pMatlab: Parallel Matlab Toolbox

Ed Hall

edhall@virginia.edu

Research Computing Support Center
Wilson Hall, Room 244
University of Virginia
Phone: 243-8800
Email: Res-Consult@Virginia.EDU

URL: www.itc.virginia.edu/researchers/services.html
Outline of Talk

- pMatlab Motivation
- Parallel Computing Background
- MatlabMPI
- pMatlab and Global Array Semantics
- Working with Distributed Matrices
- Example pMatlab Programs
- Developing, Testing, Debugging, Scaling
- pMatlab at UVA
- References
pMatlab Motivation

- Scientists and Engineers routinely push the limits of computing systems to tackle larger, more detailed problems.

- These computational problems either take too long or require too much memory on single (or dual) processor systems.

- One option for improving performance is parallel processing.

- Yet writing accurate, efficient, high performance parallel code is still highly non-trivial, requiring a substantial amount of time and energy.

- Primary concern of most scientists and engineers is conduct research, not to write code.
MATLAB is the dominant programming language for implementing numerical computations and is widely used for algorithm development, simulation, data reduction, testing and evaluation.

High productivity because one line of MATLAB code can typically replace multiple lines of C or Fortran code.

MATLAB supports a range of operating systems and processor architectures, providing portability and flexibility.

Excellent platform on which to create an accessible parallel computing framework.

The MIT Lincoln Laboratory has developed two libraries, **MatlabMPI** and **pMatlab**.

Allows MATLAB users to run multiple instances of MATLAB to address larger problems or to speed up their programs.

*Global array semantics* is the programming model used by pMatlab to simplify parallel programming.
Parallel Programming

What makes parallel programming so difficult?

Parallel programming introduces new complications not present in serial programming such as:

- Keeping track of which processor data is located on.
- Determining which processor data needs to be delivered to or received from.
- Distributing data such that communication between processors is minimized.
- Distributing computation such that computation on each processor is maximized.
- Synchronizing data between processors (if necessary).
- Debugging and deciphering communication between processors.
Parallel programming improves performance by breaking down a problem into smaller subproblems that are distributed to multiple processors. The benefits are two-fold.

- Total amount of computation performed by each individual processor is reduced, resulting in faster computation.
- The size of the problem can be increased by using more memory available on multiple processors.

There are several basic obstacles that restrict how much parallelization can improve performance.

Amdahl’s Law computes the theoretical maximum performance improvement for parallel programs. Let $f$ be the fraction of time spent on serial operations and $p$ is the number of processors.

$$speedup \leq \frac{1}{f + \frac{(1-f)}{p}}$$ (1)
Parallel Programming

Further obstacles that restrict how much parallelization can improve performance.

- Dividing and distributing a problem to multiple processors incurs an overhead absent in serial programs. Much of the overhead incurred by distributing a problem is caused by communication.

- It is not unusual to see performance increase, peak, then decrease as the number of processors grows, due to an increase in communication overhead.

- One way to mitigate the effects of communication on performance is to reduce the total number of individual messages by sending a few large messages rather than sending many small messages.

- In the case of pMatlab and MatlabMPI, overhead per message can be large; reduce the total amount of communication but also reduce the total number of messages,
Parallel Speedup

• Ratio of the time on 1 CPU divided by the time on N CPUs
  – If no communication is required, then speedup scales linearly with N
  – If communication is required, then the non-communicating part should scale linearly with N

• Speedup typically plotted vs number of processors
  – Linear (ideal)
  – Superlinear (achievable in some circumstances)
  – Sublinear (acceptable in most circumstances)
  – Saturated (usually due to communication)
**pMatlab Overview**

**pMatlab** and **MatlabMPI** follow *single-program multiple-data* (SPMD) model for constructing parallel programs.

In the SPMD model, the same program runs on each processor (“single-program”) but each program instance processes different data (“multiple-data”).

Flexibility and scalability of SPMD model has made it the dominant parallel programming model.

- SPMD codes can be tightly coupled (fine-grain parallelism) simulations of evolution equations predicting chemical processes, weather and combustion.
- Or “embarrassingly parallel” (coarse-grain) programs searching for genes, data strings, and patterns in data sequences or processing medical images or signals.
- The Message Passing Interface (MPI) is the most popular technology for writing parallel SPMD programs.
- In MPI, the program is started on all processors. The program is responsible for distributing data among processors, then each processor processes its section of the data, communicating among themselves as needed.

