Matlab MPI
Parallel Programming with MatlabMPI
Reference Manual

By
Dr. Jeremy Kepner

Document Generated by Universal Report www.omegacomputer.com
Abstract

Matlab is the dominate 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 a 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 "look and feel" on top of standard Matlab file i/o, resulting in a "pure" Matlab implementation that is exceedingly small (100 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).

REQUIREMENTS
- Matlab license
- File system visible to all processors

On shared memory systems, MatlabMPI only requires a single Matlab license as each user is allowed to have many Matlab sessions. On distributed memory systems, MatlabMPI will require one Matlab license per machine. Because MatlabMPI uses file i/o for communication, there must be a directory that is visible to every machine (this is usually also required in order to install Matlab). This directory defaults to the directory that the program is launched from, but can be changed within the MatlabMPI program.
Contents

1 Global Information ................................................. 5
2 Quantitative Information ........................................... 5
3 List of Files ......................................................... 6
4 List of Routines ..................................................... 7
5 Routines Description ................................................ 8
  5.1 MatMPI_Buffer_file() ............................................ 8
  5.2 MatMPI_Comm_dir() ............................................... 8
  5.3 MatMPI_Comm_init() .............................................. 8
  5.4 MatMPI_Comm_settings() ........................................ 9
  5.5 MatMPI_Commands() ............................................. 9
  5.6 MatMPI_Delete_all() ........................................... 10
  5.7 MatMPI_lock_file() ............................................ 10
  5.8 MatMPI_Save_messages() ..................................... 10
  5.9 MPI_Abort() ..................................................... 11
  5.10 MPI_Bcast() .................................................... 11
  5.11 MPI_Comm_rank() ............................................. 12
  5.12 MPI_Finalize() ................................................. 12
  5.13 MPI_Init() ..................................................... 13
  5.14 MPI_Probe() .................................................... 13
  5.15 MPI_Recv() .................................................... 14
  5.16 MPI_Run() ..................................................... 14
  5.17 MPI_Send() ..................................................... 15
6 ROUTINES BODY ..................................................... 16
  6.1 MatMPI_Buffer_file() ............................................ 16
  6.2 MatMPI_Comm_dir() ............................................... 16
  6.3 MatMPI_Comm_init() .............................................. 16
  6.4 MatMPI_Comm_settings() .................................... 18
  6.5 MatMPI_Commands() ............................................. 19
  6.6 MatMPI_Delete_all() ........................................... 21
  6.7 MatMPI_lock_file() ............................................ 22
  6.8 MatMPI_Save_messages() ..................................... 22
  6.9 MPI_Abort() ..................................................... 23
  6.10 MPI_Bcast() .................................................... 24
  6.11 MPI_Comm_rank() ............................................. 26
  6.12 MPI_Finalize() ................................................ 26
  6.13 MPI_Init() ..................................................... 27
  6.14 MPI_Probe() .................................................... 27
  6.15 MPI_Recv() .................................................... 28
  6.16 MPI_Run() ..................................................... 29
6.17 MPI_Send() ........................................... 32
1 Global Information

<table>
<thead>
<tr>
<th>Project Name</th>
<th>Matlab MPI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project Owner</td>
<td>Dr. Jeremy Kepner</td>
</tr>
<tr>
<td>Starting Date</td>
<td></td>
</tr>
<tr>
<td>Ending Date</td>
<td></td>
</tr>
<tr>
<td>Programming Environment</td>
<td>Matlab</td>
</tr>
<tr>
<td>Technical Team</td>
<td>Dr. Jeremy Kepner</td>
</tr>
<tr>
<td>Overview Comment</td>
<td>Parallel Programming with MatlabMPI</td>
</tr>
</tbody>
</table>

Table 1: General Information

2 Quantitative Information

<table>
<thead>
<tr>
<th>Total number of files</th>
<th>19 file(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total number of routines</td>
<td>17 line(s)</td>
</tr>
<tr>
<td>Total number of lines</td>
<td>1257 line(s)</td>
</tr>
<tr>
<td>Total size</td>
<td>44 Kbyte(s)</td>
</tr>
</tbody>
</table>

Table 2: Quantitative Information
3 List of Files

<table>
<thead>
<tr>
<th>N</th>
<th>File Name</th>
<th>Location</th>
<th>Line(s)</th>
<th>Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>MatlabMPI.m</td>
<td>MatlabMPI</td>
<td>69</td>
<td>3296</td>
</tr>
<tr>
<td>2</td>
<td>MatMPI_Buffer_file.m</td>
<td>MatlabMPI</td>
<td>36</td>
<td>1566</td>
</tr>
<tr>
<td>3</td>
<td>MatMPI_Comm_dir.m</td>
<td>MatlabMPI</td>
<td>41</td>
<td>1488</td>
</tr>
<tr>
<td>4</td>
<td>MatMPI_Comm_init.m</td>
<td>MatlabMPI</td>
<td>112</td>
<td>4084</td>
</tr>
<tr>
<td>5</td>
<td>MatMPI_Comm_settings.m</td>
<td>MatlabMPI</td>
<td>60</td>
<td>2528</td>
</tr>
<tr>
<td>6</td>
<td>MatMPI_Commands.m</td>
<td>MatlabMPI</td>
<td>105</td>
<td>3935</td>
</tr>
<tr>
<td>7</td>
<td>MatMPI_Delete_all.m</td>
<td>MatlabMPI</td>
<td>66</td>
<td>2087</td>
</tr>
<tr>
<td>8</td>
<td>MatMPI_Lock_file.m</td>
<td>MatlabMPI</td>
<td>37</td>
<td>1547</td>
</tr>
<tr>
<td>9</td>
<td>MatMPI_Save_messages.m</td>
<td>MatlabMPI</td>
<td>42</td>
<td>1643</td>
</tr>
<tr>
<td>10</td>
<td>MPI_Abort.m</td>
<td>MatlabMPI</td>
<td>79</td>
<td>2503</td>
</tr>
<tr>
<td>11</td>
<td>MPI_Bcast.m</td>
<td>MatlabMPI</td>
<td>127</td>
<td>3806</td>
</tr>
<tr>
<td>12</td>
<td>MPI_Comm_rank.m</td>
<td>MatlabMPI</td>
<td>33</td>
<td>1341</td>
</tr>
<tr>
<td>13</td>
<td>MPI_Comm_size.m</td>
<td>MatlabMPI</td>
<td>33</td>
<td>1350</td>
</tr>
<tr>
<td>14</td>
<td>MPI_Finalize.m</td>
<td>MatlabMPI</td>
<td>34</td>
<td>1355</td>
</tr>
<tr>
<td>15</td>
<td>MPI_Init.m</td>
<td>MatlabMPI</td>
<td>34</td>
<td>1346</td>
</tr>
<tr>
<td>16</td>
<td>MPI_Probe.m</td>
<td>MatlabMPI</td>
<td>89</td>
<td>2943</td>
</tr>
<tr>
<td>17</td>
<td>MPI_Recv.m</td>
<td>MatlabMPI</td>
<td>64</td>
<td>2149</td>
</tr>
<tr>
<td>18</td>
<td>MPI_Run.m</td>
<td>MatlabMPI</td>
<td>145</td>
<td>4743</td>
</tr>
<tr>
<td>19</td>
<td>MPI_Send.m</td>
<td>MatlabMPI</td>
<td>51</td>
<td>1856</td>
</tr>
<tr>
<td></td>
<td>TOTAL</td>
<td></td>
<td>1257</td>
<td>45566</td>
</tr>
</tbody>
</table>

