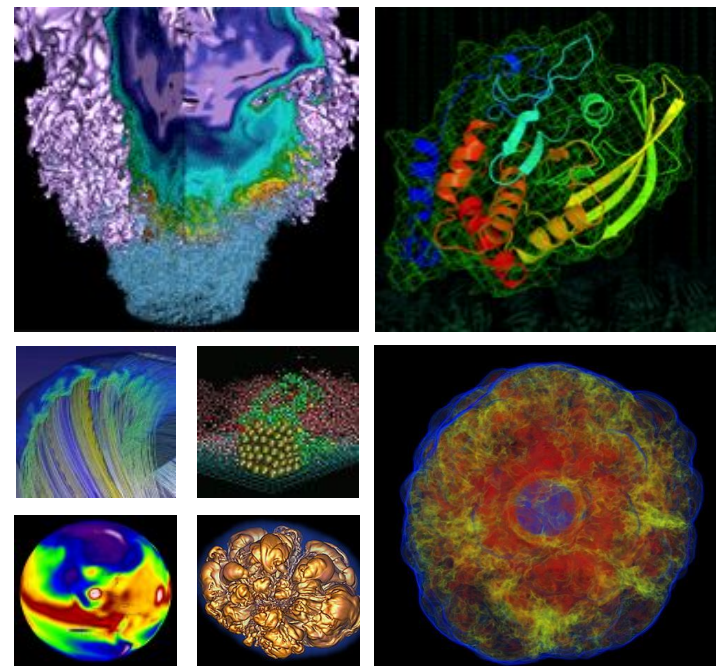


Evaluating Portability of OpenMP for SNAP using Roofline analysis



Neil Mehta, Rahul Gayatri, Yasaman Ghadar,
Christopher Knight, and Jack Deslippe



Adapting to Exascale



System	Perlmutter	Aurora	Frontier
Host	AMD Milan	Intel Xeon SR	AMD EPYC
Device	NVIDIA A100	Intel Xe Ponte Vecchio	AMD Radeon Instinct

Test-bed	Cori	JLSE Iris	Tulip
Host	Intel Skylake	Intel Skylake	AMD EPYC
Device	NVIDIA A100	Intel Gen9	AMD MI60
Compiler	LLVM 13/NVHPC 21.3	oneapi (20201008)	rocm 3.6.0 (aomp 11.0)

- **OpenMP 4.5** directives, as it requires less intensive code modifications and have compiler support by **all major GPU vendors**

Introduction to TestSNAP



- **J** determines **bispectrum**
- **TestSNAP proxy app** mimics computational load
- Test performance for **J = 2, 8, and 14** (**ECP FOM** problem size for **EXAALT MD project**)
- Number of atoms: 2000 atoms
- Number of steps: 100

```
for(int natom = 0; natom < num_atoms; ++natom)
{
    // build neighbor-list for all atoms
    build_neighborlist();

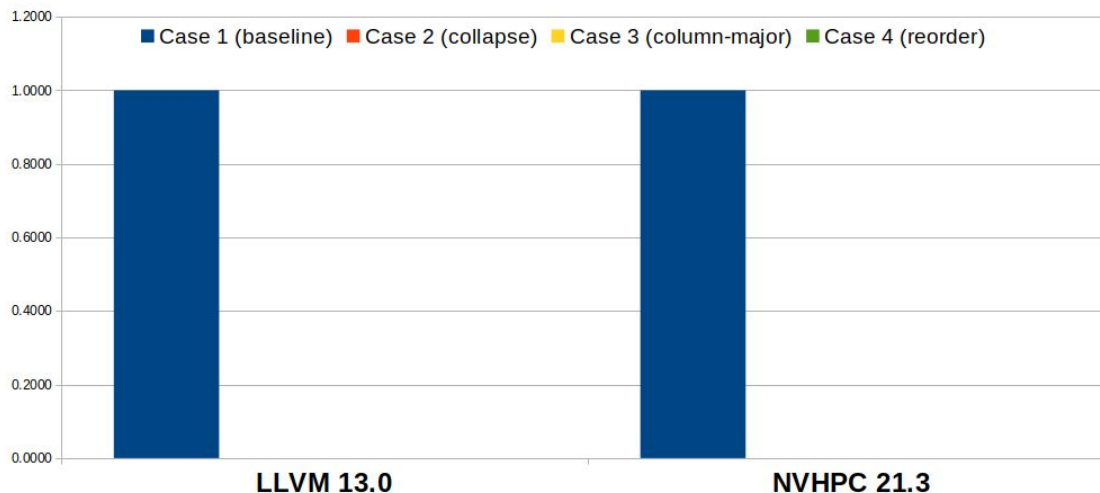
    // compute atom specific coefficients
    compute_U(); //Ulist[idx_max] and Ulisttot[idx_max]
    compute_Y(); //Ylist[idx_max]

    // for each (atom,neighbor) pair
    for(int nbor = 0; nbor < num_nbor; ++nbor)
    {
        compute_dU(); //dUlist[idx_max][3]
        compute_dE(); //dElist[3]
        update_forces()
    }
}
```

Kernel optimizations for OpenMP (1/4)

- Arrays created using **classes** that include pointer to **contiguous block of memory**
- Case 1: **baseline**

```
void add_uarraytot()  
{  
#pragma omp target teams distribute parallel for  
    for(int natom = 0; natom < num_atoms; ++natom)  
        for(int nbor = 0; nbor < num_nbor; ++nbor)  
            for(int j = 0; j < idxu_max; ++j)  
                ulisttot(natom,j) += ulist(natom,nbor,j);  
}
```



Run times:

llvm A100: 0.358 s

nvc++ A100 : 0.321 s

Kernel optimizations for OpenMP (2/4)

