Содержание

Цель работы

Освоить технику использования стандартов OpenMP и MPI в рамках решения задачи релаксации, используя итерационный метод Якоби.

Постановка задачи

Провести коррекцию намеренно допущенных ошибок в условии задачи с целью оценки уровня навыков и знаний студентов в определении необходимого метода релаксации и улучшении производительности.

Разработать нескольких программных версий, внедряющих параллельные вычисления для оптимизации и ускорения указанного метода.

Алгоритм

Итерационный метод Якоби для параллельных вычислений в частном случае (по условию $\omega = 1, f = 0$) представляет собой следующий алгоритм:

1. Устанавливается заданная точность и максимальное число итераций.
2. Создается вспомогательная матрица для хранения значений, полученных на предыдущей итерации.
3. Запускается итеративный процесс модификации матрицы, который завершается при достижении установленной точности или после выполнения заданного максимального числа итераций.
4. Внутри итерационного цикла осуществляется заполнение матрицы средними значениями соответствующих элементов из вспомогательной матрицы.

Алгоритм был улучшен путем исключения дополнительного цикла для копирования значений матриц или операции swap в отношении указателей на матрицы. Вместо этого определение вспомогательной матрицы осуществляется на основе четности итерации, что способствует более эффективной реализации.

Методы решения и описание программ

- ## OpenMP: распределение витков циклов

Для параллельной работы программы потребовались следующие модификации исходного кода:

1. Изменить порядок работы с элементами матрицы: внешний цикл должен итерировать первую размерность, а внутренний – вторую, для оптимизации и ускорения работы программы
2. Локализовать или сделать общими переменные i, j, eps для избежания ошибок одновременного изменения
3. Изменить алгоритм итерации функции relax() согласно методу Якоби: значения для вычислений берутся из вспомогательной матрицы
4. Создать директиву для циклов в функции init():
   `#pragma omp parallel for schedule(static) collapse(2)`
   `schedule(static)` – оптимизация статического распределения итераций: каждый поток отвечает за определенный диапазон итераций, и эти диапазоны определены заранее
   `collapse(2)` – оптимизация объединения 2 вложенных циклов в 1 для целей параллельной обработки
5. Создать директиву для циклов в функции relax():
   `#pragma omp parallel for reduction(max:local_eps) schedule(static) collapse(2)`
   `reduction(max:local_eps)` – редукция операции max на переменной local_eps
6. Создать директиву для циклов в функции verify():
   `#pragma omp parallel for reduction(+:s) schedule(static) collapse(2)`
   `reduction(+:s)` - редукция операции сложения на переменной s

- **OpenMP: механизм задач**

Для реализации механизма задач потребовалось организовать вычисления так, чтобы каждый поток обрабатывал значения в определенном интервале строк, указанном в полученной задаче. В следствии данных обстоятельств было принято решение внести соответствующие изменения, в директивы были добавлены следующие настройки:

`single` – секция для выполнения только одним потоком

`task` – секция для создания задач

`shared` – создание общих переменных для всех потоков

`firstprivate` – создание локальных переменных для каждого потока

`atomic, critical` – обеспечение атомарности операций

Изначально каждому потоку определялась ровно одна задача. С этой особенностью программа работала не в той же степени эффективно, как предыдущая, поэтому было принято решение исследовать оптимальное количество создаваемых задач исходя из общего количества процессов и размерности матрицы.

Была замечена линейная зависимость оптимального количества задач **T** для **каждого** потока от размерности матрицы **N** при фиксированном числе потоков **P** (в данном случае **P = 20**):

Время выполнения программы OpenMP: механизм задач (сек)