Table 3: List of Files
4 List of Routines

<table>
<thead>
<tr>
<th>N</th>
<th>Routine Name</th>
<th>Lines</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>MatMPI_Buffer_file()</td>
<td>36 lines</td>
</tr>
<tr>
<td>2</td>
<td>MatMPI_Comm_dir()</td>
<td>41 lines</td>
</tr>
<tr>
<td>3</td>
<td>MatMPI_Comm_init()</td>
<td>112 lines</td>
</tr>
<tr>
<td>4</td>
<td>MatMPI_Comm_settings()</td>
<td>60 lines</td>
</tr>
<tr>
<td>5</td>
<td>MatMPI_Commands()</td>
<td>105 lines</td>
</tr>
<tr>
<td>6</td>
<td>MatMPI_Delete_all()</td>
<td>66 lines</td>
</tr>
<tr>
<td>7</td>
<td>MatMPI_lock_file()</td>
<td>37 lines</td>
</tr>
<tr>
<td>8</td>
<td>MatMPI_Save_messages()</td>
<td>42 lines</td>
</tr>
<tr>
<td>9</td>
<td>MPI_Abort()</td>
<td>79 lines</td>
</tr>
<tr>
<td>10</td>
<td>MPI_Bcast()</td>
<td>127 lines</td>
</tr>
<tr>
<td>11</td>
<td>MPI_Comm_rank()</td>
<td>33 lines</td>
</tr>
<tr>
<td>12</td>
<td>MPI_Finalize()</td>
<td>34 lines</td>
</tr>
<tr>
<td>13</td>
<td>MPI_Init()</td>
<td>34 lines</td>
</tr>
<tr>
<td>14</td>
<td>MPI_Probe()</td>
<td>89 lines</td>
</tr>
<tr>
<td>15</td>
<td>MPI_Recv()</td>
<td>64 lines</td>
</tr>
<tr>
<td>16</td>
<td>MPI_Run()</td>
<td>145 lines</td>
</tr>
<tr>
<td>17</td>
<td>MPI_Send()</td>
<td>51 lines</td>
</tr>
</tbody>
</table>

Table 4: List of Routines
5 Routines Description

5.1 MatMPI_Buffer_file()

Routine Name     MatMPI_Buffer_file()
Routine Location  MatlabMPI\MatMPI_Buffer_file.m
Routine Objective
Routine Arguments { buffer_file, comm, dest, source, tag }
Routine Outputs  { buffer_file }
Routine Size      36 Line(s)
Routine Author
Routine Date
Routine Comment
MatMPI_Buffer_file - Helper function for creating buffer file name.

buffer_file = MatMPI_Buffer_file(source,dest,tag,comm)

Parent Routines
• MPI_Bcast()
• MPI_Recv()
• MPI_Send()

Child Routines

5.2 MatMPI_Comm_dir()

Routine Name     MatMPI_Comm_dir()
Routine Location  MatlabMPI\MatMPI_Comm_dir.m
Routine Objective
Routine Arguments { dir, new_comm, old_comm }
Routine Outputs  { new_comm }
Routine Size      41 Line(s)
Routine Author
Routine Date
Routine Comment
MatMPI_Comm_dir - function for changing communication directory.

new_comm = MatMPI_Comm_dir(old_comm,dir)

Parent Routines
Child Routines

5.3 MatMPI_Comm_init()

Routine Name     MatMPI_Comm_init()
Routine Location  MatlabMPI\MatMPI_Comm_init.m
Routine Objective
Routine Arguments { machines, MPI_COMM_WORLD, n_proc }
Routine Outputs  { MPI_COMM_WORLD }

Parent Routines
Child Routines
Routine Size: 112 Line(s)
Routine Author
Routine Date
Routine Comment
MatMPI_Comm_init - Creates generic communicator.

\[ \text{MPI COMM WORLD} = \text{MatMPI Comm init(n_proc, machines)} \]

**Parent Routines**
- MPI_Run()

**Child Routines**
- MatMPI_Comm_settings()

### 5.4 MatMPI_Comm_settings()

**Routine Name** MatMPI_Comm_settings()
**Routine Location** MatlabMPI\MatMPI_Comm_settings.m
**Routine Objective** Function for setting values in the MPI Communicator.
**Routine Arguments**
\{ machine_db_settings \}
**Routine Outputs**
\{ machine_db_settings \}
**Routine Size** 60 Line(s)

