**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 ПРАКТИЧЕСКОЙ РАБОТЫ**

«ПАРАЛЛЕЛЬНАЯ РЕАЛИЗАЦИЯ РЕШЕНИЯ СИСТЕМЫ ЛИНЕЙНЫХ АЛГЕБРАИЧЕСКИХ УРАВНЕНИЙ С ПОМОЩЬЮ OPENMP»

студентки 2 курса, группы 22204

**Клочихиной Софьи Павловны**

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:

А. Ю. Власенко

Новосибирск 2024

**СОДЕРЖАНИЕ**

ЦЕЛЬ 3

ЗАДАНИЕ 4

ОПИСАНИЕ РАБОТЫ 5

ЗАКЛЮЧЕНИЕ 9

ПРИЛОЖЕНИЕ.

Полный листинг параллельной программы на C++ 10

Полный листинг скрипта SLURM для параллельной программы 13

Полный листинг параллельной программы на C++ со schedule 14

Полный листинг скрипта SLURM для программы со schedule 16

**ЦЕЛЬ**

1. Реализовать решение системы линейных алгебраических уравнений вида методом простых итераций с использованием технологии OpenMP.

**ЗАДАНИЕ**

ВАРИАНТ 1

1. Последовательную программу из предыдущей практической работы, реализующую итерационный алгоритм решения системы линейных алгебраических уравнений вида Ax=b, распараллелить с помощью OpenMP.   
   ОБЯЗАТЕЛЬНОЕ УСЛОВИЕ: создается одна параллельная секция #pragma omp parallel, охватывающая весь итерационный алгоритм.
2. Замерить время работы программы на вычислительном кластере НОЦ «Газпромнефть-НГУ» на 1, 2, 4, 8, 12, 16 потоках. 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.
3. Провести исследование на определение оптимальных параметров #pragma omp for schedule(...) при некотором фиксированном размере задачи и количестве потоков.

**ОПИСАНИЕ РАБОТЫ**

1. На языке программирования С++ были написаны 2 программы: параллельная и параллельная со schedule, в которых использовалась библиотека OpenMP, позволяющая сделать программу параллельной. Время выполнения было замерено с помощью функции omp\_get\_wtime();
2. Был написан скрипт с использованием SLURM, с помощью которого обе программы компилировались и запускались:





1. Минимальные выходные данные параллельной программы:

* На 1 потоке:



* На 2 потоках:



* На 4 потоках:



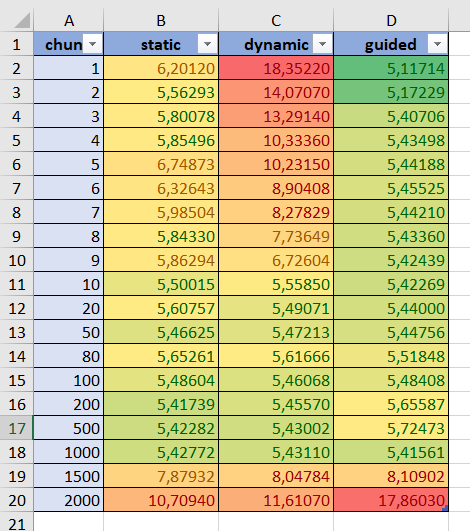
* На 8 потоках:



* На 16 потоках:



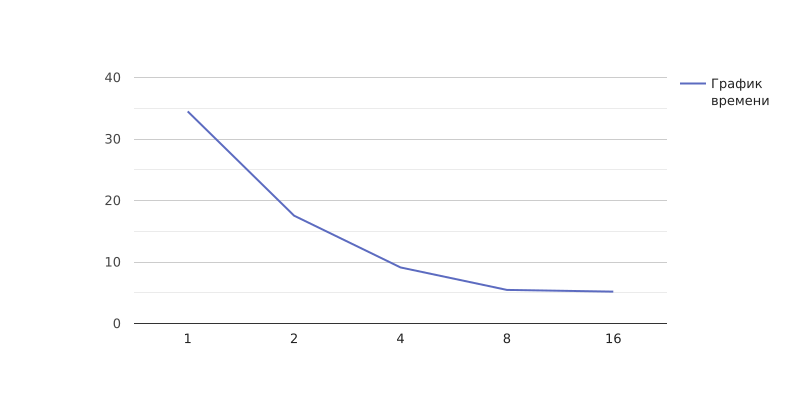
1. Также было проведено исследование на определение оптимальных параметров schedule при N = 4000 и на 4 потоках:

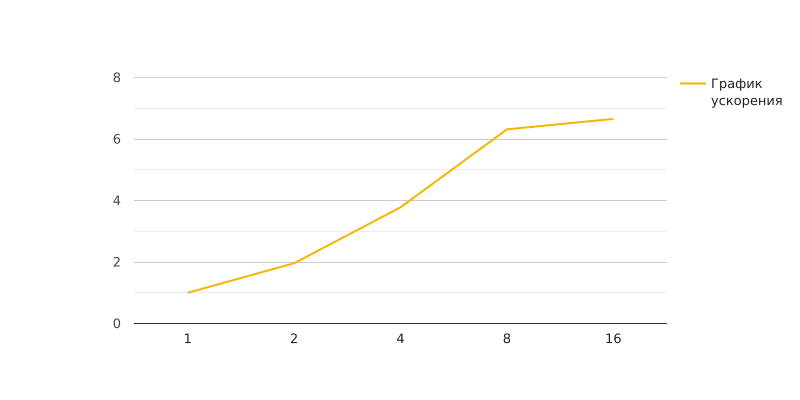


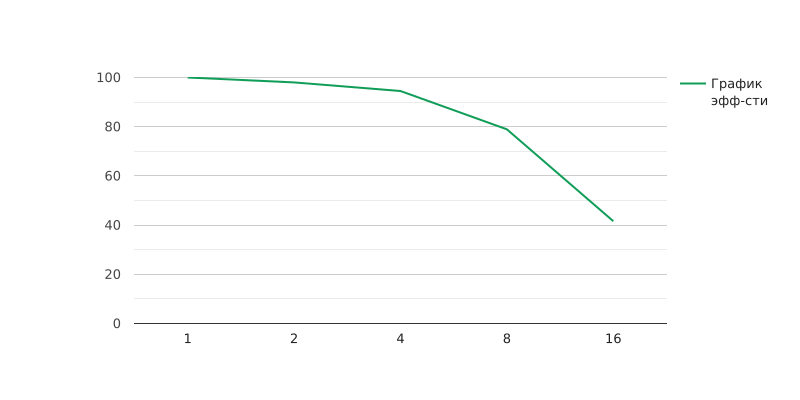
1. Было выполнено построение графиков времени, ускорения, эффективности и график параметров schedule, где:

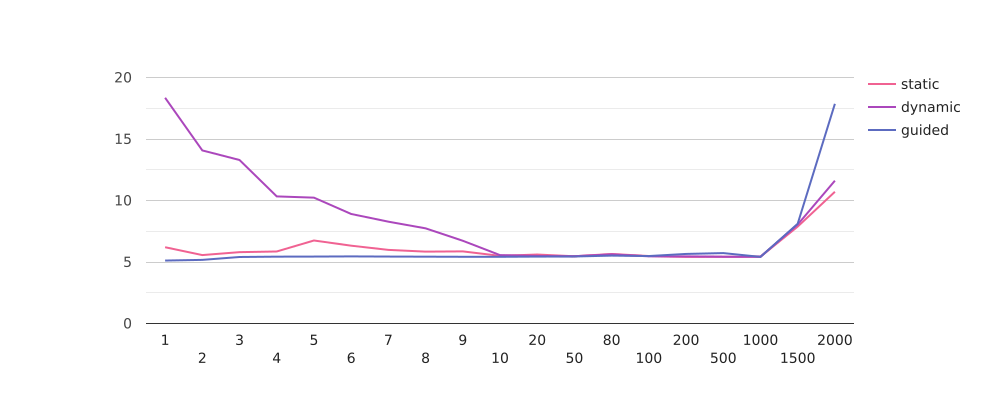
**Ускорение**: Sp = T1 / Tp, где T1 - время работы ПОСЛЕДОВАТЕЛЬНОЙ программы. Tp - время работы параллельной программы на p процессах/потоках.

**Эффективность**: Ep = Sp / p \* 100%.









**ЗАКЛЮЧЕНИЕ**

Распараллеленная программа работает в несколько раз быстрее последовательной за счёт того, что потоки обрабатывают свои данные одновременно и частями.

Как выяснилось опытным путём, оптимальным параметром schedule является guided с размером chunk = 1.

1. **Приложение 1.** Полный листинг параллельной программы на C++.

