GPU Architectures

A CPU Perspective

Derek Hower AMD Research 5/21/2013

Presented by **Jason Power**

Goals

Data Parallelism: What is it, and how to exploit it?
- Workload characteristics

Execution Models / GPU Architectures
- MIMD (SPMD), SIMD, **SIMT**

GPU Programming Models
- Terminology translations: CPU ←→ AMD GPU ←→ Nvidia GPU
- Intro to OpenCL

Modern GPU Microarchitectures
- i.e., programmable GPU pipelines, not their fixed-function predecessors

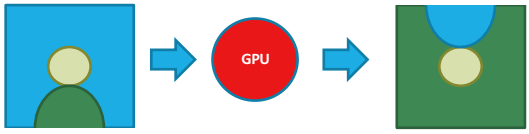
Advanced Topics: (Time permitting)
- **The Limits of GPUs:** What they can and cannot do
- **The Future of GPUs:** Where do we go from here?

Data Parallel Execution on GPUs

Data Parallelism, Programming Models, SIMT

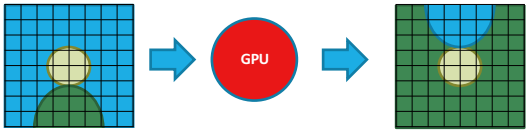
Graphics Workloads

Streaming computation



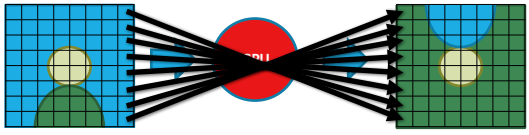
Graphics Workloads

Streaming computation ***on pixels***



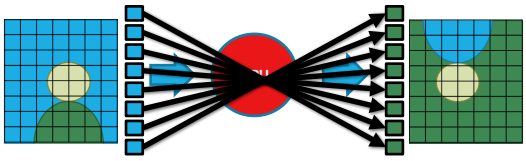
Graphics Workloads

Identical, Streaming computation ***on pixels***

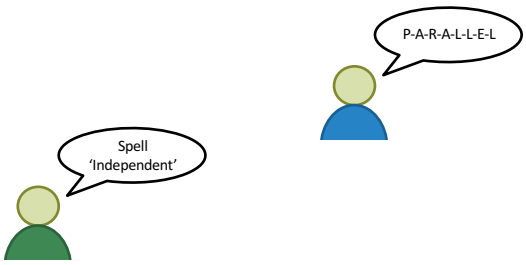


Graphics Workloads

Identical, Independent, Streaming computation ***on pixels***

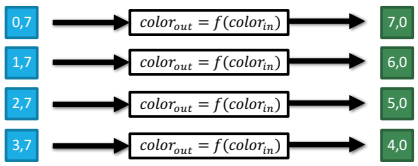


Architecture Spelling Bee



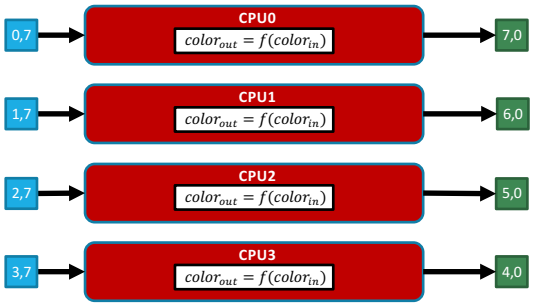
Generalize: Data Parallel Workloads

Identical, Independent computation ***on multiple data inputs***



Naïve Approach

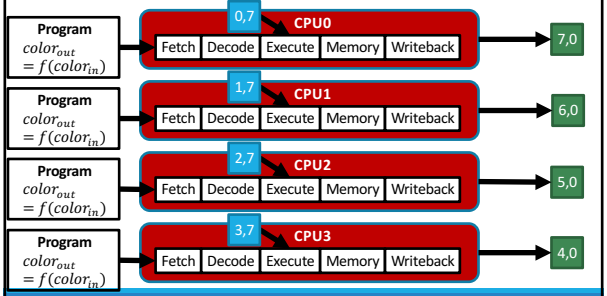
Split **independent** work over **multiple** processors



