SERIAL AND PARALLEL IMPLEMENTATION OF MATRIX MULTIPLICATION AND INVERSION

COURSE OF ADVANCED COMPUTER ARCHITECTURE

A.Y. 2021/2022

Riccardo Crescenti
Riccardo Doveri

INTRODUCTION

- The goal of this project is to implement both serial algorithms, parallelize them using open MPI and perform a performance and speed up analysis executed both in a local machine and through scalability tests on clusters built using Google Cloud Platform.
- We decided to use standard algorithm so we could operate with matrices of any dimensions. The only constraints came from mathematical point of view
- Product: the number of columns of the first matrix must be equal to the number of columns of the second matrix.
- Inversion: matrix has to be NxN, otherwise it can’t be inverted. LU decomposition method was used since it has less computational cost ($\frac{8}{3} n^3$)

MATRIX PRODUCT

Algorithm

Algorithm 1 Matrix Multiplication

INPUT: Matrices A and B

OUTPUT: Matrix C

for $i = 0, 1, 2, \ldots, m$ **do**
    for $j = 0, 1, 2, \ldots, p$ **do**
        $sum = 0$
        for $k = 0, 1, 2, \ldots, n$ **do**
            $sum += A_{i,k} \cdot B_{k,j}$
        end for
        $C_{ij} = sum$
    end for
end for

Domain decomposition

	A00	A01	A02	A03
A	A10	A11	A12	A13
	A20	A21	A22	A23
	A30	A31	A32	A33

	B00	B01	B02	B03
B	B10	B11	B12	B13
	B20	B21	B22	B23
	B30	B31	B32	B33

	C00	C01	C02	C03
C	C10	C11	C12	C13
	C20	C21	C22	C23
	C30	C31	C32	C33

A TUTTI
PR. 0
PR 1

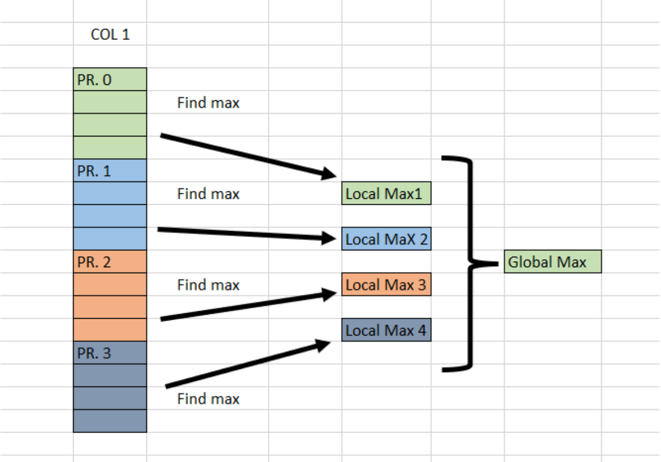
MATRIX INVERSION

3 parts:

- LU decomposition with pivoting (less parallelizable, must proceed column by column)
- Check determinant
- LUP inversion (highly parallelizable, column blocks division)

