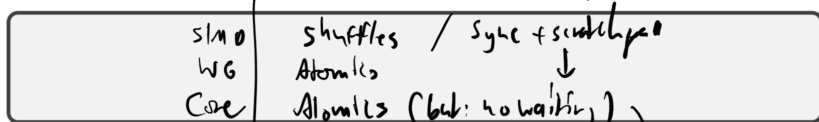
Announcements:

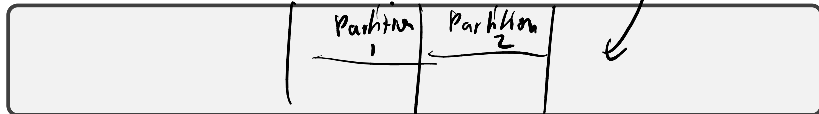
- Grid size 2D/3D
- HW3 + contest
- Grades
- HW2 answers
- Project!

GPU: Communication

What forms of communication exist at each scope?



Can we just do locking like we might do on a CPU?



indep. fw progress

also good for NUMA

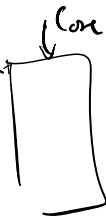
GPU Programming Model: Commentary

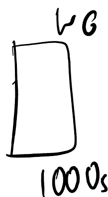
- "Vector" / "Warp" / "Wavefront"
  - Important hardware granularity
  - Poorly/very implicitly represented
- What is the impact of reconvergence?

Performance: Limits to Concurrency

What limits the amount of concurrency exposed to GPU hardware?

- num of scheduling slots
- size of the register file (variable-size per WI!)
- size of scratchpad
- ILP
- Group size





Memory Systems: Recap

Processor

Memory

D0..15

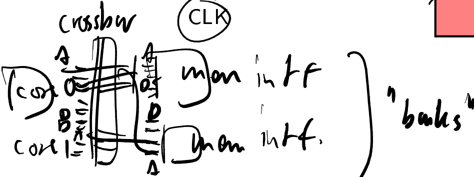
A0..15

$R/\bar{W}$

CLK

core0

core1



Parallel Memories

Problem: Memory chips have only one data bus.
So how can multiple threads read multiple data items from memory simultaneously?

- Split a wide bus (off-chip/global)
- Multiple multiplexed memory intf. (on-chip)

Where does banking show up?

- scratchpad
- register file

Memory Banking

Fill in the access pattern:

Bank

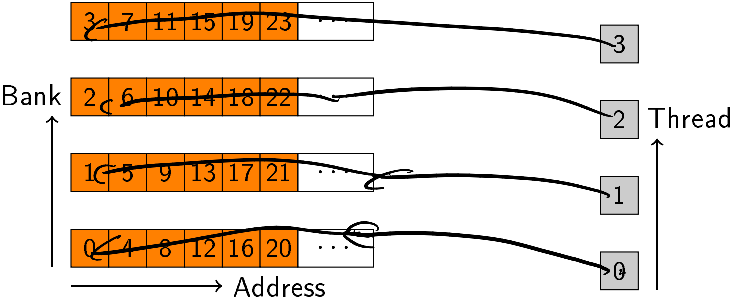
Address

SIMD lanes

~~Thread~~

Memory Banking

Fill in the access pattern:




```
local_variable[lid(0)]
```

Memory Banking

Fill in the access pattern:

3	7	11	15	19	23	· · ·
2	6	10	14	18	22	· · ·
1	5	9	13	17	21	· · ·
0	4	8	12	16	20	· · ·

Bank

Address

Thread: 3, 2, 1, 0

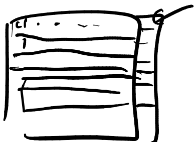
```
local_variable[BANK_COUNT*lid(0)]
```

4

Memory Banking

Fill in the access pattern:

```
local_variable[(BANK_COUNT+1)*lid(0)]
```



Memory Banking

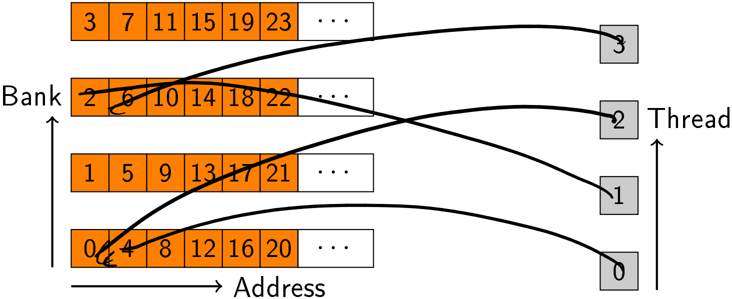
Fill in the access pattern:

Address

Thread

```
local_variable[ODD_NUMBER*lid(0)]
```

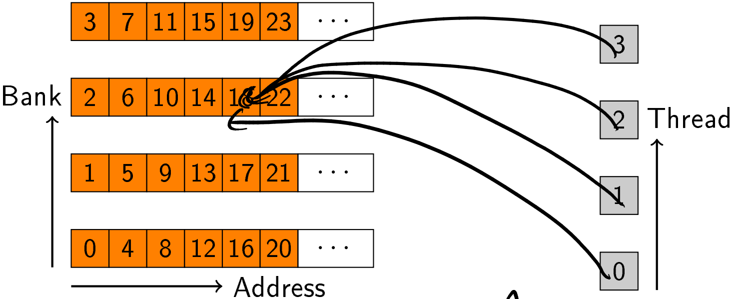
Memory Banking




```
local_variable[2*lid(0)]
```

