**ОТЧЕТ О ВЫПОЛНЕНИИ ЛАБОРАТОРНОЙ РАБОТЫ**

«Параллельная реализация решения системы   
линейных алгебраических уравнений с помощью OpenMP»

студентки 2 курса, 18209 группы

**Сологубовой Анастасии**

Используя **метод минимальных невязок**, решим систему линейных алгебраических уравнений. В методе минимальных невязок преобразование решения на каждом шаге задается следующими формулами:

,

,

, где

.

**Число доступных на процессоре ядер: 8.**

**Вариант 1**

Для каждого распараллеливаемого цикла создается отдельная параллельная секция #pragma omp parallel for.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Число задействованных ядер** | 1 | 2 | 4 | 8 | 12 |
| Время выполнения вычислений | 24,512213 | 13,432044 | 8,677915 | 7,126785 | 10,443215 |
| Ускорение | 0 | 1,824905651 | 2,824666178 | 3,439448924 | 2,347190305 |
| Эффективность | 0 | 0,9124528255 | 0,7061665446 | 0,4299311155 | 0,1955991921 |
| Диаграмма | | | Диаграмма | | |

**Вариант 2**

Создается одна параллельная секция #pragma omp parallel, охватывающая весь итерационный алгоритм.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Число задействованных ядер** | 1 | 2 | 4 | 8 | 12 |
| Время выполнения вычислений | 20,099386 | 8,157286 | 6,050708 | 4,094172 | 3,767990 |
| Ускорение | 0 | 2,463979564 | 3,321823826 | 4,909267613 | 5,33424611 |
| Эффективность | 0 | 1,231989782 | 0,8304559566 | 0,6136584516 | 0,4445205092 |
| Диаграмма | | | Диаграмма | | |

**Исследование на определение оптимальных параметров   
#pragma omp for schedule(...)**

После замера времени выполнения программы на 4 потоках с разными параметрами #pragma omp for schedule(...) получили следующие приблизительные данные:

|  |  |  |
| --- | --- | --- |
| Первый параметр schedule | Число итераций цикла на поток = 4 | Число итераций цикла на поток = 625 |
| Без schedule | 5,8 | |
| static | 9,9 | 5,9 |
| dynamic | 6,1 | 5,7 |
| guided | 8,5 | 5,7 |
| runtime | 5,7 | |

Т.к. итерации циклов в данной программе равносильны, т.е. выполняются примерно за одно и то же время, то использование #pragma omp for schedule бессмысленно. Применение директивы pragma с этим условием на практике показало, что время выполнения программы почти не менялось.

**Вывод:** Второй вариант распараллеливания с одной параллельной секцией #pragma omp parallel, охватывающей весь итерационный алгоритм, оказался более выигрышным по ускорению и эффективности, чем первый вариант, использующий #pragma omp parallel for перед каждым циклом for. Во втором варианте #pragma omp for лишь указывает, что итерации цикла следует разделить между потоками внутри существующей секции, поэтому время на вход в параллельную секцию, как в первом варианте, не затрачивается.

**ПРИЛОЖЕНИЕ 1 (Вариант задания 1)**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <omp.h>

#include <stdio.h> //printf

#include <stdlib.h>

#include <math.h>

#include <time.h>

#define N 2500

#define Eps 0.000001

//metod minimalnih nevyazok

void printVec(float \*vec, int size) {

for (int i = 0; i < size; i++){

printf("%f ", vec[i]);

}

printf("\n");

}

int main() {

int size = N;

float E = Eps;

float tau = 0.01;

float \*A = (float\*)malloc(size \* size \* sizeof(float));

float x[size], b[size], y[size], Ay[size], tmp[size];

FILE \*inA = fopen("inData/matA.bin", "rb");

FILE \*inB = fopen("inData/vecB.bin", "rb");

fread(A, sizeof(float), size \* size, inA);

fread(b, sizeof(float), size, inB);

fclose(inA);

fclose(inB);

int i = 0, j = 0;

float lenY = 0, lenB = 0, res = 1;

float scal1 = 0, scal2 = 0;

for (i = 0; i < size; i++){

x[i] = 0;

lenB += (b[i] \* b[i]);

}

lenB = sqrt(lenB);

struct timespec start, end;

clock\_gettime(CLOCK\_MONOTONIC\_RAW, &start);

while (res >= E){

#pragma omp single

{

scal1 = 0;

scal2 = 0;

tau = 0;

lenY = 0;

}

#pragma omp parallel for

for (i = 0; i < size; i++){

y[i] = 0;

for (j = 0; j < size; j++){

y[i] += (\*(A + i \* size + j)) \* (\*(x + j));

//y[i] += A[i \* size + j] \* x[j];

}

}

#pragma omp parallel for

for (i = 0; i < size; i++) {

y[i] = y[i] - b[i];

}

#pragma omp parallel for reduction(+: lenY)

for (i = 0; i < size; i++) {

lenY += (y[i] \* y[i]);

