CUDA Control

Design Document

SIGNALPOP LLC

June 2, 2017
Authored by: Dave Brown

CUDA Control

Design Document

Contents

Chapter 1 - Overview

The CUDA Control is designed to help better integrate CUDA based modules into .NET C# applications. To do this, the CUDA control leverages ATL ActiveX technologies and OLE Automation which is easily integrated into any .NET C# application along with a very simple DLL interface used on the '.cu' CUDA side of the equation. Such an interface allows for easy reuse of the .NET integration software all the while supporting any CUDA based DLL which then makes for easy debugging and increases the overall CUDA software development task.

This document describes the design of the CUDA Control, and how it interfaces with .NET applications on one side and the simple CUDA custom DLL on the other.

Chapter Summary

The following chapters are in this document.

Chapter 1 - Overview; this chapter.

Chapter 2 - Module Interactions; describes the interactions that take place between each module in the system.

Chapter 3 - Use Cases; describes several common use cases that occur when using the CUDA Control.

Chapter 4 - Module Interfaces - .NET; describes the OLE Automation interface used by .NET C# applications to program the CUDA Control.

Chapter 5 - Module Interfaces - CUDA DLL; describes the simple interface used to talk to any CUDA based DLL that implements the CUDA code.

Chapter 2 - Module Interactions

The following shows the modules and how they interact with the system.

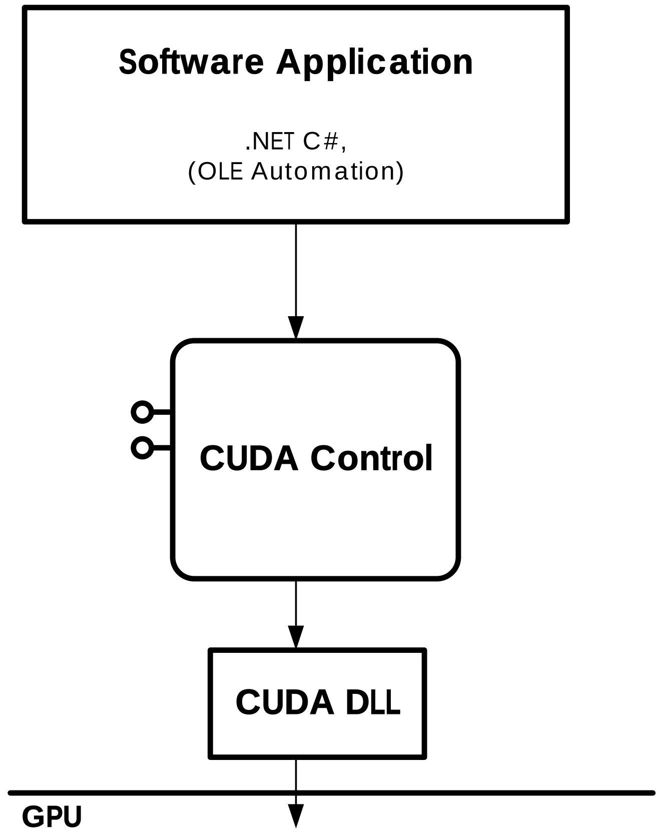

FIGURE 1 CUDA CONTROL MODULE INTERACTIONS

The modules making up the system are described as follows.

Module	Description
Software Application	The software application is the software that uses the CUDA Control to process data using the CUDA DLL, but doing so from a .NET application via an OLE Automation interface.
CUDA Control	The CUDA Control is an ActiveX control supporting an OLE Automation interface to facilitate data transfers between the software application and the CUDA DLL.
CUDA DLL	The CUDA DLL is a standard DLL with a set of very simple exported DLL functions. The main purpose of the DLL is to implement the native CUDA algorithms (i.e. the Kernel) that is to run on the CUDA enabled GPU.
GPU	The GPU is the Graphics Processing Unit that supports CUDA. For example, the NVIDIA GT9800 is an example GUP that supports the CUDA software interfaces.

