МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

имени М.В. Ломоносова

Факультет вычислительной математики и кибернетики

Практикум по курсу

"Суперкомпьютеры и параллельная обработка данных"

Разработка параллельной версии программы для транспонирования матриц

ОТЧЕТ

о выполненном задании

студентки 328 учебной группы факультета ВМК МГУ

Шелепнёвой Дарьи Дмитриевны

Москва, 2021 г.

Оглавление

1 Постановка задачи

Ставится задача транспонирования матрицы.

Дана матрица $A \in R^{n \times m}$, $n, m \in N$, требуется получить матрицу $B \in R^{m \times n}$, где $B = A_{ij}^T = A_{ji}$.

Требуется:

1. Реализовать параллельный алгоритм транспонирования матрицы с помощью технологии параллельного программирования OpenMP.
2. Исследовать масштабируемость полученных программ и построить графики зависимости времени выполнения программ от числа используемых потоков и объёма входных данных.

2 Описание алгоритма транспонирования матрицы

2.1 Основа: последовательный алгоритм

Простейший алгоритм транспонирования матрицы имеет следующий вид:

```
double **
transpose(double **a, int n, int m)
{
    double **b = malloc(sizeof(double *) * m);
    int i, j;
    for (i = 0; i < m; i++)
    {
        b[i] = malloc(sizeof(double) * n);
        for (j = 0; j < n; j++)
        {
            b[i][j] = a[j][i];
        }
    }
    return b;
}
```

Этот алгоритм имеет сложность $O(nm)$.

2.2 Параллельный алгоритм

Разбиваем задачу вычисления конечной матрицы на подзадачи по вычислению строк и распределяем их по потокам. Разбиение на вычисление отдельных полей не производим, т.к. размеры матриц при вычислениях и так будут на порядки превышать число потоков.

В OpenMP модификация кода сводится к добавлению ***omp parallel for.***

```
double **
transpose(double **a, int n, int m, int nThreads)
{
    double **b = malloc(sizeof(double *) * m);
    int i, j;
#pragma omp parallel for private(i, j) shared(a, b) num_threads(nThreads)
    for (i = 0; i < m; i++)
    {
        b[i] = malloc(sizeof(double) * n);
        for (j = 0; j < n; j++)
        {
            b[i][j] = a[j][i];
        }
    }
    return b;
}
```

Код всей программы можно посмотреть по ссылке
https://github.com/DariaShel/skipod/blob/main/transpose.c

3 Результаты замеров времени выполнения

Ниже приведены результаты замеров времени выполнения алгоритма на суперкомпьютере Polus в табличной форме и наглядно на графиках.

Программа запускалась со следующими параметрами:

- Матрица $A \in R^{n \times n}, n \in \{1000,\ 2000, 3000, 4000, 5000, 6000, 7000, 8000, 9000, 10000\}$
- Количество потоков $nThread \in \{1, 2, 4, 8, 16, 32, 64\}$

