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/](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:
  \[\Rightarrow\text{MPI}\]

- **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;
}
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)

```c
#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

```plaintext
[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
```
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.
We Stopped Here Last Time
• Hello World on Cluster of 4 Servers
• MPI_Send/MPI_Recv/MPI_COMM_WORLD
• Potato (Exercise)
• Pi (Example)
• One-To-Many, Many-To-One Communication
• Load-Balancing and Scheduling
• MPI on AWS
• 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
Hello World on Hadoop01

• 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
Tutorial

User

Hadoop01 (manager)

passwordless ssh

Hadoop02 (worker)

Hadoop03 (worker)

Hadoop04 (worker)

ssh
User

```
mpicc helloWorld.c -o hello
```

Hadoop01 (manager)

Hadoop02 (worker)

Hadoop03 (worker)

Hadoop04 (worker)
User

ssh

Hadoop01 (manager)

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

rsync

Hadoop03 (worker)

rsync

Hadoop02 (worker)

rsync

Hadoop04 (worker)
User

```
mpirun -np 4 --hostfile hosts ./hello
```
Deploy HelloWorld on the Hadoop Cluster
We Stopped Here Last Time
Go Over Java Multithreaded Game of Life Homework
Communication Patterns

Manager

Worker 0
Worker 1
Worker 2
Worker 3
Communication Patterns

Manager

Worker 0
Worker 1
Worker 2
Worker 3
Go Ahead

We're Done!

Running Life()

Copy Dish
Thread Run()

/*
 * This will apply life repeatedly to the half of the dish array that
 * belongs to this thread.
 *
 * @see java.lang.Thread#run()
 */

public void run() {
    int noRows = dataN.dish.length; // number of rows in dish
    int startRow = Id * noRows / N; // where thread starts computing life
    int endRow = ((Id+1) * noRows / N); // where thread stop computing life
    if (Id == N-1) endRow = noRows; // if last thread, make sure include last

    int command = 1;
    while (command != 0) {
        life(startRow, endRow); // send a token, wait for token
        try {
            toManager.put(Id);
            command = fromManager.take();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}

getcopy GameOfLifeNThreads.java
Manager Main Loop

// go through generations
for ( int i = 0; i < gens; i++ ) {
    if ( verbose ) System.out.println( "generation " + i );

    // wait for N threads to finish their computation of life
    for ( int t=0; t<N; t++ ) {
        try {
            int c = QFromThreads.take();
            if ( verbose ) System.out.println( " Received \"Done\" message from Thread " + c );
        } catch (InterruptedException e) {
        }
    }

    if ( verbose ) System.out.println( " Heard from all threads..." );

    // copy newGen to dish
    for ( int row=0; row < dataN.dish.length; row++ )
        dataN.dish[row] = dataN.newGen[row];

    if ( verbose ) System.out.println( " dish copied" );
    //print( dataN.dish );

    // tell N threads to resume computation
    for ( int t=0; t<N; t++ ) {
        int command = 1;
        if (i==gens-1)
            command = 0;
        try {
            QToThreads[t].put( command );
        } catch (InterruptedException e) {
        }
    }

    if ( verbose ) System.out.println( " all threads alerted to continue..." );
}
CSC352 Game of Life in Java, N Threads

// Manager Main Loop
try {
    for (int i = 0; i < gens; i++) {
        // go through generations
        for (int t = 0; t < N; t++) {
            if (verbose) System.out.println("generation "+i);
            try {
                if (verbose) System.out.println("all threads alerted to continue...");
                for (int command = 0; command < NThreads.length; command++) {
                    try {
                        if (verbose) System.out.println("received new command from Thread "+command);
                        if (verbose) System.out.println("Received "+command);"
                    } catch (InterruptedException e) {
                        if (verbose) System.out.print(e);
                        System.out.println("Message from Thread "+command);
                    }
                }
            } catch (Exception e) {
                if (verbose) System.out.println(e);
                System.out.println("Message from Thread "+command);
            }
            try {
                // tell N threads to resume computation
                for (int command = 0; command < NThreads.length; command++) {
                    try {
                        if (verbose) System.out.println("task of command "+command);
                        NThreads[command].put(command);
                    } catch (InterruptedException e) {
                        if (verbose) System.out.println(e);
                        System.out.println("Message from Thread "+command);
                    }
                }
            } catch (Exception e) {
                if (verbose) System.out.println(e);
                System.out.println("Message from Thread "+command);
            }
            try {
                // wait for N threads to finish their computation
                for (int command = 0; command < NThreads.length; command++) {
                    try {
                        if (verbose) System.out.println("thread "+command);
                        int c = QFromThreads.takeO();
                        if (verbose) System.out.println("issue for Thread "+command);
                    } catch (InterruptedException e) {
                        if (verbose) System.out.println(e);
                        System.out.println("Message from Thread "+command);
                    }
                }
            } catch (Exception e) {
                if (verbose) System.out.println(e);
                System.out.println("Message from Thread "+command);
            }
            try {
                if (verbose) System.out.println("Debugging message from Thread "+command);
                // all threads alerted to continue...");
            } catch (Exception e) {
                if (verbose) System.out.println(e);
                System.out.println("Message from Thread "+command);
            }
        }
    }
} catch (Exception e) {
    if (verbose) System.out.println(e);
    System.out.println("Message from Thread "+command);
}
2048 rows, 80 cells wide
MacBook Pro 2014
4 cores, 2.8 GHz
Performance

Threads | Speedup
---|---
2 | 0.895231676
4 | 1.088301802
8 | 1.152712505
16 | 2.026521961
32 | 1.976106799
64 | 1.186445303
128 | 0.981590412
256 | 1.584739771
512 | 0.921974566
1024 | 0.510000713
Questions

• What serial program should be used to compute the speedup?

• What does the speedup curve tell us?

• What is the general shape of the speedup curve? How could we make it "smoother"?

• What could be done to the Java program to make it faster?  *Think hard…*
Back to MPI!
Point-to-Point Blocking Communication
### MPI_Send

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

- **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.

  ![Derived Data Types](https://computing.llnl.gov/tutorials/mpi/#Derived_Data_Types)

  - `MPI_CHAR`
  - `MPI_SHORT`
  - `MPI_INT`
  - `MPI_LONG`
  - `MPI_UNSIGNED`
  - `MPI_FLOAT`
  - `MPI_DOUBLE`

  [https://computing.llnl.gov/tutorials/mpi/](https://computing.llnl.gov/tutorials/mpi/)
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 );
```
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
• Implement the Hot Potato game in MPI and run it on Aurora. Make the potato go once around. (What is a good data-type to use for the potato?)

