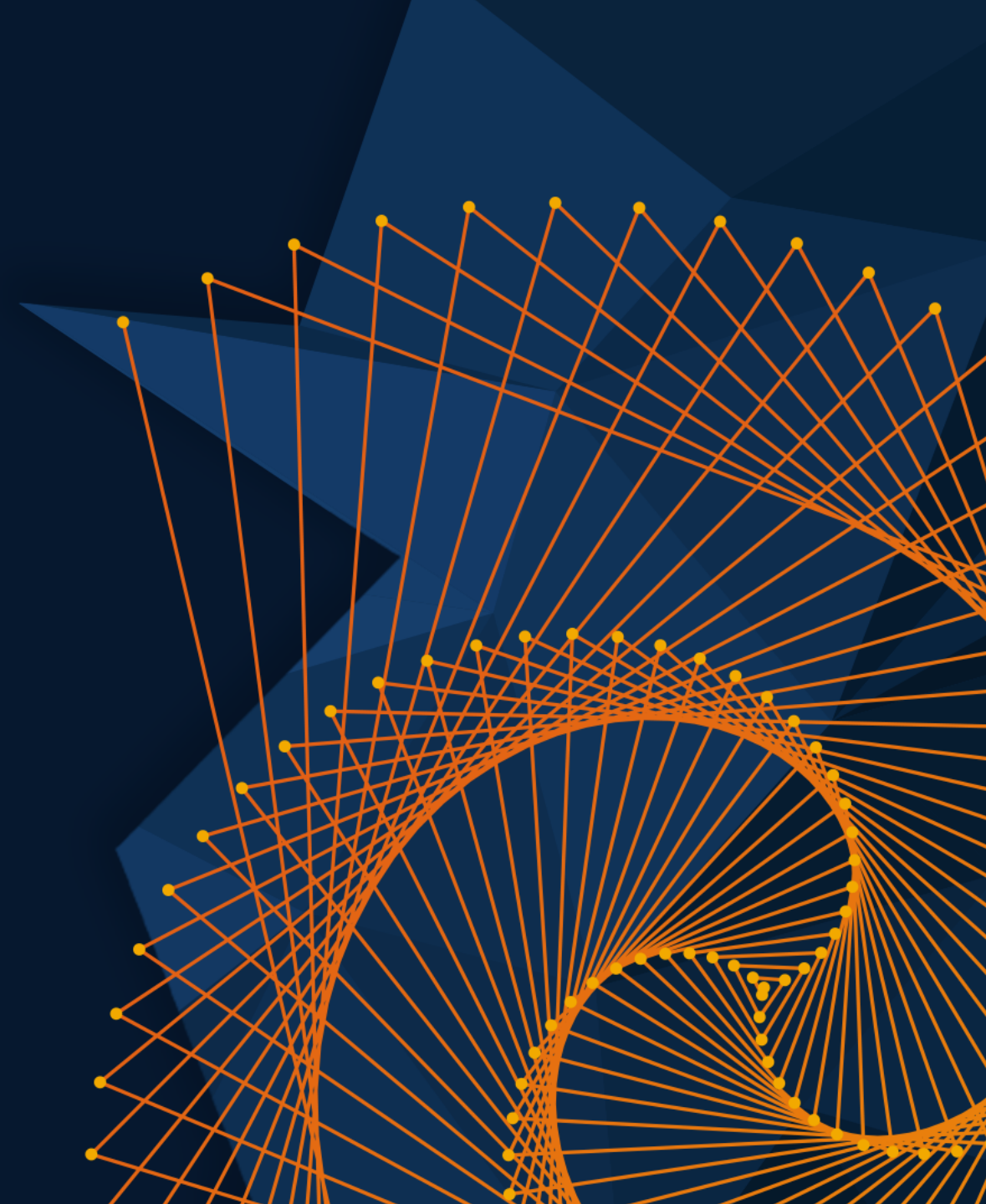


MATLAB EXPO

2024.06.11 | 그랜드 인터컨티넨탈 서울 파르나스

GPU기반 AI 어플리케이션 개발하기

신행재, 매스웍스코리아



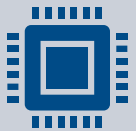
Key Takeaways



GPU Coder generates CUDA code from MATLAB & Simulink



Accelerate MATLAB & Simulink simulations



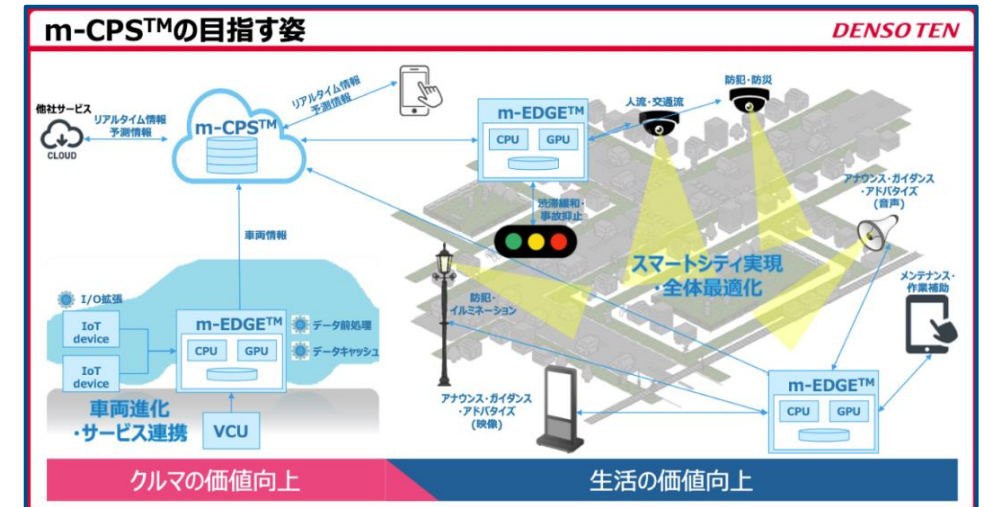
Deploy algorithms (signal/deep learning,...) to embedded GPUs

DENSO TEN Uses MATLAB to Develop Mobile Cyber-Physical System

DENSO TEN is developing mobility solutions using a cyber-physical system. The system consists of edge computers on vehicles and cloud AI. Mobility data collected from VCU and edge computers will be analyzed in the cloud to derive solutions for mobility problems.

Key Outcomes/Results

- DENSO TEN used MATLAB to develop and deploy a production-ready system that spanned hardware and cloud applications without manual recoding
- MATLAB enabled DENSO TEN to easily implement complex algorithms that utilized AI, image processing, probability calculations, and statistics
- MATLAB Production Server enabled DENSO TEN to run sophisticated analytics in a centralized location on the cloud



Mobility solution powered by edge devices and a centralized cloud to analyze and predict traffic flow.

"The superiority of MATLAB in data processing and visualization and its ability to consistently realize the entire process from conception to implementation is a major attraction and is the reason why we chose MATLAB for this project."

- Natsuki Yokoyama, DENSO TEN

Drass Develops Deep Learning System for Real-Time Object Detection in Maritime Environments

Challenge

Help ship operators monitor sea environments and detect objects, obstacles, and other ships

Solution

Create an object-detection deep learning model that can be deployed on ships and run in real time

Results

- Data labeling automated
- Development time reduced
- Flexible and reproducible framework established

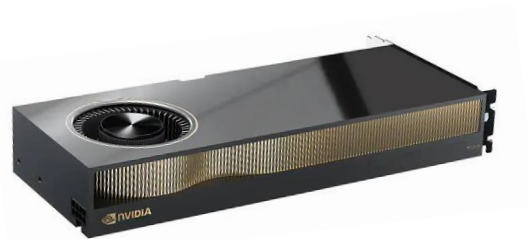
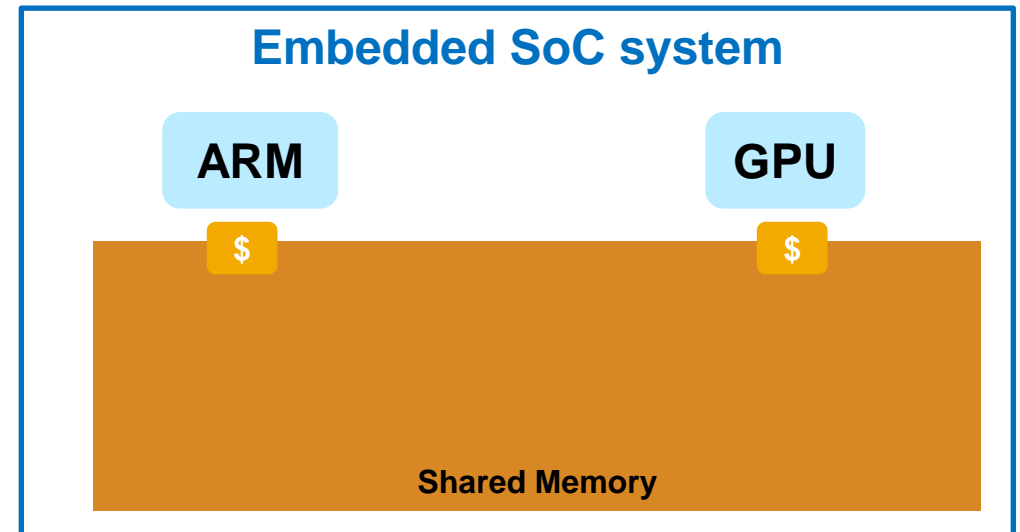
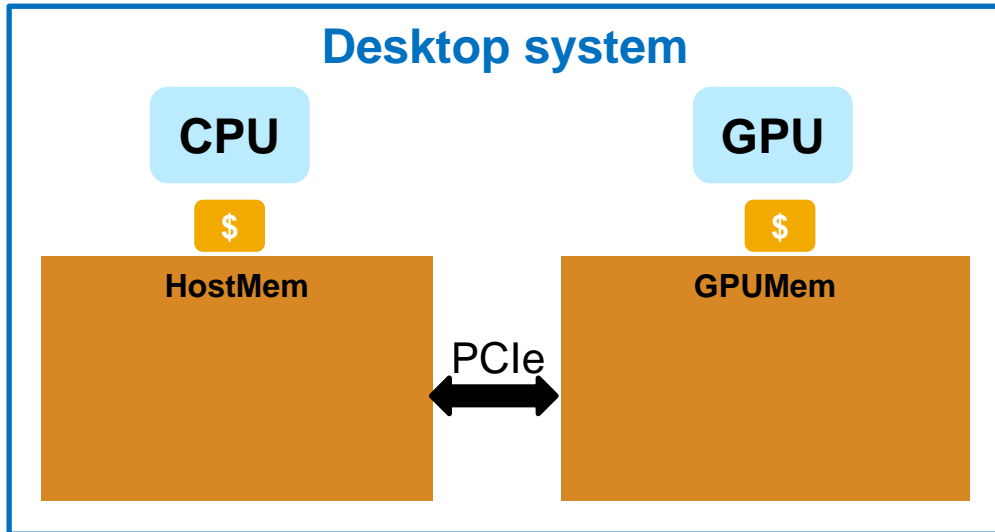


First day of object detection tests with optronic system prototype.

