GpuPy: Accelerating NumPy With a GPU

Washington State University
School of Electrical Engineering and Computer Science

Benjamin Eitzen - eitzenb@eecs.wsu.edu
Robert R. Lewis - bobl@tricity.wsu.edu

08.18.06

Snakes On A GPU

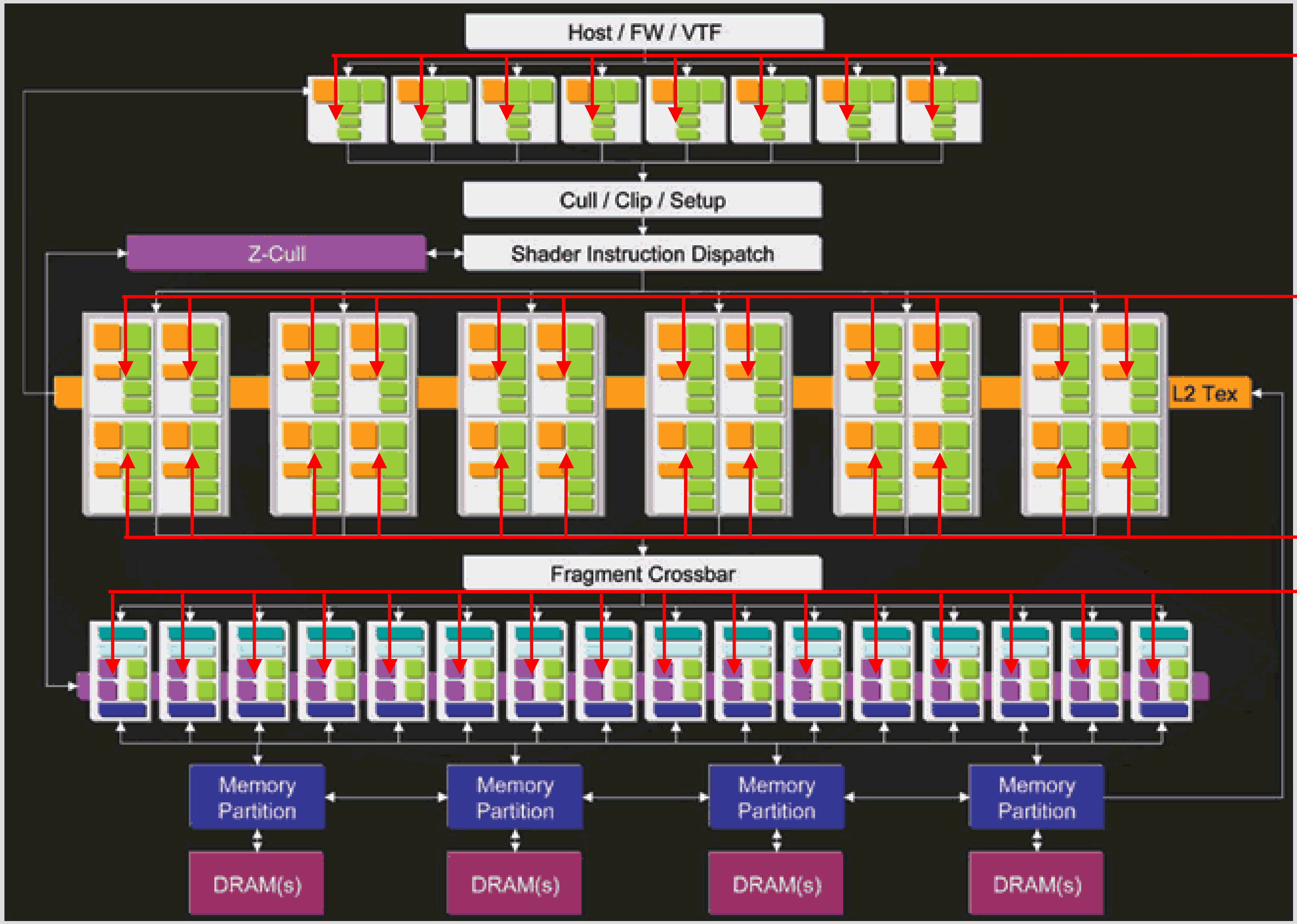
Presentation Outline

- What is a GPU?
- How can a GPU be used to do general purpose computations?
- How can NumPy take advantage of a GPU?

