Parallel Computing with MATLAB

Narfi Stefansson
Parallel Computing Development Manager
MathWorks
Agenda

- Products and terminology
- GPU capabilities
- Multi-process capabilities
- How are customers using this?
Parallel Computing with MATLAB on CPU

User’s Desktop

Parallel Computing Toolbox

MATLAB Workers

Compute Cluster

MATLAB Distributed Computing Server
Evolving With Technology Changes

GPU

Single processor

Multicore

Multiprocessor

Cluster

Grid, Cloud
Why GPUs and why now?

- Double support
  - Single/double performance inline with expectations
- Operations are IEEE Compliant
- Cross-platform support now available
What came in R2010b?

- Parallel Computing Toolbox
  - GPU support
  - Broader distributed array algorithm support (QR, rectangular \)

- MATLAB Distributed Computing Server
  - GPU support
  - Run as user with MathWorks job manager
  - Non-shared file system support

- Simulink®
  - Real-Time Workshop® support with PCT and MDCS
What came in R2011a?

- **Parallel Computing Toolbox**
  - Deployment of local workers
  - More GPU support
  - More distributed array algorithm support

- **MATLAB Distributed Computing Server**
  - Enhanced support for Microsoft HPC Server
  - More GPU support
  - Remote service start in Admin Center
GPU Support

- Call GPU(s) from MATLAB or toolbox/server worker
- Support for CUDA 1.3 enabled devices and up
Programming Parallel Applications

Level of control

- Minimal

Required effort

- None

- Straightforward

- Involved
Summary of Options for Targeting GPUs

**Level of control**

- Minimal
- Some
- Extensive

**Parallel Options**

- Use GPU arrays with MATLAB built-in functions
- Execute custom functions on elements of the GPU array
- Create kernels from existing CUDA code and PTX files
GPU Array Functionality

- Array data stored in GPU device memory
- Algorithm support for over 100 functions
- Integer, single, double, real and complex support
Example:

**GPU Arrays**

```
>> A = someArray(1000, 1000);
>> G = gpuArray(A); % Push to GPU memory
...
>> F = fft(G);
>> x = G\b;
...
>> z = gather(x); % Bring back into MATLAB
```
GPUArray Function Support

- >100 functions supported
  - fft, fft2, ifft, ifft2
  - Matrix multiplication (A*B)
  - Matrix left division (A\b)
  - LU factorization
  - '
  - abs, acos, ..., minus, ..., plus, ..., sin, ...
  - conv, conv2, filter
  - indexing
GPU Array benchmarks

<table>
<thead>
<tr>
<th>A\b*</th>
<th>Tesla C1060</th>
<th>Tesla C2050 (Fermi)</th>
<th>Quad-core Intel CPU</th>
<th>Ratio (Fermi:CPU)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single</td>
<td>191</td>
<td>250</td>
<td>48</td>
<td>5:1</td>
</tr>
<tr>
<td>Double</td>
<td>63.1</td>
<td>128</td>
<td>25</td>
<td>5:1</td>
</tr>
<tr>
<td>Ratio</td>
<td>3:1</td>
<td>2:1</td>
<td>2:1</td>
<td></td>
</tr>
</tbody>
</table>

* Results in Gflops, matrix size 8192x8192. Limited by card memory. Computational capabilities not saturated.
# GPU Array benchmarks

<table>
<thead>
<tr>
<th>MTIMES</th>
<th>Tesla C1060</th>
<th>Tesla C2050 (Fermi)</th>
<th>Quad-core Intel CPU</th>
<th>Ratio (Fermi:CPU)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single</td>
<td>365</td>
<td>409</td>
<td>59</td>
<td>7:1</td>
</tr>
<tr>
<td>Double</td>
<td>75</td>
<td>175</td>
<td>29</td>
<td>6:1</td>
</tr>
<tr>
<td>Ratio</td>
<td>4.8:1</td>
<td>2.3:1</td>
<td>2:1</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FFT</th>
<th>Tesla C1060</th>
<th>Tesla C2050 (Fermi)</th>
<th>Quad-core Intel CPU</th>
<th>Ratio (Fermi:CPU)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single</td>
<td>50</td>
<td>99</td>
<td>2.29</td>
<td>43:1</td>
</tr>
<tr>
<td>Double</td>
<td>22.5</td>
<td>44</td>
<td>1.47</td>
<td>30:1</td>
</tr>
<tr>
<td>Ratio</td>
<td>2.2:1</td>
<td>2.2:1</td>
<td>1.5:1</td>
<td></td>
</tr>
</tbody>
</table>
Example:

arrayfun: Element-Wise Operations

>> y = arrayfun(@foo, x); % Execute on GPU

function y = foo(x)
y = 1 + x.*(1 + x.*(1 + x.*(1 + ...
  x.*(1 + x.*(1 + x.*(1 + x.*(1 + ...
    x.*(1 + x./9)./8)./7)./6)./5)./4)./3)/2);
Some arrayfun benchmarks

Note: Due to memory constraints, a different approach is used at N=15 and above.

CPU[4] = multithreading enabled
CPU[1] = multithreading disabled
Example:
Invoking CUDA Kernels

% Setup
kern = parallel.gpu.CUDAKernel('myKern.ptx', cFcnSig)

% Configure
kern.ThreadBlockSize=[512 1];
kern.GridSize=[1024 1024];

% Run
[c, d] = feval(kern, a, b);
Example:

Corner Detection on the CPU

dx = cdata(2:end-1,3:end) - cdata(2:end-1,1:end-2);
dy = cdata(3:end,2:end-1) - cdata(1:end-2,2:end-1);
dx2 = dx.*dx;
dy2 = dy.*dy;
dxy = dx.*dy;

gaussHalfWidth = max( 1, ceil( 2*gaussSigma ) );
ssq = gaussSigma^2;
t = -gaussHalfWidth : gaussHalfWidth;
gaussianKernel1D = exp(-t.*t)/(2*ssq)/(2*pi*ssq);  % The Gaussian 1D filter
gaussianKernel1D = gaussianKernel1D / sum(gaussianKernel1D);
smooth_dx2 = conv2( gaussianKernel1D, gaussianKernel1D, dx2, 'valid' );
smooth_dy2 = conv2( gaussianKernel1D, gaussianKernel1D, dy2, 'valid' );
smooth_dxy = conv2( gaussianKernel1D, gaussianKernel1D, dxy, 'valid' );

det = smooth_dx2 .* smooth_dy2 - smooth_dxy .* smooth_dxy;
trace = smooth_dx2 + smooth_dy2;
score = det - 0.25*edgePhobia*(trace.*trace);

1. Calculate derivatives

2. Smooth using convolution

3. Calculate score
Example:

Corner Detection on the GPU

cdata = gpuArray( cdata );

dx = cdata(2:end-1,3:end) - cdata(2:end-1,1:end-2);
dy = cdata(3:end,2:end-1) - cdata(1:end-2,2:end-1);
dx2 = dx.*dx;
dy2 = dy.*dy;
dxy = dx.*dy;

gaussHalfWidth = max( 1, ceil( 2*gaussSigma ) );
ssq = gaussSigma^2;
t = -gaussHalfWidth : gaussHalfWidth;
gaussianKernel1D = exp(-(t.*t)/(2*ssq))/(2*pi*ssq); % The Gaussian 1D filter
gaussianKernel1D = gaussianKernel1D / sum(gaussianKernel1D);
smooth_dx2 = conv2( gaussianKernel1D, gaussianKernel1D, dx2, 'valid' );
smooth_dy2 = conv2( gaussianKernel1D, gaussianKernel1D, dy2, 'valid' );
smooth_dxy = conv2( gaussianKernel1D, gaussianKernel1D, dxy, 'valid' );

det = smooth_dx2 .* smooth_dy2 - smooth_dxy .* smooth_dxy;
trace = smooth_dx2 + smooth_dy2;
score = det - 0.25*edgePhobia*(trace.*trace);

score = gather( score );

0. Move data to GPU
4. Bring data back
arrayfun

Can execute entire scalar programs on the GPU
(while, if, for, break, &, &&, ...)

function [logCount,t] = mandelbrotElem( x0, y0, r2, maxIter)
    % Evaluate the Mandelbrot function for a single element
    z0 = complex( x0, y0 );
    z = z0;
    count = 0;
    while count <= maxIter && (z*conj(z) <= r2)
        z = z*z + z0;
        count = count + 1;
    end
    % . . . Etc. . . .
Summary of Options for Targeting GPUs

Level of control

- Minimal
- Some
- Extensive

Parallel Options

- Use GPU arrays with MATLAB built-in functions
- Execute custom functions on elements of the GPU array
- Create kernels from existing CUDA code and PTX files
Parallel Computing enables you to …

Larger Compute Pool

Speed up Computations

Larger Memory Pool

Work with Large Data
Programming Parallel Applications

**Level of control**
- Minimal
- Some
- Extensive

**Parallel Options**
- Support built into Toolboxes
- High-Level Programming Constructs: (e.g. parfor, batch, distributed)
- Low-Level Programming Constructs: (e.g. Jobs/Tasks, MPI-based)
Parallel Computing with MATLAB on CPU
Parallel Support in Optimization Toolbox

- **Functions:**
  - `fmincon`
    - Finds a constrained minimum of a function of several variables
  - `fminimax`
    - Finds a minimax solution of a function of several variables
  - `fgoalattain`
    - Solves the multiobjective goal attainment optimization problem

- Functions can take finite differences in parallel in order to speed the estimation of gradients
Tools with Built-in Support

- Optimization Toolbox
- Global Optimization Toolbox
- Statistics Toolbox
- SystemTest
- Simulink Design Optimization
- Bioinformatics Toolbox
- Model-Based Calibration Toolbox
- ...


Directly leverage functions in Parallel Computing Toolbox
Programming Parallel Applications

**Level of control**

- Minimal
- Some
- Extensive

**Parallel Options**

- Support built into Toolboxes
- High-Level Programming Constructs: (e.g. parfor, batch, distributed)
- Low-Level Programming Constructs: (e.g. Jobs/Tasks, MPI-based)
Running Independent Tasks or Iterations

- Ideal problem for parallel computing
- No dependencies or communications between tasks
- Examples include parameter sweeps and Monte Carlo simulations
Example: Parameter Sweep of ODEs

- Solve a 2\textsuperscript{nd} order ODE

\[ m \ddot{x} + b \dot{x} + k x = 0 \]

(1,2,...) (1,2,...)

- Simulate with different values for \(b\) and \(k\)
- Record peak value for each run
- Plot results
Summary of Example

- Mixed task-parallel and serial code in the same function
- Ran loops on a pool of MATLAB resources
The Mechanics of `parfor` Loops

```matlab
a = zeros(10, 1);
parfor i = 1:10
    a(i) = i;
end
a
```

Pool of MATLAB Workers
Parallel Computing enables you to …

Larger Compute Pool

Speed up Computations

Larger Memory Pool

Work with Large Data
Client-side Distributed Arrays

Remotely Manipulate Array from Desktop

Distributed Array Lives on the Cluster
Enhanced MATLAB Functions That Operate on Distributed Arrays

