GPU PERFORMANCE ANALYSIS

Bob Crovella, 8/18/2020

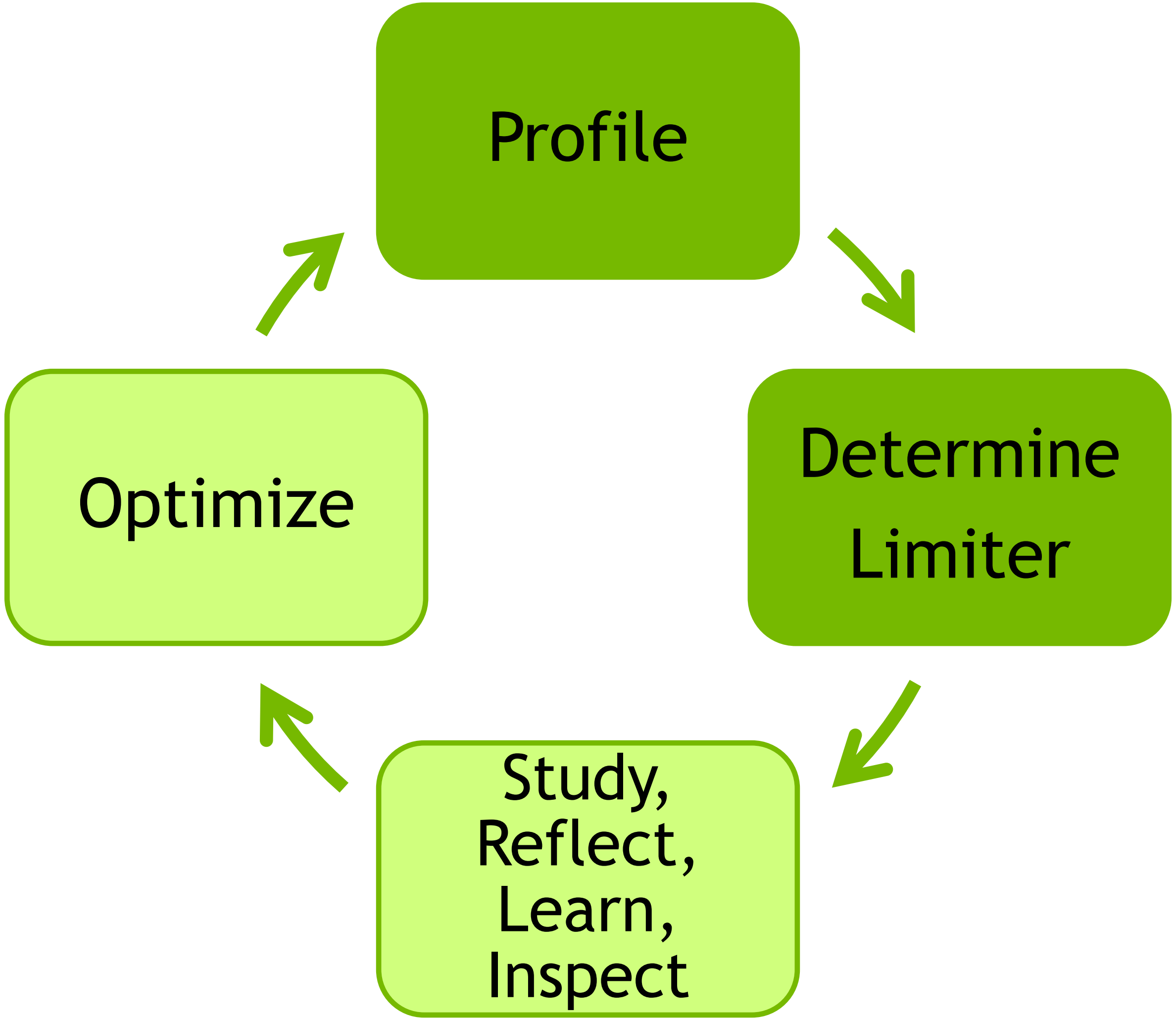
AGENDA

- Analysis Driven Optimization
- Understanding Performance Limiters
- Metrics Review
- Memory Bound Analysis
- Compute Bound Analysis
- Future Sessions
- Further Study
- Homework