Data Parallelism: A MIMD Approach

Multiple **I**nstruction **M**ultiple **D**ata

Split **independent** work over **multiple** processors



Data Parallelism: A MIMD Approach

Multiple **I**nstruction **M**ultiple **D**ata

Split **independent** work over **multiple** processors

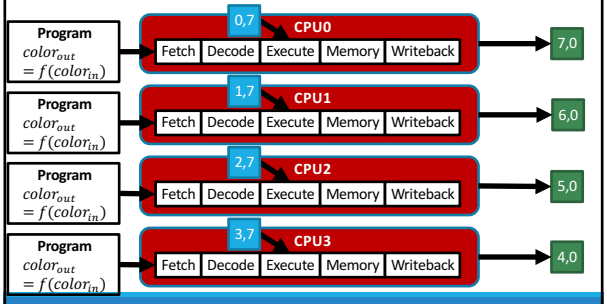
When work is **identical** (same program):

Single **P**rogram **M**ultiple **D**ata (**SPMD**)
(Subcategory of MIMD)

Data Parallelism: An SPMD Approach

Single Program Multiple Data

Split **identical, independent** work over **multiple** processors

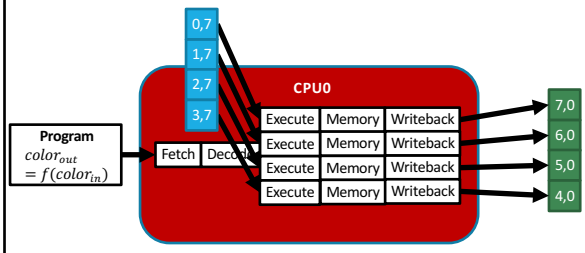


Data Parallelism: A SIMD Approach

Single Instruction Multiple Data

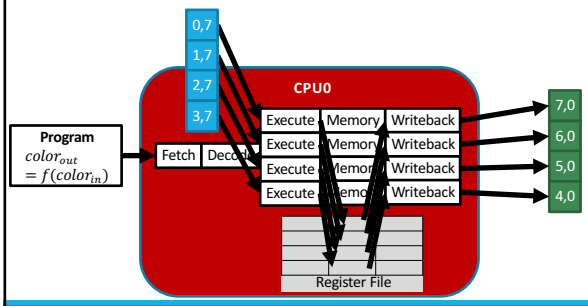
Split **identical, independent** work over **multiple** execution units (lanes)

More efficient: Eliminate redundant fetch/decode



SIMD: A Closer Look

One Thread + Data Parallel Ops → Single PC, single register file

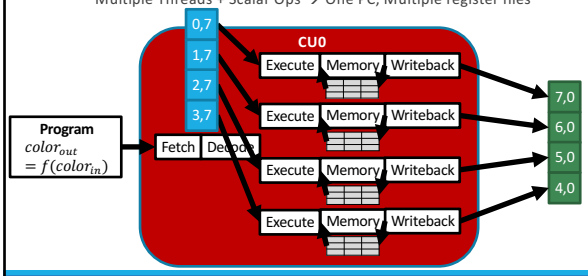


Data Parallelism: A SIMT Approach

Single Instruction Multiple Thread

Split **identical, independent** work over **multiple *lockstep*** threads

Multiple Threads + Scalar Ops → One PC, Multiple register files



Terminology Headache #1

It's common to interchange 'SIMD' and 'SIMT'

