**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. Сравнить время выполнения решения системы линейных алгебраических уравнений вида методом простых итераций последовательной и параллельной программ.

**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:



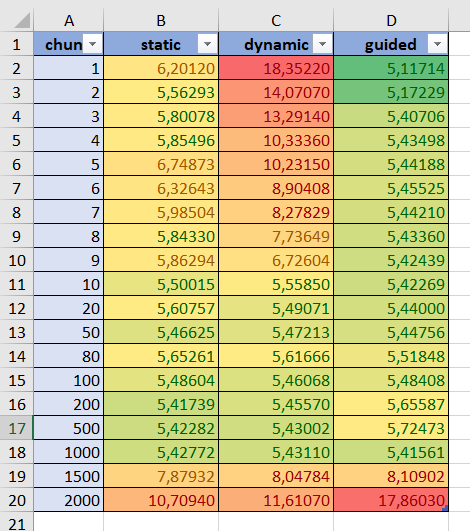
* На 8 потоках:



* На 16 потоках:



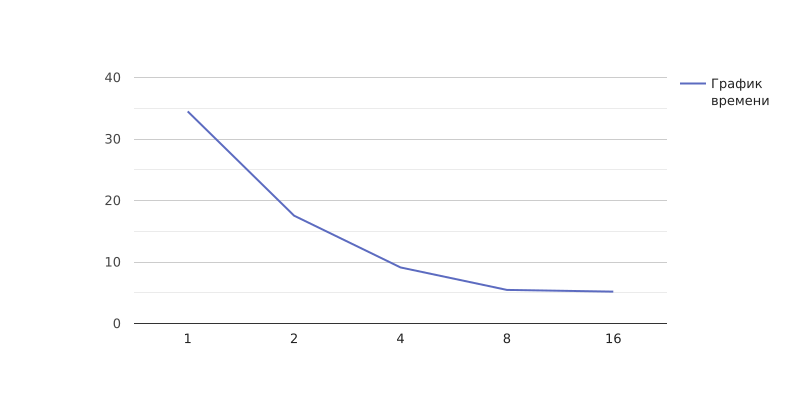
1. Также было проведено исследование на определение оптимальных параметров schedule:

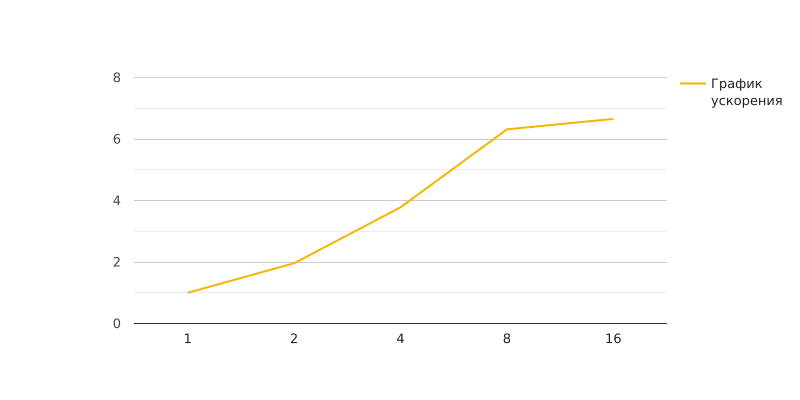


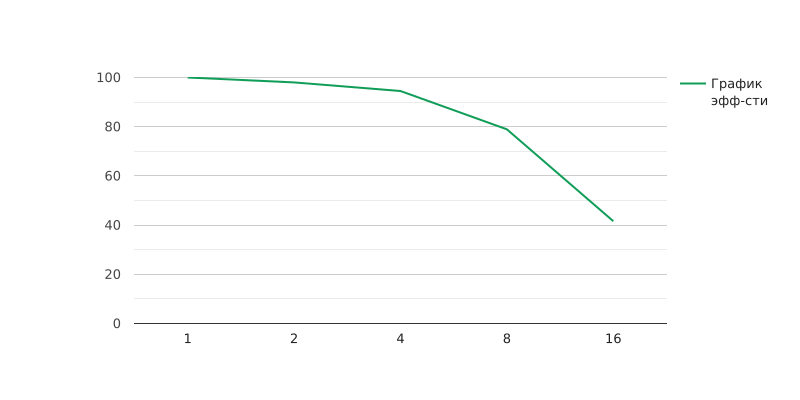
1. Было выполнено построение графиков времени, ускорения, эффективности и график параметров schedule, где:

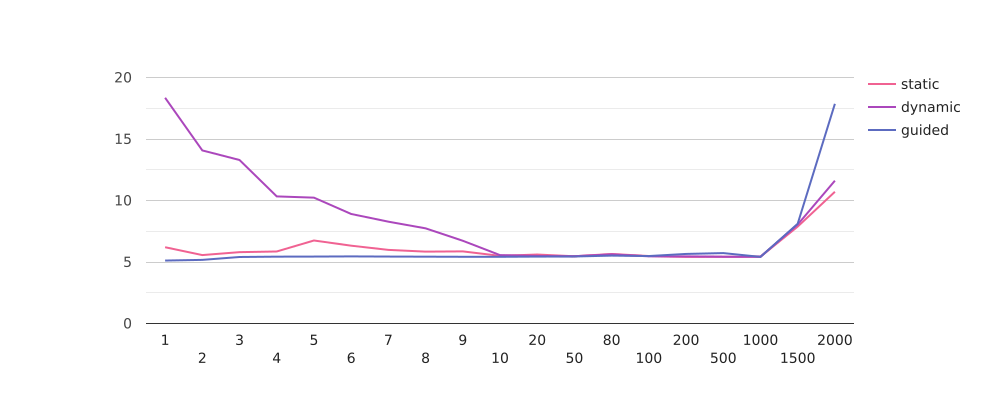
**Ускорение**: Sp = T1 / Tp, где T1 - время работы ПОСЛЕДОВАТЕЛЬНОЙ программы. Tp - время работы параллельной программы на p процессах/потоках.

