

---

# Computing Strategies and Tricks

---



Ian Buck

Graphics Lab  
Stanford University

---

**Strategies & Tricks:**

**DirectX or OpenGL?**



---

- DirectX

- + Render to Texture
  - SetRenderTarget()
- + Write once run anywhere
- + Debugging tools
- Short programs
  - Only 512 instr limit
- Readback is slow!
  - ~50 MB/sec

- OpenGL

- + 0 to N texture addressing
  - GL\_TEXTURE\_RECTANGLE\_EXT
- + Vendor Features
- + Readback is fast
- Render-to-Texture not finalized
  - SuperBuffers
  - GL\_EXT\_render\_target
- Specialized float formats for ATI and NV



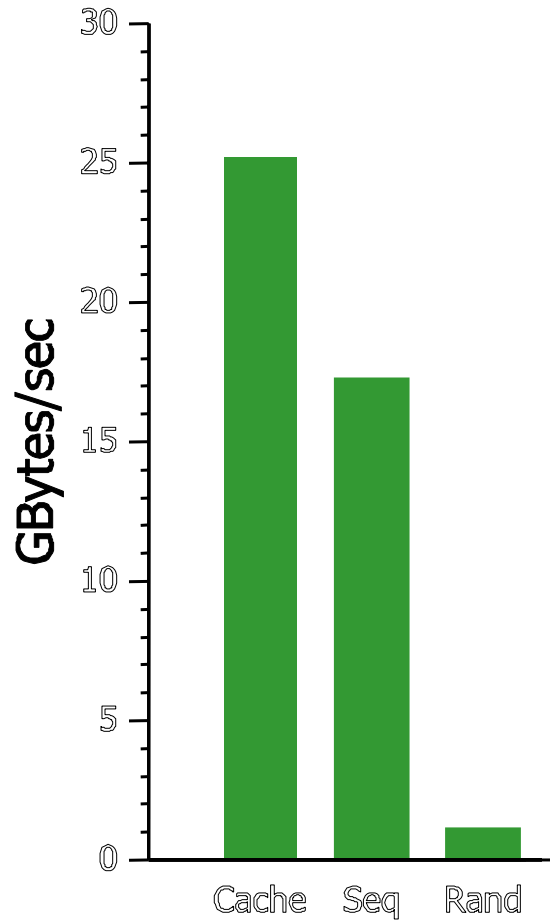
---

**Strategies & Tricks:**

**Understanding  
Performance**



# Locality, Locality, Locality



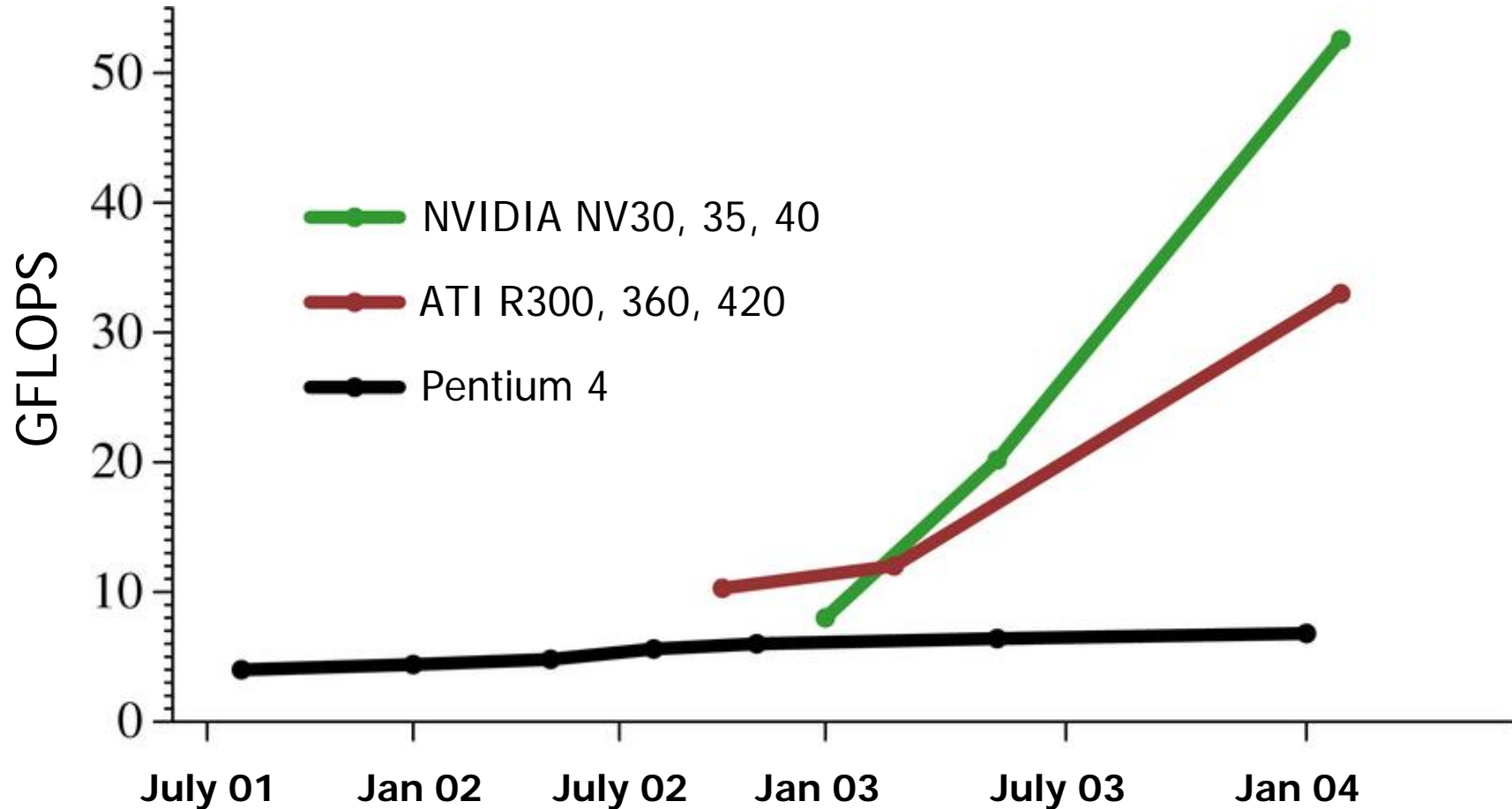
**NVIDIA FP32  
1-Component Textures**

- CPU
  - Pentium 4 3.0 GHz
  - 44 GB/sec peak Cache
  - 6 GB/sec peak Seq
- Output Reuse
  - CPU can cache outputs
  - GPU must write outputs to memory

