

```

1  /* 10. Multiply two square matrices (1000,2000 or 3000 dimensions). Compare
2  the performance of a sequential and parallel algorithm using open MP.*/
3
4  #include <iostream>
5  #include <vector>
6  #include <omp.h>  // OpenMP
7  #include <ctime>
8
9  using namespace std;
10
11 // Function to multiply two matrices sequentially
12 void multiplySequential(const vector<vector<int>>& A, const vector<vector<int>>& B, vector<vector<int>>& C, int N) {
13     for (int i = 0; i < N; ++i) {
14         for (int j = 0; j < N; ++j) {
15             C[i][j] = 0;
16             for (int k = 0; k < N; ++k) {
17                 C[i][j] += A[i][k] * B[k][j];
18             }
19         }
20     }
21 }
22
23 // Function to multiply two matrices using OpenMP
24 void multiplyParallel(const vector<vector<int>>& A, const vector<vector<int>>& B, vector<vector<int>>& C, int N) {
25     #pragma omp parallel for collapse(2)
26     for (int i = 0; i < N; ++i) {
27         for (int j = 0; j < N; ++j) {
28             C[i][j] = 0;
29             for (int k = 0; k < N; ++k) {
30                 C[i][j] += A[i][k] * B[k][j];
31             }
32         }
33     }
34 }
35
36 int main() {
37     int N; // Matrix size (1000, 2000, 3000)
38     cout << "Enter matrix size (e.g., 1000, 2000, 3000): ";
39     cin >> N;
40     clock_t start, end;
41
42     // Initialize matrices A, B, and C
43     vector<vector<int>> A(N, vector<int>(N, 1)); // Matrix A with all elements = 1
44     vector<vector<int>> B(N, vector<int>(N, 1)); // Matrix B with all elements = 1
45     vector<vector<int>> C(N, vector<int>(N, 0)); // Matrix C to store result
46
47     // Sequential multiplication
48     start = clock(); // double start = omp_get_wtime();
49     multiplySequential(A, B, C, N);
50     end = clock(); // double end = omp_get_wtime();
51     double durationSeq = double(end - start) / CLOCKS_PER_SEC; // double durationSeq = end - start;

```

```

52     cout << "Time taken for sequential multiplication: " << durationSeq << " seconds" << endl;
53
54     // Parallel multiplication using OpenMP
55     start = clock(); //start = omp_get_wtime();
56     multiplyParallel(A, B, C, N);
57     end = clock(); //end = omp_get_wtime();
58     double durationPar = double(end - start) / CLOCKS_PER_SEC; //double durationPar = end - start;
59     cout << "Time taken for parallel multiplication (OpenMP): " << durationPar << " seconds" << endl;
60
61     return 0;
62 }
63
64 /*
65 //OUTPUT:
66 Enter matrix size (e.g., 1000, 2000, 3000): 1000
67 Time taken for sequential multiplication: 6.82819 seconds
68 Time taken for parallel multiplication (OpenMP): 0.62969 seconds
69 */
70
71
72
73 /*6. Assume you have n robots which pick mangoes in a farm.
74 Wapt calculate the total number of mangoes picked by n robots parallely using MPI.*/
75
76 #include <mpi.h>
77 #include <iostream>
78 #include <cstdlib>
79 #include <ctime>
80
81 using namespace std;
82
83 int main(int argc, char** argv) {
84     MPI_Init(&argc, &argv); // Initialize MPI environment
85
86     // Get number of processes and rank of current process
87     int world_size, world_rank;
88     MPI_Comm_size(MPI_COMM_WORLD, &world_size);
89     MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
90
91     // Initialize random seed and generate random number of mangoes
92     srand(static_cast<unsigned>(time(0)) + world_rank); //srand(world_rank);
93     int mangoes_picked = rand() % 101;
94
95     cout << "Robot " << world_rank << " picked " << mangoes_picked << " mangoes." << endl;
96
97     // Use MPI_Reduce to calculate the total mangoes picked
98     int total_mangoes = 0;
99     MPI_Reduce(&mangoes_picked, &total_mangoes, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);
100
101     // Display the total mangoes picked by all robots (only on root process)
102     if (world_rank == 0) {

