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

Frank Graeber
Application Engineering
MathWorks Germany
Speed up the serial code within core MATLAB

- Use the MATLAB Profiler to identify performance bottlenecks

- Techniques for addressing performance
  - Vectorization
  - Preallocation

- Consider readability and maintainability
  - Looping vs. matrix operations
  - Subscripted vs. linear vs. logical
  - … and more …

Public Webinar: https://www.mathworks.com/company/events/webinars/wbnr71341.html
The Quest for Performance

Once you improved the performance of your serial MATLAB code …

- Do you need to reduce your run-time further?
- Do you need to solve larger problems?

- If so, do you have access to a …
  - Multi-core or multi-processor computer?
  - Computer cluster or cloud?
  - Graphics processing unit (GPU)?
Utilizing Additional Processing Power

Built-in multithreading
- Automatically enabled in MATLAB since R2008a
- Multiple threads in a single MATLAB computation engine

Parallel Computing using explicit techniques
- Multiple computation engines on CPU controlled by a single session
- Parallel programming constructs for interactive and batch processing
- GPU programming for massively parallel problems
Parallel Computing for the Desktop

- Implement & test parallel applications
- Gain speed on multicore systems
- Up to 12 local workers (computation engines)
- Take advantage of GPUs
- Prototype code for your cluster

Parallel Computing Toolbox
Scaling Up to Clusters and Clouds

Desktop Computer

Local

MATLAB Desktop (Client)

Computer Cluster

Cluster

Scheduler

Parallel Computing Toolbox

MATLAB Distributed Computing Server
Going Beyond Serial MATLAB Applications

MATLAB Desktop (Client)

Pool of Workers

Worker

Worker

Worker

Worker

Worker
Programming Parallel Applications (CPU)

- Built-in support with Toolboxes
- Simple programming constructs: parfor, batch, distributed, ...
- Advanced programming constructs: createJob, labSend, spmd, ...

Ease of Use

Greater Control
Tools Providing Parallel Computing Support

- Optimization Toolbox, Global Optimization Toolbox
- Statistics Toolbox
- Signal Processing Toolbox
- Neural Network Toolbox
- Image Processing Toolbox
- ... and more ...

Directly leverage functions in Parallel Computing Toolbox

Example: Parameter Sweep of ODEs

Parallel for-loops Simulink

- Parameter sweep of ODE system
  - Damped spring oscillator (also in Simulink)
  - Sweep through different values of damping and stiffness
  - Record peak value for each simulation

- Convert `for` to `parfor`

- Use pool of MATLAB workers

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

\[
m = 5, b = 2, k = 2
\]

\[
m = 5, b = 5, k = 5
\]
### Benchmark: Parameter Sweep of ODEs

**Scaling case study**

<table>
<thead>
<tr>
<th>Workers</th>
<th>Number of combinations</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>961</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>16</td>
<td>0.6</td>
</tr>
<tr>
<td>32</td>
<td>0.4</td>
</tr>
<tr>
<td>64</td>
<td>0.5</td>
</tr>
<tr>
<td>96</td>
<td>0.5</td>
</tr>
<tr>
<td>128</td>
<td>0.6</td>
</tr>
</tbody>
</table>

Processor: Intel Xeon E5-2670
16 cores per node
GPU Programming with MATLAB

- **Massively parallel:**
  - Calculations can be broken into hundreds or thousands of independent units of work
  - Problem size takes advantage of many GPU cores

- **Computationally intensive:**
  - Computation time significantly exceeds CPU/GPU data transfer time

One computation engine (client or worker) per GPU
GPU Programming Options

- Built-in support with Toolboxes
- Simple programming constructs: `gpuArray`, `gather`
- Advanced programming constructs: `arrayfun`, `bsxfun`, `spmd`
- Interface for experts: `CUDAKernel`, `MEX` support
Example: Solving 2D Wave Equation

GPU Computing

- Solve 2\textsuperscript{nd} order wave equation using spectral methods:
  \[
  \frac{\partial^2 u}{\partial t^2} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}
  \]