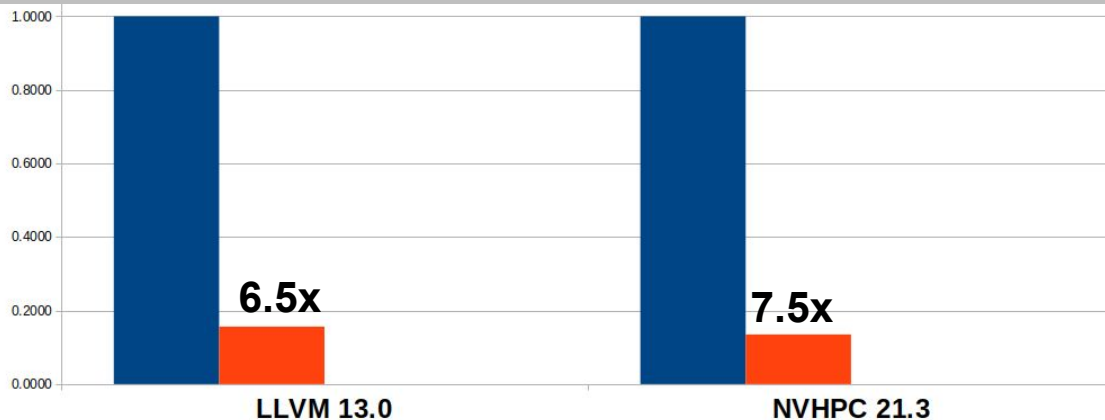
- Exploit the ability to **collapse** nested **for** loops
- Case 2: **collapse**

```
void add_uarraytot()  
{  
  #pragma omp target teams distribute parallel for collapse(2)  
  for(int natom = 0; natom < num_atoms; ++natom)  
    for(int nbor = 0; nbor < num_nbor; ++nbor)  
      for(int j = 0; j < idxu_max; ++j)  
      {  
        #pragma omp atomic  
        ulisttot(natom,j) += ulist(natom,nbor,j);  
      }  
}
```

Run times:

llvm A100: 0.0559 s

nvc++ A100 : 0.0432 s



Kernel optimizations for OpenMP (3/4)



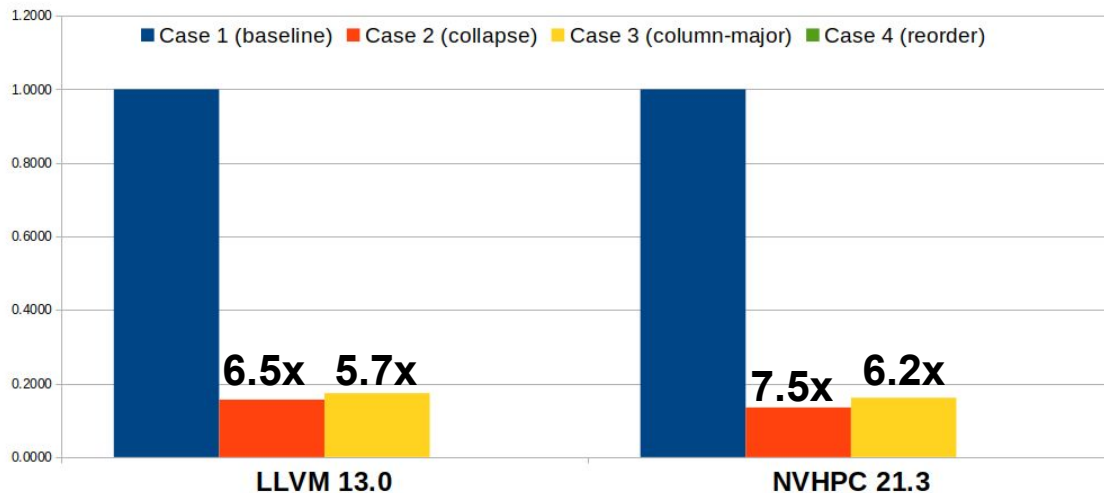
- **Column major data access:** atom loop as **fastest** moving index causes **performance degradation**
- **Case 3: column major**

Run times:

llvm A100: 0.0622 s

nvc++ A100 : 0.0516 s

```
operator()(int in1, int in2) {return dptr[in2*n1 + in1];}
```



Kernel optimizations for OpenMP (4/4)

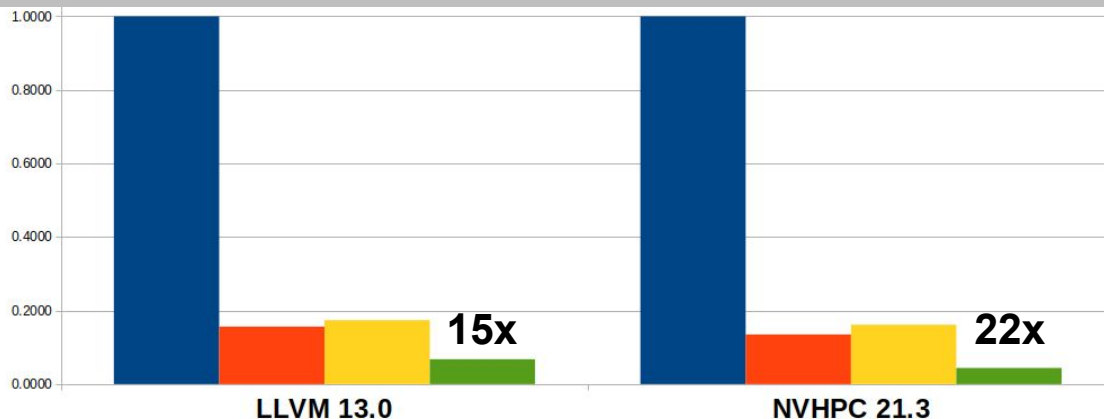
- Make **atom loop**
(**fastest moving index**)
as inner most loop
- Case 4: **reorder loop**

Run times:

llvm A100: 0.0241 s

nvc++ A100 : 0.0141 s

```
void add_uarraytot()  
{  
#pragma omp target teams distribute parallel for collapse(2)  
    for(int nbor = 0; nbor < num_nbor; ++nbor)  
        for(int natom = 0; nbor < num_atom: ++natom)  
            for(int j = 0; j < idxu_max; ++j)  
            {  
                #pragma omp atomic  
                ulisttot(natom,j) += ulist(natom,nbor,j);  
            }  
}
```