N \ T	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
2000	0,13	0,12	0,11	0,10	**0,09**	0,11	0,13	0,17	0,19	0,24	0,43	0,75	1,03	1,39	2,07
4000	0,45	0,44	0,43	0,42	0,41	**0,40**	0,42	0,45	0,51	0,57	0,74	1,05	1,33	1,72	2,31
6000	1,12	1,07	1,02	0,95	0,94	0,93	**0,90**	0,95	0,99	1,03	1,11	1,27	1,59	2,23	2,78
8000	1,82	1,72	1,67	1,66	1,65	1,64	1,64	**1,61**	1,64	1,70	1,76	1,88	2,12	2,60	3,16
10000	3,28	3,07	2,93	2,84	2,61	2,55	2,53	2,52	**2,50**	3,53	3,59	2,65	2,77	3,01	3,49
12000	4,13	4,01	3,94	3,86	3,79	3,71	3,65	3,63	3,61	**3,57**	3,65	3,81	4,29	4,93	6,21
14000	5,71	5,60	5,48	5,33	5,26	5,17	5,09	5,06	5,02	5,01	**4,78**	5,07	5,52	6,04	6,68
16000	8,11	7,92	7,69	7,43	7,21	7,05	6,88	6,73	6,59	6,47	6,41	**6,27**	6,38	6,61	6,94
18000	10,32	9,92	9,68	9,37	9,04	8,89	8,75	8,61	8,47	8,34	8,28	8,22	**8,19**	8,34	8,56
20000	12,18	12,15	11,97	11,83	11,52	11,29	11,17	10,98	10,89	10,82	10,76	10,72	10,71	**10,70**	10,76

Также удалось обнаружить обратную пропорциональность между числом потоков **P** и оптимальным количеством задач **T** для **каждого** из них при фиксированном **N** (в данном случае **N = 2000**):

Время выполнения программы OpenMP: механизм задач (сек)

P \ T	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
10	0,153	0,152	0,134	0,123	0,121	0,116	0,114	0,113	0,112	0,112	0,111	**0,109**	0,112	0,114	0,117	0,116	0,117	0,117	0,123	0,127
20	0,129	0,128	0,106	0,104	0,101	**0,100**	0,105	0,106	0,107	0,110	0,114	0,115	0,112	0,116	0,114	0,114	0,122	0,118	0,134	0,209
40	0,145	0,134	**0,129**	0,137	0,234	0,318	0,386	0,480	0,540	0,625	0,703	0,770	0,825	1,010	1,197	1,336	1,577	1,864	2,023	2,124

- Следовательно, приходим к выводу: $\boldsymbol{P \times T = const}$ при фиксированном **N**
- Но также величина $\boldsymbol{P \times T}$ является **общим** количеством задач (***tasks_num***)
- Значит **общее** оптимальное количество задач зависит только от размерности матрицы **N**
- Нас интересует именно величина ***tasks_num***, поскольку задачи в программе OpenMP распределяются самостоятельно между всеми потоками
- Уже была проведена работа по вычислению оптимального количества задач для **каждого** потока при **P = 20** для матриц различной размерности
- Значит можем вычислить **общее** оптимальное количество задач, просто домножив полученный результат на **P = 20**:

	2000	4000	6000	8000	10000	12000	14000	16000	18000	20000
T	5	6	7	8	9	10	11	12	13	14
$T \times P$	100	120	140	160	180	200	220	240	260	280

- Получаем следующую зависимость:

$$\boldsymbol{tasks\_num = \frac{N}{100} + 80}$$

Следует отметить важность данного наблюдения, благодаря которому удалось уменьшить время выполнения программы в среднем на 10-15%.

- **MPI**

В данной реализации алгоритма осуществляются параллельные вычисления на всех этапах. Каждый процесс программы обрабатывает только те строки матрицы, которые ему выделены. С целью оптимизации производительности значительно сокращен обмен данными между процессами: передаются лишь граничные строки каждому соседу, а также актуальные значения достигнутой точности. В заключительной части программы реализована редукция суммы верификации к главному процессу.

Тесты и сбор статистики

С помощью различных тестов мной была проверена корректность работы каждой версии программы, и я приступил к исследованию эффективности полученных параллельных программ на суперкомпьютере Polus.

