ACM/CS 114
Parallel algorithms for scientific applications

Michael A. G. Aïvázis

California Institute of Technology

Winter 2012

Timing

- the function

```
1 double MPI_Wtime();
```

  returns the time in seconds from some arbitrary time in the past
  - guaranteed not to change only for the duration of the process
- you can compute the elapsed time for any program segment by making calls at the beginning and the end and computing the difference
- no guarantees about synchronized clocks among different processes
- you can compute the clock resolution by using

```
1 double MPI_Wtick();
```

Other collective operations

- `MPI_Scan` computes partial reductions: the p^{th} process receives the result from processes 0 through $p - 1$

```
int MPI_Scan(
      void* send_buffer, void* recv_buffer,
      int count, MPI_Datatype datatype, MPI_Op operation,
      MPI_Comm communicator
      );
```

- `MPI_Reduce` collects the result at only the given process `root`

```
int MPI_Reduce(
      void* send_buffer, void* recv_buffer,
      int count, MPI_Datatype datatype, MPI_Op operation,
      int root, MPI_Comm communicator
      );
```

- synchronization is also a global operation:

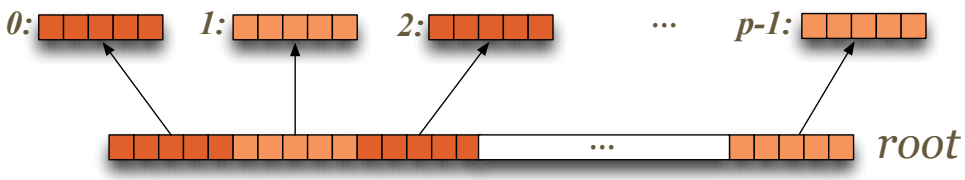
```
int MPI_Barrier(MPI_Comm communicator);
```

  participating processes block at a barrier until they have all reached it

Scatter

- `MPI_Scatter` sends data from `root` to all processes

```
int MPI_Scatter(
      void* send_buffer, int send_count, MPI_Datatype send_datatype,
      void* recv_buffer, int recv_count, MPI_Datatype recv_datatype,
      int root, MPI_Comm communicator
      );
```

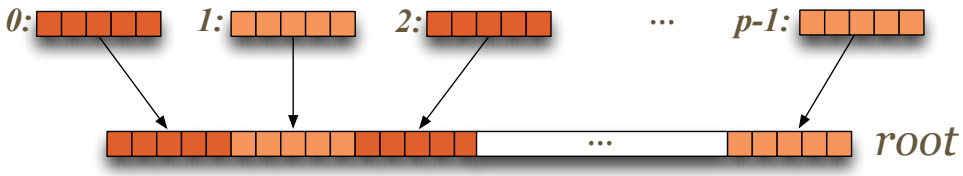


- it is as if the data in `send_buffer` were split in p segments, and the i^{th} process receives the i^{th} segment
- the `send_xxx` arguments are only meaningful for `root`; they are ignored for other processes
- the arguments `root` and `communicator` must be passed identical values by all processes

Gather

- the converse is `MPI_Gather` with `root` receiving data from all processes

```
1 int MPI_Gather(
2       void* send_buffer, int send_count, MPI_Datatype send_datatype,
3       void* recv_buffer, int recv_count, MPI_Datatype recv_datatype,
4       int root, MPI_Comm communicator
5       );
```



- it is as if p messages, one from each processes, were concatenated in rank order and placed at `recv_buffer`
- the `recv_xxx` arguments are only meaningful for `root`; they are ignored for other processes
- the arguments `root` and `communicator` must be passed identical values by all processes

Broadcasting operations

- `MPI_Alltoall` sends data from all processes to all processes in a global scatter/gather

```
int MPI_Alltoall(
      void* send_buffer, int send_count, MPI_Datatype send_datatype,
      void* recv_buffer, int recv_count, MPI_Datatype recv_datatype,
      MPI_Comm communicator
      );
```

- use `MPI_Bcast` to send the contents of a buffer from `root` to all processes in a communicator

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

Data movement patterns for the collective operations