

# Matlab MPI

Parallel Programming with MatlabMPI  
Reference Manual



By  
Dr. Jeremy Kepner

Document Generated by Universal Report [www.omegacomputer.com](http://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 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

<b>1</b>	<b>Global Information</b>	<b>5</b>
<b>2</b>	<b>Quantitative Information</b>	<b>5</b>
<b>3</b>	<b>List of Files</b>	<b>6</b>
<b>4</b>	<b>List of Routines</b>	<b>7</b>
<b>5</b>	<b>Routines Description</b>	<b>8</b>
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
<b>6</b>	<b>ROUTINES BODY</b>	<b>16</b>
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

<i>Parallel Programming with MatlabMPI</i>	4
6.17 MPI_Send() . . . . .	32

## 1 Global Information

<b>Project Name</b>	Matlab MPI
<b>Project Owner</b>	Dr. Jeremy Kepner
<b>Starting Date</b>	
<b>Ending Date</b>	
<b>Programming Environment</b>	Matlab
<b>Technical Team</b>	Dr. Jeremy Kepner
<b>Overview Comment</b>	Parallel Programming with MatlabMPI

Table 1: General Information

## 2 Quantitative Information

<b>Total number of files</b>	19 file(s)
<b>Total number of routines</b>	17 line(s)
<b>Total number of lines</b>	1257 line(s)
<b>Total size</b>	44 Kbyte(s)

Table 2: Quantitative Information

### 3 List of Files

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

Table 3: List of Files

## 4 List of Routines

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

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 }

**Routine Size**            112 Line(s)  
**Routine Author**  
**Routine Date**  
**Routine Comment**  
 MatMPI\_Comm\_init - Creates generic communicator.

MPI\_COMM\_WORLD = 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**  
**Routine Arguments** { machine\_db\_settings }  
**Routine Outputs**        { machine\_db\_settings }  
**Routine Size**            60 Line(s)  
**Routine Author**  
**Routine Date**  
**Routine Comment**

Function for setting values in the MPI Communicator.  
 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**  
**Routine Arguments** { defscmmands, m\_file, MPI\_COMM\_WORLD, rank,  
 unix\_command }  
**Routine Outputs**        { defscmmands, unix\_command }  
**Routine Size**            105 Line(s)  
**Routine Author**  
**Routine Date**  
**Routine Comment**

MatMPI\_Commands - Commands to launch a matlab script remotely.

[defscmmands, unix\_command] = ...

MatMPI\_Commands(m\_file,rank,MPI\_COMM\_WORLD)

**Parent Routines**

- MPI\_Run()

**Child Routines**

## 5.6 MatMPI\_Delete\_all()

**Routine Name** MatMPI\_Delete\_all()

**Routine Location** MatlabMPI\MatMPI\_Delete\_all.m

**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()

**Routine Name** MatMPI\_lock\_file()

**Routine Location** MatlabMPI\MatMPI\_Lock\_file.m

**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()

**Routine Name** MatMPI\_Save\_messages()

**Routine Location** MatlabMPI\MatMPI\_Save\_messages.m

**Routine Objective**

**Routine Arguments** { new\_comm, old\_comm, save\_message\_flag }

**Routine Outputs** { new\_comm }

**Routine Size** 42 Line(s)

**Routine Author**

**Routine Date**

**Routine Comment**

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).
```

**Parent Routines**

**Child Routines**

## 5.9 MPI\_Abort()

**Routine Name** MPI\_Abort()

**Routine Location** MatlabMPI\MPI\_Abort.m

**Routine Objective**

**Routine Arguments**

**Routine Outputs**

**Routine Size** 79 Line(s)

**Routine Author**

**Routine Date**

**Routine Comment**

MPI\_Abort - Aborts any currently running MatlabMPI sessions.