What Is a GPU?

- Stands for “Graphics Processing Unit”
- It is the main component on a video card
- Provides hardware acceleration for graphics APIs such as OpenGL and DirectX

GeForce 7800 GTX



GeForce 7800 GTX block diagram. Source: NVIDIA

8 programmable vertex shaders

24 programmable fragment shaders

16 raster units (non-programmable)

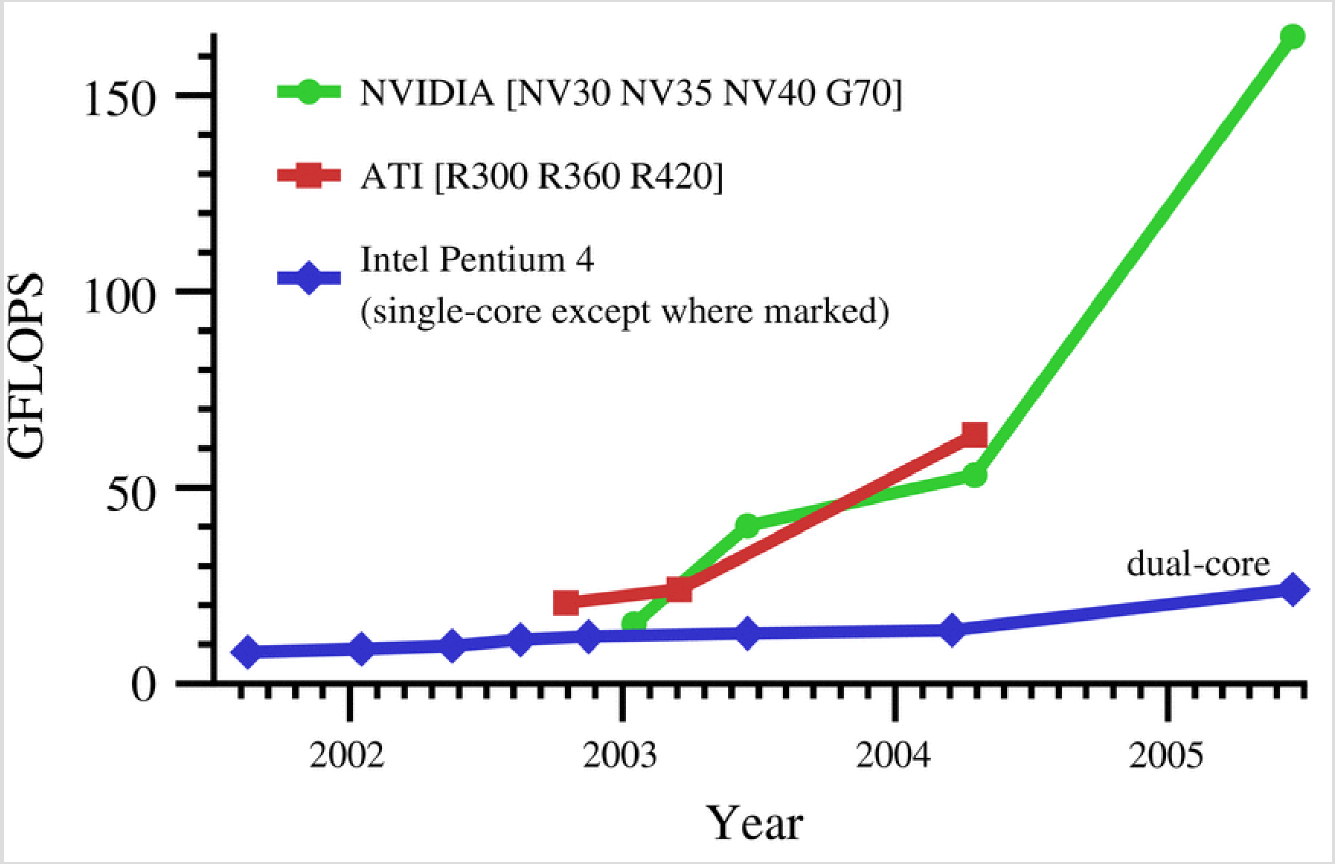
Strengths of a GPU

- Large capacity for floating point calculations
- Dedicated hardware for many graphics-related operations (linear algebra, trigonometry, etc.)
- Highly parallel
- Improving more rapidly than traditional CPUs
- Programmable

Weaknesses of a GPU

- Only single-precision floating point values are supported.
- Data must be copied to the GPU's memory before calculations can be done.
- Programming is conceptually more difficult than on a traditional CPU.

