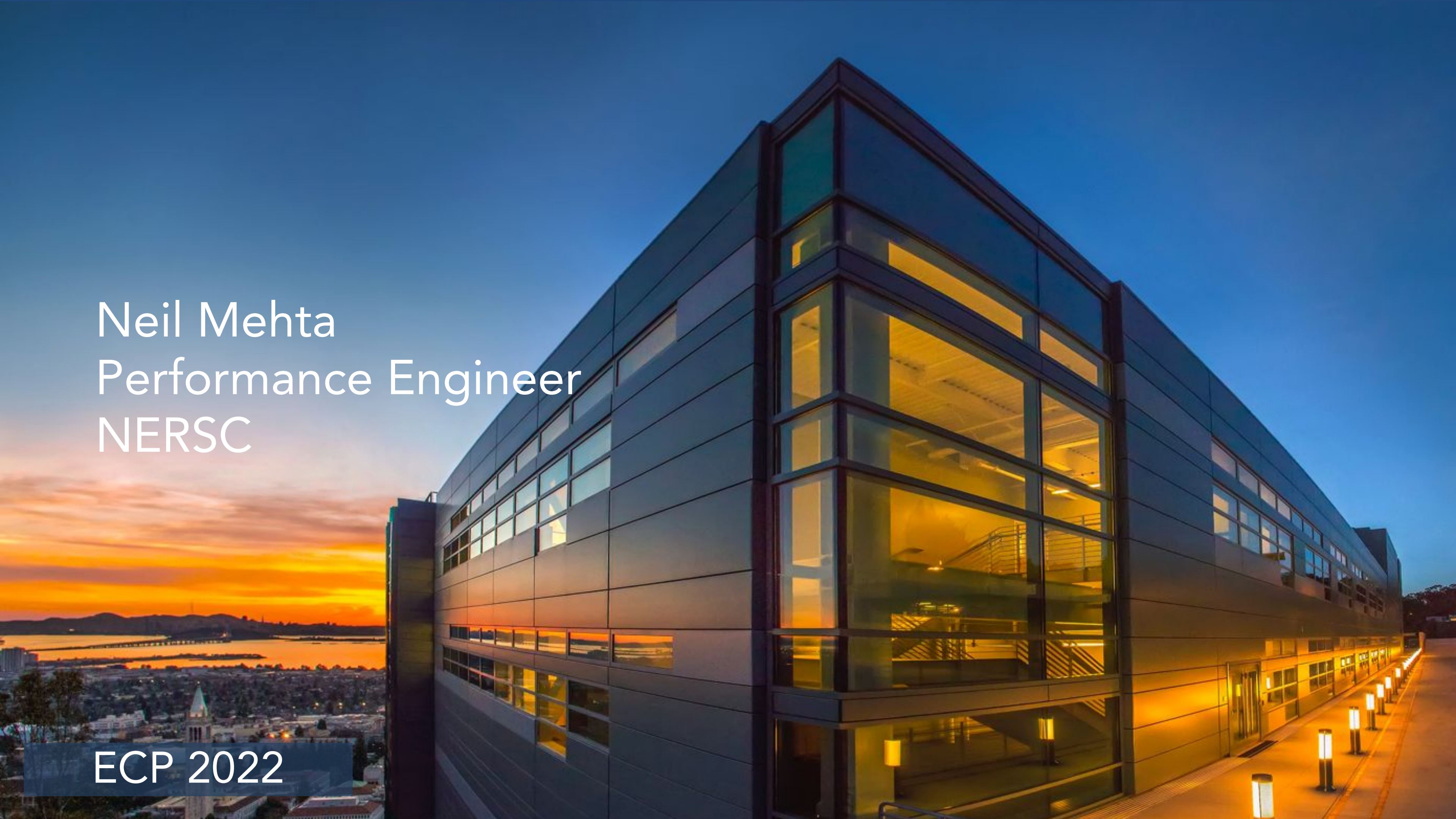
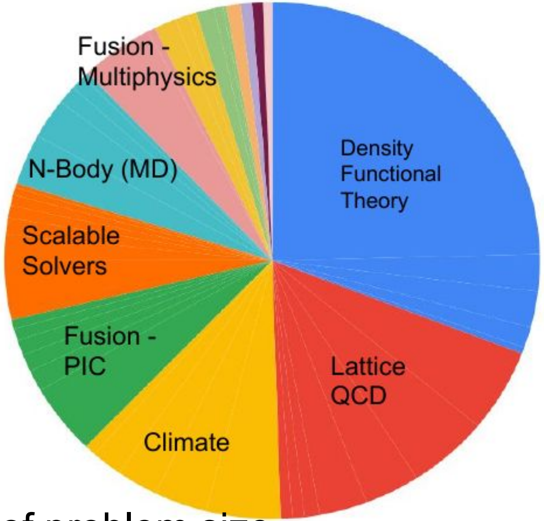


Neil Mehta
Performance Engineer
NERSC

ECP 2022

# Motivation for targeting MD code

- **LAMMPS** developed under the auspices of **DOE** and multi-lab collaboration
- Beneficiary of **Exa-scale Computing Project (ECP).** Under ECP umbrella project **EXAALT**



**For Users:**

- **Efficiency matters,**
  less resources required
- Better optimization of application on **compute resources (GPU vs CPU)**

**For Developers:**

- **Performance portability** independent of problem size
- Better **compiler** design **feedback**

# Methodology to generate rooflines

## Obtaining roofline ceilings

- Using empirical values from Empirical Roofline Toolkit (https://bitbucket.org/berkeleylab/cs-roofline-toolkit/)

## Obtaining kernel specific roofline data

- Using Nsight Compute
- Using custom scripts (https://gitlab.com/NERSC/roofline-on-nvidia-gpus/)

## Obtaining application data

- Measure three quantities: time, FLOPs, and data movement (bytes)
- Calculate:

$$\textbf{Arithmetic Intensity (FLOPs/byte)} = \frac{\textbf{FLOPs}}{\textbf{data movement}}$$

$$\textbf{Performance (GFLOP/s)} = \frac{\textbf{FLOPs}}{\textbf{time}}$$

*https://www.nersc.gov/assets/Uploads/RooflineHack-2020-mechanism-v2.pdf

# Collecting roofline metrics using NCU (1/2)

**Time**

- sm__cycles_elapsed.avg
- sm__cycles_elapsed.avg.per_second

**Accumulate code runtime**

**Memory**

- dram__bytes.sum
- lts__t_bytes.sum
- l1tex__t_bytes.sum

**Accumulate L1, L2, and DRAM memory transfers**

# Collecting roofline metrics using NCU (2/2)

## Compute measurements

### Double precision

- sm__sass_thread_inst_executed_op_dadd_pred_on.sum
- sm__sass_thread_inst_executed_op_dmul_pred_on.sum
- sm__sass_thread_inst_executed_op_dfma_pred_on.sum

### Single precision

- sm__sass_thread_inst_executed_op_fadd_pred_on.sum
- sm__sass_thread_inst_executed_op_fmul_pred_on.sum
- sm__sass_thread_inst_executed_op_ffma_pred_on.sum

### Half precision

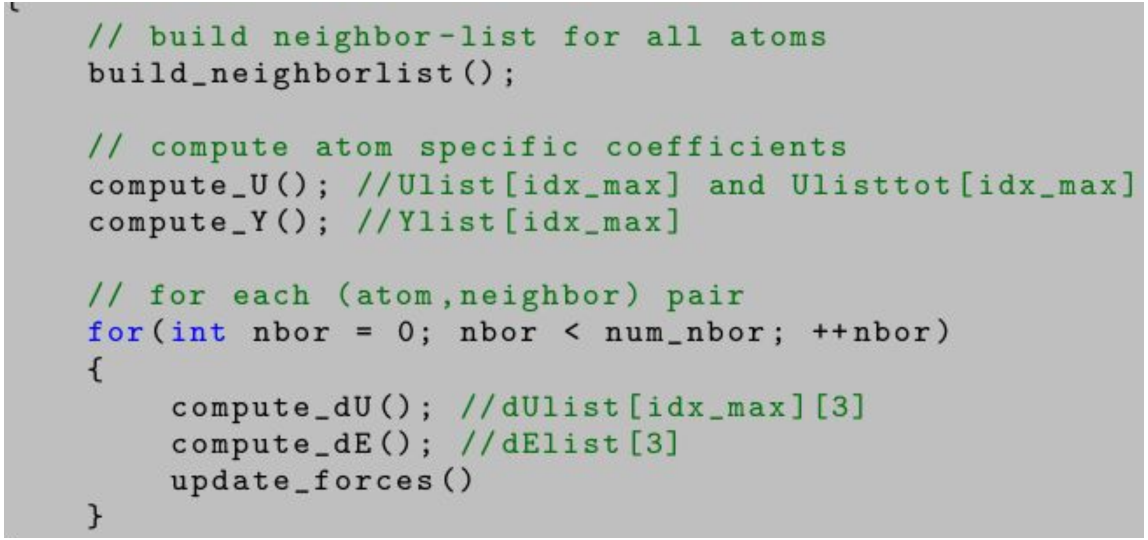
- sm__sass_thread_inst_executed_op_hadd_pred_on.sum
- sm__sass_thread_inst_executed_op_hmul_pred_on.sum
- sm__sass_thread_inst_executed_op_hfma_pred_on.sum

**Accumulate add, mul, and fused add mul instructions**

# Introduction to TestSNAP

- **TestSNAP** proxy app **mimics computational load** of **LAMMPS SNAP**
- **Four** dominant **kernels**
- Number of **atoms: 2000**
- Number of **steps: 100**
- **Profiling on NVIDIA A100**

```
$ ssh -Y username@perlmutter-p1.nersc.gov
$ cd $SCRATCH
$ git clone https://github.com/FitSNAP/TestSNAP.git
$ git checkout OpenMP4.5
```

