Co-designing for Sparseness: Simulating Next-Generation Memory Systems

Jeffrey Young, PhD · Principal Research Scientist · Partnership for Advanced Computing Environments

Jered Trujillo-Dominguez, Kevin Sheridan, Galen Shipman (LANL)

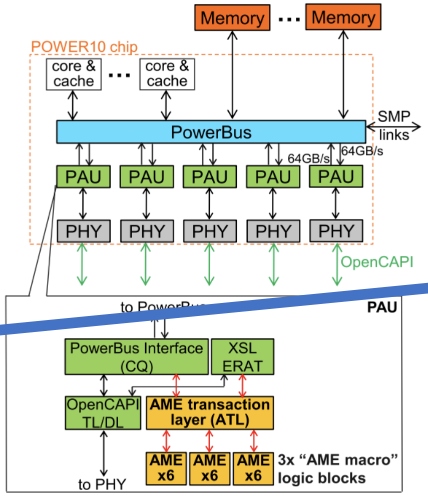
Connor Radelja, Christopher Scott (GT)

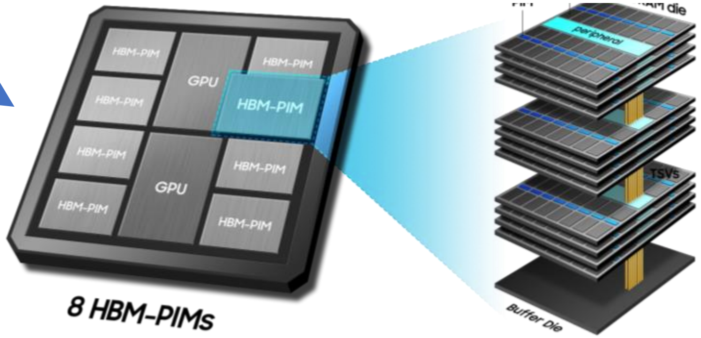
Patrick Lavin (SNL)

August 14th, 2024

Georgia Tech

Motivation





The "Cambrian Explosion" of new accelerators has led to many domain specific accelerators – not just in AI but also for data movement!

Motivation

	Memory subsystem						Floating Point		
	L1	L2	L3	DRAM	DRAM BW	Mem Latency	DP FLOPs	Vectorization	Non-FP
Flag 3D Ale							2.50%	7.10%	97.50%
PartiSN 42 groups							26.20%	90.40%	73.80%
Jayenne DDMC Hohlraum							14.30%	0.20%	85.70%
xRAGE Shaped Charge							6.50%	14.00%	93.50%
Application 1							7.80%	19.20%	92.20%
Application 2							8.10%	17.60%	91.90%

Hardware Bottlenecks for LANL HPC codes [1]

However, we still have little understanding of how future memory accelerators might affect codes of interest because:

- Finding appropriate regions of interest (ROI) for the memory system is challenging to infer even with tools like SimPoints and LoopPoint
- Some applications that we might want to benchmark can't be fully shared to extract meaningful traces or ROIs

Our ideal workflow would allow us to 1) capture relevant memory accesses from real-world applications, 2) benchmark them on real systems, and 3) simulate new hardware models

[1] G. Shipman, et al., Assessing the Memory Wall in Complex Codes, MCHPC 2022 doi: 10.1109/MCHPC56545.2022.00009

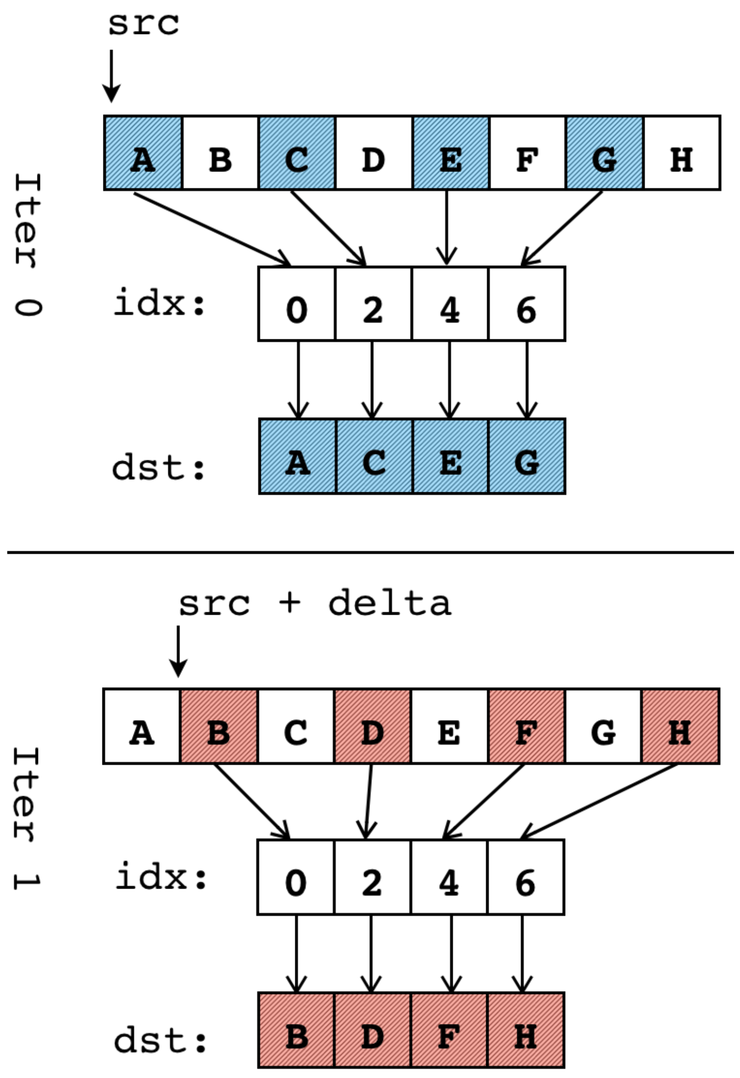
Spatter - Version 1.0

The basis of Spatter is two kernels; one for gather and another for scatter.

