

01.cpp

```
1
2
3 #include <iostream>
4 #include "mpi.h"
5 using namespace std;
6
7 int main(int argc, char **argv)
8 {
9     char name[1024];
10    int length=1024, major, minor;
11
12    MPI_Init(&argc, &argv);
13
14    MPI_Get_processor_name(name, &length);
15    cout << "\nHello from " << name;
16
17    MPI_Get_version(&major, &minor);
18    cout << "\nMPI Version " << major << "." << minor;
19    cout << "\nFrom header: Version " << MPI_VERSION << "." << MPI_SUBVERSION;
20
21    MPI_Finalize();
22
23    return 0;
24 }
25
```

02.cpp

```
26
27 #include <iostream>
28 #include "mpi.h"
29
30 int main(int argc, char **argv)
31 {
32     int rank, size;
33     MPI_Init(&argc, &argv);
34
35     MPI_Comm_rank(MPI_COMM_WORLD, &rank);
36     MPI_Comm_size(MPI_COMM_WORLD, &size);
37
38     std::cout << "\nSize=" << size << ", MyRank:" << rank;
39
40     MPI_Finalize();
41     return 0;
42 }
43
```

03.cpp

```
44
45 #include <iostream>
46 #include <cstdlib>
47 #include "mpi.h"
48
49 /* This program uses multiple processors to compute sum of some integers */
50 int main(int argc, char **argv)
51 {
52     int rank, size, upper=0;
53     int nPerNode;
54     int begin, end, i;
55     unsigned int sum, allSum;
56     double t1, t2;
57
58     MPI_Init(&argc, &argv);
59
60     // Get processor configurations
61     MPI_Comm_rank(MPI_COMM_WORLD, &rank);
62     MPI_Comm_size(MPI_COMM_WORLD, &size);
63
64     // Interactively enter a number from root when necessary
65     if(rank==0) {
66         if(argc>1) upper = atoi(argv[1]);
67         if(upper<=1) {
68             std::cout << "\nThis program computes the sum between 1 and the integer you're going to enter.";
69             std::cout << "\nPlease enter an integer: ";
70             std::cin >> upper;
71         }
72     }
73
74     // Record the start time
75     t1 = MPI_Wtime();
76
77     // Notify every participating processor the integer
```

```

78 MPI_Bcast(&upper, 1, MPI_INT, 0, MPI_COMM_WORLD);
79
80 // Now divide the work, we're assuming every processor is equally powerful (or weak)
81 // 50 -> 4 nodes --> nPerNode = 12: 1 - 12, 13 - 24, 25 - 36, 37 - 50
82 nPerNode = upper / size;
83 begin = rank * nPerNode + 1;
84 end = begin + nPerNode - 1;
85
86 // special case for the last processor
87 if(rank == (size-1)) {
88     end = upper;
89 }
90
91 // Now every node does its job
92 sum = 0;
93 for(i=begin; i<=end; i++) {
94     sum += i;
95 }
96
97 // Now collect results back to root
98 MPI_Reduce(&sum, &allSum, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);
99
100 // Calculate the total wall-clock time
101 if(rank==0) {
102     t2 = MPI_Wtime() - t1;
103     std::cout << "\n" << size << ", sum(1.." << upper << " = " << allSum << ", " << t2 << " seconds.";
104 }
105
106 // Clean up
107 MPI_Finalize();
108
109 return 0;
110 }
111

```

112 04.cpp

```

113 #include <iostream>
114 #include <cstdlib>
115 #include "mpi.h"
116 using namespace std;
117
118 double intf(double x);
119 double f(double x);
120
121 // This program integrates f(x) between [lower] and [upper] with N divisions.
122 // Algorithm: Trapezoidal rule
123 // function f is the function to be integrated
124 // function intf is the analytical solution
125
126 // 3x^2 + 2x + 3 --> x^3 + x^2 + 3x
127 double intf(double x) {
128     return x*x*x + x*x + 3*x;
129 }
130
131 // 3x^2 + 2x + 3 --> x^3 + x^2 + 3x
132 double f(double x) {
133     return 3*x*x + 2*x + 3;
134 }
135
136 // Main program
137 int main(int argc, char **argv)
138 {
139     int size, rank;
140     double upper, lower;
141     int N;
142     int i;
143     double myLower, X, L, L2;
144     double fLower, fUpper;
145     double area, answer, analytical;
146     double t1, t2;
147
148     // Initialization
149     MPI_Init(&argc, &argv);
150
151     // Get processor configurations
152     MPI_Comm_rank(MPI_COMM_WORLD, &rank);
153     MPI_Comm_size(MPI_COMM_WORLD, &size);
154
155     // Get some data

