IAMCS Parallel Programming
Tutorial 1: MPI + Matlab/Octave

Xingfu Wu

Department of Computer Science & Engineering
Texas A&M University

Blocker 503, 2:00pm, March 22, 2013
Survey

- Who is mainly programming in
  - Matlab
  - Octave

- Who has experience with parallel processing in Matlab?
  - If yes, who has used MPI + Matlab?
  - Any other parallel programming model(s) used?
## World Top 10 Supercomputers (06/2012)

<table>
<thead>
<tr>
<th>Rank</th>
<th>Site</th>
<th>Computer/Year Vendor</th>
<th>Cores</th>
<th>$R_{\text{max}}$</th>
<th>$R_{\text{peak}}$</th>
<th>Power</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>DOE/NNSA/LLNL United States</td>
<td>Sequoia - BlueGene/Q, Power BQC 16C 1.60 GHz, Custom / 2011 IBM</td>
<td>1572864</td>
<td>16324.75</td>
<td>20132.86</td>
<td>7890.0</td>
</tr>
<tr>
<td>2</td>
<td>RIKEN Advanced Institute for Computational Science (AICS) Japan</td>
<td>K computer, SPARC64 VIII fx 2.0GHz, Tofu interconnect / 2011 Fujitsu</td>
<td>705024</td>
<td>10510.00</td>
<td>11280.38</td>
<td>12659.9</td>
</tr>
<tr>
<td>3</td>
<td>DOE/SC/Argonne National Laboratory United States</td>
<td>Mira - BlueGene/Q, Power BQC 16C 1.60GHz, Custom / 2012 IBM</td>
<td>786432</td>
<td>8162.38</td>
<td>10066.33</td>
<td>3945.0</td>
</tr>
<tr>
<td>4</td>
<td>Leibniz Rechenzentrum Germany</td>
<td>SuperMUC - iDataPlex DX360 M4, Xeon E5-2680 8C 2.70GHz, Infiniband FDR / 2012 IBM</td>
<td>147456</td>
<td>2897.00</td>
<td>3185.05</td>
<td>3422.7</td>
</tr>
<tr>
<td>5</td>
<td>National Supercomputing Center in Tianjin China</td>
<td>Tianhe-1A - NUDT YH MPP, Xeon X5670 6C 2.93 GHz, NVIDIA 2050 / 2010 NUDT</td>
<td>186368</td>
<td>2566.00</td>
<td>4701.00</td>
<td>4040.0</td>
</tr>
<tr>
<td>6</td>
<td>DOE/SC/Oak Ridge National Laboratory United States</td>
<td>Jaguar - Cray XK5, Opteron 8274 16C 2.20GHz, Cray Gemini interconnect, NVIDIA 2050 / 2009 Cray Inc.</td>
<td>298592</td>
<td>1941.00</td>
<td>2627.51</td>
<td>5142.0</td>
</tr>
<tr>
<td>7</td>
<td>CINECA Italy</td>
<td>Fermi - BlueGene/Q, Power BQC 16C 1.60GHz, Custom / 2012 IBM</td>
<td>163840</td>
<td>1725.49</td>
<td>2097.15</td>
<td>821.9</td>
</tr>
<tr>
<td>8</td>
<td>Forschungszentrum Juelich (FZJ) Germany</td>
<td>JuQUEEN - BlueGene/Q, Power BQC 16C 1.60GHz, Custom / 2012 IBM</td>
<td>131072</td>
<td>1380.39</td>
<td>1677.72</td>
<td>657.5</td>
</tr>
<tr>
<td>9</td>
<td>CEA/TGCC-GENCI France</td>
<td>Curie thin nodes - Bullx B510, Xeon E5-2680 8C 2.70GHz, Infiniband QDR / 2012 Bull</td>
<td>77184</td>
<td>1359.00</td>
<td>1667.17</td>
<td>2251.0</td>
</tr>
<tr>
<td>10</td>
<td>National Supercomputing Centre in Shenzhen (NSCS) China</td>
<td>Nebulae - Dawning TC3600 Blade System, Xeon X5650 6C 2.66GHz, Infiniband QDR, NVIDIA 2050 / 2010 Dawning</td>
<td>120640</td>
<td>1271.00</td>
<td>2984.30</td>
<td>2580.0</td>
</tr>
</tbody>
</table>

Source: [www.top500.org](http://www.top500.org) (R: Tflops  Power: KW)
# World Top 10 Supercomputers (11/2012)

