|  |  |  |
| --- | --- | --- |
| 1 | #include <stdio.h>  #include "mpi.h"  int main(int argc, char \*\*argv)  { printf("Before MPI\_INIT\n");  MPI\_Init(&argc, &argv);  printf("Parallel section\n");  MPI\_Finalize();  printf("After MPI\_FINALIZE\n");  return 0;  } | $ mpirun -np 1 ./ex1  Before MPI\_INIT  Parallel section |
| 2 | #include <stdio.h>  #include "mpi.h"  int main(int argc, char \*\*argv)  {  int rank, size;  MPI\_Init(&argc, &argv);  MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);  MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);  printf ("process %d, size %d\n", rank, size);  MPI\_Finalize();  return 0;  } | $ mpirun -np 1 ./ex2  process 0, size 1  $ mpirun -np 4 ./ex2  process 2, size 4  process 3, size 4  process 0, size 4  process 1, size 4 |
| 3 | #include <stdio.h>  #include "mpi.h"  #define MAX 100  int main(int argc, char \*\*argv)  { int rank, size, n, i, ibeg, iend;  MPI\_Init(&argc, &argv);  MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);  MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);  n=(MAX-1)/size+1;  ibeg=rank\*n+1;  iend=(rank+1)\*n;  for(i=ibeg; i<=((iend>MAX)?MAX:iend); i++)  printf ("process %d, %d^2=%d\n", rank, i, i\*i);  MPI\_Finalize();  return 0;  } | $ mpirun -np 2 ./ex3  process 0, 1^2=1  process 0, 2^2=4  process 0, 3^2=9  …  process 0, 48^2=2304  process 0, 49^2=2401  process 0, 50^2=2500  process 1, 51^2=2601  process 1, 52^2=2704  ...  process 1, 98^2=9604  process 1, 99^2=9801  process 1, 100^2=10000 |
| 4 | #include <stdio.h>  #include "mpi.h"  #define NTIMES 100  int main(int argc, char \*\*argv)  {  double time\_start, time\_finish, tick;  int rank, i;  int len;  char \*name;  name = (char\*)malloc(MPI\_MAX\_PROCESSOR\_NAME\*sizeof(char));  MPI\_Init(&argc, &argv);  MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);  MPI\_Get\_processor\_name(name, &len);  tick = MPI\_Wtick();  time\_start = MPI\_Wtime();  for (i = 0; i<NTIMES; i++)  time\_finish = MPI\_Wtime();  printf ("processor %s, process %d: tick= %lf, time= %lf\n", name, rank, tick, (time\_finish-time\_start)/NTIMES);  MPI\_Finalize();  return 0;  } | $ mpirun -np 4 ./ex4  processor andrew-VirtualBox, process 0: tick= 0.000000, time= 0.000000  processor andrew-VirtualBox, process 1: tick= 0.000000, time= 0.000000  processor andrew-VirtualBox, process 2: tick= 0.000000, time= 0.000000  processor andrew-VirtualBox, process 3: tick= 0.000000, time= 0.000000 |
| 5 | #include "mpi.h"  #include <stdio.h>  int main(int argc, char \*\*argv)  { int rank;  float a, b;  MPI\_Status status;  MPI\_Init(&argc, &argv);  MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);  a = 0.0;  b = 0.0;  if(rank == 0)  { b = 1.0;  MPI\_Send(&b, 1, MPI\_FLOAT, 1, 5, MPI\_COMM\_WORLD);  MPI\_Recv(&a, 1, MPI\_FLOAT, 1, 5, MPI\_COMM\_WORLD, &status);  }  if(rank == 1)  { a = 2.0;  MPI\_Recv(&b, 1, MPI\_FLOAT, 0, 5, MPI\_COMM\_WORLD, &status);  MPI\_Send(&a, 1, MPI\_FLOAT, 0, 5, MPI\_COMM\_WORLD);  }  printf("process %d a = %f, b = %f\n", rank, a, b);  MPI\_Finalize();  return 0;  } | $ mpirun -np 2 ./ex5  process 0 a = 2.000000, b = 1.000000  process 1 a = 2.000000, b = 1.000000 |
| 6 | #include "mpi.h"  #include <stdio.h>  int main(int argc, char \*\*argv)  { int size, rank, a, b;  MPI\_Status status;  MPI\_Init(&argc, &argv);  MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);  MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);  a = rank;  b = -1;  if((rank%2) == 0){  if(rank<size-1)  MPI\_Send(&a, 1, MPI\_INT, rank+1, 5, MPI\_COMM\_WORLD);  }  else  MPI\_Recv(&b, 1, MPI\_INT, rank-1, 5, MPI\_COMM\_WORLD, &status);  printf("process %d a = %d, b = %d\n", rank, a, b);  MPI\_Finalize();  return 0;  } | $ mpirun -np 2 ./ex6  process 0 a = 0, b = -1  process 1 a = 1, b = 0  $ mpirun -np 4 ./ex6  process 2 a = 2, b = -1  process 0 a = 0, b = -1  process 1 a = 1, b = 0  process 3 a = 3, b = 2 |
