NHR FAU

FAU Friedrich-Alexander-Universität Erlangen-Nürnberg

Introduction to Parallel Programming with MPI

Dr. Alireza Ghasemi, Dr. Georg Hager

Erlangen National High Performance Computing Center

Collective Operations

Motivation

Dot product: basic reduction using MPI collectives

The inner product of two vectors: $\vec{a}.\vec{b} = \sum_{i=1}^{N} a_i\, b_i$

It can be calculated with the following loop in C programming language:

```
dot=0.0;
for(int i=0;i<N;i++) dot+=a[i]*b[i];
```

$N_l = N/N_{procs}$ →

```
dot=0.0;
for(int i=0;i<Nl;i++) dot+=a[i]*b[i];
```

Motivation

Dot product: basic reduction using MPI collectives

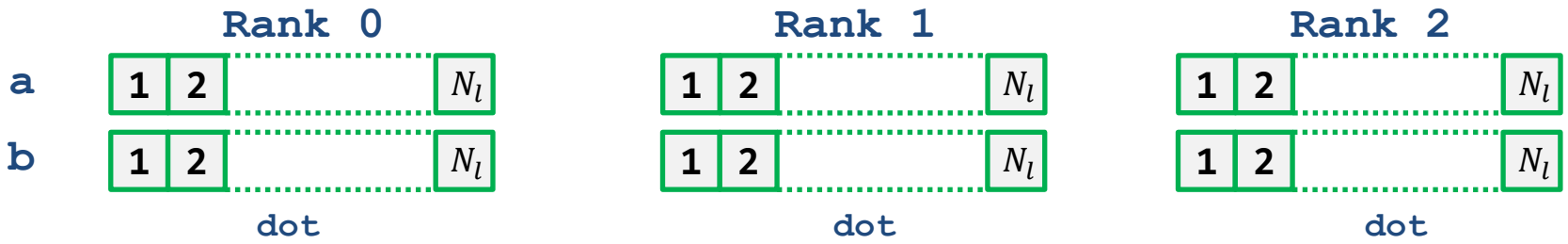
The inner product of two vectors: $\vec{a}.\vec{b} = \sum_{i=1}^{N} a_i\, b_i$

It can be calculated with the following loop in C programming language:

```
dot=0.0;
for(int i=0;i<N;i++) dot+=a[i]*b[i];
```

$N_l = N/N_{procs}$ →

```
dot=0.0;
for(int i=0;i<Nl;i++) dot+=a[i]*b[i];
```



Gathering **dots** onto rank 0 in **dot_arr**, then `dot_arr[0]+dot_arr[1]+dot_arr[2]`

Is there a better solution?

Global operations: reduction

- Compute results over distributed data

```
MPI_Reduce(sendbuf, recvbuf, count,
           datatype, MPI_Op op, root, comm)
```

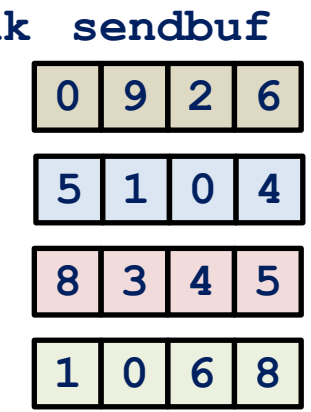
- Result in `recvbuf` only available on root process
- Perform operation on all `count` elements of an array

Global operations: reduction

- Compute results over distributed data

```
MPI_Reduce(sendbuf, recvbuf, count,
           datatype, MPI_Op op, root, comm)
```

- Result in `recvbuf` only available on root process
- Perform operation on all `count` elements of an array

