

08.cpp

```
1  #include <cassert>
2  #include <cmath>
3  #include <cstdlib>
4  #include <iostream>
5
6
7  #include "mpi.h"
8  #define maxNP 32
9  using namespace std;
10
11 // randomly generate a vector
12 double *genVector(const int N, const int seed)
13 {
14     int i;
15     double *A;
16
17     A = new (nothrow) double[N];
18     assert(A);
19
20     srand(seed);
21     for(i=0;i<N;i++) {
22         A[i] = ((double) rand() / RAND_MAX);
23     }
24
25     return A;
26 }
27
28 // get a vector size either from command line or from console
29 int getSize(int argc, char **argv)
30 {
31     int length=0;
32     // Determine the vector size
33     if(argc>1) {
34         length = atoi(argv[1]);
35     }
36     while(length<=1) {
37         cout << "\nPlease enter the length of the vector: ";
38         cin >> length;
39     }
40
41     return length;
42 }
43
44 // Calculate inner product of two vectors, version 2
45 int main(int argc, char **argv)
46 {
47     int rank, size;
48     double *v1=NULL;
49     int length = 0;
50     double t1, t2;
51     int work;
52     int count[maxNP], displ[maxNP];
53
54     int mySize, i;
55     double *myData;
56     double localSum, vLength;
57
58     // Initialization
59     MPI_Init(&argc, &argv);
60
61     // Get environment information
62     MPI_Comm_rank(MPI_COMM_WORLD, &rank);
63     MPI_Comm_size(MPI_COMM_WORLD, &size);
64
65     if(size > maxNP) {
66         if(rank==0) {
67             cout << "\nThis program is not capable of using more than " << maxNP << " processes.";
68         }
69         MPI_Finalize();
70         return 255;
71     }
72
73     if(rank==0) {
74         length = getSize(argc, argv);
75         v1 = genVector(length, 100);
76
77         // determine how work is going to be distributed
78         work = length / size;
79         displ[0] = 0;
80         for(i=0;i<size-1;i++) {
81             count[i] = work;
82             displ[i+1] = displ[i] + work;
83         }
84         count[i] = length - (size-1) * work;
```

```

85     }
86
87     t1 = MPI_Wtime();
88
89     // let everyone knows how much work they have to do
90     MPI_Scatter(count, 1, MPI_INT, &mySize, 1, MPI_INT, 0, MPI_COMM_WORLD);
91
92     // every one allocate memory
93     myData = new double[mySize];
94
95     // I am assuming memory allocation will success, which is not good!!!
96     MPI_Scatterv(v1, count, displ, MPI_DOUBLE, myData, mySize, MPI_DOUBLE, 0, MPI_COMM_WORLD);
97
98     // now every node does computation
99     localSum = 0.0;
100    for(i=0;i<mySize;i++) {
101        localSum += myData[i] * myData[i];
102    }
103
104    // sum all squares
105    MPI_Allreduce(&localSum, &vLength, 1, MPI_DOUBLE, MPI_SUM, MPI_COMM_WORLD);
106    vLength = sqrt(vLength);
107
108    if(vLength != 0.0) {
109        // Now normalize the vector
110        for(i=0;i<mySize;i++) {
111            myData[i] /= vLength;
112        }
113    }
114
115    // now send results back to the root
116    MPI_Gatherv(myData, mySize, MPI_DOUBLE, v1, count, displ, MPI_DOUBLE, 0, MPI_COMM_WORLD);
117
118    t2 = MPI_Wtime() - t1;
119    if(rank==0) {
120        cout << "\nNormalization takes " << t2 << " seconds.";
121    }
122
123    // Finalize
124    MPI_Finalize();
125
126    delete []v1;
127
128    return 0;
129 }

```

130 09a_noBarrier.cpp

```

131 #include <iostream>
132 #include "mpi.h"
133
134 int main(int argc, char **argv) {
135     int i;
136     int rank;
137
138     MPI_Init(&argc, &argv);
139     MPI_Comm_rank(MPI_COMM_WORLD, &rank);
140     for(i=0;i<10;i++) {
141         std::cout << rank << ": " << i << std::endl;
142     }
143     std::cout << rank << ": " << " done!" << std::endl;
144
145     MPI_Finalize();
146
147     return 0;
148 }
149

