МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САМАРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
ИМЕНИ АКАДЕМИКА С.П. КОРОЛЕВА»

Институт информатики, математики и электроники
Факультет информатики
Кафедра технической кибернетики

Основы параллельных вычислений

Отчет по лабораторной работе №3

Студенты группы 6408 Лобачев Н.В.,
Черемшанцев Л.Д.
Преподаватель Головашкин Д.Л.

САМАРА 2017

1 ВВЕДЕНИЕ

В данной работе мы будем исследовать выполнение операции параллельного gaxpy на кольце, при условии того, что матрица A хранится по блочным строкам. В ходе работы будет получен график зависимости ускорения вычислений от размерности матрицы для выявления особенностей, возникающих при распараллеливании алгоритма.

2 ВЫЧИСЛИТЕЛЬНЫЕ АЛГОРИТМЫ

Для реализации параллельного gaxpy на кольце с матрицей, хранящийся по блочным строкам, мы воспользуемся следующим алгоритмом.

Инициализация {mu – номер процессора, p – количество процессоров, n – размерность, r=n/p; row=(mu-1)r+1:mu*r; x_{loc}=x(row); y_{loc}=y(row); A_{loc}=A(row,:); (left, right)}

for t = 1:p

send(x_{loc}, right);

recv(x_{loc}, left);

tau = mu – t;

if tau≤0 then

tau = tau + p;

$y_{loc}=y_{loc} + A_{loc}(:, (tau-1)r+1:tau*r)* x_{loc}$;

end;

3 ВЫЧИСЛИТЕЛЬНЫЙ ЭКСПЕРИМЕНТ

3.1 Цель эксперимента

Вычислительный эксперимент проводится для выявления особенностей, возникающих при распараллеливании алгоритма.

3.2 Средства эксперимента

Эксперимент проводился с использованием следующих инструментов:

ПК: Intel Core i7-4700HQ 2.40GHz, 8gb RAM DDR4.

ПО: C++, Ubuntu 16.04 64 bit, MPI.

Данные средства эксперимента были выбраны потому, что в языке программирования C++ присутствует реализация векторных вычислений.

3.3 Параметры эксперимента

Эксперимент будем проводить с матрицами и векторами разной размерности, заполненными случайными числами. Начальный размер матриц – 100×100 – подобран так, чтобы ускорение приблизительно равнялось единице. Максимальный размер матриц – 900×900 – обусловлен особенностями MPI на , из-за которых происходит ошибка сегментации при большем размере матриц. Результаты представлены в виде таблицы 1, содержащей значения времени работы параллельного и последовательного алгоритма, и на рисунке 1, на котором отображена зависимость ускорения вычислений ($\frac{\text{Время последовательного алгоритма}}{\text{Время параллельного алгоритма}}$) от размерности матрицы и векторов.

3.4 Теоретические ожидания

В ходе вычислительного эксперимента ожидаются следующие результаты: при больших размерностях ускорение вычисления должно стремиться к двум. Также возможно, что на маленьких размерностях параллельный алгоритм будет медленные последовательного, что обусловлено затратами на пересылку данных в MPI.

3.5 Результаты

Таблица 1 – Время работы алгоритмов

Размер перемножаемых матриц	Время работы, с	
	Последовательный	Параллельный
100	0.000036	0.000047
200	0.000145	0.000095
300	0.000277	0.000202
400	0.000492	0.000317
500	0.000895	0.000388
600	0.001072	0.000554
700	0.001936	0.000714
800	0.001977	0.000973
900	0.002942	0.001239