“From data annotation to choosing, training, testing, and fine-tuning our deep learning model, MATLAB had all the tools we needed—and GPU Coder enabled us to rapidly deploy to our NVIDIA GPUs even though we had limited GPU experience.”

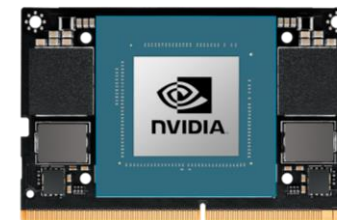
- Valerio Imbriolo, Drass Group

Types of GPUs



Desktop GPUs
(and Cloud GPUs)

Power Consumption:
300W~ vs 15~75W

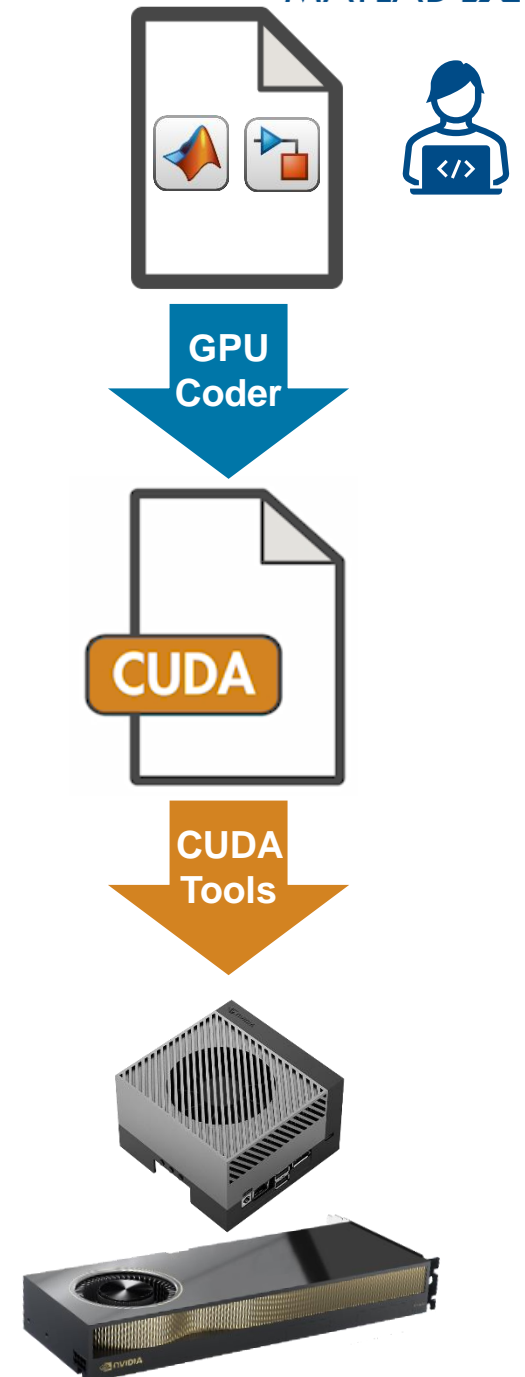


Embedded GPUs

Industrial module: [10-year Operating Lifetime](#)

CUDA code generation

- Generate optimized CUDA code from MATLAB and Simulink for deep learning, embedded vision, and autonomous systems
- Generated CUDA is portable across NVIDIA desktop GPUs
- Prototype algorithms on modern GPUs including the Nvidia Data Center GPUs and Jetson AGX Orin
- Accelerate computationally intensive portions of your MATLAB code and Simulink models using generated CUDA code

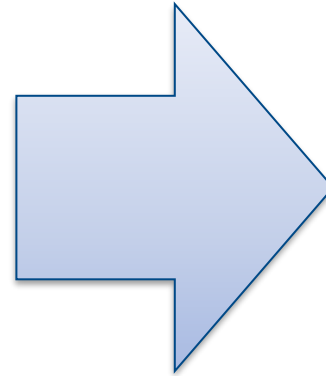




Why Use CUDA code generation ?

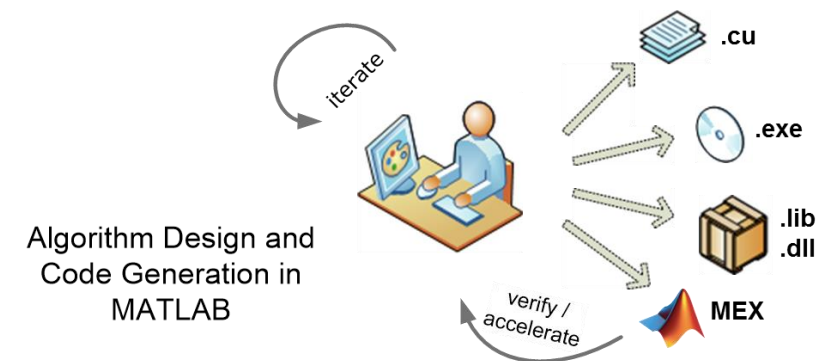
Pains: Hand code

- **Cannot code in CUDA**
- Time consuming
- Manual Coding Errors
- Multiple implementations
- Expensive



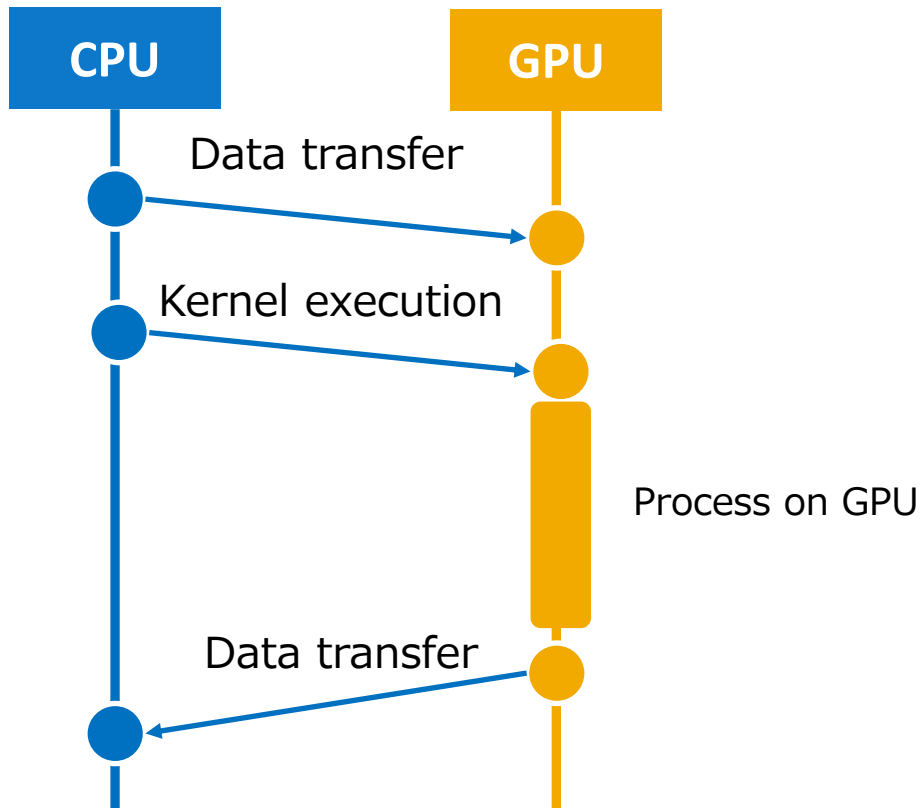
Solution: GPU Coder

- Automatically convert to CUDA
- Get to CUDA faster
- Eliminate manual coding errors
- Maintain Single “Truth”
- Stay within MATLAB/Simulink at a higher level



What is CUDA?

Run Hello World on GPU



```

__global__ void helloFromGPU()
{
    printf("Hello World from GPU!\n");
}

int main(int argc, char **argv)
{
    printf("Hello World from CPU!\n");
    helloFromGPU<<<1, 10>>>();
    return 0;
}

```

Microsoft Visual Studio Debug Console

```

Hello World from CPU!
Hello World From GPU!
Hello World From GPU!
Hello World From GPU!
Hello World From GPU!
Hello World From GPU!
Hello World From GPU!
Hello World From GPU!
Hello World From GPU!
Hello World From GPU!

```

A red bracket on the left side of the console output groups the ten "Hello World From GPU!" lines, with an arrow pointing to the `helloFromGPU` call in the code above.

- Kernel call(special syntax)

```
kernelFunc<<<Block_dim, Thread_dim>>>(a, b, c);
```

For example, if you could do this ...

Linear Algebra routine, SAXPY example

Scalarized MATLAB

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

Vectorized MATLAB

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

GPU Coder



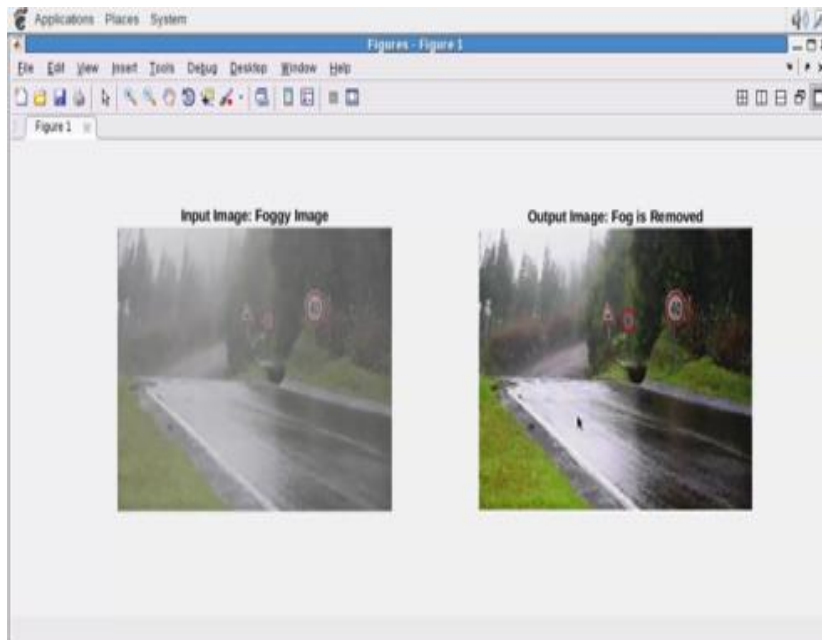
```
static __global__ __launch_bounds__(512, 1) void saxpy_kernel1(const real32_T *y,
    const real32_T *x, real32_T a, real_T *z)
{
    int i = blockIdx.x*blockDim.x + threadIdx.x;
    if (!(i >= 1048576)) {
        z[i] = (real_T)(a * x[i] + y[i]);
    }
}

void saxpy(real32_T a, const real32_T x[1048576], const real32_T y[1048576],
    real_T z[1048576])
{
    real32_T *gpu_y;
    real32_T *gpu_x;
    real_T *gpu_z;
    cudaMalloc(&gpu_z, 8388608UL);
    cudaMalloc(&gpu_x, 4194304UL);
    cudaMalloc(&gpu_y, 4194304UL);
    cudaMemcpy((void *)gpu_y, (void *)&y[0], 4194304UL, cudaMemcpyHostToDevice);
    cudaMemcpy((void *)gpu_x, (void *)&x[0], 4194304UL, cudaMemcpyHostToDevice);
    saxpy_kernel1<<<dim3(2048U, 1U, 1U), dim3(512U, 1U, 1U)>>>(gpu_y, gpu_x, a,
        gpu_z);
    cudaMemcpy((void *)&z[0], (void *)gpu_z, 8388608UL, cudaMemcpyDeviceToHost);
    cudaFree(gpu_y);
    cudaFree(gpu_x);
    cudaFree(gpu_z);
}
```