```

150 09b_barrier.cpp

```

151 #include <iostream>
152 #include "mpi.h"
153
154 int main(int argc, char **argv) {
155     int i;
156     int rank;
157
158     MPI_Init(&argc, &argv);
159     MPI_Comm_rank(MPI_COMM_WORLD, &rank);
160     for(i=0;i<10;i++) {
161         std::cout << rank << ": " << i << std::endl;
162         MPI_Barrier(MPI_COMM_WORLD);
163     }
164     std::cout << rank << ": " << " done!" << std::endl;

```

```
165 MPI_Finalize();
166
167
168 return 0;
169 }
170
```

171 10_scan.cpp

```
172 #include <iostream>
173 #include "mpi.h"
174 #define N 3
175
176 int main(int argc, char **argv) {
177     int i;
178     int rank;
179     int a[N], b[N];
180
181     MPI_Init(&argc, &argv);
182     MPI_Comm_rank(MPI_COMM_WORLD, &rank);
183     for(i=0;i<N;i++) {
184         a[i] = i+rank*N;
185     }
186
187     std::cout << "\n" << rank << ": a[]=";
188     for(i=0;i<N;i++) {
189         std::cout << a[i] << ", ";
190     }
191
192     MPI_Scan(a, b, N, MPI_INT, MPI_SUM, MPI_COMM_WORLD);
193
194     std::cout << "\n" << rank << ": b[]=";
195     for(i=0;i<N;i++) {
196         std::cout << b[i] << ", ";
197     }
198
199     MPI_Finalize();
200
201     return 0;
202 }
203
```

204

205 11_ring_1.cpp

```
206 #include "mpi.h"
207
208 int main(int argc, char **argv)
209 {
210     int rank, size;
211     int otherRank;
212     int tag = 100;
213     int to, from;
214     MPI_Status status;
215
216     // Initialization
217     MPI_Init(&argc, &argv);
218
219     // Get environment information
220     MPI_Comm_rank(MPI_COMM_WORLD, &rank);
221     MPI_Comm_size(MPI_COMM_WORLD, &size);
222
223     // Pass its rank to the right until it gets it back
224     otherRank = rank;
225     to = (rank + 1) % size;
226     if(rank != 0) {
227         from = (rank - 1);
228     }
229     else {
230         from = size - 1;
231     }
232
233     do {
234         MPI_Send(&otherRank, 1, MPI_INT, to, tag, MPI_COMM_WORLD);
235         MPI_Recv(&otherRank, 1, MPI_INT, from, tag, MPI_COMM_WORLD, &status);
236     } while(otherRank != rank);
237
238     // Finalize
239     MPI_Finalize();
240
241     return 0;
242 }
243
```

244 12_ring_2.cpp

```
245 #include <iostream>
246 #include "mpi.h"
247
248 int main(int argc, char **argv)
249 {
250     int rank, size;
251     int otherRank, tmp;
252     int tag = 100;
253     int to, from;
254     double t1, t2;
255     MPI_Status status;
256
257     // Initialization
258     MPI_Init(&argc, &argv);
259
260     // Get environment information
261     MPI_Comm_rank(MPI_COMM_WORLD, &rank);
262     MPI_Comm_size(MPI_COMM_WORLD, &size);
263
264     // Pass its rank to the right until it gets it back
265     otherRank = rank;
266     to = (rank + 1) % size;
267     if(rank != 0) {
268         from = (rank - 1);
269     }
270     else {
271         from = size - 1;
272     }
273
274     t1 = MPI_Wtime();
275     do {
276         if(rank % 2 == 0) {
277             MPI_Send(&otherRank, 1, MPI_INT, to, tag, MPI_COMM_WORLD);
278             MPI_Recv(&otherRank, 1, MPI_INT, from, tag, MPI_COMM_WORLD, &status);
279             if(rank==0) std::cout << "[" << otherRank << "]";
280         }
281         else {
282             MPI_Recv(&tmp, 1, MPI_INT, from, tag, MPI_COMM_WORLD, &status);
283             MPI_Send(&otherRank, 1, MPI_INT, to, tag, MPI_COMM_WORLD);
284             otherRank = tmp;
285         }
286     }
```

```

286     } while(otherRank != rank);
287     t2 = MPI_Wtime();
288
289     if(rank==0) {
290         std::cout << "\nTime: " << t2-t1 << std::endl;
291     }
292
293
294     // Finalize
295     MPI_Finalize();
296
297     return 0;
298 }
299

