Algorithmenentwurf leicht gemacht: Neuigkeiten in der Signalverarbeitung in MATLAB

MATLAB EXPO 2015
DEUTSCHLAND

Marco Roggero
Senior Application Engineer
Key Takeaways

System Objects:

- can accelerate your MATLAB code
- are very easy to create, configure and execute
- support floating and fixed-point data types as well as automatic C/C++ and HDL code generation
Working with System Objects

```
>> Audiofilereader = dsp.AudioFileReader

Audiofilereader =

    System: dsp.AudioFileReader

    Properties:
        Filename: 'C:\MATLAB\2015a\toolbox\dsp\dsp\speech_dft.mp3'
        PlayCount: 1
        SampleRate: 22050
        SamplesPerFrame: 1024
        OutputDataType: 'double'
```
How to create test bench in MATLAB

%% Create and Initialize
SamplesPerFrame = 1024;
Fs = 44100;

Microphone = dsp.AudioRecorder('SamplesPerFrame', SamplesPerFrame);
Spectra = dsp.SpectrumAnalyzer('SampleRate', Fs);

%% Stream processing loop
tic;
while toc < 20
    % Read frame from microphone
    audioIn = step(Microphone);

    % View audio spectrum
    step(Spectra, audioIn);
End

%% Terminate
release(Microphone)
release(Spectra)
Batch processing

All the data

Work on all the data at once…

Deliver all at once
Systems Objects enable stream processing

- Each frame available as soon as processed
  - Reduced memory footprint
  - Near real-time / Life analysis
- Typical applications
  - Communications Systems
  - Audio / video signal processing
  - Data acquisition
Stream processing in MATLAB

- Streaming techniques* process continuous data from a captured signal or large file by dividing it into “frames” and fully processes each frame before the next one arrives
  - Memory efficient

- Streaming algorithms in DSP System Toolbox provide
  - Implicit data buffering, state management and indexing
  - Simulation speed-up by reducing overhead
Where to find System Objects?

DSP System Toolbox

Computer Vision System Toolbox

Phased Array System Toolbox

Communications System Toolbox
Antenna Toolbox

Released with 2015a version

Antenna library elements

- dipole: Creates a strip dipole on the Y-Z plane
- dipoleFolded: Creates a folded dipole antenna
- dipoleVes: Creates a V-dipole antenna
- bowtieTriangular: Creates a planar bowtie antenna on the Y-Z plane
- bowtieRounded: Creates a rounded bowtie dipole on the Y-Z plane
- monopole: Creates a monopole over a rectangular ground plane
- monopoleTopHat: Creates a capacitively loaded monopole over a rectangular ground plane
- invertedF: Creates an inverted-F antenna over a rectangular ground plane
- invertedL: Creates an inverted-L antenna over a rectangular ground plane
- loopCircular: Creates a circular loop on the X-Y plane
- loopRectangular: Creates a rectangular loop on the X-Y plane
- spiralArchimedean: Creates an Archimedean spiral antenna on the X-Y plane
- spiralEquispiral: Creates an equiangular spiral antenna on the X-Y plane
- helix: Creates a helix antenna on a circular ground plane
- yagiUda: Creates a Yagi-Uda array along the Z-axis
- cavity: Creates a cavity backed antenna
- reflector: Creates a reflector backed antenna
- patchMicrostrip: Creates a rectangular microstrip patch antenna
- slot: Creates a rectangular slot antenna on a ground plane
- pifa: Creates a planar inverted-F antenna on a rectangular ground plane
- vivaldi: Creates a vivaldi notch antenna on a ground plane
Signal processing example for algorithm’s - performance comparison
Compare algorithm execution time with different implementations:

<table>
<thead>
<tr>
<th>Versions of the Algorithm</th>
<th>Elapsed Time (sec)</th>
<th>Acceleration Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Traditional MATLAB functions</td>
<td>18.0953</td>
<td>1.0000</td>
</tr>
<tr>
<td>2. System Objects in-lieu of MATLAB functions</td>
<td>3.6914</td>
<td>4.9020</td>
</tr>
<tr>
<td>3. MATLAB Coder MEX version</td>
<td>1.5241</td>
<td>11.8725</td>
</tr>
</tbody>
</table>
Accelerating algorithm execution

- Pre-allocation and vectorization
- Pre-defined efficient implementations of algorithms
- Generate MEX files automatically with MATLAB Coder
- Parallel computations on multicore computers, GPUs, and clusters
Options to speeding up MATLAB algorithms:

- Toolbox Functions
- Toolbox Functions + PCT (parfor)
- System Objects
- System Objects + PCT (parfor)
- System Objects + Code Generation
- System Objects + Code Generation + PCT (parfor)

PCT: Parallel Computing Toolbox
Dual core machine used for this example

Faster
Why Engineers 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
Automatic code generation and System Objects

C/C++ code Generation supported for nearly all System Objects of DSP, Communication and Computer Vision System Toolboxes

Support of floating and fixed-point data types

Code Replacement libraries enable generation of target optimized embedded C code

Large library of System Objects supported from HDL code generation
Key Takeaways

System Objects:

- can accelerate your MATLAB code
- are very easy to create, configure and execute
- support floating and fixed-point data types as well as automatic C/C++ and HDL code generation