Dr.Dobb's
THE WORLD OF SOFTWARE DEVELOPMENT

CUDA, Supercomputing for the Masses: Part 19

Parallel Nsight Part 1: Configuring and Debugging Applications

July 28, 2010
URL:http://www.drdobbs.com/parallel/cuda-supercomputing-for-the-masses-part/226300200

In CUDA, Supercomputing for the Masses Part 18, I demonstrated how to achieve very high rendering and compute performance by mixing CUDA and OpenGL in the same program and through the use of primitive restart, an OpenGL extension CUDA programmers can exploit to bypass PCIe bottlenecks to increase rendering performance by nearly 100 frames per second. This is the first of a two-part article that focuses on how to use NVIDIA's highly anticipated Visual Studio based Parallel Nsight debugging and profiling environment for Microsoft Windows to create and profile applications. Specifically, this article discusses the thinking behind Parallel Nsight; how to install and configure the software; plus walk through the steps to create a CUDA project from scratch and debug it. The example code from Part 14 that was used to demonstrate cuda-gdb will be built with Visual Studio and debugged with Parallel Nsight. The next article uses the Parallel Nsight 1.0 analysis capabilities to compare the Part 18 primitive restart OpenGL example with more conventional OpenGL rendering methods.

Regular readers of this series will note that the use of Visual Studio represents a departure from the previous articles in this tutorial series, which utilized the Linux tool chain to edit and create CUDA applications. With the release of Parallel Nsight, NVIDIA has made a commitment to the debugging and profiling needs of a huge base of Microsoft Windows developers ranging from game developers to commercial High Performance Computing (HPC) users. In addition to CUDA, Parallel Nsight also provides developers the ability to analyze and debug HLSL textures plus OpenCL application tracing is also supported. All the features discussed in this article are part of the standard version that is available without charge. The analysis features require the professional version, which must be purchased.

The Thought Behind Parallel Nsight

Parallel Nsight was designed from the very beginning to fully support remote debugging. Instead of building an application and running it under a debugger, Parallel Nsight utilizes a host machine to build the executable with Visual Studio 2008 service pack 1 or higher. (Visual Studio 2010 support is coming soon.) The executable is then exported to a target machine via the Parallel Nsight monitor process, which runs the application and handles debugging and tracing operations. Some form of network connection is assumed between the host and target machine(s) be they virtual or real network links. Of course, a remote execution model fits naturally into debugging applications for both cloud and cluster environments. Secure connections are supported, which opens up interesting possibilities for remote application debugging and profiling at customer sites.

Rather than relying on specific hardware provided debugging capabilities, Parallel Nsight takes the different approach of generating code that is patched into the executable. The benefit is that only those debugging and performance tracing capabilities requested by the user are introduced when and where desired. This actually is an extremely flexible model that can minimally impact the application(s) running on the monitor machine. Future capabilities can be added by the Parallel Nsight team as needed and arbitrary hardware capabilities (e.g. new counters and signals) can be exploited in newer generations of hardware to reduce overhead and provide greater insight into application behavior. Basically, Parallel Nsight is designed to give developers and hardware designers the flexibility needed to best meet future needs -- however unanticipated they might be at the current time.

As is to be expected, there is a fair amount of integration with the Visual Studio GUI. This is nice as the mouse can be used to click sections of code to set or remove breakpoints. Mouse-overs work and the scroll wheel can be used to zoom in on application traces to see finer details.

Parallel Nsight provides three basic capabilities:

- Debugging CUDA kernels. Similar to cuda-gdb, Parallel Nsight lets you set breakpoints, examine variables on the GPU and check for memory errors. This video demonstrates the debugging capabilities in action.
- Tracing CUDA applications so the programmer can understand where and how the application is spending time be it in the operating system, calculating with CUDA, transferring data, or working on the host processors. Check out this video to get a sense of the tracing and analysis capabilities.
- Shader debugging. Parallel Nsight 1.0 supports debugging Direct3D HLSL shaders as in this video at this URL. (Note that debugging OpenGL shaders is not supported at this time.) To debug an HLSL shader, the user clicks on Start Graphics Debugging and then uses either the shaders toolwindow to open a shader or use Pixel History to work backward from a render target to a particular shader. In other words, it is possible to see geometry for every draw call, watch the frame getting built up, and review the pixel history by clicking on a render target's pixel to show all the draw calls that touched that particular pixel! However, debugging shaders will not be discussed further in this article.

Of course, Visual Studio provides C/C++ debugging as well. Please note that CUDA debugging, shader debugging, and trace analysis are considered as separate operations. This means that Parallel Nsight 1.0 does not have the ability to have C/C++ breakpoints and CUDA C/C++ breakpoints both hit at the same time. Instead, the debugging session must be redone using either Parallel Nsight or Visual Studio. (A helpful post in the NVIDIA forums can provide some of this mixed breakpoint functionality.)

Information Sources about Parallel Nsight

Following are several excellent sources of information about Parallel Nsight:

- The NVIDIA Parallel Nsight forum is an excellent place to look for information and post questions.
- The Parallel Nsight Developer Zone is the main entry point for Parallel Nsight on the NVIDIA site. It provides links to documentation, videos and webinars.