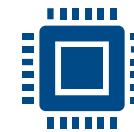
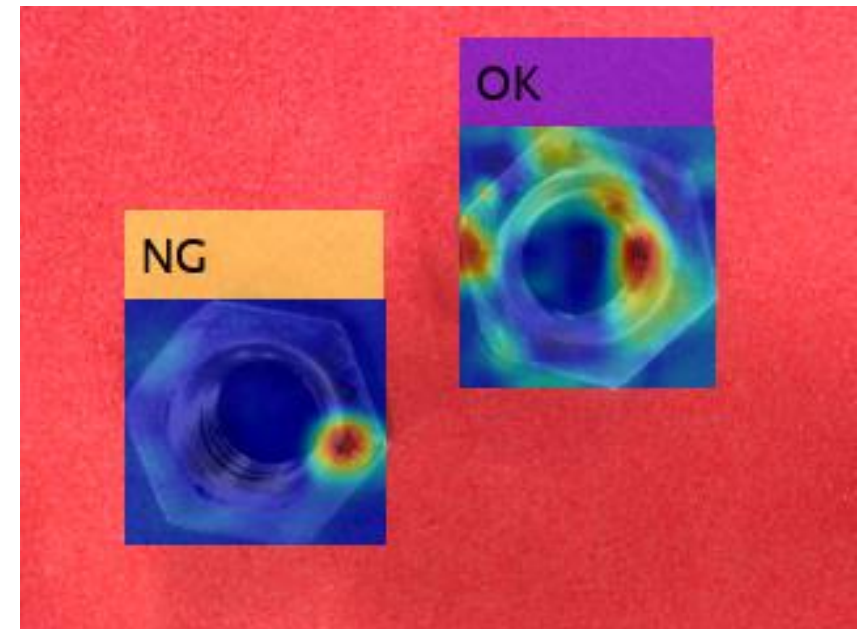
Automatic compilation from a highly extensible language to a high performance language

Two Application examples

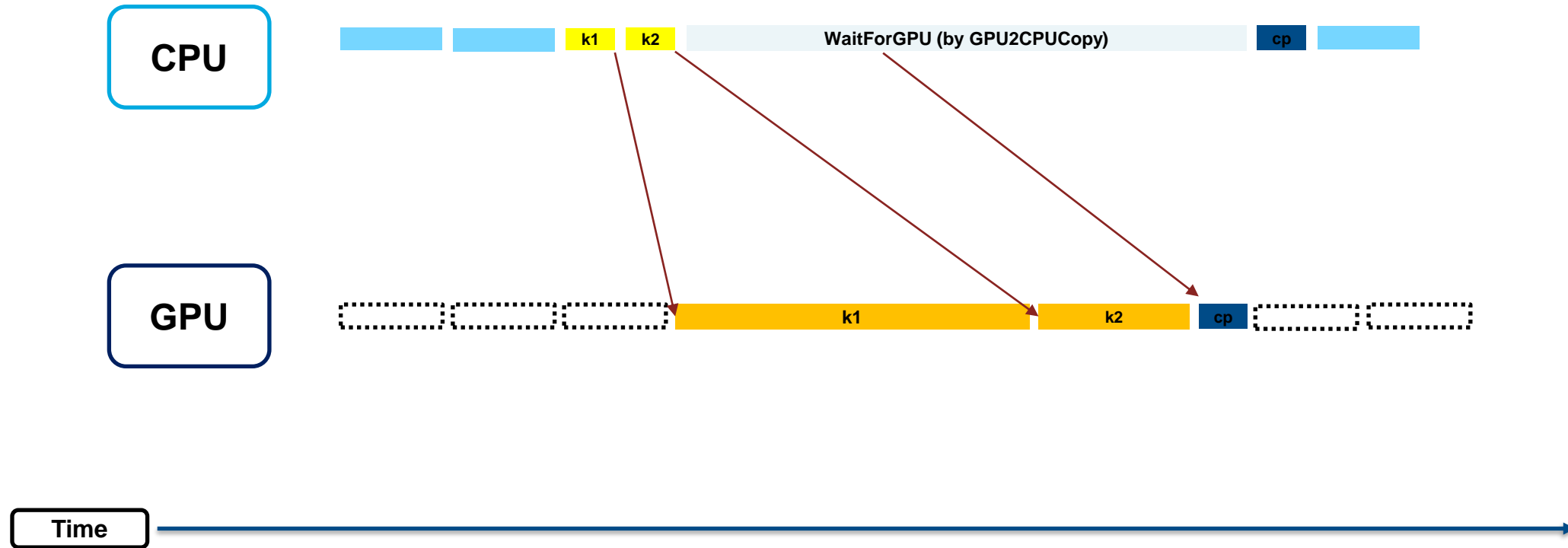
1) Fog Rectification



2) Anomaly Detection



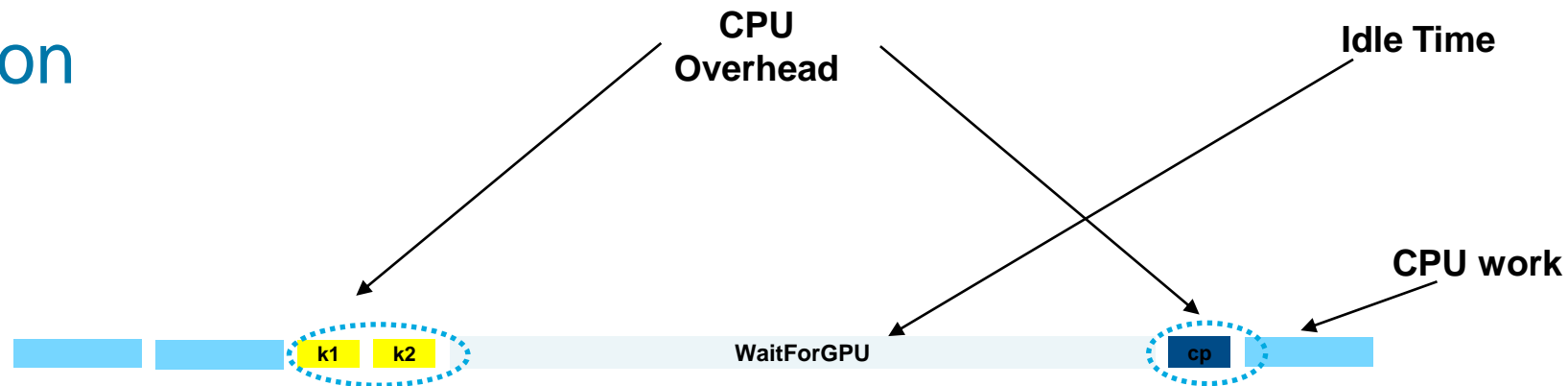
Analyze CPU/GPU interaction for performance improvements



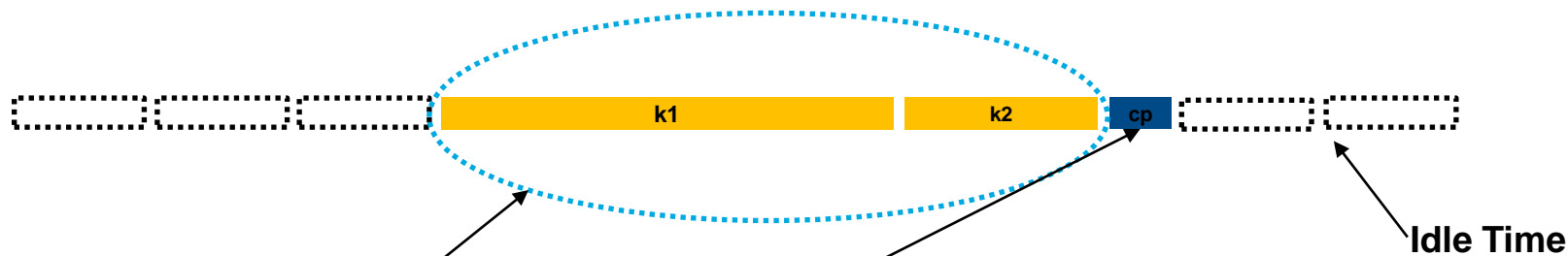
CPU/GPU Action



CPU



GPU



GPU Utilization

Kernel Time

Copy Time

Example 1: Fog rectification + GPU Performance Analyzer



MATLAB R2023a

HOME PLOTS APPS LIVE EDITOR INSERT VIEW

Code Control Task Section Break Text Table of Contents Code Example Image Hyperlink Equation

CODE SECTION TEXT IMAGE LINK EQUATION

C:\Users\cstockha\OneDrive - MathWorks\demos\conferences\MAC\2023\fog_rectification\forRecording

Live Editor - C:\Users\cstockha\OneDrive - MathWorks\demos\conferences\MAC\2023\fog_rectification\forRecording\GPUExecutionProfilingOfTheGeneratedCodeExample.mlx

GPUExecutionProfilingOfTheGeneratedCodeExample.mlx fog_rectification.m

Analyze Performance of the Generated CUDA Code

This example shows you how to analyze and optimize the performance of the generated CUDA® code by using the `gpuPerformanceAnalyzer` function.

The GPU Coder Performance Analyzer runs a software-in-the-loop (SIL) execution that collects metrics on CPU/GPU activities in the generated code and provides a chronological timeline plot to visualize, identify, and mitigate performance bottlenecks in the generated CUDA code. This example generates the performance analysis report for the *Fog Rectification* example from GPU Coder. For more information, see [Fog Rectification](#).

