



Parallel Programming with MPI: Tips and Techniques

Peter Jimack

School of Computing, University of Leeds, UK

<http://www.comp.leeds.ac.uk/pkj>

Contents of Lectures

1. The Distributed Memory Paradigm.
2. Message Passing and MPI.
3. Getting Started with MPI.
4. An Example with One-to-All and All-to-One Communication.
5. An Example with Point-to-Point Communication.
6. An Example with Non-Blocking Communication.

1. The Distributed Memory Paradigm

- Each processor has its own local memory associated with it, which only it can access.
- To execute computations on any data, that data must be in the local memory.
- A mechanism is required for allowing items of data to be made available to other processors.
- This is an abstraction of the parallel computer – if your particular computer has physically shared memory you can still program according to this abstraction.
- Computers which do have physically distributed memory include:
 - IBM Blue Gene,
 - Earth Simulator,
 - networks of workstations.

2. Message Passing and MPI

- The most common mechanism for transferring data between processors with their own distributed memory is to pass a message between these processors.
- This requires the cooperation of each processor involved in the communication.
- It is implemented by extensions to conventional programming languages, such as C, C++ or FORTRAN, known as “message passing libraries”.
- MPI (Message Passing Interface) is a definition of a message passing library.
- Its implementation and availability across a wide variety of platforms has led to genuine portability of parallel code over recent years (i.e. since 1994).

3. Getting Started with MPI

```
#include <stdio.h>
#include "mpi.h"
main(int argc, char** argv)
{
    int nprocs, nid;

    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
    MPI_Comm_rank(MPI_COMM_WORLD, &nid);
    if (nid == 0)
        printf("Hello from processor %i of %i \n", nid, nprocs);
    MPI_Finalize();
}
```

4. One-toAll and All-to-One Communication

We now introduce message passing into a program...

```
#include <stdio.h>
#include "mpi.h"
main(int argc, char** argv)
{
    int i, N, noprocs, nid;
    float sum = 0, Gsum;

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &nid);
    MPI_Comm_size(MPI_COMM_WORLD, &noprocs);
```

```

if(nid == 0){
    printf("Enter the no. of terms N -> ");
    scanf("%d",&N);
}
MPI_Bcast(&N,1,MPI_INT,0,MPI_COMM_WORLD);
for(i = nid; i < N; i += nprocs)
    if(i % 2)
        sum -= (float) 1 / (i + 1);
    else
        sum += (float) 1 / (i + 1);
MPI_Reduce(&sum,&Gsum,1,MPI_FLOAT,MPI_SUM,0,
           MPI_COMM_WORLD);
if(nid == 0)
    printf("Est. of ln(2) is %f \n",Gsum);
MPI_Finalize();
}

```

4. Point-to-Point Communication

Messages do not have to involve every process in a communicator...

```
#include <stdio.h>
#include "mpi.h"
main(int argc, char** argv)
{
    int nprocs, nid, i, n, size;
    float *a, *b, sum = 0.0, Gsum;
    FILE *fp;
    MPI_Status status;

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &nid);
    MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
```

```
if(nid == 0){
    fp = fopen("DotData.Txt","rt");
    fscanf(fp,"%d",&n);
    MPI_Bcast(&n,1,MPI_INT,0,MPI_COMM_WORLD);
    if(n % nprocs){
        printf("nprocs not a mult. of n.\n");
        MPI_Abort(MPI_COMM_WORLD,-1);
    }
    a = (float*)calloc(n,sizeof(float));
    b = (float*)calloc(n,sizeof(float));
    for(i = 0; i < n; i++)
        fscanf(fp,"%f %f",&a[i],&b[i]);
    fclose(fp);
}
```

```
for(i = 1, size = n / noprocs; i < noprocs; i++){
    MPI_Send(&a[size*i],size,MPI_FLOAT,i,10,MPI_COMM_WORLD);
    MPI_Send(&b[size*i],size,MPI_FLOAT,i,20,MPI_COMM_WORLD);
}
}
else{
    MPI_Bcast(&n,1,MPI_INT,0,MPI_COMM_WORLD);
    if(n % noprocs){
        printf("noprocs not a mult. of n.\n");
        MPI_Abort(MPI_COMM_WORLD,-1);
    }
    size = n / noprocs;
    a = (float*)calloc(size,sizeof(float));
    b = (float*)calloc(size,sizeof(float));
```

```
MPI_Recv(&a[0],size,MPI_FLOAT,0,10,MPI_COMM_WORLD,
        &status);
MPI_Recv(&b[0],size,MPI_FLOAT,0,20,MPI_COMM_WORLD,
        &status);
}
for(i = 0; i < size; i++)
    sum += a[i] * b[i];
MPI_Reduce(&sum,&Gsum,1,MPI_FLOAT,MPI_SUM,0,
          MPI_COMM_WORLD);
if(nid == 0)
    printf("Inner product is %f \n",Gsum);
MPI_Finalize();
}
```

5. Non-Blocking Communication

- The next program solves the PDE

$$\nabla^2 u = f \quad \text{in } \Omega = (0, 1) \times (0, 1),$$

subject to the BC: $u = g$ on $\partial\Omega$.

