Peter Gruber
2nd Homework
HPSC 2004

Exercise: MPI-example: A simple Jacobi iteration
Problem Description: In this example, we want to compute an approximate

solution for the two-dimension Laplace Equation $u_{xx} + u_{yy} = 0$.
$u(x, y)$ is the solution function to be found, we will calculate approximate value at 12×12 grid points arranged in a square.
Vales at boundary points are constant (given by boundary conditions). Approximate values for all interior grid points are calculated iteratively using the formula

$$u_{i,j}^{(new)} = \frac{u_{i+1,j}^{(old)} + u_{i-1,j}^{(old)} + u_{i,j+1}^{(old)} + u_{i,j-1}^{(old)}}{4}$$

[For this purpose, we will simple believe that this formula does indeed lead to a solution for the Laplace Equation]
The total amount of work to be done is split on four processors, where each of them is responsible for a 3×12 submatrix. After every iteration, processors have to exchange values for uppermost and lowermost lines as as these values ar needed for the next iteration.
Boundary conditions and initial (starting) values are given by the following scheme:

$$\begin{pmatrix}
-1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\
3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 \\
3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 \\
2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\
2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\
2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1
\end{pmatrix}$$

For convergence testing, $diffnorm := \sqrt{\sum_{i,j=1}^{12} \left(x_{i,j}^{old} - x_{i,j}^{new}\right)^2}$ is calculated after every iteration. The partial values of every process are combined by a root process that determines, by testing $diffnorm \leq 0.01$, whether the computation process can be terminated.
The implementation of this algorithm may look like this:

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

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

Some necessary includes are done and the maximum number $maxn$ is set to 12

```
int main( argc, argv ) int argc; char **argv; {
    int        rank, value, size, errcnt, toterr, i, j, itcnt;
    int        i_first, i_last;
    MPI_Status status;
    double     diffnorm, gdiffnorm;
    double     xlocal[(12/4)+2][12];
    double     xnew[(12/3)+2][12];
```

Some variables are declared and initialized. $xlocal$ is a 5×12 matrix in which results during calculations are kept as well as some additional information for each processor that is the reference values from neighbouring processors or boundary condition. I would suppose to set $xnew[(12/4) + 2][12]$ to obtain the same dimensions like $xlocal$ as this variables could be named $xold$ and $xnew$.

```
    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 );
```

The number of processors (4) and the rank $(0, 1, 2, 3)$ are determined.

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

    /* Note that top and bottom processes have one less row of interior
       points */
    i_first = 1;
    i_last  = maxn/size;
    if (rank == 0)        i_first++;
    if (rank == size - 1) i_last--;
```

As the first and last processors have one less row of interior points there has to be a difference when calculating partial results. So the first i used for interior points is set in i_first and the last in i_last.

```
    /* 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[i_first-1][j] = -1;
xlocal[i_last+1][j] = -1;
}
```

The first part of the for-conditions fills the rows 1 to 3 with the rank of the processor. Then for all middle-ranks (1 and 2) the rows 0 and 4 are filled with -1. For the first (0) rank the row 1 instead of 0 is filled up with -1 which means that the first values will be overwritten. The same with the last (3) rank where row 3 will be reset with -1.
This leads to three possibilities for *xlocal*:

- rank = 1

	0	1	2	3	...	11
0	-1	-1	-1	-1	...	-1
1	rank	rank	rank	rank	...	rank
2	rank	rank	rank	rank	...	rank
3	rank	rank	rank	rank	...	rank
4	-1	-1	-1	-1	...	-1

- rank = 3

	0	1	2	3	...	11
0	-1	-1	-1	-1	...	-1
1	rank	rank	rank	rank	...	rank
2	rank	rank	rank	rank	...	rank
3	-1	-1	-1	-1	...	-1
4						

- rank = 0

	0	1	2	3	...	11
0						
1	-1	-1	-1	-1	...	-1
2	rank	rank	rank	rank	...	rank
3	rank	rank	rank	rank	...	rank
4	-1	-1	-1	-1	...	-1

```
itcnt = 0;
do {
/* Send up unless I'm at the top, then receive from below */
/* Note the use of xlocal[i] for &xlocal[i][0] */
if (rank < size - 1)
    MPI_Send( xlocal[maxn/size], maxn, MPI_DOUBLE, rank + 1, 0,
          MPI_COMM_WORLD );
if (rank > 0)
    MPI_Recv( xlocal[0], maxn, MPI_DOUBLE, rank - 1, 0,
          MPI_COMM_WORLD, &status );
/* Send down unless I'm at the bottom */
if (rank > 0)
    MPI_Send( xlocal[1], maxn, MPI_DOUBLE, rank - 1, 1,
```

```
            MPI_COMM_WORLD );
    if (rank < size - 1)
        MPI_Recv( xlocal[maxn/size+1], maxn, MPI_DOUBLE, rank + 1, 1,
              MPI_COMM_WORLD, &status );
```

In this section the rows 1 and 3 of *xlocal* of processors 1 and 2 are sent to the next processor. As processors 0 and 3 only have one neighbour there is only one message needed.
This exchange of information is needed because values of neighbouring processors are needed for the next iteration step.

```
    /* Compute new values (but not on boundary) */
    itcnt ++;
    diffnorm = 0.0;
    for (i=i_first; i<=i_last; i++)
        for (j=1; j<maxn-1; j++) {
        xnew[i][j] = (xlocal[i][j+1] + xlocal[i][j-1] +
                  xlocal[i+1][j] + xlocal[i-1][j]) / 4.0;
        diffnorm += (xnew[i][j] - xlocal[i][j]) *
                    (xnew[i][j] - xlocal[i][j]);
        }
```

Here the values are calculated as given in the assignment and written to *xnew*.

```
    /* Only transfer the interior points */
    for (i=i_first; i<=i_last; i++)
        for (j=1; j<maxn-1; j++)
        xlocal[i][j] = xnew[i][j];
```

The new values of interior points are transferred to *xlocal* to be ready for use for the next iteration.

```
    MPI_Allreduce( &diffnorm, &gdiffnorm, 1, MPI_DOUBLE, MPI_SUM,
                MPI_COMM_WORLD );
    gdiffnorm = sqrt( gdiffnorm );
    if (rank == 0) printf( "At iteration %d, diff is %e\n", itcnt,
                   gdiffnorm );
    } while (gdiffnorm > 1.0e-2 && itcnt < 100);
```

All norms of differences are summed up. If the processor is root it prints the number of iteration and the error expressed by $diffnorm$ (as explained above). The iteration stops after reaching a certain epsilon or 100 iterations.

```
    MPI_Finalize( );
    return 0;
}
```

MPI-environment is closed and 0 is returned.