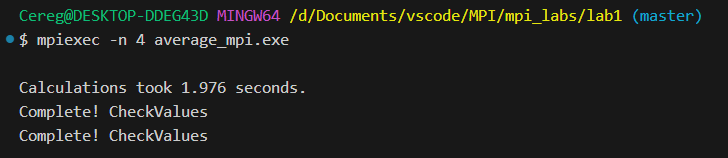
1. «Перемножение матриц»

В работе необходимо составить и проверить алгоритм перемножения матриц. Для проверки алгоритма целесообразно использовать небольшие матрицы (до 3×3 включительно), результат перемножения вывести на экран. После успешной проверки задаются большие (1000х1000) матрицы как входные данные, наполнение матриц случайно и необязательно к выводу на экран.

На скриншотах выполнения программы должна быть видна разница во времени исполнения в последовательном и в параллельном режимах.



1. /\*
2. \* Программа вычисляет средние значения элементов матрицы по строкам и столбцам
3. \* с использованием параллельных вычислений через MPI.
4. \* Проверяет корректность результатов, сравнивая с эталонными значениями из файлов.
5. \*/
6. #**include** <vector>
7. #**include** <thread>
8. #**include** <stdio.h>
9. #**include** <exception>
10. #**include** <locale.h>
11. #**include** <iostream>
12. #**include** <fstream>
13. #**include** <time.h>
14. #**include** <omp.h>
15. #**include** <mpi.h>
16. **using** **namespace** std;
17. // Тип обработки: по строкам или по столбцам
18. **enum class** **eprocess\_type**
19. {
20. by\_rows = 0,
21. by\_cols
22. };
23. // Инициализирует матрицу данными из файла matrix.txt
24. void **InitMatrix**(double \*\*matrix, const size\_t numb\_rows, const size\_t numb\_cols)
25. {
26. ifstream myfile;
27. myfile.open("matrix.txt");
28. **if** (myfile.is\_open())
29. {
30. **for** (size\_t i = 0; i < numb\_rows; ++i)
31. {
32. **for** (size\_t j = 0; j < numb\_cols; ++j)
33. {
34. myfile >> matrix[i][j];
35. }
36. }
37. }
38. myfile.close();
39. }
40. // Выводит матрицу в консоль (для отладки)
41. void **PrintMatrix**(double \*\*matrix, const size\_t numb\_rows, const size\_t numb\_cols)
42. {
43. printf("Generated matrix:\n");
44. **for** (size\_t i = 0; i < numb\_rows; ++i)
45. {
46. **for** (size\_t j = 0; j < numb\_cols; ++j)
47. {
48. printf("%lf ", matrix[i][j]);
49. }
50. printf("\n");
51. }
52. }
53. /\*\*
54. \* Вычисляет средние значения по строкам или столбцам матрицы с использованием MPI.
55. \*
56. \* @param proc\_type Способ обработки (строки/столбцы)
57. \* @param matrix Исходная матрица
58. \* @param numb\_rows Количество строк
59. \* @param numb\_cols Количество столбцов
60. \* @param average\_vals Массив для сохранения вычисленных средних значений
61. \* @param true\_vals Массив с эталонными значениями для проверки
62. \*/
63. void **FindAverageValues**(eprocess\_type proc\_type, double \*\*matrix, const size\_t numb\_rows,
64. const size\_t numb\_cols, double \*average\_vals, double \*true\_vals)
65. {
66. int rank, size;
67. MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);
68. MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);
69. **if** (proc\_type == eprocess\_type::by\_rows)
70. {
71. // Распределение строк между процессами
72. int chunk = numb\_rows / size;
73. int start = rank \* chunk;
74. int end = (rank == size - 1) ? numb\_rows : start + chunk;
75. double \*local\_average = **new** double[chunk]();
76. // Локальное вычисление средних значений для своего блока строк
77. **for** (size\_t i = start; i < end; ++i)
78. {
79. double sum = 0.0;
80. **for** (size\_t j = 0; j < numb\_cols; ++j)
81. {
82. sum += matrix[i][j];
83. }
84. local\_average[i - start] = sum / numb\_cols;
85. }
86. // Сбор результатов на процессе 0
87. MPI\_Gather(
88. local\_average, chunk, MPI\_DOUBLE, // Отправляемые данные
89. average\_vals, chunk, MPI\_DOUBLE, // Буфер для приема
90. 0, MPI\_COMM\_WORLD);
91. **delete**[] local\_average;
92. // Загрузка эталонных значений для проверки (только на процессе 0)
93. **if** (rank == 0)
94. {
95. ifstream **result1**("result\_rows.txt");
96. **for** (size\_t i = 0; i < numb\_rows; ++i)