**Routine Comment**
User can edit these values to customize the internals MatlabMPI.

**Parent Routines**
- MatMPI_Comm_init()
- MPI_Abort()

**Child Routines**

### 5.5 MatMPI_Commands()

**Routine Name** MatMPI_Commands()
**Routine Location** MatlabMPI\MatMPI_Commands.m
**Routine Objective**
MatMPI_Commands - Commands to launch a matlab script remotely.
**Routine Arguments**
\{ defcommands, m_file, MPI_COMM_WORLD, rank, unix_command \}
**Routine Outputs**
\{ defcommands, unix_command \}
**Routine Size** 105 Line(s)

**Routine Comment**

\[[\text{defcommands, unix_command}] = ...\]
MatMPI_Commands(m_file,rank,MPI_COMM_WORLD)

Parent Routines
• MPI_Run()

Child Routines

5.6 MatMPI_Delete_all()

<table>
<thead>
<tr>
<th>Routine Name</th>
<th>MatMPI_Delete_all()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Routine Location</td>
<td>MatlabMPI\MatMPI_Delete_all.m</td>
</tr>
</tbody>
</table>

Routine Objective
Routine Arguments
Routine Outputs
Routine Size  66 Line(s)
Routine Author
Routine Date
Routine Comment
MatMPI_Delete_all - Deletes leftover MatlabMPI files.

MatMPI_Delete_all()
Parent Routines
Child Routines

5.7 MatMPI_lock_file()

<table>
<thead>
<tr>
<th>Routine Name</th>
<th>MatMPI_lock_file()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Routine Location</td>
<td>MatlabMPI\MatMPI_Lock_file.m</td>
</tr>
</tbody>
</table>

Routine Objective
Routine Arguments { comm, dest, lock_file, source, tag }
Routine Outputs { lock_file }
Routine Size  37 Line(s)
Routine Author
Routine Date
Routine Comment
MatMPI_lock_file - function for creating lock file name.

lock_file = MatMPI_lock_file(source,dest,tag,comm)
Parent Routines
Child Routines

5.8 MatMPI_Save_messages()

<table>
<thead>
<tr>
<th>Routine Name</th>
<th>MatMPI_Save_messages()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Routine Location</td>
<td>MatlabMPI\MatMPI_Save_messages.m</td>
</tr>
</tbody>
</table>

Routine Objective
Routine Arguments { new_comm, old_comm, save_message_flag }
Routine Outputs { new_comm }
MatMPI_Save_messages - Toggles deleting or saving messages.

new_comm = MatMPI_Save_messages(old_comm, save_message_flag)

MatlabMPI helper function for setting the fate of messages.

save_message_flag = 1 (save messages).
save_message_flag = 0 (delete messages: default).

5.9 MPI_Abort()

MPI_Abort - Aborts any currently running MatlabMPI sessions.

Will abort any currently running MatlabMPI sessions.
by looking for leftover Matlab jobs and killing them.
Cannot be used after MatMPI_Delete_all.

5.10 MPI_Bcast()

Routine Name    MPI_Bcast()
Routine Location MatlabMPI\MPI_Bcast.m
Routine Objective
Routine Arguments { comm, source, tag, varargin, varargout }
Routine Outputs  { varargout }
Routine Size     127 Line(s)
Routine Date
Routine Comment
Child Routines   • MatMPI_Comm_settings()
Routine Comment

MPI_Bcast - broadcast variables to everyone.

[var1, var2, ...] = ...
MPI_Bcast(source, tag, comm, var1, var2, ...)

Broadcast variables to everyone in comm.

Sender blocks until all the messages are received, unless MatMMPI_Save_messages(1) has been called.

Parent Routines
Child Routines
• MPI_Comm_rank()
• MPI_Recv()
• MatMMPI_Buffer_file()

5.11 MPI_Comm_rank()

Routine Name         MPI_Comm_rank()
Routine Location     MatlabMPI\MPI_Comm_rank.m
Routine Objective     Routine Arguments { comm, rank }
Routine Outputs       { rank }
Routine Size          33 Line(s)
Routine Comment       MPI_Comm_rank - returns the rank of the current processor.

rank = MPI_Comm_rank(comm)

Parent Routines
• MPI_Bcast()
• MPI_Probe()
• MPI_Recv()
• MPI_Send()

Child Routines

5.12 MPI_Finalize()

Routine Name         MPI_Finalize()
Routine Location     MatlabMPI\MPI_Finalize.m
Routine Objective     Routine Arguments
Routine Outputs       Routine Size            34 Line(s)
Routine Author
Routine Date
Routine Comment
MPI_Finalize - Called at the end of a MatlabMPI program.

MPI_Finalize()

Called at the end of an MPI program (currently empty).
Parent Routines
Child Routines

5.13 MPI_Init()

Routine Name       MPI_Init()
Routine Location   MatlabMPI\MPI_Init.m
Routine Objective
Routine Arguments
Routine Outputs
Routine Size       34 Line(s)
Routine Author
Routine Date
Routine Comment
MPI_Init - Called at the start of an MPI program.

MPI_Init()

Called at the beginning of an MPI program (currently empty).
Parent Routines
Child Routines

5.14 MPI_Probe()

Routine Name       MPI_Probe()
Routine Location   MatlabMPI\MPI_Probe.m
Routine Objective
Routine Arguments  { comm, message_rank, message_tag, source, tag }
Routine Outputs   { message_rank, message_tag }
Routine Size       89 Line(s)
Routine Author
Routine Date
Routine Comment
MPI_Probe - Returns a list of all messages waiting to be received.

