Optimizing and Accelerating Your MATLAB Code

Sofia Mosesson
Senior Application Engineer
Agenda

- Optimizing for loops and using vector and matrix operations
- Indexing in different ways
- Finding and addressing bottlenecks
- Generating C code and incorporating it into your application
- Utilizing additional hardware and processing power
Example: Block Processing Images

- Calculate a function at grid points
- Take the mean of larger blocks
- Analyze and improve performance
Effect of Not Preallocating Memory

\[ x(1) = 4 \]
\[ x(2) = 7 \]
\[ x(3) = 12 \]
Benefit of Preallocation

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

- Execution Engine (>=R2015b)
  - All MATLAB code is just-in-time compiled
  - Improves “Nth run” performance

- Commercial Libraries
  - BLAS: Basic Linear Algebra Subroutines
  - LAPACK: Linear Algebra Package
  - IPP: Intel Performance Primitives
  - FFTW: Fastest Fourier Transform in the West
Other Best Practices

- Avoid “clear all”
  - Use “clear” or “clearvars” instead

- Avoid “introspection” functions
  - E.g. “dbstack”, “inputname”, “exist”, “whos”

Agenda

- Optimizing for loops and using vector and matrix operations
- Indexing in different ways
- Finding and addressing bottlenecks
- Generating C code and incorporating it into your application
- Utilizing additional hardware and processing power
**Example: Indexing**

- Loop over rows versus columns
- Use `find` command
- Use logical indexing

\[
x = \begin{bmatrix}
0.123 & 0.469 & 0.811 \\
0.532 & 0.103 & 0.775 \\
0.320 & 0.382 & 0.295
\end{bmatrix}
\]

\[
y = x; \\
\text{if } x(r,c) < 0.5 \\
y(r,c) = 0 \\
\text{end}
\]

\[
y = \begin{bmatrix}
0 & 0 & 0.811 \\
0.532 & 0 & 0.775 \\
0 & 0 & 0
\end{bmatrix}
\]
Matrix processing in memory

What is the fastest way to process MATLAB matrices with \texttt{for} loops? (i.e. de-vectorize)

a) Down the columns

b) Along the rows

c) Doesn't matter

\begin{verbatim}
>> x = magic(3)
x =
   8 1 6
   3 5 7
   4 9 2
\end{verbatim}
Agenda

- Optimizing for loops and using vector and matrix operations
- Indexing in different ways
  - Finding and addressing bottlenecks
- Generating C code and incorporating it into your application
- Utilizing additional hardware and processing power
Example: Addressing Bottlenecks

- Run and time program
- Identify bottlenecks
- Improve run time
Profiler

- Total number of function calls
- Time per function call
- Self time in a function call
- Statement coverage of code
Best Practices

- Minimize file I/O
- Reuse existing graphics components
- Avoid printing to Command Window
Agenda

- Optimizing `for` loops and using vector and matrix operations
- Indexing in different ways
- Finding and addressing bottlenecks
- Generating C code and incorporating it into your application
- Utilizing additional hardware and processing power
Why Engineers Translate MATLAB to C

- Implement C code on processors or hand off to software engineers
- Accelerate MATLAB algorithms
Challenges with Manual Translation of MATLAB to C

- Separate functional and implementation specifications
  - Leads to multiple implementations which are inconsistent
  - Hard to modify requirements during development
  - Difficult to keep MATLAB code and C code in sync

- Manual coding errors

- Time consuming and expensive process
Automatic Translation of MATLAB to C

- 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

- Speedup factor will vary

- When you **may** see a speedup:
  - Often for communications or signal processing
  - Likely for loops with states or when vectorization is not possible
  - Always for fixed point

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

- New functions and features are supported each release

<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 numbers</td>
<td>• Arithmetic, relational, and logical operators</td>
<td>• MATLAB functions and subfunctions</td>
</tr>
<tr>
<td>• N-dimensional arrays</td>
<td>• Integer math</td>
<td>• Program control (if, for, while, switch)</td>
<td>• Variable-length argument lists</td>
</tr>
<tr>
<td>• Subscripting</td>
<td>• Double/single-precision</td>
<td></td>
<td>• Function handles</td>
</tr>
<tr>
<td>• Frames</td>
<td>• Fixed-point arithmetic</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Persistent variables</td>
<td>• Characters</td>
<td></td>
<td>Supported algorithms</td>
</tr>
<tr>
<td>• Global variables</td>
<td>• Structures</td>
<td>• More than 1100 MATLAB operators (R2015b), functions, and System objects for:</td>
<td></td>
</tr>
</tbody>
</table>

http://www.mathworks.com/help/coder/language-supported-for-code-generation.html
Agenda

- Optimizing for loops and using vector and matrix operations
- Indexing in different ways
- Finding and addressing bottlenecks
- Generating C code and incorporating it into your application
- Utilizing additional hardware and processing power
Parallel Computing Paradigm

Multicore Desktops

MATLAB Desktop
(client)

Multicore Desktop

Worker 5

Worker 6

Worker 1

Worker 2

Worker 3

Worker 4
Parallel Computing Paradigm
Cluster Hardware

Cluster of computers

MATLAB Desktop (client)
Parallel Computing Paradigm

NVIDIA GPUs

MATLAB Desktop (client)

Using NVIDIA GPUs

GPU cores

Device Memory
Explicit Parallelism: Independent Tasks or Iterations

Parallel for loops

- Ideal problem for parallel computing
- No dependencies or communications between tasks
- Examples: parameter sweeps, Monte Carlo simulations
Example: Parameter Sweep
Parallel for-loops

- Deflection of customizable truss
  - Initial dynamic load, damping
  - Parameters:
    - Height of truss
    - Cross sectional area of truss elements

- Convert for to parfor

- Use pool of MATLAB workers
Offloading Serial Computations

- \( \text{job} = \text{batch}(...) \);
Offloading and Scaling Computations

- $\text{job} = \text{batch}(\ldots, 'Pool', n)$;
Migrate to Cluster / Cloud

- Use MATLAB Distributed Computing Server
- Change hardware without changing algorithm
Speed-up using NVIDIA GPUs

- **Ideal Problems**
  - Massively Parallel and/or Vectorized operations
  - Computationally Intensive
  - Algorithm consists of supported functions

- **300+ GPU-enabled MATLAB functions**

- **Additional GPU-enabled Toolboxes**
  - Neural Networks
  - Image Processing
  - Communications
  - Signal Processing

--- Learn More ---
Example: Wave Equation

- Solve 2nd order wave equation with spectral methods
- Use CPU and GPU
Optimizing for loops and using vector and matrix operations

Indexing in different ways

Finding and addressing bottlenecks

Generating C code and incorporating it into your application

Utilizing additional hardware and processing power
Key Takeaways

- Consider the performance benefits of vector and matrix operations
- Analyze your code for bottlenecks to address the critical areas
- Leverage MATLAB Coder to speed up functions with generated C code
- Leverage parallel computing tools to take advantage of additional hardware
Some Other Valuable Resources

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

- Accelerating MATLAB algorithms and applications


- MATLAB Question and Answers Site: MATLAB Answers