```

300 13_ring_3.cpp

```

301 #include <iostream>
302 #include "mpi.h"
303
304 int main(int argc, char **argv)
305 {
306     int rank, size;
307     int otherRank, tmp;
308     int tag = 100;
309     int to, from;
310     double t1, t2;
311     MPI_Status status;
312
313     // Initialization
314     MPI_Init(&argc, &argv);
315
316     // Get environment information
317     MPI_Comm_rank(MPI_COMM_WORLD, &rank);
318     MPI_Comm_size(MPI_COMM_WORLD, &size);
319
320     // Pass its rank to the right until it travels back
321     otherRank = rank;
322     to = (rank + 1) % size;
323     if(rank != 0) {
324         from = (rank - 1);
325     }
326     else {
327         from = size - 1;
328     }
329
330     t1 = MPI_Wtime();
331     do {
332         if(rank % 2 == 0) {
333             MPI_Sendrecv_replace(&otherRank, 1, MPI_INT, to, tag, from,
334                 tag, MPI_COMM_WORLD, &status);
335             /*
336              MPI_Sendrecv(&otherRank, 1, MPI_INT, to, tag,
337                  &otherRank, 1, MPI_INT, from, tag,
338                  MPI_COMM_WORLD, &status);
339             */
340             if(rank == 0) std::cout << "[" << otherRank << " ";
341         }
342         else {
343             MPI_Recv(&tmp, 1, MPI_INT, from, tag, MPI_COMM_WORLD, &status);
344             MPI_Send(&otherRank, 1, MPI_INT, to, tag, MPI_COMM_WORLD);
345             otherRank = tmp;
346         }
347     } while(otherRank != rank);
348     t2 = MPI_Wtime();
349     if(rank==0) {
350         std::cout << "\nTime: " << t2-t1 << std::endl;
351     }
352
353     // Finalize
354     MPI_Finalize();
355
356     return 0;
357 }
358
359

```

360

14_ring_4.cpp

```

361 #include <iostream>
362 #include "mpi.h"
363
364 int main(int argc, char **argv)
365 {
366     int rank, size;
367     int otherRank, tmp;
368     int tag = 100;
369     int to, from;
370     double t1, t2;
371     MPI_Request request[2];
372     MPI_Status status[2];
373
374     // Initialization
375     MPI_Init(&argc, &argv);
376
377     // Get environment information
378     MPI_Comm_rank(MPI_COMM_WORLD, &rank);
379     MPI_Comm_size(MPI_COMM_WORLD, &size);
380
381     // Pass its rank to the right until it gets it back
382     otherRank = rank;
383     to = (rank + 1) % size;
384     if(rank != 0) {
385         from = (rank - 1);
386     }
387     else {
388         from = size - 1;
389     }
390
391     t1 = MPI_Wtime();
392     do {
393         MPI_Isend(&otherRank, 1, MPI_INT, to, tag, MPI_COMM_WORLD, &(request[0]));
394         MPI_Irecv(&tmp, 1, MPI_INT, from, tag, MPI_COMM_WORLD, &(request[1]));
395         MPI_Waitall(2, request, status);
396         otherRank = tmp;
397         if(rank==0) std::cout << "[" << otherRank << "];"
398     } while(otherRank != rank);
399     t2 = MPI_Wtime();
400
401     if(rank==0) {
402         std::cout << "\nTime: " << t2-t1 << std::endl;
403     }
404     // Finalize
405     MPI_Finalize();
406
407     return 0;
408 }
409
410

```

411

15_ring_5.cpp

```

412 #include <iostream>
413 #include "mpi.h"
414
415 int main(int argc, char **argv)
416 {
417     int rank, size;
418     int periods;
419     MPI_Comm newCom;
420
421     int to, from;
422
423     int otherRank, tmp;
424     int tag = 100;
425     MPI_Request request[2];
426     MPI_Status status[2];
427
428     // Initialization
429     MPI_Init(&argc, &argv);
430
431     // Get environment information
432     MPI_Comm_rank(MPI_COMM_WORLD, &rank);
433     MPI_Comm_size(MPI_COMM_WORLD, &size);
434
435     // setup ring topology (1: one-dimension,
436     //     &size --> number of processes in each dimension
437     //     &periods --> whether the grid is periodic or not
438     //     0 --> ranking may not be re-ordered
439     periods = 1;