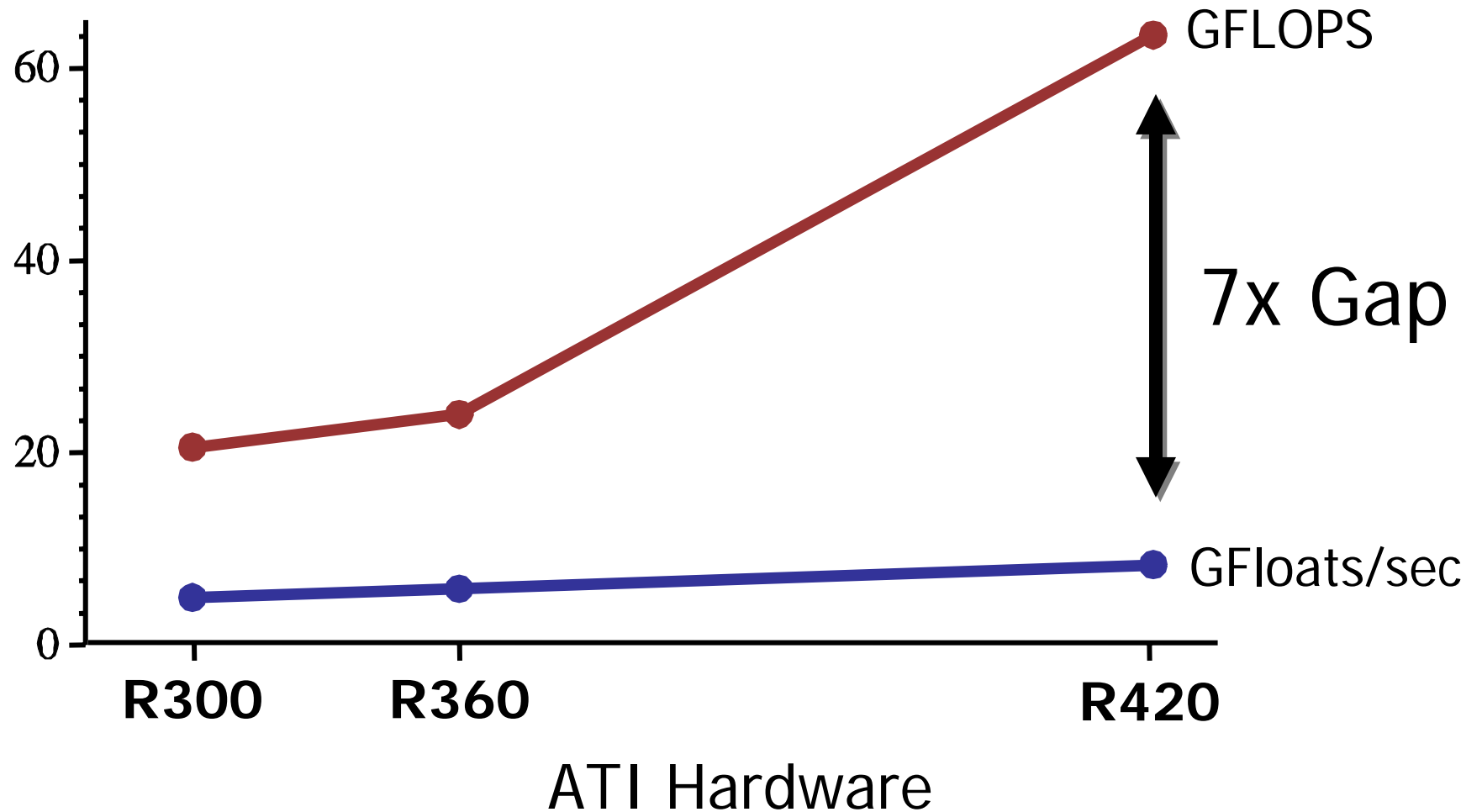


# Compute Performance

multiplies per second



# Bandwidth Gap



# Compute vs. Bandwidth

---

Arithmetic Intensity =  
Compute-to-Bandwidth ratio

## Graphics Pipeline

- Vertex
  - BW: 1 vertex = 32 bytes;
  - OP: 100-500 f32-ops / vertex
- Fragment
  - BW: 1 fragment = 10 bytes
  - OP: 300-1000 i8-ops/fragment



# Considering Readback

---

- GPUs need to download and readback results
  - Time to complete = download + compute + readback
  - Not a problem on CPU
- Readback
  - Getting a lot better!
    - > 600 MB/sec NVIDIA OpenGL
    - GL\_UNSIGNED\_BYTE: BGRA
    - Floating Point: RGBA



---

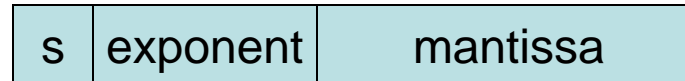
**Strategies & Tricks:**

# **Understanding Floating Point**



# Floating Point Precision

---



$\text{sign} * 1.\text{mantissa} * 2^{(\text{exponent}+\text{bias})}$

- NVIDIA FP32