| 7 | #include <stdio.h>  #include "mpi.h"  int main(int argc, char \*\*argv)  {  int size, rank, next, prev, rbuf;  MPI\_Status status;  MPI\_Init(&argc, &argv);  MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);  MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);  next = rank+1;  if(next == size) next = MPI\_PROC\_NULL;  int msg = rank\*10;  MPI\_Send(&msg, 1, MPI\_INT, next, 5, MPI\_COMM\_WORLD);  prev = rank-1;  if(prev == -1) prev = MPI\_PROC\_NULL;  rbuf = -1;  MPI\_Recv(&rbuf, 1, MPI\_INT, prev, 5, MPI\_COMM\_WORLD, &status);  printf("process %d rbuf = %d\n", rank, rbuf);  MPI\_Finalize();  return 0;  } | $ mpirun -np 4 ./ex7  process 0 rbuf = -1  process 1 rbuf = 0  process 3 rbuf = 20  process 2 rbuf = 10 |
| 8 | #include "mpi.h"  #include <stdio.h>  int main(int argc, char \*\*argv)  {  int BUFSIZE = sizeof(int) + MPI\_BSEND\_OVERHEAD;  char \*buf;  int rank, ibufsize, rbuf, size;  MPI\_Status status;  MPI\_Init(&argc, &argv);  MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);  MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);  if(rank == 0){  MPI\_Buffer\_attach(malloc(BUFSIZE), BUFSIZE);  MPI\_Bsend(&rank, 1, MPI\_INT, 1, 5, MPI\_COMM\_WORLD);  MPI\_Buffer\_detach(&buf, &ibufsize);  free(buf);  }  if(rank == 1){  MPI\_Recv(&rbuf, 1, MPI\_INT, 0, 5, MPI\_COMM\_WORLD, &status);  printf("Process 1 received %d from process %d\n", rbuf, status.MPI\_SOURCE);  }  MPI\_Finalize();  return 0;  } | $ mpirun -np 2 ./ex8  Process 1 received 0 from process 0 |
| 9 | #include <stdio.h>  #include "mpi.h"  int main(int argc, char \*\*argv)  { int rank, size, ibuf;  MPI\_Status status;  float rbuf;  MPI\_Init(&argc, &argv);  MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);  MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);  ibuf = rank;  rbuf = 1.0 \* rank;  if(rank==1) MPI\_Send(&ibuf, 1, MPI\_INT, 0, 5, MPI\_COMM\_WORLD);  if(rank==2) MPI\_Send(&rbuf, 1, MPI\_FLOAT, 0, 5, MPI\_COMM\_WORLD);  if(rank==0){  MPI\_Probe(MPI\_ANY\_SOURCE, 5, MPI\_COMM\_WORLD, &status);  if(status.MPI\_SOURCE == 1){  MPI\_Recv(&ibuf, 1, MPI\_INT, 1, 5, MPI\_COMM\_WORLD, &status);  MPI\_Recv(&rbuf, 1, MPI\_FLOAT, 2, 5, MPI\_COMM\_WORLD, &status);  }  else if(status.MPI\_SOURCE == 2){  MPI\_Recv(&rbuf, 1, MPI\_FLOAT, 2, 5, MPI\_COMM\_WORLD, &status);  MPI\_Recv(&ibuf, 1, MPI\_INT, 1, 5, MPI\_COMM\_WORLD, &status);  }  printf("Process 0 recv %d from process 1, %f from process 2\n", ibuf, rbuf);  }  MPI\_Finalize();  return 0;  } |  |
| 10 | #include <stdio.h>  #include "mpi.h"  #define NMAX 1000000  #define NTIMES 10  int main(int argc, char \*\*argv)  {  int rank, size, i, n, lmax;  double time\_start, time, bandwidth, max, a[NMAX];  MPI\_Status status;  MPI\_Init(&argc, &argv);  MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);  MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);  time\_start = MPI\_Wtime();  n = 0;  max = 0.0;  lmax = 0;  while(n<=NMAX){  time\_start = MPI\_Wtime();  for(i = 0; i<NTIMES; i++){  if(rank==0){  MPI\_Send(a, n, MPI\_DOUBLE, 1, 1, MPI\_COMM\_WORLD);  MPI\_Recv(a, n, MPI\_DOUBLE, 1, 1, MPI\_COMM\_WORLD, &status);  }  if(rank==1){  MPI\_Recv(a, n, MPI\_DOUBLE, 0, 1, MPI\_COMM\_WORLD, &status);  MPI\_Send(a, n, MPI\_DOUBLE, 0, 1, MPI\_COMM\_WORLD);  }  }  time = (MPI\_Wtime()-time\_start)/(2\*NTIMES);  bandwidth = (sizeof(double)\*n\*1.0/(1024\*1024\*time));  if(max<bandwidth){  max = bandwidth;  lmax = sizeof(double)\*n;  }  if(rank==0)  if(n==0) printf("latency = %lf seconds\n", time);  else printf("%d bytes, bandwidth = %lf Mb/s\n", (int)sizeof(double)\*n, bandwidth);  if(n==0) n = 1;  else n = 2\*n;  }  if(rank==0) printf("max bandwidth = %lf Mb/s length = %d bytes\n", max, lmax);  MPI\_Finalize();  return 0;} |  |