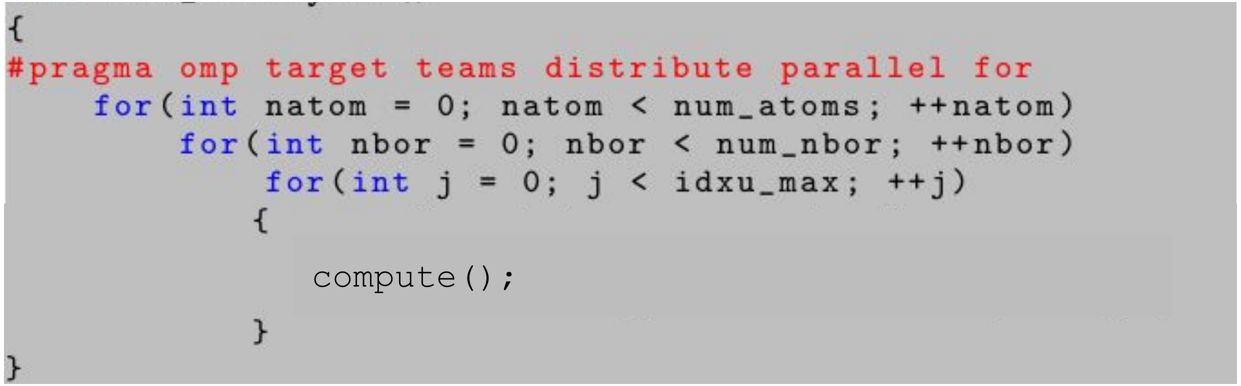


```
    // build neighbor-list for all atoms
    build_neighborlist();

    // 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 optimization for OpenMP (1/4)

**Arrays created using classes that include pointer to contiguous block of memory**

**Case 1: baseline**



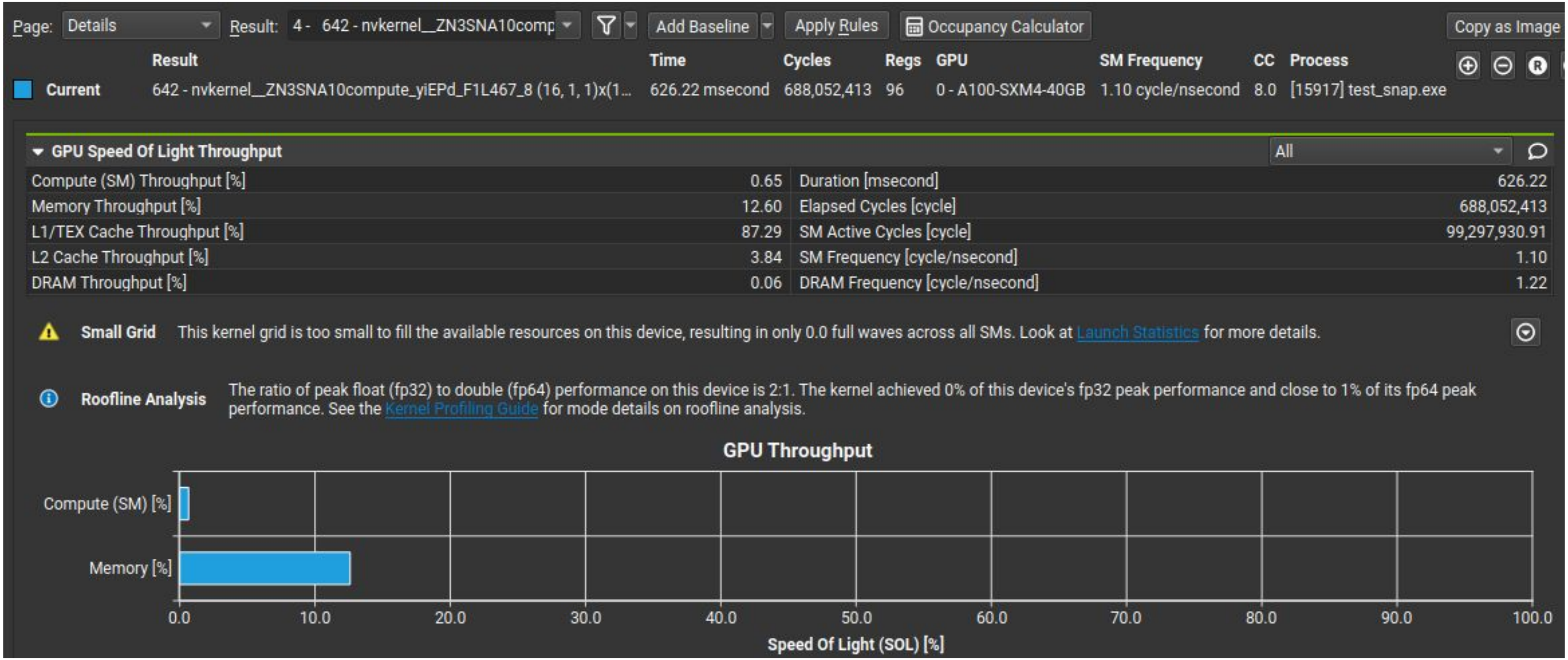
```
{
#pragma omp target teams distribute parallel for
    for(int natom = 0; natom < num_atoms; ++natom)
        for(int nbor = 0; nbor < num_nbor; ++nbor)
            for(int j = 0; j < idxu_max; ++j)
            {

              compute();
            }
}
```

**<u>Grind times:</u>**
**(ms/atm-step)**

**nvc++ : 0.321**