  - s23e8
- ATI 24-bit float
  - s16e7
- NVIDIA FP16
  - s10e5



# Floating Point Precision

---

- Common Bug
  - Pack large 1D array in 2D texture
  - Compute 1D address in shader
  - Convert 1D address into 2D
  
- FP precision will leave unaddressable texels!

## Largest Counting Number

NVIDIA FP32: 16,777,217

ATI 24-bit float: 131,073

NVIDIA FP16: 2,049



---

**Strategies & Tricks:**

# Reductions



# Reductions

---

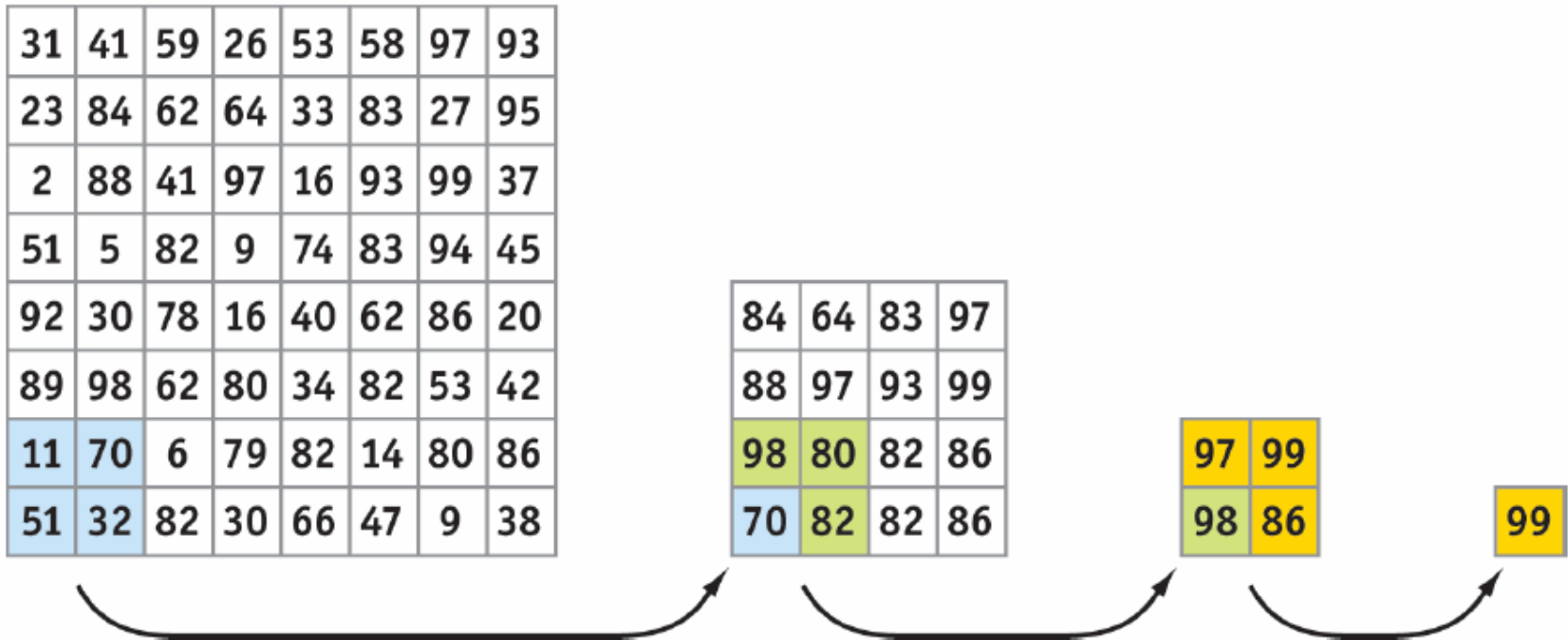
- Problem: How to compute the largest element of a vector?

```
float main(float2 texcoord : TEX0,  
uniform samplerRECT img) : COLOR  
{  
    float a, b, c, d;  
    a = fltexRECT(img, texcoord);  
    b = fltexRECT(img, texcoord + float2(0, 1));  
    c = fltexRECT(img, texcoord + float2(1, 0));  
    d = fltexRECT(img, texcoord + float2(1, 1));  
    return max(max(a, b), max(c, d));  
}
```



