Parallel and Distributed Computing: MPI part 1

Erik Saule
esaule@uncc.edu

Parallel and Distributed Computing

Outline

Learning Outcomes

After this lecture you will be able to

- Explain why MPI was a useful standard in developing HPC systems
- Write a basic Hello World distributed application in MPI
- Use collectives

The assignment will make you

- Write an MPI Hello World
- Use collective to synchronize basic code
- Use communicators and collective to solve a complex problem.

Sources

Most of the slides in this deck are taken from

MPI for Dummies

Tutorial on MPI programming, Foundations
Victor Eijkhout `eijkhout@tacc.utexas.edu`
TACC training, 2017

Pavan Balaji
Computer Scientist
Argonne National Laboratory
Email: balaji@mcs.anl.gov
Web: http://www.mcs.anl.gov/~balaji

Torsten Hoefler
Assistant Professor
ETH Zurich
Email: htor@inf.ethz.ch
Web: http://www.unixer.de/

link to slides at the end of this deck

Outline

Standardizing Message-Passing Models with MPI

- Early vendor systems (Intel's NX, IBM's EUI, TMC's CMMD) were not portable (or very capable)
- Early portable systems (PVM, p4, TCGMSG, Chameleon) were mainly research efforts
  - Did not address the full spectrum of message-passing issues
  - Lacked vendor support
  - Were not implemented at the most efficient level
- The MPI Forum was a collection of vendors, portability writers and users that wanted to standardize all these efforts

What is MPI?

- MPI: Message Passing Interface
  - The MPI Forum organized in 1992 with broad participation by:
    - Vendors: IBM, Intel, TMC, SGI, Convex, Meiko
    - Portability library writers: PVM, p4
    - Users: application scientists and library writers
    - MPI-1 finished in 18 months
  - Incorporates the best ideas in a "standard" way
    - Each function takes fixed arguments
    - Each function has fixed semantics
      - Standardizes what the MPI implementation provides and what the application can and cannot expect
      - Each system can implement it differently as long as the semantics match
- MPI is not…
  - a language or compiler specification
  - a specific implementation or product

