



Московский государственный университет имени М. В. Ломоносова

Факультет вычислительной математики и кибернетики

Кафедра суперкомпьютеров и квантовой информатики

**Отчёт по заданию 2:**

**Разработка параллельной версии решения СЛАУ  
методом отражения и обратным ходом метода  
Гаусса с использованием MPI**

Аксой Тевфик Огузхан

323 группа

## Постановка задачи

- Написать последовательный алгоритм, приводящий матрицу в верхний треугольный вид методом отражения
- Найти элементы вектора столбца  $x$  решив приведенную в верхний треугольный вид матрицу методом обратного хода Гаусса
- Найти и вывести норму невязки  $\|Ax - b\|$
- Распараллелить алгоритмы приведения матрицы в верхний треугольный вид методом отражения и обратного хода метода Гаусса.
- Измерить времена выполнений параллельных алгоритмов
- Запустить программу на вычислительном комплексе IBM Polus с разными размерами входных данных и количествами MPI процессов. Сделать выводы о времени выполнения для различных размеров матриц и привести таблицы и графики ускорения и эффективности параллельной программы и сравнить данные с OMP.

## Кратко алгоритм решения

В начале выделяются памяти для столбцов матрицы и определяется какому процессу будет принадлежать какой столбец (циклическое распределение столбцов матрицы). После этого, произвольные значения заполняют столбцы матрицы  $A$  и вектор  $b$ . Далее:

1. Методом отражения матрица приводится в верхнетреугольную форму после  $N$  шагов, в каждом из которых после вычисления вектора  $v$  по  $U(v) = E - 2v^2$  данный процесс рассылает всем остальным процессам вектор  $v$  и соответственно каждый процесс меняет значения своих столбцов.
2. При обратном ходе Гаусса выполняющимися в обратном направлении, процесс, которому принадлежит данный столбец, выполняет соответствующую операцию `MPI_Reduce` для получения значения вектора  $x$  умноженные на соответствующие числа из матрицы  $A$ .
3. После обратного хода Гаусса, на каждом процессе хранятся соответствующие значения вектора  $x$  и каждый процесс отправляет свою часть вектора  $x$  мастер процессу по рангу 0.

## Компиляция и запуск программы

*В программе используются такие особенности C++11, как лямбда функции, равномерные распределения чисел, чтобы создать рандомные числа в стиле современного C++ и синонимы шаблонов и типов (type alias, alias template)*

Программа компилируется на Polus с помощью [Shell скрипта](#)\*, который выполняет скрипт для получения доступа к IBM компилятору и [Makefile](#)\*, который компилирует программу. Для того, чтобы использовать, нужно выполнить скрипт вводя команду в терминале «. **mpi\_source.sh**».

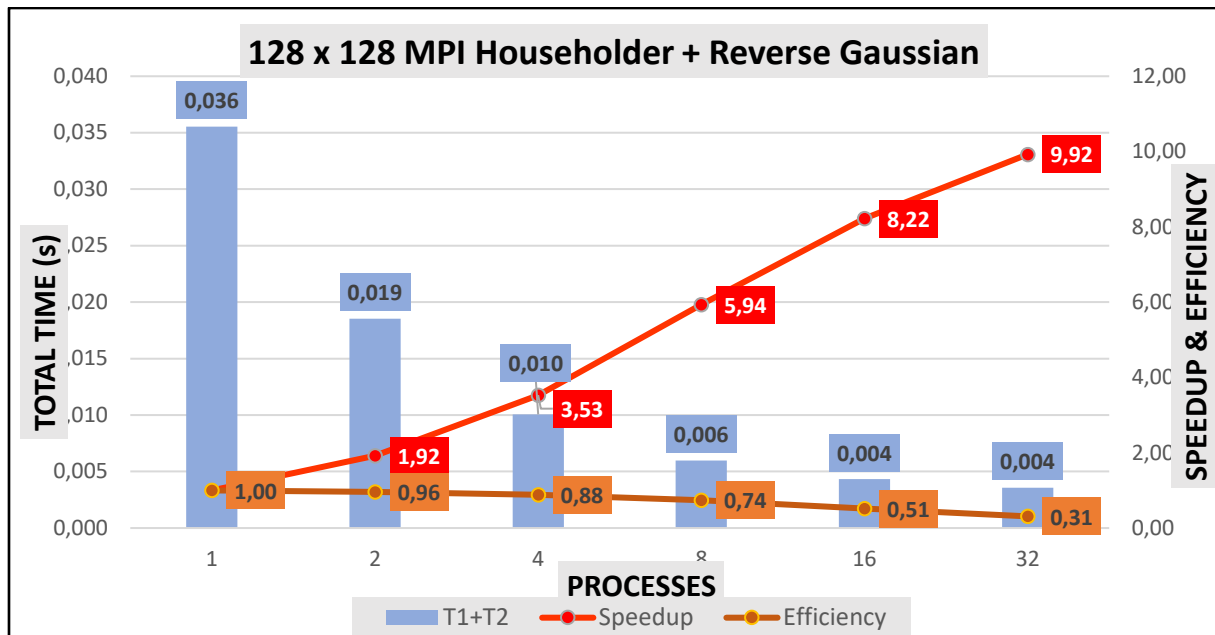
После компиляции программы можно запустить [Python скрипт](#)\*, который создаёт планировщик «**mpijob.lsf**» для размещения работы на очередь по указанным данным и вызывает команду «**bsub < mpijob.lsf**». Так же можно при запуске скрипта указать число MPI процессов и размеры матриц.