# Reductions

- Multi-pass solution
  - Requires reduction to be associative
  - Works for Sum, Max, Mul, Matrix multiply, etc...



---

## Strategies & Tricks:

# Implementing Scatter

$$a[i] = p$$



# Scatter Techniques

---

- Problem:  $a[i] = p$ 
  - Indirect write
  - Can't set the  $x,y$  of fragment in pixel shader
  - Often want to do:  $a[i] += p$



# Scatter Techniques

---

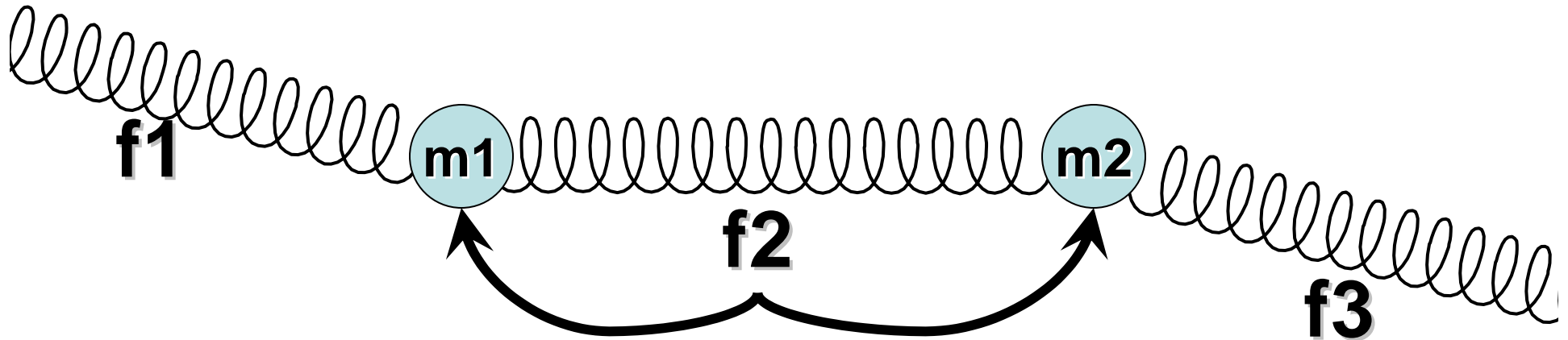
- Solution 1: Convert to Gather

for each spring

$f$  = computed force

`mass_force[left] += f;`

`mass_force[right] -= f;`



# Scatter Techniques

---

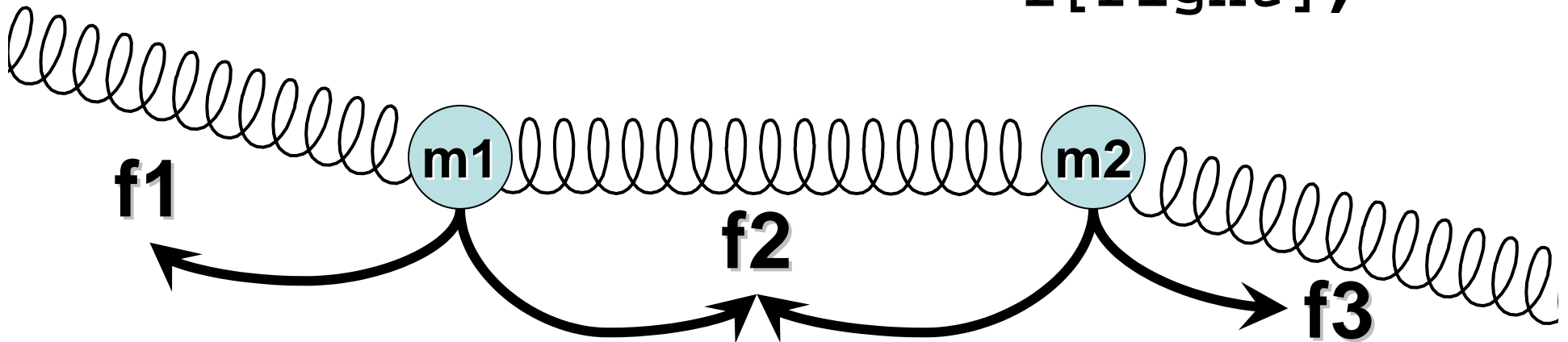
- Solution 1: Convert to Gather

**for each spring**

**f = computed force**

**for each mass**

**mass\_force = f[left] -  
f[right];**



# Scatter Techniques

---

- Solution 2: Address Sorting
  - Sort & Search
    - Shader outputs destination address and data
    - Bitonic sort based on address
    - Run binary search shader over destination buffer
      - Each fragment searches for source data



# Scatter Techniques

---

- Solution 3: Vertex processor
  - Render points
    - Use vertex shader to set destination
    - or just read back the data and re-issue
  - Vertex Textures
    - Render data and address to texture
    - Issue points, set point x,y in vertex shader using address texture
    - Requires texld instruction in vertex program



---

**Strategies & Tricks:**

# Conditionals



# Conditionals

---

- Problem:

```
if (a) b = f();  
else  b = g();
```

- Limited fragment shader conditional support



# Conditionals

---

- Solution 1: Predication

- Execute both `if (a) b = f();`
- f and g `else b = g();`

- Use LRP instruction

- LRP b, a, f, g `b = a ? f : g`
- Executes all conditional code



# Conditionals

---

- Solution 1: Predication

- Use DP4 instruction

- DP4 b.x, a, f

- Executes all conditional code

**a** = ( 0 , 1 , 0 , 0 )

**f** = ( **x** , **y** , **z** , **w** )

```
if (a.x) b = x;  
else if (a.y) b = y;  
else if (a.z) b = z;  
else if (a.w) b = w;
```

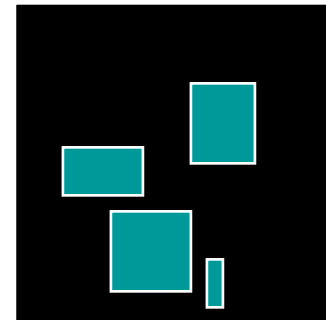


# Conditionals

---

- Solution 2: Using early Z-kill
  - Set Z buffer to a
    - Clear Z to 1.0f
    - Render quad at z=0.3
    - Evaluate conditional and kill to set Z

```
if (a) b = f();  
else  b = g();
```

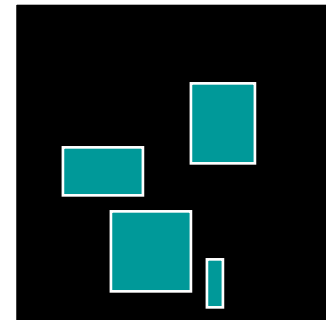


# Conditionals

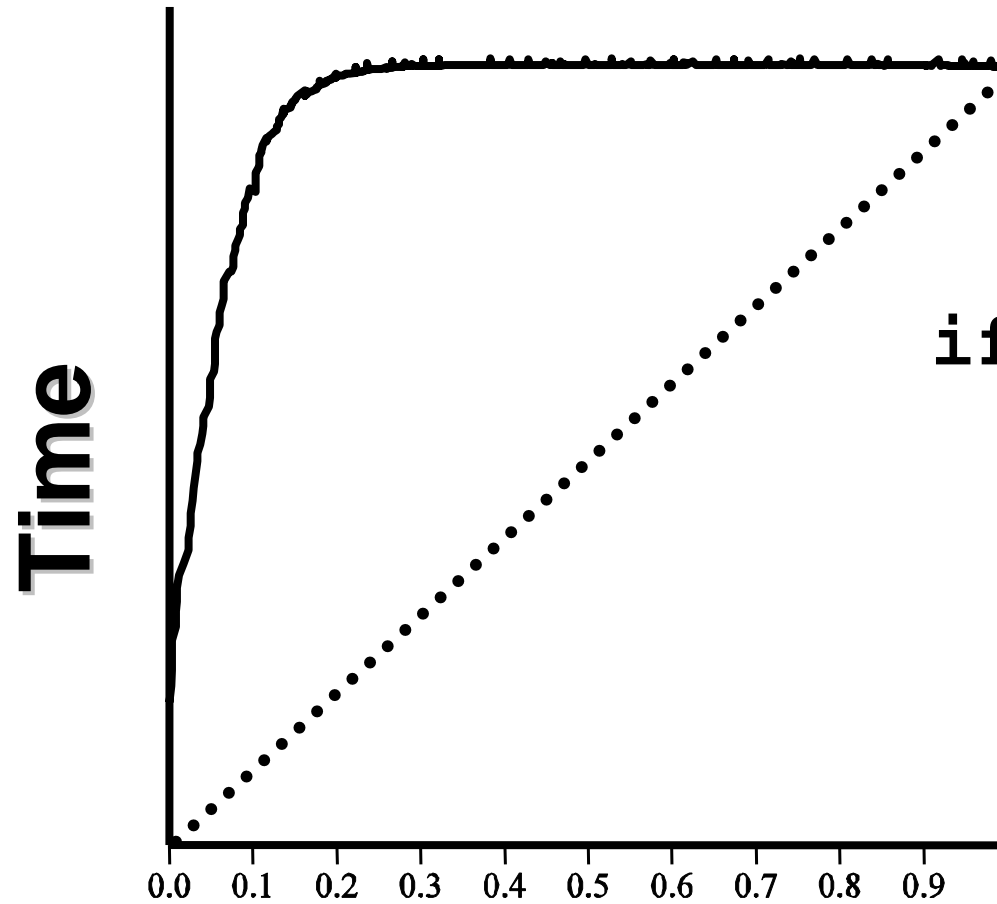
---

- Solution 2: Using early Z-kill
  - Set Z buffer to a
  - Z-test can prevent shader execution
    - `glEnable(GL_DEPTH_TEST)`
  - Good only if locality in conditional

```
if (a) b = f();  
else  b = g();
