МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ
УНИВЕРСИТЕТ
имени М.В.Ломоносова
Факультет вычислительной математики и кибернетики

Компьютерный практикум по учебному курсу
«Технологическая практика»

Разработка параллельного алгоритма для решения системы линейных уравнений методом отражений

ОТЧЕТ
о выполненном задании

студента 323 учебной группы факультета ВМК МГУ
Новикова Дмитрия Андреевича

Москва
2022 год

Содержание

1. Постановка задачи

Одной из основных задач вычислительной линейной алгебры является задача решения систем линейных уравнений (СЛАУ). Вычисления, связанные с решением СЛАУ применяются во многих сферах человеческой деятельности.

Целью данной работы является разработка программы, вычисляющей решение системы линейных алгебраических уравнений больших размерностей. Используя технологии OpenMP и MPI требуется разработать алгоритм, позволяющий достаточно ускорить этот процесс. За основу берется метод отражений решения СЛАУ.

2. Цели и задачи практической работы

1) Реализовать OpenMP-версию параллельного алгоритма.

2) Реализовать MPI-версию параллельного алгоритма.

3) Протестировать работу алгоритма на кластера Polus.

4) Привести сравнительные результаты работы OpenMP и MPI версий алгоритма (таблицы и графики).

3. Алгоритм для решения системы линейных уравнений методом отражений

Линейная система n уравнений с n неизвестными $x_1, x_2, \ldots, x_n$ может быть представлена в виде:

$$\begin{cases} \mathrm{a}_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = b_1 \\ \mathrm{a}_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n = b_2 \\ \ldots \\ \mathrm{a}_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn}x_n = b_n \end{cases}$$

В более кратком (матричном) виде система имеет вид:

$$A \cdot x = b$$

,где A - плотная вещественная матрица размера $n \times n$, векторы b и x состоят из n элементов.

Матрицей отражения называется матрица:

$$U(w) = I - 2ww^T$$

,где I - единичная матрица размера $n \times n$, w вектор-столбец такой, что $||w|| = 1$, вектор w^T - транспонированный вектор w.

Через $\mathbf{a}^{(k)} = (a_{kk}, \ldots, a_{nk})$ обозначается вектор из $(n-k+1)$ элементов k-ого столбца матрицы A, тогда существует вектор:

$$w^{(k)} = \pm \frac{\mathbf{a}^{(k)} - ||\mathbf{a}^{(k)}||e_k}{||\mathbf{a}^{(k)} - ||\mathbf{a}^{(k)}||e_k||}$$

,где $e_k = (1, 0, \ldots, 0)$ вектор из $(n-k+1)$ элементов.

Умножаестя левая и правая части системы уравнений $A^{(k-1)} \cdot x = b^{(k-1)}$ на матрицу $U^{(k)} = U(w^{(k)})$ слева, получается новая система уравнений $A^{(k)} \cdot x = b^{(k)}$, где $A^{(k)} = U^{(k)}U^{(k-1)} \cdots U^{(1)}A$, а $b^{(k-1)} = U^{(k)}U^{(k-1)} \cdots U^{(1)}b$.

$$A^{(k)} = \begin{pmatrix} ||\mathbf{a}^{(1)}|| & c_{12} & c_{13} & \cdots & c_{1,k-1} & c_{1k} & c_{1,k+1} & \cdots & c_{1n} \\ 0 & ||\mathbf{a}^{(2)}|| & c_{23} & \cdots & c_{2,k-1} & c_{2k} & c_{2,k+1} & \cdots & c_{2n} \\ 0 & 0 & ||\mathbf{a}^{(3)}|| & \cdots & c_{3,k-1} & c_{3k} & c_{3,k+1} & \cdots & c_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & ||\mathbf{a}^{(k-1)}|| & c_{k-1,k} & c_{k-1,k+1} & \cdots & c_{k-1,n} \\ 0 & 0 & 0 & \cdots & 0 & ||\mathbf{a}^{(k)}|| & c_{k,k+1} & \cdots & c_{k,n} \\ 0 & 0 & 0 & \cdots & 0 & 0 & a_{k+1,k+1} & \cdots & a_{k+1,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 0 & 0 & a_{n,k+1} & \cdots & a_{n,n} \end{pmatrix}$$

После n шагов процесса система примет вид $R \cdot x = y$, где R - верхнетреугольная матрица. $R = A^{(n)} = U^{(n)}U^{(n-1)} \cdots U^{(1)}A$, $y = b^{(n)} = U^{(n)}U^{(n-1)} \cdots U^{(1)}b$. Полученную систему можно решить используя обратный ход метода Гаусса.

