

# GPU Computation Strategies & Tricks

John Owens  
UC Davis

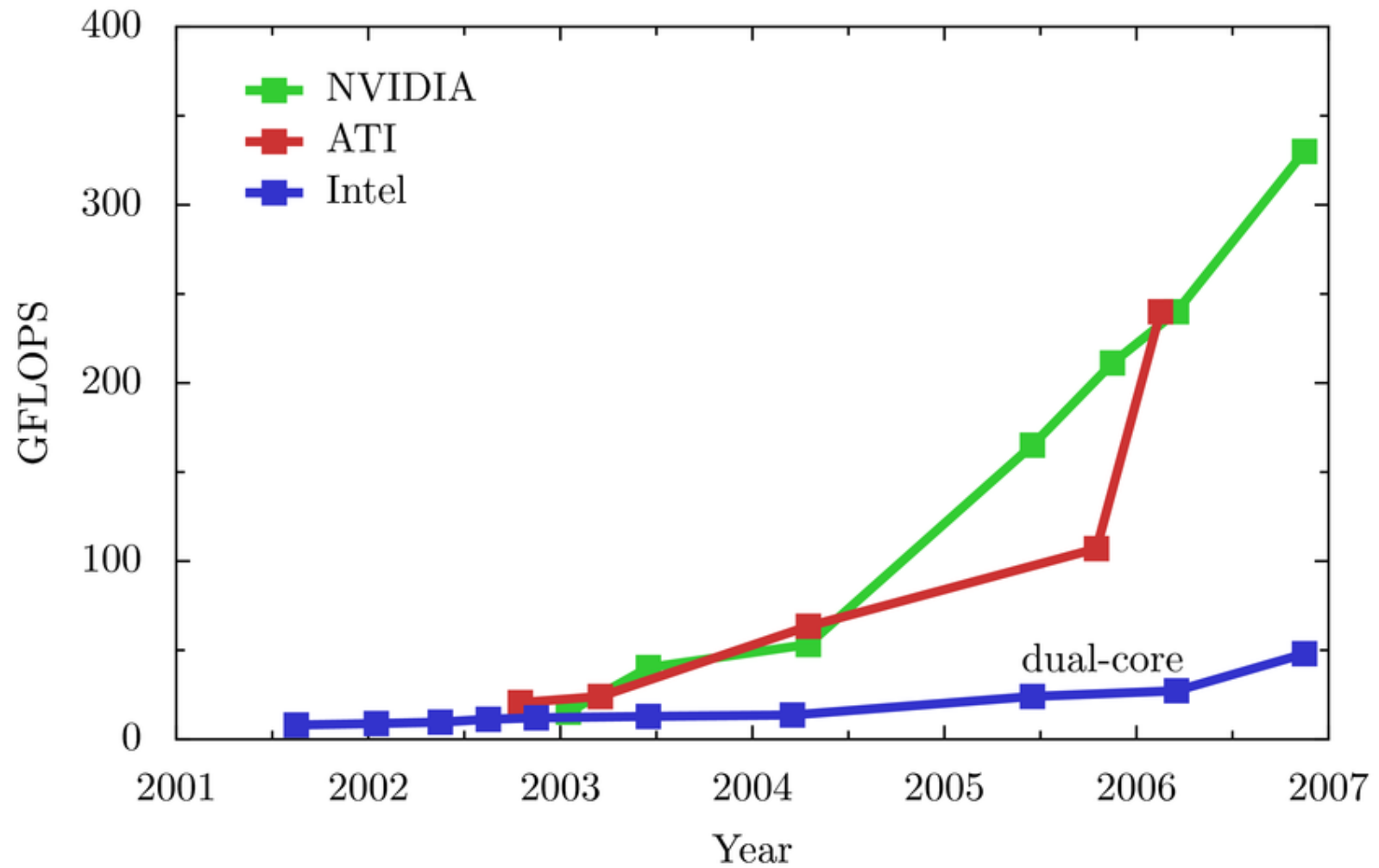


# Outline

---

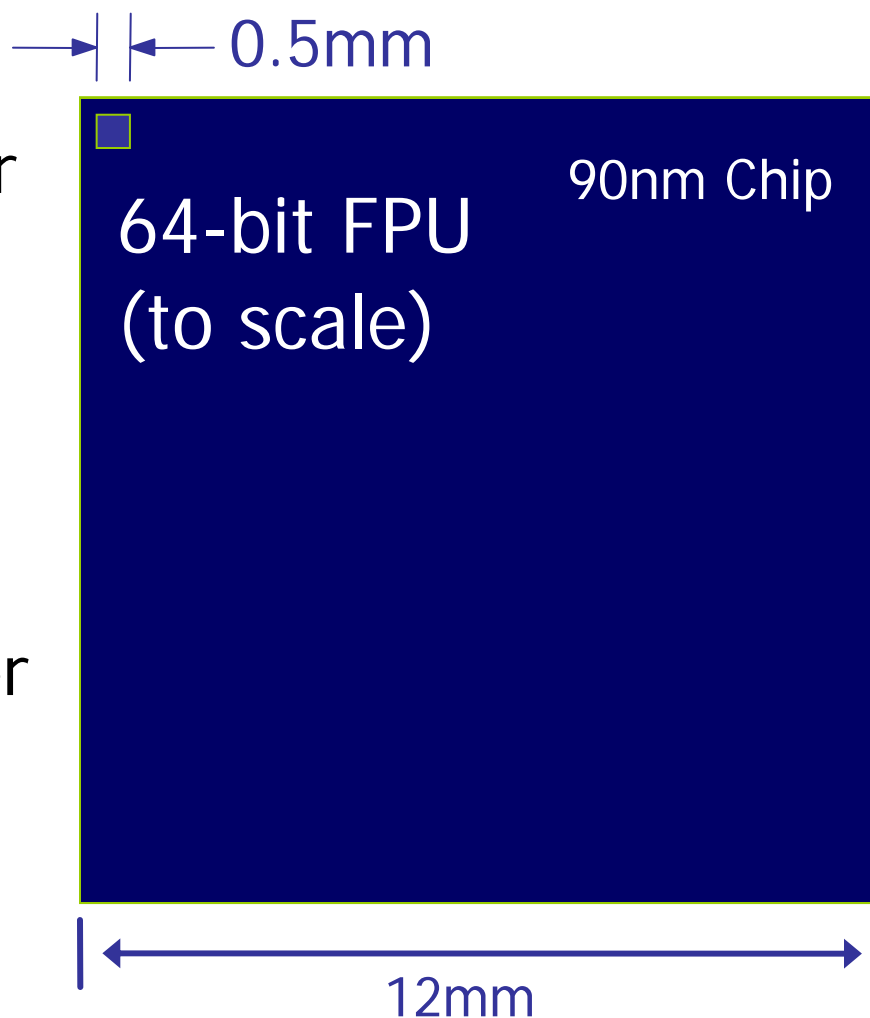
- Big-picture: GPU performance guidelines
- Scatter
- Conditionals