\* ссылка

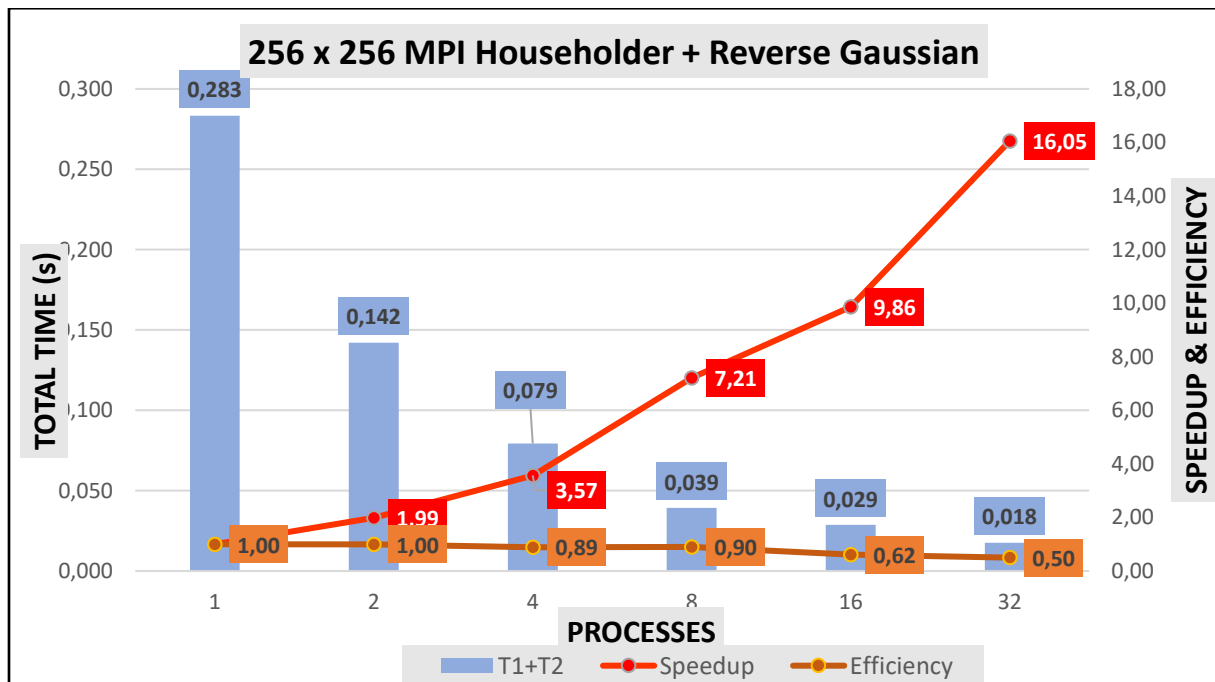
## Результаты и выводы

После запуска и обработки программы в Полусе, файл с результатами «**result.out**» на свой компьютер скопировал, дальше для составления графиков по результатам, перенёс данные от результатов в Excel с помощью [скрипта](#). В Excel составил графики ускорения в зависимости от количества процессов. В графику данные о T1+T2 добавил, а не отдельно, так как T2 очень маленькие и по сравнению с T1 невидимыми становятся.

