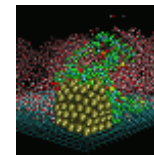
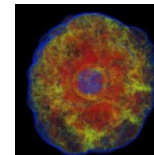
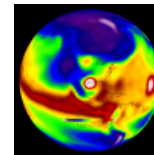
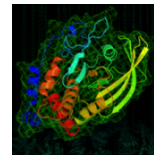
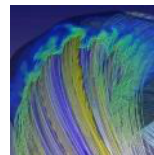
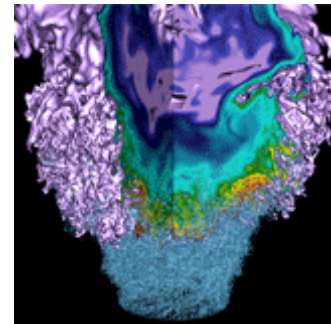


Hierarchical Roofline Analysis on GPUs



Charlene Yang
Lawrence Berkeley National Laboratory
ECP 2020, Houston

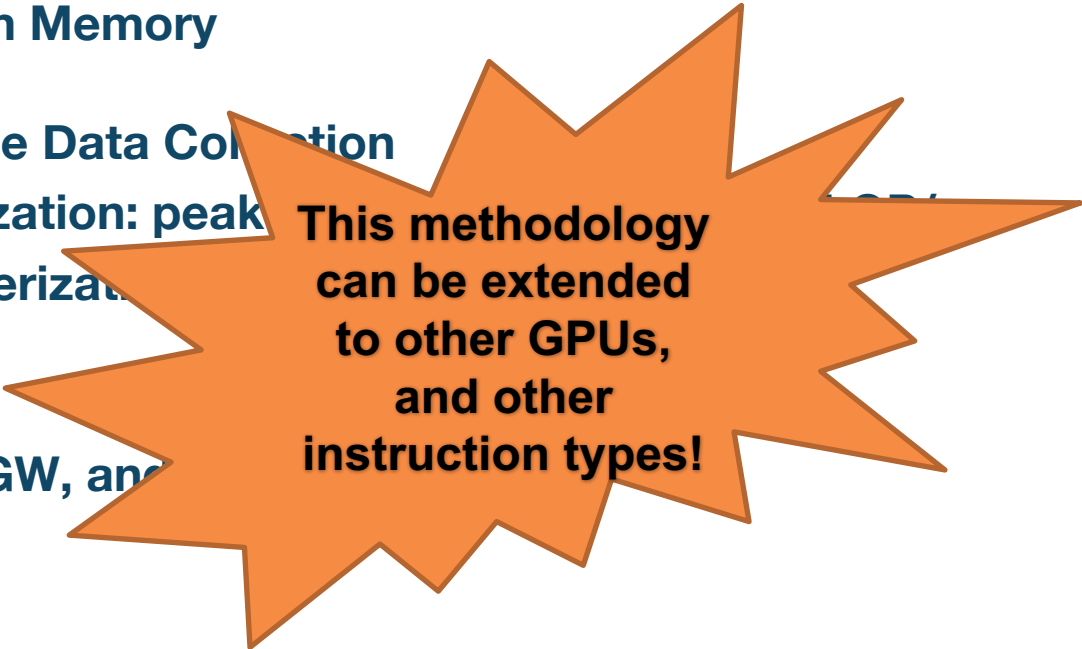
- Hierarchical Roofline on NVIDIA GPUs
 - L1, L2, HBM, System Memory

- Methodology for Roofline Data Collection

- Machine characterization: peak performance, memory bandwidth, and latency
- Application characterization: peak performance, memory bandwidth, and latency

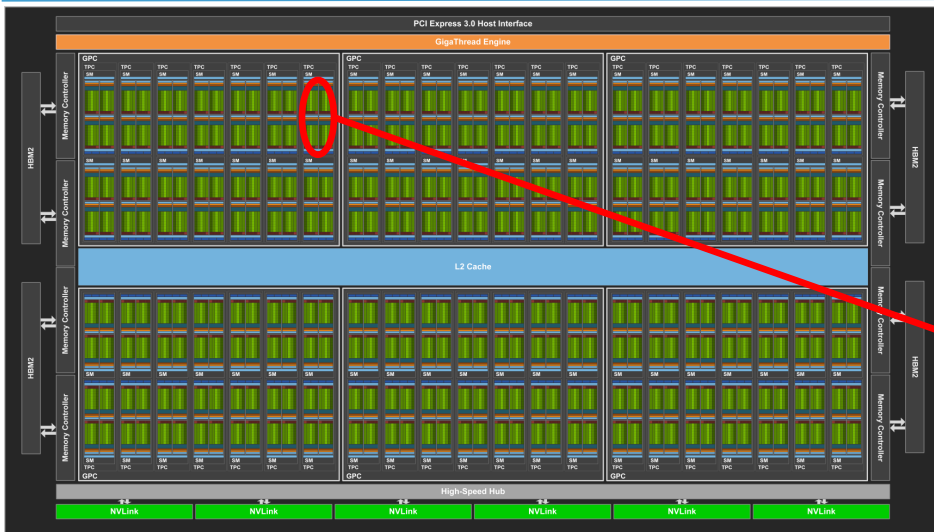
- Two Examples

- GPP from BerkeleyGW, and

An orange starburst callout with a black outline is positioned on the right side of the slide, overlapping the text of the second bullet point. It contains the following text:

**This methodology
can be extended
to other GPUs,
and other
instruction types!**

GPU Architecture: Tesla V100



80 SMs
12800 CUDA Cores **16/32GB HBM2**
640 Tensor Cores **900GB/s HBM2**

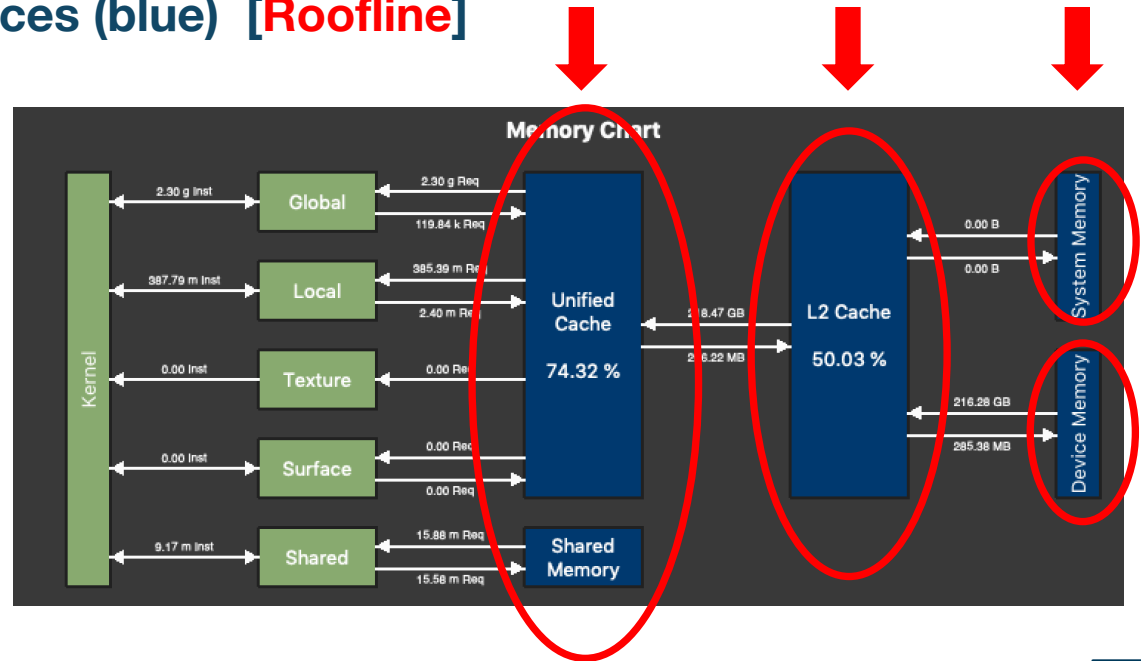
FP32 units	64	INT32 units	64
FP64 units	32	Tensor Cores	8
Registers	256KB	Unified Cache	128KB
Max Threads	2048	Thread Blocks	32



