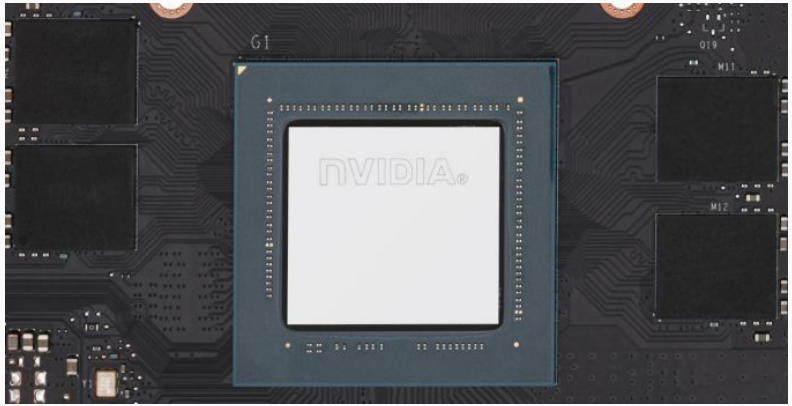
Energy-efficient Mechanisms for Managing ThreadContext in Throughput Processors

Mark Gebhart, Daniel R. Johnson, David Tarjan, Stephen W. Keckler, WIlliam J. Dally, Erik Lindholm, Kevin Skadron (UT, UIUC, NVIDIA, Stanford, UoV)
Presented by: Nick from CoffeeBeforeArch

Overview

Overview

- GPUs have complicated thread schedulers
  - Lots of thread contexts
- GPUs have large register files
  - Lots of threads!
- Exploit the fact registers are live for only a short window
  - Register File Cache (RFC)
- Limit the cost of the RFC
  - 2-level thread scheduler

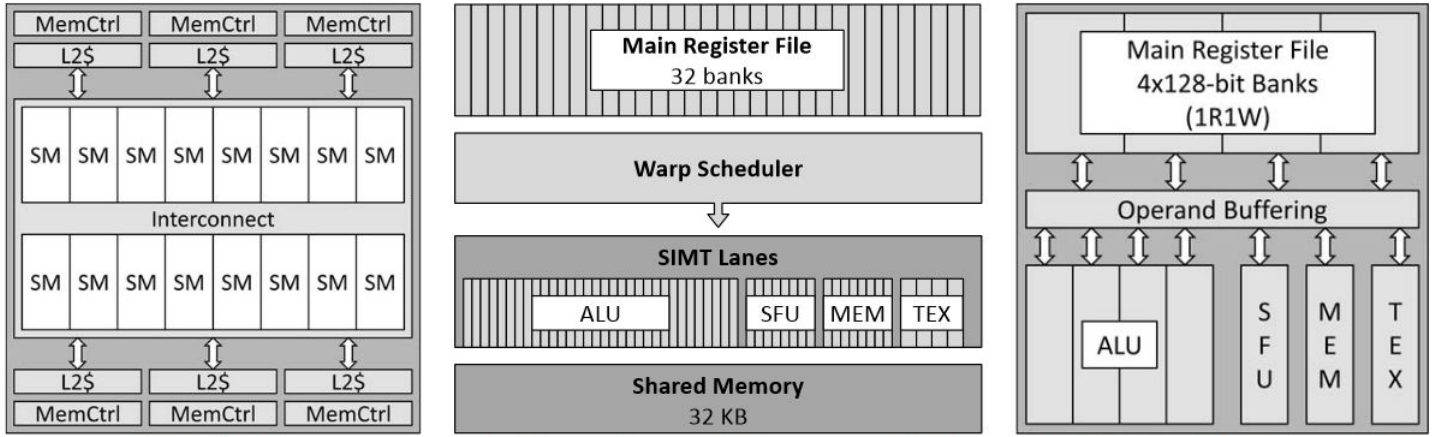


Background

Background

- GPUs exploit a fundamentally different type of parallelism
  - DLP vs. ILP
- Threads must be selected for execution
  - 2MB for total on-chip state (Fermi)
- Accessing large register file
  - Potential issue with power limitations

