The Goal

• Hierarchical Roofline
  – L1, L2, device memory, system memory (NVLINK or PCIe), peer-to-peer GPU, NIC, ...

• Data precisions
  – double, single, half, quarter, ...

• Instruction types
  – floating-point, integer, ...
  – FMA, mul/add, ...

• Execution units
  – CUDA core/Tensor core
Methodology

1. **Roofline ceilings**
   - theoretical values for peak bandwidth/computational throughput,
   - or, empirical values from Empirical Roofline Toolkit (ERT)
     [https://bitbucket.org/berkeleylab/cs-roofline-toolkit/](https://bitbucket.org/berkeleylab/cs-roofline-toolkit/)

2. **Application data**
   - measure three quantities: time, FLOPs, data movement (Bytes)
   - calculate AI and throughput:
     \[
     \text{Arithmetic Intensity} = \frac{\text{FLOPs}}{\text{data movement (x: FLOPs/Byte)}}
     \]
     \[
     \text{Performance} = \frac{\text{FLOPs}}{\text{time (y: GFLOP/s)}}
     \]

3. **Roofline chart**
   - Nsight Compute
   - or, customized scripts: [https://gitlab.com/NERSC/roofline-on-nvidia-gpus/](https://gitlab.com/NERSC/roofline-on-nvidia-gpus/)
1. Empirical Roofline Toolkit (ERT)

- Sweeps through a range of configurations
- Produces a more realistic set of peaks in real environment

(left) working set sizes
(top right) bandwidths, (top bottom) FLOP/s
2. Application Data (manual)

For tighter integration with your workflow, you can collect relevant metrics and plot Roofline charts manually:

```bash
srun -n1 nv-nsight-cu-cli -k <kernels> --metrics <metrics> -o output ./app
```

### Nsight Compute metrics and arithmetics

<table>
<thead>
<tr>
<th></th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>( \frac{\text{sm_cycles_elapsed.\ avg}}{\text{sm_cycles_elapsed.\ avg.\ per_second}} )</td>
</tr>
<tr>
<td>FLOPs</td>
<td>CUDA core [x: d for DP, f for SP, and h for HP]</td>
</tr>
<tr>
<td></td>
<td>( \text{sm_sass_thread_inst_executed_op_xadd_pred_on.sum} + )</td>
</tr>
<tr>
<td></td>
<td>( \text{sm_sass_thread_inst_executed_op_xfma_pred_on.sum} * 2 + )</td>
</tr>
<tr>
<td></td>
<td>( \text{sm_sass_thread_inst_executed_op_xmul_pred_on.sum} )</td>
</tr>
<tr>
<td></td>
<td>Tensor core</td>
</tr>
<tr>
<td></td>
<td>( \text{sm_inst_executed_pipe_tensor.sum} * 512 )</td>
</tr>
<tr>
<td>Bytes</td>
<td>device memory:</td>
</tr>
<tr>
<td></td>
<td>( \text{dram_bytes.sum} )</td>
</tr>
<tr>
<td></td>
<td>L2: ( \text{lts_t_bytes.sum} )</td>
</tr>
<tr>
<td></td>
<td>L1: ( \text{lltex_t_bytes.sum} )</td>
</tr>
</tbody>
</table>

[https://gitlab.com/NERSC/roofline-on-nvidia-gpus/](https://gitlab.com/NERSC/roofline-on-nvidia-gpus/)
OLD: nvprof-based

Runtime:
Time per invocation of a kernel
```
nvprof --print-gpu-trace ./application
```
Average time over multiple invocations
```
nvprof --print-gpu-summary ./application
```

FLOPs:
CUDA Core: Predication aware and complex-operation aware (such as divides)
```
nvprof --kernels 'kernel_name' --metrics 'flop_count_xx' ./application e.g.
flop_count_{dp/dp_add/dp_mul/dp_fma, sp*, hp*}
```
Tensor Core: (more details later)
```
--metrics tensor_precision_fu_utilization
```
0-10 integer range, 0-0, 10-125 TFLOP/s; multiply by run time -> FLOPs