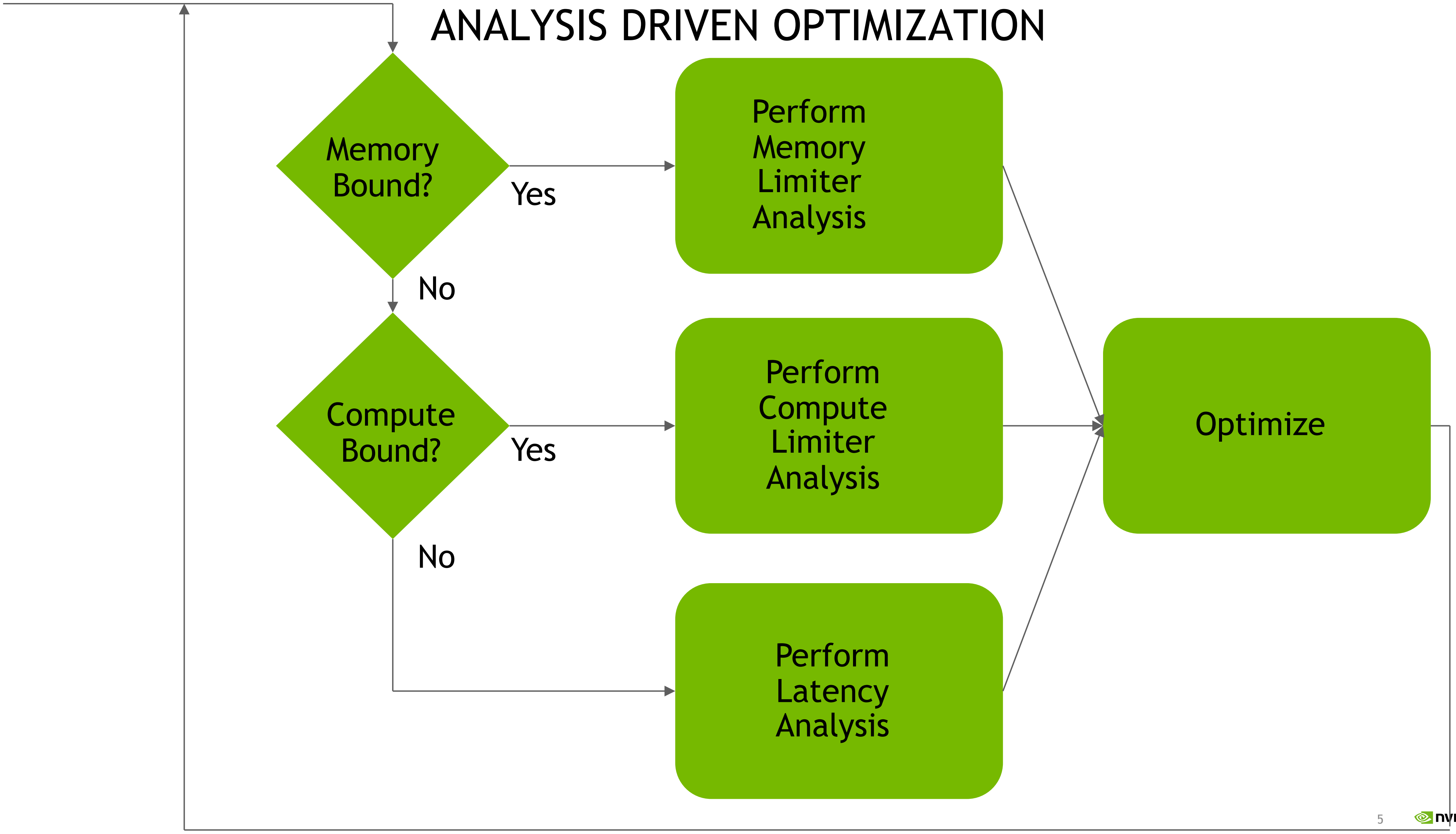
REVIEW: TOP-LEVEL PERFORMANCE CODING OBJECTIVES

