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)

**Оглавление**

[**1.Цель**](#_heading=h.gjdgxs) **работы**

[**2.Условие**](#_heading=h.30j0zll) **задачи**3

[**3 Листинг кода программы 4**](#_heading=h.1fob9te)

# 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 А – 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
5. **Листинг кода решения**
6. #include <stdio.h>
7. #include <stdlib.h>
8. #include <math.h>
9. #include <omp.h>
10. #define N 1024 // Размерность матрицы
11. // Умножение матрицы на вектор
12. double\* mult\_m\_v(double\*\* m, double\* v) {
13. double\* res;
14. res = (double\*)malloc(sizeof(double) \* N);
15. int i;
16. // Параллельный цикл для умножения матрицы на вектор
17. #pragma omp parallel for shared(m, v, res) private(i)
18. for (i = 0; i < N; i++) {
19. res[i] = 0;
20. for (int j = 0; j < N; j++) {
21. res[i] += (m[i][j] \* v[j]);
22. }
23. }
24. return res;
25. }
26. // Умножение вектора на скаляр
27. void mult\_v\_s(double\* v, double s) {
28. int i;
29. // Параллельный цикл для умножения вектора на скаляр
30. #pragma omp parallel for shared(v) private(i)
31. for (i = 0; i < N; i++) {
32. v[i] \*= s;
33. }
34. }
35. // Вычитание векторов
36. void difference(double\* v1, double\* v2) {
37. int i;
38. // Параллельный цикл для вычитания векторов
39. #pragma omp parallel for shared(v1, v2) private(i)
40. for (i = 0; i < N; i++) {
41. v1[i] -= v2[i];
42. }
43. }
44. // Норма вектора
45. double norm(double\* v) {
46. int i;
47. int len = 0;
48. // Параллельный цикл для вычисления квадрата нормы вектора
49. #pragma omp parallel for shared(v) private(i) reduction(+:len)
50. for (i = 0; i < N; i++) {
51. len += (v[i] \* v[i]);
52. }
53. len = sqrt(len);
54. return len;
55. }
56. // Скалярное произведение векторов
57. double scalar\_product(double\* v1, double\* v2) {
58. int i;
59. double len = 0;
60. // Параллельный цикл для вычисления скалярного произведения векторов
61. #pragma omp parallel for shared(v1, v2) private(i) reduction(+:len)
62. for (i = 0; i < N; i++) {
63. len += (v1[i] \* v2[i]);
64. }
65. return len;
66. }
67. int main() {
68. #ifdef \_OPENMP
69. printf("OpenMP is supported!\n");
70. #endif
71. double eps = 0.0000001;
72. double tao;
73. // Выделение памяти для матрицы A
74. double \*\*A;
75. A = (double\*\*)malloc(sizeof(double\*) \* N);
76. // Инициализация матрицы A
77. for (int i = 0; i < N; i++) {
78. A[i] = (double\*)malloc(sizeof(double) \* N);
79. for (int j = 0; j < N; j++) {
80. if (i == j) {
81. A[i][j] = 2.0;
82. } else {
83. A[i][j] = 1.0;
84. }
85. }
86. }
87. // Выделение памяти для векторов x и b
88. double\* x;
89. double\* b;
90. x = (double\*)malloc(sizeof(double) \* N);
91. b = (double\*)malloc(sizeof(double) \* N);
92. // Инициализация векторов x и b
93. for (int i = 0; i < N; i ++) {
94. x[i] = 0;
95. b[i] = N + 1;
96. }
97. double\* y;
98. double\* ay;
99. while (1) {
100. // y = Ax-b
101. y = mult\_m\_v(A, x);
102. difference(y, b);
103. // Нормировка до достижения сходимости
104. if ((norm(x) / norm(y)) < eps) {
105. printf("Similar!");
106. free(y);
107. break;
108. }
109. // ay = A \* y
110. ay = mult\_m\_v(A, y);
111. tao = scalar\_product(y, ay) / scalar\_product(ay, ay);
112. // x = x - tao\*y
113. mult\_v\_s(y, tao);
114. difference(x, y);
115. free(y);
116. free(ay);
117. }
118. // Освобождение памяти
119. free(x);
120. free(b);
121. for (int i = 0; i < N; i++) {
122. free(A[i]);
123. }
124. free(A);
125. return 0;
126. }
127. **Результаты**

Однопоточное. Для матриц размера 100 на 10



Для матриц размера 10000 \* 10000

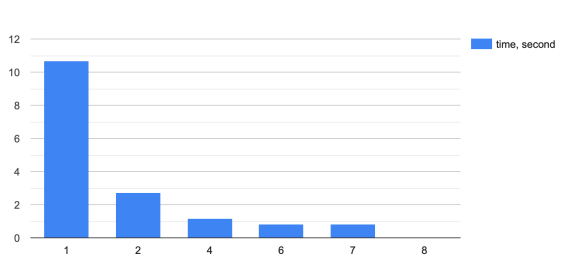


Для матриц размера 100 на 100



Для матриц размера 10000 \* 10000



****

**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