<u>Gather kernel:</u>
for i in 0..N:
    reg = gather(src + delta*i, idx)

<u>Scatter kernel:</u>
for i in 0..N:
    scatter(dst + delta*i, idx, reg)

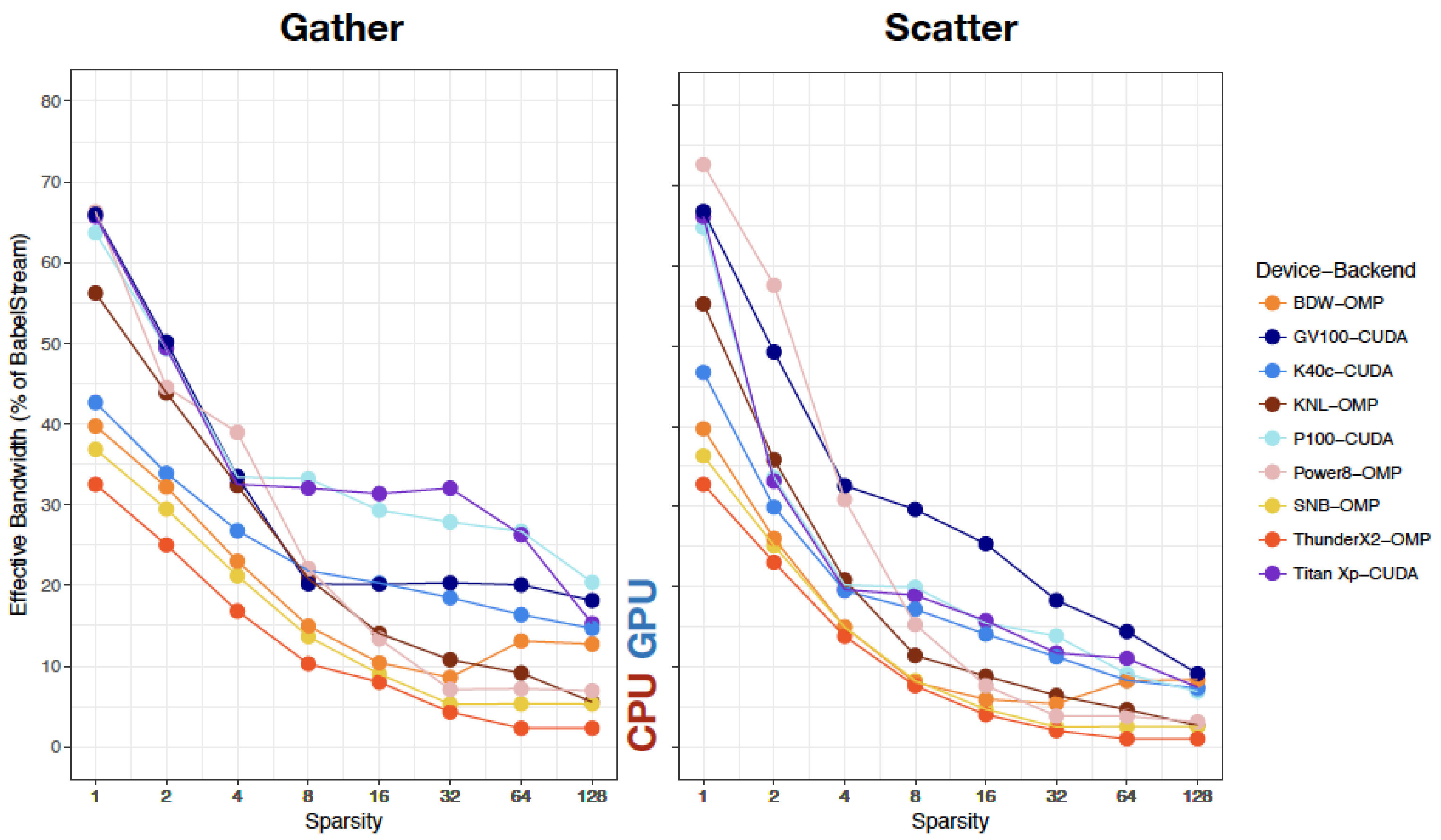
The delta and the pattern in idx specify the *memory access pattern.*



Spatter - Version 1.0 Results

Initial tests focused on CPU and GPU analysis using OpenMP and CUDA backends

Vector and scalar modes along with prototype backends for SYCL

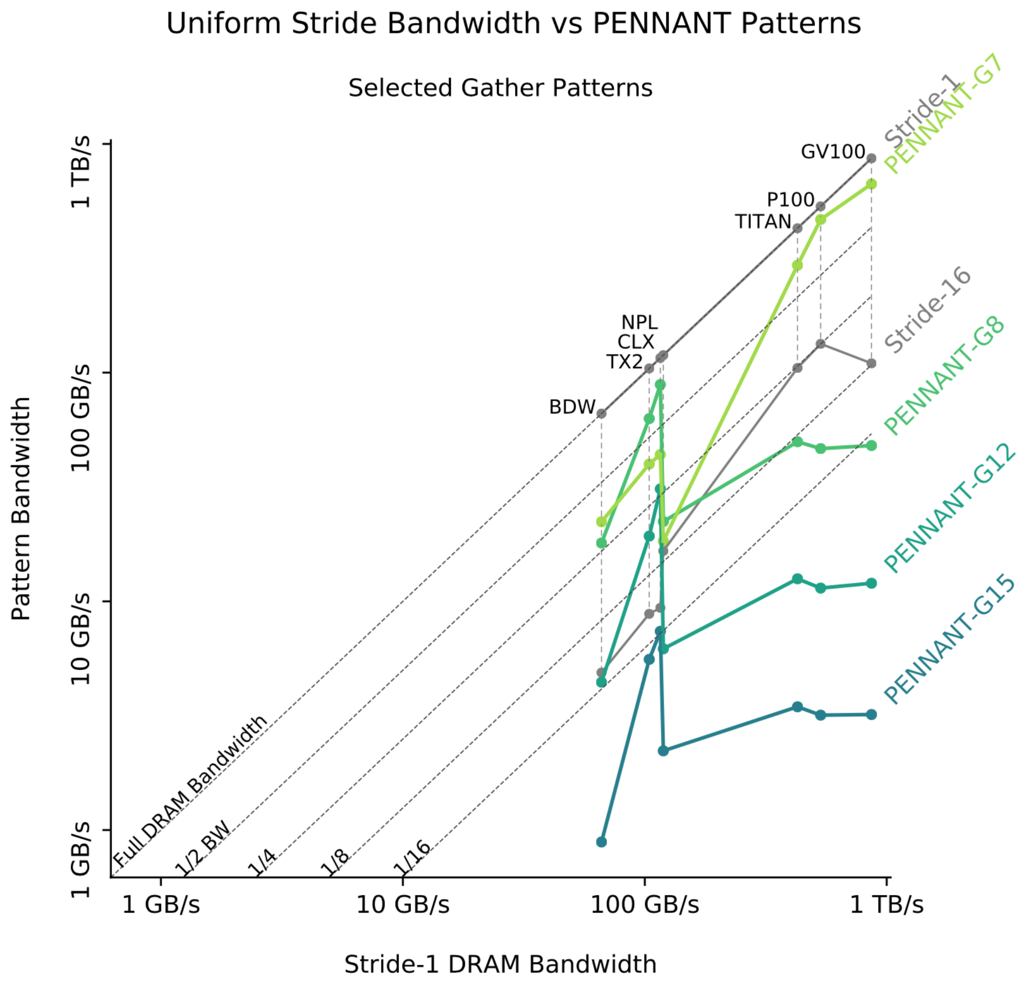


Georgia Tech

Spatter - Version 1.0 Results

Bandwidth-Bandwidth plots and application inputs for common HPC applications allowed for investigating "Peak Bandwidth versus Pattern Bandwidth"