Обратный ход последовательно проходит по всем строкам в порядке убывания их номеров (от n до 1). Для каждой такой строки производится вычисление соответствующей координаты вектора x по следующим формулам:

$$x_n = \frac{y_n}{r_{nn}},$$
$$x_i = \frac{y_i - \sum\limits_{j=i+1}^{n} r_{ij}x_j}{r_{ii}}, i = \overline{n-1, 1}$$

4. Запуск и компиляция программы

Запуск и компиляция программ проводились на вычислительном комплексе IBM Polus, исходный код программы находился в файле `parallel_algorithm.c`.

4.1. OpenMP

1) Компиляция:
   `xlc_r -Wall -Werror -qsmp=omp slae_generation.c  -o main -lm`

2) Запуск:
   Для постановки задач использовался специально разработанный скрипт mpisubmit.pl.
   `mpisubmit.pl -t[число потоков] ./main[параметры исполняемого файла]`

Отладка и тестирование проводились на своем компьютере.

1) Компиляция:
   ```
   CFLAGS = -Wall -Werror -O0 -fopenmp -lpthread
   gcc $(CFLAGS) parallel_algorithm.c -o main -lm -fsanitize=address
   ```

2) Запуск:
   `./main[параметры исполняемого файла]`

4.2. MPI

1) Компиляция:
   `mpixlc -Wall -Werror parallel_algorithm.c -o main -lm`

2) Запуск:
   Для постановки задач использовался специально разработанный скрипт mpisubmit.pl.
   `mpisubmit.pl -p[число процессов] ./main[параметры исполняемого файла]`

Отладка и тестирование проводились на своем компьютере.

1) Компиляция:
   ```
   CFLAGS = -Wall -Werror -O0
   mpicc $(CFLAGS) parallel_algorithm.c -o main -lm -fsanitize=address
   ```

2) Запуск:
   `mpirun -np[число процессов] ./main[параметры исполняемого файла]`

5. Результаты замеров времени выполнения

Size	500		1000	
$Type(T_{num})$	`MPI(1)`	`OpenMP(1)`	`MPI(1)`	`OpenMP(1)`
T_1	6.871283e-01	1.253363e-01	5.588139e+00	9.410177e-01
T_2	1.719333e-03	8.506667e-04	5.364667e-03	2.321667e-03
T_{all}	6.888477e-01	1.261870e-01	5.593504e+00	9.433393e-01
Sol_{norm}	0.000000e+00	1.023740e-10	0.000000e+00	7.190720e-10
Res_{norm}	1.873230e-07	3.229460e-09	3.436420e-07	9.689020e-09
$Type(T_{num})$	`MPI(2)`	`OpenMP(2)`	`MPI(2)`	`OpenMP(2)`
T_1	3.871730e-01	7.659967e-02	2.739257e+00	5.015967e-01
T_2	2.214000e-03	2.217000e-03	4.804333e-03	5.073667e-03
T_{all}	3.893870e-01	7.881667e-02	2.744062e+00	5.066703e-01
Sol_{norm}	0.000000e+00	1.289760e-10	0.000000e+00	7.371390e-09
Res_{norm}	7.740070e-08	2.703250e-09	4.699960e-07	8.091670e-09
$S(T_{num})$	1.769057e+00	1.601019e+00	2.038403e+00	1.861840e+00
$E(T_{num})$	8.845283e-01	8.005096e-01	1.019202e+00	9.309202e-01
$Type(T_{num})$	`MPI(4)`	`OpenMP(4)`	`MPI(4)`	`OpenMP(4)`
T_1	2.416713e-01	5.784100e-02	1.406691e+00	3.265320e-01
T_2	2.396667e-03	2.659667e-03	4.282667e-03	5.513667e-03
T_{all}	2.440680e-01	6.050067e-02	1.410974e+00	3.320457e-01
Sol_{norm}	0.000000e+00	0.000000e+00	0.000000e+00	1.210300e-10
Res_{norm}	8.679310e-08	2.556380e-09	3.594950e-07	7.713900e-09
$S(T_{num})$	2.822360e+00	2.085713e+00	3.964287e+00	2.840993e+00
$E(T_{num})$	7.055899e-01	5.214281e-01	9.910716e-01	7.102482e-01
$Type(T_{num})$	`MPI(8)`	`OpenMP(8)`	`MPI(8)`	`OpenMP(8)`
T_1	2.008533e-01	5.154833e-02	7.733993e-01	2.763010e-01
T_2	2.746667e-03	3.247333e-03	4.073667e-03	6.540000e-03
T_{all}	2.036000e-01	5.479567e-02	7.774730e-01	2.828410e-01
Sol_{norm}	0.000000e+00	0.000000e+00	0.000000e+00	1.732760e-09
Res_{norm}	7.604410e-08	2.514710e-09	3.649170e-07	7.165570e-09
$S(T_{num})$	3.383338e+00	2.302865e+00	7.194467e+00	3.335228e+00
$E(T_{num})$	4.229173e-01	2.878581e-01	8.993084e-01	4.169035e-01
$Type(T_{num})$	`MPI(16)`	`OpenMP(16)`	`MPI(16)`	`OpenMP(16)`
T_1	1.703210e-01	5.222367e-02	5.113233e-01	1.982547e-01
T_2	4.041667e-03	4.163333e-03	4.648333e-03	8.575000e-03
T_{all}	1.743627e-01	5.638700e-02	5.159717e-01	2.068297e-01
Sol_{norm}	0.000000e+00	0.000000e+00	0.000000e+00	1.824540e-09
Res_{norm}	8.033700e-08	2.515810e-09	3.501280e-07	7.252840e-09
$S(T_{num})$	3.950660e+00	2.237874e+00	1.084072e+01	4.560948e+00
$E(T_{num})$	2.469163e-01	1.398671e-01	6.775450e-01	2.850592e-01