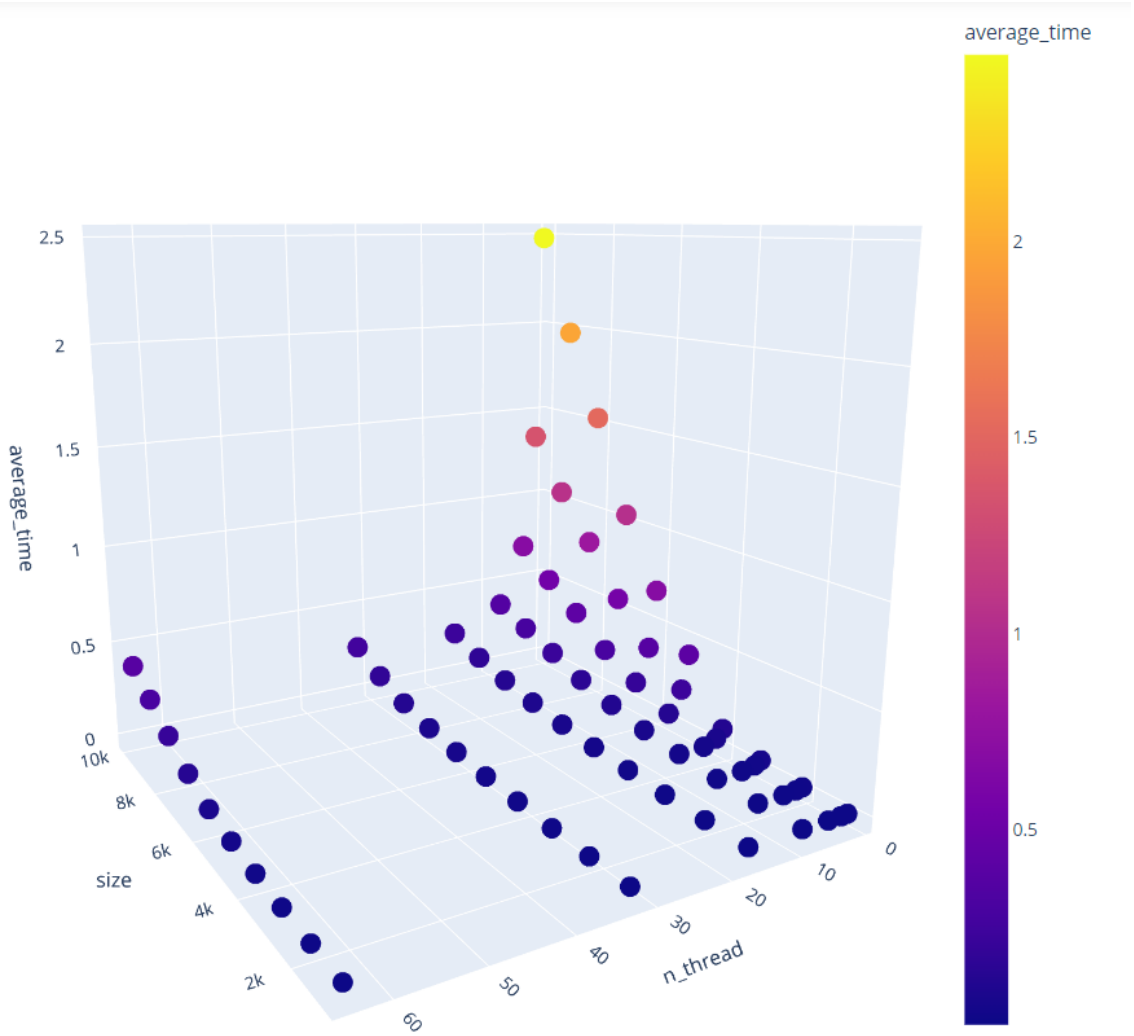
Было взято среднее значение времени за 5 запусков для каждой конфигурации.

