Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное
учреждение высшего образования
«Самарский национальный исследовательский университет
имени академика С.П. Королева»

Институт информатики и кибернетики

Кафедра технической кибернетики

Отчет по лабораторной работе № 2
Вычисления “GAXPY” на языке программирования C++»

Курс: «Параллельные алгоритмы»

Студент: Грязнов Илья Евгеньевич
(фамилия, имя отчество)
Группы 6131-010402D

Самара 2022

Содержание

Задание 3
Теоретическая часть 4
Ход выполнения работы 7
Вывод 10
Список использованных источников 11
Листинг программы 12
Часть, выполненная на C++ 12
Часть, выполненная в Octave 14

Задание

В целях ознакомления с операцией gaxpy и приобретения навыков программирования параллельных вычислений самостоятельно реализовать метод вычисления данной операции на языке программирования C++. Записать входные и выходные данные, произвести аналогичные вычисления с теми же данными при помощи уже реализованной библиотеки BLAS.

Теоретическая часть

Умножение матриц является одной из базовых операций вычислительной линейной алгебры. К ней неизменно сводятся блочные алгоритмы матричных разложений, которые в свою очередь широко применяются при решении систем линейных алгебраических уравнений и алгебраической проблемы собственных значений – двух основных задач упомянутой предметной области.

Операцию gaxpy (general A x plus y.) принято записывать в виде

$$z=Ax+y$$

Через нее удобно выражать умножение матриц, матричные разложения и итерационные методы решения СЛАУ.

Положим, что вектора и матрица из gaxpy для удобства составления параллельного алгоритма разбиты на блоки: $z_i, y_i \in \mathbb{R}^{\alpha \times 1}$, где 1≤i, j≤p, α=n/p, β=m/p, а p – количество задач искомого алгоритма. Построение параллельного алгоритма начинается с распределения блоков матрицы A между задачами; Выбранный способ распределения служит для дальнейшей классификации алгоритмов.

Распределение матрицы по блочным строкам

Пусть результатом вычислений, заданных задачей μ (1≤μ≤p) параллельного алгоритма, будет μ-ый блок вектора z – zμ=Aμx+yμ, где $A_\mu \in \mathbb{R}^{\alpha \times m}$ есть μ-ая блочная строка матрицы A. Тогда выделяя из строки и x отдельные блоки перепишем последнее выражение через блочное скалярное произведение блочной строки A и вектора x:

$$z_\mu = \sum_{j=1}^{p} A_{\mu j} x_j + y_\mu$$

Таким образом, для организации вычислений по ф. (1) в памяти задачи μ необходимо разместить все указанные в (1) блоки: Aμj, xj (1≤j≤p) и yμ как на рисунке ниже. То есть всю блочную строку Aμ и весь вектор x. Искомый блок zμ сформируем поверх блока yμ (алгоритм с замещением начальных данных результатами расчетов).

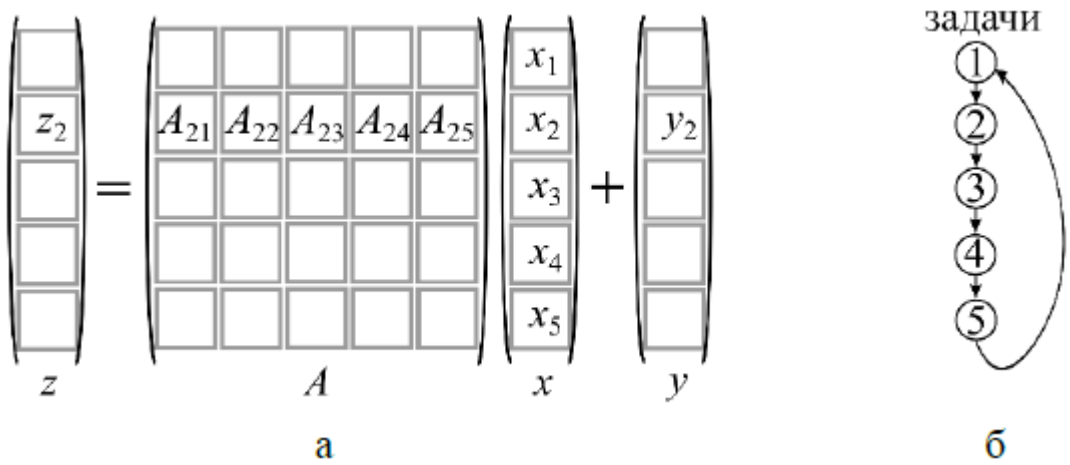

Рис. 1. Иллюстрации к параллельным алгоритмам gaxpy с распределением матрицы A между задачами по блочным строкам.

Необходимости в обмене данными между задачами при таком их распределении не возникает; платой за отсутствие коммуника-ций является дублирование вектора *x* в памяти всех задач.