	Index	Delta	Index Type
PENNANT	[2,484,482,0,4,486,484,2,6,488,486,4,8,490,488,6]	2	N/A
LULESH	[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]	1	Stride-1
NEKBONE	[0,6,12,18,24,30,36,42,48,54,60,66,72,78,84,90]	3	Stride-6
AMG	[1333,0,1,2,36,37,38,72,73,74,1296,1297,1298,1332, 1334,1368]	1	Mostly Stride-1



The Need for an Open Source Workflow

1. Trace G/S Instructions - pulled from a proprietary simulator

Gather 0x0040, [0, 2, 4, 6]
Gather 0x0044, [0, 2, 4, 6]
Gather 0x0048, [0, 2, 4, 6]
Scatter 0x004C, [1, 1, 5, 5]

2. Change base address to delta

Gather _, [0, 2, 4, 6]
Gather 4, [0, 2, 4, 6]
Gather 4, [0, 2, 4, 6]
Scatter 4, [1, 1, 5, 5]

3. Aggregate Counts

Gather 4, [0, 2, 4, 6], 2
Scatter 4, [1, 1, 5, 5], 1

← Spatter's Format

Previous work relied on proprietary tools to create Spatter application inputs. We'd like to have a common open source workflow!

GS Patterns

Sliding window approach is used to track non-trivial memory accesses within an application trace or region of interest (ROI)

$$
\begin{array}{cccccc}
 & \text{maddr1} & \text{maddr2} & \dots & \text{maddrN} & \textit{iaddr} \\
\textbf{ScatterWindow} = & 0x0480 & 0x0488 & \dots & 0x1488 & 0x0001 \\
 & 0x1780 & 0x1788 & \dots & 0x0783 & 0x0003 \\
 & \dots & \dots & \dots & \dots & \dots \\
 & 0x9580 & 0x5588 & \dots & 0x3581 & 0x1019
\end{array}
$$

Multiple filters are used to keep the most relevant access patterns for the final pattern output

Filter #	Description
1	Index distances are only -1, 0, and/or 1
2	No symbol
3	Not in top 10 window appearance counts
4	Less than 1024 instances
5	Less than 6 unique index distances and less than 50% out of bounds distances*

*Filter rules apply to each full memory access sequence. Sub-sequences are not removed. *Default out of bounds distances in $(-\infty, -513]$ or $[513, \infty)$.*

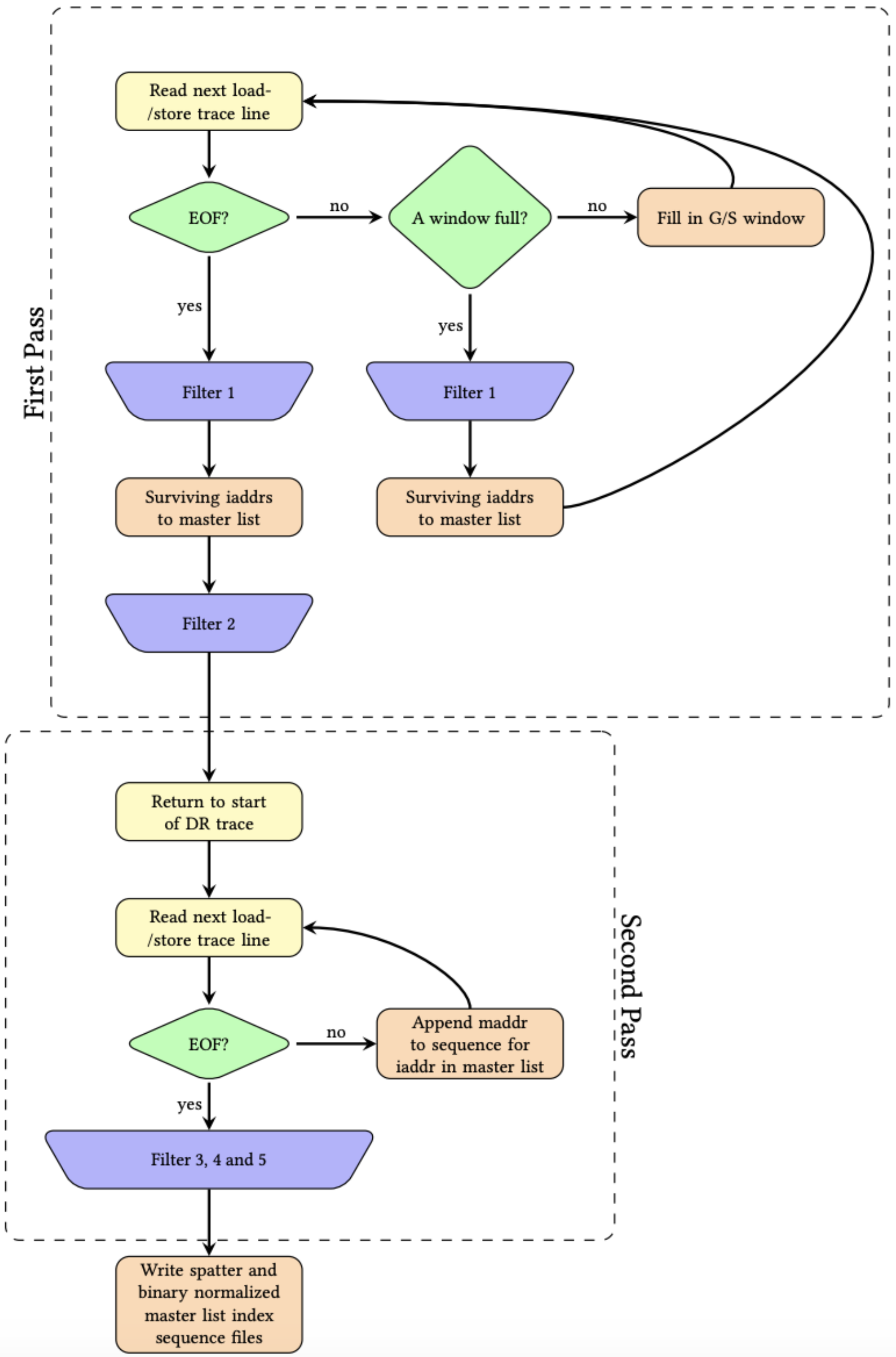
GS Patterns codebase at https://github.com/lanl/gs_patterns and https://github.com/hpcgarage/gs_patterns

GS Patterns

Two passes are used to apply all filters and output the patterns of interest

Filter #	Description
1	Index distances are only -1, 0, and/or 1
2	No symbol
3	Not in top 10 window appearance counts
4	Less than 1024 instances
5	Less than 6 unique index distances and less than 50% out of bounds distances*

*Filter rules apply to each full memory access sequence. Sub-sequences are not removed. *Default out of bounds distances in* $(-\infty, -513]$ *or* $[513, \infty)$.