Size	2000		3000	
$Type(T_{num})$	MPI(1)	OpenMP(1)	MPI(1)	OpenMP(1)
T_1	4.643742e+01	7.929052e+00	1.599954e+02	2.758517e+01
T_2	2.103133e-02	6.271000e-03	5.493233e-02	1.161200e-02
T_{all}	4.645845e+01	7.935323e+00	1.600504e+02	2.759678e+01
Sol_{norm}	0.000000e+00	2.813760e-10	0.000000e+00	2.432480e-09
Res_{norm}	1.771470e-06	2.693160e-08	6.263430e-06	4.946880e-08
$Type(T_{num})$	MPI(2)	OpenMP(2)	MPI(2)	OpenMP(2)
T_1	2.296636e+01	3.931020e+00	7.953244e+01	1.374619e+01
T_2	1.463400e-02	1.102533e-02	3.289267e-02	1.807367e-02
T_{all}	2.298099e+01	3.942046e+00	7.956534e+01	1.376426e+01
Sol_{norm}	0.000000e+00	1.219450e-09	0.000000e+00	1.570870e-09
Res_{norm}	2.067800e-06	2.397690e-08	4.276880e-06	4.340320e-08
$S(T_{num})$	2.021603e+00	2.012996e+00	2.011559e+00	2.004959e+00
$E(T_{num})$	1.010802e+00	1.006498e+00	1.005779e+00	1.002480e+00
$Type(T_{num})$	MPI(4)	OpenMP(4)	MPI(4)	OpenMP(4)
T_1	1.109976e+01	2.181314e+00	3.846372e+01	7.218577e+00
T_2	1.049767e-02	1.207833e-02	2.045800e-02	1.948467e-02
T_{all}	1.111025e+01	2.193393e+00	3.848418e+01	7.238062e+00
Sol_{norm}	0.000000e+00	2.153600e-10	0.000000e+00	1.566540e-09
Res_{norm}	1.802590e-06	2.222400e-08	3.878630e-06	4.092040e-08
$S(T_{num})$	4.181583e+00	3.617830e+00	4.158861e+00	3.812731e+00
$E(T_{num})$	1.045396e+00	9.044576e-01	1.039715e+00	9.531827e-01
$Type(T_{num})$	MPI(8)	OpenMP(8)	MPI(8)	OpenMP(8)
T_1	5.689080e+00	1.825700e+00	2.004854e+01	4.740271e+00
T_2	8.441333e-03	1.396433e-02	1.654267e-02	2.187533e-02
T_{all}	5.697522e+00	1.839664e+00	2.006508e+01	4.762146e+00
Sol_{norm}	0.000000e+00	5.639510e-10	0.000000e+00	1.796760e-09
Res_{norm}	1.749150e-06	2.098650e-08	4.085013e-06	4.002760e-08
$S(T_{num})$	8.154151e+00	4.313462e+00	7.976563e+00	5.795031e+00
$E(T_{num})$	1.019269e+00	5.391828e-01	9.970703e-01	7.243788e-01
$Type(T_{num})$	MPI(16)	OpenMP(16)	MPI(16)	OpenMP(16)
T_1	3.066948e+00	1.091246e+00	1.145867e+01	3.690117e+00
T_2	8.284333e-03	1.743400e-02	1.553233e-02	2.637633e-02
T_{all}	3.075232e+00	1.108680e+00	1.147421e+01	3.716494e+00
Sol_{norm}	0.000000e+00	4.433740e-10	0.000000e+00	1.691290e-09
Res_{norm}	1.842920e-06	2.127300e-08	4.088343e-06	3.801050e-08
$S(T_{num})$	1.510730e+01	7.157449e+00	1.394871e+01	7.425489e+00
$E(T_{num})$	9.442062e-01	4.473405e-01	8.717943e-01	4.640930e-01