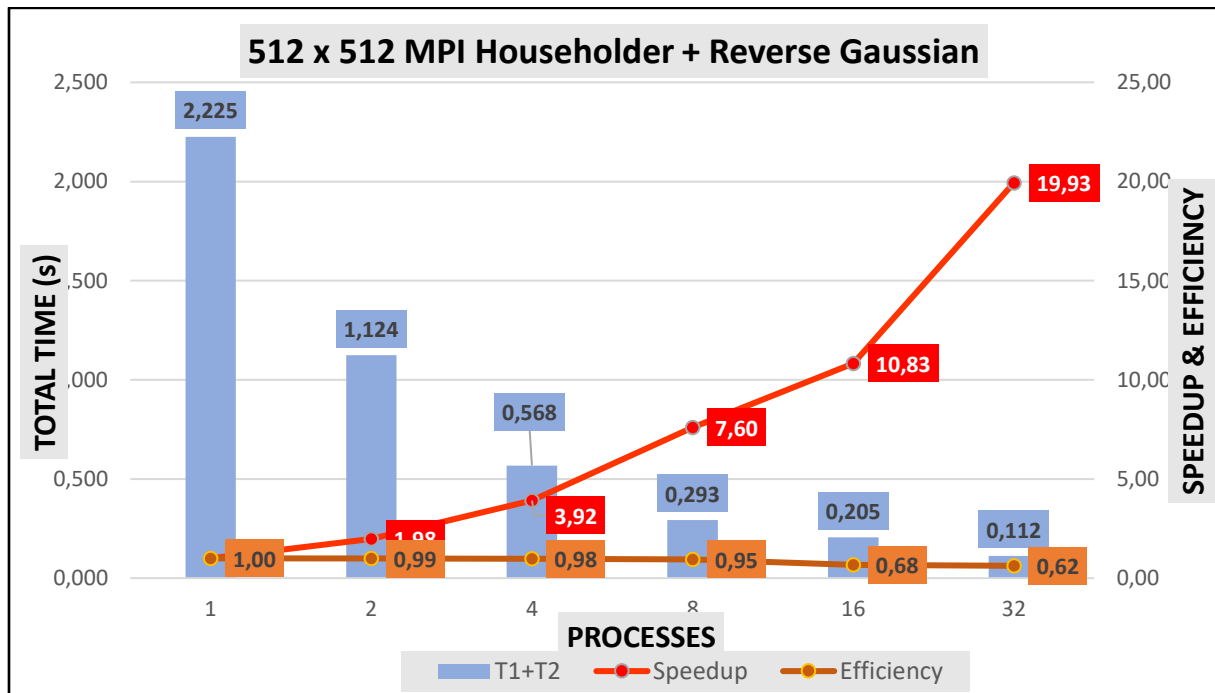
Size	Processes	T1	T2	T1+T2	Speedup	Efficiency
128x128	1	0,0353	0,0002	0,036	1,00	1,00
128x128	2	0,0182	0,0003	0,019	1,92	0,96
128x128	4	0,0096	0,0004	0,010	3,53	0,88
128x128	8	0,0056	0,0004	0,006	5,94	0,74
128x128	16	0,0038	0,0005	0,004	8,22	0,51
128x128	32	0,0028	0,0008	0,004	9,92	0,31