```
MPI_Abort()
```

Will abort any currently running MatlabMPI sessions.

by looking for leftover Matlab jobs and killing them.

Cannot be used after MatMPI\_Deleta.all.

**Parent Routines**

**Child Routines** • MatMPI\_Comm\_settings()

## 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 Author**

**Routine Date**

**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 MatMPI\_Save\_messages(1) has been called.

**Parent Routines**

- Child Routines**
- MPI\_Comm\_rank()
  - MPI\_Recv()
  - MatMPI\_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 Author**  
**Routine Date**  
**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 MPI\_Recv()

**Routine Name**        MPI\_Recv()  
**Routine Location**    MatlabMPI\MPI\_Recv.m  
**Routine Objective**  
**Routine Arguments** { comm, source, tag, varargout }  
**Routine Outputs**    { varargout }  
**Routine Size**        64 Line(s)  
**Routine Author**  
**Routine Date**  
**Routine Comment**

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)

**Parent Routines**

• MPI\_Bcast()

**Child Routines**      • MPI\_Comm\_rank()

• MatMPI\_Buffer\_file()

### 5.16 MPI\_Run()

**Routine Name**        MPI\_Run()  
**Routine Location**    MatlabMPI\MPI\_Run.m  
**Routine Objective**  
**Routine Arguments** { defscommands, m\_file, machines, n\_proc }  
**Routine Outputs**  
**Routine Size**        145 Line(s)  
**Routine Author**  
**Routine Date**  
**Routine Comment**

MPI\_Run - Run m\_file on multiple processors.

```
defscommands = MPI_Run( 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 defscmmands will contain the commands that need to be run locally, via eval(defscmmands).

#### Parent Routines

- Child Routines**
- MatMPIComm\_init()
  - MatMPICommands()

## 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**
- MPIComm\_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:
10: for i=1:n
11:     new\_comm.machine\_db.dir{1,i} = dir;
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:         machine\_db.id\_stop(1,i) = machine\_db.id\_start(1,i) +
54:         machine\_db.n\_proc(1,i) -1;
55:     else
56:         machine\_db.id\_start(1,i) = machine\_db.id\_stop(1,i-1) + 1;
57:         machine\_db.id\_stop(1,i) = machine\_db.id\_start(1,i) +
58:         machine\_db.n\_proc(1,i) -1;

```

```

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('\n');
11:
12: % Create filename each Matlab job will run at startup.
13: defsbse = ['MatMPI/defs' num2str(rank)];
14: defsfle = [defsbse '.m'];
15: comm\_mat\_file = 'MatMPI/MPI\_COMM\_WORLD.mat';
16: outfile = ['MatMPI/' m\_file '.' num2str(rank) '.out'];
17:

```

```

18: % Get single quote character.
19: q = strrep(' ' ' ', ' ', ' ');
20:
21: % Create Matlab MPI setup commands.
22: commands{1} = ['global MPI\_COMM\_WORLD;' nl];
23: commands{2} = ['load ' q comm\_mat\_file q '']; nl];
24: commands{3} = ['MPI\_COMM\_WORLD.rank = ' num2str(rank) '']; nl];
25: commands{4} = ['delete(' q defsfile q ')]; nl];
26: commands{5} = [m\_file '']; nl];
27:
28: defscommands = '';
29:
30: % Get name of host.
31: machine\_id = MPI\_COMM\_WORLD.machine\_id(1,rank+1);
32: machine = MPI\_COMM\_WORLD.machine\_db.machine{1,machine\_id};
33: remote\_launch = MPI\_COMM\_WORLD.machine\_db.remote\_launch{1,machine\_id};
34: remote\_flags = MPI\_COMM\_WORLD.machine\_db.remote\_flags{1,machine\_id};
35: matlab\_command = MPI\_COMM\_WORLD.machine\_db.matlab\_command{1,machine\_id};
36:
37: % Print name of machine we are launching on.
38: disp(['Launching MPI rank: ' num2str(rank) ' on: ' machine]);
39:
40: % Create base matlab command.
41: matlab\_command = [matlab\_command ' < ' defsfile ' > ' outfile ];
42: % matlab\_command = [matlab\_command ' -r ' defsfile ' -logfile ' outfile
43: ' > /def/null'];
44: % matlab\_command = [matlab\_command ' -r ' defsfile ' -logfile ' outfile
45: ];
46: % matlab\_command = [matlab\_command ' -r ' defsfile ' > ' outfile ];
47:
48: % Determine how to run script and where to send output.
49: host = getenv('HOST');
50: if (strcmp(machine,host))
51:     if (rank == 0)
52:         % Run defsfile script interactively.
53:         defscommands = [commands{1} commands{2} commands{3} commands{5}];
54:         unix\_command = nl;
55:     else
56:         % Write commands to a .m text file.
57:         fid = fopen(defsfile,'wt');
58:         n\_command = size(commands,2);
59:         for i\_command=1:n\_command
60:             fwrite(fid,commands{i\_command});
61:         end
62:         fclose(fid);
63:

```

