###### МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

###### ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

###### НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

###### Факультет информационных технологий

**Кафедра параллельных вычислений**

ОТЧЕТ

О ВЫПОЛНЕНИИ ЛАБОРАТОРНОЙ РАБОТЫ

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

студента 2 курса, группы 19212

**Хомченко Станислава Евгеньевича**

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:

Ажбаков Артём Альбертович

# **ЗАДАНИЕ**

1. Последовательную программу из лабораторной работы 1, реализующую итерационный алгоритм решения системы линейных алгебраических уравнений вида Ax=b, распараллелить с помощью OpenMP. Реализовать два варианта программы:

* Вариант 1: для каждого распараллеливаемого цикла создается отдельная параллельная секция #pragma omp parallel for,
* Вариант 2: создается одна параллельная секция #pragma omp parallel, охватывающая весь итерационный алгоритм.

Уделить внимание тому, чтобы при запуске программы на различном числе OpenMP-потоков решалась одна и та же задача (исходные данные заполнялись одинаковым образом).

1. Замерить время работы двух вариантов программы при использовании различного числа процессорных ядер: от 1 до числа доступных в узле. Построить графики зависимости времени работы программы, ускорения и эффективности распараллеливания от числа используемых ядер. Исходные данные и параметры задачи подобрать таким образом, чтобы решение задачи на одном ядре занимало не менее 30 секунд.
2. Провести исследование на определение оптимальных параметров #pragma omp for schedule(...) при некотором фиксированном размере задачи и количестве потоков.
3. На основании полученных результатов сделать вывод о целесообразности использования первого или второго варианта.

# **ОПИСАНИЕ РАБОТЫ**

В качестве алгоритма для реализации был выбран метод простой итерации:

# Написание программы на языке С++(последовательная).

1. Провести исследование на определение оптимальных параметров #pragma omp for schedule(…).
2. Написание двух программ при помощи OpenMP(параллельные).
3. Замер времени, нахождение ускорения и эффективности.
4. Вывод о проделанной работе.

# **Приложение 1.** *Листинг программы Вариант 1: для каждого распараллеливаемого цикла создаётся отдельная параллельная секция #pragma omp parallel for*

#include <iostream>  
#include <stdlib.h>  
#include <math.h>  
#include <omp.h>  
  
using namespace std;  
  
#define N 15000  
#define eps 1e-6  
  
void matrMul(double\* A, double\* x, double\* xnew) {  
#pragma omp parallel for //schedule (static)=(//type)  
 for (int i = 0; i < N; i++) {  
 xnew[i] = 0;  
 for (int j = 0; j < N; j++) {  
 xnew[i] += A[j + i \* N] \* x[j];  
 }  
 }  
}  
  
int main() {  
 double startTime = omp\_get\_wtime( );  
  
 double \*x = NULL, \*b = NULL, \*xnew = NULL, tau = 0.0001, norm = 0, norm\_b = 0, sum = 0;  
  
 x = (double \*)calloc(N, sizeof(double));  
 b = (double \*)malloc((N + 1) \* sizeof(double));  
 xnew = (double \*)calloc(N, sizeof(double));  
  
#pragma omp parallel for  
 for (int i = 0; i < N; ++i) {  
 b[i] = N + 1;  
 }  
  
 double \*A = nullptr;  
 A = (double \*)malloc(N \* N \* sizeof(double));  
#pragma omp parallel for  
 for (int i = 0; i < N; i++) {  
 for (int j = 0; j < N; j++) {  
 A[i \* N + j] = (i == j) ? 2.0 : 1.0;  
 }  
 }  
  