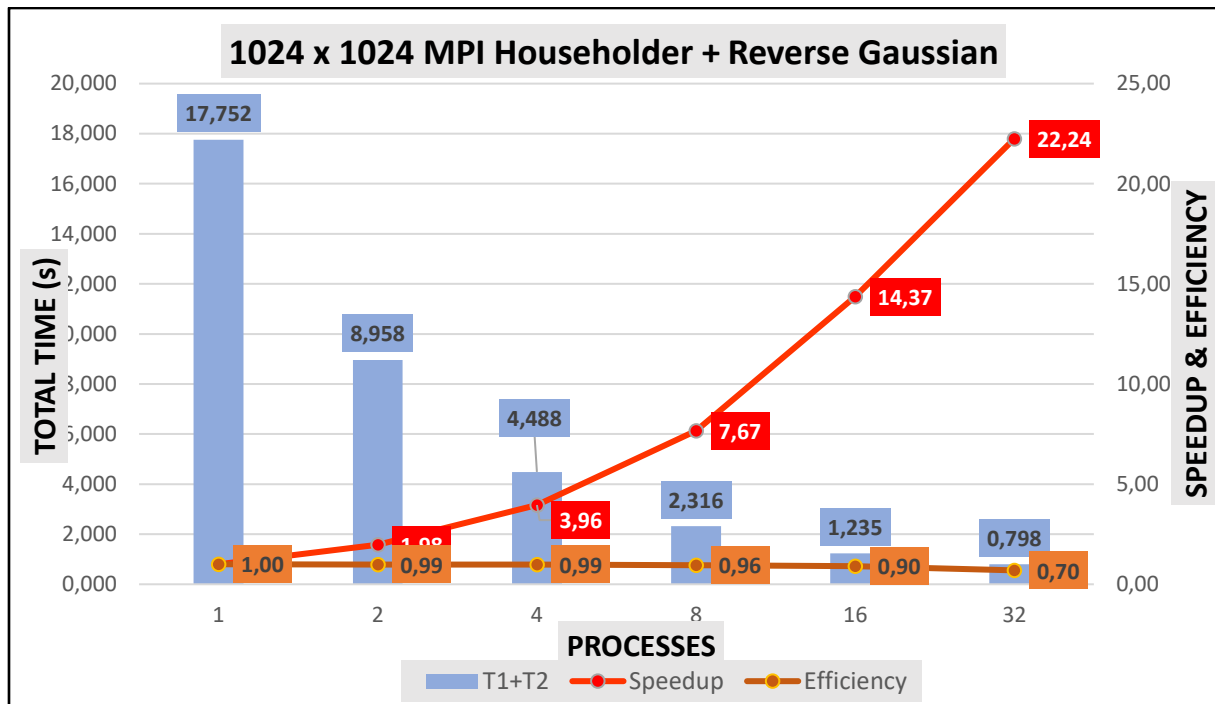
Size	Processes	T1	T2	T1+T2	Speedup	Efficiency
256x256	1	0,2825	0,0007	0,283	1,00	1,00
256x256	2	0,1413	0,0008	0,142	1,99	1,00
256x256	4	0,0786	0,0007	0,079	3,57	0,89
256x256	8	0,0384	0,0009	0,039	7,21	0,90
256x256	16	0,0278	0,0009	0,029	9,86	0,62
256x256	32	0,0161	0,0015	0,018	16,05	0,50