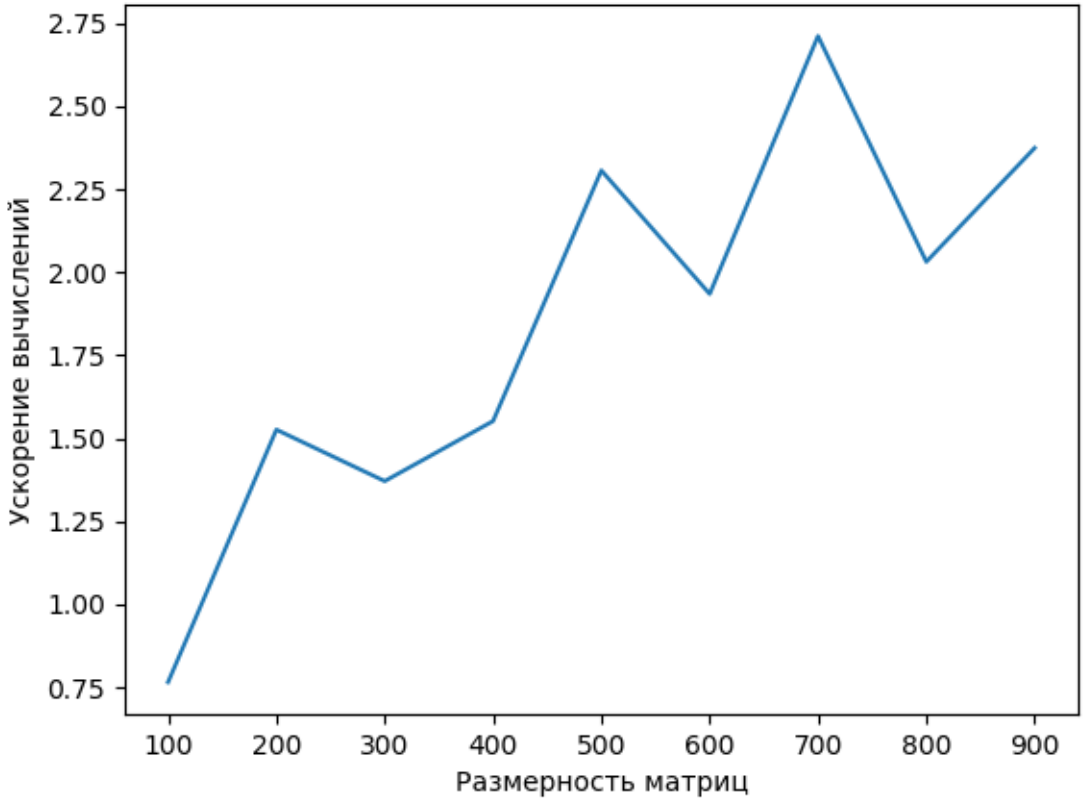

Рисунок 1 – График зависимости ускорения вычислений от размерности матрицы

3.6 Описание результатов

Из графика, изображенного на рисунке 1, видно, что при увеличении размерности увеличивается ускорение. Небольшие проседания ускорения обусловлены работой операционной системы.

3.7 Анализ результатов

На основании проведенных экспериментов после оценки результатов можно сделать следующие выводы.

Все результаты являются приблизительными, так как на времени работы алгоритма сказывается и загруженность процессора в данный момент. Поэтому в разный момент времени, вычисления для одного и того же алгоритма для той же размерности матрицы занимает различное время.

Рост и спад ускорения в эксперименте происходит в зависимости от доли параллельного кода, который, в свою очередь, зависит от размера матрицы.

4 ЗАКЛЮЧЕНИЕ

Цель эксперимента в ходе лабораторной работы была нами достигнута. Мы выяснили, что при распараллеливании вычислительного алгоритма следует учитывать размерности матриц и векторов, которые планируется использовать. Также необходимо грамотно оценивать значения ускорения от использования параллельного алгоритма при написании программ.