```
$ salloc -C gpu -t 240 -c 10 -G 1 -q regular -A <project>
$ module load nvhpc/22.2 (module load cuda/11.3.0)
$ ncu -o profile_snap --set full ./testsnap.exe -ns 100
$ ncu-ui profile_snap.ncu-rep
```

# Kernel optimization for OpenMP (1/4)



| | Result | Time | Cycles | Regs | GPU | SM Frequency | CC | Process |
|---|---|---|---|---|---|---|---|---|
| 🟦 **Current** | 642 - nvkernel__ZN3SNA10compute_yiEPd_F1L467_8 (16, 1, 1)x(1... | 626.22 msecond | 688,052,413 | 96 | 0 - A100-SXM4-40GB | 1.10 cycle/nsecond | 8.0 | [15917] test_snap.exe |

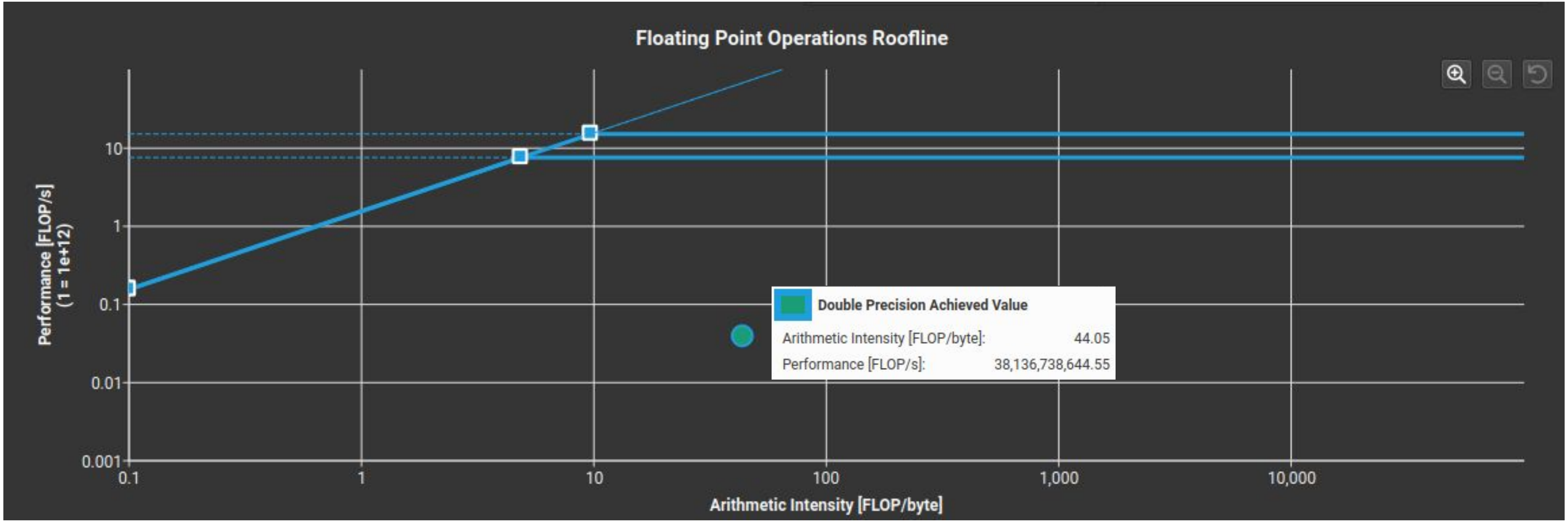
**GPU Speed Of Light Throughput**

| | | | |
|---|---|---|---|
| Compute (SM) Throughput [%] | 0.65 | Duration [msecond] | 626.22 |
| Memory Throughput [%] | 12.60 | Elapsed Cycles [cycle] | 688,052,413 |
| L1/TEX Cache Throughput [%] | 87.29 | SM Active Cycles [cycle] | 99,297,930.91 |
| L2 Cache Throughput [%] | 3.84 | SM Frequency [cycle/nsecond] | 1.10 |
| DRAM Throughput [%] | 0.06 | DRAM Frequency [cycle/nsecond] | 1.22 |

⚠ **Small Grid** This kernel grid is too small to fill the available resources on this device, resulting in only 0.0 full waves across all SMs. Look at Launch Statistics for more details.

ⓘ **Roofline Analysis** The ratio of peak float (fp32) to double (fp64) performance on this device is 2:1. The kernel achieved 0% of this device's fp32 peak performance and close to 1% of its fp64 peak performance. See the Kernel Profiling Guide for mode details on roofline analysis.

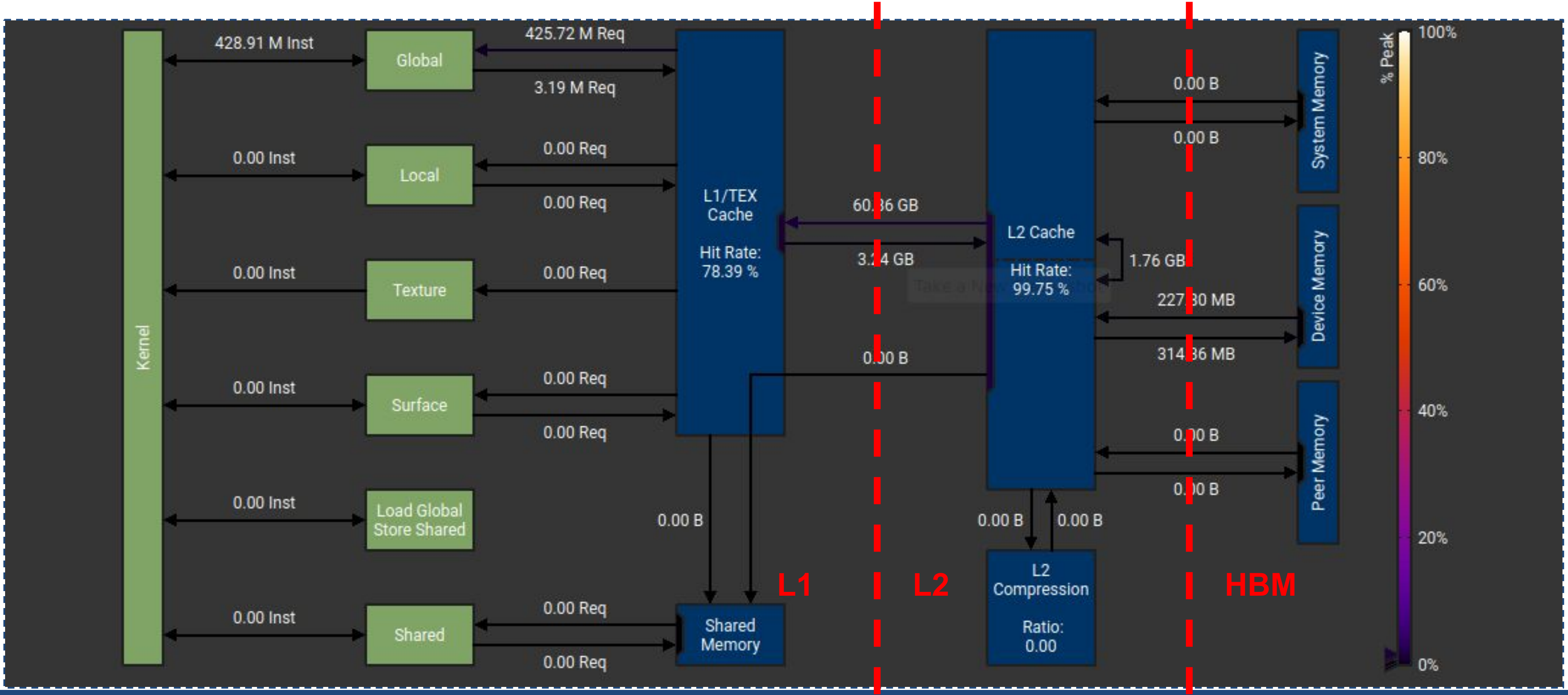
# Kernel optimization for OpenMP (1/4)



**Double Precision Achieved Value**

| | |
|---|---|
| Arithmetic Intensity [FLOP/byte]: | 44.05 |
| Performance [FLOP/s]: | 38,136,738,644.55 |

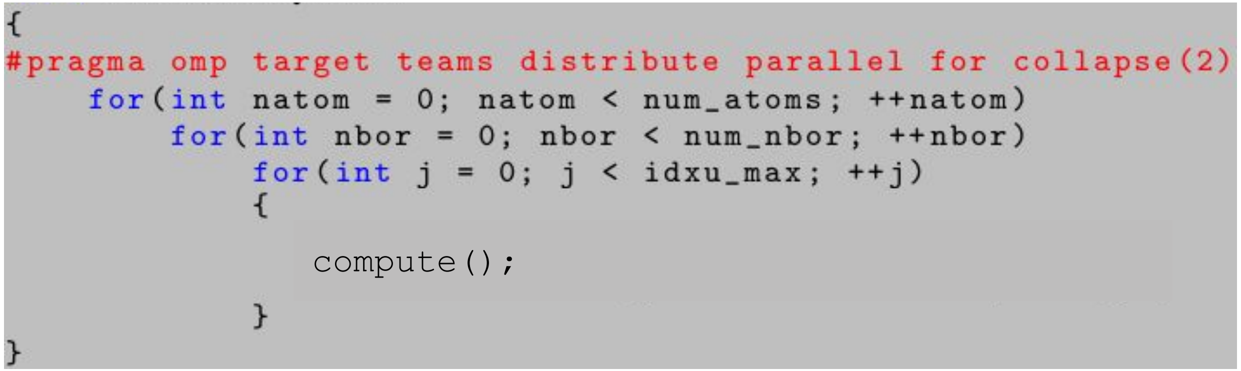