Memory Banking

Fill in the access pattern:




```
local_variable[f(gid(0))]
```

Looks bad

But: important special case with hw help

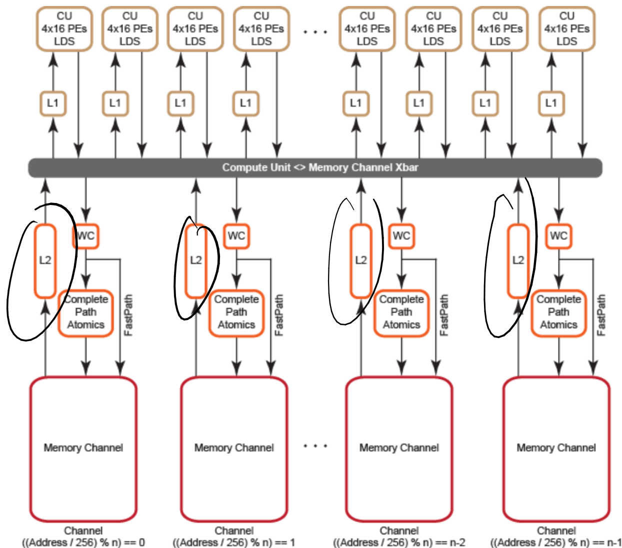
Memory Banking: Observations

- Factors of two in the stride: generally bad
- In a conflict-heavy access pattern, padding can help
  - Usually not a problem since scratchpad is transient by definition
- Word size (bank offset) may be adjustable (Nvidia) ← 32 bit / 64 bit

Given that unit strides are beneficial on global memory access, how do you realize a transpose?

load from global w/ unit stride
large stride access in scratchpad
store to g w/ unit stride

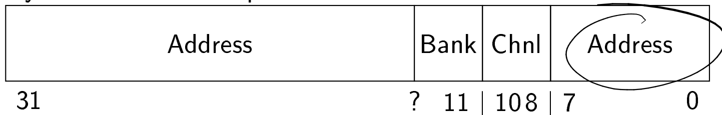
GPU Global Memory System



[GCN Optimization Manual, AMD]

GPU Global Memory Channel Map: Example

Byte address decomposition:



Implications:

- Transfers between compute unit and channel have granularity
  - Reasonable guess: warp/wavefront size $\times$ 32bits
  - Should strive for good utilization ('*Coalescing*')
- Channel count often *not* a power of two -> complex mapping
  - *Channel conflicts* possible
- Also *banked*
  - *Bank conflicts* also possible

GPU Global Memory: Performance Observations

Key quantities to observe for GPU global memory access:

- number of channels hit / lane stride
- group stride
- utilization

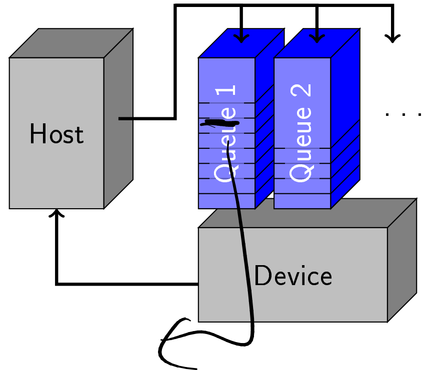
Are there any guaranteed-good memory access patterns?

- unit strides

- Need to consider access pattern *across entire device*
- *GPU caches:* Use for *spatial*, not for temporal locality
- Switch available: L1/Scratchpad partitioning
  - Settable on a per-kernel basis
- Since GPUs have meaningful caches at this point:
  Be aware of cache annotations (see later)

Host-Device Concurrency

- Host and Device run asynchronously
- Host submits to queue:
  - Computations
  - Memory Transfers
  - Sync primitives
  - ...
- Host can wait for:
  - *drained* queue
  - Individual "events"
- Profiling



Timing GPU Work

How do you find the execution time of a GPU kernel?

submit work
drain queue
start timer
submit work
drain queue
stop timer

How do you do this asynchronously?

markers

Host-Device Data Exchange

- Sad fact: Must get data onto device to compute
  - Transfers can be a bottleneck
  - If possible, overlap with computation
  - Pageable memory incurs difficulty in GPU-host transfers, often entails (another!) CPU side copy
  - “Pinned memory”: unpageable, avoids copy
    - Various *system-defined* ways of allocating pinned memory
- “Unified memory”:
  - GPU directly accesses host memory
  - “Fine grain”: Byte-for-byte coherent
  - “Coarse grain”: Per-buffer fences

Performance: Ballpark Numbers?

Bandwidth host/device:

Bandwidth on device:

Flop throughput?

Kernel launch overhead?