}

#pragma omp parallel for

for (i = 0; i < size; i++){

Ay[i] = 0;

for (j = 0; j < size; j++){

Ay[i] += (\*(A + i \* size + j)) \* (\*(y + j));

//Ay[i] += A[i \* size + j] \* y[j];

}

}

#pragma omp parallel for reduction(+: scal1)

for (i = 0; i < size; i++) {

scal1 += (y[i] \* Ay[i]);

}

#pragma omp single

{

scal1 = scal1;

}

#pragma omp parallel for reduction(+: scal2)

for (i = 0; i < size; i++) {

scal2 += (Ay[i] \* Ay[i]);

}

#pragma omp single

{

tau = scal1 / scal2;

}

#pragma omp parallel for

for (i = 0; i < size; i++) {

x[i] = x[i] - tau \* y[i];

}

#pragma omp single

{

lenY = sqrt(lenY);

res = lenY/lenB;

}

}

clock\_gettime(CLOCK\_MONOTONIC\_RAW, &end);

printf("Time taken: %lf sec.\n",end.tv\_sec-start.tv\_sec+ 0.000000001\*(end.tv\_nsec-start.tv\_nsec));

FILE \*inX = fopen("inData/vecX.bin", "rb");

fread(tmp, sizeof(float), size, inX);

fclose(inX);

/\* float difference = -1;

for (i = 0; i < size; i++){

double d = abs(tmp[i] - x[i]);

if (d > difference) difference = d;

//printf("x[%d] dif: %lf\n", i, d);

}

//printVec(x, size);

printf("%lf\n", difference);

\*/

free(A);

return 0;

}

**ПРИЛОЖЕНИЕ 2 (Вариант задания 2)**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <omp.h>

#include <stdio.h> //printf

#include <stdlib.h>

#include <math.h>

#include <time.h>

#define N 2500

#define Eps 0.000001

//metod minimalnih nevyazok

void printVec(float \*vec, int size) {

for (int i = 0; i < size; i++){

printf("%f ", vec[i]);

}

printf("\n");

}

int main() {

int size = N;

float E = Eps;

float tau = 0.01;

float \*A = (float\*)malloc(size \* size \* sizeof(float));

float x[size], b[size], y[size], Ay[size], tmp[size];

FILE \*inA = fopen("inData/matA.bin", "rb");

FILE \*inB = fopen("inData/vecB.bin", "rb");

fread(A, sizeof(float), size \* size, inA);

fread(b, sizeof(float), size, inB);

fclose(inA);

fclose(inB);

int i = 0, j = 0;

float lenY = 0, lenB = 0, res = 1;

float scal1 = 0, scal2 = 0;

for (i = 0; i < size; i++){

x[i] = 0;

lenB += (b[i] \* b[i]);

}

lenB = sqrt(lenB);

struct timespec start, end;

clock\_gettime(CLOCK\_MONOTONIC\_RAW, &start);

while (res >= E){

#pragma omp parallel

{

#pragma omp single

{

scal1 = 0;

scal2 = 0;

tau = 0;

lenY = 0;

}

#pragma omp for

for (i = 0; i < size; i++){

y[i] = 0;

for (j = 0; j < size; j++){

y[i] += (\*(A + i \* size + j)) \* (\*(x + j));

//y[i] += A[i \* size + j] \* x[j];

}

}

#pragma omp for

for (i = 0; i < size; i++) {

y[i] = y[i] - b[i];

}

#pragma omp for reduction(+: lenY)

for (i = 0; i < size; i++) {

lenY += (y[i] \* y[i]);

}

#pragma omp for

for (i = 0; i < size; i++){

Ay[i] = 0;

for (j = 0; j < size; j++){

Ay[i] += (\*(A + i \* size + j)) \* (\*(y + j));

//Ay[i] += A[i \* size + j] \* y[j];

}

}

#pragma omp for reduction(+: scal1)

for (i = 0; i < size; i++) {

scal1 += (y[i] \* Ay[i]);

}

#pragma omp single

{

scal1 = scal1;

}

#pragma omp for reduction(+: scal2)

for (i = 0; i < size; i++) {

scal2 += (Ay[i] \* Ay[i]);

}

#pragma omp single

{

tau = scal1 / scal2;

}

#pragma omp for

for (i = 0; i < size; i++) {

x[i] = x[i] - tau \* y[i];

}

#pragma omp single

{

lenY = sqrt(lenY);

res = lenY/lenB;

}

}

}

clock\_gettime(CLOCK\_MONOTONIC\_RAW, &end);

printf("Time taken: %lf sec.\n",end.tv\_sec-start.tv\_sec+ 0.000000001\*(end.tv\_nsec-start.tv\_nsec));

FILE \*inX = fopen("inData/vecX.bin", "rb");

fread(tmp, sizeof(float), size, inX);

fclose(inX);

/\* float difference = -1;

for (i = 0; i < size; i++){

double d = abs(tmp[i] - x[i]);

if (d > difference) difference = d;

//printf("x[%d] dif: %lf\n", i, d);

}

//printVec(x, size);

printf("%lf\n", difference);

\*/

free(A);

return 0;

}