- Central differences give the following discrete problem:

$$u[i+1][j] + u[i-1][j] + u[i][j+1] + u[i][j-1] - 4u[i][j] = h^2 f[i][j]$$

for i and j in $\{1, \dots, N-1\}$.

- Jacobi iteration then yields the iterative formula

$$new[i][j] = (old[i+1][j] + old[i-1][j] + old[i][j+1] + old[i][j-1] - h^2 f[i][j]) / 4$$

for i and j in $\{1, \dots, N-1\}$.

- For simplicity we take $f \equiv 0$ and $g \equiv 1$.
- To obtain a sol. in parallel the rows are partitioned between the procs.
- To apply the iterative formula communication is required after each loop.
- Since communication only effects top and bottom row of each block the update of the other rows can be completed simultaneously.
- This overlapping of communication with computation requires non-blocking sends and receives.

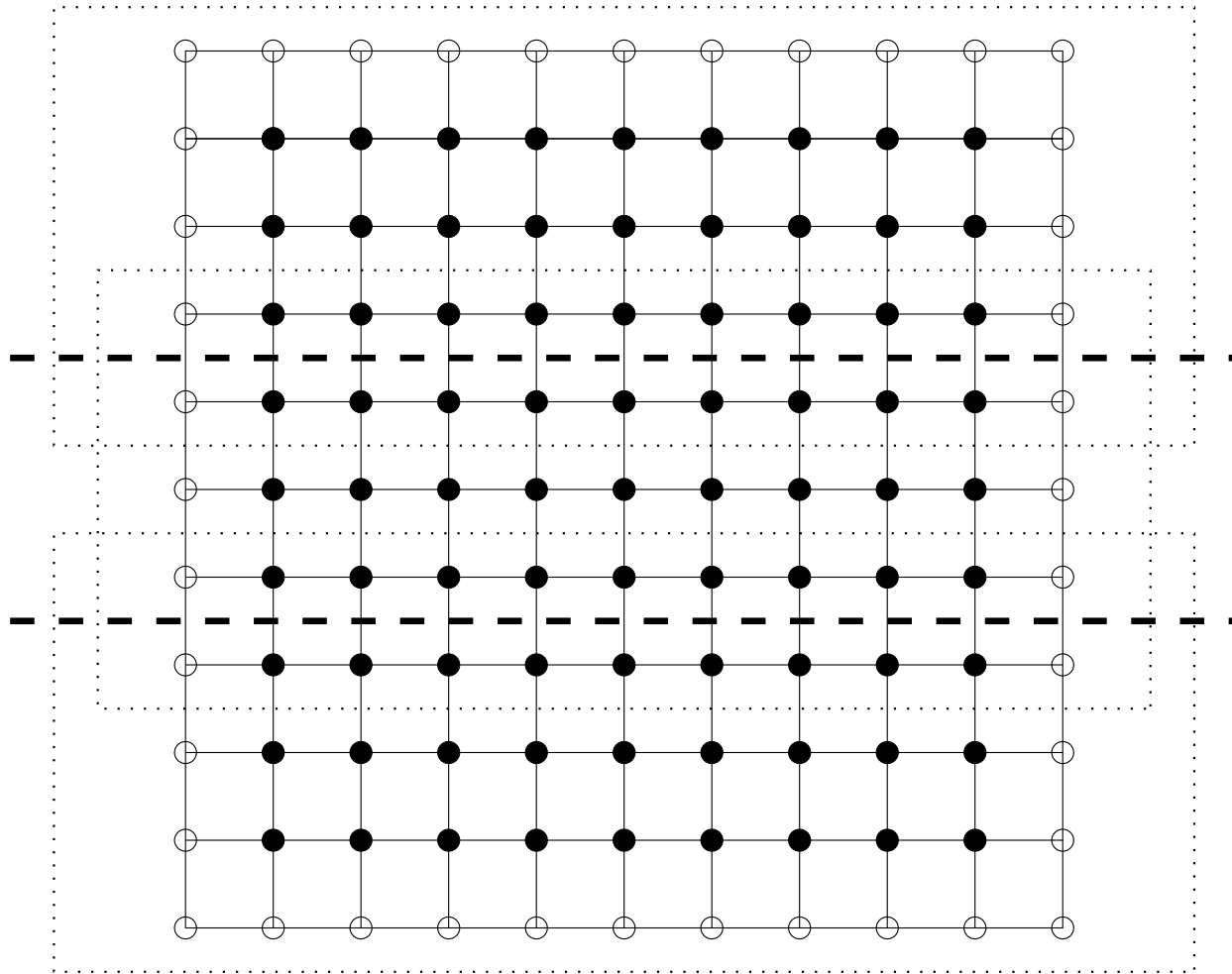


Figure 1: Partition of a finite difference mesh across 3 processes.

```
#include <stdio.h>
#include "mpi.h"
#define N 100
#define Tol 0.00001

float **matrix(int m, int n)
{
    int i;
    float **ptr;

    ptr = (float**)calloc(m, sizeof(float *));
    for(i = 0; i < m ;i++)
        ptr[i]=(float*)calloc(n, sizeof(float));
    return (ptr);
}
```

```

float iteration(float **old, float **new, int start,
               int finish)
{
    float diff, maxerr = 0;
    int i, j;

    for(i = start; i < finish; i++)
        for(j = 1; j < N; j++){
            new[i][j] = 0.25 * (old[i+1][j]+old[i-1][j]+
                               old[i][j+1]+old[i][j-1]);
            diff = new[i][j] - old[i][j];
            if(diff < 0) diff = -diff;
            if(maxerr < diff) maxerr = diff;
        }
    return (maxerr);
}

```

```
main(int argc, char** argv)
{
    float **new, **old, **tmp, maxerr, err, maxerrG;
    int noprocs, nid, remainder, size, i, j;
    char str[20];
    FILE *fp;
    MPI_Status status;
    MPI_Request req_send10, req_send20, req_recv10, req_recv20;

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &nid);
