Roofline Analysis on NVIDIA GPUs

Samuel Williams
Computational Research Division
Lawrence Berkeley National Lab
SWWilliams@lbl.gov

Material/slides provided by Max Katz, Charlene Yang, Jonathan Madsen, Tan Nguyen, Nan Ding, and Khaled Ibrahim
Reminder: Roofline is made of two components

- **Machine Model**
  - Lines defined by peak GB/s and GF/s *(Benchmarking)*
  - Unique to each architecture
  - Common to all apps on that architecture
Reminder: Roofline is made of two components

- **Machine Model**
  - Lines defined by peak GB/s and GF/s *(Benchmarking)*
  - Unique to each architecture
  - Common to all apps on that architecture

- **Application Characteristics**
  - Dots defined by application GFLOPs, GBs, and run time *(Application Instrumentation)*
  - Unique to each application
  - Unique to each architecture
Two Approaches:

**Original Approach**
- Empirical Roofline Toolkit (ERT)
  - GFLOP/s, GB/s, etc…

**Fully Integrated Approach**
- Nsight Compute
  - Existing Analytical Capabilities + Roofline Modeling, Profiling, and Visualization

**Profiling**
- Nsight Compute
  - Kernel metrics:
    - GFLOPs, GBs, and seconds

**Visualization**
- Python Scripts
  - Manipulate metrics and plot
Empirical Roofline Toolkit (ERT)
Machine Characterization

- “Theoretical Performance” numbers can be highly optimistic…
  - Pin BW vs. sustained bandwidth
  - TurboMode / Underclock for AVX
  - compiler failings on high-AI loops.

- LBL developed the Empirical Roofline Toolkit (ERT)…
  - Characterize CPU/GPU systems
  - Peak Flop rates
  - Bandwidths for each level of memory
  - MPI+OpenMP/CUDA == multiple GPUs

- Provides a sanity check on programmers, compilers, vendors

https://bitbucket.org/berkeleylab/cs-roofline-toolkit
https://github.com/cyanguwa/nersc-roofline
https://crd.lbl.gov/departments/computer-science/PAR/research/roofline
ERT Configuration Files

**Kernel.c**
- actual compute
- customizable

**Driver.c**
- setup
- call kernels
- loop over parameters

**config script**
- set up ranges of parameters

**job script**
- submit the job and run it

---

Empirical Roofline Toolkit (ERT).
https://bitbucket.org/berkeleylab/cs-roofline-toolkit/
ERT on NVIDIA GPUs

- Last level of memory is ‘DRAM’ (ERT calls HBM DRAM on V100)
- Enumerates all detected caches as L1, L2, etc…
- Uncacheable memory/WT caches are not detected (ERT misses L1 on V100 and calls the L2 the L1)
- Empirical ceilings are 8-28% lower than the theoretical numbers.
Profiling with NVProf
Nsight Compute
Application Characterization with Nsight (1)

Usage:
ncu -k [regexp] --metrics [metrics] --csv ./application

Kernel Run time:
- Time per invocation of a kernel:
  \[\text{sm\_cycles\_elapsed.\text{avg} / sm\_cycles\_elapsed.\text{avg}.\text{per\_second}}\]
#FLOPs:

- For {Double, Single, Half} precision, sum the following metrics:
  
  \[
  \text{sm\_sass\_thread\_inst\_executed\_op\_}\{d,f,h\}\text{add\_pred\_on\_sum} + \\
  \text{sm\_sass\_thread\_inst\_executed\_op\_}\{d,f,h\}\text{mul\_pred\_on\_sum} + \\
  2\times\text{sm\_sass\_thread\_inst\_executed\_op\_}\{d,f,h\}\text{fma\_pred\_on\_sum}
  \]

- To calculate FLOPs from Tensor Cores (Volta):
  
  \[512\times\text{sm\_inst\_executed\_pipe\_tensor\_sum}\]

- (far more accurate than NVProf’s discretized tensor utilization)
#Bytes

- Measure bytes for each level of the memory hierarchy
- Scale transactions where necessary

<table>
<thead>
<tr>
<th>Level</th>
<th>Metrics</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1 Cache</td>
<td>l1tex__t_bytes.sum</td>
</tr>
<tr>
<td>Shared Memory (included in L1)</td>
<td>(l1tex__data_pipe_lsu_wavefronts_mem_shared_op_ld.sum + l1tex__data_pipe_lsu_wavefronts_mem_shared_op_st.sum)*32</td>
</tr>
<tr>
<td>Atomics (included in L1)</td>
<td>(l1tex__t_set_accesses_pipe_lsu_mem_global_op_atom.sum + l1tex__t_set_accesses_pipe_lsu_mem_global_op_red.sum)*32</td>
</tr>
<tr>
<td>L2 Cache</td>
<td>lts__t_bytes.sum</td>
</tr>
<tr>
<td>Device Memory</td>
<td>dram__bytes.sum</td>
</tr>
<tr>
<td>System Memory (PCIe)</td>
<td>(lts__t_sectors_aperture_sysmem_op_read.sum + lts__t_sectors_aperture_sysmem_op_write.sum)*32</td>
</tr>
</tbody>
</table>
Visualization
You must combine ERT and Nsight data

