白皮书

从 MATLAB 生成 CUDA 代码：加速基于 GPU 的嵌入式视觉和深度学习算法

MathWorks®

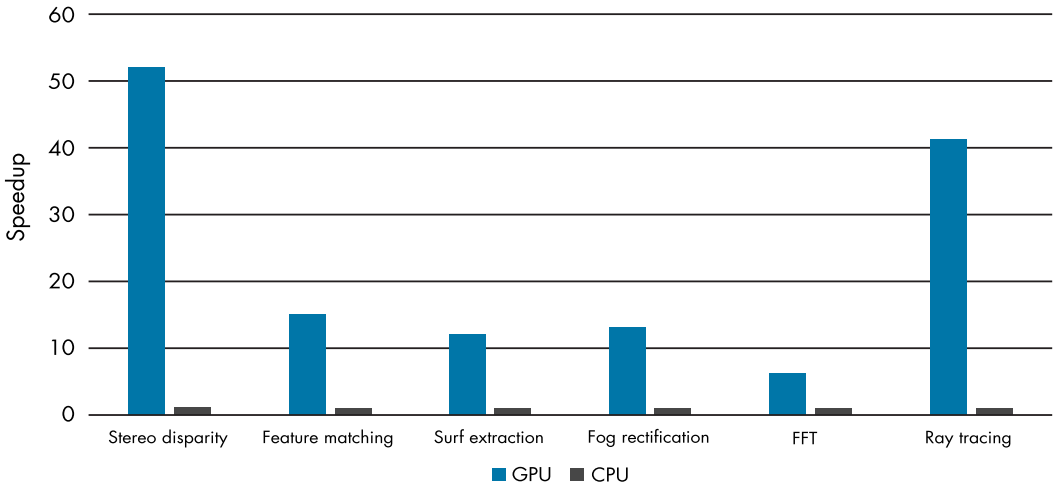
介绍使用 MATLAB 进行 GPU 代码生成

GPU 计算能实现大规模的数据并行运算，因此有助于深度学习应用的增长。通过利用你算法中的数据并行性，GPU得以大幅加快算法执行速度。对于嵌入式视觉、雷达以及深度学习等领域内的计算密集型算法，这是达到性能指标的关键所在。

GPU 加速运算遵循异构编程模型；应用程序中可并行化的部分被映射到GPU的核 (kernel) 上，而GPU内通常有成百上千个核可以同时运行，而代码的顺序部分在 CPU 上运行。

GPU Coder™ 通过从 *MATLAB* 代码自动生成可在 NVIDIA® GPU 上执行的 CUDA® 代码，从而加速现有算法。生成的 CUDA 代码对并行进行了优化，同时最大限度地减少 CPU 和 GPU 之间数据传输的开销。在代码生成过程中，GPU Coder 分析 CPU 和 GPU 部分之间的数据依赖关系。通过这项分析自动确定，GPU和CPU间必须传输的最小一组数据集。CPU 和 GPU 之间的内存数据传输瓶颈是 GPU 加速性能受限的主要原因；只有最大限度地减少数据传输量，性能才会显著提高。自动化工作流程可避免 CUDA 手动编程可能带来的麻烦和容易出现的错误。