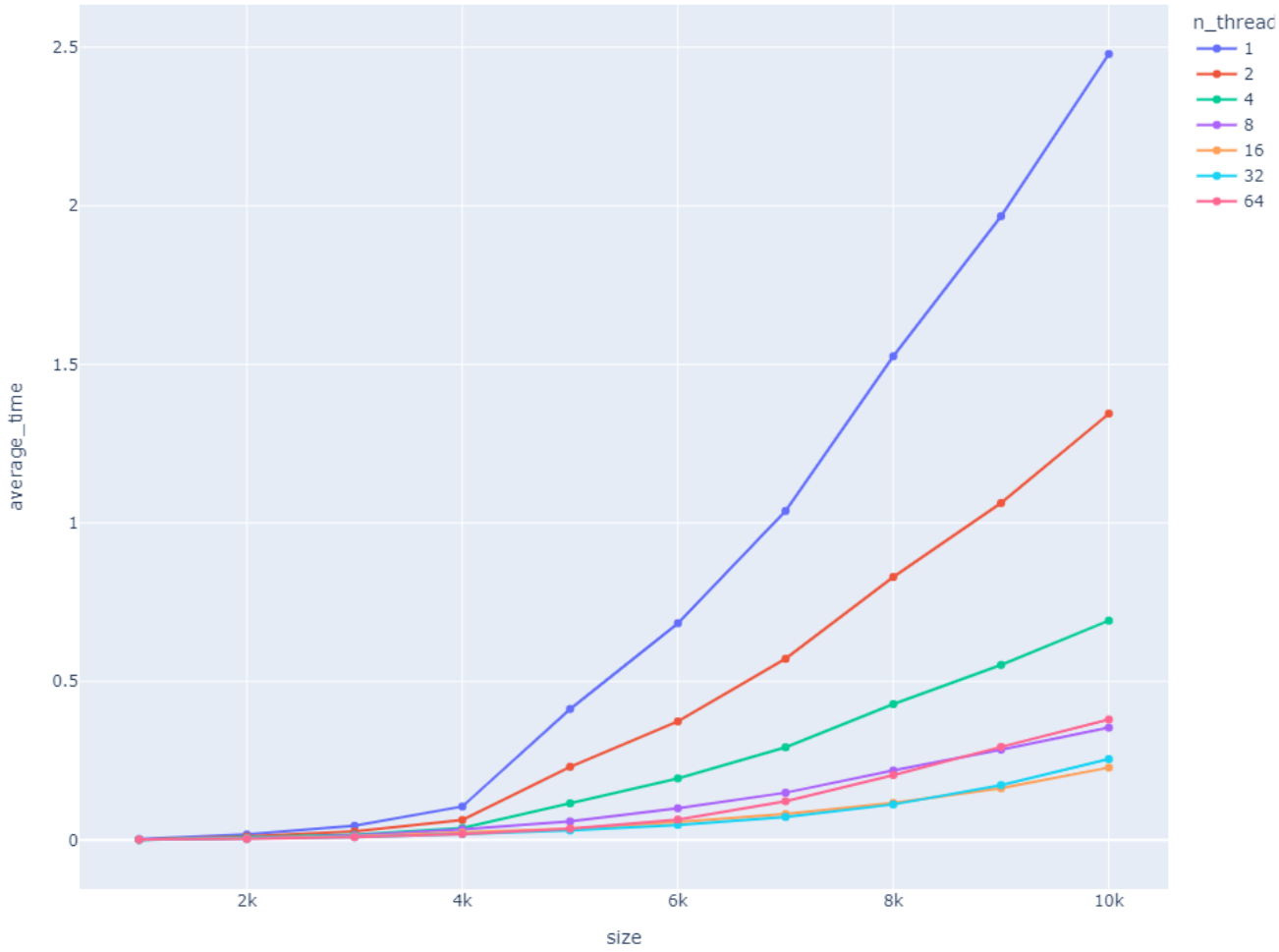
3.1 Таблицы

size	n_thread	average_time	size	n_thread	average_time	size	n_thread	average_time	size	n_thread	average_time	size	n_thread	average_time
1000	1	0.003237	2000	1	0.017415	3000	1	0.044771	4000	1	0.105689	5000	1	0.413145
1000	2	0.000920	2000	2	0.011337	3000	2	0.026812	4000	2	0.063245	5000	2	0.230878
1000	4	0.000769	2000	4	0.007021	3000	4	0.016373	4000	4	0.037896	5000	4	0.115746
1000	8	0.000771	2000	8	0.005574	3000	8	0.014286	4000	8	0.034011	5000	8	0.058552
1000	16	0.000791	2000	16	0.004837	3000	16	0.011223	4000	16	0.023616	5000	16	0.036033
1000	32	0.000968	2000	32	0.004030	3000	32	0.009348	4000	32	0.018923	5000	32	0.030078
1000	64	0.001563	2000	64	0.003520	3000	64	0.008856	4000	64	0.019144	5000	64	0.035809

size	n_thread	average_time	size	n_thread	average_time	size	n_thread	average_time	size	n_thread	average_time	size	n_thread	average_time
6000	1	0.683482	7000	1	1.037804	8000	1	1.525624	9000	1	1.967304	10000	1	2.478779
6000	2	0.374073	7000	2	0.571479	8000	2	0.829646	9000	2	1.062822	10000	2	1.344648
6000	4	0.194368	7000	4	0.292600	8000	4	0.428459	9000	4	0.552159	10000	4	0.691871
6000	8	0.100035	7000	8	0.149152	8000	8	0.219062	9000	8	0.285058	10000	8	0.354458
6000	16	0.056551	7000	16	0.081613	8000	16	0.117176	9000	16	0.163063	10000	16	0.228351
6000	32	0.047744	7000	32	0.072746	8000	32	0.112184	9000	32	0.173082	10000	32	0.255214
6000	64	0.064573	7000	64	0.122549	8000	64	0.204893	9000	64	0.293878	10000	64	0.380011

3.2 Графики





4 Анализ результатов

Из графиков и таблицы можно сделать вывод, что при увеличении количества потоков, особенно на больших размерах матриц, скорость выполнения алгоритма заметно увеличивается. Однако лучший результат получается распараллеливанием не на максимальное число потоков, а на 16. Это происходит из-за того, что при слишком большом количестве потоков тратятся ресурсы на управление ими, что приводит к дополнительным затратам времени.

5 Выводы

Выполнена работа по разработке параллельной версии алгоритма транспонирования матриц. Изучена технология написания параллельного алгоритма OpenMP. Проанализировано время выполнения алгоритмов на вычислительной системе Polus.

Технология OpenMP достаточно удобна в использовании и даёт значительный прирост производительности на рассчитанных на многопоточные вычисления системах.