Необходимо было сравнить время выполнения программ в зависимости от:

- Размерности обрабатываемых данных (N = 2000, 4000, 6000, 8000, 10000, 12000, 14000, 16000, 18000, 20000)
- Количества потоков/процессов (P = 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 20, 40, 60, 80, 100, 120, 140, 160)
- Различных оптимизаций компилятора (-O0, -O1, -O2, -O3, -Ofast)

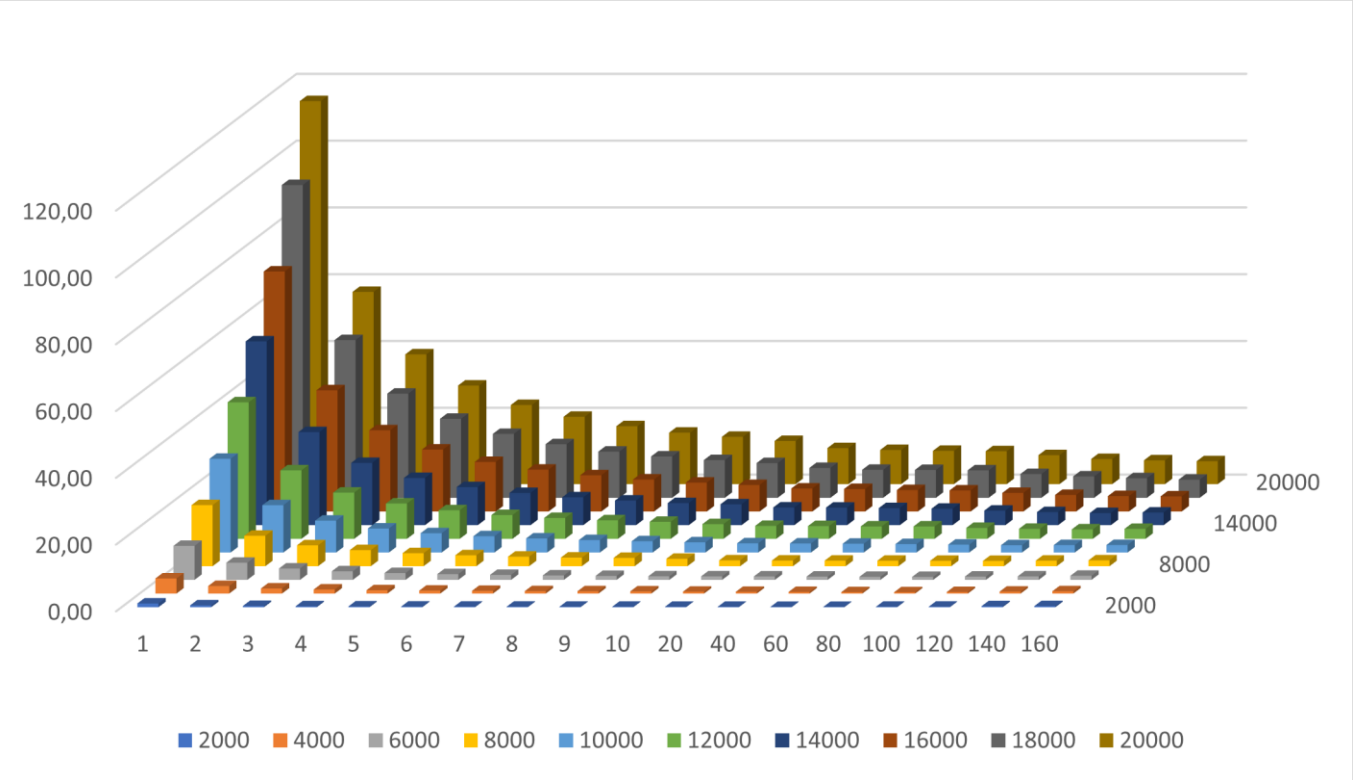
Тестирование программ OpenMP производилось с помощью следующих команд:

- gcc -std=c99 -Wall -fopenmp {-O0, -O1, -O2, -O3, -Ofast} {for.c, task.c}
- OMP_NUM_THREADS=P ./a.out

