Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Отчёт по домашнему заданию по дисциплине "Программирование параллельных процессов"

Тема Распараллеливание с помощью MPI и OpenMP

Студент Варламова Е. А.

Группа ИУ7-33М

Оценка (баллы) _____

Преподаватели Ковтушенко А.П.

Москва — 2024 г.

СОДЕРЖАНИЕ

1 | Задание и описание алгоритма

1.1 Задание

Задание 16. Разработать программу вычисления матрицы обратной заданной на основе метода Жордана. Обосновать проектое решение (выбор алгоритма). Обеспечить равномерную загрузку процессоров. Результат вывести в текстовый файл. Исследовать зависимость времени счета от размерности задачи и количества процессоров.

1.2 Метод Жордана

Алгоритм Гаусса-Жордана (метод Жордана) нахождения обратной матрицы – это модификация метода Гаусса, который позволяет вычислить обратную матрицу A^{-1} для невырожденной квадратной матрицы A размера $n \times n$. Он выполняется путем одновременного применения элементарных преобразований строк к исходной матрице A и единичной матрице I размера $n \times n$, размещенной справа от A.

Шаги алгоритма:

1. Создание расширенной матрицы

   Создаем расширенную матрицу $[A|I]$ путем конкатенации исходной матрицы A и единичной матрицы I того же размера:

$$\left[\begin{array}{cccc|cccc} a_{11} & a_{12} & \cdots & a_{1n} & 1 & 0 & \cdots & 0 \\ a_{21} & a_{22} & \cdots & a_{2n} & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} & 0 & 0 & \cdots & 1 \end{array}\right]$$

2. Прямой ход (приведение к диагональному виду)

   Цель этого шага — преобразовать левую часть расширенной матрицы (A) в единичную матрицу I путем применения элементарных преобразований строк. Эти преобразования включают:

(a) **Обмен местами строк:** Строки можно менять местами.

(b) **Умножение строки на число:** Строку можно умножить на ненулевое число.

(c) **Прибавление к строке другой строки, умноженной на число:** К одной строке можно прибавить другую строку, умноженную на произвольное число.

Эти преобразования применяются одновременно к левой и правой частям расширенной матрицы. Процесс повторяется для каждой строки матрицы A.

На i-ом шаге:

(a) Выбираем максимальный элемент в столбце i и меняем строку с максимальным элементом со строкой i. Если $a_{ii} = 0$, то матрица вырождена.

(b) Делим i-ую строку на a_{ii} (для приведения ведущего элемента к 1).

(c) Прибавляем к каждой строке j, где $j \neq i$, i-ую строку, умноженную на $-a_{ji}$ (для обнуления элементов в i-ом столбце).

3. Обратная матрица

   После завершения прямого хода, левая часть расширенной матрицы будет единичной матрицей I, а правая часть будет содержать обратную матрицу A^{-1}:

$$\left[\begin{array}{cccc|cccc} 1 & 0 & \cdots & 0 & b_{11} & b_{12} & \cdots & b_{1n} \\ 0 & 1 & \cdots & 0 & b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 & b_{n1} & b_{n2} & \cdots & b_{nn} \end{array}\right]$$

   где $B = [b_{ij}] = A^{-1}$.

4. Проверка: Для проверки результата можно выполнить умножение $A \times A^{-1}$ и $A^{-1} \times A$. Результатом должно быть единичная матрица I.

   Замечание: Если в процессе выполнения алгоритма на каком-либо шаге ведущий элемент равен нулю, и нельзя найти строку для обмена, то матрица A является вырожденной (сингулярной), и обратной матрицы не существует.

1.3 Предлагаемый метод параллелизации

Необходимо:

1. Разделить столбцы исходной матрицы между процессорами: пусть нулевой процессор обрабатывает столбцы $0, N_{proc}, 2 \cdot N_{proc}...$, первый – $1, N_{proc}+1, 2 \cdot N_{proc}+1...$ и т. д. Распределяем столбцы исходной и дополненной матрицы в соотвестствии с такой схемой с помощью MPI_Scatter на нулевом процессоре.