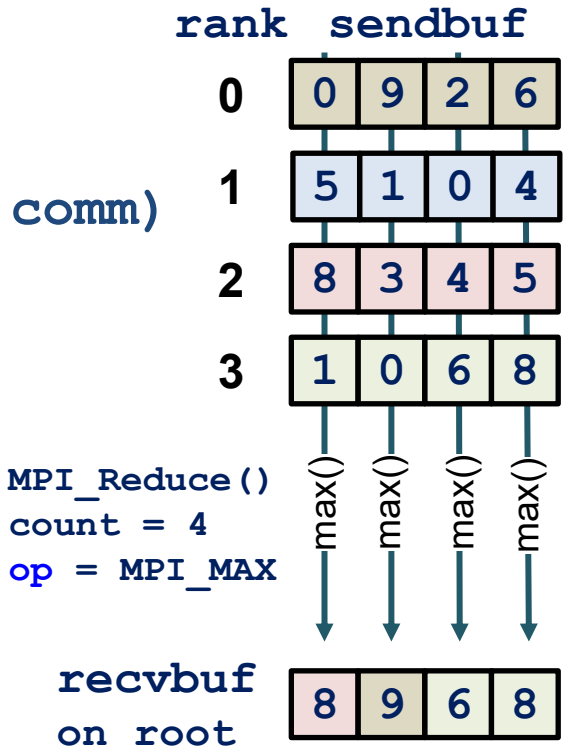


Global operations: reduction

- Compute results over distributed data

```
MPI_Reduce(sendbuf, recvbuf, count,
           datatype, MPI_Op op, root, comm)
```

- Result in `recvbuf` only available on root process
- Perform operation on all `count` elements of an array

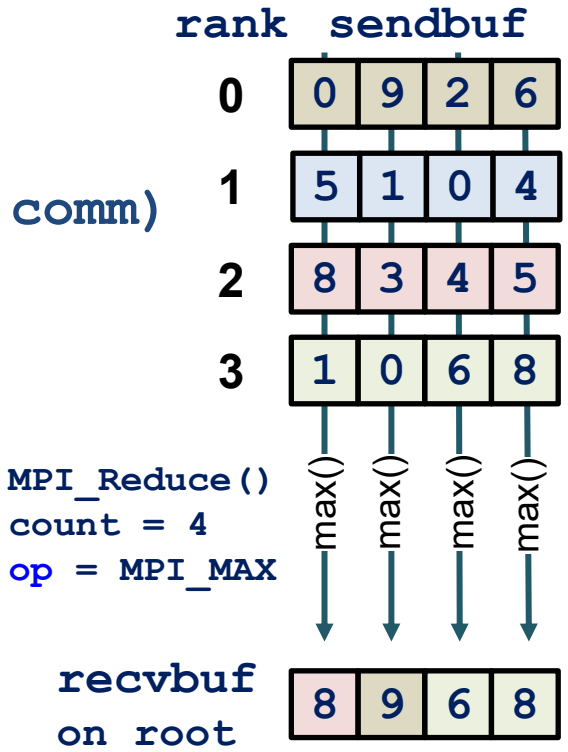


Global operations: reduction

- Compute results over distributed data

```
MPI_Reduce(sendbuf, recvbuf, count,
           datatype, MPI_Op op, root, comm)
```

- Result in `recvbuf` only available on root process
- Perform operation on all `count` elements of an array
- If all ranks need the result, then use `MPI_Allreduce()`
- If the 12 predefined ops are not enough use `MPI_Op_create`/`MPI_Op_free` to create own ones



Global operations – predefined operators

Name	Operation	Name	Operation
`MPI_SUM`	Sum	`MPI_PROD`	Product
`MPI_MAX`	Maximum	`MPI_MIN`	Minimum
`MPI_LAND`	Logical AND	`MPI_BAND`	Bit-AND
`MPI_LOR`	Logical OR	`MPI_BOR`	Bit-OR
`MPI_LXOR`	Logical XOR	`MPI_BXOR`	Bit-XOR
`MPI_MAXLOC`	Maximum+Position	`MPI_MINLOC`	Minimum+Position

- Define own operations with `MPI_Op_create`/`MPI_Op_free`
- MPI assumes that the operations are associative
  → be careful with floating-point operations

“In-place” buffer specification

Override local input buffer with a result

“In-place” buffer specification

Override local input buffer with a result

MPI_Reduce

```
int partial_sum = …, total_sum;
MPI_Reduce(&partial_sum, &total_sum,
           1, MPI_INT,
           MPI_SUM, root, comm);
```

```
int partial_sum = …, total_sum;
if (rank == root) {
  total_sum = partial_sum;
  MPI_Reduce(MPI_IN_PLACE, &total_sum,
             1, MPI_INT,
             MPI_SUM, root, comm);
}
else {
  MPI_Reduce(&partial_sum, &total_sum,
             1, MPI_INT,
             MPI_SUM, root, comm);
}
```