Size	5000	
$Type(T_{num})$	`MPI(1)`	`OpenMP(1)`
T_1	8.938382e+02	3.282270e+02
T_2	1.694940e-01	2.840400e-02
T_{all}	8.940077e+02	3.282554e+02
Sol_{norm}	0.000000e+00	7.115230e-10
Res_{norm}	1.475310e-05	1.062330e-07
$Type(T_{num})$	`MPI(2)`	`OpenMP(2)`
T_1	3.834333e+02	1.649767e+02
T_2	8.819833e-02	3.505700e-02
T_{all}	3.835215e+02	1.650117e+02
Sol_{norm}	0.000000e+00	1.127420e-09
Res_{norm}	1.188690e-05	9.299140e-08
S(T_{num})	2.331049e+00	1.989285e+00
E(T_{num})	1.165525e+00	9.946426e-01
$Type(T_{num})$	`MPI(4)`	`OpenMP(4)`
T_1	1.859774e+02	8.414319e+01
T_2	4.930833e-02	3.441400e-02
T_{all}	1.860267e+02	8.417760e+01
Sol_{norm}	0.000000e+00	1.084650e-09
Res_{norm}	1.102550e-05	8.797520e-08
S(T_{num})	4.805803e+00	3.899557e+00
E(T_{num})	1.201451e+00	9.748893e-01
$Type(T_{num})$	`MPI(8)`	`OpenMP(8)`
T_1	9.459825e+01	4.554694e+01
T_2	3.324500e-02	3.797000e-02
T_{all}	9.463149e+01	4.558491e+01
Sol_{norm}	0.000000e+00	7.401560e-10
Res_{norm}	1.200040e-05	8.723150e-08
S(T_{num})	9.447253e+00	7.200965e+00
E(T_{num})	1.180907e+00	9.001207e-01
$Type(T_{num})$	`MPI(16)`	`OpenMP(16)`
T_1	5.716643e+01	3.002337e+01
T_2	2.987400e-02	4.502533e-02
T_{all}	5.719631e+01	3.006840e+01
Sol_{norm}	0.000000e+00	1.835970e-09
Res_{norm}	1.182980e-05	8.366810e-08
S(T_{num})	1.563051e+01	1.091696e+01
E(T_{num})	9.769071e-01	6.823098e-01