<table>
<thead>
<tr>
<th>Rank</th>
<th>Site</th>
<th>System</th>
<th>Cores</th>
<th>Rmax (TFlop/s)</th>
<th>Rpeak (TFlop/s)</th>
<th>Power (kW)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>DOE/SC/Oak Ridge National Laboratory</td>
<td>Titan - Cray XK7, Opteron 6274 16C 2.200GHz, Cray Gemini interconnect, NVIDIA K20x Cray Inc.</td>
<td>560840</td>
<td>17590.0</td>
<td>27112.5</td>
<td>8209</td>
</tr>
<tr>
<td>2</td>
<td>DOE/NNSA/LLNL United States</td>
<td>Sequoia - BlueGene/Q, Power BQC 16C 1.60GHz, Custom IBM</td>
<td>1572864</td>
<td>16324.8</td>
<td>20132.7</td>
<td>7890</td>
</tr>
<tr>
<td>3</td>
<td>RIKEN Advanced Institute for Computational Science (AICS) Japan</td>
<td>K computer, SPARC64 VIIIfx 2.0GHz, Tofu interconnect Fujitsu</td>
<td>705024</td>
<td>10510.0</td>
<td>11280.4</td>
<td>12860</td>
</tr>
<tr>
<td>4</td>
<td>DOE/SC/Argonne National Laboratory United States</td>
<td>Mira - BlueGene/Q, Power BQC 16C 1.60GHz, Custom IBM</td>
<td>786432</td>
<td>8162.4</td>
<td>10066.3</td>
<td>3945</td>
</tr>
<tr>
<td>5</td>
<td>Forschungszentrum Juelich (FZJ) Germany</td>
<td>JUQUEEN - BlueGene/Q, Power BQC 16C 1.600GHz, Custom Interconnect IBM</td>
<td>393216</td>
<td>4141.2</td>
<td>5033.2</td>
<td>1970</td>
</tr>
<tr>
<td>6</td>
<td>Leibniz Rechenzentrum Germany</td>
<td>SuperMUC - iDataPlex DX360M4, Xeon E5-2680 8C 2.70GHz, Infiniband FDR IBM</td>
<td>147456</td>
<td>2897.0</td>
<td>3185.1</td>
<td>3423</td>
</tr>
<tr>
<td>7</td>
<td>Texas Advanced Computing Center/Univ. of Texas United States</td>
<td>Stampede - PowerEdge C8220, Xeon E5-2680 8C 2.700GHz, Infiniband FDR, Intel Xeon Phi Dell</td>
<td>204900</td>
<td>2660.3</td>
<td>3959.0</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>National Supercomputing Center in Tianjin China</td>
<td>Tianhe-1A - NUDT YH MPP, Xeon X5670 8C 2.93 GHz, NVIDIA 2050 NUDT</td>
<td>186368</td>
<td>2566.0</td>
<td>4701.0</td>
<td>4040</td>
</tr>
<tr>
<td>9</td>
<td>CINECA Italy</td>
<td>Fermi - BlueGene/Q, Power BQC 16C 1.60GHz, Custom IBM</td>
<td>163840</td>
<td>1725.5</td>
<td>2097.2</td>
<td>822</td>
</tr>
<tr>
<td>10</td>
<td>IBM Development Engineering United States</td>
<td>DARPA Trial Subset - Power 775, POWER7 8C 3.83GHz, Custom Interconnect IBM</td>
<td>63360</td>
<td>1515.0</td>
<td>1944.4</td>
<td>3576</td>
</tr>
</tbody>
</table>

Source: [www.top500.org](http://www.top500.org) (R: Tflops  Power: KW)
Outline

- Parallel Matlab/Octave
  - Matlab, Octave
  - Parallel Computing Toolbox
  - MatlabMPI and its extension
- IAMCS Clusters: Brazos and Hurr
- Try to parallelize your own code
- Q&A
MATLAB

- MATLAB: MATrix LABoratory, the language of technical computing developed by MathWorks (http://www.mathworks.com/)

- An interactive environment for algorithm development, data visualization, data analysis, and numerical computation

- Widely used in a lot of applications in engineering, math., stat., biology, and so on.
Octave

- GNU Octave is a high-level interactive language, primarily intended for numerical computations that is mostly compatible with Matlab (http://www.gnu.org/software/octave/)
- It provides a convenient command line interface for solving linear and nonlinear problems numerically
- The name Octave is actually the name of one of the author's former professors
Comparison: Matlab and Octave

- Free Octave available under the GNU General Public License (GNU GPL); Commercial Matlab
- As free alternatives, Octave re-implements a MATLAB subset
- MATLAB clone: Octave’s syntax is very similar to MATLAB so that most programs are easily portable.
- Octave for open source community all over the world
Differences: Matlab and Octave

- Octave's parser allows some syntax that Matlab's does not, so programs written for Octave might not run in Matlab.
  - Octave supports the use of both single and double quotes. Matlab only supports single quotes, which means parsing errors will occur if you try to use double quotes.