GPU vs. CPU



Source: SIGGRAPH 2005 GPGPU Course

GPU Programmability

- How can a GPU be used to do general purpose computations?
- Newer GPUs can directly execute programs written in high-level languages (Cg, GLSL, HLSL) or in assembler.
- Programs are called shaders and will seem familiar to programmers who have used C and/or assembler.

How it works

- GPUs execute a program once for each pixel that is drawn to the screen.
- The coordinates of the pixel being drawn index an element of an array.
- By rendering a rectangle that fills the entire screen, the GPU can be made to execute a program once for each element in an array.

Data Types

- Most GPUs support 1- 2- 3- or 4-vectors of 32-bit floating point values.
- Some operators act in parallel on all elements of the vector, others act on single elements.
- Data is passed to the GPU in the form of a texture, which is essentially an array.

High-Level Code for Addition

Cg:

```
float4 main(float2 p : TEXCOORD0,
            uniform samplerRECT arg1,
            uniform samplerRECT arg2) : COLOR
{
   return texRECT(arg1, p) + texRECT(arg2, p);
}
```

GLSL:

```
uniform sampler2DRect arg1;
uniform sampler2DRect arg2;
void main(void)
{
   vec2 p = gl_TexCoord[0].xy;
   gl_FragColor =
       texture2DRect(arg1, p) + texture2DRect(arg2, p);
}
```

Assembler for Addition

- The two examples from the previous slide both produce the following assembler code.

arbfp1:

```
PARAM c[1] = { program.local[0] };
TEMP R0;
TEMP R1;
TEX R1, fragment.texcoord[0], texture[1], RECT;
TEX R0, fragment.texcoord[0], texture[0], RECT;
ADD result.color, R0, R1;
END
```

GpuPy

- GpuPy extends NumPy so that it can take advantage of a GPU.
- Many of the operations that NumPy provides can be performed efficiently on a GPU.

How it works

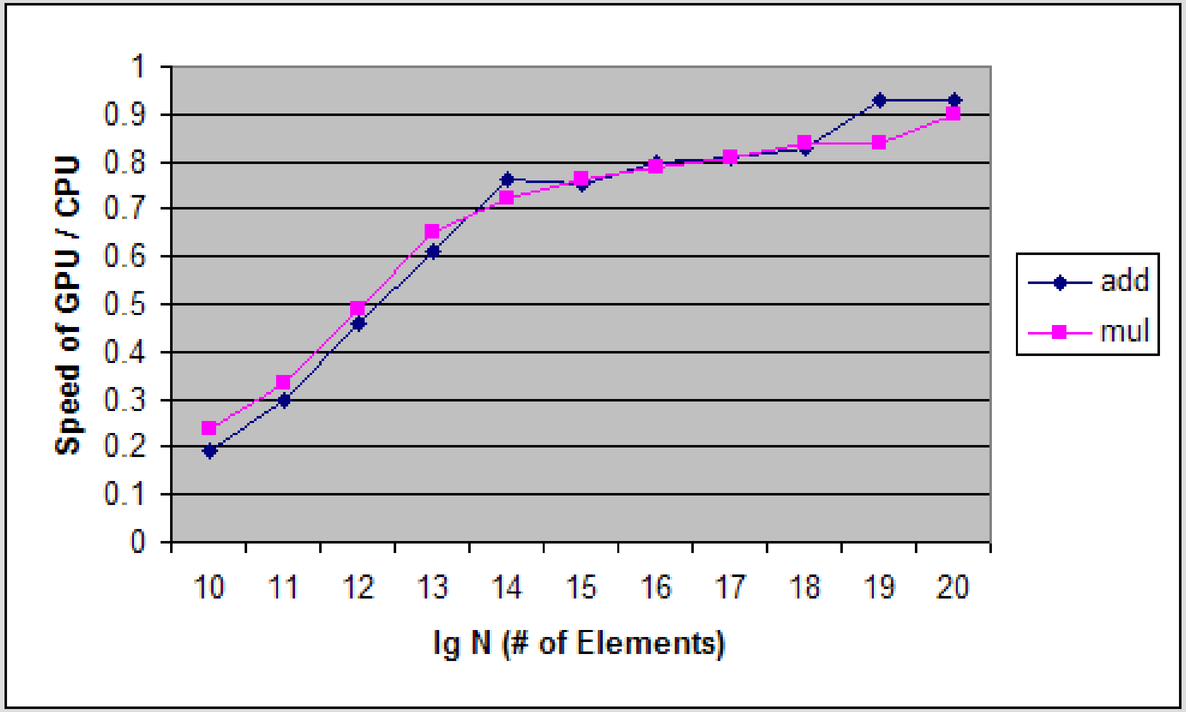
- GpuPy overrides NumPy's default implementations of Float32 operations with GPU-based implementations.
- GpuPy can be transparent, so if a GPU is present on the system where the script is being run, it will simply run faster.
- This means that existing scripts can take advantage of a GPU without any changes.