- Run both on CPU and GPU

- Using \texttt{gpuArray} and overloaded functions (e.g., \texttt{fft} and \texttt{ifft})
### Comparison of Code

<table>
<thead>
<tr>
<th>WaveEqu_GPU.m</th>
<th>WaveEqu_GPU.m</th>
</tr>
</thead>
<tbody>
<tr>
<td>H:\ML_SRC\2012b\WaveEqu_GPU\GUI\WaveEqu_GPU.m</td>
<td>H:\ML_SRC\2012b\WaveEqu_GPU\GUI\WaveEqu_GPU.m</td>
</tr>
<tr>
<td>Mon Jun 24 17:03:26 CEST 2013</td>
<td>Tue Jun 18 11:03:21 CEST 2013</td>
</tr>
</tbody>
</table>

4 differences found. Use the tooltip buttons to navigate to them.

```matlab
1 function WaveEqn_GPU(h,h2,object,hText,Hx) % function WaveEqn_GPU(h,h2,object,hText,Hx)
2 % solving 2nd order Wave equation using Spectral Methods % solving 2nd order Wave equation using Spectral Methods
3 % This example solves a 2nd order wave equation: utt = ux + u  % This example solves a 2nd order wave equation: utt = ux + u
4 % on the boundaries. It uses a 2nd order central differences. It uses a 2nd order central finite differences
5 [1] unmodified lines hidden

43 WuyyTL = repmat(1./(1-y(i1).*2),1,length(i1)); 43 WuyyTL = repmat(1./(1-y(i1).*2),1,length(i1)); 43
44 WuyyTR = repmat(y(i1)./(1-y(i1).*2),1,length(i1)); 44 WuyyTR = repmat(y(i1)./(1-y(i1).*2),1,length(i1)); 44
45 % % Converting variables to gpuArrays to get variables in GPU 45 % Converting variables to gpuArrays to get variables in GPU
46 46
47 dt = gpuArray(dt); 47 dt = gpuArray(dt);
48 > vV = gpuArray(vV); 49 > vV = gpuArray(vV);
49 > WLT = gpuArray(WLT); 49 > WLT = gpuArray(WLT);
50 > MLT = gpuArray(MLT); 50 > MLT = gpuArray(MLT);
51 > WRT = gpuArray(WRT); 51 > WRT = gpuArray(WRT);
52 > WRT = gpuArray(WRT); 52 > WRT = gpuArray(WRT); 52
53 > WUT = gpuArray(WUT); 53 > WUT = gpuArray(WUT); 53
54 > WUT = gpuArray(WUT); 54 > WUT = gpuArray(WUT); 54
55 > WuyyTL = gpuArray(WuyyTL); 55 > WuyyTL = gpuArray(WuyyTL); 55
56 > WuyyTR = gpuArray(WuyyTR); 56 > WuyyTR = gpuArray(WuyyTR); 56
57 % % start time-stepping % start time-stepping
58 58
59 n = 1; 59 n = 1;
```

Number of matching lines: 62
## 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>0.15</td>
<td>0.32</td>
</tr>
<tr>
<td>128 x 128</td>
<td>0.13</td>
<td>0.15</td>
<td>0.88</td>
</tr>
<tr>
<td>256 x 256</td>
<td>0.47</td>
<td>0.15</td>
<td>3.12</td>
</tr>
<tr>
<td>512 x 512</td>
<td>2.22</td>
<td>0.27</td>
<td>8.10</td>
</tr>
<tr>
<td>1024 x 1024</td>
<td>10.80</td>
<td>0.88</td>
<td>12.31</td>
</tr>
<tr>
<td>2048 x 2048</td>
<td>54.60</td>
<td>3.84</td>
<td>14.22</td>
</tr>
</tbody>
</table>

Intel Xeon Processor W3690 (3.47GHz), NVIDIA Tesla K20 GPU
Learn More

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

- Analyze your code for bottlenecks using the MATLAB profiler
- Consider performance benefit of vector and matrix operations in MATLAB
- Leverage parallel computing tools on
  - Multicore CPUs
  - Compute clusters or clouds
  - Graphics processing units (GPUs)

On the use of multiple GPUs: