Parallel Matlab: The Next Generation

Dr. Jeremy Kepner / MIT Lincoln Laboratory
Ms. Nadya Travinin / MIT Lincoln Laboratory

This work is sponsored by the Department of Defense under Air Force Contract F19628-00-C-0002. Opinions, interpretations, conclusions, and recommendations are those of the author and are not necessarily endorsed by the United States Government.
Outline

• Introduction
  • Motivation
  • Challenges

• Approach

• Performance Results

• Future Work and Summary
Motivation: DoD Need

• Cost

  = 4 lines of DoD code

• DoD has a clear need to rapidly develop, test and deploy new techniques for analyzing sensor data
  – Most DoD algorithm development and simulations are done in Matlab
  – Sensor analysis systems are implemented in other languages
  – Transformation involves years of software development, testing and system integration

• MatlabMPI allows any Matlab program to become a high performance parallel program
Challenges: Why Has This Been Hard?

• **Productivity**
  – Most users will not touch any solution that requires other languages (even cmex)

• **Portability**
  – Most users will not use a solution that could potentially make their code non-portable in the future

• **Performance**
  – Most users want to do very simple parallelism
  – Most programs have long latencies (do not require low latency solutions)
Outline

• Introduction

• Approach
  • MatlabMPI messaging
  • pMatlab programming

• Performance Results

• Future Work and Summary
• Can build a parallel library with a few messaging primitives
  • MatlabMPI provides this messaging capability:
    
    ```
    MPI_Send(dest, comm, tag, X);
    X = MPI_Recv(source, comm, tag);
    ```

• Can build an 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 functionality

• “Core Lite” Parallel computing requires eight capabilities
  – `MPI_Run` launches a Matlab script on multiple processors
  – `MPI_Comm_size` returns the number of processors
  – `MPI_Comm_rank` returns the id of each processor
  – `MPI_Send` sends Matlab variable(s) to another processor
  – `MPI_Recv` receives Matlab variable(s) from another processor
  – `MPI_Init` called at beginning of program
  – `MPI_Finalize` called at end of program

• Additional convenience functions
  – `MPI_Abort` kills all jobs
  – `MPI_Bcast` broadcasts a message
  – `MPI_Probe` returns a list of all incoming messages
  – `MPI_cc` passes program through Matlab compiler
  – `MatMPI_Delete_all` cleans up all files after a run
  – `MatMPI_Save_messages` toggles deletion of messages
  – `MatMPI_Comm_settings` user can set MatlabMPI internals
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

- 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
Example: Basic Send and Receive

- Initialize
  - Get processor ranks

```matlab
MPI_Init; % Initialize MPI.
comm = MPI_COMM_WORLD; % Create communicator.
comm_size = MPI_Comm_size(comm); % Get size.
my_rank = MPI_Comm_rank(comm); % Get rank.
source = 0; % Set source.
dest = 1; % Set destination.
tag = 1; % Set message tag.

if(comm_size == 2) % Check size.
  if (my_rank == source) % If source.
    data = 1:10; % Create data.
    MPI_Send(dest,tag,comm,data); % Send data.
  end
  if (my_rank == dest) % If destination.
    data=MPI_Recv(source,tag,comm); % Receive data.
  end
end

MPI_Finalize; % Finalize Matlab MPI.
exit; % Exit Matlab
```

- Uses standard message passing techniques
- Will run anywhere Matlab runs
- Only requires a common file system
pMatlab Goals

- **Allow a Matlab user to write parallel programs with the least possible modification to their existing Matlab programs**
  
- New parallel concepts should be intuitive to Matlab users
  - parallel matrices and functions instead of message passing
  - Matlab*P interface

- **Support the types of parallelism we see in our applications**
  - data parallelism (distributed matrices)
  - task parallelism (distributed functions)
  - pipeline parallelism (conduits)

- **Provide a single API that potentially a wide number of organizations could implement (e.g. Mathworks or others)**
  - unified syntax on all platforms

- **Provide a unified API that can be implemented in multiple ways,**
  - Matlab*P implementation
  - Multimatlab
  - matlab-all-the-way-down implementation
  - unified hybrid implementation (desired)
Structure of pMatlab Programs

**Initialize globals**

- `pMATLAB_Init;`

**Map to sets of processors**
- `mapX = map([1:N/2],{},[1:N/2]);`
- `mapY = map([N/2 1],{},[N/2+1:N]);`
- `X = ones(n, mapX);`
- `Y = zeros(n, mapY);`
- `Y(:, :) = fft(X);`

**Clear globals**

- `pMATLAB_Finalize;`

**Distributed matrices**

**Parallel FFT and “Corner Turn” Redistribution**

- Can parallelize code by changing a few lines
- Built on top of MatlabMPI (pure Matlab)
- Moving towards Matlab*P interface
pMatlab Library Functionality