- See the link for detailed differences:
Parallel Computing Toolbox™ lets you solve computationally and data-intensive problems using multicore processors, GPUs, and computer clusters.

High-level constructs—parallel for-loops (parfor), special array types, and parallelized numerical algorithms—let you parallelize MATLAB® applications without CUDA or MPI.
Parallel Computing Toolbox

- The toolbox provides twelve workers (MATLAB computational engines) to execute applications locally on a multicore desktop.

- Without changing the code, you can run the same application on a computer cluster using MATLAB Distributed Computing Server™ (DCS).

- MATLAB Distributed Computing Server™: MATLAB and Simulink® based applications
Parallel Computing Toolbox

Multicore Desktop with GPUs

Parallel Computing Toolbox
Local Workers
Simulink, Blocksets, and Other Toolboxes
MATLAB

Computer Cluster
MATLAB Distributed Computing Server
Workers
Scheduler

Limitations: Parallel Computing Toolbox

- The toolbox provides only twelve workers (MATLAB computational engines) to execute applications locally.
- It requires MATLAB Distributed Computing Server™ for a computer cluster.
  - Efficient hardware support, License
- Low program portability on another cluster without MATLAB DCS.
- Need a general, portable parallel Matlab.
MATLABMPI

Parallel Programming with MatlabMPI

Dr. Jeremy Kepner
kepner@l.ll.mit.edu

I. INTRODUCTION

Matlab is the dominant programming language for implementing numerical computations and is widely used for algorithm development, simulation, data reduction, testing and system evaluation. Many of these computations could benefit from faster execution on a parallel computer. There have been many previous attempts to provide an efficient mechanism for running Matlab programs on parallel computers. These efforts have faced numerous challenges and none have received widespread acceptance.

In the world of parallel computing the Message Passing Interface (MPI) is the de facto standard for implementing programs on multiple processors. MPI defines C and Fortran language functions for doing point-to-point communication in a parallel program. MPI has proven to be an effective model for implementing parallel programs and is used by many of the world’s most demanding applications (weather modeling, weapons simulation, aircraft design, etc.).

MatlabMPI is set of Matlab scripts that implement a subset of MPI and allow any Matlab program to be run on a parallel computer. The key innovation of MatlabMPI is that it implements the widely used MPI "lock and feel" on top of standard Matlab file I/O, resulting in a "pure" Matlab implementation that is exceedingly small (~300 lines of code). Thus, MatlabMPI will run on any combination of computers that Matlab supports. In addition, because of its small size, it is simple to download and use (and modify if you like).

Download: MatlabMPI_v1.2.tar.gz

MatlabMPI Page Contents
- Introduction
- Download
- Requirements
- Installing and Running
- Launching and File I/O
- Error Handling
- Running on Linux
- Running on MacOSX
- Running on Windows
- Other Optimizations
- Running in Batch Mode
- Other Settings
- Diagnostics and Troubleshooting
- First-Time User’s Rules of Thumb
- Files

pMatlab: Parallel Matlab Toolbox

pMatlab provides a set of Matlab data structures and functions that implement distributed Matlab arrays
What is MatlabMPI?

- It is a MATLAB implementation of the MPI standards that allows any MATLAB program to exploit multiple processors.
- It implements, the basic MPI functions that are the core of the MPI point-to-point communications with extensions to other MPI functions using pure M-file implementation.
- It runs anywhere MATLAB/Octave runs.
- Principal developer: Dr. Kepner (MIT Lincoln Laboratory)
General Requirements

- As MatlabMPI uses Matlab file I/O for communication, a common file system must be visible to every machine/processor.

- On shared memory platforms: single MATLAB license is enough since any user is allowed to launch many MATLAB sessions.

