MPI: Numerical Integration, P2P and Collective Communication

Kameswararao Anupindi

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

March, 2024

Multiple CC calls

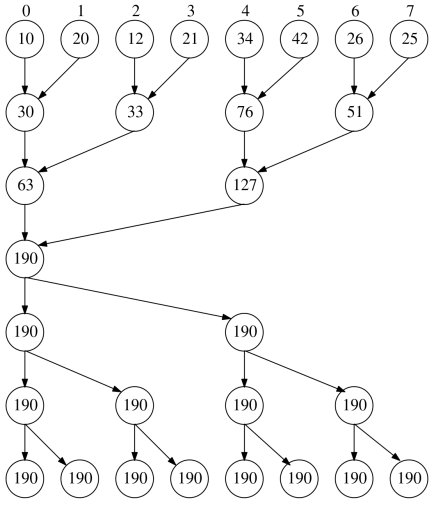
```
Process 0:
  a = 1, b = 0, c = 2, d = 0;
  dest_process = 0;

  MPI_Reduce(&a, &b, ..., 0, comm);
  MPI_Reduce(&c, &d, ..., 0, comm);

Process 1:
  a = 1, b = 0, c = 2, d = 0;
  dest_process = 0;

  MPI_Reduce(&c, &d, ..., 0, comm);
  MPI_Reduce(&a, &b, ..., 0, comm);

Process 2:
  a = 1, b = 0, c = 2, d = 0;
  dest_process = 0;

  MPI_Reduce(&a, &b, ..., 0, comm);
  MPI_Reduce(&c, &d, ..., 0, comm);
```

Reduction on the same variable

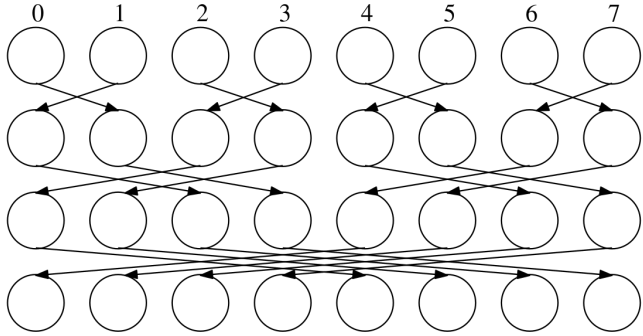
```
MPI_Reduce(&x, &x, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);
```

- Illegal in MPI
- Produces unpredictable result.

MPI_Allreduce: Tree and Reverse-tree



MPI_Allreduce: Butterfly

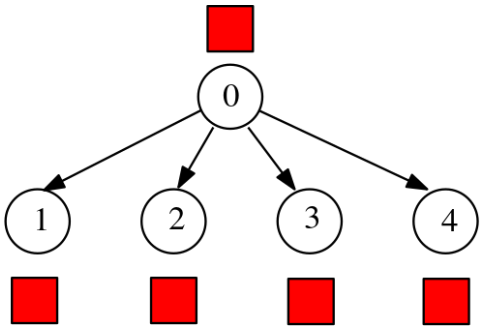


MPI_Allreduce function prototype

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

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

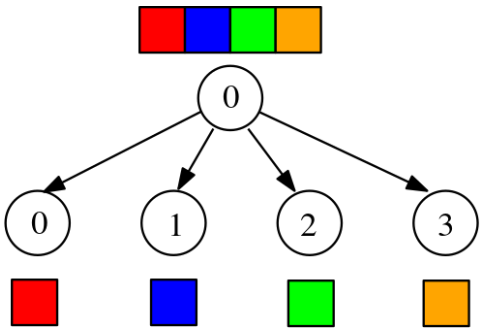
Collective communication: Broadcast




```
MPI_Bcast(
          void *data,
          int count,
          MPI_Datatype datatype,
          int root,
          MPI_Comm communicator)
```

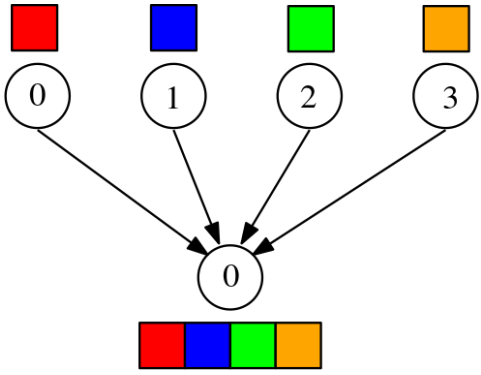
- Use a tree-structured communication instead!
- data_p is an input argument on root (send_proc) and output on the other processes.