# Kernel optimization for OpenMP (1/4)



# Kernel optimization for OpenMP (2/4)

**Exploit the ability to collapse nested *for* loops**

**Case 2: collapse**



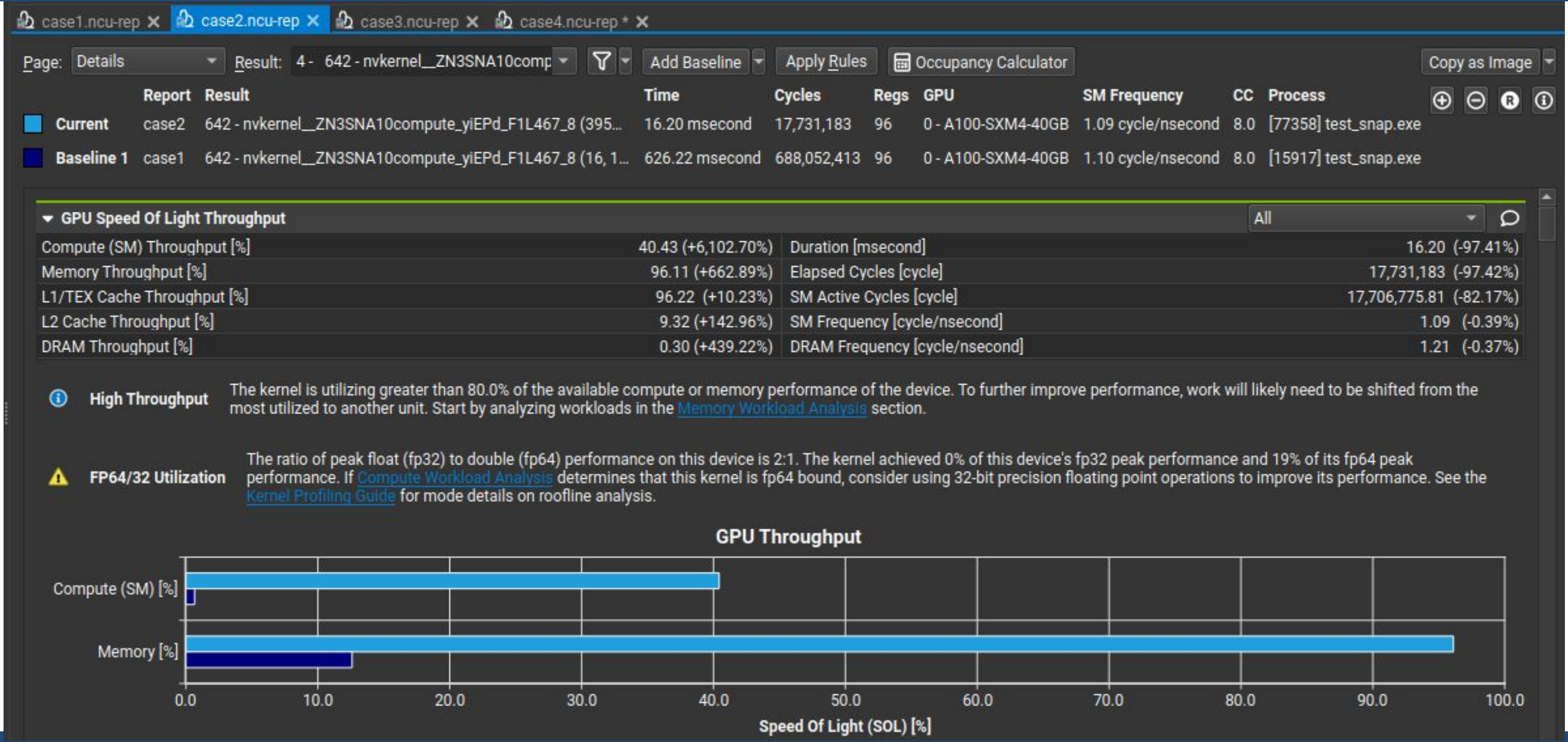
```
{
#pragma omp target teams distribute parallel for collapse(2)
    for(int natom = 0; natom < num_atoms; ++natom)
        for(int nbor = 0; nbor < num_nbor; ++nbor)
            for(int j = 0; j < idxu_max; ++j)
            {
                compute();
            }
}
```

**Grind times:**

**(ms/atm-step)**

**nvc++ : 0.0342 (9.5x)**

# Kernel optimization for OpenMP (2/4)



Page: Details
Result: 4 - 642 - nvkernel__ZN3SNA10comp

| | Report | Result | Time | Cycles | Regs | GPU | SM Frequency | CC | Process |
|---|---|---|---|---|---|---|---|---|---|
| **Current** | case2 | 642 - nvkernel__ZN3SNA10compute_yiEPd_F1L467_8 (395... | 16.20 msecond | 17,731,183 | 96 | 0 - A100-SXM4-40GB | 1.09 cycle/nsecond | 8.0 | [77358] test_snap.exe |
| **Baseline 1** | case1 | 642 - nvkernel__ZN3SNA10compute_yiEPd_F1L467_8 (16, 1... | 626.22 msecond | 688,052,413 | 96 | 0 - A100-SXM4-40GB | 1.10 cycle/nsecond | 8.0 | [15917] test_snap.exe |

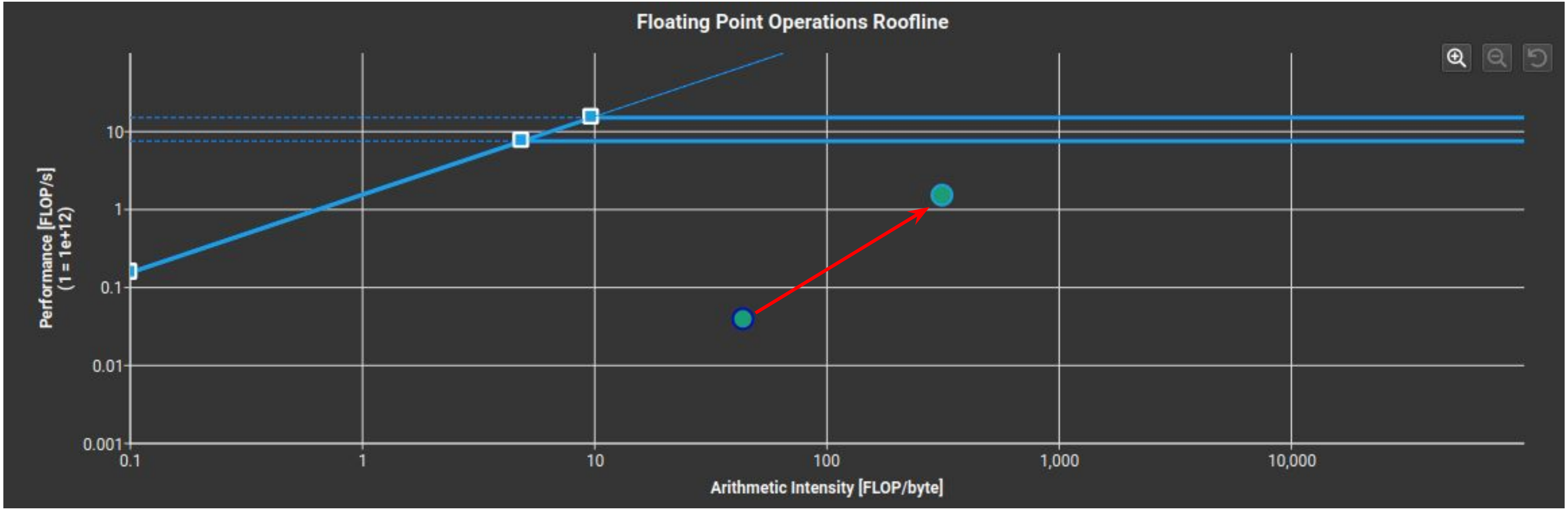
**GPU Speed Of Light Throughput**

| | | | |
|---|---|---|---|
| Compute (SM) Throughput [%] | 40.43 (+6,102.70%) | Duration [msecond] | 16.20 (-97.41%) |
| Memory Throughput [%] | 96.11 (+662.89%) | Elapsed Cycles [cycle] | 17,731,183 (-97.42%) |
| L1/TEX Cache Throughput [%] | 96.22 (+10.23%) | SM Active Cycles [cycle] | 17,706,775.81 (-82.17%) |
| L2 Cache Throughput [%] | 9.32 (+142.96%) | SM Frequency [cycle/nsecond] | 1.09 (-0.39%) |
| DRAM Throughput [%] | 0.30 (+439.22%) | DRAM Frequency [cycle/nsecond] | 1.21 (-0.37%) |