#pragma omp parallel for reduction(+:sum)  
 for (int i = 0; i < N; i++)  
 sum += b[i] \* b[i];  
 norm\_b = sqrt(sum);  
  
 matrMul(A, x, xnew);  
#pragma omp parallel for  
 for (int i = 0; i < N; i++)  
 xnew[i] = xnew[i] - b[i];  
#pragma omp parallel for reduction(+:sum)  
 for (int i = 0; i < N; i++)  
 sum += xnew[i] \* xnew[i];  
 double norm\_0 = sqrt(sum);  
#pragma omp parallel for  
 for (int i = 0; i < N; i++)  
 xnew[i] = -tau \* xnew[i];  
#pragma omp parallel for  
 for (int i = 0; i < N; i++)  
 xnew[i] = x[i] + xnew[i];  
#pragma omp parallel for  
 for (int i = 0; i < N; i++)  
 x[i] = xnew[i];  
  
 sum = 0;  
 matrMul(A, x, xnew);  
#pragma omp parallel for  
 for (int i = 0; i < N; i++)  
 xnew[i] = xnew[i] - b[i];  
#pragma omp parallel for reduction(+:sum)  
 for (int i = 0; i < N; i++)  
 sum += xnew[i] \* xnew[i];  
 norm = sqrt(sum);   
 if (norm\_0 <= norm)  
 tau = -tau;  
  
 norm /= norm\_b;  
  
 while (norm >= eps) {  
 sum = 0;  
 matrMul(A, x, xnew);  
  
#pragma omp parallel for  
 for (int i = 0; i < N; i++)  
 xnew[i] = xnew[i] - b[i];  
  
#pragma omp parallel for reduction(+:sum)  
 for (int i = 0; i < N; i++)  
 sum += xnew[i] \* xnew[i];  
 norm = sqrt(sum) / norm\_b;  
  
#pragma omp parallel for  
 for (int i = 0; i < N; i++)  
 xnew[i] = -tau \* xnew[i];  
  
#pragma omp parallel for  
 for (int i = 0; i < N; i++)  
 xnew[i] = x[i] + xnew[i];  
  
#pragma omp parallel for  
 for (int i = 0; i < N; i++)  
 x[i] = xnew[i];   
 }  
  
 double endTime = omp\_get\_wtime( );  
  
 cout << "Res: " << "\n";  
 for (int i = 0; i < N; i++) {  
 cout << "x[" << i << "] = " << xnew[i] << "\n";  
 }  
 cout << "Time: "<< (double)(endTime - startTime) << endl;  
  
 return 0;  
}

# **Приложение 2.** *Листинг программы Вариант 2: создается секция #pragma omp parallel, охватывающая весь итерационный алгоритм*

#include <iostream>  
#include <stdlib.h>  
#include <math.h>  
#include <omp.h>  
  
using namespace std;  
  
#define N 15000  
#define eps 1e-6  
  
void matrMul(double\* A, double\* x, double\* xnew) {  
#pragma omp for  
 for (int i = 0; i < N; i++) {  
 xnew[i] = 0;  
 for (int j = 0; j < N; j++) {  
 xnew[i] += A[j + i \* N] \* x[j];  
 }  
 }  
  
}  
  
int main() {  
 double startTime = omp\_get\_wtime( );  
  
 double \*x = nullptr, \*b = nullptr, \*xnew = nullptr, tau = 0.0001, norm = 0, norm\_b = 0, sum = 0;  
  
 x = (double \*)calloc(N, sizeof(double));  
 b = (double \*)malloc((N + 1) \* sizeof(double));  
 xnew = (double \*)calloc(N, sizeof(double));  
  
#pragma omp for  
 for (int i = 0; i < N; ++i) {  
 b[i] = N + 1;  
 }  
  
 double \*A = nullptr;  
 A = (double \*)malloc(N \* N \* sizeof(double));  
 for (int i = 0; i < N; i++) {  
 for (int j = 0; j < N; j++) {  
 A[i \* N + j] = (i == j) ? 2.0 : 1.0;  
 }  
 }  
  
 for (int i = 0; i < N; i++)  
 sum += b[i] \* b[i];  
 norm\_b = sqrt(sum);  
  