MPI is an industry standard. pMatlab and MatlabMPI use the MPI model,
**Message Passing Interface (MPI)**

MPI is a basic networking library tailored to parallel systems, enabling processes in a parallel program to communicate with each other. Details of the underlying network protocols and infrastructure are hidden from the programmer.

In message passing, all processes are independent entities, i.e. each process has its own private memory space. Processes communicate and share data by passing messages. Communication is performed explicitly by directly invoking send and receive operations in the code.

To distinguish sub-processes from each other, each process is given a unique identifier called a rank. MPI programs use ranks to identify which data should be processed by each process or what operations should be run on each process.

MatlabMPI is a MATLAB implementation of the MPI standard developed at the Lincoln Laboratory to emulate the “look and feel” of MPI. MatlabMPI is written in pure MATLAB code (i.e. no MEX files) and uses MATLAB’s file I/O functions to perform communication, allowing it to run on any system that MATLAB supports.

Despite MPI’s popularity, writing parallel programs with MPI and MatlabMPI can still be difficult. Explicit communication requires careful coordination of sending and receiving messages and data that must be divided among processors can require complex index computations.
MatlabMPI: Point-to-point Communication

- Any messaging system can be implemented using file I/O
- File I/O provided by Matlab via load and save functions
  - Takes care of complicated buffer packing/unpacking problem
  - Allows basic functions to be implemented in ~250 lines of Matlab code

```
MPI_Send (dest, tag, comm, variable);
variable = MPI_Recv (source, tag, comm);
```

- **Sender** saves variable in Data file, then creates Lock file
- **Receiver** detects Lock file, then loads Data file
Global Array Semantics

Message passing provides an extremely flexible programming paradigm that allows programmers to parallelize a wide range of data structures, including arrays.

*Global array semantics* is a parallel programming model in which the programmer views a distributed array as a single global array rather than multiple, independent arrays located on different processors, e.g., as with MPI.

The ability to view related data distributed across processors as a single array more closely matches the serial programming model, thus making the transition from serial to parallel programming much smoother.

A global array library can be implemented using message passing libraries such as MPI or MatlabMPI. The program calls functions from the global array library. The global array library determines if and how data must be redistributed and calls functions from the message passing library to perform the communication.

Communication is hidden from the programmer; arrays are automatically redistributed when necessary, without the knowledge of the programmer.
pMatlab Description

pMatlab brings global array semantics to MATLAB, using the message passing capabilities MatlabMPI.

Goal of pMatlab is to move beyond basic messaging (and its inherent programming complexity) towards higher level parallel data structures and functions.

Parallel functionality can be added to existing serial programs with minor modifications.

Distributed matrices/vectors are created by using “maps” that describe data distribution.

“Automatic” parallel computation and data distribution is achieved via operator overloading.

“Pure” Matlab implementation.
MatlabMPI & pMatlab Software Layers

Application

Parallel Library

Parallel Hardware

- Can build a parallel library with a few messaging primitives
  - MatlabMPI provides this messaging capability:
    ```
    MPI_Send(dest, comm, tag, X);
    MPI_Recv(source, comm, tag, X);
    ```

- Can build a application with a few parallel structures and functions
  - pMatlab provides parallel arrays and functions
    ```
    X = ones(n, mapX);
    Y = zeros(n, mapY);
    Y(:, :) = fft(X);
    ```
MatlabMPI Example

MatlabMPI code for FFT and cornerturn

```matlab
if (my_rank==0) | (my_rank==1) | (my_rank==2) | (my_rank==3)
    A_local=rand(M,N/4);
endif (my_rank==4) | (my_rank==5) | (my_rank==6) | (my_rank==7)
    B_local=zeros(M/4,N);
end
tag = 0;
if (my_rank==0) | (my_rank==1) | (my_rank==2) | (my_rank==3)
    A_local=fft(A_local);
    for ii = 0:3
        MPI_Send(ii+4, tag, comm, A_local(ii*M/4 + 1:(ii+1)*M/4,:));
    end
end
if (my_rank==4) | (my_rank==5) | (my_rank==6) | (my_rank==7)
    for ii = 0:3
        B_local(:, ii*N/4 + 1:(ii+1)*N/4) = MPI_Recv(ii, tag, comm);
    end
end
```

pMatlab: Parallel Matlab Toolbox – p. 12/30
pMatlab code for FFT and cornerturn

```matlab
mapA = map([1 4],{},[0:3]);
mapB = map([4 1],{},[4:7]);
A = rand(M,N,mapA);
B = zeros(M,N,mapB)
B(:,:, :) = fft(A);
```
Distributed Matrices