Background



(a) Full chip (b) Streaming multiprocessor (SM) (c) 4-Wide SIMT lane detail

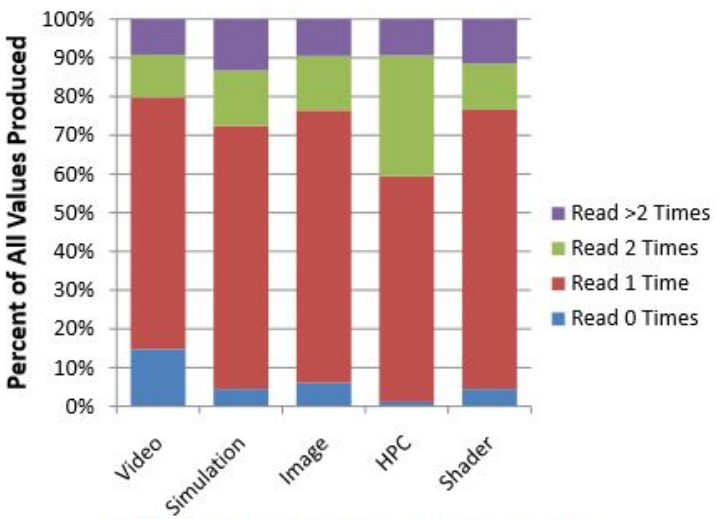
Figure 1: Contemporary GPU architecture.

Trends in Value Use

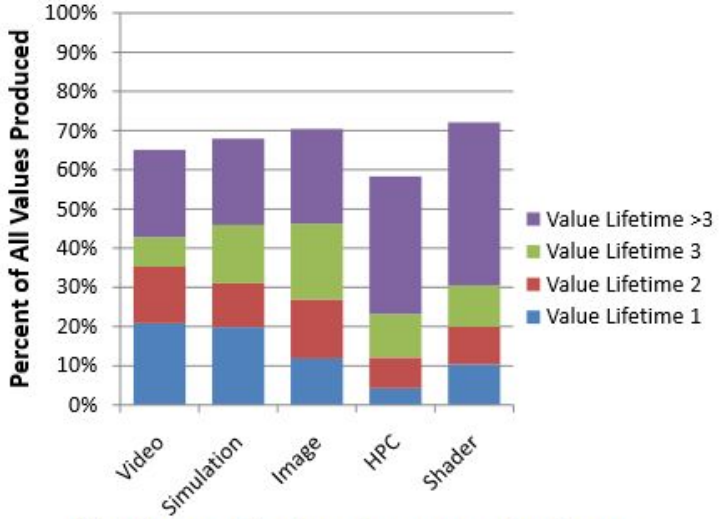
Trends in Value Use

- Only a fractional portion of registers are live at any given moment
  - Does this imply our RF is too big?
    - Later!
- Take samples of real world apps for...
  - Graphics
  - Computation
- Large portions of values are referenced only a single time
  - Reference is shortly after creation!

Trends in Value Use



(a) Number of reads per register value.



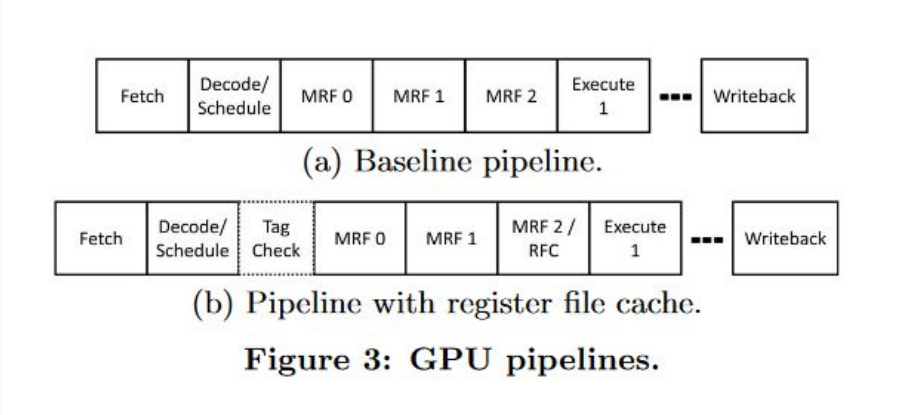
(b) Lifetime of values that are read only once.

Figure 2: Value usage characterization.

Register File Cache

Register File Cache

- Add another (small) cache for register file accesses
  - Reduce MRF accesses
- Allocation
  - Every result
- Replacement
  - FIFO
- Evictions
  - Don’t write back “dead” values
  - Compile-time knowledge



(a) Baseline pipeline.

