



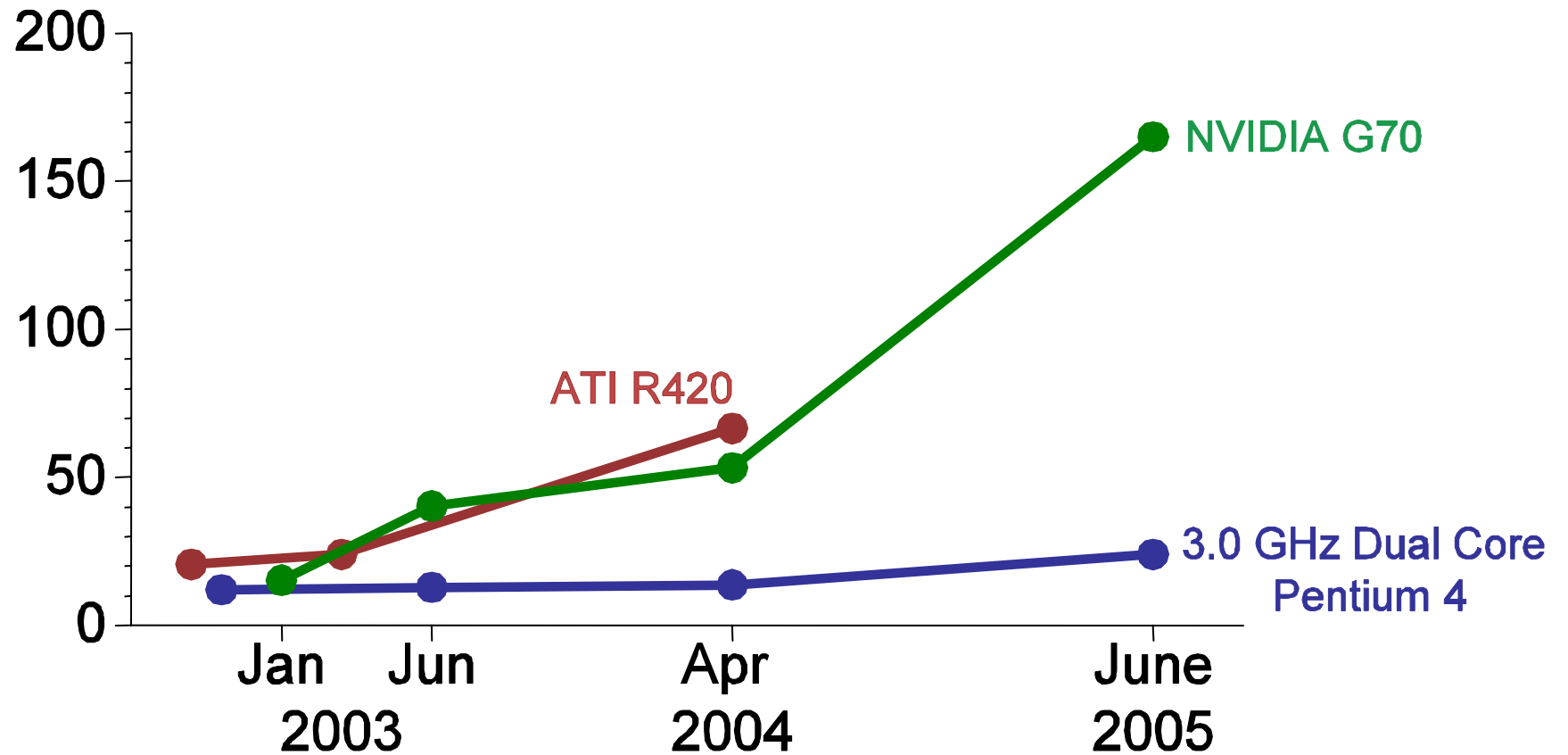
# GPU Computation Strategies & Tricks

Ian Buck  
NVIDIA

**GP GPU**

# Recent Trends

---



# Compute is Cheap

---

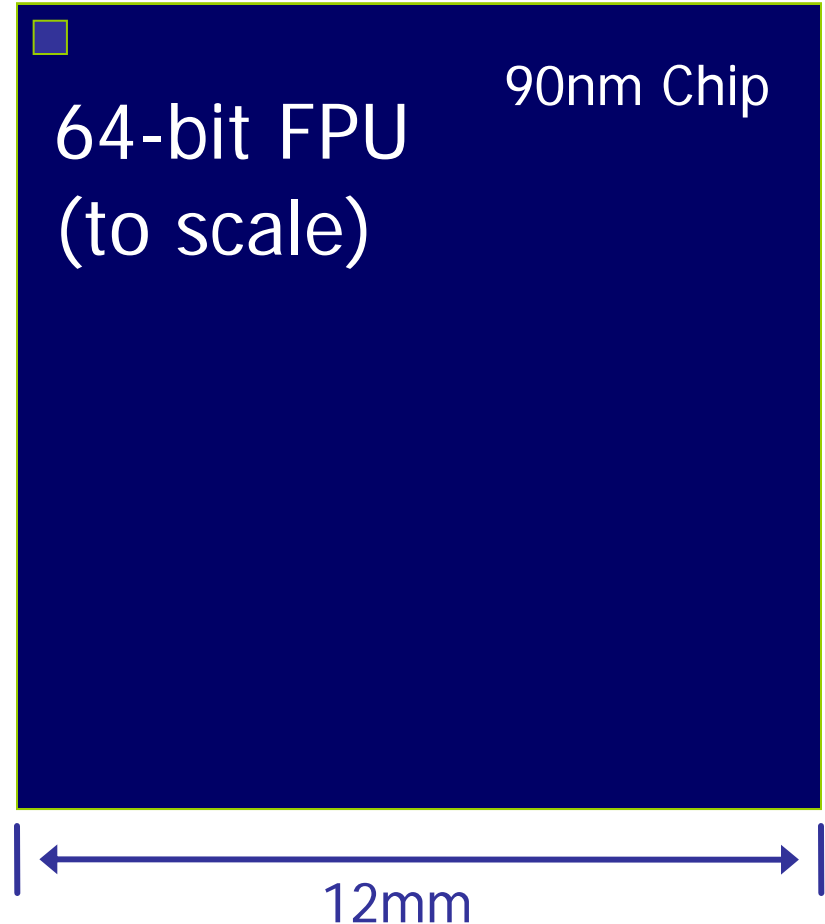
- **parallelism**

- to keep 100s of ALUs per chip busy

- **shading is highly parallel**

- millions of fragments per frame

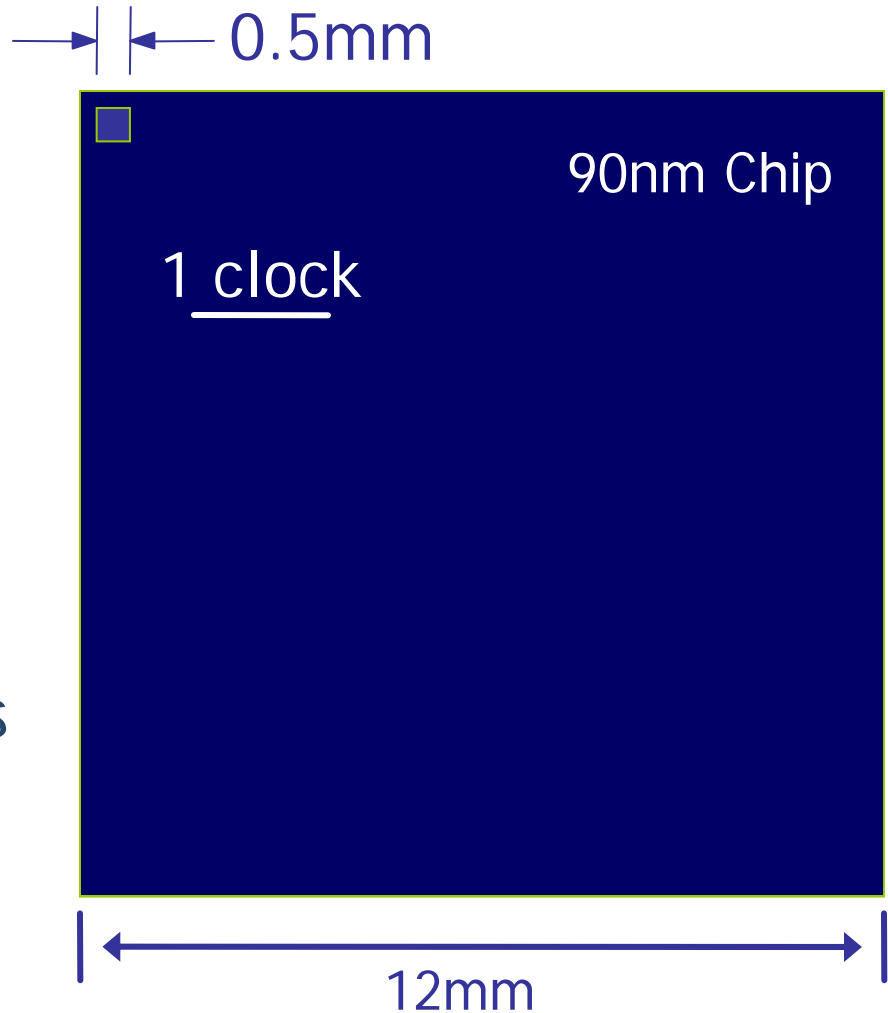
→ | | ← 0.5mm



# ...but Bandwidth is Expensive

---

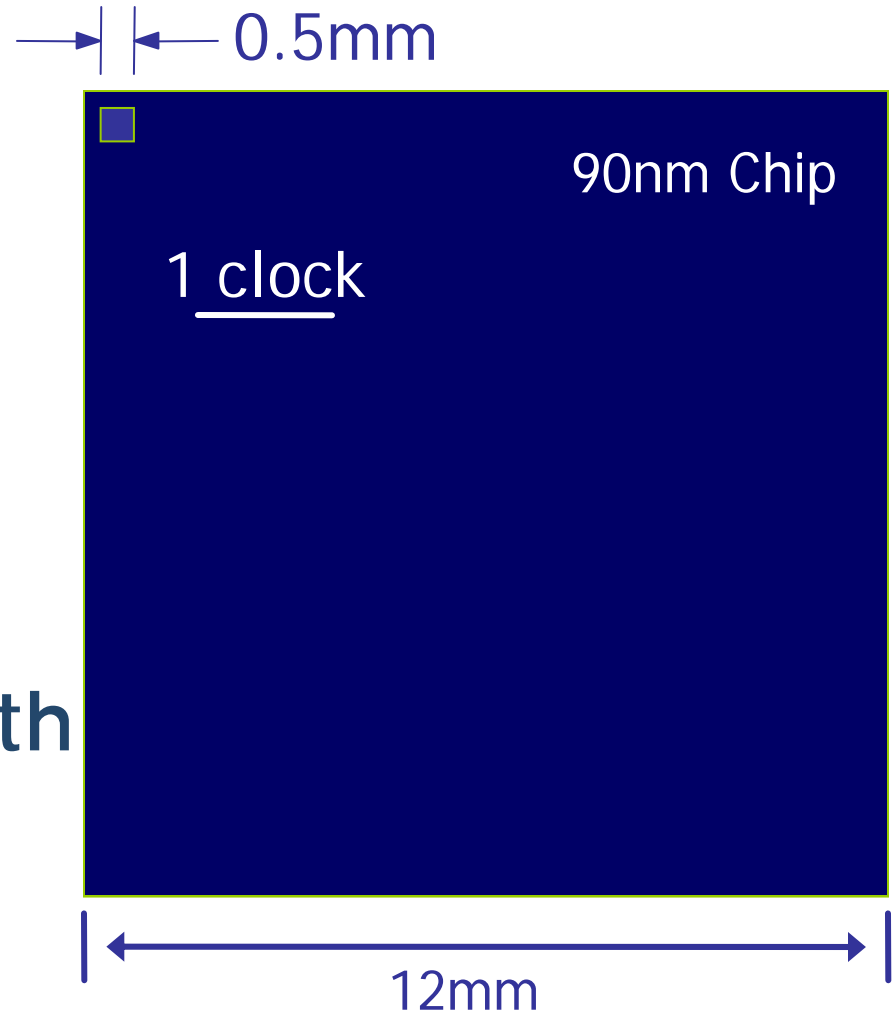
- **latency tolerance**
  - to cover 500 cycle remote memory access time
- **locality**
  - to match 20Tb/s ALU bandwidth to ~100Gb/s chip bandwidth



# Optimizing for GPUs

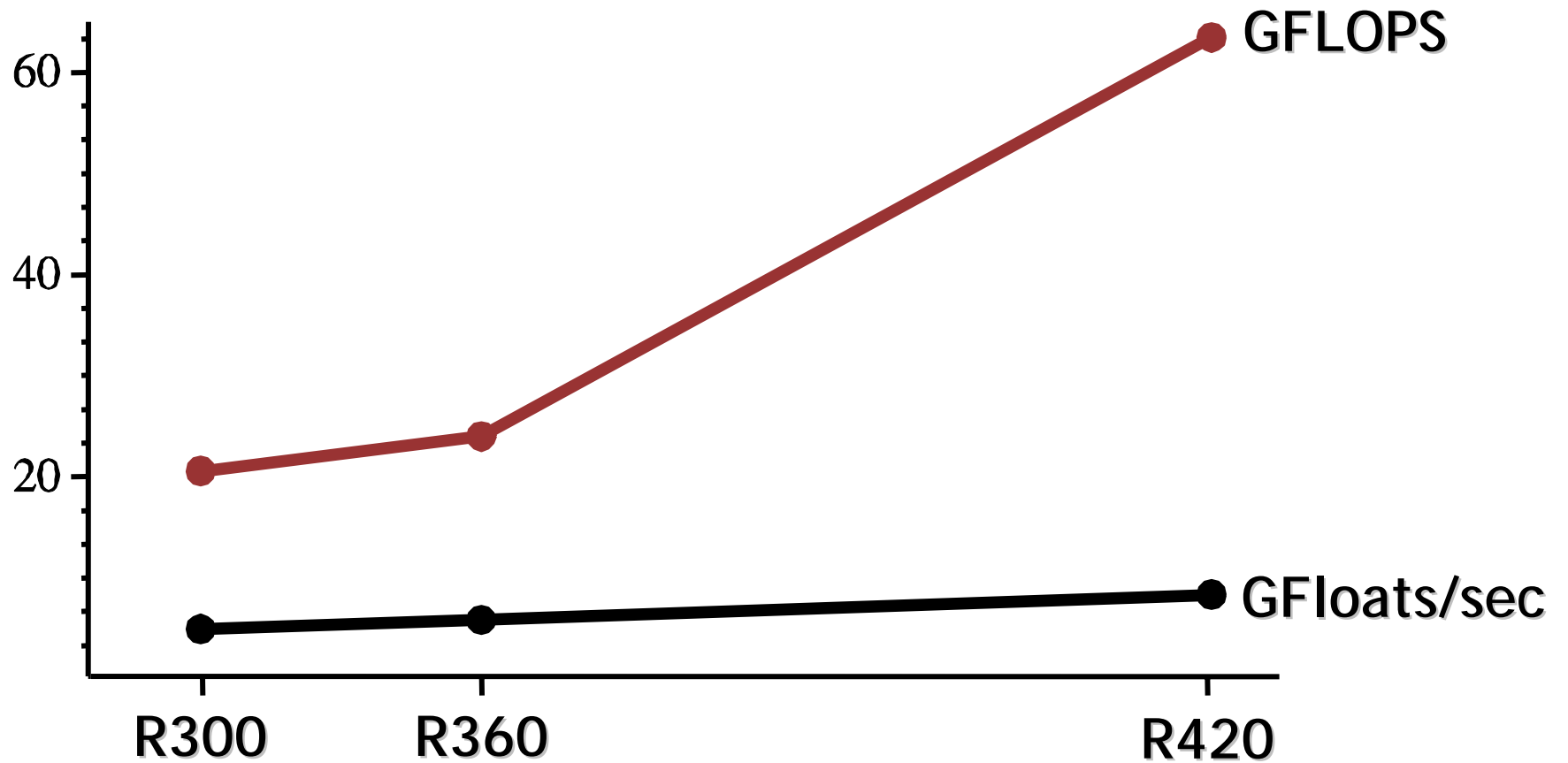
---

- shading is compute intensive
  - 100s of floating point operations
  - output 1 32-bit color value
- compute to bandwidth ratio
  - arithmetic intensity



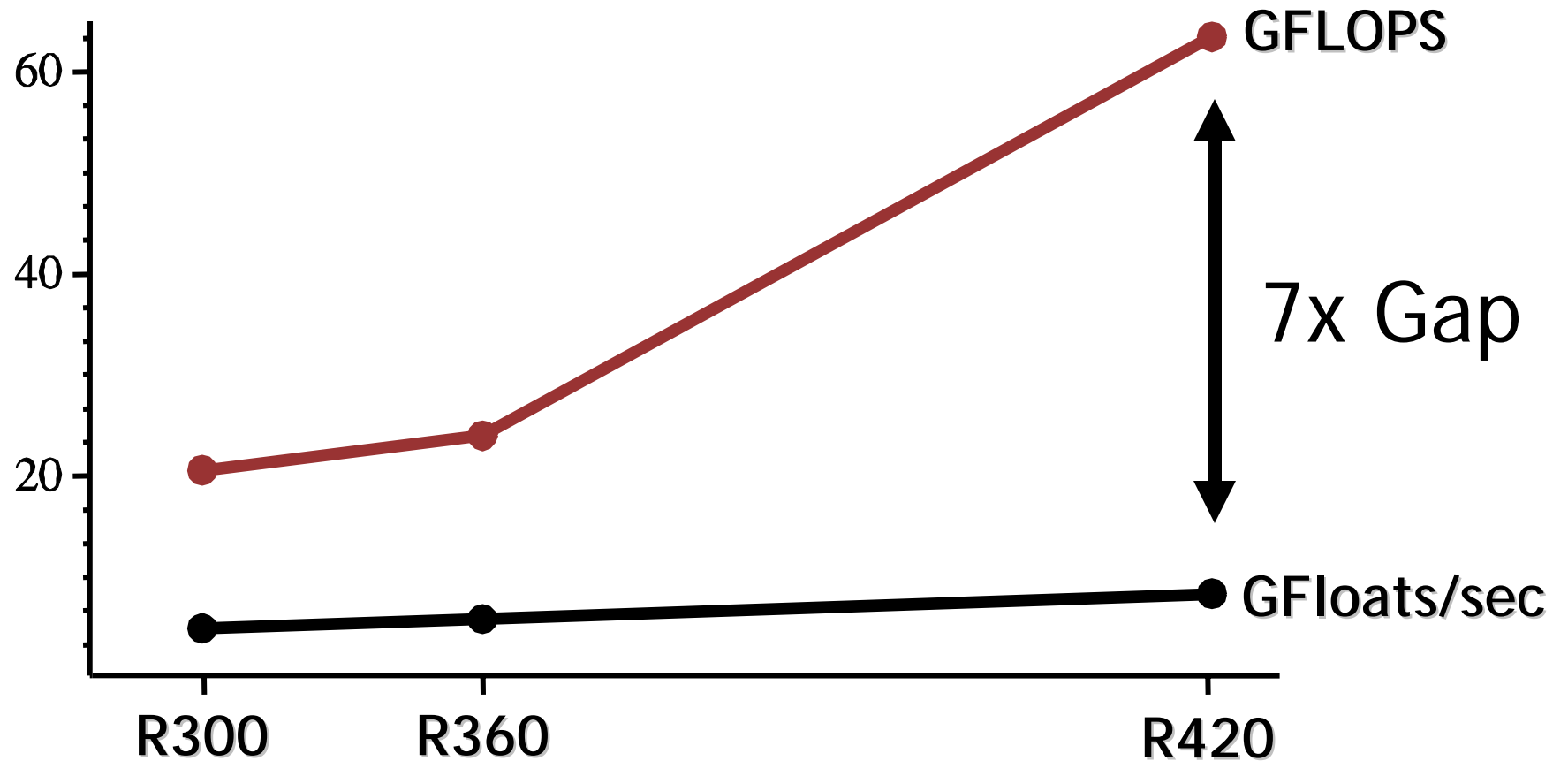
# Compute vs. Bandwidth

---



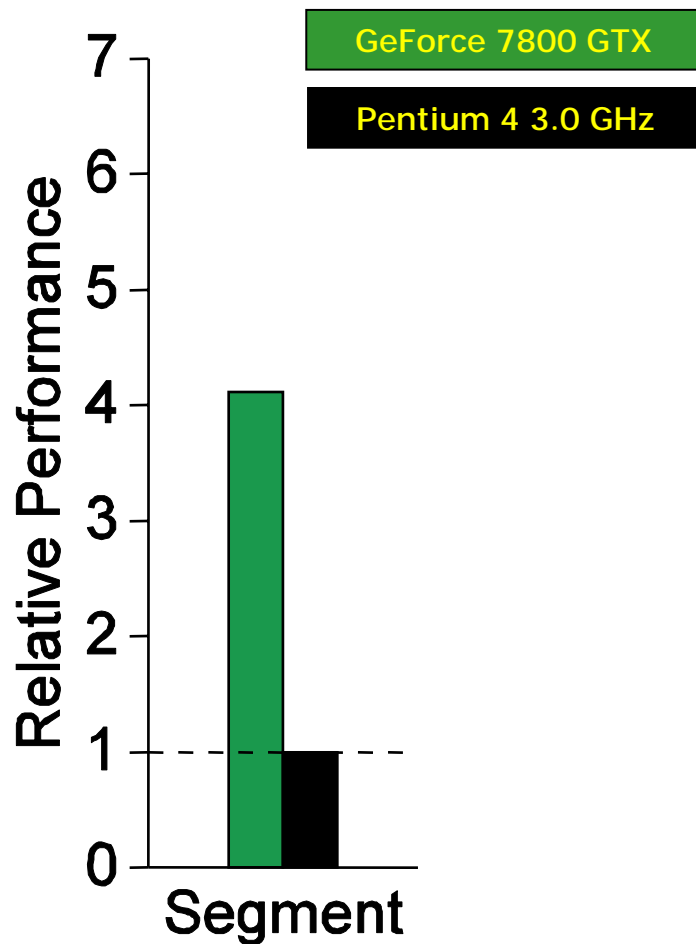
# Arithmetic Intensity

---



# Arithmetic Intensity

---

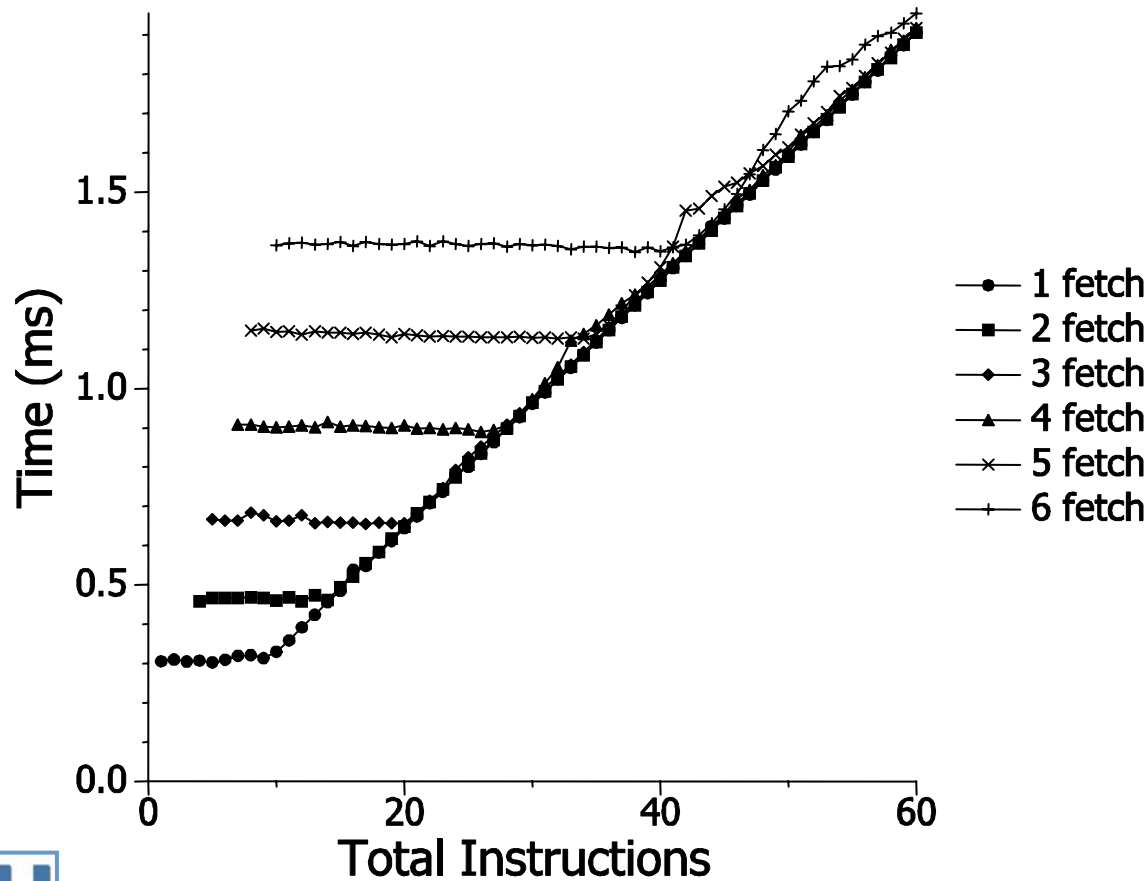


GPU wins when...

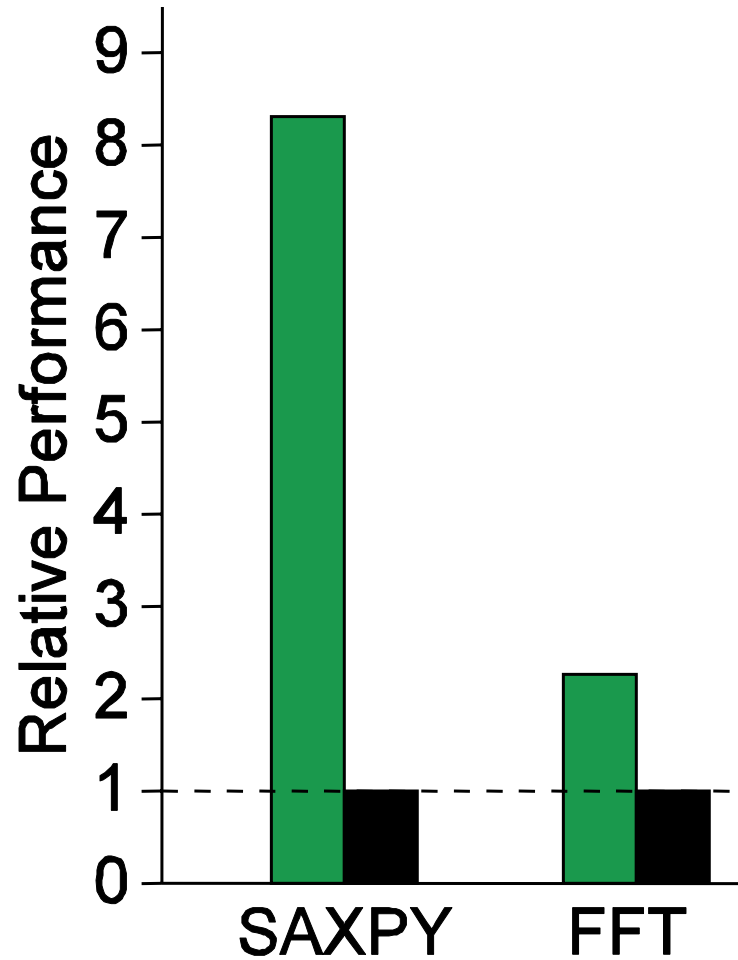
- Arithmetic intensity
  - ✓ Segment
  - 3.7 ops per word
  - 11 GFLOPS

# Arithmetic Intensity

- Overlapping computation with communication



# Memory Bandwidth



GPU wins when...

- Streaming memory bandwidth

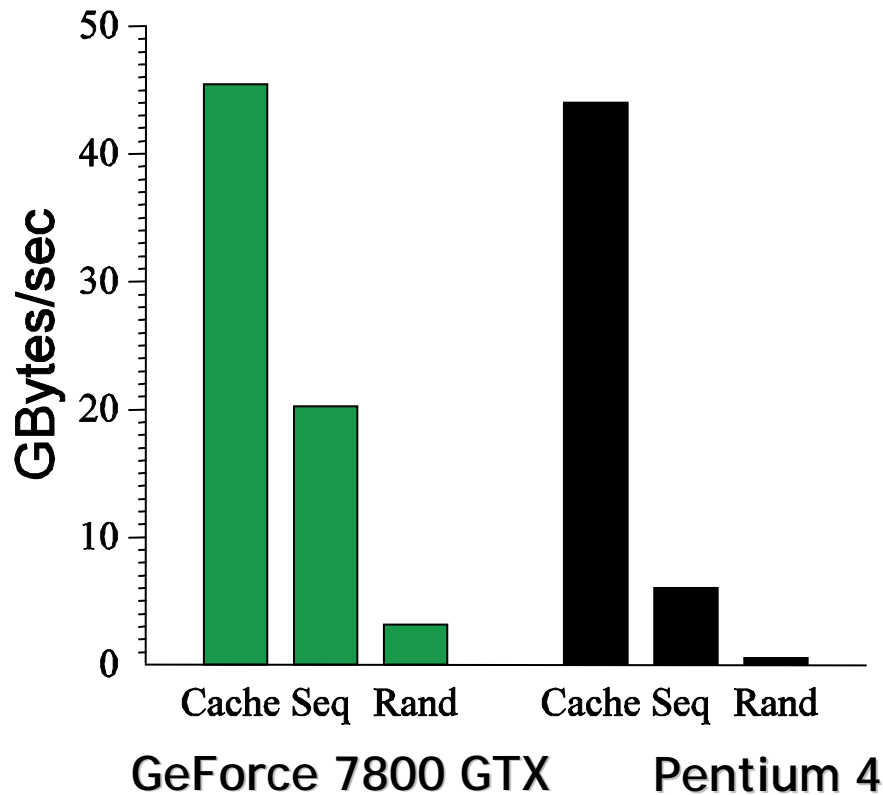
✓ SAXPY

✗ FFT

GeForce 7800 GTX

Pentium 4 3.0 GHz

# Memory Bandwidth

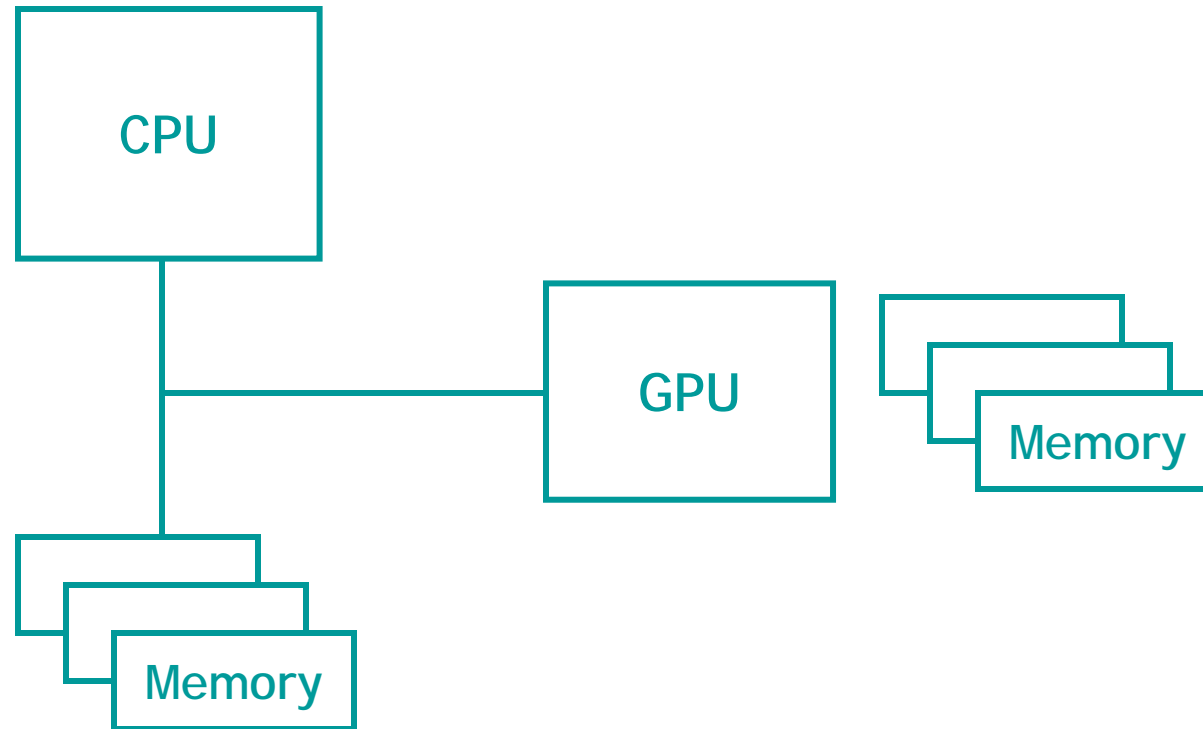


- Streaming Memory System
  - Optimized for sequential performance
- GPU cache is limited
  - Optimized for texture filtering
  - Read-only
  - Small
- Local storage
  - CPU >> GPU

# Computational Intensity

---

- Considering GPU transfer costs:  $T_r$



# Computational Intensity

---

- Considering GPU transfer costs:  $T_r$ 
  - Computational intensity:  $\gamma$

$$\gamma \equiv K_{\text{gpu}} / T_r$$

work per word  
transferred

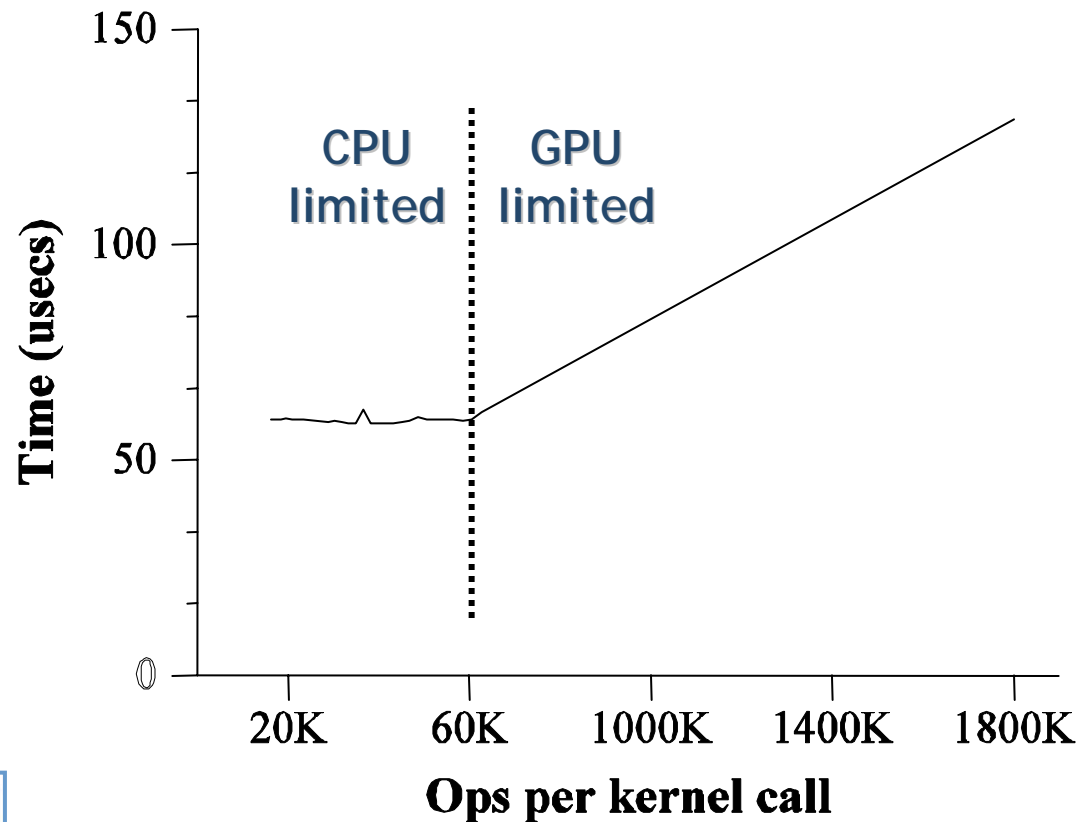
- to outperform the CPU:

speedup:  $s \equiv K_{\text{cpu}} / K_{\text{gpu}}$

$$\gamma > \frac{1}{s - 1}$$

# Kernel Overhead

- Considering CPU cost to issuing a kernel
  - Generating compute geometry
  - Graphics driver



# 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

# 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

---

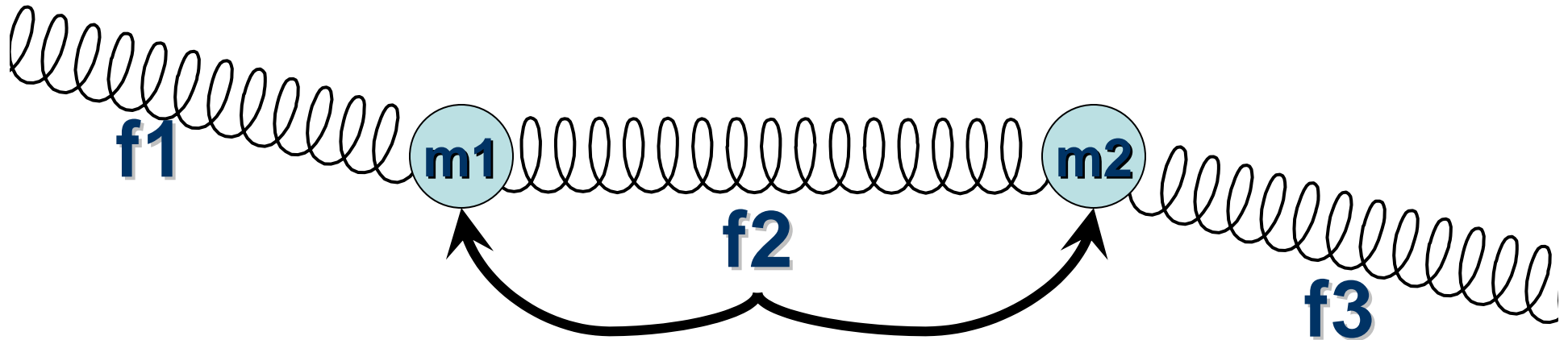
- Solution 1: Convert to Gather

`for each spring`

`f = computed force`

`mass_force[left] += f;`

`mass_force[right] -= f;`

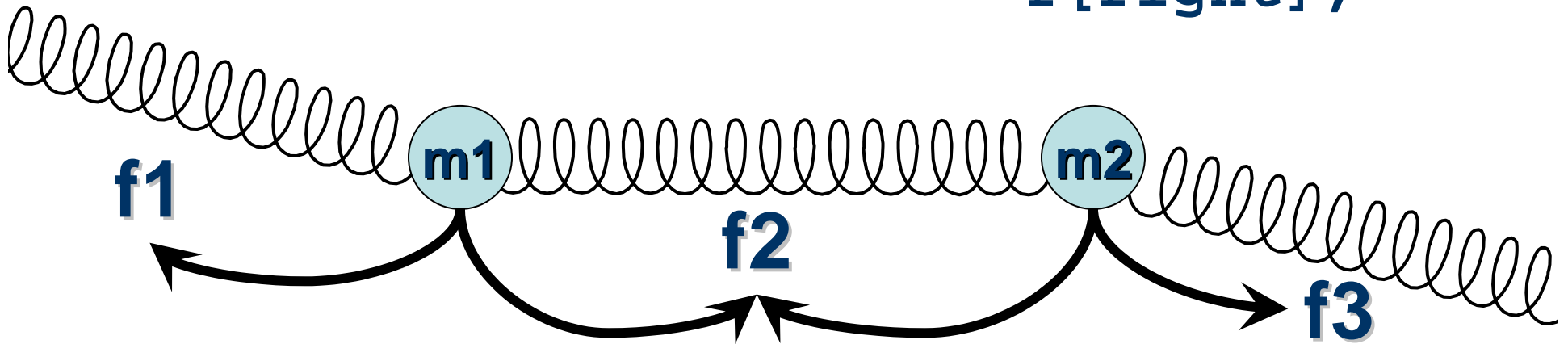


# Scatter Techniques

---

- 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

# Pre-computation

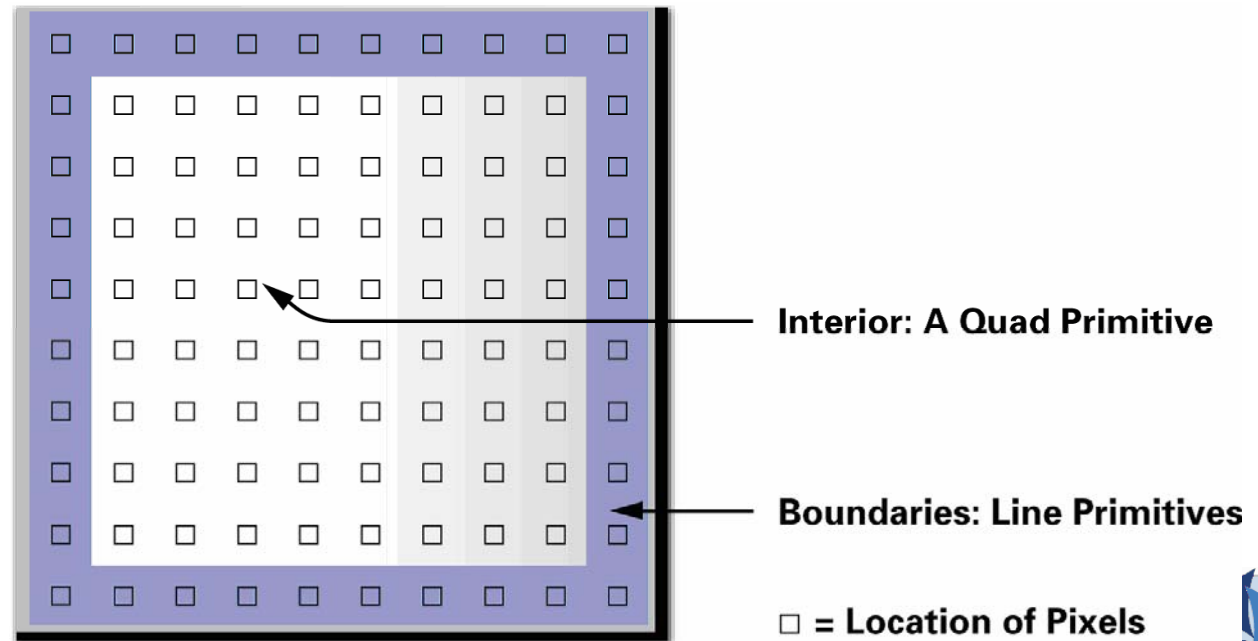
---

- Pre-compute anything that will not change every iteration!
- Example: static obstacles in fluid sim
  - When user draws obstacles, compute texture containing boundary info for cells
  - Reuse that texture until obstacles are modified
  - Combine with Z-cull for higher performance!

# Static Branch Resolution

---

- Avoid branches where outcome is fixed
  - One region is always true, another false
  - Separate FPs for each region, no branches
- Example: boundaries



# Branching with Occlusion Query

---

- Use it for iteration termination

Do

```
{ // outer loop on CPU
```

```
  BeginOcclusionQuery
```

```
  {
```

```
    // Render with fragment program that
```

```
    // discards fragments that satisfy
```

```
    // termination criteria
```

```
  } EndQuery
```

```
} While query returns > 0
```

- Can be used for subdivision techniques

# Conditionals

---

- Predication

- Execute both
- f and g

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

- Use CMP instruction

- CMP b, -a, f, g
- Executes all conditional code

# Conditionals

---

- Predication

- Use DP4 instruction

$\mathbf{a} = (0, 1, 0, 0)$

$\mathbf{f} = (\mathbf{x}, \mathbf{y}, \mathbf{z}, \mathbf{w})$

- DP4 b.x, a, f

- Executes all conditional code

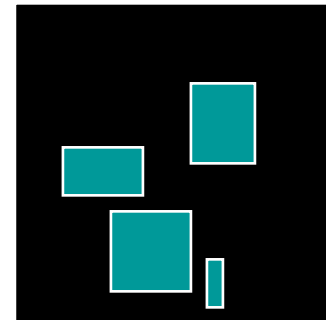
```
if (a.x) b = x;  
else if (a.y) b = y;  
else if (a.z) b = z;  
else if (a.w) b = w;
```

# Conditionals

---

- Using the depth buffer
  - Set Z buffer to a
  - Z-test can prevent shader execution
    - glEnable(GL\_DEPTH\_TEST)
  - Locality in conditional

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



# Conditionals

---

- Using the depth buffer
  - Optimization disabled with:

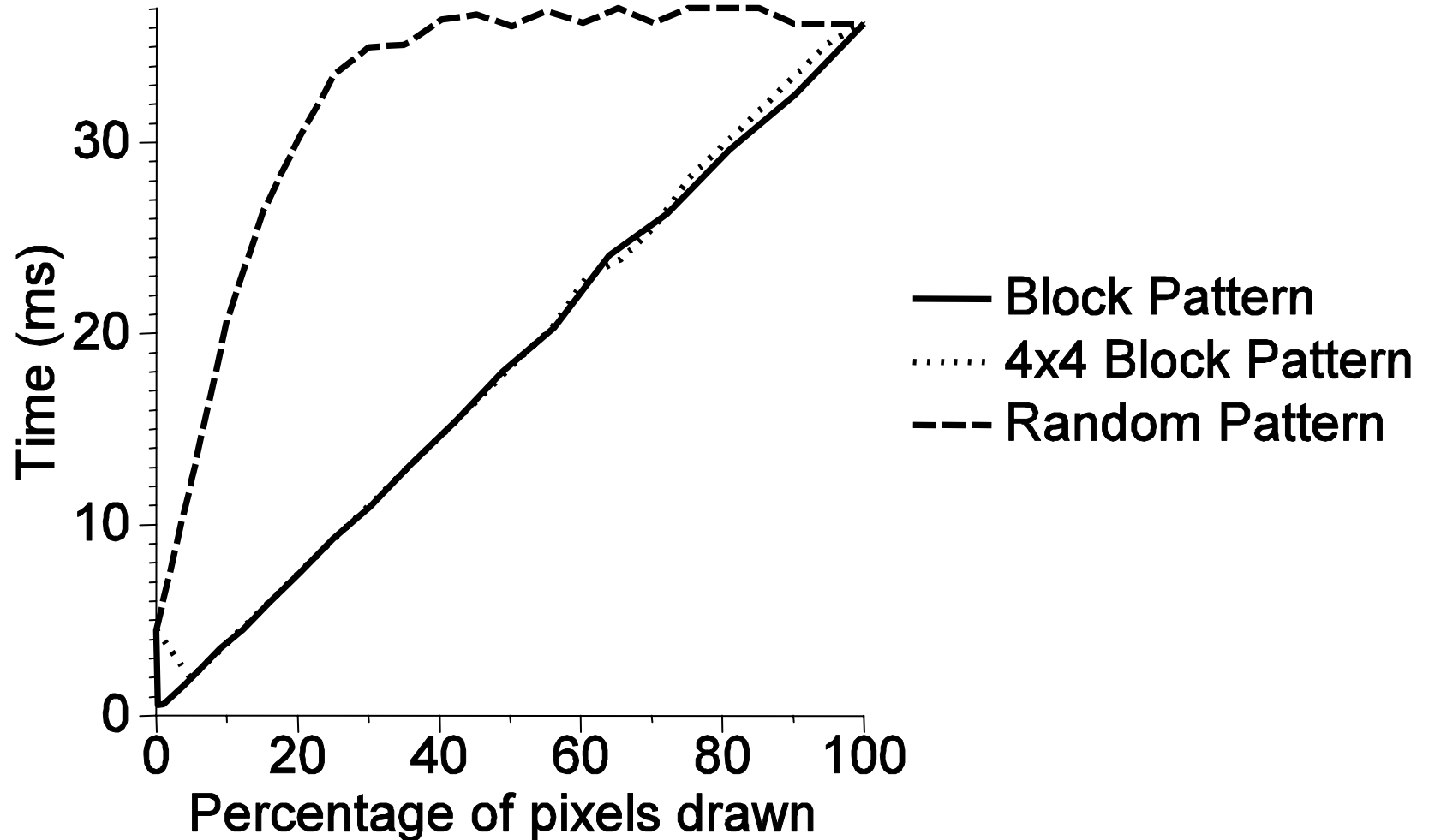
## ATI:

- Writing Z in shader
- Enabling Alpha test
- Using texkill in shader

## NVIDIA:

- Changing depth test direction in frame
- Writing stencil while rejecting based on stencil
- Changing stencil func/ref/mask in frame

# Depth Conditionals



# Conditionals

---

- Conditional Instructions

- Available with NV\_fragment\_program2

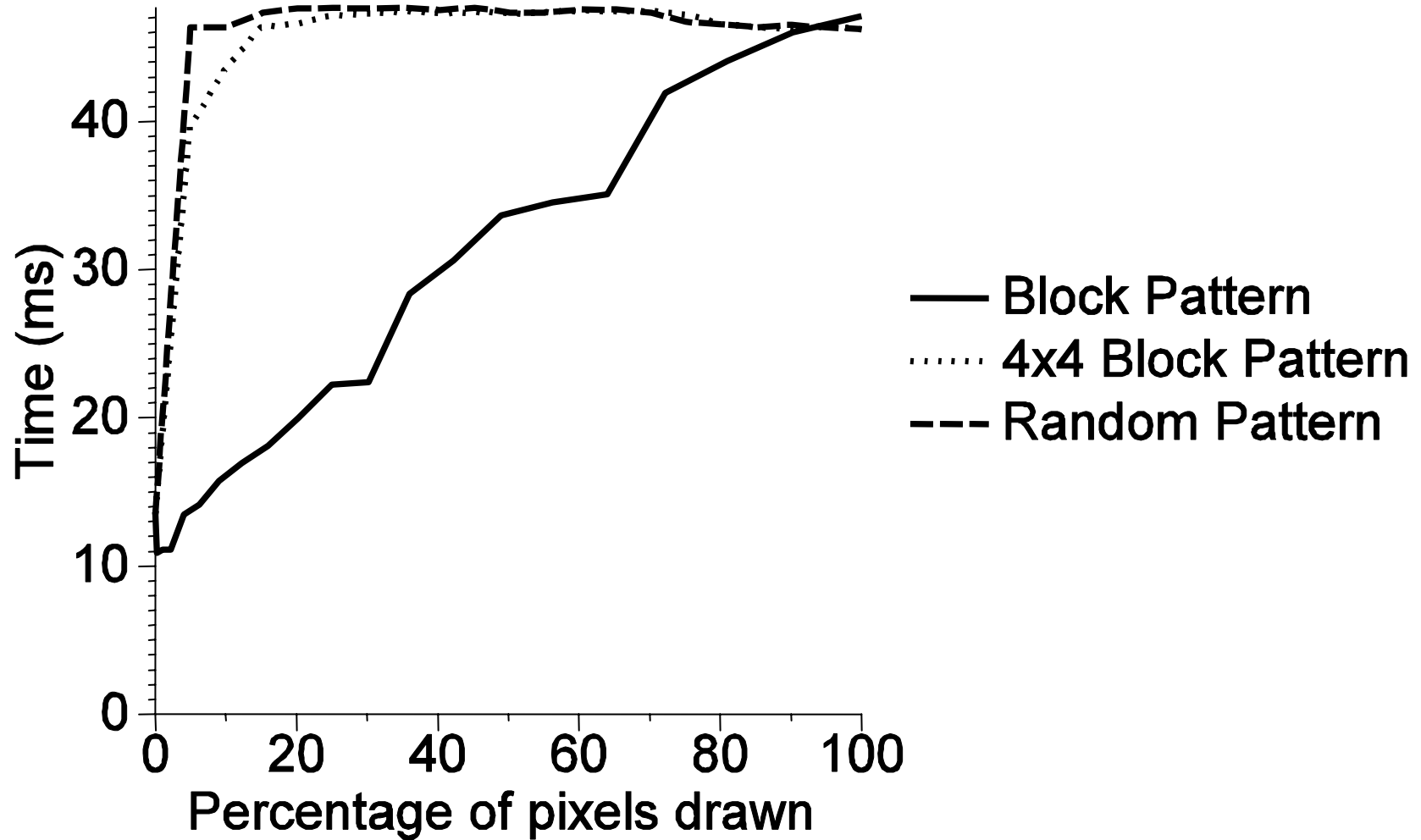
```
MOVC CC, R0;  
IF GT.x;  
MOV R0, R1; # executes if R0.x > 0  
ELSE;  
MOV R0, R2; # executes if R0.x <= 0  
ENDIF;
```

# GeForce 6/7 Series Branching

---

- **True, SIMD branching**
  - Lots of incoherent branching can hurt performance
  - Should have coherent regions of ~1000 pixels
    - That is only about 30x30 pixels, so still very useable!
- **Don't ignore overhead of branch instructions**
  - Branching over < 5 instructions may not be worth it
- **Use branching for early exit from loops**
  - Save a lot of computation

# Conditional Instructions



# Branching Techniques

---

- **Fragment program branches can be expensive**
  - No true fragment branching on GeForce FX or Radeon 9x00-X850
  - SIMD branching on GeForce 6/7 Series
    - Incoherent branching hurts performance
- **Sometimes better to move decisions up the pipeline**
  - Replace with math
  - Occlusion Query
  - Static Branch Resolution
  - Depth Buffer
  - Pre-computation