Тестирование программы MPI производилось с помощью следующих команд:

- mpicc -std=c99 -Wall {-O0, -O1, -O2, -O3, -Ofast} mpi.c
- mpirun -np P ./a.out

Время выполнения программы OpenMP: распределение витков циклов (сек)

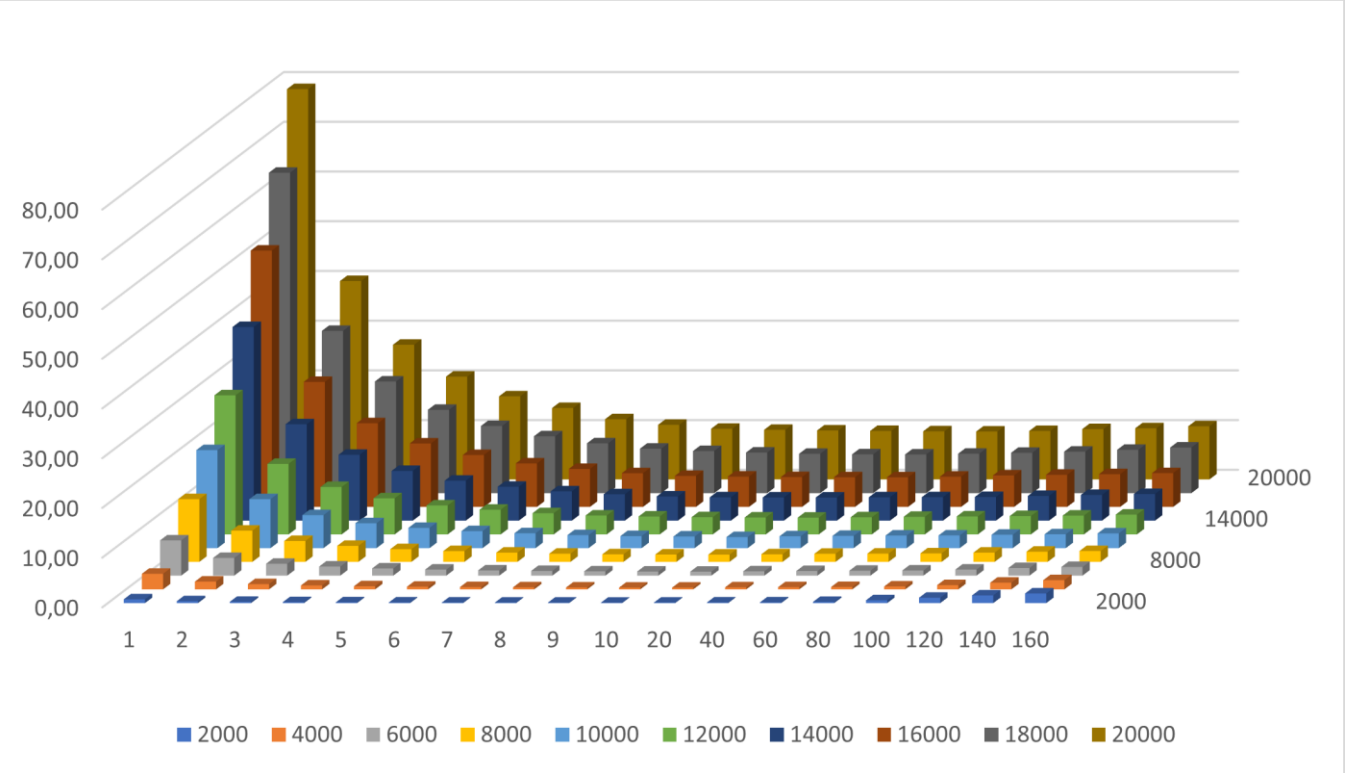


N \ P	1	2	3	4	5	6	7	8	9	10	20	40	60	80	100	120	140	160
2000	1,11	0,56	0,37	0,29	0,23	0,18	0,16	0,15	0,14	0,13	0,12	0,12	0,11	0,09	0,13	0,16	0,31	0,34
4000	4,52	2,26	1,52	1,21	0,98	0,87	0,81	0,71	0,68	0,64	0,56	0,50	0,47	0,41	0,37	0,46	0,57	0,68
6000	10,19	5,13	3,41	2,56	2,04	1,72	1,49	1,32	1,18	1,08	1,04	1,02	0,98	0,90	0,88	0,98	1,04	1,22
8000	18,29	9,11	6,26	4,82	3,90	3,27	2,86	2,57	2,49	2,22	1,74	1,69	1,66	1,64	1,61	1,58	1,65	1,74
10000	28,08	14,17	9,53	7,19	5,78	4,88	4,21	3,72	3,37	3,04	2,76	2,71	2,63	2,52	2,35	2,18	2,22	2,26
12000	40,89	20,52	13,84	10,58	8,56	7,12	6,26	5,55	5,10	4,41	3,88	3,82	3,72	3,71	3,27	3,00	2,83	2,99
14000	55,03	27,86	18,64	14,13	11,40	9,64	8,33	7,36	6,61	6,25	5,30	5,25	5,11	4,93	4,36	3,92	3,64	3,78
16000	71,87	36,31	24,32	18,55	14,84	12,54	10,86	9,58	8,66	7,95	6,88	6,73	6,41	6,31	5,56	4,96	4,61	4,53
18000	93,67	47,23	31,19	23,66	19,13	16,04	13,84	12,37	11,30	10,40	8,95	8,38	8,33	8,23	7,12	6,40	5,88	5,44
20000	114,70	57,57	38,86	29,53	23,69	20,18	17,35	15,41	14,17	12,93	10,79	10,22	9,96	9,85	8,68	7,54	7,13	6,86

С увеличением числа потоков наблюдается повышение производительности. Тем не менее, существует ограничение, и использование избыточного числа потоков для обработки относительно небольших объемов данных оказывается неоптимальным. Это связано с необходимостью затрат на поддержание синхронизации между потоками, что влечет за собой увеличение использования ресурсов. В результате количество задач, порученных каждому потоку, уменьшается до такой степени, что выделение отдельного процесса для них становится нецелесообразным. Для решения задач релаксации с матрицами:

- Размерности N = 2000 наиболее эффективное количество процессов P = 80
- Размерности N = 4000, 6000 наиболее эффективное количество процессов P = 100
- Размерности N = 8000, 10000 наиболее эффективное количество процессов P = 120
- Размерности N = 12000, 14000 наиболее эффективное количество процессов P = 140
- Размерности N = 16000, 18000, 20000 наиболее эффективное количество процессов P = 160

Время выполнения программы OpenMP: механизм задач (сек)