In addition to Internet resources, Parallel Nsight installs a wealth of documentation into Visual Studio. (An online version of the installed user guide can be found here.) The release notes in particular are important as the Parallel Nsight team is rapidly improving the software and adding capabilities. First time users will find the example projects included in the host install to be the quickest way to start working with Parallel Nsight without requiring any project configuration. Please note that under Windows 7, the C:\ProgramData directory that contains the walkthrough projects is a hidden directory. The Visual Studio document, "Walkthrough: Debugging A CUDA Application" contains more information about using the Parallel Nsight installed samples. It is accessed via the Start Page | Getting Started window | How Do I…? | NVIDIA GPU Development | Parallel Nsight User Guide | Existing Projects and CUDA C Debugging | Walkthroughs.

Starting Microsoft Visual Studio 2008 after installing Parallel Nsight will show a start page similar to the following screenshot. Highlighted in Figure 1 are three key areas that first-time Parallel Nsight users need to be aware of: How do I …?, The Solution Explorer window, and the Nsight tab on the toolbar.

[Click image to view at full size]

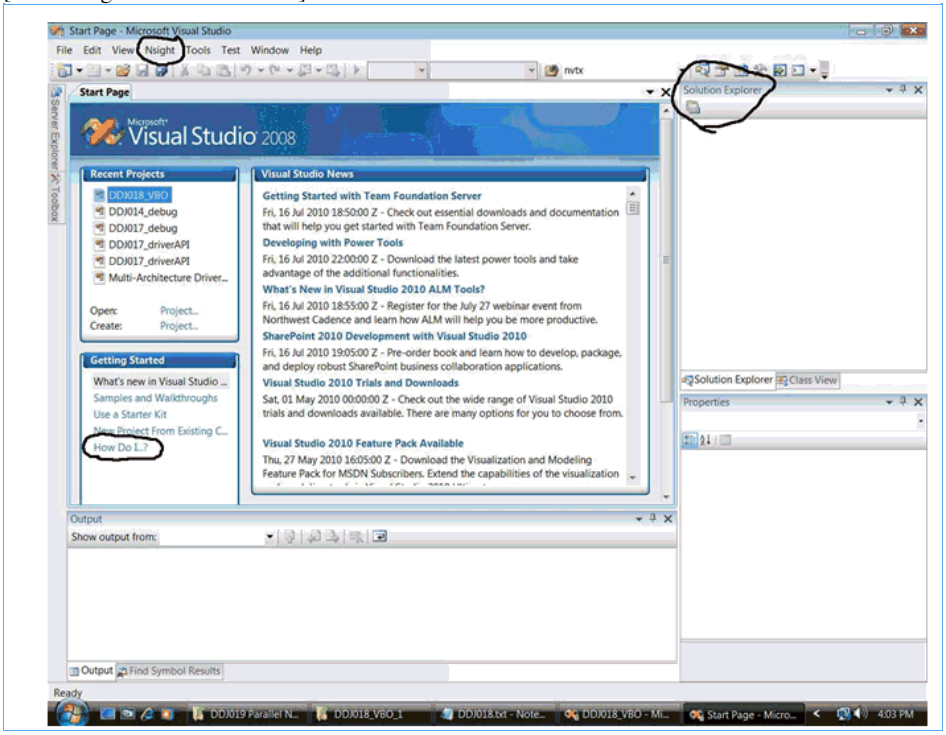

Figure 1

Clicking on the How do I…? link and expanding the Contents window, lets users browse the documents provided by the Parallel Nsight team as can be seen in Figure 2. Documents include walkthroughs based on some projects that were installed along with the documentation. Be aware that the name or address of the target machine must be specified before these walkthroughs can be used. This article, along with the Parallel Nsight documents, describes how to set the remote machine name. The How to Set Build Options link is an extremely important document as Parallel Nsight v1.0 requires the user manually configure many of the options needed to build, package, and run the application on the remote machine.

[Click image to view at full size]

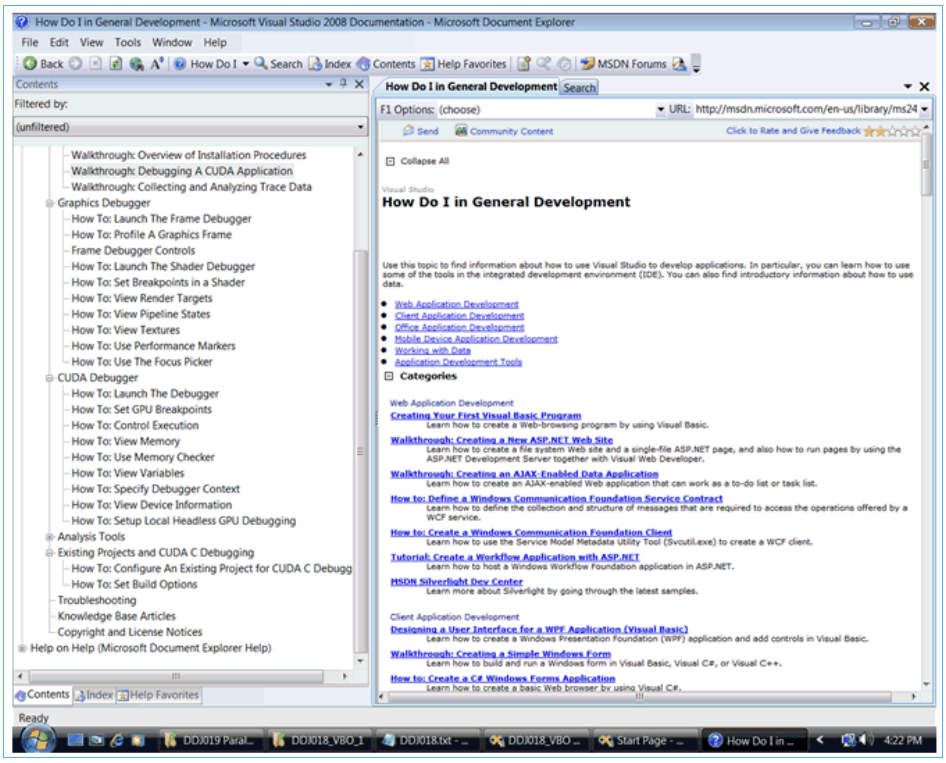

Figure 2

Installing Parallel Nsight

Parallel Nsight can be downloaded from its Developer Zone homepage.