GPU Architecture: Tesla V100



- Logical memory spaces (green)
- Physical memory spaces (blue) [**Roofline**]
 - Level 1
 - Level 2
 - HBM (DRAM)
 - PCIe/NVLink

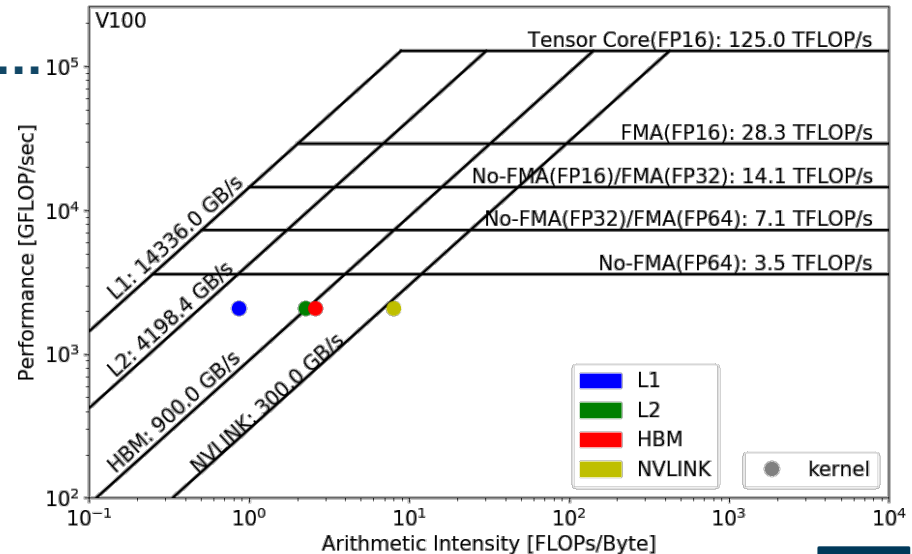


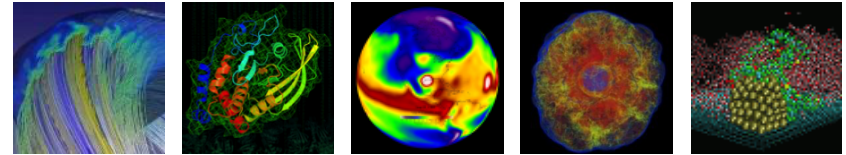
Goal: Construct Hierarchical Roofline



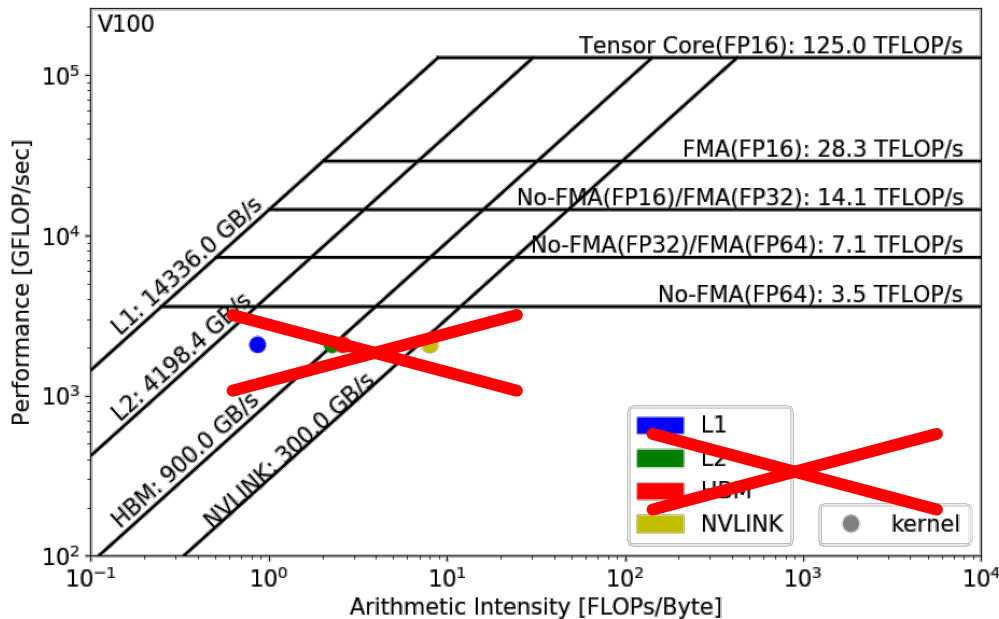
To construct a Roofline on NVIDIA GPUs

- that incorporates the full memory hierarchy
 - L1, L2, HBM, System Memory (NVLink/PCIe)
- also instruction types, data types...
 - FMA/no-FMA/IntOps/...
 - FP64, FP32, FP16, ...
 - CUDA core/Tensor core
 - ...





Methodology to Collect Roofline Data



How to get the ceilings?

