Introduction to MPI

D. Thiebaut
Inspiration Reference

- **MPI** by Blaise Barney, Lawrence Livermore National Lab
  
  [https://computing.llnl.gov/tutorials/mpi/](https://computing.llnl.gov/tutorials/mpi/)
Some Background

• REVIEW: Flynn’s taxonomy of computer architecture (1966): old, and faded, but everybody seems to know it!

• SISD (uniprocessor)

• SIMD (GPU)

• MISD (rare)

• MIMD (everything else!)
• Multi-core, Many-core, Distributed systems, Clusters are all MIMD

- **SPMD**: Single Process/Multiple Data:
  
  $$\Rightarrow$$ **MPI**

- **MPMD**: Multiple Programs/Multiple Data:
MPI: Message Passing Interface

- MPI is a *specification*. Not a library
- MPI libraries implement the specification
- Processes communicate with each other:
  - Synchronization (barriers)
  - Data exchange
- MPI is **large** (430 functions in MPI-3)
- MPI is **small** (~6 functions)
Old but Vibrant!

- **Top500**: Great majority uses MPI
- **From the Top500 Q&A:**

Q: Where can I get the software to generate performance results for the Top500?

A: There is software available that has been optimized and many people use to generate the Top500 performance results. This benchmark attempts to measure the best performance of a machine in solving a system of equations. The problem size and software can be chosen to produce the best performance. A copy of that software can be downloaded from:

http://www.netlib.org/benchmark/hpl/

In order to run this you will need MPI and an optimized version of the BLAS. For MPI you can see: http://www-unix.mcs.anl.gov/mpi/mpich/download.html and for the BLAS see: http://www.netlib.org/atlas/.
Advantages of MPI

• Supported in many languages (Mostly C, but not just C)

• Supports heterogeneous computer systems

• Provides access to advanced parallel systems

• Portable (install it on your Mac or Windows PC!)
C Tutorial

(See separate set of slides)
Hello world!
(version 1)

// minimalist hello
// world program
// D. Thiebaut
#include <mpi.h>
#include <stdio.h>

int main( int argc, char *argv[] ) {
    MPI_Init( &argc, &argv );
    printf( "Hello world!\n" );
    MPI_Finalize();
    return 0;
}

Computer Science
Dominique Thiebaut
Compile & Run

[15:33:05] ~/mpi/352$: mpicc -o hello1 hello1.c
[15:33:41] ~/mpi/352$: mpirun -np 1 ./hello1
Hello world!
[15:33:48] ~/mpi/352$: mpirun -np 2 ./hello1
Hello world!
Hello world!
[15:33:53] ~/mpi/352$: mpirun -np 4 ./hello1
Hello world!
Hello world!
Hello world!
Hello world!
[15:33:57] ~/mpi/352$: 
Different Syntaxes

- mpirun -np 2 ./hello
- mpirun -n 2 ./hello
- mpiexec -n 2 ./hello
- mpiexec -np 2 ./hello
Hello World!
(Version 2: more interesting)

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

int main(int argc, char *argv[]) {
    int numprocs, rank, namelen;
    char processor_name[MPI_MAX_PROCESSOR_NAME];

    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Get_processor_name(processor_name, &namelen);

    printf("Process %d on %s out of %d\n", rank,
            processor_name, numprocs);

    MPI_Finalize();
}
Compile & Run

[15:42:00] ~/mpi/352$: mpirun -np 2 ./hello2
Process 0 on MacDom2.local out of 2
Process 1 on MacDom2.local out of 2
• MPI functions return values, either an error code or MPI_SUCCESS

• An error causes all processes to stop

• Two important MPI functions:
  - **MPI_Comm_size**: # of processes enrolled
  - **MPI_Comm_rank**: rank of current process
Exercise

Hello World on Aurora

• Create your first MPI Hello World! Program on Aurora (or your own laptop if you have installed MPI on it) compile it, and run it.
Follow the tutorial at this URL
http://www.science.smith.edu/dftwiki/index.php/Setup_MPI_on_Hadoop_Cluster

to run MPI on a cluster of 4 servers

See next slides for process overview
User

ssh

Hadoop01 (manager)

passwordless ssh

Hadoop02 (worker)

Hadoop03 (worker)

Hadoop04 (worker)
User

SSH

Hadoop01 (manager)

mpicc helloWorld.c -o hello

Hadoop02 (worker)