Installing Parallel Nsight is fairly straightforward as described in the How To: … sections of the online documentation. Basically, the user downloads a host package and runs the install. Similarly, the package for the target machine is downloaded and installed. The user can select between 32- or 64-bit downloads so long as the installations are consistent between the host and target machines. Note: It is possible to create and debug 32-bit applications using 64-bit installations.

Version 1.0 of Parallel Nsight has explicit software requirements:

- Both host and target machines must be running Vista, Windows 7 or Windows Server 2008 operating systems. Windows XP will not work.
- Parallel Nsight will only work with Visual Studio 2008 service pack 1 or better. Parallel Nsight will not work with Visual Studio 2005 or 2010. The express version of Visual Studio 2008 does not support the plug-in model required by Parallel Nsight, so it cannot be used with Parallel Nsight.

Since Parallel Nsight is based on a remote debugging model that utilizes both a host and at least one target machine, some thought needs to be given to how the software will be installed and used.

By far the simplest and best performing configuration is to use Parallel Nsight on a dual-monitor workstation that supports a virtual machine and a graphics card like a Quadro FX 5800 that allows each operating system to directly access the GPU. With this configuration, Visual Studio builds occur on a powerful workstation, communications with the monitor process happen quickly over a virtual network, and accessing the target machine is as easy as looking at the second screen or moving the mouse seamlessly onto the target virtual machine to change application behavior.

Many users will actually utilize two separate machines, one for the host and one for the target machine. The host system must run Visual Studio so it should be powerful enough to perform the software build quickly and not bottleneck the development process. Smaller projects can probably be built on a laptop but larger projects will likely require a higher performance workstation. Similarly, the target machine should provide the capabilities envisioned for a typical application runtime environment.

Communication between host and target machines via a WiFi network link will work, but note that starting the application requires copying the executable plus any needed dlls from the host to the target machine. Some applications might also require moving data and configuration files as well. Parallel Nsight performs an incremental data move so only those files that have changed will be moved from the host to the target machine. Heavy bandwidth consumption can occur when the project is first sent to the target machine or when the entire project is rebuilt. Modified files are identified via an rsync-like mechanism to help reduce bandwidth consumption after the initial target machine setup.

Similarly, the network connecting the machines should allow the monitor process to return large amounts of data to the host in a timely fashion, which can happen when performing device queries and application trace operations. Version 1.0 of Parallel Nsight only supports tracing that begins when the application program starts and completes when the application either exits or is killed. There is no way to set software triggers to begin and end the tracing activity so very large amounts of data can be returned to the host -- especially when it takes awhile to get to interesting application behavior or when the voluminous Trace All option is selected. Host machine disk storage requirements can also be impacted.

It is important to note that the target machine cannot be controlled with Remote Desktop. Remote Desktop appears to be a natural way to work with the target

computer, but it affects how applications can interact with the GPU. While the monitor program can be started under Remote Desktop, it will not be able to access the GPU.

If Parallel Nsight is to be used with applications that require user input or viewing of output, then the host and monitor machines should be in close physical proximity or accessible via a KVM. (A KVM is a switch that connects two or more computers to the same keyboard, mouse, and monitor.) Interactive graphics based applications in particular require direct access to the target machine. If the application does not require any user interactions (e.g. it performs a calculation and exits) then there is no need to have physical access to the target machine. For these applications, the monitor program can be started on target machine boot or remotely via some remote access software such as cygwin and sshd.

Configurations that utilize two or more GPUs within a single system are also possible, but they are limited to non-graphical CUDA debugging.

Table 1, taken from the developer zone hardware and software requirements page summarizes various hardware configurations along with the current software requirements.

[Click image to view at full size]

Hardware Configuration	Single GPU System	Dual GPU System	Two systems, each with GPU	Dual GPU System SLI MultiOS
CUDA C/C++ Parallel Debugger		✓	✓	✓
D3D Shader Debugger			✓	✓
D3D Graphics Inspector	✓	✓	✓	✓
Analyzer	✓	✓	✓	✓
Requirements				
Supported Tesla GPUs	C1060, S1070,C2050, C2070			2 x800 Quadro GPUs only
Supported Quadro GPUs	FX 3800, FX 4800, FX 5800 and others			
Supported GeForce GPUs	9 series or better			
Software Requirements	Windows 7, Windows Vista SP1, or Windows HPC Server 2008 R2 (32 or 64 bit) .NET framework 3.5 with Service Pack 1 Visual Studio 2008 Standard or better, with Visual Studio Service Pack 1 installed			

Table 1

Configuring Visual Studio and Parallel Nsight to Debug an Application

Most readers will want to create a new CUDA application to debug and trace their code with Parallel Nsight. For this reason, we now provide a simple example demonstrating how to create an entirely new project in Visual Studio that can build and debug a CUDA program. The source code from Part 14 that was use to demonstrate cuda-gdb will be built and debugged with Parallel Nsight.