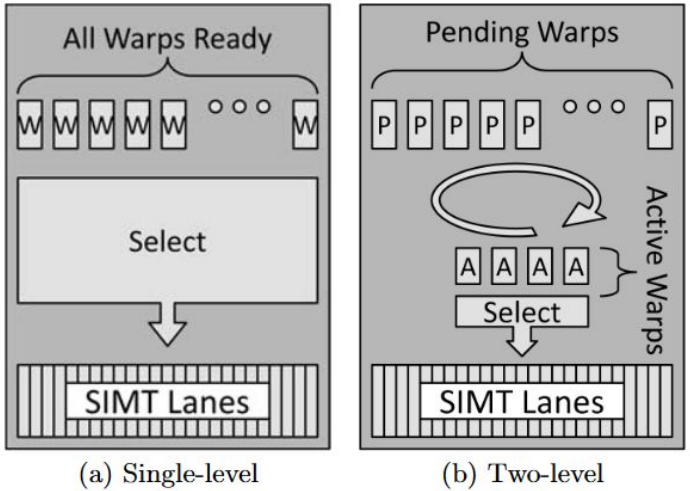
(b) Pipeline with register file cache.

Figure 3: GPU pipelines.

2-Level Scheduling

2-Level Scheduling

- Even a 6-entry RFC is is expensive!
  - 2MB in Fermi
- Limit the pool of threads we have to store at any given moment
- 2 Levels of threads
  - 1 Level to hide low-latency operations
  - 1 Level to high high-latency operations

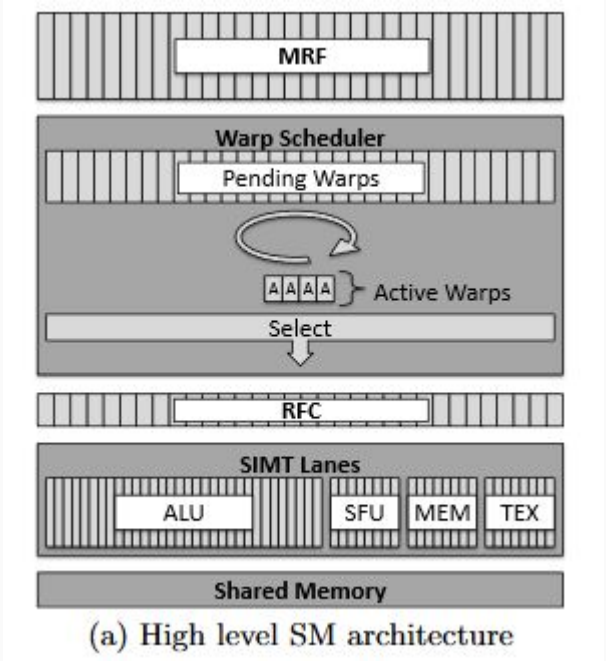


(a) Single-level (b) Two-level

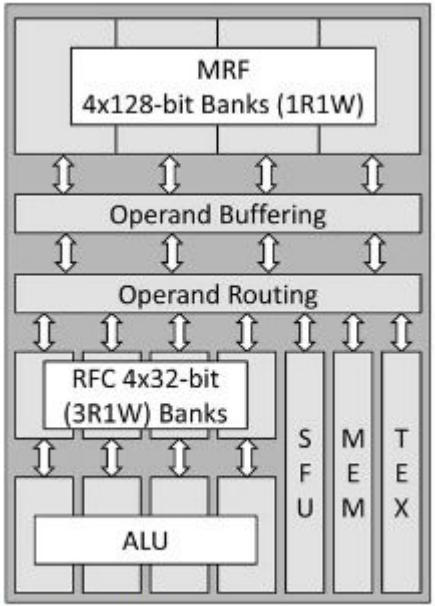
Figure 4: Warp schedulers.