- On distributed memory platforms: one MATLAB license per machine/node.
Basic Concepts of MatlabMPI

- MatlabMPI “Core Lite” requires eight capabilities and additional convenience functions.
- MatlabMPI implements the fundamental communication operations in MPI using MATLAB’s file I/O functions.
- Any message passing can be implemented using file I/O provided by Matlab via load and save functions.
Basic Concepts of MatlabMPI

- **Messages**: MATLAB variables transferred from one processor to another

- **Synchronous transfer**: A function call does not return until the message is sent or received
### MPI: Message Passing Interface

**The minimal set of MPI routines**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI_Init</td>
<td>Initializes MPI.</td>
</tr>
<tr>
<td>MPI_Finalize</td>
<td>Terminates MPI.</td>
</tr>
<tr>
<td>MPI_Comm_size</td>
<td>Determines the number of processes.</td>
</tr>
<tr>
<td>MPI_Comm_rank</td>
<td>Determines the label of calling process.</td>
</tr>
<tr>
<td>MPI_Send</td>
<td>Sends a message.</td>
</tr>
<tr>
<td>MPI_Recv</td>
<td>Receives a message.</td>
</tr>
</tbody>
</table>

**mpirun**: launches MPI processes on multiple processors
MatlabMPI

The minimal set of MatlabMPI routines

- **MPI_Init**: Initializes MPI.
- **MPI_Finalize**: Terminates MPI.
- **MPI_Comm_size**: Determines the number of processes.
- **MPI_Comm_rank**: Determines the label of calling process.
- **MPI_Send**: Sends Matlab variables to another process.
- **MPI_Recv**: Receives Matlab variables from another process.

**MPI_Run**: launches multiple copies of a Matlab script on multiple processors

**MatMPI_Delete_all**: cleans up all files after a run
MatlabMPI Functionality

- “Core Lite” requires eight capabilities
  - MPI_Init: initializes MPI at beginning of program
  - MPI_Finalize: finalizes MPI at end of program
  - MPI_Comm_size: returns the number of processes
  - MPI_Comm_rank: returns the id of each process
  - MPI_Send: sends Matlab variable(s) to another process
  - MPI_Recv: receives Matlab variable(s) from another process
  - MPI_Run: launches multiple copies of a Matlab script on multiple processors
  - MatMPI_Delete_all: cleans up all files after a run
MatlabMPI Functionality

- Additional convenience functions
  - MPI_Abort: kills all jobs
  - MPI_Bcast: broadcasts a message
  - MPI_Probe: returns a list of all incoming messages
  - MPI_cc: passes program through Matlab compiler
  - MatMPI_Save_messages: toggles deletion of messages
  - MatMPI_Comm_settings: user can set MatlabMPI internals
Extending MatlabMPI

- Extend MatlabMPI to support the cluster systems with PBS schedulers
  - Original MatlabMPI just supports clusters without any scheduler, and treat each node as local or remote
  - We modified it (MPI_Run) to support PBS schedulers

- Add additional functions
  - **MPI_Barrier**: Blocks until all processes in the communicator have reached this routine
  - **MPI_Reduce**: Reduces values on all processes to a single value
MPI: Point-to-Point Communication Mechanism

Process 0

User Mode

... Call mpi_send(sendbuf, dest=1)
(blacked)
Now sendbuf can be reused
...

Kernel Mode

sysbuf

Copying data from sendbuf to sysbuf
Send data from sysbuf to dest

Process 1

User Mode

... Call mpi_recv(recvbuf, src=0)
(blacked)
Now recvbuf contains valid data
...

Kernel Mode

sysbuf

Receive data from src to sysbuf
Copying data from sysbuf to recvbuf
MatlabMPI: Point-to-Point Communication Mechanism

Source: Dr. Kepner’s talks: ”Parallel Matlab: the Next Generation”
Example: MatlabMPI (Hello World)

% Initialize MPI
MPI_Init;

% Create the MPI communicator
comm= MPI_COMM_WORLD;

% Get size and rank
size = MPI_Comm_size(comm);
rank= MPI_Comm_rank(comm);

disp(['Hello world from process : ',num2str(rank),' of ', num2str(size)]);

% Finalize MatlabMPI
MPI_Finalize;

*Output: Hello world from process 0 of 2
Hello world from process 1 of 2*
How to Write a MatlabMPI Code

1. Initialize MPI
   MPI_Init;
2. Create MPI communicator
   comm = MPI_COMM_WORLD;
3. Get size and rank
   size = MPI_Comm_size(comm);
   rank = MPI_Comm_rank(comm);
4. Some computation for each processor
   disp(['Hello world from process : ',num2str(rank),’ of ‘,
         num2str(size)]);
5. Communication using MPI for data exchange if needed
   MPI_Send, MPI_Recv…
6. Some computation for each processor using Matlab
7. Finalize MPI
   MPI_Finalize;
A Simple Matlab Code: Summation