# Recent Trends



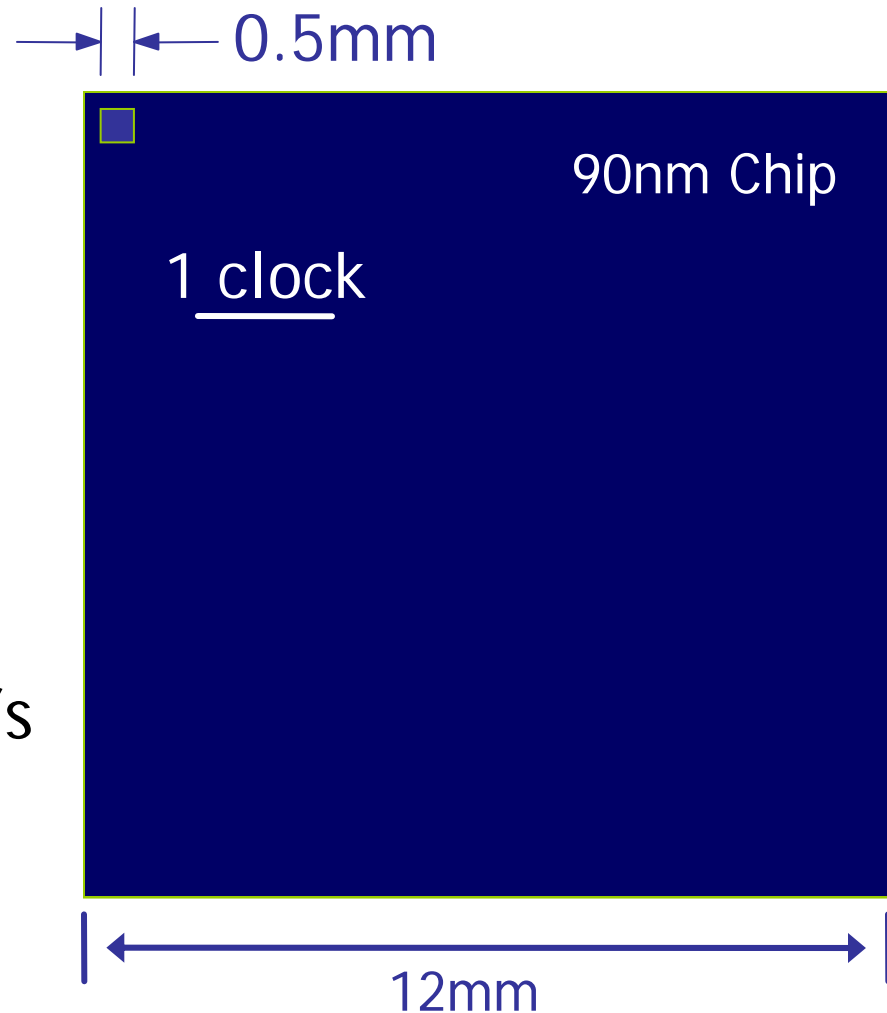
# Compute is Cheap

- **parallelism**
  - to keep 100s of ALUs per chip busy
- **shading is highly parallel**
  - millions of fragments per frame



