Deploying Deep Learning Networks to Embedded GPUs and CPUs
MATLAB Deep Learning Framework

Access Data
- Manage large image sets
- Automate image labeling
- Easy access to models

Design + Train
- Acceleration with GPU’s
- Scale to clusters

Deploy
- Automate compilation to GPUs and CPUs using GPU Coder:
  - 5x faster than TensorFlow
  - 2x faster than MXNet
Design Deep Learning & Vision Algorithms

Transfer Learning Workflow

Images → Transfer Learning → New Classifier

- Load Reference Network
- Modify Network Structure
- Learn New Weights

Labels: Hot dogs, Pizzas, Ice cream, Chocolate cake, French fries
Example: Transfer Learning in MATLAB

1. **Set up training dataset**
   ```matlab
   cifarFolder = 'cifar10Train';
   categories = {'Cars', 'Trucks', 'BigTrucks', 'Suvs', 'Vans'};
   imds = imageDatastore(fullfile(cifarFolder, categories),...
                     'LabelSource', 'filenames');
   imds = splitEachLabel(imds, 500, 'randomize'); % we only need 500 images per class
   imds.ReadFcn = @readFunctionTrain;
   ``

2. **Load Reference Network**
   ```matlab
   net = alexnet;
   layers = net.Layers;
   ``

3. **Modify Network Structure**
   ```matlab
   % modify network
   layers = layers(1:end-3);
   layers(end+1) = fullyConnectedLayer(64, 'Name', 'special_2');
   layers(end+1) = reluLayer;
   layers(end+1) = fullyConnectedLayer(5, 'Name', 'fc8_2');
   layers(end+1) = softmaxLayer;
   layers(end+1) = classificationLayer();
   ``

4. **Learn New Weights**
   ```matlab
   % train!
   options = trainingOptions('sgdm', ...%
                            'LearnRateSchedule', 'none', ...%
                            'InitialLearnRate', 0.001, ...%
                            'MaxEpochs', 20, ...%
                            'MiniBatchSize', 128);
   myConvnet = trainNetwork(imds, layers, options);
   ```
Scaling Up Model Training Performance

Training on the AWS (EC2)

Multiple GPU support

MATLAB is more than 4x faster than TensorFlow

Single GPU performance

MATLAB EXPO 2018
Visualizing and Debugging Intermediate Results

- Many options for visualizations and debugging
- Examples to get started

Filters

Layer Activations

Activations

Deep Dream

Training Accuracy Visualization

Convolution → Activation → Pooling → Convolution → Activation → Pooling → ... → Convolution → Activation → Pooling

Feature Visualization

MATLAB EXPO 2018
GPU Coder for Deployment

Accelerated implementation of parallel algorithms on GPUs & CPUs

Deep Neural Networks
Deep Learning, machine learning

Image Processing and Computer Vision
Image filtering, feature detection/extraction

Signal Processing and Communications
FFT, filtering, cross correlation,

5x faster than TensorFlow
2x faster than MXNet

60x faster than CPUs for stereo disparity

20x faster than CPUs for FFTs
GPUs and CUDA

C/C++

CUDA kernels

CPU Memory Space

GPU Memory Space

I/O

SECURITY ENGINES
4K60 VIDEO ENCODER
60 VIDEO DECODER
AUDIO ENGINE
2D ENGINE
DISPLAY ENGINES
128-bit LPDDR4
BOOT and PM PROC
GigE Ethernet MAC
IMAGE PROC (ISP)
Safety Engine

GPU CUDA Cores

ARM Cortex

MATLAB EXPO 2018
Challenges of Programming in CUDA for GPUs

- Learning to program in CUDA
  - Need to rewrite algorithms for parallel processing paradigm

- Creating CUDA kernels
  - Need to analyze algorithms to create CUDA kernels that maximize parallel processing

- Allocating memory
  - Need to deal with memory allocation on both CPU and GPU memory spaces

- Minimizing data transfers
  - Need to minimize while ensuring required data transfers are done at the appropriate parts of your algorithm
GPU Coder Helps You Deploy to GPUs Faster

- Library function mapping
- Loop optimizations
- Dependence analysis

- Data locality analysis
- GPU memory allocation

- Data-dependence analysis
- Dynamic memcpy reduction
GPU Coder Generates CUDA from MATLAB: saxpy

Scalarized MATLAB

