MATLAB EXPO 2018

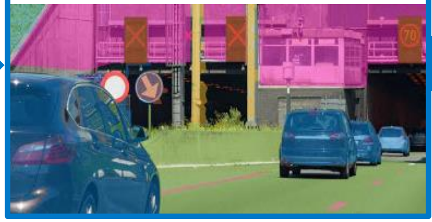
Deploying Deep Learning Networks to Embedded GPUs and CPUs

Rainer Mümmler

MATLAB Deep Learning Framework

Access Data

Design + Train



Deploy

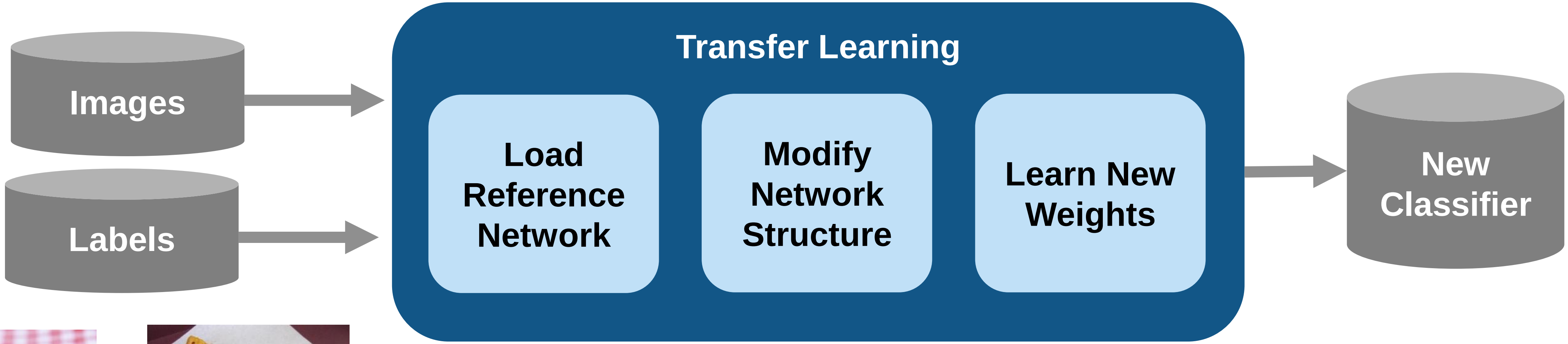
- **Manage** large image sets
- **Automate** image labeling
- **Easy access** to models

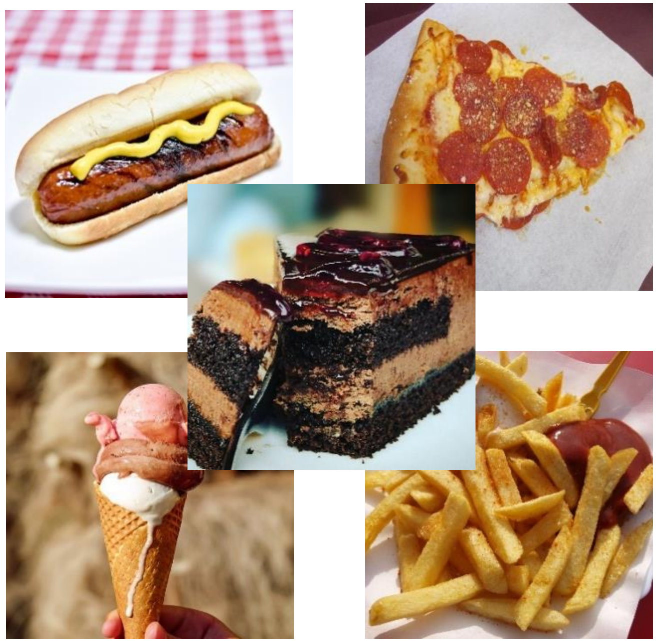
- **Acceleration** with GPU's
- **Scale** to clusters

- **Automate compilation to GPUs and CPUs using GPU Coder**

Design Deep Learning & Vision Algorithms

Transfer Learning Workflow





Training Data

Labels: Hot dogs, Pizzas, Ice cream, Chocolate cake, French fries

Example: Transfer Learning in MATLAB

Set up training dataset

```
%% set up training dataset
cifarFolder = 'cifar10Train';
categories = {'Cars', 'Trucks', 'BigTrucks', 'Suvs', 'Vans'};
imds = imageDatastore(fullfile(cifarFolder, categories), ...
    'LabelSource', 'foldernames');

imds = splitEachLabel(imds, 500, 'randomize'); % we only need 500 images per class
imds.ReadFcn = @readFunctionTrain;
```

Load Reference Network

```
%% load reference network
net = alexnet;
layers = net.Layers;
```

Modify Network Structure

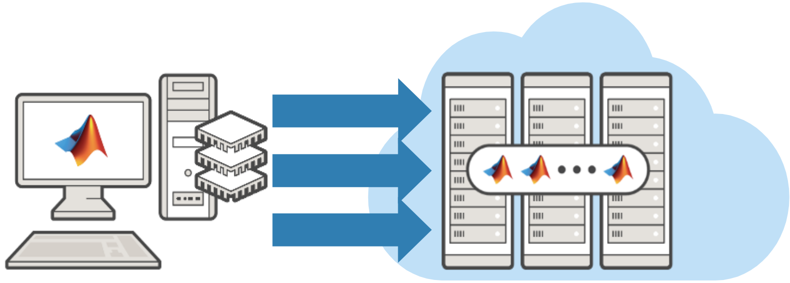
```
%% modify network
layers = layers(1:end-3);

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();
```

Learn New Weights

```
%% train!
options = trainingOptions('sgdm', ...
    'LearnRateSchedule', 'none',...
    'InitialLearnRate', .0001,...
    'MaxEpochs', 20, ...
    'MiniBatchSize', 128);

myConvnet = trainNetwork(imds, layers, options);
```

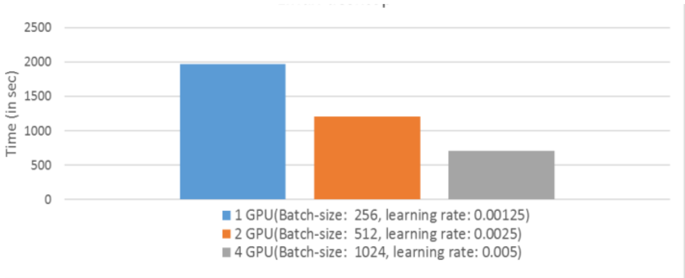
Scaling Up Model Training Performance

`'ExecutionEnvironment', 'parallel' );`



Training on the AWS (EC2)

`'ExecutionEnvironment', 'multi-gpu' );`

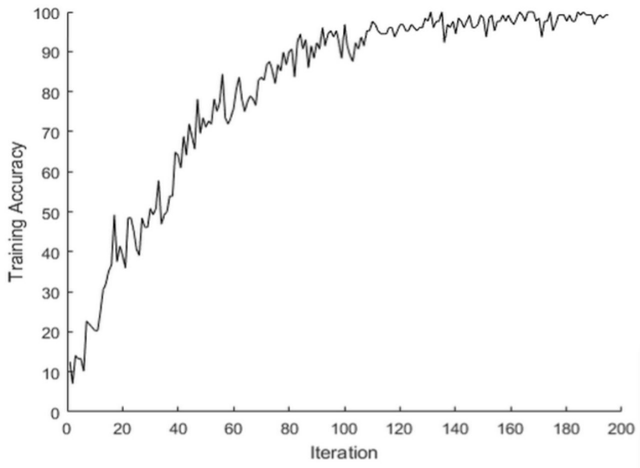


Multiple GPU support

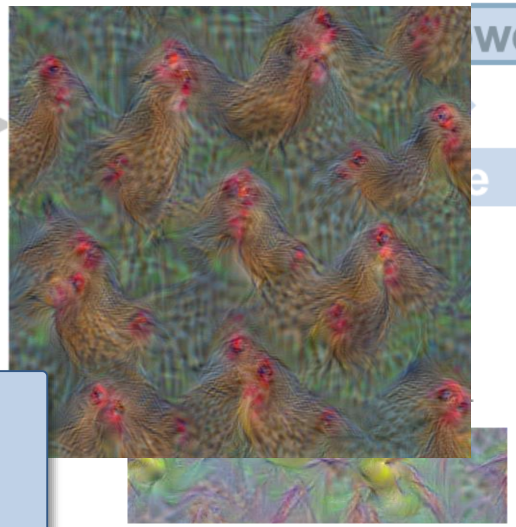
```
opts = trainingOptions('sgdm', ...
    'MaxEpochs', 100, ...
    'MiniBatchSize', 250, ...
    'InitialLearnRate', 0.00005, ...
    'ExecutionEnvironment', 'auto' );
```