- Make efficient use of the memory subsystem
  - Efficient use of global memory (coalesced access)
  - Intelligent use of the memory hierarchy
    - shared, constant, texture, caches, etc.
- Expose enough parallelism (work) to saturate the machine and hide latency
  - Threads/blocks
  - Occupancy
  - Work per thread
  - Execution efficiency

ANALYSIS DRIVEN OPTIMIZATION



ANALYSIS DRIVEN OPTIMIZATION



TOP-LEVEL PERFORMANCE BEHAVIOR - LIMITERS

- Memory Bound – A code is memory bound, when the measured memory system performance is at or close to the expected maximum. (saturate memory bus)
- Compute Bound – A code is compute bound when the compute instruction throughput is at or close to the expected maximum.
- Latency Bound – One of the indicators for a latency bound code is when neither of the above are true.
- (Analysis-driven) Optimization uses the above determination to direct code refactoring efforts in the first stage.
- Limiting behavior of a code may change over the duration of its execution cycle.
- It’s desirable to analyze small sections of code e.g. one kernel at a time

METRICS FOR DETERMINING COMPUTE VS. MEMORY BOUND

https://docs.nvidia.com/nsight-compute/NsightComputeCli/index.html#nvprof-metric-comparison

Latency metrics:
“sm efficiency”: smsp__cycles_active.avg.pct_of_peak_sustained_elapsed

Memory metrics:
“dram utilization”: dram__throughput.avg.pct_of_peak_sustained_elapsed
“L2 utilization”: lts__t_sectors.avg.pct_of_peak_sustained_elapsed
“shared utilization”:
l1tex__data_pipe_lsu_wavefronts_mem_shared.avg.pct_of_peak_sustained_elapsed

Compute metrics:
“DP Utilization”: smsp__inst_executed_pipe_fp64.avg.pct_of_peak_sustained_active
“SP Utilization”: smsp__pipe_fma_cycles_active.avg.pct_of_peak_sustained_active
“HP Utilization”: smsp__inst_executed_pipe_fp16.avg.pct_of_peak_sustained_active
“TC Utilization”: sm__pipe_tensor_op_hmma_cycles_active.avg.pct_of_peak_sustained_active
“Integer Utilization”:
smsp__sass_thread_inst_executed_op_integer_pred_on.avg.pct_of_peak_sustained_active

MEMORY BOUND

- A code can be memory bound when either it is limited by memory bandwidth or latency. We will lump memory latency bound codes in with the general latency case.
- For a memory bandwidth bound code, we will seek to optimize usage of the various memory subsystems, taking advantage of the memory hierarchy where possible.
    - Optimize use of global memory
    - Under data reuse scenarios, make (efficient) use of higher levels of the memory hierarchy, and optimize these usages (L2 cache, shared memory).
    - Take advantage of cache "diversification" using special GPU caches – constant cache, read-only cache, texture cache/memory, surface memory.
- For a code that is memory bandwidth bound, we can compute the actual throughput vs. peak theoretical

COMPUTE BOUND

- A code is compute bound when the performance of a particular type of compute instruction/operation is at or near the limit of the functional unit servicing that type
- Optimization strategy involves optimizing the use of that functional unit type, as well as (possibly) seeking to shift the compute load to other types
- For a code that is dominated by a particular type (e.g. single precision floating point multiply/add) we can compare the actual throughput vs. peak theoretical.

