Mit MATLAB auf der Überholspur – Methoden zur Beschleunigung von MATLAB Anwendungen

Michael Glaßer
Application Engineering
MathWorks Germany
Key Takeaways

1. Speed up your serial code within core MATLAB
2. Easily parallelize your MATLAB code
3. Scale your parallel applications to a cluster or cloud
Acceleration Strategies Applied in MATLAB

- Best coding practices
  - Use the Code Analyzer

```matlab
1 - i = 0;
2
3 - for t = 0:.01:10
4 -     i = i + 1;
5 -     y(i) = sin(t);
6 - end
```
Acceleration Strategies Applied in MATLAB

- **Best coding practices**
  - Use the Code Analyzer
  - Preallocation

```matlab
1 - i = 0;
2 - for t = 0:.01:10
3 -   i = i + 1;
4 - end
```

'y' appears to change size on every loop iteration (within a script). Consider preallocating for speed. [Details ▼]
Acceleration Strategies Applied in MATLAB

- **Best coding practices**
  - Use the Code Analyzer
  - Preallocation

```matlab
i = 0;
y = zeros(100,1);
for t = 0:.01:10
    i = i + 1;
y(i) = sin(t);
end
```
Acceleration Strategies Applied in MATLAB

- **Best coding practices**
  - Use the Code Analyzer
  - Preallocation
  - Vectorization
Acceleration Strategies Applied in MATLAB

- **Best coding practices**
  - Use the Code Analyzer and Profiler
  - Preallocation
  - Vectorization

### Lines where the most time was spent

<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>4</td>
<td>( y(i) = \sin(t); )</td>
<td>1000001</td>
<td>0.198 s</td>
<td>52.5%</td>
<td><img src="#" alt="Time Plot" /></td>
</tr>
<tr>
<td>3</td>
<td>( i = i + 1; )</td>
<td>1000001</td>
<td>0.093 s</td>
<td>24.7%</td>
<td><img src="#" alt="Time Plot" /></td>
</tr>
<tr>
<td>5</td>
<td>end</td>
<td>1000001</td>
<td>0.086 s</td>
<td>22.6%</td>
<td><img src="#" alt="Time Plot" /></td>
</tr>
<tr>
<td>2</td>
<td>for ( t = 0:0.01:10e3 )</td>
<td>1</td>
<td>0 s</td>
<td>0%</td>
<td><img src="#" alt="Time Plot" /></td>
</tr>
<tr>
<td>1</td>
<td>( i = 0; )</td>
<td>1</td>
<td>0 s</td>
<td>0%</td>
<td><img src="#" alt="Time Plot" /></td>
</tr>
<tr>
<td>All other lines</td>
<td></td>
<td></td>
<td>0 s</td>
<td>0%</td>
<td><img src="#" alt="Time Plot" /></td>
</tr>
<tr>
<td>Totals</td>
<td></td>
<td></td>
<td>0.377 s</td>
<td>100%</td>
<td><img src="#" alt="Time Plot" /></td>
</tr>
</tbody>
</table>
Acceleration Strategies Applied in MATLAB

- **Best coding practices**
  - Use the Code Analyzer and Profiler
  - Preallocation
  - Vectorization

- **Integration with other Languages**
  - C/C++, Fortran
  - Precompiled MEX Files (MATLAB Coder)

- **More Hardware**
  - CPUs, GPUs
  - Clusters, Clouds

Parallel Computing with MATLAB

Pool of Workers

MATLAB Desktop (Client)

Parallel Computing Toolbox
Parallel Computing with MATLAB
Programming Parallel Applications

- Built-in support with Toolboxes

```matlab
% Fmincon options
options = optimset('UseParallel',true);

% Train neural network
net1 = train(net,x,t,'UseGpu','yes');
```
Tools Providing built-in Parallel Computing Support

- Optimization Toolbox
- Global Optimization Toolbox
- Statistics Toolbox
- Signal Processing Toolbox
- Neural Network Toolbox
- Image Processing Toolbox
- Communications System Toolbox
- Simulink Control Design
- …

Directly leverage functions in Parallel Computing Toolbox

Programming Parallel Applications

- Built-in support with Toolboxes
- Simple programming constructs:
  - CPU: parfor, batch, distributed
Parallel for-Loops

- Convert a for-loop to a \texttt{parfor} loop

```matlab
%% Parameter Sweep
for ii = 1:numel(kGrid)
    % Solve ODE
    [~,Y] = ode45(@(t,y) odesystem(t, y, m, ...
            bGrid(ii), kGrid(ii)), ... % input params
    [0, 25], ... % simulate for 25 seconds
    [0, 1]) ; % initial conditions
    peakVals(ii) = max(Y(:,1));
end
```
Parallel for-Loops

- Convert a for-loop to a `parfor` loop

```matlab
%% Parameter Sweep
parfor ii = 1:numel(kGrid)
    % Solve ODE
    [~,Y] = ode45(@(t,y) odesystem(t, y, m, ...
        bGrid(ii), kGrid(ii)), ... % input params
    [0, 25], ... % simulate for 25 seconds
    [0, 1]); % initial conditions
    peakVals(ii) = max(Y(:,1));
end
```

- Iterations are automatically run in parallel in separate MATLAB sessions (parallel pool)
Benchmark: Parameter Sweep of ODEs
Scaling case study for a fixed problem size with a cluster