% sequential summation using Matlab/Octave

n = 10000000;
s = 0;

% a large loop

for i=1:n
    s = s+i;
end

% print the total summation
disp(['Total summation: ', num2str(s)]);
How to Parallelize a Matlab Code

Design a parallel algorithm for the code
1. Initialize MPI
   MPI_Init;
2. Create MPI communicator
   comm= MPI_COMM_WORLD;
3. Get size and rank
   size = MPI_Comm_size(comm);
   rank= MPI_Comm_rank(comm);
4. Parallelize some computation
5. Communication using MPI for data exchange if needed
   MPI_Send, MPI_Recv…
6. Some computation for each processor using Matlab
7. Finalize MPI
   MPI_Finalize;
n = 100000000;
s = 0;
\% Initialize MPI
MPI_Init;
\% Create MPI communicator
comm = MPI_COMM_WORLD;
\% Get size and rank
size = MPI_Comm_size(comm);
rank = MPI_Comm_rank(comm);
master = 0;

\% Split the for loop into size partitions
m = n / size;
r = mod(n, m);
if ( rank == size-1 )
    se = (rank + 1)*m + r;
else
    se = (rank + 1)*m;
end
\% Each process works on a partition of the loop independently
s1 = s;
for i=(rank * m)+1:se
    s1 = s1 + i;
end
\% print the partial summation on each process
disp(['Partial summation: ', num2str(s1), ' on process ', num2str(rank)]);
Parallelize the Simple Code

% Create a unique tag id for this message (very important in Matlab MPI!).
tag = 1000 + rank;

% Send the partial summation from each process to the master (rank=0)
if ( rank > 0 )
    MPI_Send( master, tag, comm, s1 );
else
%  Receive the partial summation from each process (rank > 0) and add them
    s = s1;
    for j=1:size-1
        tag1 = 1000 + j;
        s1 = MPI_Recv( j, tag1, comm );
        s = s + s1;
    end
end

% print the total summation on the master (rank = 0)
if (rank == 0)
    disp(['Total summation : ',num2str(s)]);
end

% Finalize MatlabMPI
MPI_Finalize;
Parallelize the Simple Code

% Reduce the values on all processes to a single value on master (rank =0)
s = MPI_Reduce( s1, MPI_SUM, 0, comm);

% print the total summation on the master (rank = 0)
if (rank == 0)
    disp(['Total summation : ',num2str(s)]);
end
% Finalize MatlabMPI
MPI_Finalize;

Note: MPI_SUM means summation. It also supports MPI_PROD (multiplication), MPI_MIN (minimum), and MPI_MAX (maximum)
Partitioning and Load Balancing

- Good partition
- Bad partition
- Execution time
- Parallel processes (P1, P2, P3)
Load Balancing

- Load balancing refers to the practice of distributing work among tasks so that all tasks are kept busy all the time.

- If data is fairly homogeneous, an equal partition of the work among different tasks suffices, otherwise a dynamic work assignment strategy needs to be used.

- An important consideration which can be controlled by communication is load balancing.
Load Balancing

- **Static**: done once, by the programmer
  - block, cyclic, etc.
  - good for regular, predictable, data parallel

- **Dynamic**: done frequently at runtime
  - task queue
  - Good for irregular, unpredictable execution times
  - usually high overhead

- **Semi-static**: done once or occasionally at run-time
Static Load Balancing

- In general, each process executes approx. \( N/P \) chunks for a loop of length \( N \) and \( P \) processes.

- Distribute iterations in blocks of size "chunk" over the processes in a round-robin fashion.

- A loop of length 16 on 4 processors:
  - General (chunk=4): 1-4 5-8 9-12 13-16
  - chunk = 2: 1-2 3-4 5-6 7-8 9-10 11-12 13-14 15-16
Dynamic Load Balancing

- Split a work into fixed portions; size is controlled by the value of chunk
- When a process finishes, it starts on the next portion of work controlled by a dynamic scheduler
- A loop of length 16 on 4 processors
  - Chunk=2: 1-2 3-4 5-6 7-8
  - 9-10 11-12 13-14 15-16
Partitioning and Load Balancing

Iteration Number

0 50 100 150 200 250 300 350 400 450 500

dynamic, 5

static
Partitioning 2D Mesh (FEM)
Partitioning 3D Mesh

1-D

2-D

3-D
Surface and Communication
(128x128x128 on 16 processors)

