**Лабораторная работа №3**

**«Умножение матрицы на матрицу в MPI 2D решетка»**

**Задание к лабораторной работе №3:**

1. Реализовать параллельный алгоритм умножения матрицы на матрицу при 2D решетке.

2. Исследовать производительность параллельной программы в зависимости от размера матрицы и размера решетки.

3. Выполнить профилирование программы с помощью MPE при использовании 16-и ядер.

**Выполнение:**

Для измерений были выбраны матрицы размером 2000x2000.

Зависимость времени работы программы от количества узлов:

|  |  |
| --- | --- |
| 1 x 1 | 46,892 |
| 2 x 1 | 24,35 |
| 2 x 2 | 21,581 |
| 2 x 4 | 18,557 |
| 4 x 4 | 11,361 |

Зависимость времени работы программы от расположения узлов:

Были выбраны различные варианты построения сетки на 16 узлах.

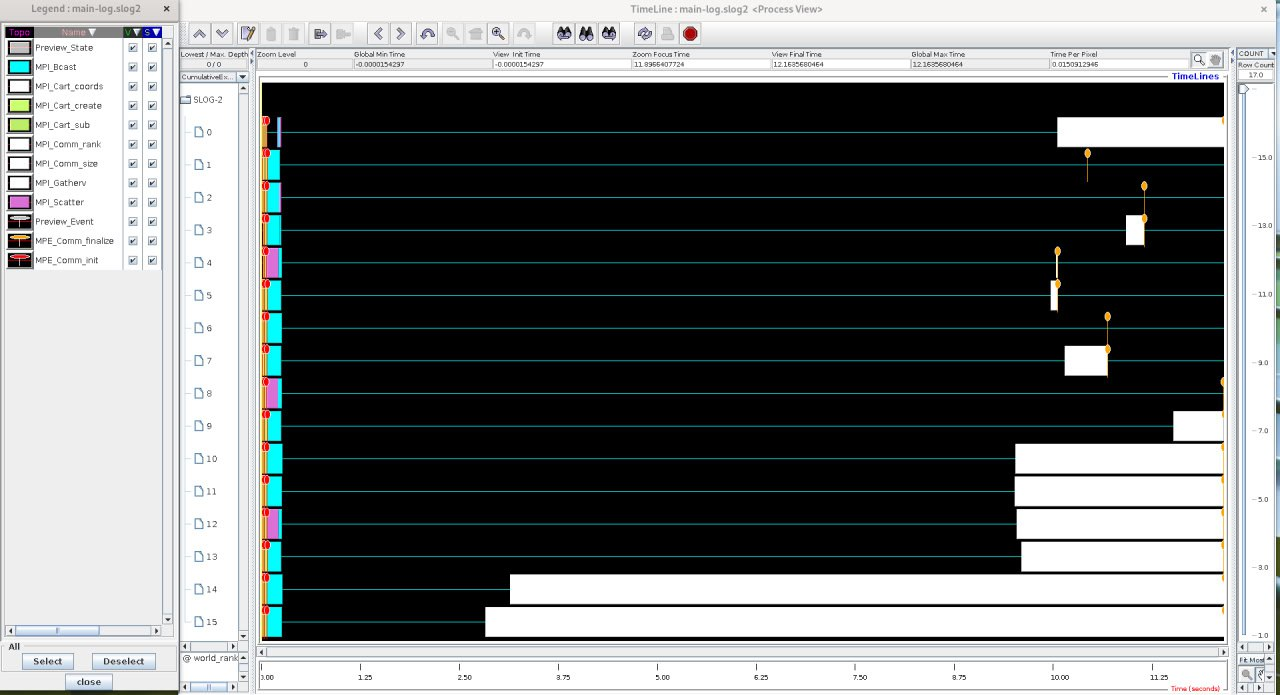
|  |  |
| --- | --- |
| 4 x 4 | 11,361 |
| 2 x 8 | 11,668 |
| 1 x 16 | 11,408 |