[message_rank, message_tag] = MPI_Probe( source, tag, comm )

Source and tag can be an integer or a wildcard '*'.
Parent Routines
Child Routines

- MPI_Comm_rank()

5.15 MPIRecv()

Routine Name: MPIRecv()
Routine Location: MatlabMPI\MPIRecv.m
Routine Objective: Receives message from source.
Routine Arguments: 
{ comm, source, tag, varargout }
Routine Outputs: 
{ varargout }
Routine Size: 64 Line(s)
Routine Author: 
Routine Date: 
Routine Comment: 
MPIRecv - Receives message from source.

[var1, var2, ...] = MPIRecv( source, tag, comm )

Receives message from source with a given tag and returns the variables in the message.

source can be an integer from 0 to comm_size-1
tag can be any integer
comm is an MPI Communicator (typically a copy of MPI_COMM_WORLD)

Parent Routines
- MPI_Bcast()
Child Routines
- MPI_Comm_rank()
- MatMPI_Buffer_file()

5.16 MPIRun()

Routine Name: MPIRun()
Routine Location: MatlabMPI\MPIRun.m
Routine Objective: Runs n_proc copies of m_file on machines, where
Routine Arguments: 
{ defcommands, m_file, machines, n_proc }
Routine Outputs: 
Routine Size: 145 Line(s)
Routine Author: 
Routine Date: 
Routine Comment: 
MPIRun - Run m_file on multiple processors.

defcommands = MPIRun( m_file, n_proc, machines )

Runs n_proc copies of m_file on machines, where
machines = ;
Run on a local processor.

machines = 'machine1' 'machine2')
Run on a multi processors.

machines = 'machine1:dir1' 'machine2:dir2')
Run on a multi processors and communicate using via dir1 and dir2,
which must be visible to both machines.

If machine1 is the local cpu, then defscmds will contain
the commands that need to be run locally, via eval(defscmds).

Parent Routines
Child Routines • MatMPI_Comm_init()
• MatMPI_Commands()

5.17 MPI_Send()

Routine Name MPI_Send()
Routine Location MatlabMPI\MPI_Send.m
Routine Objective
Routine Arguments { comm, dest, tag, varargin }
Routine Outputs
Routine Size 51 Line(s)
Routine Author
Routine Date
Routine Comment
MPI_Send - Sends variables to dest.

MPI_Send( dest, tag, comm, var1, var2, ...)
Send message containing variables to dest with a given tag

dest can be an integer from 0 to comm.size-1
tag can be any integer
comm is an MPI Communicator (typically a copy of MPI_COMM_WORLD)
Parent Routines
Child Routines • MPI_Comm_rank()
• MatMPI_Buffer_file()
6 ROUTINES BODY

6.1 MatMPI_Buffer_file()
1: function buffer_file = MatMPI_Buffer_file(source,dest,tag,comm)
2: % MatMPI_Buffer_file - Helper function for creating buffer file name.
3: %
4: % buffer_file = MatMPI_Buffer_file(source,dest,tag,comm)
5: %
6: 
7: machine_id = comm.machine_id(1,dest+1);
8: dir = comm.machine_db.dir{1,machine_id};
9: buffer_file = [dir,'/p',num2str(source),'_p',num2str(dest),'_t',
10: num2str(tag),'_buffer.mat'];

6.2 MatMPI_Comm_dir()
1: function new_comm = MatMPI_Comm_dir(old_comm,dir)
2: % MatMPI_Comm_dir - function for changing communication directory.
3: %
4: % new_comm = MatMPI_Comm_dir(old_comm,dir)
5: %
6: new_comm = old_comm;
7: 
8: n = new_comm.machine_db.n_machine;
9: for i=1:n
10: new_comm.machine_db.dir{1,i} = dir;
11: end
12: end
13: 
14: new_comm;

6.3 MatMPI_Comm_init()
1: function MPI_COMM_WORLD = MatMPI_Comm_init(n_proc,machines)
2: % MatMPI_Comm_init - Creates generic communicator.
3: %
4: % MPI_COMM_WORLD = MatMPI_Comm_init(n_proc,machines)
5: %
6: 
7: % Get number of machines to launch on.
8: n_machines = size(machines,2);
9: n_m = max(n_machines,1);
10: 
11: % Set default target machine.
12: host = getenv('HOST');
13:  machine = host;
14:  
15:  \% Initialize comm.
16:  MPI\_COMM\_WORLD.rank = -1;
17:  MPI\_COMM\_WORLD.size = n\_proc;
18:  MPI\_COMM\_WORLD.save\_message\_flag = 0;
19:  MPI\_COMM\_WORLD.group = (1:n\_proc)-1;
20:  MPI\_COMM\_WORLD.machine\_id = zeros(1,n\_proc);
21:  
22:  \% Initialize machine database.
23:  machine\_db.n\_machine = n\_m; \% Number of machines.
24:  machine\_db.type = cell(1,n\_m); \% Unix or Windows.
25:  machine\_db.machine = cell(1,n\_m); \% Machine names.
26:  machine\_db.dir = cell(1,n\_m); \% Communication directory.
27:  machine\_db.matlab\_command = cell(1,n\_m); \% Matlab command.
28:  machine\_db.remote\_launch = cell(1,n\_m); \% Remote launch command.
29:  machine\_db.remote\_flags = cell(1,n\_m); \% Remote launch flags.
30:  machine\_db.n\_proc = zeros(1,n\_m); \% \# processes on this machine.
31:  machine\_db.id\_start = zeros(1,n\_m); \% Start index.
32:  machine\_db.id\_stop = zeros(1,n\_m); \% Stop index.
33:  
34:  \% Start setting up machine id.
35:  for i\_rank=0:n\_proc-1
36:     i\_machine = mod(i\_rank,n\_m) + 1;
37:     machine\_db.n\_proc(1,i\_machine) = machine\_db.n\_proc(1,i\_machine) + 1;
38:  end
39:  
40:  \% Get possibly user settings.
41:  machine\_db\_settings = MatMPI\_Comm\_settings;
42:  
43:  \% Set machine\_db values.
44:  for i=1:n\_m
45:     machine\_db.type{1,i} = machine\_db\_settings.type;
46:     machine\_db.machine{1,i} = host;
47:     machine\_db.dir{1,i} = [pwd '/MatMPI'];
48:     machine\_db.matlab\_command{1,i} = machine\_db\_settings.matlab\_command;
49:     machine\_db.remote\_launch{1,i} = machine\_db\_settings.remote\_launch;
50:     machine\_db.remote\_flags{1,i} = machine\_db\_settings.remote\_flags;
51:     if (i == 1)
52:         machine\_db.id\_start(1,i) = 1;
53:     else
54:         machine\_db.id\_start(1,i) = machine\_db.id\_start(1,i-1) + 1;
55:     end
56:     machine\_db.id\_stop(1,i) = machine\_db.id\_start(1,i) +
57:     machine\_db.n\_proc(1,i) -1;
58:  end
59: end
60:
61: id\_start = machine\_db.id\_start(1,i);
62: id\_stop = machine\_db.id\_stop(1,i);
63:
64: MPI\_COMM\_WORLD.machine\_id(1,id\_start:id\_stop) = i;
65:
66: \% Check if there is a machines list.
67: if (n\_machines > 0)
68: machine = machines{i};
69: machine\_db.machine{1,i} = machine;
70:
71: \% Check if there is a directory appended.
72: dir\_sep = findstr(machine,':');
73: if (dir\_sep)
74: machine\_piece = machine(1,1:dir\_sep-1);
75: dir\_piece = machine(1,(dir\_sep+1):end);
76: machine\_db.machine{1,i} = machine\_piece;
77: machine\_db.dir{1,i} = dir\_piece;
78: end
79: end
80: end
81:
82: \% Add machine\_db to communicator.
83: MPI\_COMM\_WORLD.machine\_db = machine\_db;
84:
85: \% Write out.
86: comm\_mat\_file = 'MatMPI/MPI\_COMM\_WORLD.mat';
87: save(comm\_mat\_file,'MPI\_COMM\_WORLD');