**pMatlab** introduces a new datatype: the *distributed matrix*, or *dmat*

- **dmat** is the fundamental data storage datatype in pMatlab, equivalent to double in MATLAB.
- pMatlab can support dmat objects with two, three, or four dimensions.
- dmat objects must be explicitly constructed in pMatlab via a *constructor function*.
- Constructor functions accept the same set of parameters as their corresponding MATLAB functions, with the addition of a map parameter.
- The map parameter accepts a map object which describes how to distribute the dmat object across multiple processors.

```
D = ones(M, P);        % Create a MxM dmat object of ones with map P
D = zeros(M, N, R, P); % Create a MxNxB dmat object of zeros using
                       % map P
D = rand(M, N, R, S, P); % Create a MxNxBxS dmat object of random
                        % numbers with map P
D = spalloc(M, N, X, P); % Create a MxN sparse dmat with room to hold
                         % X non-zero values with map P
```
Maps

A map object in pMatlab defines how and where the dmat is to be distributed.

Each map object has four components:

- The grid is a vector of integers that specifies how each dimension of a dmat is broken up. For example, if the grid is [2 3], the first dimension is broken up between 2 processors and the second dimension is broken up between 3 processors,

- The distribution specifies how to distribute each dimension of the dmat among processors. There are three types of distributions:
  - Block – Each processor contains a single contiguous block of data
  - Cyclic – Data are interleaved among processors
  - Block-cyclic – Contiguous blocks of data are interleaved among processors.

- The processor list specifies on which ranks the object should be distributed.

- Overlap is a vector of integers that specify the amount of data overlap between processors for each dimension. Only block distributions can have overlap. Overlap is useful in situations when data on the boundary between two processors are required by both, e.g. convolution.
Example Data Distributions

Figure 13 – Example of a map grid.

Figure 14 – Examples of different data distributions.
pMatlab Maps and Distributed Matrices

• Map Example

mapA = map([1 2], {}, [0:1]);  % Specifies that cols be dist. over 2 procs
mapB = map([1 2], {}, [2:3]);

A = rand(m,n, mapA);  % Create random distributed matrix
B = zeros(m,n, mapB);  % Create empty distributed matrix
B(:,:) = A;  % Copy and redistribute data from A to B.

• Grid and Resulting Distribution
Using MATLAB’s function overloading feature, pMatlab is able to provide new implementations of existing MATLAB functions.

These overloaded functions provide the same functionality as their MATLAB equivalents but operate on the dmats rather than doubles.

pMatlab overloads a number of basic MATLAB functions. Most of these overloaded functions implement only a subset of the available functionality of each function.

A number of arithmetic operators, such as addition (+), matrix multiplication (*), element-wise multiplication (.*), and equality (==) have been overloaded.

```
N = 1000000;
M1 = map([1 Ncpus], {}, 0:Ncpus-1);
% Add two dmats
A = rand(N, M1); % NxN dmat mapped to M1
B = rand(N, M1); % NxN dmat mapped to M1
C1 = A + B; % Result is mapped to M1
```
Working with Distributed Matrices

Several mathematical functions, including simple functions such as absolute value (abs) and advanced functions such as fft have been overloaded in pMatlab.

\[
N = 1000000;
M1 = \text{map}([1 \ \text{Ncpus}], \{\}, \ 0: \text{Ncpus-1});
M2 = \text{map}([\text{Ncpus} \ 1], \{\}, \ 0: \text{Ncpus-1});
\]

% Absolute value
A = \text{rand}(N, \ M1); \quad % \text{NxN} \ \text{dmat} \text{ mapped to} \ M1
B1 = \text{abs}(A); \quad % \text{Result is mapped to} \ M1
B2(:,:,\ :) = \text{abs}(A); \quad % \text{Result is remapped to} \ M2

% FFT
D = \text{rand}(N, \ M1); \quad % \text{NxN} \ \text{dmat} \text{ mapped to} \ M1
E1 = \text{fft}(D); \quad % \text{Result is mapped to} \ M1
The notion of matrices distributed across multiple processors raises the issue of global and local scopes of a dmat. The **global scope** of a dmat refers to the matrix as a single entity, while **local scope** refers to only the section of the dmat residing on the local processor.

