Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования

Московский государственный технический университет имени Н.Э.Баумана
(МГТУ им. Н.Э.Баумана)

ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №3

«Параллельная реализация решения системы линейных алгебраических уравнений с помощью OpenMP»

Выполнил: Митрошкин А.А.
(Фамилия И.О. студента)

ИУ9-51Б
(Индекс группы)

Преподаватель: Царев.А. С.
(Фамилия И.О. преподавателя)

(Подпись)

Москва, 2023

Оглавление

1. Цель работы. Сравнить время работы вычисления матрицы на одном потоке и нескольких при помощи mpi. 2. Условия задачи Пусть есть система из N линейных алгебраических уравнений в виде Ax=b, где A – матрица коэффициентов уравнений размером N×N, b – вектор правых частей размером N, x – искомый вектор решений размером N. Решение системы уравнений итерационным методом состоит в выполнении следующих шагов. 1. Задается x0 – произвольное начальное приближение решения (вектор с произвольными начальными значениями). 2. Приближение многократно улучшается с использованием формулы вида xn+1 = f(xn), где функция f определяется используемым методом1. 3. Процесс продолжается, пока не выполнится условие g(xn) < ε, где функция g определяется используемым методом, а величина ε задает требуемую точность. 1. Написать программу, которая реализует итерационный алгоритм решения системы линейных алгебраических уравнений вида Ax=b в соответствии с выбранным вариантом. Здесь A – матрица размером N×N, x и b – векторы длины N. Тип элементов – double. 2. Программу распараллелить с помощью MPI с разрезанием матрицы A по строкам на близкие по размеру, возможно не одинаковые, части. Соседние строки матрицы должны располагаться в одном или в соседних MPI-процессах. Реализовать два варианта программы: 1: векторы x и b дублируются в каждом MPI-процессе, 2: векторы x и b разрезаются между MPI-процессами аналогично матрице A. (только для сдающих после срока) Уделить внимание тому, чтобы при запуске программы на различном числе MPI-процессов решалась одна и та же задача (исходные данные заполнялись одинаковым образом). 3. Замерить время работы двух вариантов программы при использовании различного числа процессорных ядер: 1,2, 4, 8, 16. Построить графики зависимости времени работы программы, ускорения и эффективности распараллеливания от числа используемых ядер. Исходные данные, параметры N и ε подобрать таким образом, чтобы решение задачи на одном ядре занимало не менее 30

секунд. Также параметр N разрешено подобрать таким образом, чтобы он нацело делился на на 1,2,4,8 и 16. 4. На основании полученных результатов сделать вывод о целесообразности использования одного или второго варианта программы.

1. Написать программу, которая реализует итерационный алгоритм решения системы линейных алгебраических уравнений вида Ax=b в соответствии с выбранным вариантом. Здесь A – матрица размером N×N, x и b – векторы длины N. Тип элементов – double. 2. Программу распараллелить с помощью MPI с разрезанием матрицы A по строкам на близкие по размеру, возможно не одинаковые, части. Соседние строки матрицы должны располагаться в одном или в соседних MPI-процессах. Реализовать два варианта программы: 1: векторы x и b дублируются в каждом MPI-процессе,
2. Векторы x и b разрезаются между MPI-процессами аналогично матрице A. (только для сдающих после срока) Уделить внимание тому, чтобы при запуске программы на различном числе MPI-процессов решалась одна и та же задача (исходные данные заполнялись одинаковым образом).
3. Замерить время работы двух вариантов программы при использовании различного числа процессорных ядер: 1,2, 4, 8, 16. Построить графики зависимости времени работы программы, ускорения и эффективности распараллеливания от числа используемых ядер. Исходные данные, параметры N и ε подобрать таким образом, чтобы решение задачи на одном ядре занимало не менее 30 секунд. Также параметр N разрешено подобрать таким образом, чтобы он нацело делился на 1,2,4,8 и 16.
4. На основании полученных результатов сделать вывод о целесообразности использования одного или второго варианта программы

3. Листинг кода решения

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <omp.h>

#define N 1024 // Размерность матрицы

// Умножение матрицы на вектор
double* mult_m_v(double** m, double* v) {
    double* res;
    res = (double*)malloc(sizeof(double) * N);
    int i;

    // Параллельный цикл для умножения матрицы на вектор
```

```
18.#pragma omp parallel for shared(m, v, res) private(i)
19.    for (i = 0; i < N; i++) {
20.        res[i] = 0;
21.
22.        for (int j = 0; j < N; j++) {
23.            res[i] += (m[i][j] * v[j]);
24.        }
25.    }
26.    return res;
27.}
28.
29.// Умножение вектора на скаляр
30.void mult_v_s(double* v, double s) {
31.    int i;
32.
33.    // Параллельный цикл для умножения вектора на скаляр
34.#pragma omp parallel for shared(v) private(i)
35.    for (i = 0; i < N; i++) {
36.        v[i] *= s;
37.    }
38.}
39.
40.// Вычитание векторов
41.void difference(double* v1, double* v2) {
42.    int i;
43.
44.    // Параллельный цикл для вычитания векторов
45.#pragma omp parallel for shared(v1, v2) private(i)
46.    for (i = 0; i < N; i++) {
47.        v1[i] -= v2[i];
48.    }
49.}
50.
51.// Норма вектора
52.double norm(double* v) {
53.    int i;
54.    int len = 0;
55.
56.    // Параллельный цикл для вычисления квадрата нормы вектора
57.#pragma omp parallel for shared(v) private(i) reduction(+:len)
58.    for (i = 0; i < N; i++) {
59.        len += (v[i] * v[i]);
60.    }
61.
62.    len = sqrt(len);
63.    return len;
64.}
65.
66.// Скалярное произведение векторов
67.double scalar_product(double* v1, double* v2) {
```