Visualizing and Debugging Intermediate Results

Training Accuracy Visualization



Deep Dream



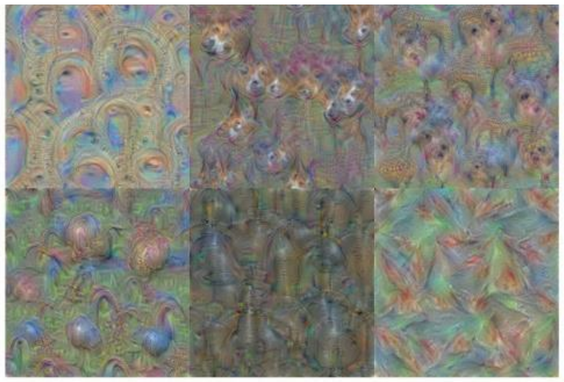
- Many options for visualizations and debugging
- Examples to get started

Filters

Layer Activations

Activations

Feature Visualization



GPU Coder for Deployment

MATLAB

GPU Coder

Accelerated implementation of parallel algorithms on GPUs & CPUs

Intel MKL-DNN Library

NVIDIA CUDA C/C++

ARM Compute Library

R2018b
CPUs with MATLAB Coder

Deep Neural Networks

Deep Learning, machine learning

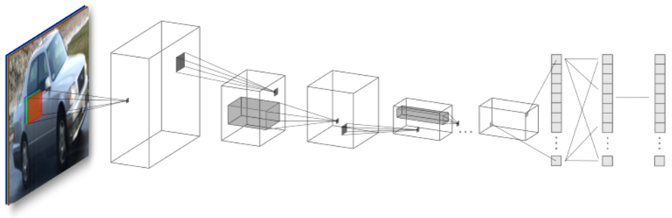


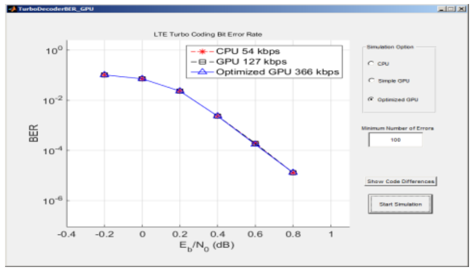
Image Processing and Computer Vision

Image filtering, feature detection/extraction

60x faster than CPUs for stereo disparity

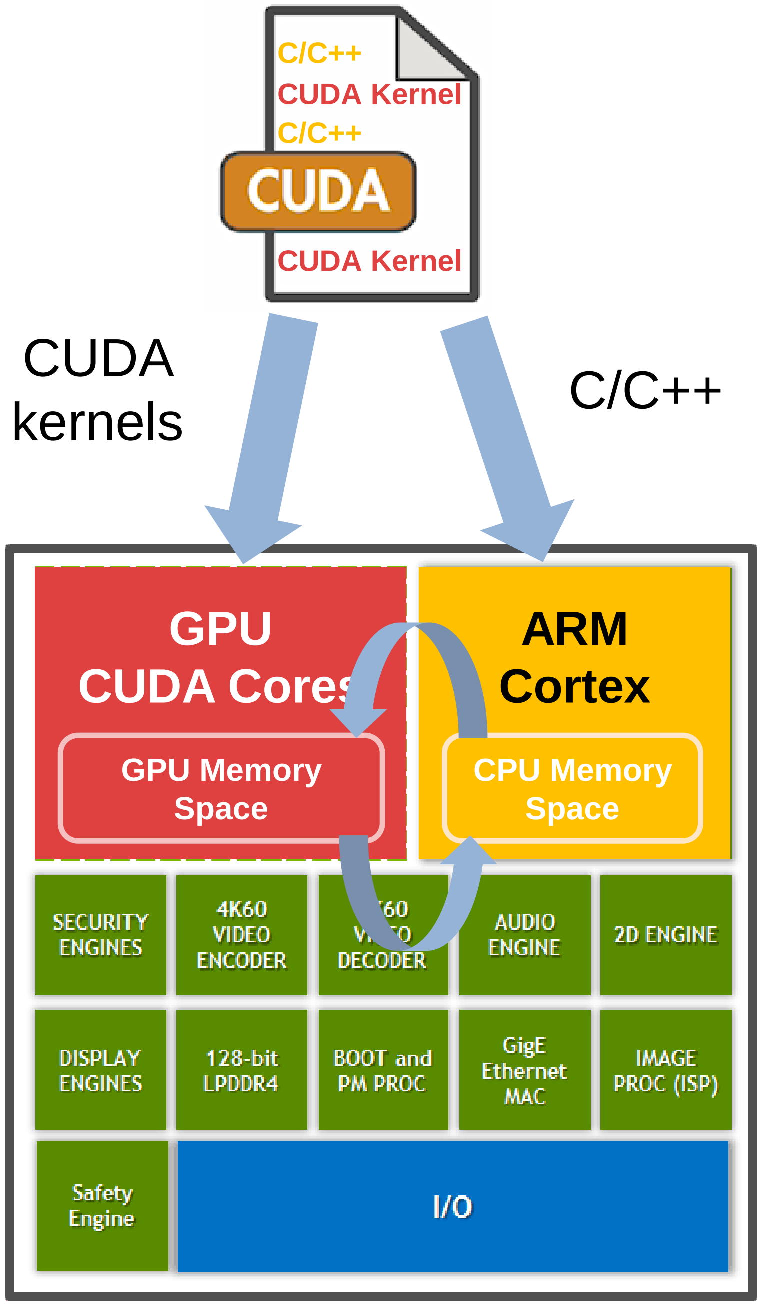
Signal Processing and Communications

FFT, filtering, cross correlation,



20x faster than CPUs for FFTs

GPUs and CUDA



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

MATLAB

GPU Coder

- CUDA Kernel creation
  - Library function mapping
  - Loop optimizations
  - Dependence analysis
- Memory allocation
  - Data locality analysis
  - GPU memory allocation
- Data transfer minimization
  - Data-dependence analysis
  - Dynamic memcpy reduction

NVIDIA CUDA C/C++

GPU Coder Generates CUDA from MATLAB: saxpy

Scalarized MATLAB

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

GPU Coder

Vectorized MATLAB

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

CUDA

```
cudaMalloc(&gpu_z, 8388608UL);
cudaMalloc(&gpu_x, 4194304UL);
cudaMalloc(&gpu_y, 4194304UL);
cudaMemcpy((void *)gpu_y, (void *)y, 4194304UL, cudaMemcpyHostToDevice);
cudaMemcpy((void *)gpu_x, (void *)x, 4194304UL, cudaMemcpyHostToDevice);
saxpy_kernel1<<<dim3(2048U, 1U, 1U), dim3(512U, 1U, 1U)>>>(gpu_y, gpu_x, a,
  gpu_z);
cudaMemcpy((void *)z, (void *)gpu_z, 8388608UL, cudaMemcpyDeviceToHost);
cudaFree(gpu_y);
cudaFree(gpu_x);
cudaFree(gpu_z);
```

CUDA kernel for GPU parallelization

```
static __global__ __launch_bounds__(512, 1) void saxpy_kernel1(const real32_T *y,
  const real32_T *x, real32_T a, real_T *z)
{
  int32_T i;

  i = (int32_T)(((((gridDim.x * gridDim.y * blockIdx.z + gridDim.x * blockIdx.y)
                  + blockIdx.x) * (blockDim.x * blockDim.y * blockDim.z) +
                 threadIdx.z * blockDim.x * blockDim.y) + threadIdx.y *
                blockDim.x) + threadIdx.x);
  if (!(i >= 1048576)) {
    z[i] = (real_T)(a * x[i] + y[i]);
  }
}
```

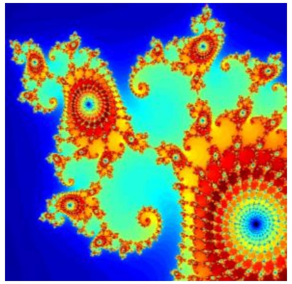
Loops and matrix operations are directly compiled into kernels

Generated CUDA Optimized for Memory Performance

Kernel data allocation is automatically optimized

```
z = z0;
for n = 0:maxIterations
    z = z.*z + z0;
    inside = abs( z )<=2;
    count = count + inside;
end
count = log( count );
```

GPU Coder



Mandelbrot space

CUDA kernel for GPU parallelization

```
static __global__ __launch_bounds__(512, 1) void kernel3(creal_T *z0, real_T
  *count, creal_T *z)
{
  real_T z_im;
  real_T y[1000000];
  int32_T threadIdX;
  threadIdX = (int32_T)(blockDim.x * blockIdx.x + threadIdx.x);
  if (!(threadIdX >= 1000000)) {
    z_im = z[threadIdX].re * z[threadIdX].im + z[threadIdX].im * z[threadIdX].re;
    z[threadIdX].re = (z[threadIdX].re * z[threadIdX].re - z[threadIdX].im *
                       z[threadIdX].im) + z0[threadIdX].re;
    z[threadIdX].im = z_im + z0[threadIdX].im;
    y[threadIdX] = hypot(z[threadIdX].re, z[threadIdX].im);
    count[threadIdX] += (real_T)(y[threadIdX] <= 2.0);
  }
}
```