Chapter 3 - Use Cases

The following sections describe several common uses of the CUDA Control and shows how the modules making up the system interact during each use case.

Initialization

There are two phases to initialization: 1.) Loading the CUDA DLL to be used, and 2.) direction the CUDA DLL to initialize any memory that is to be used on the GPU.

Loading

The initial step of using the CUDA Control is to load the CUDA DLL that it is to use. This step typically occurs when the CUDA Control is first used.

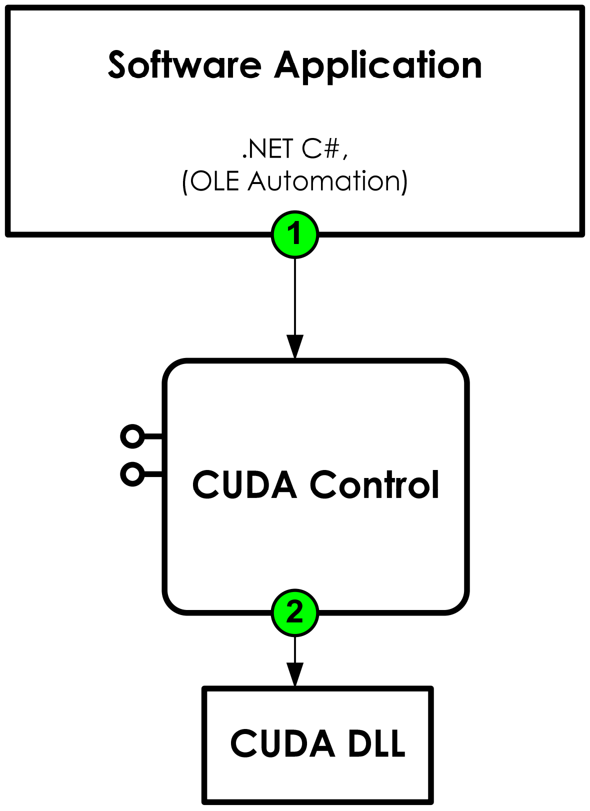

FIGURE 2 LOADING THE CUDA DLL

When loading the CUDA Control the following steps occur.

1.) First the software application using the CUDA Control must direct the CUDA Control to load the DLL at which time the application passes the name of the DLL to be used to the CUDA Control.

2.) When directed, the CUDA Control dynamically loads the CUDA DLL and sets a function pointer to each of the CUDA DLL's exposed functions.

At this time the CUDA Control is ready to be used.

CUDA

After the DLL is loaded, the CUDA Control may be used to direct the CUDA DLL to initialize any memory used on the GPU.

