Basics of MPI

1. Message Passing Interface
   - one program to code manager/worker model
   - hello world!
   - broadcasting an integer
2. Moving Data from Manager to Workers
   - broadcasting an array of doubles
   - code to broadcast an array of doubles
3. MPI for Python
   - bindings of MPI for Python
   - hello world with mpi4py
   - broadcasting data
   - MPI wrappers for Julia

MCS 572 Lecture 4
Introduction to Supercomputing
Jan Verschelde, 4 September 2024

Basics of MPI

processors and processes

A parallel program is a collection of concurrent processes.
A process (also called a job or task) is a sequence of instructions.

Usually, there is a 1-to-1 map between processes and processors.
If there are more processes than processors,
then processes are executed in a time sharing environment.

We use the SPMD model: Single Program, Multiple Data.

Every node executes the same program.
Every node has a unique identification number (id)
— the root node has number zero —
and code can be executed depending on the id.

The root node is the manager, the other nodes are workers.

MPI = Message Passing Interface

MPI = Message Passing Interface
is a standard specification for interprocess communication
for which several implementations exist.

Start a C program with

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

to use the functionality of MPI.

`Open MPI` is an open source implementation `www.open-mpi.org`
of all features of the MPI-3.1 standard.

In this lecture we use MPI in simple interactive programs, e.g.:
as `mpicc` and `mpirun` are available on laptop computers.

Basics of MPI

running `mpi_hello_world`

We use a `makefile` to compile, and then run with 10 processes:

```
$ make mpi_hello_world
mpicc mpi_hello_world.c -o /tmp/mpi_hello_world

$ mpirun -np 10 /tmp/mpi_hello_world
Hello world from processor 2 out of 10.
Hello world from processor 8 out of 10.
Hello world from processor 0 out of 10.
Hello world from processor 1 out of 10.
Hello world from processor 3 out of 10.
Hello world from processor 4 out of 10.
Hello world from processor 5 out of 10.
Hello world from processor 6 out of 10.
Hello world from processor 9 out of 10.
Hello world from processor 7 out of 10.
$
```

mpi_hello_world.c

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

int main ( int argc, char *argv[] )
{
   int i,p;

   MPI_Init(&argc,&argv);
   MPI_Comm_size(MPI_COMM_WORLD,&p);
   MPI_Comm_rank(MPI_COMM_WORLD,&i);

   printf("Hello world from processor %d out of %d.\n",
          i,p);

   MPI_Finalize();

   return 0;
}
```

initializing and cleaning up

```
#include <mpi.h>

int main ( int argc, char *argv[] )
{
   MPI_Init(&argc,&argv);
   MPI_Finalize();
   return 0;
}
```

The `MPI_Init` processes command line arguments:

1. `argc` is the number of arguments,
2. `argv` contains the arguments,
   `argv[0]` is the name of the program.

`MPI_Finalize()` cleans up the environment.

the universe

`MPI_COMM_WORLD` is a predefined named constant handle to refer to the universe of p processors with labels from 0 to $p - 1$.

- `MPI_Comm_size` returns the number of processors.
- `MPI_Comm_rank` returns the label of a node.

For example:

```
int i,p;

MPI_Comm_size(MPI_COMM_WORLD,&p);
MPI_Comm_rank(MPI_COMM_WORLD,&i);
```

Basics of MPI

1. Message Passing Interface
   - one program to code manager/worker model
   - hello world!
   - broadcasting an integer
2. Moving Data from Manager to Workers
   - broadcasting an array of doubles
   - code to broadcast an array of doubles
3. MPI for Python
   - bindings of MPI for Python
   - hello world with mpi4py
   - broadcasting data
   - MPI wrappers for Julia

broadcasting an integer

Many parallel programs follow a manager/worker model.

As an example, we broadcast an integer:

1. Node with id 0 (manager) prompts for an integer.
2. The integer is *broadcasted* over the network:
   $\rightarrow$ the number is sent to all processors in the universe.
3. Every worker node prints the number to screen.

Application: broadcast dimension of data before sending the data.

running the program