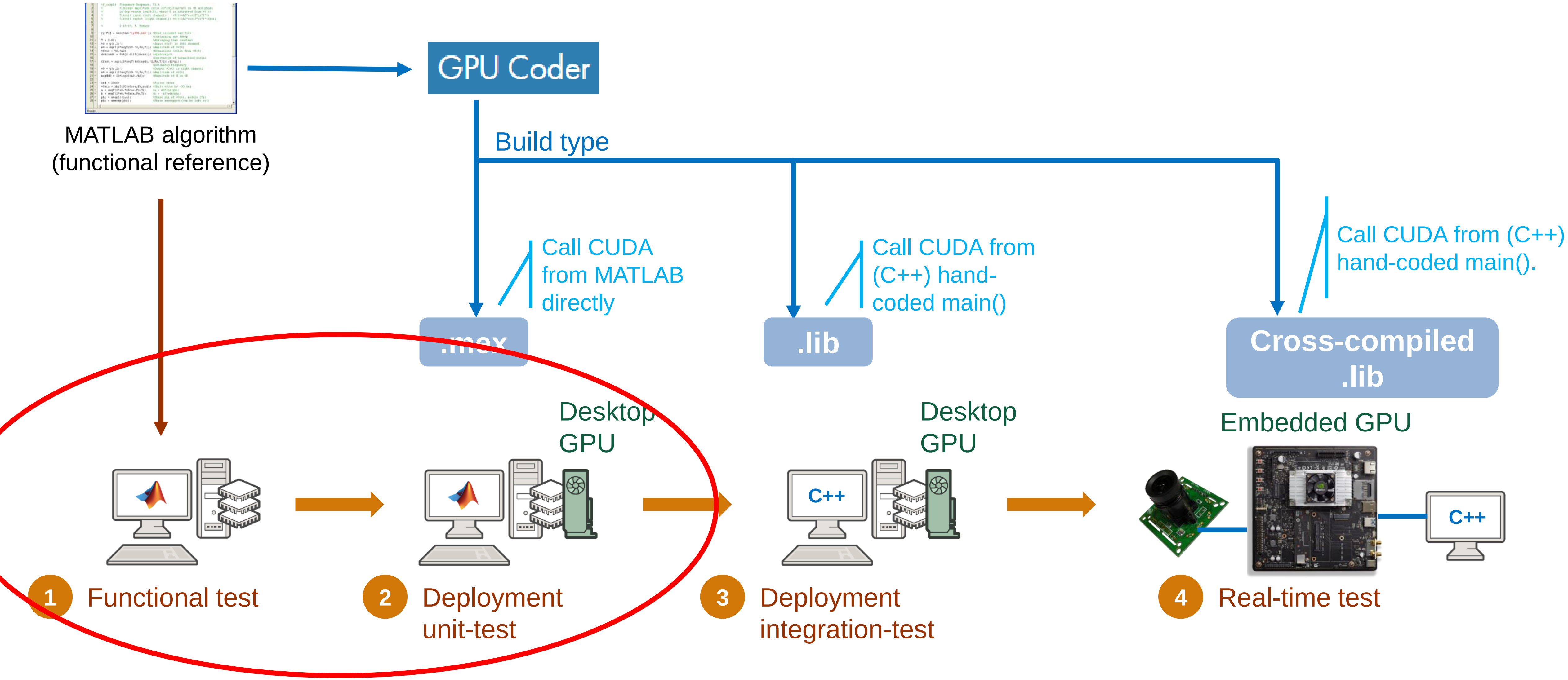
CUDA

...
...

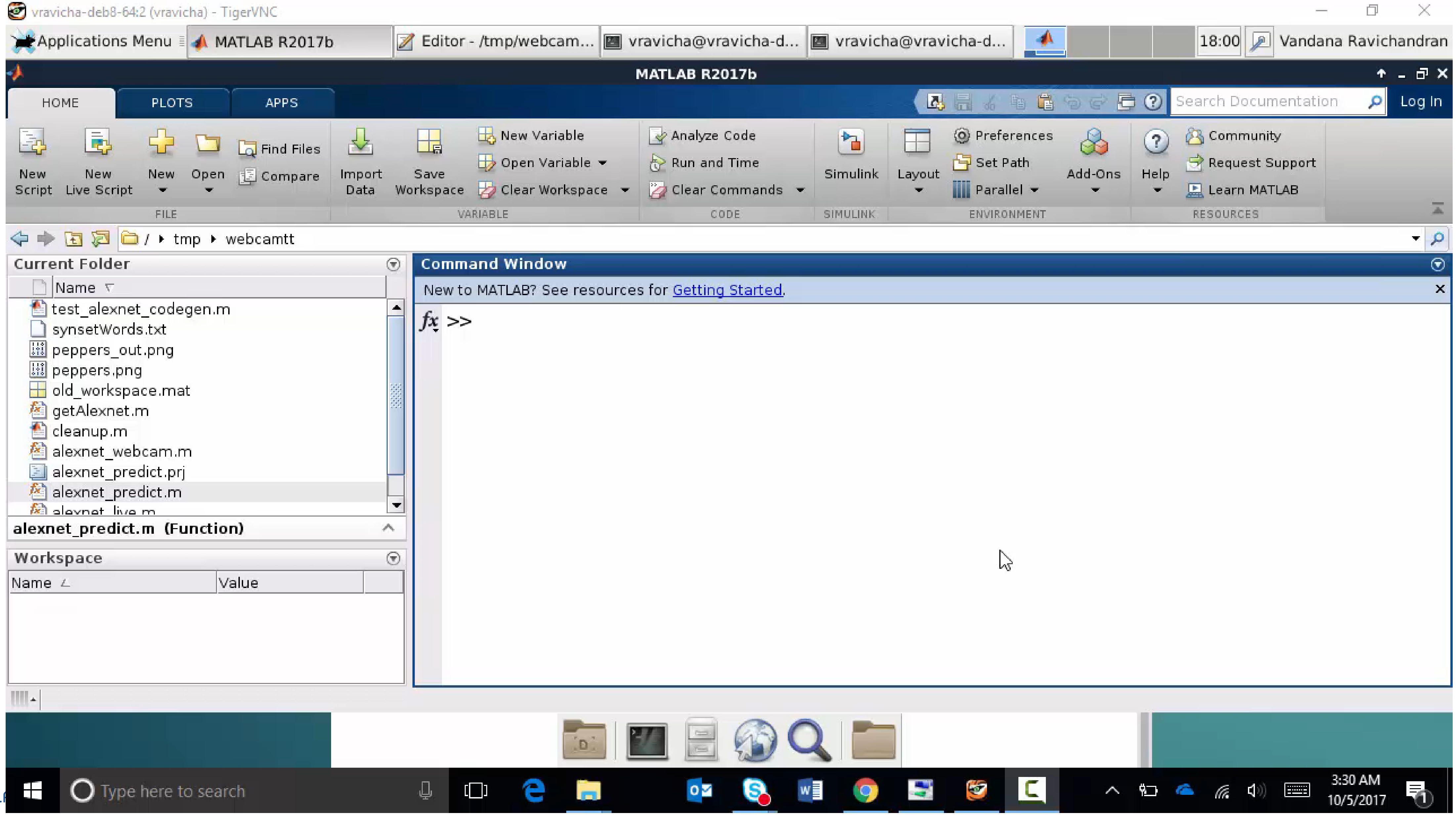
```
cudaMalloc(&gpu_xGrid, 8000000U);
cudaMalloc(&gpu_yGrid, 8000000U);

/*  mandelbrot computation */
cudaMemcpy(gpu_yGrid, yGrid, 8000000U, cudaMemcpyHostToDevice);
cudaMemcpy(gpu_xGrid, xGrid, 8000000U, cudaMemcpyHostToDevice);
kernel1<<<dim3(1954U, 1U, 1U), dim3(512U, 1U, 1U)>>>(gpu_yGrid, gpu_xGrid,
  gpu_z, gpu_count, gpu_z0);
for (n = 0; n < (int32_T)(maxIterations + 1.0); n++) {
  kernel3<<<dim3(1954U, 1U, 1U), dim3(512U, 1U, 1U)>>>(gpu_z0, gpu_count,
    gpu_z);
}

kernel2<<<dim3(1954U, 1U, 1U), dim3(512U, 1U, 1U)>>>(gpu_count);
cudaMemcpy(count, gpu_count, 8000000U, cudaMemcpyDeviceToHost);
cudaFree(gpu_yGrid);
```

...
...