At the moment there is no project wizard so all configuration of the build, environmental variables and remote application packaging must be done by hand. For this reason, accessing the properties menu in the Windows Solution window is important because that is where much of this work is performed. Once a project is configured, the Nsight tab will provide access to the functionality of the Parallel Nsight debugging and analysis capabilities.

There are two important things to note when configuring a Parallel Nsight solution for debugging and analysis:

- Plan on taking some time to correctly configure a Parallel Nsight/Visual Studio CUDA project. The Parallel Nsight team is working on a true CUDA project system for Visual Studio, which will make CUDA C/C++ a first-class citizen rather than a VC++ project compiling CUDA code. This will considerably ease the configuration and build process and make project setup much easier as it will require many fewer steps plus be significantly more robust against both human error and version changes. However, release 1.0 does not have this capability.
- Succinctly, package management for the target machine is done by hand. Since Parallel Nsight uses a remote execution model, building the executable is only part of the process required to use Parallel Nsight. The user is also responsible for aggregating everything needed so the application can be shipped to the remote machine and run correctly. DLLs in particular are a challenge as even simple applications can require access to a large number of disparate libraries.
    - Identifying the required dlls can require a significant effort on the part of the developer for even very simple applications. Third-party dlls present additional challenges as the user may choose to have more than one copy on their system, with each application using a different version. Similarly, 32- and 64-bit issues are a problem. Even the simple examples provided in this article series require gathering multiple libraries such as glut, glew, cudart and cutil from various system and Internet locations. Considerable time can be wasted by mistakenly copying a dll with the correct name but incorrect version or type. As a result, the solution will build but fail to run on the target machine.
        - Specification of the appropriate paths to the header files is also complicated by the library/dll challenges and is exacerbated by the need to separately define the paths for both the Visual Studio and nvcc compilers.
    - Once the dlls are found, Parallel Nsight is configured so the remote application will find them so long as they are in the same directory as the executable (or perhaps in the working directory), or in the system PATH. There are two ways to gather the dlls once they have been identified.

- A kludgy way to handle this issue is to hand copy the dlls to the project Debug directory once the project is created.
- A more elegant approach is to specify one or more commands in the Visual Studio project properties field Build Events | Post-Build Event | Command line to gather any needed dlls. Note: the commands specified in this field are executed after each successful solution build, which can adversely impact build time.

Following are the steps required to create a new Parallel Nsight project using the simple debugging example used in Part 14, which discussed debugging with cuda-gdb.

1. Start Visual Studio
2. Click on File | New | Win32 Console Application

    a. Specify the name as DDJ014_debug and click OK
    b. Click Finish
3. Right click on project in the Solution Explorer window and select properties
4. Add the following to Inherited Project Property Sheets
   $(VCInstallDir)VCProjectDefaults\NsightCudaToolkit.v31.vsprops
    1. Click the arrow next to Linker and select General
    2. Add the following to Additional Library Directives
       $(NSIGHT_CUDA_TOOLKIT)/lib/$(PlatformName)
    3. Click on Input
    4. Add the following to Additional Dependencies cudart.lib
    5. Click OK
    6. Click on Build Events
    7. Click on Post-Build Event
    8. Copy/paste the following into Command Line
       copy "$(NSIGHT_CUDA_TOOLKIT)\bin\cudart*_*.dll" "$(TargetDir)"
5. Copy the following source file (source page in DDJ014) and save to AssignScaleVectorWithError.cu
6. Right click on Source Files

    a. Add | Existing Item ...
    b. Select AssignScaleVectorWithError.cu in the DDJ014_debug directory
    c. Note: a Matching Custom Build Rules dialogue will appear a mouseover occurs over the names in the Rule File column
    d. Click to select the one that has NsightCudaRuntimeApi.v31.rules (or the latest version) in the name
    e. Click OK
7. Click on AssignScaleVectorWithError.cu and edit the source to change int main() to void myMain().
8. Click on DDJ014_debug.cpp and edit the source to add
    a. Add the forward declaration extern void myMain(); before int _tmain
    b. Add a call to myMain(); before the line return 0;
    c. The source should look like this

```
// DDJ014_debug.cpp : Defines the entry point for the console application.
//

#include "stdafx.h"

extern void myMain();

int _tmain(int argc, _TCHAR* argv[])
{
        myMain();
        return 0;
}
```

- Right click on project in the Solution Explorer window and select Nsight User Properties

a. Change localhost in Connection Name to the remote IP address (e.g. 10.37.130.3) or the name of the target machine that is running the Parallel Nsight monitor.

Note: It is important to verify that the compiler code generation flags are identical for both the C/C++ compiler and nvcc otherwise link errors can occur.

- Verify that Property | C/C++ | Code Generation | Runtime Library is set to /Mtd.
- Right-click on a .cu file and select Properties | Parallel Nsight | Host | Runtime Library and verify the flag is set to /MTd.

Build the project: CTRL + Shift + B

If you wish, you can create a 64-bit version. The copy command for the post-build processing copies both 32- and 64-bit versions of cudart.lib for the target machine to use.

