

CS 264 - Lab 2

Richard Edgar

Lab 2: Using Textures

- * Goal of this lab is familiarisation with CUDA textures
- * Textures are cached on multiprocessors
- * Can be useful as look up tables

Skeleton Code

- * **Download code from**
<http://cs264.org/files/cs264-lab2.tgz>
- * **Unpack it and go into the**
LinearTexture/src
directory
- * **Edit the Makefile to point at nvcc**

Skeleton Code

- * The code
 - * Creates a look up table
 - * Uses this table to fill in array of other values

Skeleton Code

- * When you run the code, get two output files
 - * lookup.out (the look up table used)
 - * interpolate.out (interpolated values)
- * Compile and run the code

Running the Code

```
[rge21@chimera src]$ ../bin/LinearTextures
Texturing from Linear Arrays
=====

Enter number of values in the lookup table
5
Enter first and last values in the lookup table
-2 2
Enter number of polynomial coefficients
3
Enter polynomial coefficients
First coefficient is that on the zeroth power
-1 0 1
```

$$-1 + 0x + x^2$$

lookup.out

x	f(x)
-2	3
-1	0
0	-1
1	0
2	3

Running the Code

```
Enter number of values in output table  
21  
Enter start and stop values for output table  
-3 3  
[rge21@chimera src]$
```

**Produce 21 interpolated
values for x in range [-3,3]**

**Result stored in interpolate.out
Also includes exact value of the polynomial**

Using Textures

- * Textures have to be
 - * Set up
 - * Used
 - * Released
- * File 'lineartexture.cu' provides skeleton functions for these

Creating the Texture

- * Textures declared at file scope

```
texture<float, 1, cudaReadModeElementType> dtl_f;
```

Has to be accessible to device & host

- * We will also need an array of floats to hold the actual values

Creating the Texture

- * In the routine "Texture Initialise"
 - * Allocate the look up table on the device
 - * Copy the values over
- * Next, we must associate the array with the texture

Creating the Texture

- * **We need a channel descriptor**

```
cudaChannelFormatDesc cd_f =  
cudaCreateChannelDesc<float>();
```

- * **We also need to set the addressing mode**

```
dtl_f.normalized = false;  
dtl_f.addressMode[0] = cudaAddressModeClamp;  
dtl_f.filterMode = cudaFilterModePoint;
```

- * **Finally, bind the texture**

```
cudaBindTexture( 0, dtl_f, <your array>, cd_f,  
nVals*sizeof(float) );
```

Releasing the texture

- * Skeleton routine 'TextureFinalise'
- * Start by unbinding the texture
`cudaUnbindTexture(dtl_f);`
- * Can then free your array

Using the Texture

- * CPU routine 'TextureInterpolate'
- * Kernel 'DoLookup'
- * Aim is to get an interpolated value $fs[i]$ for every $xs[i]$ supplied

Using the Texture

- * CPU routine has to
 - * Put input xs on the device
 - * Run the kernel to do the interpolation
 - * Retrieve the fs from the device
- * Write this yourself

Using the texture

- * Kernel has to use the texture
- * Fill in the appropriate f value
- * Use

```
myf = tex1Dfetch( dt1_f, myx);
```
- * Look at your results

What went wrong?

- * Your results were incorrect
- * Problem is that you didn't know how texture was indexed
- * Only passed in the f values, not corresponding x values

Texture Indexing

- * We are using textures indexed by floating point values
- * $f[0]$ is assumed to lie at $x = 0.5$
 $f[1]$ is assumed to lie at $x = 1.5$
 $f[n-1]$ is assumed to lie at $x = n - 0.5$
- * This means we have to adjust our call to `tex1Dfetch`

Texture Indexing

- * To get the right value we need
 $(n-1)(myx-xFirst) / xRange + 0.5f$
in our call to `tex1Dfetch`
- * Have to set up `n`, `xFirst` and `xRange` in `TextureInitialise` routine
- * Use device constants

Limitations

- * We set up `addressMode` and `filterMode`
- * These aren't useful for linear textures
- * To activate them, use `cudaArrays`

Cuda Arrays

- * Cuda Arrays are opaque objects
- * Can't access directly
- * Look at TextureCudaArray skeleton

Cuda Arrays

- * Want to change code to use Cuda Array instead of normal array

- * Start with declaration

```
texture<float, 1, cudaReadModeElementType> dt_f;  
static cudaArray* dca_f;
```

Initialising the Array

- * Slightly different to normal array

```
cudaChannelFormatDesc cd_f = cudaCreateChannelDesc<float>();  
cudaMallocArray( &dca_f, &cd_f, nVals, 1);  
cudaMemcpyToArray( dca_f, 0, 0, f, nVals*sizeof(float),  
                  cudaMemcpyHostToDevice);
```

- * Can still see standard structure

Binding the Texture

* Similar to before

```
dt_f.normalized = false;  
dt_f.addressMode[0] = cudaAddressModeClamp;  
dt_f.filterMode = cudaFilterModeLinear;  
  
cudaBindTextureToArray( dt_f, dca_f, cd_f );
```

Releasing

* Slightly different call

```
cudaUnbindTexture( dt_f );  
cudaFreeArray( dca_f );
```

Using the Texture

- * Have to use `tex1D`, not `tex1Dfetch`

- * Otherwise identical

```
myf = tex1D( dt_f, myx );
```

Results

