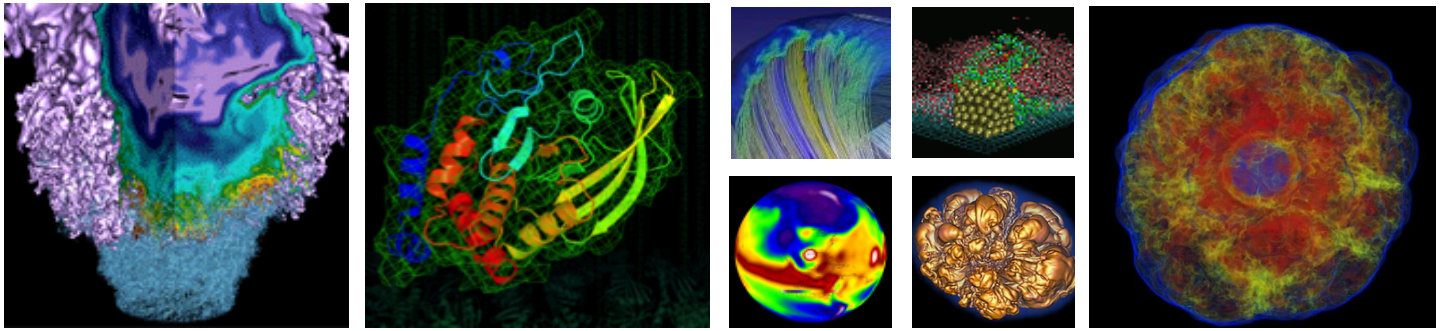


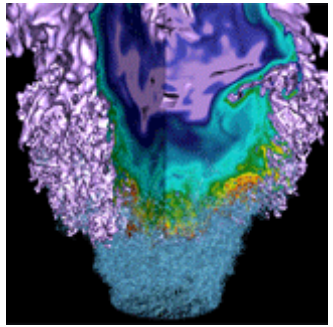
Use Cases of Roofline Analysis

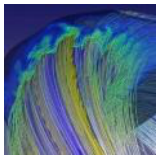
Charlene Yang @NERSC

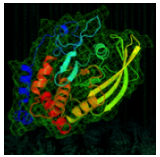
U.S. DEPARTMENT OF ENERGY | Office of Science

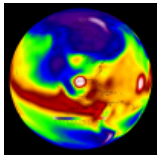
BERKELEY LAB

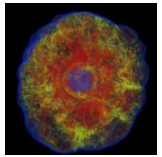
Use Case 1: BerkeleyGW

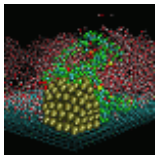






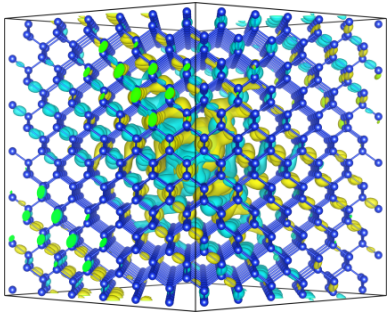


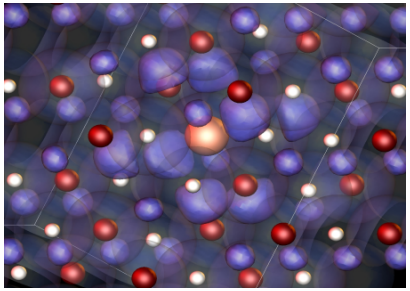


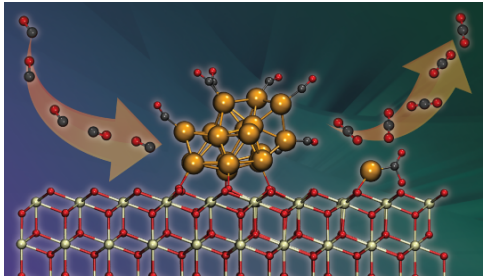


NERSC

Material Science/Chemistry at Exascale

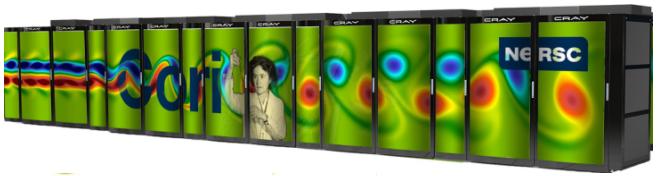






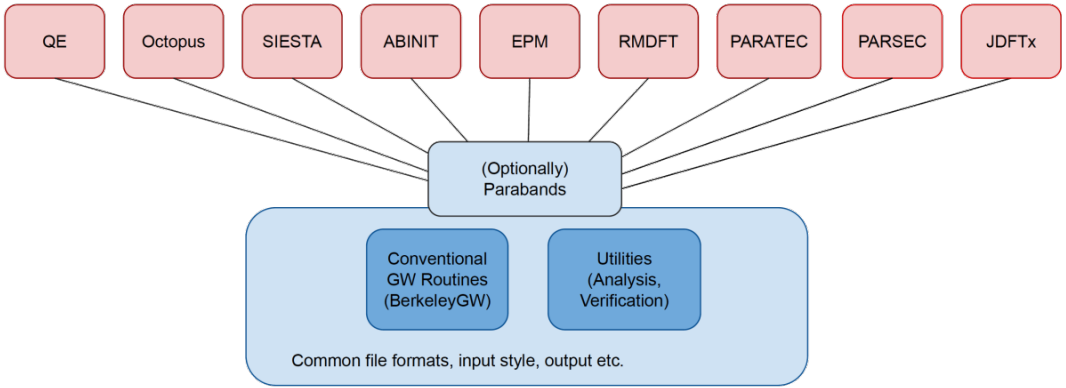
Mat. Sci & Chem apps like **VASP, Quantum ESPRESSO, NWChem, GAMESS, QMCPACK, BerkeleyGW, and CP2K** are some of the most heavily used apps at DOE facilities.

They are being used to design and understand the fundamental components of **Quantum Computers, Solar Cells, OLEDs, Batteries, Catalysts, Bio-Energy, Semiconductors, Sensors, Hydrogen Storage, Carbon Sequestration**



BerkeleyGW

- A massively parallel package for GW calculations
- Sits on top of DFT codes
- Studies Excited-State properties of materials
  - Photovoltaics
  - LEDs
  - Quantum Computers
  - Junctions / Interfaces
  - Defect Energy Levels
  - ....



BerkeleyGW

C2SEPEM

Sigma-GPP

Pseudo Code

```
do n1 = 1, nbands                n' e.g. 2763
   do igp = 1, ngpown            G' e.g. 6633
      do ig = 1, ncouls          G  e.g. 26529
         do iw = 1, nw           E  e.g. 3
            compute:   1. mixed data types
                          e.g. complex double, double, integer
                       2. various memory access patterns
                          e.g. (ig,igp)(ig,n1)(igp,n1)(iw,n1)(n1)
                       3. complex number divisions
                       4. nw is very small, will be unrolled

            reduction: 1. complex numbers
                       2. all top 3 loops, billions of iterations
```

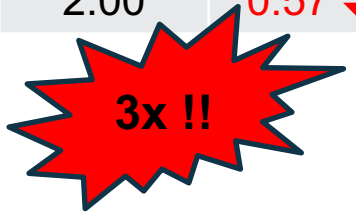
Optimization Path

9 Steps:

1. Collapse n', G', and G loops
2. Bring n' loop in; collapse only G' and G
3. Adjust threadblock size
4. Reduce branching; pull iw loop outside
5. Swap indices to suite parallelisation
6. Simplify code
7. Replace div. with rcp. and mul.
8. Replace abs with power of 2
9. Cache blocking

```
!$ACC PARALLEL LOOP REDUCTION(+: )
do n1 = 1, nbands
   do igp = 1, ngpown
      do ig = 1, ncouls
         do iw = 1, nw
            compute and reduction
```

	TFLOPs	Time	TFLOP/s
v1.collapse3	3.71	1.63	2.27
v9.block	2.00	0.57	3.50



V1. Naïve Implementation

- Collapse the first 3 loops to gain parallelism

```
!$ACC PARALLEL LOOP COLLAPSE(3) REDUCTION(+: )
do n1 = 1, nbands
   do igp = 1, ngpown
      do ig = 1, ncouls
         do iw = 1, nw        #unrolled
            compute and reduction
```