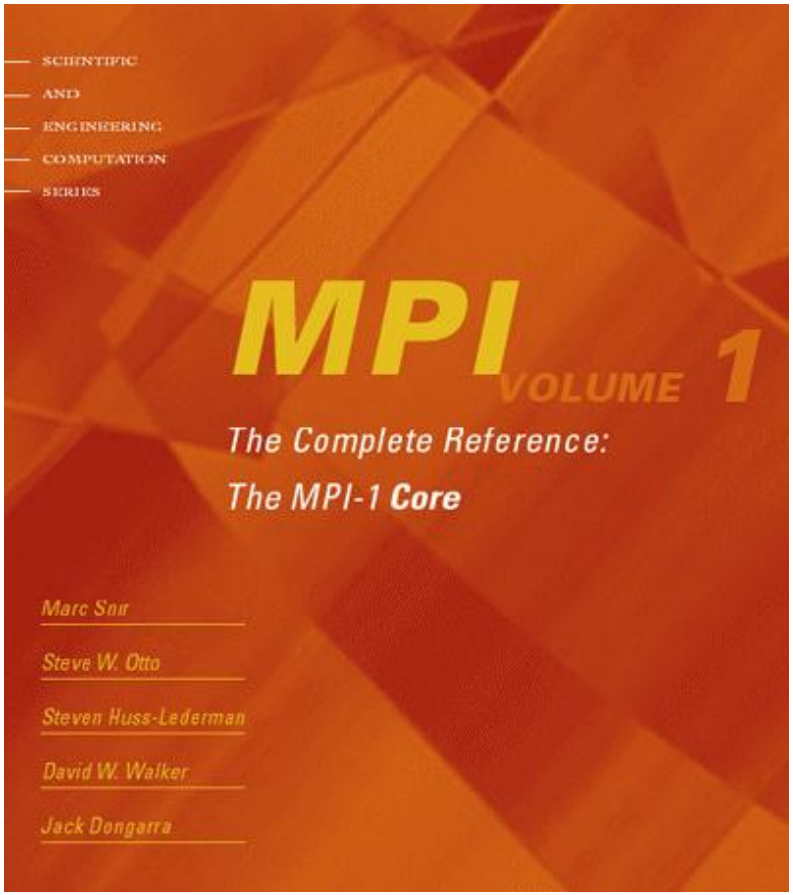
What is in MPI-1

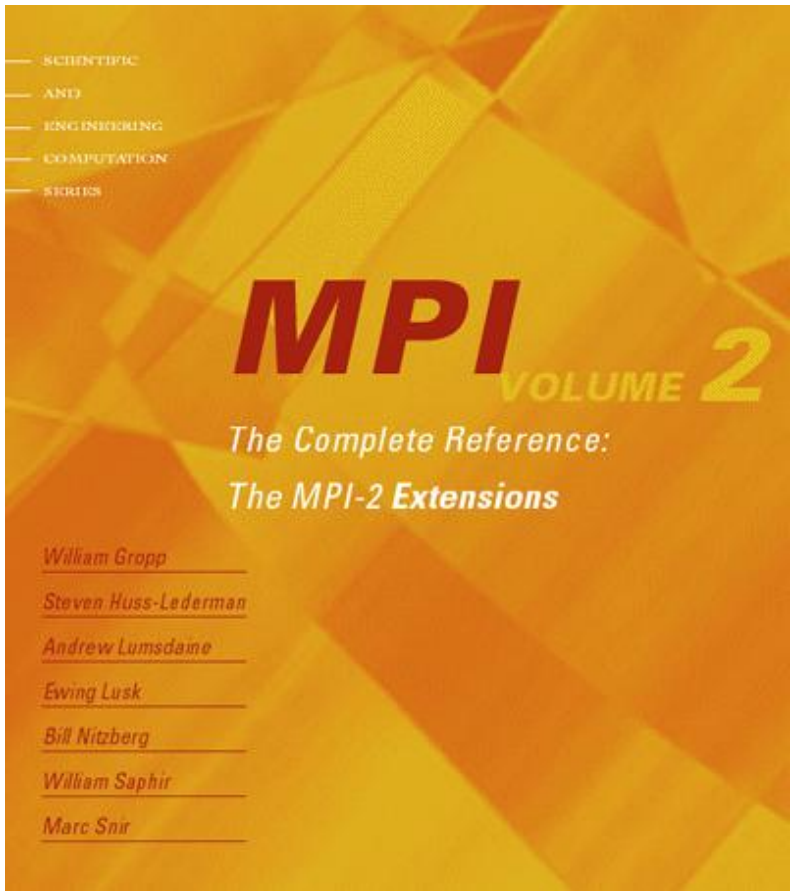
- Basic functions for communication (100+ functions)
- Blocking sends, receives
- Nonblocking sends and receives
- Variants of above
- Rich set of collective communication functions
  - Broadcast, scatter, gather, etc
  - Very important for performance; widely used
- Datatypes to describe data layout
- Process topologies
- C, C++ and Fortran bindings
- Error codes and classes

Following MPI Standards

- MPI-2 was released in 2000
  - Several additional features including MPI + threads, MPI-I/O, remote memory access functionality and many others
- MPI-2.1 (2008) and MPI-2.2 (2009) were recently released with some corrections to the standard and small features
- MPI-3 (2012) added several new features to MPI
- The Standard itself:
  - at http://www.mpi-forum.org
  - All MPI official releases, in both postscript and HTML
- Other information on Web:
  - at http://www.mcs.anl.gov/mpi
  - pointers to lots of material including tutorials, a FAQ, other MPI pages

