Agenda

▪ What is CUDA code?
▪ What is GPU Coder?
▪ Why use GPU Coder?
▪ How to use GPU Coder?
▪ How fast is GPU Coder?
▪ Key takeaways
Algorithm Design to Embedded Deployment Workflow

MATLAB algorithm (functional reference) → ? → Embedded GPU

C++
GPUs and CUDA
What is CUDA code?

CUDA extends C/C++ code with constructs for parallel computing
What does CUDA code look like?

void foo(const real_T A[100000000], const real_T B[100000000],
          real_T C[100000000])
{
    real_T *gpu_B;
    real_T *gpu_A;
    real_T *gpu_C;
    cudaMalloc(&gpu_C, 800000000ULL);
    cudaMalloc(&gpu_A, 800000000ULL);
    cudaMalloc(&gpu_B, 800000000ULL);
    cudaMemcpy((void *)gpu_B, (void *)&B[0], 800000000ULL,
                cudaMemcpyHostToDevice);
    cudaMemcpy((void *)gpu_A, (void *)&A[0], 800000000ULL,
                cudaMemcpyHostToDevice);
    foo_kernel1<<<dim3(313U, 313U, 1U), dim3(32U, 32U,
           1U)>>>(gpu_B, gpu_A, gpu_C);
    cudaMemcpy((void *)&C[0], (void *)gpu_C, 800000000ULL,
                cudaMemcpyDeviceToHost);
    cudaFree(gpu_B);
    cudaFree(gpu_A);
    cudaFree(gpu_C);
}

static __global__ __launch_bounds__(1024, 1)
void foo_kernel1(const real_T *B,
                 const real_T *A, real_T *C)
{
    uint32_T threadId;
    int32_T i0;
    threadId = (uint32_T)mwGetGlobalThreadIndex();
    i0 = (int32_T)threadId;
    if (!(i0 >= 100000000)) {
        C[i0] = A[i0] * B[i0];
    }
}

function C = foo(A,B)
C = A*B;
Challenges for the CUDA programmer

▪ 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
What is GPU Coder?

- Generates **CUDA** code for NVIDIA GPUs

- Also generates code for Deep Neural Networks for Intel CPUs and ARM Cortex-A platforms.
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;
```

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, real_T *z) {
    int32_T i;
    i = blockIdx.x*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
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 z(1000000);
    int32_T threadIdx;
    threadIdx = (int32_T)(blockDim.x * blockIdx.x + threadIdx.x);
    if ((threadIdx.x >= 1000000)) {
        z_im = z[threadIdx.x.re * z[threadIdx.x.im + z[threadIdx.x.im] * z[threadIdx.x.im]].re - z[threadIdx.x.im].re * z[threadIdx.x.im].re - z[threadIdx.x.im].im * z[threadIdx.x.im].im);
    } else {
        z_im = z_im + z[0][threadIdx.x.im].im;
        y[threadIdx.x] = hypot(z[threadIdx.x].re, z[threadIdx.x].im);
        count[threadIdx.x] += (real_T)y[threadIdx.x] * (x, y);
    }
}
```

CUDA

```c
... ...
```
Algorithm Design to Embedded Deployment Workflow

MATLAB algorithm (functional reference)

MATLAB algorithm

Build type

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

GPU Coder

Call CUDA from MATLAB directly

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

Cross-compiled .lib

Embedded GPU

Desktop GPU

Desktop GPU

C++

C++

C++

C++

Call CUDA from MATLAB directly

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

**Functional test**
- Test in MATLAB on host
- Call CUDA from MATLAB directly

**Deployment unit-test**
- Test generated code in MATLAB on host + GPU
- Call CUDA from (C++) hand-coded main()

**Deployment integration-test**
- Test generated code within C/C++ app on host + GPU
- CallCUDA from (C++) hand-coded main(). Cross-compiled on host with Linaro toolchain

**Real-time test**
- Test generated code within C/C++ app on Tegra target

Build type
- .mex
- .lib

Cross-compiled .lib

MATLAB Expo 2018
Alexnet Deployment to Tegra: Cross-Compiled with ‘lib’

Two small changes
1. Change build-type to ‘lib’
2. Select cross-compile toolchain

<table>
<thead>
<tr>
<th>Build type:</th>
<th>Static Library</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output file name:</td>
<td>alexnet_predict</td>
</tr>
<tr>
<td>Language:</td>
<td>C, C++</td>
</tr>
<tr>
<td>Hardware Board:</td>
<td>MATLAB Host Computer</td>
</tr>
<tr>
<td>Device:</td>
<td>Generic, MATLAB Host Computer</td>
</tr>
<tr>
<td>Toolchain:</td>
<td>Automatically locate an installed toolchain</td>
</tr>
<tr>
<td>NVIDIA CUDA</td>
<td>gmake (64-bit Linux)</td>
</tr>
<tr>
<td>NVIDIA CUDA for Jetson Tegra K1 v6.5</td>
<td>gmake (64-bit Linux)</td>
</tr>
<tr>
<td>NVIDIA CUDA for Jetson Tegra X1 v7.0</td>
<td>gmake (64-bit Linux)</td>
</tr>
<tr>
<td>NVIDIA CUDA for Jetson Tegra X2 v8.0</td>
<td>gmake (64-bit Linux)</td>
</tr>
</tbody>
</table>
End-to-End Application: Lane Detection

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

GPU coder generates code for whole application
GPU Coder for Deployment

Accelerated implementation of parallel algorithms on GPUs & CPUs

Deep Neural Networks
Deep Learning, machine learning

5x faster than TensorFlow
2x faster than MXNet

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

Deep Neural Networks
Deep Learning, machine learning

5x faster than TensorFlow
2x faster than MXNet

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
Deep Learning Network Support (with Neural Network Toolbox)

SeriesNetwork

Networks:
- MNist
- Alexnet
- YOLO
- VGG
- Lane detection
- Pedestrian detection

GPU Coder: R2017b

DAGNetwork

Networks:
- GoogLeNet
- ResNet
- SegNet
- DeconvNet
- Object detection
- Semantic segmentation

GPU Coder: R2018a
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
- Ray tracing: 18x speedup
- Frangi filter: 3x speedup
- Stereo disparity: 50x speedup
- SURF feature extraction: 700x speedup
Alexnet Inference on NVIDIA Titan Xp

Frames per second vs Batch Size

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

Software:
- TensorFlow (1.6.0)
- MXNet (1.1.0)
- GPU Coder + cuDNN
- GPU Coder + TensorRT (3.0.1)
- GPU Coder + TensorRT (3.0.1, int8)
VGG-16 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)
Alexnet Inference on Jetson TX2: Frame-Rate Performance

Frames per second vs. Batch Size

- **TensorRT (2.1)**: 1.15x improvement
- **MATLAB GPU Coder (R2017b)**: 2x improvement
- **C++ Caffe (1.0.0-rc5)**
Alexnet Inference on Jetson TX2: Memory Performance

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

![Graph showing peak memory usage vs batch size for different inference methods.](image-url)
Key Takeaways

- GPU Coder automates the process of writing CUDA code for general algorithms – not only Deep Learning

- GPU Coder generates code for DNN for multiple platforms

- GPU Coder performs in most times better than other common Deep Learning platforms