	TFLOPs	Time (sec)	TFLOP/s
v1.collapse3	3.71	1.63	2.27

V2. More Compute Per Thread

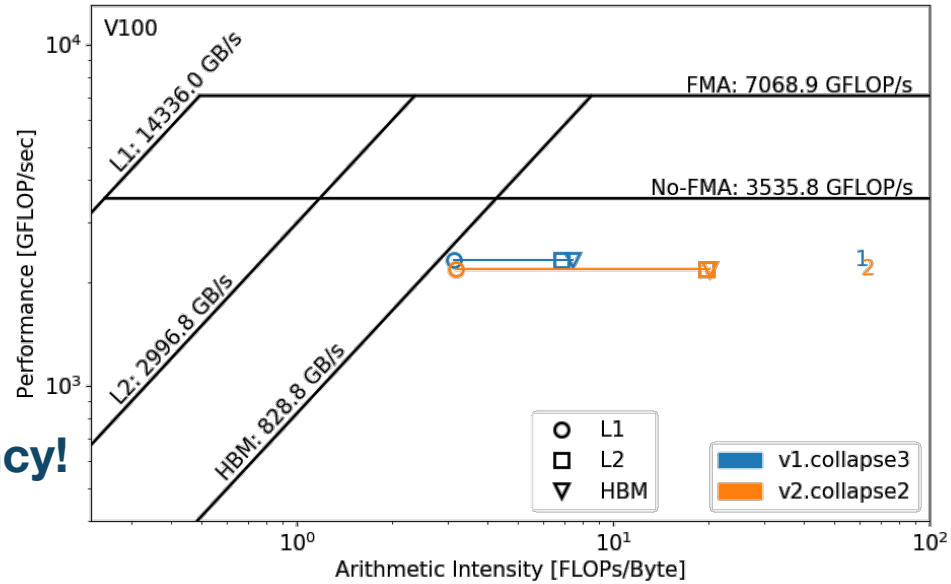
- Move n' loop in, and collapse the first 2 loops

```
!$ACC PARALLEL LOOP COLLAPSE(2) REDUCTION(+: )
do igp = 1, ngpown
   do ig = 1, ncouls
      do n1 = 1, nbands       #unrolled too!
         do iw = 1, nw        #unrolled
           compute and reduction
```

	TFLOPs	Time	TFLOP/s
v1.collapse3	3.71	1.63	2.27
v2.collapse2	3.71	1.73	2.15

V2. More Compute Per Thread

- L2/HBM AI increases!
- Very low occupancy
  - 8 warps per SM
  - Register count at 186
- Need more warps to hide latency!



Active Warps Per Scheduler [warp]	2.00	Instructions Per Active Issue Slot [inst/cycle]	1
Eligible Warps Per Scheduler [warp]	0.35	No Eligible [%]	70.01
Issued Warp Per Scheduler	0.30	One or More Eligible [%]	29.99

V3. Increase Threadblock Size

- Force threadblock size to be 512, instead of the default 128

```
!$ACC PARALLEL LOOP COLLAPSE(2) VECTOR_LENGTH(512) REDUCTION(+: )
```

- Register spills but performance may not be bad!

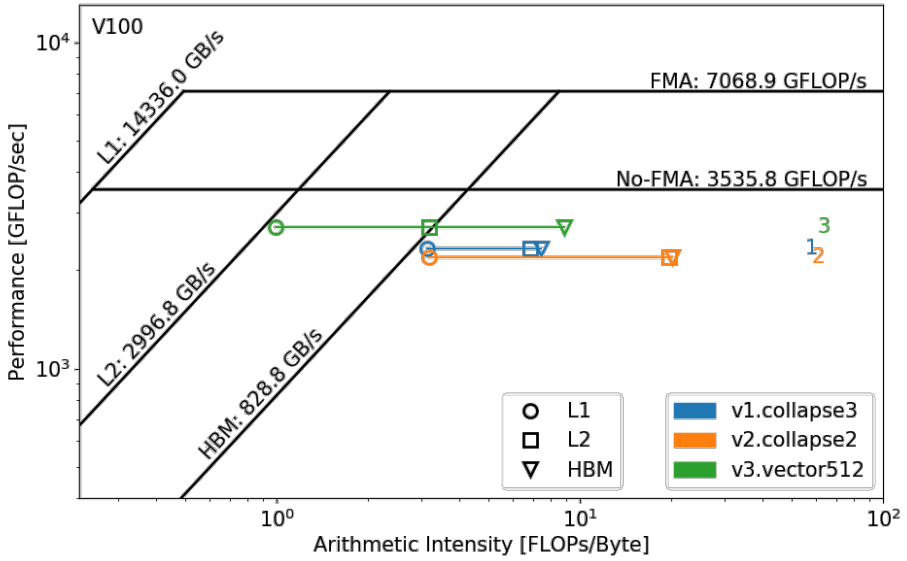
```
    0 bytes stack frame, 0 bytes spill stores, 0 bytes spill loads
ptxas info    : Used 186 registers, 624 bytes cmem[0], 32 bytes cmem[2]
```

```
    104 bytes stack frame, 188 bytes spill stores, 168 bytes spill loads
ptxas info    : Used 128 registers, 624 bytes cmem[0], 32 bytes cmem[2]
```

V3. Increase Threadblock Size

- More bandwidth bound now but latency hiding is successful!

	TFLOPs	Time	TFLOP/s
v2.collapse2	3.71	1.73	2.15
v3.vector512	3.71	1.40	2.65



Active Warps Per Scheduler [warp]	4.00	Instructions Per Active Issue Slot [inst/cycle]	1
Eligible Warps Per Scheduler [warp]	0.67	No Eligible [%]	58.21
Issued Warp Per Scheduler	0.42	One or More Eligible [%]	41.79

V4. Reduce Branching

- Bring iw loop outside of the kernel

```
do iw = 1, nw                          #reduce branching
!$ACC PARALLEL LOOP COLLAPSE(2) VECTOR_LENGTH(512) REDUCTION(+: )
do igp = 1, ngpown
   do ig = 1, ncouls
      do n1 = 1, nbands          #unrolled
         compute and reduction
```

- Fewer variables to be reduced -> lower register pressure

```
    0 bytes stack frame, 0 bytes spill stores, 0 bytes spill loads
ptxas info    : Used 122 registers, 600 bytes cmem[0], 32 bytes cmem[2]
```

V4. Reduce Branching

- Aggregated data for all kernels

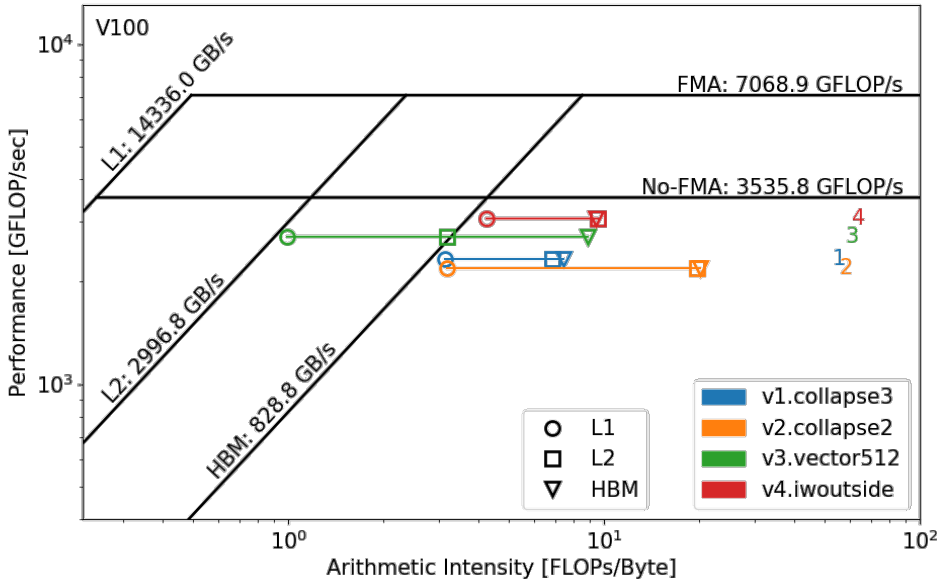
- BRA instruction count

  14,278,897,053

  5,975,051,812 x 2

16%

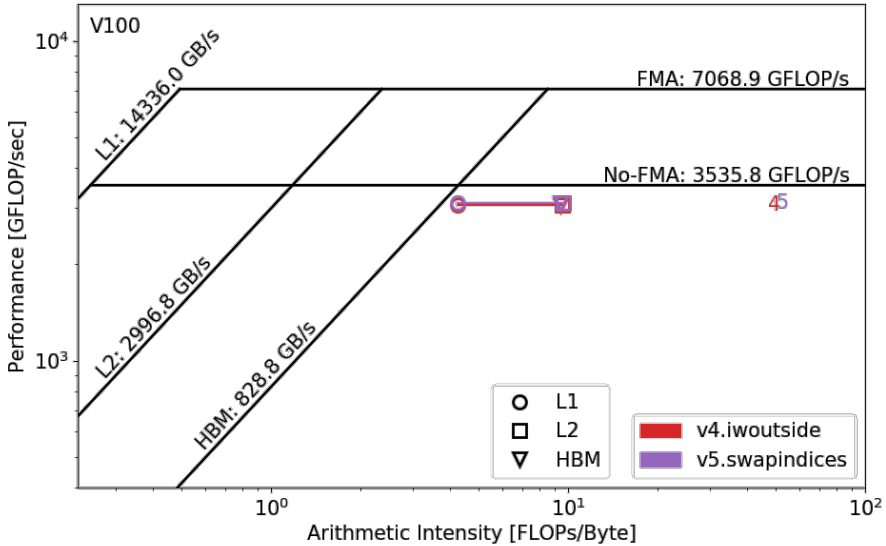
	TFLOPs	Time	TFLOP/s
v3.vector512	3.71	1.40	2.65
v4.iwoutside	3.52	1.17	3.00