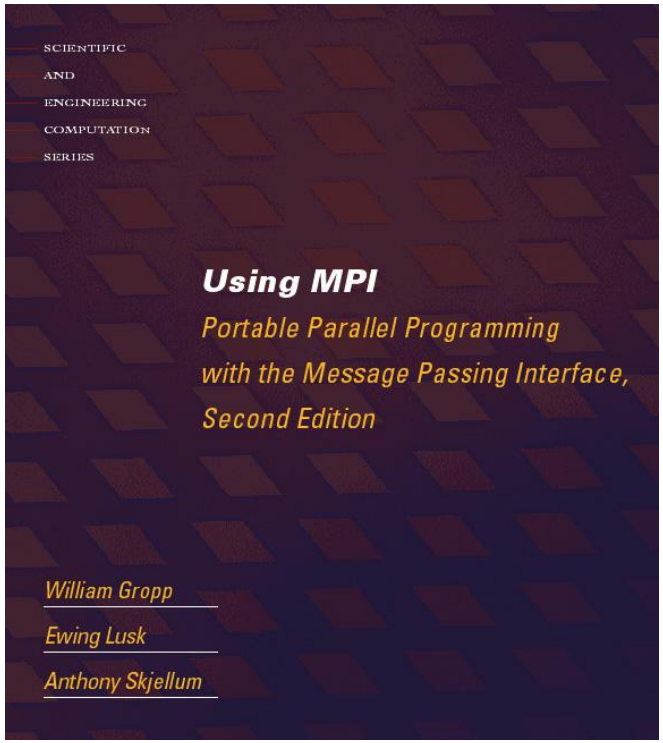
The MPI Standard (1 & 2)

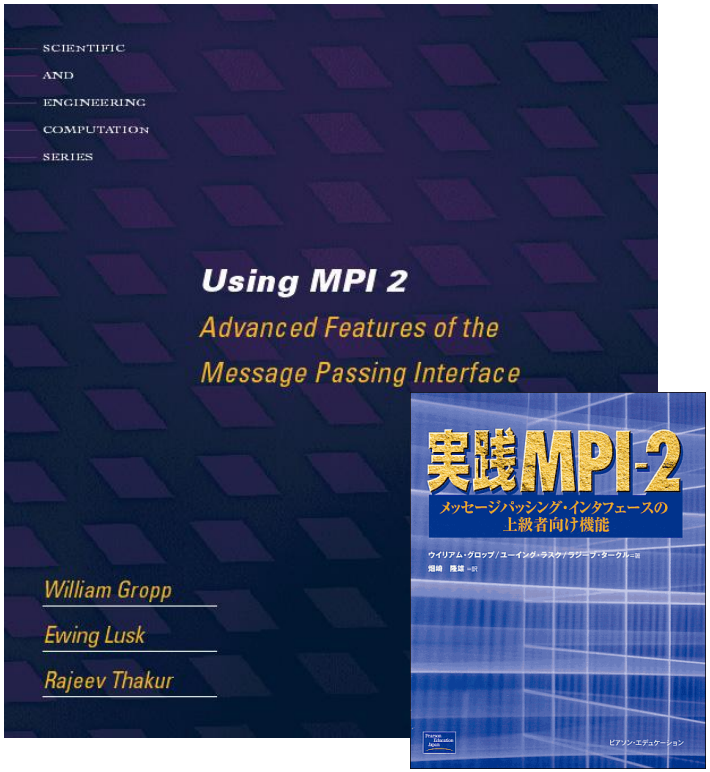




Pavan Balaji and Torsten Hoefler, PPoPP, Shenzhen, China (02/24/2013)

Tutorial Material on MPI-1 and MPI-2





http://www.mcs.anl.gov/mpi/usingmpi
http://www.mcs.anl.gov/mpi/usingmpi2

These books available through the Atkins library, see end slide.

Reasons for Using MPI

- **Standardization** - MPI is the only message passing library which can be considered a standard. It is supported on virtually all HPC platforms. Practically, it has replaced all previous message passing libraries
- **Portability** - There is no need to modify your source code when you port your application to a different platform that supports (and is compliant with) the MPI standard
- **Performance Opportunities** - Vendor implementations should be able to exploit native hardware features to optimize performance
- **Functionality** – Rich set of features
- **Availability** - A variety of implementations are available, both vendor and public domain
  - MPICH is a popular open-source and free implementation of MPI
  - Vendors and other collaborators take MPICH and add support for their systems
    - Intel MPI, IBM Blue Gene MPI, Cray MPI, Microsoft MPI, MVAPICH, MPICH-MX

Pavan Balaji and Torsten Hoefler, PPoPP, Shenzhen, China (02/24/2013)

Outline

Compiling and running

MPI compilers are usually called `mpicc`, `mpif90`, `mpicxx`.