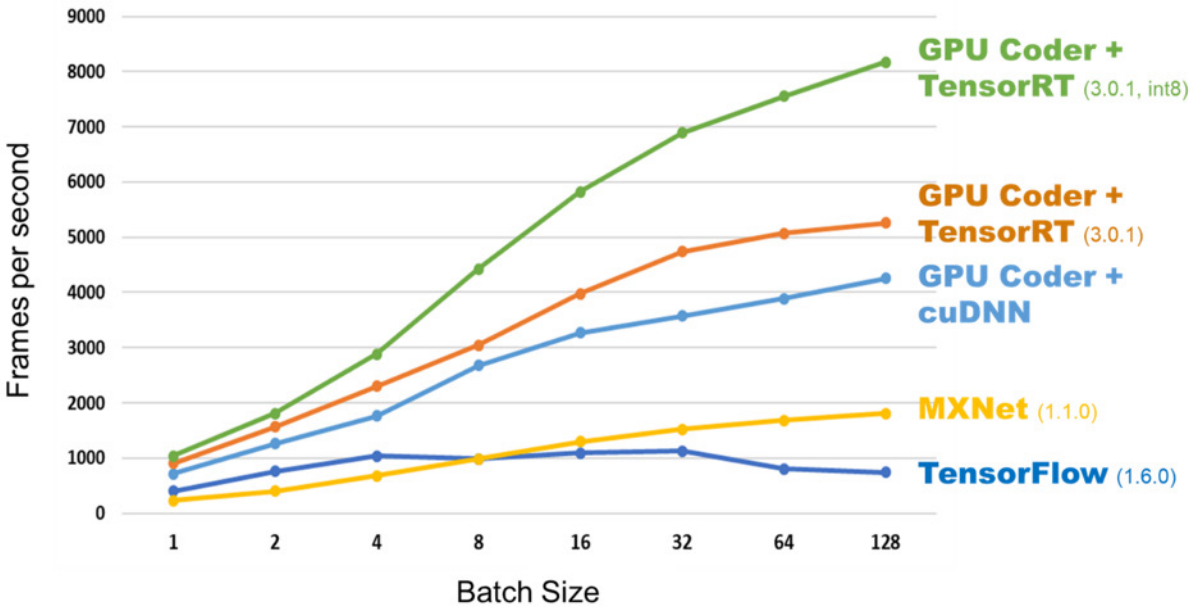
GPU Coder 可以加速在台式机、云或嵌入式设备 (包括 NVIDIA Jetson® 和 NVIDIA Drive® 平台) 上基于 GPU 的算法。图 1 和图 2 突出显示这些性能提升。对于常见图像处理运算 (如 SURF 特征提取、立体视差、去雾) 和常见信号处理运算 (如 FFT)，应用程序可以实现两个数量级的性能提升。



测试平台	MATLAB 2018a
CPU	Intel Xeon CPU E5-1650 v3 @ 3.50 GHz
GPU	NVIDIA Pascal TITAN V (Volta 架构)
CUDA	9.0 版

图 1. 与 CPU 上运行的 C 代码相比，GPU 上运行的由 GPU Coder 生成的 CUDA 代码的常见图像处理和信号处理性能基准。

图 2 显示与其他流行的深度学习框架相比，通过 GPU Coder 生成的 CUDA 代码的推理性能。由 GPU Coder 生成的代码运行速度比 TensorFlow™ 快五倍，是 Apache MXNet 的两倍。



测试平台	MATLAB 2018a
CPU	Intel Xeon CPU E5-1650 v4 @ 3.60 GHz
GPU	NVIDIA Pascal TITAN Xp
cuDNN	v7

图 2. 深度学习性能基准：运行于 NVIDIA Titan® Xp 上的 GPU Coder、TensorFlow 和 MXNet 的 AlexNet 推理性能比较。

注意，典型的深度学习算法由调用一个或多个经过训练的网络进行推理的用户逻辑组成。使用 GPU Coder，您可以从整个应用程序自动生成代码，而无需进行任何手动编程。例如，对于非常常见的图像分类任务，用户逻辑可能包括调整图像大小和更改色度空间等预处理步骤，以及绘制边框等后处理步骤。GPU Coder 能从整个应用程序生成代码（图 3）。