2. Теперь в цикле по столбцам матрицы необходимо делать следующее на каждом процессоре (прямой ход метода):

   (a) Если текущий обрабатываемый столбец матрицы принадлежит текущему процессору, то на этом процессоре найти максимальный элемент в столбце и его номер. Разослать этот столбец и номер максимума всем процессорам с помощью MPI_Bcast.

   (b) Все процессоры, получив столбец и максимум должны поменять во всех своих столбцах текущую строку (строку с номером текущего обрабатываемого столбца) со строкой с максимальным элементом. Далее каждый процессор делит во всех своих столбцах текущую строку на максимальный элемент. При этом все дейтствия он совершает над основной и дополненной матрицами

   (c) Далее все процессоры совершают вычисления, которые позволят обнулить все элементы ведущего столбца. При этом все действия он совершает над основной и дополненной матрицами. **Этот шаг можно выполнить в несколько потоков для увеличения производительности**.

3. Собрать с помощью MPI_Gather на нулевом процессоре все столбцы дополненной матрицы (это части обратной матрицы).

2 | Результаты исследования

Были проведены замеры времени со следующими параметрами:

— число узлов – от 1 до 10 с шагом 1;

— размер матрицы $(N * N)$, N менялось от 100 до 1000 с шагом 100, 2000, 3000, 5000.

— реализация – без параллелизации, параллелизация с помощью MPI, параллелизация с помощью MPI и OpenMP.

2.1 Обычная реализация

Был разработан не параллельный алгоритм. Результаты его работы представлены на рисунке 2.1.

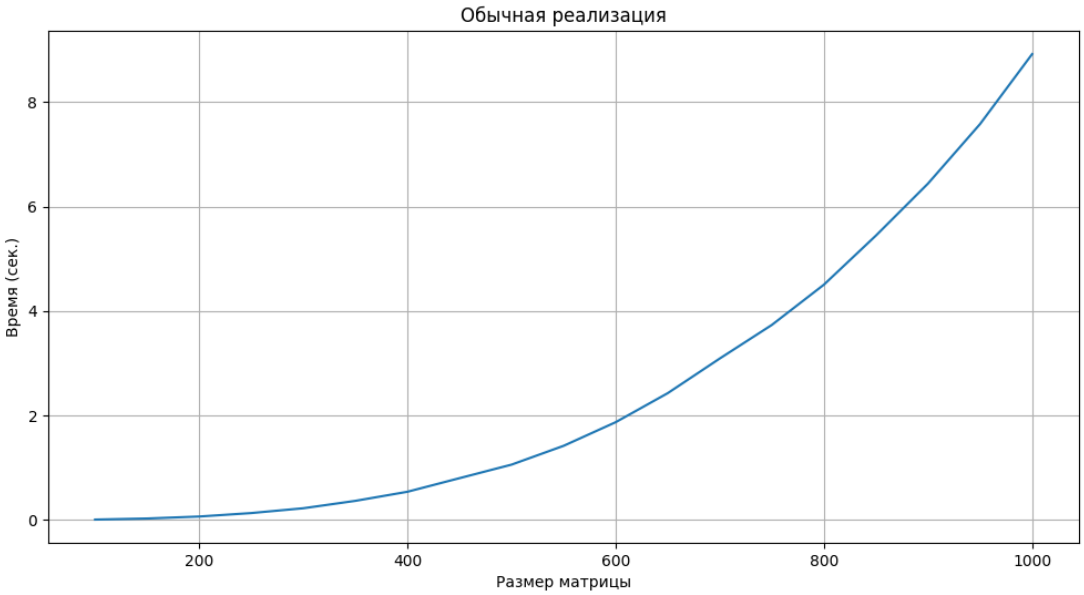

Рис. 2.1: Обычная реализация

Видим, что при размере матрицы 1000 время достигает 8.5 секунд.

