

Санкт-Петербургский политехнический университет Петра Великого
Институт прикладной математики и механики
Высшая школа прикладной математики и вычислительной физики

Введение в технологии суперкомпьютерных вычислений

Отчёт по лабораторной работе №4

Работу
выполнил:
Тептев М. А.
Группа:
5030301/00102
Преподаватель:
Гатаулин Я. А.

Санкт-Петербург
2023

Содержание

1. Задание	3
2. Характеристики компьютера	3
3. Исследование последовательной версии программы	3
4. Исследование параллельной версии программы	3
5. Вывод	7
6. Коды программ	8

1. Задание

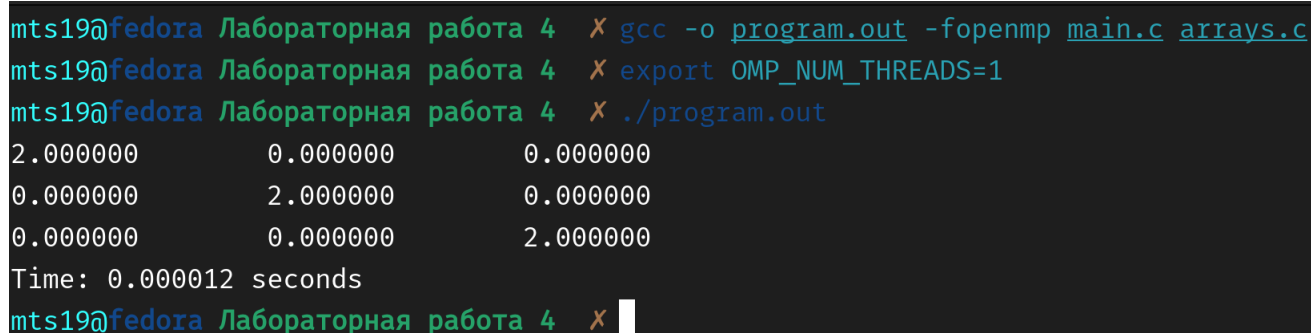
Реализовать и распараллелить средствами OpenMP операцию сложения(вычитание) матриц большой размерности. Сравнить время работы программ, определить ускорение и эффективность распараллеливания.

2. Характеристики компьютера

Количество ядер: 4, количество потоков: 8, объём оперативной памяти: 8Гб.

3. Исследование последовательной версии программы

Разработана последовательная версия программы, для сравнения с тестовым решением суммы двух единичных матриц размера 3×3 (рисунок 3.1). Как видно, программа работает верно, поэтому можем приступить к исследованию параллельной версии программы.



```
mts19@fedora Лабораторная работа 4 X gcc -o program.out -fopenmp main.c arrays.c
mts19@fedora Лабораторная работа 4 X export OMP_NUM_THREADS=1
mts19@fedora Лабораторная работа 4 X ./program.out
2.000000      0.000000      0.000000
0.000000      2.000000      0.000000
0.000000      0.000000      2.000000
Time: 0.000012 seconds
mts19@fedora Лабораторная работа 4 X
```

Рисунок 3.1. Результат работы последовательной версии программы для матрицы 3×3 .

4. Исследование параллельной версии программы

Также сравним с тестовым решением суммы двух единичных матриц размера 3×3 параллельный алгоритм при $TN = 8$ (рисунок 4.1). Как видно, программа работает верно.

```
mts19@fedora Лабораторная работа 4 X export OMP_NUM_THREADS=8
mts19@fedora Лабораторная работа 4 X ./program.out
2.000000      0.000000      0.000000
0.000000      2.000000      0.000000
0.000000      0.000000      2.000000
Time: 0.000846 seconds
```

Рисунок 4.1. Результат работы параллельной версии программы для матрицы 3×3 и $TN = 8$.

Приступим к изучению влияния различных условий на время работы параллельной программы. Первый эксперимент направлен на изучение влияния размерности матрицы на время выполнения при постоянном числе нитей $TN = 8$ (рисунок 4.2). Как видно, при увеличении размерности матрицы увеличивается и время работы.