```



# Conditionals



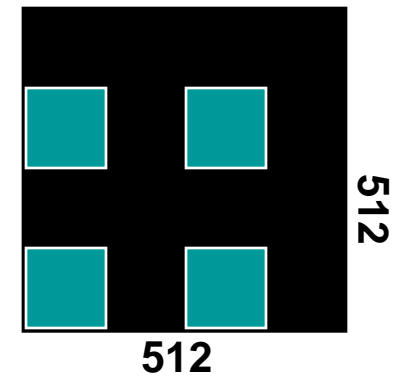
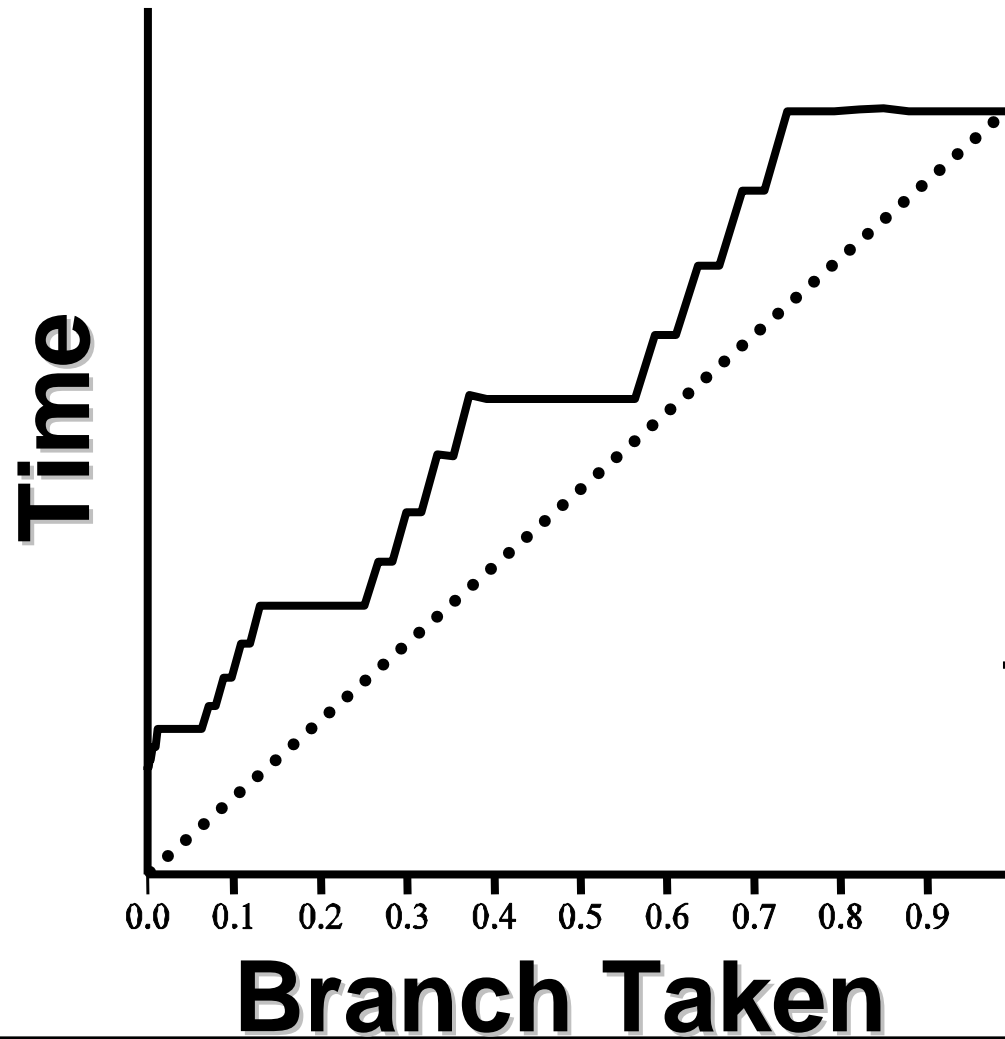
```
if (rand)  
b = f();
```

**Branch Taken**

**ATI X800**



# Conditionals



```
if (block)  
    b = f();
```

ATI X800



# Conditionals

---

## ● Solution 2: Using early Z-kill

- Very Sensitive!