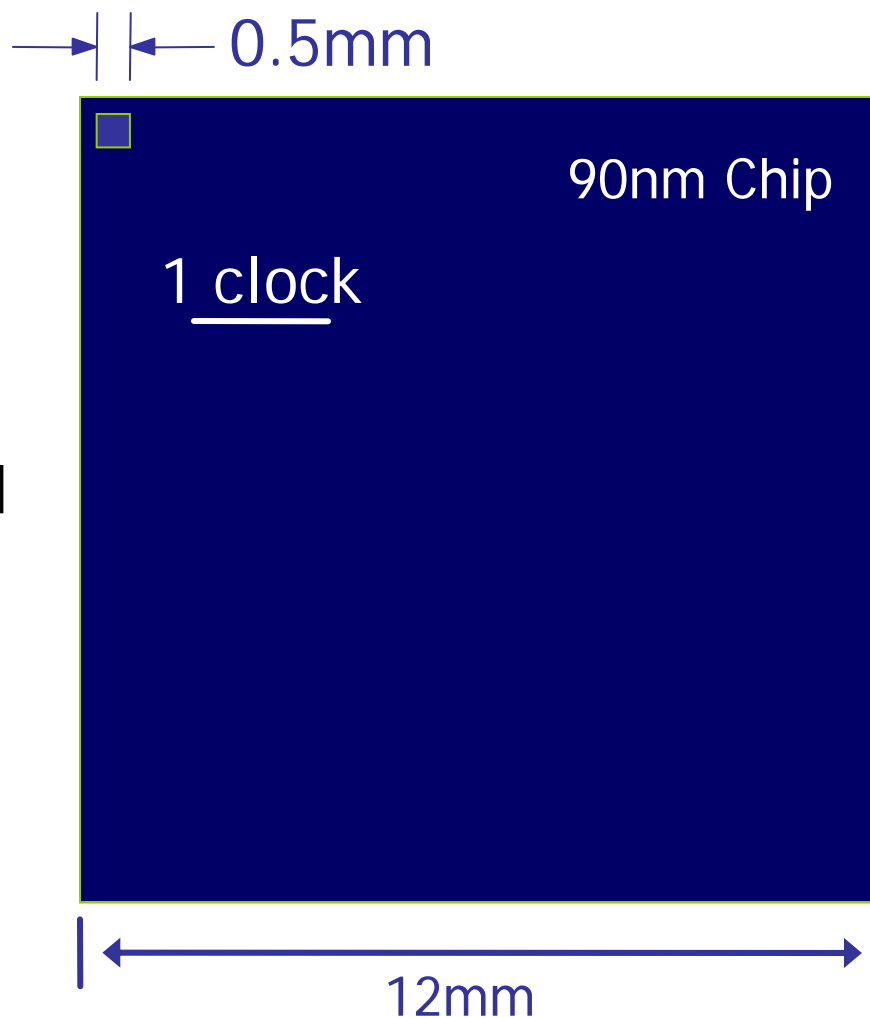
# ...but Bandwidth is Expensive

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

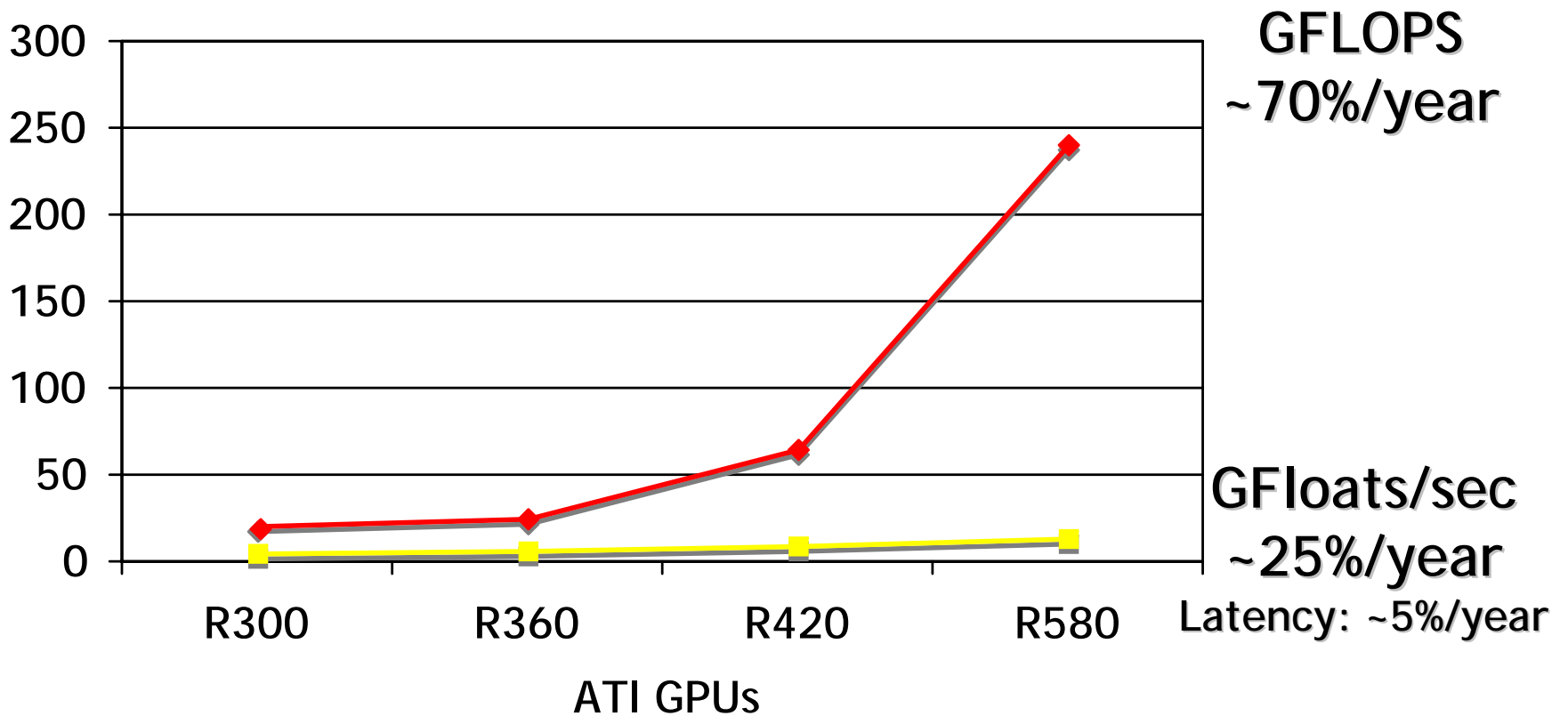


# Optimizing for GPUs

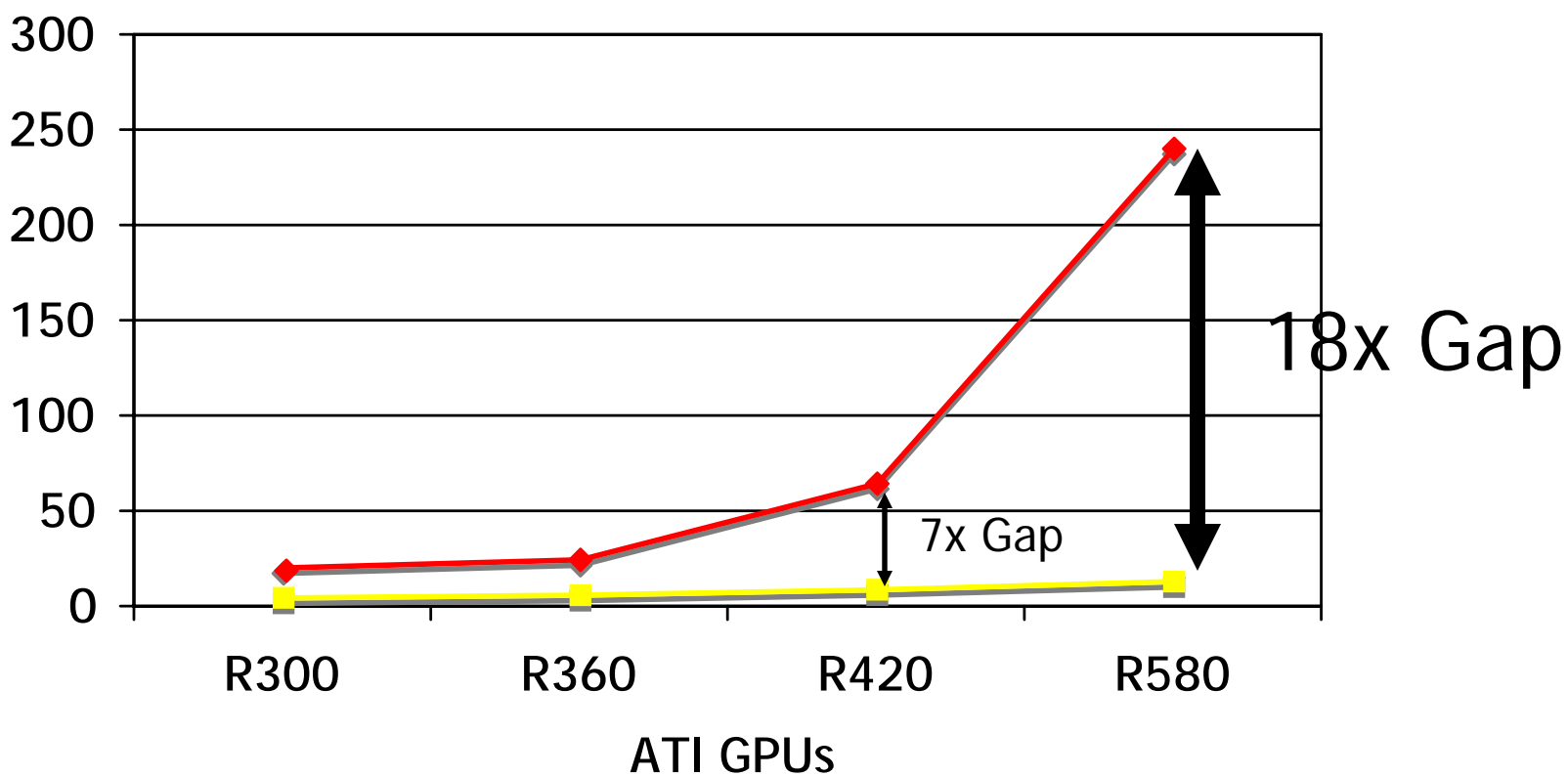
- Shading is compute intensive
  - 100s of floating point operations
  - Common case, output 1 32-bit color value
    - With MRTs: 16 32b FP outputs
- Compute to bandwidth ratio
  - arithmetic intensity