Data Parallel Execution Models

MIMD/SPMD

Multiple **independent** threads

SIMD/Vector

One thread with wide execution datapath

SIMT

Multiple **lockstep** threads

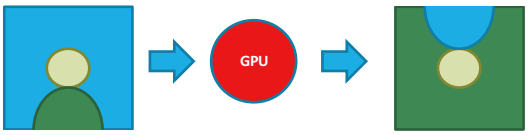
Execution Model Comparison

	MIMD/SPMD	SIMD/Vector	SIMT
			
Example Architecture	Multicore CPUs	x86 SSE/AVX	GPUs
Pros	More general: supports TLP	Can mix sequential & parallel code	Easier to program Gather/Scatter operations Can mix seq. & parallel (kinda)
Cons	Inefficient for data parallelism	Gather/Scatter can be awkward	Divergence kills performance

GPUs and Memory

Recall: GPUs perform ***Streaming*** computation →

Streaming memory access



DRAM latency: 100s of GPU cycles

How do we keep the GPU busy (*hide memory latency*)?

Hiding Memory Latency

Options from the CPU world:

~~Caches~~ ✗
- Need spatial/temporal locality

~~OoO/Dynamic Scheduling~~ ✗
- Need ILP

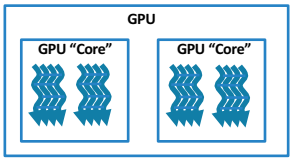
Multicore/Multithreading/SMT ✓
- Need independent threads

Multicore Multithreaded SIMT

Many SIMT "threads" grouped together into GPU "Core"

SIMT threads in a group ≈ SMT threads in a CPU core
- Unlike CPU, groups are exposed to programmers

Multiple GPU "Cores"



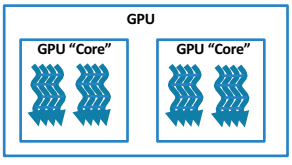
Multicore Multithreaded SIMT

Many SIMT "threads" grouped together into GPU "Core"

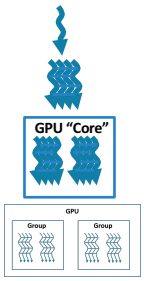
SIMT threads in a group ≈ SMT threads in a CPU core
- Unlike CPU, groups are exposed to programmers

Multiple GPU "Cores"

This is a GPU Architecture (Whew!)



GPU Component Names



AMD/OpenCL	Derek's CPU Analogy
Processing Element	Lane
SIMD Unit	Pipeline
Compute Unit	Core
GPU Device	Device

GPU Programming Models

OpenCL

GPU Programming Models

CUDA – **C**ompute **U**nified **D**evice **A**rchitecture
- Developed by Nvidia -- proprietary
- First serious GPGPU language/environment

OpenCL – **Open** **C**omputing **L**anguage
- From makers of OpenGL
- Wide industry support: AMD, Apple, Qualcomm, Nvidia (begrudgingly), etc.

C++ AMP – **C++** **A**ccelerated **M**assive **P**arallelism
- Microsoft
- Much higher abstraction that CUDA/OpenCL

OpenACC – **Open** **Acc**elerator
- Like OpenMP for GPUs (semi-auto-parallelize serial code)
- Much higher abstraction than CUDA/OpenCL

GPU Programming Models

CUDA – **C**ompute **U**nified **D**evice **A**rchitecture
- Developed by Nvidia -- proprietary
- First serious GPGPU language/environment

OpenCL – **Open** **C**omputing **L**anguage
- From makers of OpenGL
- Wide industry support: AMD, Apple, Qualcomm, Nvidia (begrudgingly), etc.

C++ AMP – **C++** **A**ccelerated **M**assive **P**arallelism
- Microsoft
- Much higher abstraction that CUDA/OpenCL

OpenACC – **Open** **Acc**elerator
- Like OpenMP for GPUs (semi-auto-parallelize serial code)
- Much higher abstraction than CUDA/OpenCL

OpenCL

Early CPU languages were light abstractions of physical hardware
- E.g., C

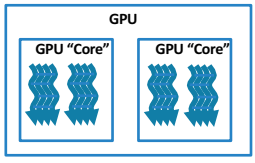
Early GPU languages are light abstractions of physical hardware
- OpenCL + CUDA