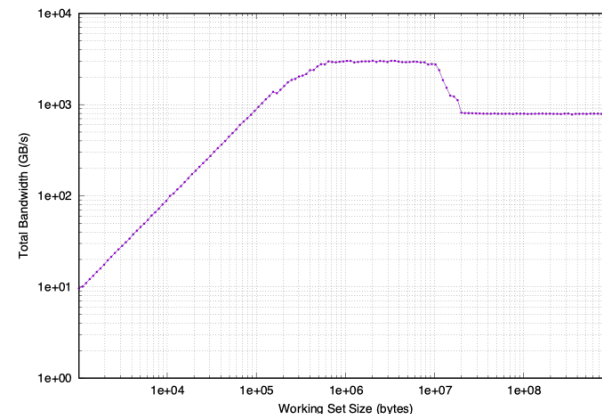
- compute and bandwidth

Theoretical vs Empirical

Empirical Roofline Toolkit (ERT)

- runs micro benchmarks
- **More Realistic**
- power constraints, *etc*

- **Empirical Roofline Toolkit (ERT)**
 - Different than the architecture specs, **MORE REALISTIC**
 - Reflects **actual** execution environment (power constraints, *etc*)
 - Sweeps through a range of configurations, and **statistically stable**
 - Data elements per thread
 - FLOPs per data element
 - Threadblocks/threads
 - Trails per dataset
 - *etc*



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

Machine Characterization



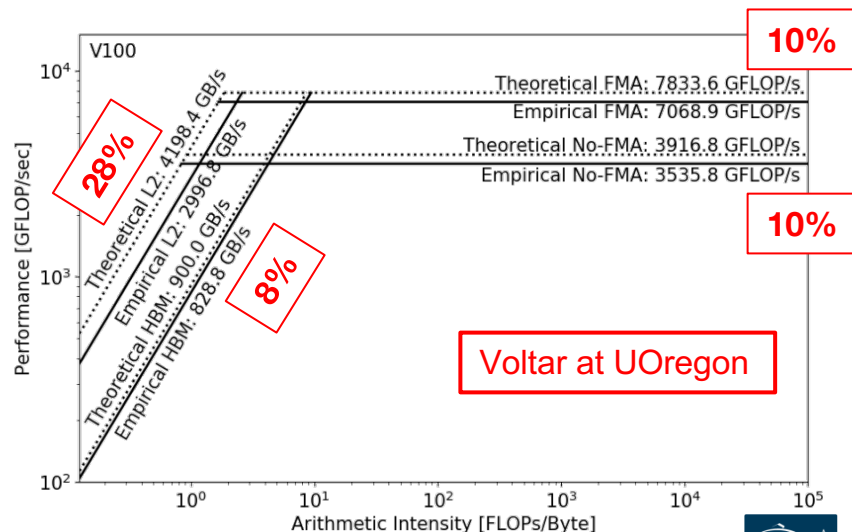
- ERT can't detect all the ceilings yet - IN DEVELOPMENT!
- Theoretical **compute** ceilings on V100:
 - FP64 FMA: 80 SMs x 32 FP64 cores x 1.53 GHz x 2 = 7.83 TFLOP/s
 - FP64 No-FMA: 80 SMs x 32 FP64 cores x 1.53 GHz = 3.92 TFLOP/s
- Theoretical **memory** bandwidths on V100:
 - HBM: 900 GB/s
 - L2: ~4.1 TB/s

Bad News:

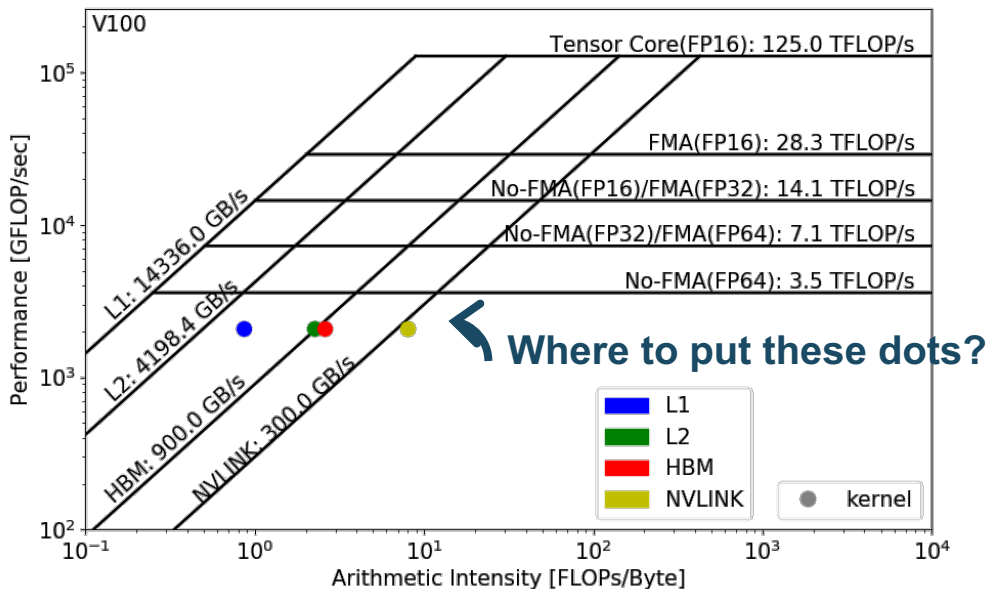
- you may never achieve 7.8 TFLOP/s

Good News:

- you may be closer to the ceiling than you think



Application Characterization



Require three raw measurements:

- Runtime
- FLOPs
- Bytes (on each cache level)

to calculate AI and GFLOP/s:

$$\text{Arithmetic Intensity} = \frac{\text{nvprof FLOPs}}{\text{nvprof Data Movement}}$$

(x: FLOPs/Byte)

$$\text{Performance} = \frac{\text{nvprof FLOPs}}{\text{Runtime}}$$

(y: GFLOP/s)

Currently the methodology is based on **nvprof**

But we are working with NVIDIA on an **Nsight**-based methodology!!

- **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, 10-125TFLOP/s; multiply by run time -> FLOPs

Application Characterization



- Bytes for different cache levels in order to construct hierarchical Roofline:
 - Bytes = (read transactions + write transactions) x transaction size
 - `nvprof --kernels 'kernel_name' --metrics 'metric_name' ./application`

Level	Metrics	Transaction Size
First Level Cache*	<code>gld_transactions, gst_transactions, atomic_transactions, local_load_transactions, local_store_transactions, shared_load_transactions, shared_store_transactions</code>	32B
Second Level Cache	<code>l2_read_transactions, l2_write_transactions</code>	32B
Device Memory	<code>dram_read_transactions, dram_write_transactions</code>	32B
System Memory	<code>system_read_transactions, system_write_transactions</code>	32B

- **Note:** surface and texture transactions are ignored here for HPC applications

Example Output



```
[cjyang@voltar source]$ nvprof --kernels "1:7:smooth_kernel:1" --metrics  
flop_count_dp --metrics gld_transactions --metrics gst_transactions --  
metrics l2_read_transactions --metrics l2_write_transactions --metrics  
dram_read_transactions --metrics dram_write_transactions --metrics  
systemem_read_bytes --metrics systemem_write_bytes ./hpgmg-fv-fp 5 8
```

context : stream : kernel : invocation

- Export to CSV: `--csv -o nvprof.out`

Invocations	Metric Name	Metric Description	Min	Max	Avg
Device "Tesla V100-PCI-E-16GB (0)"					
Kernel: void smooth_kernel<int=6, int=32, int=4, int=8>(level_type, int, int, double, double, int, double*, double*)					
1	flop_count_dp	Floating Point Operations(Double Precision)	30277632	30277632	30277632
1	gld_transactions	Global Load Transactions	4280320	4280320	4280320
1	gst_transactions	Global Store Transactions	73728	73728	73728
1	l2_read_transactions	L2 Read Transactions	890596	890596	890596
1	l2_write_transactions	L2 Write Transactions	85927	85927	85927
1	dram_read_transactions	Device Memory Read Transactions	702911	702911	702911
1	dram_write_transactions	Device Memory Write Transactions	151487	151487	151487
1	systemem_read_bytes	System Memory Read Bytes	0	0	0
1	systemem_write_bytes	System Memory Write Bytes	160	160	160