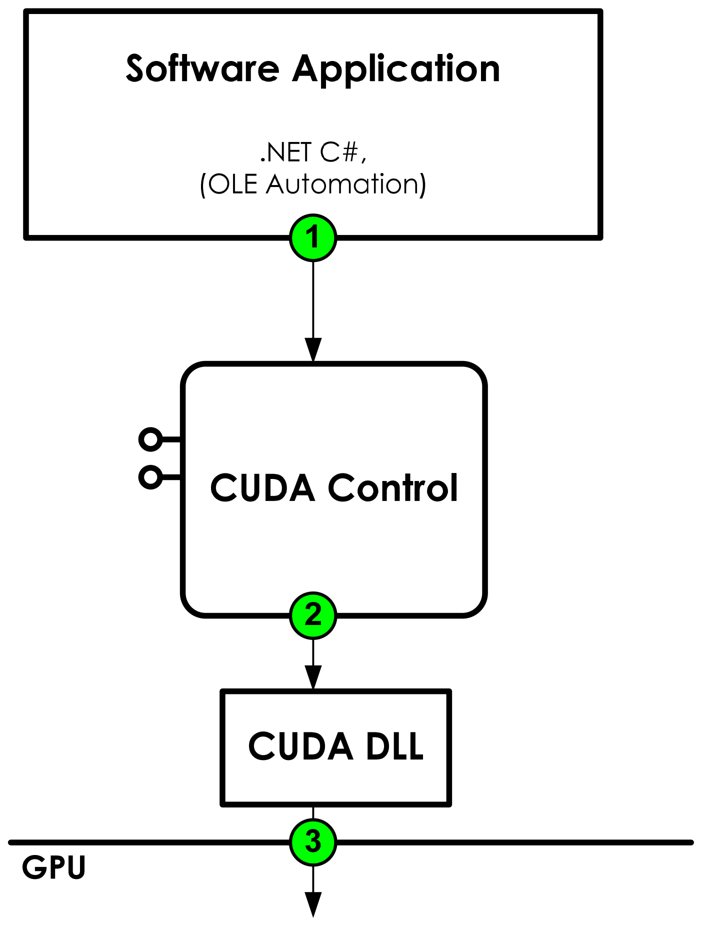

FIGURE 3 INITIALIZING THE CUDA GPU

The following steps occur when initializing the GPU memory.

1.) First the software application directs the CUDA Control to initialize itself.

2.) Next, the CUDA Control delegates the call to the CUDA DLL.

3.) And the CUDA DLL then directs the GPU to allocate and initialize memory to be used when later running the CUDA parallel algorithm implemented within the DLL.

Running

Once the CUDA Control and CUDA DLL are initialized, they are ready to process data using the algorithm implemented within the CUDA DLL.

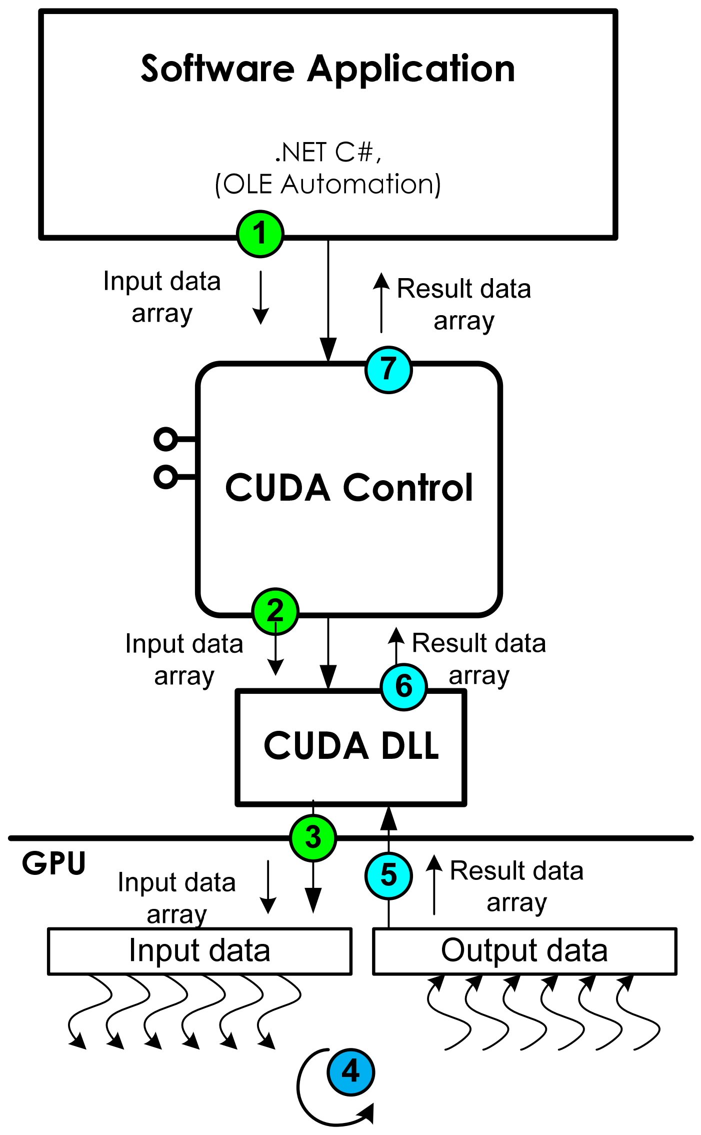

FIGURE 4 RUNNING A FUNCTION

The following steps take place when running a function.