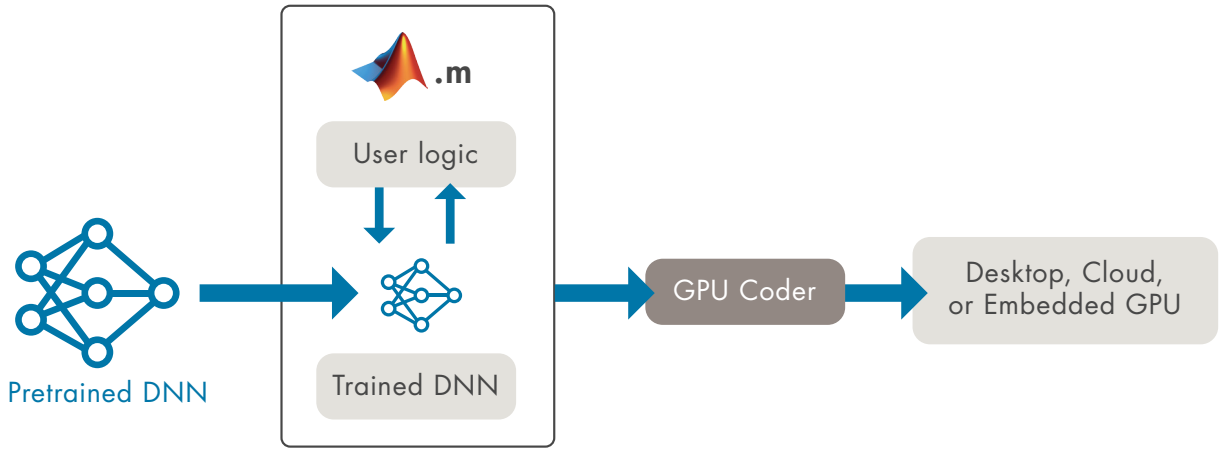


图 3. 使用 GPU Coder 生成整个端到端应用程序的代码。

高层级工作流程

图 4 显示从算法设计、代码生成到部署的典型工作流程。第一步先准备 MATLAB 代码用于代码生成，然后测试生成的代码，确保其与原始 MATLAB 代码的功能表现一致。在第三步中，您可以生成代码作为 MEX 文件（替换原始 MATLAB 算法以达到加速目的），也可以作为源代码形式的 CUDA 代码，或者作为静态或动态库，以便集成到较大 CUDA 项目中。可选的第四步是进一步优化 MATLAB 代码，以提升所生成代码的性能。

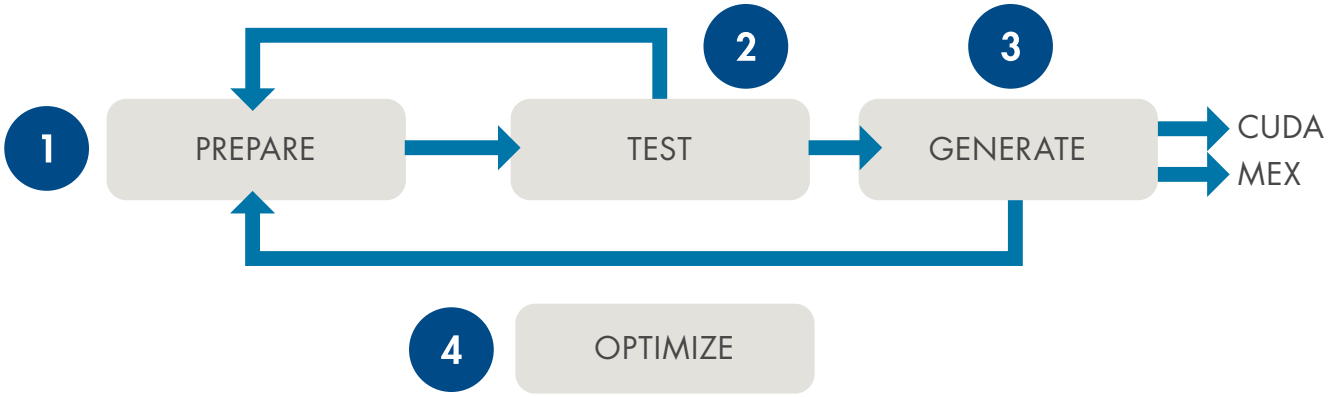


图 4. 从 MATLAB 代码生成 CUDA 代码的工作流程。

交通信号检测应用

本节使用基于深度学习的*交通信号检测示例*，更详细地说明代码生成工作流程。进行交通信号检测和识别的三个基本步骤是检测、非最大值抑制 (NMS) 和识别。您可以通过 MATLAB 算法开始：

- 读取和预处理测试图像
- 调用目标检测网络来检测交通信号
- 使用 NMS 算法来抑制重叠的检测
- 调用识别网络进行交通信号识别
- 显示带边框的输出图像和相应检测标签

