Non-rigid Displacement Vector Fields
Agenda

- Leveraging the power of vector and matrix operations
- Addressing bottlenecks
- Generating and incorporating C code
- Utilizing additional processing power
- Summary
Example: Block Processing Images

- Evaluate function at grid points
- Reevaluate function over larger blocks
- Compare the results
- Evaluate code performance
Effect of Not Preallocating Memory

\[ x = 4 \]
\[ x(2) = 7 \]
\[ x(3) = 12 \]
Benefit of Preallocation

\[ x = \text{zeros}(3,1) \]
\[ x(1) = 4 \]
\[ x(2) = 7 \]
\[ x(3) = 12 \]
Benefit of Preallocation

\[ x = \text{zeros}(3,1) \]
\[ x(1) = 4 \]
\[ x(2) = 7 \]
\[ x(3) = 12 \]
MATLAB Underlying Technologies

- Commercial libraries
  - BLAS: Basic Linear Algebra Subroutines (multithreaded)
  - LAPACK: Linear Algebra Package
  - etc.
MATLAB Underlying Technologies

- JIT/Accelerator
  - Improves looping
  - Generates on-the-fly multithreaded code
  - Continually improving
Summary of Example: Tools

- Used built-in timing functions: `tic`, `toc`

```plaintext
>> tic; v = eig(rand(1000)); toc
Elapsed time is 1.033879 seconds.
>>
```
Summary of Example: Tools

- Used built-in timing functions: `timeit`

```matlab
>> t = timeit(@(())svd(rand(1000)))
t =
    0.4190
>>
```
Summary of Example: Tools

- Used Code Analyzer to find suboptimal code

```matlab
kAvg (Nx, Ny, Nxavg, Nyavg)

5 Nx, Ny = 1500, Nxavg,

Create a surface and look at resolution.
```

```matlab
mysurf(i, j) = 5*cos((xp+yp)*2*pi)
```

Explanation

The variable 'mysurf' appears to change size on every loop iteration. Consider allocating a larger array and copying the old array contents to the new array as it grows.
Summary of Example: Techniques

- Preallocated arrays

```matlab
>> x = zeros(3,1)
```

![Preallocation example](image)
Summary of Example: Techniques

- Vectorized code

```matlab
% Precomputation of inputs
[ygrid, xgrid] = meshgrid(y, x);

mysurf = 5*cos((xgrid+ygrid)*2*pi) + ...
        2*sin(xgrid*2*pi) + 2*cos(xgrid*2*pi);
```
Other Best Practices

- Minimize dynamically changing path

```plaintext
>> cd (...)```

Other Best Practices

- Minimize dynamically changing path

\[ \text{Instead use:} \]

\[ \text{\texttt{addpath(...)}} \]
\[ \text{\texttt{fullfile(...)}} \]
Other Best Practices

- Use the functional load syntax

```matlab
>> load('myvars.mat')
```
Other Best Practices

- Use the functional load syntax

```plaintext
>> load('myvars.mat')
```

Instead use:

```plaintext
>> x = load('myvars.mat')
x =
    a: 5
    b: 'hello'
```
Other Best Practices

- Minimize changing variable class

```matlab
>> x = 1;  
>> x = 'hello';
```
Other Best Practices

- Minimize changing variable class