V5. Swap Indices

```
do iw = 1, nw
!$ACC PARALLEL LOOP
do igp = 1, ngpown
   do ig = 1, ncouls
      do n1 = 1, nbands
       wx_array(iw,n1) to (n1,iw)
```

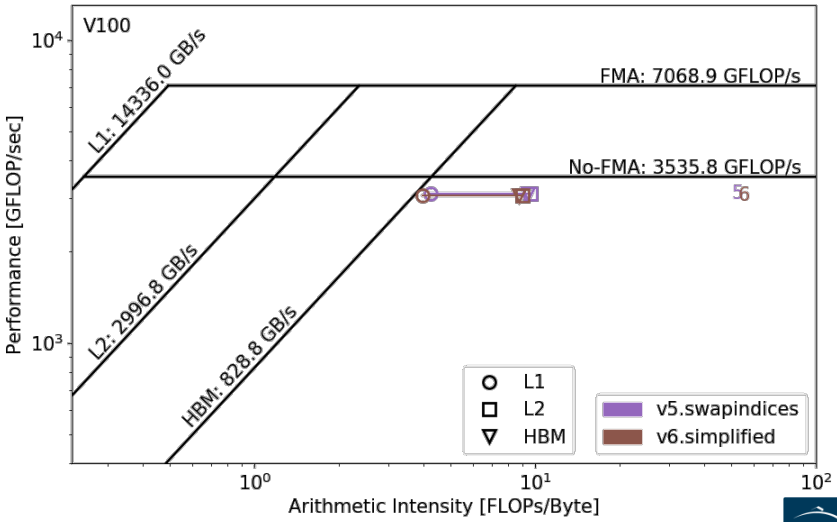
	TFLOPs	Time	TFLOP/s
v4.iwoutside	3.52	1.17	3.00
v5.swapindices	3.52	1.16	3.03



V6. Simplify Code

- Fewer instructions -> less work
  - Pull repeated instructions outside the loop
  - Use temporary variables to hold intermediate values for reuse
- Less branches -> better programming
  - 3 branches is more than 1 branch
    worse than 2 branches!

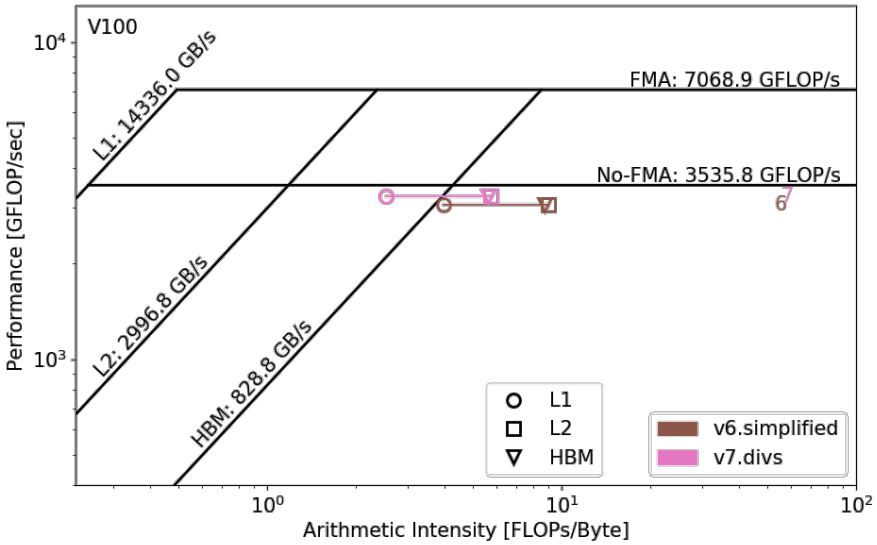
	TFLOPs	Time	TFLOP/s
v5.swapindices	3.52	1.16	3.03
v6.simplify	3.30	1.10	3.00



V7. Replace Divides

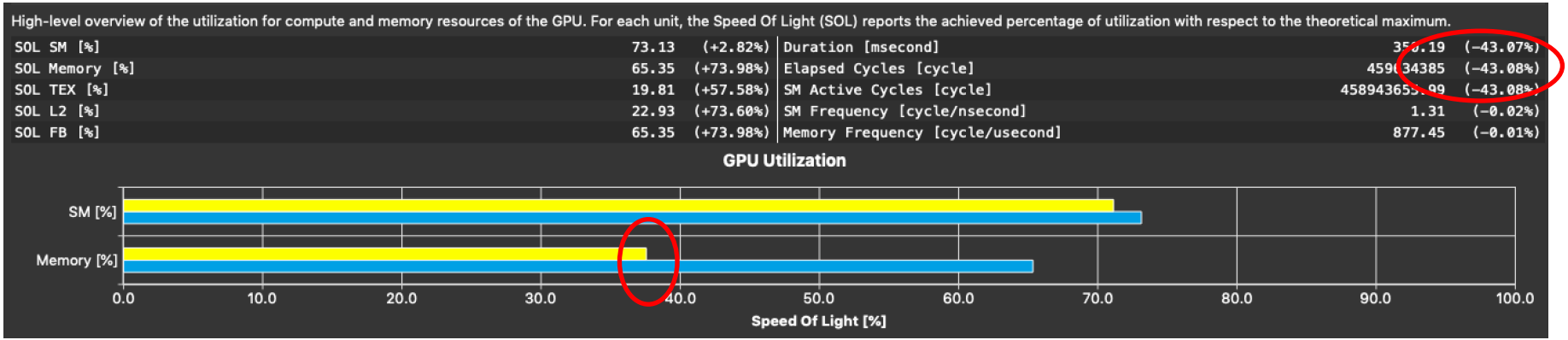
- Replace (complex) div. with (double) rcp. and (complex) mul.
- Lower instruction count: 40%
- More bandwidth bound now!

	TFLOPs	Time	TFLOP/s
v6.simplify	3.30	1.10	3.00
v7.divs	2.09	0.66	3.18



V7. Replace Divides

- Can be confirmed by Nsight Compute profiles



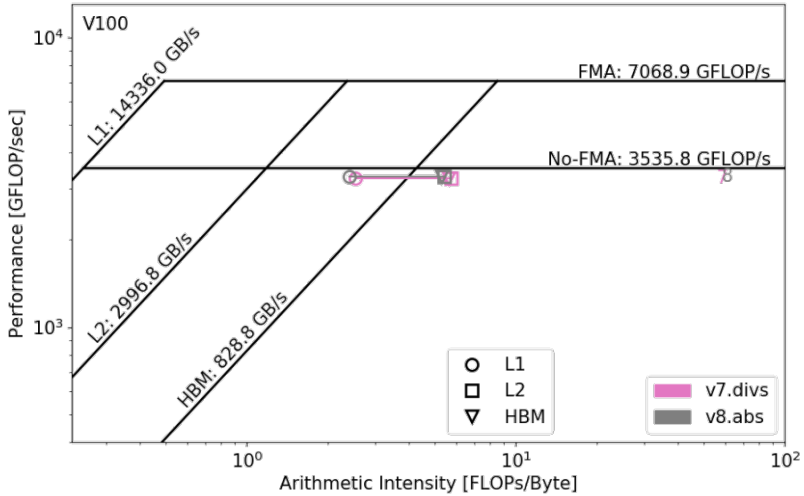
V8. Replace abs(x) with x**2

```
complex(DP) ssx
if (abs(ssx) .le. ssxcutoff) then
```

```
real(DP) ssxpower
if (ssxpower .le. ssxcutoff **2) then
```

- $\sqrt{x^2 + y^2} < z$ → $x^2 + y^2 < z^2$
- sqrt(complex) vs power of 2
- Causing pipeline to wait