<table>
<thead>
<tr>
<th>Type of Function</th>
<th>Function Names</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data functions</td>
<td><code>cumprod</code>, <code>cumsum</code>, <code>fft</code>, <code>max</code>, <code>min</code>, <code>prod</code>, <code>sum</code></td>
</tr>
<tr>
<td>Data type functions</td>
<td><code>cast</code>, <code>cell2mat</code>, <code>cell2struct</code>, <code>celldisp</code>, <code>cellfun</code>, <code>char</code>, <code>double</code>,</td>
</tr>
<tr>
<td></td>
<td><code>fieldnames</code>, <code>int16</code>, <code>int32</code>, <code>int64</code>, <code>int8</code>, <code>logical</code>, <code>num2cell</code>,</td>
</tr>
<tr>
<td></td>
<td><code>rmfield</code>, <code>single</code>, <code>struct2cell</code>, <code>swapbytes</code>, <code>typecast</code>, <code>uint16</code>,</td>
</tr>
<tr>
<td></td>
<td><code>uint32</code>, <code>uint64</code>, <code>uint8</code></td>
</tr>
<tr>
<td>Elementary and trigonometric functions</td>
<td><code>abs</code>, <code>acos</code>, <code>acosd</code>, <code>acosh</code>, <code>acot</code>, <code>acotd</code>, <code>acoth</code>, <code>acsc</code>, <code>acscd</code>, <code>acsch</code>, <code>angle</code>,</td>
</tr>
<tr>
<td></td>
<td><code>asec</code>, <code>asecd</code>, <code>asech</code>, <code>asin</code>, <code>asind</code>, <code>asinh</code>, <code>atan</code>, <code>atan2</code>, <code>atand</code>,</td>
</tr>
<tr>
<td></td>
<td><code>atanh</code>, <code>ceil</code>, <code>complex</code>, <code>conj</code>, <code>cos</code>, <code>cosd</code>, <code>cosh</code>, <code>cot</code>, <code>cotd</code>,</td>
</tr>
<tr>
<td></td>
<td><code>coth</code>, <code>csc</code>, <code>csd</code>, <code>csch</code>, <code>exp</code>, <code>expm1</code>, <code>fix</code>, <code>floor</code>, <code>hypot</code>, <code>imag</code>,</td>
</tr>
<tr>
<td></td>
<td><code>isreal</code>, <code>log</code>, <code>log10</code>, <code>log1p</code>, <code>log2</code>, <code>mod</code>, <code>nextpow2</code>, <code>ninthroot</code>,</td>
</tr>
<tr>
<td></td>
<td><code>pow2</code>, <code>real</code>, <code>reallog</code>, <code>realpow</code>, <code>realsqrt</code>, <code>rem</code>, <code>round</code>, <code>sec</code>, <code>secd</code>,</td>
</tr>
<tr>
<td></td>
<td><code>sech</code>, <code>sign</code>, <code>sin</code>, <code>sind</code>, <code>sinh</code>, <code>sqrt</code>, <code>tan</code>, <code>tand</code>, <code>tanh</code></td>
</tr>
<tr>
<td>Elementary matrices</td>
<td><code>cat</code>, <code>diag</code>, <code>eps</code>, <code>find</code>, <code>isempty</code>, <code>isequal</code>, <code>isequalwithnan</code>,</td>
</tr>
<tr>
<td></td>
<td><code>isfinite</code>, <code>isinf</code>, <code>isnan</code>, <code>length</code>, <code>ndims</code>, <code>size</code>, <code>tril</code>, <code>triu</code></td>
</tr>
<tr>
<td>Matrix functions</td>
<td><code>chol</code>, <code>eig</code>, <code>lu</code>, <code>norm</code>, <code>normest</code>, <code>svd</code></td>
</tr>
<tr>
<td>Array operations</td>
<td><code>all</code>, <code>and</code>, <code>any</code>, <code>bitand</code>, <code>bitor</code>, <code>bitxor</code>, <code>ctranspose</code>, <code>end</code>, <code>eq</code>, <code>ge</code>, <code>gt</code>, <code>horzcat</code>,</td>
</tr>
<tr>
<td></td>
<td><code>ldivide</code>, <code>le</code>, <code>lt</code>, <code>minus</code>, <code>mldivide</code>, <code>mrdivide</code>, <code>mtimes</code>, <code>ne</code>, <code>not</code>, <code>or</code>, <code>plus</code>,</td>
</tr>
<tr>
<td></td>
<td><code>power</code>, <code>rdivide</code>, <code>subsasgn</code>, <code>subsindex</code>, <code>subsref</code>, <code>times</code>, <code>transpose</code>,</td>
</tr>
<tr>
<td></td>
<td><code>uminus</code>, <code>uplus</code>, <code>vertcat</code>, <code>xor</code></td>
</tr>
<tr>
<td>Sparse matrix functions</td>
<td><code>full</code>, <code>issparse</code>, <code>nnz</code>, <code>nonzeros</code>, <code>nzmax</code>, <code>sparse</code>, <code>spfun</code>, <code>spions</code></td>
</tr>
<tr>
<td>Special functions</td>
<td><code>dot</code></td>
</tr>
</tbody>
</table>
Programming Parallel Applications