These are not separate compilers, but scripts around the regular C/Fortran compiler. You can use all the usual flags.

Run your program with something like

```
mpiexec -n 4 hostfile ... yourprogram arguments
mpirun -np 4 hostfile ... yourprogram arguments
```

At TACC:

`ibrun yourprog`

the number of processes is determined by SLURM.

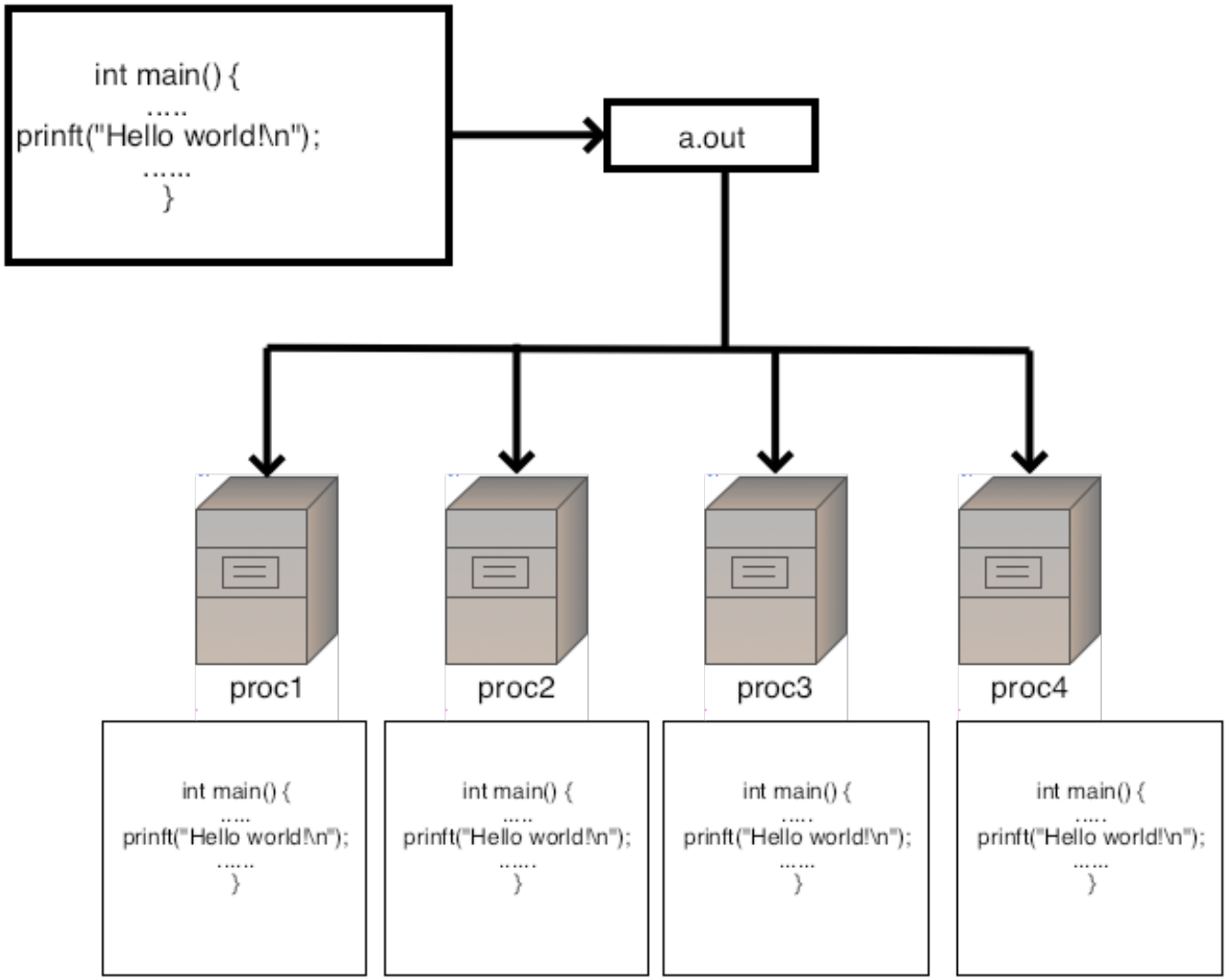
Do I need a supercomputer?

