MATLAB EXPO 2018

Deploying Deep Learning Networks to Embedded GPUs and CPUs

성 호 현 부장
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

Training Data
Example: Transfer Learning in MATLAB

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

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

% 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();

% train!
options = trainingOptions('sgdm', ... 
    'LearnRateSchedule', 'none', ...
    'InitialLearnRate', .0001, ...
    'MaxEpochs', 20, ...
    'MiniBatchSize', 128);
myConvnet = trainNetwork(imsd, layers, options);
```
Scaling Up Model Training Performance

Training on the AWS (EC2)

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

Multiple GPU support

MATLAB is more than 4x faster than TensorFlow.
Visualizing and Debugging Intermediate Results

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

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
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

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

Vectorized MATLAB

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

CUDA

```
cudaMalloc(&gpu_z, 8388608UL);
cudaMalloc(&gpu_x, 4149304UL);
cudaMalloc(&gpu_y, 4149304UL);
cudaMemcpy((void*)gpu_y, (void*)y, 4149304UL, cudaMemcpyHostToDevice);
cudaMemcpy((void*)gpu_x, (void*)x, 4149304UL, 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 * blockDim.x + blockDim.x - 1) / blockDim.x) + threadIdx.x);

    if (i < gridDim.x) {
        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

CUDA kernel for GPU parallelization

```c
static __global__ __launch_bounds__ (512, 1) void kernel3(real_T *z0, real_T *z, real_T *count, real_T *c)
{
    real_T z_im;
    real_T y(10000000);
    int32_T threadIdx;
    threadIdx = (int32_T)(blockIdx.x * blockDim.x + threadIdx.x);
    if ((threadIdx <= 10000000) || (threadIdx >= 10000000)) {
        z_im = z[threadIdx].re * z[threadIdx].im + z[threadIdx].im * z[threadIdx].re;
        z[threadIdx].re = (z[threadIdx].re) * z[threadIdx].im - z[threadIdx].im * z[threadIdx].re;
        z[threadIdx].im = z_im + y[threadIdx].im;
        if (z_im < 0) return;
        count[threadIdx] += (real_T)(y[threadIdx].re) < 2.0;
    }
}
```

CUDA

```c
... ...
```

Mandelbrot space

MATLAB EXPO 2016
Algorithm Design to Embedded Deployment Workflow

MATLAB algorithm
(functional reference)

MATLAB

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

CPU

GPU Coder

Build type

Call CUDA from MATLAB directly

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

Cross-compiled .lib

Desktop GPU

.mex

.lib

Desktop GPU

Embedded GPU

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)

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)

MATLAB 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

.mex

.Tesla GPU

.Tegra GPU

.C++

Cross-compiled on host with Linaro toolchain

Real-time test

Cross-compiled .lib

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

Build type

Call CUDA from MATLAB directly

Call CUDA from (C++) hand-coded main()
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: \( y = ax^2 + bx + c \)

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, Object detection, Semantic segmentation
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
Deploying to CPUs

Deep Learning Networks

GPU Coder

Desktop CPU

NVIDIA TensorRT & cuDNN Libraries

Raspberry Pi board
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

Frames per second

Batch Size

Testing platform

CPU: Intel(R) Xeon(R) CPU E5-1650 v4 @ 3.60GHz
GPU: Pascal Titan Xp
cuDNN: v7

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

Testing platform
- CPU: Intel(R) Xeon(R) CPU E5-1650 v4 @ 3.60GHz
- GPU: Pascal Titan Xp
cuDNN v7

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)**
Alexnet Inference on Jetson TX2: Frame-Rate Performance

TensorRT (2.1) 1.15x
MATLAB GPU Coder (R2017b)
2x
C++ Caffe (1.0.0-rc5)

Frames per second

Batch Size

To be updated with R2018a benchmarks soon
Contact Bill.Chou@mathworks.com for more information
Alexnet Inference on Jetson TX2: Memory Performance

- **C++ Caffe (1.0.0-rc5)**
- **MATLAB GPU Coder (R2017b)**
- **TensorRT 2.1**
  (using giexec wrapper)

To be updated with R2018a benchmarks soon

Contact Bill.Chou@mathworks.com for more information

MATLAB EXPU 2018
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.