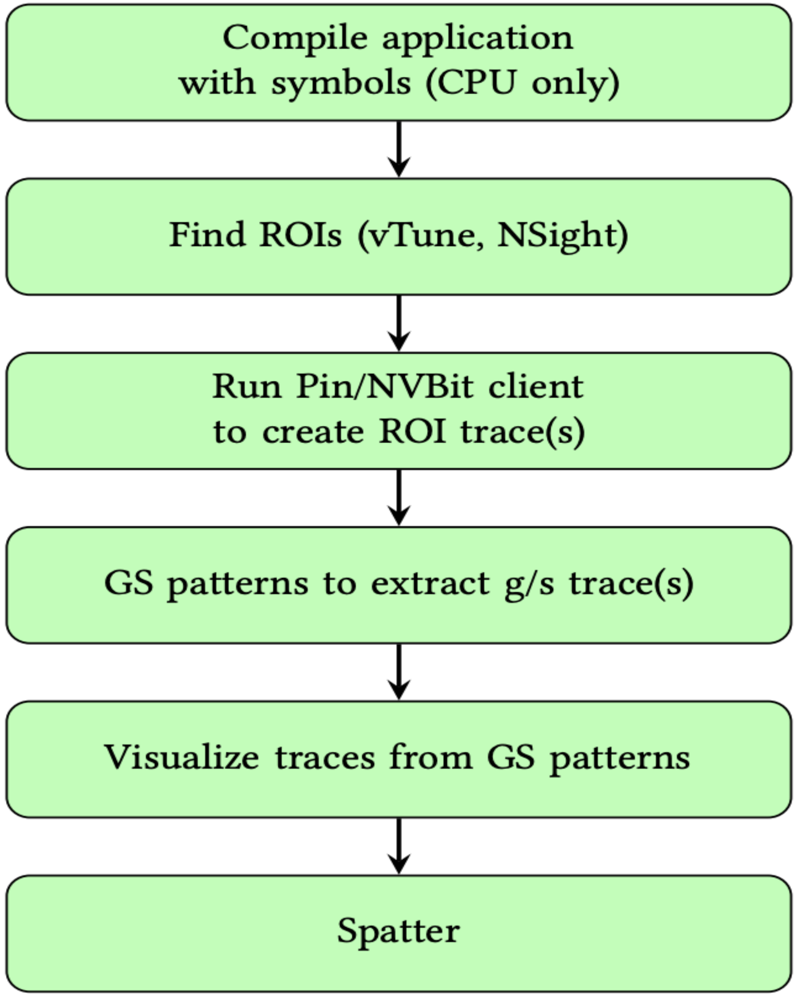


GS Patterns Workflow

Further extensions to GS Patterns refactored code to use C++ and a plugin infrastructure for PinTool, NVBit

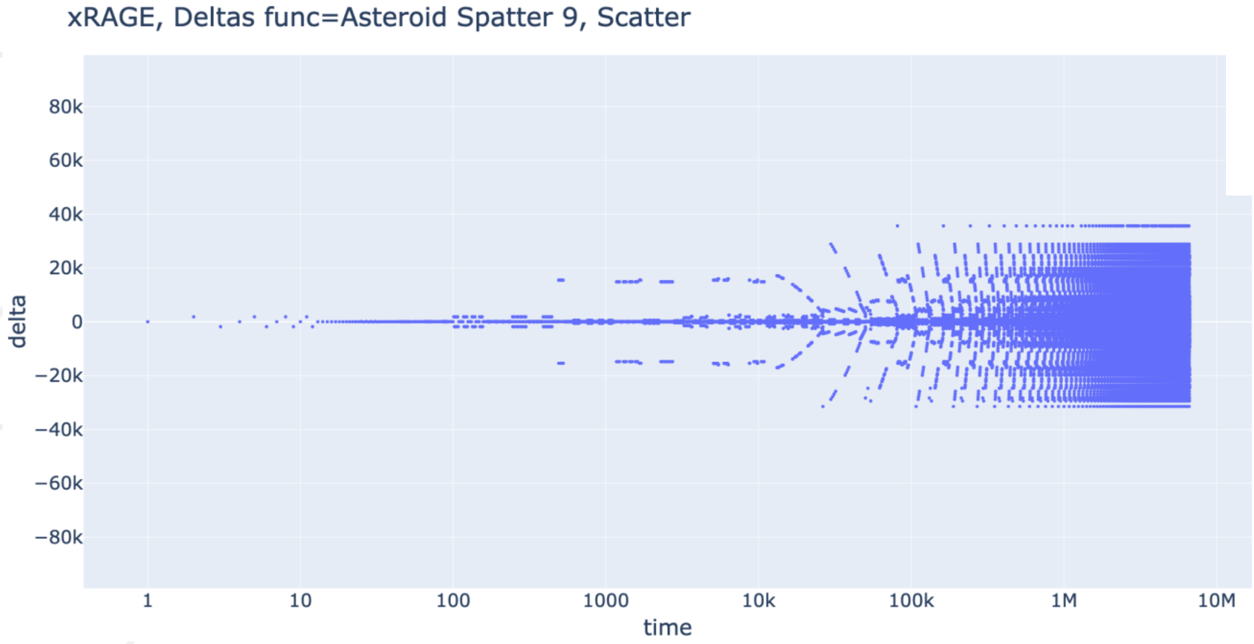
Currently ROI analysis speeds up the overall GS Patterns workflow but is a somewhat manual process to run and annotate codes

