

```

1  /*6. Assume you have n robots which pick mangoes in a farm.
2  WAPT calculate the total number of mangoes picked by n robots parallely using MPI.*/
3
4  #include <mpi.h>
5  #include <iostream>
6  #include <cstdlib>
7  #include <ctime>
8
9  using namespace std;
10
11 int main(int argc, char** argv) {
12     MPI_Init(&argc, &argv);          // Initialize MPI environment
13
14     // Get number of processes and rank of current process
15     int world_size, world_rank;
16     MPI_Comm_size(MPI_COMM_WORLD, &world_size);
17     MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
18
19     // Initialize random seed and generate random number of mangoes
20     srand(static_cast<unsigned>(time(0)) + world_rank); //srand(world_rank); //ensures each process generates a diff
    random no. //or, srand((unsigned int)time(0) + world_rank);
21     int mangoes_picked = rand() % 101; //random number between 0 and 100, representing the number of mangoes picked by the
    robot
22
23     cout << "Robot " << world_rank << " picked " << mangoes_picked << " mangoes." << endl;
24
25     // Use MPI_Reduce to calculate the total mangoes picked
26     int total_mangoes = 0;
27     MPI_Reduce(&mangoes_picked, &total_mangoes, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);
28
29     // Display the total mangoes picked by all robots (only on root process)
30     if (world_rank == 0)
31         cout << "Total mangoes picked by all robots: " << total_mangoes << endl;
32
33     MPI_Finalize();
34     return 0;
35 }
36
37 /*
38 Robot 2 picked 25 mangoes.
39 Robot 4 picked 6 mangoes.
40 Robot 0 picked 15 mangoes.
41 Robot 1 picked 29 mangoes.
42 Robot 3 picked 49 mangoes.
43 Total mangoes picked by all robots: 124
44 */
45
46
47
48
49 /*7. Design a program that implements application of

```

```

50 MPI Collective Communications.*/
51
52 #include <iostream>
53 #include <mpi.h>
54 #include <vector>
55 #include <cstdlib>
56
57 using namespace std;
58
59 int main(int argc, char** argv) {
60     MPI_Init(&argc, &argv);
61
62     int world_size, world_rank;
63     MPI_Comm_size(MPI_COMM_WORLD, &world_size);
64     MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
65
66     int n = 10;
67     vector<int> local_array(n);
68     int local_sum = 0, total_sum = 0;
69
70     srand(static_cast<unsigned>(time(0)) + world_rank);
71     for (int i = 0; i < n; ++i) {
72         local_array[i] = rand() % 100;
73         local_sum += local_array[i];
74     }
75     cout << "Process " << world_rank << " local array: ";
76     for (int i : local_array) cout << i << " ";
77     cout << "\nProcess " << world_rank << " local sum: " << local_sum << endl;
78
79     MPI_Reduce(&local_sum, &total_sum, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);
80     MPI_Bcast(&total_sum, 1, MPI_INT, 0, MPI_COMM_WORLD);
81
82     if (world_rank == 0) {
83         double average = static_cast<double>(total_sum) / (n * world_size);
84         cout << "Total sum: " << total_sum << endl;
85         cout << "Average: " << average << endl;
86     }
87     MPI_Finalize();
88     return 0;
89 }
90
91 /*
92 $ gedit akhpc7.cpp
93 $ mpic++ akhpc7.cpp
94 $ mpirun -np 5 ./a.out
95
96 Process 0 local array: 44 81 71 89 74 44 98 41 1 5
97 Process 0 local sum: 548
98 Process 3 local array: 62 42 67 68 15 31 40 14 96 2
99 Process 3 local sum: 437
100 Process 4 local array: 36 65 76 67 42 93 64 14 68 25