**High Throughput** The kernel is utilizing greater than 80.0% of the available compute or memory performance of the device. To further improve performance, work will likely need to be shifted from the most utilized to another unit. Start by analyzing workloads in the Memory Workload Analysis section.

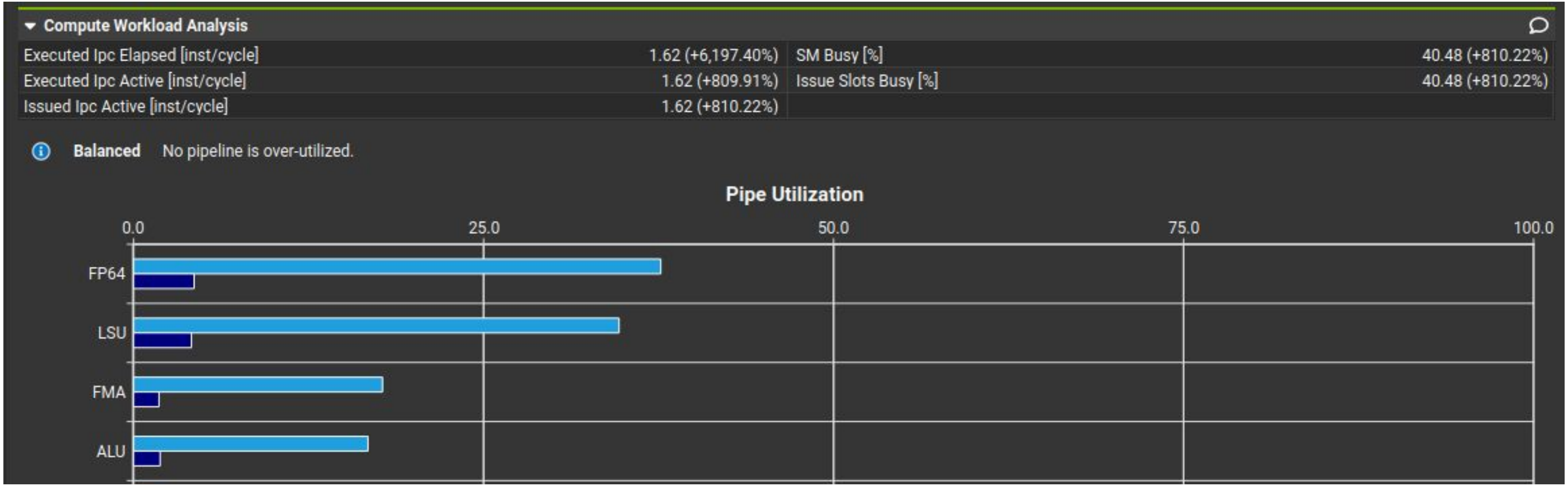
**FP64/32 Utilization** The ratio of peak float (fp32) to double (fp64) performance on this device is 2:1. The kernel achieved 0% of this device's fp32 peak performance and 19% of its fp64 peak performance. If Compute Workload Analysis determines that this kernel is fp64 bound, consider using 32-bit precision floating point operations to improve its performance. See the Kernel Profiling Guide for mode details on roofline analysis.

# Kernel optimization for OpenMP (2/4)



- **Improvement in AI and Performance due to atom and neighbor loop fusing**

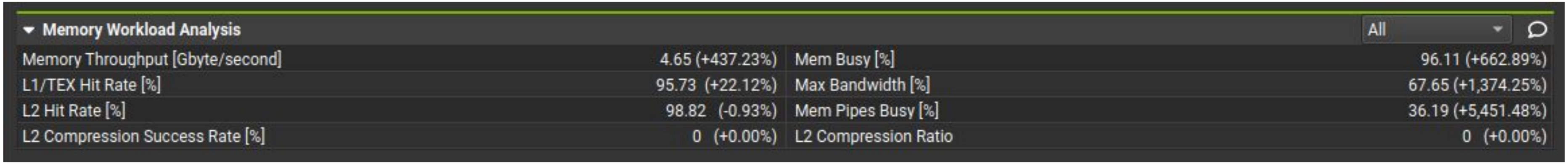
# Kernel optimization for OpenMP (2/4)



**Compute Workload Analysis**

| | | | |
|---|---|---|---|
| Executed Ipc Elapsed [inst/cycle] | 1.62 (+6,197.40%) | SM Busy [%] | 40.48 (+810.22%) |
| Executed Ipc Active [inst/cycle] | 1.62 (+809.91%) | Issue Slots Busy [%] | 40.48 (+810.22%) |
| Issued Ipc Active [inst/cycle] | 1.62 (+810.22%) | | |

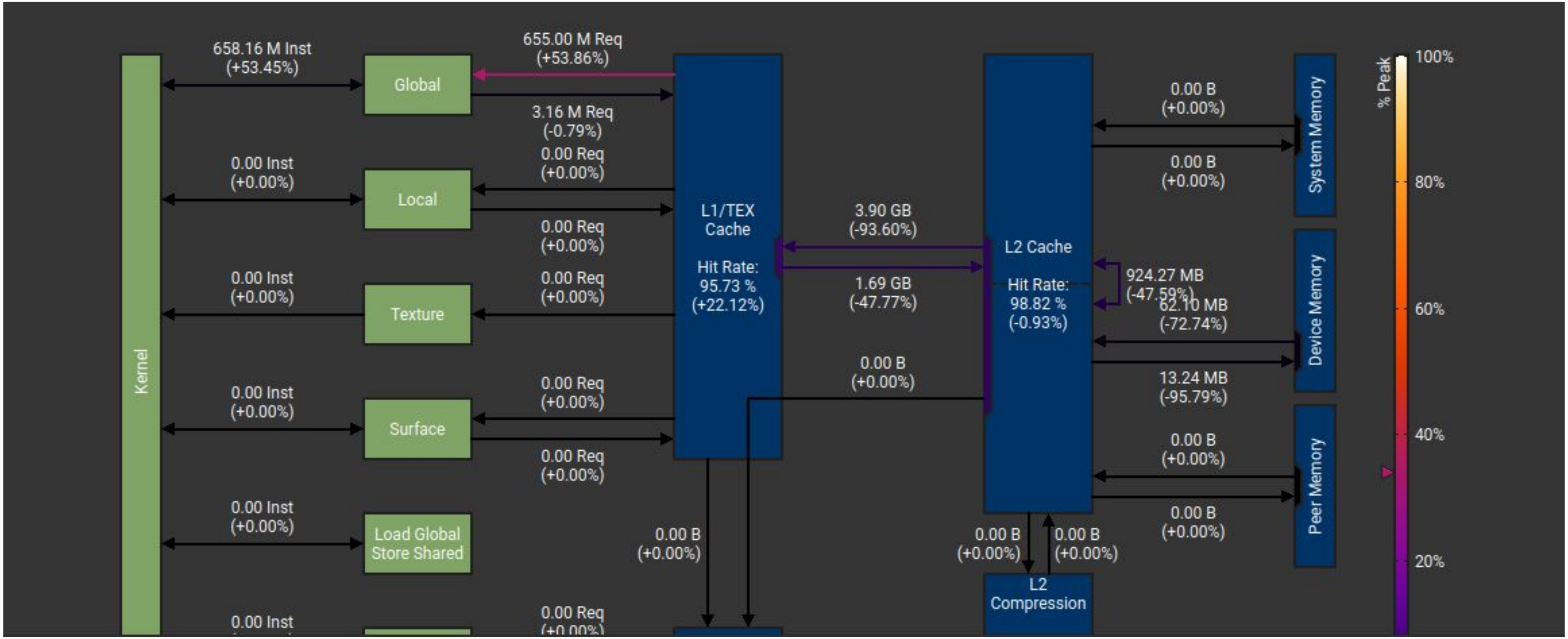
**Balanced** No pipeline is over-utilized.



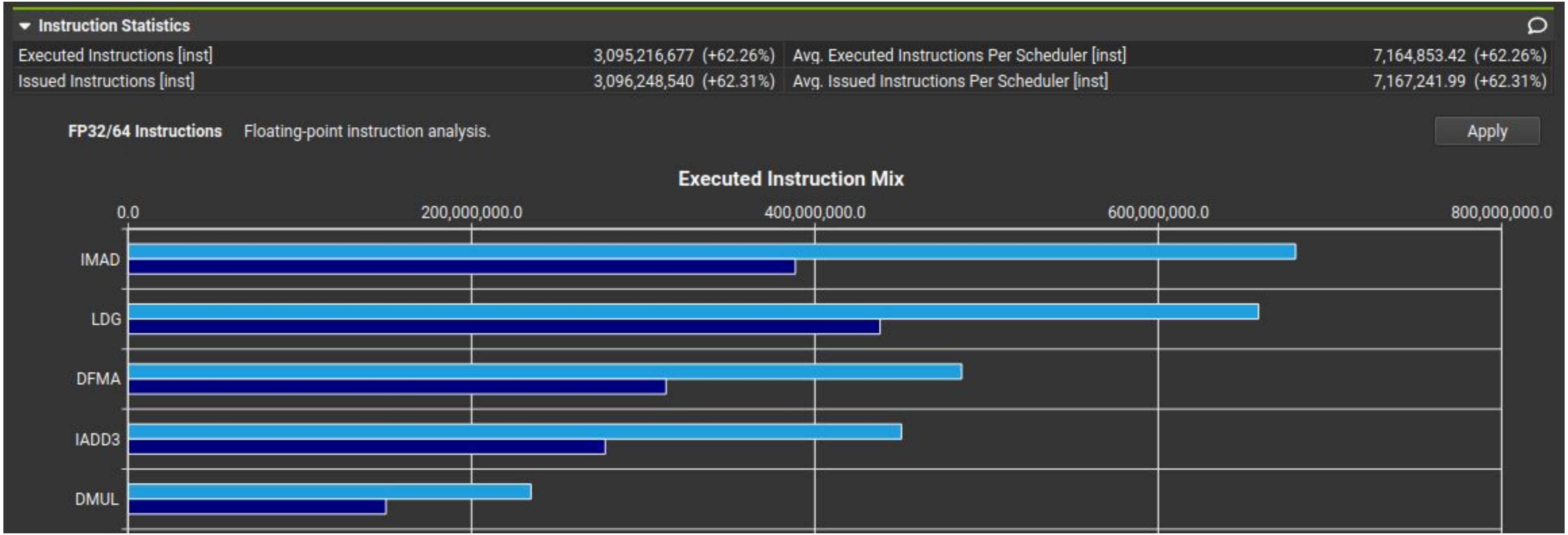
**Memory Workload Analysis**

| | | | |
|---|---|---|---|
| Memory Throughput [Gbyte/second] | 4.65 (+437.23%) | Mem Busy [%] | 96.11 (+662.89%) |
| L1/TEX Hit Rate [%] | 95.73 (+22.12%) | Max Bandwidth [%] | 67.65 (+1,374.25%) |
| L2 Hit Rate [%] | 98.82 (-0.93%) | Mem Pipes Busy [%] | 36.19 (+5,451.48%) |
| L2 Compression Success Rate [%] | 0 (+0.00%) | L2 Compression Ratio | 0 (+0.00%) |