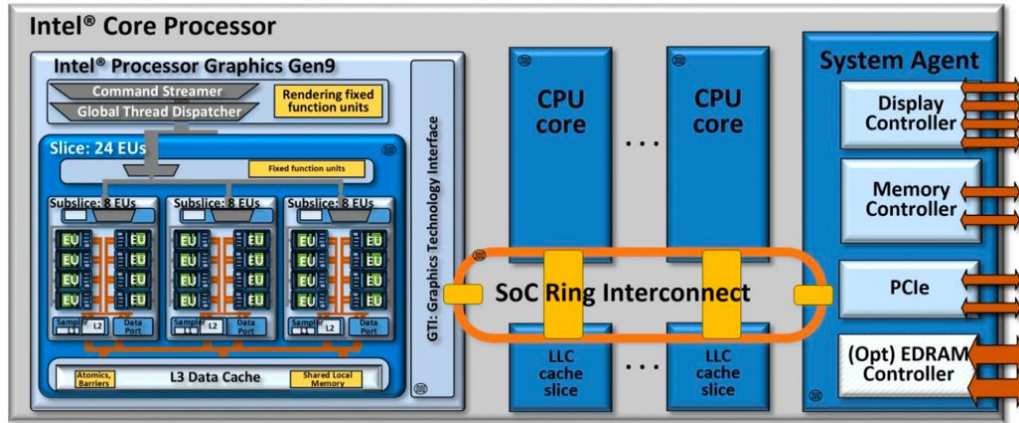
TestSNAP profiling data



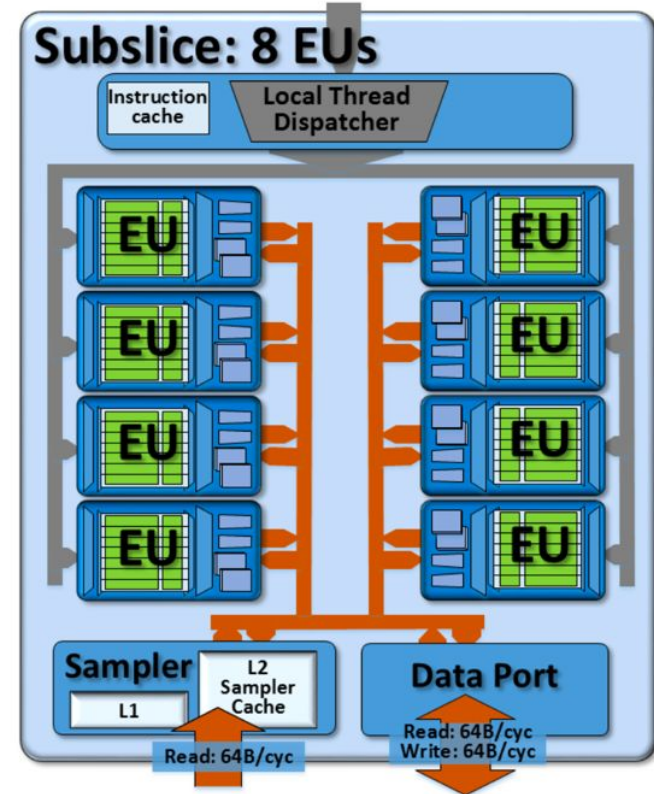
Version	Intel Gen9		AMD MI60		NVIDIA A100	
Rank	Time (%)	Kernel	Time (%)	Kernel	Time (%)	Kernel
1	65.65	compute_Y	57.01	compute_Y	56.30	compute_Y
2	19.15	compute_dU	31.53	compute_dU	22.80	compute_dU
3	10.58	compute_U	8.61	compute_U	14.30	compute_dE
4	4.02	compute_dE	2.44	compute_dE	5.70	memcpy HtoD
5	0.41	WriteBuffer	0.29	zero_uarraytot	0.70	zero_uarraytot

Version	Serial (Skylake)		OpenMP offload GPU		
	LLVM/11	ICX	Gen9	MI60	A100
Step time (s/step)	9.7671	9.8669	1.8215	0.1394	0.0281
Grind time (ms/atm-stp)	4.8835	4.9334	0.9107	0.0697	0.0141
compute_U (s)	0.6211	0.6221	0.1975	0.0153	0.0041
compute_Y (s)	7.6839	7.6789	1.2005	0.0748	0.0151
compute_dU (s)	1.2008	1.3363	0.3484	0.0389	0.0061
compute_dE (s)	0.2604	0.2288	0.0741	0.0086	0.0017

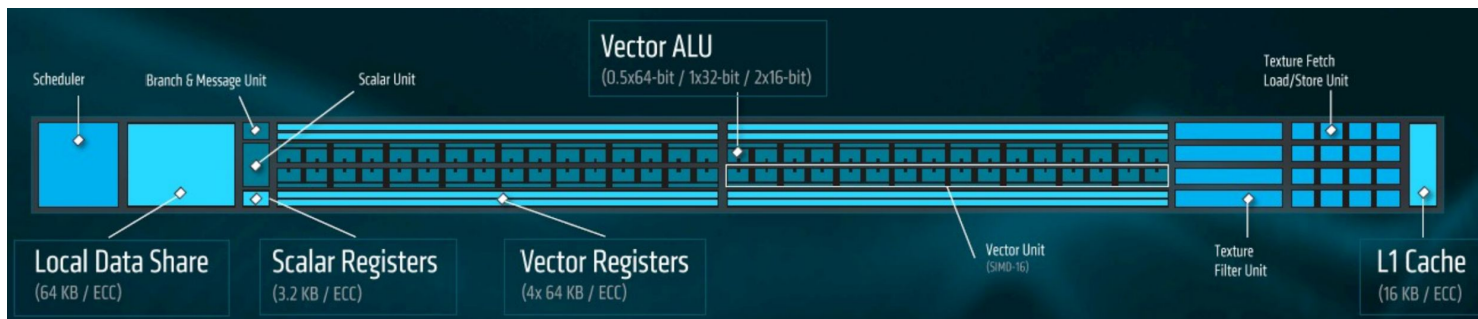
GPU architecture of Intel Gen9



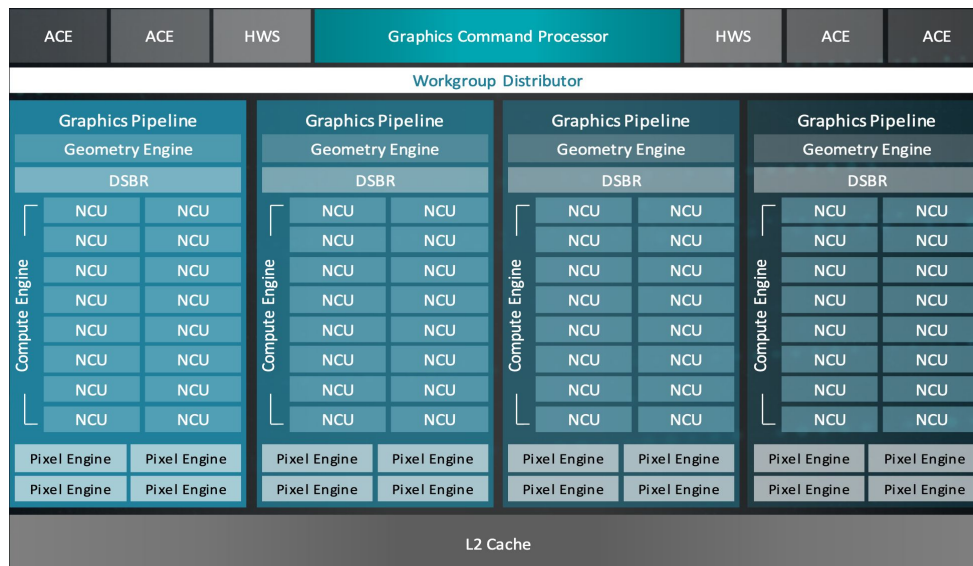
- Integrated GPU architecture
- 3 GPU slices, each having 3 sub-slices
- Each subslice has 8 execution unit (EU)
- Each EU has 7 thread
- Total of **504 threads** per GPU



