**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**

«Умножение матрицы на матрицу в MPI 2D решетке»

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 10

ПРИЛОЖЕНИЕ.

Полный листинг параллельной программы на C++ 11

Полный листинг скрипта SLURM для параллельной программы 14

**ЦЕЛЬ**

1. Освоить концепции MPI-коммуникаторов и декартовых топологий, а также концепции производных типов данных.

**ЗАДАНИЕ**

1. Реализовать параллельный алгоритм умножения матрицы на матрицу  
   при 2D решетке процессов с соблюдением требований.
2. Исследовать производительность параллельной программы при  
   фиксированном размере матрицы в зависимости от и размера решетки:  
   2x12, 3x8, 4x6, 6x4, 8x3, 12x2. Размер матриц подобрать таким образом,  
   чтобы худшее из времен данного набора было не менее 30 сек.
3. Выполнить профилирование программы при использовании 8-и ядер  
   с решетками 2x4, 4x2.

Общий алгоритм:

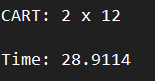
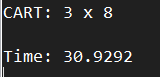
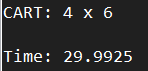
1. Создание решетки процессов p1 x p2.
2. Генерация матриц A[n1 x n2] и B[n2 x n3] на процессе с координатами (0;0) как одномерных массивов.
3. Раздача матрицы A по горизонтальным полосам на вертикальную линейку процессов (0;0), (1;0), (2;0), ..., (p1 - 1; 0) при помощи MPI\_Scatter.
4. Определение нового производного типа данных для выбора из матрицы B вертикальных полос.
5. Раздача матрицы B по вертикальным полосам на горизонтальную линейку процессов (0;0), (0;1), (0;2), ..., (0; p2 – 1) таким образом, что каждому процессу высылается только 1 элемент производного типа.
6. Каждый из процессов в левой вертикальной колонке ( (1;0), (2;0), ..., (p1 - 1; 0) ) при помощи MPI\_Bcast раздает свою полосу матрицы A всем процессам своей горизонтали. Т.е. процесс (1;0) раздает свою полосу процессам (1;1), (1;2),...
7. То же с полосами матрицы B, которые процессы первой горизонтали раздают по своим вертикальным столбцам решетки процессов (MPI\_Bcast).
8. Теперь на каждом процессе есть по полосе А и по столбцу B, перемножаем, получаем миноры С.
9. Собираем всю С на процессе (0;0).

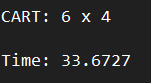
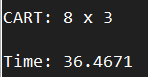
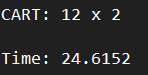
**ОПИСАНИЕ РАБОТЫ**

1. Была написана программа с использованием технологии MPI. Время выполнения было замерено с помощью функции MPI\_Wtime();
2. Был написан скрипт с использованием SLURM, с помощью которого программа компилировалась и запускалась:

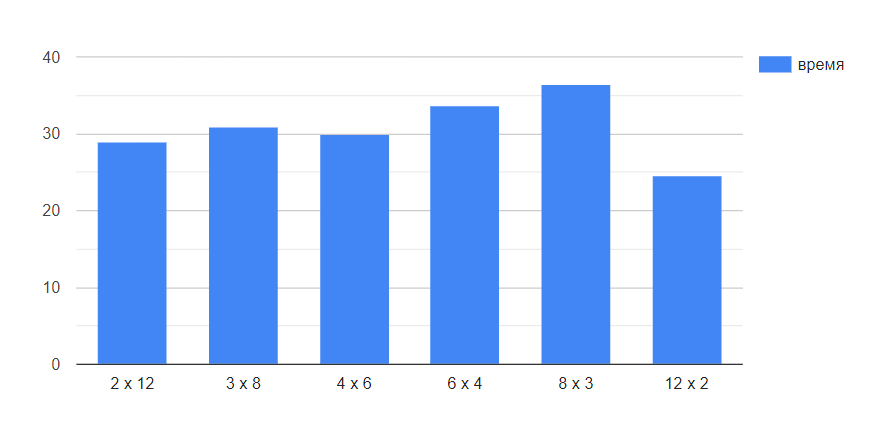


1. Время работы программы с разными входными данными получилось следующим:

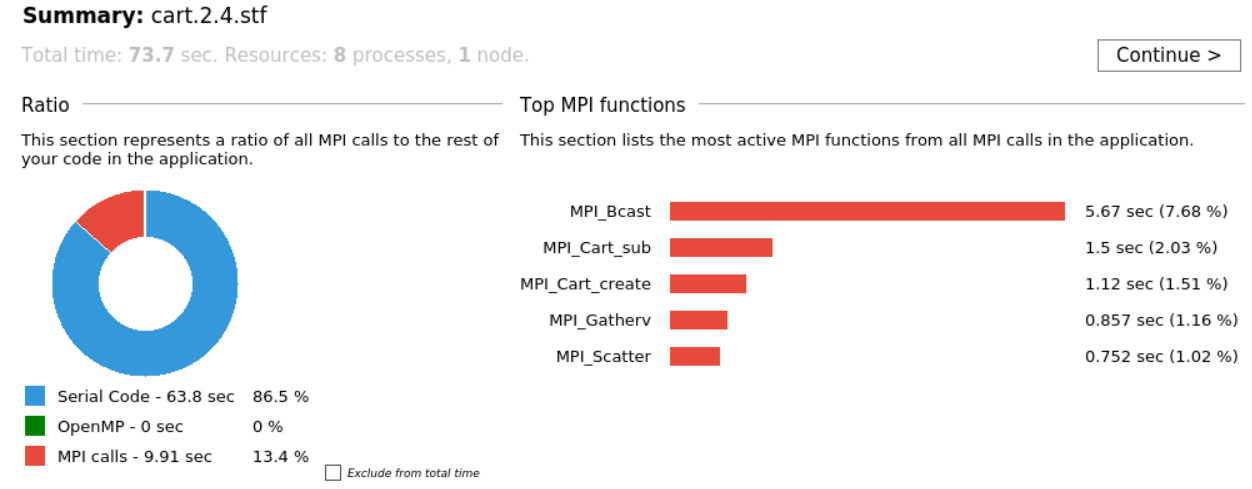
  

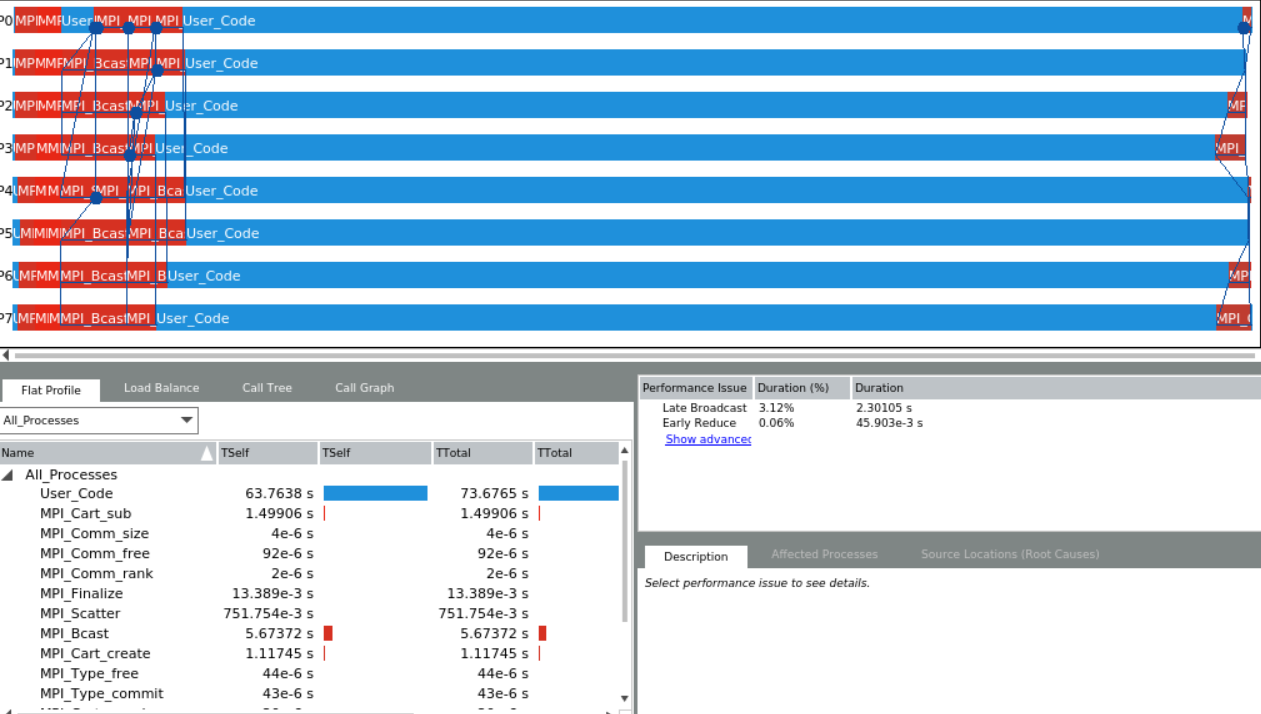
1. Сравнение времени работы программы на разных входных данных:

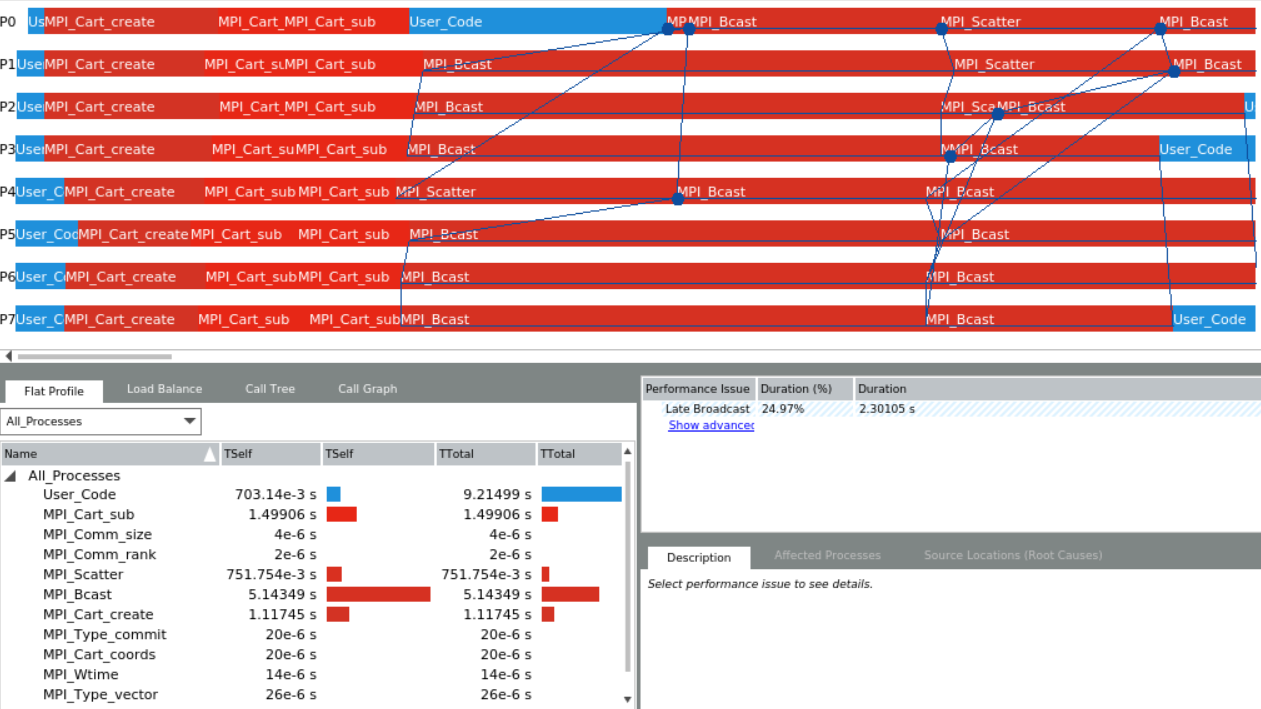


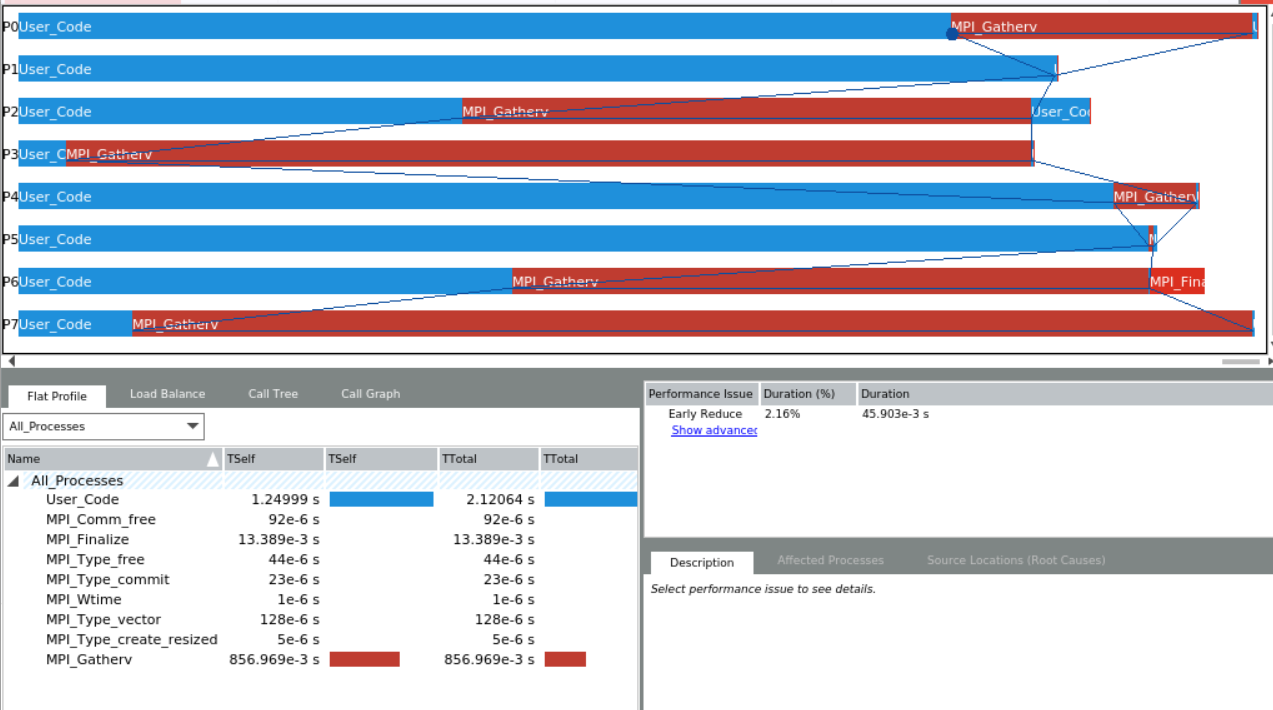
1. Было выполнено профилирование на 8-ми процессах:

Для решётки 2 х 4:

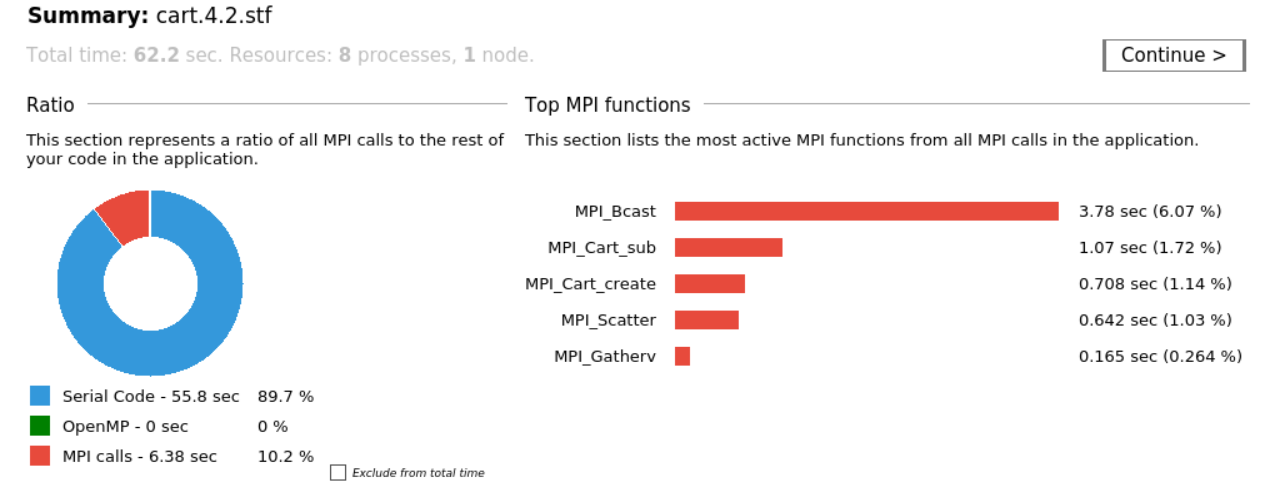


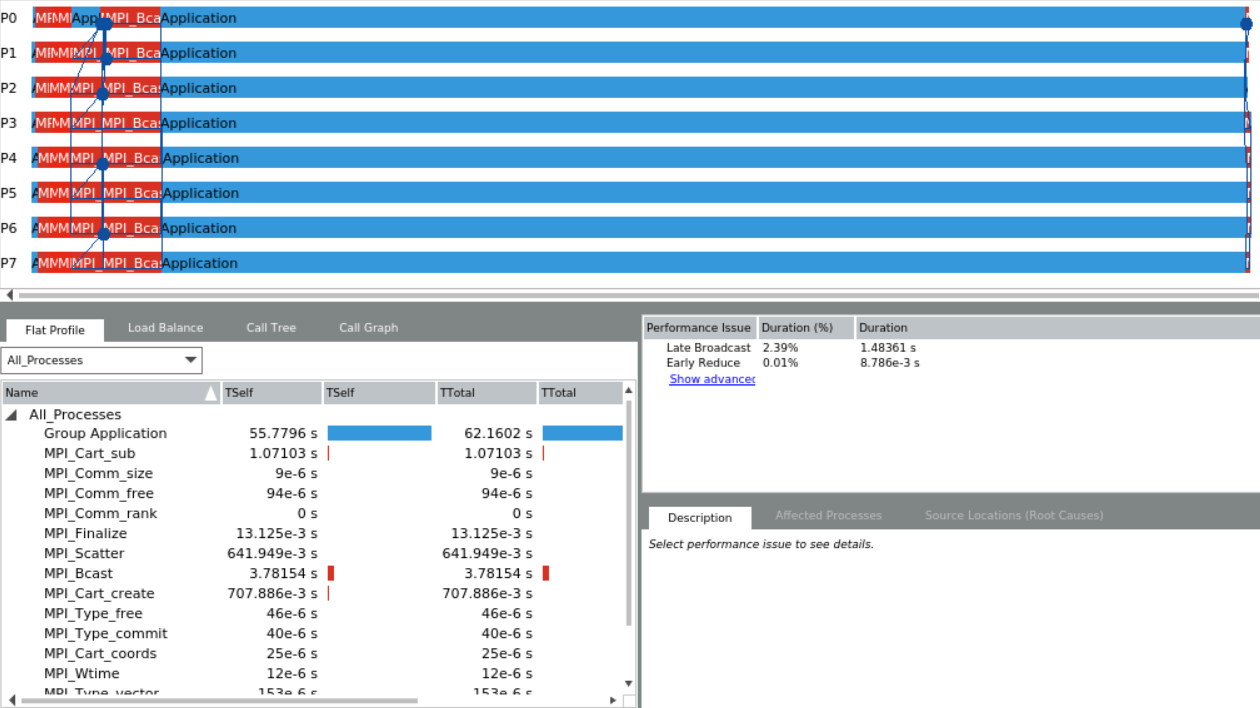






Для решётки 4 х 2:









**ЗАКЛЮЧЕНИЕ**

Были освоены концепции MPI коммуникаторов и декартовых топологий, а

также концепции производных типов данных.