### ATI:

- Z out in shader
- alpha test enabled
- texkill is shader

### NV3X & NV4X:

- Changing depth test direction invalidates for remainder of frame

### NV3X:

- Alpha test, alpha-to-coverage, user clip planes,
- Pixel kill in shader (KIL), shader alters Z

### NV4X:

- Writing stencil while rejecting based on stencil
- Changing stencil func/ref/mask invalidates for remainder of frame



# Conditionals

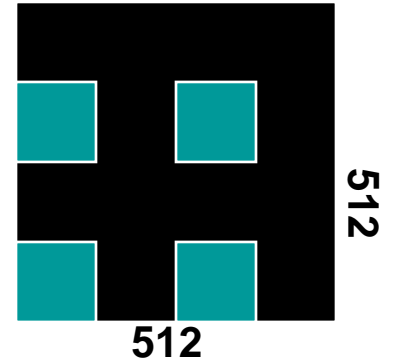
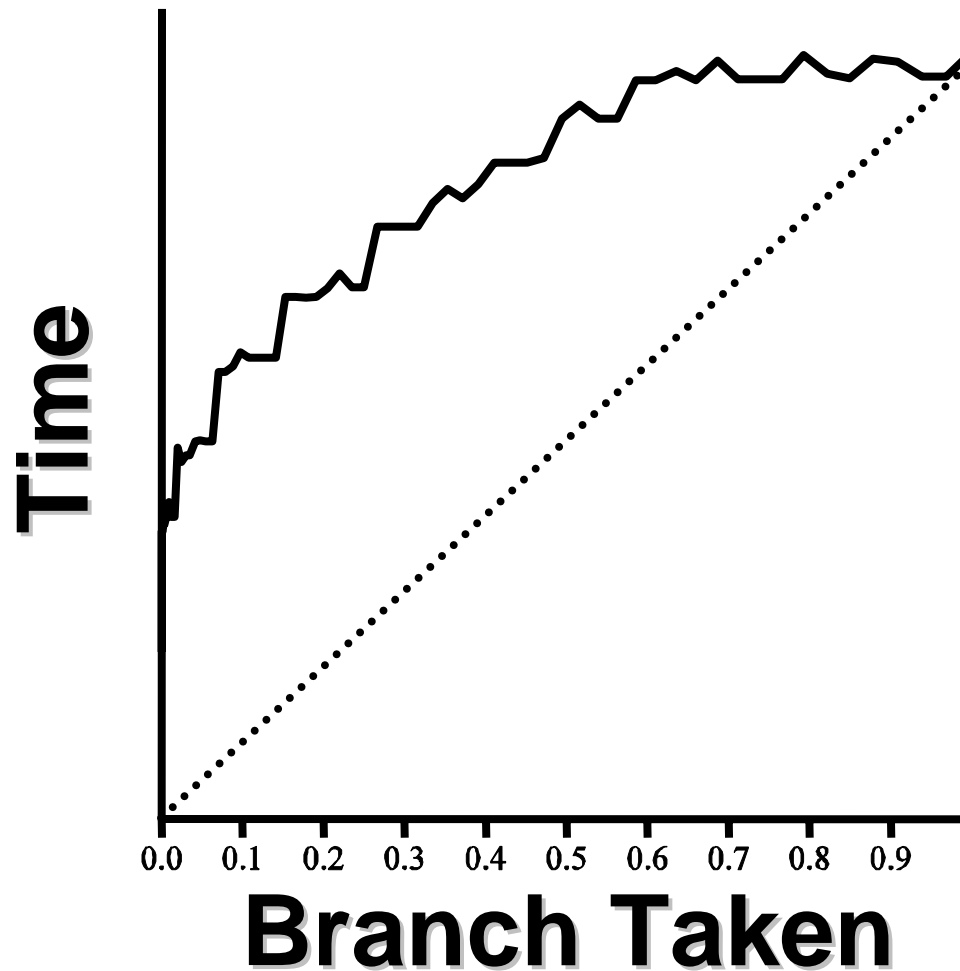
---

- Solution 3: Conditional Instructions
  - Available with NV\_fragment\_program2

```
MOVC CC, R0;  
IF GT.x;  
MOV R0, R1; # executes if R0.x > 0  
ELSE;  
MOV R0, R2; # executes if R0.x <= 0  
ENDIF;
```



# Conditionals



```
if (block)  
    b = f();
```

**GeForce 6800**



---

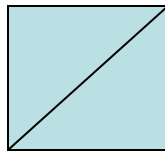
# Strategies & Tricks: Optimizing Execution



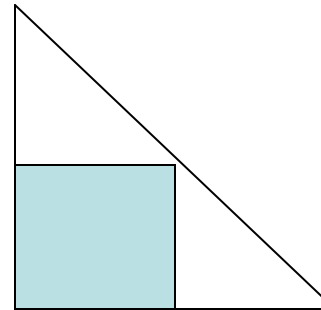
# Optimizing Execution

---

- Two methods for GPGPU shader execution



```
glBegin(GL_QUADS);  
glVertex2f(left, bottom);  
glVertex2f(right, bottom);  
glVertex2f(right, top);  
glVertex2f(left, top);  
glEnd();
```

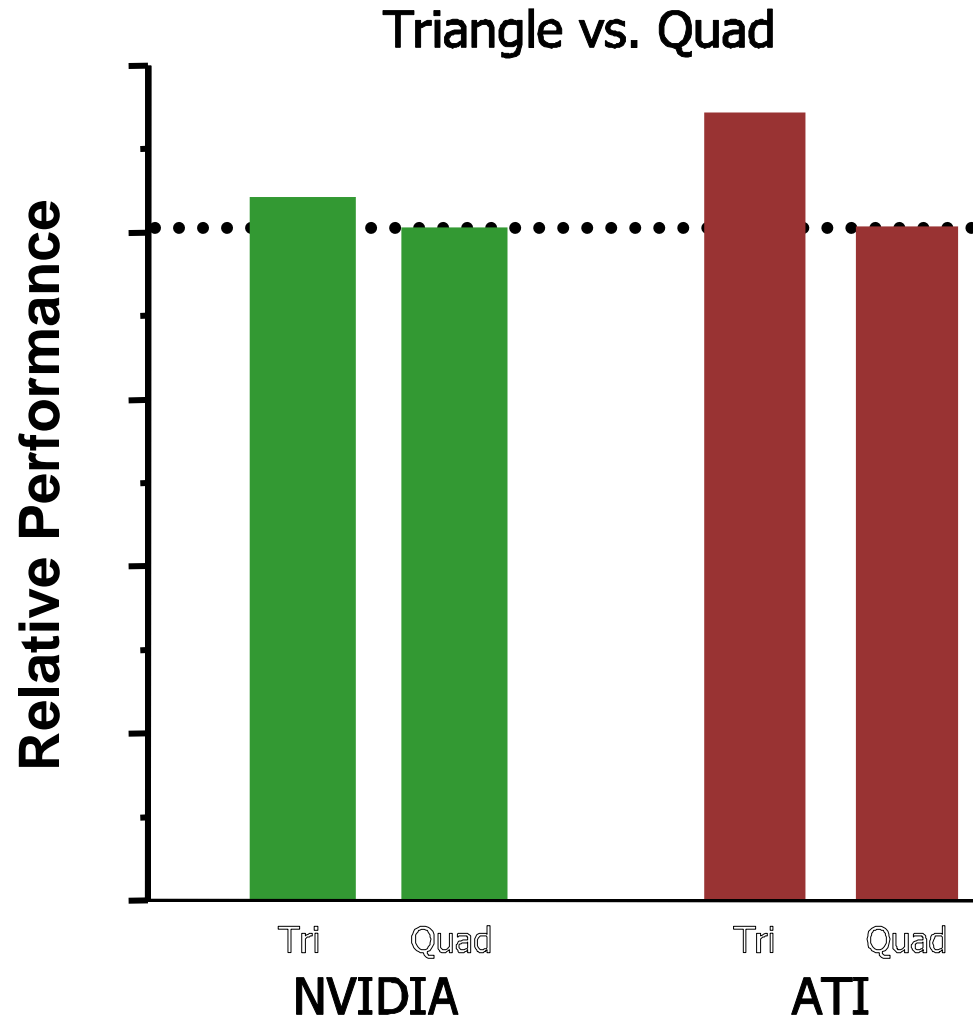


```
glViewport(0,0,width,height)  
glBegin(GL_TRIANGLE);  
glVertex2f( 0, 0);  
glVertex2f(width*2, 0);  
glVertex2f( 0, height*2);  
glEnd();
```



# Optimizing Execution

---



---

# Strategies & Tricks: Multiple Outputs



# Multiple Outputs

---

- Software solution
  - Let driver, cgc, or fxc do dead code elimination

```
kernel void foo (float3 a<>,
                 float3 b<>, ...,
                 out float3 x<>,
                 out float3 y<>)
```

- Works well if shader is separable

```
kernel void foo1(float3 a<>,
                 float3 b<>, ...,
                 out float3 x<>)
```

```
kernel void foo2(float3 a<>,
                 float3 b<>, ...,
                 out float3 y<>)
```



# GPUBench

- <http://graphics.stanford.edu/projects/gpubench>
- <http://sourceforge.net/projects/gpubench>