```
$ make broadcast_integer
mpicc broadcast_integer.c -o /tmp/broadcast_integer

$ mpirun -np 3 /tmp/broadcast_integer
Type an integer number...
123
Node 1 writes the number n = 123.
Node 2 writes the number n = 123.
$
```

MPI_Bcast

An example of the `MPI_Bcast` command:

```
int n;
MPI_Bcast(&n,1,MPI_INT,0,MPI_COMM_WORLD);
```

There are five arguments:

1. address of the element(s) to broadcast,
2. number of elements that will be broadcasted,
3. type of all the elements,
4. message label,
5. universe.

headers and subroutine declarations

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

void manager ( int *n );
/* code executed by the manager node 0,
 * prompts the user for an integer number n */

void worker ( int i, int n );
/* code executed by the i-th worker node,
 * who will write the integer number n to screen */
```

the main program

```
int main ( int argc, char *argv[] )
{
   int myid,numbprocs,n;

   MPI_Init(&argc,&argv);
   MPI_Comm_size(MPI_COMM_WORLD,&numbprocs);
   MPI_Comm_rank(MPI_COMM_WORLD,&myid);

   if (myid == 0) manager(&n);

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

   if (myid != 0) worker(myid,n);

   MPI_Finalize();

   return 0;
}
```

code for the subroutines

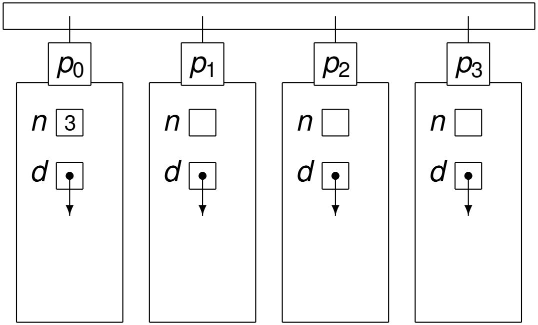
```
void manager ( int *n )
{
   printf("Type an integer number... \n");
   scanf("%d",n);
}

void worker ( int i, int n )
{
   printf("Node %d writes the number n = %d.\n",i,n);
}
```

Basics of MPI

1. Message Passing Interface
   - one program to code manager/worker model
   - hello world!
   - broadcasting an integer
2. Moving Data from Manager to Workers
   - broadcasting an array of doubles
   - code to broadcast an array of doubles
3. MPI for Python
   - bindings of MPI for Python
   - hello world with mpi4py
   - broadcasting data
   - MPI wrappers for Julia

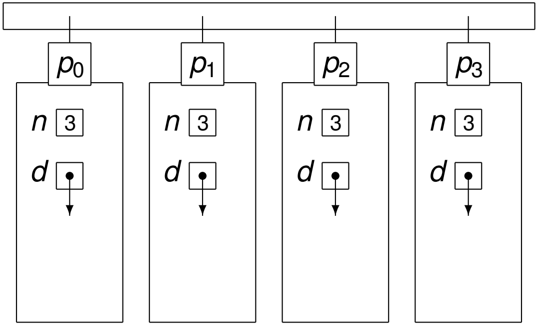
before broadcasting the dimension

Before broadcasting the dimension *n* to all nodes on a 4-processor distributed memory computer.



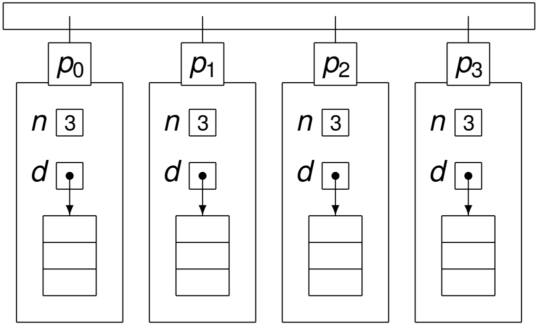
before array allocation

Before allocating an array of 3 doubles
on a 4-processor distributed memory computer.