MATRIX INVERSION – domain decomposition

Local maxima



LU decomposition

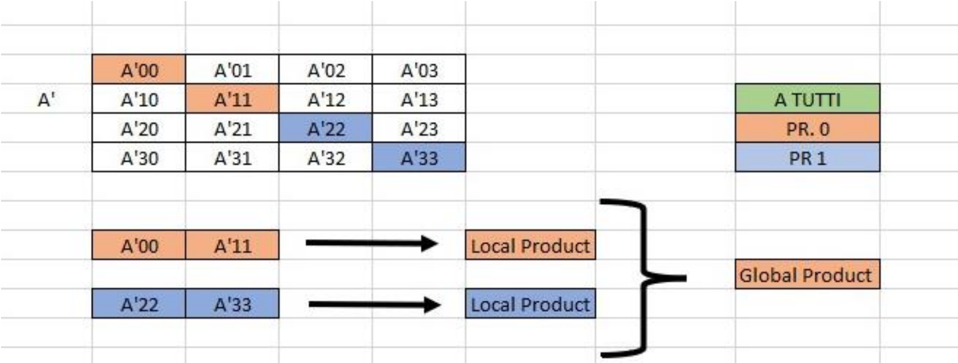
FIRST ITERATION

					A TUTTI
	A00	A01	A02	A03	PR. 0
A	A10	A11	A12	A13	PR 1
	A20	A21	A22	A23	
	A30	A31	A32	A33	

SECOND ITERATION

	A'00	A01	A02	A03
A'	A'10	A11	A12	A13
	A'20	A21	A22	A23
	A'30	A31	A32	A33

Check determinant



Inversion

					A TUTTI
	A'00	A'01	A'02	A'03	PR. 0
A'	A'10	A'11	A'12	A'13	PR 1
	A'20	A'21	A'22	A'23	
	A'30	A'31	A'32	A'33	

	Ainv00	Ainv01	Ainv02	Ainv03
Ainv	Ainv10	Ainv11	Ainv12	Ainv13
	Ainv20	Ainv21	Ainv22	Ainv23
	Ainv30	Ainv31	Ainv32	Ainv33

MATRIX INVERSION – versions 2 and 3

- The inversion function from A' to Ainv showed improvements visible immediately
- On the other hand, the functions for calculating the maximum and determining check, for the cases tested, were not particularly efficient
- We created a v2 version of the program in which we have replaced the portion of code in parallel with the serial one.
- In the v3 version we performed the LU decomposition, maximum search, and determinant check functions in serial, keeping only the true inversion function in parallel.
- We think this is due to over-parallelization for relatively large matrices (our tests are limited to cases of 3000x3000). Based on our studies in the case of even larger matrices we think that this difference will not be so deep and the v1 will overcome v3.

ESTIMATING THE SPEED UP

- Thanks to the Amdahl's law, we can approximate the speed up for the 2 algorithms
- We estimated 97.8% of the product and about 84% of inversion

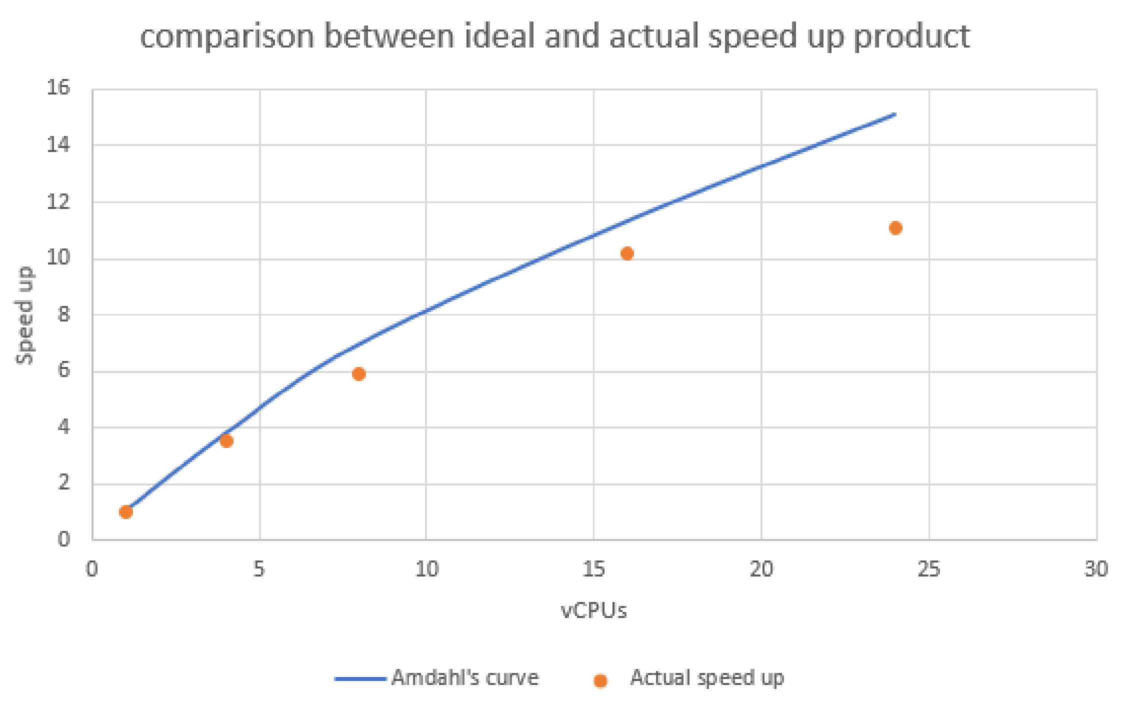
- P is the fraction of parallelizable code
- S is the not parallelizable code
- N is the number of cores