Hadoop03 (worker)

Hadoop04 (worker)
User

ssh

Hadoop01 (manager)

rsync -azv hello hadoop02
rsync -azv hello hadoop03
rsync -azv hello hadoop04

Hadoop02 (worker)

Hadoop03 (worker)

Hadoop04 (worker)
User

ssh

mpirun -np 4 --hostfile hosts ./hello

Hadoop01 (manager)

mpirun

Hadoop02 (worker)

mpirun

Hadoop03 (worker)

mpirun

Hadoop04 (worker)
// pi.c
#include <stdlib.h>
#include <stdio.h>

double f( double x ) { return 4.0 / ( 1 + x*x ); }

int main( int argc, char *argv[] ) {
    int N, i;
    double deltaX, sum;

    if ( argc < 2 ) {
        printf( "Syntax %s N\n", argv[0] );
        exit(1);
    }

    N = atoi( argv[1] );
    sum = 0;
    deltaX = 1.0/N;

    for ( i = 0; i < N; i++ )
        sum += f( i * deltaX );

    printf( "%d iterations: Pi = %1.6f\n", N, sum*deltaX );
    }

cc -o pi pi.c
./pi 100000
100000 iterations: Pi = 3.141603

getcopy handout/mpi/pi.c
Manager/Worker Paradigm

Manager

Worker

Work/Data

Distributes work, Schedules, Load-balances, Gathers results
Manager/Worker Paradigm

Manager

Work/Data

Worker

Computer Science

Dominique Thiebaut
Computing Pi
(Parallel version in MPI)

Manager/Worker setup

```
mpirun -np 2 ./pi2 N
```
Computing Pi
(Parallel version in MPI)

- Manager/Worker setup

```
mpirun -np 2 ./pi2 N
```

Manager

Worker

`main` Send  `main` Recv

N/2
Computing Pi

(Parallel version in MPI)

- Manager/Worker setup

```bash
mpirun -np 2 ./pi2 N
```

Manager

Worker

main

compute sum w/ N/2 samples

N/2
Computing Pi
(Parallel version in MPI)

- Manager/Worker setup

```bash
mpirun -np 2 ./pi2 N
```

Manager

Worker

```
main
```

```
compute sum w/ N/2 samples
```

```
print result
```

```
Send
```

```
Recv
```

```
N/2
```

```
sum
```

```
main
```

```
compute sum w/ N/2 samples
```

```
```
int main(int argc, char *argv[]) {
    int N, myId, noProcs, nameLen, i;
    char procName[MPI_MAX_PROCESSOR_NAME];

    if ( argc<2 ) {
        printf( "Syntax: mpirun -np 2 pi2 N\n" );
        return 1;
    }
    N = atoi( argv[1] );

    //--- start MPI ---
    MPI_Init( &argc, &argv );
    MPI_Comm_rank( MPI_COMM_WORLD, &myId );
    MPI_Comm_size( MPI_COMM_WORLD, &noProcs );
    MPI_Get_processor_name( procName, &nameLen );
    printf( "Process %d of %d started on %s. N = %d\n",
            myId, noProcs, procName, N );
    //--- farm out the work: 1 manager, several workers ---
    if ( myId == MANAGER )
        doManager( N );
    else
        doWorker( );

    //--- close up MPI ---
    MPI_Finalize();
    return 0;
}
void doManager( int n ) {
    double sum0 = 0, sum1;
    double deltaX = 1.0/n;
    int i;
    MPI_Status status;

    //--- first send n to worker ---
    MPI_Send( &n, 1, MPI_INT, WORKER, 0, MPI_COMM_WORLD );

    //--- perform 1st half of the work ---
    for ( i=0; i< n/2; i++ )
        sum0 += f( i * deltaX );

    //--- wait for other half from worker ---
    MPI_Recv( &sum1, 1, MPI_DOUBLE, MPI_ANY_SOURCE, 0, MPI_COMM_WORLD, &status );

    //--- output result ---
    printf( "%d iterations: Pi = %1.6f\n", n, ( sum0 + sum1 )*deltaX );
}
///== W O R K E R ==

void doWorker( ) {
    int i, n;
    MPI_Status status;
    double sum = 0, deltaX;

    //--- get n from manager ---
    MPI_Recv( &n, 1, MPI_INT, MPI_ANY_SOURCE, 0, MPI_COMM_WORLD, &status );