• Make each node of the cluster increment the potato as it passes around. Make the manager print the value of the potato when it gets it.

• Modify the MPI program so that the number of rounds can be specified on the command line.

```c
#include <stdlib.h>

int N;
N = atoi( argv[1] );
```
/* hotPotato.c 
D. Thiebaut 
Hot potato passing around, with N processes.

The manager (Process 0) passes the potato to Worker 1, 
which passes it to worker 2, etc... until the manager 
gets it back again and quits.
*/

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

#define MANAGER 0

int main ( int argc, char *argv[] ) {
    int rank, size;
    int potato = 1213;
    MPI_Status status;  // required variable for receive routines

    MPI_Init (&argc, &argv);  /* starts MPI */
    MPI_Comm_rank (MPI_COMM_WORLD, &rank);  /* get current process id */
    MPI_Comm_size (MPI_COMM_WORLD, &size);  /* get number of processes */

    if ( rank == MANAGER ) {
        //        buffer   #items  item-size src/dest       tag   world
        MPI_Send( &potato, 1,      MPI_INT,  1,             0,    MPI_COMM_WORLD );
        printf( "Manager sent potato!\n" );
        MPI_Recv( &potato, 1,      MPI_INT,  size-1,        0,    MPI_COMM_WORLD, &status );
        printf( "Manager got potato back!\n" );
    } else {
        MPI_Recv( &potato, 1,      MPI_INT,  rank-1,        0, MPI_COMM_WORLD, &status );
        printf( "Worker %d got potato!\n", rank );
        MPI_Send( &potato, 1,      MPI_INT,  (rank+1)%size, 0, MPI_COMM_WORLD );
    }

    MPI_Finalize();
    return 0;
}
/* hotPotato.c
D. Thiebaut
Hot potato passing around, with N processes.