2.2 Параллельная реализация с помощью MPI

В таблице 2.1 представлены результаты замеров и расчёт эффективности и ускорения. Видим, что при размере матрицы 1000 и одном процессоре время достигает 9.5 секунд, что эквивалентно обычной реализации с учётом накладных расходов.

Таблица 2.1: Реализация MPI

P	П/П	1	2	3	4	5	6	7	8	9	10
100	T	0.009	0.022	0.031	0.130	0.056	0.246	0.086	0.131	0.105	0.117
	A	1.000	0.412	0.291	0.070	0.162	0.037	0.106	0.070	0.086	0.077
	E	1.000	0.206	0.097	0.017	0.032	0.006	0.015	0.009	0.010	0.008
200	T	0.074	0.062	0.046	0.069	0.039	0.128	0.039	0.040	0.036	0.038
	A	1.000	1.192	1.595	1.069	1.886	0.575	1.882	1.825	2.025	1.960
	E	1.000	0.596	0.532	0.267	0.377	0.096	0.269	0.228	0.225	0.196
300	T	0.250	0.164	0.119	0.124	0.094	0.182	0.081	0.109	0.081	0.086
	A	1.000	1.524	2.106	2.018	2.652	1.372	3.074	2.281	3.074	2.909
	E	1.000	0.762	0.702	0.505	0.530	0.229	0.439	0.285	0.342	0.291
400	T	0.660	0.349	0.248	0.237	0.176	0.166	0.151	0.173	0.140	0.130
	A	1.000	1.894	2.666	2.785	3.749	3.967	4.376	3.815	4.715	5.094
	E	1.000	0.947	0.889	0.696	0.750	0.661	0.625	0.477	0.524	0.509
500	T	1.186	0.641	0.456	0.366	0.316	0.282	0.262	0.239	0.219	0.221
	A	1.000	1.852	2.602	3.239	3.756	4.202	4.523	4.960	5.406	5.365
	E	1.000	0.926	0.867	0.810	0.751	0.700	0.646	0.620	0.601	0.537
600	T	2.004	1.082	0.745	0.600	0.500	0.456	0.415	0.377	0.341	0.318
	A	1.000	1.852	2.688	3.338	4.007	4.395	4.830	5.319	5.882	6.297
	E	1.000	0.926	0.896	0.835	0.801	0.733	0.690	0.665	0.654	0.630
700	T	3.196	1.665	1.183	0.965	0.781	0.670	0.614	0.559	0.517	0.503
	A	1.000	1.920	2.702	3.311	4.094	4.770	5.206	5.714	6.183	6.352
	E	1.000	0.960	0.901	0.828	0.819	0.795	0.744	0.714	0.687	0.635
800	T	4.856	2.515	1.793	1.422	1.215	1.354	1.305	1.064	1.062	1.031
	A	1.000	1.931	2.708	3.416	3.996	3.588	3.722	4.564	4.574	4.709
	E	1.000	0.966	0.903	0.854	0.799	0.598	0.532	0.571	0.508	0.471
900	T	6.913	3.576	2.537	1.958	1.665	1.451	1.315	1.178	1.075	1.053
	A	1.000	1.933	2.725	3.530	4.153	4.764	5.257	5.868	6.429	6.568
	E	1.000	0.967	0.908	0.883	0.831	0.794	0.751	0.733	0.714	0.657
1000	T	9.437	4.831	3.480	2.674	2.231	1.921	1.716	1.632	1.425	1.315
	A	1.000	1.953	2.712	3.529	4.230	4.911	5.498	5.784	6.625	7.175
	E	1.000	0.977	0.904	0.882	0.846	0.819	0.785	0.723	0.736	0.718
2000	T	77.109	38.812	26.773	19.949	16.269	13.955	12.400	10.999	9.495	9.258
	A	1.000	1.987	2.880	3.865	4.740	5.525	6.218	7.011	8.121	8.329
	E	1.000	0.993	0.960	0.966	0.948	0.921	0.888	0.876	0.902	0.833
3000	T	255.889	129.530	87.857	67.072	53.526	46.474	39.062	35.160	31.380	30.405
	A	1.000	1.976	2.913	3.815	4.781	5.506	6.551	7.278	8.155	8.416
	E	1.000	0.988	0.971	0.954	0.956	0.918	0.936	0.910	0.906	0.842
5000	T	1173.671	591.561	403.794	303.560	244.541	210.244	175.006	158.841	141.443	126.934
	A	1.000	1.984	2.907	3.866	4.799	5.582	6.706	7.389	8.298	9.246
	E	1.000	0.992	0.969	0.967	0.960	0.930	0.958	0.924	0.922	0.925