```

```

103         cout << "Total mangoes picked by all robots: " << total_mangoes << endl;
104     }
105
106     MPI_Finalize();
107     return 0;
108 }
109
110 /*
111 //OUTPUT:
112 Robot 2 picked 25 mangoes.
113 Robot 4 picked 6 mangoes.
114 Robot 0 picked 15 mangoes.
115 Robot 1 picked 29 mangoes.
116 Robot 3 picked 49 mangoes.
117 Total mangoes picked by all robots: 124
118 */
119
120
121
122 /*7. Design a program that implements application of
123 MPI Collective Communications.*/
124
125 #include <iostream>
126 #include <mpi.h>
127 #include <vector>
128 #include <cstdlib>
129
130 using namespace std;
131
132 int main(int argc, char** argv) {
133     MPI_Init(&argc, &argv);
134     int world_size, world_rank;
135     MPI_Comm_size(MPI_COMM_WORLD, &world_size);
136     MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
137     int n = 10;
138     vector<int> local_array(n);
139     int local_sum = 0, total_sum = 0;
140     srand(static_cast<unsigned>(time(0)) + world_rank);
141     for (int i = 0; i < n; ++i) {
142         local_array[i] = rand() % 100;
143         local_sum += local_array[i];
144     }
145     cout << "Process " << world_rank << " local array: ";
146     for (int i : local_array) cout << i << " ";
147     cout << "\nProcess " << world_rank << " local sum: " << local_sum << endl;
148     MPI_Reduce(&local_sum, &total_sum, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);
149     MPI_Bcast(&total_sum, 1, MPI_INT, 0, MPI_COMM_WORLD);
150     if (world_rank == 0) {
151         double average = static_cast<double>(total_sum) / (n * world_size);
152         cout << "Total sum: " << total_sum << endl;
153         cout << "Average: " << average << endl;

```

```

154     }
155     MPI_Finalize();
156     return 0;
157 }
158
159 /*
160 //OUTPUT:
161
162 student@student-HP-Pro-Tower-280-G9-E-PCI-Desktop-PC:~$ gedit akhpc7.cpp
163 student@student-HP-Pro-Tower-280-G9-E-PCI-Desktop-PC:~$ mpic++ akhpc7.cpp
164 student@student-HP-Pro-Tower-280-G9-E-PCI-Desktop-PC:~$ mpirun -np 5 ./a.out
165 Invalid MIT-MAGIC-COOKIE-1 key
166 Process 0 local array: 44 81 71 89 74 44 98 41 1 5
167 Process 0 local sum: 548
168 Process 3 local array: 62 42 67 68 15 31 40 14 96 2
169 Process 3 local sum: 437
170 Process 4 local array: 36 65 76 67 42 93 64 14 68 25
171 Process 4 local sum: 550
172 Process 2 local array: 36 8 70 74 74 29 58 11 69 79
173 Process 2 local sum: 508
174 Process 1 local array: 13 99 77 4 54 82 64 48 5 52
175 Process 1 local sum: 498
176 Total sum: 2541
177 Average: 50.82
178 */
179
180
181
182 // 8. Implement Cartesian Virtual Topology in MPI.
183
184 #include <iostream>
185 #include <mpi.h>
186 #include <vector>
187 using namespace std;
188
189 int main(int argc, char** argv) {
190     MPI_Init(&argc, &argv);
191
192     int world_size, world_rank;
193     MPI_Comm_size(MPI_COMM_WORLD, &world_size);
194     MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
195
196     int dims[2] = {0, 0};
197     MPI_Dims_create(world_size, 2, dims);
198
199     MPI_Comm cart_comm;
200     int period[2] = {0, 0}; // Periodicity in both dimensions
201     MPI_Cart_create(MPI_COMM_WORLD, 2, dims, period, true, &cart_comm);
202
203     int coords[2];
204     MPI_Comm_rank(cart_comm, &world_rank);