High-Level Architecture

High-Level Architecture



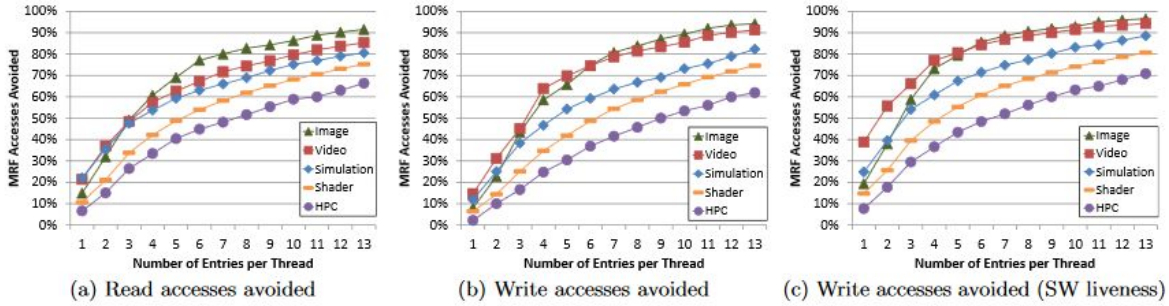
(a) High level SM architecture



(b) Detailed microarchitecture

Results

Results



(a) Read accesses avoided (b) Write accesses avoided (c) Write accesses avoided (SW liveness)

Figure 6: Reduction of MRF accesses by baseline register file cache.

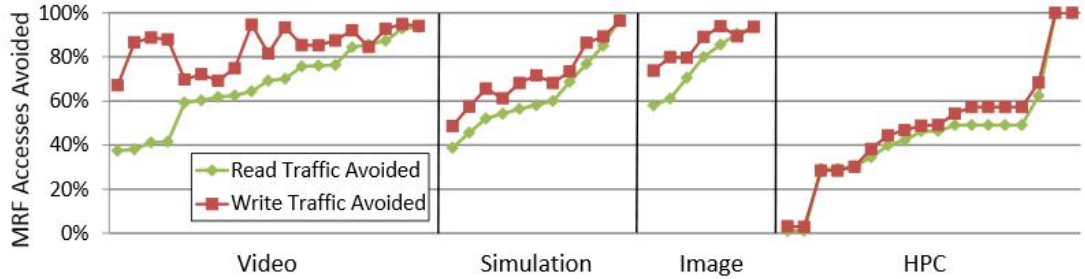
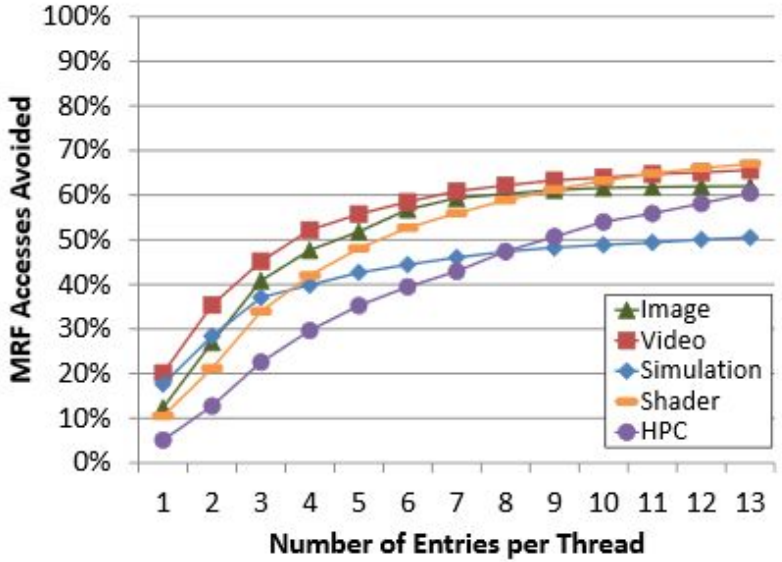
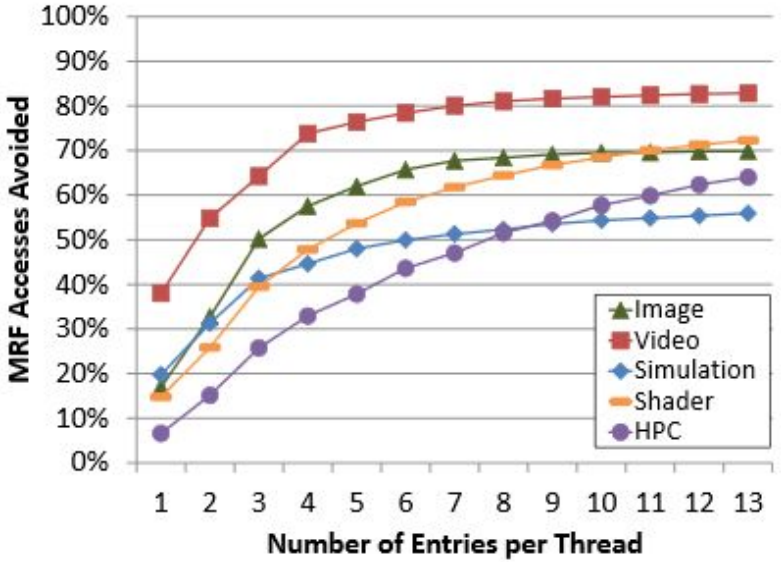


