CUDA to SZG, or a Tale of Three Programs

A. Dakkak, W. Davis, C. Boren[1]

Introduction & Purpose:

The primary purpose of our endeavours was a proof of concept for a joint CUDA/SZG program. CUDA is the C programming language extension that allows programs or parts of programs to be run on an Nvidia graphics processing unit. Due to the computing needs of intensive graphics, the GPU (which is Single Instruction, Multi Data or SIMD) has become very capable of parallelizing programs. In short, it allows us to do many simple/straightforward calculations fast. These include solving the Navier-Stokes equations found in FluidsGL. Syzygy (or SZG), on the other side of our interface, is a framework for allowing OpenGL and Python programs to be run in the 3 dimensional virtual environments at the CAVE and CUBE at the Beckman Institute. SZG is more than capable of handling the OpenGL portion of a program, but it can struggle with heavy computation due to its distributed nature. This provides the motivation for our interface.

Method:

We decided to use the program FluidsGL[2] as our model due to the previous work done on it by Jared Schaber in the spring. His work included abstracting the computation from the display portion[3] and then piping it over a socket. All that was left was to port to a Windows32/SZG environment. Unfortunately, Schaber used BSD sockets and other n*x functions that prohibited an easy port. At this juncture there were two efforts: 1)A proper method lead by Will Davis and 2) a hacking, nasty method by Chase Boren.

The issue that caused the division was the way Windows Sockets work. Until Windows 2003 SDK, when the Windows recv function is called, the program opens the socket and writes data to a buffer for a short amount of time. It does not fill up the buffer before returning, which is a native BSD sockets option. Schaber uses this option in his program which allowed the array of particles that is FluidsGL to be shipped properly. After 2003 this option was enabled in Windows, so for our purposes, it does exist in the Windows API.

Davis' method involved the option on recv that returns the number of bytes received. We could print out the bytes sent on the server, so we would run it in a loop, updating the array and calling recv again. This worked, although we received strange artifacts on the top and bottom edges. When we ran this version in standalone mode, it worked acceptably well and also in standalone mode when logged in to the cluster, the distributed computing network environment of SZG. However,

[1]All baREUcuda, illiMath REU 2008, Abdul Dakkak PI, Professor George Francis Director

[2]The program is included in the Nvidia-Cuda SDK

[3]A process now known as Schaberization

running it in a distributed exchange mode (which is how it would normally run in immersive environments) caused the program to freeze.

Boren attempted to use the Windows API with the option of filling the recv buffer before returning. We were using the MinGW build environment, which is somewhat outdated and therefore does not utilize the full API. The Cygwin build system does. We replaced all the MinGW libraries with Cygwin libraries and it worked on Windows Vista and Windows XP Professional x64. However, the program ran slowly on XP for no apparent reason.

At this point a side note becomes necessary. Both methods used the exact same program with the exception of the sockets implementation. The size of the array was 128 X 128 or $2^7 * 2^7$. When CUDA calculates two to an odd power, there can be artifacts due to integer truncation in the machine. For example 2^3 is truncated to 7 instead of 8 sometimes if the result is assigned to an integer. So by changing our dimensions to, for example, 140 X 140 we eliminated the slowness in one method and the artifacts in the other. There was no change in compatibility or SZG performance.

In an effort to find a reason for the poor SZG performance, we came across the SZG's so-called portability layer. It provides an abstraction of sockets for SZG, so we started pursuing this venue. The portability layer is poorly documented but we obtained SZGSalamiMan[4], which has an example of SZG sockets in it. It natively allows us to fill the buffer before returning, besides working in SZG, so it fit our needs. We could not, though, compile SZG code into our server because it uses the special CUDA compiler. To reconcile BSD Sockets with the SZG sockets interface, we constructed a relay. It is very small program that basically passes data between the server and the client by communicating via BSD sockets on the server side and SZG sockets on the client side. The first try worked in standalone and when logged in to the Phleet but again failed in distributed exchange. This was due to the fact that the master and the slaves were all simultaneously requesting a return handshake from the relay. Under the guidance of Jim Crowell, the socket binding subroutine was changed to only be called by the master in the appropriate callback and the program worked in all environments.

Usage

At present we are tied to the GNU/Linux Sun machine (the sun) in the lab in Altgeld Hall room 102. On the sun, the CUDA server and the Relay must be started separately. You need two shells, either through ssh or otherwise. In the first shell the following command needs to be executed(assuming home is cfgauss home directory):

```
 % ~/NVIDIA_CUDA_SDK/bin/release/linux/fluidzDy 8675
```

and then in the other shell:

[4]Courtesy of Jim Crowell

```
%bash
$source ~/Desktop/cboren2/szg/zmaths/szgenv
$fluidsRelay 127.0.0.1 8675 5309
```

[5][6] Then on the console in the CAVE:

```
$dex vcbox fluidzDy 130.126.108.215 5309
```

In the CAVE you can use the controller and button one (or zero depending on who is counting) and you can disturb the fluid. It will then behave according to a slightly modified Navier-Stokes equation.

[5]sun uses tcsh and we need bash

[6]The port numbers are a reference to the 1982 song *Jenny* by Tommy Tutone