<table>
<thead>
<tr>
<th>Partition Size per Processor</th>
<th>group1</th>
<th>group2</th>
<th>group3</th>
<th>group4</th>
</tr>
</thead>
<tbody>
<tr>
<td>128x128x8</td>
<td>128x64x16, 128x16x64</td>
<td>32x32x128</td>
<td>64x64x32</td>
<td></td>
</tr>
<tr>
<td>128x8x128</td>
<td>64x128x16, 64x16x128</td>
<td>32x128x32</td>
<td>64x32x64</td>
<td></td>
</tr>
<tr>
<td>8x128x128</td>
<td>16x128x64, 16x64x128</td>
<td>128x32x32</td>
<td>32x64x64</td>
<td></td>
</tr>
<tr>
<td>Surface Area</td>
<td>36864</td>
<td>22528</td>
<td>18432</td>
<td>16384</td>
</tr>
</tbody>
</table>

1D  2D  2D  3D

The surface area is related to communication amount
Execution Models for Running Multiple Processes

- **SPMD: Single Program Multiple Data**
  - There is only one program
  - Each process uses the same m-file working on different sets of data

- **MPMD: Multiple Programs Multiple Data**
  - Different programs for different processes
  - The processes collaborate to solve the same problem
Examples: SPMD, MPMD

1. SPMD
2. MPMD: Master/Worker
3. MPMD: Coupled Analysis
# How to Measure Execution Time

<table>
<thead>
<tr>
<th>Timer</th>
<th>Usage</th>
<th>Wallclock/</th>
<th>Resolution</th>
<th>Languages</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>CPU Time</td>
<td></td>
<td>any</td>
</tr>
<tr>
<td><code>time</code></td>
<td>shell script</td>
<td>both</td>
<td>1/100th second</td>
<td>any</td>
</tr>
<tr>
<td><code>gettimeofday</code></td>
<td>subroutine</td>
<td>wallclock</td>
<td>micro-second</td>
<td>C/C++</td>
</tr>
<tr>
<td><code>MPI_Wtime</code></td>
<td>subroutine</td>
<td>wallclock</td>
<td>micro-second</td>
<td>C/C++, Fortran</td>
</tr>
<tr>
<td><code>tic, toc</code></td>
<td>Matlab script</td>
<td>wallclock</td>
<td>second</td>
<td>Matlab, Octave</td>
</tr>
</tbody>
</table>
Matlab/Octave tic, toc

- tic starts clock to measure the time
- toc stops the clock to measure the time elapsed in seconds
- toc displays the time elapsed since the tic command
Example: use of tic and toc

% sequential summation using Matlab/Octave

tic;
n = 100000000;
k = 1000;
s = 0;
for i=1:n
    s = s+i;
    for j=1:k
        j++;
    end
end
t1 = toc;
disp(['Total summation: ', num2str(s), ' takes ', num2str(t1), ' seconds']);
Outline

- Parallel Matlab/Octave
  - Matlab, Octave
  - Parallel Computing Toolbox
  - MatlabMPI, pMatlab

- IAMCS Clusters: Brazos and Hurr

- Try to parallelize your own code

- Q&A
IAMCS Clusters: Brazos/Hurr

- Total of 300 nodes, 2400 cores

- Brazos:
  - 172 nodes, each with two quad-core Intel Xeon (Harpertown) processors with 2.5GHz and 16GB per node

- Hurr:
  - 128 nodes, each with two quad core AMD Opteron (Shanghai) processors with 2.5GHz and 32GB per node
Remote Login Brazos

- ssh username@brazos.tamu.edu
- See brazos.tamu.edu for the details
- Setup Environments (.bash_profile)
  - Matlab/Octave, R
  - MatlabMPI
  - Brazos: only two Matlab licenses
- Examples:
  - cp –r /data/wuxf/examples/MatlabMPI
List in MatlabMPI

- **doc/** original MatlabMPI document
- **examples/** original MatlabMPI examples
- **matlab/** directory for MatlabMPI configuration to support Matlab
- **.octaverc** MatlabMPI configuration file to support Octave
- **README** original MatlabMPI readme
- **readme.brazos** readme for Brazos/Hurr
- **README.pc** original file
- **src/** original MatlabMPI source files
- **tests/** test examples and pbs scripts to run the codes on Brazos/Hurr
Configure MatlabMPI to support MPI + Matlab codes

- To add the path of MatlabMPI to Matlab, copy the directory matlab to your home directory as follows:
  \[ \text{cp -r matlab ~} \]

- Then, edit \texttt{matlab/startup.m} to change \\texttt{"/data/wuxf/examples/MatlabMPI/src"} to \\texttt{"yourdirectory/MatlabMPI/src"}, and \\texttt{"/home/wuxf/matlab"} to \\texttt{"yourhomedirectory/matlab"}.