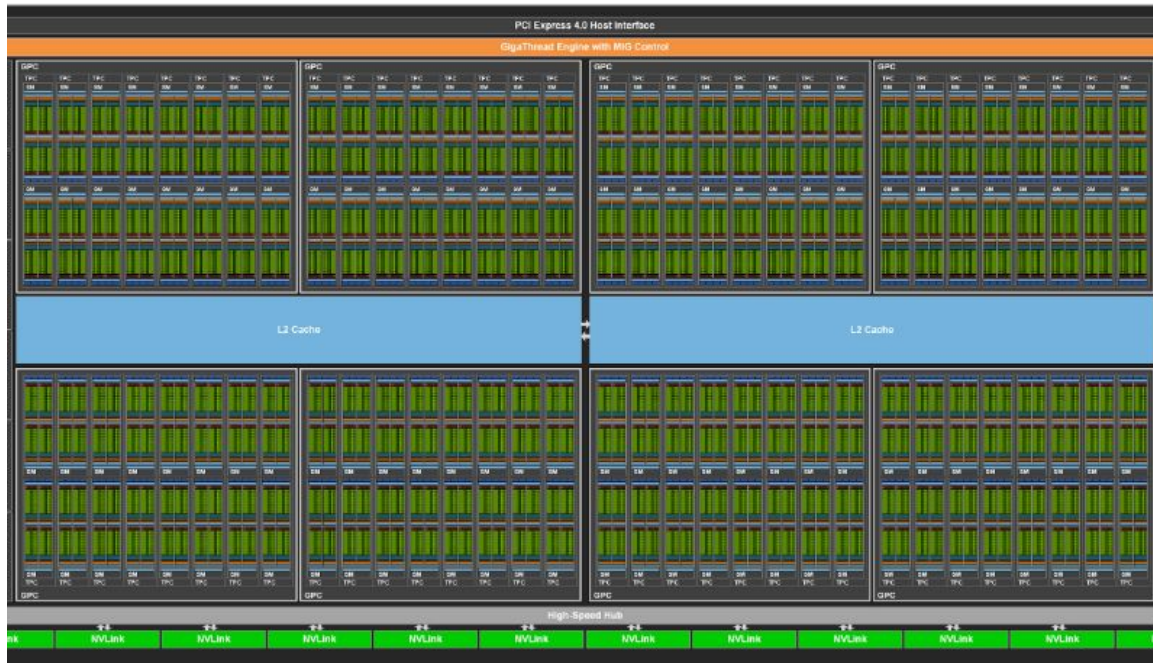
GPU architecture of AMD MI60



- Each GPU has 64 ‘Next-gen Compute Units’ or ‘NCUs’
- Analogous to NVIDIA warps, on AMD GPU, each ‘wavefront’ consists of 64 work items, i.e. threads
- MI60 capable of launching **4096 work items**



GPU architecture of NVIDIA A100



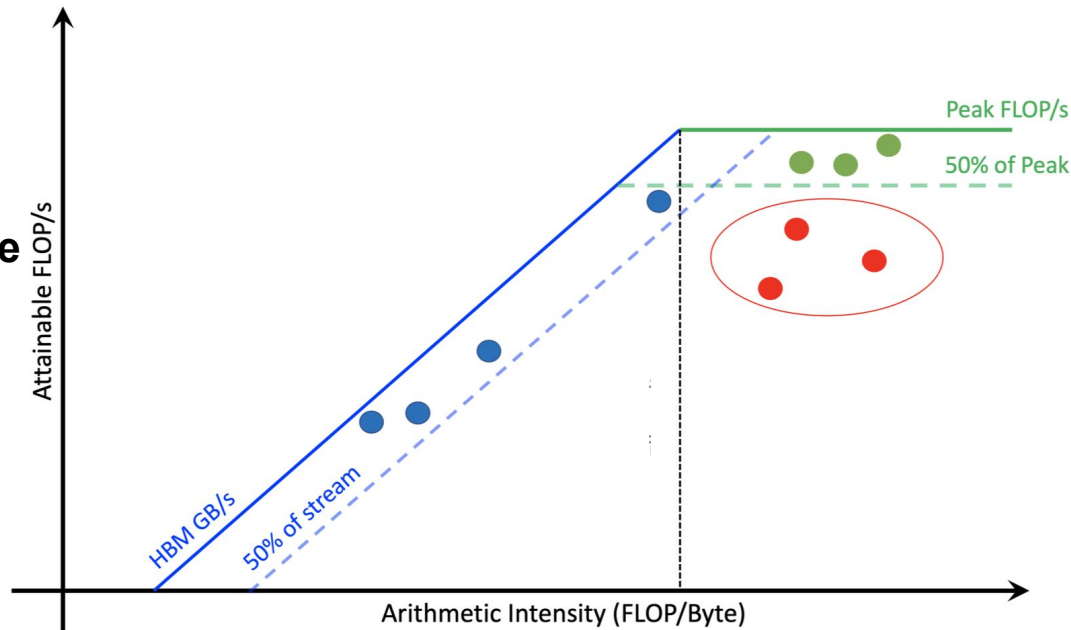
- 108 SM per GPU
- Total of **6912 FP32 cores** per GPU