Plot Roofline with Python



- Calculate Arithmetic Intensity and GFLOP/s performance
 - x coordinate: Arithmetic Intensity
 - y coordinate: GFLOP/s performance

$$\text{Performance (GFLOP/s)} = \frac{\text{nvprof FLOPs}}{\text{Runtime}}, \quad \text{Arithmetic Intensity (FLOPs/Byte)} = \frac{\text{nvprof FLOPs}}{\text{nvprof Data Movement}}$$

- Plot Roofline with Python Matplotlib
 - Example scripts:
 - <https://gitlab.com/cyang.lbl/roofline-on-nvidia-gpus/tree/master/ExamplePlots>
 - Tweak as needed for more complex Rooflines

Plot Roofline with Python

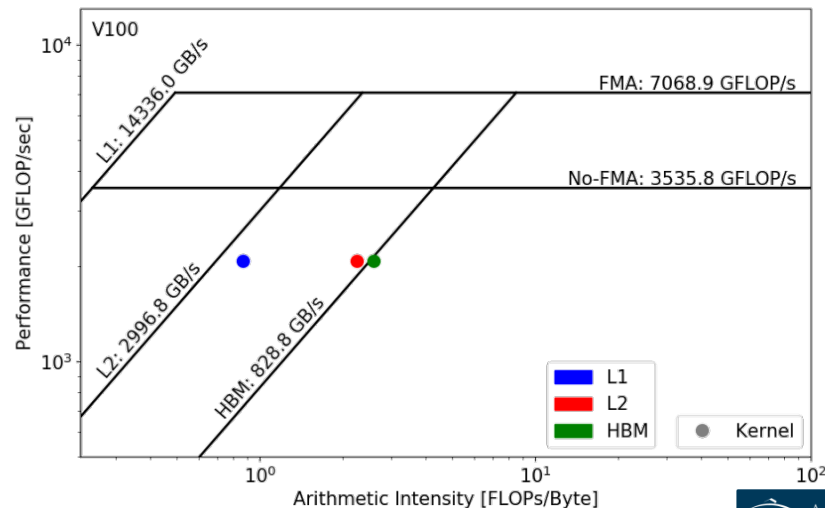


- Quick example: `plot_roofline.py data.txt`
- Accepts space-delimited list for values
- Use quotes to separate names/labels

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'
```



Recap: Methodology to Construct Roofline

1. Collect Roofline ceilings

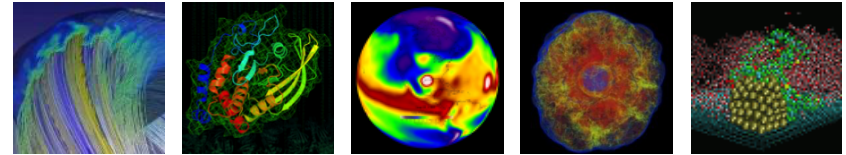
- ERT: <https://bitbucket.org/berkeleylab/cs-roofline-toolkit>
- **compute** (FMA/no FMA) and **bandwidth** (DRAM, L2, ...)

2. Collect application performance

- `nvprof: --metrics, --events, --print-gpu-trace`
- **FLOPs**, **bytes** (DRAM, L2, ...), **runtime**

3. Plot Roofline with Python Matplotlib

- **arithmetic intensity**, **GFLOP/s** performance, **ceilings**
- example scripts: <https://gitlab.com/cyang.lbl/roofline-on-nvidia-gpus/>



Roofline Analysis: Two Examples

Example 1: GPP



- GPP (General Plasmon Pole) kernel from BerkeleyGW (Material Science)
- Small problem size: 512 2 32768 20
- Tensor-contraction, abundant parallelism, large reductions
- Low FMA counts, divides, complex double data type, HBM data 1.5GB

Pseudo Code

```
do band = 1, nbands           #blockIdx.x
  do igp = 1, ngpown          #blockIdx.y
    do ig = 1, ncouls         #threadIdx.x
      do iw = 1, nw           #unrolled
        compute; reductions
```

Example 1: GPP



- Highly parameterizable
 1. Varying `nw` from 1 to 6 to increase **arithmetic intensity**
 - FLOPs increases, but data movement stays (at least for HBM)

Pseudo Code

```
do band = 1, nbands           #blockIdx.x
  do igp = 1, ngpown          #blockIdx.y
    do ig = 1, ncouls         #threadsIdx.x
      do iw = 1, nw         #unrolled
        compute; reductions
```

2. Compiling with and without FMA to study impact of **instruction mix**
 - `-fmad=true/false`

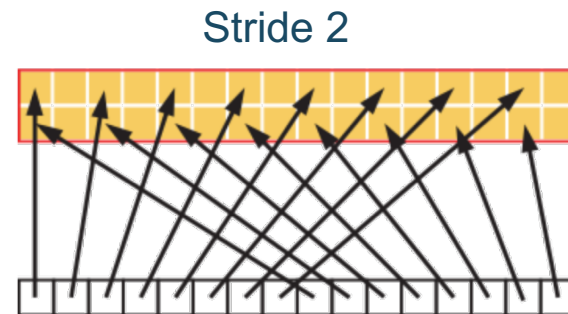
Example 1: GPP



- Highly parameterizable
- 3. Striding `ig` loop to analyze impact of **memory coalescing**
 - Split `ig` loop to two loops and place the 'blocking' loop outside

Pseudo Code

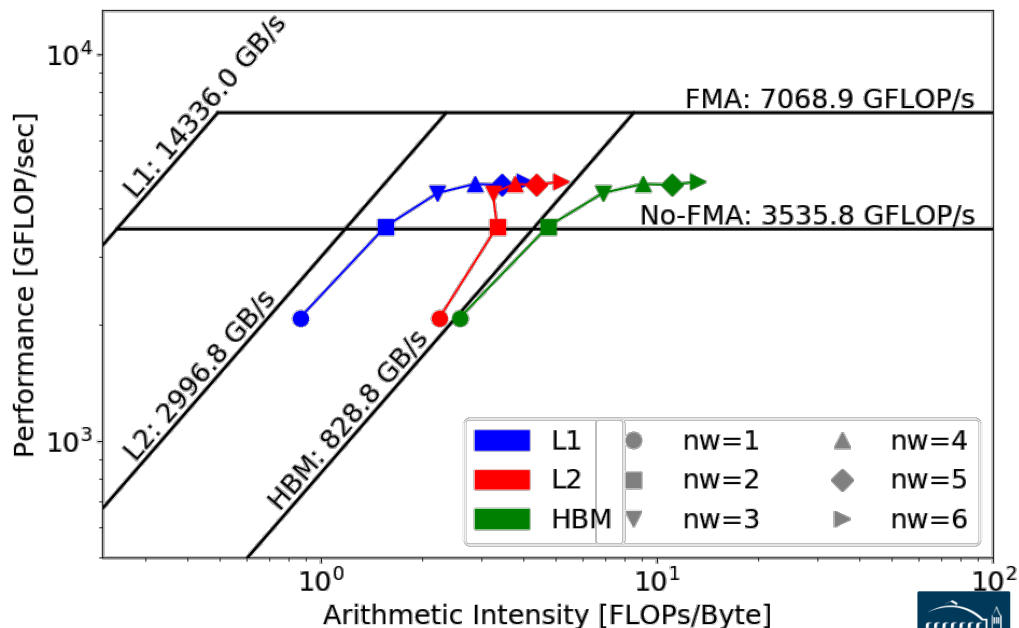
```
do band = 1, nbands           #blockIdx.x
  do igp = 1, ngpown          #blockIdx.y
    do igs = 0, stride - 1
      do ig = 1, ncouls/stride #threadIdx.x
        do iw = 1, nw         #unrolled
          compute; reductions
```



Example 1: GPP Analysis



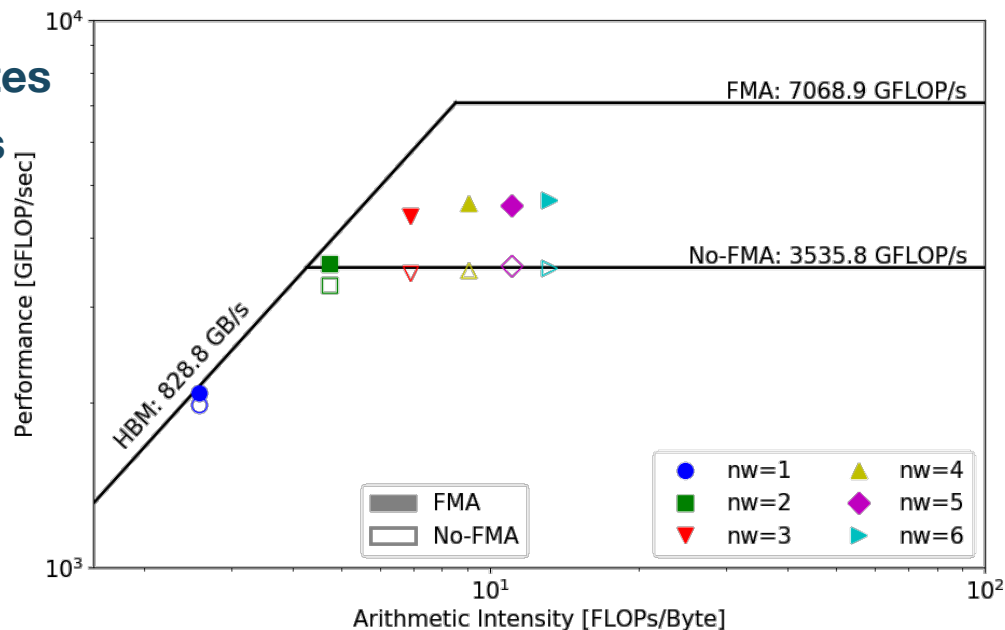
- Hierarchical Roofline, i.e. bytes are HBM, L2 and unified L1 cache bytes
 - GPP is HBM bound at low nw 's and compute bound at high nw 's
 - FLOPs $\propto nw$
 - HBM bytes: constant
 - L2 bytes: increasing at $\alpha > 1$
 - L1 bytes: constant
- Hierarchical Roofline captures more details about **cache locality**



Example 1: GPP Analysis



- **HBM Roofline, i.e. bytes are HBM bytes**
 - No-FMA performance converges to no-FMA ceiling, but FMA performance is still far from the FMA ceiling
 - Not reaching FMA ceiling due to lack of FMA instructions



Example 1: GPP Analysis



- At $n_w=6$, GPP has $\alpha = \frac{\text{FMA FP64 instr.}}{\text{FMA FP64 instr.} + \text{non-FMA FP64 instr.}} = 60\%$ of FMA instructions

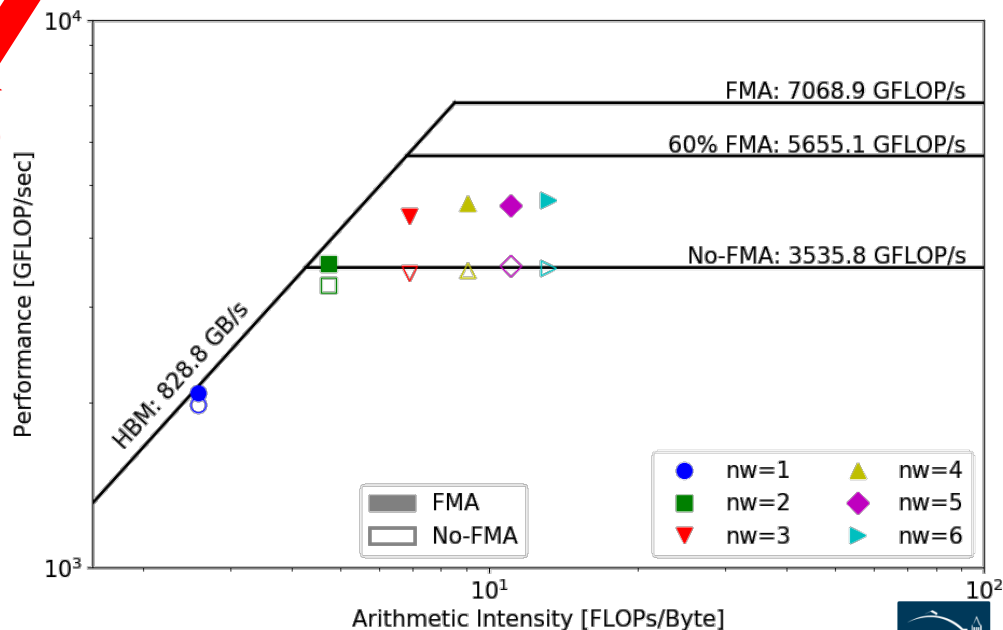
- Expected performance is

$$\beta = \frac{\alpha \times 2 + (1 - \alpha)}{2} = 80\% \text{ of peak}$$

But at $n_w=6$, GPP only achieves 66%

- Other FP/non-FP instructions may be taking up the instruction issue/execution pipeline

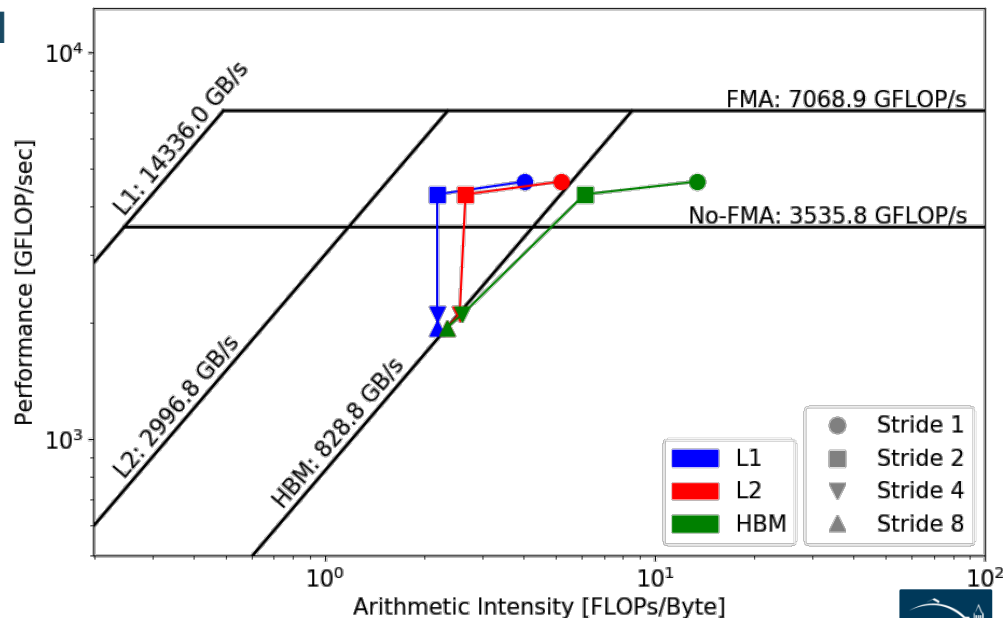
- Roofline captures effects of **instruction mix**



Example 1: GPP Analysis



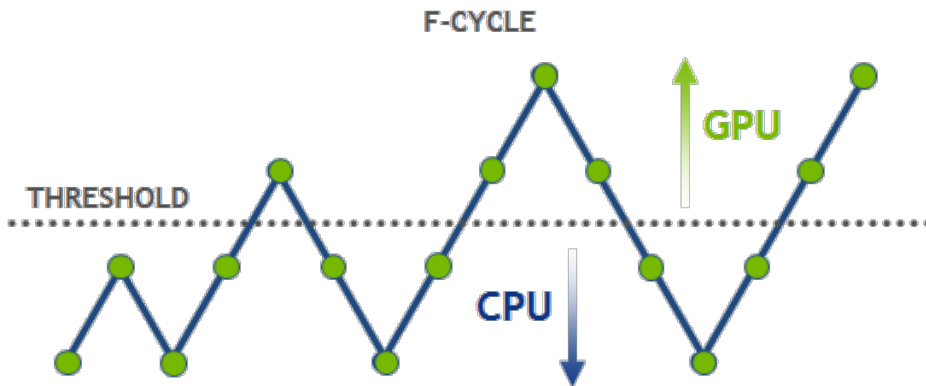
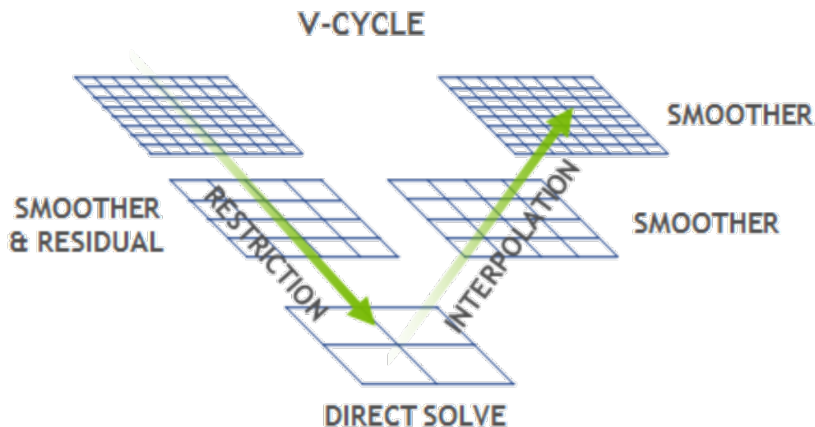
- Hierarchical Roofline, i.e. bytes are HBM, L2 and unified L1 cache bytes
 - L1/L2 bytes doubles from stride 1 to 2, but stays almost constant afterwards
 - at $n_w=6$, GPP moves from compute bound to bandwidth bound
 - Eventually all converge to HBM
- Roofline captures effects of suboptimal **memory coalescing**



Example 2: HPGMG



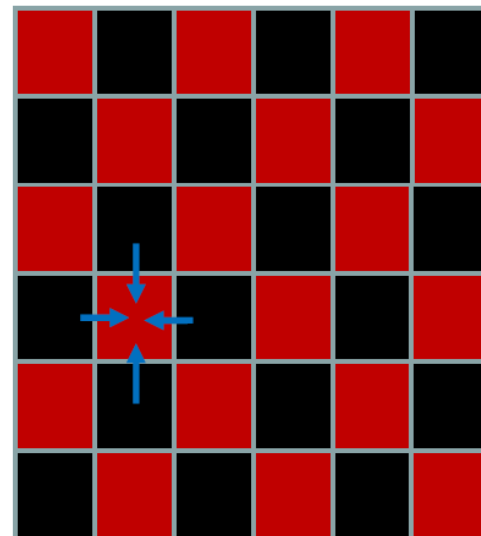
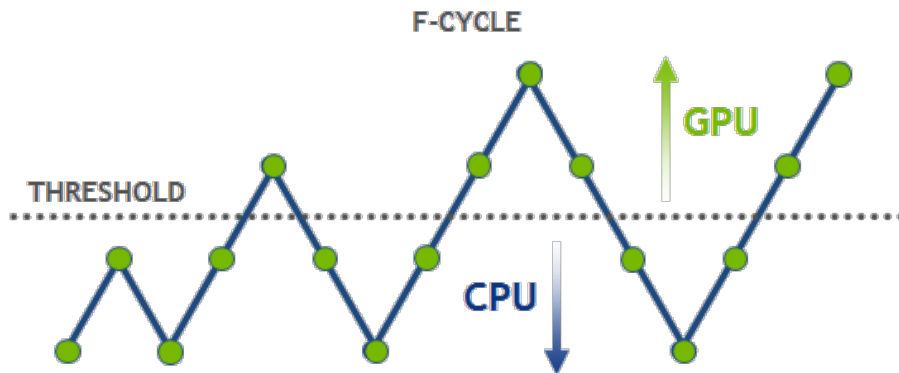
- HPGMG (High-performance Geometric Multigrid) from Adaptive Mesh Refinement code
- <https://bitbucket.org/nsakharnykh/hpgmg-cuda>
- Stencil code, F-cycles and V-cycles, GSRB smoother kernel (Gauss-Seidel Red-Black)



Example 2: HPGMG



- Hybrid GPU and CPU code
 - Example: `hpgmg-fv 7 8`
 - 128^3 box x 8, Level 5-8 run on GPU, Level 1-4 on CPU
- Three versions of GSRB kernel
 - GSRB_FP, GSRB_BRANCH, GSRB_STRIDE2



Example 2: HPGMG



GSRB_FP