LATENCY BOUND

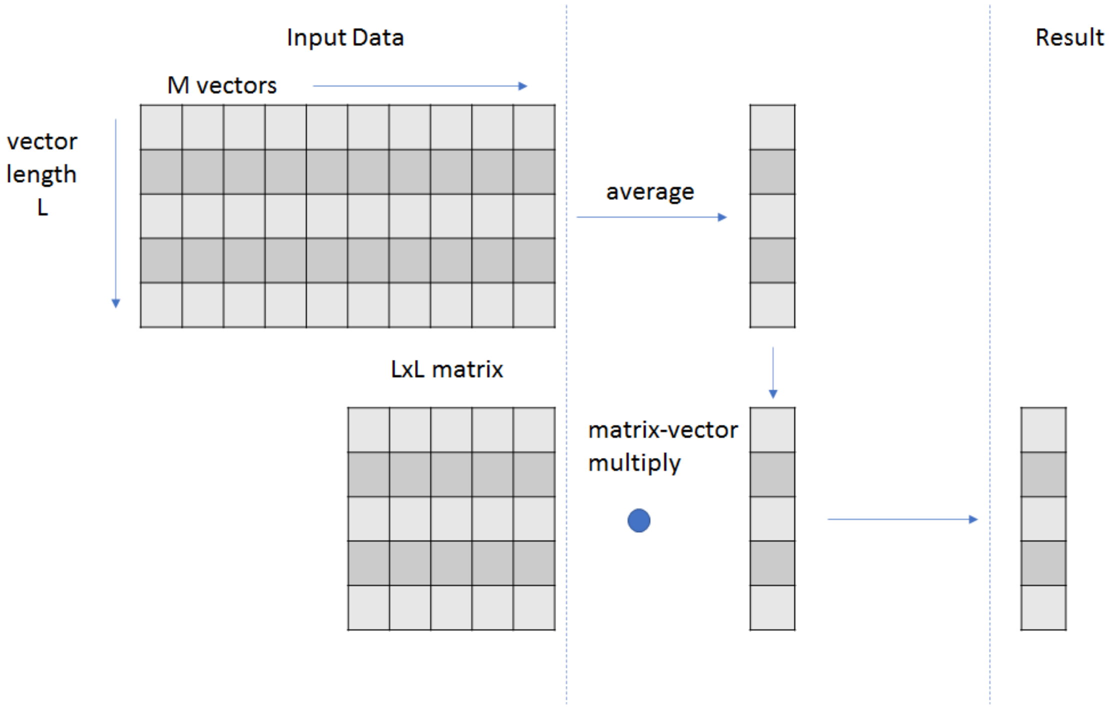
- A code is latency bound when the GPU cannot keep busy with the available/exposed parallel work.
- The general strategy for a latency bound code will be to expose more parallel work
  - Make sure that you are launching a large number of threads
  - Increase the work per thread (e.g. via a loop over input elements)
  - Use “vector load” to allow a single thread to process multiple input elements
  - Strive for maximum occupancy

WHAT IS OCCUPANCY?

- A measure of the actual thread load in an SM, vs. peak theoretical/peak achievable
- CUDA includes an occupancy calculator spreadsheet
- Higher occupancy is sometimes a path to higher performance
- Achievable occupancy is affected by limiters to occupancy
- Primary limiters:
    - Registers per thread (can be reported by the profiler, or can get at compile time)
    - Threads per threadblock
    - Shared memory usage

WALK-THRU

- What the code does:



This process is repeated N times, using N sets of input vectors, reusing the matrix, producing N result vectors.

