MPI: Numerical Integration, P2P and Collective Communication

Kameswararao Anupindi

Department of Mechanical Engineering
Indian Institute of Technology Madras (IITM)

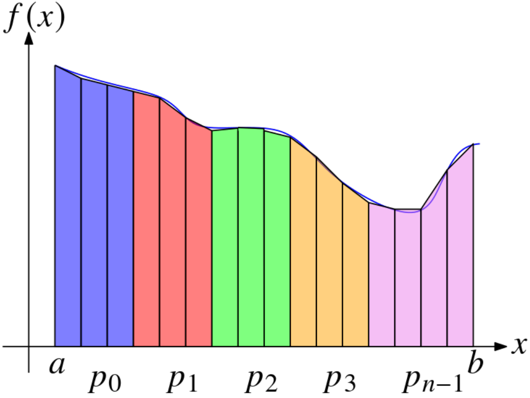
March, 2024

A few potential pitfalls of MPI_Send/MPI_Recv

Process A	Process B
x	MPI_Recv
MPI_Send	x

- Non-matching tags
- Rank of the destination process is **the same** as that of the source.

The Trapezoidal Rule approximation



$$\int_a^b f(x)dx = \frac{h}{2}\left[f(x_0) + f(x_n) + 2\left(f(x_1) + f(x_2) \ldots + f(x_{n-1})\right)\right] \tag{1}$$

The Trapezoidal Rule using MPI in C

```c
/* MPI parallel version of trapezoidal rule */
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include<mpi.h>

#define PI 3.14159265358

double func(double x)
{
  return (1.0 + sin(x));
}

double trapezoidal_rule(double la, double lb, double ln, double h)
{
  double total;
  double x;
  int i;

  total = (func(la) + func(lb))/2.0;
  for(i = 1; i <= ln-1; i++) /* sharing the work, use only local_n */
    {
      x = la + i*h;
      total += func(x);
    }
  total = total * h;

  return total;                   /* total for each thread, private */
}
```

The Trapezoidal Rule using MPI in C contd...

```
int main(int argc, char* argv[])
{
  double a, b, final_result, la, lb, lsum, h;
  int myid, nprocs, proc;
  int n, ln;

  MPI_Init(NULL, NULL);
  MPI_Comm_rank(MPI_COMM_WORLD, &myid);   /* myrank of the process */
  MPI_Comm_size(MPI_COMM_WORLD, &nprocs); /* size of the communicator */

  n = 1024;                      /* number of trapezoids.. */
  a = 0.0;
  b = PI;                        /* hard-coded.. */
  final_result = 0.0;

  h = (b-a)/n;
  ln = n/nprocs;                 /* nprocs evenly divides number of trapezoids */

  la = a + myid*ln*h;
  lb = la + ln*h;
  lsum = trapezoidal_rule(la, lb, ln, h); /* every process calls this function... */
```

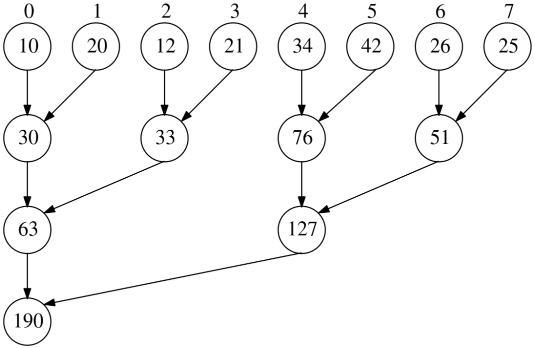
The Trapezoidal Rule using MPI in C contd...

```c
  if (myid != 0)
    {
      MPI_Send(&lsum, 1, MPI_DOUBLE, 0, 0, MPI_COMM_WORLD);
    }
  else                          /* process 0 */
    {
      final_result = lsum;
      for (proc = 1; proc < nprocs; proc++)
        {
          MPI_Recv(&lsum, 1, MPI_DOUBLE, proc, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
          final_result += lsum;
        }
    }

  if (myid == 0)                /* output is only printed by process 0 */
    {
      printf("\n The area under the curve (1+sin(x)) between 0 to PI is equal to %lf \n\n", final_result);
    }

  MPI_Finalize();
  return 0;
}
```