| 11 | #include <stdio.h>  #include "mpi.h"  int main(int argc, char \*\*argv)  { int rank, size, prev, next;  int buf[2];  MPI\_Request reqs[4];  MPI\_Status stats[4];  MPI\_Init(&argc,&argv);  MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);  MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);  prev = rank - 1;  next = rank + 1;  if(rank==0) prev = size - 1;  if(rank==size - 1) next = 0;  MPI\_Irecv(&buf[0], 1, MPI\_INT, prev, 5, MPI\_COMM\_WORLD, &reqs[0]);  MPI\_Irecv(&buf[1], 1, MPI\_INT, next, 6, MPI\_COMM\_WORLD, &reqs[1]);  MPI\_Isend(&rank, 1, MPI\_INT, prev, 6, MPI\_COMM\_WORLD, &reqs[2]);  MPI\_Isend(&rank, 1, MPI\_INT, next, 5, MPI\_COMM\_WORLD, &reqs[3]);  MPI\_Waitall(4, reqs, stats);  printf("process %d prev = %d next=%d\n", rank, buf[0], buf[1]);  MPI\_Finalize();  return 0;  } |  |
| 12 | #include <stdio.h>  #include "mpi.h"  #define N 1000  #define MAXPROC 128  void slave(double \*a, int n, int rank, int size)  {  int i, ibeg, iend, h;  h = n / size;  ibeg = h\*rank;  iend = (h+1)\*rank;  for (i = ibeg; i<iend;i++)  {  a[i] = rank;  }  /\* обработка локальной части массива a \*/  }  void master(double \*a, int n)  {  int i;  for (i = 0; i<n;i++)  {  printf ("%lf ",a[i]);  }    /\* обработка массива a \*/  }  int main(int argc, char \*\*argv)  { int rank, size, num, k, i, indices[MAXPROC], source;  MPI\_Request req[MAXPROC];  MPI\_Status statuses[MAXPROC];  double a[N][MAXPROC];  MPI\_Init(&argc, &argv);  MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);  MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);  if(rank!=0)  {  slave((double\*)a, N, rank, size);  MPI\_Send(a, N, MPI\_DOUBLE, 0, 5, MPI\_COMM\_WORLD);  }  else  { for(i = 0; i<size-1; i++)  MPI\_Irecv(&a[0][i], N, MPI\_DOUBLE, i, 5, MPI\_COMM\_WORLD, &req[i]);    MPI\_Waitsome(size-1, req, &num, indices, statuses);  for (i = 0; i< num; i++){  source = statuses[i].MPI\_SOURCE;  master(&a[0][source], N);  MPI\_Irecv(&a[0][source], N, MPI\_DOUBLE, source, 5, MPI\_COMM\_WORLD, &req[source]);    }  }  MPI\_Finalize();  return 0;  } |  |
| 13 | #include <stdio.h>  #include "mpi.h"  #define N 9  #define MAXPROC 64  int main(int argc, char \*\*argv)  { int rank, size, nl, i, j, ii, jj, ir, irr;  double c;  MPI\_Status status;  MPI\_Request req[MAXPROC\*2];  MPI\_Init(&argc, &argv);  MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);  MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);  nl = (N-1)/size+1;  double \*a = (double \*) calloc(size\*nl\*nl, sizeof(double));  double \*b = (double \*) calloc(size\*nl\*nl, sizeof(double));  for(i = 0; i<nl; i++){  ii = i+rank\*nl;  if(ii<N)  for(j = 0; j<N; j++)  \*(a+j\*nl+i)=100\*ii+j;  }  for(ir = 0; ir<size; ir++){  MPI\_Irecv(b+ir\*nl\*nl, nl\*nl, MPI\_DOUBLE, ir, 1, MPI\_COMM\_WORLD, &req[ir]);  MPI\_Isend(a+ir\*nl\*nl, nl\*nl, MPI\_DOUBLE, ir, 1, MPI\_COMM\_WORLD, &req[ir+size]);  }  for(irr = 0; irr<size; irr++){  MPI\_Waitany(size, req, &ir, &status);  ir = status.MPI\_SOURCE;  for(i = 0; i<nl; i++){  for(j = i+1; j<nl; j++){  c = \*(b+i\*nl+j+ir\*nl\*nl);  \*(b+i\*nl+j+ir\*nl\*nl) = \*(b+j\*nl+i+ir\*nl\*nl);  \*(b+j\*nl+i+ir\*nl\*nl) = c;  }  }  }  for(i = 0; i<nl; i++){  ii = i+rank\*nl;  if(ii<N)  for(j = 0; j<N; j++)  printf("process %d : a[%d][%d] = %lf b[%d][%d] = %lf\n", rank, ii, j, \*(a+j\*nl+i), ii, j, \*(b+j\*nl+i));  }  free(a);  free(b);  MPI\_Finalize();  return 0;  } | $ mpirun -np 2 ./ex13  process 0 : a[0][0] = 0.000000 b[0][0] = 0.000000  process 0 : a[0][1] = 1.000000 b[0][1] = 100.000000  process 0 : a[0][2] = 2.000000 b[0][2] = 200.000000  …  process 0 : a[3][6] = 306.000000 b[3][6] = 603.000000  process 0 : a[3][7] = 307.000000 b[3][7] = 703.000000  process 0 : a[3][8] = 308.000000 b[3][8] = 803.000000  process 0 : a[4][0] = 400.000000 b[4][0] = 4.000000  process 0 : a[4][1] = 401.000000 b[4][1] = 104.000000  process 0 : a[4][2] = 402.000000 b[4][2] = 204.000000  …  process 1 : a[5][0] = 500.000000 b[5][0] = 5.000000  process 1 : a[5][1] = 501.000000 b[5][1] = 105.000000  …  process 1 : a[8][6] = 806.000000 b[8][6] = 608.000000  process 1 : a[8][7] = 807.000000 b[8][7] = 708.000000  process 1 : a[8][8] = 808.000000 b[8][8] = 808.000000 |