“In-place” buffer specification

Override local input buffer with a result

MPI_Reduce

```
int partial_sum = …, total_sum;
MPI_Reduce(&partial_sum, &total_sum,
           1, MPI_INT,
           MPI_SUM, root, comm);
```

```
int partial_sum = …, total_sum;
if (rank == root) {
  total_sum = partial_sum;
  MPI_Reduce(MPI_IN_PLACE, &total_sum,
             1, MPI_INT,
             MPI_SUM, root, comm);
}
else {
  MPI_Reduce(&partial_sum, &total_sum,
             1, MPI_INT,
             MPI_SUM, root, comm);
}
```

MPI_Allreduce

```
int partial_sum = …, total_sum;
MPI_AllReduce(&partial_sum, &total_sum,
              1, MPI_INT,
              MPI_SUM, comm);
```

```
int partial_sum = …, total_sum;

total_sum = partial_sum;
MPI_AllReduce(MPI_IN_PLACE, &total_sum,
              1, MPI_INT,
              MPI_SUM, comm);
```

MPI_IN_PLACE cheat sheet

Function	MPI_IN_PLACE argument	@ rank(s)	Comment [MPI 3.0]
MPI_GATHER	send buffer	root	Recv value at root already in the correct place in receive buffer.
MPI_GATHERV	send buffer	root	Recv value at root already in the correct place in receive buffer.
MPI_SCATTER	receive buffer	root	Root-th segment of send buffer is not moved.
MPI_SCATTERV	receive buffer	root	Root-th segment of send buffer is not moved.
MPI_ALLGATHER	send buffer	all	Input data at the correct place were process would receive its own contribution.
MPI_ALLGATHERV	send buffer	all	Input data at the correct place were process would receive its own contribution.
MPI_ALLTOALL	send buffer	all	Data to be sent is taken from receive buffer and replaced by received data, data sent/received must be of the same type map specified in receive count/receive type.
MPI_ALLTOALLV	send buffer	all	Data to be sent is taken from receive buffer and replaced by received data. Data sent/received must be of the same type map specified in receive count/receive type. The same amount of data and data type is exchanged between two processes.
MPI_REDUCE	send buffer	root	Data taken from receive buffer, replaced with output data.
MPI_ALLREDUCE	send buffer	all	Data taken from receive buffer, replaced with output data.

Summary of MPI collective communication

- MPI (blocking) collectives
  - All ranks in communicator must call the function
- Communication and synchronization
  - Barrier, broadcast, scatter, gather, and combinations thereof
- Global operations
  - Reduce, allreduce, some more…
- In-place buffer specification `MPI_IN_PLACE`
  - Save some space if needed

Quiz:

1. Do you recommend an application developer to use `MPI_Gather` followed by operations on the `root` process rather than using `MPI_Reduce`? Why?

   a. Yes

   b. No

Quiz:

1. Do you recommend an application developer to use **`MPI_Gather`** followed by operations on the **`root`** process rather than using **`MPI_Reduce`**? Why?

   a. Yes b. No

Answer: b., because it is very likely that it is slower than **`MPI_Reduce`**.

Quiz:

1. Do you recommend an application developer to use **`MPI_Gather`** followed by operations on the **`root`** process rather than using **`MPI_Reduce`**? Why?

   a. Yes b. No

Answer: b., because it is very likely that it is slower than **`MPI_Reduce`**.

2. Is the argument **`recvbuf`** in **`MPI_Reduce`** significant on every process calling it?

   a. Yes b. No

Quiz:

1. Do you recommend an application developer to use **`MPI_Gather`** followed by operations on the **`root`** process rather than using **`MPI_Reduce`**? Why?

   a. Yes b. No

Answer: b., because it is very likely that it is slower than **`MPI_Reduce`**.

2. Is the argument **`recvbuf`** in **`MPI_Reduce`** significant on every process calling it?

   a. Yes b. No

Answer: b., it is significant only at root.