6.4 MatMPI\_Comm\_settings()
1: function machine\_db\_settings = MatMPI\_Comm\_settings()
2: \%
3: \% Function for setting values in the MPI Communicator.
4: \% User can edit these values to customize the internals
5: \% MatlabMPI.
6: \%
7:
8: \% Set to 'unix' or 'windows'.
9: \% 'windows' currently doesn't work.
10: machine\_db\_settings.type = 'unix';
11:
12: \% Matlab command and launch flags.
13:
14: \% Generic.
15: machine\_db\_settings.matlab\_command = ' matlab -display null -nojvm -
16: nosplash ';
17: 
18: % Lincoln cluster common.
19: % machine\_db\_settings.matlab\_command = ' /tools/matlab/bin/matlab -
20: display null -nojvm -nosplash ';
21: % Lincoln cluster local.
22: % machine\_db\_settings.matlab\_command = ' /local/matlab12.1/bin/matlab -
23: display null -nojvm -nosplash ';
24: % LCS Cluster local.
25: % machine\_db\_settings.matlab\_command = ' /usr/local/bin/matlab -display
26: null -nojvm -nosplash ';
27: % Boston University.
28: % machine\_db\_settings.matlab\_command = ' /usr/local/IT/matlab-6.1/bin/
29: matlab -display null -nojvm -nosplash ';
30: % MHPCC local copy.
31: % machine\_db.matlab\_command{1,i} = ' /scratch/tempest/users/kepner/
32: matlab6/bin/matlab -display null -nojvm -nosplash ';
33: 
34: % Remote launch command.
35: % To use ssh, change ' rsh ' to ' ssh ' in line below.
36: % machine\_db\_settings.remote\_launch = ' ssh ';
37: machine\_db\_settings.remote\_launch = ' rsh ';
38: 
39: % Remote launch flags.
40: machine\_db\_settings.remote\_flags = ' -n ';

6.5 MatMPI\_Commands()
1: function [defscmds, unix\_command] = ...
2: MatMPI\_Commands(m\_file,rank,MPI\_COMM\_WORLD)
3: % MatMPI\_Commands - Commands to launch a matlab script remotely.
4: %
5: [defscmds, unix\_command] = ...
6: MatMPI\_Commands(m\_file,rank,MPI\_COMM\_WORLD)
7: %
8: 
9: % Set newline string.
10: nl = sprintf('
');
11: 
12: % Create filename each Matlab job will run at startup.
13: defsbase = ['MatMPI/defs' num2str(rank)];
14: defsfile = [defscmds '.m'];
15: comm\_mat\_file = 'MatMPI/MPI\_COMM\_WORLD.mat';
16: outfile = ['MatMPI/' m\_file '.' num2str(rank) '.out'];
17:
% Get single quote character.
q = strrep(' ', ', ', ', ', ', '); 

% Create Matlab MPI setup commands.
commands{1} = ['global MPI_COMM_WORLD; ' nl];
commands{2} = ['load ' q comm_mat_file q '; ' nl];
commands{3} = ['MPI_COMM_WORLD.rank = ' num2str(rank) '; ' nl];
commands{4} = ['delete(' q defsfile q '); ' nl];
commands{5} = [m_file '; ' nl];

defscommands = ' ';

% Get name of host.
machine_id = MPI_COMM_WORLD.machine_id(1,rank+1);
machine = MPI_COMM_WORLD.machine_db.machine{1,machine_id};
remote_launch = MPI_COMM_WORLD.machine_db.remote_launch{1,machine_id};
remote_flags = MPI_COMM_WORLD.machine_db.remote_flags{1,machine_id};
matlab_command = MPI_COMM_WORLD.machine_db.matlab_command{1,machine_id};

disp(['Launching MPI rank: ' num2str(rank) ' on: ' machine]);

% Print name of machine we are launching on.
matlab_command = [matlab_command ' < ' defsfile ' > ' outfile ];