- ERT provides compute (horizontal lines) and bandwidth (diagonal lines) ceilings
- NVProf data must be manipulated

\[
\text{AI (x coordinate)} = \frac{\text{NVprof GFLOPs}}{\text{NVprof GBytes}}
\]

\[
\text{GFLOP/s (y coordinate)} = \frac{\text{NVprof GFLOPs}}{\text{NVprof seconds}}
\]

- Plot Using Python script. e.g.

  https://gitlab.com/NERSC/roofline-on-nvidia-gpus/-/tree/master/custom-scripts
You must combine ERT and Nsight data

% cat data.txt

# all data is space delimited
memroofs 14336.0 2996.8 828.758
mem_roof_names 'L1' 'L2' 'HBM'
comproofs 7068.86 3535.79
comp_roof_names 'FMA' 'No-FMA'

# omit the following if only plotting roofs
# AI: arithmetic intensity; GFLOPs: performance
AI 0.87 2.25 2.58
GFLOPs 2085.756683
labels 'Kernel'

% plot_roofline.py data.txt
Nsight Compute (Roofline integration)
Nsight has integrated Roofline Analysis

- Nsight’s view of V100’s memory architecture
  - Green boxes are logical regions
  - Blue boxes are physical levels

- Roofline is calculated based on the data movement between physical levels

- Roofline in GUI:
  
  ncu -k [kernel] --metrics [metrics]/application
Nsight has integrated Roofline Analysis

- Automatically plots Roofline (DRAM shown below)
- Allows you to compare multiple versions of a kernel on the same Roofline (tracks progress towards optimality)
Complements Existing SOL and PTX analysis

- speed-of-light analysis comparisons with baseline
- Source/PTX/SASS analysis and correlation
Scripting interface for Custom Rooflines

- Nsight includes a scripting interface where users can define their own custom Rooflines (e.g. hierarchical Roofline)
  
https://docs.nvidia.com/nsight-compute/CustomizationGuide/index.html#sections

- NERSC/NVIDIA have provided example scripts:
  
https://gitlab.com/NERSC/roofline-on-nvidia-gpus

  e.g.

  ncu -f -o myprofile --section-folder ..../ncu-section-files
  --sectionSpeedOfLight_HierarchicalDoubleRooflineChart ./application
Which approach should I use?

**Original Approach**

**Empirical Roofline Toolkit (ERT)**
- GFLOP/s, GB/s, etc...

**Python Scripts**
- Manipulate metrics and plot

**Fully Integrated Approach**

**Nsight Compute**

**Existing Analytical Capabilities**
- Roofline Modeling, Profiling, and Visualization

**Nsight Compute**

**Kernel metrics:**
- GFLOPs, GBs, and seconds

**Best starting point for developers**

*Benchmarking, cross-architecture, custom plots, etc...*
Questions?
NVProf
(pre-Volta, soon to be deprecated)
Application Characterization with NVProf (1)

Run time:
- Time per invocation of a kernel
  \[\text{nvprof} --\text{print-gpu-trace} \ ./\text{application}\]
- Average time over multiple invocations
  \[\text{nvprof} --\text{print-gpu-summary} \ ./\text{application}\]

#FLOPs:
- CUDA Core: Predication aware and complex-operation aware (such as divides)
  \[\text{nvprof} --\text{kernels} \ [\text{kernel\_name}] --\text{metrics} \ [\text{flop\_count\_xx}] ./\text{application}\]
  e.g. \(\text{flop\_count\_\{dp/dp\_add/dp\_mul/dp\_fma, sp*, hp\}}\)
- Tensor Cores:
  \[--\text{metrics} \ \text{tensor\_precision\_fu\_utilization}\]
  Note: integer in the range of 0-10, 0=0, 10=125TFLOP/s; multiply by run time -> #FLOPs
#Bytes

- Measure bytes for each level of the memory hierarchy
- Bytes = (read transactions + write transactions) * transaction size
- Preface with your favor launcher (srun, mpirun, jsrun, etc…)

```
nvprof --kernels [kernel_name] --metrics [metric_name] ./application
```

<table>
<thead>
<tr>
<th>Level</th>
<th>Metrics</th>
<th>Transaction Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>First Level Cache</td>
<td>gld_transactions, gst_transactions, local_load_transactions, local_store_transactions, atomic_transactions</td>
<td>32B</td>
</tr>
<tr>
<td>Shared Memory</td>
<td>shared_load_transactions, shared_store_transactions</td>
<td>128B</td>
</tr>
<tr>
<td>Second Level Cache</td>
<td>l2_read_transactions, l2_write_transactions</td>
<td>32B</td>
</tr>
<tr>
<td>Device Memory</td>
<td>dram_read_transactions, dram_write_transactions</td>
<td>32B</td>
</tr>
<tr>
<td>System Memory</td>
<td>system_read_transactions, system_write_transactions</td>
<td>32B</td>
</tr>
</tbody>
</table>
You can specify specific context(1), stream(7), and invocation(1) ...

--kernels “1:7:smooth_kernel:1”

Nominally just get an output table

Alternately, you can output to a csv...

--csv -o nvprof.out