1.) First the software application calls the 'Invoke' method on the CUDA Control, passing into the method an array of input data.

2.) The CUDA Control converts the OLE Automation SAFEARRAY of data into a native C++ array of data and passes the native C++ data to the CUDA DLL.

3.) The CUDA DLL passes the native data to the GPU using CUDA function calls.

4.) The GPU runs the kernel in parallel thus processing the data and producing the output data.

5.) The output data is then returned to the CUDA DLL.

6.) The CUDA DLL passes the native data back to the CUDA Control.

7.) And the CUDA Control converts the native data into a SAFEARRAY which is then returned to the software application.

Clean-up

Clean-up occurs when the software application releases the CUDA Control.

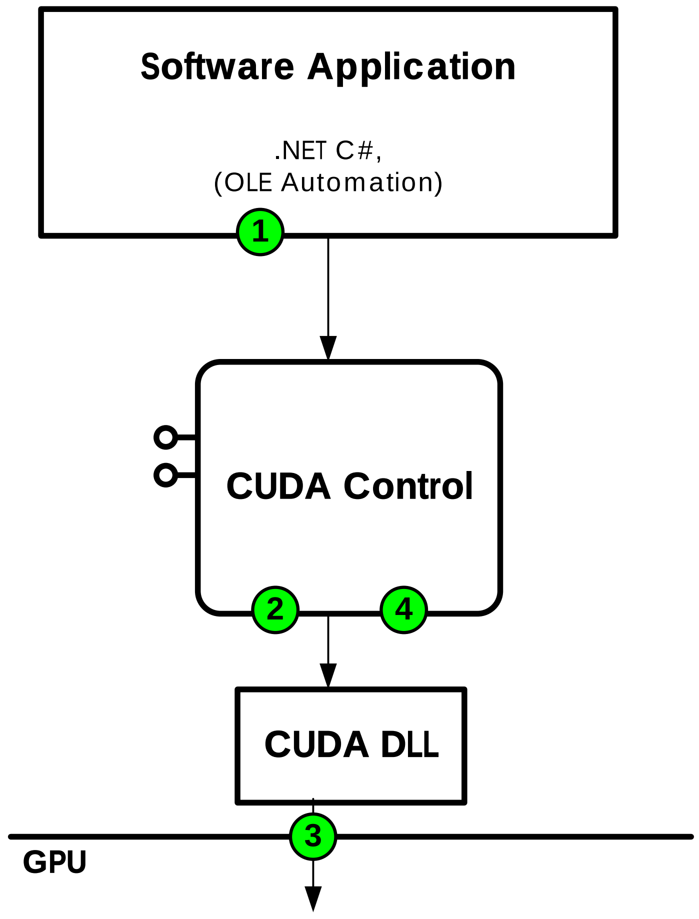

FIGURE 5 CLEAN-UP

The following steps take place during clean-up.

1.) When the software application is done using the CUDA Control, the control is automatically released during the software applications garbage collection process.

2.) When released, the CUDA Control, directs the CUDA DLL to free any memory used.

3.) When directed, the CUDA DLL frees any memory used on the GPU.

4.) After all memory is freed up, the CUDA Control unloads the CUDA DLL.

Chapter 4 - Module Interfaces - .NET

This section describes the OLE Automation interface used by the .NET software application to communicate with the CUDA Control.

IKernel Interface

The **IKernel** interface is used to setup and run kernels on a CUDA based GPU. The following methods are in this interface.

Load - this method is used to load a new CUDA DLL module.

Initialize - this method is used to initialize any memory used by the CUDA DLL.

Cleanup - this method is used to free any memory used by the CUDA DLL.

Run - this method is used to run a kernel implemented within the CUDA DLL.

IKernel.Load

This method is used to load the CUDA DLL that implements one or more kernel functions that run on the CUDA based GPU.

