GPU Performance Observability

Status and Opportunities for Improvement

Scalable Tools Workshop 2026

Topics

- **Assessing and attributing thread and wave utilization**
- **Monitoring and attributing memory access costs: problems and solutions**
  - loads/stores
  - TLB misses
  - Page faults
- **Monitoring and attributing data movement**
- **Benchmark suite**
- **Next steps**

Thread and Wave Utilization

- **Concerns: divergence and occupancy can change dynamically within a single kernel**
  - An if/else can cause thread divergence
  - Wave utilization may decrease in a computation if threads if many waves early return
- **AMD MI300**
  - Each “stochastic” HW sample reports bit masks identifying active waves and active threads
  - Enables precise attribution of thread/wave utilization attribution to code regions
    - Expected use cases
      - LLM mega kernels
      - LLNL’s Mercury Monte Carlo solver (e.g., 2,000 GPU functions, 100,000 lines in a single kernel)
  - HPCToolkit demo
    - Divergence
      - Slow: thread index mod 2 splits even/odd threads → 50% thread utilization
      - Fast: all threads in block take same path → 100% thread utilization
    - Occupancy
      - Slow: 1-thread block yields 1.6% utilization
      - Fast: 256-thread block yields 100% utilization
- **NVIDIA and Intel lack mechanisms for measuring and attributing fine-grain thread and wave utilization**

Monitoring Memory Accesses

- **How are outstanding memory accesses tracked today?**
  - AMD: explicit wait count instructions
  - NVIDIA: barrier bits embedded in instructions
  - Intel: scoreboard IDs assigned per load
- **Identifying problematic loads?**
  - NVIDIA and Intel: post-processing can help identify problematic loads
    - When stalled on an add using R3 + R4, hardware reports a stall but not which register's load caused it
    - Hardware has this information internally but does not expose it
    - GPA's post-processing heuristic: whichever load is closer (by instruction distance) is the likely cause
  - AMD waitcnt use lacks ability to identify problematic loads

Memory Stall Classification

- **Types of memory stalls**
  - Latency stalls (waiting for data)
  - Request issue stalls (pipeline full)
- **Today**
  - AMD
    - Latency stalls have no breakdown by memory domain
    - Can synthesize memory request stalls
      - NOT SELECTED/PIPE BUSY + pipeline kind to classify request stall kind (flat memory, LDS, texture)
      - Better for AMD to do this in rocprofiler-sdk not HPCToolkit
  - NVIDIA
    - Latency stalls are attributed by memory domain
    - Request issue stalls lack memory-type detail
  - Intel
    - Latency stalls only report scoreboard ID; no breakdown by memory domain
    - Request issue stalls: lack memory-type detail
- **What's missing**
  - No attribution of which level of the memory hierarchy satisfied a load
    - Cannot distinguish L2 hit vs. main memory vs. LDS at sample time
    - CPU equivalents (Intel PEBS, AMD IBS, IBM marked instruction sampling) do provide this
  - GPU HW support to directly identify uncoalesced accesses

Improving Identification of Slow GPU Memory Accesses?

- **AMD GPUs**
  - Improved compiler support could help
    - compiler could “macro-expand” a waitcnt so that there is one for each count level
      - a count level would then tracks an individual load
      - proposed by John M-C and Hsuan-heng Manrico ~3 years ago, but not acted upon by AMD
  - AMD HW could expose what current waitcnt is in a PC sample
    - supported by rocprofiler-sdk PC sampling design but not available on MI300A
  - Proposed AMD HW enhancement
    - Expose a bitmap of outstanding memory accesses at wait count instructions
      - Would show exactly which loads/stores still pending, not just a count
      - Thoughts
        - In-order global loads make this tractable
        - Tracking flat memory (LDS-capable) is more complex
- **NVIDIA GPUs**
  - API for analyzing GPU binaries would help post-mortem analysis

Improving Attribution of TLB Misses and Page Faults?

- **Memory access delays**
  - Translation Lookaside Buffer (TLB) miss kinds
    - Soft TLB miss: page table walk only
    - Hard miss: data not in memory, requires transfer from CPU
  - Page fault in unified memory (occurs in OpenMP code without data mapping pragmas)
- **Today**
  - AMD: high page fault rate detectable via TLB counters but not attributed to instruction
  - Intel: kernel driver separation means no source-line attribution
  - NVIDIA: TLB misses can report page address but not the PC that caused it
- **Gap**
  - Can correlate page address to an array allocation but not to a specific source line
  - Unified memory page faults: kernel-level attribution only, no source line
- **Workaround**
  - precise attribution may be less critical in simple kernels
  - high TLB miss rate often implies matrix transpose needed

Bandwidth Observability for Future Systems?

- **ORNL Discovery (2028)**
  - MI430X GPUs + next generation AMD EPYC CPU to be delivered 2028
  - Advertised as “bandwidth everywhere” for AI/HPC
- **Tools today**
  - Can measure bandwidth at kernel launch level (e.g., total Infinity Fabric usage per kernel)
  - No attribution inside a kernel to specific loop nests or source lines
  - SDK does not currently expose inter-chiplet or inter-die bandwidth metrics
- **Ask: attribute bandwidth utilization to source lines, or at minimum to loop nests within mega kernels**
  - Acknowledged as very challenging; two years before a realistic answer

GpuPerfSuite

- **Proposed community “bad/good GPU kernels” repository**
  - gitlab.com/hpctoolkit/GpuPerfSuite
  - Each benchmark demonstrates one anti-pattern with a known performance signature
  - Could serve as training data for LLM-based performance advisors
  - Reference: Cavazos et al. paper using hardware counter signatures to classify performance issues
  - CuPTI-based benchmark paper also mentioned as a source to mine (59 problem classes)
- **Draft benchmark set assembled last week using Claude**
  - Each covers one performance pathology (divergence, occupancy, coalescing, etc.)
  - Missing: page fault scenarios, unified memory abuse, bandwidth stress
  - translated HIP version to CUDA and SYCL via Claude
- **Overlap with concurrent session on tool overhead measurement**
  - standardization discussion ongoing

Next Steps

- **Collaborate on GpuPerfSuite**
  - Useful for tool validation and LLM training data
  - Enhancements
    - Add page fault and unified memory scenarios
      - Needed to test TLB miss and bandwidth attribution gaps
    - Mine CuPTI benchmark paper for additional problem classes
- **Fix LLVM attribution of optimized code to “line 0”**
  - Leverage AMD contacts with LLVM compiler team
  - Push for approximate line mapping
    - Example: common subexpression optimization
      - Today: combining subexpression computations from lines 4 and 10 maps to line 0
      - Wanted: combining subexpression computations from lines 4 and 10 maps to line 4 or 10

References

- **John Cavazos, Grigori Fursin, Felix Agakov, Edwin Bonilla, Michael F. P. O'Boyle, and Olivier Temam. 2007. Rapidly Selecting Good Compiler Optimizations using Performance Counters. In Proceedings of the International Symposium on Code Generation and Optimization (CGO '07). IEEE Computer Society, USA, 185–197. https://doi.org/10.1109/CGO.2007.32**
- **Yuyang Bi, Junming Cao, You Lu, Bihuan Chen, Tianwei Gan, Dingji Wang, and Xin Peng. 2026. Understanding Performance Problems in CUDA Programs. Proc. ACM Softw. Eng. 3, FSE, Article FSE016 (July 2026), 22 pages. https://doi.org/10.1145/3797143**