```matlab
for i = 1:length(x)
    z(i) = a .* x(i) + y(i);
end
```

Vectorized MATLAB

```matlab
z = a .* x + y;
```

Loops and matrix operations are directly compiled into kernels

CUDA kernel for GPU parallelization

```c
static __global__ __launch_bounds__(512, 1) void saxpy_kernel4(const real32_T *y,
        const real32_T *x, real32_T a, real32_T *z)
{
    int32_T i;

    i = (int32_T)(((gridDim.x * blockDim.z * blockDim.y) + blockDim.x) * blockDim.y * blockDim.z) -
        blockDim.x * blockDim.y - blockDim.z - threadIdx.y * blockDim.x - threadIdx.z;

    if (i < 1048576) {
        z[i] = (real32_T)(a * x[i] + y[i]);
    }
}
```
Generated CUDA Optimized for Memory Performance

Kernel data allocation is automatically optimized

CUDA kernel for GPU parallelization

```c
static __global__ __launch_bounds__(512, 1) void kernel3(real_T *z0, real_T *z1, real_T *z2, real_T *z3)
{
    real_T z_im;
    real_T y[1000000];
    int32_T threadIdx;
    threadIdx = (int32_T)(blockIdx.x * blockDim.x + threadIdx.x);
    if (threadIdx >= 1000000) {
        z_im = z[threadIdx].re * z[threadIdx].im + z[threadIdx].im * z[threadIdx].re;
        y[threadIdx].re = (z[threadIdx].re) * z[threadIdx].re - (z[threadIdx].im) * z[threadIdx].im;
        y[threadIdx].im = z[threadIdx].im * z[threadIdx].re;
    } else {
        y[threadIdx].re = hypot(z[threadIdx].re, z[threadIdx].im);
        y[threadIdx].im = z[threadIdx].im / y[threadIdx].re;
        z[threadIdx].re = (z[threadIdx].re) / y[threadIdx].re - (z[threadIdx].im) * z[threadIdx].im / y[threadIdx].re;
    }
}
```

CUDA

```matlab
... % mandelbrot computation
cudaMalloc((void**)&gpu_ygrid, 8000000U);
cudaMalloc((void**)&gpu_xgrid, 8000000U);