Understanding Roofline model



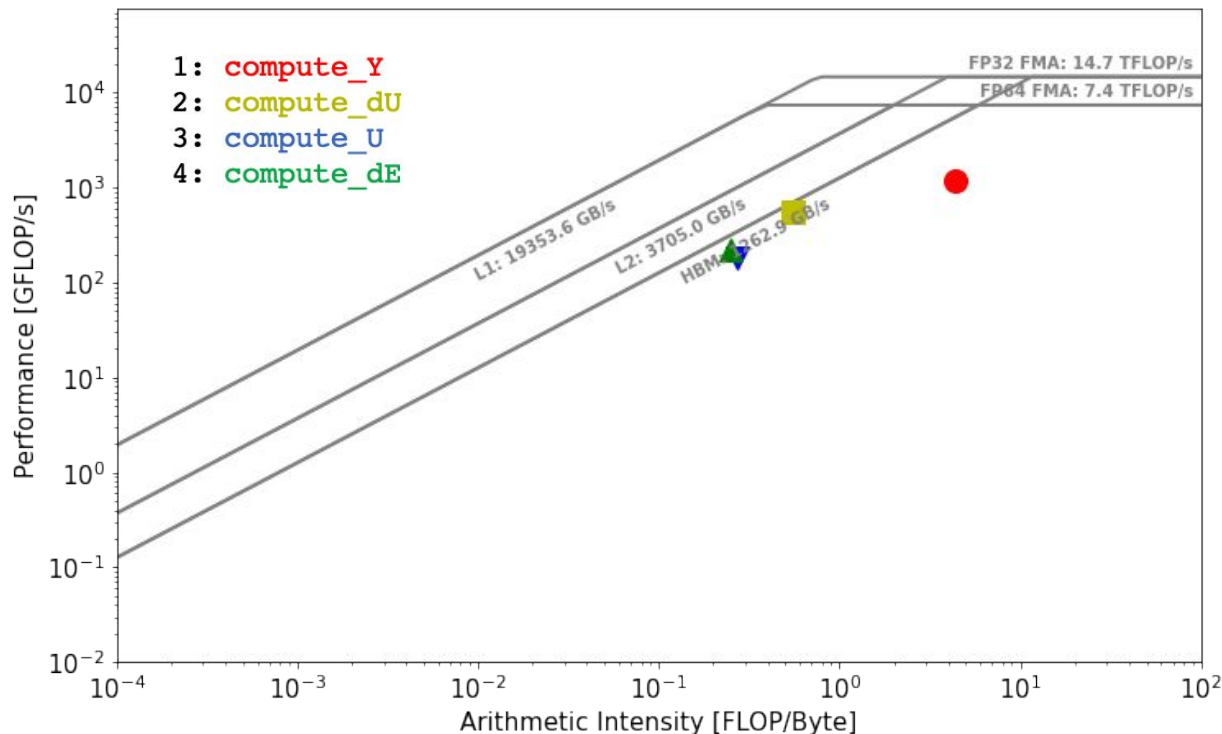
- Region to the left of 'machine balance line' represents **memory bound**
- Region to the right represents **compute bound**
- Kernels shown in blue and green are close to peak memory bandwidth and compute throughput, representing bounded performance
- **Red kernels** are neither compute or memory bound and show potential for greater optimization
- Ideally kernel should shift upwards and rightwards



Roofline from NVIDIA A100 - NVHPC



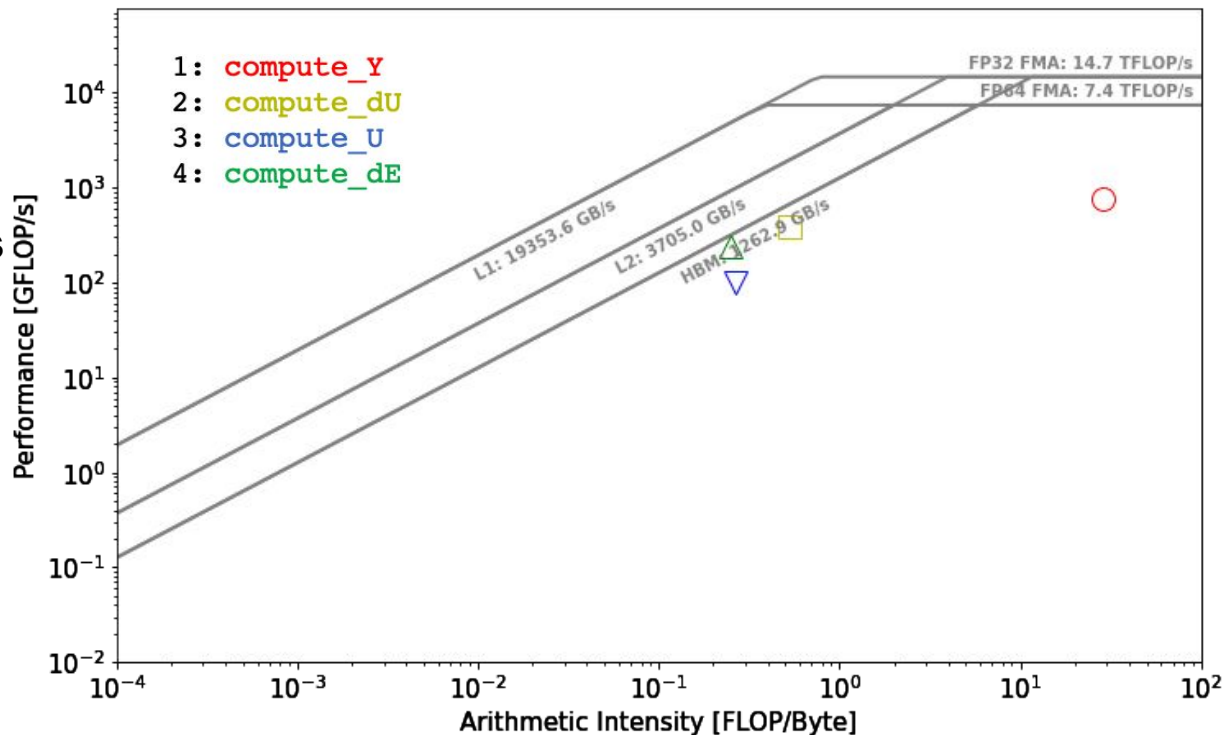
- Kernels are **memory bound** on A100
- Compute_y kernel close to machine balance line
- Other three kernels are bound by HBM bandwidth