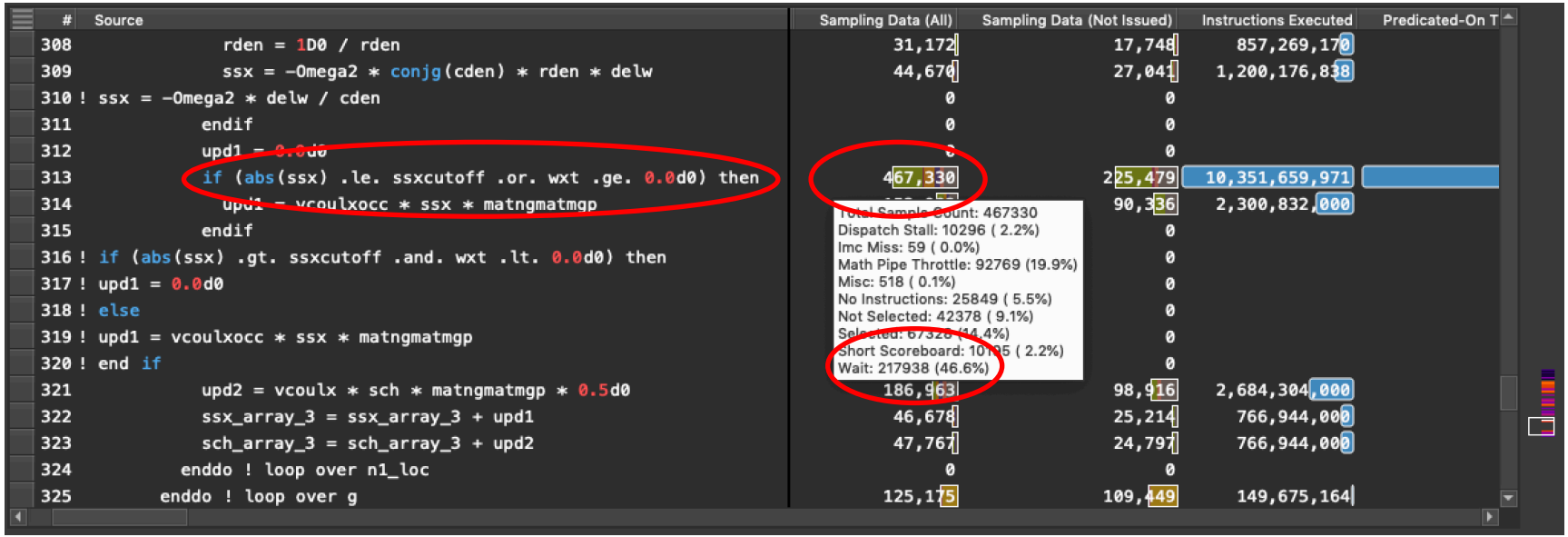
	TFLOPs	Time	TFLOP/s
v7.divs	2.09	0.66	3.18
v8.abs	1.99	0.62	3.23



V8. Replace abs(x) with x**2

Before:

- **Wait:** warp stalled waiting on a fixed latency execution dependency



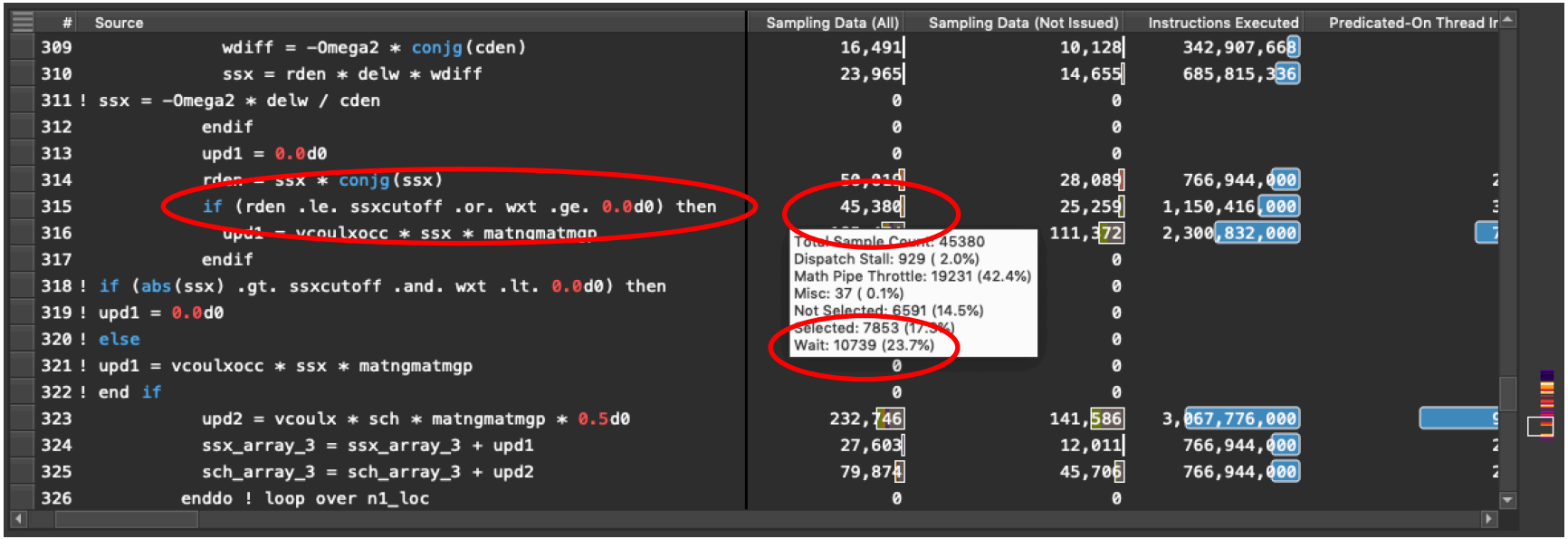
#	Source	Sampling Data (All)	Sampling Data (Not Issued)	Instructions Executed	Predicated-On T
308	rden = 1D0 / rden	31,172	17,748	857,269,170	
309	ssx = -Omega2 * conjg(cden) * rden * delw	44,670	27,041	1,200,176,838	
310	! ssx = -Omega2 * delw / cden	0	0		
311	endif	0	0		
312	upd1 = 0.0d0	0	0		
313	if (abs(ssx) .le. ssxcutoff .or. wxt .ge. 0.0d0) then	467,330	225,479	10,351,659,971	
314	upd1 = vcoulxocc * ssx * matngmatmgp		90,336	2,300,832,000	
315	endif		0		
316	! if (abs(ssx) .gt. ssxcutoff .and. wxt .lt. 0.0d0) then		0		
317	! upd1 = 0.0d0		0		
318	! else		0		
319	! upd1 = vcoulxocc * ssx * matngmatmgp		0		
320	! end if		0		
321	upd2 = vcoulx * sch * matngmatmgp * 0.5d0	186,963	98,916	2,684,304,000	
322	ssx_array_3 = ssx_array_3 + upd1	46,678	25,214	766,944,000	
323	sch_array_3 = sch_array_3 + upd2	47,767	24,797	766,944,000	
324	enddo ! loop over n1_loc	0	0		
325	enddo ! loop over g	125,175	109,449	149,675,164	

Total Sample Count: 467330
Dispatch Stall: 10296 (2.2%)
Imc Miss: 59 (0.0%)
Math Pipe Throttle: 92769 (19.9%)
Misc: 518 (0.1%)
No Instructions: 25849 (5.5%)
Not Selected: 42378 (9.1%)
Selected: 67328 (14.4%)
Short Scoreboard: 10195 (2.2%)
Wait: 217938 (46.6%)

V8. Replace abs(x) with x**2

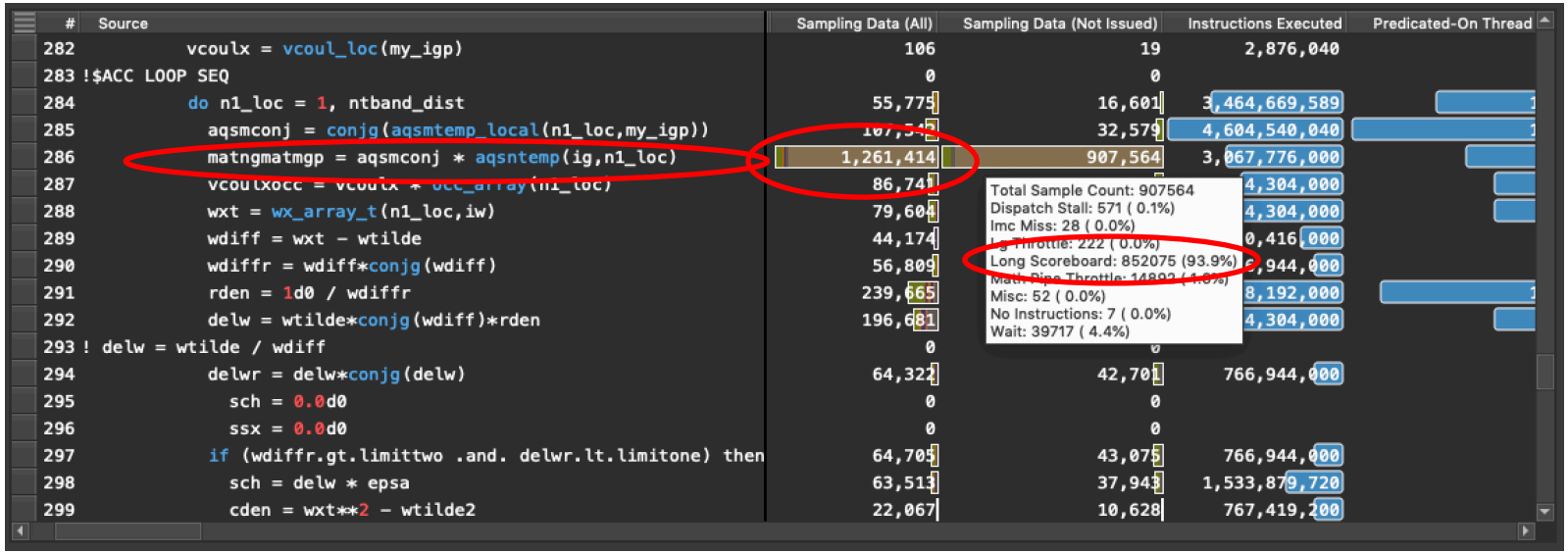
After:

- Wait: 46.6% -> 23.7%



V9. Cache Blocking

- **Non-coalesced memory access for `aqsntemp`**
- **Causing Long Scoreboard Warp State**
  - Warp stalled waiting for L1TEX (local, global, surface, tex) memory operation



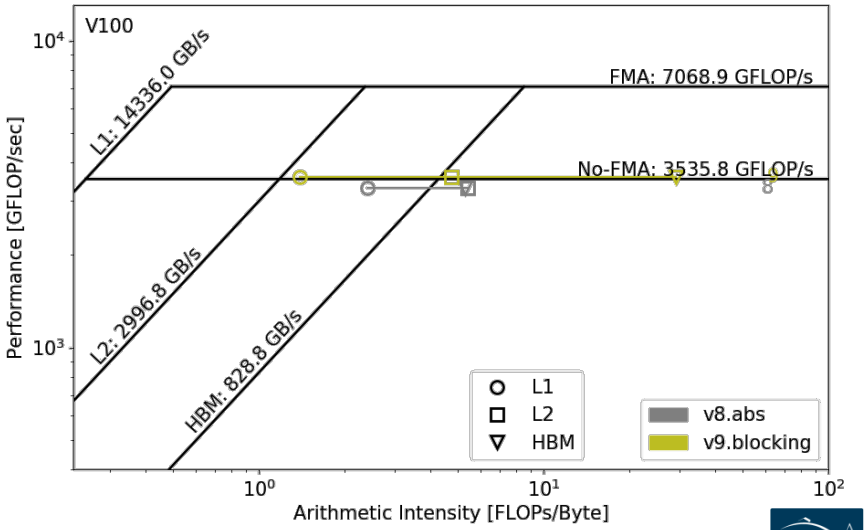
#	Source	Sampling Data (All)	Sampling Data (Not Issued)	Instructions Executed	Predicated-On Thread
282	vcoulx = vcoul_loc(my_igp)	106	19	2,876,040	
283	!$ACC LOOP SEQ	0	0		
284	do n1_loc = 1, ntband_dist	55,775	16,601	3,464,669,589	
285	aqsmconj = conjg(aqsmtemp_local(n1_loc,my_igp))	107,545	32,579	4,604,540,040	
286	matngmatmgp = aqsmconj * aqsntemp(ig,n1_loc)	1,261,414	907,564	3,067,776,000	
287	vcoulxocc = vcoulx * occ_array(n1_loc)	86,741		4,304,000	
288	wxt = wx_array_t(n1_loc,iw)	79,604		4,304,000	
289	wdiff = wxt - wtilde	44,174		0,416,000	
290	wdiffr = wdiff*conjg(wdiff)	56,809		2,944,000	
291	rden = 1d0 / wdiffr	239,665		8,192,000	
292	delw = wtilde*conjg(wdiff)*rden	196,681		4,304,000	
293	! delw = wtilde / wdiff	0	0		
294	delwr = delw*conjg(delw)	64,322	42,701	766,944,000	
295	sch = 0.0d0	0	0		
296	ssx = 0.0d0	0	0		
297	if (wdiffr.gt.limittwo .and. delwr.lt.limitone) then	64,705	43,075	766,944,000	
298	sch = delw * epsa	63,513	37,943	1,533,879,720	
299	cden = wxt**2 - wtilde2	22,067	10,628	767,419,200	

V9. Cache Blocking

- Break loops into chunks and reuse data across threadblocks
- Increase L2 hit rate

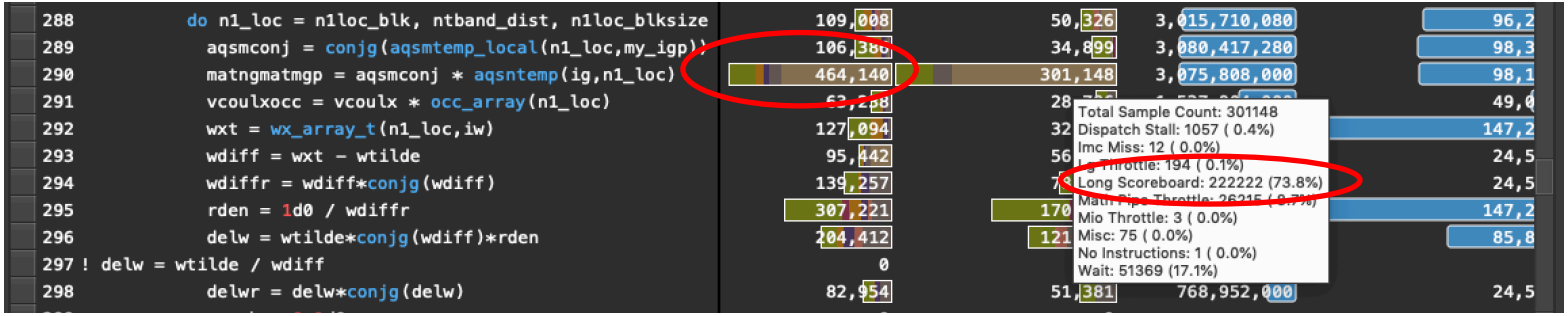
```
!$ACC LOOP GANG VECTOR
do ig_blk = 1, ig_blksize
!$ACC LOOP SEQ
  do ig = ig_blk, ncouls, ig_blksize
```

	TFLOPs	Time	TFLOP/s
v8.abs	1.99	0.62	3.23
v9.block	2.00	0.57	3.50



V9. Cache Blocking

- Less Long Scoreboard samples and higher L2/L1 hit rate



Memory Workload Analysis

Detailed analysis of the memory resources of the GPU. Memory can become a limiting factor for the overall kernel performance when fully utilizing the involved hardware units (Mem Busy), exhausting the available communication bandwidth between those units (Max Bandwidth), or by reaching the maximum throughput of issuing memory instructions (Mem Pipes Busy).

Memory Throughput [Gbyte/second]	131.85 (-78.99%)	Mem Busy [%]	31.10 (+29.96%)
L1 Hit Rate [%]	71.38 (+27.27%)	Max Bandwidth [%]	28.44 (-59.32%)
L2 Hit Rate [%]	88.89 (+3272.88%)	Mem Pipes Busy [%]	12.84 (+59.44%)

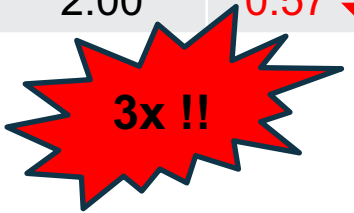
Summary

9 Steps to Optimize Sigma-GPP

1. Collapse n', G', and G loops
2. Bring n' loop in; collapse only G' and G
3. Adjust threadblock size
4. Reduce branching; pull iw loop outside
5. Swap indices to suite parallelisation
6. Simplify code
7. Replace div. with rcp. and mul.
8. Replace abs with power of 2
9. Cache blocking

```
!$ACC PARALLEL LOOP REDUCTION(+: )
do n1 = 1, nbands
   do igp = 1, ngpown
      do ig = 1, ncouls
         do iw = 1, nw
            compute and reduction
```