 matrMul(A, x, xnew);  
 for (int i = 0; i < N; i++)  
 xnew[i] = xnew[i] - b[i];  
 for (int i = 0; i < N; i++)  
 sum += xnew[i] \* xnew[i];  
 double norm\_0 = sqrt(sum);  
 for (int i = 0; i < N; i++)  
 xnew[i] = -tau \* xnew[i];  
 for (int i = 0; i < N; i++)  
 xnew[i] = x[i] + xnew[i];  
 for (int i = 0; i < N; i++)  
 x[i] = xnew[i];  
  
 sum = 0;  
 matrMul(A, x, xnew);  
 for (int i = 0; i < N; i++)  
 xnew[i] = xnew[i] - b[i];  
 for (int i = 0; i < N; i++)  
 sum += xnew[i] \* xnew[i];  
 norm = sqrt(sum);  
  
 if (norm\_0 <= norm)  
 tau = -tau;  
  
 norm /= norm\_b;  
  
#pragma omp parallel  
 while (norm >= eps) {  
 sum = 0;  
 matrMul(A, x, xnew);  
  
#pragma omp for  
 for (int i = 0; i < N; i++)  
 xnew[i] = xnew[i] - b[i];  
  
#pragma omp shared(sum) for reduction(+:sum)  
 for (int i = 0; i < N; i++)  
 sum += xnew[i] \* xnew[i];  
 norm = sqrt(sum) / norm\_b;  
  
#pragma omp for  
 for (int i = 0; i < N; i++)  
 xnew[i] = -tau \* xnew[i];  
  
#pragma omp for  
 for (int i = 0; i < N; i++)  
 xnew[i] = x[i] + xnew[i];  
  
#pragma omp for  
 for (int i = 0; i < N; i++)  
 x[i] = xnew[i];  
 }  
  
 double endTime = omp\_get\_wtime( );  
  
 cout << "Res: " << "\n";  
 for (int i = 0; i < N; i++) {  
 cout << "x[" << i << "] = " << xnew[i] << "\n";  
 }  
 cout << "Time: "<< (double)(endTime - startTime) << endl;  
  
 return 0;  
}

# **Приложение 3.** *Графики программ*

# Время исполнения программы на Си на 1 ядре составляет 32,6311 sec.

ускорение: Sp = T1 / Tp, где T1 – время работы последовательной программы на 1 ядре. Tp - время работы параллельной программы на p ядрах. Эффективность: Ep = (Sp / p) \* 100%

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Количество ядер | Вариант 1 | Вариант 2 | Ускорение 1 | Ускорение 2 | Эффективность 1 | Эффективность 2 |
| 1 | 36,7661 | 35,5383 | 0,88753226 | 0,918195299 | 88,75323% | 91,81953% |
| 2 | 19,853 | 19,2764 | 1,64363572 | 1,692800523 | 82,18179% | 84,64003% |
| 4 | 12,591 | 11,3255 | 2,591621 | 2,881206128 | 64,79052% | 72,03015% |
| 8 | 8,526 | 7,38381 | 3,82724607 | 4,419276769 | 47,84058% | 55,24096% |
| 12 | 8,26005 | 6,9502 | 3,95047245 | 4,694987195 | 32,92060% | 39,12489% |

Первый вариант – синий, второй – оранжевый.

Количество потоков 8.

|  |  |  |  |
| --- | --- | --- | --- |
|  | static | guided | dynamic |
| 64 | 5,5711 | 6,13662 | 5,66602 |
| 128 | 5,56984 | 5,79151 | 6,07904 |
| 256 | 5,60768 | 5,80332 | 5,80378 |
| 512 | 5,96601 | 6,22135 | 5,97647 |

# **ВЫВОД**

За время освоения лабораторной я научился пользоваться OpenMP. Время,

полученное при замерах, во втором варианте работы программы(создается

одна параллельная секция #pragma omp parallel, охватывающая весь

итерационный алгоритм) быстрее, чем в первом варианте (для каждого

распараллеливаемого цикла создается отдельная параллельная секция

#pragma omp parallel for).