```

64:         % Create command to run defsfile locally and pipe output to another
65:         file.
66:         unix\_command = [matlab\_command ' &' nl 'touch MatMPI/pid.' machine
67:         '.$!' nl];
68:     end
69: else
70:
71:     % Write commands to a .m text file.
72:     fid = fopen(defsfile,'wt');
73:     n\_command = size(commands,2);
74:     for i\_command=1:n\_command
75:         fwrite(fid,commands{i\_command});
76:     end
77:     fclose(fid);
78:
79:     % Create command to run defsfile locally and pipe output to another
80:     file.
81:     unix\_command = [matlab\_command ' &' nl 'touch MatMPI/pid.' machine
82:     '.$!' nl];
83:
84: end
85:

```

## 6.6 MatMPI\_Delete\_all()

```

1: function MatMPI\_Delete\_all()
2: % MatMPI\_Delete\_all - Deletes leftover MatlabMPI files.
3: %
4: % MatMPI\_Delete\_all()
5: %
6: %
7:
8: % First load MPI\_COMM\_WORLD.
9: load 'MatMPI/MPI\_COMM\_WORLD.mat';
10:
11: % Set newline string.
12: nl = sprintf('\n');
13: % Get single quote character.
14: q = strep(' ' ',' ',' ');
15:
16: % Get number of machines.
17: n\_m = MPI\_COMM\_WORLD.machine\_db.n\_machine;
18:
19: % Loop backwards over each machine.
20: for i\_m=n\_m:-1:1
21:

```

```

22:     % Get number of processes to launch on this machine.
23:     n\_proc\_i\_m = MPI\_COMM\_WORLD.machine\_db.n\_proc(1,i\_m);
24:
25:     if (n\_proc\_i\_m >= 1)
26:
27:         % Get communication directory.
28:         comm\_dir = MPI\_COMM\_WORLD.machine\_db.dir{1,i\_m};
29:
30:         % Delete buffer and lock files in this directory.
31:         delete([comm\_dir ' /p*\_p*\_t*\_buffer.mat']);
32:         delete([comm\_dir ' /p*\_p*\_t*\_lock.mat']);
33:     end
34:
35: end
36:
37: % Delete MatMPI directory.
38: delete('MatMPI/*');
39: delete('MatMPI');

```

## 6.7 MatMPI\_lock\_file()

```

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:
7:   machine\_id = comm.machine\_id(1,dest+1);
8:   dir = comm.machine\_db.dir{1,machine\_id};
9:   lock\_file = [dir,'/p',num2str(source),'\_p',num2str(dest),'\_t',
10:  num2str(tag),'\_lock.mat'];

```

## 6.8 MatMPI\_Save\_messages()

```

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:
10: new\_comm = old\_comm;
11:
12: new\_comm.save\_message\_flag = save\_message\_flag;

```

```

13:
14:   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\_Deleta\_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
45:         'kill -9' pid q];
46:         disp(unix\_command);
47:         unix(unix\_command);
48:     end
49:
50:     end
51:
52: end

```

## 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 = '';