Occasionally, pMatlab users may wish to directly manipulate a dmat object. Thus, pMatlab provides additional functions that allow the user to access the contents of dmat object without using overloaded pMatlab functions.

The **local** and **put_local** functions give the user direct access to the local contents of dmat objects. Each processor in a dmat’s map contains a section of the dmat. **local** returns a matrix that contains the section of the dmat that resides on the local processor. Conversely, **put_local** writes the contents of a matrix into that processor’s section of the dmat.

The **local** and **put_local** functions allow the user to perform operations not implemented in pMatlab while still taking advantage of pMatlab’s ability to distribute data and computation.
Launching pMatlab

The recommended method of launching pMatlab programs is to use a Matlab launch script, e.g. `RUN.m`. It must be located and run in the same directory as the pMatlab application.

```matlab
% RUN.m is a generic script for running pMatlab scripts.
% Define number of processors to use
    Ncpus = 4;
% Name of the script you want to run
    mFile = 'sample_application';
% Define cpus.
% Empty implies run on host.
    cpus = {};
% Get path to PBS node file on ITC Linux cluster
    pbs_path=getenv('PBS_NODEFILE');
% Specify machine names to run remotely.
    cpus = textread(pbs_path,'%s');
% Specify which machines to run on
    cpus = {'node1.ll.mit.edu', ...
            'node2.ll.mit.edu', ...
            'node3.ll.mit.edu', ...
            'node4.ll.mit.edu'};
```
Launching pMatlab

The RUN.m script continued

% Abort left over jobs
MPI_Abort;
pause(2.0);
% Delete left over MPI directory
MatMPI_Delete_all;
pause(2.0);
% Define global variables
global pMATLAB;
% Run the script.
[‘Running: ’ mFile ‘ on ’ num2str(Ncpus) ’ cpus’]
eval(MPI_Run(mFile, Ncpus, cpus));

On the ITC Linux clusters, RUN.m would be run like any other Matlab script from within the PBS command file.

A MatMPI subdirectory is automatically created for pMatlab job files. Only one pMatlab job can be run from the directory at a time.
A Sample pMatlab Application

pMatlab sample_application.m that performs a parallel FFT

N = 2^10; % NxN Matrix size.
% Turn parallelism on or off.
PARALLEL = 1; % Can be 1 or 0. OK to change.

% Create Maps.
mapX = 1; mapY = 1;
if (PARALLEL)
    % Initialize pMatlab.
    pMatlab_Init;
    Ncpus = pMATLAB.comm_size;
    my_rank = pMATLAB.my_rank;
    % Break up channels.
    mapX = map([1 Ncpus], {}, 0:Ncpus-1);
    mapY = map([1 Ncpus], {}, 0:Ncpus-1);
end
A Sample pMatlab Application

```
pMatlab sample_application.m continued

    % Allocate data structures.
    X = rand(N,N,mapX);
    Y = zeros(N,N,mapY);

    % Do fft. Changes Y from real to complex.
    Y(:,:, :) = fft(X);
    % Finalize the pMATLAB program
    disp('SUCCESS');

    if (PARALLEL)
        pMatlab_Finalize;
    end
```
Developing pMatlab Code

Several guidelines that pMatlab users should follow when developing pMatlab applications.

- Write code that can be run independent of pMatlab. pMatlab code should be written in such a manner that the pMatlab library can be easily disabled to run the application serially, i.e. on a single processor.

- In parallel computing there are generally two types of errors, those resulting from algorithmic choice and those resulting from parallel implementation.

- After the code has been debugged in serial, any new bugs in the parallel code can be assumed to arise from the parallel implementation.

- Write scalable code. Avoid writing code that depends on a specific number of processors. Generally, the more the code is parameterized the less rewriting, rebuilding, retesting is required.