% Determine how to run script and where to send output.
host = getenv('HOST');
if (strcmp(machine,host))
if (rank == 0)
defscommands = [commands{1} commands{2} commands{3} commands{5}];
unix_command = nl;
else
% Run defsfile script interactively.
defscommands = [commands{1} commands{2} commands{3} commands{5}];
unix_command = nl;
else
% Write commands to a .m text file.
fid = fopen(defsfile,'wt');
n_command = size(commands,2);
for i_command=1:n_command
fwrite(fid,commands{i_command});
end
fclose(fid);
else
% Create command to run defsfile locally and pipe output to another
unix_command = [matlab_command ' & nl 'touch MatMPI/pid.' machine '.! nl];
end
else
% Write commands to a .m text file.
fid = fopen(defsfile,'wt');
n_command = size(commands,2);
for i_command=1:n_command
fwrite(fid,commands{i_command});
end
fclose(fid);
% Create command to rundefsfile locally and pipe output to another
unix_command = [matlab_command ' & nl 'touch MatMPI/pid.' machine '.! nl];
end

6.6 MatMPI_Delete_all()

function MatMPI_Delete_all()
% MatMPI_Delete_all - Deletes leftover MatlabMPI files.
% MatMPI_Delete_all()
% %
% % First load MPI_COMM_WORLD.
load 'MatMPI/MPI_COMM_WORLD.mat';
% Set newline string.
nl = sprintf('
');
% Get single quote character.
q = strrep(' ','\','}');
% Get number of machines.

n_m = MPI_COMM_WORLD.machine_db.n_machine;
% Loop backwards over each machine.
for i_m=n:-1:1

% Get number of processes to launch on this machine.
\n\texttt{\textcolor{blue}{n\_proc\_i\_m = MPI\_COMM\_WORLD.machine\_db.n\_proc(1,i\_m);}}
\nif (n\_proc\_i\_m >= 1)
\n% Get communication directory.
\texttt{\textcolor{blue}{comm\_dir = MPI\_COMM\_WORLD.machine\_db.dir(1,i\_m);}}
\n% Delete buffer and lock files in this directory.
\texttt{\textcolor{blue}{delete([\textcolor{red}{comm\_dir 'p\_p\_t\_buffer.mat'}]);}}
\texttt{\textcolor{blue}{delete([\textcolor{red}{comm\_dir 'p\_p\_t\_lock.mat'}]);}}
\nend
\nend
\n% Delete MatMPI directory.
\texttt{\textcolor{blue}{delete('MatMPI/*');}}
\texttt{\textcolor{blue}{delete('MatMPI');}}

\section{MatMPI\_lock\_file()}
\begin{verbatim}
1: function lock\_file = MatMPI\_lock\_file(source,dest,tag,comm)
2: % MatMPI\_lock\_file - function for creating lock file name.
3: %
4: % lock\_file = MatMPI\_lock\_file(source,dest,tag,comm)
5: %
6: machine\_id = comm.machine\_id(1,dest+1);
7: dir = comm.machine\_db.dir(1,machine\_id);
8: lock\_file = [dir,'/p',num2str(source),'p',num2str(dest),'t',num2str(tag),'_lock.mat'];
\end{verbatim}

\section{MatMPI\_Save\_messages()}
\begin{verbatim}
1: function new\_comm = MatMPI\_Save\_messages(old\_comm,save\_message\_flag)
2: % MatMPI\_Save\_messages - Toggles deleting or saving messages.
3: %
4: % new\_comm = MatMPI\_Save\_messages(old\_comm,save\_message\_flag)
5: %
6: % MatlabMPI helper function for setting the fate of messages.
7: % save\_message\_flag = 1 (save messages).
8: % save\_message\_flag = 0 (delete messages: default).
9: new\_comm = old\_comm;
10: new\_comm.save\_message\_flag = save\_message\_flag;
\end{verbatim}
new\_comm;

6.9 MPI\_Abort()

1: function MPI\_Abort()
2: % MPI\_Abort - Aborts any currently running MatlabMPI sessions.
3: %
4: % MPI\_Abort()
5: %
6: % Will abort any currently running MatlabMPI sessions.
7: % by looking for leftover Matlab jobs and killing them.
8: % Cannot be used after MatMPI\_Delete\_all.
9: %
10:
11: % Get possibly user defined settings.
12: machine\_db\_settings = MatMPI\_Comm\_settings;
13:
14: % Get list of pid files.
15: pid\_files = dir('MatMPI/pid.*.*');
16: s = size(pid\_files);
17: n\_files = s(1);
18:
19: % Create single quote.
20: q = strrep(' ','','');
21:
22: % Check if there are any files
23: if (n\_files < 1)
24: disp('No pid files found');
25: else
26:
27: % Loop over each file.
28: for i\_file=1:n\_files
29:
30: % Get file name.
31: file\_name = pid\_files(i\_file).name;
32:
33: % Check if there is a directory appended.
34: dir\_sep = findstr(file\_name,'.'));
35: if (dir\_sep)
36:
37: % Parse file name.
38: machine = file\_name(1,(dir\_sep(1)+1):(dir\_sep(end)-1));
39: pid = file\_name(1,(dir\_sep(end)+1):end);
40:
41: % Kill process.
42: % To use ssh, change 'rsh ' to 'ssh ' in line below.
43: % unix\_command = ['rsh ' machine ' ' q 'kill -9 ' pid q];
44: unix\_command = [ machine\_db\_settings.remote\_launch machine ' ' q 'kill -9 ' pid q];
45: disp(unix\_command);
46: unix(unix\_command);
47: end
48: 
49: end
50: 
51: end
52: 

