April 8, 2025

Announcements

- Project proposals
- HW3

Goals

Review

- Andreas hates GPUs!

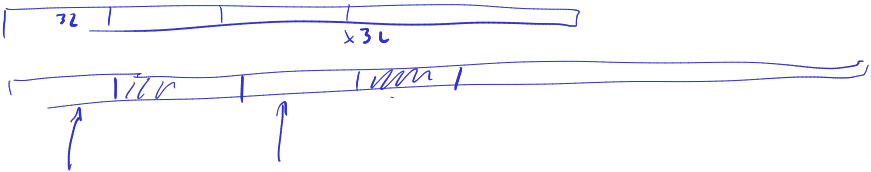
GPU Programming Model: Commentary

Advantages:

- Scalability
  - in cores, in lanes (?)
- Concurrency is part of the model

Disadvantages:

- Subgroup granularity
- Sync model is weak
- SIMD reconvergence?
- Changing widths in a kernel
- WG-wide ops implies


32
x3L

Performance: Limits to Concurrency

Occupancy / ILP

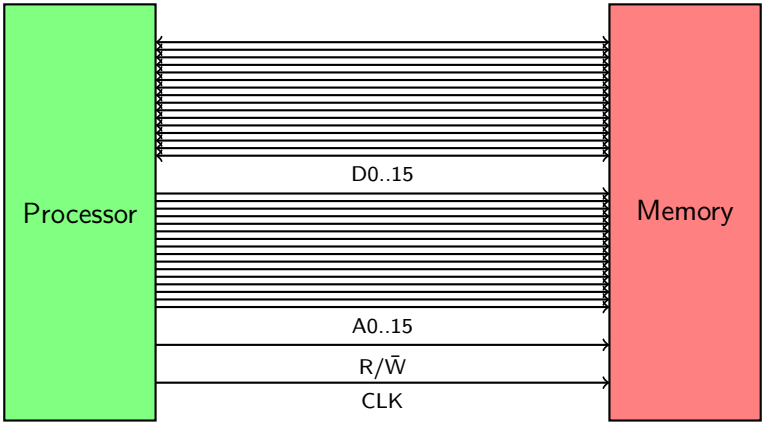
What limits the amount of concurrency exposed to GPU hardware?

- Synchronization (made worse by big groups)
- Divergence
- Register space (big groups steal flexibility from the machine)
- Amount of scratchpad
- #scheduler slots
- # in flight requests
- available in problem

- ILP

Hiding latency comes at the expense of state space

Memory Systems: Recap



Parallel Memories

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

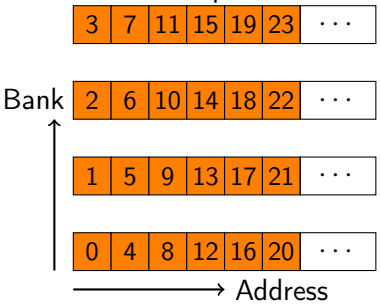
- share an address bus
- photocopy it ("banking") separate A and D bus

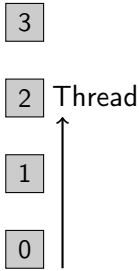
Where does banking show up?

- Scratchpad / local
- Registers
- Global

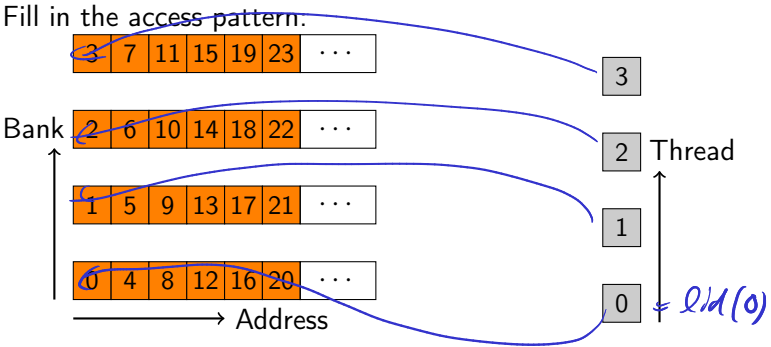
Memory Banking

Fill in the access pattern:



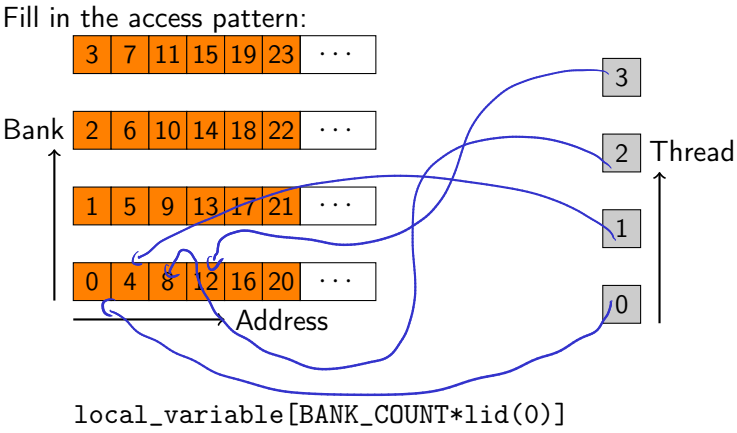


Memory Banking

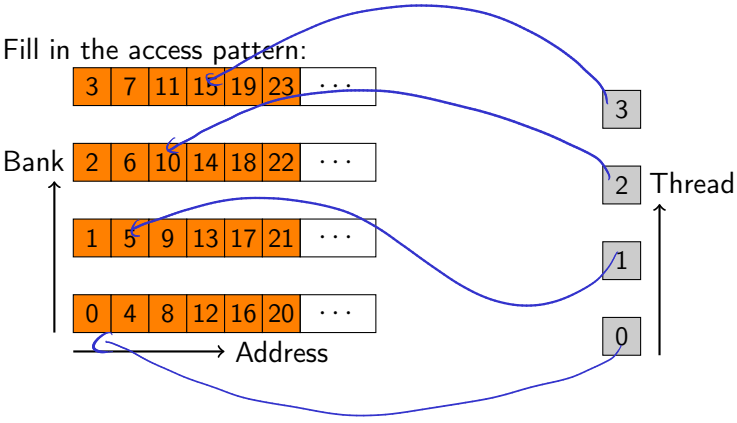



```
local_variable[lid(0)]
```

Memory Banking

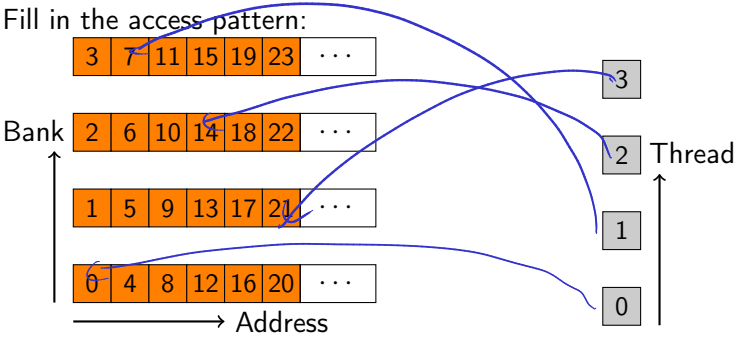


Memory Banking




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

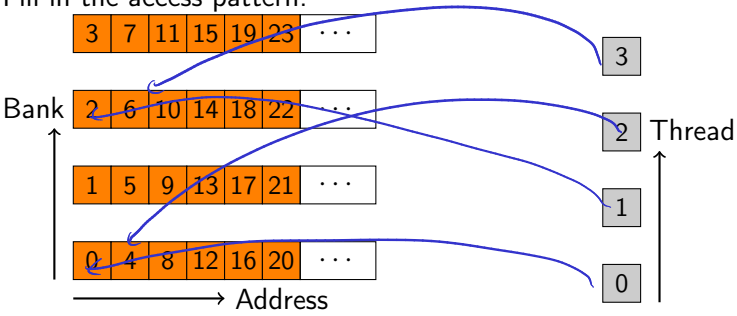
Memory Banking




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

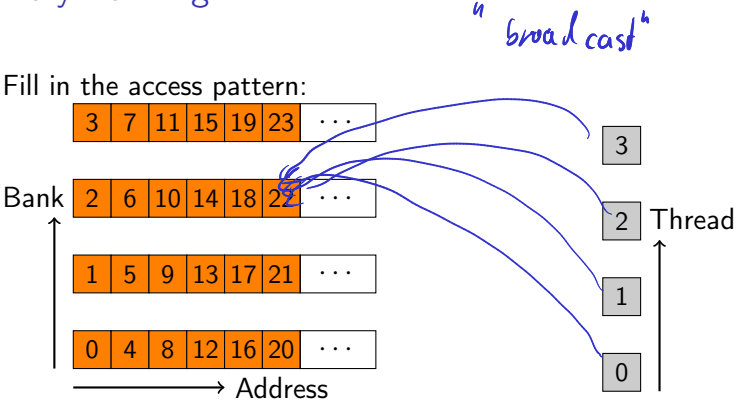
Memory Banking

Fill in the access pattern:




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

Memory Banking



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

Memory Banking: Observations

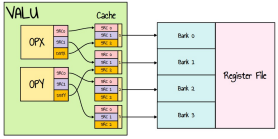
- Factors of two in the stride: generally bad
- In a conflict-heavy access pattern, padding can help
  - Usually not a problem since scratchpad is transient by definition
- Word size (bank offset) may be adjustable (Nvidia)

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

(old-timey)

From global w/ unit stride
Go to scratchpad with bad stride,
Read from scratchpad with 1 stride
to global w/ unit stride.