**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++.
2. #**include** <iostream>
3. #**include** <cmath>
4. #**include** <omp.h>
5. const double EPSILON = 0.000001;
6. const int N = 7000;
7. const float TAU = 0.00001;
8. const int MAX\_ITERATION\_NUM = 10000;
9. void **createMatrix**(double\* matrix) {
10. **for** (int i = 0; i < N; i++) {
11. **for** (int j = i; j < N; j++) {
12. matrix[i \* N + j] = (double)rand() / RAND\_MAX \* 2.0 - 1.0;
13. **if** (i == j)
14. matrix[i \* N + j] = matrix[i \* N + j] + N;
15. }
16. **for** (int j = 0; j < i; j++)
17. matrix[i \* N + j] = matrix[j \* N + i];
18. }
19. }
20. void **createVector**(double\* vector) {
21. **for** (int i = 0; i < N; i++)
22. vector[i] = (double)rand() / RAND\_MAX \* 10.0 - 5.0;
23. }
24. void **setMatrixParts**(int\* linesPerThread, int\* offsets, int size){
25. int lines = N / size, rest = N % size, offset = 0;
26. **for** (int i = 0; i < size; ++i) {
27. rest ? (linesPerThread[i] = lines + 1, rest--) : linesPerThread[i] = lines;
28. offsets[i] = offset;
29. offset += linesPerThread[i];
30. }
31. }
32. double **countAccuracy**(const double\* vector) {
33. double accuracy = 0;
34. **for** (int i = 0; i < N; i++)
35. accuracy += vector[i] \* vector[i];
36. **return** sqrt(accuracy);
37. }
38. int **main**(int argc, char\*\* argv) {
39. srand(time(nullptr));
40. double begin, end;
41. //omp\_set\_num\_threads(1);
42. int size = omp\_get\_max\_threads();
43. **auto**\* A = **new** double[N \* N];
44. **auto**\* x = **new** double[N];
45. **auto**\* x\_new = **new** double[N];
46. **auto**\* b = **new** double[N];
47. **auto**\* linesPerThread = **new** int[size];
48. **auto**\* offsets = **new** int[size];
49. int iter\_count = 0;
50. double accuracy = 1;
51. setMatrixParts(linesPerThread, offsets, size);
52. begin = omp\_get\_wtime();
53. createMatrix(A);
54. createVector(x);
55. createVector(b);
56. double B\_accuracy = countAccuracy(b);
57. #**pragma** omp parallel shared(accuracy)
58. {
59. int thread\_id = omp\_get\_thread\_num();
60. int first\_index = offsets[thread\_id];
61. int last\_index = first\_index + linesPerThread[thread\_id];
62. **while** (accuracy > EPSILON && iter\_count < MAX\_ITERATION\_NUM) {
63. **for** (int i = first\_index; i < last\_index; i++) {
64. x\_new[i] = 0;
65. **for** (int j = 0; j < N; j++)
66. x\_new[i] += A[i \* N + j] \* x[j];
67. }
68. **for** (int i = first\_index; i < last\_index; i++)
69. x\_new[i] = x\_new[i] - b[i];
70. #**pragma** omp barrier
71. #**pragma** omp single
72. accuracy = countAccuracy(x\_new) / B\_accuracy;
73. /\*#pragma omp master
74. printf("accuracy = %.6lf\n", accuracy);\*/
75. **for** (int i = first\_index; i < last\_index; i++) {
76. x\_new[i] = x[i] - TAU \* x\_new[i];
77. x[i] = x\_new[i];
78. }
79. #**pragma** omp single
80. iter\_count++;
81. }
82. }
83. **delete**[] A;
84. **delete**[] x;
85. **delete**[] x\_new;
86. **delete**[] b;
87. **delete**[] linesPerThread;
88. **delete**[] offsets;
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 2.** Полный листинг скрипта SLURM для параллельной программы.
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.%j.out # Name of stdout output file (%j expands to %jobId)
99. #SBATCH -N 1 # Total number of nodes requested
100. #SBATCH -n 1 # Total number of mpi tasks requested
101. #SBATCH -t 00:10:00 # Run time (hh:mm:ss) - 1 minute
102. module load mpi/intelmpi
103. **if** [ "$#" -eq 0 ]; **then**
104. echo "Usage: $0 <num\_threads>"
105. exit 1
106. **fi**
107. num\_threads=$1
108. export OMP\_NUM\_THREADS=$num\_threads
109. g++ -fopenmp slauOMP.cpp -o slauOMP
110. echo -e "\n1st run:\n"
111. ./slauOMP
112. echo -e "\n2nd run:\n"
113. ./slauOMP
114. echo -e "\n3rd run:\n"
115. ./slauOMP
116. **Приложение 3.** Полный листинг параллельной программы на C++ со schedule.
117. #**include** <iostream>
118. #**include** <cmath>
119. #**include** <omp.h>
120. const double EPSILON = 0.000001;
121. const int N = 4000;
122. const float TAU = 0.00001;
123. const int MAX\_ITERATION\_NUM = 10000;
124. void **createMatrix**(double\* matrix) {
125. **for** (int i = 0; i < N; i++) {
126. **for** (int j = i; j < N; j++) {
127. matrix[i \* N + j] = (double)rand() / RAND\_MAX \* 2.0 - 1.0;
128. **if** (i == j)
129. matrix[i \* N + j] = matrix[i \* N + j] + N;
130. }
131. **for** (int j = 0; j < i; j++)
132. matrix[i \* N + j] = matrix[j \* N + i];
133. }
134. }
135. void **createVector**(double\* vector) {
136. **for** (int i = 0; i < N; i++)
137. vector[i] = (double)rand() / RAND\_MAX \* 10.0 - 5.0;
138. }
139. double **countAccuracy**(const double\* vector) {
140. double accuracy = 0;
141. #**pragma** omp parallel for reduction(+ : accuracy) schedule(runtime)
142. **for** (int i = 0; i < N; i++)
143. accuracy += vector[i] \* vector[i];
144. **return** sqrt(accuracy);
145. }
146. int **main**(int argc, char\*\* argv) {
147. srand(time(nullptr));
148. double begin, end;
149. **auto**\* A = **new** double[N \* N];
150. **auto**\* x = **new** double[N];
151. **auto**\* x\_new = **new** double[N];
152. **auto**\* b = **new** double[N];
153. int iter\_count = 0;
154. double accuracy = 1;
155. begin = omp\_get\_wtime();
156. createMatrix(A);
157. createVector(x);
158. createVector(b);
159. double B\_accuracy = countAccuracy(b);
160. **while** (accuracy > EPSILON && iter\_count < MAX\_ITERATION\_NUM) {
161. #**pragma** omp parallel for schedule(runtime)
162. **for** (int i = 0; i < N; i++) {
163. x\_new[i] = 0;
164. **for** (int j = 0; j < N; j++)
165. x\_new[i] += A[i \* N + j] \* x[j];
166. }
167. #**pragma** omp parallel for schedule(runtime)
168. **for** (int i = 0; i < N; i++)
169. x\_new[i] = x\_new[i] - b[i];
170. accuracy = countAccuracy(x\_new) / B\_accuracy;
171. #**pragma** omp parallel for schedule(runtime)
172. **for** (int i = 0; i < N; i++) {
173. x\_new[i] = x[i] - TAU \* x\_new[i];
174. x[i] = x\_new[i];
175. }
176. iter\_count++;
177. }
178. **delete**[] A;
179. **delete**[] x;
180. **delete**[] x\_new;
181. **delete**[] b;
182. end = omp\_get\_wtime();
183. std::cout << "iter\_count = " << iter\_count << std::endl;
184. std::cout << "total time is " << end - begin << " seconds" << std::endl;
185. **return** 0;
186. }
187. **Приложение 4.** Полный листинг скрипта SLURM для программы со schedule.
188. #!/bin/bash
189. #SBATCH -J lab3 # Job name
190. #SBATCH -p compclass # Queue name (or "compclass\_unstable", or "gpuserv", or "a100serv")
191. #SBATCH -o lab3.schedule.%j.out # Name of stdout output file (%j expands to %jobId)
192. #SBATCH -e lab3.schedule.%j.err # Name of stderr output file (%j expands to %jobId)
193. #SBATCH -N 1 # Total number of nodes requested
194. #SBATCH -n 1 # Total number of mpi tasks requested
195. #SBATCH -t 00:30:00 # Run time (hh:mm:ss)
196. module load mpi/intelmpi
197. export OMP\_NUM\_THREADS=4
198. g++ -fopenmp schedule.cpp -o schedule
199. params=(1 2 3 4 5 6 7 8 9 10 20 50 80 100 200 500 1000 1500 2000)
200. **for** i **in** "${params[@]}"; **do**
201. echo -e "\nRunning with OMP\_SCHEDULE = static,$i"
202. export OMP\_SCHEDULE=static,$i
203. ./schedule
204. echo -e "\nRunning with OMP\_SCHEDULE = dynamic,$i"
205. export OMP\_SCHEDULE=dynamic,$i
206. ./schedule
207. echo -e "\nRunning with OMP\_SCHEDULE = guided,$i"
208. export OMP\_SCHEDULE=guided,$i
209. ./schedule
210. **done**