```

```

156     if(rank == 0) {
157         if(argc==4) {
158             lower = atof(argv[1]);
159             upper = atof(argv[2]);
160             N      = atoi(argv[3]);
161         }
162         else {
163             cout << "\nLower bound: ";
164             cin >> lower;
165             cout << "\nUpper bound: ";
166             cin >> upper;
167             cout << "\nDivision per processor: ";
168             cin >> N;
169         }
170     }
171
172     // Timing
173     t1 = MPI_Wtime();
174
175     // Broadcast information
176     MPI_Bcast(&lower, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD);
177     MPI_Bcast(&upper, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD);
178     MPI_Bcast(&N, 1, MPI_INT, 0, MPI_COMM_WORLD);
179
180     // Now each processor decides what to do
181     L = (upper - lower) / size;
182     myLower = rank * L + lower;
183     L /= N;
184     L2 = L * 0.5;
185
186     // Now calculate area of each division
187     area = 0;
188     fLower = f(myLower);
189     X = myLower;
190     for(i=0;i<N;i++) {
191         X += L;
192         fUpper = f(X);
193         area += (fLower + fUpper) * L2;
194         fLower = fUpper;
195     }
196
197     // Now sum the area up.
198     MPI_Reduce(&area, &answer, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
199
200     // timing
201     t2 = MPI_Wtime();
202
203     // The answer is:
204     if(rank == 0) {
205         analytical = intf(upper) - intf(lower);
206         cout << "\nThe answer is " << answer << "(" << analytical << ")";
207         cout << "\nThe error is " << (answer - analytical) / analytical * 100;
208         cout << "\nThe solution time: " << t2-t1 << " seconds.";
209     }
210
211     // Cleanup
212     MPI_Finalize();
213
214     return 0;
215 }

```

05.cpp

```

217 #include <iostream>
218 #include <cstdlib>
219 #include "mpi.h"
220
221 using namespace std;
222
223 // This program generates the desired number of random numbers, and
224 // calculate the mean, the max, and the min. of these random numbers
225 int main(int argc, char **argv)
226 {
227     int rank, size;
228     int N;
229     int myShare, i;
230     double *data;
231     double myMax, myMin, mySum;
232
233

```

```

234 int totalN;
235 double allMax, allMin, allSum;
236
237 // Initialization
238 MPI_Init(&argc, &argv);
239
240 // Get processor configurations
241 MPI_Comm_rank(MPI_COMM_WORLD, &rank);
242 MPI_Comm_size(MPI_COMM_WORLD, &size);
243
244 // Get necessary input data
245 if(rank == 0) {
246     if(argc == 2) {
247         N = atoi(argv[1]);
248     }
249     while(N <= 1) {
250         cout << "\nEnter the number of random numbers you want: (must be larger than 1)";
251         cin >> N;
252     }
253 }
254
255 // Announce the number of data to be generated
256 MPI_Bcast(&N, 1, MPI_INT, 0, MPI_COMM_WORLD);
257
258 // Determine how many random numbers I need to generate
259 myShare = N / size;
260
261 // Last process may need to generate different number of data than others
262 if(rank == (size - 1)) {
263     myShare = N - (size-1) * myShare;
264 }
265
266 // allocate memory
267 data = new double[myShare];
268
269 if(data != NULL) {
270     // generate data
271     // set random number seed, otherwise, all processes use the same default
272     // random seed (1), and produce identical output of random numbers.
273     srand(rank);
274     for(i=0;i<myShare;i++) {
275         data[i] = (double) rand()/RAND_MAX;
276         cout << "(" << data[i] << ")";
277     }
278
279     // Find the max. and min. of the data, and calculate sum
280     myMax = data[0];
281     myMin = data[0];
282     mySum = data[0];
283     for(i=1;i<myShare;i++) {
284         mySum += data[i];
285         if(data[i] > myMax) myMax = data[i];
286         if(data[i] < myMin) myMin = data[i];
287     }
288
289     // free memory and discard data
290     delete []data;
291 }
292 else {
293     // failed to allocate memory
294     myShare = 0;
295 }
296
297 // Now, collect data back to root (rank 0) process
298 MPI_Reduce(&myShare, &totalN, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);
299 MPI_Reduce(&myMax, &allMax, 1, MPI_DOUBLE, MPI_MAX, 0, MPI_COMM_WORLD);
300 MPI_Reduce(&myMin, &allMin, 1, MPI_DOUBLE, MPI_MIN, 0, MPI_COMM_WORLD);
301 MPI_Reduce(&mySum, &allSum, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
302
303 if(rank==0) {
304     cout << "\nN: " << totalN;
305     cout << "\nMax: " << allMax;
306     cout << "\nMin: " << allMin;
307     cout << "\nAvg: " << allSum/totalN;
308 }
309 MPI_Finalize();
310
311 return 0;
312 }

```