Memory Banking: AMD RDNA3 Registers



Manual dual-issue:

```
OpCodeX DSTX, SRCX0, SRCX1
    :: OpCodeY DSTY, SRCY0, SRCY1
```

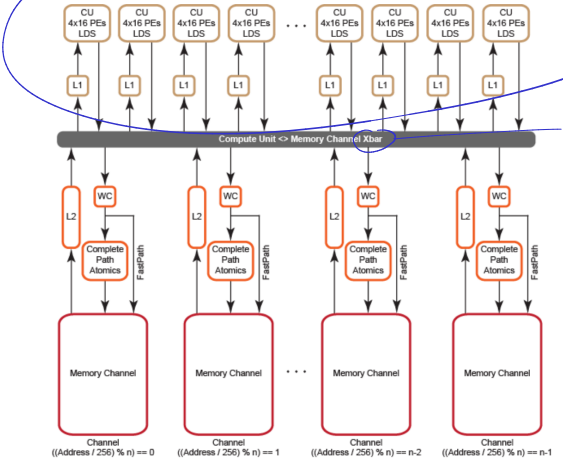
- Insns must be independent
- `SRCX0` and `SRCY0` must use different VGPR banks
- Dest VGPRs: one must be even and the other odd

RDNA3 specific:

- There are 4 VGPR banks (indexed by `SRC[1:0]`), and each bank has a cache.
- Each cache has 3 read ports: one dedicated to `SRC0`, one dedicated to `SRC1` and one for `SRC2`.
- A cache can read all 3 of them at once, but it can't read two SRC0's at once (or SRC1/2).

[Vince '25]

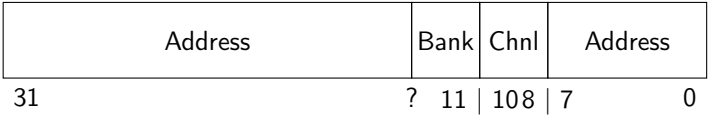
GPU Global Memory System



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:

- Stride
- Utilization

Are there any guaranteed-good memory access patterns?

Unit stride is happy stride

- 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
  - ...
  - *Batches* of these
    - *Mutable* batches of these
    - Nvidia: "CUDA Graphs"
    - OpenCL: "Command buffers"
- Host can wait for:
  - *drained* queue
  - Individual "events"
- Profiling