- Avoid rank-dependent code. Some exceptions to this rule:
  - One is related to the agg function which returns the entire contents of a dmat to the leader processor.
  - Writing rank dependent code is associated with I/O for debugging.
Testing and Debugging pMatlab Code

Testing

- A failure in just one MATLAB process can cause the entire program to hang.
- All non-leader MATLAB processes redirect their output and error messages to .out files in the MatMPI directory.
- Users should make a habit of routinely checking the contents of the MatMPI/*.out files when they suspect their programs have hung.

Debugging

- The MATLAB debugger can be used to assist in debugging pMatlab programs in a limited fashion. The debugger runs on only the leader process on the user’s machine.
- Nonleader processes will simply continue past breakpoints set by the user until they reach a parallelized operation which requires communication with the leader. However, this asynchronous behavior will not affect the behavior of the program or the debugger.
Another tenet of good software engineering is that programs should not be run on full scale inputs immediately. Rather, programs should initially be run on a small test problem to verify functionality and to validate against known results before running at full scale.

The following is the recommended procedure for scaling up a pMatlab application. This procedure gradually adds complexity to running the application.

1. Run with 1 processor on the user’s local machine with the pMatlab library disabled. This tests the basic serial functionality of the code.

2. Run with 1 processor on the local machine with pMatlab enabled. Tests that the pMatlab library has not broken the basic functionality of the code.

3. Run with 2 processors on the local machine. Tests the program’s functionality works with more than one processor without network communication.

4. Run with 2 processors on multiple machines. Test that the program works with network communication.

5. Run with 4 processors on multiple machines.

6. Increase the number of processors as appropriate.
### Scaling pMatlab Code

<table>
<thead>
<tr>
<th>In pMatlab code</th>
<th>In RUN.m</th>
</tr>
</thead>
<tbody>
<tr>
<td>PARALLEL = 0;</td>
<td>Ncpus = 1; cpus =</td>
</tr>
<tr>
<td></td>
<td>{ };</td>
</tr>
<tr>
<td>PARALLEL = 1;</td>
<td>Ncpus = 1; cpus =</td>
</tr>
<tr>
<td></td>
<td>{ };</td>
</tr>
<tr>
<td>PARALLEL = 1;</td>
<td>Ncpus = 2; cpus =</td>
</tr>
<tr>
<td></td>
<td>{ };</td>
</tr>
<tr>
<td>PARALLEL = 1;</td>
<td>Ncpus = 2; cpus =</td>
</tr>
<tr>
<td></td>
<td>{ 'node1', 'node2' };</td>
</tr>
<tr>
<td>PARALLEL = 1;</td>
<td>Ncpus = 4; cpus =</td>
</tr>
<tr>
<td></td>
<td>{ 'node1', 'node2' };</td>
</tr>
<tr>
<td>PARALLEL = 1;</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>cpus = { 'node1', 'node2' };</td>
</tr>
</tbody>
</table>

Figure 26 – Example sequence of parameters for scaling parallel programs.
Figure B.1. Measured and predicted LU performance. The LU performance model is shown for several different configurations of Gigabit Ethernet (GigE) and InfiniBand (IB) networks. The lowest performing network agrees well with the pMatlab data. As the network is improved the performance should approach that of the C-MPI code, which uses the more efficient (and more complex) 2D block cyclic distribution.
Using pMaltab at UVA

pMaltab installed in /common/matlab/pMatlab on ITC Linux Clusters.

To use it, create a `matlab` subdirectory of your home directory containing a `startup.m` file with the following lines:

```matlab
addpath /common/matlab/pMatlab/MatlabMPI/src
addpath /common/matlab/pMatlab/src
```

To test, type “help pMaltab” at the Matlab prompt.

See examples in the directory /common/matlab/pMatlab/examples of the ITC Linux clusters.

Use pMaltab on ITC clusters for large memory jobs, requiring memory distributed across several nodes.

For “embarrassingly parallel” jobs use the Matlab compiler. See the URL http://www.itc.virginia.edu/research/matlab/compiler.html
MatlabMPI and pMatlab References

Parallel Programming with MatlabMPI:
http://www.ll.mit.edu/MatlabMPI

pMatlab: Parallel Matlab Toolbox:
(used extensively in this talk)
http://www.ll.mit.edu/pmatlab

Sourcebook of Parallel Computing
Edited by Dongara, Foster, Fox, Gropp, Kennedy, Torczon and White
(Available to peruse in Wilson 244)