Drawbacks

- Currently, the GPU-based implementations only support 32-bit floating point values, so only Float32 and Complex64 can be accelerated.
- Simple operations aren't any faster than software (yet).

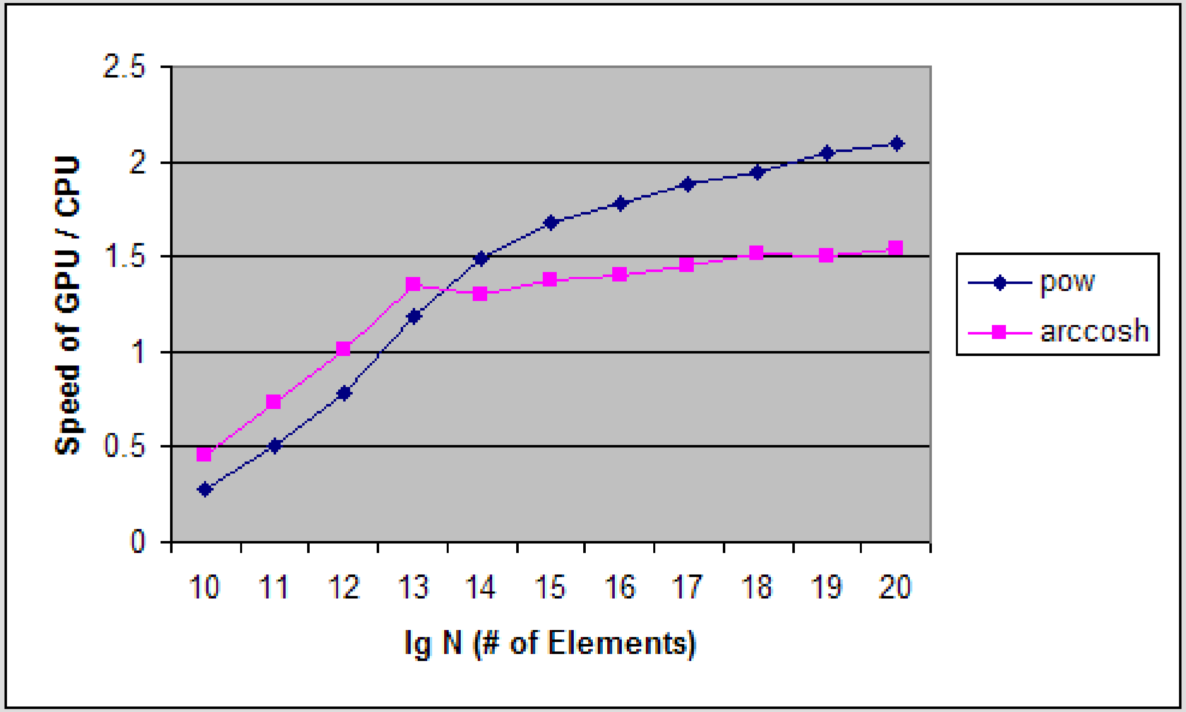
Performance Results 1

- Some operations are never faster on a GPU.