6. Анализ эффективности

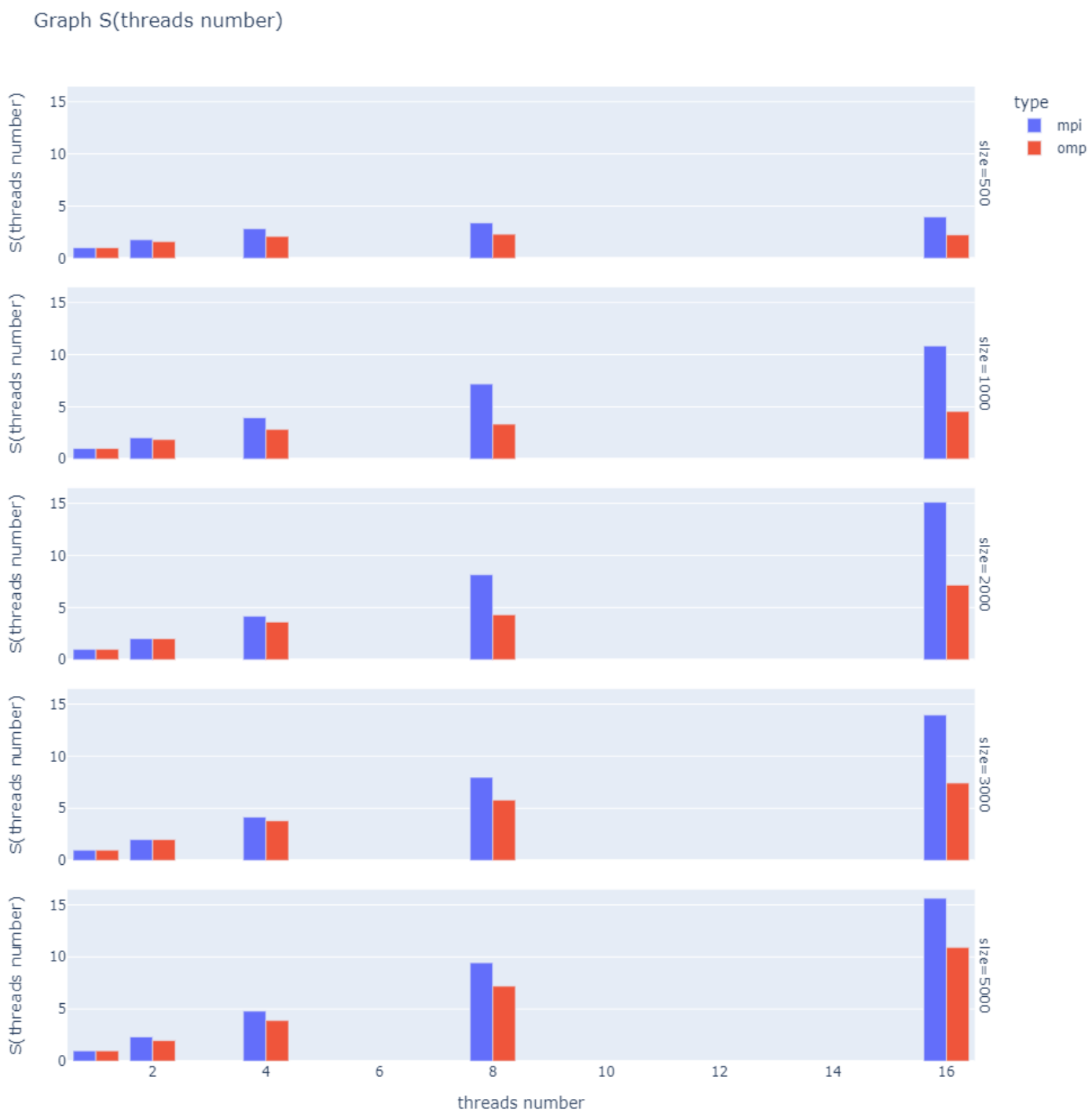

Рис. 1: S(threads number)