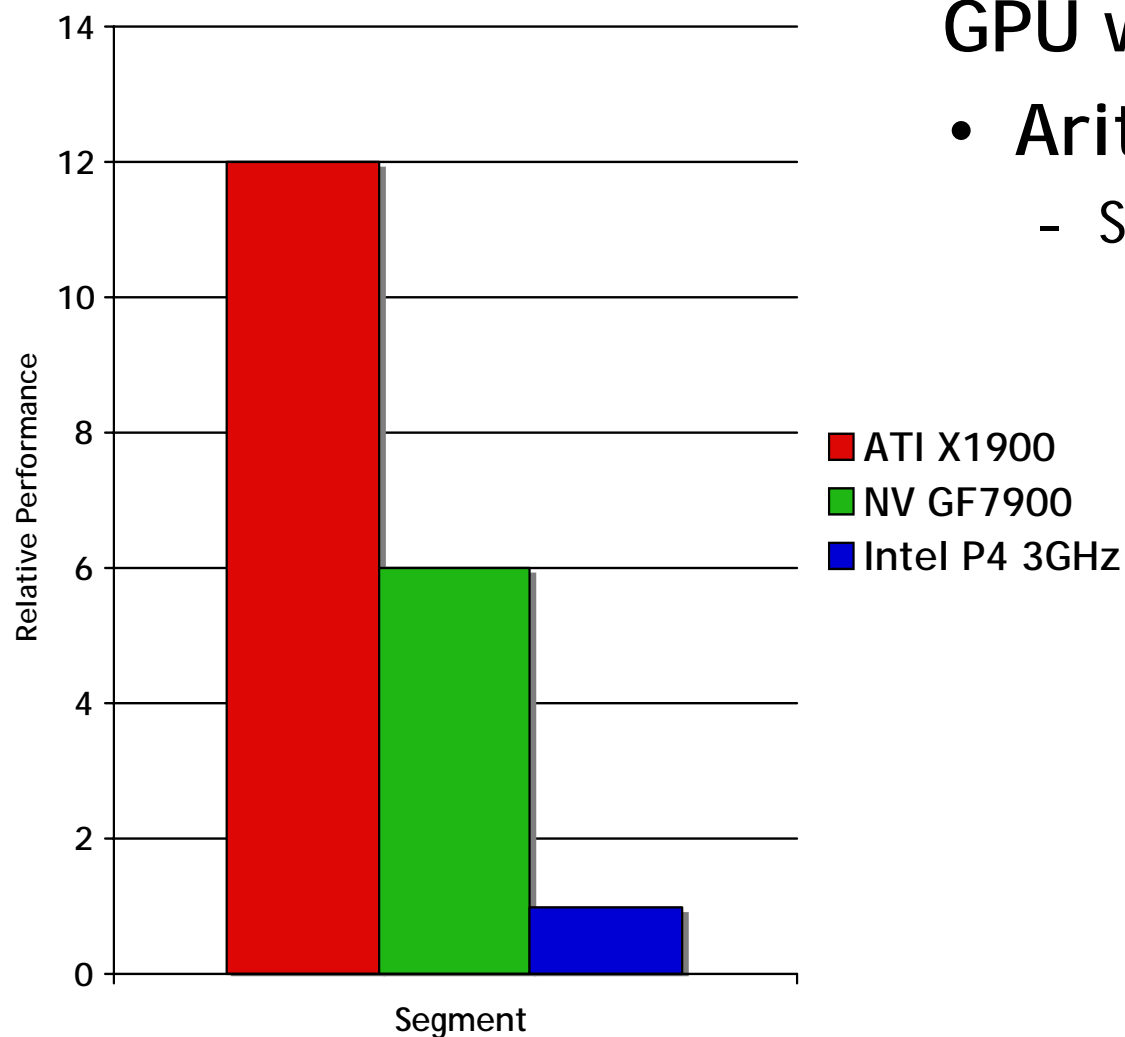
# Compute vs. Bandwidth



# Arithmetic Intensity



# Arithmetic Intensity

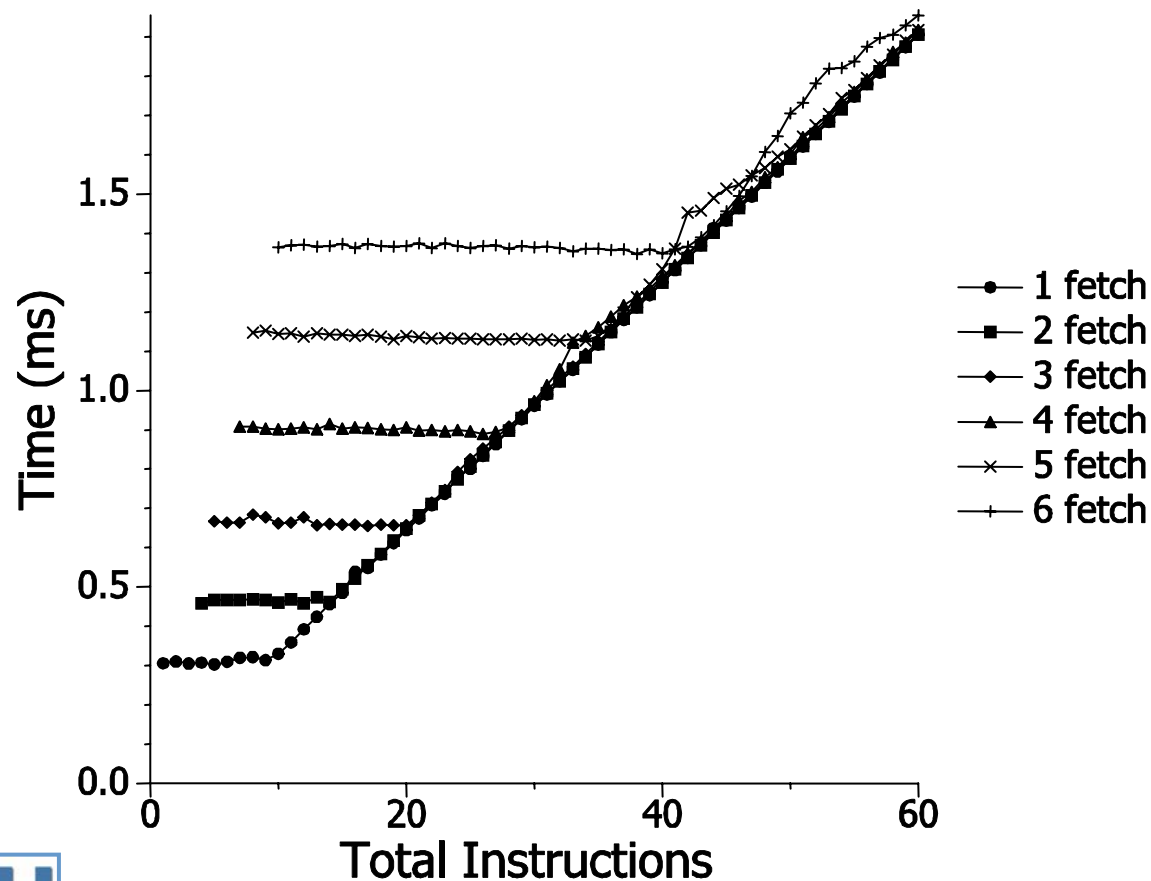


GPU wins when ...