```

```

440 MPI_Cart_create(MPI_COMM_WORLD, 1, &size, &periods, 1, &newCom);
441
442 // 0: direction, 1: displacement
443 // &from, &to
444 MPI_Cart_shift(newCom, 0, 1, &from, &to);
445
446 // Initially pass its rank to the right until it gets it back
447 otherRank = rank;
448
449 do {
450     MPI_Isend(&otherRank, 1, MPI_INT, to, tag, newCom, &(request[0]));
451     MPI_Irecv(&tmp, 1, MPI_INT, from, tag, newCom, &(request[1]));
452     MPI_Waitall(2, request, status);
453     otherRank = tmp;
454     if(rank==0) std::cout << "[" << otherRank << "];";
455 } while(otherRank != rank);
456
457 // Finalize
458 MPI_Finalize();
459
460 return 0;
461 }
462

```

463 16_cart.cpp

```

464 #include <iostream>
465 #include "mpi.h"
466 using namespace std;
467
468 #define UP 0
469 #define DOWN 1
470 #define LEFT 2
471 #define RIGHT 3
472
473 int main(int argc, char **argv)
474 {
475     int myRank1, myRank2;
476     int size[2]={3,3};
477     int periods[2] = {1,0};
478     int reorder=1;
479     MPI_Comm newCom;
480
481     int myCoord[2];
482     int tmp[2];
483     int maxdims=2;
484
485     int who;
486     int ngrbr[4];
487
488     // Initialization
489     MPI_Init(&argc, &argv);
490
491     // get my rank in the World communicator
492     MPI_Comm_rank(MPI_COMM_WORLD, &myRank1);
493
494     // setup 2D cartesian topology (1: one-dimension,
495     MPI_Cart_create(MPI_COMM_WORLD, 2, size, periods, reorder, &newCom);
496
497     // get my rank in the newly created cartesian communicator
498     MPI_Comm_rank(newCom, &myRank2);
499
500     // query my coord in the created cartesian topology
501     MPI_Cart_coords(newCom, myRank2, maxdims, myCoord);
502
503     cout << "\n" << myRank1 << " : (" << myRank2 << ", " << myCoord[0] << ", " << myCoord[1] << ")";
504
505     // Query who is at coord (1,2)
506     tmp[0] = 1;
507     tmp[1] = 2;
508     MPI_Cart_rank(newCom, tmp, &who);
509     cout << "\n" << who << " is at (1,2)";
510
511     // Find out about neighbors
512     MPI_Cart_shift(newCom, 0, 1, &ngrbr[LEFT], &ngrbr[RIGHT]);
513     MPI_Cart_shift(newCom, 1, 1, &ngrbr[UP], &ngrbr[DOWN]);
514     cout << "\n" << myRank2 << "(" << ngrbr[LEFT] << ", " << ngrbr[RIGHT] << " <<
515         ", " << ngrbr[UP] << ", " << ngrbr[DOWN] << ") as neighbors.";
516
517     MPI_Finalize();
518
519     return 0;
520 }

```

521

17_group.cpp

```
522 #include <iostream>
523 #include "mpi.h"
524 using namespace std;
525
526 int main(int argc, char **argv)
527 {
528     int rank, size;
529     int newSize;
530     int *gRanks;
531     MPI_Group gWorld, newGroup;
532     MPI_Comm newComm;
533
534     int i, pts;
535     int newRank;
536
537     // Initialization
538     MPI_Init(&argc, &argv);
539
540     // Get environment information
541     MPI_Comm_rank(MPI_COMM_WORLD, &rank);
542     MPI_Comm_size(MPI_COMM_WORLD, &size);
543
544     // Get the member of MPI_COMM_WORLD to g1 & g2 group handles
545     MPI_Comm_group(MPI_COMM_WORLD, &gWorld);
546
547     // Now we want g1 to have only even ranks
548     // and g2 to have only odd ranks
549
550     newSize = size / 2;
551     if(size%2!=0) newSize++;
552
553     gRanks = new int[newSize];
554
555     // assign ranks to be in the different group
556     if(rank%2==0) {
557         for(i=0,pts=0,newSize=0;i<size;i+=2,pts++,newSize++) {
558             gRanks[pts] = i;
559         }
560     }
561     else {
562         for(i=1,pts=0,newSize=0;i<size;i+=2,pts++,newSize++) {
563             gRanks[pts] = i;
564         }
565     }
566
567     // Create new groups from gWorld.
568     MPI_Group_incl(gWorld, newSize, gRanks, &newGroup);
569
570     // Create group communicator for sub-group communication
571     MPI_Comm_create(MPI_COMM_WORLD,newGroup,&newComm);
572
573     MPI_Comm_rank(newComm, &newRank);
574     cout << "\nold=" << rank << ", new=" << newRank;
575
576     // Now we can do collective communication with new communicators
577     MPI_Bcast(&rank, 1, MPI_INT, 0, newComm);
578     cout << "\nbroadcasted rank=" << rank;
579
580     MPI_Comm_free(&newComm);
581     delete []gRanks;
582
583     // Finalize
584     MPI_Finalize();
585
586     return 0;
587 }
588
589
```