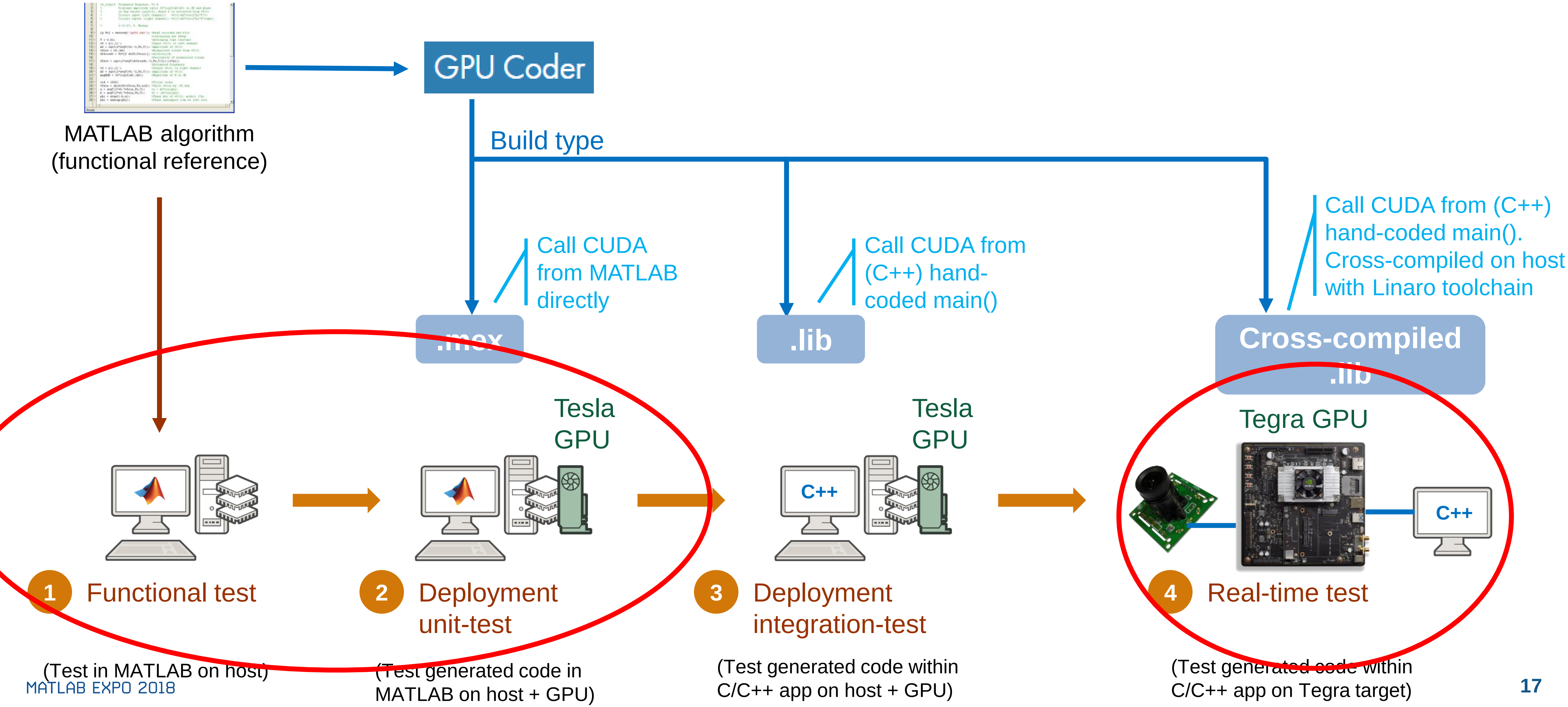
Algorithm Design to Embedded Deployment Workflow



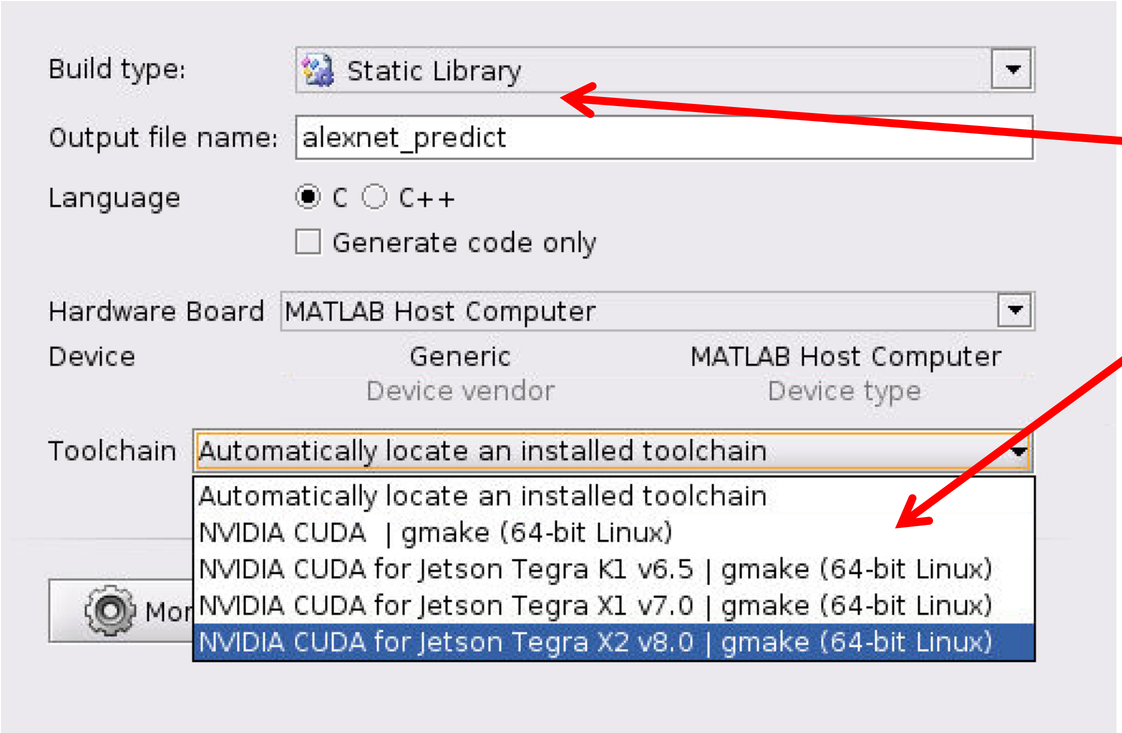
Demo: Alexnet Deployment with ‘mex’ Code Generation



Algorithm Design to Embedded Deployment on Tegra GPU



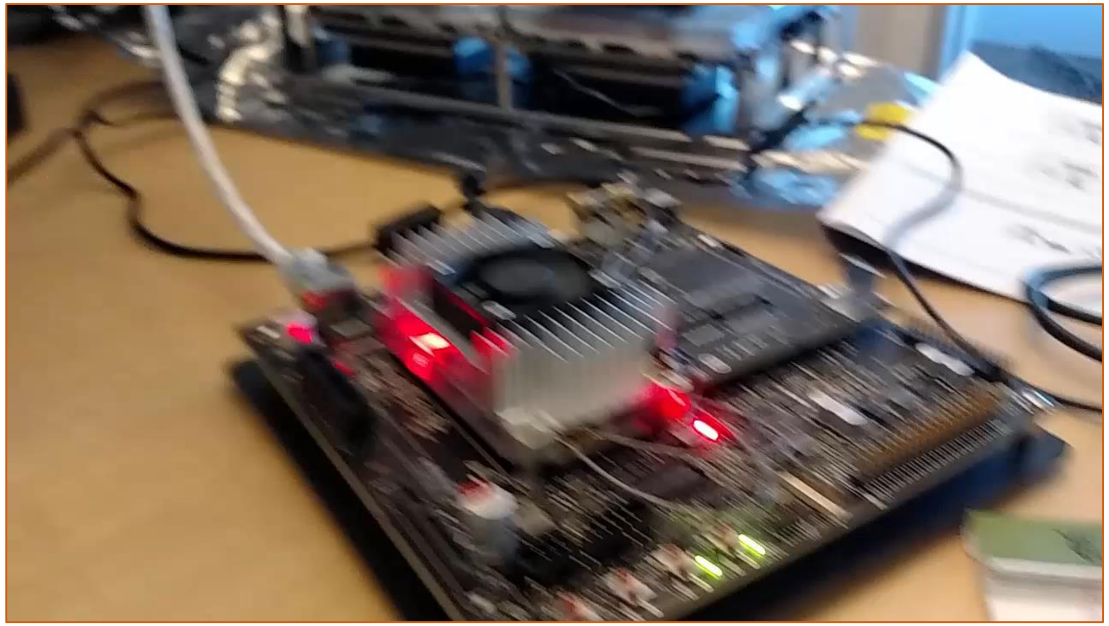
Alexnet Deployment to Tegra: Cross-Compiled with 'lib'



<u>Two small changes</u>

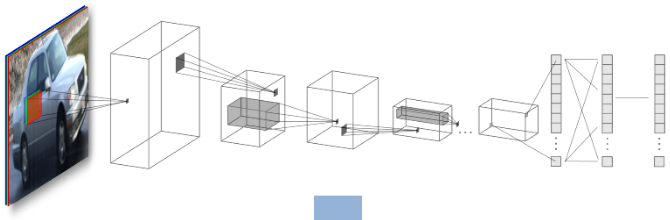
1. Change build-type to 'lib'