OpenCL

Early CPU languages were light abstractions of physical hardware
- E.g., C

Early GPU languages are light abstractions of physical hardware
- OpenCL + CUDA

GPU Architecture

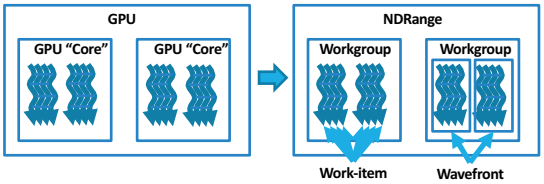


OpenCL

Early CPU languages were light abstractions of physical hardware
- E.g., C

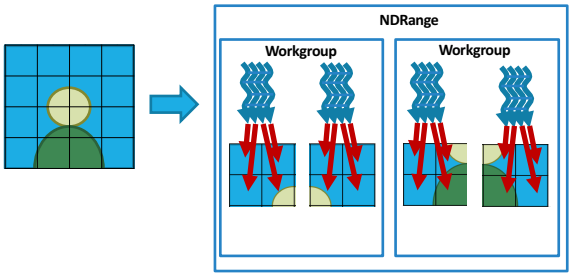
Early GPU languages are light abstractions of physical hardware
- OpenCL + CUDA

GPU Architecture **OpenCL Model**



NDRange

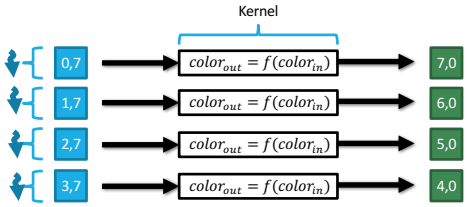
N-Dimensional (N = 1, 2, or 3) index space
- Partitioned into workgroups, wavefronts, and work-items



Kernel

Run an NDRange on a **kernel** (i.e., a function)

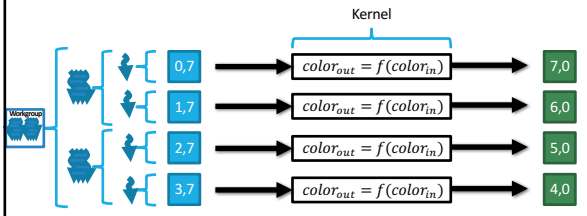
Same kernel executes for each work-item
- Smells like MIMD/SPMD



Kernel

Run an NDRange on a **kernel** (i.e., a function)

Same kernel executes for each work-item
- Smells like MIMD/SPMD...**but beware, it's not!**



OpenCL Code

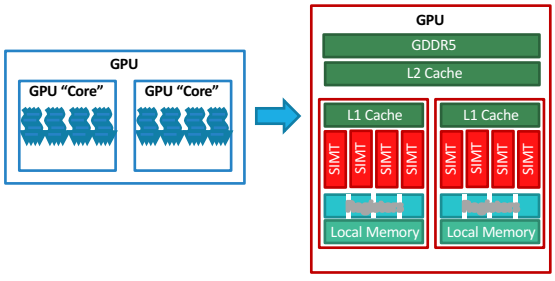
```
__kernel
void flip_and_recolor(__global float3 **in_image,
                      __global float3 **out_image,
                      int img_dim_x, int img_dim_y)
{
    int x = get_global_id(1); // get work-item id in dim 1
    int y = get_global_id(2); // get work-item id in dim 2

    out_image[img_dim_x - x][img_dim_y - y] =
       recolor(in_image[x][y]);
}
```


GPU Microarchitecture

AMD Graphics Core Next

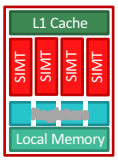
GPU Hardware Overview



Compute Unit – A GPU Core

Compute Unit (CU) – Runs *Workgroups*
- Contains 4 SIMT Units
- Picks one SIMT Unit per cycle for scheduling

SIMT Unit – Runs *Wavefronts*
- Each SIMT Unit has 10 wavefront instruction buffer
- Takes 4 cycles to execute one wavefront