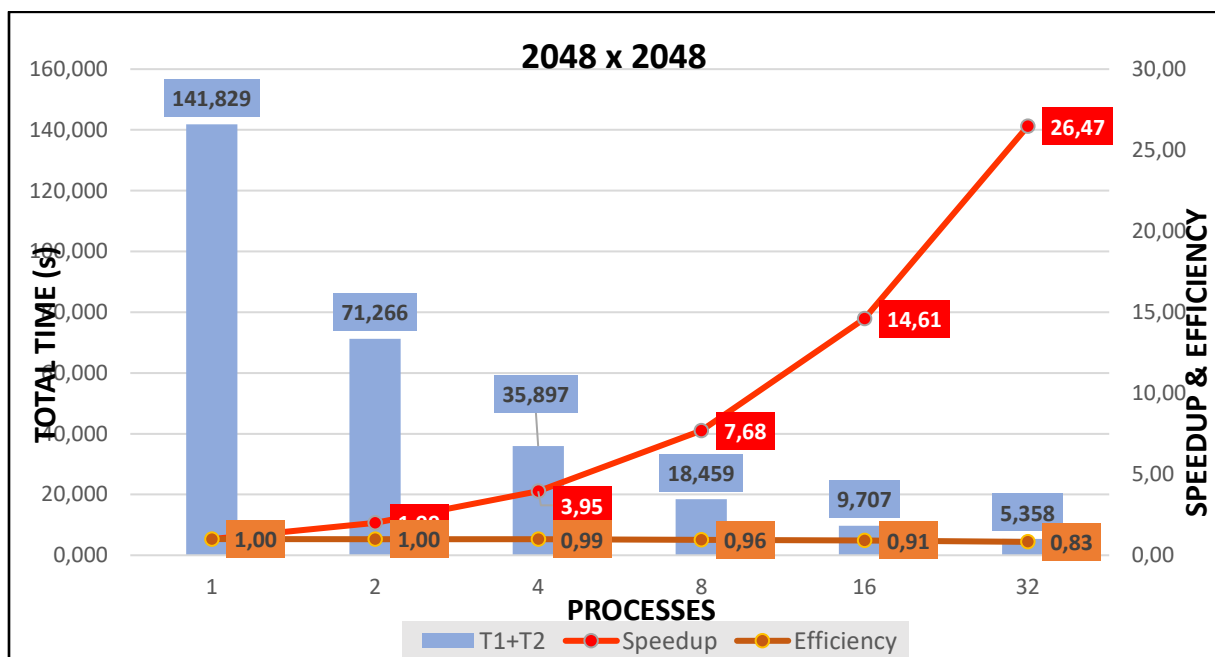
Size	Processes	T1	T2	T1+T2	Speedup	Efficiency
512x512	1	2,2225	0,0028	2,225	1,00	1,00
512x512	2	1,1220	0,0022	1,124	1,98	0,99
512x512	4	0,5663	0,0017	0,568	3,92	0,98
512x512	8	0,2910	0,0017	0,293	7,60	0,95
512x512	16	0,2039	0,0016	0,205	10,83	0,68
512x512	32	0,1093	0,0024	0,112	19,93	0,62



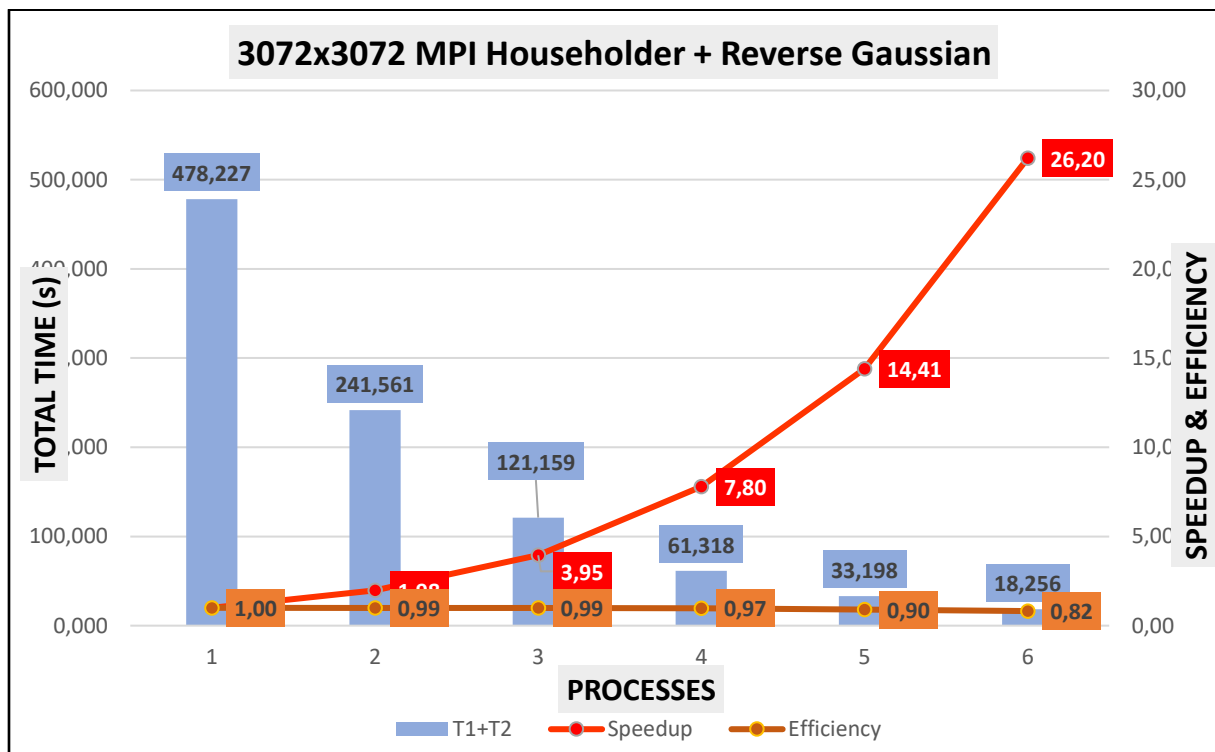
Size	Processes	T1	T2	T1+T2	Speedup	Efficiency
1024x1024	1	17,740	0,011	17,752	1,00	1,00
1024x1024	2	8,951	0,007	8,958	1,98	0,99
1024x1024	4	4,483	0,005	4,488	3,96	0,99
1024x1024	8	2,312	0,004	2,316	7,67	0,96
1024x1024	16	1,232	0,003	1,235	14,37	0,90
1024x1024	32	0,794	0,004	0,798	22,24	0,70