- Test your MatlabMPI codes in Matlab environment
Configure MatlabMPI to support MPI + Octave codes

- To add the path of MatlabMPI to Octave, copy the file .octaverc to your home directory as follows:

  cp .octaverc ~

- Then, edit the file .octaverc to change "/data/wuxf/examples/MatlabMPI/src" to "yourdirectory/MatlabMPI/src", and "/home/wuxf/matlab" to "yourhomedirectory/matlab".

- Test your MatlabMPI codes in Octave environment
Running the Code

- **Interactive Mode**
  - What most of us are used to
  - Like we are "interacting" with the computer
  - Use command line to run the program immediately

- **Batch Mode**
  - The programs (jobs) are not performed immediately, but instead, put onto a queue to execute later
  - Use a PBS scheduler to schedule a job to run
Running a Code (interactive)

- In Matlab environment

```
[wuxf@brazos:/data/wuxf/examples/MatlabMPI/tests]$ matlab
>> eval( MPI_Run('hello',8,{}) );
Launching MPI rank: 7 on: brazos.tamu.edu
Launching MPI rank: 6 on: brazos.tamu.edu
Launching MPI rank: 5 on: brazos.tamu.edu
Launching MPI rank: 4 on: brazos.tamu.edu
Launching MPI rank: 3 on: brazos.tamu.edu
Launching MPI rank: 2 on: brazos.tamu.edu
Launching MPI rank: 1 on: brazos.tamu.edu
Launching MPI rank: 0 on: brazos.tamu.edu
unix_launch =
/bin/sh ./MatMPI/Unix_Commands.brazos.tamu.edu.0.sh &
Hello world from process : 0 of 8
>>
```
MPI_Run: Run MatlabMPI m_file

- Launches a Matlab script (m_file) on multiple processors

MPI_Run( m_file, n, machines )

- Runs n copies of m_file on machines, where
  - machines = {} means to run on n local / reserved processors
  - machines = {'machine1' 'machine2' ... 'machinen'} means to run on n processors: machine1, ..., and machinen.
Running a Code (interactive)

- In Matlab environment

Note: Before running another program basic.m, execute MatMPI_Delete_all.

```
>> MatMPI_Delete_all;
>> eval( MPI_Run('basic',2,{} ) );
```

Launching MPI rank: 1 on: brazos.tamu.edu
Launching MPI rank: 0 on: brazos.tamu.edu

unix_launch =
/bin/sh ./MatMPI/Unix_Commands.brazos.tamu.edu.0.sh &
my_rank: 0
SUCCESS

>>
Running a Code (interactive)

- In Octave environment

```
[wuxf@brazos:/data/wuxf/examples/MatlabMPI/tests]$ octave
octave:1> eval(MPI_Run('hello', 8, {}));
Launching MPI rank: 7 on: brazos.tamu.edu
Launching MPI rank: 6 on: brazos.tamu.edu
Launching MPI rank: 5 on: brazos.tamu.edu
Launching MPI rank: 4 on: brazos.tamu.edu
Launching MPI rank: 3 on: brazos.tamu.edu
Launching MPI rank: 2 on: brazos.tamu.edu
Launching MPI rank: 1 on: brazos.tamu.edu
Launching MPI rank: 0 on: brazos.tamu.edu
unix_launch = /bin/sh ./MatMPI/Unix_Commands.brazos.tamu.edu.0.sh &
Hello world from process : 0 of 8
octave:2>
```
Running a Code (interactive)

- In Octave environment

Note: Before running another program `basic.m`, execute `MatMPI_Delete_all`.

```
octave:2> MatMPI_Delete_all
remove entire contents of ./MatMPI? (yes or no) yes
octave:3> eval(MPI_Run('basic',2, {}));
Launching MPI rank: 1 on: brazos.tamu.edu
Launching MPI rank: 0 on: brazos.tamu.edu
unix_launch = /bin/sh ./MatMPI/
Unix_Commands.brazos.tamu.edu.0.sh &
my_rank: 0
SUCCESS
octave:4>
```
Running a Code (batch)