# Kernel optimization for OpenMP (2/4)



# Kernel optimization for OpenMP (2/4)



**Instruction Statistics**

| | | | |
|---|---|---|---|
| Executed Instructions [inst] | 3,095,216,677 (+62.26%) | Avg. Executed Instructions Per Scheduler [inst] | 7,164,853.42 (+62.26%) |
| Issued Instructions [inst] | 3,096,248,540 (+62.31%) | Avg. Issued Instructions Per Scheduler [inst] | 7,167,241.99 (+62.31%) |

**FP32/64 Instructions** Floating-point instruction analysis.

# Kernel optimization for OpenMP (3/4)

**Column major data access: atom loop as fastest moving index causes performance degradation**

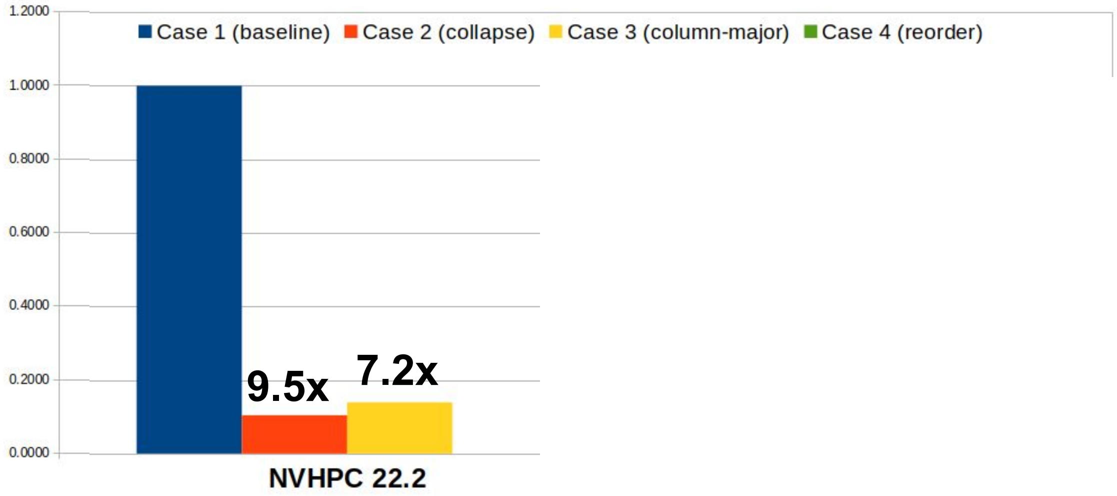
**Case 3: column major**

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

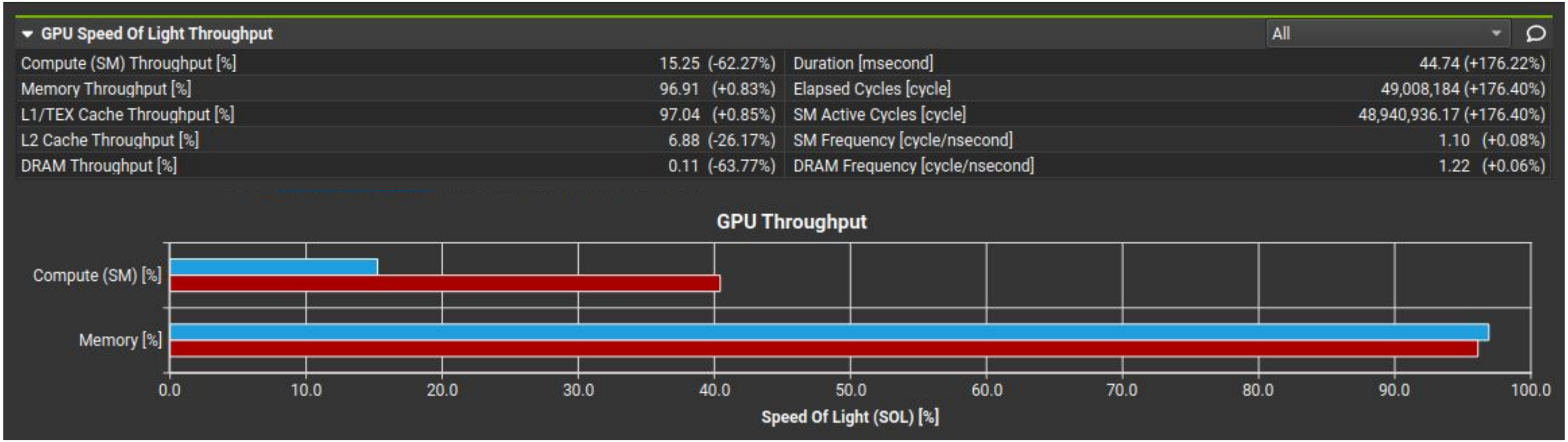
**Grind times:**

**(ms/atm-step)**

**nvc++ : 0.0457 (7.2x)**

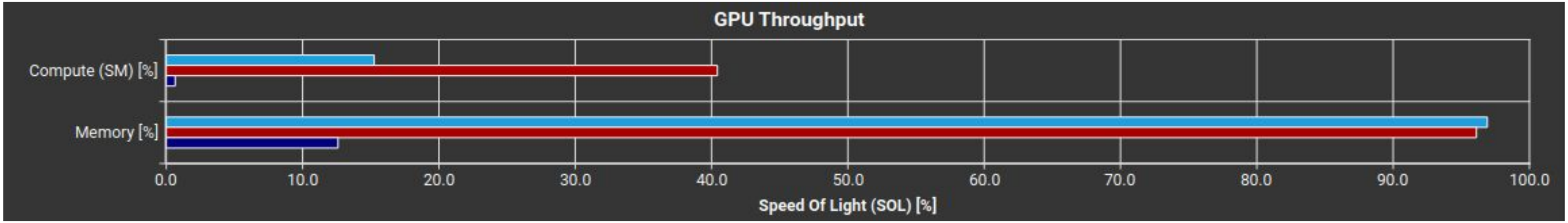


# Kernel optimization for OpenMP (3/4)

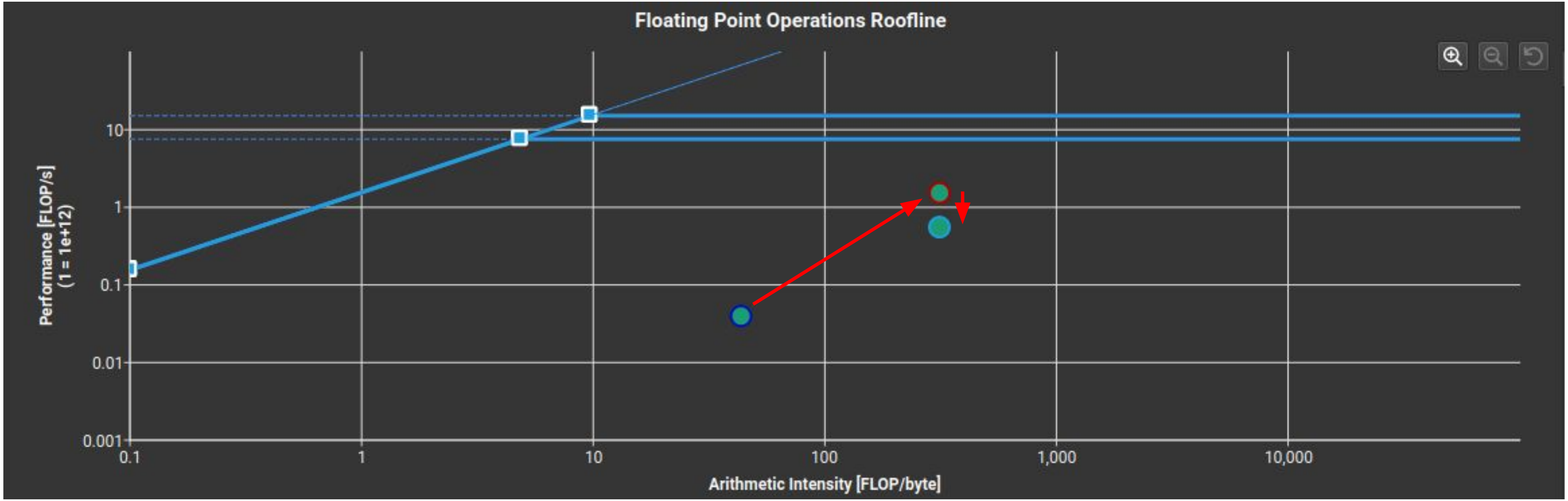


**GPU Speed Of Light Throughput**

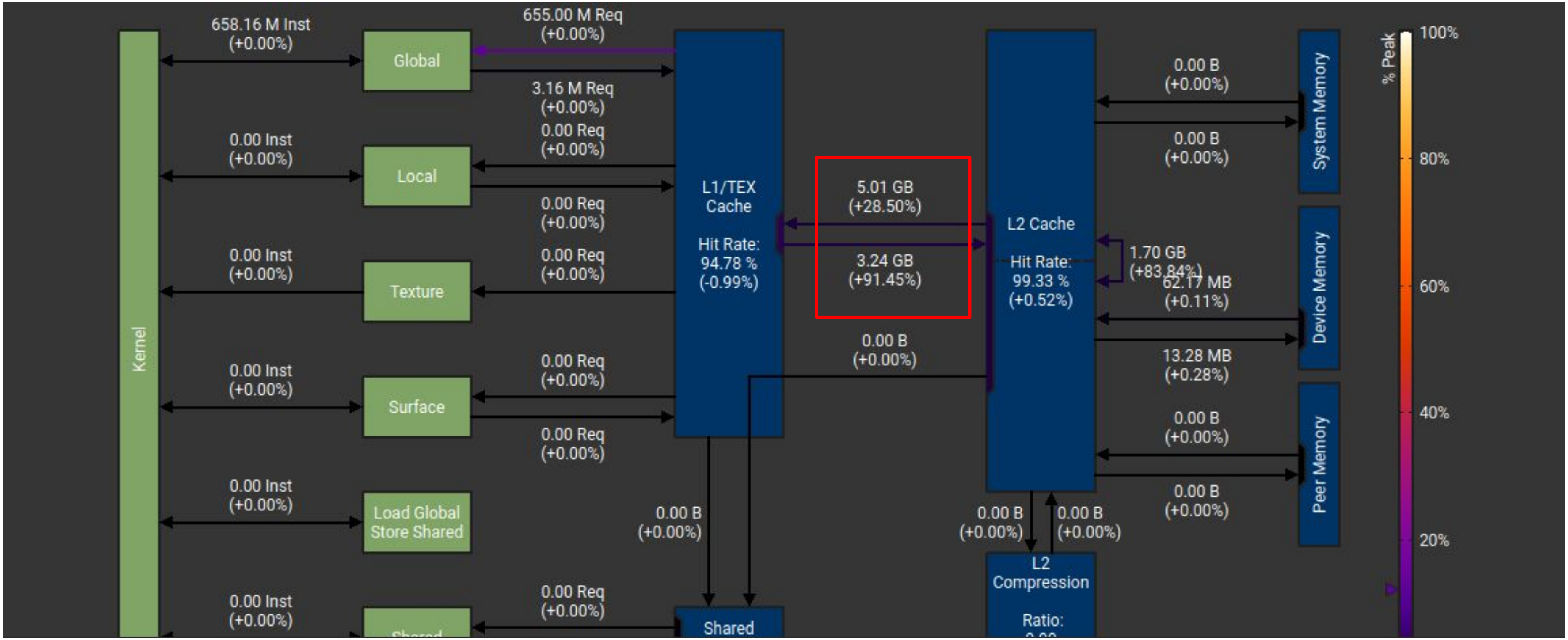
| | | | |
|---|---|---|---|
| Compute (SM) Throughput [%] | 15.25 (-62.27%) | Duration [msecond] | 44.74 (+176.22%) |
| Memory Throughput [%] | 96.91 (+0.83%) | Elapsed Cycles [cycle] | 49,008,184 (+176.40%) |
| L1/TEX Cache Throughput [%] | 97.04 (+0.85%) | SM Active Cycles [cycle] | 48,940,936.17 (+176.40%) |
| L2 Cache Throughput [%] | 6.88 (-26.17%) | SM Frequency [cycle/nsecond] | 1.10 (+0.08%) |
| DRAM Throughput [%] | 0.11 (-63.77%) | DRAM Frequency [cycle/nsecond] | 1.22 (+0.06%) |



# Kernel optimization for OpenMP (3/4)



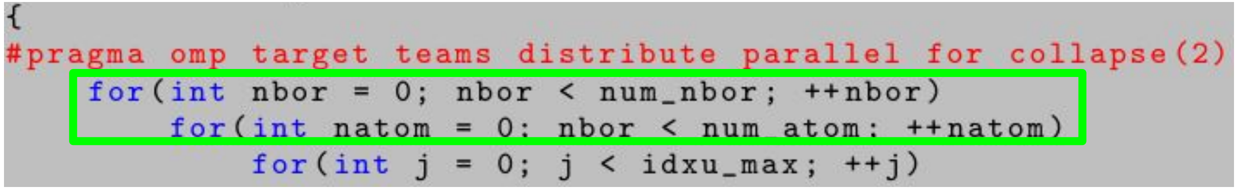
# Kernel optimization for OpenMP (3/4)



# Kernel optimization for OpenMP (4/4)

**Make atom loop (fastest moving index) as inner most loop**

**Case 4: reorder loop**

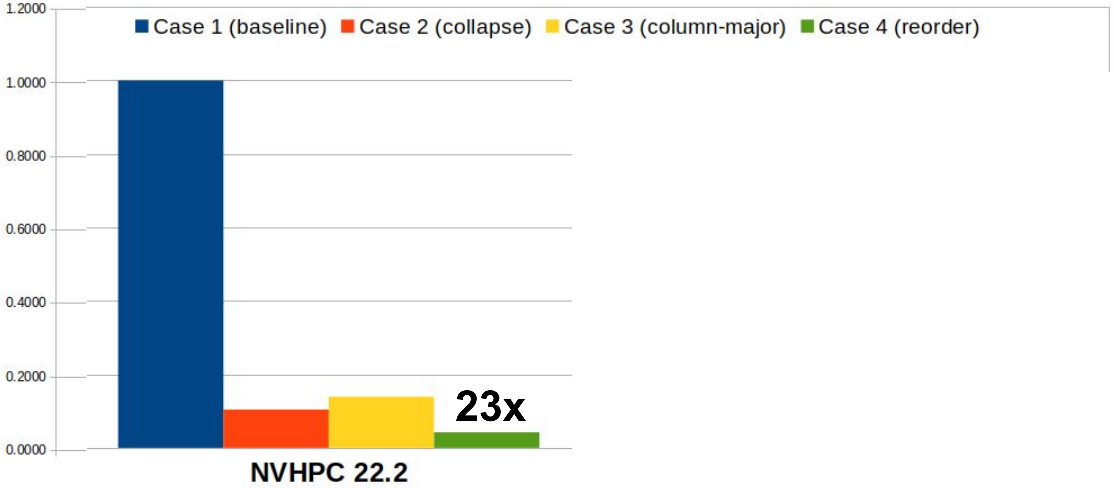