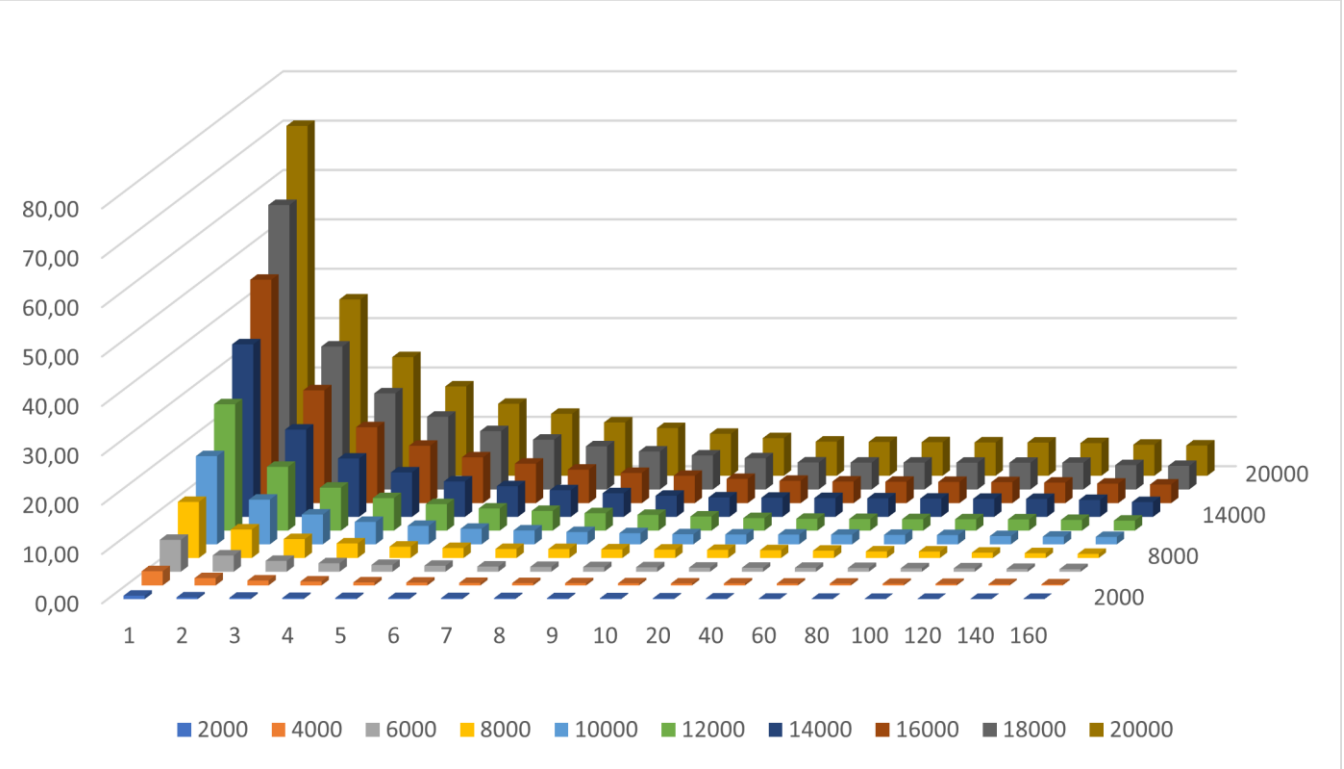


N \ P	1	2	3	4	5	6	7	8	9	10	20	40	60	80	100	120	140	160
2000	0,77	0,38	0,26	0,20	0,16	0,13	0,12	0,11	0,10	0,09	0,11	0,12	0,15	0,24	0,57	1,08	1,55	1,94
4000	3,13	1,57	1,06	0,79	0,63	0,53	0,46	0,41	0,37	0,36	0,35	0,41	0,45	0,48	0,58	0,85	1,34	1,85
6000	7,08	3,55	2,39	1,82	1,44	1,21	1,05	0,94	0,85	0,82	0,79	0,86	0,92	0,98	1,03	1,21	1,53	1,74
8000	12,62	6,27	4,20	3,22	2,55	2,17	1,86	1,66	1,52	1,49	1,46	1,53	1,63	1,66	1,72	1,86	2,06	2,18
10000	19,71	9,85	6,64	5,01	4,08	3,42	2,99	2,62	2,43	2,34	2,23	2,39	2,47	2,53	2,55	2,70	2,79	2,96
12000	27,98	14,19	9,53	7,25	5,83	4,97	4,26	3,80	3,60	3,46	3,40	3,36	3,44	3,54	3,61	3,68	3,78	3,99
14000	38,96	19,39	13,25	10,00	8,04	6,82	5,90	5,30	4,87	4,69	4,66	4,65	4,70	4,72	4,76	4,97	5,17	5,36
16000	51,56	25,14	16,84	12,76	10,43	8,74	7,64	6,78	6,24	6,11	6,01	5,97	5,95	6,09	6,34	6,45	6,59	6,80
18000	64,44	32,64	22,45	16,77	13,51	11,44	10,02	8,94	8,47	8,16	7,91	7,81	7,78	7,93	8,12	8,35	8,71	9,18
20000	78,51	39,90	27,09	20,67	16,71	14,36	12,11	10,98	10,19	9,98	9,85	9,75	9,67	9,63	9,75	10,12	10,29	10,70