    MPI_Comm_size(MPI_COMM_WORLD, &noprocs);
    remainder = (N - 1) % noprocs;
    size = (N - 1 - remainder)/noprocs;
```

```

if(nid < remainder)
    size = size + 2;
else
    size = size + 1;
new = matrix(size+1,N+1);
old = matrix(size+1,N+1);
for(i = 0; i < size + 1; i++)
    new[i][0] = new[i][N] = old[i][0] = old[i][N] = 1;
if(nid == 0)
    for(j = 1; j < N; j++)
        new[0][j] = old[0][j] = 1;
if(nid == noprocs - 1)
    for(j = 1; j < N; j++)
        new[size][j] = old[size][j] = 1;
maxerr = iteration(old,new,1,size);
MPI_Allreduce(&maxerr,&maxerrG,1,MPI_FLOAT,MPI_MAX,
              MPI_COMM_WORLD);

```

```

while(maxerrG > Tol){
    tmp = new; new = old; old = tmp;
    req_send10 = req_recv20 = MPI_REQUEST_NULL;
    if(nid < noprocs-1){
        MPI_Isend(&old[size-1][1],N-1,MPI_FLOAT,nid+1,10,
                MPI_COMM_WORLD,&req_send10);
        MPI_Irecv(&old[size][1],N-1,MPI_FLOAT,nid+1,20,
                MPI_COMM_WORLD,&req_recv20);
    }
    req_send20 = req_recv10 = MPI_REQUEST_NULL;
    if(nid > 0){
        MPI_Isend(&old[1][1],N-1,MPI_FLOAT,nid-1,20,
                MPI_COMM_WORLD,&req_send20);
        MPI_Irecv(&old[0][1],N-1,MPI_FLOAT,nid-1,10,
                MPI_COMM_WORLD,&req_recv10);
    }
}

```

```
maxerr = iteration(old,new,2,size-1);
if(nid < noprocs-1)
    MPI_Wait(&req_recv20,&status);
err = iteration(old,new,size-1,size);
if(err > maxerr)
    maxerr = err;
if(nid > 0)
    MPI_Wait(&req_recv10,&status);
err = iteration(old,new,1,2);
if(err > maxerr)
    maxerr = err;
MPI_Allreduce(&maxerr,&maxerrG,1,MPI_FLOAT,MPI_MAX,
              MPI_COMM_WORLD);
}
```

```
sprintf(str, "Solution%d.Txt", nid);
fp = fopen(str, "wt");
if(nid == 0)
    for(j = 0; j < N + 1; j++)
        fprintf(fp, "%6.4f\n", new[0][j]);
for(i = 1; i < size; i++)
    for(j = 0; j < N + 1; j++)
        fprintf(fp, "%6.4f\n", new[i][j]);
if(nid == noprocs - 1)
    for(j = 0; j < N + 1; j++)
        fprintf(fp, "%6.4f\n", new[size][j]);
fclose(fp);
MPI_Finalize();
}
```