590

591 **18_Armstrong.cpp**

```
592 #include <iostream>
593 #include <climits>
594 #include <cstdlib>
595 using namespace std;
596 #include "mpi.h"
597
598 int isArmstrong1(unsigned long num){
599     return 1==1;
600 }
601
602 int isArmstrong(unsigned long num) {
603     int i, j, p, count, product;
604     unsigned long n, sum;
605     int digits[25]; // in fact, 20 at most...
606
607     count=0;
608     n = num;
609     while(n>=1) {
610         digits[count] = n % 10;
611         n /= 10;
612         count++;
613     }
614     p = count;
615
616     sum = 0;
617     for(j=0;j<count;j++) {
618         product = 1;
619         for(i=0;i<p;i++) product *= digits[j];
620         sum += product;
621     }
622
623     return num==sum;
624 }
625
626 int manager(int size, int count) {
627     unsigned long i, out[2], block=100, answer;
628     int no=0;
629     MPI_Status status;
630
631     // for distributing work
632     for(i=100;i<=ULONG_MAX-block;i) {
633         MPI_Recv(&answer, 1, MPI_LONG, MPI_ANY_SOURCE, 100, MPI_COMM_WORLD, &status);
634         if(answer!=0) {
635             cout << "\n" << answer;
636             no++;
637             if(no>=count) break;
638         }
639         else {
640             out[0] = i;
641             out[1] = block;
642             MPI_Send(out, 2, MPI_LONG, status.MPI_SOURCE, 101, MPI_COMM_WORLD);
643             i += block;
644         }
645     }
646     // for terminating workers
647     for(i=1;i<size;i) {
648         MPI_Recv(&answer, 1, MPI_LONG, MPI_ANY_SOURCE, 100, MPI_COMM_WORLD, &status);
649         if(answer!=0) {
650             // cout << "ans=" << answer << endl;
651             continue;
652         }
653         out[0] = 0;
654         out[1] = 0;
655         MPI_Send(out, 2, MPI_LONG, status.MPI_SOURCE, 101, MPI_COMM_WORLD);
656         i++;
657     }
658
659     return no;
660 }
661
662 void worker() {
663     unsigned long work[2], i;
664     unsigned long answer=0;
665     MPI_Status status;
666
667     for(;;) {
668         MPI_Send(&answer, 1, MPI_LONG, 0, 100, MPI_COMM_WORLD);
669         MPI_Recv(work, 2, MPI_LONG, 0, 101, MPI_COMM_WORLD, &status);
670         // cout << work[0] << ", " << work[1] << endl;
671         if(work[1]==0) break;
672         answer = 0;
673         for(i=work[0]; i<work[0]+work[1]; i++) {
```

```
674     if(isArmstrong(i)) {
675         MPI_Send(&i, 1, MPI_LONG, 0, 100, MPI_COMM_WORLD);
676     }
677 }
678 }
679 }
680
681 int main(int argc, char **argv) {
682     int rank, size;
683     int count=1;
684     double t1,t2;
685     // Initialization
686     MPI_Init(&argc, &argv);
687
688     // Get environment information
689     MPI_Comm_rank(MPI_COMM_WORLD, &rank);
690     MPI_Comm_size(MPI_COMM_WORLD, &size);
691
692     if(argc>1) {
693         count=atoi(argv[1]);
694     }
695     if(count<1) count=1;
696
697     t1=MPI_Wtime();
698     if(rank==0) {
699         count=manager(size, count);
700         t2=MPI_Wtime();
701         cout << "\nFound " << count << " Armstrong numbers";
702         cout << ", and took " << t2-t1 << " seconds.";
703     }
704     else {
705         worker();
706     }
707
708     // Finalize
709     MPI_Finalize();
710
711     return 0;
712 }
713
714
```