Условные обозначения:

— Р – размер;

— П/П – показатель/процессоры;

— Т – время;

— А – ускорение;

— Е – эффективность.

Расчёты:

— ускорение рассчитано как отношение времени на одном процессоре ко времени (на 1, 2, ... 10 процессорах);

— эффективность рассчитана как отношение ускорения к количеству использованных процессоров.

На рисунках 2.2-2.4 представлены графики зависимости времени, ускорения и эффективности от количества процессоров для разных размеров матрицы.

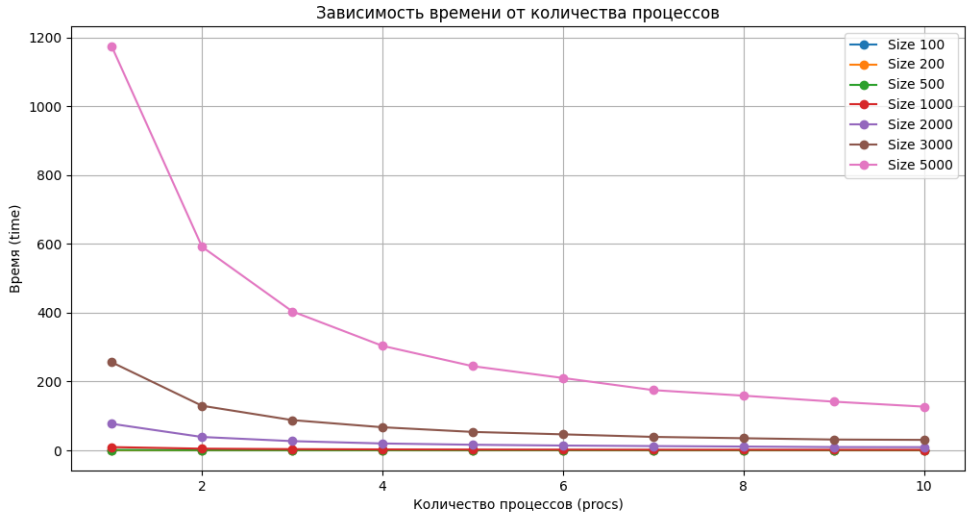

Рис. 2.2: MPI-реализация: график времени от количества процессоров

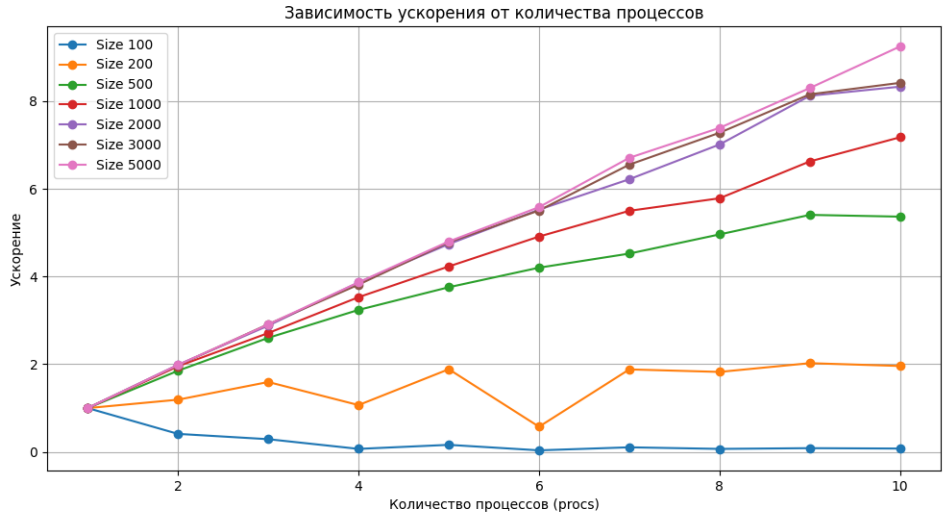

Рис. 2.3: MPI-реализация: график ускорения от количества процессоров

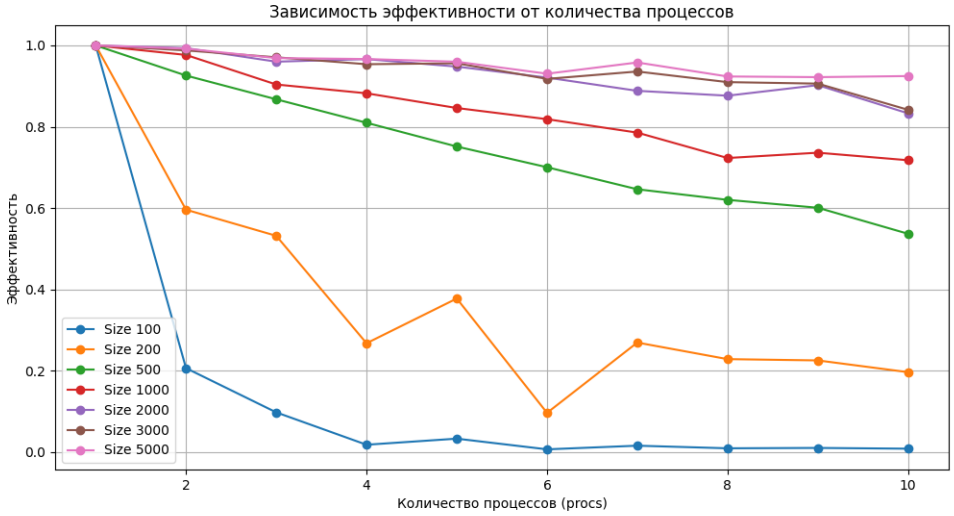

Рис. 2.4: MPI-реализация: график эффективности от количества процессоров

Видно, что на матрице размером 5000 эффективность параллелизации достигает 92%. А максимальное ускорение в 9.2 раза соответственно.

2.3 Openmp + MPI реализация

На размере матрицы 3000 были проведены исследования зависисмости времени исполнения от количества процессоров и потоков. Результаты представлены в таблице 2.2.

Условные обозначения: П – потоки, П/П – показатель/процессоры, Т – время, А – ускорение, Е – эффективность.