Figure 7: Per-trace reduction in MRF accesses with a 6 entry RFC per thread (one point per trace).

Results



(a) MRF read accesses avoided



(b) MRF write accesses avoided (SW liveness)

Figure 11: Effectiveness of RFC when combined with two-level scheduler.

Results

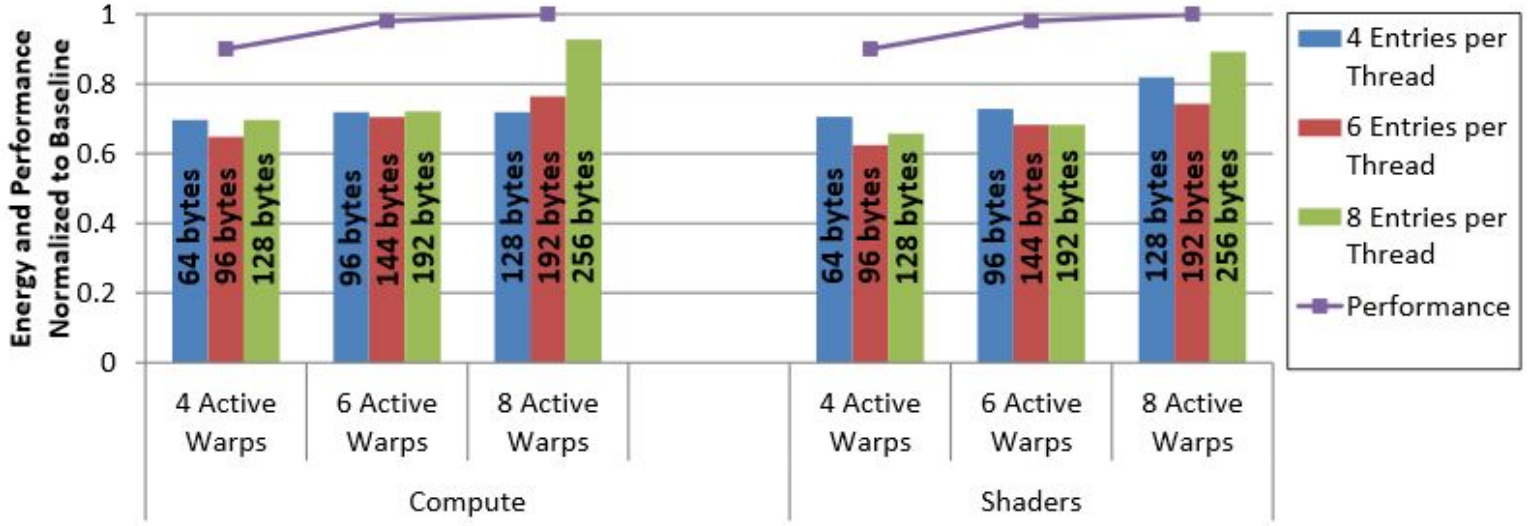


Figure 12: Energy savings due to MRF traffic reduction, bars show register file access energy consumed relative to baseline without RFC (lower is better), lines show performance (higher is better).

Conclusions

Conclusions

- Power is a concern for GPU scaling
- Can reduce RF energy by 36% (3.3 Watts)
- Easily extensible
  - Compiler optimizations

- And better yet...

Conclusions

- It exists today!

> Maxwell, however, provides something to make up for this and at the same time offers the capability to significantly reduce register bank traffic and overall chip power draw. This is the operand reuse cache. The operand reuse cache has 8 bytes of data per source operand slot. An instuction like FFMA has 3 source operand slots. Each time you issue an instruction there is a flag you can use to specify if each of the operands is going to be used again. So the next instruction that uses the same register *in the same operand slot* will not have to go to the register bank to fetch its value. And with this feature you can see how a register bank conflict can be averted.