Roofline from NVIDIA A100 - LLVM



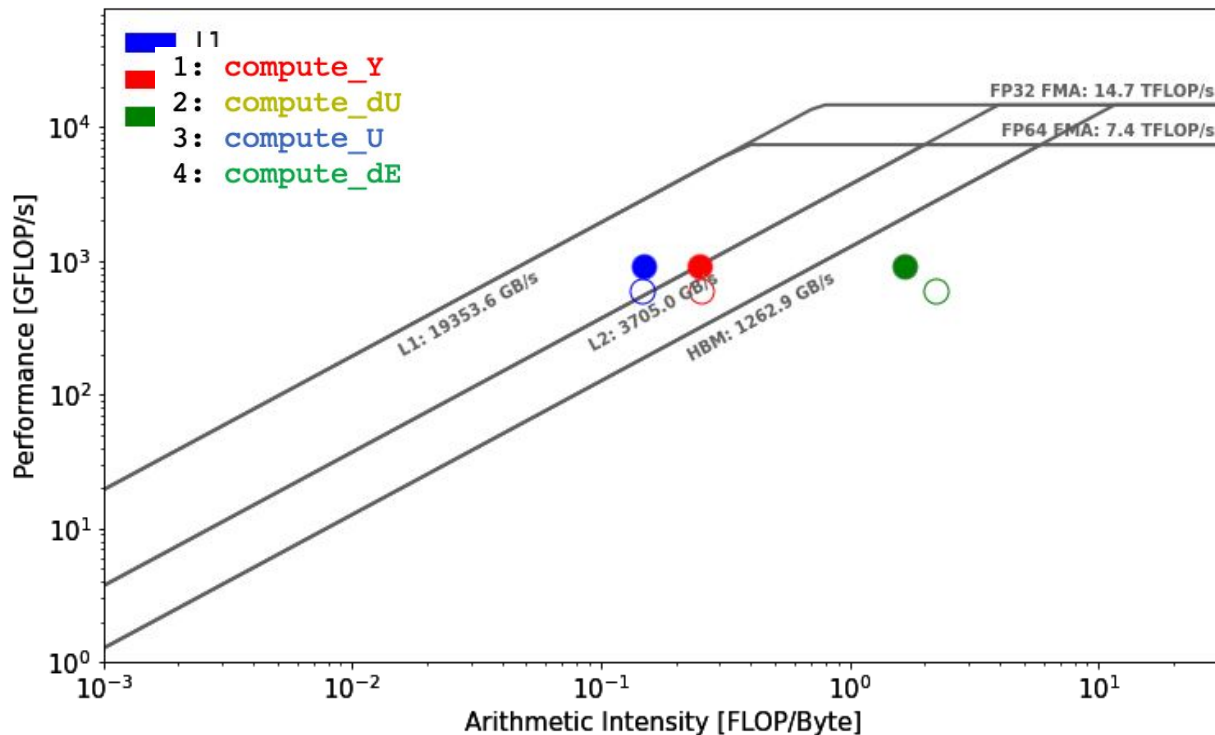
- Similar **memory bound** result for 3 out of 4 kernels as NVHPC
- Compute_Y however is **compute_bound**
- May be due to better **memory transfer protocol** but performance is lower



Hierarchical roofline from A100



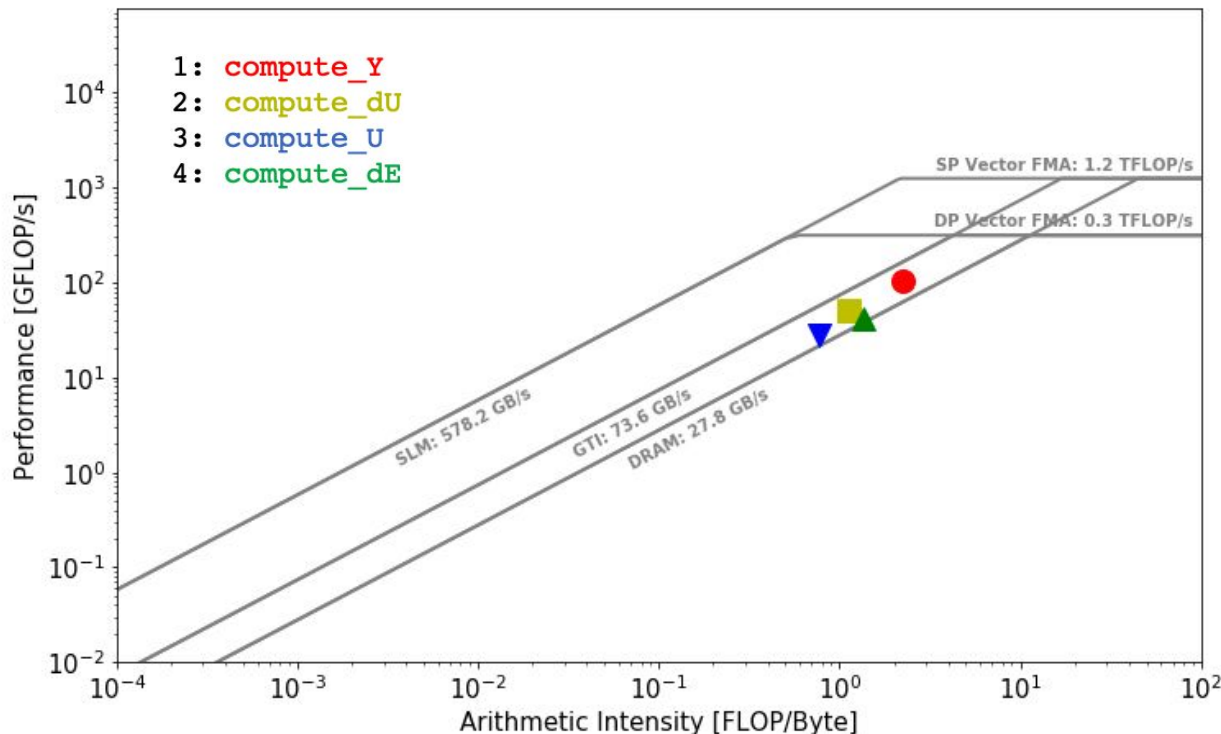
- **High reuse** between **DRAM** and **L2**
- **Poor** cache utilization between **L2** and **L1**
- Lower AI does not correspond to lower code runtime
- Code **AI** capped by **L2** cache