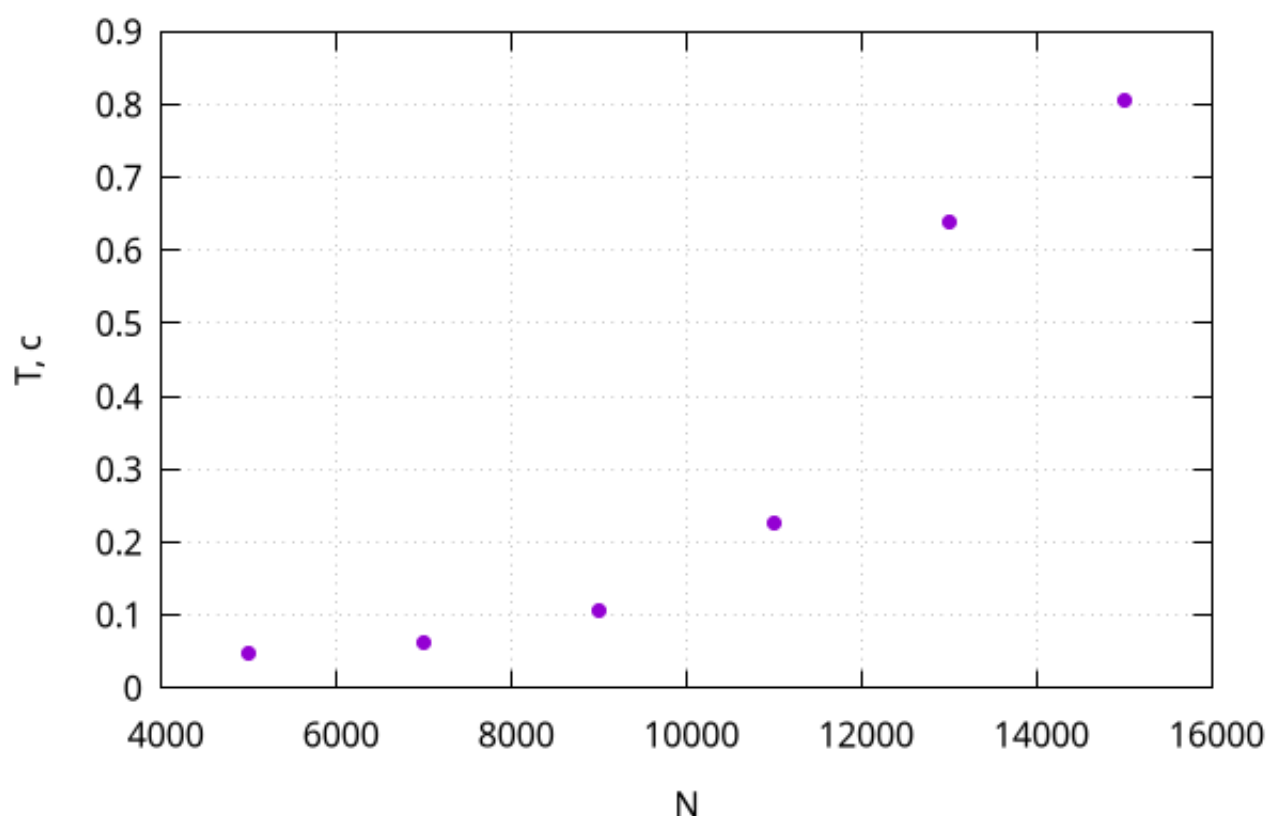


Рисунок 4.2. Зависимость времени выполнения программы от размерности матрицы для числа нитей $TN = 8$.

Теперь возьмём за основу размерность матрицы 15000×15000 и для неё исследуем зависимость времени выполнения, ускорения и эффективности от количества нитей (рисунок 4.3, рисунок 4.4, рисунок 4.5). Исследуя зависимости видно, что сначала увеличение количества нитей существенно влияет на скорость выполнения программы, но потом наступает некоторое насыщение и большого выигрыша не происходит.