The manager (Process 0) passes the potato to Worker 1, which passes it to worker 2, etc... until the manager gets it back again and quits.

*/

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

#define MANAGER 0

int main ( int argc, char *argv[] ) {
    int rank, size;
    int potato = 1213;
    MPI_Status status; // required variable for receive routines

    MPI_Init (&argc, &argv); /* starts MPI */
    MPI_Comm_rank (MPI_COMM_WORLD, &rank); /* get current process id */
    MPI_Comm_size (MPI_COMM_WORLD, &size); /* get number of processes */

    if ( rank == MANAGER ) {
        //        buffer   #items  item-size src/dest       tag   world
        MPI_Send( &potato, 1,      MPI_INT,  1,             0,    MPI_COMM_WORLD );
        printf( "Manager sent potato!\n" );
        MPI_Recv( &potato, 1,      MPI_INT,  size-1,        0,    MPI_COMM_WORLD, &status );
        printf( "Manager got potato back!\n" );
    } else {
        MPI_Recv( &potato, 1,      MPI_INT,  rank-1,        0, MPI_COMM_WORLD, &status );
        printf( "Worker %d got potato!\n", rank );
        MPI_Send( &potato, 1,      MPI_INT,  (rank+1)%size, 0, MPI_COMM_WORLD );
    }

    MPI_Finalize();
    return 0;
}
We Stopped Here Last Time
Misc Notes on Paper Presentations

Hadoop Web Console

### Hadoop110 Hadoop Map/Reduce Administration

- **State:** RUNNING
- **Started:** Thu Dec 16 11:55:22 EST 2010
- **Version:** 0.19.2, r786567
- **Compiled:** Tue Jun 30 12:40:50 EDT 2009 by root
- **Identifier:** 2010121611505

#### Cluster Summary

<table>
<thead>
<tr>
<th>Maps</th>
<th>Reduces</th>
<th>Total Submissions</th>
<th>Nodes</th>
<th>Map Task Capacity</th>
<th>Reduce Task Capacity</th>
<th>Avg. Tasks/Node</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>3</td>
<td>10</td>
<td>40</td>
<td>40</td>
<td>8.00</td>
</tr>
</tbody>
</table>

#### Scheduling Information

<table>
<thead>
<tr>
<th>Queue Name</th>
<th>Scheduling Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>default</td>
<td>N/A</td>
</tr>
</tbody>
</table>

#### Running Jobs

```
none
```

#### Completed Jobs

<table>
<thead>
<tr>
<th>JobId</th>
<th>Priority</th>
<th>User</th>
<th>Name</th>
<th>Map % Complete</th>
<th>Map Total</th>
<th>Maps Completed</th>
<th>Reduce % Complete</th>
<th>Reduce Total</th>
<th>Reduces Completed</th>
<th>Job Scheduling Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>job_201012161156_00003</td>
<td>NORMAL</td>
<td>hadoop</td>
<td>wordcount</td>
<td>100.00%</td>
<td>40</td>
<td>40</td>
<td>100.00%</td>
<td>10</td>
<td>10</td>
<td></td>
</tr>
</tbody>
</table>

#### Failed Jobs

<table>
<thead>
<tr>
<th>JobId</th>
<th>Priority</th>
<th>User</th>
<th>Name</th>
<th>Map % Complete</th>
<th>Map Total</th>
<th>Maps Completed</th>
<th>Reduce % Complete</th>
<th>Reduce Total</th>
<th>Reduces Completed</th>
<th>Job Scheduling Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>job_201012161156_00002</td>
<td>NORMAL</td>
<td>hadoop</td>
<td>wordcount</td>
<td>100.00%</td>
<td>40</td>
<td>40</td>
<td>100.00%</td>
<td>10</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

### Local Logs

- Log directory: Job Tracker History
- Hadoop, 2011.
Ganglia Console for Hadoop
Typical Map-Reduce Job

500 GB Task Timeline

- Running Tasks vs. Seconds
- Graph shows the distribution of running tasks over time for a 500 GB task.
In reference to “Nobody ever got fired…”

PROCESSING WIKIPEDIA DUMPS
A Case-Study comparing the XGrid and MapReduce Approaches