Roofline from Intel Gen9



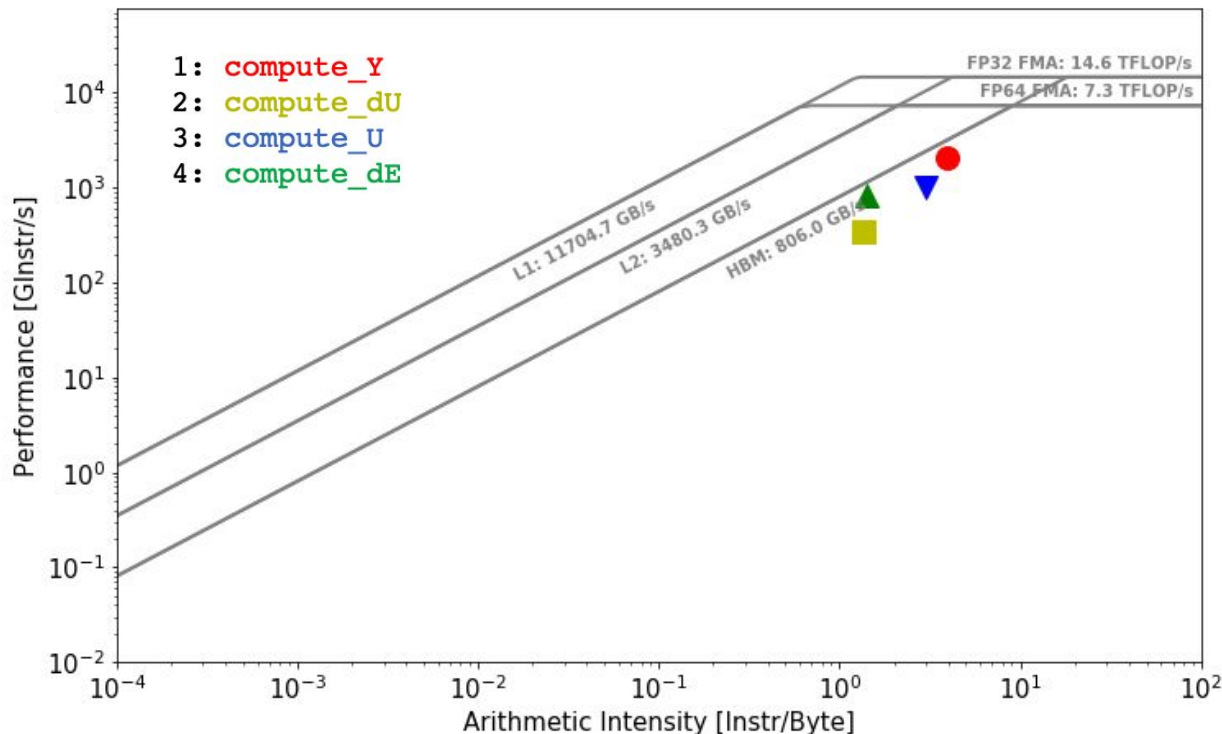
- Kernels are close to the DRAM bandwidth line, indicating **memory bound**
- Theoretically, roofline **cannot cross** bandwidth line
- Kernels crossing DRAM bandwidth indicates **eDRAM memory usage**
- Indicates performance **capped by memory bandwidth** and not compute capability



Roofline from AMD MI60



- Kernels again close to the DRAM bandwidth line, indicating **memory bound**
- Kernels are **closer to compute bound** regime
- All in the same order of magnitude



- **OpenMP offload directives can be used successfully to write portable code across all three GPUs**
- **Compiler maturity plays important role in code performance**
- **Profiles indicate expected difference in the performance between all three GPUs**
- **Code optimization play a key role in profiling and vice versa**
- **Kernels on all three GPUs are memory bound**
- **Differences in how compilers and GPU hardware address memory transfer leading to either drop and increase in AI**

TestSNAP code available at <https://github.com/FitSNAP/TestSNAP/tree/OpenMP4.5>

Acknowledgement



- **We would like to thank Dr. Aidan Thompson for providing us the initial TestSNAP code, which was then highly modified for this work.**
- **We would like to thank Drs. Danny Perez, Noah Reddell, and Nicholas Malaya for enabling us access and providing compute resources on the DOE's Cray Tulip machine.**
- **This research used resources of the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357.**



Joint Laboratory for System Evaluation (JLSE)

<https://jlse.anl.gov>



We gratefully acknowledge the computing resources provided and operated by the Joint Laboratory for System Evaluation (JLSE) at Argonne National Laboratory.



