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)

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

```
4. import time
5. import numpy as np
6. from numpy.linalg import norm, det
7. import sys
8. from mpi4py import MPI
9.
10.# Установка начального значения для генерации случайных чисел
11.np.random.seed(42)
12.
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
MATRIX_SIZE = 2 ** 13
MATRIX_SPLIT = int(sys.argv[1])

# Создание симметричной положительно определенной матрицы 'a'
a = np.zeros((MATRIX_SIZE, MATRIX_SIZE), dtype=np.double)
for i in range(MATRIX_SIZE):
    for j in range(MATRIX_SIZE):
        if i == j:
            a[i, j] = 2
        else:
            a[i, j] = 1

# Выбор тестового случая в зависимости от аргумента командной строки
if sys.argv[2] == "1":
    # Тест 1: установка значений векторов b и x
    b = np.ones(MATRIX_SIZE, dtype=np.double) * (2 ** 13 + 1)
    x = np.zeros(MATRIX_SIZE, dtype=np.double)
elif sys.argv[2] == "2":
    # Тест 2: генерация случайного вектора u и вычисление вектора b
    u = np.zeros(MATRIX_SIZE, dtype=np.double)
    for i in range(MATRIX_SIZE):
        u[i] = np.random.random()
    b = np.matmul(a, u[:, None]).T[0]
    x = np.zeros(MATRIX_SIZE, dtype=np.double)

# Установка значения epsilon для оценки точности вычислений
epsilon = 1e-5

# Функция для умножения матрицы на вектор
def mult_matrix_by_vector(m, v):
    v = v[:, None]
    # Буфер для вычислений
    part_a = np.empty(shape=(MATRIX_SIZE // MATRIX_SPLIT, MATRIX_SIZE),
dtype=np.double)

    comm.Scatter(m, part_a, root=0)
    # Умножение части матрицы на в  ектор
    part_a = part_a @ v
    # Выделение места под результат
    res = None
    if rank == 0:
        res = np.empty(shape=(MATRIX_SIZE, 1), dtype=np.double)
    # Сбор результатов на процессе с rank=0
    comm.Gather(part_a, res, root=0)

    return comm.bcast(res, root=0).T[0]

# Основная функция программы
def main():
```

```
67.     global x
68.
69.     old_crit = 0  # Значение критерия на предыдущей итерации
70.     i = 0  # Счетчик итераций
71.     while True:
72.         i += 1
73.         y = mult_matrix_by_vector(a, x) - b
74.         ay = mult_matrix_by_vector(a, y)
75.         flag = False
76.         if rank == 0:
77.             crit = norm(y) / norm(b)
78.             if crit < epsilon or crit == old_crit:
79.                 flag = True
80.             else:
81.                 old_crit = crit
82.                 tao = (y.dot(ay)) / (ay.dot(ay))
83.                 x = x - tao * y
84.
85.         # Рассылка флага о завершении и проверка условия выхода
86.         if comm.bcast(flag, root=0):
87.             break
88.         x = comm.bcast(x, root=0)
89.
90. # Запуск основной функции при выполнении скрипта
91. if __name__ == '__main__':
92.     t = time.time()
93.     main()
94.     # Вывод результата времени выполнения для процесса с rank=0
95.     if rank == 0:
96.         print(MATRIX_SPLIT, time.time() - t)
97.
```

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

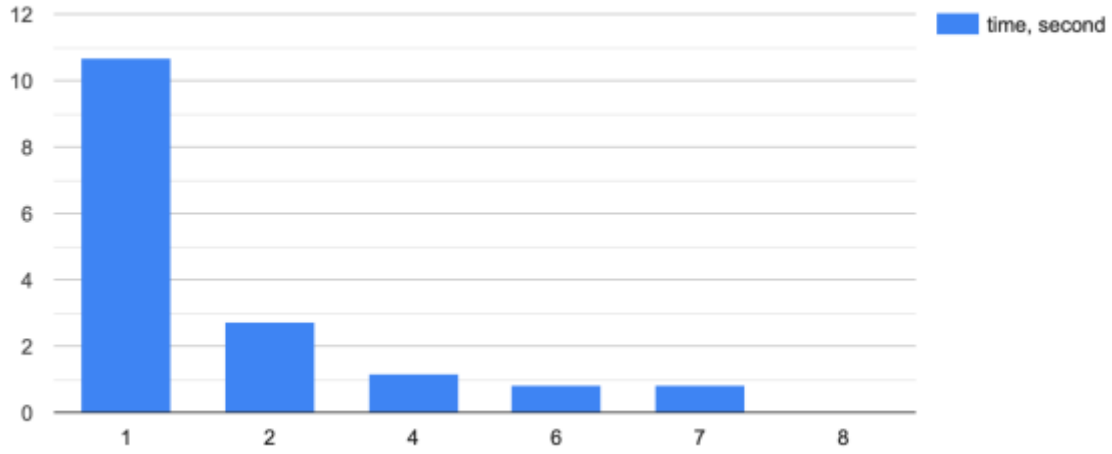
```
2023/11/18 09:47:39 Завершено после 83 итераций
Program time: 10.681274625s
```

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