FUTURE SESSIONS

- Cooperative Groups

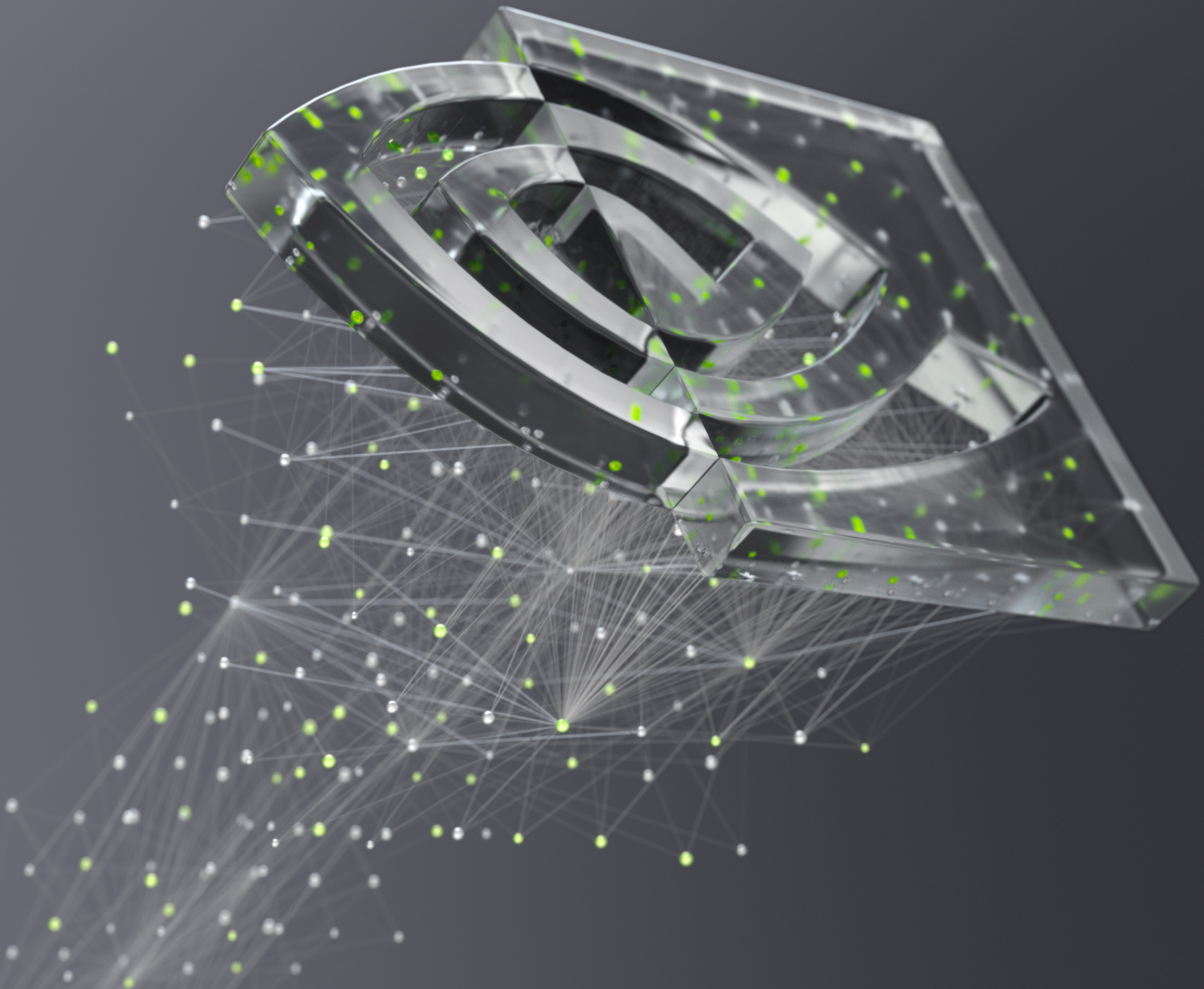
FURTHER STUDY

- Analysis Driven optimization:
  - http://on-demand.gputechconf.com/gtc/2012/presentations/S0514-GTC2012-GPU-Performance-Analysis.pdf
  - http://www.nvidia.com/content/GTC-2010/pdfs/2012_GTC2010.pdf
  - Google “gtc cuda optimization”
- New tools blogs:
  - https://developer.nvidia.com/blog/migrating-nvidia-nsight-tools-nvvp-nvprof/
  - https://developer.nvidia.com/blog/transitioning-nsight-systems-nvidia-visual-profiler-nvprof/
  - https://developer.nvidia.com/blog/using-nsight-compute-to-inspect-your-kernels/

HOMEWORK

- Log into Summit (ssh username@home.ccs.ornl.gov -> ssh summit)
- Clone GitHub repository:
  - Git clone git@github.com:olcf/cuda-training-series.git
- Follow the instructions in the readme.md file:
  - https://github.com/olcf/cuda-training-series/blob/master/exercises/hw7/readme.md

- Prerequisites: basic linux skills, e.g. ls, cd, etc., knowledge of a text editor like vi/emacs, and some knowledge of C/C++ programming



NVIDIA