Simple Parallel Data Structures-4 Solutions continue
by
William Gropp and Ewing Lusk

Exercise: Using nonblocking operations

```
#include <stdio.h>
#include "mpi.h"

/* This example handles a 12 x 12 mesh, on 4 processors only. */
/* This using nonblocking operations */
#define maxn 12

int main( argc, argv )
int argc;
char **argv;
{
   int rank, value, size, errcnt, toterr, i, j;
   MPI_Request r[4];
   int nreq;
   MPI_Status statuses[4];
   double x[12][12];
   double xlocal[(12/4)+2][12];

   MPI_Init( &argc, &argv );

   MPI_Comm_rank( MPI_COMM_WORLD, &rank );
   MPI_Comm_size( MPI_COMM_WORLD, &size );

   if (size != 4) MPI_Abort( MPI_COMM_WORLD, 1 );

   /* xlocal[][0] is lower ghostpoints, xlocal[][maxn+2] is upper */

   /* Fill the data as specified */
   for (i=1; i<=maxn/size; i++)
          for (j=0; j<maxn; j++)
             xlocal[i][j] = rank;
   for (j=0; j<maxn; j++) {
          xlocal[0][j] = -1;
          xlocal[maxn/size+1][j] = -1;
   }

   /* Send up unless I'm at the top, then receive from below */
   /* Note the use of xlocal[i] for &xlocal[i][0] */
   nreq = 0;
```

```
  if (rank < size - 1)
        MPI_Isend( xlocal[maxn/size], maxn, MPI_DOUBLE, rank + 1, 0,
                  MPI_COMM_WORLD, &r[nreq++] );
  if (rank > 0)
        MPI_Irecv( xlocal[0], maxn, MPI_DOUBLE, rank - 1, 0,
MPI_COMM_WORLD,
                  &r[nreq++] );
  /* Send down unless I'm at the bottom */
  if (rank > 0)
        MPI_Isend( xlocal[1], maxn, MPI_DOUBLE, rank - 1, 1,
MPI_COMM_WORLD,
                   &r[nreq++] );
  if (rank < size - 1)
        MPI_Irecv( xlocal[maxn/size+1], maxn, MPI_DOUBLE, rank + 1, 1,
                  MPI_COMM_WORLD, &r[nreq++] );

  MPI_Waitall( nreq, r, statuses );

  /* Check that we have the correct results */
  errcnt = 0;
  for (i=1; i<=maxn/size; i++)
        for (j=0; j<maxn; j++)
          if (xlocal[i][j] != rank) errcnt++;
  for (j=0; j<maxn; j++) {
        if (xlocal[0][j] != rank - 1) errcnt++;
        if (rank < size-1 && xlocal[maxn/size+1][j] != rank + 1) errcnt++;
  }

  MPI_Reduce( &errcnt, &toterr, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD );
  if (rank == 0) {
        if (toterr)
          printf( "! found %d errors\n", toterr );
        else
          printf( "No errors\n" );
  }

  MPI_Finalize( );
  return 0;
}
```

Output

```
% mpicc -o exchng exchng.c
% mpirun -np 4 exchng
No errors
```

Makefile

```
# Generated automatically from Makefile.in by configure.
ALL: exchng
exchng: exchng.c
        mpicc -o exchng exchng.c
```

Exercise: Shifting data around

```
#include <stdio.h>
#include "mpi.h"

/* This example handles a 12 x 12 mesh, on 4 processors only. */
#define maxn 12

int main( argc, argv )
int argc;
char **argv;
{
   int rank, value, size, errcnt, toterr, i, j;
   int up_nbr, down_nbr;
   MPI_Status status;
   double x[12][12];
   double xlocal[(12/4)+2][12];

   MPI_Init( &argc, &argv );

   MPI_Comm_rank( MPI_COMM_WORLD, &rank );
   MPI_Comm_size( MPI_COMM_WORLD, &size );

   if (size != 4) MPI_Abort( MPI_COMM_WORLD, 1 );

   /* xlocal[][0] is lower ghostpoints, xlocal[][maxn+2] is upper */

   /* Fill the data as specified */
   for (i=1; i<=maxn/size; i++)
          for (j=0; j<maxn; j++)
             xlocal[i][j] = rank;
   for (j=0; j<maxn; j++) {
          xlocal[0][j] = -1;
          xlocal[maxn/size+1][j] = -1;
   }

   /* Send up and receive from below (shift up) */
   /* Note the use of xlocal[i] for &xlocal[i][0] */
   /* Note that we use MPI_PROC_NULL to remove the if statements that
     would be needed without MPI_PROC_NULL */
   up_nbr = rank + 1;
   if (up_nbr >= size) up_nbr = MPI_PROC_NULL;
   down_nbr = rank - 1;
   if (down_nbr < 0) down_nbr = MPI_PROC_NULL;
   MPI_Sendrecv( xlocal[maxn/size], maxn, MPI_DOUBLE, up_nbr, 0,
                     xlocal[0], maxn, MPI_DOUBLE, down_nbr, 0,
                     MPI_COMM_WORLD, &status );

   /* Send down and receive from above (shift down) */
   MPI_Sendrecv( xlocal[1], maxn, MPI_DOUBLE, down_nbr, 1,
                     xlocal[maxn/size+1], maxn, MPI_DOUBLE, up_nbr, 1,
                     MPI_COMM_WORLD, &status );

   /* Check that we have the correct results */
   errcnt = 0;
   for (i=1; i<=maxn/size; i++)
          for (j=0; j<maxn; j++)
             if (xlocal[i][j] != rank) errcnt++;
   for (j=0; j<maxn; j++) {
          if (xlocal[0][j] != rank - 1) errcnt++;
          if (rank < size-1 && xlocal[maxn/size+1][j] != rank + 1) errcnt++;
   }

   MPI_Reduce( &errcnt, &toterr, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD );
   if (rank == 0) {
          if (toterr)
             printf( "! found %d errors\n", toterr );
          else
             printf( "No errors\n" );
   }

   MPI_Finalize( );
   return 0;
}
```

