Collective Communications

Ned Nedialkov

McMaster University

24 January 2023

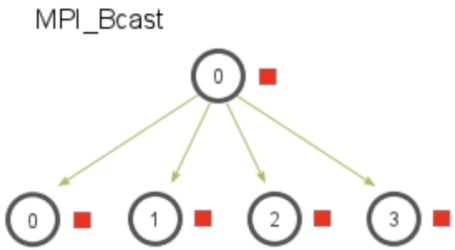
Outline

- Broadcast
- Reduce
- Scatter
- Gather
- Example: parallel $A \times b$

Broadcast

Broadcast: a single process sends the same data to all processes in a communicator. See also

▸ MPI Broadcast and Collective Communication




```
int  keywordstyleMPI_Bcast(void *buffer, int count,
        keywordstyleMPI_Datatype datatype, int root,
        keywordstyleMPI_Comm comm);
```

`buffer`	starting address of buffer (in/out)
`count`	number of entries in buffer
`datatype`	data type of buffer
`root`	rank of broadcast root
`comm`	communicator

- keywordstyleMPI_Bcast sends a copy of the message on process with rank root to each process in comm.
- Must be called in each process.
- Data is sent in root and received by all other processes.
- buffer is 'in' parameter in root and 'out' parameter in the rest of processes.
- Cannot receive broadcasted data with keywordstyleMPI_Recv.

Example: reading and broadcasting data

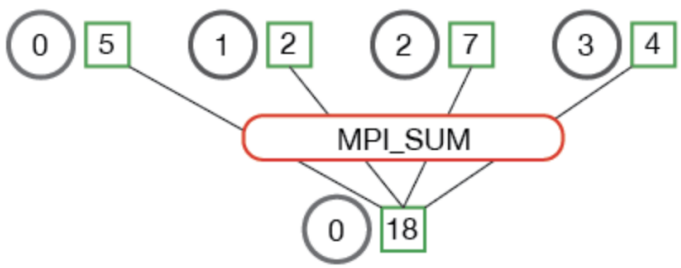
Code adapted from P. Pacheco, Parallel Programming with MPI

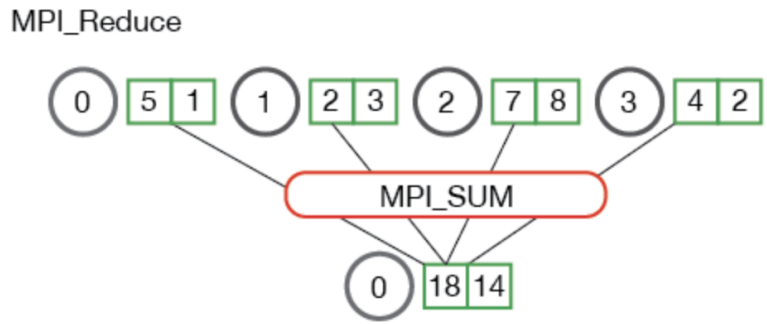
```
/* getdata2.c */
#include "mpi.h"
#include <stdio.h>
void Get_data2(double *a, double *b, int *n, int myrank) {
   if (myrank == 0) {
      printf("Enter a, b, and n\n");
      scanf("%lf %lf %d", a, b, n);
   }
    keywordstyleMPI_Bcast(a, 1,  keywordstyleMPI_DOUBLE, 0,
           keywordstyleMPI_COMM_WORLD);
    keywordstyleMPI_Bcast(b, 1,  keywordstyleMPI_DOUBLE, 0,
           keywordstyleMPI_COMM_WORLD);
    keywordstyleMPI_Bcast(n, 1,  keywordstyleMPI_INT, 0,
           keywordstyleMPI_COMM_WORLD);
}
```

Reduce

Data from all processes are combined using a binary operation.

E.g. with `keywordstyleMPI_SUM`





See also ▸ MPI Reduce and Allreduce

```
int  keywordstyleMPI_Reduce(void *sendbuf, void *recvbuf, int count,
        keywordstyleMPI_Datatype datatype,
        keywordstyleMPI_Op op, int root,  keywordstyleMPI_Comm comm);
```

`sendbuf`	address of send buffer
`recvbuf`	address of receive buffer
`count`	number of entries in send buffer
`datatype`	data type of elements in send buffer
`op`	reduce operation; predefined, `keywordstyleMPI_MIN`, `keywordstyleMPI_MAX`, `keywordstyleMPI_SUM`, ..., or user defined
`root`	rank of root process
`comm`	communicator

Must be called in all processes in a communicator.

Example: trapezoid with reduce

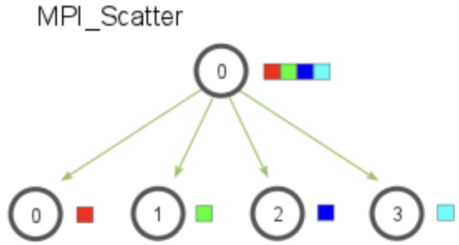
Code adapted from P. Pacheco, Parallel Programming with MPI

```
/* redtrap.c */
#include "mpi.h"
#include <stdio.h>
extern void Get_data2(double *a_ptr, double *b_ptr, int *n_ptr, int
        myrank);
extern double Trap(double local_a, double local_b, int local_n, double h);
int main(int argc, char **argv) {
   int myrank, p;
   double a, b, h;
   int n;
   double local_a, local_b;
   int local_n;
   double integral, total;
    keywordstyleMPI_Init(&argc, &argv);
    keywordstyleMPI_Comm_rank( keywordstyleMPI_COMM_WORLD, &myrank);
    keywordstyleMPI_Comm_size( keywordstyleMPI_COMM_WORLD, &p);
   Get_data2(&a, &b, &n, myrank);
   h = (b - a) / n;
   local_n = n / p;
```

```
   local_a = a + myrank * local_n * h;
   local_b = local_a + local_n * h;
   integral = Trap(local_a, local_b, local_n, h);
   // sum up from all processes
    keywordstyleMPI_Reduce(&integral, &total, 1,  keywordstyleMPI_DOUBLE,
           keywordstyleMPI_SUM, 0,  keywordstyleMPI_COMM_WORLD);
   if (myrank == 0) {
      printf("With n = %d trapezoids, our estimate\n", n);
      printf("of the integral from %f to %f = %f\n", a, b, total);
   }
    keywordstyleMPI_Finalize();
   return 0;
}
```

Scatter

Sends chunks of data
to all processes in a communicator. See also ▸ MPI Broadcast and Collective Communication




```
int  keywordstyleMPI_Scatter(void *sendbuf, int sendcount,
        keywordstyleMPI_Datatype sendtype, void *recvbuf, int recvcount,
        keywordstyleMPI_Datatype recvtype, int root,
        keywordstyleMPI_Comm comm);
```

`sendbuf`	starting address of send buffer (significant only at root)
`sendcount`	number of elements sent to each process (significant only at root)
`sendtype`	data type of `sendbuf` elements (significant only at root)
`recvbuf`	address of receive buffer
`recvcount`	number of elements for any single receive
`recvtype`	data type of `recvbuf` elements
`root`	rank of sending process
`comm`	communicator

- Assume p processes.
- keywordstyleMPI_Scatter splits data at sendbuf on root into p segments
- each of sendcount elements, and
- sends these segments to processes $0, 1, \ldots, p-1$ in order.

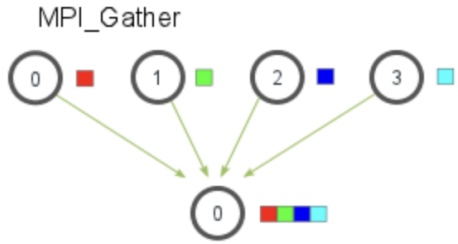
```
int  keywordstyleMPI_Scatterv(void *sendbuf, int *sendcounts, int *displs,
        keywordstyleMPI_Datatype sendtype, void *recvbuf, int recvcount,
        keywordstyleMPI_Datatype recvtype, int root,
        keywordstyleMPI_Comm comm)
```

`sendcounts`	integer array (of size p) specifying the number of elements to send to each process
`displs`	integer array (of size p) Entry `i` specifies the displacement (relative to `sendbuf`) from which to take the outgoing data to process `i`
`recvbuf`	address of receive buffer (output)

For an illustration, see

▸ Examples using MPI_SCATTER, MPI_SCATTERV

Gather



Gathers
together data from a group of processes.

```
int  keywordstyleMPI_Gather(void *sendbuf, int sendcount,
        keywordstyleMPI_Datatype sendtype, void *recvbuf, int recvcount,
        keywordstyleMPI_Datatype recvtype, int root,
        keywordstyleMPI_Comm comm )
```