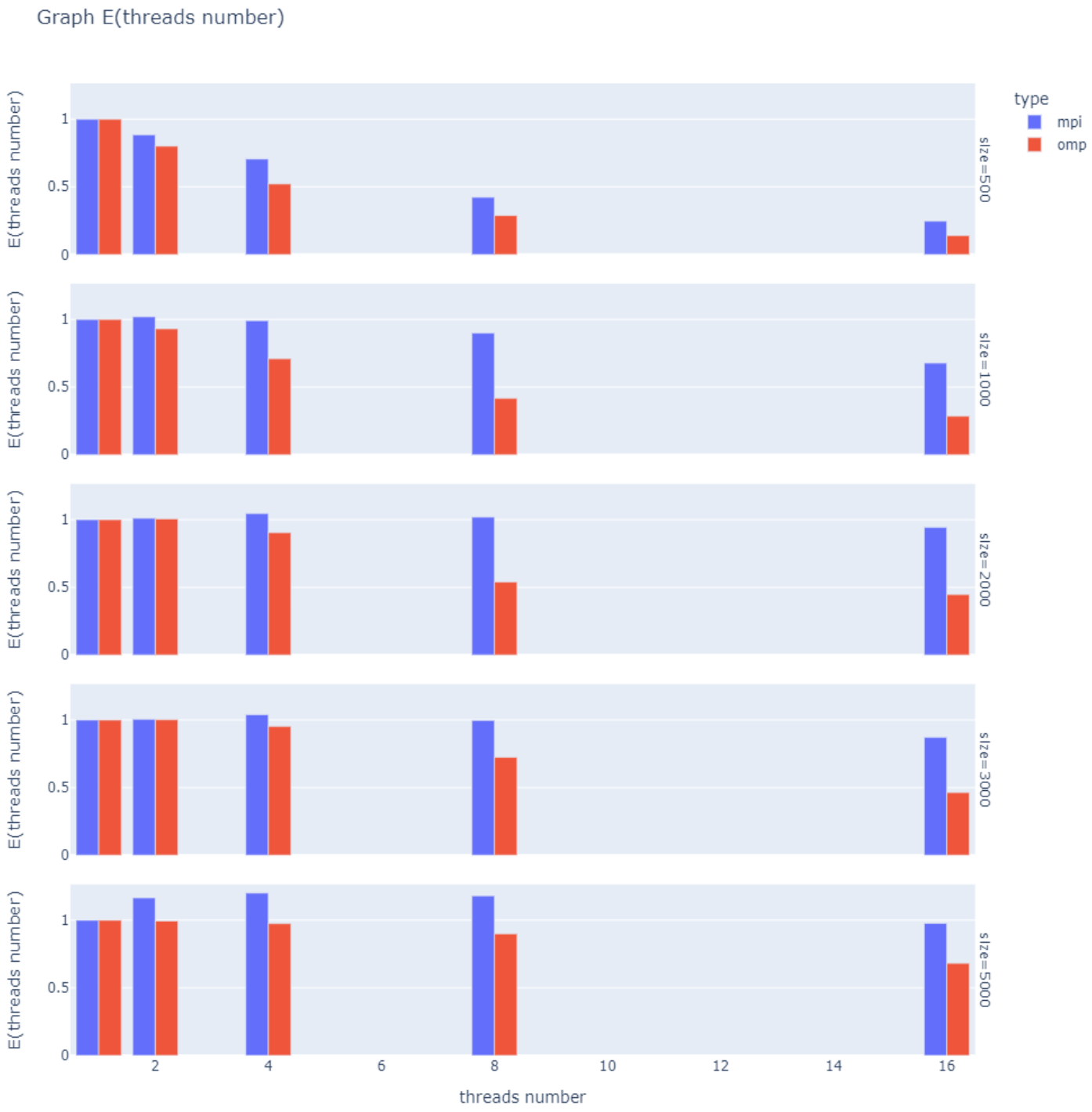

Рис. 2: E(threads number)

7. Теоретическая оценка времени выполнения

A - матрица размера $n \times n$, задача запускается на p процессах. Время выполнения сложения или умножения двух чисел равно τ_c , а время затрачиваемое на передачу одного числа от одного процесса к другому равно τ_s.

7.1. Распределение матрицы по процессам

Каждый процесс получает $\frac{n}{p}$ столбцов матрицы A, затрачиваемое время:

$$\frac{n}{p} * n \log_2(p) * \tau_s$$

7.2. Приведение матрицы к верхнетреугольному виду

Рассмотрим k - шаг алгоритма.

Для вычисления вектора w потребуется $(n-k)$ операций сложения, $(n-k)$ - умножения и $(n-k)$ - деления.

Сложность распространение вектора w по всем процессам составляет $(n-k)log_2(p)$ операций.

Для вычисления произведения матрицы отражения на подматрицу матрицы A размера $(n-k) \times (n-k)$ потребуется $2 * (n-k) * \frac{(n-k)}{p}$ операций сложения и столько же умножений.

Тогда затрачиваемое время всего процесса:

$$\sum_{k=1}^{n} (3 * (n-k) + 4 * (n-k) * \tfrac{(n-k)}{p}) * \tau_c + (n-k)log_2(p) * \tau_s$$

7.3. Оратный ход метода Гаусса

Рассмотрим k - шаг алгоритма.

Время вычисления значения x_k вектора результата составляет:

$$2 * \tfrac{n}{p} * \tau_c + (p) * \tau_c + log_2(p) * \tau_s$$

Тогда затрачиваемое время всего процесса:

$$\sum_{k=1}^{n} 2 * \tfrac{n}{p} * \tau_c + (p) * \tau_c + log_2(p) * \tau_s$$

7.4. Итоговое время

$$\frac{n*(8n^2+9np+6p^2-9p+4)*\tau_c}{6p} + \frac{n*(n+1)log_2(p)*\tau_s}{2} \approx$$
$$\approx (\tfrac{4n^3}{3p} + O(n^2/p)) * \tau_c + (\tfrac{n^2*log_2(p)}{2} + O(n)) * \tau_s$$