| 14 | #include <stdio.h>  #include "mpi.h"  int main(int argc, char \*\*argv)  {  int size, rank, prev, next;  int reqs[4];  int rbuf[2], sbuf[2];  MPI\_Init(&argc, &argv);  MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);  MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);  prev = rank - 1;  next = rank + 1;  if(rank == 0) prev = size - 1;  if(rank == size - 1) next = 0;  MPI\_Recv\_init(rbuf[0], 1, MPI\_FLOAT, prev, 5, MPI\_COMM\_WORLD, reqs[0]);  MPI\_Recv\_init(rbuf[1], 1, MPI\_FLOAT, next, 6, MPI\_COMM\_WORLD, reqs[1]);  MPI\_Send\_init(sbuf[0], 1, MPI\_FLOAT, prev, 6, MPI\_COMM\_WORLD, reqs[2]);  MPI\_Send\_init(sbuf[1], 1, MPI\_FLOAT, next, 5, MPI\_COMM\_WORLD, reqs[3]);  for(i=...){  sbuf[0] =...;  sbuf[1] =...;  MPI\_Startall(4, reqs);  ...  MPI\_Waitall(4, reqs, stats);  ...  }  MPI\_Request\_free(reqs[0]);  MPI\_Request\_free(reqs[1]);  MPI\_Request\_free(reqs[2]);  MPI\_Request\_free(reqs[3]);  MPI\_Finalize();  } |  |
| 15 | #include <stdio.h>  #include "mpi.h"  int main(int argc, char \*\*argv)  {  int rank, size, prev, next;  int buf[2];  MPI\_Status status1;  MPI\_Status status2;  MPI\_Init(&argc, &argv);  MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);  MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);  prev = rank - 1;  next = rank + 1;  if(rank==0) prev = size - 1;  if(rank==size-1) next = 0;  MPI\_Sendrecv(&rank, 1, MPI\_INT, prev, 6, &buf[1], 1, MPI\_INT, next, 6, MPI\_COMM\_WORLD, &status2);  MPI\_Sendrecv(&rank, 1, MPI\_INT, next, 5, &buf[0], 1, MPI\_INT, prev, 5, MPI\_COMM\_WORLD, &status1);  printf( "process %d prev=%d next=%d\n", rank, buf[0], buf[1]);  MPI\_Finalize();  } | $ mpirun -np 4 ./example15a  process 3 prev=2 next=0  process 0 prev=3 next=1  process 1 prev=0 next=2  process 2 prev=1 next=3 |
| 16 | #include <stdio.h>  #include "mpi.h"  #define MAXPROC 128  #define NTIMES 10000  int main(int argc, char \*\*argv)  {  int rank, size, i, it;  int ibuf[MAXPROC];  double time\_start, time\_finish;  MPI\_Request req[2\*MAXPROC];  MPI\_Status statuses[MAXPROC];  MPI\_Init(&argc, &argv);  MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);  MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);  if(rank==0){  for(i = 1; i<size; i++){  MPI\_Recv\_init(&ibuf[i], 0, MPI\_INT, i, 5, MPI\_COMM\_WORLD, &req[i]);  MPI\_Send\_init(&rank, 0, MPI\_INT, i, 6, MPI\_COMM\_WORLD, &req[size+i]);  }  time\_start = MPI\_Wtime();  for(it = 0; it<NTIMES; it++){  MPI\_Startall(size-1, &req[1]);  MPI\_Waitall(size-1, &req[1], statuses);  MPI\_Startall(size-1, &req[size+1]);  MPI\_Waitall(size-1, &req[size+1], statuses);  }  }  else{  MPI\_Recv\_init(&ibuf[0], 0, MPI\_INT, 0, 6, MPI\_COMM\_WORLD, &req[0]);  MPI\_Send\_init(&rank, 0, MPI\_INT, 0, 5, MPI\_COMM\_WORLD, &req[1]);  time\_start = MPI\_Wtime();  for(it = 0; it<NTIMES; it++){  MPI\_Start(&req[1]);  MPI\_Wait(&req[1], statuses);  MPI\_Start(&req[0]);  MPI\_Wait(&req[0], statuses);  }  }  time\_finish = MPI\_Wtime()-time\_start;  printf("rank = %d model time = %lf\n", rank, time\_finish/NTIMES);  time\_start = MPI\_Wtime();  for(it = 0; it<NTIMES; it++) MPI\_Barrier(MPI\_COMM\_WORLD);  time\_finish = MPI\_Wtime()-time\_start;  printf("rank = %d barrier time = %lf\n", rank, time\_finish/NTIMES);  MPI\_Finalize();  } | mpirun -np 4 ./example16a  rank = 0 model time = 0.026705  rank = 1 model time = 0.026706  rank = 3 model time = 0.026706  rank = 2 model time = 0.026706  rank = 0 barrier time = 0.035401  rank = 1 barrier time = 0.035401  rank = 2 barrier time = 0.035400  rank = 3 barrier time = 0.035403 |