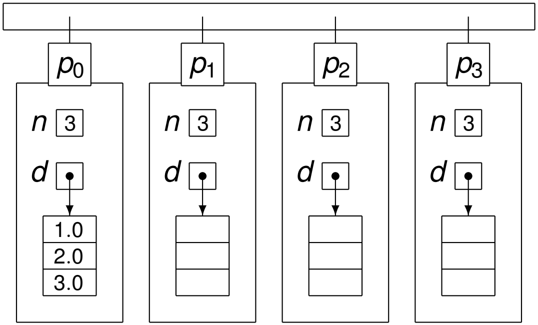
after array allocation

After allocating an array of 3 doubles
on a 4-processor distributed memory computer.



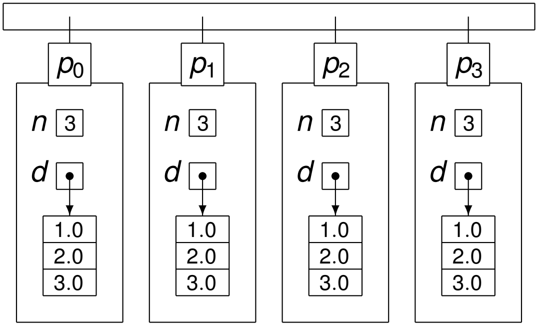
before the broadcast

Before broadcasting an array of 3 doubles
on a 4-processor distributed memory computer.



after the broadcast

After broadcasting an array of 3 doubles
on a 4-processor distributed memory computer.



Basics of MPI

1. Message Passing Interface
   - one program to code manager/worker model
   - hello world!
   - broadcasting an integer

2. Moving Data from Manager to Workers
   - broadcasting an array of doubles
   - code to broadcast an array of doubles

3. MPI for Python
   - bindings of MPI for Python
   - hello world with mpi4py
   - broadcasting data
   - MPI wrappers for Julia

headers and subroutine declarations

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

void define_doubles ( int n, double *d );
/* defines the values of the n doubles in d */

void write_doubles ( int myid, int n, double *d );
/* node with id equal to myid
   writes the n doubles in d */
```

We include `stdlib.h` for memory allocation.

broadcasting the dimension

```
int main ( int argc, char *argv[] )
{
   int myid,numbprocs,n;
   double *data;

   MPI_Init(&argc,&argv);
   MPI_Comm_size(MPI_COMM_WORLD,&numbprocs);
   MPI_Comm_rank(MPI_COMM_WORLD,&myid);

   if (myid == 0)
   {
      printf("Type the dimension ...\n");
      scanf("%d",&n);
   }
   MPI_Bcast(&n,1,MPI_INT,0,MPI_COMM_WORLD);
```

allocating memory

The main program continues:

```
   data = (double*)calloc(n,sizeof(double));
```

Every node must allocate memory!

```
   if (myid == 0) define_doubles(n,data);

   MPI_Bcast(data,n,MPI_DOUBLE,0,MPI_COMM_WORLD);

   if (myid != 0) write_doubles(myid,n,data);

   MPI_Finalize();
   return 0;
}
```

subroutine definitions

```
void define_doubles ( int n, double *d )
{
   int i;

   printf("defining %d doubles ...\n", n);
   for(i=0; i < n; i++) d[i] = (double)i;
}

void write_doubles ( int myid, int n, double *d )
{
   int i;

   printf("Node %d writes %d doubles : \n", myid,n);
   for(i=0; i < n; i++) printf("%lf\n",d[i]);
}
```

Basics of MPI

Message Passing Interface for Python

MPI for Python provides bindings of MPI for Python,
allowing any Python program to exploit multiple processors.

- object oriented interface follows closely MPI-2 C++ bindings;
- supports point-to-point and collective communications
  - of any pickable Python object,
  - as well as numpy arrays and builtin bytes, strings.

`mpi4py` gives the standard MPI “look and feel” in Python scripts
to develop parallel programs.

Often, only a small part of the code needs the efficiency of a compiled language. Python handles memory, errors, and user interaction.

Available at github, installs with pip,
the current version is 3.1.4, Nov 2, 2022.
L. Dalcin, R. Paz, and M. Storti: **MPI for Python.**
Journal of Parallel and Distributed Computing, 65(9):1108–1115, 2005.

Basics of MPI

hello world with mpi4py

```
from mpi4py import MPI

SIZE = MPI.COMM_WORLD.Get_size()
RANK = MPI.COMM_WORLD.Get_rank()
NAME = MPI.Get_processor_name()