要生成此算法的 CUDA 代码，您可以从准备此 MATLAB 脚本进行代码生成开始。

准备 MATLAB 代码进行 GPU 代码生成

要准备 MATLAB 算法进行代码生成，请按以下步骤操作：

1. **从测试平台中分离算法**。首先从测试平台中分离算法，并将算法代码转换为函数。将您的算法分离成深度学习网络推理函数以及预处理和后处理函数也是不错的选择。
2. **找出算法的计算密集部分**。使用 *MATLAB Profiler 剖析*并找出算法中的热点部分。将热点隔离到单独的函数中，有助于下一步操作。
3. **指示 GPU Coder 将热点自动映射成 CUDA 内核程序**。在热点函数的开头插入 `coder.gpu.kernelfun` 编译控制指令。这就指示 GPU Coder 从这些计算量大的函数分析和创建高效的 CUDA 内核程序。
4. **确保算法与代码生成兼容**。使用 GPU Coder 应用（图 5）完成此过程。该应用首先运行代码生成就绪工具（code generation readiness tool） 来检测任何代码生成兼容性问题，包括不支持的函数和结构：

- **不支持的函数**：您需要从 MATLAB 算法中移除或替换*不支持的函数*才可生成代码。
- **无界可变大小数据**：GPU Coder 从有界的变量生成代码。如果代码生成器无法确定数组的大小或者确定大小有变化，则该维度为可变大小（图 6）。GPU Coder 报告出无界变量或无界可变大小数组的警告。在这种情况下，您需要指定上界，以便编码器能够静态地分配内存：
  - 指定可变大小输入的上界：使用 `coder.typeof` 结构指定可变大小输入的上界。
  - 指定本地变量的上界：要限制用来指定可变大小数组维度的变量值，请使用带有关系运算符的断言函数或使用 `coder.varsize` 函数，指定函数中本地变量的所有实例的上界。

5. **检查潜在的运行时错误**。在检查代码生成兼容性之后，建议执行运行时检查来检测和修复运行时错误。运行 CPU 运行时检查，以确认内存完整性，验证数组边界，并执行维度检查。运行 GPU 运行时检查，识别寄存器溢出并确认堆栈大小一致性。

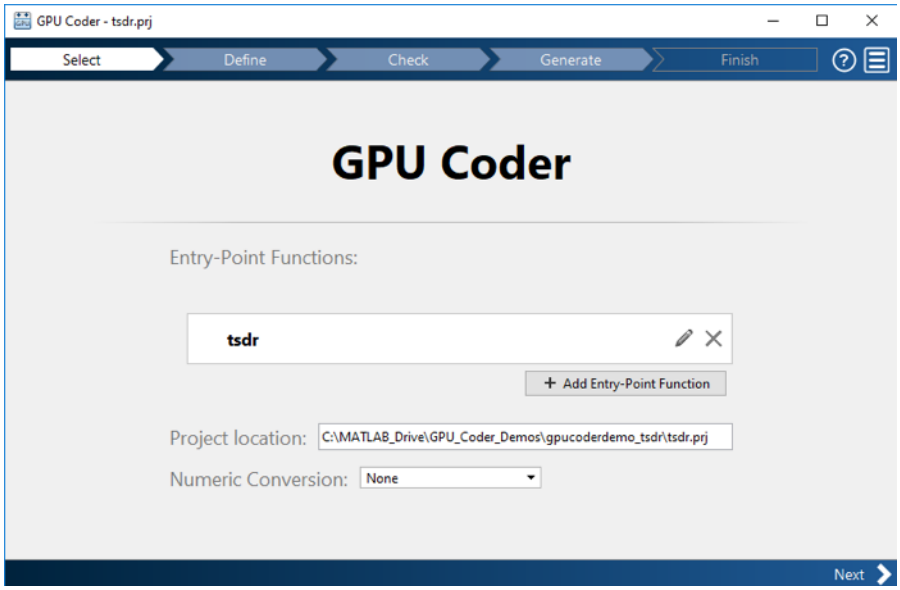


图 5. GPU Coder 应用。

Summary | All Messages (0) | Variables | Target Build Log

Order	Variable	Type	Size	Class	Complex
1	C	Output	1 x :100	double	No
2	A	Input	1 x :100	double	No
3	B	Input	1 x :?	double	No