```

```
33:     for i=0:comm\_size-1
34:         % Don't do source.
35:         if (i ~= source)
36:
37:             % Create buffer link name.
38:             buffer\_link = MatMPI\_Buffer\_file(my\_rank,i,tag,comm);
39:
40:             % Append to link\_command.
41:             link\_command = [link\_command 'ln -s ' buffer\_file ' ' buffer\_link
42:                 '; '];
43:         end
44:     end
45:
46:     % Create symbolic link to data\_file.
47:     %   unix(link\_command);
48:
49:     % Write commands unix commands to .sh text file
50:     % to fix Matlab's problem with very long commands sent to unix().
51:     unix\_link\_file = ['MatMPI/Unix\_Link\_Commands\_t' num2str(tag) '.sh'];
52:     fid = fopen(unix\_link\_file,'wt');
53:     fwrite(fid,link\_command);
54:     fclose(fid);
55:     unix(['/bin/sh ' unix\_link\_file]);
56:     delete(unix\_link\_file);
57:
58:     % Loop over everyone in comm and create lock file.
59:     for i=0:comm\_size-1
60:         % Don't do source.
61:         if (i ~= source)
62:             % Get lock file name.
63:             lock\_file = MatMPI\_Lock\_file(my\_rank,i,tag,comm);
64:
65:             % Create lock file.
66:             fclose(fopen(lock\_file,'w'));
67:         end
68:     end
69:
70:
71:     % Check if the message is to be saved.
72:     if (not(comm.save\_message\_flag))
73:
74:         % Loop over lock files.
75:         % Delete buffer\_file when lock files are gone.
76:         % Loop over everyone in comm and create lock file.
77:         for i=0:comm\_size-1
78:             % Don't do source.
```

```

79:         if (i ~= source)
80:             % Get lock file name.
81:             lock\_file = MatMPI\_Lock\_file(my\_rank,i,tag,comm);
82:
83:             % Spin on lock file until it is deleted.
84:             loop = 0;
85:             while exist(lock\_file) ~= 0
86: %                 pause(0.01);
87:                 loop = loop + 1;
88:             end
89:
90:         end
91:     end
92:
93:
94:     % Delete buffer file.
95:     if (not(comm.save\_message\_flag))
96:         delete(buffer\_file);
97:     end
98:
99: end
100:
101: end

```

### 6.11 MPI\_Comm\_rank()

```

1: function rank = MPI\_Comm\_rank(comm)
2: % MPI\_Comm\_rank - returns the rank of the current processor.
3: %
4: %   rank = MPI\_Comm\_rank(comm)
5: %
6:   rank = comm.rank;

```

### 6.12 MPI\_Finalize()

```

1: function MPI\_Finalize()
2: % MPI\_Finalize - Called at the end of a MatlabMPI program.
3: %
4: %   MPI\_Finalize()
5: %
6: %   Called at the end of an MPI program (currently empty).
7: %

```

### 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'];

```

```

35:     tag\_str = '\_lock.mat';
36:     source\_len = length(source\_str);
37:     dest\_len = length(dest\_str);
38:     tag\_len = length(tag\_str);
39:
40:     % Step through each file name and strip out rank and tag.
41:     for i\_file=1:n\_files
42:
43:         % Get file name.
44:         file\_name = message\_files(i\_file).name;
45:
46:         % Find location of each of the strings.
47:         source\_pos = findstr(file\_name,source\_str);
48:         dest\_pos = findstr(file\_name,dest\_str);
49:         tag\_pos = findstr(file\_name,tag\_str);
50:
51:         % If we have found the location than extract rank and tag.
52:         if (source\_pos & dest\_pos & tag\_pos)
53:
54:             message\_rank(i\_file) = str2num(file\_name(1,(source\_len+
55:             1):(dest\_pos-1)));
56:             message\_tag(i\_file) = str2num(file\_name(1,(dest\_pos+
57:             dest\_len):(tag\_pos-1)));
58:
59:         end
60:
61:     end
62:
63: end

```

## 6.15 MPI\_Recv()

```

1: function varargout = MPI\_Recv( source, tag, comm )
2: % MPI\_Recv - Receives message from source.
3: %
4: % [var1, var2, ...] = MPI\_Recv( source, tag, comm )
5: %
6: %     Receives message from source with a given tag
7: %     and returns the variables in the message.
8: %
9: %     source can be an iteger from 0 to comm\_size-1
10: %     tag can be any integer
11: %     comm is an MPI Communicator (typically a copy of MPI\_COMM\_WORLD)
12: %
13:

```

```

14: % Get processor rank.
15: my\_rank = MPI\_Comm\_rank(comm);
16:
17: % Get file names.
18: buffer\_file = MatMPI\_Buffer\_file(source,my\_rank,tag,comm);
19: lock\_file = MatMPI\_Lock\_file(source,my\_rank,tag,comm);
20:
21: % Spin on lock file until it is created.
22: loop = 0;
23: while exist(lock\_file) ~= 2
24:     loop = loop + 1;
25: end
26:
27: % Read all data out of buffer\_file.
28: buf = load(buffer\_file);
29:
30: % Delete buffer and lock files.
31: if (not(comm.save\_message\_flag))
32:     delete(buffer\_file);
33:     delete(lock\_file);
34: end
35:
36: % Get variable out of buf.
37: varargout = buf.varargin;

```

## 6.16 MPI\_Run()