- Click the dropdown next to Win32

a. Click Configuration Manager
b. Click the drop-down next to Win32

c. Click New
d. Select x64 in New Platform
e. Click OK
f. Click Close

- 10. Right click on project in the Solution Explorer window and select properties

a. Click on Host
b. Select Target Machine Platform and from the dropdown menu select x64
c. Click OK

- On the top toolbar menu

a. Build | Build Clean
b. Build | Build

Click on the filename AssignScaleVectorWithError.cu and set a breakpoint next to the first line in `scale(float *v_d, int n, float scaleFactor)`. The break point can be set by clicking in the gray area next to the source line. A red breakpoint circle will appear in the gray area next to the source line. (Note: one way to remove a breakpoint is to click on the red circle again.)

Start Parallel Nsight by clicking Nsight | Start CUDA Debugging and a console window will start on the target machine. Meanwhile a yellow arrow will appear in the red breakpoint circle next to the source line as in Figure 3.

[Click image to view at full size]

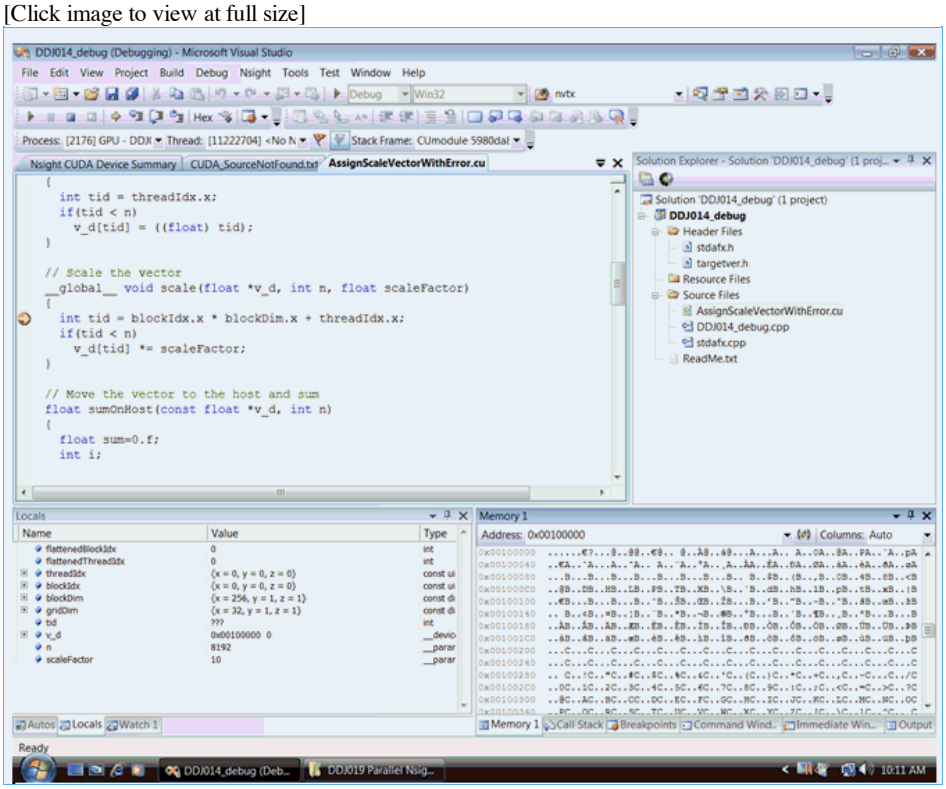

Figure 3

The variable `v_d` can be typed or dragged to the Address field in the Memory 1 window at the bottom lower side. This will specify the memory address to examine. The user can specify that `v_d` is a floating-point array by right clicking in the Memory 1 window and selecting 32-bit Floating Point as the type. Scrolling down shows that `v_d` is filled with zeros after the value of 255. This indicates there is an error in `assign()`, which should fill `v_d` with consecutive integers from 0 to n-1. As can be seen in the Locals window, n is 8192. Per the discussion in Part 14, the calculation of tid in `assign()` is incorrect and must be changed to the form shown in `scale()` for the program to run correctly.

[Click image to view at full size]

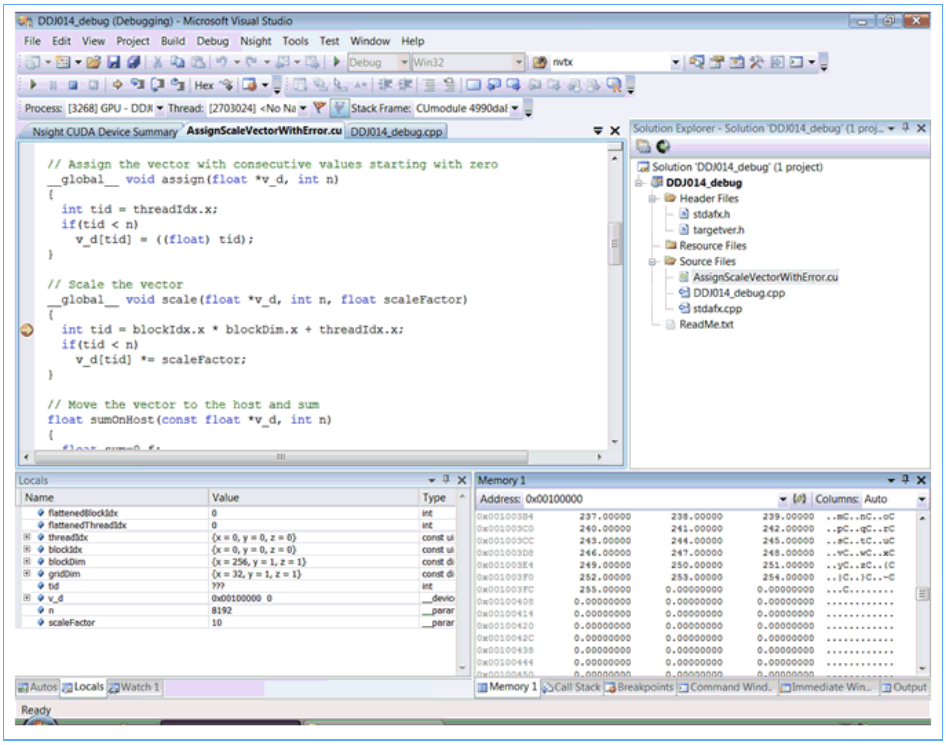

Figure 4

Pressing F5 (or clicking Debug | Continue top toolbar) highlights two items to note in the v1.0 release:

- Parallel Nsight version 1.0 does not support console redirection of text from the target machine. Thus the "TEST FAILED" message printed by the application is not accessible from Visual Studio or visible to the user on the target console because the window closes so quickly.
- It is not possible in this example to set a functional breakpoint on the `printf()` statements in the host side code because AssignScaleVectorWithError.cu was compiled with nvcc.

Debug Focus, Conditional Breakpoints and Memory Checking

Setting a break point in the assignment of `v_d` in the method `assign()` in Figure 5 lets us see how Parallel Nsight Debug Focus works. Pressing F11 moves the yellow arrow indicating program execution to the close curly bracket. The display of `v_d` in the Memory 1 window is also updated. Note that all the values from 1-31 are highlighted in red indicating that this warp changed those values in `v_d`. The value at index zero is not highlighted in red as it did not change value. It already contained the value of zero.

[Click image to view at full size]

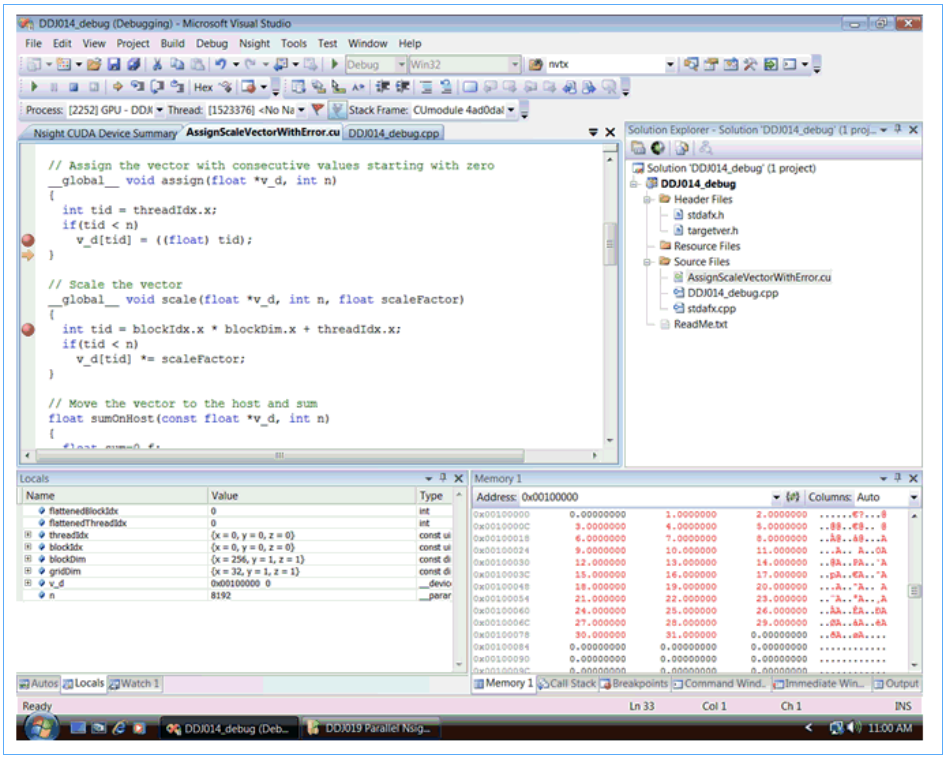

Figure 5

Notice that the values after 31 are still zero. This happens because Parallel Nsight has not executed the warp that assigns locations 32-63.

Two important characteristics to note about breakpoints:

- Program execution stops when any warp hits the breakpoint. As this example demonstrates, a partial result might be displayed as the other threads might not yet completed their work.
- Any warp can trigger the breakpoint. Do not assume that the Block 0(0,0,0) Warp 0(0,0,0) will always trigger the breakpoint.

Clicking on Nsight | Device Summary allows is to select whatever block and warp desired on the GPU. (To quickly make a selection in large grids, click on Nsight | Debug Focus … and type in the block and thread.) As in Figure 5, Warp 0(0,0,0) is highlighted.