```
68.     int i;
69.     double len = 0;
70.
71.     // Параллельный цикл для вычисления скалярного произведения векторов
72.#pragma omp parallel for shared(v1, v2) private(i) reduction(+:len)
73.     for (i = 0; i < N; i++) {
74.         len += (v1[i] * v2[i]);
75.     }
76.
77.     return len;
78.}
79.
80.int main() {
81.#ifdef _OPENMP
82.     printf("OpenMP is supported!\n");
83.#endif
84.
85.     double eps = 0.0000001;
86.     double tao;
87.
88.     // Выделение памяти для матрицы A
89.     double **A;
90.     A = (double**)malloc(sizeof(double*) * N);
91.
92.     // Инициализация матрицы A
93.     for (int i = 0; i < N; i++) {
94.         A[i] = (double*)malloc(sizeof(double) * N);
95.
96.         for (int j = 0; j < N; j++) {
97.             if (i == j) {
98.                 A[i][j] = 2.0;
99.             } else {
100.                    A[i][j] = 1.0;
101.                }
102.            }
103.        }
104.
105.        // Выделение памяти для векторов x и b
106.        double* x;
107.        double* b;
108.        x = (double*)malloc(sizeof(double) * N);
109.        b = (double*)malloc(sizeof(double) * N);
110.
111.        // Инициализация векторов x и b
112.        for (int i = 0; i < N; i ++) {
113.            x[i] = 0;
114.            b[i] = N + 1;
115.        }
116.
117.        double* y;
```

```
118.         double* ay;
119.
120.         while (1) {
121.             // y = Ax-b
122.             y = mult_m_v(A, x);
123.             difference(y, b);
124.
125.             // Нормировка до достижения сходимости
126.             if ((norm(x) / norm(y)) < eps) {
127.                 printf("Similar!");
128.                 free(y);
129.                 break;
130.             }
131.
132.             // ay = A * y
133.             ay = mult_m_v(A, y);
134.
135.             tao = scalar_product(y, ay) / scalar_product(ay, ay);
136.             // x = x - tao*y
137.             mult_v_s(y, tao);
138.             difference(x, y);
139.
140.             free(y);
141.             free(ay);
142.         }
143.
144.         // Освобождение памяти
145.         free(x);
146.         free(b);
147.
148.         for (int i = 0; i < N; i++) {
149.             free(A[i]);
150.         }
151.
152.         free(A);
153.
154.         return 0;
155.     }
156.
```

4. Результаты

Однопоточное. Для матриц размера 100 на 10

```
2023/11/18 09:47:08 Завершено после 3324 итераций
Program time: 57.247875ms
```

Для матриц размера 10000 * 10000

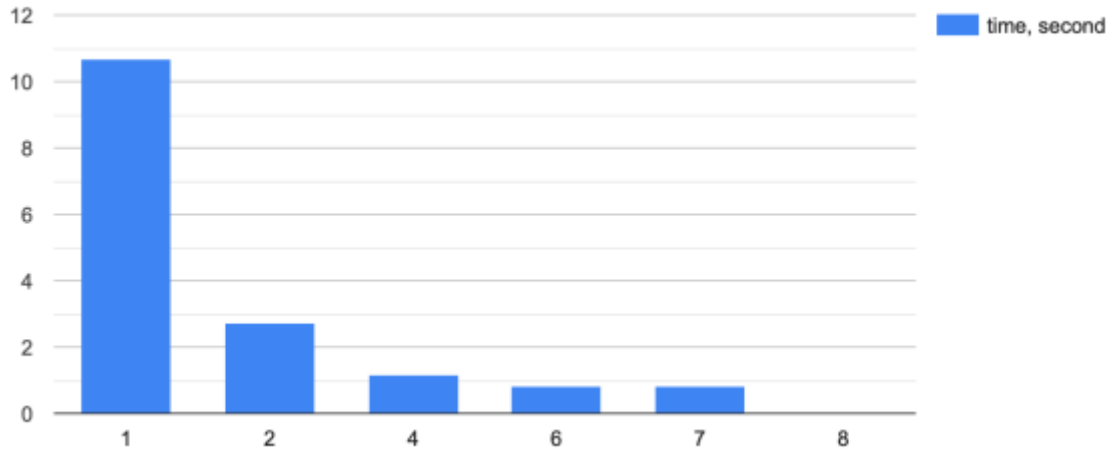
```
2023/11/18 09:47:39 Завершено после 83 итераций
Program time: 10.681274625s
```

Для матриц размера 100 на 100

```
Success
make run p=4 n=10000  1.15s user 0.35s system 91% cpu 1.633 total
```

Для матриц размера 10000 * 10000

```
Success
make run p=4 n=10000  1.15s user 0.35s system 91% cpu 1.633 total
```



Характеристики компьютера

12th Gen Intel(R) Core(TM) i5-12450H 2.50 GHz
16,0 ГБ (доступно: 15,7 ГБ)
64-разрядная операционная система, процессор x64

Вывод:

Можно заметить, что результат улучшается с увеличением ядер, но при этом больше процессов, чем число ядер мы использовать не можем. Именно их параллельная работа позволяет считать матрицу параллельно и улучшать время работы