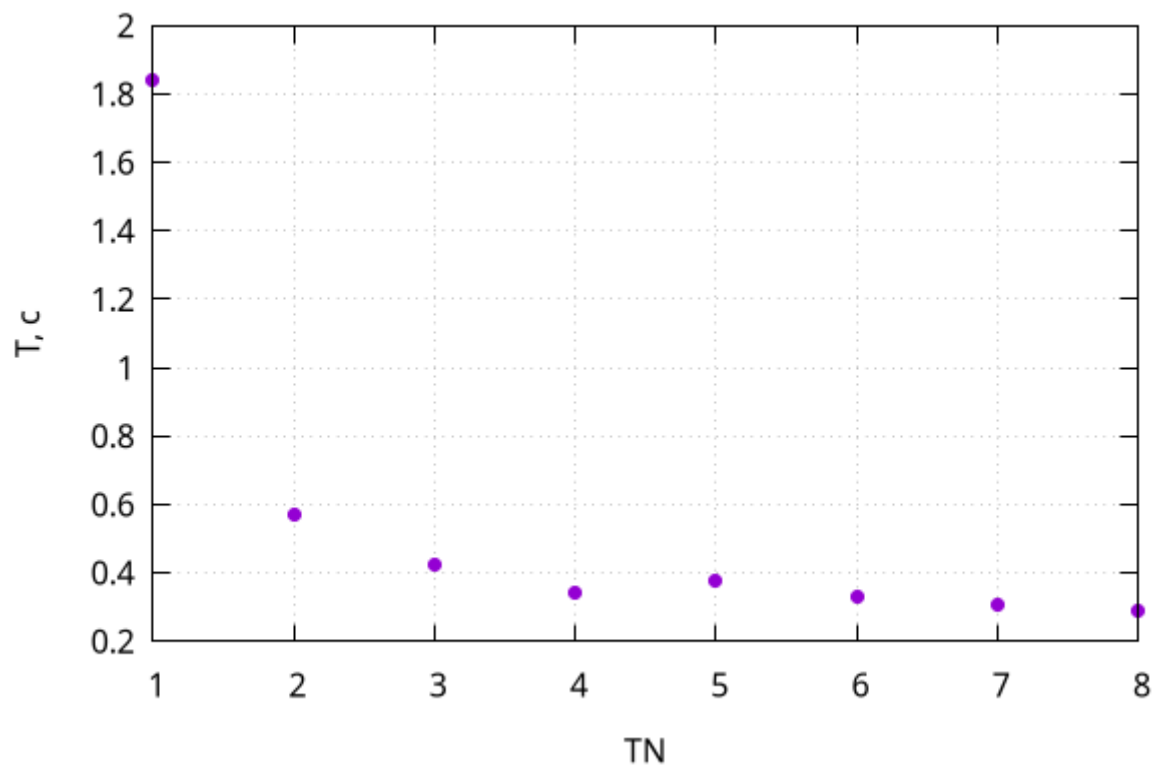


Рисунок 4.3. Зависимость времени выполнения программы от числа нитей для размерности матрицы 15000x15000.