`sendbuf`	starting address of send buffer
`sendcount`	number of elements in send buffer
`sendtype`	data type of `sendbuf` elements
`recvbuf`	address of receive buffer (significant only at root)
`recvcount`	number of elements for any single receive (significant only at root)
`recvtype`	data type of `recvbuf` elements (significant only at root)
`root`	root rank of receiving process
`comm`	communicator

- keywordstyleMPI_Gather collects data, stored at sendbuf, from each process in comm and stores the data on root at recvbuf
- Data is received from processes in order, i.e. from process 0, then from process 1 and so on
- Usually sendcount, sendtype are the same as recvcount, recvtype
- root and comm must be the same on all processes
- The receive parameters are significant only on root
- Amount of data sent/received must be the same

```
int  keywordstyleMPI_Allgather(void *sendbuf, int sendcount,
        keywordstyleMPI_Datatype sendtype,void *recvbuf, int recvcount,
        keywordstyleMPI_Datatype recvtype,  keywordstyleMPI_Comm comm)
```

- The block of data sent from the ith process is received by every process and placed in the ith block of the buffer recvbuf

```
int  keywordstyleMPI_Gatherv(void *sendbuf, int sendcount,
        keywordstyleMPI_Datatype sendtype,void *recvbuf, int
        *recvcounts,int *displs,
        keywordstyleMPI_Datatype recvtype,int root,
        keywordstyleMPI_Comm comm)
```

- The "opposite" of keywordstyleMPI_Scatterv

Example: parallel matrix times vector

Compute $y = A \cdot b$, where A is an $m \times n$ matrix and b is an n-vector using p processes.

Assume process 0 has A and b

1. Process 0
   - distributes A by rows
   - broadcasts b

   Denote the submatrix of A on process i by A_i.
2. Process i computes $y_i = A_i b$ serially.
3. Process 0 gathers the y_i for obtain y.

Example: parallel $A \times b$

Distribution

- p may not divide m exactly.
- Number of local rows, on process i, can be computed as

$$m_i = m \div p + \begin{cases} 1 & \text{if } i < m \mod p \\ 0 & \text{otherwise} \end{cases}$$

  - $\div$ is integer division
  - $\mod$ is remainder
- In C

```
#define NUM_LOCAL_ROWS(i, p, m) (m / p + ((i < m % p) ? 1 : 0))
```

- Process 0 scatters m_i rows to process i.
  That is, $m_i \times n$ items.

  Process 0 sends

to process	at displacement from A	# items
0	$d_0 = 0$	$s_0 = m_0 n$
1	$d_1 = s_0$	$s_1 = m_1 n$
2	$d_2 = d_1 + s_1$	$s_2 = m_2 n$
$\vdots$		
$p-1$	$d_{p-1} = d_{p-2} + s_{p-2}$	$s_{p-1} = m_{p-1} n$

Distributing A

```
#define NUM_LOCAL_ROWS(i, p, m) (m / p + ((i < m % p) ? 1 : 0))
```

```
   // Distribute matrix A of size num_rows x num_cols to p processes.
   // A is stored as one-dimensional array of size num_rows x num_cols.
   // A_local is of size num_local_rows x num_cols
   int *displacements, *counts;
   if (myrank == 0) {
      displacements = (int *)malloc(sizeof(int) * p);
      counts = (int *)malloc(sizeof(int) * p);
      counts[0] = NUM_LOCAL_ROWS(0, p, num_rows) * num_cols;
      displacements[0] = 0;
      for (int i = 1; i < p; i++) {
         displacements[i] = displacements[i - 1] + counts[i - 1];
         counts[i] = NUM_LOCAL_ROWS(i, p, num_rows) * num_cols;
      }
   }
    keywordstyleMPI_Scatterv(A, counts, displacements,
           keywordstyleMPI_DOUBLE, A_local, num_local_rows * num_cols,
           keywordstyleMPI_DOUBLE, 0,
             keywordstyleMPI_COMM_WORLD);
```

Example: parallel $A \times b$

Gathering y

```
    if (myrank == 0) {
       for (int i = 0; i < p; i++)
          counts[i] = NUM_LOCAL_ROWS(i, p, num_rows);
       for (int i = 1; i < p; i++)
          displacements[i] = displacements[i - 1] + counts[i - 1];
    }
     keywordstyleMPI_Gatherv(y_local, num_local_rows,
            keywordstyleMPI_DOUBLE, y, counts, displacements,
            keywordstyleMPI_DOUBLE, 0,  keywordstyleMPI_COMM_WORLD);
```