```
for(int k=klo; k<(klo+kdim); k++){
  const int ijk = i + j*jStride + k*kStride;
  const double *__restrict__ RedBlack =
    level.RedBlack_FP + ghosts*(1+jStride)
    + ((k^color000)&1)*kStride;
  const double Ax = apply_op_ijk();
  const double lambda = Dinv_ijk();
  const int ij = i + j*jStride;
  xo[ijk] = X(ijk) + RedBlack[ij]*lambda*(rhs[ijk]-
Ax);
}
```

GSRB_BRANCH

```
for(int k=klo; k<(klo+kdim); k++){
  const int ijk = i + j*jStride + k*kStride;
  if(((i^j^k^color000^1)&1)){
    const double Ax = apply_op_ijk();
    const double lambda = Dinv_ijk();
    xo[ijk] = X(ijk) + lambda*(rhs[ijk]-Ax);
  }else{
    xo[ijk] = X(ijk);
  }
}
```

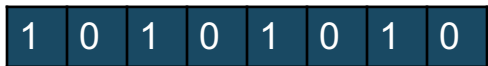


8 elements



8 elements

Sweep



8 threads



8 threads

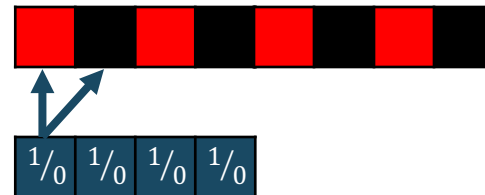
- GSRB_BRANCH has half the FLOPs as GSRB_FP but the same HBM/L1/L2 bytes

Example 2: HPGMG



GSRB_STRIDE2