MESSAGE = "Hello from %d of %d on %s." \
  % (RANK, SIZE, NAME)
print MESSAGE
```

running the script

Programs that run with MPI are executed with `mpiexec`.

To run `mpi4py_hello_world.py` by 3 processes:

```
$ mpiexec -n 3 python mpi4py_hello_world.py
Hello from 0 of 3 on ...
Hello from 2 of 3 on ...
Hello from 1 of 3 on ...
$
```

Three Python interpreters are launched.

Each interpreter executes the script,
printing the hello message.

some basic MPI concepts and commands

`MPI.COMM_WORLD` is a predefined intracommunicator.
An intracommunicator is a group of processes.
All processes within an intracommunicator have a unique number.

Methods of the intracommunicator `MPI.COMM_WORLD`:

- `Get_size()` returns the number of processes.
- `Get_rank()` returns rank of executing process.

Even though every process runs the same script,
the test `if MPI.COMM_WORLD.Get_rank() == i:`
allows to specify particular code for the *i*-th process.

`MPI.Get_processor_name()`
returns the name of the calling processor.

Basics of MPI

broadcasting data

A collective communication involves every process
in the intracommunicator.

A broadcast is a collective communication in which

- one process sends the same data to all processes,
- all processes receive the same data.

In `mpi4py`, a broadcast is done with the `bcast` method.

An example:

```
$ mpiexec -n 3 python mpi4py_broadcast.py
0 has data {'pi': 3.141592653589793l, 'e': 2.718281828459045l}
1 has data {'pi': 3.141592653589793l, 'e': 2.718281828459045l}
2 has data {'pi': 3.141592653589793l, 'e': 2.718281828459045l}
$
```

the script `mpi4py_broadcast.py`

```
from mpi4py import MPI

COMM = MPI.COMM_WORLD
RANK = COMM.Get_rank()

if(RANK == 0):
    DATA = {'e'  : 2.718281828459045,
            'pi' : 3.141592653589793 }
else:
    DATA = None # DATA must be defined

DATA = COMM.bcast(DATA, root=0)
print(RANK, 'has data', DATA)
```

Basics of MPI

1. Message Passing Interface
   - one program to code manager/worker model
   - hello world!
   - broadcasting an integer

2. Moving Data from Manager to Workers
   - broadcasting an array of doubles
   - code to broadcast an array of doubles

3. MPI for Python
   - bindings of MPI for Python
   - hello world with mpi4py
   - broadcasting data
   - MPI wrappers for Julia

MPI wrappers for Julia

`MPI.jl` is a Julia interface to MPI, inspired by mpi4py.

Available at `https://github.com/JuliaParallel`.

Its installation requires a shared binary installation
of a C MPI library, supporting the MPI 3.0 standard or later.

The MPI.jl is a Julia package, install as `using MPI`.

Simon Byrne, Lucas C. Wilcox, and Valentin Churavy:
MPI.jl: Julia bindings for the Message Passing Interface.
In *JuliaCon Proceedings*, 1(1), 68, 2021.

the Julia program `mpi_hello_world.jl`

Adapted from `JuliaParallel/MPI.jl`, from the docs/examples:

```
using MPI
MPI.Init()

comm = MPI.COMM_WORLD
myid = MPI.Comm_rank(comm)
size = MPI.Comm_size(comm)

print("Hello from $myid of $size.\n")

MPI.Barrier(comm)
```

Run with `mpiexecjl`, locate and adjust path.

Summary + Exercises

Visit `http://www.mpi-forum.org/docs/`

the original MPI book is available at
`http://www.netlib.org/utk/papers/mpi-book/mpi-book.html`

Exercises: (use C, Python, or Julia)

0. Install MPI and/or mpi4py, MPI.jl on your own computer.
1. Adjust hello world so that after you type in your name once, when prompted by the manager node, every node salutes you, using the name you typed in.
2. We measure the wall clock time using `time mpirun` in the broadcasting of an array of doubles. To avoid typing in the dimension *n*, either define *n* as a constant in the program or redirect the input from a file that contains *n*. For increasing number of processes and *n*, investigate how the wall clock time grows.