**Level of control**
- Minimal
- Some
- Extensive

**Parallel Options**
- Support built into Toolboxes
  - High-Level Programming Constructs: (e.g. parfor, batch, distributed)
  - Low-Level Programming Constructs: (e.g. Jobs/Tasks, MPI-based)
spmd blocks

```matlab
spmd
    % single program across workers
end
```

- Mix parallel and serial code in the same function
- Run on a pool of MATLAB resources
- **Single** Program runs simultaneously across workers
  - Distributed arrays, message-passing
- **Multiple** Data spread across multiple workers
  - Data stays on workers
Client-side Distributed Arrays and SPMD

- Client-side distributed arrays
  - Class `distributed`
  - Can be created and manipulated directly from the client.
  - Simpler access to memory on labs
  - Client-side visualization capabilities

- `spmd`
  - Block of code executed on workers
  - Worker specific commands
  - Explicit communication between workers
  - Mixture of parallel and serial code
MPI-Based Functions in Parallel Computing Toolbox™

Use when a high degree of control over parallel algorithm is required

- High-level abstractions of MPI functions
  - `labSendReceive`, `labBroadcast`, and others
  - Send, receive, and broadcast any data type in MATLAB

- Automatic bookkeeping
  - Setup: communication, ranks, etc.
  - Error detection: deadlocks and miscommunications

- Pluggable
  - Use any MPI implementation that is *binary*-compatible with MPICH2
Scheduling Jobs and Tasks
Support for Schedulers

Direct Support

Open API for others

TORQUE
Programming Parallel Applications

**Level of control**

- Minimal
- Some
- Extensive

**Parallel Options**

- Support built into Toolboxes
- High-Level Programming Constructs: (e.g. parfor, batch, distributed)
- Low-Level Programming Constructs: (e.g. Jobs/Tasks, MPI-based)
Research Engineers Advance Design of the International Linear Collider with MathWorks Tools

Challenge
Design a control system for ensuring the precise alignment of particle beams in the International Linear Collider

Solution
Use MATLAB, Simulink, Parallel Computing Toolbox, and Instrument Control Toolbox software to design, model, and simulate the accelerator and alignment control system

Results
- Simulation time reduced by an order of magnitude
- Development integrated
- Existing work leveraged

"Using Parallel Computing Toolbox, we simply deployed our simulation on a large group cluster. We saw a linear improvement in speed, and we could run 100 simulations at once. MathWorks tools have enabled us to accomplish work that was once impossible."

Dr. Glen White
Queen Mary, University of London

Link to user story
Edwards Air Force Base Accelerates Flight Test Data Analysis Using MATLAB and MathWorks Parallel Computing Tools

Challenge
Accelerate performance and flying qualities flight test data analysis for unmanned reconnaissance aircraft

Solution
Use MathWorks parallel computing tools to execute MATLAB flight data processing algorithms on a 16-node computer cluster

Results
- Analysis completed 16 times faster
- Application parallelized in minutes
- Program setup time reduced from weeks to days

Parallel Computing Toolbox and MATLAB Distributed Computing Server provided a one-for-one time savings with the number of processors used. For example, with a 16-processor cluster, throughput was 16 times higher, enabling Edwards AFB engineers to accomplish in hours tasks that used to take days.

Link to user story
Argonne National Laboratory Develops Powertrain Systems Analysis Toolkit with MathWorks Tools

Challenge
Evaluate designs and technologies for hybrid and fuel cell vehicles

Solution
Use MathWorks tools to model advanced vehicle powertrains and accelerate the simulation of hundreds of vehicle configurations

Results
- Distributed simulation environment developed in one hour
- Simulation time reduced from two weeks to one day
- Simulation results validated using vehicle test data

“We developed an advanced framework and scalable powertrain components in Simulink, designed controllers with Stateflow, automated the assembly of models with MATLAB scripts, and then distributed the complex simulation runs on a computing cluster – all within a single environment.”

Sylvain Pagerit
Argonne National Laboratory