Size	Processes	T1	T2	T1+T2	Speedup	Efficiency
2048x2048	1	141,7810	0,0479	141,829	1,00	1,00
2048x2048	2	71,2389	0,0268	71,266	1,99	1,00
2048x2048	4	35,8813	0,0160	35,897	3,95	0,99
2048x2048	8	18,4488	0,0101	18,459	7,68	0,96
2048x2048	16	9,6987	0,0078	9,707	14,61	0,91
2048x2048	32	5,3491	0,0092	5,358	26,47	0,83



Size	Processes	T1	T2	T1+T2	Speedup	Efficiency
3072x3072	1	478,1050	0,1225	478,227	1,00	1,00
3072x3072	2	241,4990	0,0615443	241,561	1,98	0,99
3072x3072	4	121,1260	0,0330	121,159	3,95	0,99
3072x3072	8	61,2979	0,0198	61,318	7,80	0,97
3072x3072	16	33,1844	0,0140	33,198	14,41	0,90
3072x3072	32	18,2414	0,0144	18,256	26,20	0,82



Полученные данные показывают, что когда имеем большие размеры матриц, тогда ускорения и эффективность распараллеливания становится больше.

Для размеров матриц (которые я запускал с OMP) ускорение для одинакового размера процессов/нитей (до 32 для процессов) полученное ускорение немного отличается и ускорение в программе с MPI немного выше и соответственно более эффективно.

	128 x 128					
	OpenMP			MPI		
THREADS/PROCESSES	Time	Speedup	Efficiency	Time	Speedup	Efficiency
1	2,55423	1,00	1,00	0,03553	1,00	1,00
2	1,33601	1,91	0,96	0,01853	1,92	0,96
4	0,72688	3,51	0,88	0,01006	3,53	0,88
8	0,43849	5,83	0,73	0,00599	5,94	0,74
16	0,44027	5,80	0,36	0,00432	8,22	0,51
32	0,31906	8,01	0,25	0,00358	9,92	0,31

	256 x 256					
	OpenMP			MPI		
THREADS/PROCESSES	Time	Speedup	Efficiency	Time	Speedup	Efficiency
1	39,073	1,00	1,00	0,2832	1,00	1,00
2	19,947	1,96	0,98	0,1421	1,99	1,00
4	10,500	3,72	0,93	0,0793	3,57	0,89
8	5,968	6,55	0,82	0,0393	7,21	0,90
16	4,634	8,43	0,53	0,0287	9,86	0,62
32	3,737	10,46	0,33	0,0176	16,05	0,50

	512 x 512					
	OpenMP			MPI		
THREADS/PROCESSES	Time	Speedup	Efficiency	Time	Speedup	Efficiency
1	617,7820	1,00	1,00	2,225	1,00	1,00
2	313,1790	1,97	0,99	1,124	1,98	0,99
4	159,8080	3,87	0,97	0,568	3,92	0,98
8	87,2225	7,08	0,89	0,293	7,60	0,95
16	65,5784	9,42	0,59	0,205	10,83	0,68
32	51,0318	12,11	0,38	0,112	19,93	0,62