Алгоритм 1 Параллельный алгоритм gaxpy без коммуникаций с распределением матрицы по блочным строкам

Инициализация{row=(μ-1)α+1:μα; Aloc=A(row,:); xloc=x; yloc=y(row)}

for *j=1:p*% проход по слагаемым из суммы в ф. (1)

a=(j-1)β+1:jβ% индексы, задающие блочный столбец *j*

yloc=yloc+Aloc(:,a)xloc(a)% вычисление слагаемого *j* из (1)

end for

Переходя к построению следующего алгоритма откажемся от дублирования вектора x в памяти всех задач. Пусть при начальном распределении данных задаче μ достанется блок xμ. Тогда необходимо предусмотреть обмен блоками этого вектора между задачами алгоритма; такой, чтобы в памяти любой задачи побывали все блоки x, как необходимые для вычислений по (1).

Алгоритм 2 Параллельный алгоритм gaxpy на кольце с распределением матрицы по блочным строкам:

Инициализация{row=(μ-1)α+1:μα; Aloc=A(row,:); xloc=x((μ-1)β+1:μβ); yloc=y(row); left, right}

for *j=1:p*% проход по слагаемым из суммы в ф. (1)

send(xloc,right)% отсылка *xloc* правому соседу

recv(xloc,left)% прием *xloc* от левого соседа

% определение индекса полученного блока x*τ*

τ=μ-j; ***if*** *τ≤0* ***then*** *τ=τ+p* ***end if***

a=(τ-1)β+1:τβ % индексы, задающие блочный столбец τ

yloc=yloc+Aloc(:,a)xloc% вычисление слагаемого τ из (1)

end for

В новом варианте инициализации иначе определяется вектор *xloc*, теперь в памяти задачи μ будет находиться лишь один блок вектора *x*; перед началом вычислений это *x*μ, которому в алгоритме при инициализации соответствует *x((μ-1)β+1:μβ)*. Переменные *left, right* – номера соседей μ-ой задачи, с которыми она будет обмениваться данными. При организации коммуникаций на процессорном кольце *left*=μ-1 и *right*=μ+1 при 1<μ<*p*; у первой задачи левый сосед – задача *p*, у последней правый – задача 1.

Ход выполнения работы

Первым делом был реализован код программы на C++, отвечающий за создание квадратной матрицы заданного размера и двух векторов, длинной соответственно подходящей для выполнения операции gaxpy.

Далее был реализована часть выполняющая непосредственно вычислительные операции для данных объектов с числовыми значениями типа double (число с плавающей точкой). Сама реализация программы представлена ниже в приложении (после списка используемой литературы).

Аналогично ходу выполнения первой лабораторной работы выполним вычисления операции gaxpy средствами стандартной библиотеки blas будем осуществлять при помощи GNU Octave (GUI) и сравненим полученные результаты. Для этого полученные в консольном приложении, написанном на C++ данные записываем в txt файл и передаем в Octave (Код реализуемый в Octave так же представлен в приложении ниже). Где мы открываем саму программу считываем данные и производим вычисления, результаты функции norm(A) представлены ниже.

Для наглядности, взяв небольшую матрицу 10x10, выведем результат первой итерации:

Матица 10x10:

Выходные данные после операции gaxpy в Octave:

```
>> outOctave
outOctave =
   2.6023
   3.4979
   4.0201
   2.8039
   3.0511
   2.9796
   3.3037
   3.8030
   3.4963
   2.7234
```

Выходные данные после операции gaxpy на C++:

```
>> output
output =

   2.6023
   3.4979
   4.0201
   2.8039
   3.0511
   2.9796
   3.3037
   3.8030
   3.4963
   2.7234
```

Результат нормировки:

```
>> result = norm(output - outOctave);
>> result
result = 2.1342e-06
```

Далее не будет выводить полностью вектор, ввиду своего размера, а будут представлены только результаты нормировки.

Матица 100x100:

```
>> result = norm(output - outOctave);
>> result
result = 2.1852e-05
>>
```

Матица 250x250:

```
>> result = norm(output - outOctave);
>> result
result = 6.4403e-05
>>
```

Матица 500x500:

```
>> result = norm(output - outOctave);
>> result
result = 8.2579e-04
>>
```

Размерность матрицы	Погрешность вычислений
Матрица 10x10	2.1342e-06
Матица 100x100	2.1852e-05
Матица 250x250	6.4403e-05
Матица 500x500 float	8.2579e-04
Матица 500x500 double	1.5890e-04

Таблица 1. Результаты работы операции gaxpy

Вывод

В рамках второй лабораторной работы по изучению параллельных алгоритмов была реализована операция gaxpy на языке C++ в консольном приложении с дальнейшим вычислением и сравнением выходных результатов с работой стандартной библиотеки blas в приложении GNU Octave (GUI).

Результаты оказались отличными от первой работы ввиду особенностей языка и реализации алгоритма вычисления. Метод нормировки выходных данных показал, что разница в вычислениях между вышеуказанными операциями хоть и ничтожно мала заметно увеличилась в отличии от первой работы. Так же были произведены вычисления на разных типах данных double и float. Для первых матриц был применен тип double для повышения точности, для матрицы размером “500x500” был применен тип данных float. Данный способ был реализован ввиду того, что в языке C++ память выделяется в стеке потока, в отличии от C#, где память сразу выделяется в куче, ввиду чего возникают сложности с объявлением чрезмерно большого массива (матрицы). В конце был реализован подсчет для матрицы размером “500x500” с типом double. Данная матица была фактически реализована через вектор.