Dominique Thiébaut, Yang Li, Diana Jaunzeikare, Alexandra Cheng, Elyshia Raelen Recto, Gillian Riggs, Xia Ting Zhao, Tonje Stolpestad, Cam Le T Nguyen
Dept. Computer Science, Smith College, Northampton, MA, USA
{dthiebau, yli2, djaunzei, acheng, erecto, griggs, xzhao, tstolpes, lnguyen}@smith.edu

Keywords: Grid Computing, XGrid, Hadoop, wikipedia, data mining, performance

Abstract: We present a simple comparison of the performance of three different cluster platforms: Apple’s XGrid, and Hadoop the open-source version of Google’s MapReduce as the total execution time taken by each to parse a 27-GByte XML dump of the English wikipedia. A local hadoop cluster of Linux workstation, as well as an Elastic MapReduce cluster rented from Amazon are used. We show that for this specific workload, XGrid yields the fastest execution time, with the local Hadoop cluster a close second. The overhead of fetching data from Amazon’s Simple Storage System (S3), along with the inability to skip the reduce, sort, and merge phases on Amazon penalizes this platform targeted for much larger data sets.

1 INTRODUCTION

The aim of this paper is to find the fastest parallel computer cluster available at Smith College for processing Wikipedia dumps and for generating word statistics. We implement the parsing of the XML dump of the English wikipedia, and the gathering of word usage on an Apple XGrid system, and on two Hadoop clusters, one local, the other hosted on Amazon, each report on the performance obtained from paper we investigate whether hadoop is a contender, and under which conditions the execution time of the parsing process is comparable on both frameworks.

When low-cost, powerful, and easily accessible parallel computational platforms are available, it is important to better understand their source of performance and pick the best one for a solving a given problem.
Back to MPI…
An Example: Computing Pi With MPI
// 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 );
}

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

copy mpi/pi.c
Manager/Worker Paradigm

Manager

Distributes work, Schedules, Load-balances, Gathers results

Worker

Work/Data
Manager/Worker Paradigm

Manager

Work/Data

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

manager/ -np 2 ./pi2 N

Manager

Worker

main

Send

main

Recv

N/2

Computer Science
Dominique Thiebaut
Computing Pi
(Parallel version in MPI)

- Manager/Worker setup

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

Manager

Worker
Computing Pi
(Parallel version in MPI)

- Manager/Worker setup

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

Manager

Worker

Diagram:
- Parallel version in MPI
- Manager/Worker setup
- ```bash
  mpirun -np 2 ./pi2 N
  ```
- Main
- Compute sum with N/2 samples
- Print result
- Send
- Recv
- Sum
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 );
}

Computing Pi
(Parallel version in MPI)
 Worker Function
(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
Exercise

- Modify the MPI Pi computation program so that the Manager uses 2 Workers to do the computation. The Manager doesn't compute anything, except summing up the results. Worker 0 computes the sum from \([0, N/2)\), and Worker 1 computes the sum from \([N/2, N)\). Develop your code on your laptop or on aurora, for faster results.

- Run your program for different values of \(N\) and compare its execution time to the execution time of the original version. Faster? Slower?
One-to-Many, Many-to-One Communication
**Communication**

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

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

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

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

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

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

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

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

MPI_Recv(__, __, __, MPI_ANY_SOURCE, __, __, __)
Exercise

• Run your version of the Pi program with 2, 10, 20 Processes:
  • On the hadoop cluster, using 4 processors.
  • On your laptop or Aurora
• Compare the execution times…
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

Manager

Worker 1

Worker 2

time
Communication vs. Computation
All-Important Graph: Communication vs Computation

Fixed # Procs

Granularity of Data
All-Important Graph: Communication vs Computation

Granularity of Data

Comm-overhead/packet

Computation/packet
All-Important Graph: Communication vs Computation
Scheduling w/ Manager/Workers

- Approach:
  - Define a Protocol for exchanging data
  - Figure out how to start // computation
  - Control steady state
  - Define condition for when to stop
  - Make sure special cases are handled
Exercise

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

• See if you can find a combination of $n$, the size and the chunk size that reduces the execution time.
Reference

• List of all MPI functions
Installation and Tutorials

• Running MPI on AWS

• Tutorial 1 (Install StarCluster)

• Tutorial 2 (Compute Pi on AWS/MPI)

(See next slide…)
Amazon Machine Image

Instance

AWS