1. **Приложение 1.** Полный листинг параллельной программы на C++.
2. #**include** <iostream>
3. #**include** <mpi.h>
4. int N1;
5. int N2;
6. int N3;
7. const int X = 0;
8. const int Y = 1;
9. const int NDIMS = 2;
10. const int n1\_mult = 500;
11. const int n3\_mult = 750;
12. void **initNs**(const int\* dims) {
13. N1 = dims[X] \* n1\_mult;
14. N3 = dims[Y] \* n3\_mult;
15. N2 = (N3 + N1) / 2;
16. }
17. void **generateMatrix**(double\* matrix, int row, int column) {
18. **for**(int i = 0; i < row; i++)
19. **for**(int j = 0; j < column; j++)
20. matrix[i \* column + j] = (double)rand() / RAND\_MAX \* 20.0 - 10.0;
21. }
22. void **createCartComm**(MPI\_Comm& cart, int size, int argc, char\*\* argv, int\* dims) {
23. **if** (argc <= 2)
24. MPI\_Dims\_create(size, NDIMS, dims);
25. **else** {
26. dims[X] = strtol(argv[1], nullptr, 10);
27. dims[Y] = strtol(argv[2], nullptr, 10);
28. **if** (dims[X] \* dims[Y] != size) exit(EXIT\_FAILURE);
29. }
30. bool reorder = true;
31. int periodic[NDIMS] = {};
32. MPI\_Cart\_create(MPI\_COMM\_WORLD, NDIMS, dims, periodic, reorder, &cart);
33. }
34. void **createSubComms**(MPI\_Comm& cart, MPI\_Comm& rows, MPI\_Comm& columns) {
35. int remain\_dims[NDIMS];
36. remain\_dims[X] = false, remain\_dims[Y] = true;
37. MPI\_Cart\_sub(cart, remain\_dims, &rows);
38. remain\_dims[X] = true, remain\_dims[Y] = false;
39. MPI\_Cart\_sub(cart, remain\_dims, &columns);
40. }
41. void **mult**(double\* C\_part, const double\* A\_part, const double\* B\_part, int A\_rows, int B\_cols) {
42. **for** (int i = 0; i < A\_rows; i++)
43. **for** (int j = 0; j < N2; j++)
44. **for** (int k = 0; k < B\_cols; k++)
45. C\_part[i \* B\_cols + k] += A\_part[i \* N2 + j] \* B\_part[j \* B\_cols + k];
46. }
47. void **gatherC**(double\* C, double\* C\_part, MPI\_Comm& cart, int size, int dim\_x, int dim\_y) {
48. MPI\_Datatype C\_block, C\_blocktype;
49. int\* recvcounts = **new** int [size];
50. int\* displs = **new** int [size];
51. **for** (int i = 0; i < dim\_x; i++)
52. **for** (int j = 0; j < dim\_y; j++) {
53. recvcounts[i \* dim\_y + j] = 1;
54. displs[i \* dim\_y + j] = i \* n1\_mult \* dim\_y + j;
55. }
56. MPI\_Type\_vector(n1\_mult, n3\_mult, N3, MPI\_DOUBLE, &C\_block);
57. MPI\_Type\_commit(&C\_block);
58. MPI\_Type\_create\_resized(C\_block, 0, n3\_mult \* sizeof(double), &C\_blocktype);
59. MPI\_Type\_commit(&C\_blocktype);
60. MPI\_Gatherv(C\_part, n1\_mult \* n3\_mult, MPI\_DOUBLE, C, recvcounts, displs, C\_blocktype, 0, MPI\_COMM\_WORLD);
61. MPI\_Type\_free(&C\_block);
62. MPI\_Type\_free(&C\_blocktype);
63. **delete**[] displs;
64. **delete**[] recvcounts;
65. }
66. int **main**(int argc, char\*\* argv) {
67. int size, rank;
68. double start, end;
69. double\* A;
70. double\* B;
71. double\* C;
72. double\* A\_part;
73. double\* B\_part;
74. double\* C\_part;
75. MPI\_Comm cart;
76. MPI\_Comm rows;