Syntax	`bool IKernel.Load(string strDLLFileName);`
Parameters	***string*** *strDLLFileName* - specifies the DLL file name of the CUDA DLL that implements one or more kernels to run on the GPU.
Return Value	***bool*** - True is returned on successful load, False otherwise.
Exceptions	An exception is thrown if the CUDA DLL cannot be loaded or is in an unrecognized form. See Chapter 4 - Module Interfaces - CUDA DLL for a description of the expected functions that must be implemented on the CUDA DLL.

IKernel.Initialize

This method is used to initialize any memory or other resources used by the CUDA Control or underlying CUDA DLL.

Syntax	`bool IKernel.Initialize(float[] rgparam);`
Parameters	***float[]*** *rgparam* - specifies the initialization parameters used to allocate memory used by the 'Run' method.
Return Value	***bool*** - True is returned on successful initialization, False otherwise.
Exceptions	An exception is thrown if the CUDA DLL cannot be initialized.

IKernel.Cleanup

This method is used to release any memory or other resources used by the CUDA Control or underlying CUDA DLL.

Syntax	`bool IKernel.Cleanup();`
Return Value	***bool*** - True is returned on successful cleanup, False otherwise.

IKernel.Run

This method is used to run a kernel implemented within the CUDA DLL.

Syntax	`float[] IKernel.Run(int nIdx, float[] rginput);`
Parameters	***int*** *nIdx* - specifies which kernel to implement. The default value of '0' specifies to run the default kernel. Other values are only used when the CUDA DLL implements more than one kernel.
	float[] *rginput* **-** specifies the input data to be processed by the kernel. A value of 'null' specifies to use any input data sent in a previous call to 'Run'.
Return Value	***float[]*** - The output data produced by the call to the 'Run' method, is returned.
Exceptions	An exception is thrown if the CUDA DLL fails to run the 'Run' method.

Chapter 5 - Module Interfaces - CUDA DLL

This section describes the DLL functions exported by the CUDA DLL and used by the CUDA Control to communicate with the CUDA DLL.

DLL_Invoke

The CUDA DLL implements and exports a single entry point called 'DLL_Invoke'. This method is called with differing parameter signatures depending on the function's use. See sections below for the various function uses.

Syntax

```
long DLL_Invoke(int nIdx,
                float* pInput, long lInput,
                float* pOutput, long* plOutput,
                LPTSTR szErr, long lszErrMax);
```

Parameters

int *nIdx* - specifies which kernel to implement. The default value of '0' specifies to run the default kernel. Other values are only used when the CUDA DLL implements more than one kernel.

float* *pInput* **-** specifies the input data to be processed by the kernel. A value of 'null' specifies to use any input data sent in a previous call to 'DLL_Invoke'.

long *lInput* **-** specifies the number of 'double' values in the 'pInput' array.

float* *pOutput* **-** specifies the output produced by the kernel, if any.

long* *plOutput* **-** specifies the number of 'double' values in the 'pOutput' array, if any.

LPTSTR *szErr* **-** specifies a buffer where any error description text is place in the event an error occurs..

long *lszErrMax* **-** specifies the maximum number of characters that can be copied into the 'szErr' buffer, including the trailing NULL.

Return Value

long - specifies the result of the function, where a value of '0' specifies success and any other value specifies an error code.

DLL_Invoke Indices

The following standard function indices should be supported by the CUDA DLL.

CUDA_DLL_INITIALIZE (-1)

The CUDA_DLL_INITIALIZE index directs the CUDA DLL to initialize any memory to be used within the DLL and/or on the GPU. The following parameter signature is used for this function index.

`pInput[0]` Specifies the size of the input array's that will be sent on each run of the kernel.

CUDA_DLL_CLEANUP (-2)

The CUDA_DLL_CLEANUP index directs the CUDA DLL to free any memory or other resources used.

CUDA_DLL_RUN (0)

The CUDA_DLL_RUN index directs the CUDA DLL to run the kernel on the data.