Collective communication: Scatter




```
MPI_Scatter(
          void *send_data,
          int send_count,
          MPI_Datatype datatype,
          void *receive_data,
          int receive_count,
          MPI_Datatype datatype,
          int root,
          MPI_Comm communicator)
```

Collective communication: Gather




```
MPI_Gather(
          void *send_data,
          int send_count,
          MPI_Datatype datatype,
          void *receive_data,
          int receive_count,
          MPI_Datatype datatype,
          int root,
          MPI_Comm communicator)
```

Broadcast example program

```
if (myid == 0)
  buf = 327;


MPI_Bcast(&buf, 1, MPI_INT, 0, MPI_COMM_WORLD);

if (myid == 0)
  printf("\n Broadcasted values on processors are:\n");

printf("\t (%d, %d)\n", myid, buf);
```

Gather example program

```
int send_buf, *recv_buf;
if (myid == 0)
  {
    recv_buf = (int *)malloc(size*sizeof(int));
  }


send_buf = 100+myid*myid;


MPI_Gather(&send_buf, 1, MPI_INT, recv_buf, 1, MPI_INT, 0, MPI_COMM_WORLD);

if (myid == 0)
  {
    printf("\n Received values on host process are:\n");
    for(i=0; i<size; i++)
      printf("\t %d", recv_buf[i]);
    printf("\n");
  }

if (myid == 0)
  free(recv_buf);
```

Scatter example program

```
int *send_buf, recv_buf;
if (myid == 0)
  {
    send_buf = (int *)malloc(size*sizeof(int));
    for(i=0; i<size; i++)
      send_buf[i] = 100+i*5+i;
  }


MPI_Scatter(send_buf, 1, MPI_INT, &recv_buf, 1, MPI_INT, 0, MPI_COMM_WORLD);

if (myid == 0)
  printf("\n Received values on processors are:\n");


printf("\t (%d, %d)", myid, recv_buf);

if (myid == 0)
  free(send_buf);
```

Matrix - Vector Multiplication using block decomposition

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \times \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

- Row block-decomposition
- Consider

$$\begin{bmatrix} x_1 & x_2 & \cdots & x_n \end{bmatrix}^T$$

  has block-decomposition as well
- How to arrange that each process has access to all components of $[x]$?

MPI_Allgather

```c
int MPI_Allgather(
                  const void* send_buf_p,
                  int send_count,
                  MPI_Datatype send_type,
                  void* recv_buf_p,
                  int recv_count,
                  MPI_Datatype recv_type,
                  MPI_Comm communicator);
```

```
MPI_Allgather(sendbuf, sendcount, sendtype, recvbuf, recvcount, recvtype, comm, ierror)
TYPE(*), DIMENSION(..), INTENT(IN) :: sendbuf
TYPE(*), DIMENSION(..) :: recvbuf
INTEGER, INTENT(IN) :: sendcount, recvcount
TYPE(MPI_Datatype), INTENT(IN) :: sendtype, recvtype
TYPE(MPI_Comm), INTENT(IN) :: comm
INTEGER, OPTIONAL, INTENT(OUT) :: ierror
```

Motivation for MPI Derived Datatypes

```
double x[1000];

if (myid == 0)
  for(i = 0; i < 1000; i++)
    MPI_Send(&x[i], 1, MPI_DOUBLE, 1, 0, comm);
else
  for(i = 0; i < 1000; i++)
    MPI_Recv(&x[i], 1, MPI_DOUBLE, 0, 0, comm, &status);
```

```
double x[1000];

if (myid == 0)
  MPI_Send(x, 1000, MPI_DOUBLE, 1, 0, comm);
else
  MPI_Recv(x, 1000, MPI_DOUBLE, 0, 0, comm, &status);
```

MPI Data consolidation

- The **count** argument
- MPI Derived Datatypes
- MPI_Pack and MPI_Unpack

MPI Derived Datatypes

- Collection of data items in memory by storing both **type** and **relative memory locations**

Variable	Address
a	24
b	40
c	48

```
{(MPI_DOUBLE,0), (MPI_DOUBLE,16), (MPI_DOUBLE,24)}
```

Building MPI Derived Datatypes

```
int MPI_Type_create_struct(
             int            count,
             int            array_of_blocklengths[],
             MPI_Aint       array_of_displacements[],
             MPI_Datatype   array_of_types[],
             MPI_Datatype*  new_type_p);

int array_of_blocklengths[3] = {1, 1, 1};

array_of_blocklengths[0] = 5;

array_of_displacements[] = {0, 16, 24};
```

Building MPI Derived Datatypes contd...

```
int MPI_Get_address(
    void*       location_p,
    MPI_Aint*   address_p);

MPI_Aint a_addr, b_addr, c_addr;

MPI_Get_address(&a, &a_addr);
array_of_displacements[0] = 0;

MPI_Get_address(&b, &b_addr);
array_of_displacements[1] = b_addr - a_addr;

MPI_Get_address(&c, &c_addr);
array_of_displacements[2] = c_addr - b_addr;
```

Building MPI Derived Datatypes contd...

```
MPI_Datatype array_of_types[3] = {MPI_DOUBLE, MPI_DOUBLE, MPI_INT);
....
....
MPI_Datatype new_mpi_t;
...
MPI_Type_create_struct(3, array_of_blocklengths,
    array_of_displacements, array_of_types, &new_mpi_t);
...
int MPI_Type_commit(MPI_Datatype* new_mpi_type_p);
...
...
MPI_Bcast(&a, 1, new_mpi_t, 0, comm);
...
...
int MPI_Type_free(MPI_Datatype* old_mpi_type_p);
```