```matlab
>> x = 1;
>> x = 'hello';
```

instead use:
```matlab
>> x = 1;
>> xnew = 'hello';
```
Agenda

- Leveraging the power of vector and matrix operations
- Addressing bottlenecks
- Generating and incorporating C code
- Utilizing additional processing power
- Summary
Example: Fitting Data

- Load data from multiple files
- Extract a specific test
- Fit a spline to the data
- Write results to Microsoft Excel
Summary of Example: Tools

- Profiler
  - Total number of function calls
  - Time per function call

<table>
<thead>
<tr>
<th>Line Number</th>
<th>Code</th>
<th>Calls</th>
<th>Total Time</th>
<th>% Time</th>
<th>Time Plot</th>
</tr>
</thead>
<tbody>
<tr>
<td>79</td>
<td>xlswrite(fid,xlsName,[splineTli...</td>
<td>10</td>
<td>11.630 s</td>
<td>66.2%</td>
<td></td>
</tr>
<tr>
<td>37</td>
<td>textscan(fid, '%f %f \n',nTli...</td>
<td>590</td>
<td>2.642 s</td>
<td>15.0%</td>
<td></td>
</tr>
<tr>
<td>74</td>
<td>saveas(gcf, fullfile('PlotFigs...</td>
<td>10</td>
<td>2.115 s</td>
<td>12.0%</td>
<td></td>
</tr>
<tr>
<td>58</td>
<td>figure;</td>
<td>10</td>
<td>0.526 s</td>
<td>3.0%</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>nTimes = textscan(fid,'%s',1);</td>
<td>10</td>
<td>0.191 s</td>
<td>1.1%</td>
<td></td>
</tr>
</tbody>
</table>

All other lines: 0.481 s 2.7%

Totals: 17.692 s 100%

Children (called functions)

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Function Type</th>
<th>Calls</th>
<th>Total Time</th>
<th>% Time</th>
<th>Time Plot</th>
</tr>
</thead>
<tbody>
<tr>
<td>xlswrite</td>
<td>function</td>
<td>10</td>
<td>11.630 s</td>
<td>66.2%</td>
<td></td>
</tr>
<tr>
<td>saveas</td>
<td>function</td>
<td>10</td>
<td>2.114 s</td>
<td>12.0%</td>
<td></td>
</tr>
<tr>
<td>subplot</td>
<td>function</td>
<td>20</td>
<td>0.054 s</td>
<td>0.3%</td>
<td></td>
</tr>
</tbody>
</table>
Summary of Example: Techniques

- Target significant bottlenecks
  - Reduce file I/O
  - Disk is slow compared to RAM
  - When possible, use **load** and **save** commands
Summary of Example: Techniques

- Target significant bottlenecks
  - Reuse figure
  - Avoid printing to command window
Steps for Improving Performance

- First focus on getting your code working
- Then speed up the code within core MATLAB
- Consider other languages (i.e. C or Fortran MEX files) and additional processing power
Agenda

- Leveraging the power of vector and matrix operations
- Addressing bottlenecks
- Generating and incorporating C code
- Utilizing additional processing power
- Summary
Why engineers and scientists translate MATLAB to C today?

**Integrate** MATLAB algorithms w/ existing C environment using source code and static/dynamic libraries

**Prototype** MATLAB algorithms on desktops as standalone executables

**Accelerate** user-written MATLAB algorithms

**Implement** C code on processors or hand-off to software engineers
Challenges with Manual Translation from MATLAB to C

- Separate functional and implementation specification
- Leads to multiple implementations that are inconsistent
- Hard to modify requirements during development
- Difficult to keep reference MATLAB code and C code in-sync
Challenges with Manual Translation from MATLAB to C

- Manual coding errors
Challenges with Manual Translation from MATLAB to C

- Time consuming and expensive
With MATLAB Coder, design engineers can

- Maintain one design in MATLAB
- Design faster and get to C quickly
- Test more systematically and frequently
- Spend more time improving algorithms in MATLAB
Acceleration using MEX

- Speed-up factor will vary
  - When you **may** see a speedup
    - Often for Communications and Signal Processing
    - Always for Fixed-point
    - Likely for loops with states or when vectorization isn’t possible

- When you **may not** see a speedup
  - MATLAB implicitly multithreads computation
  - Built-functions call IPP or BLAS libraries
# Supported MATLAB Language Features and Functions

<table>
<thead>
<tr>
<th>Matrices and Arrays</th>
<th>Data Types</th>
<th>Programming Constructs</th>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Matrix operations</td>
<td>• Complex</td>
<td>• Arithmetic,</td>
<td>• MATLAB functions and sub-functions</td>
</tr>
<tr>
<td>• N-dimensional arrays</td>
<td>numbers</td>
<td>relational, and</td>
<td></td>
</tr>
<tr>
<td>• Subscripting</td>
<td>• Integer math</td>
<td>logical operators</td>
<td></td>
</tr>
<tr>
<td>• Frames</td>
<td>• Double/single-precision</td>
<td>Program control</td>
<td>• Variable length argument lists</td>
</tr>
<tr>
<td>• Persistent variables</td>
<td>• Fixed-point arithmetic</td>
<td>(if, for, while, switch )</td>
<td>• Function handles</td>
</tr>
<tr>
<td>• Global variables</td>
<td>• Characters</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Structures</td>
<td>• Structures</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Numeric class</td>
<td>• Numeric class</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Variable-sized data</td>
<td>• System objects</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• MATLAB Classes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• System objects</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