Bytes: on different cache levels
```
Bytes = (read transactions + write transactions) x transaction size
```
```
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><code>gld_transactions, gst_transactions, atomic_transactions, local_load_transactions, local_store_transactions, shared_load_transactions, shared_store_transactions</code></td>
<td>32B</td>
</tr>
<tr>
<td>Second Level Cache</td>
<td><code>l2_readTransactions, l2_write_transactions</code></td>
<td>32B</td>
</tr>
<tr>
<td>Device Memory</td>
<td><code>dram_read_transactions, dram_write_transactions</code></td>
<td>32B</td>
</tr>
<tr>
<td>System Memory</td>
<td><code>system_read_transactions, system_write_transactions</code></td>
<td>32B</td>
</tr>
</tbody>
</table>
## OLD: NCU-based (CUDA 10)

### Quantities
- **Runtime**
  - `smsp__cycles_elapsed.sum`,
  - `smsp__cycles_elapsed.sum.per_second`,
  - `smsp__pipe_tensor_op_hmma_cycles_active.sum`,
  - `smsp__pipe_tensor_op_hmma_cycles_active.sum.per_second`  

### Arithmetics
- `sum/sum.per_second`

### FLOPs
- `smsp__sass_thread_inst_executed_op_dadd_pred_on.sum`,
- `smsp__sass_thread_inst_executed_op_dmul_pred_on.sum`,
- `smsp__sass_thread_inst_executed_op_dfma_pred_on.sum`,
- `smsp__sass_thread_inst_executed_op_fadd_pred_on.sum`,
- `smsp__sass_thread_inst_executed_op_fmul_pred_on.sum`,
- `smsp__sass_thread_inst_executed_op_ffma_pred_on.sum`,
- `smsp__sass_thread_inst_executed_op_hadd_pred_on.sum`,
- `smsp__sass_thread_inst_executed_op_hmul_pred_on.sum`,
- `smsp__sass_thread_inst_executed_op_hfma_pred_on.sum`

### Metrics
- `#d: double precision`
- `#f: single precision`
- `#h: half precision`
- `fma x 2 + mul + add`

### Similar to before, Bytes for different cache levels:

<table>
<thead>
<tr>
<th>Level</th>
<th>Metrics</th>
<th>Transaction Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>First Level Cache*</td>
<td><code>l1tex_t_sectors_pipe_lsu_mem_global_op_ld.sum</code>, <code>l1tex_t_sectors_pipe_lsu_mem_global_op_st.sum</code>, <code>l1tex_t_set_accesses_pipe_lsu_mem_global_op_red.sum</code>, <code>l1tex_t_sectors_pipe_lsu_mem_local_op_ld.sum</code>, <code>l1tex_t_sectors_pipe_lsu_mem_local_op_st.sum</code>, <code>smsp_inst_executed_op_shared_ld.sum</code>, <code>smsp_inst_executed_op_shared_st.sum</code></td>
<td>32B</td>
</tr>
<tr>
<td>Second Level Cache</td>
<td><code>lts_t_sectors_op_read.sum</code>, <code>lts_t_sectors_op_write.sum</code></td>
<td>32B</td>
</tr>
<tr>
<td>Device Memory</td>
<td><code>dram_sectors_write.sum</code>, <code>dram_sectors_read.sum</code></td>
<td>32B</td>
</tr>
<tr>
<td>System Memory</td>
<td><code>lts_t_sectors_aperture_sysmem_op_read.sum</code>, <code>lts_t_sectors_aperture_sysmem_op_write.sum</code></td>
<td>32B</td>
</tr>
</tbody>
</table>

### Runtime and FLOPs:
```
nv-nsight-cu-cli -k `kernel_name` --metrics `metrics_name` ./application
```
Otherwise, Nsight Compute provides a set of section files for automatic Roofline data collection

- metrics used are a bit different than in the previous table, but they will produce the same results
- SpeedOfLight_Hierarchical{Double,Single,Half,Tensor}RooflineChart.section