- In Matlab environment (run.pbs)

```bash
#!/bin/sh
#PBS -l walltime=00:10:00
#PBS -q iamcs
#PBS -l nodes=2:ppn=8
#PBS -j oe
#PBS -N matlabmpi-hello
Nprocs=16
### go to your working directory
cd /data/wuxf/examples/MatlabMPI/tests
### copy the node list file to your working directory
cp $PBS_NODEFILE ./.machines.m
### run the program
/apps/matlab/R2010b/bin/matlab -nojvm -nosplash -display null < run-hello.m
```
machines.m

- reserved machine list file

c0533b
c0533b
c0533b
c0533b
c0533b
c0533b
c0533b
c0533b
c0615a
c0615a
c0615a
c0615a
c0615a
c0615a
c0615a
c0615a
c0615a
Running a Code (batch)

- In Matlab environment (run-hello.m)

  % Read the node list file into a array
  machines = textread('machines.m','%s','delimiter','\n','whitespace','"');
  % Re-arrange the dimension of the nodes to 1xn
  machines2 = reshape(machines,1,[]);
  % Get the length of node list
  node_num = size(machines);
  % Clean up matlabmpi
  MatMPI_Delete_all
  % Run your program
  eval(MPI_Run('hello',node_num(1) , machines2));

- Submit a MatlabMPI job
  - qsub run.pbs
Running a Code (batch)

In Octave environment (orun.pbs)

```bash
#!/bin/sh
#PBS -l walltime=00:10:00
#PBS -q iamcs
#PBS -l nodes=4:ppn=8
#PBS -j oe
#PBS -N octavempi-hello
Nprocs=32
### go to your working directory
cd /data/wuxf/examples/MatlabMPI/tests
### copy the node list file to your working directory
cp $PBS_NODEFILE ./machines.m

### run the program
/apps/octave-3.2.4/bin/octave < ohello.m
```
In Octave environment (ohello.m)

% Read the node list file into an array
machines = textread('machines.m','%s','delimiter','\n','whitespace','"');

% Re-arrange the dimension of the nodes to 1xn
machines2 = reshape(machines,1,[]);

% Get the length of node list
node_num = size(machines);

% Clean up matlabmpi
MatMPI_Delete_all

% Run your program
eval(MPI_Run('hello',node_num(1), machines2));

Submit a MatlabMPI job

`qsub orun.pbs`
Submit a batch job

- TORQUE Resource Manager provides control over batch jobs and distributed computing resources.
- It is an advanced open-source product based on the OpenPBS.

- Submit a batch job
  - `qsub file.pbs`

- Look at the job status
  - `qstat -u username`
  - `qdel jobid` (delete the job with jobid)
Outline

- Parallel Matlab/Octave
  - Matlab, Octave
  - Parallel Computing Toolbox
  - MatlabMPI

- IAMCS Clusters: Brazos and Hurr

- Try to parallelize your own code

- Q&A
Case Study: Parallel 4D CT Imaging Processing

- Optical flow applications using traditional Lucas-Kanade and Horn-Schunck methods
- The original sequential source code is written in Matlab from Dr. Ding
- It requires large memory and takes a long time to simulate
- We use MatlabMPI to parallelize them
Datasets ([http://pubimage.hcuge.ch:8080/DATA/MAGIX.zip](http://pubimage.hcuge.ch:8080/DATA/MAGIX.zip))

Alias Name: MAGIX, Modality: CT 64, File Size: 69 MB, Description: 4D Cardiac CT.

4D Cardiac CT Imaging (512x512x76x1)
A Slice of the 4D Cardiac CT Image

Sensation 64
2006/10/19 14:37:39
Cardiaque(Cardiaque_standard (Adulte))
A10026587077
Cir CardiacCirc 3.0 E20f 0-90% STANDARD_DECLEINEMENT 30 %
Series: 10
Instance: 457
Image Size: 512 x 512
Zoom: 98 Angle: 0

TP60PC1123
FOV: 206.00 mm
Thk: 3.00 mm
Loc: 82.50 mm
0.40/0.40 mm
E20f
120.00 kVp
550 mA

http://prophesy.cs.tamu.edu
Execution Time (s) of Parallel 4D CT Applications on SystemG

Execution Time of 4D CT on SystemG

Data-6x6x3

Data-3x3x1
Useful Links

- **Matlab**: [http://www.mathworks.com](http://www.mathworks.com)
- **IAMCS Matlab technical seminars** from the IAMCS website
Download the Tutorial Slides

IAMCS Parallel Programming
Tutorial 2: Parallel R

Xingfu Wu

Department of Computer Science & Engineering
Texas A&M University

Blocker 503, 2:00pm, April 5, 2013