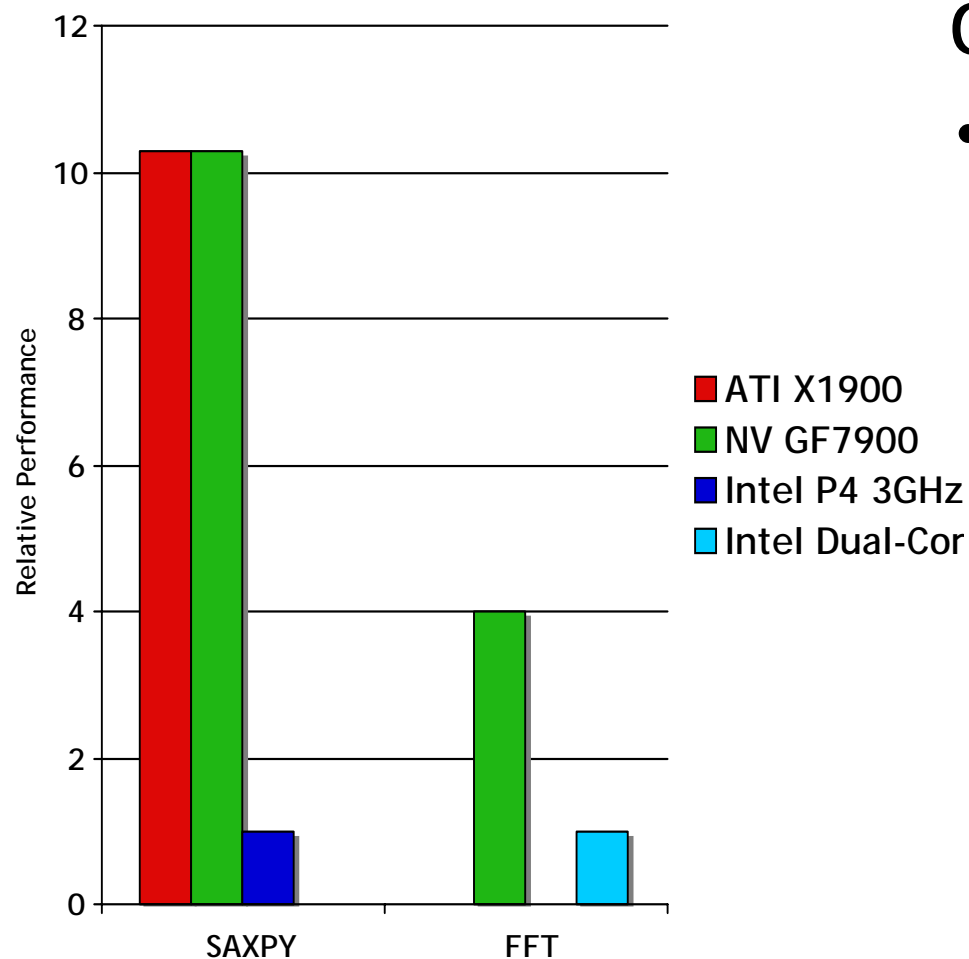
- Arithmetic intensity
  - Segment (PMLSeg):
    - 3.7 ops per word
    - 31.3 GFLOPS (ATI)

# Arithmetic Intensity

- Overlapping computation with communication



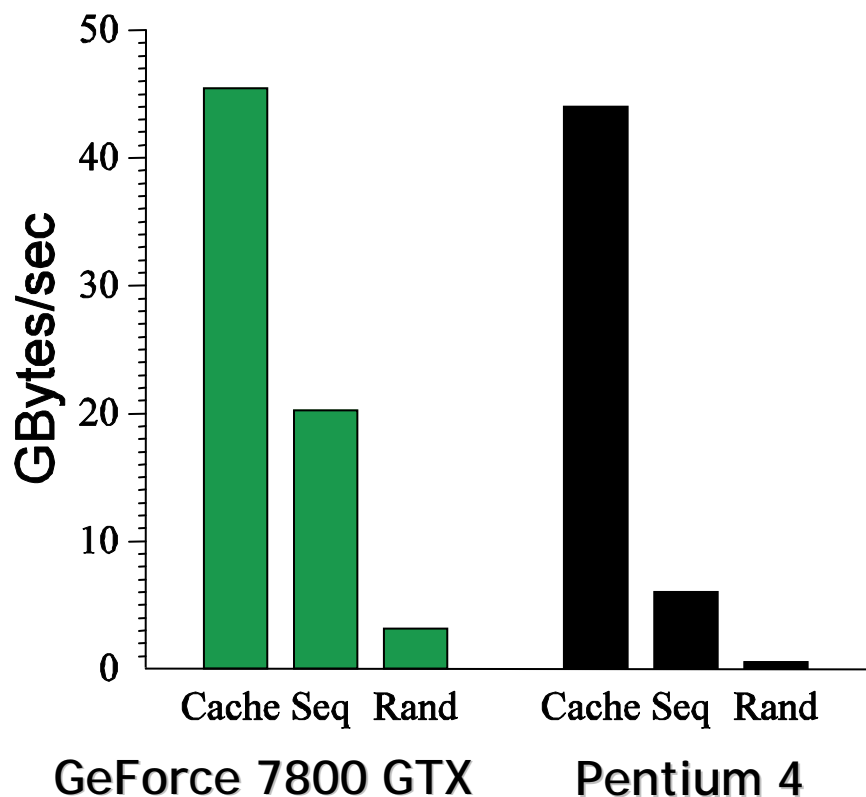
# Memory Bandwidth



GPU wins when ...

- Streaming memory bandwidth
  - SAXPY
  - FFT

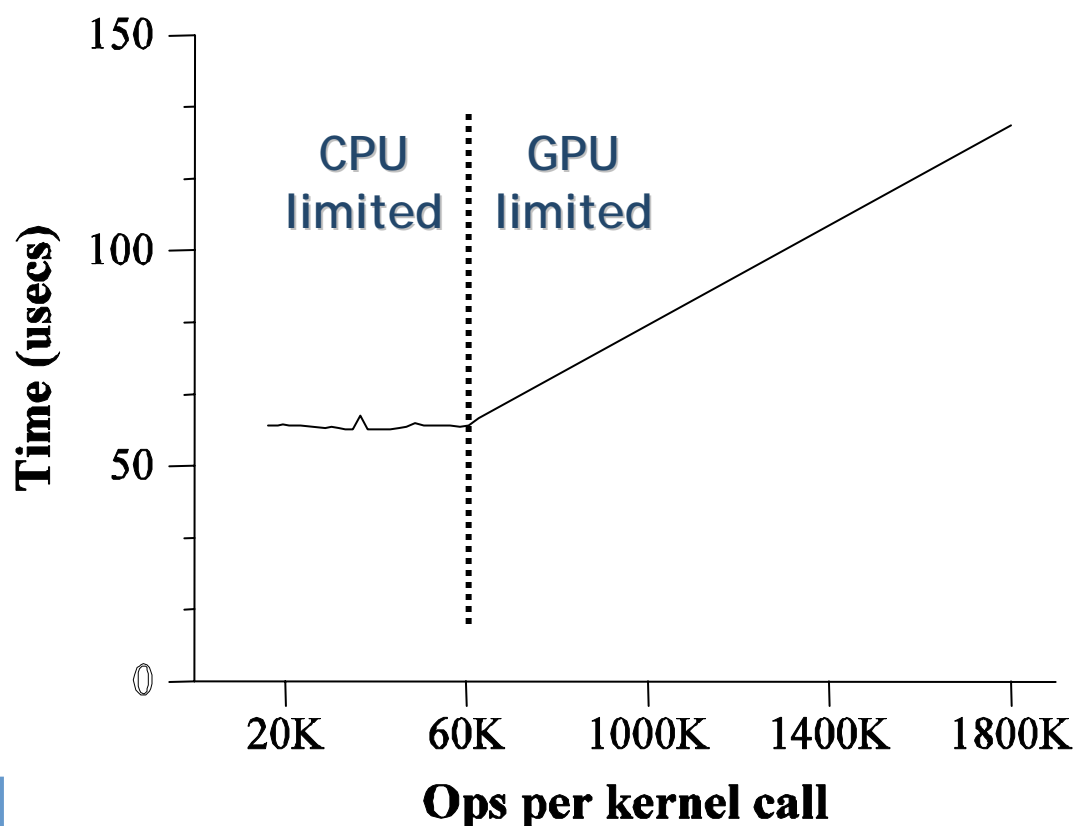
# Memory Bandwidth



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

# 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, ATI FP32**
  - s23e8
- **(Legacy) 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
FP32:	16,777,217
FP24:	131,073
FP16:	2,049
- **DX10 has both 32b FP and 32b integers**