```

```

205     MPI_Cart_coords(cart_comm, world_rank, 2, coords);
206
207     int north, south, east, west;
208     MPI_Cart_shift(cart_comm, 0, 1, &north, &south);
209     MPI_Cart_shift(cart_comm, 1, 1, &west, &east);
210
211     int value = world_rank;
212     cout << "Process " << world_rank << " at (" << coords[0] << ", " << coords[1] << ") has value: " << value << endl;
213
214     if (north != MPI_PROC_NULL)
215         MPI_Send(&value, 1, MPI_INT, north, 0, cart_comm);
216     if (south != MPI_PROC_NULL)
217         MPI_Recv(&value, 1, MPI_INT, south, 0, cart_comm, MPI_STATUS_IGNORE);
218
219
220     if (west != MPI_PROC_NULL)
221         MPI_Send(&value, 1, MPI_INT, west, 0, cart_comm);
222     if (east != MPI_PROC_NULL)
223         MPI_Recv(&value, 1, MPI_INT, east, 0, cart_comm, MPI_STATUS_IGNORE);
224
225     MPI_Finalize();
226     return 0;
227 }
228
229 /*
230 //OUTPUT:
231
232 student@student-HP-Pro-Tower-280-G9-E-PCI-Desktop-PC:~$ gedit akhpc8.cpp
233 student@student-HP-Pro-Tower-280-G9-E-PCI-Desktop-PC:~$ mpic++ akhpc8.cpp
234 student@student-HP-Pro-Tower-280-G9-E-PCI-Desktop-PC:~$ mpirun -np 4 ./a.out
235 Invalid MIT-MAGIC-COOKIE-1 key
236 Process 0 at (0, 0) has value: 0
237 Process 1 at (0, 1) has value: 1
238 Process 2 at (1, 0) has value: 2
239 Process 3 at (1, 1) has value: 3
240 */
241
242
243
244 /*9. Design a MPI program that uses blocking send/receive
245 routines and non blocking send/receive routines.*/
246
247 #include <iostream>
248 #include <mpi.h>
249 #include <vector>
250
251 using namespace std;
252
253 int main(int argc, char** argv) {
254     MPI_Init(&argc, &argv);
255     int world_size, world_rank;

```

```

256 MPI_Comm_size(MPI_COMM_WORLD, &world_size);
257 MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
258
259 const int TAG = 0;
260 const int DATA_SIZE = 10;
261 vector<int> send_data(DATA_SIZE, world_rank);
262 vector<int> recv_data(DATA_SIZE);
263
264 if (world_rank == 0) {
265     cout << "Process 0 sending data: ";
266     for (int i : send_data) cout << i << " ";
267     cout << endl;
268     MPI_Send(send_data.data(), DATA_SIZE, MPI_INT, 1, TAG, MPI_COMM_WORLD);
269 }
270 else if (world_rank == 1) {
271     MPI_Recv(recv_data.data(), DATA_SIZE, MPI_INT, 0, TAG, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
272     cout << "Process 1 received data: ";
273     for (int i : recv_data) cout << i << " ";
274     cout << endl;
275 }
276
277 MPI_Request send_request, recv_request;
278
279 if (world_rank == 0) {
280     MPI_Isend(send_data.data(), DATA_SIZE, MPI_INT, 1, TAG, MPI_COMM_WORLD, &send_request);
281     cout << "Process 0 non-blocking send initiated." << endl;
282 }
283 else if (world_rank == 1) {
284     MPI_Irecv(recv_data.data(), DATA_SIZE, MPI_INT, 0, TAG, MPI_COMM_WORLD, &recv_request);
285     cout << "Process 1 non-blocking receive initiated." << endl;
286 }
287
288 if (world_rank == 0) {
289     MPI_Wait(&send_request, MPI_STATUS_IGNORE);
290     cout << "Process 0 non-blocking send completed." << endl;
291 }
292 else if (world_rank == 1) {
293     MPI_Wait(&recv_request, MPI_STATUS_IGNORE);
294     cout << "Process 1 non-blocking receive completed: ";
295     for (int i : recv_data) cout << i << " ";
296     cout << endl;
297 }
298
299 MPI_Finalize();
300 return 0;
301 }
302
303 /*
304 //OUTPUT:
305
306 student@student-HP-Pro-Tower-280-G9-E-PCI-Desktop-PC:~$ mpic++ akhpc9.cpp