5 ПРИЛОЖЕНИЕ А. КОД ПОСЛЕДОВАТЕЛЬНОЙ ПРОГРАММЫ

```
#include <cstdlib>
#include <ctime>
#include <iostream>

void printMatrix(double** m, int n)
{
	for (int i = 0; i < n; i++) {
		for (int j = 0; j < n; j++) {
			std::cout << m[i][j] << " ";
		}
		std::cout << std::endl;
	}
}
void printVector(double* v, int n)
{
	for (int i = 0; i < n; i++) {
		std::cout << v[i] << std::endl;
	}
	std::cout << std::endl;
}
double fRand(double fMin, double fMax) {
	double f = (double)rand() / RAND_MAX;
	return fMin + f*(fMax - fMin);
}
double** createMatrix(int n) {
	double** matrix = new double*[n];
	for (int i = 0; i < n; i++) {
		matrix[i] = new double[n];
	}
	return matrix;
}
double** randomMatrix(double fMin, double fMax, int n) {
	srand(time(NULL));
	double** matrix = createMatrix(n);
	for (int i = 0; i < n; i++) {
```

```
            for (int j = 0; j < n; j++) {
                  matrix[i][j] = fRand(fMin, fMax);
            }
      }

      for (int i = 0; i < n; i++) {
            double summa = 0.0;
            for (int j = 0; j < n; j++) {
                  summa += abs(matrix[i][j]);
            }
            matrix[i][i] = summa;
      }

      return matrix;
}
double* randomVector(double fMin, double fMax, int n) {
      srand(time(NULL));
      double* vector = new double[n];
      for (int i = 0; i < n; i++) {
            vector[i] = fRand(fMin, fMax);
      }
      return vector;
}
void rowGaxpy(double** a, int n, double* x, double* y) {
      for (int i = 0; i < n;i++) {
            for (int j = 0; j < n;j++) {
                  y[i] += a[i][j]*x[j];
            }
      }
}

int main() {
      double minR = 1;
      double maxR = 100;

      for(int i=100;i<=10100;i+=500) {
            std::clock_t start;
            double duration;
```

```
            double** a = new double*[i];
            for(int j=0;j<i;j++)
                  a[i] = new double[i];

            double* x = new double[i];
            double* y = new double[i];

            a = randomMatrix(minR, maxR, i);
            x = randomVector(minR,maxR,i);
            y = randomVector(minR, maxR, i);

            start = std::clock();
            rowGaxpy(a,i,x,y);
            duration = (std::clock() - start) / (double)CLOCKS_PER_SEC;
            std::cout << i <<" " << duration << std::endl << std::endl;
            for(int j=0;j<i;j++)
                  delete[] a[i];
            delete[] a;
            delete[] x;
            delete[] y;
      }
      return 0;
}
```

5 ПРИЛОЖЕНИЕ А. КОД ПАРАЛЛЕЛЬНОЙ ПРОГРАММЫ

```
#include <stdio.h>
#include <stdlib.h>
#include "/usr/local/include/mpi.h"
#define NMAX 900

int main(int argc, char** argv){
      int procRank,procNum, N=NMAX, r,*row,i,j,left,right,t,T;
      double start_time,end_time;

      double *a_loc,*y_loc,*x_loc;

      double a[NMAX*NMAX],y[NMAX],x[NMAX];

      for(i=0;i<N;i++){
            for(int j=0;j<N;j++){
                  a[i*N+j]=1.0;
            }
            y[i]=1.0;
            x[i]=1.0;
      }

      MPI_Status status;

      MPI_Init(&argc,&argv);
      MPI_Comm_size(MPI_COMM_WORLD,&procNum);
      MPI_Comm_rank(MPI_COMM_WORLD,&procRank);

      if(procRank-1<0) left=procNum-1;
      else left=procRank-1;
      if(procRank==procNum-1) right=0;
      else right=procRank+1;

      r=N/procNum;

      a_loc=(double *) malloc(r*N*sizeof(double));
      x_loc=(double *) malloc(r*sizeof(double));
      y_loc=(double *) malloc(r*sizeof(double));
```

```
row=(int *) malloc(r*sizeof(int));

for(i=0;i<r;i++){
      row[i]=procRank*r+i;
}



for(int i=0;i<r;i++){
      y_loc[i]=y[row[i]];
      x_loc[i]=x[row[i]];
      for(j=0;j<N;j++){
            a_loc[i*N+j]=a[row[i]*N+j];
      }
}

start_time=MPI_Wtime();

for(t=0;t<procNum;t++){
      MPI_Send(x_loc,r,MPI_DOUBLE,right,0,MPI_COMM_WORLD);
      MPI_Recv(x_loc,r,MPI_DOUBLE,left,0,MPI_COMM_WORLD,&status);
      T=procRank-t;
      if(T<=0) T=T+procNum;
      for(i=0;i<r;i++){
            for(j=0;j<r;j++){
                  y_loc[i]+=a_loc[i*N+(T-1)*r+j]*x_loc[j];


            }
      }
}

MPI_Gather(y_loc,r,MPI_DOUBLE,y,r,MPI_DOUBLE,0,MPI_COMM_WORLD);
end_time=MPI_Wtime();
end_time=end_time-start_time;

if(procRank==0){
      printf("\n Time: %f\n",end_time);
}
```

```
        MPI_Finalize();
        return 0;
}
```