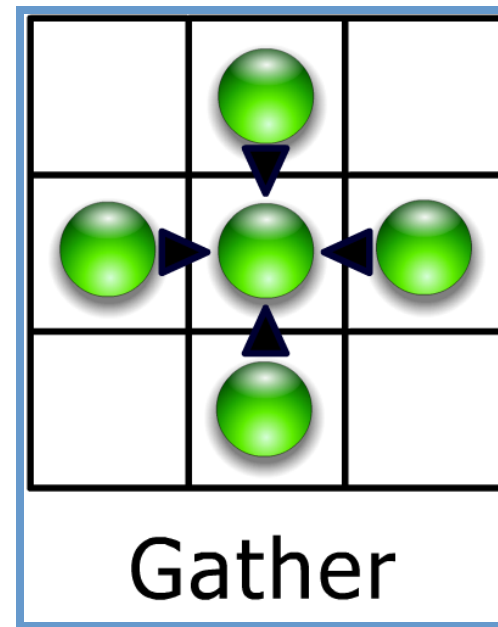
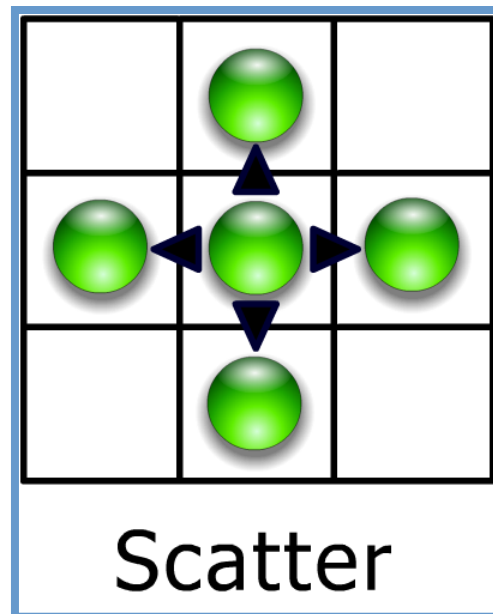
---

*Strategies & Tricks:*

# Scatter

# Scatter vs. Gather

- **Gather:**  $p = a[i]$ 
  - Vertex or Fragment programs
- **Scatter:**  $a[i] = p$ 
  - (DX9) Vertex programs only
  - Native for ATI CTM (DX9/10), NVIDIA CUDA (DX10)



# Scatter Techniques

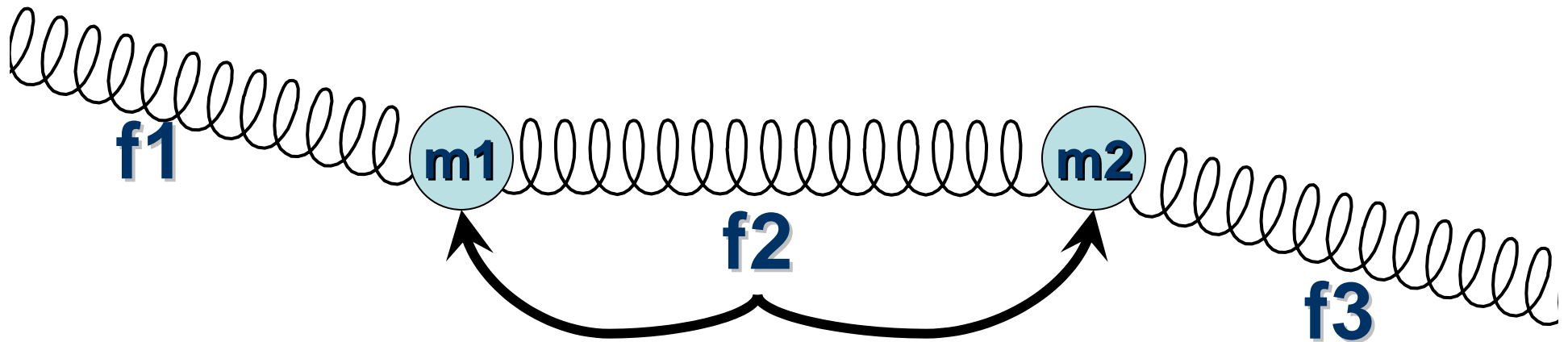
---

- **Problem:**  $a[i] = p$ 
  - Indirect write
  - Can't set the  $x, y$  of fragment in DX9 pixel shader
  - Often want to do:  $a[i] += p$
- **What about scatter-capable hardware?**
  - Still interesting because:
    - Scatter perhaps not highest performance
    - Scatter has unspecified semantics on collisions

# Scatter Techniques

- Solution 1: Convert to Gather

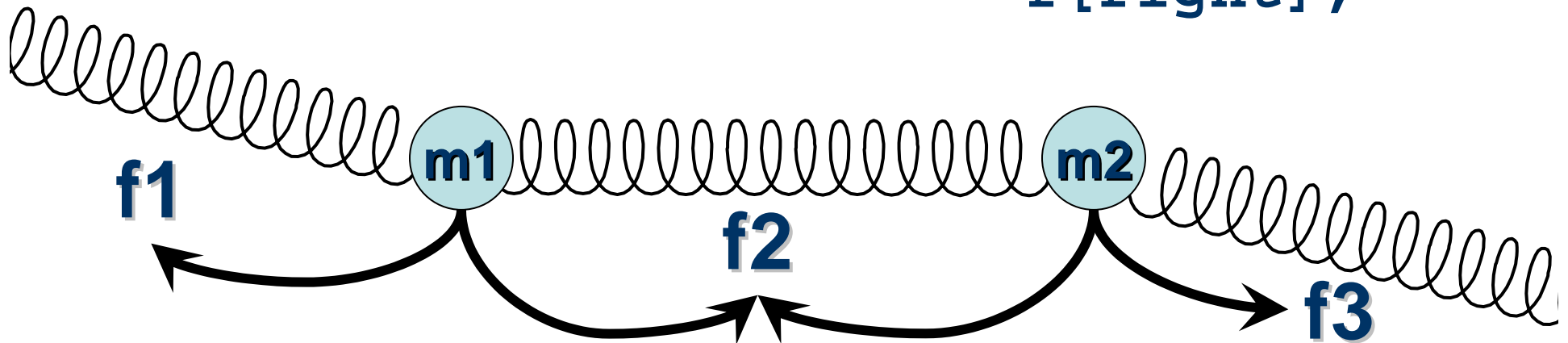
```
for each spring
  f = computed force
  mass_force[left] += f;
  mass_force[right] -= f;
```