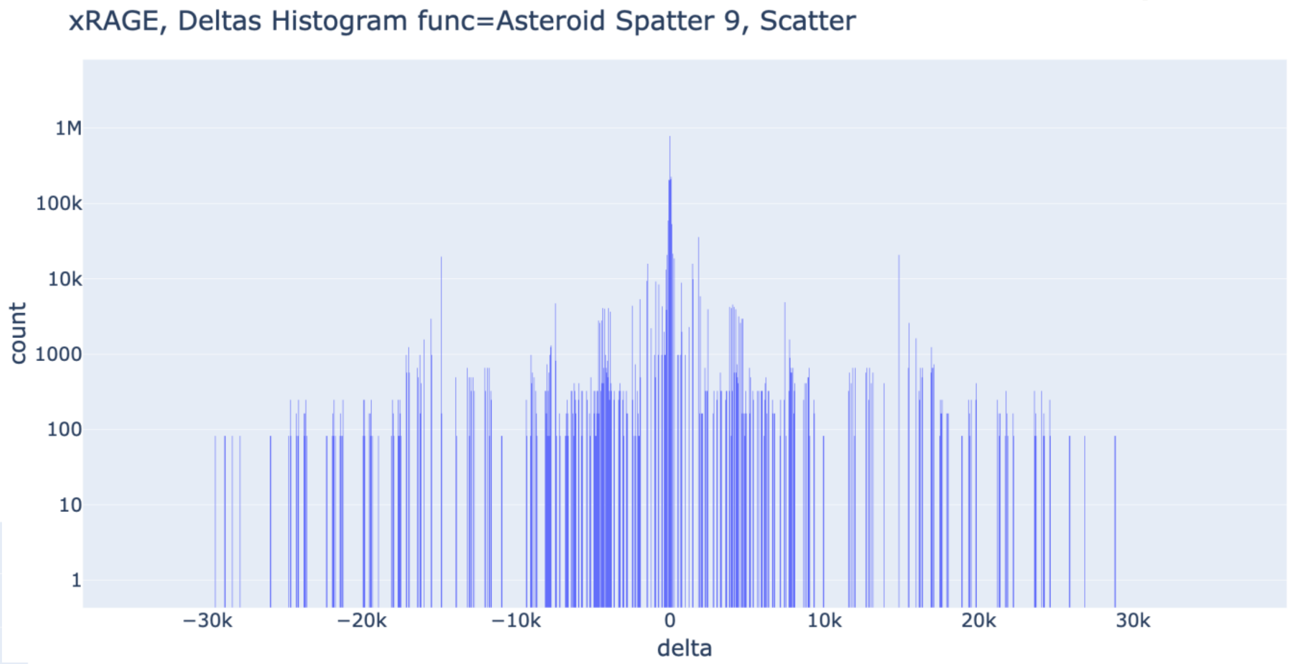
Check out collected public patterns at https://github.com/hpcgarage/spatter-patterns



Georgia Tech

Pattern Visualization Tools

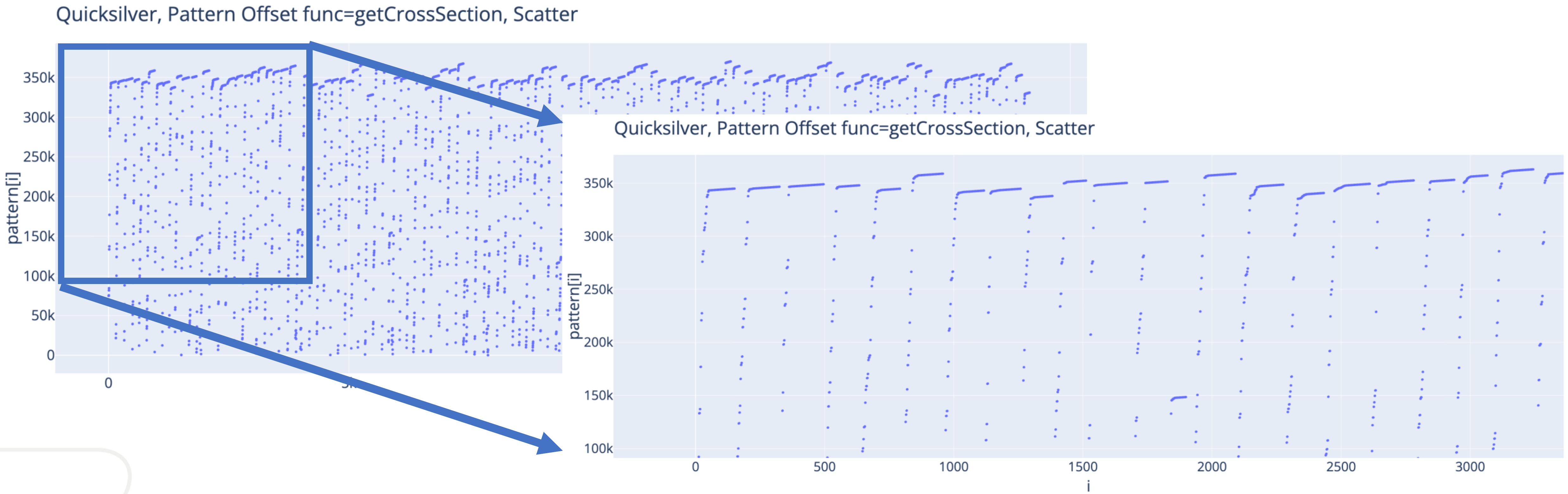




The GS Patterns JSON output can be easily visualized in a variety of different ways…