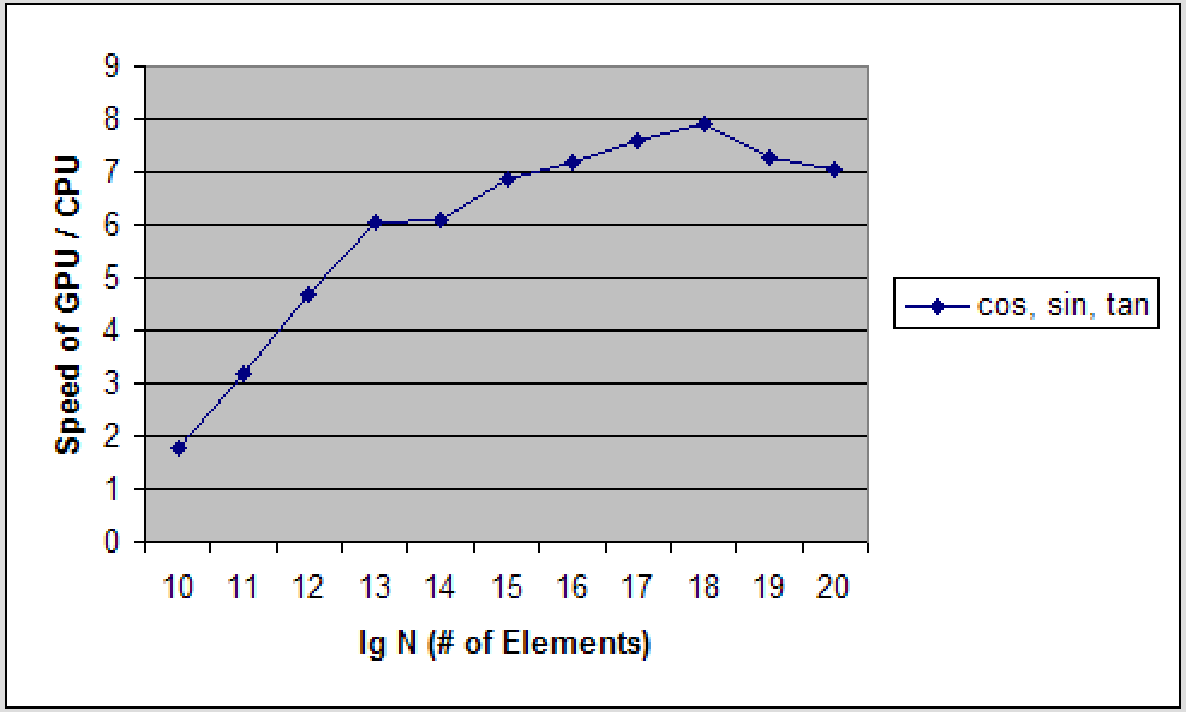
Performance Results 2

- Some operations are only faster on a GPU when the array size is large enough.