```cpp
AchievedWork {
    ValueCyclesPerSecondExpression {
        ValuePerCycleMetrics {
            Name: "smsp__sass_thread_inst_executed_op_dadd_pred_on.sum.per_cycle_elapsed"
        }
        ValuePerCycleMetrics {
            Name: "smsp__sass_thread_inst_executed_op_dmul_pred_on.sum.per_cycle_elapsed"
        }
        ValuePerCycleMetrics {
            Name: "smsp__sass_thread_inst_executed_op_dfma_pred_on.sum.per_cycle_elapsed"
        }
        ValuePerCycleMetrics {
            Name: "smsp__sass_thread_inst_executed_op_dfma_pred_on.sum.per_cycle_elapsed"
        }
        CyclesPerSecondMetric {
            Name: "smsp__cycles_elapsed.avg.per_second"
        }
    }
}
```
3. Roofline Charts

- Automatic if you are using the Roofline feature in Nsight Compute

- For more customized plots, please try scripts here
  - https://gitlab.com/NERSC/roofline-on-nvidia-gpus/
  - TWEAK them to fit your needs!
Example 1. GEMM

- A Tensor Core example using WMMA and cuBLAS
  - `convertFp32ToFp16(__half*, float*, int)`
  - `generate_seed_pseudo(unsigned long long, unsigned long long, unsigned long long, curandOrdering, curandStateXORWOW*, unsigned int*)`
  - `void gen_sequenced<curandStateXORWOW, float, int, &(float curand_uniform_noargs<curandStateXORWOW>(curandStateXORWOW*, int)), rng_config<curandStateXORWOW, (curandOrdering)101> >(curandStateXORWOW*, float*, unsigned long, unsigned long, int)`
  - `volta_s884gemm_fp16_256x128_ldg8_nn`
  - `wmma_example(__half*, __half*, float*, int, int, int, float, float)`

https://github.com/NVIDIA-developer-blog/code-samples/tree/master/posts/tensor-cores
Example 1. GEMM

- A Tensor Core example using WMMA and cuBLAS
  - convertFp32ToFp16(__half*, float*, int)
  - generate_seed_pseudo(unsigned long long, unsigned long long, unsigned long long, curandOrdering, curandStateXORWOW*, unsigned int*)
  - void gen_sequenced<curandStateXORWOW, float, int, &float curand_uniform_noargs<curandStateXORWOW>(curandStateXORWOW*, int)), rng_config<curandStateXORWOW, (curandOrdering)101> >(curandStateXORWOW*, float*, unsigned long, unsigned long, int)
  - volta_s884gemm_fp16_256x128_ldg8_nn
  - wmma_example(__half*, __half*, float*, int, int, int, float, float)

- Multiple kernels in one plot
- Shorten/reformat kernel names
- Adjust the xmin/xmax and ymin/ymax
- Prune 0-AI kernels (first 2)

https://github.com/NVIDIA-developer-blog/code-samples/tree/master/posts/tensor-cores
Example 2. PyTorch climate seg.

- Separate HBM, L2, L1 charts
- Marker size based on percentage of time, kernel count, FLOPs, etc
- AMP O0 vs O1 (CUDA core vs Tensor Core)
Example 2. PyTorch climate seg.

- Separate HBM, L2, L1 charts
- Marker size based on percentage of time, kernel count, FLOPs, etc
- AMP O0 vs O1 (CUDA core vs Tensor Core)
Example 2. PyTorch climate seg.

- Separate HBM, L2, L1 charts
- Marker size based on percentage of time, kernel count, FLOPs, etc
- AMP O0 vs O1 (CUDA core vs Tensor Core)
Example 2. PyTorch climate seg.

- Separate HBM, L2, L1 charts
- Marker size based on percentage of time, kernel count, FLOPs, etc
- AMP O0 vs O1 (CUDA core vs Tensor Core)
Example 3. GPP

Show optimization path
1. baseline collapse(3)
2. move n1_loc in and change to collapse(2)
3. vector_length 128 to 512
4. replace div by rcp
5. …
Open questions/issues

- Over-reporting when active SMs < 80?
- Integer overflow for \_red.sum metrics in Nsight Compute?
- How do we combine ceilings for mixed-precision Rooflines? overlapping pipelines?
  - current section files are based on precisions (dp, sp, hp, tensor)
- Metrics for integer Roofline, power Roofline, and instruction Roofline?
- How to treat 0-AI kernels? instruction Roofline?
- More elegant section files for Tensor Core? 512 repetitions of the same metric.
- ??
Thank You