Список использованных источников

1. Головашкин Д.Л. Векторные алгоритмы вычислительной линейной алгебры: учебной пособие – Самара: Изд-во Самарского университета, 2019. – 76 с.
2. Головашкин Д.Л. Параллельные алгоритмы вычислительной линейной алгебры: учебной пособие – Самара: Изд-во Самарского университета, 2019. – 88 с.
3. Головашкин Д.Л. Модели в теории параллельных вычислений: учебной пособие – Самара: Изд-во Самарского университета, 2019. – 96 с.
4. Голуб Дж., Ван Лоун Ч. Матричные вычисления. Пер. с англ. / Под ред. Воеводина В.В. –М.: Мир, 1999. - 548 с.

Листинг программы

Часть, выполненная на C++

```
#include <iostream>
#include <string>
#include <ctime>
#include <fstream>
#include <cmath>
#include <algorithm>
using namespace std;

int main()
{
    srand(time(NULL));
    setlocale(LC_ALL, "ru");

    //Формула gaxpy z = A*x + y
    //Заем размерность матрицы
    const int n = 250;
    const int m = 250;
    string text = "";

    cout << "Matrix shape: n=" << n << " and m=" << m << "\n";
    cout << "---------\n";
    //Созаем матрицу с заданной размерностью
    float matrix[n][m]{};

    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < m; j++)
        {
            float r = static_cast <float> (rand()) / static_cast <float> (RAND_MAX);
            matrix[i][j] = floor(r*1000000)/ 1000000;
            cout << matrix[i][j] << " ";
            text = text + to_string(matrix[i][j])  + "\n";
        }
        cout << "\n";
    }
    cout << "\n";

    ofstream streamTextOne;
    streamTextOne.open("C:\\Users\\Dark_Monk\\Desktop\\gaxpy\\matrix.txt");
    std::replace(text.begin(), text.end(), ',', '.'); // replace all ',' to '.'
    streamTextOne << text;
    cout << "---------\n";
    cout << "\n";


    // Вектор
    text = "";
    cout << "Вектор x: \n";
    cout << "---------\n";
    float vector[m];

    for (int i = 0; i < m; i++)
    {
        float r = static_cast <float> (rand()) / static_cast <float> (RAND_MAX);
        vector[i] = floor(r * 1000000) / 1000000;
        cout << vector[i] << "\n";
        text = text + to_string(vector[i]) + "\n";
    }
    ofstream streamTextTwo;
```

```
    streamTextTwo.open("C:\\Users\\Dark_Monk\\Desktop\\gaxpy\\vector.txt");
    std::replace(text.begin(), text.end(), ',', '.'); // replace all ',' to '.'
    streamTextTwo << text;
    cout << "---------\n";
    cout << "\n";


    // второй вектор
    text = "";
    cout << "Вектор y: \n";
    cout << "---------\n";
    float vectorNext[n];

    for (int i = 0; i < n; i++)
    {
        float r = static_cast <float> (rand()) / static_cast <float> (RAND_MAX);
        vectorNext[i] = floor(r * 1000000) / 1000000;
        cout << vectorNext[i] << "\n";
        text +=  to_string(vectorNext[i]) + "\n";
    }
    ofstream streamTextThree;
    streamTextThree.open("C:\\Users\\Dark_Monk\\Desktop\\gaxpy\\vectorNext.txt");
    std::replace(text.begin(), text.end(), ',', '.'); // replace all ',' to '.'
    streamTextThree << text;
    cout << "---------\n";
    cout << "\n";


    //Умножение матрицы на вектор
    float outputMatrixV[n]{};

    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < m; j++)
        {
            outputMatrixV[i] += matrix[i][j] * vector[j];
        }
    }
    //Прибавляем вектор, получаем итоговый вектор, пишем в фаил
    text = "";
    cout << "Выходной итоговый вектор после gaxpy: \n";
    cout << "---------\n";
    float output[n];

    for (int i = 0; i < n; i++)
    {
        output[i] = outputMatrixV[i] + vectorNext[i];
        cout << output[i] << "\n";
        text += to_string(output[i]) + "\n";
    }
    ofstream streamTextFour;
    streamTextFour.open("C:\\Users\\Dark_Monk\\Desktop\\gaxpy\\output.txt");
    std::replace(text.begin(), text.end(), ',', '.'); // replace all ',' to '.'
    streamTextFour << text;
    cout << "---------\n";

}
```

Часть, выполненная в Octave

```
clear all; close all;
load matrix.txt;

n = 500;
A = zeros(n, n);

for j = 1:n
  b = (j-1) * n + 1:j * n;
  A(j,:) = matrix(b);
end

load vector.txt;
load vectorNext.txt;
load output.txt;

outOctave = A * vector + vectorNext;

outOctave
output

result = norm(output - outOctave);
result
```