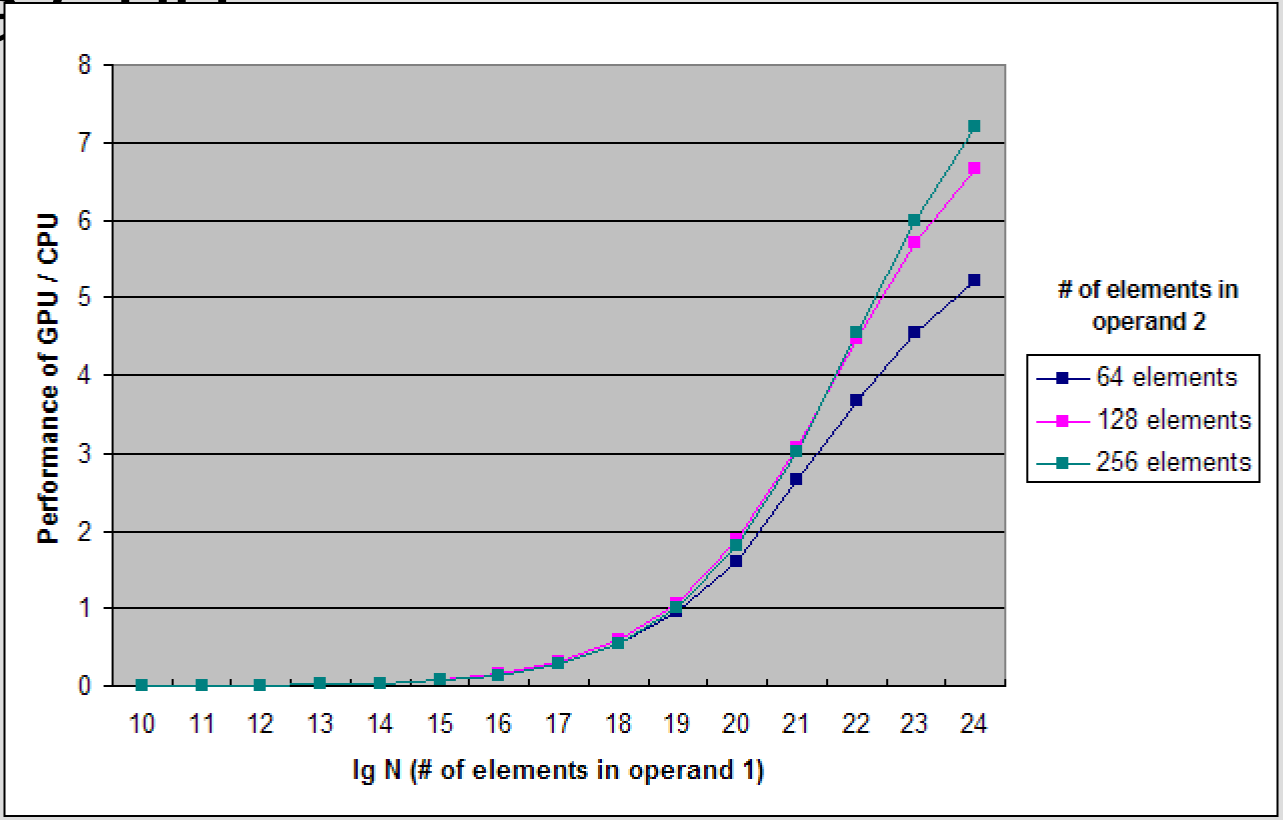
Performance Results 3

- Some operations are much faster on a GPU.