2. Select cross-compile toolchain

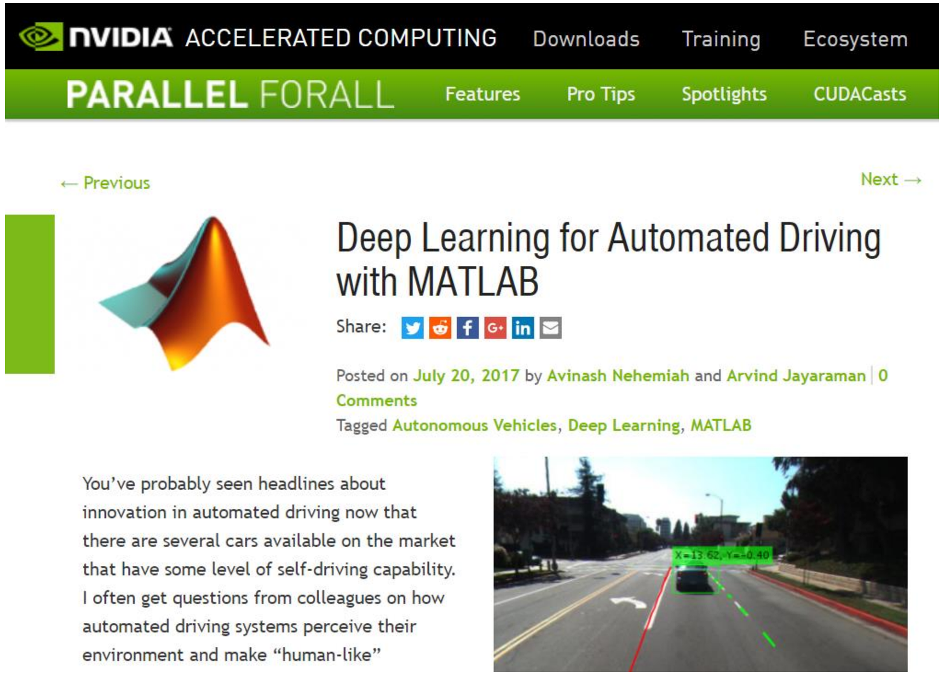


End-to-End Application: Lane Detection

Alexnet



Transfer Learning



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

Image → **Lane detection CNN**

Left lane coefficients →

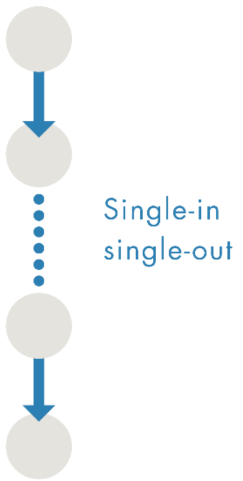
Right lane coefficients →

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



GPU Coder: R2017b

Networks:
MNist
Alexnet
YOLO
VGG
Lane detection
Pedestrian detection

DAGNetwork

GPU Coder: R2018a

Networks:
GoogLeNet
ResNet
SegNet
DeconvNet

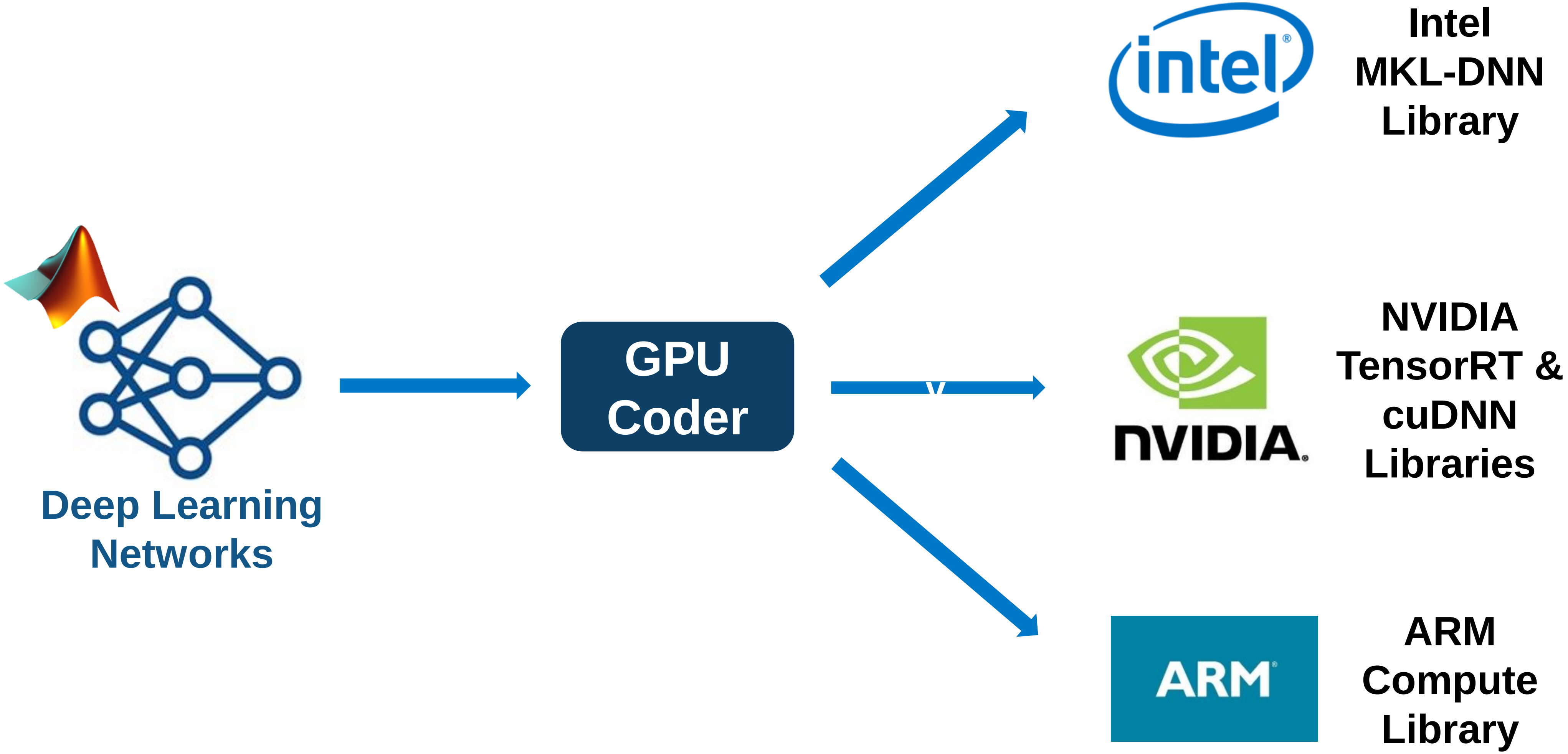
Object detection
Semantic segmentation

Semantic Segmentation

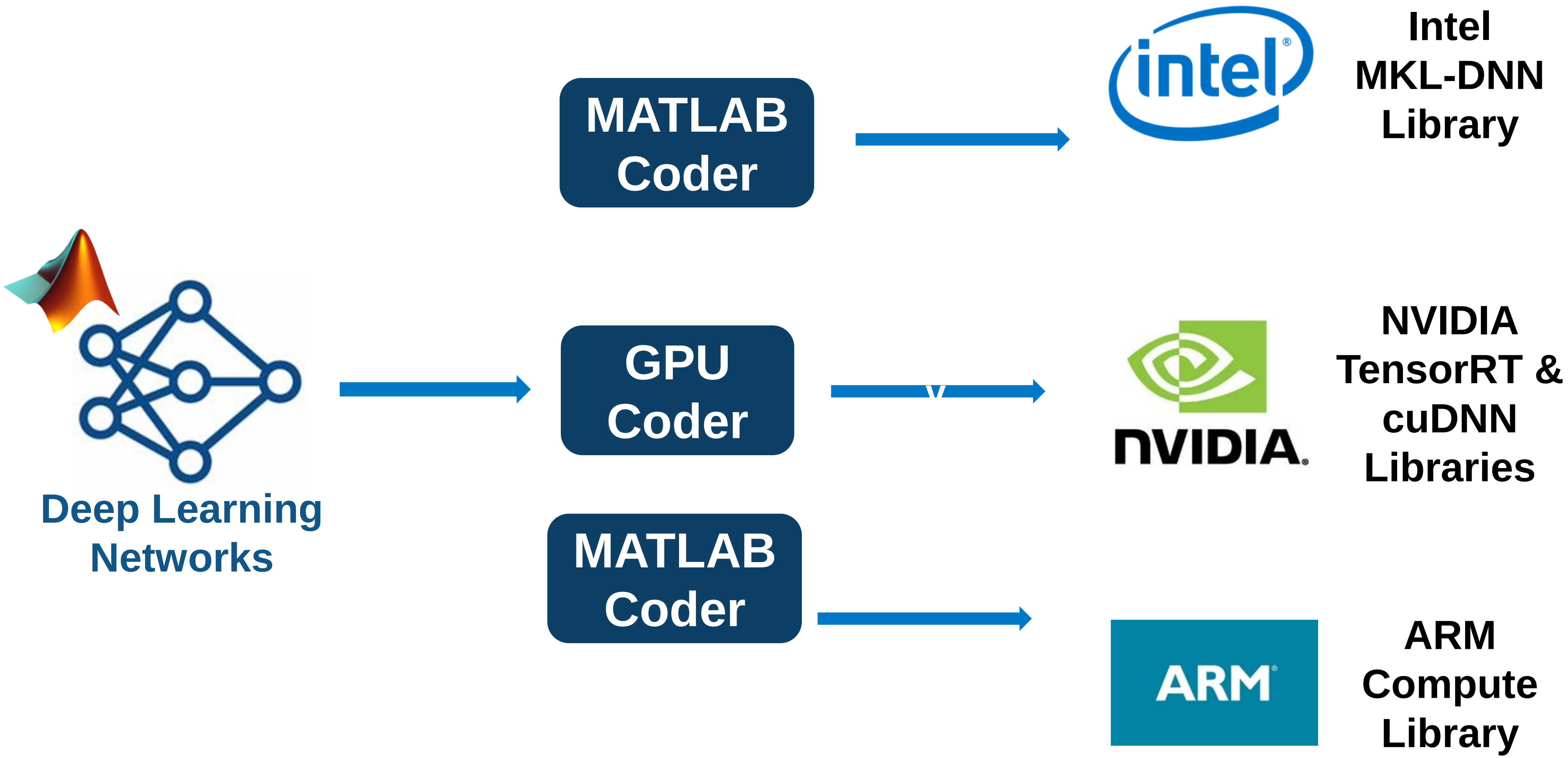
Running in MATLAB

Generated Code from GPU Coder

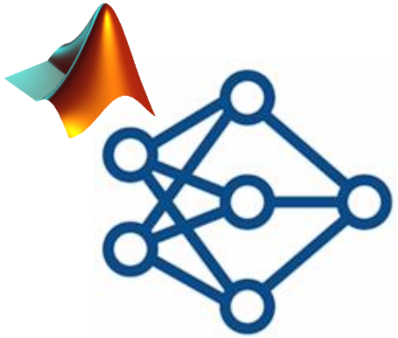
Deploying to CPUs



Deploying to CPUs



Deploying to CPUs



Deep Learning Networks

MATLAB Coder

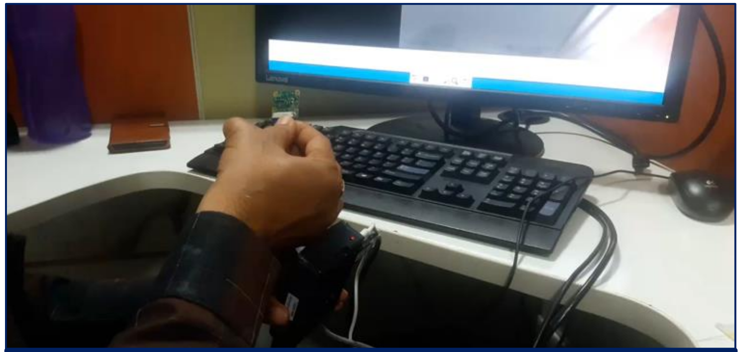
GPU Coder

MATLAB Coder

Desktop CPU

NVIDIA

NVIDIA TensorRT & cuDNN Libraries



Raspberry Pi board

How Good is Generated Code Performance

- Performance of image processing and computer vision

- Performance MATLAB / GPU Coder / TensorFlow / MXNet / PyTorch / etc.

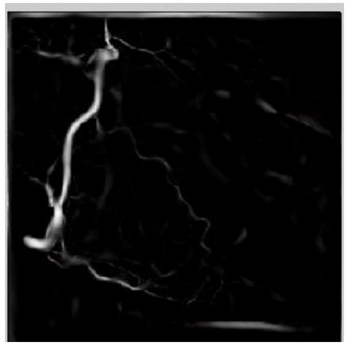
GPU Coder for Image Processing and Computer Vision