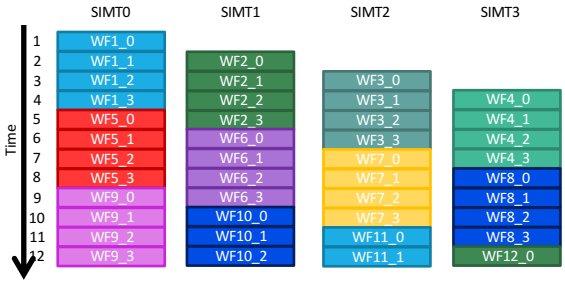


10 Wavefront x 4 SIMT Units =
40 Active Wavefronts / CU

64 work-items / wavefront x 40 active wavefronts =
2560 Active Work-items / CU

Compute Unit Timing Diagram

On average: fetch & commit one  wavefront / cycle



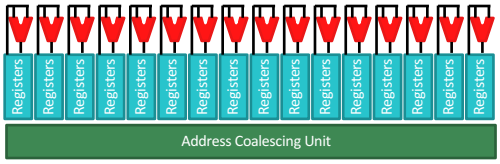
SIMT Unit – A GPU Pipeline

Like a wide CPU pipeline – except one fetch for entire width

16-wide physical ALU
- Executes 64-wavefront over 4 cycles. *Why??*

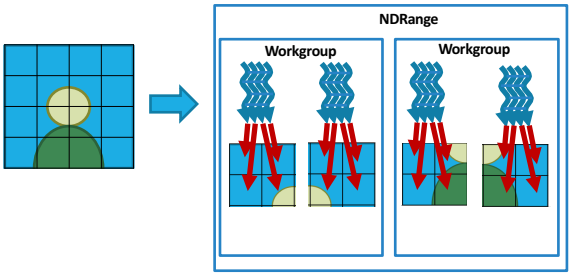
64KB register state / SIMT Unit
- Compare to x86 (Bulldozer): ~1KB of physical register file state (***~1/64 size***)

Address Coalescing Unit
- *A key to good memory performance*



Address Coalescing

Wavefront: Issue 64 memory requests



Address Coalescing

Wavefront: Issue 64 memory requests

Common case:
- work-items in same wavefront touch same cache block

Coalescing:
- Merge many work-items requests into single cache block request

Important for performance:
- Reduces bandwidth to DRAM

GPU Memory

GPUs have caches.

Not Your CPU's Cache

By the numbers: **Bulldozer – FX-8170** vs. **GCN – Radeon HD 7970**

	CPU (Bulldozer)	GPU (GCN)
L1 data cache capacity	16KB	16 KB
Active threads (work-items) sharing L1 D Cache	1	2560
L1 dcache capacity / thread	**16KB**	**6.4 bytes**
Last level cache (LLC) capacity	8MB	768KB
Active threads (work-items) sharing LLC	8	81,920
LLC capacity / thread	**1MB**	**9.6 bytes**

GPU Caches

Maximize throughput, not hide latency
- Not there for either spatial or temporal locality

L1 Cache: Coalesce requests to same cache block by different work-items
- i.e., streaming thread locality?
- Keep block around just long enough for each work-item to hit once
- Ultimate goal: **Reduce bandwidth to DRAM**

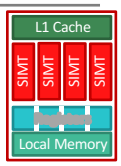
L2 Cache: DRAM staging buffer + some instruction reuse
- Ultimate goal: **Tolerate spikes in DRAM bandwidth**

If there is any spatial/temporal locality:
- Use local memory (scratchpad)

Scratchpad Memory

GPUs have scratchpads (Local Memory)
- Separate address space
- Managed by software:
  - Rename address
  - Manage capacity – manual fill/eviction

Allocated to a workgroup
- i.e., shared by wavefronts in workgroup



Example System: Radeon HD 7970

High-end part

32 Compute Units:
- 81,920 Active work-items
- 32 CUs * 4 SIMT Units * 16 ALUs = 2048 Max FP ops/cycle
- 264 GB/s Max memory bandwidth