```

```

101 Process 4 local sum: 550
102 Process 2 local array: 36 8 70 74 74 29 58 11 69 79
103 Process 2 local sum: 508
104 Process 1 local array: 13 99 77 4 54 82 64 48 5 52
105 Process 1 local sum: 498
106 Total sum: 2541
107 Average: 50.82
108 */
109
110
111
112
113 // 8. Implement Cartesian Virtual Topology in MPI.
114
115 #include <iostream>
116 #include <mpi.h>
117 #include <vector>
118 using namespace std;
119
120 int main(int argc, char** argv) {
121     MPI_Init(&argc, &argv);
122
123     int world_size, world_rank;
124     MPI_Comm_size(MPI_COMM_WORLD, &world_size);
125     MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
126
127     int dims[2] = {0, 0};
128     MPI_Dims_create(world_size, 2, dims);
129
130     MPI_Comm cart_comm;
131     int period[2] = {0, 0}; // Periodicity in both dimensions
132     MPI_Cart_create(MPI_COMM_WORLD, 2, dims, period, true, &cart_comm);
133
134     int coords[2];
135     MPI_Comm_rank(cart_comm, &world_rank); // can comment this lineeee
136     MPI_Cart_coords(cart_comm, world_rank, 2, coords);
137
138     int north, south, east, west;
139     MPI_Cart_shift(cart_comm, 0, 1, &north, &south);
140     MPI_Cart_shift(cart_comm, 1, 1, &west, &east);
141
142     int value = world_rank;
143     cout << "Process " << world_rank << " at (" << coords[0] << ", " << coords[1] << ") has value: " << value << endl;
144
145     if (north != MPI_PROC_NULL) MPI_Send(&value, 1, MPI_INT, north, 0, cart_comm);
146     if (south != MPI_PROC_NULL) MPI_Recv(&value, 1, MPI_INT, south, 0, cart_comm, MPI_STATUS_IGNORE);
147     if (west != MPI_PROC_NULL) MPI_Send(&value, 1, MPI_INT, west, 0, cart_comm);
148     if (east != MPI_PROC_NULL) MPI_Recv(&value, 1, MPI_INT, east, 0, cart_comm, MPI_STATUS_IGNORE);
149
150     MPI_Finalize();
151     return 0;

```

```

152 }
153
154 /*
155 $ gedit akhpc8.cpp
156 $ mpic++ akhpc8.cpp
157 $ mpirun -np 4 ./a.out
158
159 Process 0 at (0, 0) has value: 0
160 Process 1 at (0, 1) has value: 1
161 Process 2 at (1, 0) has value: 2
162 Process 3 at (1, 1) has value: 3
163 */
164
165
166
167
168 /*9. Design a MPI program that uses blocking send/receive
169 routines and non blocking send/receive routines.*/
170
171 #include <iostream>
172 #include <mpi.h>
173 #include <vector>
174
175 using namespace std;
176
177 int main(int argc, char** argv) {
178     MPI_Init(&argc, &argv);
179     int world_size, world_rank;
180     MPI_Comm_size(MPI_COMM_WORLD, &world_size);
181     MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
182
183     const int TAG = 0;
184     const int DATA_SIZE = 10;
185     vector<int> send_data(DATA_SIZE, world_rank);
186     vector<int> recv_data(DATA_SIZE);
187
188     if (world_rank == 0) {
189         cout << "Process 0 sending data: ";
190         for (int i : send_data) cout << i << " ";
191         cout << endl;
192         MPI_Send(send_data.data(), DATA_SIZE, MPI_INT, 1, TAG, MPI_COMM_WORLD);
193     }
194     else if (world_rank == 1) {
195         MPI_Recv(recv_data.data(), DATA_SIZE, MPI_INT, 0, TAG, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
196         cout << "Process 1 received data: ";
197         for (int i : recv_data) cout << i << " ";
198         cout << endl;
199     }
200
201     MPI_Request send_request, recv_request;
202