6.10 MPI\_Bcast()

1: function varargout = MPI\_Bcast( source, tag, comm, varargin )
2: % MPI\_Bcast - broadcast variables to everyone.
3: %
4: % [var1, var2, ...] = ...
5: % MPI\_Bcast( source, tag, comm, var1, var2, ... )
6: %
7: % Broadcast variables to everyone in comm.
8: %
9: % Sender blocks until all the messages are received,
10: % unless MatMMPI\_Save\_messages(1) has been called.
11: %
12: 
13: % Get processor rank.
14: my\_rank = MPI\_Comm\_rank(comm);
15: comm\_size = MPI\_Comm\_size(comm);
16: 
17: % If not the source, then receive the data.
18: if (my\_rank ~= source)
19: varargout = MPI\_Recv( source, tag, comm );
20: end
21: 
22: % If the source, then send the data.
23: if (my\_rank == source)
24: 
25: % Create data file.
26: buffer\_file = MatMPI\_Buffer\_file(my\_rank,source,tag,comm);
27: 
28: % Save varargin to file.
29: save(buffer\_file,'varargin');
30: 
31: % Loop over everyone in comm and create link to data file.
32: link\_command = '';
for i=0:comm_size-1
  % Don't do source.
  if (i ~= source)
    % Create buffer link name.
    buffer_link = MatMPI_Buffer_file(my_rank,i,tag,comm);
    % Append to link_command.
    link_command = [link_command 'ln -s ' buffer_file ' ' buffer_link ' ' ']
    end
  end
end

% Create symbolic link to data_file.
% unix(link_command);

% Write commands unix commands to .sh text file
% to fix Matlab's problem with very long commands sent to unix().
unix_link_file = ['MatMPI/Unix_Link_Commands_t' num2str(tag) '.sh'];
fid = fopen(unix_link_file,'wt');
fwrite(fid,link_command);
fclose(fid);
unix(['/bin/sh ' unix_link_file]);
delete(unix_link_file);

% Loop over everyone in comm and create lock file.
for i=0:comm_size-1
  % Don't do source.
  if (i ~= source)
    % Get lock file name.
    lock_file = MatMPI_Lock_file(my_rank,i,tag,comm);
    % Create lock file.
    fclose(fopen(lock_file,'w'));
    end
  end
end

% Check if the message is to be saved.
if (not(comm.save_message_flag))
  % Loop over lock files.
  % Delete buffer_file when lock files are gone.
  % Loop over everyone in comm and create lock file.
  for i=0:comm_size-1
    % Don't do source.
if (i ~= source)
    % Get lock file name.
    lock_file = MatMPI\_Lock\_file(my\_rank,i,tag,comm);

% Spin on lock file until it is deleted.
loop = 0;
while exist(lock\_file) ~= 0
    % pause(0.01);
    loop = loop + 1;
end

% Delete buffer file.
if (not(comm.save\_message\_flag))
    delete(buffer\_file);
end

% Delete buffer file.
if (not(comm.save\_message\_flag))
    delete(buffer\_file);
end

6.11 MPI\_Comm\_rank()

function rank = MPI\_Comm\_rank(comm)
% MPI\_Comm\_rank - returns the rank of the current processor.
rank = comm.rank;

6.12 MPI\_Finalize()

function MPI\_Finalize()
% MPI\_Finalize - Called at the end of a MatlabMPI program.
% Called at the end of an MPI program (currently empty).
6.13  MPI_Init()

1: function MPI\_Init()
2: \% MPI\_Init - Called at the start of an MPI program.
3: \%
4: \% MPI\_Init()
5: \%
6: \% Called at the beginning of an MPI program (currently empty).
7: \%

6.14  MPI_Probe()

1: function [message\_rank, message\_tag] = MPI\_Probe( source, tag, comm )
2: \% MPI\_Probe - Returns a list of all messages waiting to be received.
3: \%
4: \% [message\_rank, message\_tag] = MPI\_Probe( source, tag, comm )
5: \%
6: \% Source and tag can be an integer or a wildcard ’*’.  
7: \%
8: \%
9: \% Get processor rank.
10: my\_rank = MPI\_Comm\_rank(comm);
11: \%
12: \% Get lock file names.
13: lock\_file = MatMPI\_Lock\_file(source,my\_rank,tag,comm);
14: \%
15: \% Check to see if there are any messages.
16: message\_files = dir(lock\_file);
17: n\_files = length(message\_files);
18: \%
19: \% Create single quote.
20: q = strrep(’ ’,’,’,’,’,’);
21: \%
22: \% Check if there are any files
23: if (n\_files < 1)
24: \% Set default (negative) return values.
25: message\_rank = ’’;
26: message\_tag = ’’;
27: else
28: \% Create arrays to store rank and tag.
29: message\_rank = zeros(n\_files,1);
30: message\_tag = message\_rank;
31: \%
32: \% Set strings to search for (THIS IS VERY BAD, SHOULD HIDE THIS).
33: source\_str = ’p’;
34: dest\_str = [’\_p’ num2str(my\_rank) ’\_t’];
tag\_str = '\_lock.mat';
source\_len = length(source\_str);
dest\_len = length(dest\_str);
tag\_len = length(tag\_str);

% Step through each file name and strip out rank and tag.
for i\_file=1:n\_files
  % Get file name.
  file\_name = message\_files(i\_file).name;
  % Find location of each of the strings.
  source\_pos = findstr(file\_name,source\_str);
dest\_pos = findstr(file\_name,dest\_str);
tag\_pos = findstr(file\_name,tag\_str);

  % If we have found the location than extract rank and tag.
  if (source\_pos & dest\_pos & tag\_pos)
    message\_rank(i\_file) = str2num(file\_name(1,(source\_len+
1):(dest\_pos-1)));
    message\_tag(i\_file) = str2num(file\_name(1,(dest\_pos+
dest\_len):(tag\_pos-1)));
  end
end