925 MHz engine clock
- 3.79 TFLOPS single precision (accounting trickery: FMA)

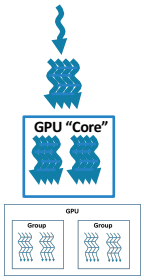
210W Max Power (Chip)
- >350W Max Power (card)
- 100W *idle power* (card)

Radeon HD 7990 - Cooking

Two 7970s on one card:
375W (AMD Official) – 450W (OEM)

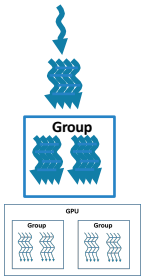
A Rose by Any Other Name…

Terminology Headaches #2-5



Nvidia/CUDA	AMD/OpenCL	Derek's CPU Analogy
CUDA Core	Processing Element	Lane
Core cluster?	SIMD Unit	Pipeline
Streaming Multiprocessor	Compute Unit	Core
GPU Device	GPU Device	Device

Terminology Headaches #6-9

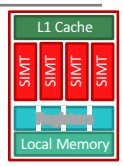


CUDA/Nvidia	OpenCL/AMD	Henn&Patt
Thread	Work-item	Sequence of SIMD Lane Operations
Warp	Wavefront	Thread of SIMD Instructions
Block	Workgroup	Body of vectorized loop
Grid	NDRange	Vectorized loop

Terminology Headache #10

GPUs have scratchpads (Local Memory)
- Separate address space
- Managed by software:
  - Rename address
  - Manage capacity – manual fill/eviction

Allocated to a workgroup
- i.e., shared by wavefronts in workgroup



Nvidia calls 'Local Memory' 'Shared Memory'. AMD sometimes calls it 'Group Memory'.

Recap

Data Parallelism: Identical, Independent work over multiple data inputs
- GPU version: Add streaming access pattern

Data Parallel Execution Models: MIMD, SIMD, SIMT

GPU Execution Model: Multicore Multithreaded SIMT

OpenCL Programming Model
- NDRange over workgroup/wavefront

Modern GPU Microarchitecture: AMD Graphics Core Next (GCN)
- Compute Unit ("GPU Core"): 4 SIMT Units
- SIMT Unit ("GPU Pipeline"): 16-wide ALU pipe (16x4 execution)
- Memory: designed to *stream*

GPUs: Great for data parallelism. Bad for everything else.

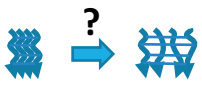
Advanced Topics

GPU Limitations, Future of GPGPU

SIMT Control Flow

Consider SIMT conditional branch:
- One PC
- Multiple data (i.e., multiple conditions)

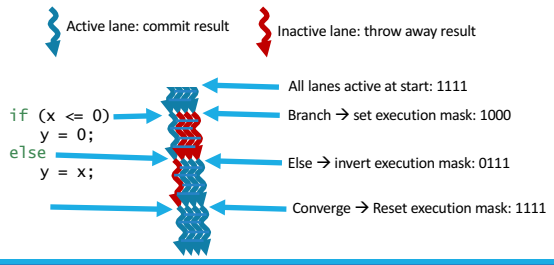
```
if (x <= 0)
   y = 0;
else
   y = x;
```



SIMT Control Flow

Work-items in wavefront run in lockstep
- *Don't all have to commit*

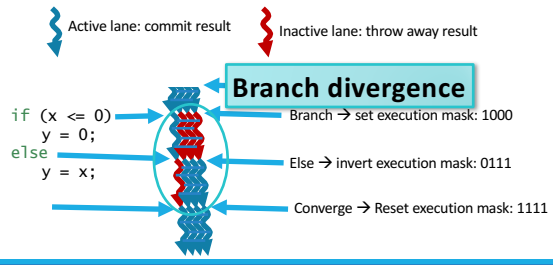
Branching through **predication**



SIMT Control Flow

Work-items in wavefront run in lockstep
- *Don't all have to commit*

Branching through **predication**



Branch Divergence

When control flow *diverges*, ***all lanes take all paths***

Divergence Kills Performance

Beware!