97. {
98. result1 >> true\_vals[i];
99. }
100. }
101. }
102. **else** **if** (proc\_type == eprocess\_type::by\_cols)
103. {
104. // Распределение столбцов между процессами
105. int chunk = numb\_cols / size;
106. int start = rank \* chunk;
107. int end = (rank == size - 1) ? numb\_cols : start + chunk;
108. double \*local\_average = **new** double[chunk]();
109. // Локальное вычисление средних значений для своего блока столбцов
110. **for** (size\_t j = start; j < end; ++j)
111. {
112. double sum = 0.0;
113. **for** (size\_t i = 0; i < numb\_rows; ++i)
114. {
115. sum += matrix[i][j];
116. }
117. local\_average[j - start] = sum / numb\_rows;
118. }
119. // Сбор результатов на процессе 0
120. MPI\_Gather(
121. local\_average, chunk, MPI\_DOUBLE,
122. average\_vals, chunk, MPI\_DOUBLE,
123. 0, MPI\_COMM\_WORLD);
124. **delete**[] local\_average;
125. // Загрузка эталонных значений для проверки
126. **if** (rank == 0)
127. {
128. ifstream **result2**("result\_cols.txt");
129. **for** (size\_t j = 0; j < numb\_cols; ++j)
130. {
131. result2 >> true\_vals[j];
132. }
133. }
134. }
135. **else**
136. {
137. throw("Incorrect process type!");
138. }
139. }
140. // Сравнивает вычисленные значения с эталонными
141. void **CheckValues**(double \*average\_vals, double \*true\_vals, const size\_t counter)
142. {
143. **for** (size\_t i = 0; i < counter; ++i)
144. {
145. **if** (std::abs(average\_vals[i] - true\_vals[i]) > 1e-9)
146. {
147. printf("Error! CheckValues\n");
148. **return**;
149. }
150. }
151. printf("Complete! CheckValues\n");
152. }
153. int **main**()
154. {
155. MPI\_Init(NULL, NULL); // Инициализация MPI
156. int rank;
157. MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);
158. srand((unsigned)time(0));
159. clock\_t start, stop;
160. start = clock();
161. // Параметры матрицы
162. const size\_t numb\_rows = 1000;
163. const size\_t numb\_cols = 1000;
164. // Выделение памяти под матрицу
165. double \*\*matrix = **new** double \*[numb\_rows];
166. **for** (size\_t i = 0; i < numb\_rows; ++i)
167. {
168. matrix[i] = **new** double[numb\_cols];
169. }
170. // Процесс 0 загружает матрицу и рассылает её всем процессам
171. **if** (rank == 0)
172. {
173. InitMatrix(matrix, numb\_rows, numb\_cols);
174. }
175. **for** (size\_t i = 0; i < numb\_rows; ++i)
176. {
177. MPI\_Bcast(matrix[i], numb\_cols, MPI\_DOUBLE, 0, MPI\_COMM\_WORLD);
178. }
179. // Выделение памяти для результатов
180. double \*average\_vals\_in\_rows = **new** double[numb\_rows]();
181. double \*average\_vals\_in\_cols = **new** double[numb\_cols]();
182. double \*true\_vals\_in\_rows = **new** double[numb\_rows];
183. double \*true\_vals\_in\_cols = **new** double[numb\_cols];
184. // Вычисление средних значений
185. FindAverageValues(eprocess\_type::by\_rows, matrix, numb\_rows, numb\_cols, average\_vals\_in\_rows, true\_vals\_in\_rows);
186. FindAverageValues(eprocess\_type::by\_cols, matrix, numb\_rows, numb\_cols, average\_vals\_in\_cols, true\_vals\_in\_cols);
187. // Проверка результатов и вывод времени работы (только на процессе 0)
188. **if** (rank == 0)
189. {
190. CheckValues(average\_vals\_in\_rows, true\_vals\_in\_rows, numb\_rows);
191. CheckValues(average\_vals\_in\_cols, true\_vals\_in\_cols, numb\_cols);
192. stop = clock();
193. cout << endl
194. << "Calculations took " << ((double)(stop - start)) / CLOCKS\_PER\_SEC << " seconds.\n";
195. }
196. // Освобождение ресурсов
197. **for** (size\_t i = 0; i < numb\_rows; ++i)
198. {
199. **delete**[] matrix[i];
200. }
201. **delete**[] matrix;
202. **delete**[] average\_vals\_in\_rows;
203. **delete**[] average\_vals\_in\_cols;
204. **delete**[] true\_vals\_in\_rows;
205. **delete**[] true\_vals\_in\_cols;
206. MPI\_Finalize(); // Завершение работы с MPI
207. **return** 0;
208. }