```

1: function defscmmands = MPI\_Run( m\_file, n\_proc, machines )
2: % MPI\_Run - Run m\_file on multiple processors.
3: %
4: % defscmmands = MPI\_Run( m\_file, n\_proc, machines )
5: %
6: %     Runs n\_proc copies of m\_file on machines, where
7: %
8: %     machines = {};
9: %         Run on a local processor.
10: %
11: %     machines = {'machine1' 'machine2'}) );
12: %         Run on a multi processors.
13: %
14: %     machines = {'machine1:dir1' 'machine2:dir2'}) );
15: %         Run on a multi processors and communicate using via dir1 and dir2,
16: %         which must be visible to both machines.
17: %
18: %     If machine1 is the local cpu, then defscmmands will contain

```

```

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};

```

```

65:
66:     % Get starting and stopping rank.
67:     i\_rank\_start = MPI\_COMM\_WORLD.machine\_db.id\_start(1,i\_m) - 1;
68:     i\_rank\_stop = MPI\_COMM\_WORLD.machine\_db.id\_stop(1,i\_m) - 1;
69:
70:     % Initialize unix command that will be run on each node.
71:     unix\_matlab = '';
72:
73:     % Loop backwards over number of processes.
74:     for i\_rank=i\_rank\_stop:-1:i\_rank\_start
75:
76:         % Build commands
77:         [defscmds, unix\_matlab\_i\_rank] = ...
78:             MatMPI\_Commands(m\_file,i\_rank,MPI\_COMM\_WORLD);
79:         unix\_matlab = [unix\_matlab unix\_matlab\_i\_rank];
80:
81:     end
82:
83:     % Create a file name.
84:     %     unix\_matlab\_file = ['MatMPI/Unix\_Commands.' machine '.sh'];
85:     unix\_matlab\_file = ['MatMPI/Unix\_Commands.' machine '.
86:         num2str(i\_rank\_start) '.sh'];
87:
88:     % Append delete command.
89:     unix\_matlab = [unix\_matlab ' rm ' unix\_matlab\_file ';' nl];
90:
91:     % Put commands in a file.
92:     fid = fopen(unix\_matlab\_file,'wt');
93:     fwrite(fid,unix\_matlab);
94:     fclose(fid);
95:
96:     % Create unix commands to launch this file.
97:     if (strcmp(machine,host))
98:         unix\_launch\_i\_m = ['/bin/sh ./' unix\_matlab\_file ' &' nl];
99:     else
100:         unix\_launch\_i\_m = [remote\_launch machine remote\_flags ...
101:             q 'cd ' pwd '; /bin/sh ./' unix\_matlab\_file ' &' q ' &' nl];
102:     end
103:
104:     unix\_launch = [unix\_launch unix\_launch\_i\_m];
105:     end
106: end
107:
108: % Execute all launches in a single unix call.
109: unix\_launch
110: % unix(unix\_launch);

```

```
111:
112:
113: % Write commands unix commands to .sh text file
114: % to fix Matlab's problem with very long commands sent to unix().
115: unix\_launch\_file = 'MatMPI/Unix\_Commands.sh';
116: fid = fopen(unix\_launch\_file,'wt');
117: fwrite(fid,unix\_launch);
118: fclose(fid);
119: unix(['/bin/sh ' unix\_launch\_file]);
120: delete(unix\_launch\_file);
```

## 6.17 MPI\_Send()

```
1: function MPI\_Send( dest, tag, comm, varargin )
2: % MPI\_Send - Sends variables to dest.
3: %
4: % MPI\_Send( dest, tag, comm, var1, var2, ...)
5: %
6: %     Send message containing variables to dest with a given tag
7: %
8: %     dest can be an iteger from 0 to comm\_size-1
9: %     tag can be any integer
10: %     comm is an MPI Communicator (typically a copy of MPI\_COMM\_WORLD)
11: %
12:
13: % Get processor rank.
14: my\_rank = MPI\_Comm\_rank(comm);
15:
16: % Create buffer and lock file.
17: buffer\_file = MatMPI\_Buffer\_file(my\_rank,dest,tag,comm);
18: lock\_file = MatMPI\_Lock\_file(my\_rank,dest,tag,comm);
19:
20: % Save buf to file.
21: save(buffer\_file,'varargin');
22:
23: % Create lock file.
24: fclose(fopen(lock\_file,'w'));
```