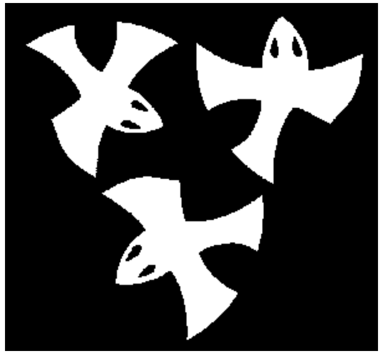
Fog removal

5x speedup

Frangi filter

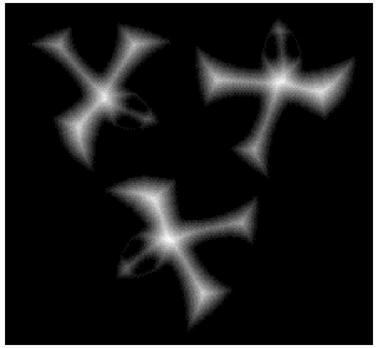
3x speedup





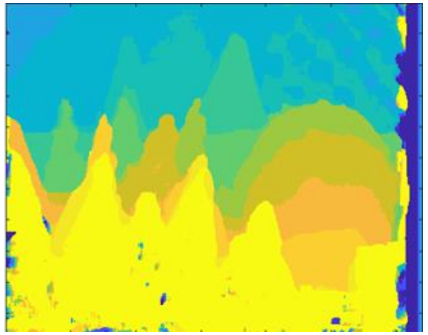
Distance transform

8x speedup



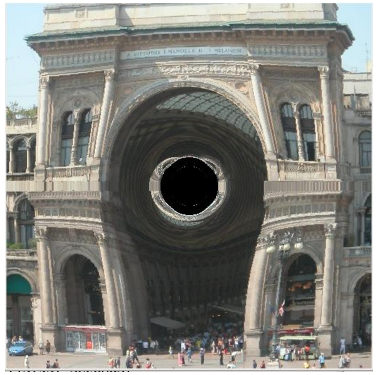
Stereo disparity

50x speedup



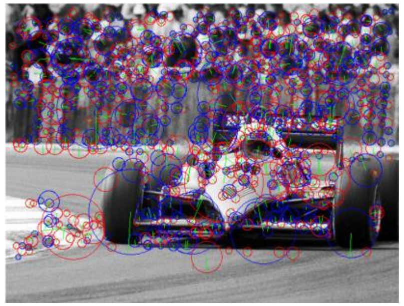
Ray tracing

18x speedup

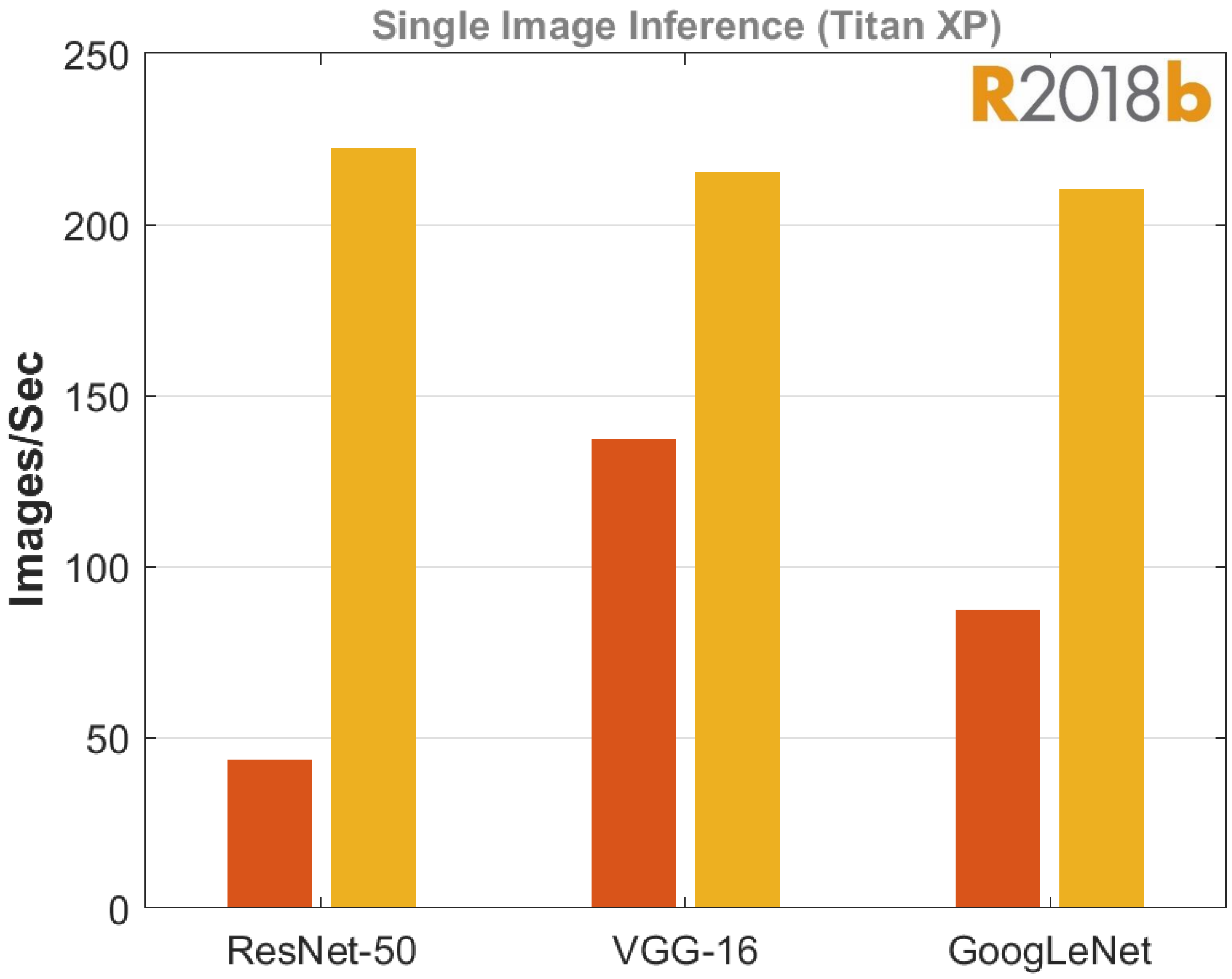


SURF feature extraction

700x speedup

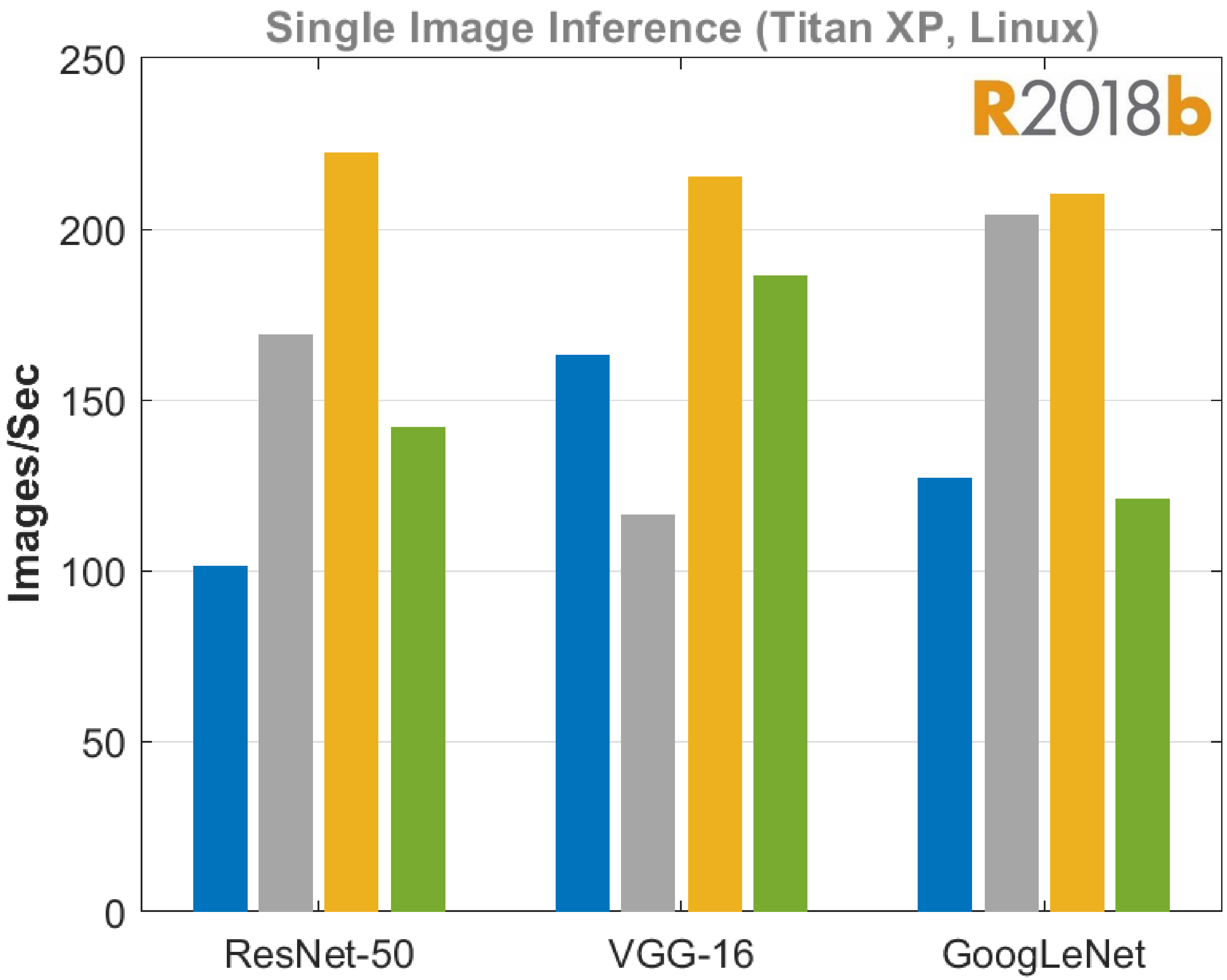


GPU Coder speeds up MATLAB about 2x



Intel® Xeon® CPU 3.6 GHz - NVIDIA libraries: CUDA9 - cuDNN 7