Таблица 2.2: Реализация OpenMP, размер матрицы 3000

П	П/П	1	2	3	4	5	6	7	8	9	10
1	Т	243.627	124.836	83.355	63.876	51.802	43.957	38.314	33.582	30.155	27.265
	А	1.000	1.952	2.923	3.814	4.703	5.542	6.359	7.255	8.079	8.936
	Е	1.000	0.976	0.974	0.954	0.941	0.924	0.908	0.907	0.898	0.894
2	Т	124.096	64.065	42.249	32.781	27.088	22.912	19.658	17.644	16.338	14.527
	А	1.963	3.803	5.766	7.432	8.994	10.633	12.393	13.808	14.911	16.771
	Е	0.982	0.951	0.961	0.929	0.899	0.886	0.885	0.863	0.828	0.839
3	Т	83.573	43.499	28.887	23.386	22.985	17.834	15.881	13.947	12.296	10.656
	А	2.915	5.601	8.434	10.418	10.599	13.661	15.341	17.468	19.813	22.862
	Е	0.972	0.933	0.937	0.868	0.707	0.759	0.731	0.728	0.734	0.762
4	Т	63.843	33.283	22.826	19.597	22.747	14.825	15.701	13.795	10.884	9.208
	А	3.816	7.320	10.673	12.432	10.710	16.434	15.517	17.660	22.384	26.457
	Е	0.954	0.915	0.889	0.777	0.536	0.685	0.554	0.552	0.622	0.661

Расчёты:

— ускорение рассчитано как отношение времени на одном процессоре с одним потоком ко времени (на 1, 2, ... 10 процессорах);

— эффективность рассчитана как отношение ускорения к количеству использованных процессоров, умноженных на количество потоков.

На рисунках 2.5-2.7 представлены графики зависимости времени, ускорения и эффективности от количества процессоров для разного количества потоков для матрицы размером $3000 * 3000$.