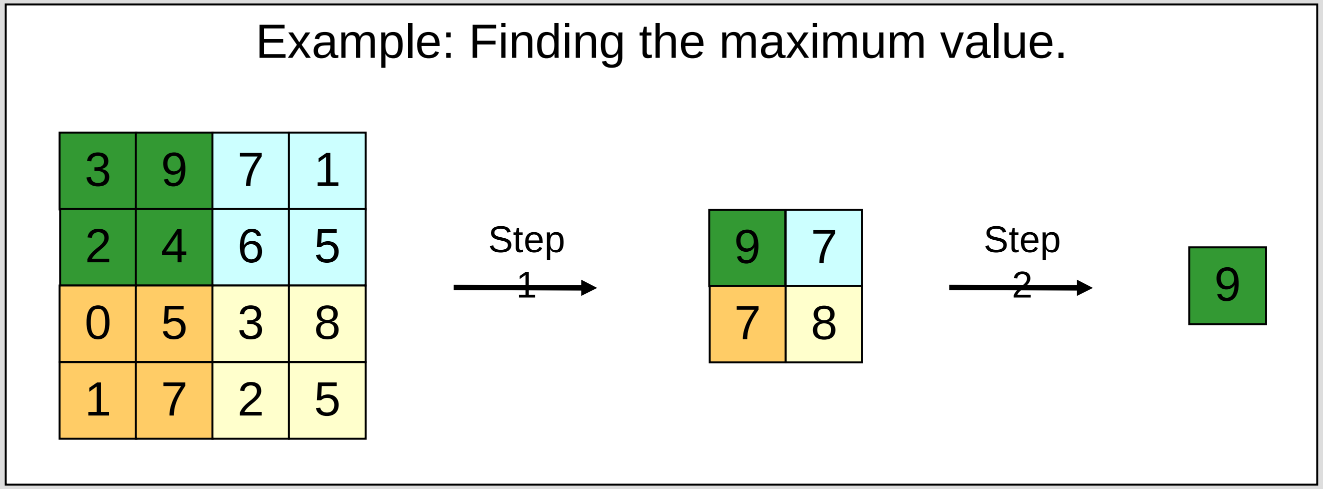
Cross Correlation

- Implementing NumPy's correlate function on a GPU



Reductions

- Reductions can be performed in parallel on the GPU.
- Reduction Requires O(lg N) steps.



More Complex Operations

- Convolution can be done on a GPU, and is a good choice because of the large number of multiplies per element copied to the GPU.
- Sorting networks.
- FFTs can also be performed on the GPU. http://gamma.cs.unc.edu/GPUFFTW

Future Work: High-Level Code for Edge Detection Filter

Cg:

```
float main(float2 p : TEXCOORD0,
           uniform samplerRECT arg1) : COLOR
{
   float v1 = texRECT(arg1, p + float2(-1, -1)).r * -1;
   float v2 = texRECT(arg1, p + float2(-1,  0)).r * -2;
   float v3 = texRECT(arg1, p + float2(-1,  1)).r * -1;
   float v4 = texRECT(arg1, p + float2( 0, -1)).r *  0;
   float v5 = texRECT(arg1, p + float2( 0,  0)).r *  0;
   float v6 = texRECT(arg1, p + float2( 0,  1)).r *  0;
   float v7 = texRECT(arg1, p + float2( 1, -1)).r *  1;
   float v8 = texRECT(arg1, p + float2( 1,  0)).r *  2;
   float v9 = texRECT(arg1, p + float2( 1,  1)).r *  1;

   return v1 + v2 + v3 + v4 + v5 + v6 + v7 + v8 + v9;
}
```

Future Work: Assembler for Edge Detection Filter

arbfp1:

```
PARAM c[2] = { { 1, 0, 2, -1 }, { -2 } };
TEMP R0;
TEMP R1;
ADD R0.xy, fragment.texcoord[0], c[0].w;
ADD R0.zw, fragment.texcoord[0].xyxy, c[0].xywy;
TEX R1.x, R0.zwzw, texture[0], RECT;
TEX R0.x, R0, texture[0], RECT;
MAD R0.y, R1.x, c[1].x, -R0.x;
ADD R1.xy, fragment.texcoord[0], c[0].wxzw;
TEX R0.x, R1, texture[0], RECT;
ADD R0.zw, fragment.texcoord[0].xyxy, c[0].xyxw;
TEX R1.x, R0.zwzw, texture[0], RECT;
ADD R0.x, R0.y, -R0;
ADD R0.y, R0.x, R1.x;
ADD R1.xy, fragment.texcoord[0], c[0].x;
TEX R0.x, R1, texture[0], RECT;
ADD R0.zw, fragment.texcoord[0].xyxy, c[0].xyxy;
TEX R1.x, R0.zwzw, texture[0], RECT;
MAD R0.y, R1.x, c[0].z, R0;
ADD result.color.x, R0.y, R0;
END
```

Future Work: Minimize Copies

- Moving data between CPU and GPU is bottleneck.
- Avoiding the transfer of intermediate values can improve performance considerably.
- Optimize texture usage.
- Combine complex operations into a single shader.
- Be aware how the GPU and driver are managing memory.

Future Work: Lazy Evaluation

- Postpone calculations until result is actually needed.
- Avoid copying data between GPU and CPU whenever possible.
- Saves expression and only evaluates it when needed.

Future Work: Miscellaneous

- Auto-configuration: Dynamically decide whether it is faster to perform the operation in software or using the GPU (or a combination of both).
- Double-precision emulation: Use the GPU's single-precision hardware to provide double or greater precision floating point values. http://hal.ccsd.cnrs.fr/ccsd-00021443

email:
eitzenb@eecs.wsu.edu
bobl@tricity.wsu.edu

on the web:
http://eecs.wsu.edu/~eitzenb/gpupy