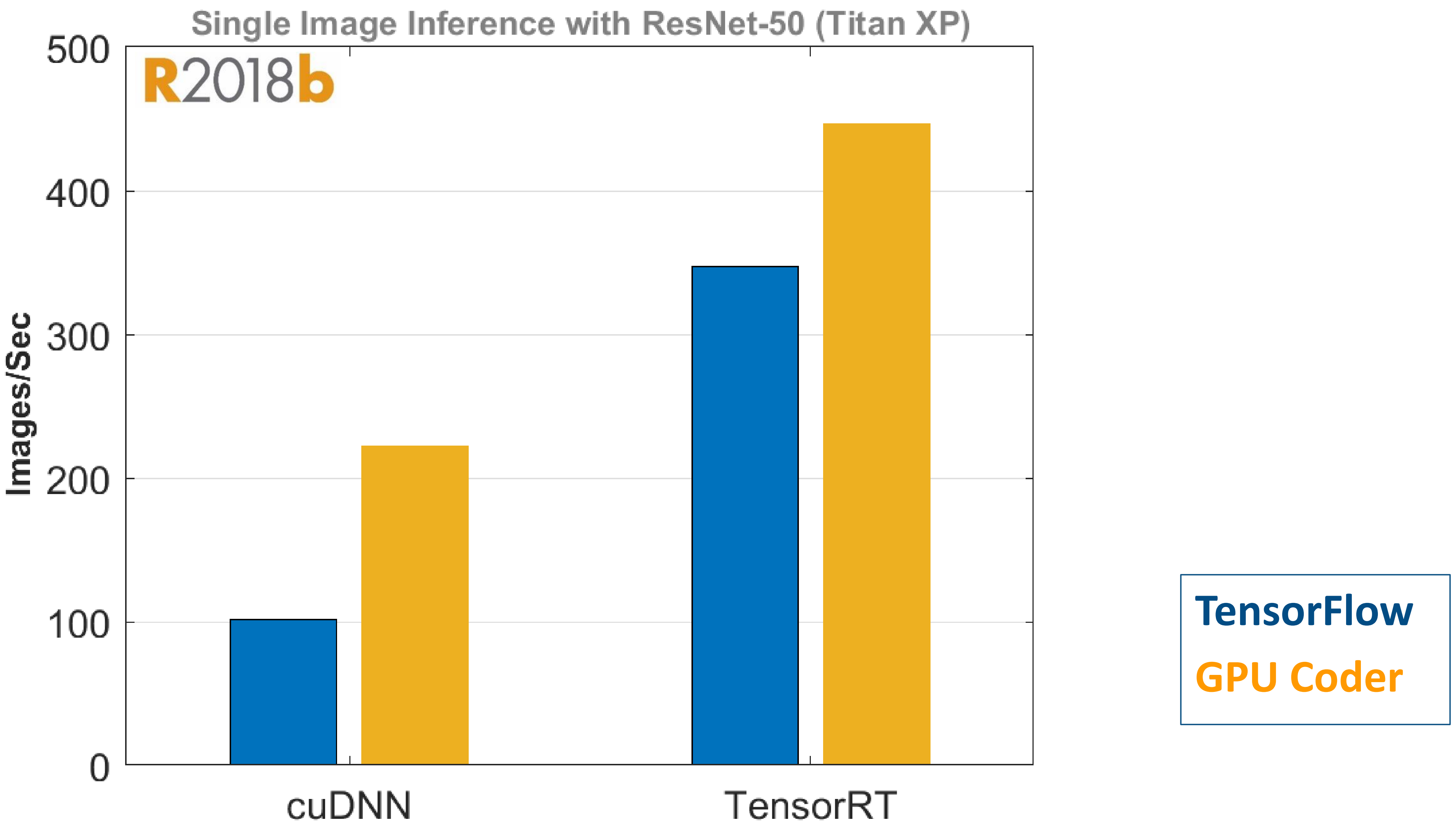
With GPU Coder, MATLAB is fast



Faster than TensorFlow, MXNet, and PyTorch

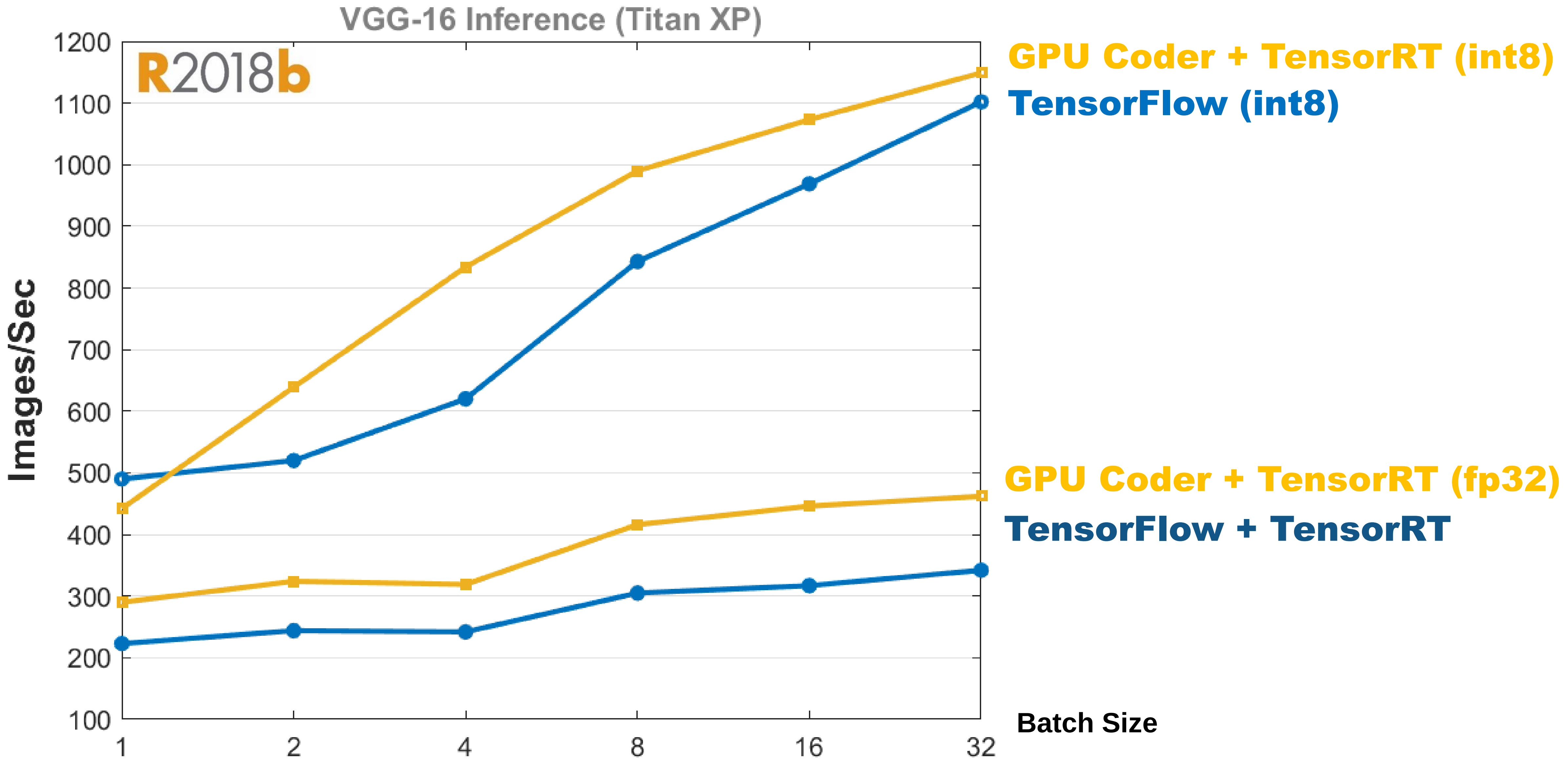
Intel® Xeon® CPU 3.6 GHz - NVIDIA libraries: CUDA9 - cuDNN 7 - Frameworks: TensorFlow 1.8.0, MXNet 1.2.1, PyTorch 0.3.1

TensorRT speeds up inference for TensorFlow and GPU Coder



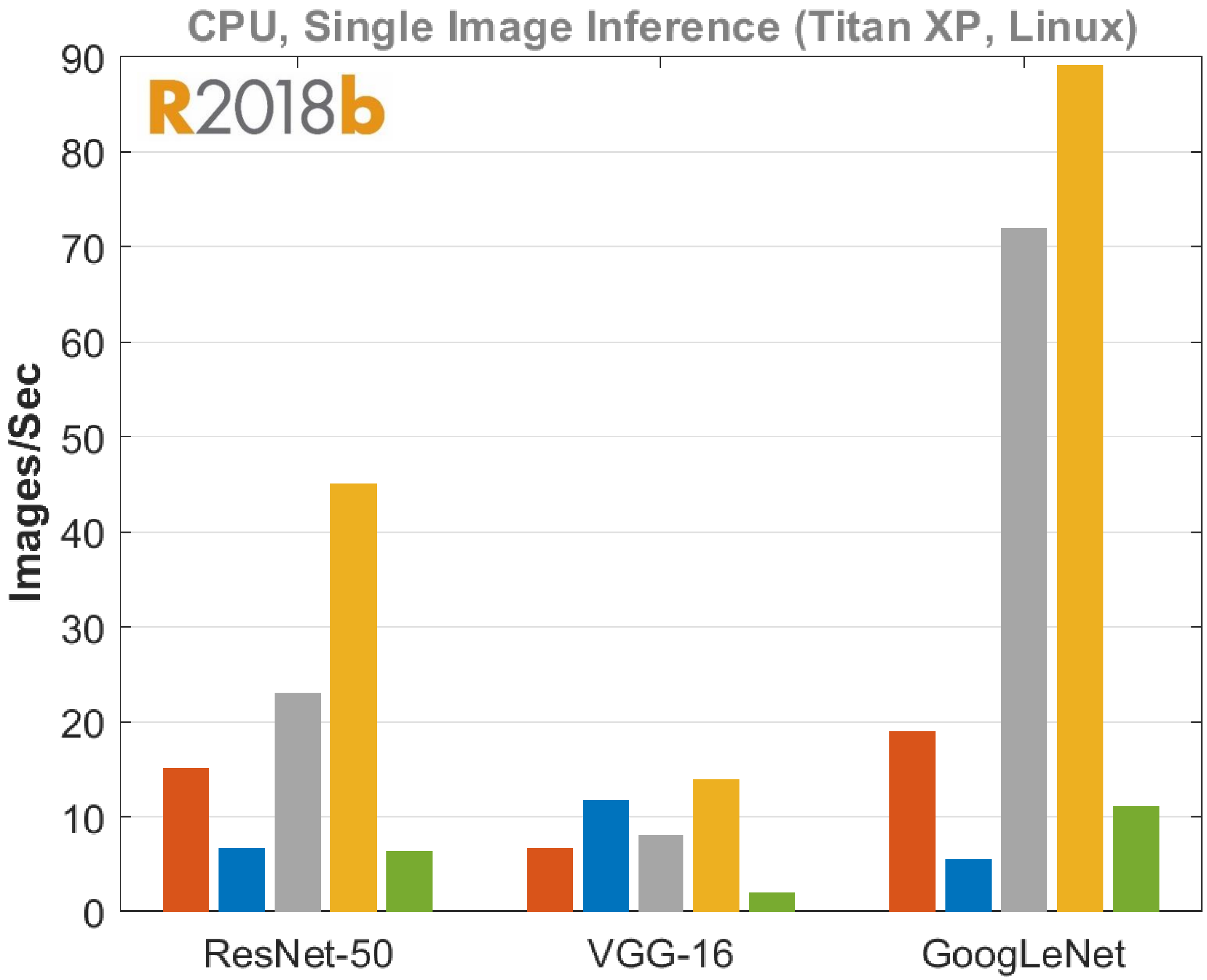
Intel® Xeon® CPU 3.6 GHz - NVIDIA libraries: CUDA9 - cuDNN 7 – TensorRT 3.0.4 - Frameworks: TensorFlow 1.8.0

Even higher Speeds with Integer Arithmetic (int8)



Intel® Xeon® CPU 3.6 GHz - NVIDIA libraries: CUDA9 - cuDNN 7 – TensorRT 3.0.4 - Frameworks: TensorFlow 1.8.0, MXNet 1.2.1, PyTorch 0.3.1

CPU Performance



GPU Coder also fastest on CPU

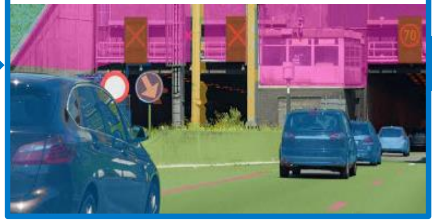
Only **MXNet** faster than **MATLAB**

Intel® Xeon® CPU 3.6 GHz - Frameworks: TensorFlow 1.6.0, MXNet 1.2.1, PyTorch 0.3.1

Design Your DNNs in MATLAB, Deploy with GPU Coder

Access Data

Design + Train



Deploy

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

- **Acceleration** with GPU's
- **Scale** to clusters

- **Automate compilation to GPUs and CPUs using GPU Coder**