| 17 | #include <stdio.h>  #include "mpi.h"  #define n 1000  int main(int argc, char \*\*argv)  {  int rank, i, size, nproc;  double time\_start, time\_finish;  double a[n], b[n], c[n];  MPI\_Status status;  MPI\_Init(&argc, &argv);  MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);  MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);  nproc = size;  for(i = 0; i<n; i++) a[i] = 1.0/size;  MPI\_Barrier(MPI\_COMM\_WORLD);  time\_start = MPI\_Wtime();  for(i = 0; i<n; i++) c[i] = a[i];  while(nproc>1){  if(rank<nproc/2){  MPI\_Recv(b, n, MPI\_DOUBLE, nproc-rank-1, 1, MPI\_COMM\_WORLD, &status);  for(i = 0; i<n; i++) c[i] = c[i] + b[i];  }  else if(rank<nproc)  MPI\_Send(c, n, MPI\_DOUBLE, nproc-rank-1, 1, MPI\_COMM\_WORLD);  nproc = nproc/2;  }  for(i = 0; i<n; i++) b[i] = c[i];  time\_finish = MPI\_Wtime()-time\_start;  if(rank==0) printf("model b[1]=%lf\n", b[1]);  printf ("rank=%d model time=%lf\n", rank, time\_finish);  for(i = 0; i<n; i++) a[i] = 1.0/size;  MPI\_Barrier(MPI\_COMM\_WORLD);  time\_start = MPI\_Wtime();  MPI\_Reduce(a, b, n, MPI\_DOUBLE, MPI\_SUM, 0, MPI\_COMM\_WORLD);  time\_finish = MPI\_Wtime()-time\_start;  if(rank==0) printf("reduce b[1]=%lf\n", b[1]);  printf("rank=%d reduce time =%lf\n", rank, time\_finish);  MPI\_Finalize();  } |  |
| 18 | #include <stdio.h>  #include "mpi.h"  #define n 1000  void smod5(void \*in, void \*inout, int \*l, MPI\_Datatype \*type){  int i;  for(i = 0; i<\*l; i++) ((int\*)inout)[i] = (((int\*)in)[i] + ((int\*)inout)[i])%5;  }  int main(int argc, char \*\*argv)  {  int rank, size, i;  int a[n];  int b[n];  MPI\_Op op;  MPI\_Init(&argc, &argv);  MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);  MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);  for(i = 0; i<n; i++) a[i] = i + rank +1;  printf("process %d a[0] = %d\n", rank, a[0]);  MPI\_Op\_create(&smod5, 1, &op);  MPI\_Reduce(a, b, n, MPI\_INT, op, 0, MPI\_COMM\_WORLD);  MPI\_Op\_free(&op);  if(rank==0) printf("b[0] = %d\n", b[0]);  MPI\_Finalize();  } |  |
| 19 | #include <stdio.h>  #include "mpi.h"  int main(int argc, char \*\*argv)  {  int rank, i, size, size1, n;  MPI\_Status status;  MPI\_Group group, group1, group2;  int ranks[128], rank1, rank2, rank3;  MPI\_Init(&argc, &argv);  MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);  MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);  MPI\_Comm\_group(MPI\_COMM\_WORLD, &group);  size1 = size/2;  for(i = 0; i<size1; i++) ranks[i] = i;  MPI\_Group\_incl(group, size1, ranks, &group1);  MPI\_Group\_excl(group, size1, ranks, &group2);  MPI\_Group\_rank(group1, &rank1);  MPI\_Group\_rank(group2, &rank2);  if(rank1==MPI\_UNDEFINED){  if(!(size%2) || rank!=size-1) MPI\_Group\_translate\_ranks(group1, 1, &rank2, group, &rank3);  else rank3 = MPI\_PROC\_NULL;  }  else MPI\_Group\_translate\_ranks(group2, 1, &rank1, group, &rank3);  MPI\_Sendrecv(&rank, 1, MPI\_INT, rank3, 1, &n, 1, MPI\_INT, rank3, 1, MPI\_COMM\_WORLD, &status);  MPI\_Group\_free(&group);  MPI\_Group\_free(&group1);  MPI\_Group\_free(&group2);  printf("process %d n=%d\n", rank, n);  MPI\_Finalize();  } |  |
| 20 | #include <stdio.h>  #include "mpi.h"  int main(int argc, char \*\*argv)  {  int rank, size, i, rbuf;  MPI\_Group group, new\_group;  MPI\_Comm new\_comm;  int ranks[128], new\_rank;  MPI\_Init(&argc, &argv);  MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);  MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);  MPI\_Comm\_group(MPI\_COMM\_WORLD, &group);  for(i=0; i<size/2; i++) ranks[i] = i;  if(rank < size/2) MPI\_Group\_incl(group, size/2, ranks, &new\_group);  else MPI\_Group\_excl(group, size/2, ranks, &new\_group);  MPI\_Comm\_create(MPI\_COMM\_WORLD, new\_group, &new\_comm);  MPI\_Allreduce(&rank, &rbuf, 1, MPI\_INT, MPI\_SUM, new\_comm);  MPI\_Group\_rank(new\_group, &new\_rank);  printf("rank=%d newrank=%d rbuf=%d\n", rank, new\_rank, rbuf);  MPI\_Finalize();  } |  |
| 21 | #include <stdio.h>  #include "mpi.h"  int main(int argc, char \*\*argv)  {  int rank, size, rank1;  MPI\_Comm comm\_revs;  MPI\_Init(&argc, &argv);  MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);  MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);  MPI\_Comm\_split(MPI\_COMM\_WORLD, 1, size-rank,&comm\_revs);  MPI\_Comm\_rank(comm\_revs, &rank1);  printf("rank = %d rank1 = %d\n", rank, rank1);  MPI\_Comm\_free(&comm\_revs);  MPI\_Finalize();  } |  |