#**include** <iostream>

#**include** <cmath>

#**include** <omp.h>

const double EPSILON = 0.000001;

const int N = 7000;

const float TAU = 0.00001;

const int MAX\_ITERATION\_NUM = 10000;

void **createMatrix**(double\* matrix) {

**for** (int i = 0; i < N; i++) {

**for** (int j = i; j < N; j++) {

matrix[i \* N + j] = (double)rand() / RAND\_MAX \* 2.0 - 1.0;

**if** (i == j)

matrix[i \* N + j] = matrix[i \* N + j] + N;

}

**for** (int j = 0; j < i; j++)

matrix[i \* N + j] = matrix[j \* N + i];

}

}

void **createVector**(double\* vector) {

**for** (int i = 0; i < N; i++)

vector[i] = (double)rand() / RAND\_MAX \* 10.0 - 5.0;

}

void **setMatrixParts**(int\* linesPerThread, int\* offsets, int size){

int lines = N / size, rest = N % size, offset = 0;

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

rest ? (linesPerThread[i] = lines + 1, rest--) : linesPerThread[i] = lines;

offsets[i] = offset;

offset += linesPerThread[i];

}

}

double **countAccuracy**(const double\* vector) {

double accuracy = 0;

**for** (int i = 0; i < N; i++)

accuracy += vector[i] \* vector[i];

**return** sqrt(accuracy);

}

int **main**(int argc, char\*\* argv) {

srand(time(nullptr));

double begin, end;

//omp\_set\_num\_threads(4);

int size = omp\_get\_max\_threads();

**auto**\* A = **new** double[N \* N];

**auto**\* x = **new** double[N];

**auto**\* x\_new = **new** double[N];

**auto**\* b = **new** double[N];

**auto**\* linesPerThread = **new** int[size];

**auto**\* offsets = **new** int[size];

int iter\_count = 0;

double accuracy = 1;

setMatrixParts(linesPerThread, offsets, size);

begin = omp\_get\_wtime();

createMatrix(A);

createVector(x);

createVector(b);

double B\_accuracy = countAccuracy(b);

int i, j;

#**pragma** omp parallel shared(accuracy) private(i, j)

{

int thread\_id = omp\_get\_thread\_num();

int first\_index = offsets[thread\_id];

int last\_index = first\_index + linesPerThread[thread\_id];

**while** (accuracy > EPSILON && iter\_count < MAX\_ITERATION\_NUM) {

**for** (i = first\_index; i < last\_index; i++) {

x\_new[i] = 0;

**for** (j = 0; j < N; j++)

x\_new[i] += A[i \* N + j] \* x[j];

}

**for** (i = first\_index; i < last\_index; i++)

x\_new[i] = x\_new[i] - b[i];

#**pragma** omp single

accuracy = 0;

#**pragma** omp for reduction(+: accuracy)

**for** (i = 0; i < N; i++)

accuracy += x\_new[i] \* x\_new[i];

**for** (i = first\_index; i < last\_index; i++) {

x\_new[i] = x[i] - TAU \* x\_new[i];

x[i] = x\_new[i];

}

#**pragma** omp single

{

accuracy = sqrt(accuracy) / B\_accuracy;

iter\_count++;

}

}

}

**delete**[] A;

**delete**[] x;

**delete**[] x\_new;

**delete**[] b;

**delete**[] linesPerThread;

**delete**[] offsets;

end = omp\_get\_wtime();

std::cout << "iter\_count = " << iter\_count << std::endl;

std::cout << "total time is " << end - begin << " seconds" << std::endl;