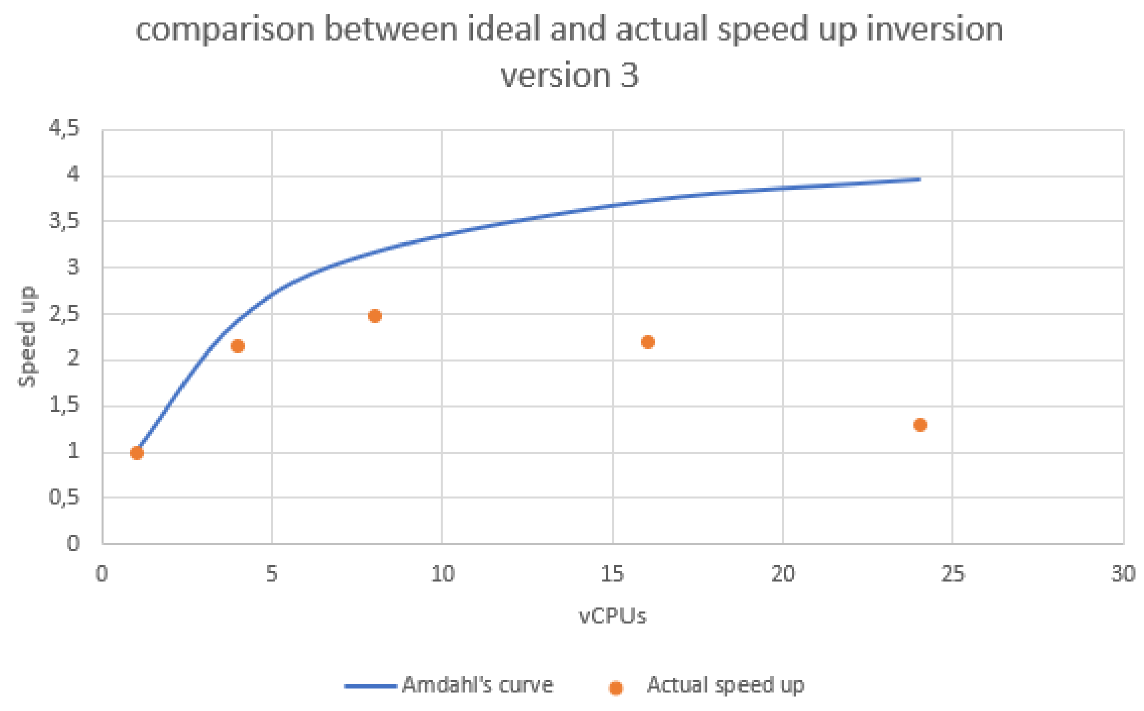
$$Speedup(N) = \frac{Time_{serial}}{Time_{parallel}(N)} = \frac{(S+P)T_{serial}}{S \cdot T_{serial} + \frac{P \cdot T_{serial}}{N}} = \frac{S+P}{S+\frac{P}{N}} = \frac{1}{S+\frac{P}{N}}$$