```
{
#pragma omp target teams distribute parallel for collapse(2)
    for(int nbor = 0; nbor < num_nbor; ++nbor)
        for(int natom = 0; nbor < num_atom; ++natom)
            for(int j = 0; j < idxu_max; ++j)
```

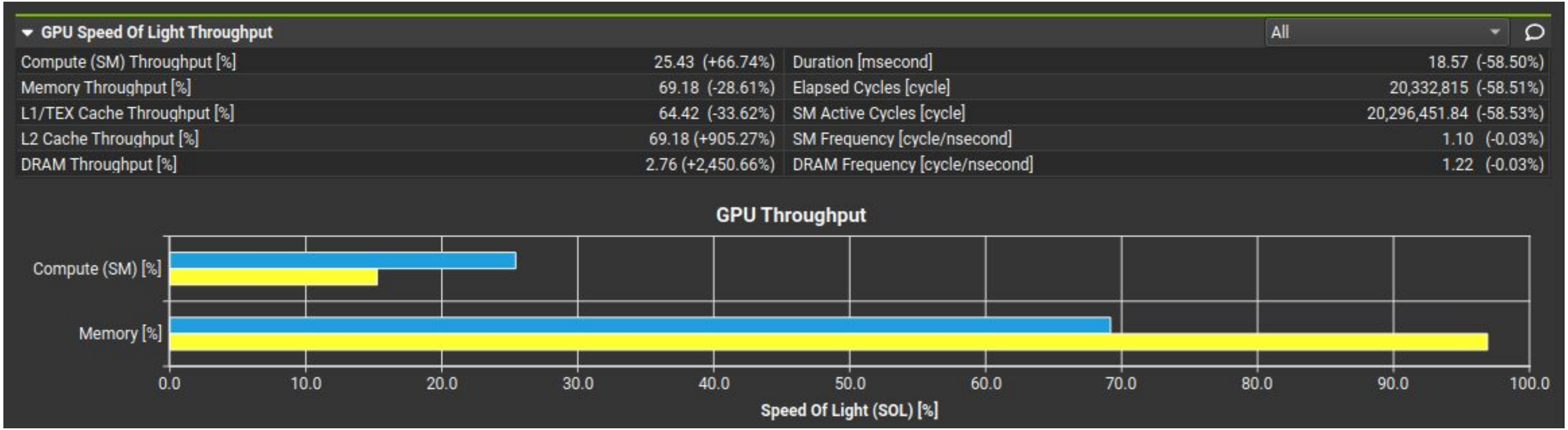
**Grind times:**

**(ms/atm-step)**

**nvc++ : 0.0139 (23x)**

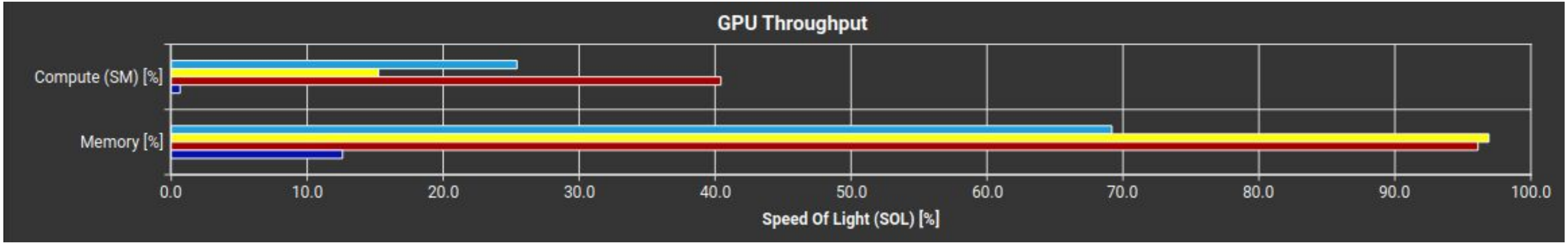


# Kernel optimization for OpenMP (4/4)

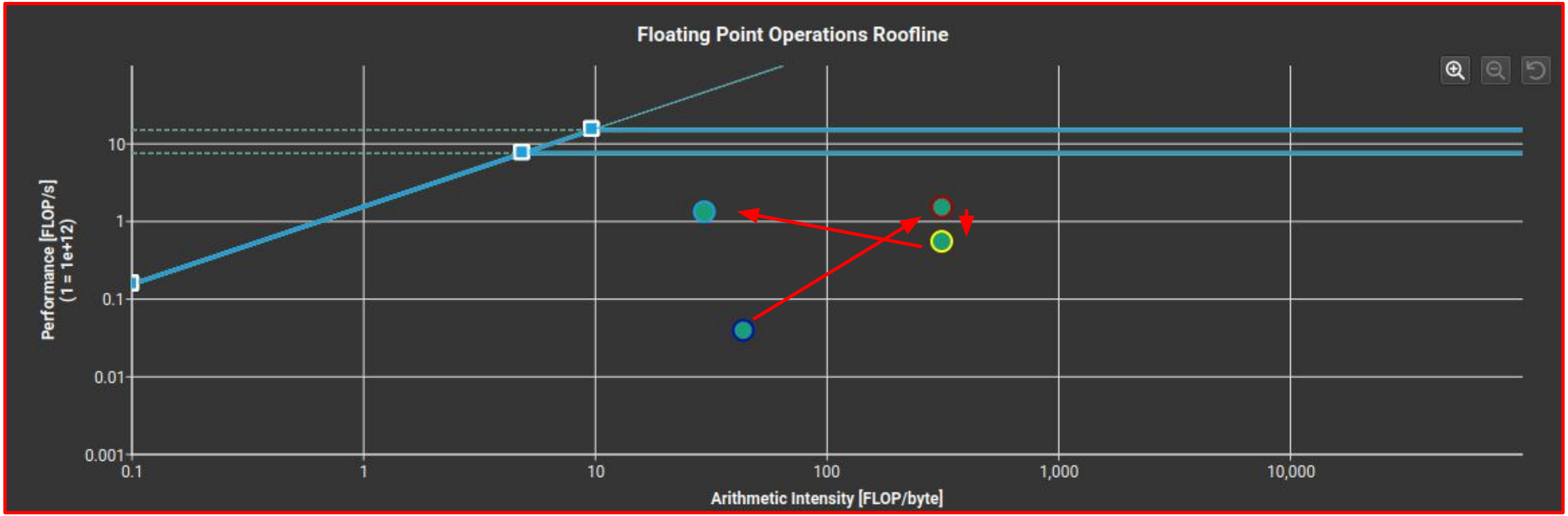


**GPU Speed Of Light Throughput**

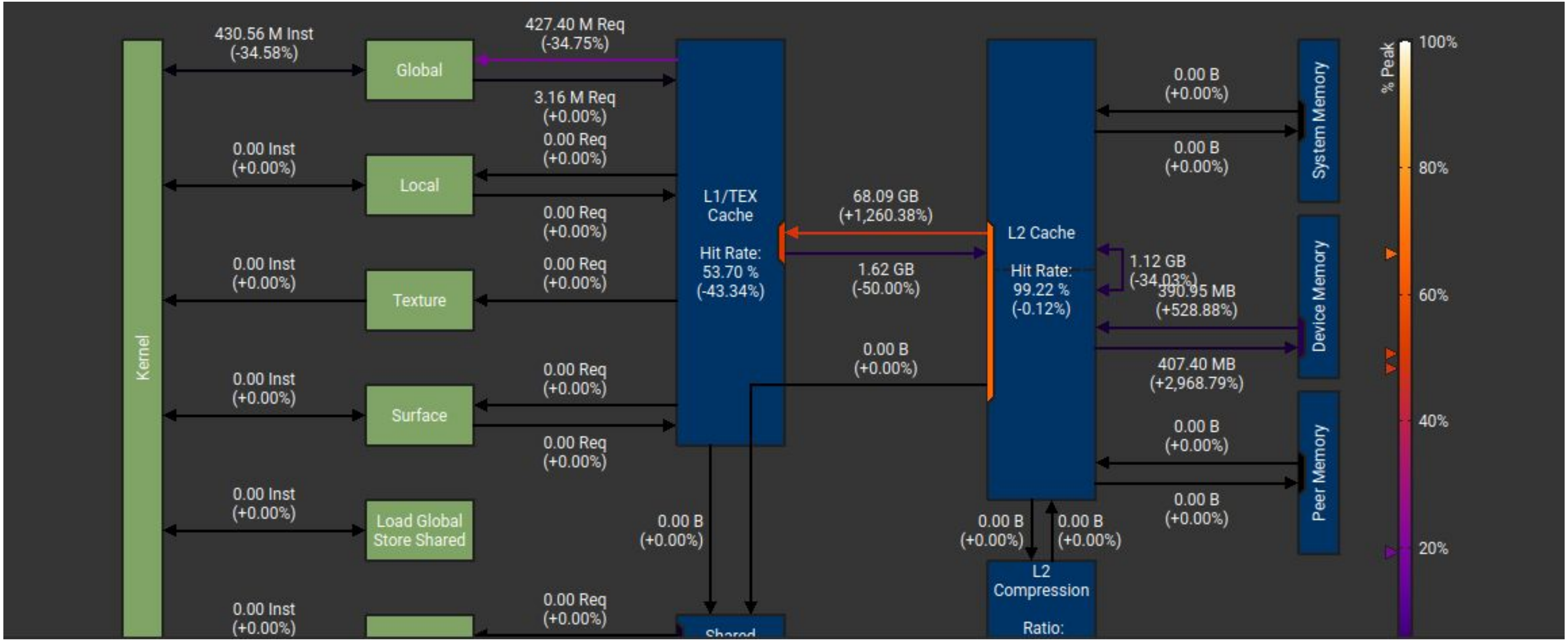
| | | | |
|---|---|---|---|
| Compute (SM) Throughput [%] | 25.43 (+66.74%) | Duration [msecond] | 18.57 (-58.50%) |
| Memory Throughput [%] | 69.18 (-28.61%) | Elapsed Cycles [cycle] | 20,332,815 (-58.51%) |
| L1/TEX Cache Throughput [%] | 64.42 (-33.62%) | SM Active Cycles [cycle] | 20,296,451.84 (-58.53%) |
| L2 Cache Throughput [%] | 69.18 (+905.27%) | SM Frequency [cycle/nsecond] | 1.10 (-0.03%) |
| DRAM Throughput [%] | 2.76 (+2,450.66%) | DRAM Frequency [cycle/nsecond] | 1.22 (-0.03%) |



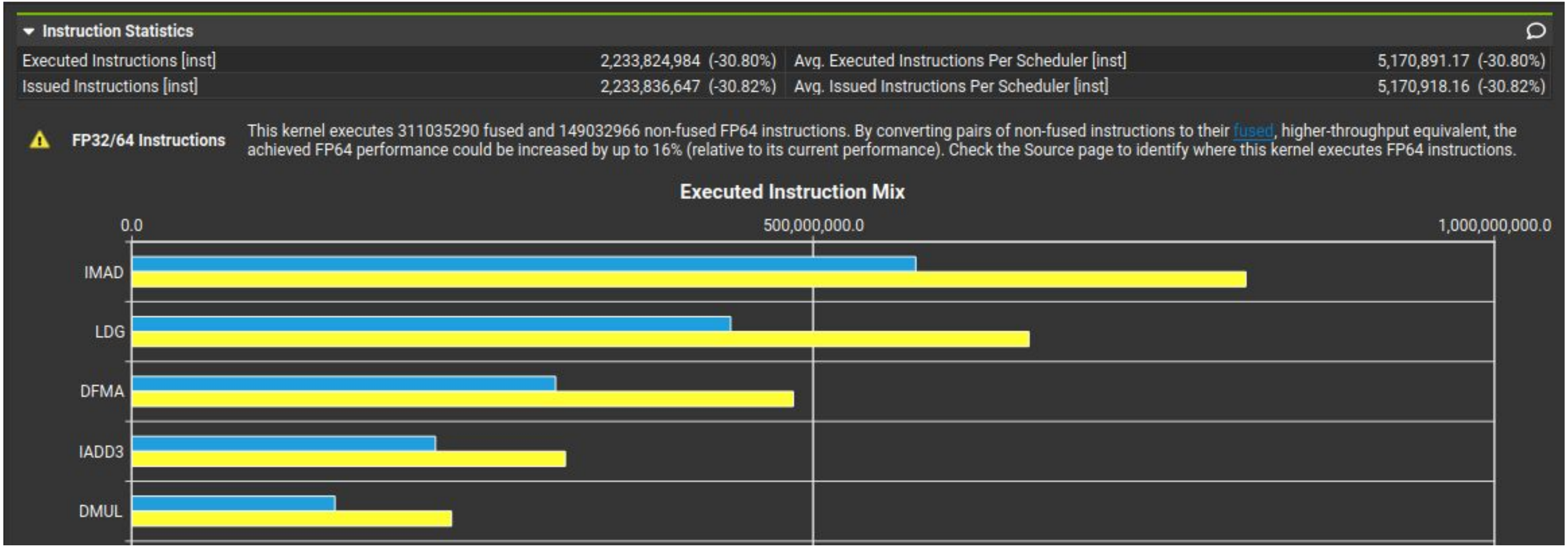
# Kernel optimization for OpenMP (4/4)



# Kernel optimization for OpenMP (4/4)



# Kernel optimization for OpenMP (4/4)



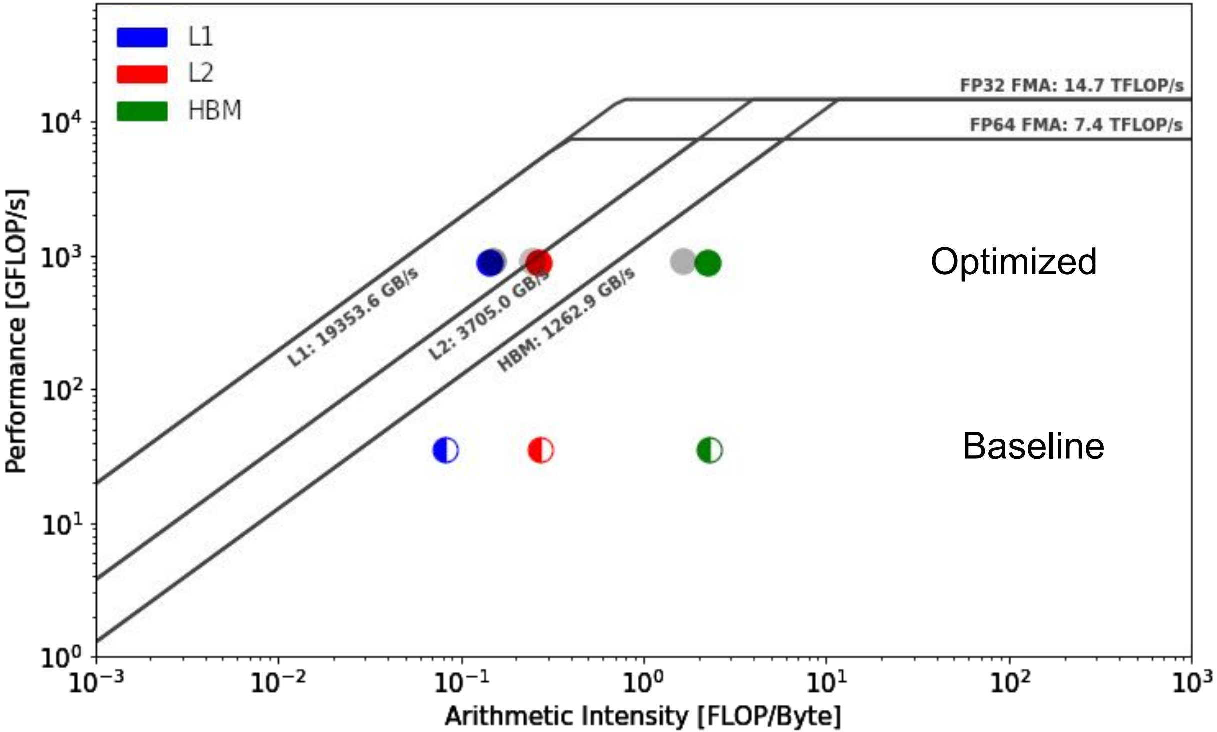
**Instruction Statistics**

| | | | |
|---|---|---|---|
| Executed Instructions [inst] | 2,233,824,984 (-30.80%) | Avg. Executed Instructions Per Scheduler [inst] | 5,170,891.17 (-30.80%) |
| Issued Instructions [inst] | 2,233,836,647 (-30.82%) | Avg. Issued Instructions Per Scheduler [inst] | 5,170,918.16 (-30.82%) |

⚠ **FP32/64 Instructions** This kernel executes 311035290 fused and 149032966 non-fused FP64 instructions. By converting pairs of non-fused instructions to their fused, higher-throughput equivalent, the achieved FP64 performance could be increased by up to 16% (relative to its current performance). Check the Source page to identify where this kernel executes FP64 instructions.

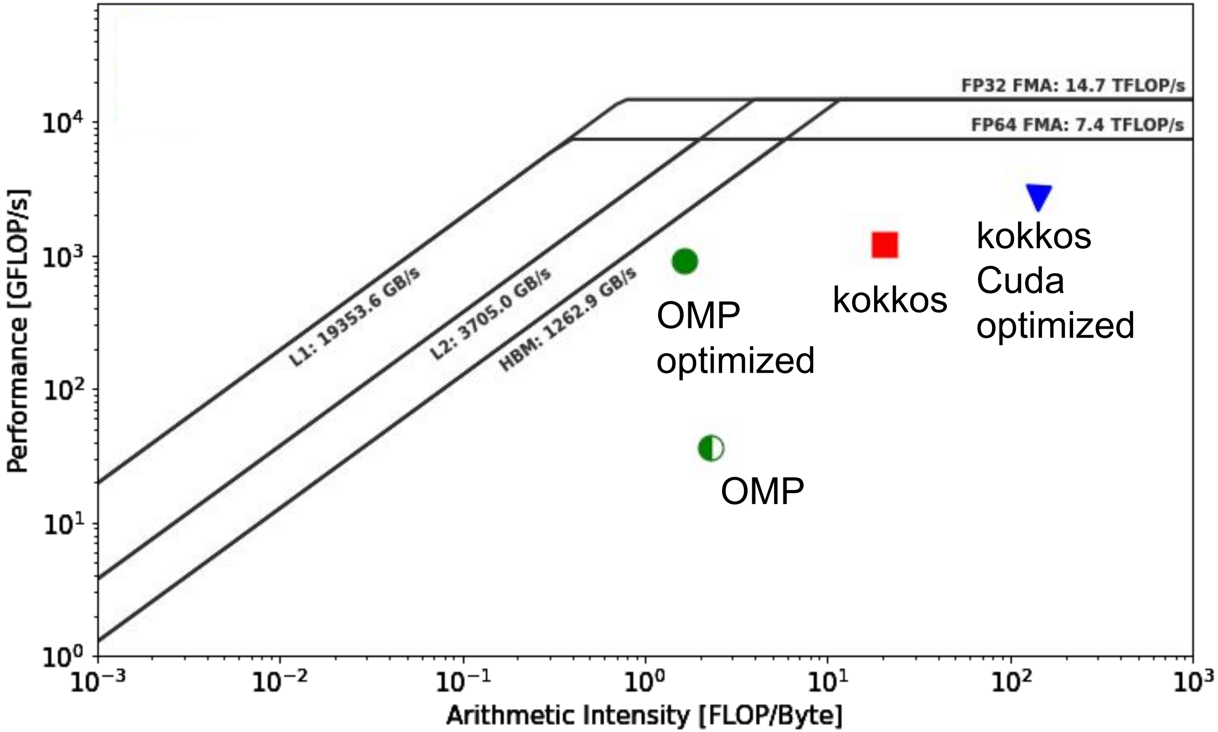
# Kernel optimization: Hierarchical roofline

- **Baseline** implementation has **poor cache locality**
- High **data reuse** between **L2** and **HBM after optimization**
- **HBM AI lower** after **optimization** for nvc due to higher data movement



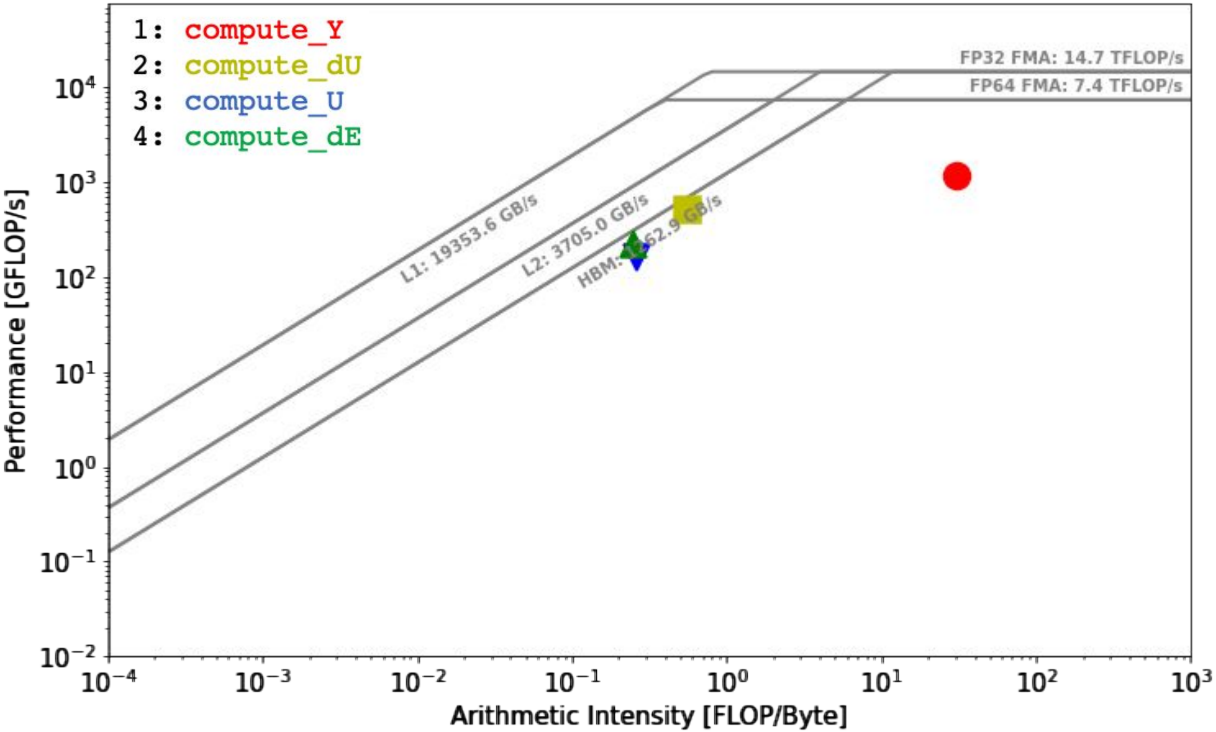
# Comparing across APIs

- **Versatile tool capable** of **comparing** across **APIs**
- **AI improvements** can come from **better data management** as well as **algorithm optimization**