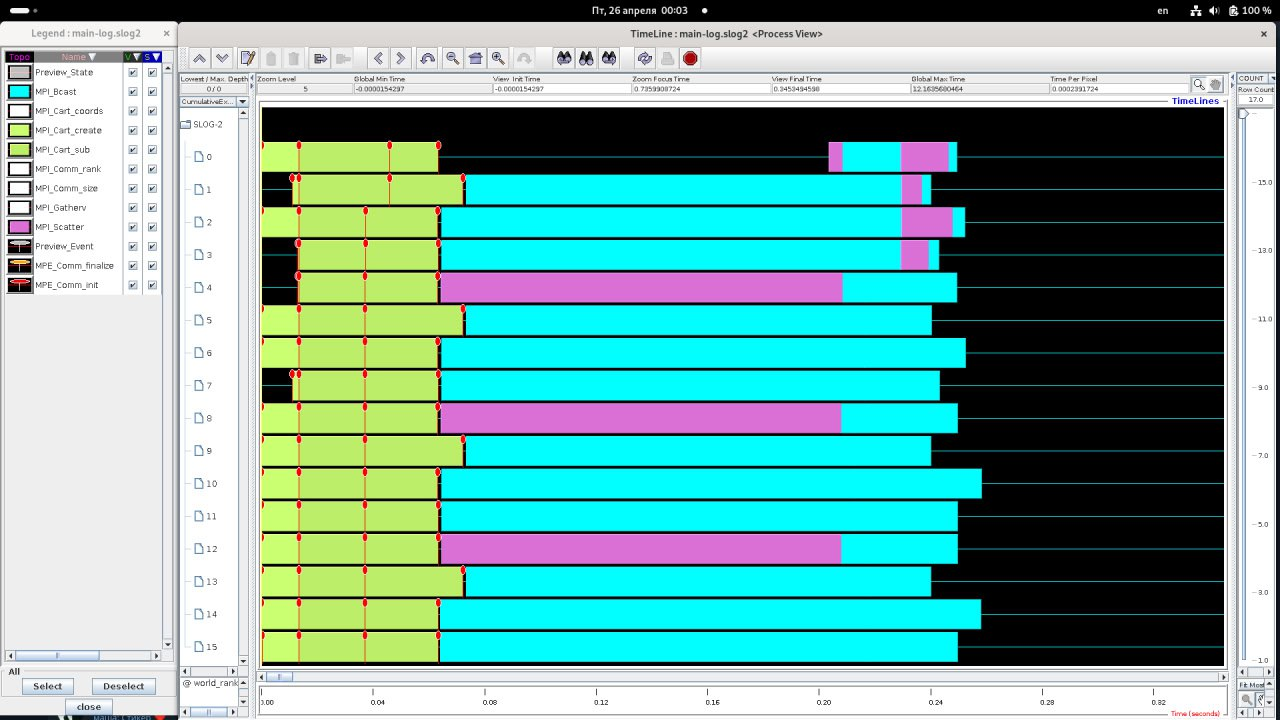
Анализ полученных данных:

Скачок во времени работы при сетке на 16 узлах, скорее всего, обусловлен загруженностью кластера во время предыдущих замеров.

Также из второй таблицы видно, что время работы программы не зависит от того, какую форму имеет сетка (квадрат, прямоугольник, линия) в пределах погрешности.