Divergence isn't just a performance problem:

```
__global int lock = 0;

void mutex_lock(…)
{
…
   // acquire lock
   while (test&set(lock, 1) == false) {
        // spin
   }
   return;
}
```


Beware!

Divergence isn't just a performance problem:

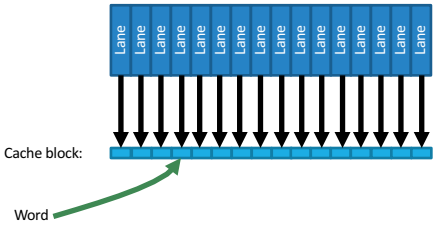
```
__global int lock = 0;

void mutex_lock(…)
```

Deadlock: work-items can't enter mutex together!

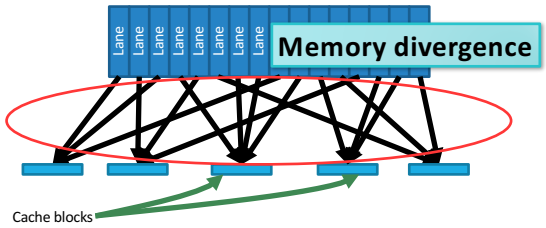
```
   // acquire lock
   while (test&set(lock, 1) == false) {
        // spin
   }
   return;
}
```


Memory Bandwidth



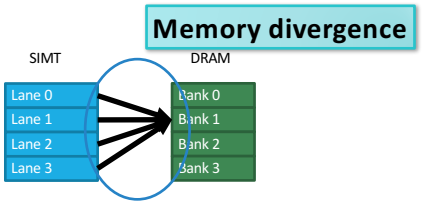
✓ -- Parallel Access

Memory Bandwidth



✘ -- Uncoalesced Access

Memory Bandwidth



✘ -- Sequential Access

Memory Divergence

One work-item stalls → entire wavefront must stall
- Cause: Bank conflicts, cache misses

Data layout & partitioning is important

Memory Divergence

One work-item stalls → entire wavefront must stall
- Cause: Bank conflicts, cache misses

Data layout & partitioning is important

Divergence Kills Performance

Communication and Synchronization

Work-items can communicate with:
- Work-items in same wavefront
  - No special sync needed…they are lockstep!
- Work-items in different wavefront, same workgroup (local)
  - Local barrier
- Work-items in different wavefront, different workgroup (global)
  - OpenCL 1.x: Nope
  - CUDA 4.x: Yes, but complicated

GPU Consistency Models

Very weak guarantee:
- Program order respected within single work-item
- All other bets are off

Safety net:
- Fence – "make sure all previous accesses are visible before proceeding"
- Built-in barriers are also fences

A wrench:
- GPU fences are *scoped* – only apply to subset of work-items in system
  - E.g., local barrier

Take-away: confusion abounds

GPU Coherence?

Notice: GPU consistency model does not require coherence
- i.e., Single Writer, Multiple Reader

Marketing claims they are coherent…

GPU "Coherence":
- Nvidia: disable private caches
- AMD: flush/invalidate entire cache at fences

GPU Architecture Research

Blending with CPU architecture:
- Dynamic scheduling / dynamic wavefront re-org
- Work-items have more locality than we think

Tighter integration with CPU on SOC:
- Fast kernel launch
  - Exploit fine-grained parallel region: Remember Amdahl's law
- Common shared memory

Reliability:
- Historically: Who notices a bad pixel?
- Future: GPU compute demands correctness

Power:
- Mobile, mobile mobile!!!

Computer Economics 101

GPU Compute is cool + gaining steam, but…
- Is a 0 billion dollar industry (to quote Mark Hill)

GPU design priorities:
1. Graphics
2. Graphics

   …

N-1. Graphics

N. GPU Compute

Moral of the story:
- GPU won't become a CPU (nor should it)