6.15 MPI\_Recv()

function varargout = MPI\_Recv( source, tag, comm )
% MPI\_Recv - Receives message from source.
% [var1, var2, ...] = MPI\_Recv( source, tag, comm )
% Receives message from source with a given tag
% and returns the variables in the message.
% source can be an integer from 0 to comm\_size-1
% tag can be any integer
% comm is an MPI Communicator (typically a copy of MPI\_COMM\_WORLD)
14: % Get processor rank.
15: my_rank = MPI_Comm_rank(comm);
16: % Get file names.
17: buffer_file = MatMPI_Buffer_file(source,my_rank,tag,comm);
18: lock_file = MatMPI_Lock_file(source,my_rank,tag,comm);
19: % Spin on lock file until it is created.
20: loop = 0;
21: while exist(lock_file) ~= 2
22:     loop = loop + 1;
23: end
24: % Read all data out of buffer_file.
25: buf = load(buffer_file);
26: % Delete buffer and lock files.
27: if (not(comm.save_message_flag))
28:     delete(buffer_file);
29:     delete(lock_file);
30: end
31: % Get variable out of buf.
32: varargout = buf.varargin;

6.16  MPI_Run()
1: function defscommands = MPI_Run( m_file, n_proc, machines )
2: % MPI_Run - Run m_file on multiple processors.
3: %
4: % defscommands = MPI_Run( m_file, n_proc, machines )
5: % Runs n_proc copies of m_file on machines, where
6: % machines = {`};
7: % Run on a local processor.
8: % machines = `{`machine1` , `machine2`} ;
9: % Run on a multi processors.
10: % machines = `{`machine1:dir1` , `machine2:dir2`} ;
11: % Run on a multi processors and communicate using via dir1 and dir2,
12: % which must be visible to both machines.
13: % If machine1 is the local cpu, then defscommands will contain
Parallel Programming with MatlabMPI

19: % the commands that need to be run locally, via eval(defscommands).
20: %
21:
22: % Check if the directory 'MatMPI' exists
23: if exist('MatMPI', 'dir') ~= 0
24: %error('MatMPI directory already exists: rename or remove with
25: MatMPI\_Delete\_all');
26: else
27: mkdir('MatMPI');
28: end
29:
30: % Create working directory.
31: % mkdir('MatMPI');
32:
33: % Get host.
34: host = getenv('HOST');
35:
36: % Get number of machines to launch on.
37: n\_machines = size(machines,2);
38:
39: % Create generic comm.
40: MPI\_COMM\_WORLD = MatMPI\_Comm\_init(n\_proc,machines);
41:
42: % Set newline string.
43: nl = sprintf('\n');
44: % Get single quote character.
45: q = strrep(' ', ' ', '');
46:
47: % Initialize unix command launch on all the different machines.
48: unix\_launch = '';
49:
50: % Get number of machines.
51: n\_m = MPI\_COMM\_WORLD.machine\_db.n\_machine;
52:
53: % Loop backwards over each machine.
54: for i\_m=n\_m:-1:1
55:
56: % Get number of processes to launch on this machine.
57: n\_proc\_i\_m = MPI\_COMM\_WORLD.machine\_db.n\_proc(1,i\_m);
58:
59: if (n\_proc\_i\_m >= 1)
60:
61: % Get machine info.
62: machine = MPI\_COMM\_WORLD.machine\_db.machine{1,i\_m};
63: remote\_launch = MPI\_COMM\_WORLD.machine\_db.remote\_launch{1,i\_m};
64: remote\_flags = MPI\_COMM\_WORLD.machine\_db.remote\_flags{1,i\_m};
% Get starting and stopping rank.
i_rank_start = MPI_COMM_WORLD.machine_db.id_start(1,i_m) - 1;
i_rank_stop = MPI_COMM_WORLD.machine_db.id_stop(1,i_m) - 1;

% Initialize unix command that will be run on each node.
unix_matlab = '';

% Loop backwards over number of processes.
for i_rank=i_rank_stop:-1:i_rank_start
    % Build commands
    [defscommands, unix_matlab_i_rank] = ...
        MatMPI_Commands(m_file,i_rank,MPI_COMM_WORLD);
    unix_matlab = [unix_matlab unix_matlab_i_rank];
end

% Create a file name.
% unix_matlab_file = ['MatMPI/Unix_Commands.' machine '.sh'];
unix_matlab_file = ['MatMPI/Unix_Commands.' num2str(i_rank_start) '.sh'];

% Append delete command.
unix_matlab = [unix_matlab ' rm ' unix_matlab_file ';' nl];

% Put commands in a file.
fid = fopen(unix_matlab_file,'wt');
fwrite(fid,unix_matlab);
fclose(fid);

% Create unix commands to launch this file.
if (strcmp(machine,host))
    unix_launch_i_m = ['/bin/sh ./' unix_matlab_file ' &' nl];
else
    unix_launch_i_m = [remote_launch machine remote_flags ...
            q 'cd ' pwd ' /bin/sh ./' unix_matlab_file ' &' q ' &' nl];
end
unix_launch = [unix_launch unix_launch_i_m];
end

% Execute all launches in a single unix call.
unix_launch

% unix(unix_launch);
% Write commands unix commands to .sh text file
% to fix Matlab’s problem with very long commands sent to unix().
unix_launch_file = 'MatMPI/Unix_Commands.sh';
fid = fopen(unix_launch_file,'wt');
fwrite(fid,unix_launch);
fclose(fid);
unix(['/bin/sh ' unix_launch_file]);
delete(unix_launch_file);

6.17 MPI_Send()

function MPI_Send( dest, tag, comm, varargin )

% MPI_Send - Sends variables to dest.

% Send message containing variables to dest with a given tag
% dest can be an integer from 0 to comm_size-1
% tag can be any integer
% comm is an MPI Communicator (typically a copy of MPI_COMM_WORLD)

% Get processor rank.
my_rank = MPI_Comm_rank(comm);

% Create buffer and lock file.
buffer_file = MatMPI_Buffer_file(my_rank,dest,tag,comm);
lock_file = MatMPI_Lock_file(my_rank,dest,tag,comm);

% Save buf to file.
save(buffer_file,'varargin');

% Create lock file.
fclose(fopen(lock_file,'w'));