# Scatter Techniques

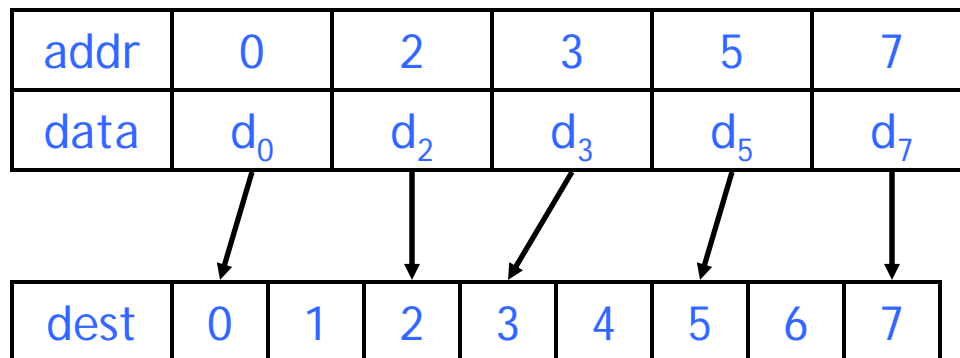
- Solution 1: Convert to Gather

```
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
      - For  $n$  items,  $O(\log n)$  passes



*Daniel Horn. Stream reduction operations for GPGPU applications. In GPU Gems 2, Mar. 2005, ch. 36, pp. 573-589.*

# 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

# Scatter Techniques

---

- **Solution 4: Native scatter in GPUs**
  - First sighted in Xbox 360 "Xenos" GPU ("MEMEXPORT", more general-purpose)
  - Currently only available using:
    - ATI CTM driver: can set (x,y) location for writes
      - Available in ATI's recent DX9 processors
    - NVIDIA CUDA on DX10 processors
  - Caveats:
    - Performance implications not yet clear
    - Uncached
    - Collisions result in undefined behavior

---

*Strategies & Tricks:*

# Conditionals

# Conditionals

---

- Problem:

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

- Above is *parallel* code
- Runs on every fragment
- GPU is good at running the same code over many fragments
- How does it handle conditionals?

# Conditionals

---

- Problem:

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

- Limited fragment shader conditional support
- Fragment shaders on GPUs are “SPMD” - single-program, multiple-data
  - Different fragments can diverge ...
    - Handled with predication - run both sides if necessary
  - ... but batches of fragments are executed in lockstep
    - Fragment program is SPMD, but batches are SIMD
- Conclusion: be smart about branching.

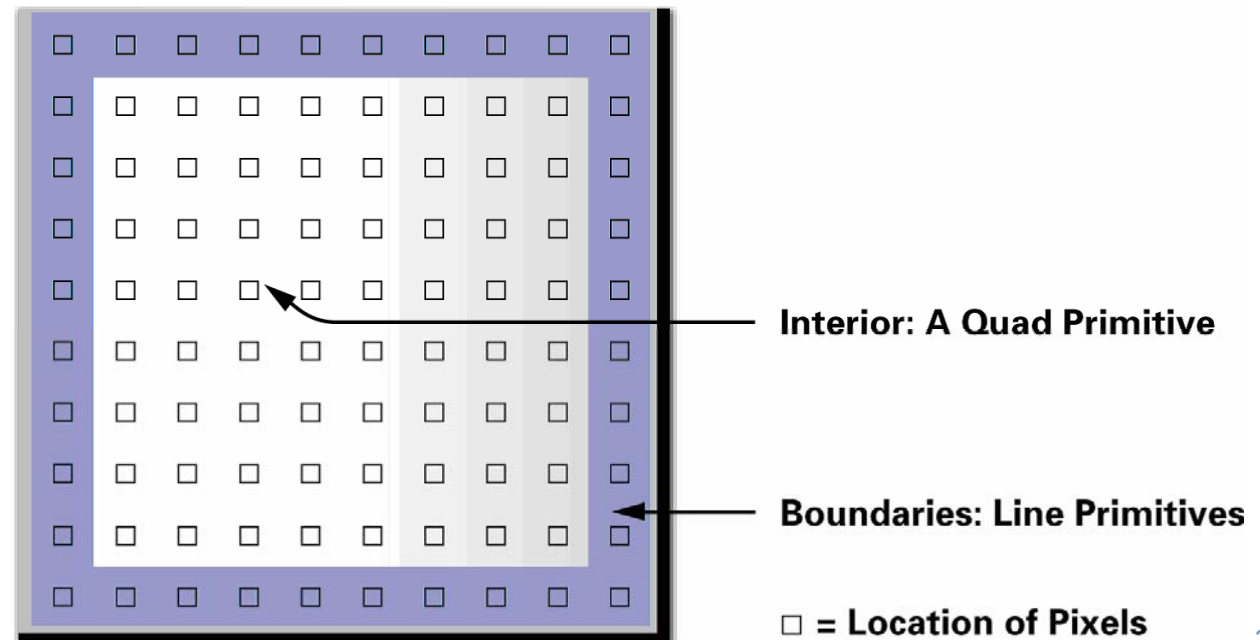
# Pre-computation

---

- 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
- Caveat: Requires CPU-GPU synchronization
  - Try to avoid waiting on synchronization
  - Issue multiple queries at once if possible

# Conditionals

---

- Predication

- Execute both
- f and g

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

- Use CMP instruction

- CMP b, -cond, f, g
- Executes all conditional code (both sides of branch)

# Conditionals

---

- Predication

- Use DP4 instruction

- DP4 b.x, a, f

- Executes all conditional code

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

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

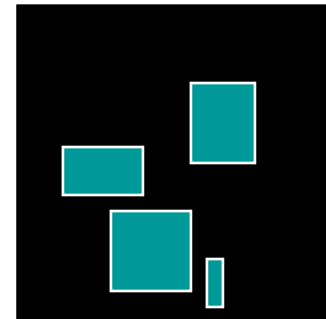
```
if (a.x) b = x;  
else if (a.y) b = y;  
else if (a.z) b = z;  
else if (a.w) b = w;
```

# Conditionals

---

- Using the depth buffer
  - Preset Z buffer to mark regions where you don't want computation
  - Z-test can prevent shader execution
    - glEnable(GL\_DEPTH\_TEST)
  - Both efficient and fine-grained