图 6. Size 字段显示所计算的最大数组大小。对于可变大小数组，冒号 (:) 表示维度为可变大小，问号 (?) 表示大小无界。

准备交通信号检测 MATLAB 脚本进行代码生成

使用上述步骤，第一步是从测试平台脚本 **tsdr _ test.m** 分离出算法代码，现在将其命名为 **tsdr** 函数。当您在 MATLAB 中剖析该函数时，您会看到该函数内没有要隔离的特定热点。这意味着您可以将整个函数映射到 GPU，方法是将 **coder.gpu.kernelfun** 编译控制指令插入到 **tsdr** 函数中。GPU Coder 应用未发现不支持的函数，但会显示该函数有无界大小可变数组。所以，您需要使用 **coder.varsize** 指定上界（图 7）。CPU 和 GPU 的运行时检查均成功完成，没有任何错误。

```
%% Run Non-Maximal Suppression on the detected bounding boxes
coder.varsize('selectedBbox',[98, 4],[1 0]);
[selectedBbox,~] = selectStrongestBbox(round(boxes),probs);
```

```
%% Recognition
persistent recognitionnet;
if isempty(recognitionnet)
     recognitionnet = coder.loadDeepLearningNetwork('RecognitionNet.mat', ...
          'Recognition');
end

coder.varsize('idx',98,1);
idx = zeros(size(selectedBbox,1),1);
inpImg = coder.nullcopy(zeros(48,48,3,size(selectedBbox,1)));
```

图 7. 使用 `coder.varsize` 指定可变大小数组的上界。

测试生成的 MEX

既然已准备好代码进行代码生成，您应该先在 MATLAB 内测试生成的 MEX，确保其与原始 MATLAB 代码的功能表现一致。为此，您需要生成一个 MEX 函数。MEX 是一种包装程序接口，让您能够在 MATLAB 中调用编译的二进制代码，并用它代替原始 MATLAB 函数，验证它与原始 MATLAB 函数具有相同的表现。

在 GPU Coder 应用中，选择 MEX 编译类型，以便生成 CUDA 代码（图 8），然后使用 NVIDIA 的 nvcc 编译器将其编译成一个 MEX 函数。在验证代码步骤中，通过从测试平台调用 MEX 函数，您可以验证生成的代码与原始 MATLAB 函数具有相同的功能。