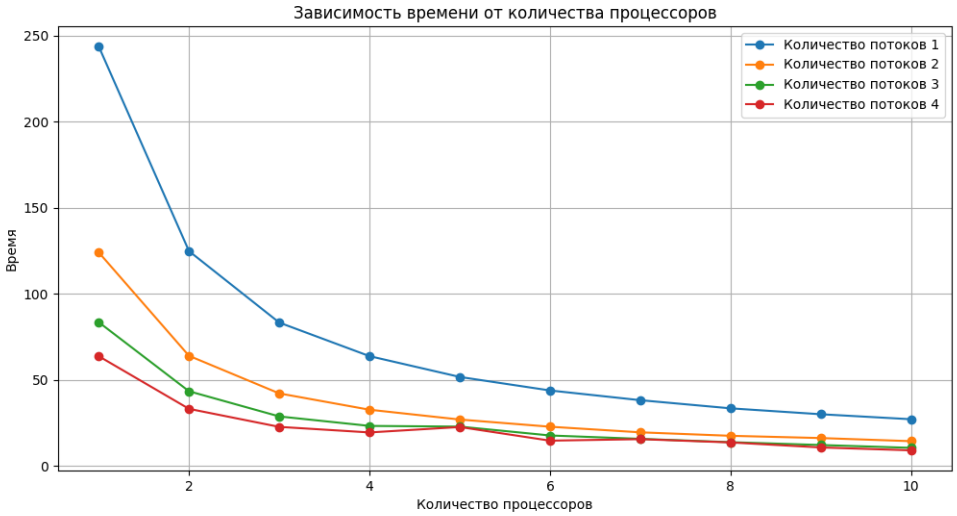

Рис. 2.5: OpenMP-реализация: график времени от количества процессоров

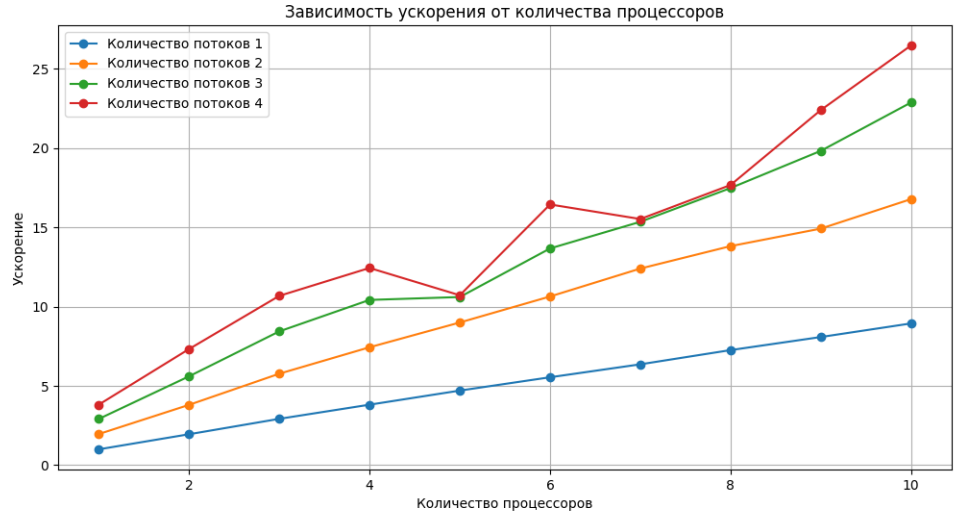

Рис. 2.6: OpenMP-реализация: график ускорения от количества процессоров

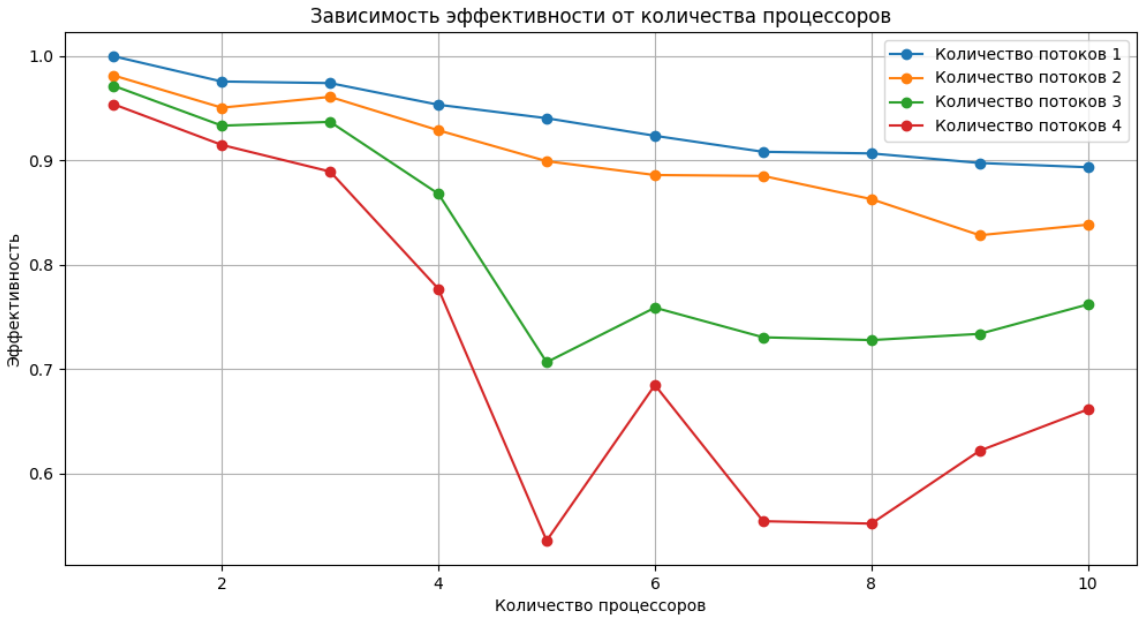

Рис. 2.7: OpenMP-реализация: график эффективности от количества процессоров

Видно, что на матрице размером 3000 максимальное ускорение (для 4 потоков и 10 ядер) составляет 26.4 раза, эффективность параллелизации при этом 60%.

ЗАКЛЮЧЕНИЕ

В ходе выполнения работы был реализован и исследован метод Жордана для поиска обратной матрицы в следующих конфигурациях:

1. Обычная реализация;
2. Параллельная реализация с помощью MPI;
3. Параллельная реализация с помощью MPI и Openmp.

Было выяснено, что на больших размерах матрицы (3000) максимальное ускорение – на 10 ядрах и 4 потоках – составляет 26.4 раза. При этом эффективность параллелизации – 60%.