    //--- do (second) half of the work ---
    deltaX = 1.0/n;
    for ( i=n/2; i< n; i++ )
        sum += f( i * deltaX );

    //-- send result to manager ---
    MPI_Send( &sum, 1, MPI_DOUBLE, MANAGER, 0, MPI_COMM_WORLD );
}
MPI_Send

```c
MPI_Send(&work, // buffer
         1, // number of items
         MPI_INT, // type of items
         rank, // Id of receiver
         tag, // message tag (must match)
         MPI_COMM_WORLD); // the communicator’s group
```

**Basic blocking send operation.** Routine returns only after the application buffer in the sending task is free for reuse. Note that this routine may be implemented differently on different systems.

https://computing.llnl.gov/tutorials/mpi/#Derived_Data_Types

<table>
<thead>
<tr>
<th>Derived Data Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI_CHAR</td>
</tr>
<tr>
<td>MPI_SHORT</td>
</tr>
<tr>
<td>MPI_INT</td>
</tr>
<tr>
<td>MPI_LONG</td>
</tr>
<tr>
<td>MPI_UNSIGNED</td>
</tr>
<tr>
<td>MPI_FLOAT</td>
</tr>
<tr>
<td>MPI_DOUBLE</td>
</tr>
</tbody>
</table>
Receive a message and block until the requested data is available in the application buffer in the receiving task.

https://computing.llnl.gov/tutorials/mpi/

http://www.mpi-forum.org/docs/mpi-11-html/node35.html#Node35
Status Structure

```c
int recvd_tag, recvd_from;
int recvd_count;
MPI_Status status;

MPI_Recv(..., ..., ..., &status);

Recvd_tag = status.MPI_TAG;
Recvd_from = status.MPI_SOURCE;
MPI_Get_count( &status,
               datatypeOfbuffer,
               &recvd_count );
```
Compile and Run

(on aurora)
mpicc -o pi2b pi2b.c
mpirun -np 2 ./pi2b 1000000

Process 0 of 2 started on MacDom2.local. N = 1000000
Process 1 of 2 started on MacDom2.local. N = 1000000
1000000 iterations: Pi = 3.141594
**Definition**

**MPI_COMM_WORLD**

- MPI_COMM_WORLD is a *Communicator*
- It contains ALL processes
- A communicator determines the scope and the "communication universe" in which a point-to-point or collective operation is to operate.
- It’s the *universe* for most MPI programs
MPI_Send(\_,\_,\_,\_,n,\_,\_)  
\hspace{1cm}  MPI_Recv(\_,\_,\_,\_,0,\_,\_,\_,\_)  
Process 0  \hspace{1cm}  Process n

MPI_Bcast(\_,\_,\_,\_,\_,\_)  
\hspace{1cm}  MPI_Recv(\_,\_,\_,\_,0,\_,\_,\_,\_)  
Process 0  \hspace{1cm}  Process n

MPI_Send(\_,\_,\_,\_,n,\_,\_)  
\hspace{1cm}  MPI_Recv(\_,\_,\_,\_,\_\_\_,\_,\_,\_,\_,\_\_,\_\_,\_)  
Process 0  \hspace{1cm}  Process n
Exercise

• Create/get your own version of the pi program and run it with np=2, 10, 20
  • On the hadoop cluster, or
  • On your laptop
Scheduling/Load-Balancing

• Similar concepts

• Goal: Maximize performance by transferring tasks from busy to idle processors

• How: Determine parallel tasks + assign tasks to processors
Communication vs. Computation
Communication vs. Computation

Coarse-grain parallelism
Communication vs. Computation

Manager

Worker 1

Worker 2

time

Computer Science

Dominique Thiebaut
All-Important Graph: Communication vs Computation

Fixed # Procs

Packet Size
All-Important Graph: Communication vs Computation

Packet Size

Communication/packet
All-Important Graph: Communication vs Computation

Packet Size

Communication/packet

Computation/packet

Comm-overhead/packet
All-Important Graph: Communication vs Computation

Packet Size

Comm-overhead/packet

Computation/packet

Dominique Thiebaut

Computer Science
Scheduling w/ Manager/Workers

- Approach:
  - Define a Protocol for exchanging data
  - How to start
  - Control steady state
  - How to stop
  - Make sure special cases are handled
Exercise

• Modify the Pi-approximation program so that the manager distributes computation in small chunks to its $n$ workers.
Reference

- List of all MPI functions