NERSC

Thank You



U.S. DEPARTMENT OF
ENERGY

Office of
Science





Backup slides



U.S. DEPARTMENT OF
ENERGY

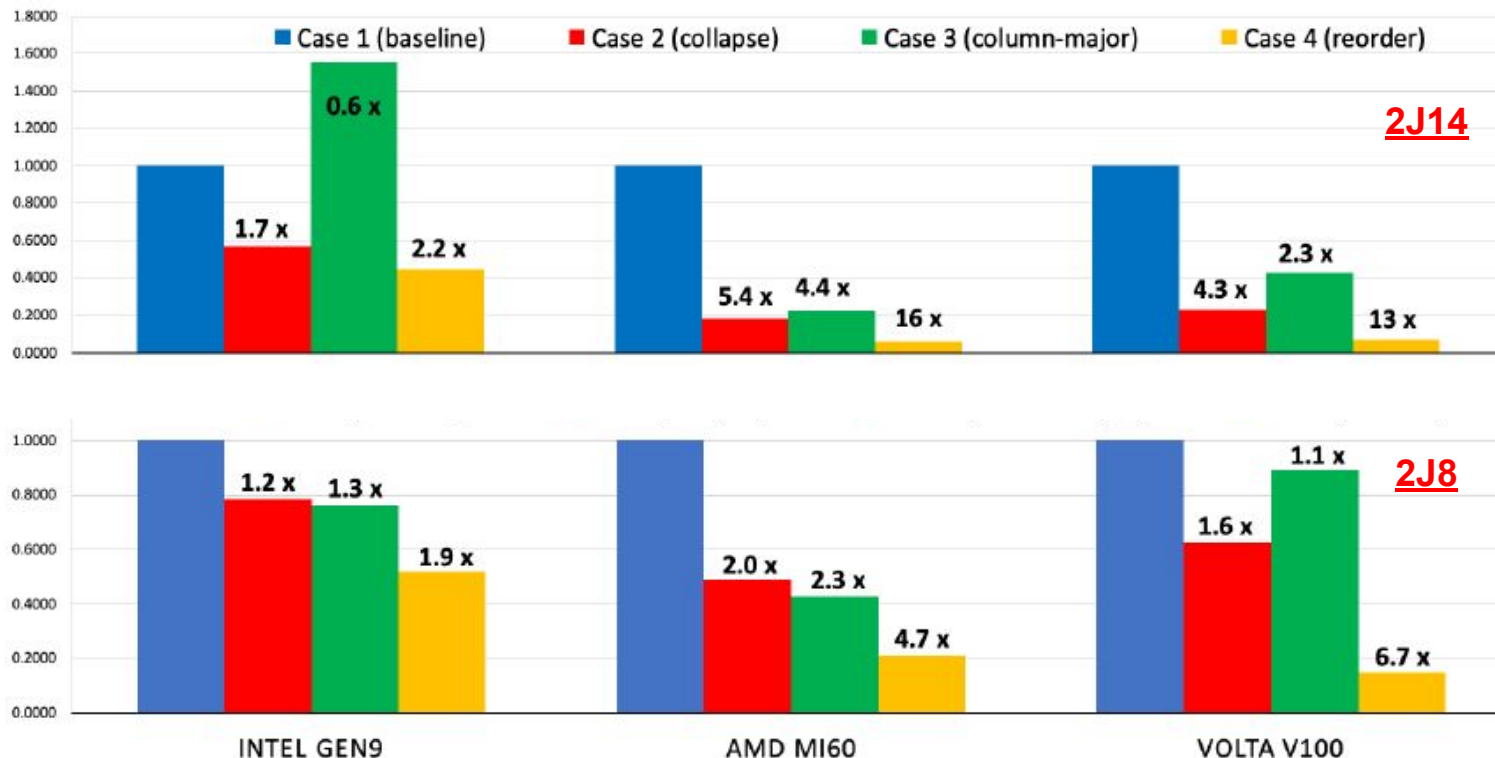
Office of
Science



Speed-ups due to optimizations



- All results normalized to baseline
- Column major data access: atom loop as fastest moving index causing performance degradation



Variadic versus non-variadic arrays



```
define ARRAY2D ArrayMD<int, 2>
define ARRAY3D ArrayMD<int, 3>
RRAY2D y(N, N);
RRAY2D x(N, N);
RRAY3D m(N, N, N);
```

```
1 #define ARRAY2D Array2D<int>
2 #define ARRAY3D Array3D<int>
3 ARRAY2D y(N, N);
4 ARRAY2D x(N, N);
5 ARRAY3D m(N, N, N);
```

- A single class to create multidimensional arrays for every dimension and data type
- Class is templated over the number of dimensions
- Variadic template pack expansion is used to calculate the offset in each dimension
- An array class is templated but only per data type
- Requires duplication of templated class for each multi-dimensional array class

Profiling data - NVIDIA V100



Metric	Baseline	With omp for	With omp simd
Kernel time (s)	10.70	1.82	1.81
Total time (s)	11.85	2.99	2.96

Metric	Variadic	Non-variadic
Kernel time (s)	1.81	7.20
Total time (s)	2.96	8.75

grid_size: 1000 x 1 x 1
block_size: 128 x 1 x 1

Profiling data - NVIDIA V100



Metric	Baseline	With omp for	With omp simd
Kernel time (s)	11.00	2.17	2.16
Total time (s)	13.10	4.15	4.16

Metric	Variadic	Non-variadic
Kernel time (s)	2.16	3.08
Total time (s)	4.16	5.81

Profiling data - Intel Gen9



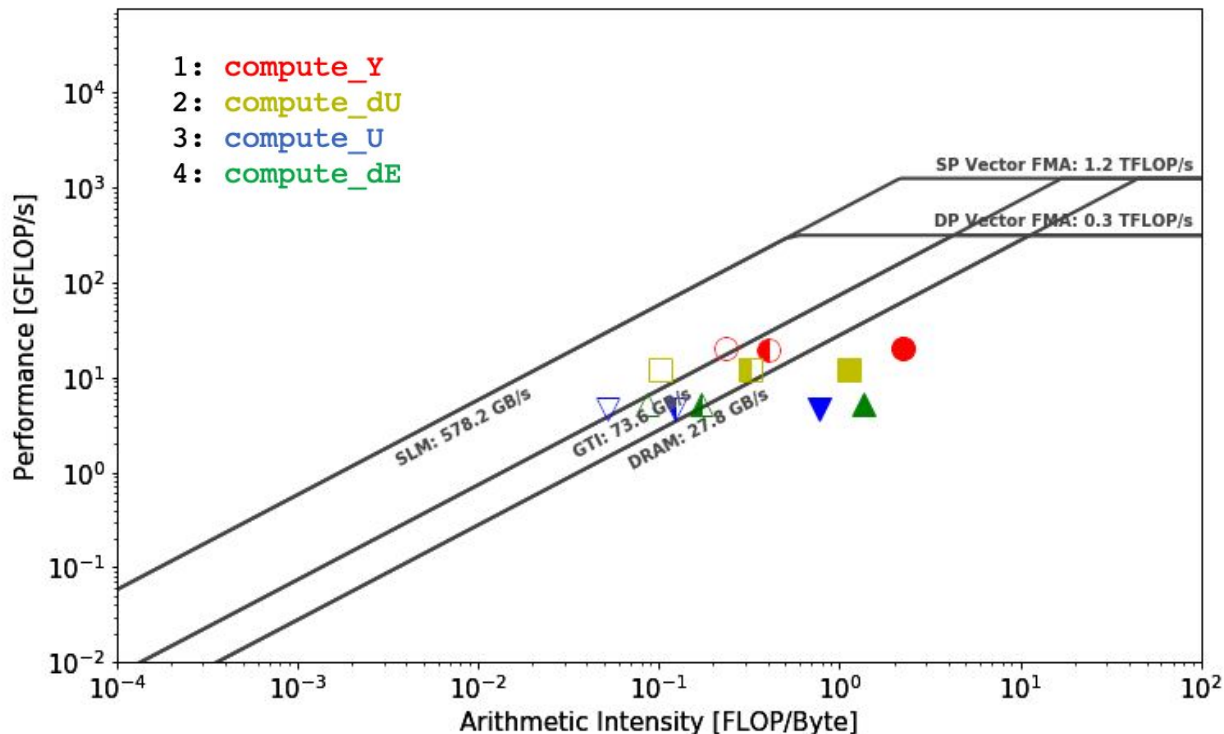
Metric	Baseline	With omp for	With omp simd
Kernel time (s)	275.43	20.36	20.41
Total time (s)	275.94	20.87	20.91

Metric	Variadic	Non-variadic
Kernel time (s)	20.41	21.36
Total time (s)	20.91	22.37

Hierarchical roofline from Gen9



- Rooflines measured for at **DRAM, GTI, and L3 (SLM)** cache level
- Reduction in AI due **lower FLOP count** for each unit of memory moved **across that memory level**
- **Larger difference** between the kernel rooflines indicate good data reuse, i.e., **good use of cache hierarchy**



● DRAM
○ L3