Fog Rectification Algorithm

To improve the foggy input image, the algorithm performs fog removal and then contrast enhancement. The diagram shows the steps of both these operations.

```

graph LR
    subgraph Fog_Removal [Fog Removal]
        A[Foggy input image] --> B[Dark Channel Estimation]
        B --> C[Air light map estimation]
        C --> D[Air light map refinement]
        D --> E[Restoration]
    end
    E --> F[RGB to Gray]
    subgraph Contrast_Enhancement [Contrast Enhancement]
        F --> G[Histogram calculation]
        G --> H[Histogram normalization]
        H --> I[CDF calculation]
        I --> J[Contrast stretching]
    end
    J --> K[Fog rectified output image]
  
```

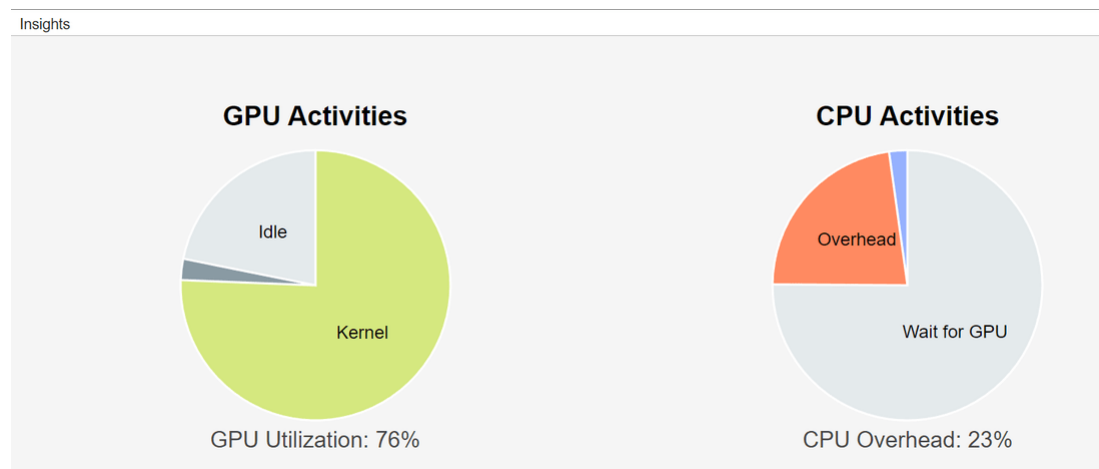
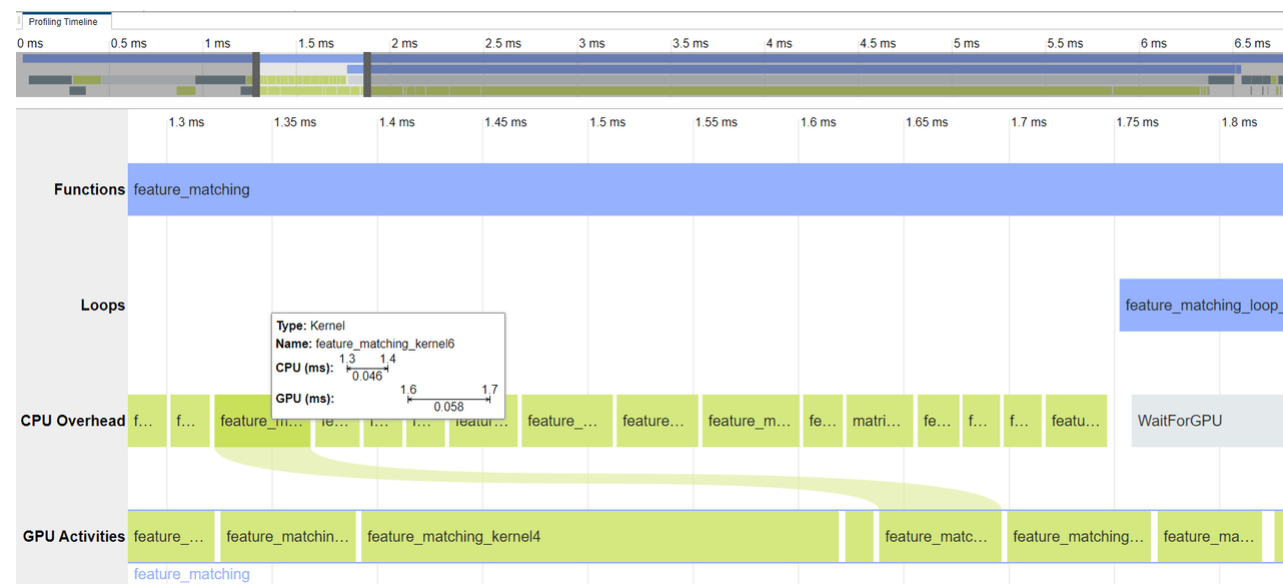
Zoom: 10... UTF-8 LF script



Code Profiling using GPU Performance Analyzer

Visualize code metrics and identify optimization and tuning opportunities

- Profile and understand GPU and CPU activities, events, and performance metrics in a chronological timeline plot
- Use the profiling info to analyze and optimize the performance of the generated CUDA



Event Statistics	
Type	Kernel
Name	feature_matching_kernel2
Start time	1.209344 ms
End time	1.253440 ms
Duration	0.044096 ms
Launch Params	
Grid size	[261, 1, 1]
Block size	[512, 1, 1]
Total threads	133.63 K
Shared memory	0 Byte
Registers per thread	16

Bidirectional Traceability



Understand how GPU Coder maps the MATLAB algorithm to CUDA kernels

The screenshot displays the MATLAB GPU Coder interface with two panes. The left pane shows the MATLAB source code for a function named `fog_rectification`. The right pane shows the corresponding CUDA code generated by the GPU Coder. A blue arrow points from the MATLAB code to the CUDA code, and a red arrow points from the CUDA code back to the MATLAB code, illustrating bidirectional traceability. A yellow highlight in the MATLAB code (lines 25-27) is connected by a blue arrow to a corresponding section of the CUDA code (lines 763-771).

```

1 function [out] = fog_rectification(input) %#codegen
2
3 % Copyright 2017-2019 The MathWorks, Inc.
4
5 coder.gpu.kernelfun;
6
7 % restoreOut is used to store the output of restoration
8 restoreOut = zeros(size(input),'double');
9
10 % Changing the precision level of input image to double
11 input = double(input)./255;
12
13 %% Dark channel Estimation from input
14 darkChannel = min(input,[],3);
15
16 % diff_im is used as input and output variable for anis
17 diff_im = 0.9*darkChannel;
18 num_iter = 3;
19
20 % 2D convolution mask for Anisotropic diffusion
21 hN = [0.0625 0.1250 0.0625; 0.1250 0.2500 0.1250; 0.062
22 hN = double(hN);
23
24 %% Refine dark channel using Anisotropic diffusion.
25 for t = 1:num_iter
26     diff_im = conv2(diff_im,hN,'same');
27 end
28
29 %% Reduction with min
30 diff_im = min(darkChannel,diff_im);
31
32 diff_im = 0.6*diff_im ;
33
34

```

```

753 // Changing the precision level of input image to d
754 cudaMemcpy(*gpu_input, input, 921600ULL, cudaMemcpyH
755 fog_rectification_kernel1<<<dim3(1800U, 1U, 1U), dim
756 *gpu_input, *b_gpu_input, *gpu_restoreOut);
757 // Dark channel Estimation from input
758 // diff_im is used as input and output variable for
759 fog_rectification_kernel2<<<dim3(600U, 1U, 1U), dim3
760 *b_gpu_input, *gpu_diff_im, *gpu_darkChannel);
761 // 2D convolution mask for Anisotropic diffusion
762 // Refine dark channel using Anisotropic diffusion.
763 for (idx = 0; idx < 3; idx++) {
764     fog_rectification_kernel3<<<dim3(605U, 1U, 1U), di
765     *gpu_expanded);
766     fog_rectification_kernel4<<<dim3(600U, 1U, 1U), di
767     *gpu_diff_im, *gpu_expanded);
768     cudaMemcpyToSymbol(const_b, b, 720ULL, 0ULL, cudaMe
769     fog_rectification_kernel5<<<dim3(15U, 20U, 1U), di
770     *gpu_expanded, *gpu_diff_im);
771 }
772 // Reduction with min
773 // Parallel element-wise math to compute
774 // Restoration with inverse Koschmieder's law
775 fog_rectification_kernel6<<<dim3(600U, 1U, 1U), dim3
776 *gpu_diff_im, *gpu_darkChannel);
777 fog_rectification_kernel7<<<dim3(600U, 1U, 1U), dim3
778 *b_gpu_input, *gpu_darkChannel, *gpu_diff_im, *g
779 fog_rectification_kernel8<<<dim3(1800U, 1U, 1U), dim
780 *gpu_restoreOut, *b_gpu_restoreOut);
781 //
782 // Stretching performs the histogram stretching of
783 // im is the input color image and p is cdf limit.
784 // out is the contrast stretched image and cdf is t
785 // density function and T is the stretching functio
786