- **Optimization** in the form of better **scratch memory usage** by **Evan Weinberg (NVIDIA)** and **Rahul Gayatri (NERSC)**

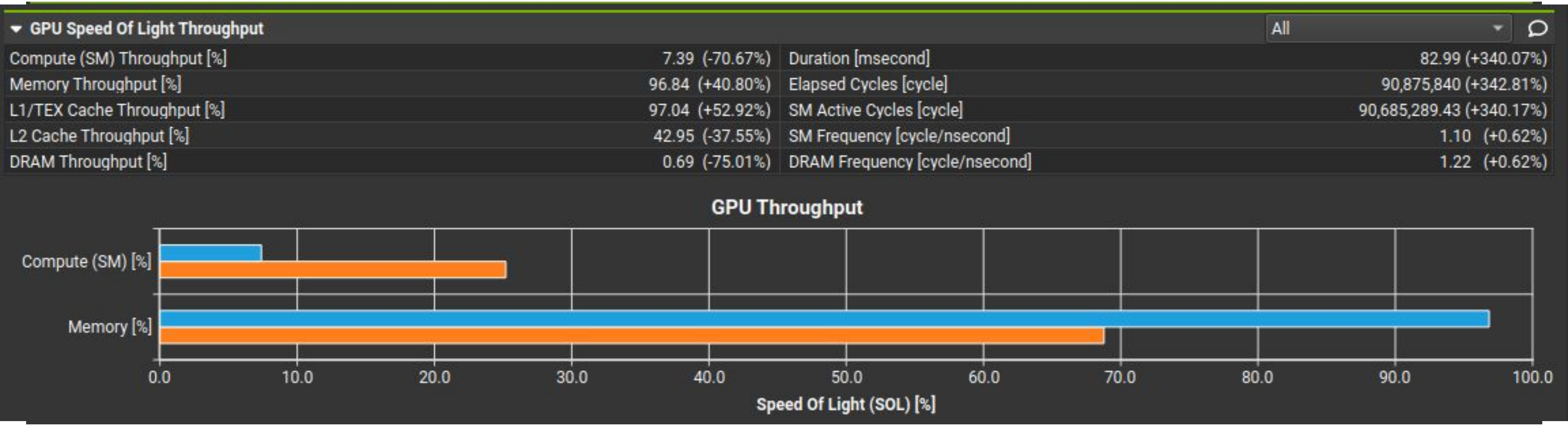


# Kokkos OMP target vs Native (1/3)

- **Note:** algorithm was **refactored** for the **Kokkos version**
- All kernels demonstrate **high AI** and are **near** or **in compute bound** regime
- **Grind times: (ms/atm-step)**
  **nvc++ : 0.0139**
  **kokkos: 0.0507**



# Kokkos OMP target vs Native (2/3)



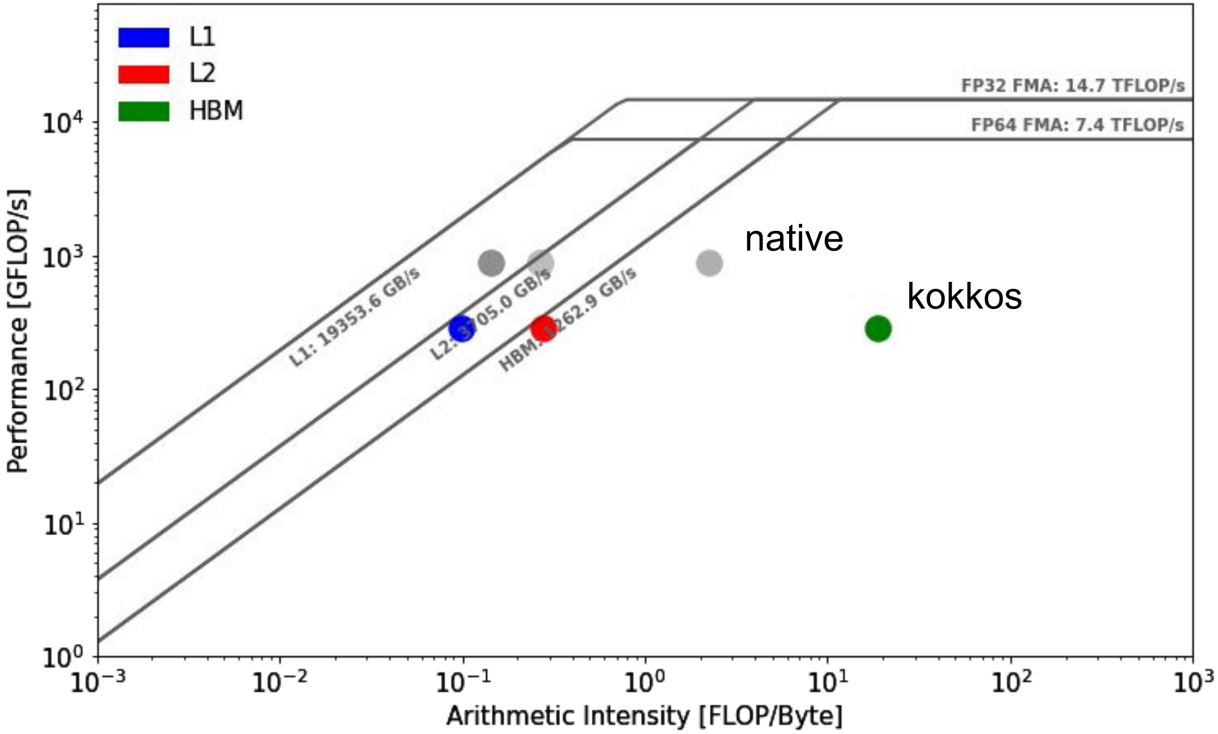
GPU Speed Of Light Throughput

|  |  |  |  |
|---|---|---|---|
| Compute (SM) Throughput [%] | 7.39 (-70.67%) | Duration [msecond] | 82.99 (+340.07%) |
| Memory Throughput [%] | 96.84 (+40.80%) | Elapsed Cycles [cycle] | 90,875,840 (+342.81%) |
| L1/TEX Cache Throughput [%] | 97.04 (+52.92%) | SM Active Cycles [cycle] | 90,685,289.43 (+340.17%) |
| L2 Cache Throughput [%] | 42.95 (-37.55%) | SM Frequency [cycle/nsecond] | 1.10 (+0.62%) |
| DRAM Throughput [%] | 0.69 (-75.01%) | DRAM Frequency [cycle/nsecond] | 1.22 (+0.62%) |

- **Blue: kokkos omp target backend**, **Orange: nvc(baseline)**
- SOL: **native openmp target (nvc)** has **higher SM %** but **Kokkos has less memory throughput % utilization at L2 and DRAm levels**
- **Kokkos** has a lot **higher instruction count** compared to **native**

# Kokkos OMP target vs Native (3/3)

- **Higher reuse** between **HBM and L2** for kokkos code but **performance** is lower
- **Higher AI** does not always mean **higher performance**



native

kokkos

# Conclusions

- Roofline analysis can provide **compute and memory efficiency** of the **code**
- Analysis can be performed **without intrusive code changes**

**<u>For Users:</u>**

- Researchers can **choose better combinations of architectures and compilers** based on **accuracy, speed, as well as efficiency**
- Rooflines helpful when **optimizing the code**

**<u>For Developers:</u>**

- **Algorithm developers** can demonstrate **platform performance portability**
- **Although rooflines** do **not** provide **complete picture**, can **help** determine architecture-dependent **compiler-optimization roadblocks**

# Acknowledgement

**Dr. Rahul Gayatri**

**Dr. Aidan Thompson**

**Dr. Danny Perez**

I would like to thank **Evan Weinberg (NVIDIA)** for his immense contribution in optimizing the TestSNAP code

# Thank you!