Output

```
% mpicc -o exchng exchng.c
% mpirun -np 4 exchng
No errors
```

Makefile

```
# Generated automatically from Makefile.in by configure.
ALL: exchng
exchng: exchng.c
          mpicc -o exchng exchng.c
```

Exercise: Exchanging data with MPI_Sendrecv

```
#include <stdio.h>
#include "mpi.h"
```

```
/* This example handles a 12 x 12 mesh, on 4 processors only. */
#define maxn 12

int main( argc, argv )
int argc;
char **argv;
{
   int rank, value, size, errcnt, toterr, i, j;
   int up_nbr, down_nbr;
   MPI_Status status;
   double x[12][12];
   double xlocal[(12/4)+2][12];

   MPI_Init( &argc, &argv );

   MPI_Comm_rank( MPI_COMM_WORLD, &rank );
   MPI_Comm_size( MPI_COMM_WORLD, &size );

   if (size != 4) MPI_Abort( MPI_COMM_WORLD, 1 );

   /* xlocal[][0] is lower ghostpoints, xlocal[][maxn+2] is upper */

   /* Fill the data as specified */
   for (i=1; i<=maxn/size; i++)
          for (j=0; j<maxn; j++)
             xlocal[i][j] = rank;
   for (j=0; j<maxn; j++) {
          xlocal[0][j] = -1;
          xlocal[maxn/size+1][j] = -1;
   }

   /* Processors 0 and 1 exchange, 2 and 3 exchange, etc.  Then
     1 and 2 exchange, 3 and 4, etc.  The formula for this is
     if (even) exchng up else down
     if (odd)  exchng up else down
     */
   /* Note the use of xlocal[i] for &xlocal[i][0] */
   /* Note that we use MPI_PROC_NULL to remove the if statements that
     would be needed without MPI_PROC_NULL */
   up_nbr = rank + 1;
   if (up_nbr >= size) up_nbr = MPI_PROC_NULL;
   down_nbr = rank - 1;
   if (down_nbr < 0) down_nbr = MPI_PROC_NULL;

   if ((rank % 2) == 0) {
          /* exchange up */
          MPI_Sendrecv( xlocal[maxn/size], maxn, MPI_DOUBLE, up_nbr, 0,
                        xlocal[maxn/size+1], maxn, MPI_DOUBLE, up_nbr, 0,
                        MPI_COMM_WORLD, &status );
```

```
    }
    else {
        /* exchange down */
        MPI_Sendrecv( xlocal[1], maxn, MPI_DOUBLE, down_nbr, 0,
                        xlocal[0], maxn, MPI_DOUBLE, down_nbr, 0,
                        MPI_COMM_WORLD, &status );
    }

    /* Do the second set of exchanges */
    if ((rank % 2) == 1) {
        /* exchange up */
        MPI_Sendrecv( xlocal[maxn/size], maxn, MPI_DOUBLE, up_nbr, 1,
                        xlocal[maxn/size+1], maxn, MPI_DOUBLE, up_nbr, 1,
                        MPI_COMM_WORLD, &status );
    }
    else {
        /* exchange down */
        MPI_Sendrecv( xlocal[1], maxn, MPI_DOUBLE, down_nbr, 1,
                        xlocal[0], maxn, MPI_DOUBLE, down_nbr, 1,
                        MPI_COMM_WORLD, &status );
    }

    /* Check that we have the correct results */
    errcnt = 0;
    for (i=1; i<=maxn/size; i++)
        for (j=0; j<maxn; j++)
          if (xlocal[i][j] != rank) errcnt++;
    for (j=0; j<maxn; j++) {
        if (xlocal[0][j] != rank - 1) errcnt++;
        if (rank < size-1 && xlocal[maxn/size+1][j] != rank + 1) errcnt++;
    }

    MPI_Reduce( &errcnt, &toterr, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD );
    if (rank == 0) {
        if (toterr)
          printf( "! found %d errors\n", toterr );
        else
          printf( "No errors\n" );
    }

    MPI_Finalize( );
    return 0;
}
```

Output

```
% mpicc -o exchng exchng.c
% mpirun -np 4 exchng
No errors
```

Makefile

```
# Generated automatically from Makefile.in by configure.
ALL: exchng
exchng: exchng.c
	mpicc -o exchng exchng.c
```