• “Core Lite” Provides distributed array storage class (up to 4D)
  – Supports reference and assignment on a variety of distributions:
    Block, Cyclic, Block-Cyclic, Block-Overlap
  Status: Available

• “Core” Overloads most array math functions
  – good parallel implementations for certain mappings
  Status: In Development

• “Core Plus” Overloads entire Matlab library
  – Supports distributed cell arrays
  – Provides best performance for every mapping
  Status: Research
Outline

• Introduction

• Approach

• Performance Results
  • MatlabMPI
  • pMatlab

• Future Work and Summary
MatlabMPI vs MPI bandwidth

- Bandwidth matches native C MPI at large message size
- Primary difference is latency (35 milliseconds vs. 30 microseconds)
MatlabMPI bandwidth scalability

Linux w/Gigabit Ethernet

- Bandwidth scales to multiple processors
- Cross mounting eliminates bottlenecks
MatlabMPI on WindowsXP

MATLAB 6.5

Command Window

>> RUN
No pid files found
Nothing to delete.
Launching MPI rank: 3 on: SLAVE
Launching MPI rank: 2 on: SLAVE
Launching MPI rank: 1 on: SLAVE
Launching MPI rank: 0 on: SLAVE

unix_launch =
start /b MatlabMPI\Dos_Commands.SLAVE.0.bat

Z:\projects\MPI-Jumpstart-Kit\MatlabMPI\pc>start /b MatlabMPI\my_rank: 0
SUCCESS

>>
• Achieved “classic” super-linear speedup on fixed problem
• Achieved speedup of ~300 on 304 processors on scaled problem
“Cognitive” Algorithms

- **Challenge:** applications requiring vast data; real-time; large memory
- **Approach:** test parallel processing feasibility using MatlabMPI software
- **Results:** algorithms rich in parallelism; significant acceleration achieved with minimal (100x less) programmer effort

<table>
<thead>
<tr>
<th>Application</th>
<th>Algorithm</th>
<th>CPUs / Speedup / Effort</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contextual vision</td>
<td>Statistical object detection</td>
<td>16 / 9.4x / 3 hrs</td>
</tr>
<tr>
<td>Text processing</td>
<td>Expectation maximization</td>
<td>14 / 9.7x / 8 hrs</td>
</tr>
<tr>
<td>Image segment.</td>
<td>Belief propagation</td>
<td>12 / 8x -( \sim ) x / 4 hrs</td>
</tr>
</tbody>
</table>

**Contextual vision**
- Image Face Map
  - Torralba (Al Lab) / Kepner (Lincoln)
  - Coarse Grained
  - Image Parallel
  - (Static Client Server)

**Text Processing**
- Words
  - Sentences
  - Medium Grained
  - Sentence Parallel
    - (Block Cyclic Dynamic Client Server)

**Image Segmentation**
- Observed
  - Recovered
  - Murphy (Al Lab) / Kepner (Lincoln)
  - Fine Grained
  - Pixel Parallel
    - (Block Nearest Neighbor Overlap)
Current MatlabMPI deployment

- Lincoln Signal processing (7.8 on 8 cpus, 9.4 on 8 duals)
- Lincoln Radar simulation (7.5 on 8 cpus, 11.5 on 8 duals)
- Lincoln Hyperspectral Imaging (~3 on 3 cpus)
- MIT LCS Beowulf (11 Gflops on 9 duals)
- MIT AI Lab Machine Vision
- OSU EM Simulations
- ARL SAR Image Enhancement
- Wash U Hearing Aid Simulations
- So. Ill. Benchmarking
- JHU Digital Beamforming
- ISL Radar simulation
- URI Heart modeling

- Rapidly growing MatlabMPI user base
- Web release creating hundreds of users

http://www.ll.mit.edu/MatlabMPI
Outline

• Introduction
• Approach
• **Performance Results**
  • MatlabMPI
  • pMatlab
• Future Work and Summary
pMatlab vs. MatlabMPI bandwidth

- Bandwidth matches underlying MatlabMPI
- Primary difference is latency (35 milliseconds vs. 70 milliseconds)
Clutter Simulation Performance

### Parallel Matlab

- Achieved “classic” super-linear speedup on fixed problem
- Serial and Parallel code “identical”

#### Fixed Problem Size (Linux Cluster)

<table>
<thead>
<tr>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Number of Processors</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 4 8 16</td>
</tr>
</tbody>
</table>