```
for(int k=klo; k<klo+kdim; k++){
  i = ilo + !((ilo^j^k^color000)&1) + threadIdx.x*2;
  if(i < ilo+idim){
    const int ijk = i + jStride + k*kStride;
    xo[ijk] = X(ijk);
  }
  i = ilo + ((ilo^j^k^color000)&1) + threadIdx.x*2;
  if(i < ilo+idim){
    const int ijk = i + j*jStride + k*kStride;
    const double Ax = apply_op_ijk();
    const double lambda = Dinv_ijk();
    xo[ijk] = X(ijk) + lambda*(rhs[ijk]-Ax);
  }
}
```



8 elements

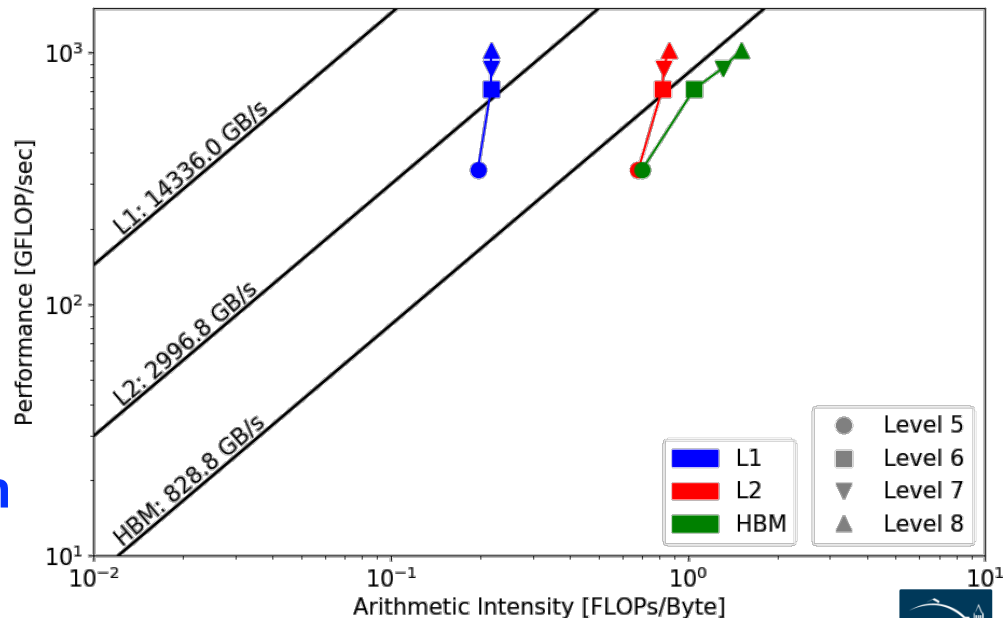
4 threads

- GSRB_STRIDE2 should have the same FLOPs as GSRB_BRANCH, but more bytes? More writes than GSRB_BRANCH.

Example 2: HPGMG Analysis



- **GSRB_FP**, Hierarchical Roofline, i.e. bytes are HBM, L2 and unified L1 cache bytes
- Highly bandwidth bound, inherent to stencil codes
- From Level 5 to Level 8:
 - HBM AI increases due to better Surface: Volume ratio
 - Roughly constant L1/L2 AI due to stencils being ‘tiled’
- Roofline captures **computational characteristics** of the **algorithm**

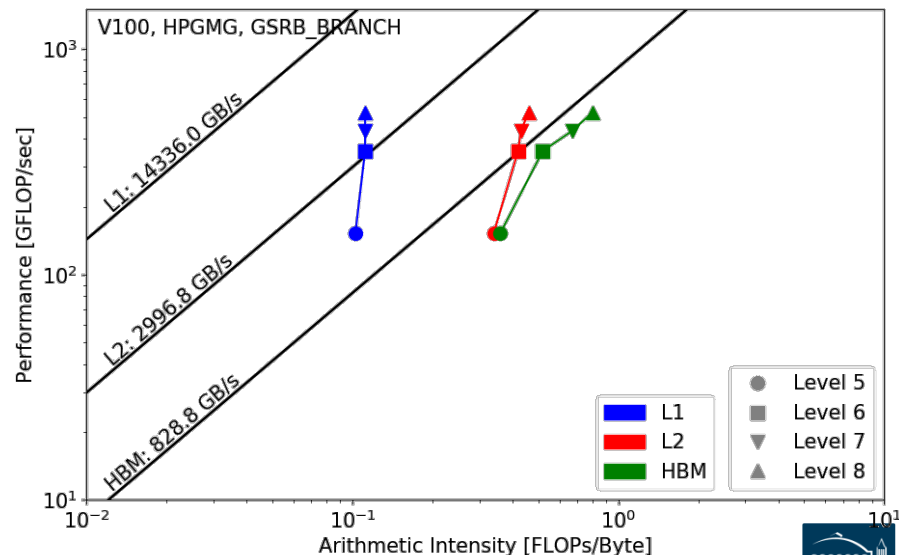
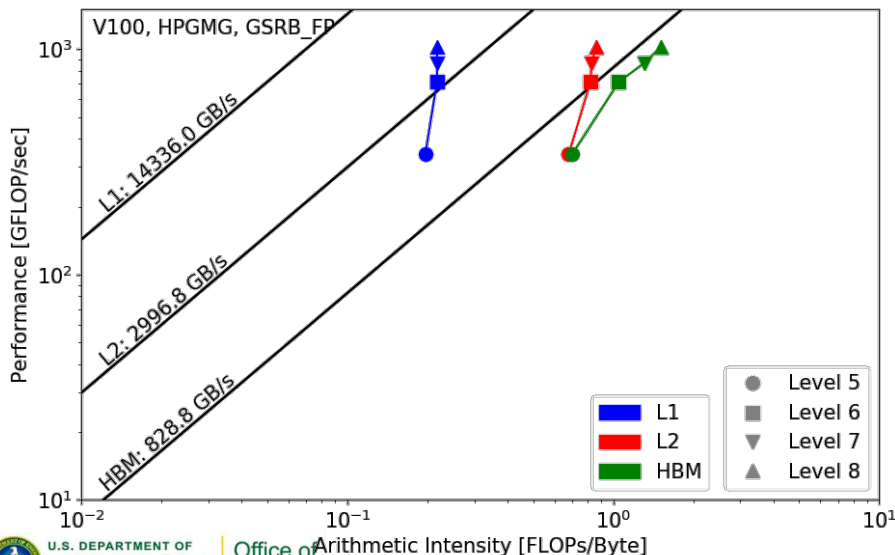


Example 2: HPGMG Analysis



GSRB_FP vs. GSRB_BRANCH

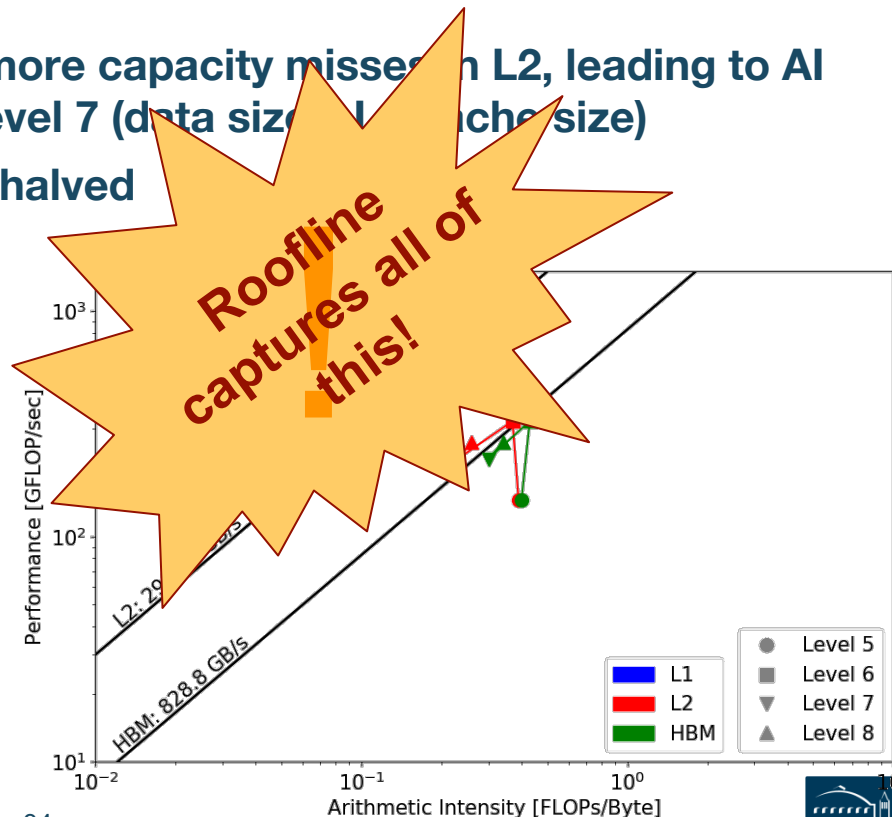
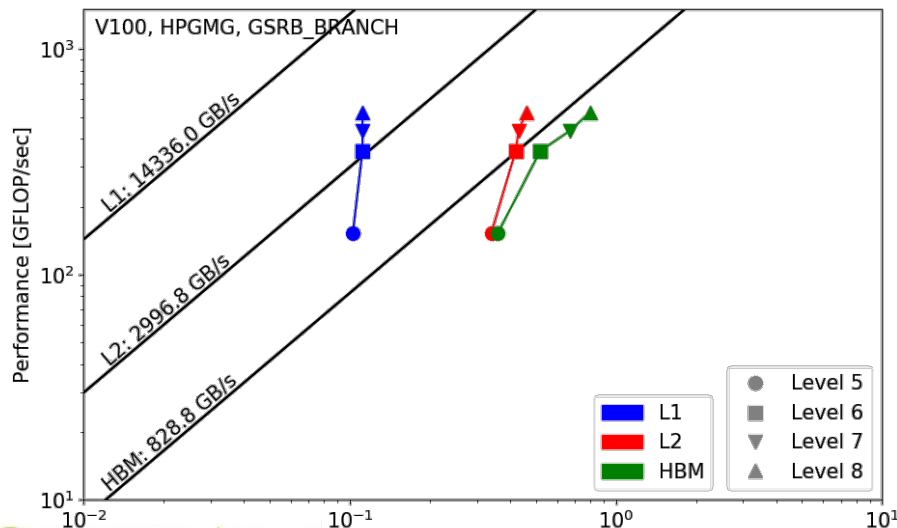
- FLOPs halves, bytes doesn't change, thus AI halves and GFLOP/s halves
- Runtime is comparable even though GFLOP/s has halved
- Same number of threads occupied, only with half predicated in GSRB_BRANCH



Example 2: HPGMG Analysis

GSRB_BRANCH vs. GSRB_STRIDE2

- Extra writes in GSRB_STRIDE2 cause more capacity misses on L2, leading to AI drop on L2 and DRAM, starting from Level 7 (data size \approx cache size)
- Runtime almost doubled and GFLOP/s halved



- **An effective methodology to construct hierarchical Roofline on NVIDIA GPUs**
 - **ERT for machine characterization**
 - **nvprof for application characterization**

- **Two examples demonstrated the value of this methodology and its ability to understand various aspects of performance on NVIDIA GPUs**
 - **cache locality, instruction mix, memory coalescing, reduced precision and Tensor Cores**
 - **GPP from BerkeleyGW, and HPGMG kernel**

Acknowledgement



- This material is based upon work supported by the Advanced Scientific Computing Research Program in the U.S. Department of Energy, Office of Science, under Award Number DE-AC02-05CH11231.
- This material is based upon work supported by the DOE RAPIDS SciDAC Institute.
- This research used resources of the National Energy Research Scientific Computing Center (NERSC), which is supported by the Office of Science of the U.S. Department of Energy under contract DE-AC02-05CH11231.



Thank You