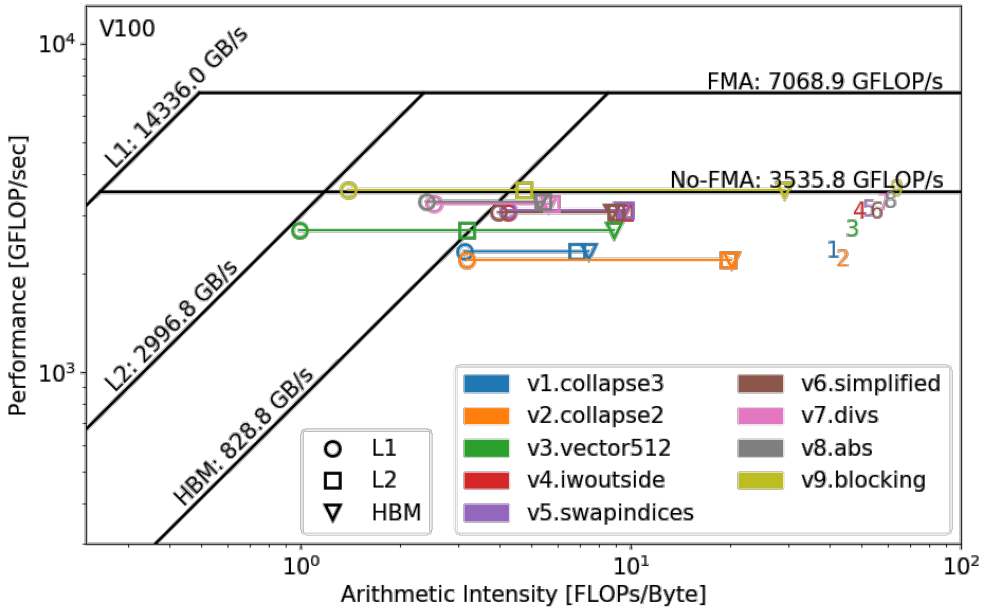
	TFLOPs	Time	TFLOP/s
v1.collapse3	3.71	1.63	2.27
v9.block	2.00	0.57	3.50



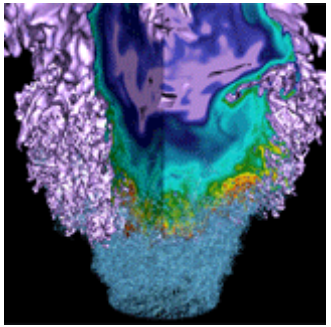
Summary

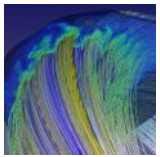
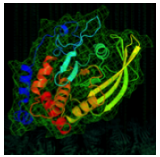
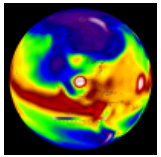
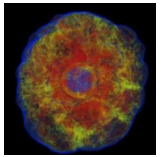
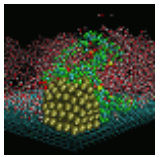
- Code is still bandwidth and latency bound
  - shared memory
  - lower register count
  - improve FMA ratio

- **Together with profilers, Roofline provides the complete solution for your performance analysis and optimization needs!**



Use Case 2:
conv2d from TensorFlow



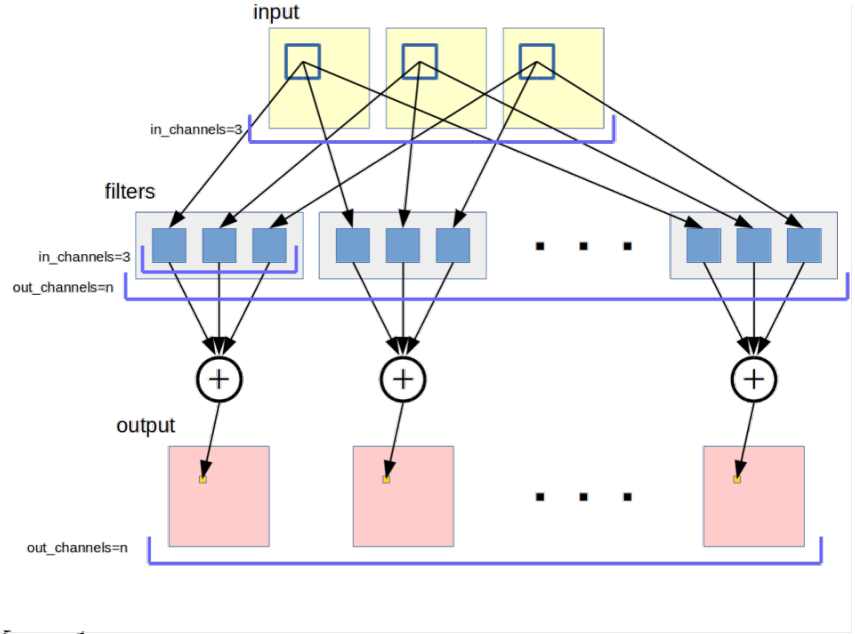
NERSC

conv2d from TensorFlow

- **Kernel tf.nn.conv2d**

TensorFlow

https://www.tensorflow.org



$$B_{nhwc} = \sum_{m=0}^{C-1} \sum_{k_h=0}^{K_H-1} \sum_{k_w=0}^{K_W-1} A_{n\ h+k_h\ w+w_h\ m}\ K_{k_h\ k_w\ m\ c}$$

conv2d from TensorFlow

exec_op:

- forward pass -- conv in 2D
- backward pass -- conv + derivative
- calibrate -- tensor generation

```
#choose operation depending on pass
if pass=="forward":
    with tf.device(gpu_dev):
    exec_op = output_result
elif pass=="backward":
    with tf.device(gpu_dev):
    opt = tf.train.Gradient\
        DescentOptimizer(0.5)
    exec_op = opt.compute\
        _gradients(output_result)
elif pass=="calibrate":
    with tf.device(gpu_dev):
    exec_op = input_image
```

```
#generate random input tensor
input_image = tf.random_uniform(shape=input_size, minval=0., maxval=1., dtype=dtype)
#create network
output_result = conv2d(input_image, 'NHWC', kernel_size, stride_size, dtype)
```

conv2d from TensorFlow

- TensorFlow autotuning mechanism
- Split the loop into ‘warm-up’ and ‘measurement’
  - 5 iters and 20 iters
- pyc.driver from PyCUDA
  - need to launch nvprof with `--profile-from-start off`

```
with tf.Session(config=...) as sess:
    ...

    #warm-up
    for i in range(n_warm):
    result = sess.run(exec_op)

    #measurement
    pyc.driver.start_profiler()
    for i in range(n_iter):
        result = sess.run(exec_op)
    pyc.driver.stop_profiler()
```

conv2d from TensorFlow

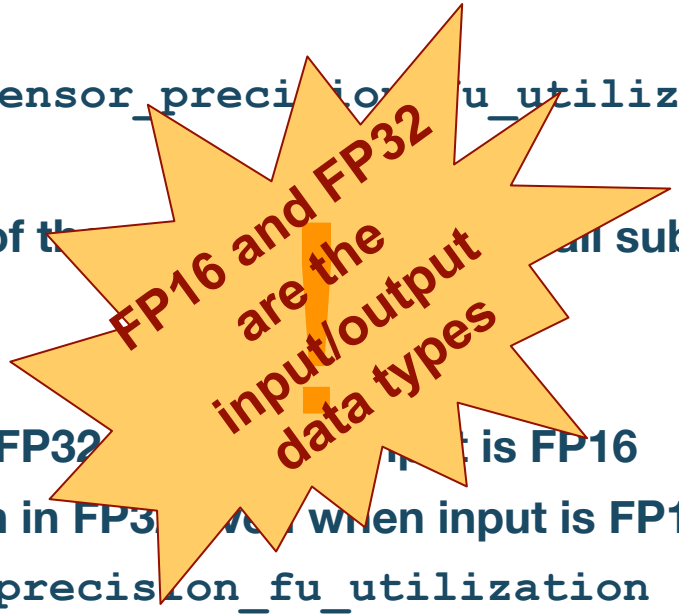
- Each TensorFlow kernel translates to a series of subkernels
  - padding, shuffling, data conversion, *etc*
- TensorFlow based on heuristics decides what subkernels to call
- cuDNN also has some algorithm selection mechanism
- We INCLUDE the housekeeping subkernels in our measurements, but EXCLUDE the autotuning subkernels

conv2d from TensorFlow

- Our FLOP count comes from

  `flop_count_sp, flop_count_hp, tensor_preci...fu_utilization`

- Byte count and run time are the sum of th... all subkernels



CAVEATS:

- Housekeeping subkernels may run in FP32 ...p is FP16
- TensorFlow may execute computation in FP3... when input is FP16
- Very coarse quantization for `tensor_precision_fu_utilization`
  - 0-10 integer range, 0 maps to 0 TFLOP/s and 10 maps to 125 TFLOP/s