---

```matlab
% Initialize
pMATLAB_Init; Ncpus=comm_vars.comm_size;

% Map X to first half and Y to second half.
mapX=map([1 Ncpus/2],{},[1:Ncpus/2])
mapY=map([Ncpus/2 1],{},[Ncpus/2+1:Ncpus]);

% Create arrays.
X = complex(rand(N,M,mapX),rand(N,M,mapX));
Y = complex(zeros(N,M,mapY));

% Initialize coefficients
coefs = ...
weights = ...

% Parallel filter + corner turn.
Y(:,:,1) = conv2(coefs,X);
% Parallel matrix multiply.
Y(:,:,1) = weights*Y;

% Finalize pMATLAB and exit.
pMATLAB_Finalize; exit;
```
Eight Stage Simulator Pipeline

Parallel Data Generator
- Initialize
- Inject targets
- Conolve with pulse
- Channel response

Parallel Signal Processor
- Pulse compress
- Beamform
- Detect targets

Example Processor Distribution
- 0, 1
- 2, 3
- 4, 5
- 6, 7
- all

Matlab Map Code
- map3 = map([2 1], {}, 0:1);
- map2 = map([1 2], {}, 2:3);
- map1 = map([2 1], {}, 4:5);
- map0 = map([1 2], {}, 6:7);

Goal: create simulated data and use to test signal processing
parallelize all stages; requires 3 “corner turns”
pMatlab allows serial and parallel code to be nearly identical
Easy to change parallel mapping; set map=1 to get serial code
pMatlab Code

```matlab
pMATLAB_Init; SetParameters; SetMaps; %Initialize.
Xrand = 0.01*squeeze(complex(rand(Ns,Nb, map0),rand(Ns,Nb, map0)));
X0 = squeeze(complex(zeros(Ns,Nb, map0)));
X1 = squeeze(complex(zeros(Ns,Nb, map1)));
X2 = squeeze(complex(zeros(Ns,Nc, map2)));
X3 = squeeze(complex(zeros(Ns,Nc, map3)));
X4 = squeeze(complex(zeros(Ns,Nb, map3)));
...
for i_time=1:NUM_TIME % Loop over time steps.
    X0(:,:) = Xrand; % Initialize data
    for i_target=1:NUM_TARGETS
        [i_s i_c] = targets(i_time,i_target,:);
        X0(i_s,i_c) = 1; % Insert targets.
    end
    X1(:,:) = conv2(X0,pulse_shape,'same'); % Convolve and corner turn.
    X2(:,:) = X1*steering_vectors; % Channelize and corner turn.
    X3(:,:) = conv2(X2,kernel,'same'); % Pulse compress and corner turn.
    X4(:,:) = X3*steering_vectors'; % Beamform.
    [i_range,i_beam] = find(abs(X4) > DET); % Detect targets
end
pMATLAB_Finalize; % Finalize.
```

- **Implicitly Parallel Code**
- **Required Change**
Outline

• Introduction
• Approach
• Performance Results

• Future Work and Summary
pMatlab achieves high performance with very little effort
Airborne Sensor “QuickLook” Capability

- RAID Disk Recorder
- Data Files
- 28 CPU Bladed Cluster Running pMatlab
- SAR GMTI ...
  (new)
- Analyst Workstation Running Matlab
- Streaming Sensor Data

28 CPU Bladed Cluster Running pMatlab

Beam Reconstruct Performance

- Linear
- pMatlab
- w/Hyperthreading

MIT Lincoln Laboratory
pMatlab Future Work

1. Demonstrate in a large multi-stage framework

2. Incorporate Expert Knowledge into Standard Components

3. Port pMatlab to HPEC systems
Summary

- MatlabMPI has the basic functions necessary for parallel programming
  - Size, rank, send, receive, launch
  - Enables complex applications or libraries

- Performance can match native MPI at large message sizes

- Demonstrated scaling into hundreds of processors

- pMatlab allows user’s to write very complex parallel codes
  - Built on top of MatlabMPI
  - Pure Matlab (runs everywhere Matlab runs)
  - Performance comparable to MatlabMPI

- Working with MIT LCS, Ohio St. and UCSB to define a unified parallel Matlab interface
Acknowledgements

• Support
  – Charlie Holland DUSD(S&T) and John Grosh OSD
  – Bob Bond and Ken Senne (Lincoln)

• Collaborators
  – Nadya Travinin (Lincoln)
  – Stan Ahalt and John Nehrbass (Ohio St.)
  – Alan Edelman and Ron Choy (MIT LCS)
  – John Gilbert (UCSB)
  – Antonio Torralba and Kevin Murphy (MIT AI Lab)

• Centers
  – Maui High Performance Computing Center
  – Boston University
  – MIT Earth and Atmospheric Sciences
Web Links

MatlabMPI
http://www.ll.mit.edu/MatlabMPI

High Performance Embedded Computing Workshop
http://www.ll.mit.edu/HPEC
Beam Reconstruct Performance

- Linear
- pMatlab
- w/Hyperthreading

Speedup vs. Number of Processors