<table>
<thead>
<tr>
<th>Workers</th>
<th>Computation (minutes)</th>
<th>Speed-up</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>173</td>
<td>1</td>
</tr>
<tr>
<td>16</td>
<td>13</td>
<td>13</td>
</tr>
<tr>
<td>32</td>
<td>6.4</td>
<td>27</td>
</tr>
<tr>
<td>64</td>
<td>3.2</td>
<td>55</td>
</tr>
<tr>
<td>96</td>
<td>2.1</td>
<td>83</td>
</tr>
<tr>
<td>128</td>
<td>1.6</td>
<td>109</td>
</tr>
<tr>
<td>160</td>
<td>1.3</td>
<td>134</td>
</tr>
<tr>
<td>192</td>
<td>1.1</td>
<td>158</td>
</tr>
</tbody>
</table>

Processor: Intel Xeon E5-2670
16 cores per node
Programming Parallel Applications

- Built-in support with Toolboxes

- Simple programming constructs:
  - CPU: `parfor`, `batch`, `distributed`
  - GPU: `gpuArray`, `gather`
Example: Corner Detection on the GPU
Example: Corner Detection on the GPU (still on CPU)

1. Calculate derivatives

\[
dx = \text{cdata}(2:\text{end}-1,3:\text{end}) - \text{cdata}(2:\text{end}-1,1:\text{end}-2);
dy = \text{cdata}(3:\text{end},2:\text{end}-1) - \text{cdata}(1:\text{end}-2,2:\text{end}-1);
dx2 = dx.*dx;
dy2 = dy.*dy;
dxy = dx.*dy;
\]

2. Smooth using convolution

\[
\text{gaussHalfWidth} = \max(1, \text{ceil}(2*\text{gaussSigma})));
\text{ssq} = \text{gaussSigma}^2;
t = -\text{gaussHalfWidth} : \text{gaussHalfWidth} ;
\text{gaussianKernel1D} = \exp\left(-\frac{(t.*t)}{2*\text{ssq}}\right)/(2*\pi*\text{ssq}) ; \quad \% \text{The Gaussian 1D filter}
\text{gaussianKernel1D} = \text{gaussianKernel1D} / \text{sum(gaussianKernel1D)} ;
\text{smooth}_\text{dx2} = \text{conv2}(\text{gaussianKernel1D}, \text{gaussianKernel1D}, \text{dx2}, 'valid' ) ;
\text{smooth}_\text{dy2} = \text{conv2}(\text{gaussianKernel1D}, \text{gaussianKernel1D}, \text{dy2}, 'valid' ) ;
\text{smooth}_\text{dxy} = \text{conv2}(\text{gaussianKernel1D}, \text{gaussianKernel1D}, \text{dxy}, 'valid' ) ;
\]

3. Calculate score

\[
\text{det} = \text{smooth}_\text{dx2} .* \text{smooth}_\text{dy2} - \text{smooth}_\text{dxy} .* \text{smooth}_\text{dxy} ;
\text{trace} = \text{smooth}_\text{dx2} + \text{smooth}_\text{dy2} ;
\text{score} = \text{det} - 0.25*\text{edgePhobia}*(\text{trace}.*\text{trace}) ;
\]
Example: Corner Detection on the GPU

cdata = gpuArray( cdata ); Move data to GPU

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 ); Bring data back to RAM
Example: Corner Detection on the GPU

Intel Xeon Processor X5650, NVIDIA Tesla C2050 GPU
Benchmark: Solving 2D Wave Equation

CPU vs GPU

<table>
<thead>
<tr>
<th>Grid Size</th>
<th>CPU (s)</th>
<th>GPU (s)</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>64 x 64</td>
<td>0.05</td>
<td>0.11</td>
<td>0.4</td>
</tr>
<tr>
<td>128 x 128</td>
<td>0.14</td>
<td>0.11</td>
<td>1.3</td>
</tr>
<tr>
<td>256 x 256</td>
<td>0.83</td>
<td>0.12</td>
<td>7.2</td>
</tr>
<tr>
<td>512 x 512</td>
<td>4.40</td>
<td>0.24</td>
<td>18</td>
</tr>
<tr>
<td>1024 x 1024</td>
<td>18.79</td>
<td>0.82</td>
<td>23</td>
</tr>
<tr>
<td>2048 x 2048</td>
<td>75.03</td>
<td>3.67</td>
<td>20</td>
</tr>
</tbody>
</table>

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

- Built-in support with Toolboxes

- Simple programming constructs:
  - CPU: `parfor`, `batch`, `distributed`
  - GPU: `gpuArray`, `gather`

- Advanced programming constructs:
  - CPU: `createJob`, `labSend`, `spmd`, ...
  - GPU: `arrayfun`, `CUDAKernel`, `MEX`
Scale Up to Clusters and Clouds

Desktop Computer

Local

MATLAB Desktop (Client)

Computer Cluster

Cluster

Scheduler

Scale Up to Clusters and Clouds
Take Advantage of Cluster Hardware

- **Offload computation:**
  - Free up desktop
  - Access better computers

- **Scale speed-up:**
  - Use more cores
  - Go from hours to minutes

- **Scale memory:**
  - Solve larger problems without re-coding algorithms
  - Utilize distributed arrays
Scale Up to Clusters and Clouds
Scale Up to Clusters and Clouds
Scale Up to Clusters and Clouds
Learn More

http://www.mathworks.de/parallel-computing
http://www.mathworks.de/products/parallel-computing/
Key Takeaways

1. Speed up your serial code within core MATLAB
2. Easily parallelize your MATLAB code
3. Scale your parallel applications to a cluster or cloud