[Click image to view at full size]

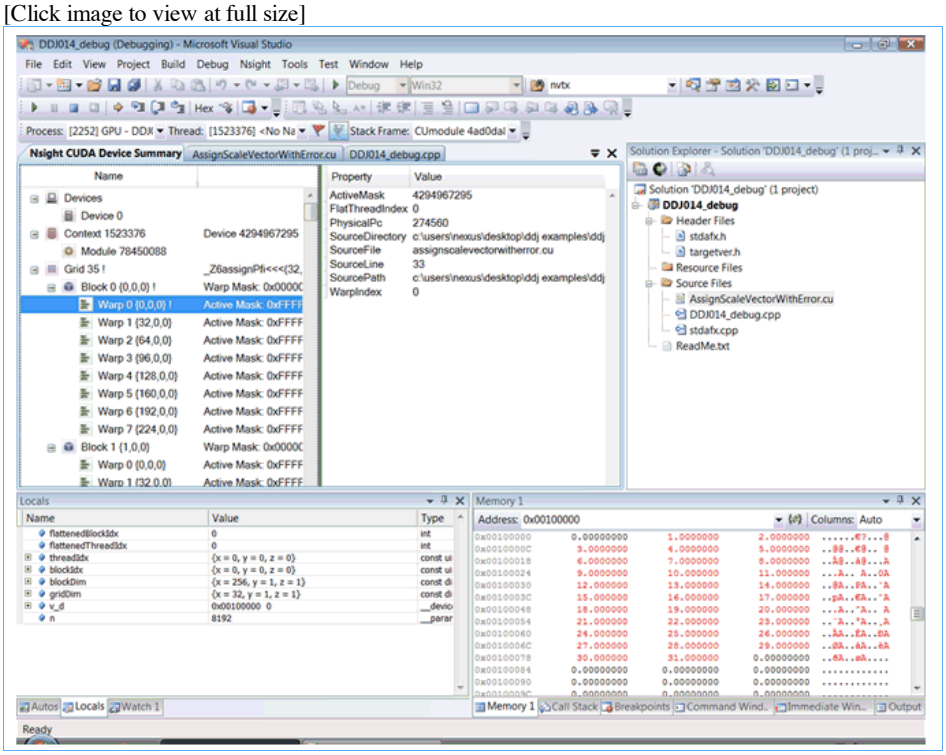

Figure 6

Clicking on the following line, Warp 1 (32,0,0) and double-clicking on AssignScaleVectorWithError.cu in the Solution Explorer window shows that the yellow arrow is still in the breakpoint circle. This indicates that warp 1 in this block has not run. Pressing F11 advances the arrow and shows the values from 32-62 are updated. Selecting any warp in any block aside from block 0 shows that tid never exceeds 255, which confirms the incorrect assignment of tid.

It is also possible to set conditional breakpoints on the predefined CUDA variables like `blockIdx` and `threadIdx`. Parallel Nsight 1.0 does not support conditional breakpoints on data, but it does support setting breakpoints on a memory address.

To set a conditional breakpoint:

- Click in the gray area next to the source line to set a breakpoint.
- Right-click and select Condition … from the menu that appears.
- Type in the condition. A plus ('+') will be added to the breakpoint circle.

In Figure 7, the condition `@blockidx(10,0,0)` is required to be true. The Parallel Nsight CUDA debugging was started and F11 was pressed to advance the execution by the warp. As can be seen, tid is incorrectly set to zero. Pressing F5 to continue shows that `scale()` correctly calculates tid as 2560.

[Click image to view at full size]

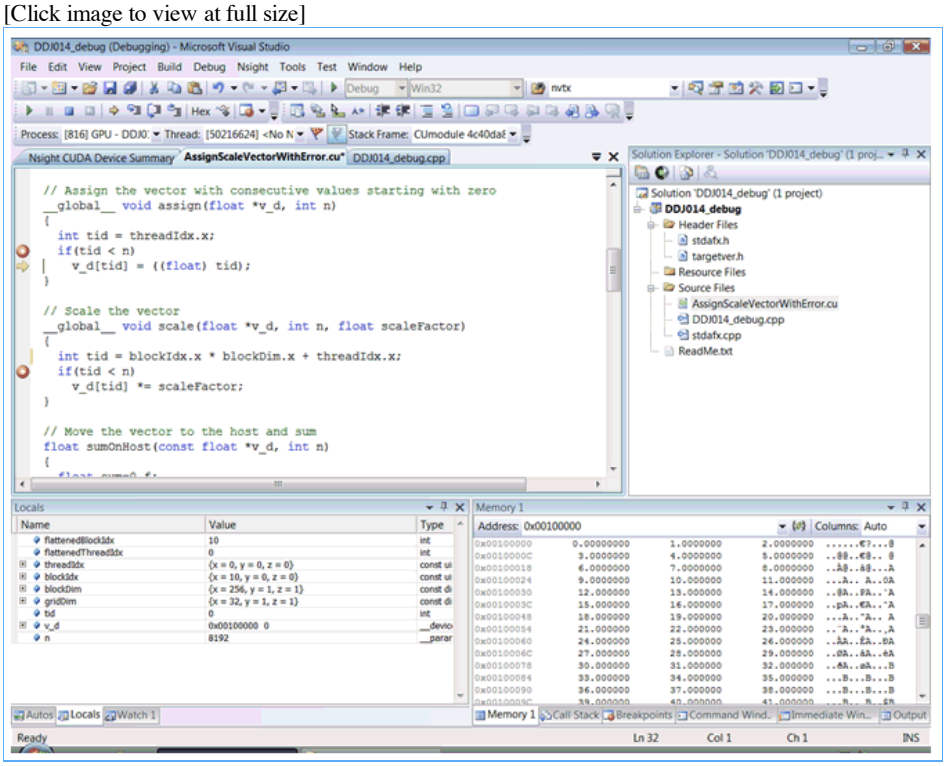

Figure 7

Just like the Linux cuda-gdb debugger, Parallel Nsight also supports out-of-bounds memory checking. Multiplying tid by a 1000 as shown in the code snippet below purposely causes an out-of-bounds condition that is caught by Parallel Nsight with memory checking enabled.

```
// Scale the vector
__global__ void scale(float *v_d, int n, float scaleFactor)
{
  int tid = blockIdx.x * blockDim.x + threadIdx.x;
  if(tid < n)
    v_d[1000*tid] *= scaleFactor;
}
```

Figure 8 illustrates how Parallel Nsight flags the out-of-bounds memory errors.