The Trapezoidal Rule - Enhancements - Dealing with input and output

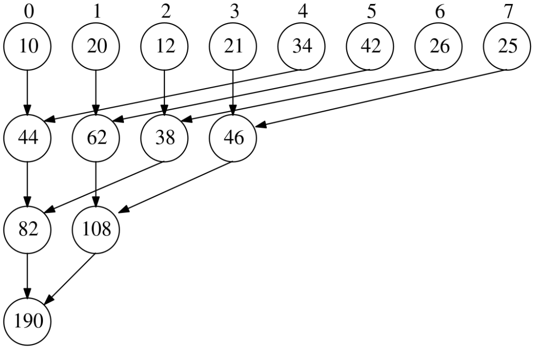
```
if (myid == 0)
  {
    printf("\n Enter the lower limit, upper limit and n");
    scanf(&a, &b, &n);

    for (proc = 1, proc<nprocs; proc++)
      {
        MPI_Send(&a, ....);
        MPI_Send(&b, ....);
        MPI_Send(&n, ....);
      }
  }
else
  {
    MPI_Recv(&a, 1, ...);
    MPI_Recv(&b, 1, ...);
    MPI_Recv(&n, 1, ...);
  }



return 0;
}
```

The Trapezoidal Rule - Enhancements - Calculating global sum



- Original sum: 7 receives and adds
- Tree sum: 3 receives and adds
- if nprocs = 1024, tree sum would do only 10 receives and adds

The Trapezoidal Rule - Calculating global sum - another way



- Several possibilities exist
- A method works best for small trees, and another for large trees!
- A method may work best for system A, and another for system B.
- MPI provides a **global sum** that works the best in the form of **Collective Communication**.

Collective Communication - MPI_Reduce

```c
int MPI_Reduce(
               void* input_data_p,
               void* output_data_p,
               int count,
               MPI_Datatype datatype,
               MPI_Op operator,
               int root,
               MPI_Comm communicator);
```

```fortran
MPI_Reduce(sendbuf, recvbuf, count, datatype, op, root, comm, ierror)
TYPE(*), DIMENSION(:), INTENT(IN) :: sendbuf
TYPE(*), DIMENSION(:) :: recvbuf
INTEGER, INTENT(IN) :: count, root
TYPE(MPI_Datatype), INTENT(IN) :: datatype
TYPE(MPI_Op), INTENT(IN) :: op
TYPE(MPI_Comm), INTENT(IN) :: comm
INTEGER, OPTIONAL, INTENT(OUT) :: ierror
```

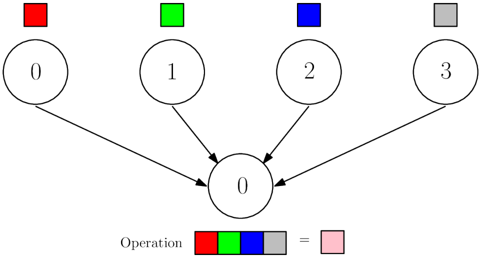
Collective Communication - MPI_Reduce

```
MPI_Reduce(&lsum, &final_result, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
```

```
call MPI_Reduce(lsum, final_result, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD, mpierror);
```

MPI_MAX	MPI_LOR
MPI_MIN	MPI_BAND
MPI_SUM	MPI_BOR
MPI_PROD	MPI_MAXLOC
MPI_LAND	MPI_MINLOC

Collective communication: Reduce




```
MPI_Reduce(
            void *send_buffer,
            void *receive_buffer,
            int count,
            MPI_Datatype datatype,
            MPI_Op operator,
            int root,
            MPI_Comm communicator)
```

Difference between Collective and P2P communications

- All the processes must call the same MPI Collective Communication (CC)
- The arguments passed by each process to MPI CC must be *compatible*
- All processes must supply an output_data_p, although this is needed only on *root*
- While P2P are matched using *communicator* and *tags*, MPI CC are matched solely on the basis of *communicator* and order of calling.