| 22 | #include <stdio.h>  #include "mpi.h"  void master(MPI\_Comm comm)  {  int size, i, r;  MPI\_Status status;  MPI\_Comm\_remote\_size(comm, &size);  for(i=0; i<size; i++){  MPI\_Send(&i, 1, MPI\_INT, i, 5, comm);  }  MPI\_Recv(&r, 1, MPI\_INT, 0, 6, comm, &status);  printf("sum = %d\n", r);  }  void slave(MPI\_Comm comm, MPI\_Comm intercomm)  {  int s, r, rank;  MPI\_Status status;  MPI\_Comm\_rank(comm, &rank);  MPI\_Recv(&s, 1, MPI\_INT, 0, 5, intercomm, &status);  MPI\_Reduce(&s, &r, 1, MPI\_INT, MPI\_SUM, 0, comm);  if(rank==0) MPI\_Send(&r, 1, MPI\_INT, 0, 6, intercomm);  }  int main(int argc, char \*\*argv)  {  int rank, size, color, i, rank1;  MPI\_Comm newcomm, newintercomm;  MPI\_Init(&argc, &argv);  MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);  MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);  color=(rank==0);  MPI\_Comm\_split(MPI\_COMM\_WORLD, color, rank, &newcomm);  MPI\_Intercomm\_create(newcomm, 0, MPI\_COMM\_WORLD, color, 1, &newintercomm);  if(rank==0) master(newintercomm);  else slave(newcomm, newintercomm);  MPI\_Comm\_free(&newintercomm);  MPI\_Comm\_free(&newcomm);  MPI\_Finalize();  } |  |
| 23 | #include <stdio.h>  #include "mpi.h"  #define MAXPROC 128  #define MAXEDGES 512  int main(int argc, char \*\*argv)  {  int rank, rank1, i, size;  int a, b;  MPI\_Status status;  MPI\_Comm comm\_graph;  int index[MAXPROC], edges[MAXEDGES];  int num, neighbors[MAXPROC];  MPI\_Init(&argc, &argv);  MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);  MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);  for(i = 0; i<size; i++) index[i] = size+i-1;  for(i = 0; i<size-1; i++){  edges[i] = i+1;  edges[size+i-1] = 0;  }  MPI\_Graph\_create(MPI\_COMM\_WORLD, size, index, edges, 1, &comm\_graph);  MPI\_Graph\_neighbors\_count(comm\_graph, rank, &num);  MPI\_Graph\_neighbors(comm\_graph, rank, num, neighbors);  for(i = 0; i<num; i++){  MPI\_Sendrecv(&rank, 1, MPI\_INT, neighbors[i], 1, &rank1, 1, MPI\_INT, neighbors[i], 1, comm\_graph, &status);  printf("process %d communicate with process %d\n", rank, rank1);  }  MPI\_Finalize();  } |  |
| 24 | #include <stdio.h>  #include "mpi.h"  #define N 8  void work(int n, int nl, int size, int rank)  {  int ii, matr\_rev;  double a[n][nl], b[n][nl];  int i, j;  MPI\_Status status;  for(j = 0; j<nl; j++)  for(i = 0; i<n; i++){  b[i][j] = 0.0;  ii = j+rank\*nl;  a[i][j] = 100\*ii+i;  }  MPI\_Type\_vector(nl, n, -n, MPI\_DOUBLE, &matr\_rev);  MPI\_Type\_commit(&matr\_rev);  MPI\_Sendrecv(&a[0][nl-1], 1, matr\_rev, size-rank-1, 1, b, nl\*n, MPI\_DOUBLE, size-rank-1, 1, MPI\_COMM\_WORLD, &status);  for(j = 0; j<nl; j++)  for(i = 0; i<n; i++)  printf("process %d : %d %d %f %f\n", rank, j+rank\*nl, i, a[i][j], b[i][j]);  }  int main(int argc, char \*\*argv)  {  int rank, size,nl;  MPI\_Init(&argc, &argv);  MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);  MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);  nl = (N-1)/size+1;  work(N, nl, size, rank);  MPI\_Finalize();  } |  |
| 25 | #include <stdio.h>  #include "mpi.h"  int main(int argc, char \*\*argv)  {  int size, rank, position, i;  float a[10];  char b[10], buf[100];  MPI\_Init(&argc, &argv);  MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);  MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);  for(i = 0; i<10; i++){  a[i] = rank + 1.0;  if(rank==0) b[i]='a';  else b[i] = 'b';  }  position=0;  if(rank==0){  MPI\_Pack(a, 10, MPI\_FLOAT, buf, 100, &position, MPI\_COMM\_WORLD);  MPI\_Pack(b, 10, MPI\_CHAR, buf, 100, &position, MPI\_COMM\_WORLD);  MPI\_Bcast(buf, 100, MPI\_PACKED, 0, MPI\_COMM\_WORLD);  }  else{  MPI\_Bcast(buf, 100, MPI\_PACKED, 0, MPI\_COMM\_WORLD);  position=0;  MPI\_Unpack(buf, 100, &position, a, 10, MPI\_FLOAT, MPI\_COMM\_WORLD);  MPI\_Unpack(buf, 100, &position, b, 10, MPI\_CHAR, MPI\_COMM\_WORLD);  }  for(i = 0; i<10; i++) printf("process %d a=%f b=%c\n", rank, a[i], b[i]);  MPI\_Finalize();  } |  |