Данная реализация требует существенно меньше времени для выполнения аналогичных задач по сравнению с предыдущей версией, при условии использования не более 40 потоков. Также программа выделяется своей способностью достичь оптимальной производительности при использовании менее значительного количества потоков, в пределе от 20 до 80, что представляет собой значительное отличие от предыдущей реализации. Для решения задач релаксации с матрицами:

- Размерности N = 2000 наиболее эффективное количество процессов P = 10
- Размерности N = 4000, 6000, 8000, 10000 наиболее эффективное количество процессов P = 20
- Размерности N = 12000, 14000 наиболее эффективное количество процессов P = 40
- Размерности N = 16000, 18000 наиболее эффективное количество процессов P = 60
- Размерности N = 20000 наиболее эффективное количество процессов P = 80

Время выполнения программы MPI (сек)



N \ P	1	2	3	4	5	6	7	8	9	10	20	40	60	80	100	120	140	160
2000	0,70	0,36	0,28	0,24	0,23	0,22	0,21	0,20	0,19	0,17	0,15	0,12	0,09	0,07	0,06	0,06	0,05	0,06
4000	2,88	1,49	1,00	0,80	0,61	0,57	0,51	0,47	0,45	0,42	0,40	0,39	0,37	0,36	0,33	0,31	0,26	0,22
6000	6,46	3,29	2,19	1,72	1,38	1,21	1,05	0,98	0,93	0,89	0,83	0,81	0,79	0,76	0,70	0,68	0,58	0,53
8000	11,37	5,78	3,84	2,96	2,31	2,03	1,83	1,81	1,75	1,70	1,65	1,58	1,49	1,33	1,32	1,08	0,98	0,84
10000	17,87	9,05	6,04	4,53	3,73	3,15	2,81	2,48	2,26	2,04	1,99	1,97	1,95	1,91	1,86	1,67	1,56	1,50
12000	25,59	12,92	8,70	6,59	5,36	4,47	3,98	3,52	3,19	2,86	2,53	2,39	2,34	2,31	2,29	2,27	2,16	2,03
14000	34,96	17,68	11,82	9,01	7,17	6,22	5,40	4,80	4,28	3,95	3,89	3,82	3,75	3,71	3,67	3,64	3,44	3,00
16000	45,26	22,86	15,38	11,63	9,28	8,00	6,80	6,12	5,55	4,92	4,53	4,39	4,34	4,31	4,25	4,17	3,96	3,78
18000	57,63	28,95	19,47	14,75	11,82	10,09	8,75	7,69	6,89	6,33	5,46	5,44	5,43	5,42	5,41	5,41	4,93	4,78
20000	70,87	35,72	24,04	18,10	14,58	12,56	10,80	9,67	8,50	7,62	6,92	6,85	6,80	6,75	6,73	6,61	6,30	6,13

Данная программа продемонстрировала наивысший уровень эффективности среди всех реализаций метода релаксации, проявляя более короткое время выполнения как для ограниченного, так и для высокого числа параллельных процессов. Оптимальным числом используемых процессов для максимальной эффективности оказался диапазон от 140 до 160.

Сравнение влияния различных оптимизаций на программу OpenMP: for

O \ P	1	2	3	4	5	6	7	8	9	10	20	40	60	80	100	120	140	160
O0	801,47	400,82	265,08	200,38	160,17	133,49	115,23	101,55	90,87	82,07	57,23	40,05	36,42	36,11	31,16	30,03	29,17	27,13
O1	274,87	136,12	92,00	69,15	56,26	46,84	40,83	36,07	32,78	29,67	22,55	16,84	16,24	15,47	13,71	11,47	10,37	9,83
O2	111,78	56,46	37,33	28,44	23,15	19,22	16,92	15,11	13,55	12,75	10,97	10,21	10,13	10,00	8,92	8,02	7,50	7,34
O3	118,14	60,91	40,36	31,57	24,30	20,57	17,97	15,86	14,66	13,37	11,18	10,52	10,38	10,22	9,13	7,93	7,25	7,05
FAST	114,70	57,57	38,86	29,53	23,69	20,18	17,35	15,41	14,17	12,93	10,79	10,22	9,96	9,85	8,68	7,54	7,13	6,86