```

GPU Coder for Image Processing and Computer Vision



Fog removal



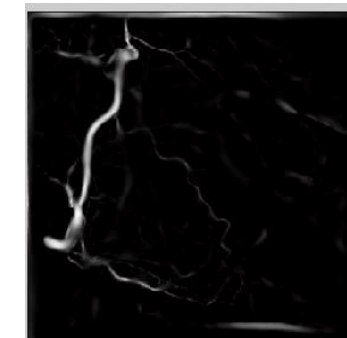
4x speedup



Frangi filter



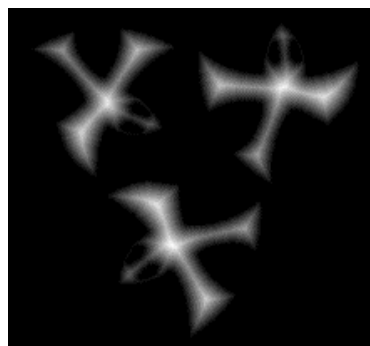
3x speedup



Distance transform



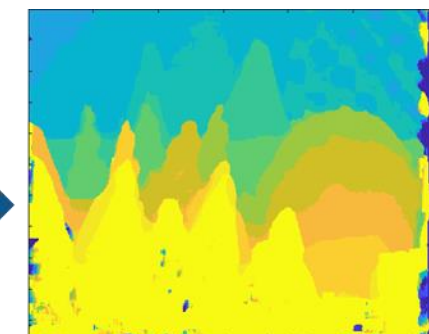
8x speedup



Stereo disparity



50x speedup



Ray tracing



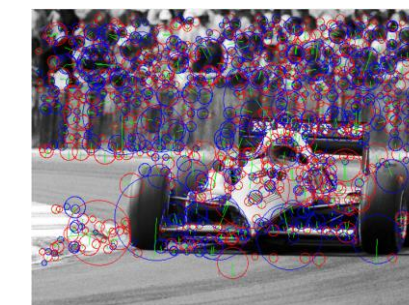
18x speedup



SURF feature extraction



700x speedup

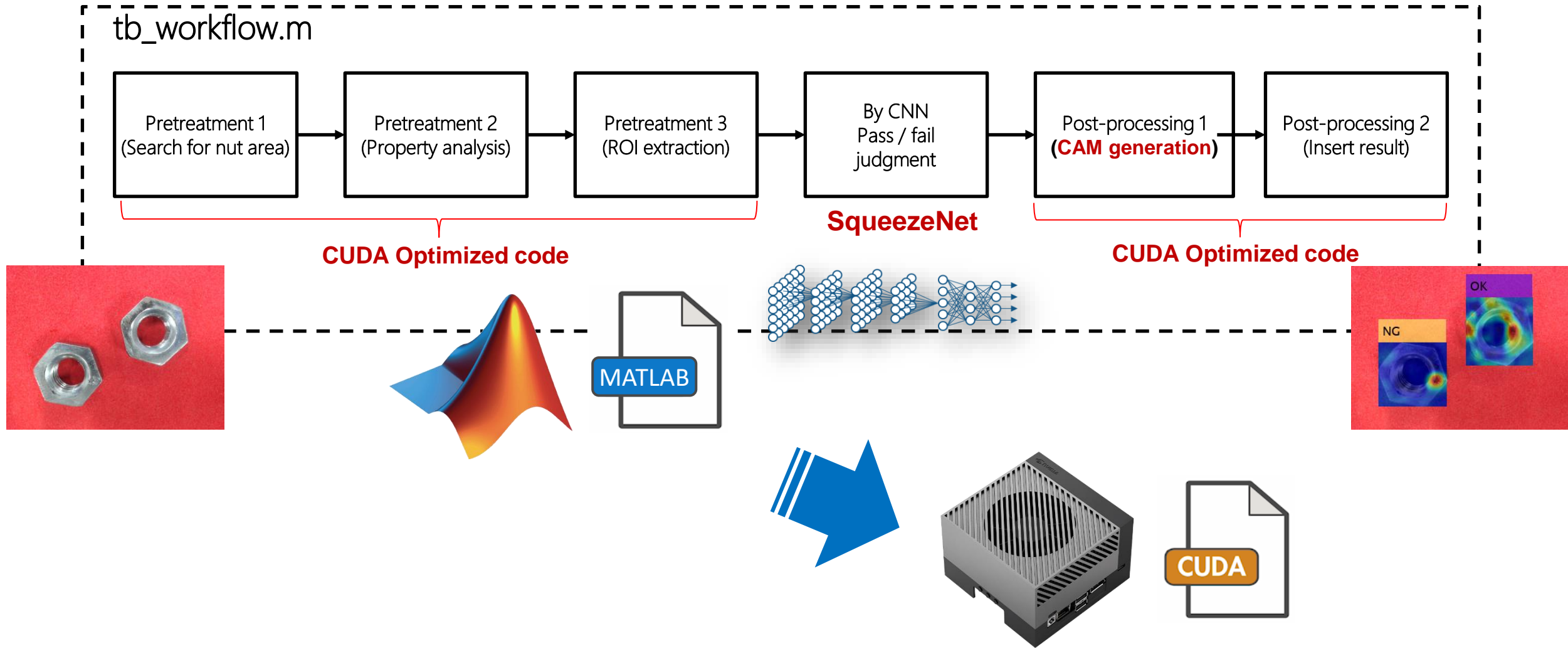


Optimizations for Generated CUDA Code



- Accelerated library support
 - cuFFT, cuBLAS, cuSolver, Thrust, cuDNN, & TensorRT
- Data Transfer Minimization
 - Analyzes data dependency between the CPU and GPU partitions to determine minimum set of locations where data must be copied between CPU and GPU using `cudaMemcpy`

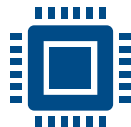
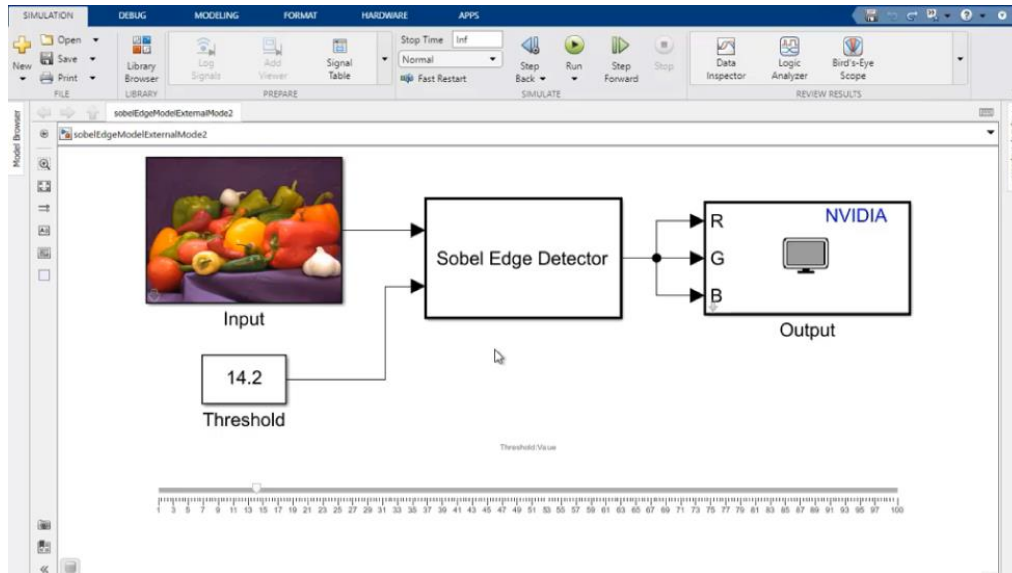
Example 2: Anomaly Detection



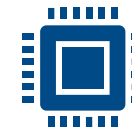
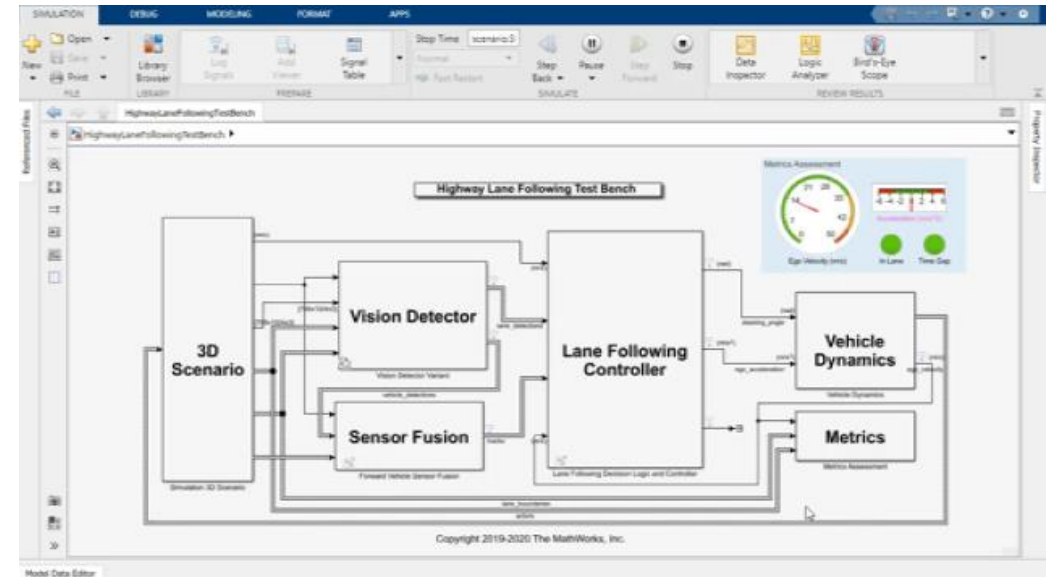
OK, but what about Simulink?

Two Application examples

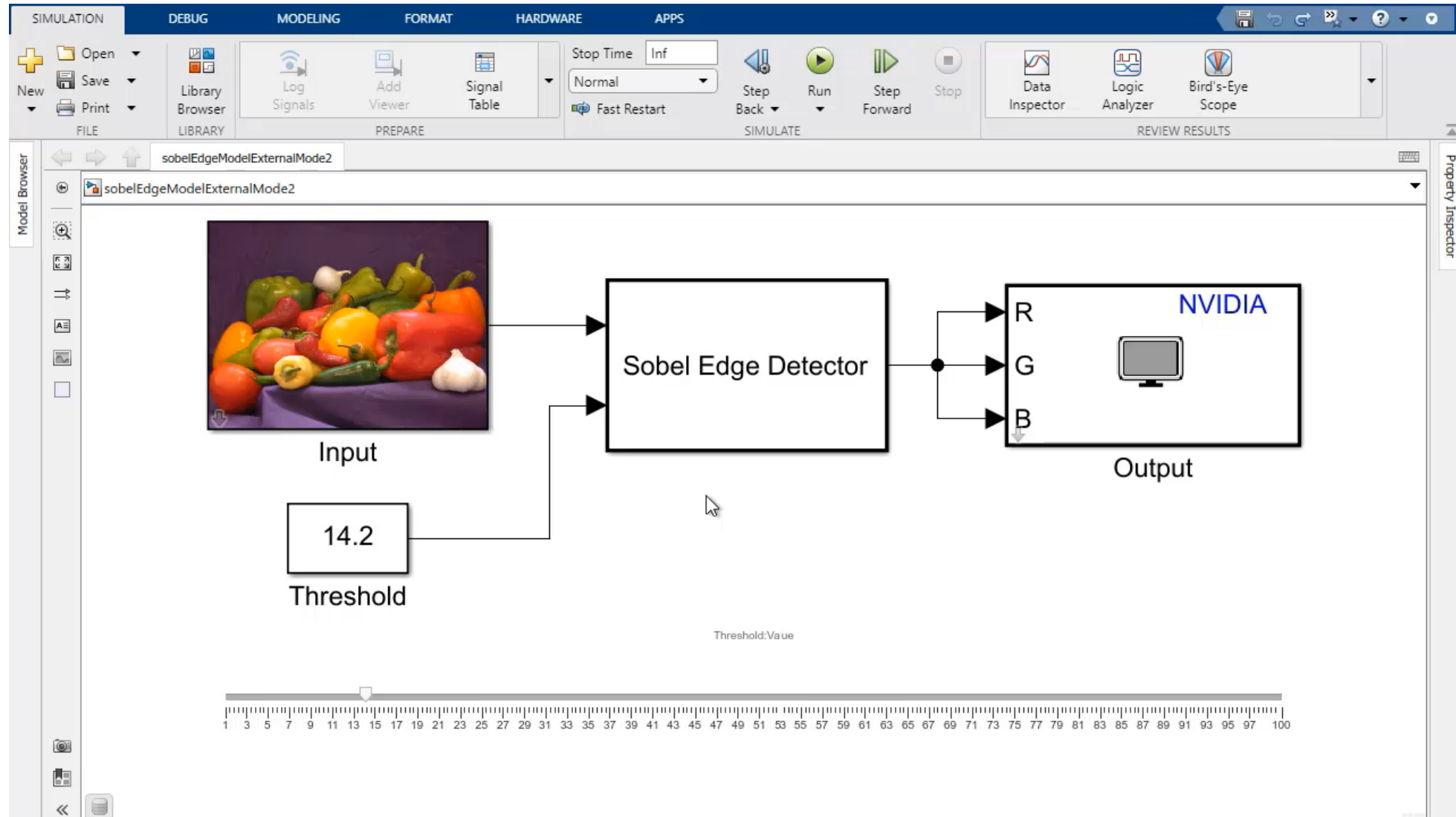
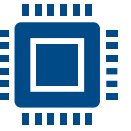
3) Sobel Edge Detection