| 26 | Master:  #include <stdio.h>  #include "mpi.h"  int main(int argc, char \*\*argv)  {  int size, rank, rank1, rank2;  MPI\_Status status;  MPI\_Comm intercomm;  char slave[10]="./slave";  MPI\_Init(&argc, &argv);  MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);  MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);  MPI\_Comm\_spawn(slave, MPI\_ARGV\_NULL, 2, MPI\_INFO\_NULL, 0, MPI\_COMM\_SELF, &intercomm, MPI\_ERRCODES\_IGNORE);  MPI\_Recv(&rank1, 1, MPI\_INT, 0, 0, intercomm, &status);  MPI\_Recv(&rank2, 1, MPI\_INT, 1, 1, intercomm, &status);  printf("Slaves %d and %d are working for %d\n", rank1, rank2, rank);  MPI\_Finalize();  return 0;  }  Slave:  #include <stdio.h>  #include "mpi.h"  int main(int argc, char \*\*argv)  {  int rank, size;  MPI\_Comm intercomm;  MPI\_Init(&argc, &argv);  MPI\_Comm\_get\_parent(&intercomm);  MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);  MPI\_Send(&rank, 1, MPI\_INT, 0, rank, intercomm);  MPI\_Finalize();  return 0;  } | $ mpirun -np 4 ./master  Slaves 0 and 1 are working for 0  Slaves 0 and 1 are working for 1  Slaves 0 and 1 are working for 3  Slaves 0 and 1 are working for 2 |
| 27 | Client:  #include <stdio.h>  #include "mpi.h"  int main(int argc, char \*\*argv)  {  int rank;  MPI\_Init(&argc, &argv);  char port\_name[MPI\_MAX\_PORT\_NAME];  MPI\_Comm intercomm;  MPI\_Lookup\_name("example", MPI\_INFO\_NULL, port\_name);  MPI\_Comm\_connect(port\_name, MPI\_INFO\_NULL, 0, MPI\_COMM\_SELF, &intercomm);  MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);  MPI\_Send(&rank, 1, MPI\_INT, 0, 0, intercomm);  MPI\_Finalize();  return 0;  }  Server:  #include <stdio.h>  #include "mpi.h"  int main(int argc, char \*\*argv)  {  int r;  char port\_name[MPI\_MAX\_PORT\_NAME];  MPI\_Init(&argc, &argv);  MPI\_Status status;  MPI\_Comm intercomm;  MPI\_Open\_port(MPI\_INFO\_NULL, port\_name);  MPI\_Publish\_name("example", MPI\_INFO\_NULL, port\_name);  MPI\_Comm\_accept(port\_name, MPI\_INFO\_NULL, 0, MPI\_COMM\_SELF, &intercomm);  MPI\_Recv(&r, 1, MPI\_INT, 0, 0, intercomm, &status);  MPI\_Unpublish\_name("example", MPI\_INFO\_NULL, port\_name);  MPI\_Close\_port(port\_name);  printf("Client is sent %d\n", r);  MPI\_Finalize();  return 0;  } | $ mpirun -np 1 ./server &  [1] 5244  $ mpirun -np 1 ./client  Client is sent 0 |
| 28 | #include <stdio.h>  #include "mpi.h"  int main(int argc, char \*\*argv)  {  int rank, size, prev, next;  int buf[2];  MPI\_Aint lb, extent;  MPI\_Win win;  MPI\_Init(&argc, &argv);  MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);  MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);  prev = rank - 1;  next = rank + 1;  if(rank==0) prev = size - 1;  if(rank==size - 1) next = 0;  MPI\_Type\_get\_extent(MPI\_INT, &lb, &extent);  MPI\_Win\_create(buf, 2\*extent, extent, MPI\_INFO\_NULL, MPI\_COMM\_WORLD, &win);  MPI\_Win\_fence(0, win);  MPI\_Put(&rank, 1, MPI\_INT, prev, 1, 1, MPI\_INT, win);  MPI\_Put(&rank, 1, MPI\_INT, next, 0, 1, MPI\_INT, win);  MPI\_Win\_fence(0, win);  MPI\_Win\_free(&win);  printf("process %d prev = %d next=%d\n", rank, buf[0], buf[1]);  MPI\_Finalize();  } | $ mpirun -np 8 ./example28a  process 0 prev = 7 next=1  process 1 prev = 0 next=2  process 4 prev = 3 next=5  process 5 prev = 4 next=6  process 3 prev = 2 next=4  process 7 prev = 6 next=0  process 6 prev = 5 next=7  process 2 prev = 1 next=3 |
| 29 | #include <stdio.h>  #include "mpi.h"  int main(int argc, char \*\*argv)  {  int rank, size, prev, next, ranks[2];  int buf[2];  MPI\_Aint lb, extent;  MPI\_Win win;  MPI\_Group group, commgroup;  MPI\_Init(&argc, &argv);  MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);  MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);  prev = rank - 1;  next = rank + 1;  if(rank==0) prev = size - 1;  if(rank==size - 1) next = 0;  MPI\_Type\_get\_extent(MPI\_INT, &lb, &extent);  MPI\_Win\_create(buf, 2\*extent, extent, MPI\_INFO\_NULL, MPI\_COMM\_WORLD, &win);  MPI\_Comm\_group(MPI\_COMM\_WORLD, &group);  ranks[0]=prev; ranks[1]=next;  MPI\_Group\_incl(group, 2, ranks, &commgroup);  