- With `mpiexec` and such, you start a bunch of processes that execute your MPI program.
- Does that mean that you need a cluster or a big multicore?
- No! You can start a large number of MPI processes, even on your laptop. The OS will use 'time slicing'.
- Of course it will not be very efficient...

How does MPI know where to run

On mamba, MPI will contact the batch scheduler (SLURM) to know where the process should run.

In a picture


int main() {
.....
prinft("Hello world!\n");
.....
}
a.out
proc1
proc2
proc3
proc4
int main() {
.....
prinft("Hello world!\n");
.....
}
int main() {
.....
prinft("Hello world!\n");
.....
}
int main() {
.....
prinft("Hello world!\n");
.....
}
int main() {
.....
prinft("Hello world!\n");
.....
}

Simple MPI Program Identifying Processes

```
#include <mpi.h>
#include <stdio.h>

int main(int argc, char ** argv)
{
    int rank, size;

    MPI_Init(&argc, &argv);

    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    printf("I am %d of %d\n", rank, size);

    MPI_Finalize();
    return 0;
}
```

Basic requirements for an MPI program

Outline

Collectives

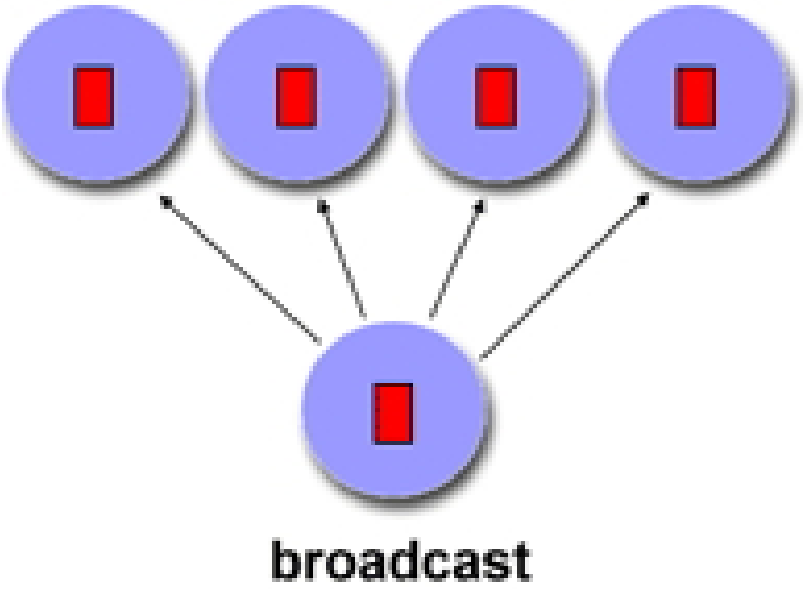
Gathering and spreading information:

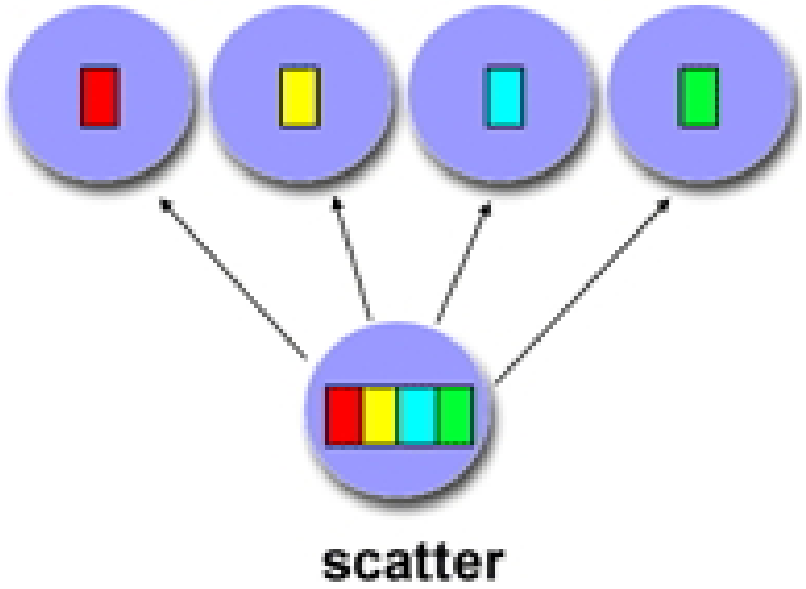
- Every process has data, you want to bring it together;
- One process has data, you want to spread it around.

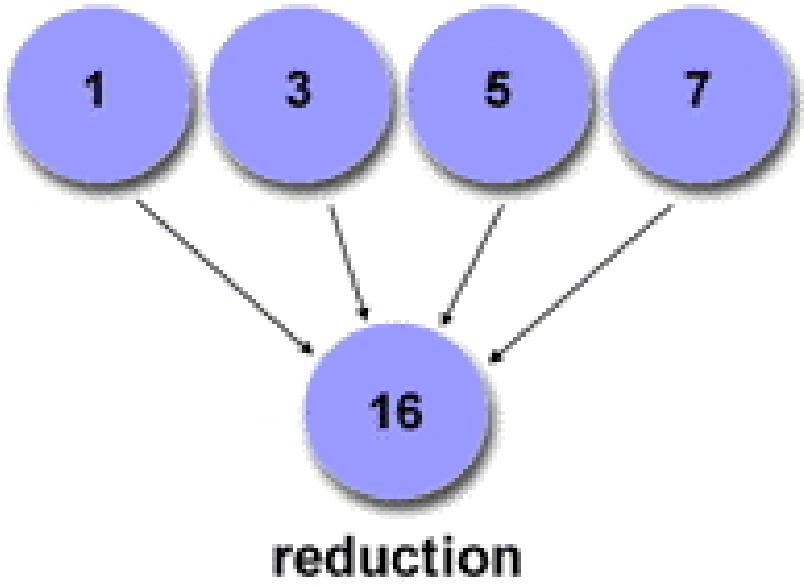
Root process: the one doing the collecting or disseminating.

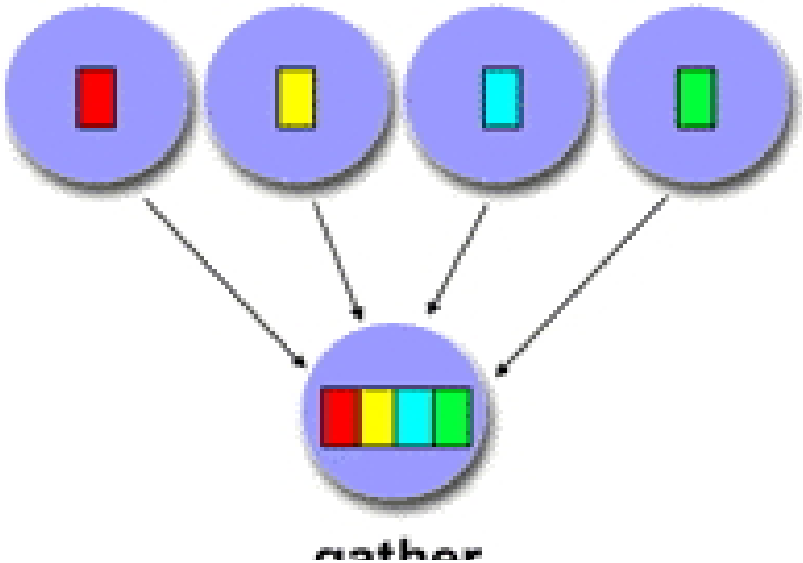
Basic cases:

- Collect data: gather.
- Collect data and compute some overall value (sum, max): reduction.
- Send the same data to everyone: broadcast.
- Send individual data to each process: scatter.


broadcast


scatter


1
3
5
7
16
reduction



MPI Collective Routines

- Many Routines: `MPI_ALLGATHER`, `MPI_ALLGATHERV`, `MPI_ALLREDUCE`, `MPI_ALLTOALL`, `MPI_ALLTOALLV`, `MPI_BCAST`, `MPI_GATHER`, `MPI_GATHERV`, `MPI_REDUCE`, `MPI_REDUCESCATTER`, `MPI_SCAN`, `MPI_SCATTER`, `MPI_SCATTERV`
- "`All`" versions deliver results to all participating processes
- "`V`" versions (stands for vector) allow the hunks to have different sizes
- `MPI_ALLREDUCE`, `MPI_REDUCE`, `MPI_REDUCESCATTER`, and `MPI_SCAN` take both built-in and user-defined combiner functions

Example: PI in C (let us try writing this)

```
#include <mpi.h>
#include <math.h>
int main(int argc, char *argv[])
{
    [...snip...]
    /* Tell all processes, the number of segments you want */
    MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);
    w   = 1.0 / (double) n;
    mypi = 0.0;
    for (i = myid + 1; i <= n; i += numprocs)
        mypi += w * sqrt(1 - (((double) i / n) * ((double) i / n));
    MPI_Reduce(&mypi, &pi, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
    if (myid == 0)
        printf("pi is approximately %.16f, Error is %.16f\n", 4 * pi,
               fabs(pi - PI25DT));
    [...snip...]
}
```

Pavan Balaji and Torsten Hoefler, PPoPP, Shenzhen, China (02/24/2013)

Elementary datatypes

C	Fortran	meaning
MPI_CHAR	MPI_CHARACTER	
MPI_SHORT	MPI_BYTE	
MPI_INT	MPI_INTEGER	
MPI_FLOAT	MPI_REAL	
MPI_DOUBLE	MPI_DOUBLE_PRECISION	
	MPI_COMPLEX	
	MPI_LOGICAL	
		optional
	MPI_REAL4	
	MPI_REAL8	

A bunch more.

MPI Built-in Collective Computation Operations

MPI_MAX	Maximum
MPI_MIN	Minimum
MPI_PROD	Product
MPI_SUM	Sum
MPI_LAND	Logical and
MPI_LOR	Logical or
MPI_LXOR	Logical exclusive or
MPI_BAND	Bitwise and
MPI_BOR	Bitwise or
MPI_BXOR	Bitwise exclusive or
MPI_MAXLOC	Maximum and location
MPI_MINLOC	Minimum and location

How to use collectives

Really check the manual for details. For instance:
`https://www.mpich.org/static/docs/v3.1/www3/MPI_Bcast.html`

Reduction

```
int MPI_Reduce
  (void *sendbuf, void *recvbuf,
   int count, MPI_Datatype datatype,
   MPI_Op op, int root, MPI_Comm comm)
```

- Buffers: `sendbuf`, `recvbuf` are ordinary variables/arrays.
- Every process has data in its `sendbuf`,
  Root combines it in `recvbuf` (ignored on non-root processes).
- `count` is number of items in the buffer: 1 for scalar.
- `MPI_Op` is `MPI_SUM`, `MPI_MAX` et cetera.