Сравнение влияния различных оптимизаций на программу OpenMP: task

O \ P	1	2	3	4	5	6	7	8	9	10	20	40	60	80	100	120	140	160
O0	725,43	363,58	244,40	182,39	147,80	123,48	106,07	93,17	83,97	75,06	51,98	37,09	32,83	32,73	28,85	26,74	26,09	25,69
O1	246,12	122,78	82,85	62,87	50,80	42,33	36,29	31,88	28,60	26,55	18,80	14,21	13,42	12,63	11,98	9,93	9,07	8,69
O2	101,84	50,85	34,20	25,86	20,86	17,44	15,33	14,11	12,57	11,36	10,52	10,28	10,12	9,76	9,93	10,24	10,30	10,80
O3	108,37	55,97	37,17	28,92	23,63	19,80	16,79	15,38	13,86	12,90	11,50	10,40	10,22	9,81	10,01	10,31	10,45	10,83
FAST	78,51	39,90	27,09	20,67	16,71	14,36	12,11	10,98	10,19	9,98	9,85	9,75	9,67	9,63	9,75	10,12	10,29	10,70

Сравнение влияния различных оптимизаций на программу MPI

O \ P	1	2	3	4	5	6	7	8	9	10	20	40	60	80	100	120	140	160
O0	446,65	224,08	148,64	111,05	89,21	75,79	67,53	57,39	51,43	46,52	29,36	25,94	23,00	21,52	20,95	18,85	18,69	18,51
O1	261,21	130,97	87,04	65,20	52,39	44,54	38,25	33,33	30,07	26,54	15,00	11,06	9,75	8,55	8,27	8,23	7,11	7,40
O2	251,74	125,88	83,91	64,32	52,34	42,61	36,27	32,62	28,09	25,52	14,60	11,56	11,22	9,57	9,06	8,55	7,28	7,19
O3	253,29	126,48	84,61	65,19	64,23	43,33	37,17	32,75	28,77	25,80	14,95	11,64	11,53	10,50	9,33	8,26	7,05	6,87
FAST	70,87	35,72	24,04	18,10	14,58	12,56	10,80	9,67	8,50	7,62	6,92	6,85	6,80	6,75	6,73	6,61	6,30	6,13

При применении более высокого уровня оптимизации проявляется тенденция к ускоренному выполнению программы, однако наблюдаются несколько интересных особенностей:

- Оптимизация O1 демонстрирует более высокую эффективность по сравнению с O2, O3 и fast в случае множественного запуска процессов, как, например, в программе OpenMP с использованием механизма задач
- В большинстве сценариев оптимизация O2 оказывает более существенное воздействие на сокращение времени выполнения программы, чем O3, иногда даже в сравнении с fast, при уменьшенном количестве потоков, как, например, в программе OpenMP с использованием ветвления циклов

Выводы

Выполнена разработка параллельной реализации метода релаксации, в ходе которой были изучены технологии написания параллельных алгоритмов OpenMP и MPI. Проанализировано время выполнения алгоритмов, при этом выявлено, что технология OpenMP предоставляет удобство использования, сопряженное с значительным увеличением производительности. MPI, в свою очередь, представляет собой более низкоуровневую технологию: разработка MPI-программы обучает основам взаимодействия вычислительных узлов суперкомпьютера. Важно отметить, что MPI оптимизирована для многопроцессорных систем, и наилучшие результаты были достигнуты с использованием MPI-реализации, запущенной на максимальном числе вычислителей суперкомпьютера Polus