**return** 0;

}

1. **Приложение 2.** Полный листинг скрипта SLURM для параллельной программы.
2. #!/bin/bash
3. #SBATCH -J lab3 # Job name
4. #SBATCH -p compclass # Queue name (or "compclass\_unstable", or "gpuserv", or "a100serv")
5. #SBATCH -o lab3.%j.out # Name of stdout output file (%j expands to %jobId)
6. #SBATCH -N 1 # Total number of nodes requested
7. #SBATCH -n 1 # Total number of mpi tasks requested
8. #SBATCH -t 00:10:00 # Run time (hh:mm:ss)
9. module load mpi/intelmpi
10. **if** [ "$#" -eq 0 ]; **then**
11. echo "Usage: $0 <num\_threads>"
12. exit 1
13. **fi**
14. num\_threads=$1
15. export OMP\_NUM\_THREADS=$num\_threads
16. g++ -fopenmp slauOMP.cpp -o slauOMP
17. echo -e "\n1st run:\n"
18. ./slauOMP
19. echo -e "\n2nd run:\n"
20. ./slauOMP
21. echo -e "\n3rd run:\n"
22. ./slauOMP
23. **Приложение 3.** Полный листинг параллельной программы на C++ со schedule.
24. #**include** <iostream>
25. #**include** <cmath>
26. #**include** <omp.h>
27. const double EPSILON = 0.000001;
28. const int N = 4000;
29. const float TAU = 0.00001;
30. const int MAX\_ITERATION\_NUM = 10000;
31. void **createMatrix**(double\* matrix) {
32. **for** (int i = 0; i < N; i++) {
33. **for** (int j = i; j < N; j++) {
34. matrix[i \* N + j] = (double)rand() / RAND\_MAX \* 2.0 - 1.0;
35. **if** (i == j)
36. matrix[i \* N + j] = matrix[i \* N + j] + N;
37. }
38. **for** (int j = 0; j < i; j++)
39. matrix[i \* N + j] = matrix[j \* N + i];
40. }
41. }
42. void **createVector**(double\* vector) {
43. **for** (int i = 0; i < N; i++)
44. vector[i] = (double)rand() / RAND\_MAX \* 10.0 - 5.0;
45. }
46. double **countAccuracy**(const double\* vector) {
47. double accuracy = 0;
48. #**pragma** omp parallel for reduction(+ : accuracy) schedule(runtime)
49. **for** (int i = 0; i < N; i++)
50. accuracy += vector[i] \* vector[i];
51. **return** sqrt(accuracy);
52. }
53. int **main**(int argc, char\*\* argv) {
54. srand(time(nullptr));
55. double begin, end;
56. **auto**\* A = **new** double[N \* N];
57. **auto**\* x = **new** double[N];
58. **auto**\* x\_new = **new** double[N];
59. **auto**\* b = **new** double[N];
60. int iter\_count = 0;
61. double accuracy = 1;
62. begin = omp\_get\_wtime();
63. createMatrix(A);
64. createVector(x);
65. createVector(b);
66. double B\_accuracy = countAccuracy(b);
67. **while** (accuracy > EPSILON && iter\_count < MAX\_ITERATION\_NUM) {
68. #**pragma** omp parallel for schedule(runtime)
69. **for** (int i = 0; i < N; i++) {
70. x\_new[i] = 0;
71. **for** (int j = 0; j < N; j++)
72. x\_new[i] += A[i \* N + j] \* x[j];
73. }
74. #**pragma** omp parallel for schedule(runtime)
75. **for** (int i = 0; i < N; i++)
76. x\_new[i] = x\_new[i] - b[i];
77. accuracy = countAccuracy(x\_new) / B\_accuracy;
78. #**pragma** omp parallel for schedule(runtime)
79. **for** (int i = 0; i < N; i++) {
80. x\_new[i] = x[i] - TAU \* x\_new[i];
81. x[i] = x\_new[i];
82. }
83. iter\_count++;
84. }
85. **delete**[] A;
86. **delete**[] x;
87. **delete**[] x\_new;
88. **delete**[] b;
89. end = omp\_get\_wtime();
90. std::cout << "iter\_count = " << iter\_count << std::endl;
91. std::cout << "total time is " << end - begin << " seconds" << std::endl;
92. **return** 0;
93. }
94. **Приложение 4.** Полный листинг скрипта SLURM для программы со schedule.
95. #!/bin/bash
96. #SBATCH -J lab3 # Job name
97. #SBATCH -p compclass # Queue name (or "compclass\_unstable", or "gpuserv", or "a100serv")
98. #SBATCH -o lab3.schedule.%j.out # Name of stdout output file (%j expands to %jobId)
99. #SBATCH -e lab3.schedule.%j.err # Name of stderr output file (%j expands to %jobId)
100. #SBATCH -N 1 # Total number of nodes requested
101. #SBATCH -n 1 # Total number of mpi tasks requested
102. #SBATCH -t 00:30:00 # Run time (hh:mm:ss)
103. module load mpi/intelmpi
104. export OMP\_NUM\_THREADS=4
105. g++ -fopenmp schedule.cpp -o schedule
106. params=(1 2 3 4 5 6 7 8 9 10 20 50 80 100 200 500 1000 1500 2000)
107. **for** i **in** "${params[@]}"; **do**
108. echo -e "\nRunning with OMP\_SCHEDULE = static,$i"
109. export OMP\_SCHEDULE=static,$i
110. ./schedule
111. echo -e "\nRunning with OMP\_SCHEDULE = dynamic,$i"
112. export OMP\_SCHEDULE=dynamic,$i
113. ./schedule
114. echo -e "\nRunning with OMP\_SCHEDULE = guided,$i"
115. export OMP\_SCHEDULE=guided,$i
116. ./schedule
117. **done**