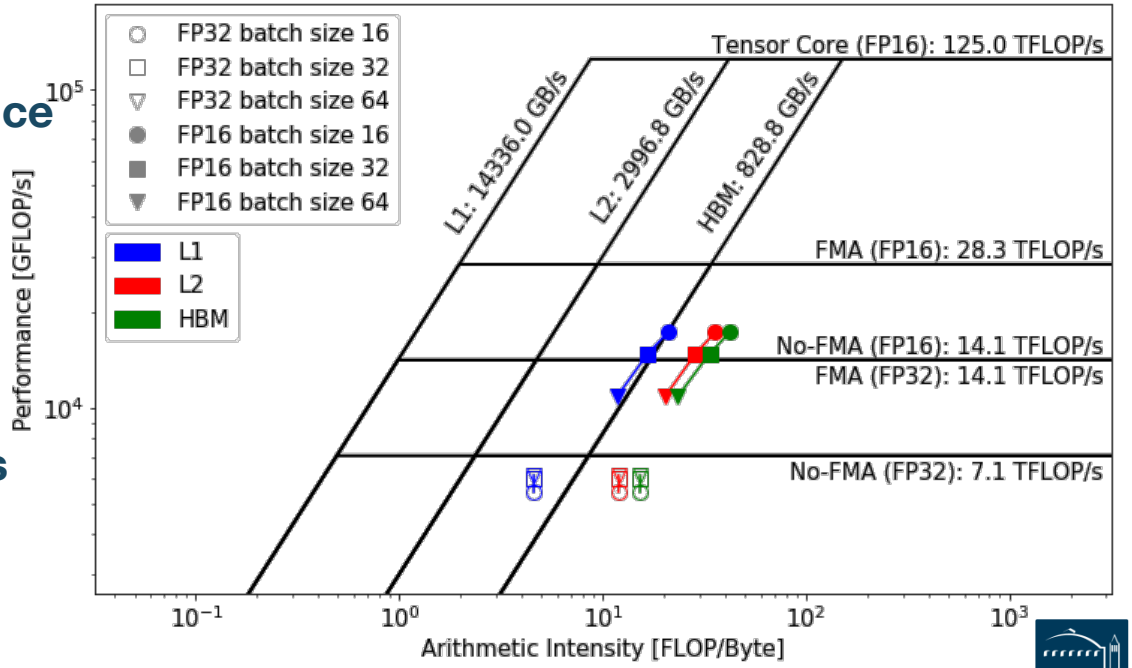
conv2d Analysis

- **Batch Size** 16, 32 and 64, forward pass

- Same underlying algorithm
  → should be same performance

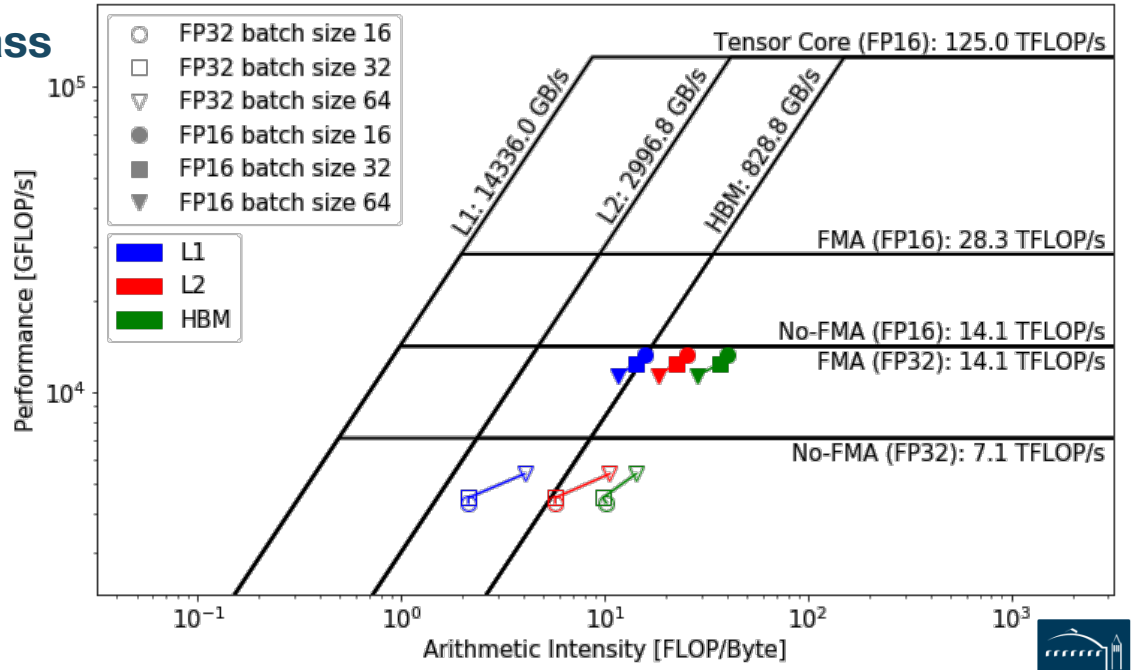
- But, housekeeping kernels are mostly bandwidth bound

- One reason TF applications are not reaching peak



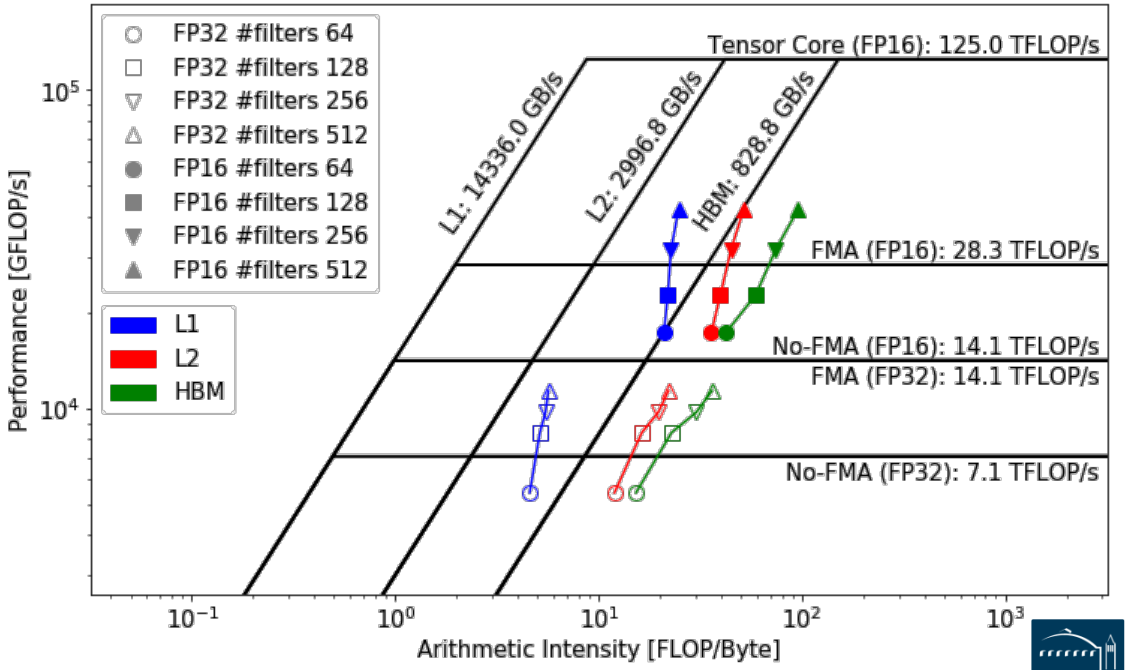
conv2d Analysis

- **Batch Size** 16, 32 and 64, backward pass
- Similar trend as forward pass
- But algorithm changes for FP32 at batch size 64, leading to slightly better performance



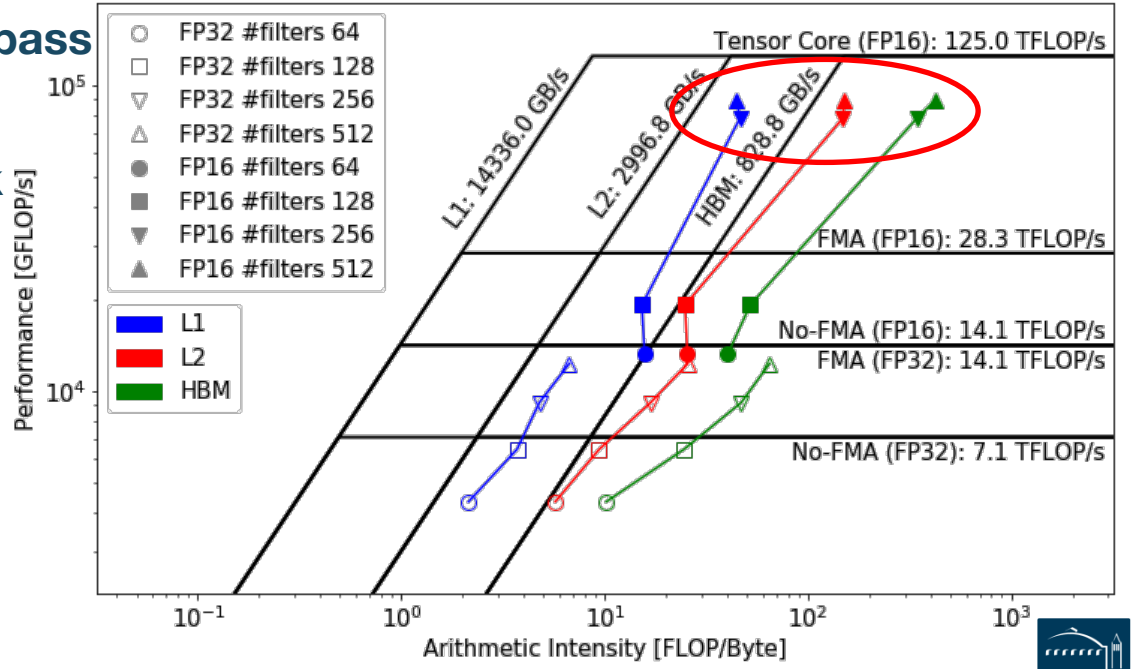
conv2d Analysis

- **Number of Output Filters** 64, 128, 256 and 512, forward pass
- Increasing intensity and performance
- Good L1 locality
- cuDNN uses shared mem