Broadcast

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

- All processes call with the same argument list
- `root` is the rank of the process doing the broadcast
- Each process allocates buffer space;
  root explicitly fills in values,
  all others receive values through broadcast call.
- Datatype is `MPI_FLOAT`, `MPI_INT` et cetera, different between C/Fortran.
- `comm` is usually `MPI_COMM_WORLD`

Gather/Scatter

```
int MPI_Gather(
  void *sendbuf, int sendcnt, MPI_Datatype sendtype,
  void *recvbuf, int recvcnt, MPI_Datatype recvtype,
  int root, MPI_Comm comm
);
int MPI_Scatter
  (void* sendbuf, int sendcount, MPI_Datatype sendtype,
   void* recvbuf, int recvcount, MPI_Datatype recvtype,
   int root, MPI_Comm comm)
```

- Compare buffers to ▸ reduce
- Scatter: the `sendcount` / Gather: the `recvcount`:
  this is not, as you might expect, the total length of the buffer; instead, it is the amount of data to/from each process.

Also: `MPI_Allgather`

Communicators

Note that the collective only enable to communicate with all the processes of the communicator.
The default communicator MPI_COMM_WORLD contains all the processes that were started.
One can create new custom communicators with MPI_Comm_spawn, or MPI_Comm_create.
But the easiest way is using MPI_Comm_split. (see code.)

Outline

Problem 1: Hello World

Just print "I am node 2 out of 10. I run on machine.uncc.edu"
A basic use of

- `MPI_Comm_rank` to get your process ID
- `MPI_Comm_size` to get the total number of processes
- `gethostname` to get the name of the machine

Problem 2: Numerical Integration

Each process get a fixed interval of points to integrate like in a static schedule.
Use of

- `MPI_Comm_rank`, `MPI_Comm_size` to know what to integrate
- `MPI_Allreduce` to accomplish the reduction

Problem 3: Iterated Dense Matrix Vector multiplication

For a matrix A, a vector x_0, and a integer k; compute x_k such that

$$x_{i+1} = Ax_i$$

There is a data partitioning problem. (Assume the number of processes is a square number.) Partition A as a grid of size $\sqrt{P} \times \sqrt{P}$.
Get the values of A and x_0 through the scaffolded functions.
After doing the local multiplication of part of x with part of A, you'll need to reduce along the lines of processes.
To do the next iteration of the calculation, you'll need to broadcast the next value of the x along the column of processes.
To do the collective, processes will need a row communicator composed of the process on the same row. And also a column communicator composed of the process on the same column.
Use of

- `MPI_Bcast` to broadcast the next x
- `MPI_Reduce` to compute the result of the multiplication
- `MPI_Comm_split` to build column and row communicators

Outline

External

Books:

- Using MPI, 3rd edition. William Gropp, Ewing Lusk and Anthony Skjellum. MIT Press. Available through the library at https://librarylink.uncc.edu/login?url=http://ieeexplore.ieee.org/xpl/bkabstractplus.jsp?bkn=6981847
- Using Advanced MPI. William Gropp, Torsten Hoefler, Rajeev Thakur and Ewing Lusk. MIT Press. Available through the library at https://librarylink.uncc.edu/login?url=http://ieeexplore.ieee.org/xpl/bkabstractplus.jsp?bkn=6981848

MPI implementations:

- MPICH https://www.mpich.org
- OpenMPI https://www.open-mpi.org/

API Documentation:

- MPICH man pages https://www.mpich.org/static/docs/v3.2/www3/index.htm
- OpenMPI documentation https://www.open-mpi.org/doc/v3.0/

Slides I used:

- Tutorial on MPI programming. Victor Eijkhout. https://bitbucket.org/VictorEijkhout/parallel-computing-book/raw/e11748c8d8ae874ed645566ba0e82aa787ecf959/EijkhoutMPIlecture.pdf
- MPI for Dummies. Pavan Balaki, Torsten Hoefler. https://htor.inf.ethz.ch/teaching/mpi_tutorials/ppopp13/2013-02-24-ppopp-mpi-basic.pdf

Tutorial:

- part one of Parallel Programming in MPI and OpenMP. Victor Eijkhout. Draft at https://bitbucket.org/VictorEijkhout/parallel-computing-book/raw/e11748c8d8ae874ed645566ba0e82aa787ecf959/EijkhoutParComp.pdf