for (n = 0; n < (int32_T)(maxIterations + 1.0); n++) {
    kernel<<<dim3(1934U, 1, 1U), dim3(S128U, 1, 1U)>>>(gpu_ygrid, gpu_xgrid, gpu_z0, gpu_z1, gpu_z2, gpu_z3);
}
... % free CUDA memory
```

Mandelbrot space

MATLAB EXPO 2016
Algorithm Design to Embedded Deployment Workflow

MATLAB algorithm (functional reference) --> GPU Coder

1. Functional test
2. Deployment unit-test
3. Deployment integration-test
4. Real-time test

Call CUDA from MATLAB directly

Build type

Call CUDA from (C++) hand-coded main()
Demo: Alexnet Deployment with ‘mex’ Code Generation
Algorithm Design to Embedded Deployment on Tegra GPU

MATLAB algorithm (functional reference)

GPU Coder

Build type

Call CUDA from MATLAB directly

Call CUDA from (C++) hand-coded main()

Cross-compiled .lib

Call CUDA from (C++) hand-coded main(). Cross-compiled on host with Linaro toolchain

1. Functional test
   (Test in MATLAB on host)

2. Deployment unit-test
   (Test generated code in MATLAB on host + GPU)

3. Deployment integration-test
   (Test generated code within C/C++ app on host + GPU)

4. Real-time test
   (Test generated code within C/C++ app on Tegra target)
Alexnet Deployment to Tegra: Cross-Compiled with ‘lib’

Two small changes
1. Change build-type to ‘lib’
2. Select cross-compile toolchain
End-to-End Application: Lane Detection

Alexnet

Transfer Learning

Output of CNN is lane parabola coefficients according to: \( y = ax^2 + bx + c \)

Image → Lane detection CNN → Left lane coefficients → Right lane coefficients → Post-processing (find left/right lane points) → Image with marked lanes

GPU coder generates code for whole application
Deep Learning Network Support (with Neural Network Toolbox)

SeriesNetwork
- Single-in single-out

GPU Coder: R2017b
- Networks: MNist, Alexnet, YOLO, VGG, Lane detection, Pedestrian detection

DAGNetwork

GPU Coder: R2018a
- Networks: GoogLeNet, ResNet, SegNet, DeconvNet, Semantic segmentation, Object detection
Semantic Segmentation

Running in MATLAB

Generated Code from GPU Coder
Deploying to CPUs

Deep Learning Networks

GPU Coder

Intel MKL-DNN Library

NVIDIA TensorRT & cuDNN Libraries

ARM Compute Library

NVIDIA TensorRT & cuDNN Libraries
Deploying to CPUs

Deep Learning Networks -> GPU Coder

Desktop CPU

NVIDIA TensorRT & cuDNN Libraries

Raspberry Pi board

MATLAB EXPO 2018
How Good is Generated Code Performance

- Performance of image processing and computer vision

- Performance of CNN inference (Alexnet) on Titan XP GPU

- Performance of CNN inference (Alexnet) on Jetson (Tegra) TX2
GPU Coder for Image Processing and Computer Vision

- Fog removal: 5x speedup
- Distance transform: 8x speedup
- Frangi filter: 3x speedup
- Stereo disparity: 50x speedup
- Ray tracing: 18x speedup
- SURF feature extraction: 700x speedup
Alexnet Inference on NVIDIA Titan Xp

<table>
<thead>
<tr>
<th>Testing platform</th>
<th>CPU</th>
<th>GPU</th>
<th>cuDNN</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU Test</td>
<td>Intel(R) Xeon(R) CPU E5-1650 v4 @ 3.60GHz</td>
<td>Pascal Titan Xp</td>
<td>v7</td>
</tr>
</tbody>
</table>

Frames per second vs Batch Size

- **GPU Coder + TensorRT (3.0.1, int8)**
- **GPU Coder + TensorRT (3.0.1)**
- **GPU Coder + cuDNN**
- **MXNet (1.1.0)**
- **TensorFlow (1.6.0)**
VGG-16 Inference on NVIDIA Titan Xp

<table>
<thead>
<tr>
<th>Testing platform</th>
<th>CPU</th>
<th>GPU</th>
<th>cuDNN</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Intel(R) Xeon(R) CPU E5-1650 v4 @ 3.60GHz</td>
<td>Pascal Titan Xp</td>
<td>v7</td>
</tr>
</tbody>
</table>

Frames per second vs Batch Size chart:
- **GPU Coder + TensorRT (3.0.1, int8)**
- **GPU Coder + TensorRT (3.0.1)**
- **GPU Coder + cuDNN MXNet (1.1.0)**
- **TensorFlow (1.6.0)**
Alexnet Inference on Jetson TX2: Frame-Rate Performance

![Graph showing performance comparison between different tools and frameworks.]

- **TensorRT (2.1)**: 1.15x
- **MATLAB GPU Coder (R2017b)**: 2x
- **C++ Caffe (1.0.0-rc5)**: Frames per second

To be updated with R2018a benchmarks soon.

Contact Bill.Chou@mathworks.com for more information.

MATLAB EXPO 2018
Alexnet Inference on Jetson TX2: Memory Performance

To be updated with R2018a benchmarks soon

Contact Bill.Chou@mathworks.com for more information

MATLAB GPU Coder (R2017b)

C++ Caffe (1.0.0-rc5)

TensorRT 2.1 (using giexec wrapper)
Design Your DNNs in MATLAB, Deploy with GPU Coder

- **Access Data**
  - Manage large image sets
  - Automate image labeling
  - Easy access to models

- **Design + Train**
  - Acceleration with GPU’s
  - Scale to clusters

- **Deploy**
  - Automate compilation to GPUs and CPUs using GPU Coder:
    - 5x faster than TensorFlow
    - 2x faster than MXNet
감사합니다.