```
if cond {b = expensive();}
```



# Conditionals

---

- Using the depth buffer
  - Optimization disabled with:

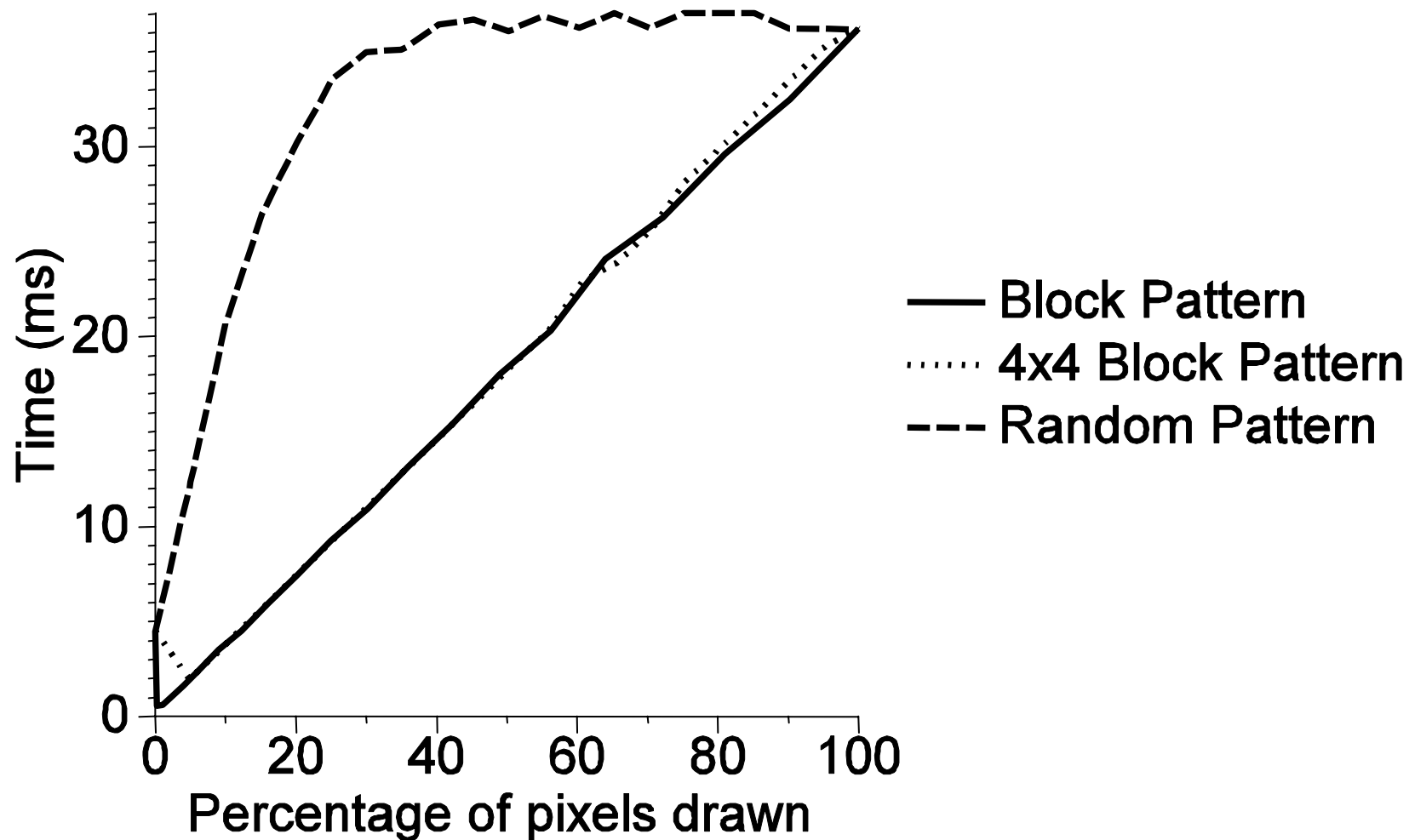
## ATI:

- 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 in PS3.0 pixel shader instruction set

```
MOVCC CC, R0;
```

```
IF GT.x;
```

```
MOV R0, R1; # executes if R0.x > 0
```

```
ELSE;
```

```
MOV R0, R2; # executes if R0.x <= 0
```

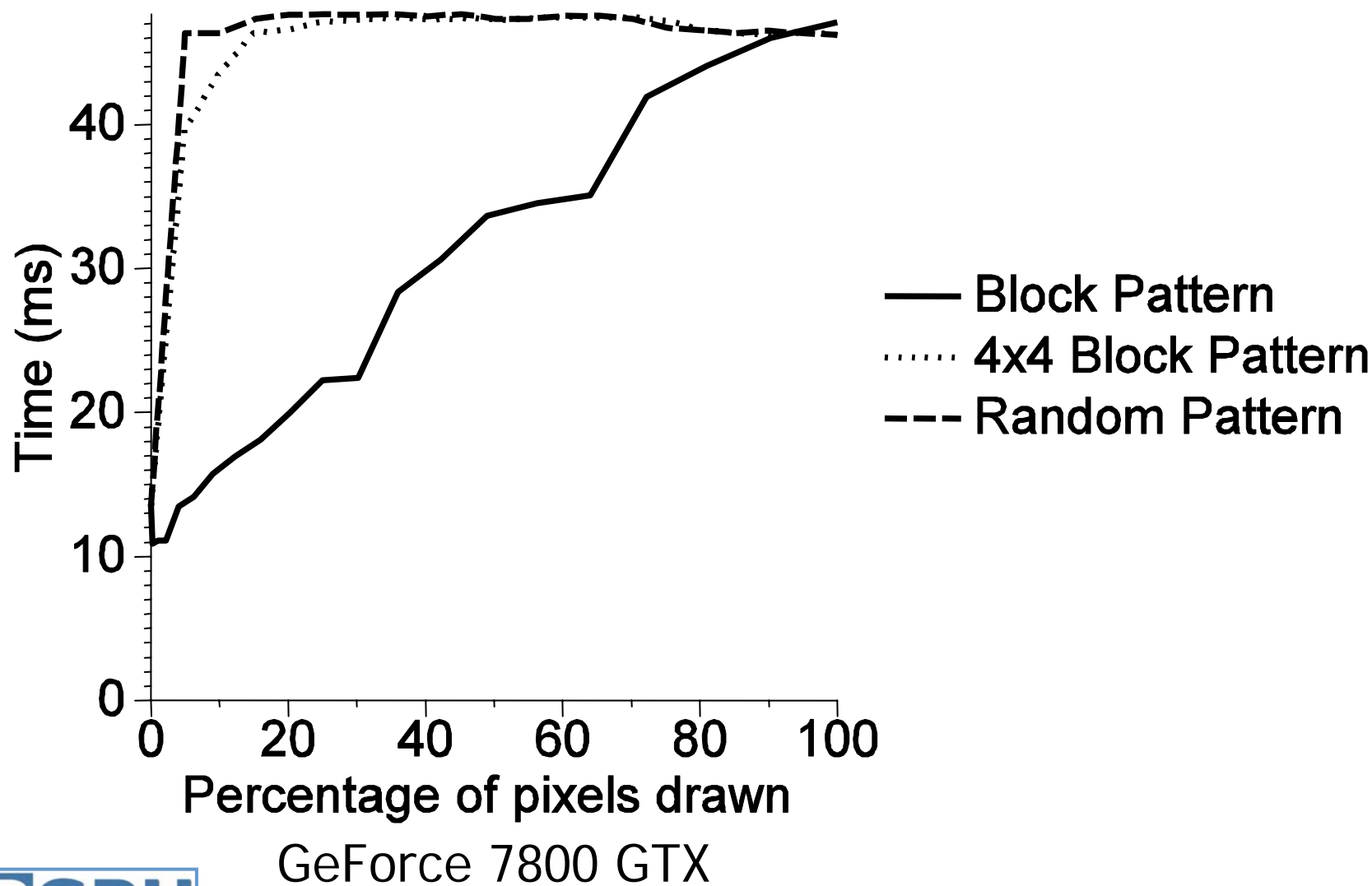
```
ENDIF;
```

# True SIMD branching

---

- **Lots of incoherent branching can hurt performance**
  - NVIDIA 7x00 has coherent regions of ~1000 pixels
    - That is only about 30x30 pixels, so still very useable!
  - ATI X1800-series GPUs and NVIDIA G80-series GPUs have ~16-pixel branch granularity
- **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**
  - Programmers want branches to have small granularity
  - Hardware wants branches to have large granularity
  - 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