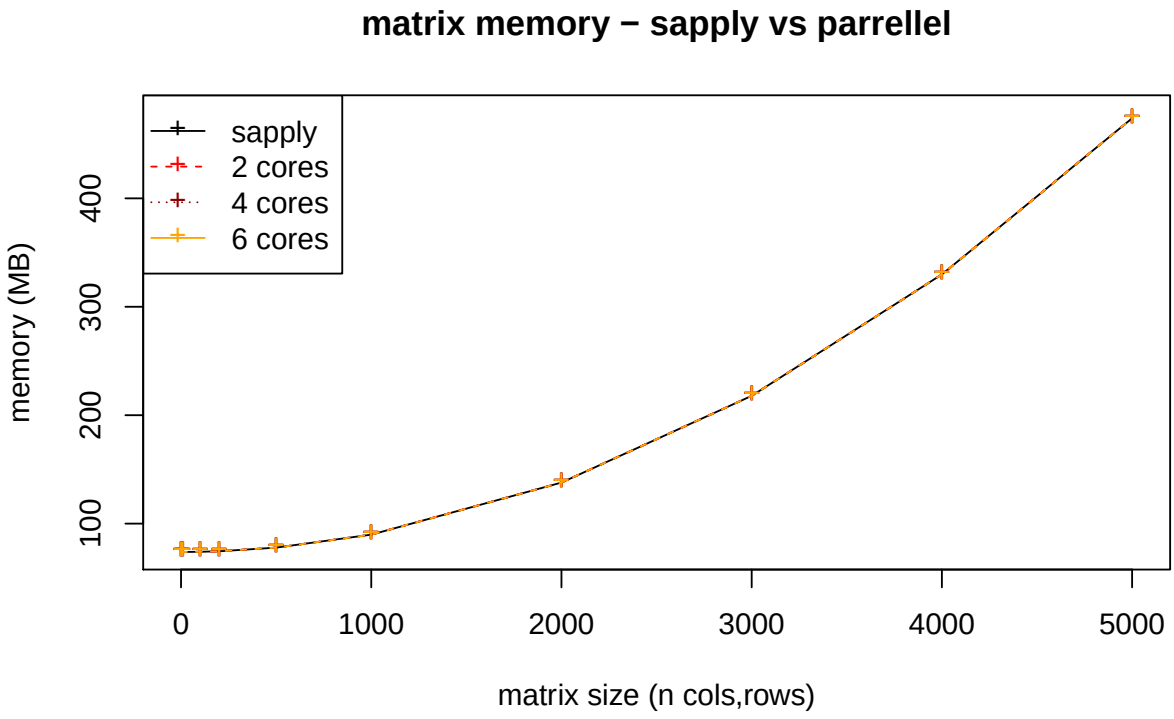
Results

sapply performce show the lowest elapse time on a 1000 x 1000 matrix, this is likely due to the overhead of running jobs in parallel. If the jobs you fire at the worker nodes take a significant amount of time then it is likely parallelization will improve overall performance. If individual jobs take only milliseconds, the overhead of constantly firing off jobs will deteriorate overall performance. Further optimising is possible by dividing the work over the nodes in such a way that the jobs are sufficiently long, say at least a few second. Parrallelisation advantages will likely be shown when runing models simultaneously that individual will run

for hours.

Computational and memory plots of 2d matrix with varying size:





Memory Limitation for larger Matrixs

Any matrix over 7000 rows/columns in R code failed to compute on personal compute, due to massive memory requrements on the RAM. To address limitation, some possiblie solutions are:

1. Rewrite R algorithm to work on submatrix, saving to disk unused data. i.e Fox algorithm.
2. Pythons PyTables and NumPy.PyTables store data on disk in HDF compression with possiblity of tens of millions of rows.
3. Vertical scaling. Larger Compute instances, such as cloud computing to access very large RAM capacity (244GB).
4. Horizontal scaling with distributed computing. This includes systems such as Apache Spark and Hadoop etc.

Computational Improvements to R code

As shown R code native sapply for matrix operations outperforms parrrllel processing and delievers quick results for large matrix. This may not be the case for other operations, if faster operations are required due to bottle necks the following mabye possible.

1. Rewrite functions in C/C++, the bottlenecks that can be address are:

- Loops that cant be easily vectorised because subsequent iterations depend on previous ones.
- Recursive functions
- Problems that require advanced data structures

2. Frameworks

- tensorflow (utilisig monder architecturd -GPU)
- mapreduce, Hadoop (clusters for big data)

3. Other improved library languagues

- GNU GSL
- Stan