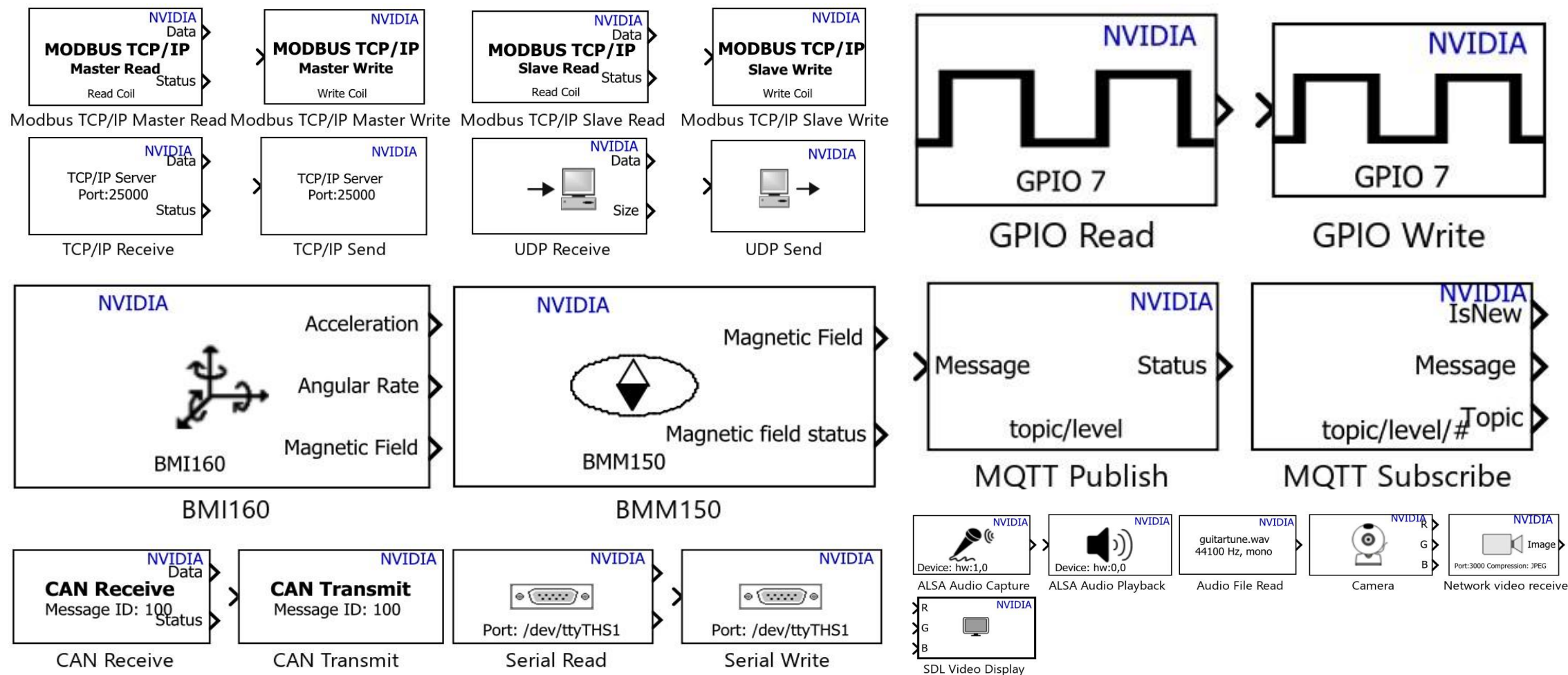
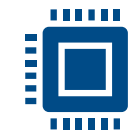
4) Highway Lane Following Model

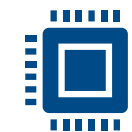


Example 3: Tuning Parameters using External Mode

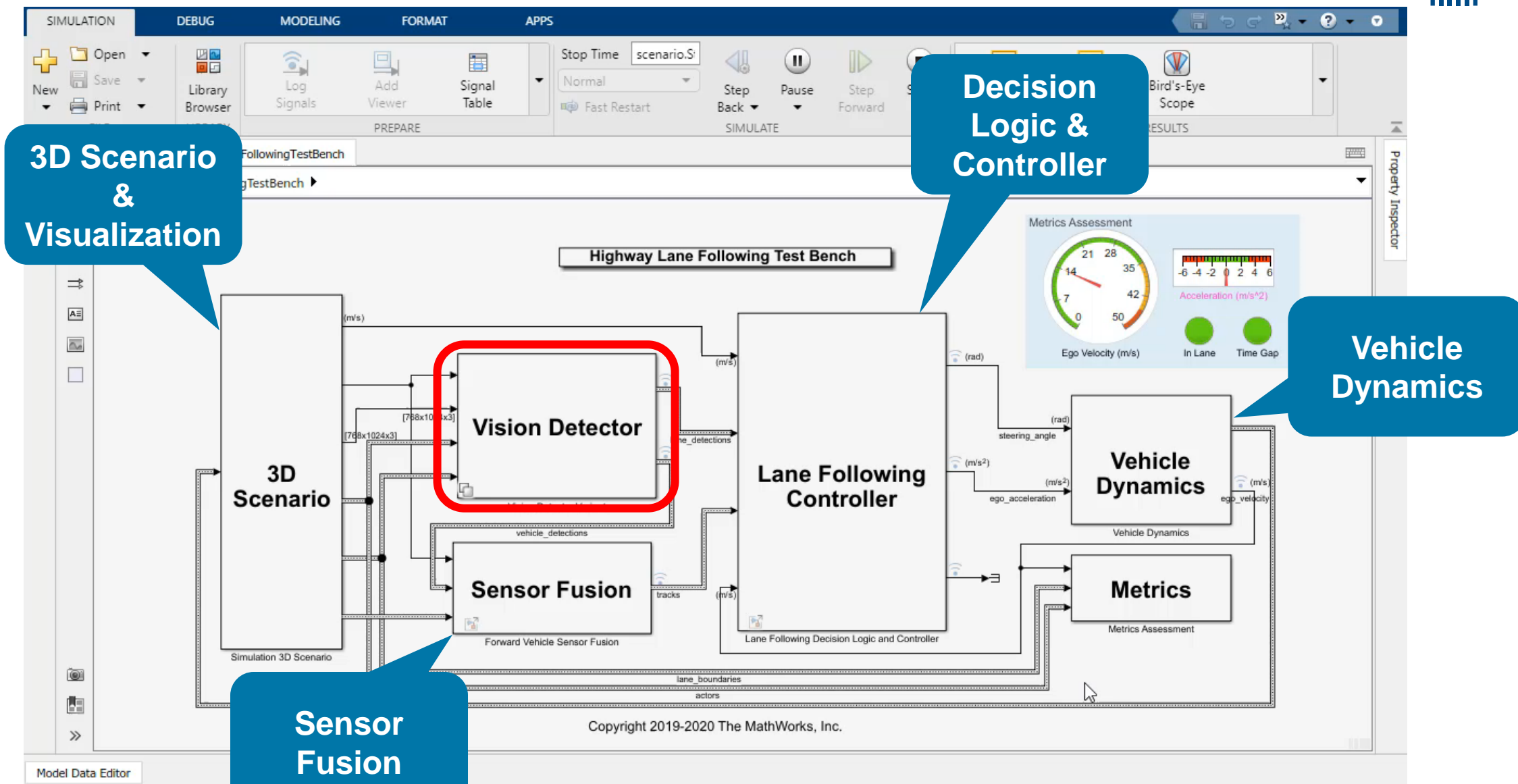


NVIDIA Peripheral Support – block library

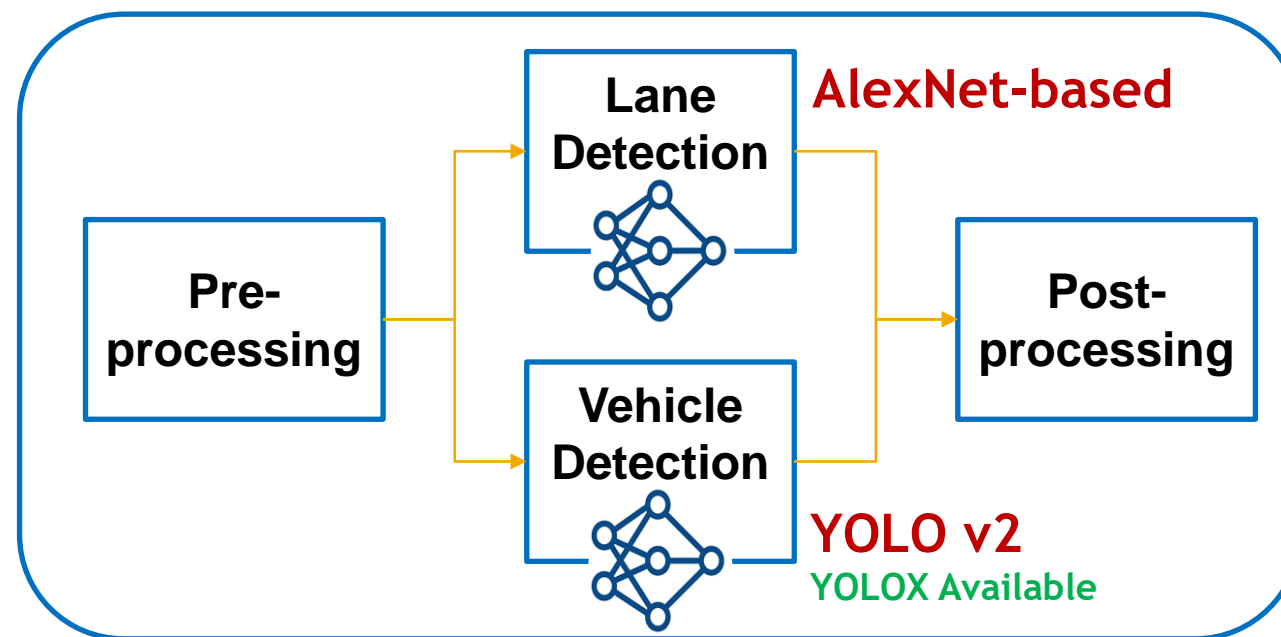
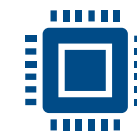