MPI\_Win\_post(commgroup, 0, win);  MPI\_Win\_start(commgroup, 0, win);  MPI\_Put(&rank, 1, MPI\_INT, prev, 1, 1, MPI\_INT, win);  MPI\_Put(&rank, 1, MPI\_INT, next, 0, 1, MPI\_INT, win);  MPI\_Win\_complete(win);  MPI\_Win\_wait(win);  MPI\_Win\_free(&win);  MPI\_Group\_free(&group);  MPI\_Group\_free(&commgroup);  printf("process %d prev = %d next=%d\n", rank, buf[0], buf[1]);  MPI\_Finalize();  } | $ mpirun -np 8 ./example29a  process 6 prev = 5 next=7  process 5 prev = 4 next=6  process 4 prev = 3 next=5  process 7 prev = 6 next=0  process 0 prev = 7 next=1  process 1 prev = 0 next=2  process 2 prev = 1 next=3  process 3 prev = 2 next=4 |
| 30 | #include <stdio.h>  #include "mpi.h"  int main(int argc, char \*\*argv)  {  int rank, size, prev, next;  int buf[2];  MPI\_Aint lb, extent;  MPI\_Win win;  MPI\_Init(&argc, &argv);  MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);  MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);  prev = rank - 1;  next = rank + 1;  if(rank==0) prev = size - 1;  if(rank==size - 1) next = 0;  MPI\_Type\_get\_extent(MPI\_INT, &lb, &extent);  MPI\_Win\_create(buf, 2\*extent, extent, MPI\_INFO\_NULL, MPI\_COMM\_WORLD, &win);  MPI\_Win\_lock(MPI\_LOCK\_EXCLUSIVE, prev, 0, win);  MPI\_Put(&rank, 1, MPI\_INT, prev, 1, 1, MPI\_INT, win);  MPI\_Win\_unlock(prev, win);  MPI\_Win\_lock(MPI\_LOCK\_EXCLUSIVE, next, 0, win);  MPI\_Put(&rank, 1, MPI\_INT, next, 0, 1, MPI\_INT, win);  MPI\_Win\_unlock(next, win);  MPI\_Win\_free(&win);  printf("process %d prev = %d next=%d\n", rank, buf[0], buf[1]);  MPI\_Finalize();  } | $ mpirun -np 8 ./example30a  process 0 prev = 7 next=1  process 1 prev = 0 next=2  process 3 prev = 2 next=4  process 2 prev = 1 next=3  process 7 prev = 6 next=0  process 6 prev = 5 next=7  process 4 prev = 3 next=5  process 5 prev = 4 next=6 |
| 31 | #include <stdio.h>  #include "mpi.h"  #define BUFSIZE 100  int main(int argc, char \*\*argv)  {  int bufsize, num, sum;  MPI\_Status status;  MPI\_File fh;  char buf[BUFSIZE];  MPI\_Init(&argc, &argv);  MPI\_File\_open(MPI\_COMM\_WORLD, "file.txt", MPI\_MODE\_RDONLY, MPI\_INFO\_NULL, &fh);  MPI\_File\_set\_view(fh, 0, MPI\_CHAR, MPI\_CHAR, "native", MPI\_INFO\_NULL);  sum = 0;  do{  MPI\_File\_read(fh, buf, BUFSIZE, MPI\_CHAR, &status);  MPI\_Get\_count(&status, MPI\_CHAR, &num);  printf("buf=%s\n", buf);  sum += num;  } while(num >= BUFSIZE);  MPI\_File\_close(&fh);  printf("%d characters are readed\n", sum);  MPI\_Finalize();  } | $ mpirun -np 8 ./example31a  buf=This is some text in file.txt!  31 characters are readed  buf=This is some text in file.txt!  buf=This is some text in file.txt!  31 characters are readed  buf=This is some text in file.txt!  31 characters are readed  buf=This is some text in file.txt!  31 characters are readed  buf=This is some text in file.txt!  31 characters are readed  31 characters are readed  buf=This is some text in file.txt!  31 characters are readed  buf=This is some text in file.txt!  31 characters are readed |
| 32 | #include <stdio.h>  #include "mpi.h"  int main(int argc, char \*\*argv)  {  int rank;  MPI\_File fh;  char buf[10];  MPI\_Init(&argc, &argv);  MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);  MPI\_File\_open(MPI\_COMM\_WORLD, "file1.txt", MPI\_MODE\_RDONLY, MPI\_INFO\_NULL, &fh);  MPI\_File\_set\_view(fh, rank\*10, MPI\_CHAR, MPI\_CHAR, "native", MPI\_INFO\_NULL);  MPI\_File\_read\_all(fh, buf, 10, MPI\_CHAR, MPI\_STATUS\_IGNORE);  printf("process %d, buf=%s\n", rank, buf);  MPI\_File\_close(&fh);  MPI\_Finalize();  } | $ mpirun -np 1 ./example32a  process 0, buf=This is an@  $ mpirun -np 2 ./example32a  process 0, buf=This is an@  process 1, buf=other text@  $ mpirun -np 4 ./example32a  process 0, buf=This is an@  process 1, buf=other text@  process 3, buf=txt!  process 2, buf= in file1.@  $ mpirun -np 5 ./example32a  process 1, buf=other text@  process 2, buf= in file1.@  process 4, buf=@  @  process 0, buf=This is an@  process 3, buf=txt! |