```

```

307 student@student-HP-Pro-Tower-280-G9-E-PCI-Desktop-PC:~$ mpirun -np 9 ./a.out
308 Process 0 sending data: 0 0 0 0 0 0 0 0 0 0
309 Process 0 non-blocking send initiated.
310 Process 0 non-blocking send completed.
311 Process 1 received data: 0 0 0 0 0 0 0 0 0 0
312 Process 1 non-blocking receive initiated.
313 Process 1 non-blocking receive completed: 0 0 0 0 0 0 0 0 0 0
314 */
315
316
317
318
319
320
321
322
323 #include<stdio.h>
324 #include<stdlib.h>
325 #include<time.h>
326 #include<mpi.h>
327 int main(int argc, char* argv[])
328 {
329     int numtasks, rank, rc, count, next, prev, sz, inmsg;
330     MPI_Status Stat;
331     time_t st, et;
332     MPI_Init(&argc, &argv);
333     MPI_Comm_size(MPI_COMM_WORLD, &numtasks);
334     sz = (numtasks / 2) * 2;
335     MPI_Comm_rank(MPI_COMM_WORLD, &rank);
336     st = clock();
337     if (rank == 0) prev = sz - 1;
338     else prev = rank - 1;
339     if (rank == sz - 1) next = 0;
340     else next = rank + 1;
341     if (rank % 2 == 0 && rank < sz) {
342         rc = MPI_Send(&rank, 1, MPI_INT, next, 0, MPI_COMM_WORLD);
343         rc = MPI_Recv(&inmsg, 1, MPI_INT, prev, 1, MPI_COMM_WORLD, &Stat);
344     }
345     else if (rank % 2 == 1 && rank < sz) {
346         rc = MPI_Recv(&inmsg, 1, MPI_INT, prev, 0, MPI_COMM_WORLD, &Stat);
347         rc = MPI_Send(&rank, 1, MPI_INT, next, 1, MPI_COMM_WORLD);
348     }
349     MPI_Barrier(MPI_COMM_WORLD);
350     et = clock();
351     if(rank==0) printf("Time taken by Blocking send/receive : %lf\n", (double)(et - st) / CLOCKS_PER_SEC);
352     MPI_Barrier(MPI_COMM_WORLD);
353     MPI_Request reqs[2];
354     MPI_Status stats[2];
355     st = clock();
356     if (rank == numtasks - 1) next = 0;
357     else next = rank + 1;

```

```
358     if (rank == 0) prev = numtasks - 1;
359     else prev = rank - 1;
360     MPI_Irecv(&inmsg, 1, MPI_INT, prev, 0, MPI_COMM_WORLD, &reqs[0]);
361     MPI_Isend(&rank, 1, MPI_INT, next, 0, MPI_COMM_WORLD, &reqs[1]);
362     MPI_Barrier(MPI_COMM_WORLD);
363     et = clock();
364     if (rank == 0) printf("Time taken by NonBlocking send/receive : %lf\n", (double)(et - st) / CLOCKS_PER_SEC);
365     MPI_Finalize();
366 }
367
```