Приложение 1. Профилирование программы





Приложение 2. Исходный код

1. #**include** <iostream>
2. #**include** <mpi.h>
3. #**include** <cmath>
4. #**include** <zconf.h>
5. #**include** <cstdio>
6. #**include** "MatrixOperations.h"
7. MPI\_Comm GridComm;
8. MPI\_Comm ColComm;
9. MPI\_Comm RowComm;
10. int GridCoords[2];
11. int ProcNum = 0;
12. int ProcRank = 0;
13. int p1 = 4;
14. int p2 = 4;
15. int n1 = 2000;
16. int n2 = 2000;
17. int n3 = 2000;
18. void **CreateGridCommunicators**(){
19. int DimSize[2];
20. int Periodic[2];
21. int SubDimension[2];
22. DimSize[0] = p1;
23. DimSize[1] = p2;
24. Periodic[0] = 0;
25. Periodic[1] = 0;
26. MPI\_Cart\_create(MPI\_COMM\_WORLD, 2, DimSize, Periodic, 0, &GridComm);
27. MPI\_Cart\_coords(GridComm, ProcRank, 2, GridCoords);
28. SubDimension[0] = 0;
29. SubDimension[1] = 1;
30. MPI\_Cart\_sub(GridComm, SubDimension, &RowComm);
31. SubDimension[0] = 1;
32. SubDimension[1] = 0;
33. MPI\_Cart\_sub(GridComm, SubDimension, &ColComm);
34. }
35. void **InitializeProcess**(double\* &pAMatrix, double\* &pBMatrix, double\* &pCMatrix,
36. double\* &pAblock, double\* &pBblock, double\* &pCblock, int &ABlockSize, int &BBlockSize ) {
37. ABlockSize = n1/p1;
38. BBlockSize = n3/p2;
39. pAblock = **new** double [n2\*ABlockSize];
40. pBblock = **new** double [n2\*BBlockSize];
41. pCblock = **new** double [ABlockSize\*BBlockSize];
42. **if** (ProcRank == 0){
43. pAMatrix = **new** double [n1\*n2];
44. pBMatrix = **new** double [n2\*n3];
45. pCMatrix = **new** double [n1\*n3];
46. DataInitialization(pAMatrix, n1, n2);
47. DataInitialization(pBMatrix, n2, n3);
48. SetToZero(pCMatrix, n1, n3);
49. }
50. SetToZero(pCblock, ABlockSize, BBlockSize);
51. }
52. void **TerminateProcess** (double\* AMatrix, double\* BMatrix,
53. double\* CMatrix, double\* Ablock, double\* Bblock, double\* Cblock){
54. **if** (ProcRank == 0){
55. **delete** [] AMatrix;
56. **delete** [] BMatrix;
57. **delete** [] CMatrix;
58. }
59. **delete** [] Ablock;
60. **delete** [] Bblock;
61. **delete** [] Cblock;
62. }
63. void **DataDistribution**(double\* AMatrix, double\* BMatrix, double\* Ablock,
64. double\* Bblock, int ABlockSize, int BBlockSize) {
65. **if** (GridCoords[1] == 0) {
66. MPI\_Scatter(AMatrix, ABlockSize \* n2, MPI\_DOUBLE, Ablock,
67. ABlockSize \* n2, MPI\_DOUBLE, 0, ColComm);
68. }
69. MPI\_Bcast(Ablock, ABlockSize \* n2, MPI\_DOUBLE, 0, RowComm);
70. MPI\_Datatype col, coltype;
71. **if** (ProcRank == 0) {
72. MPI\_Type\_vector(n2, BBlockSize, n3, MPI\_DOUBLE, &col);
73. MPI\_Type\_commit(&col);
74. MPI\_Type\_create\_resized(col, 0, BBlockSize \* sizeof(double), &coltype);
75. MPI\_Type\_commit(&coltype);
76. }

79. **if** (GridCoords[0] == 0) {
80. MPI\_Scatter(BMatrix, 1, coltype, Bblock, n2 \* BBlockSize,
81. MPI\_DOUBLE, 0, RowComm);
82. }
83. MPI\_Bcast(Bblock, BBlockSize \* n2, MPI\_DOUBLE, 0, ColComm);
84. }
85. int **main**(int argc, char\* argv[]) {
86. double\* AMatrix = NULL;
87. double\* BMatrix = NULL;
88. double\* CMatrix = NULL;
89. int ABlockSize = 0;
90. int BBlockSize = 0;
91. double \*Ablock = NULL;
92. double \*Bblock = NULL;
93. double \*Cblock = NULL;
94. MPI\_Init(&argc, &argv);
95. MPI\_Comm\_size(MPI\_COMM\_WORLD, &ProcNum);
96. MPI\_Comm\_rank(MPI\_COMM\_WORLD, &ProcRank);
97. **if** ((n1 % p1 != 0) || (n3 % p2 != 0)) {
98. **if** (ProcRank == 0) {
99. printf ("invalid grid size\n");
100. }
101. } **else** {
102. **if** (ProcRank == 0)
103. printf("Parallel matrix multiplication program, on %d processes\n", ProcNum);
104. CreateGridCommunicators();
105. }
106. InitializeProcess(AMatrix, BMatrix, CMatrix, Ablock, Bblock,
107. Cblock, ABlockSize, BBlockSize);
108. double startTime;
109. **if** (ProcRank == 0) {
110. startTime = MPI\_Wtime();
111. }
112. DataDistribution(AMatrix, BMatrix, Ablock, Bblock, ABlockSize, BBlockSize);
113. MatrixMul(Ablock, Bblock, Cblock, ABlockSize, n2, BBlockSize);
114. MPI\_Datatype block, blocktype;
115. MPI\_Type\_vector(ABlockSize, BBlockSize, n3, MPI\_DOUBLE, &block);
116. MPI\_Type\_commit(&block);
117. MPI\_Type\_create\_resized(block, 0, BBlockSize\*sizeof(double), &blocktype);
118. MPI\_Type\_commit(&blocktype);
119. int\* displ = **new** int[p1\*p2];
120. int\* rcount = **new** int[p1\*p2];
121. int BlockCount = 0;
122. int BlockSize = ABlockSize\*BBlockSize;
123. int NumCount = 0;
124. int Written;
125. int j = 0;
126. **while** (NumCount < p1\*p2\*BlockSize) {
127. Written = 0;
128. **for** (int i = 0; i < n3; i += BBlockSize) {
129. displ[j] = BlockCount;
130. rcount[j] = 1;
131. j++;
132. BlockCount++;
133. Written++;
134. }
135. NumCount += Written \* BlockSize;
136. BlockCount += Written \* (ABlockSize - 1);
137. }
138. MPI\_Gatherv(Cblock, BlockSize, MPI\_DOUBLE, CMatrix,
139. rcount, displ, blocktype, 0, MPI\_COMM\_WORLD);
140. **if** (ProcRank == 0){
141. double endTime = MPI\_Wtime();
142. printf("That took %lf seconds\n",endTime-startTime);
143. }
144. TerminateProcess(AMatrix, BMatrix, CMatrix, Ablock, Bblock, Cblock);
145. **delete** [] displ;
146. **delete** [] rcount;
147. MPI\_Finalize();
148. **return** 0;
149. }