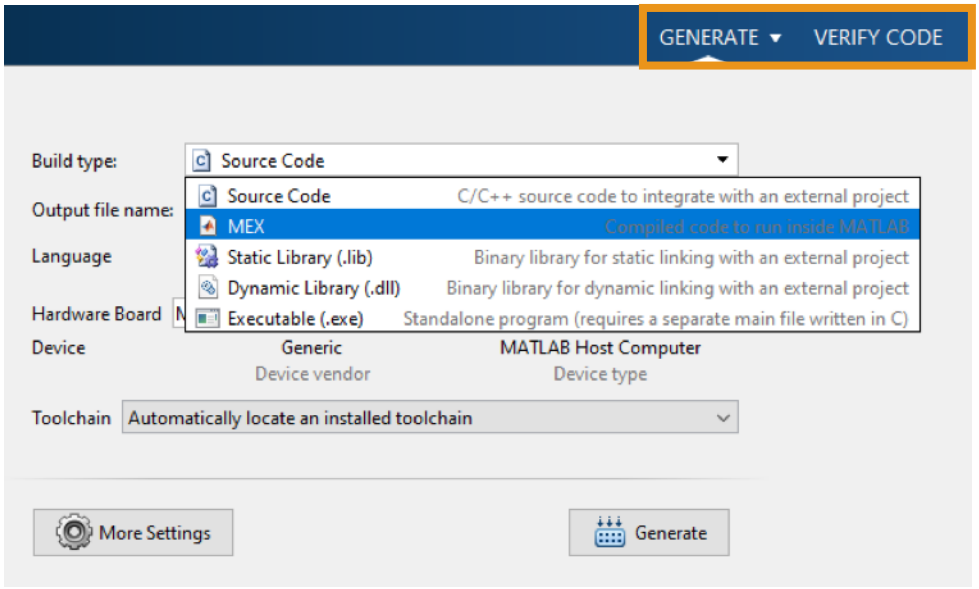


图 8. 在 GPU Coder 应用中选择编译目标以及验证代码步骤。

在此示例应用中，您生成 MEX 目标，并在验证代码步骤中使用 MEX 版本运行测试，验证所生成代码的功能表现。

在嵌入式 GPU 上部署生成的代码后，借助 GPU Coder 与 *Embedded Coder®*，您可以使用*软件在环 (SIL) 执行程序*来验证生成的代码，进一步验证所生成独立 C++ 代码的数字行为。您可以检测出 MEX 与独立的产品级质量代码之间的任何差异。

以 MEX 或源代码的形式生成代码：在台式机、云端或嵌入式 GPU 上部署

在验证生成的 MEX 函数具有预期表现后，您可以生成独立 C++ 代码或静态库，只需更改编译类型选项即可。然后，您可以将生成的库集成到您的开发环境中的大型应用程序，将您的算法部署到台式机或云端。

如果您的目标是部署到嵌入式平台，如 NVIDIA Jetson 或 Drive，您可以将 CUDA 源代码导出到目标平台，在目标平台上进行编译。对于 NVIDIA Tegra® 开发板，如 Jetson TX2、TX1 或 TK1，还可以在您的主机上交叉编译生成的代码，通过将可执行文件复制到目标，从而部署到板上。

使用 MEX 构建类型生成代码的一个重要使用案例是，通过调用在 GPU 上运行的已编译库，而不是需要解释执行的原生 MATLAB 函数，从而加速 MATLAB 中的计算速度。

生成代码并部署交通信号检测应用

在交通信号检测示例应用中，目标是在台式机上部署算法。在这种情况下，您选择编译类型为源代码，生成可集成到主文件并编译成可执行文件的独立 C++ 代码。您甚至可以更进一步，生成可以交叉编译并部署到 Jetson 板上的静态库，如 AlexNet 示例中的说明。

进一步优化代码的技巧

如果生成的代码不符合您的性能要求，您可以采取额外步骤进行优化，以提高生成代码的性能表现：

- **使用剖析 MEX**。要找出所生成代码的性能瓶颈，可以通过使用 MATLAB Profiler 剖析所生成 MEX 函数的执行时间。对所生成代码的剖析显示相应 MATLAB 函数的调用次数以及每一行花费的时间。通过对您的 MATLAB 算法进行一些修改，或通过调节代码生成选项，可以解决某些瓶颈；下面列出了一些技巧。
- **使用 `coder.gpu.kernel` 编译控制指令**。如果您的算法中有些并行循环没有在所生成代码的部分中进行并行处理，您可以在循环的前面插入 `coder.gpu.kernel` 编译控制指令，强制 GPU Coder 为该循环生成一个内核程序。注意，这是高阶知识，应小心使用。
- **使用设计模式**。对于某些反复访问某一给定输入元素来计算多个相邻输出元素的算法，可以使用诸如模板计算 (`gpucoder.stencilKernel`) 或矩阵-矩阵运算 (`gpucoder.matrixMatrixKernel`) 等设计模式来提高计算效率。这些设计模式利用 GPU 的共享内存来提升性能。请参阅*设计模式文档*了解详细信息。
- **插入自定义 CUDA 代码**。如果您有高度优化的 CUDA 代码，用于您希望合并到生成代码中的某些子函数，在这种情形下，GPU Coder 扩展 `coder.ceval` 功能，可帮助您将自定义代码与 GPU Coder 生成的代码集成在一起。请参阅*有关继承代码集成的文档*了解详细信息。

- **自定义代码生成选项**。您可以自定义代码生成配置，针对您的特定需求进一步优化代码生成，使用的选项包括：
  - 设置内存分配模式，可在离散和统一内存分配之间选择
  - 启用对 cuBLAS 和 cuSOLVER 等 CUDA 库的支持
  - 指定目标硬件的计算能力
  - 将附加标记传递给 GPU 编译器

后续步骤

- 浏览：*从 MATLAB 生成 GPU 代码*
- 下载：*GPU Coder 试用版*