# Supported Functions

- > 800 MATLAB operators and functions
- > 200 System objects for
  - Signal processing
  - Communications
  - Computer vision
More Information

- To learn more visit the product page

- On-Demand Webinar:
  
  "MATLAB to C Made Easy"

Search at

Agenda

- Leveraging the power of vector and matrix operations
- Addressing bottlenecks
- Generating and incorporating C code
- Utilizing additional processing power
- Summary
Going Beyond Serial MATLAB Applications
Parallel Computing enables you to ...

**Larger Compute Pool**

Speed up Computations

**Larger Memory Pool**

Work with Large Data
Parallel Computing on the Desktop

- Speed up parallel applications on local computer
- Take full advantage of desktop power by using CPUs and GPUs
- Separate computer cluster not required
Using Additional Cores/Processors (CPUs)

- Support built into Toolboxes
Tools Providing Parallel Computing Support

- Optimization Toolbox
- Global Optimization Toolbox
- Statistics Toolbox
- Communications System Toolbox
- Simulink Design Optimization
- Bioinformatics Toolbox
- Image Processing Toolbox
- …

Directly leverage functions in Parallel Computing Toolbox

Using Additional Cores/Processors (CPUs)

- Support built into Toolboxes
- Simple programming constructs: `parfor`, `batch`, `distributed`
Running Independent Tasks or Iterations

- Ideal problem for parallel computing
- No dependencies or communications between tasks
- Examples include parameter sweeps and Monte Carlo simulations
The Mechanics of `parfor` Loops

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

Pool of MATLAB Workers
Example: Parameter Sweep of ODEs

Parallel for-loops

- Parameter sweep of ODEs
  - Deflection of a truss under a dynamic load

\[ \begin{align*}
N = 4 \\
M \ddot{x} + C \dot{x} + K x &= F
\end{align*} \]
Example: Parameter Sweep of ODEs

- Parameter sweep of ODEs
  - Deflection of a truss under a dynamic load
  - Sweeping two parameters:
    - Number of truss elements
    - Cross sectional area of truss elements

\[ M\ddot{x} + C\dot{x} + Kx = F \]
Using Additional Cores/Processors (CPUs)

- Support built into Toolboxes
- Simple programming constructs: `parfor`, `batch`, `distributed`
- Full control of parallelization: `jobs` and `tasks`, `spmd`
Scale Up to Clusters, Grids and Clouds

Desktop Computer

Parallel Computing Toolbox

Scheduler

Computer Cluster

MATLAB Distributed Computing Server
Scheduling Work

MATLAB
SIMULINK
TOOLBOXES
BLOCKSETS

Scheduler

Worker
Worker
Worker

Work → Scheduler → Result

Worker

Worker
Offload Computations with `batch`
Offload and Scale Computations with `batch`

`batch(…,'Pool',...)`
What is a Graphics Processing Unit (GPU)

- Originally for graphics acceleration, now also used for scientific calculations
- Massively parallel array of integer and floating point processors
  - Typically hundreds of processors per card
  - GPU cores complement CPU cores
- Dedicated high-speed memory
Performance Gain with More Hardware

Using More Cores (CPUs)

Using GPUs

Device Memory

GPU cores
GPU Requirements

- Parallel Computing Toolbox requires NVIDIA GPUs
- This includes the Tesla 20-series products

<table>
<thead>
<tr>
<th>MATLAB Release</th>
<th>Required Compute Capability</th>
</tr>
</thead>
<tbody>
<tr>
<td>MATLAB R2014b</td>
<td>2.0 or greater</td>
</tr>
<tr>
<td>MATLAB R2014a and earlier releases</td>
<td>1.3 or greater</td>
</tr>
</tbody>
</table>

See a complete listing at [www.nvidia.com/object/cuda_gpus.html](http://www.nvidia.com/object/cuda_gpus.html)
Programming Parallel Applications (GPU)

- Built-in support with toolboxes
Programming Parallel Applications (GPU)

- Built-in support with toolboxes
- Simple programming constructs: `gpuArray`, `gather`
Example: Solving 2D Wave Equation

- Solve 2\textsuperscript{nd} order wave equation using spectral methods:

\[
\frac{\partial^2 u}{\partial t^2} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}
\]
Benchmark: Solving 2D Wave Equation

CPU v. GPU

Intel Xeon Processor W3550 (3.07GHz), NVIDIA Tesla K20c GPU
Programming Parallel Applications (GPU)

- Built-in support with toolboxes
- Simple programming constructs: `gpuArray`, `gather`
- Advanced programming constructs: `arrayfun`, `spmd`
- Interface for experts: `CUDAKernel`, `MEX` support

www.mathworks.com/help/distcomp/run-cuda-or-ptx-code-on-gpu
www.mathworks.com/help/distcomp/run-mex-functions-containing-cuda-code
Agenda

- Leveraging the power of vector and matrix operations
- Addressing bottlenecks
- Generating and incorporating C code
- Utilizing additional processing power

Summary
Key Takeaways

- Consider performance benefit of vector and matrix operations in MATLAB
- Analyze your code for bottlenecks and address most critical items
- Leverage MATLAB Coder to speed up applications through generated C/C++ code
- Leverage parallel computing tools to take advantage of additional computing resources
Sample of Other Performance Resources

- MATLAB documentation
  MATLAB → Advanced Software Development → Performance and Memory

- Accelerating MATLAB Algorithms and Applications

- The Art of MATLAB, Loren Shure’s blog
  blogs.mathworks.com/loren/

- MATLAB Answers
  http://www.mathworks.com/matlabcentral/answers/