CSC352
Week #7 — Spring 2017
Introduction to MPI

Dominique Thiébaut
dthiebaut@smith.edu
Introduction to MPI

D. Thiebaut
Inspiration Reference

• **MPI** by Blaise Barney, Lawrence Livermore National Lab
  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!)
MIMD

- Multi-core, Many-core, Distributed systems, Clusters are all **MIMD**

  - **SPMD**: Single Process/Multiple Data:
   
   => **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** (125 functions)
- 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)
// 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;
}
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$: 
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 this process
Exercise

- 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 (more details in class on which machine to use)
// 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
Manager/Worker Paradigm

Manager

Worker

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

Manager

Worker

Worker

Worker

Worker

Worker
Computing Pi
(Parallel version in MPI)

• Manager/Worker setup

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

Manager

Worker
Computing Pi
(Parallel version in MPI)

- Manager/Worker setup

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

Manager

Worker

main

N/2

Send

Recv...
Computing Pi
(Parallel version in MPI)

- Manager/Worker setup

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

Manager

Worker
Computing Pi
(Parallel version in MPI)

Manager/Worker setup

mpirun -np 2 ./pi2 N

Manager
Worker

compute sum w/ N/2 samples
main
compute sum w/ N/2 samples
main

N/2
Send
Recv
Sum
print result
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;
}
Computing Pi
(Parallel version in MPI)
Manager Function

```c
//=== MANAGER ===
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 );
}
```
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

MPI_Send(&work,
         1,
         MPI_INT,
         rank,
         tag,
         MPI_COMM_WORLD);

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

MPI_CHAR
MPI_SHORT
MPI_INT
MPI_LONG
MPI_UNSIGNED
MPI_FLOAT
MPI_DOUBLE
MPI_Recv

MPI_Recv(&result,       // buffer
    1,             // # items
    MPI_DOUBLE,    // item type
    MPI_ANY_SOURCE, // receive from any sender
    MPI_ANY_TAG,   // any tag
    MPI_COMM_WORLD, // default communicator
    &status);      // info about the received
                   // message

In C, status is a structure that contains three fields named MPI_SOURCE, MPI_TAG, and MPI_ERROR; the structure may contain additional fields. Thus, status.MPI_SOURCE, status.MPI_TAG and status.MPI_ERROR contain the source, tag, and error code, respectively, of the received message.

http://www.mpi-forum.org/docs/mpi-11-html/node35.html#Node35
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 beowulf or hadoop0)
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
Communication

MPI_Send(__, __, __, n, __, __)

Process 0

MPI_Recv(__, __, __, 0, __, __, __)

Process n

MPI_Bcast(__, __, __, __, __, __)

Process 0

MPI_Recv(__, __, __, 0, __, __, __)

Process n

MPI_Send(__, __, __, n, __, __)

Process 0

MPI_Recv(__, __, __, __, __, __, __)

Process n

MPI_Send(__, __, __, n, __, __)

Process 0

MPI_Recv(__, __, __, __, __, __, __, __)

Process n
Exercise

• Create your own version of the pi program (type it!) and run it with np=2
  • On your laptop
  • On aurora
Exercise

• Create your own version of the pi program (type it!) and run it with $np=10$
  • On your laptop
  • On aurora
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

Manager

Worker

Compute

Communication

time
Communication vs. Computation

Coarse-grain parallelism

Manager

Worker 1

Worker 2
Communication vs. Computation
All-Important Graph: Communication vs Computation
All-Important Graph: Communication vs Computation

Packet Size vs Communication/packet
All-Important Graph: Communication vs Computation

Packet Size

Communication/packet

Computation/packet
All-Important Graph: Communication vs Computation

Packet Size

Communication/packet

Computation/packet
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.
• List of all MPI functions