Georgia Tech

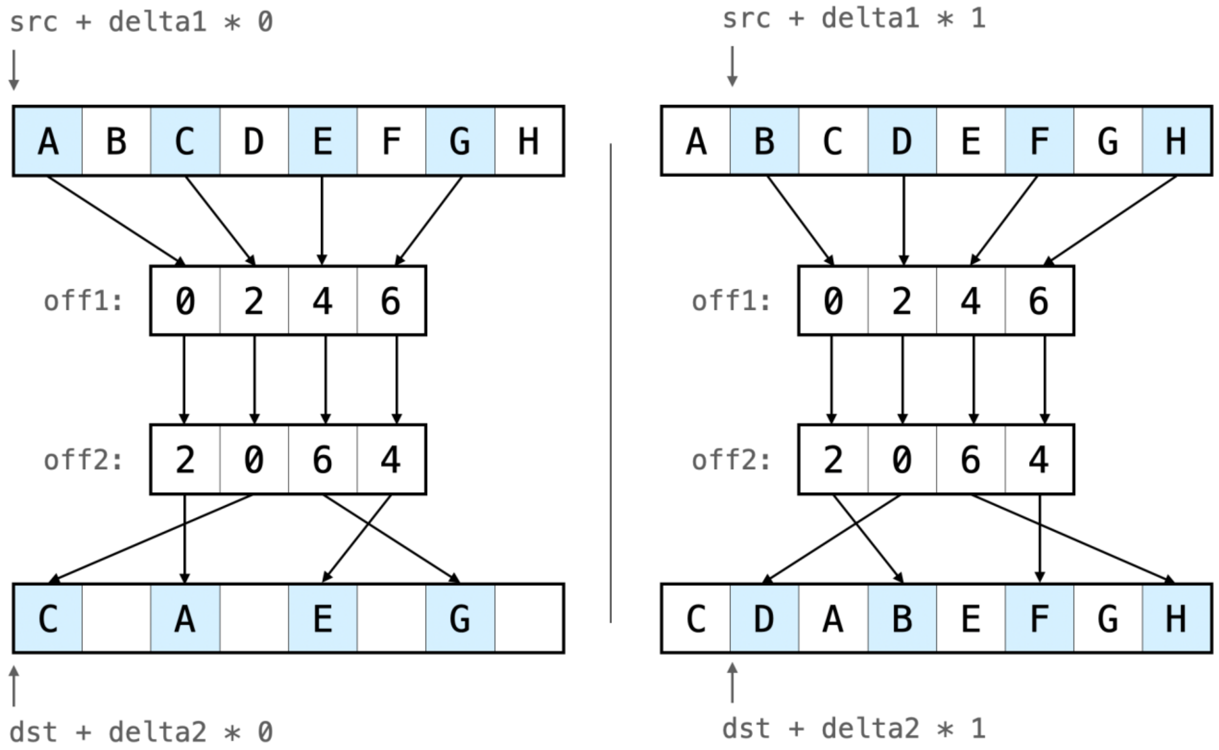
Pattern Visualization Tools



But we likely need to do more detailed statistical analysis to make more sense of these patterns..

Spatter 2.0

- Complete refactor of argument parsing, build system, and movement towards C++ design
- Addition of new kernels to better represent multiple levels of indirection - GatherScatter, MultiGather, MultiScatter
- Support for longer offset buffer lengths for improved application pattern representation
- MPI support for weak/strong scaling
- Support for atomics with scatter operations



GatherScatter Kernel Representation

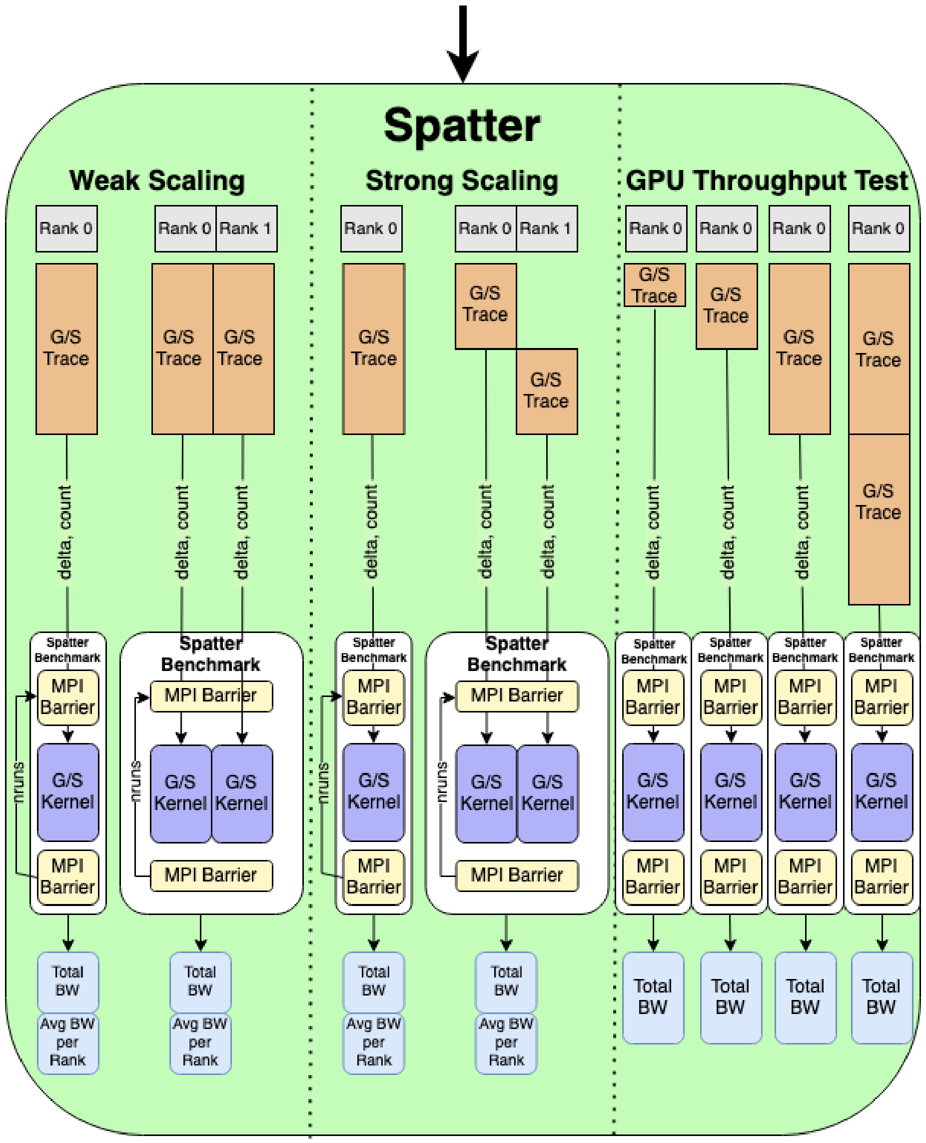
https://github.com/hpcgarage/spatter

Spatter 2.0 MPI Workflow

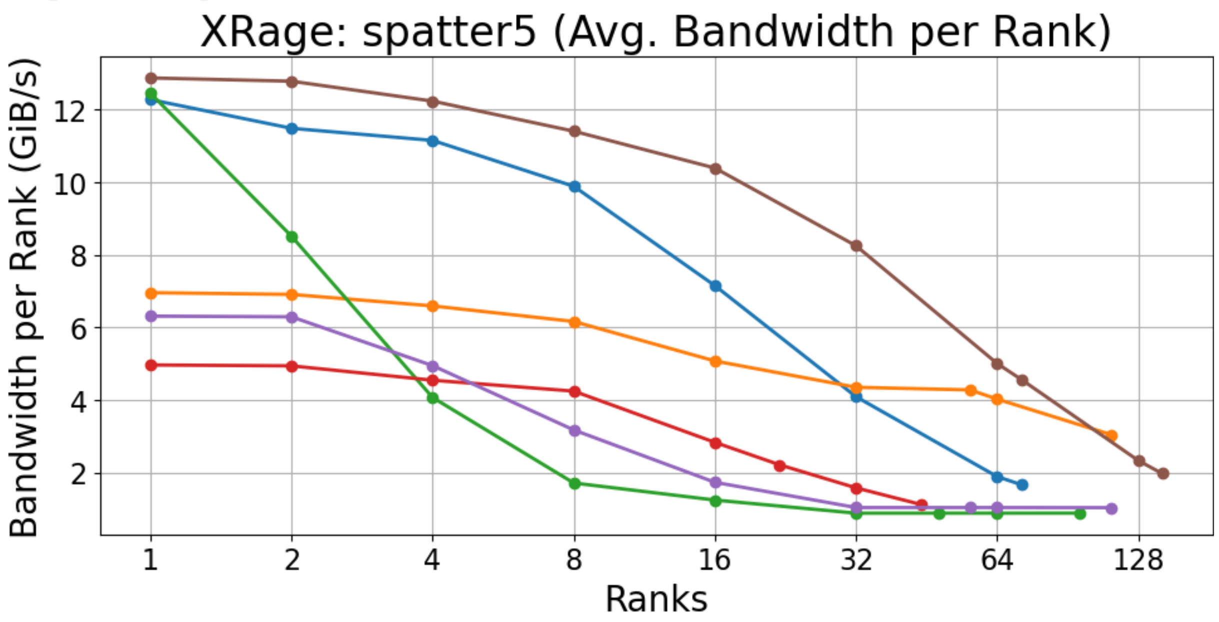
<u>Weak Scaling</u> – each MPI rank gets the same pattern and scaling scripts are used to sweep across N ranks