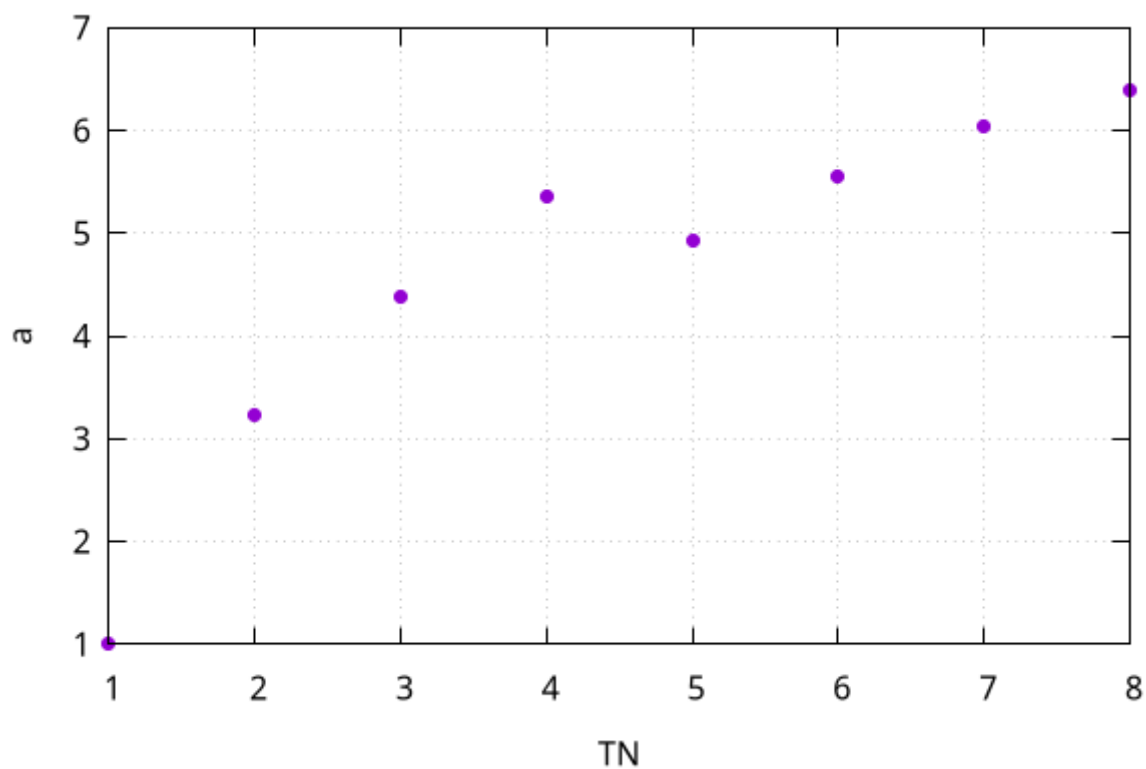


Рисунок 4.4. Зависимость ускорения ($a = \frac{T_1}{T_{tn}}$) выполнения программы от числа нитей для размерности матрицы 15000x15000.

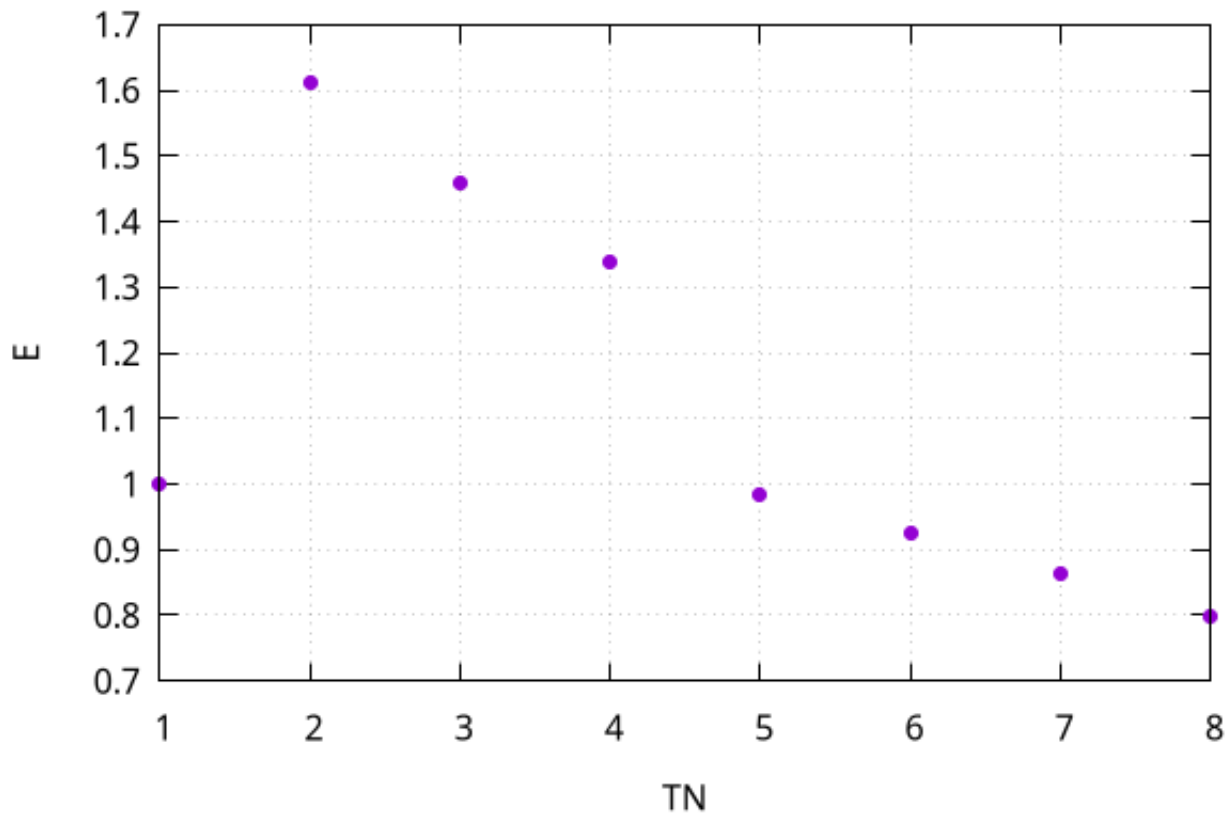


Рисунок 4.5. Зависимость эффективности ($e = \frac{a}{TN}$) выполнения программы от числа нитей для размерности матрицы 15000x15000.