[Click image to view at full size]

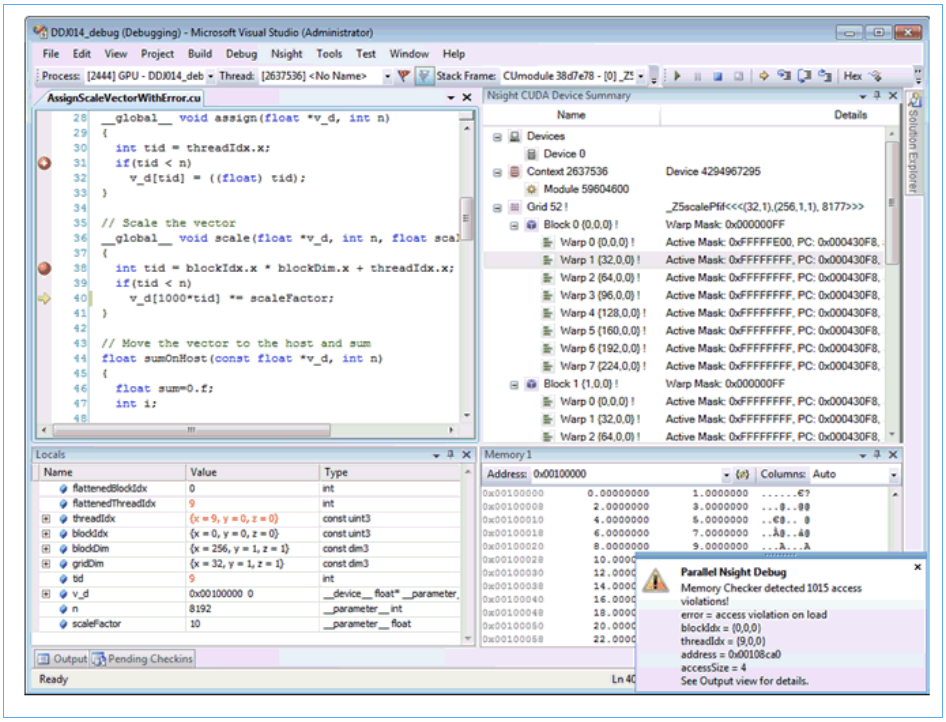

Figure 8

The pop-up window in the lower right of the screen (Figure 9) shows that the memory checker found access violations.

[Click image to view at full size]

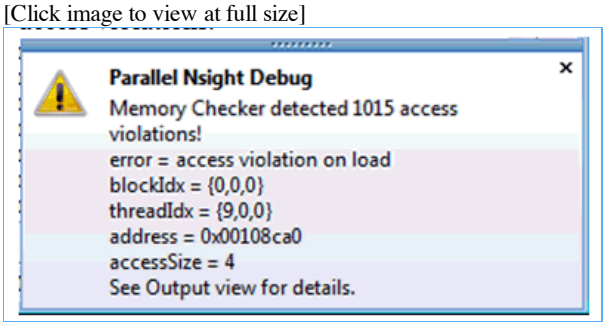

Figure 9

Summary

With version 1.0 of Parallel Nsight, NVIDIA has demonstrated a significant commitment to Visual Studio developers around the world. The thought process behind Parallel Nsight is clearly in-place to create a powerful and flexible product to support developers for many years to come. As with other NVIDIA software, it is expected that Parallel Nsight will evolve quickly in the near future to ease project creation, remote machine package management, and improve the user debugging experience by providing a more unified debugging experience. Look to the release notes to stay abreast of the latest developments. Still, the current 1.0 release is quite useful. In particular, the large amount of information that is displayed in an interactive fashion can speed debugging efforts.

The next article in this series will focus on using the trace capabilities provided by Parallel Nsight, which allow developers to gain "Nsight" into what is happening on host and GPU processing systems as well as pinpoint exactly where time spent within the various APIs. Support for tracing OpenCL applications within Parallel Nsight will also be covered.

References

- CUDA, Supercomputing for the Masses: Part 18
- CUDA, Supercomputing for the Masses: Part 17
- CUDA, Supercomputing for the Masses: Part 16
- CUDA, Supercomputing for the Masses: Part 15
- CUDA, Supercomputing for the Masses: Part 14
- CUDA, Supercomputing for the Masses: Part 13
- CUDA, Supercomputing for the Masses: Part 12
- CUDA, Supercomputing for the Masses: Part 11
- CUDA, Supercomputing for the Masses: Part 10
- CUDA, Supercomputing for the Masses: Part 9
- CUDA, Supercomputing for the Masses: Part 8

- CUDA, Supercomputing for the Masses: Part 7
- CUDA, Supercomputing for the Masses: Part 6
- CUDA, Supercomputing for the Masses: Part 5
- CUDA, Supercomputing for the Masses: Part 4
- CUDA, Supercomputing for the Masses: Part 3
- CUDA, Supercomputing for the Masses: Part 2
- CUDA, Supercomputing for the Masses: Part 1

Rob Farber is a senior scientist at Pacific Northwest National Laboratory. He has worked in massively parallel computing at several national laboratories and as co-founder of several startups. He can be reached at rmfarber@gmail.com