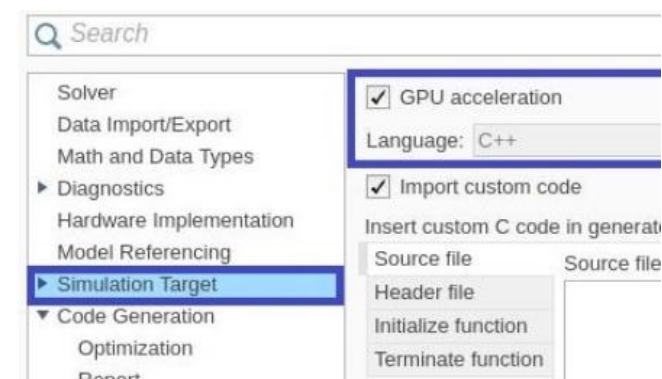
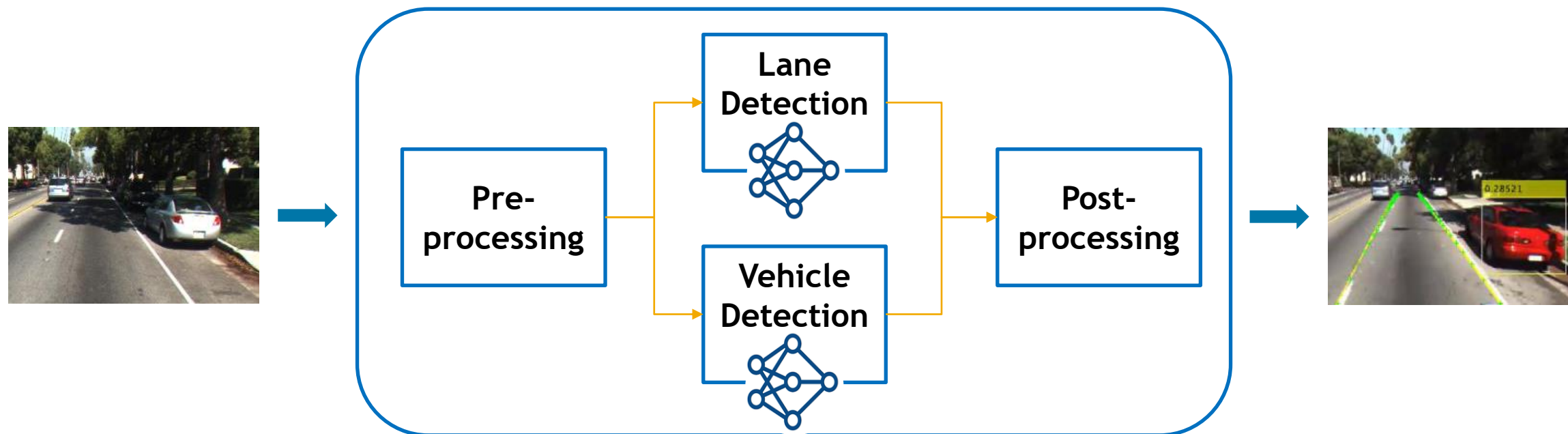
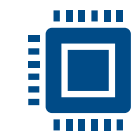
Example 4: Highway Lane Following Model



Lane and Vehicle Detection

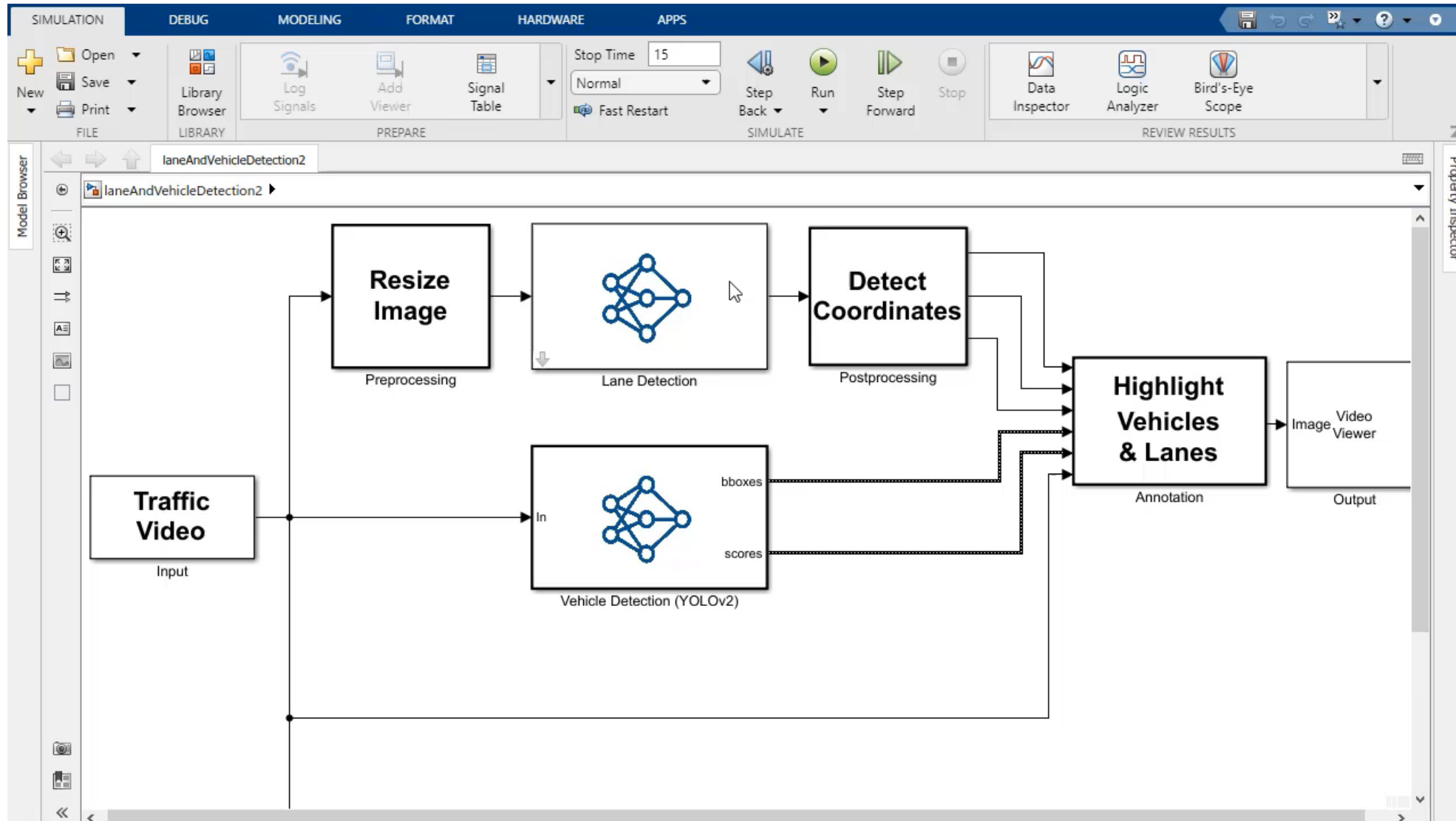
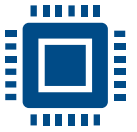


Run Simulation on Desktop GPU

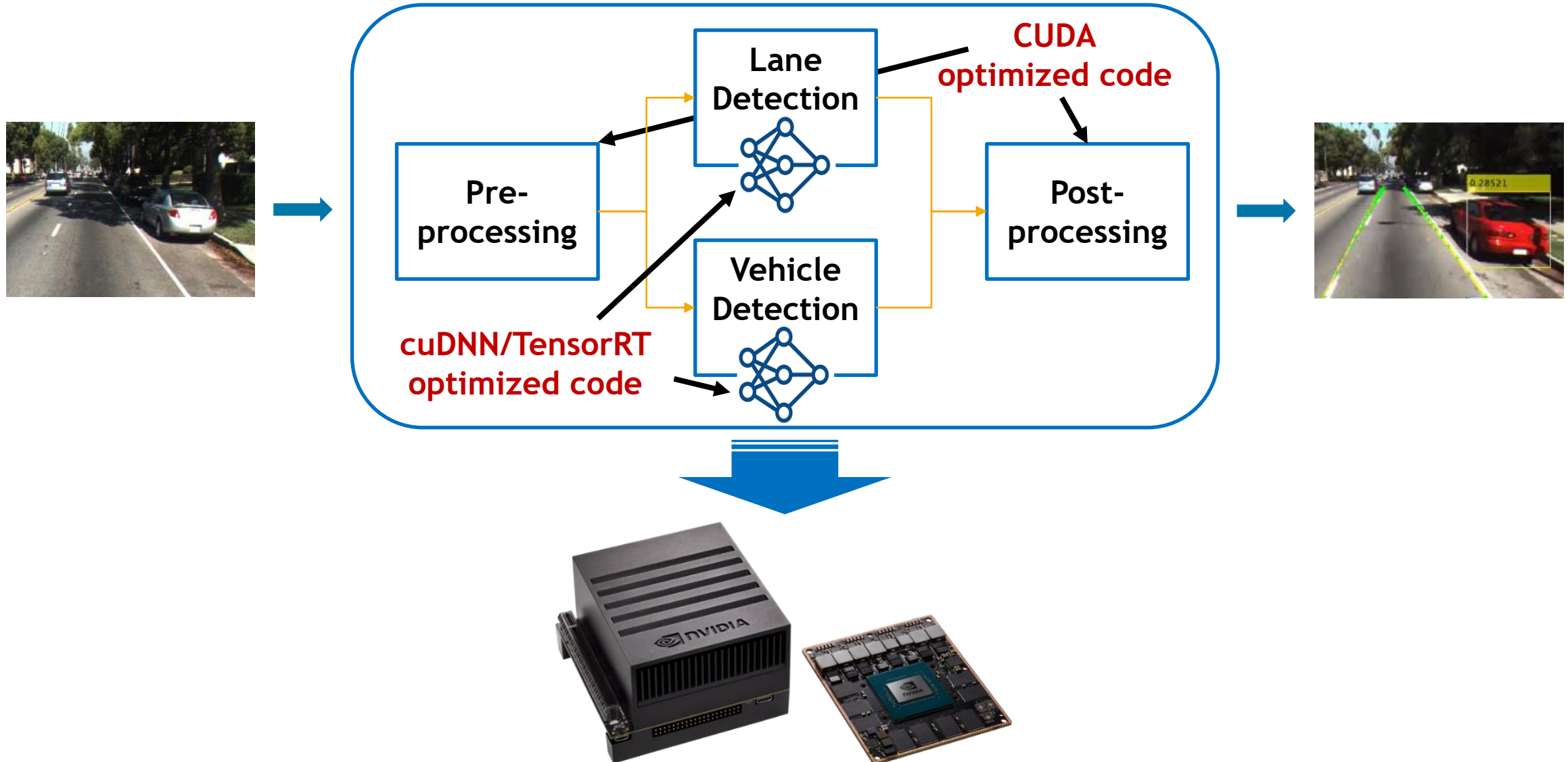
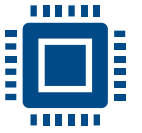