TESTING AND DEBUGGING

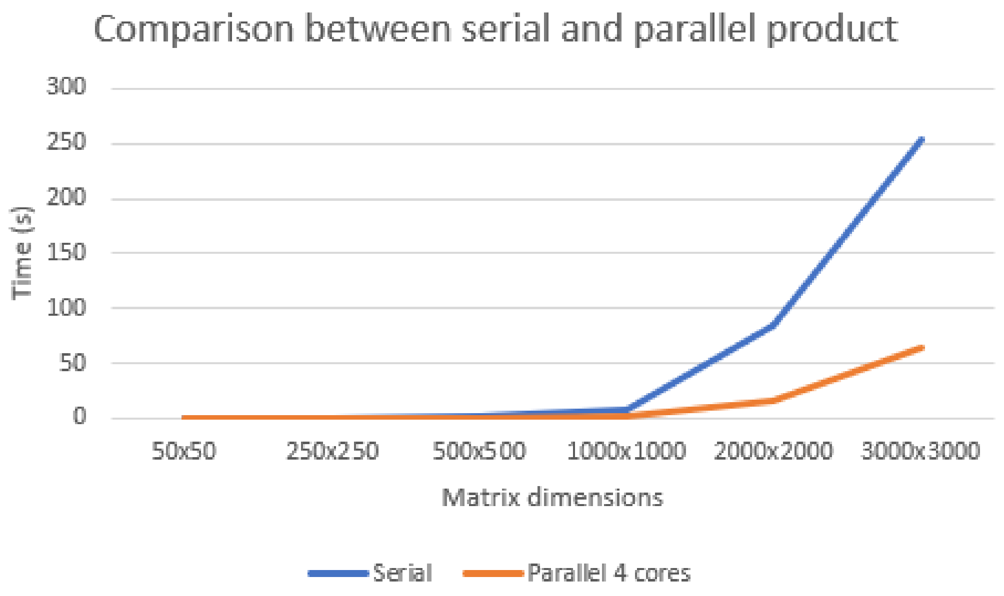
- Initially on a local virtual machine (CENTOS 8), then we moved to Google Cloud Platform
- Checking the validity of the serial algorithms for small matrices with an online calculator
- Checking validity for large matrices, created with memory allocation functions
- Checking validity of parallel algorithms for a small number of cores in local
- Scalability tests on GCP
- All matrices are generated by a program we made
- Check of possible problems as file not found, incorrect matrices, incorrect dimensions or matrix degenerated