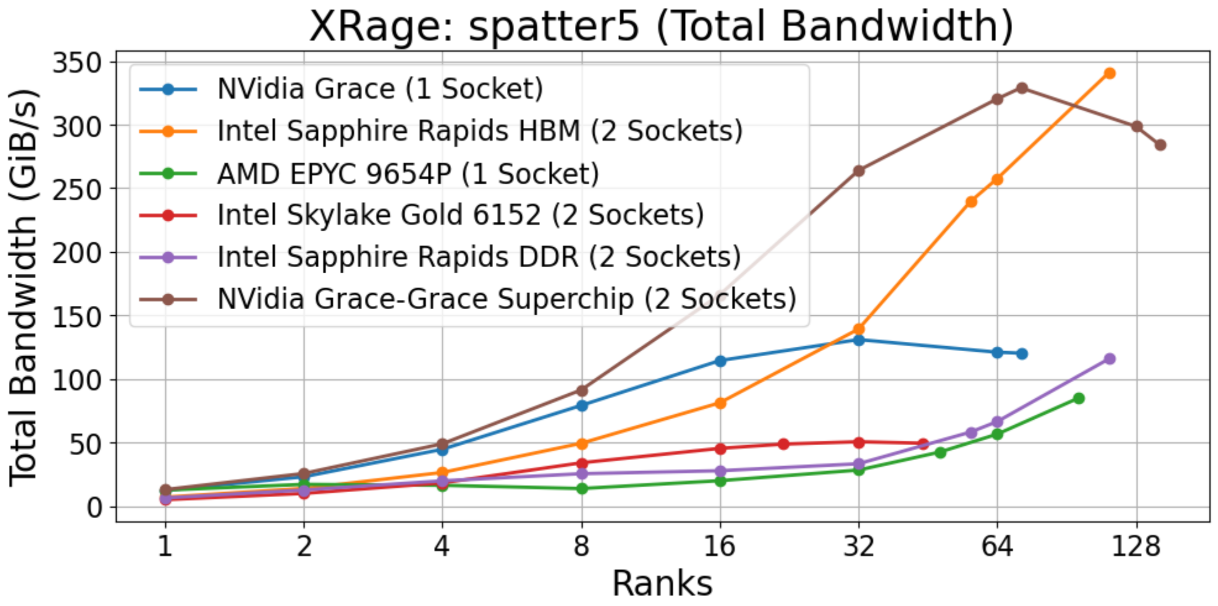
<u>Strong Scaling</u> – access patterns are partitioned across MPI ranks

<u>GPU Throughput</u> – patterns are truncated or expanded to vary amount of memory accesses and saturate GPU memory subsystem



Spatter 2.0 Results





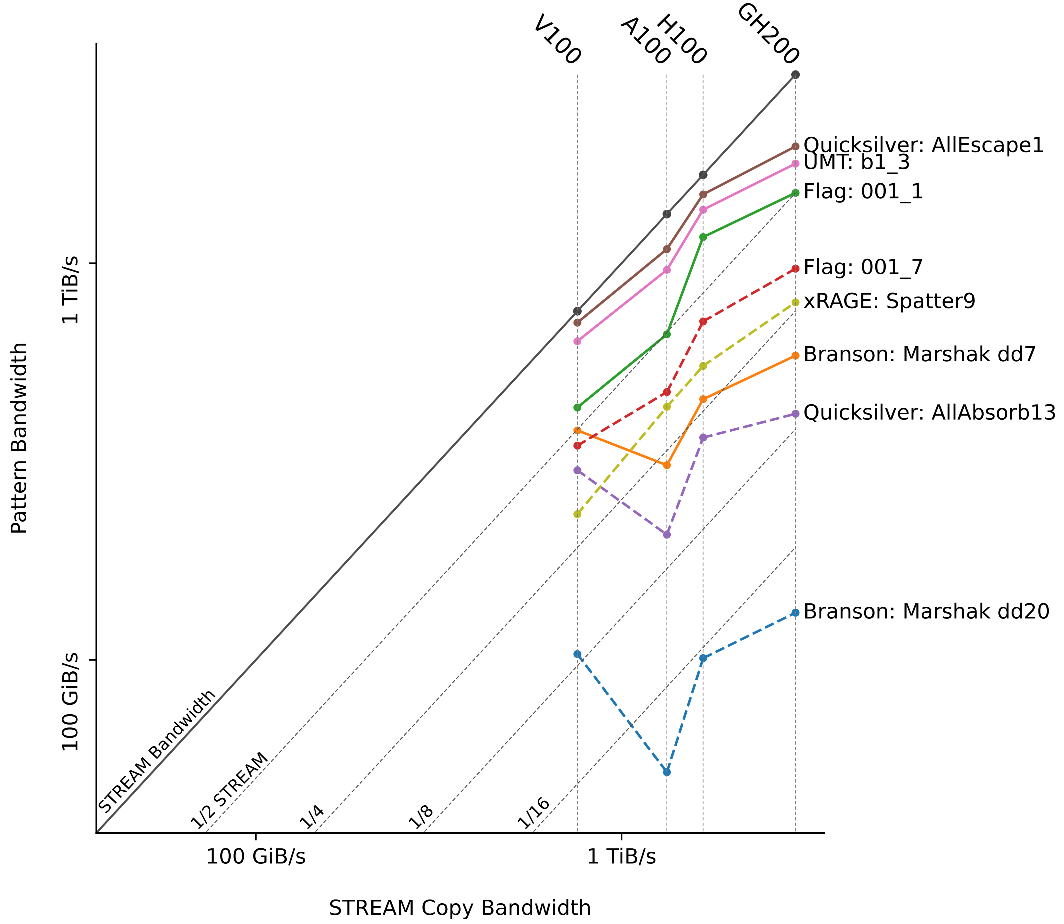
Weak Scaling MPI Tests for Xrage Gather pattern

See our MEMSYS 2024 paper in October 2024 for more details!