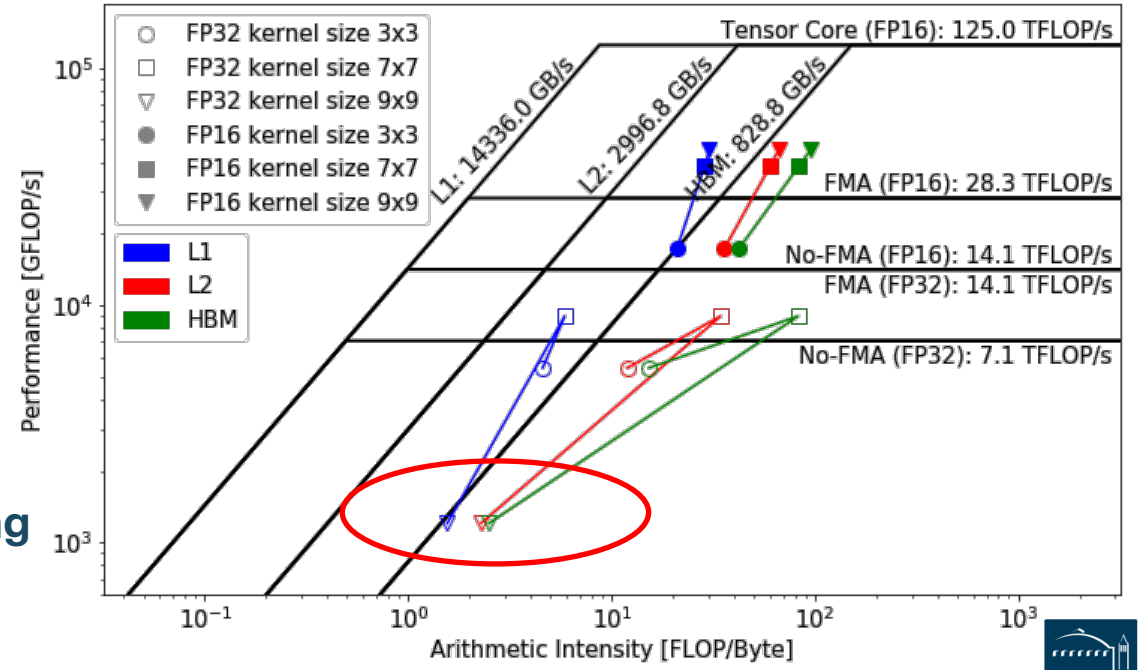
conv2d Analysis

- **Number of Output Filters** 64, 128, 256 and 512, backward pass
- Similar trend as forward pass
- Almost reaching TC peak and FP32 FMA peak



conv2d Analysis

- **Kernel Size** 3x3, 7x7 and 9x9, forward pass
- Increasing intensity and performance
- Algorithm change at 9x9
  - `wgrad` to FFT
  - may not be efficient use of FFT kernels
  - More robust autotuning

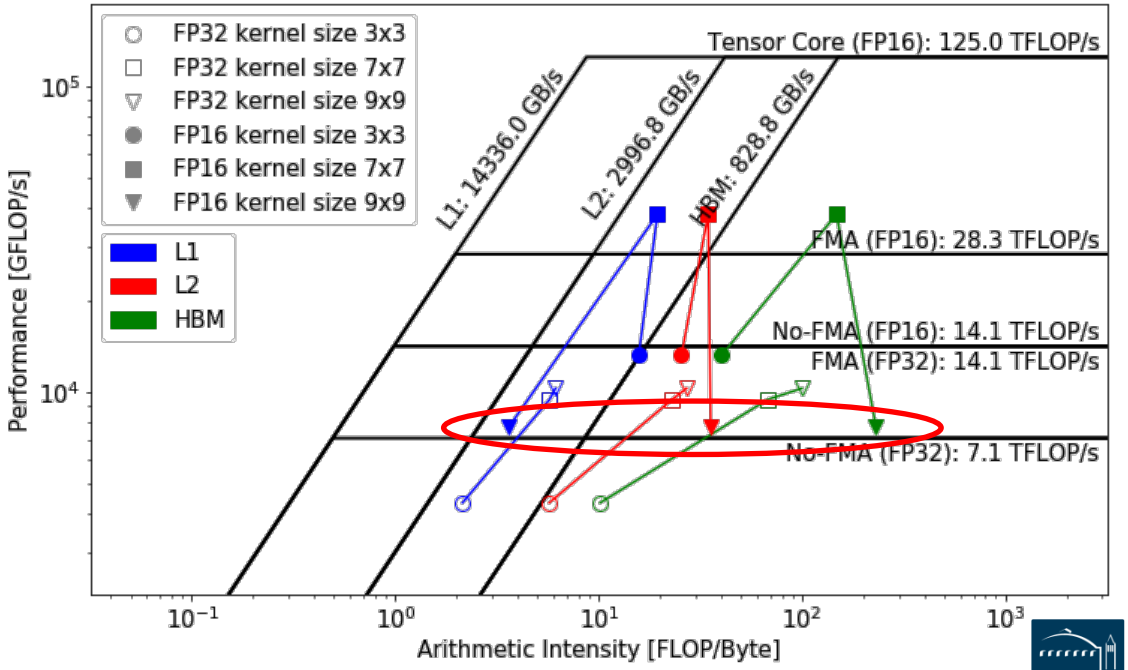


conv2d Analysis

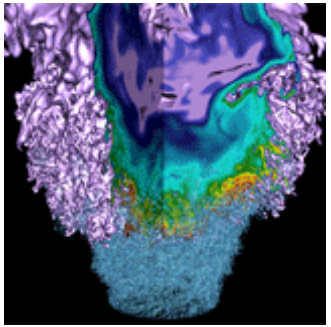
- **Kernel Size** 3x3, 7x7 and 9x9, backward pass

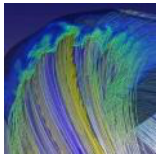
- TF decides to run in FP32 even though both input and output are in FP16; Data needs to be converted back and forth

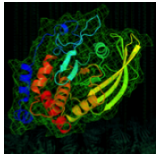
- Comply with dimension requirements

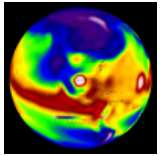


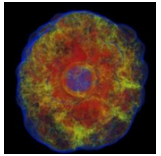
Summary

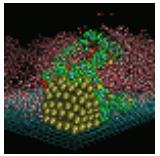












NERSC

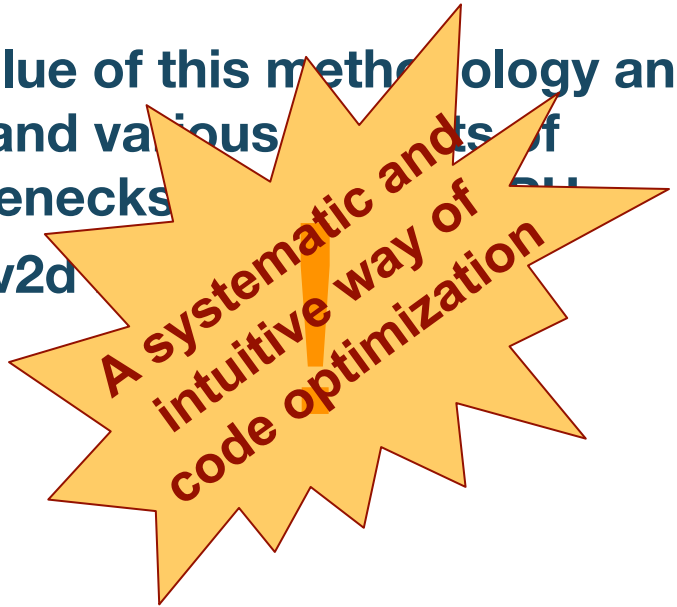
U.S. DEPARTMENT OF ENERGY | Office of Science

BERKELEY LAB

Summary

- We presented an effective methodology to collect machine and application data, and construct hierarchical Roofline on NVIDIA GPUs.

- Two use cases demonstrated the value of this methodology and showed its ability to readily understand various aspects of performance and performance bottlenecks on GPU
  - GPP from BerkeleyGW, and conv2d



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

Sam Williams, swwilliams@lbl.gov
Yunsong Wang, yunsongwang@lbl.gov
Charlene Yang, cjyang@lbl.gov