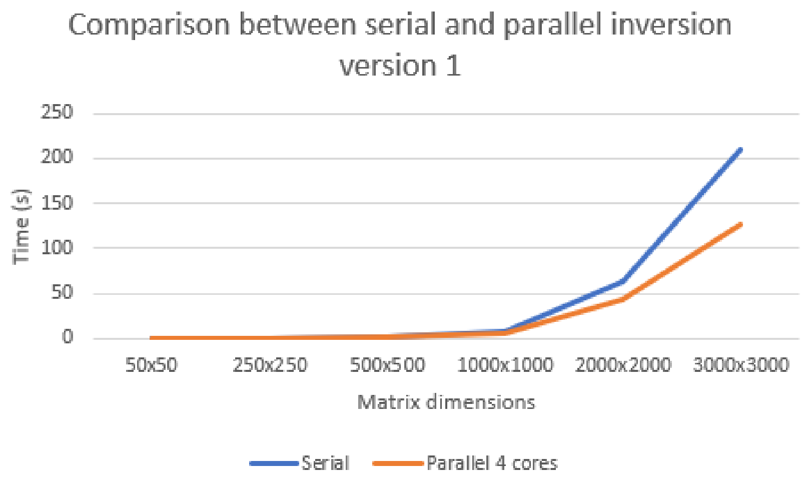
PERFORMANCE ANALYSIS – speed up

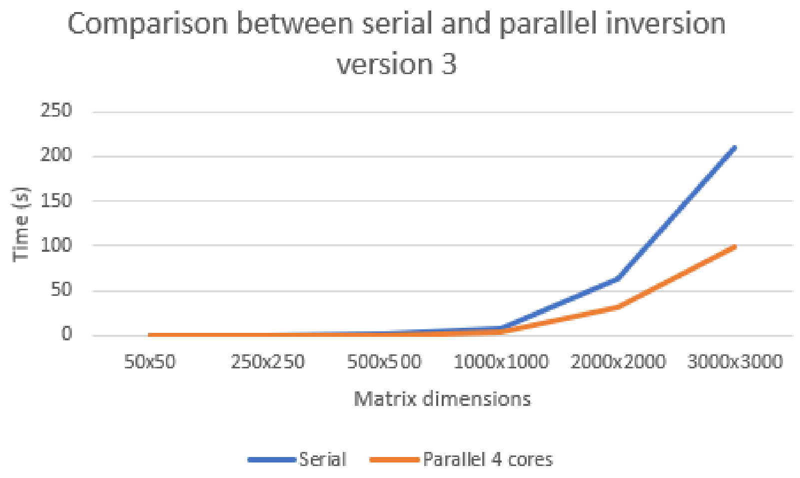




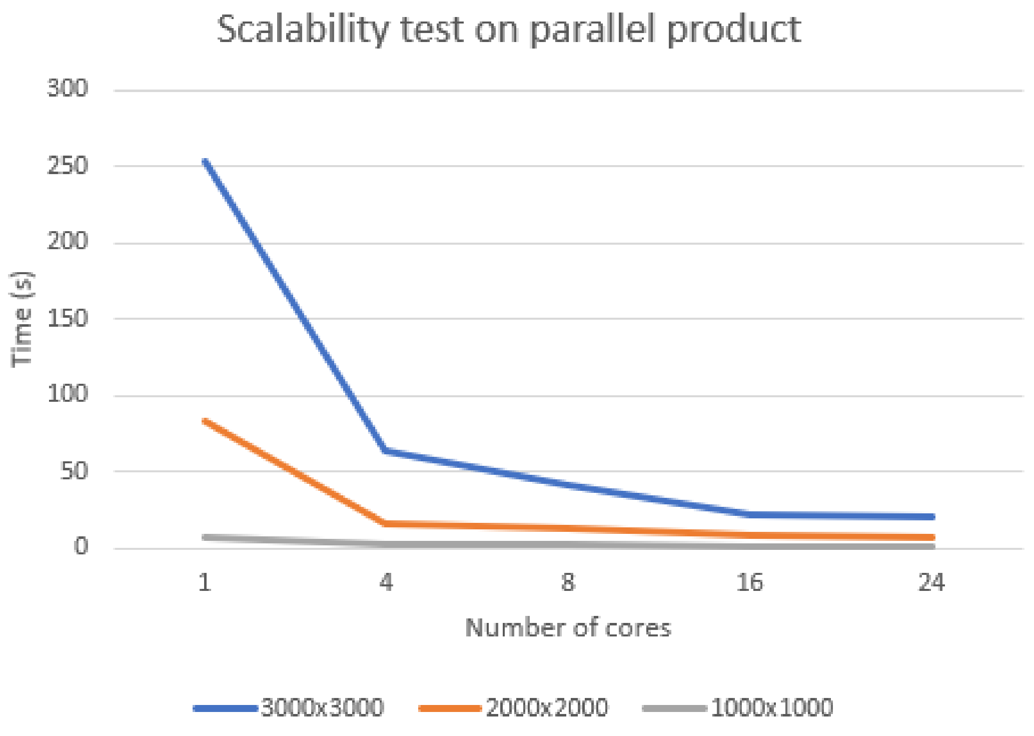
PERFORMANCE ANALYSIS – local execution

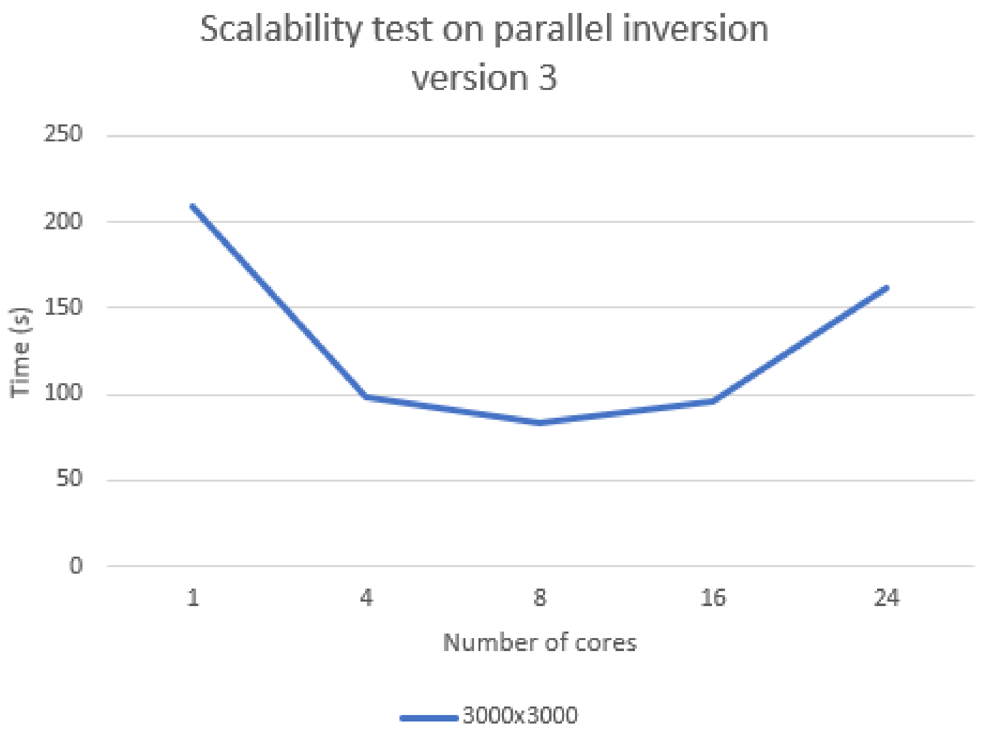






PERFORMANCE ANALYSIS – GCP execution

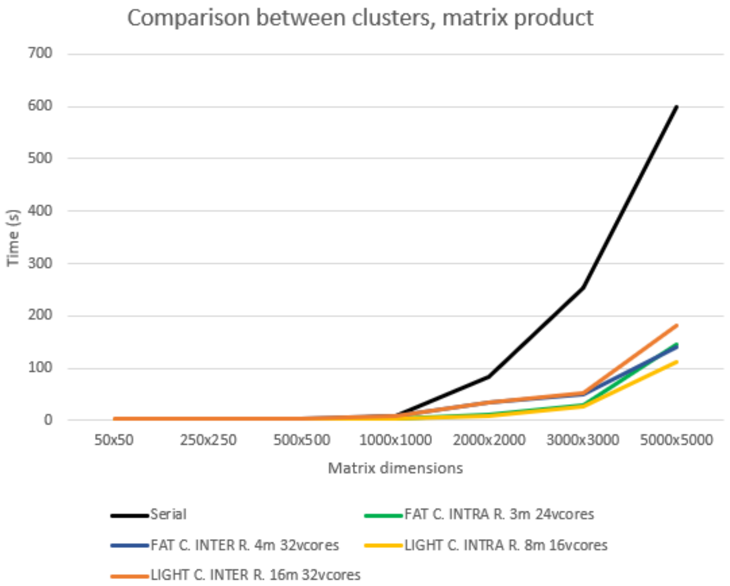


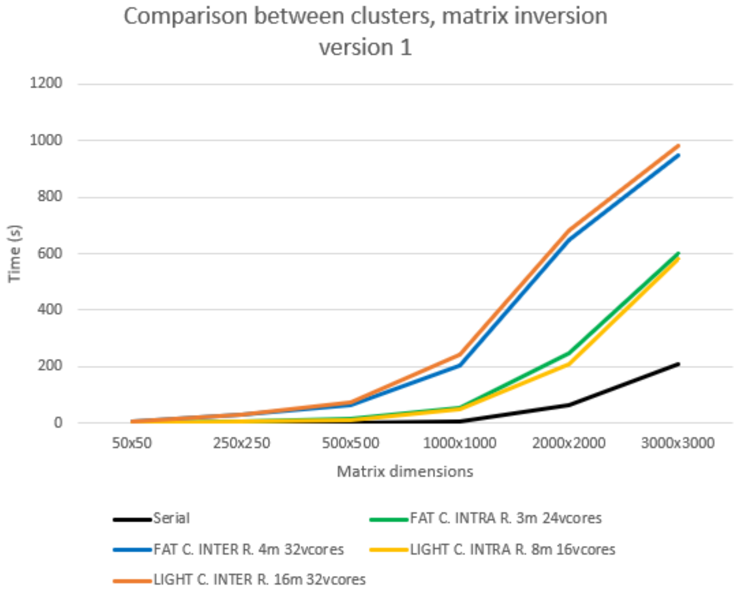


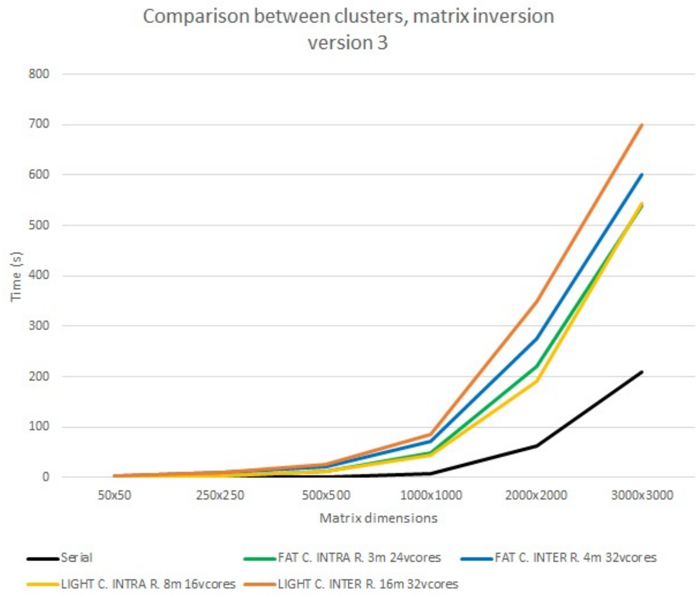
PERFORMANCE ANALYSIS – clusters

- Fat cluster:
  - 3 machines with 8 core each, means 24 total infra-regional (US central)
  - 4 machines with 8 cores each, equal to 32 inter-regional
- Light cluster:
  - 8 machines with two cores each means 16 total infra-regional cores (US central)
  - 16 machines with two cores each equals to 32 inter-regional cores

PERFORMANCE ANALYSIS – clusters







Second generation CPUs were selected, whose CPU platform is selected based on availability.
The 8-core machines for fat clusters were configured with 36GB of RAM and 50GB of disk.
The 2-core machines for light clusters were configured with 8GB of RAM and 50GB of disk.

CONCLUSIONS

- We implemented serial matrix and inversion product algorithms
- Parallelized and performed a performance and speed up analysis
- Obtained results that reflect the values studied a priori quite well
- Best and worst solution for product/inversion/clusters were found, but it would be necessary to make tests with very large matrices.