Speed up MATLAB Function blocks

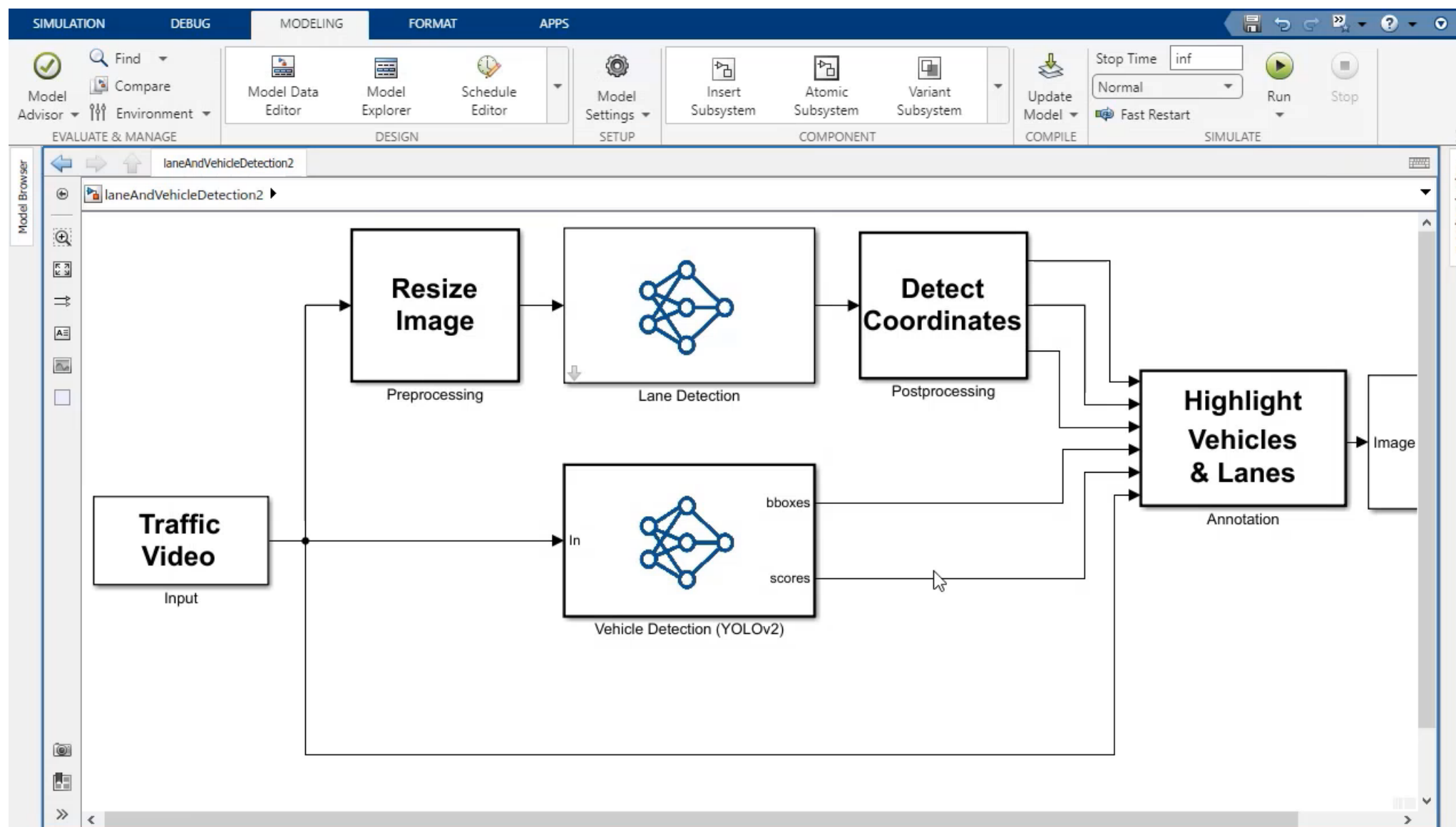
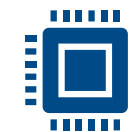
Run Simulation on Desktop GPU



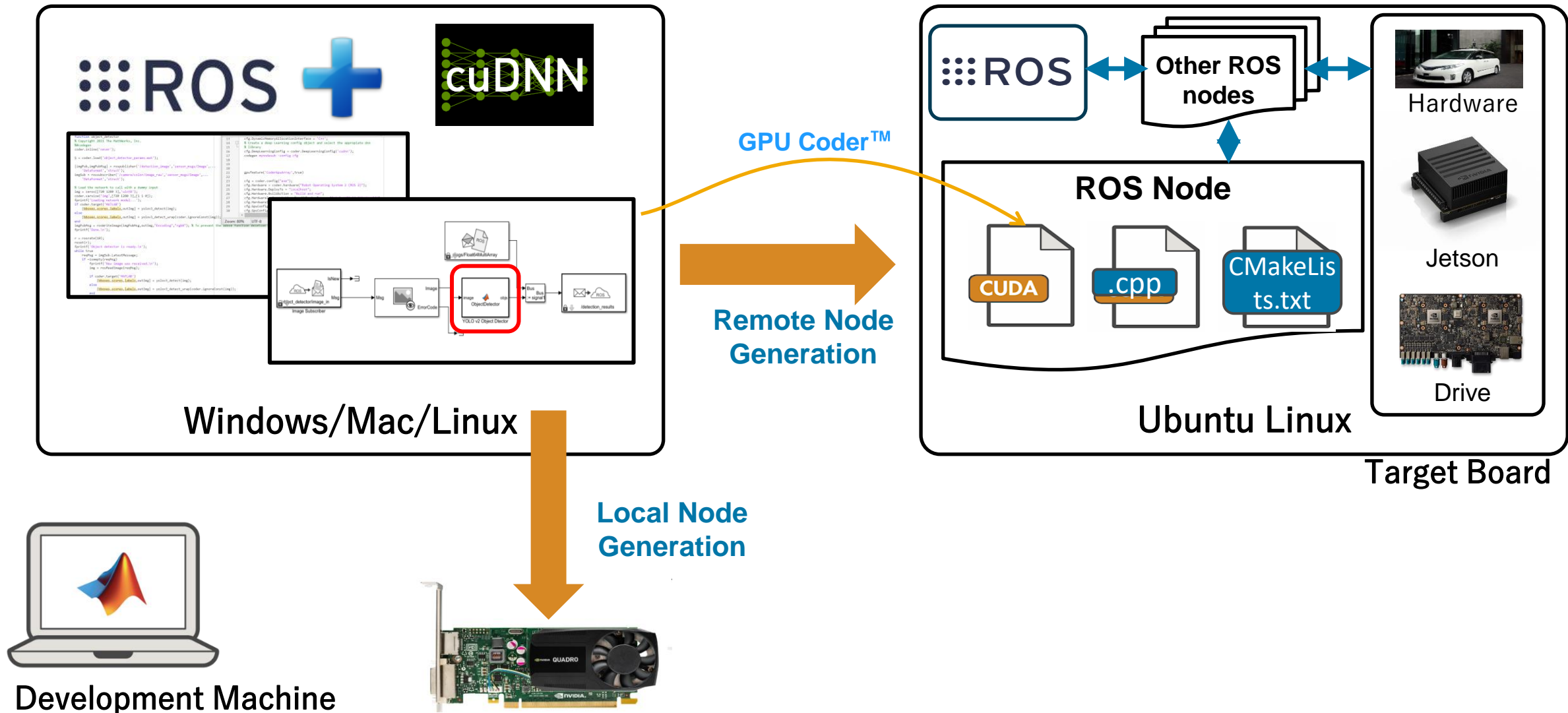
Generate CUDA Code and Run on Jetson



Generate CUDA Code and Run on Jetson



Generate CUDA Enabled ROS and ROS2 Node from MATLAB



Shipping Examples

<https://www.mathworks.com/help/gpucoder/examples.html>

Help Center

Hilfe-Center Suche

INHALTSVERZEICHNIS

« Documentation Home

« Examples

« Code Generation

Category

Fixed-Point Designer

GPU Coder

- Get Started with GPU Coder 2
- Kernel Creation 13
- Performance 3
- Deep Learning with GPU Coder 26
- Deployment 20

HDL Coder

HDL Verifier

IEC Certification Kit

MATLAB Coder

Simulink Code Inspector

Simulink Coder

Simulink PLC Coder

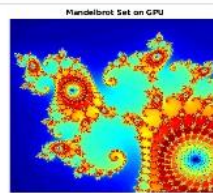
Type

- All 61
- MATLAB 48
- Simulink 13

Documentation Examples Functions Apps Videos Answers

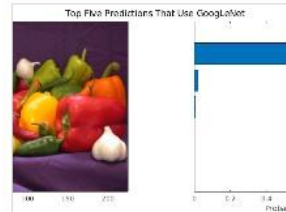
GPU Coder – Examples

Get Started with GPU Coder



GPU Code Generation: The Mandelbrot Set

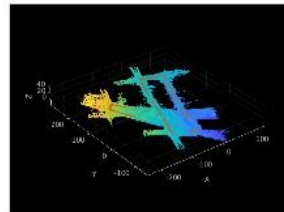
Generate CUDA® code from a simple MATLAB® function by using GPU Coder™. A Mandelbrot set implementation by using standard



Code Generation for Deep Learning Networks

Get started with CUDA code generation for image classification networks such as MobileNet-v2, ResNet, and GoogleNet.

Kernel Creation



Build a Map from Lidar Data



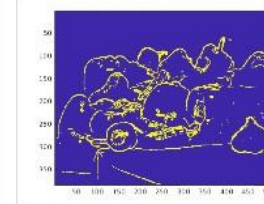
Feature Extraction Using



Feature Matching



Lane Detection on the GPU



Edge Detection with Sobel



GPU Code Generation for

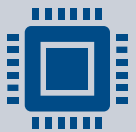
Key Takeaways



GPU Coder generates CUDA code from MATLAB & Simulink



Accelerate MATLAB & Simulink simulations



Deploy algorithms (signal/deep learning,...) to embedded GPUs

DEMO Booth



MATLAB EXPO

Edge GPU 기반 On-device AI

신행재, 매스웍스코리아

 MathWorks

Navigation icons: back, forward, search, home, refresh, close, and more options.

The slide features a dark blue background with a complex, glowing orange network of lines and nodes on the right side, resembling a neural network or data flow diagram. The text is in white, and the MathWorks logo is in its characteristic orange and blue colors.

MATLAB EXPO



© 2024 The MathWorks, Inc. MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See mathworks.com/trademarks for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