77. MPI\_Comm columns;
78. MPI\_Init(&argc, &argv);
79. MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);
80. MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);
81. int dims[NDIMS] = {};
82. int coords[NDIMS] = {};
83. createCartComm(cart, size, argc, argv, dims);
84. initNs(dims);
85. createSubComms(cart, rows, columns);
86. MPI\_Cart\_coords(cart, rank, NDIMS, coords);
87. **if** (!coords[X] && !coords[Y]) {
88. A = **new** double [N1 \* N2];
89. B = **new** double [N2 \* N3];
90. C = **new** double [N1 \* N3]{};
91. generateMatrix(A, N1, N2);
92. generateMatrix(B, N2, N3);
93. }
94. start = MPI\_Wtime();
95. int A\_part\_size = n1\_mult \* N2;
96. int B\_part\_size = N2 \* n3\_mult;
97. A\_part = **new** double [A\_part\_size];
98. B\_part = **new** double [B\_part\_size];
99. C\_part = **new** double [n1\_mult \* n3\_mult];
100. **if** (!coords[Y])
101. MPI\_Scatter(A, A\_part\_size, MPI\_DOUBLE, A\_part, A\_part\_size, MPI\_DOUBLE, 0, columns);
102. MPI\_Bcast(A\_part, A\_part\_size, MPI\_DOUBLE, 0, rows);
103. MPI\_Datatype B\_block, B\_blocktype;
104. MPI\_Type\_vector(N2, n3\_mult, N3, MPI\_DOUBLE, &B\_block);
105. MPI\_Type\_commit(&B\_block);
106. MPI\_Type\_create\_resized(B\_block, 0, n3\_mult \* sizeof(double), &B\_blocktype);
107. MPI\_Type\_commit(&B\_blocktype);
108. **if** (!coords[X])
109. MPI\_Scatter(B, 1, B\_blocktype, B\_part, n3\_mult \* N2, MPI\_DOUBLE, 0, rows);
110. MPI\_Bcast(B\_part, B\_part\_size, MPI\_DOUBLE, 0, columns);
111. mult(C\_part, A\_part, B\_part, n1\_mult, n3\_mult);
112. gatherC(C, C\_part, cart, size, dims[X], dims[Y]);
113. **if** (!coords[X] && !coords[Y]) {
114. end = MPI\_Wtime();
115. std::cout << "Time: " << end - start << std::endl;
116. **delete**[] A;
117. **delete**[] B;
118. **delete**[] C;
119. }
120. **delete**[] A\_part;
121. **delete**[] B\_part;
122. **delete**[] C\_part;
123. MPI\_Comm\_free(&cart);
124. MPI\_Comm\_free(&rows);
125. MPI\_Comm\_free(&columns);
126. MPI\_Type\_free(&B\_block);
127. MPI\_Type\_free(&B\_blocktype);
128. MPI\_Finalize();
129. **return** 0;
130. }
131. **Приложение 2.** Полный листинг скрипта SLURM для параллельной программы.
132. #!/bin/bash
133. #SBATCH -J lab4 # Job name
134. #SBATCH -p compclass # Queue name (or "compclass\_unstable", or "gpuserv", or "a100serv")
135. #SBATCH -o lab4.%j.out # Name of stdout output file (%j expands to %jobId)
136. #SBATCH -N 1 # Total number of nodes requested
137. #SBATCH -n 2 # Total number of mpi tasks requested
138. #SBATCH -t 00:10:00 # Run time (hh:mm:ss) - 1 minute
139. module load mpi/mpich-x86\_64
140. **if** [ "$#" -ne 2 ]; **then**
141. echo "Usage: $0 <dimension X size> <dimension Y size>"
142. exit 1
143. **fi**
144. dimX=$1
145. dimY=$2
146. SIZE=$((dimX \* dimY))
147. mpicxx cart.cpp -o cart
148. echo -e "CART: $dimX x $dimY\n"
149. mpirun -np $SIZE ./cart $dimX $dimY