```

```

203     if (world_rank == 0) {
204         MPI_Isend(send_data.data(), DATA_SIZE, MPI_INT, 1, TAG, MPI_COMM_WORLD, &send_request);
205         cout << "Process 0 non-blocking send initiated." << endl;
206     }
207     else if (world_rank == 1) {
208         MPI_Irecv(recv_data.data(), DATA_SIZE, MPI_INT, 0, TAG, MPI_COMM_WORLD, &recv_request);
209         cout << "Process 1 non-blocking receive initiated." << endl;
210     }
211
212     if (world_rank == 0) {
213         MPI_Wait(&send_request, MPI_STATUS_IGNORE);
214         cout << "Process 0 non-blocking send completed." << endl;
215     }
216     else if (world_rank == 1) {
217         MPI_Wait(&recv_request, MPI_STATUS_IGNORE);
218         cout << "Process 1 non-blocking receive completed: ";
219         for (int i : recv_data) cout << i << " ";
220         cout << endl;
221     }
222
223     MPI_Finalize();
224     return 0;
225 }
226
227 /*
228 $ mpic++ akhpc9.cpp
229 $ mpirun -np 9 ./a.out
230
231 Process 0 sending data: 0 0 0 0 0 0 0 0 0 0
232 Process 0 non-blocking send initiated.
233 Process 0 non-blocking send completed.
234 Process 1 received data: 0 0 0 0 0 0 0 0 0 0
235 Process 1 non-blocking receive initiated.
236 Process 1 non-blocking receive completed: 0 0 0 0 0 0 0 0 0 0
237 */
238
239
240
241
242
243 /* 10. Multiply two square matrices (1000,2000 or 3000 dimensions). Compare
244 the performance of a sequential and parallel algorithm using open MP.*/
245
246 #include <iostream>
247 #include <vector>
248 #include <omp.h>    // OpenMP
249 #include <ctime>
250
251 using namespace std;
252
253 // Function to multiply two matrices sequentially

```

```

254 void multiplySequential(const vector<vector<int>>& A, const vector<vector<int>>& B, vector<vector<int>>& C, int N) {
255     for (int i = 0; i < N; ++i) {
256         for (int j = 0; j < N; ++j) {
257             C[i][j] = 0;
258             for (int k = 0; k < N; ++k) {
259                 C[i][j] += A[i][k] * B[k][j];
260             }
261         }
262     }
263 }
264
265 // Function to multiply two matrices using OpenMP
266 void multiplyParallel(const vector<vector<int>>& A, const vector<vector<int>>& B, vector<vector<int>>& C, int N) {
267     #pragma omp parallel for collapse(2)
268     for (int i = 0; i < N; ++i) {
269         for (int j = 0; j < N; ++j) {
270             C[i][j] = 0;
271             for (int k = 0; k < N; ++k) {
272                 C[i][j] += A[i][k] * B[k][j];
273             }
274         }
275     }
276 }
277
278 int main() {
279     int N; // Matrix size (1000, 2000, 3000)
280     cout << "Enter matrix size (e.g., 1000, 2000, 3000): ";
281     cin >> N;
282     clock_t start, end;
283
284     // Initialize matrices A, B, and C
285     vector<vector<int>> A(N, vector<int>(N, 1)); // Matrix A with all elements = 1
286     vector<vector<int>> B(N, vector<int>(N, 1)); // Matrix B with all elements = 1
287     vector<vector<int>> C(N, vector<int>(N, 0)); // Matrix C to store result
288
289     // Sequential multiplication
290     start = clock(); // double start = omp_get_wtime();
291     multiplySequential(A, B, C, N);
292     end = clock(); // double end = omp_get_wtime();
293     double durationSeq = double(end - start) / CLOCKS_PER_SEC; // double durationSeq = end - start;
294     cout << "Time taken for sequential multiplication: " << durationSeq << " seconds" << endl;
295
296     // Parallel multiplication using OpenMP
297     start = clock(); //start = omp_get_wtime();
298     multiplyParallel(A, B, C, N);
299     end = clock(); //end = omp_get_wtime();
300     double durationPar = double(end - start) / CLOCKS_PER_SEC; //double durationPar = end - start;
301     cout << "Time taken for parallel multiplication (OpenMP): " << durationPar << " seconds" << endl;
302
303     return 0;
304 }

```

```
305
306  /*
307  //OUTPUT:
308  Enter matrix size (e.g., 1000, 2000, 3000): 1000
309  Time taken for sequential multiplication: 6.82819 seconds
310  Time taken for parallel multiplication (OpenMP): 0.62969 seconds
311  */
```