Теперь, используя $TN = 8$ и размерность 15000x15000, исследуем влияние оптимизатора и способа распределения итераций между нитями (таблица 4.1, таблица 4.2). Использование опций оптимизации компилятора уменьшает время работы, однако при переходе с первого уровня на третий разница почти незаметна, а способ распределения итераций между нитями не оказывает сильного влияния на время выполнения программы.

	O0	O1	O3
t, сек	0.301593	0.255691	0.255691

Таблица 4.1

Зависимость времени выполнения от опций оптимизации компилятора для $TN = 8$ и размерности матрицы 15000x15000.

	STATIC	DYNAMIC, 1500	GUIDED
t, сек	0.298245	0.326329	0.301749

Таблица 4.2

Зависимость времени выполнения от способа распределения итераций между нитями для $TN = 8$ и размерности матрицы 15000x15000.

5. Вывод

В данной работе было реализовано расспаралеливание программы для вычисления суммы двух матриц и исследование зависимости эффективности работы программы от различных опций оптимизации и ускорений, предоставленных директивой OpenMP. Основными выводами являются следующие факты:

1. При увеличении размерности матрицы увеличивается время выполнения программы.
2. С увеличением количества нитей возрастает ускорение, но падает эффективность параллелизации.
3. Способ распределения итераций между циклами не оказывает сильного влияния на время выполнения программы
4. Опции оптимизатора ускоряют время выполнения программы примерно в 1,2 раза.

6. Коды программ

```
1 #include "arrays.h"
2 #include <omp.h>
3 #include <stddef.h>
4 #include <stdio.h>
5 #include <stdlib.h>
6 #define N 3
7
8 int main() {
9     omp_set_dynamic(0);
10
11     double **A = matrix_alloc(N, N);
12     matrix_random(A, N, N);
13     for (size_t i = 0; i < N; i++) {
14         for (size_t j = 0; j < N; j++) {
15             if (i == j) {
16                 A[i][j] = 1.0;
17             } else {
18                 A[i][j] = 0.0;
19             }
20         }
21     }
22     double **B = matrix_alloc(N, N);
23     matrix_random(B, N, N);
24     for (size_t i = 0; i < N; i++) {
25         for (size_t j = 0; j < N; j++) {
26             if (i == j) {
27                 B[i][j] = 1.0;
28             } else {
29                 B[i][j] = 0.0;
30             }
31         }
32     }
33     double **C = matrix_alloc(N, N);
34
35     double start_time = omp_get_wtime();
36 #pragma omp parallel for collapse(2) shared(A, B, C)
37     for (size_t i = 0; i < N; i++) {
38         for (size_t j = 0; j < N; j++) {
39             C[i][j] = A[i][j] + B[i][j];
40         }
41     }
42     double end_time = omp_get_wtime();
43
44     free(A);
45     free(B);
46     for (size_t i = 0; i < N; i++) {
47         for (size_t j = 0; j < N; j++) {
48             printf("%lf\t", C[i][j]);
49         }
50         printf("\n");
51     }
52     free(C);
53
54     printf("Time: %f seconds\n", end_time - start_time);
55
56     return EXIT_SUCCESS;
57 }
```

Рисунок 6.1. Тестовая программа.


```

1  #include "arrays.h"
1  #include <omp.h>
2  #include <stddef.h>
3  #include <stdio.h>
4  #include <stdlib.h>
5  #define N 15000
6
7  int main() {
8      omp_set_dynamic(0);
9
10     double **A = matrix_alloc(N, N);
11     matrix_random(A, N, N);
12     double **B = matrix_alloc(N, N);
13     matrix_random(B, N, N);
14     double **C = matrix_alloc(N, N);
15
16     double start_time = omp_get_wtime();
17     #pragma omp parallel for collapse(2) shared(A, B, C)
18     for (size_t i = 0; i < N; i++) {
19         for (size_t j = 0; j < N; j++) {
20             C[i][j] = A[i][j] + B[i][j];
21         }
22     }
23     double end_time = omp_get_wtime();
24
25     free(A);
26     free(B);
27     free(C);
28
29     printf("Time: %f seconds\n", end_time - start_time);
30
31     return EXIT_SUCCESS;
32 }

```

Рисунок 6.2. Параллельный алгоритм.

