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 №2**

**«Параллельная реализация решения системы линейных алгебраических уравнений с помощью MPI»**

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. import time
7. import numpy as np
8. from numpy.linalg import norm, det
9. import sys
10. from mpi4py import MPI
11. # Установка начального значения для генерации случайных чисел
12. np.random.seed(42)
13. # Инициализация MPI (Message Passing Interface)
14. comm = MPI.COMM\_WORLD
15. rank = comm.Get\_rank()  # Номер текущего процесса
16. size = comm.Get\_size()  # Общее количество процессов
17. # Размер матрицы и размер блока берутся из аргументов командной строки
18. MATRIX\_SIZE = 2 \*\* 13
19. MATRIX\_SPLIT = int(sys.argv[1])
20. # Создание симметричной положительно определенной матрицы 'a'
21. a = np.zeros((MATRIX\_SIZE, MATRIX\_SIZE), dtype=np.double)
22. for i in range(MATRIX\_SIZE):
23. for j in range(MATRIX\_SIZE):
24. if i == j:
25. a[i, j] = 2
26. else:
27. a[i, j] = 1
28. # Выбор тестового случая в зависимости от аргумента командной строки
29. if sys.argv[2] == "1":
30. # Тест 1: установка значений векторов b и x
31. b = np.ones(MATRIX\_SIZE, dtype=np.double) \* (2 \*\* 13 + 1)
32. x = np.zeros(MATRIX\_SIZE, dtype=np.double)
33. elif sys.argv[2] == "2":
34. # Тест 2: генерация случайного вектора u и вычисление вектора b
35. u = np.zeros(MATRIX\_SIZE, dtype=np.double)
36. for i in range(MATRIX\_SIZE):
37. u[i] = np.random.random()
38. b = np.matmul(a, u[:, None]).T[0]
39. x = np.zeros(MATRIX\_SIZE, dtype=np.double)
40. # Установка значения epsilon для оценки точности вычислений
41. epsilon = 1e-5
42. # Функция для умножения матрицы на вектор
43. def mult\_matrix\_by\_vector(m, v):
44. v = v[:, None]
45. # Буфер для вычислений
46. part\_a = np.empty(shape=(MATRIX\_SIZE // MATRIX\_SPLIT, MATRIX\_SIZE), dtype=np.double)
48. comm.Scatter(m, part\_a, root=0)
49. # Умножение части матрицы на в  ектор
50. part\_a = part\_a @ v
51. # Выделение места под результат
52. res = None
53. if rank == 0:
54. res = np.empty(shape=(MATRIX\_SIZE, 1), dtype=np.double)
55. # Сбор результатов на процессе с rank=0
56. comm.Gather(part\_a, res, root=0)
57. return comm.bcast(res, root=0).T[0]
58. # Основная функция программы
59. def main():
60. global x
61. old\_crit = 0  # Значение критерия на предыдущей итерации
62. i = 0  # Счетчик итераций
63. while True:
64. i += 1
65. y = mult\_matrix\_by\_vector(a, x) - b
66. ay = mult\_matrix\_by\_vector(a, y)
67. flag = False
68. if rank == 0:
69. crit = norm(y) / norm(b)
70. if crit < epsilon or crit == old\_crit:
71. flag = True
72. else:
73. old\_crit = crit
74. tao = (y.dot(ay)) / (ay.dot(ay))
75. x = x - tao \* y
76. # Рассылка флага о завершении и проверка условия выхода
77. if comm.bcast(flag, root=0):
78. break
79. x = comm.bcast(x, root=0)
80. # Запуск основной функции при выполнении скрипта
81. if \_\_name\_\_ == '\_\_main\_\_':
82. t = time.time()
83. main()
84. # Вывод результата времени выполнения для процесса с rank=0
85. if rank == 0:
86. print(MATRIX\_SPLIT, time.time() - t)
87. **Результаты**

Однопоточное. Для матриц размера 100 на 10



Для матриц размера 10000 \* 10000

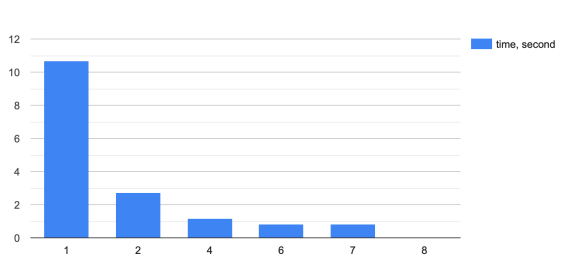


Для матриц размера 100 на 100



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