Приложение 3. MatrixOperations

1. #**include** "MatrixOperations.h"
2. #**include** <mpi.h>
3. #**include** <cstdlib>
4. #**include** <cstdio>
5. double **rand\_double**(){
6. **return** (double)rand()/RAND\_MAX\*50.0 - 2.0;
7. }
8. void **DataInitialization**(double\* pMatrix, int rowCount, int colCount) {
9. **for** (int i = 0; i < rowCount; i++){
10. **for** (int j = 0; j < colCount; j++){
11. pMatrix[i\*colCount + j] = rand\_double();
12. }
13. }
14. }
15. void **SetToZero**(double\* pMatrix, int rowCount, int colCount) {
16. **for** (int i = 0; i < rowCount; i++){
17. **for** (int j = 0; j < colCount; j++){
18. pMatrix[i\*colCount + j] = 0;
19. }
20. }
21. }
22. // Function for formatted vector output
23. void **PrintVector**(double\* pVector, int Size, int ProcNum) {
24. printf("proc #%d ", ProcNum);
25. // MPI\_Barrier(MPI\_COMM\_WORLD);
26. **for** (int i = 0; i < Size; i++)
27. printf("%7.4f ", pVector[i]);
28. // MPI\_Barrier(MPI\_COMM\_WORLD);
29. printf("\n");
30. }
31. // Function for formatted vector output
32. void **PrintVector**(int\* pVector, int Size, int ProcNum) {
33. printf("proc # %d ", ProcNum);
34. // MPI\_Barrier(MPI\_COMM\_WORLD);
35. **for** (int i = 0; i < Size; i++)
36. printf("%d ", pVector[i]);
37. // MPI\_Barrier(MPI\_COMM\_WORLD);
38. printf("\n");
39. }
40. // Function for formatted matrix output
41. void **PrintMatrix**(double\* pMatrix, int RowCount, int ColCount) {
42. int i, j; // Loop variables
43. **for** (i = 0; i < RowCount; i++) {
44. **for** (j = 0; j < ColCount; j++)
45. printf("%7.4f ", pMatrix[i \* ColCount + j]);
46. printf("\n");
47. }
48. }
49. // Function for matrix multiplication
50. void **MatrixMul**(double\* pAMatrix, double\* pBMatrix, double\* pCMatrix, int n1, int n2, int n3) {
51. int i, j, k; // Loop variables
52. **for** (i = 0; i < n1; i++) {
53. **for** (j = 0; j < n3; j++)
54. **for** (k = 0; k < n2; k++)
55. pCMatrix[i\*n3 + j] += pAMatrix[i\*n2 + k] \* pBMatrix[k\*n3 + j];
56. }
57. }