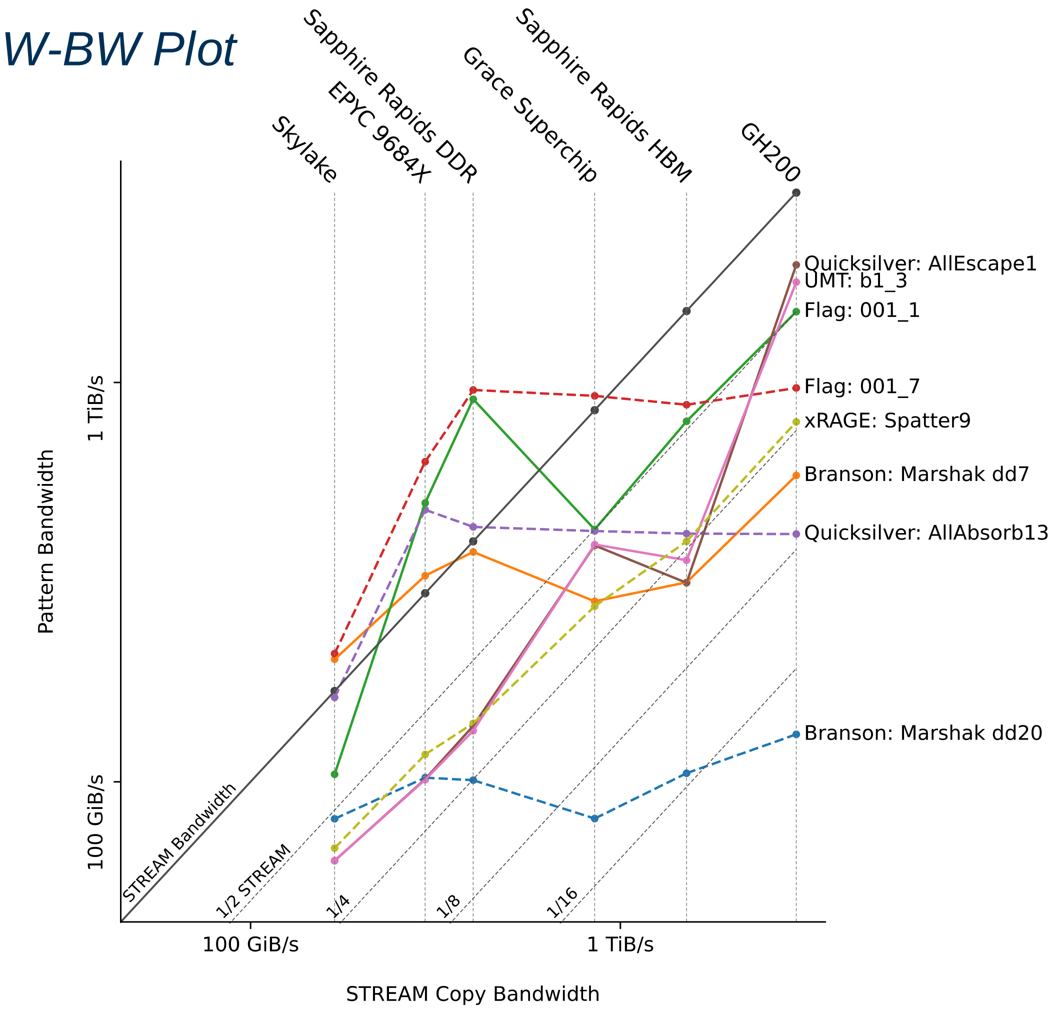
Georgia Tech

Spatter 2.0 Results

GPU BW-BW Plot



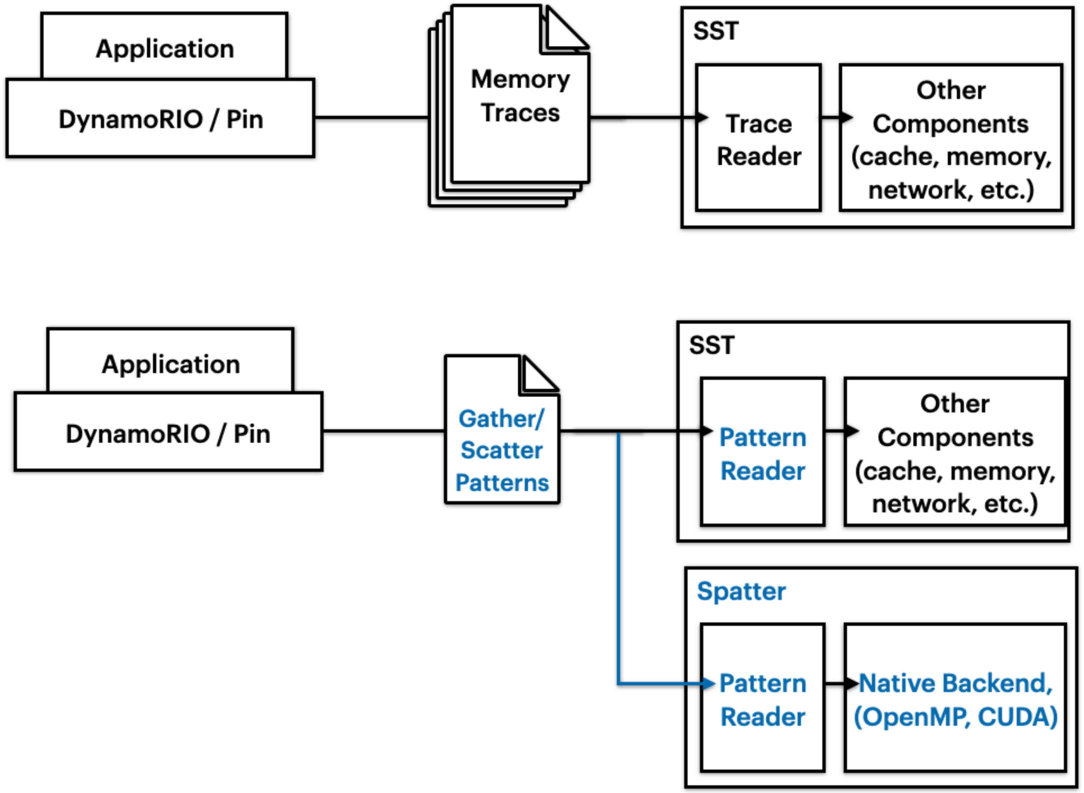
CPU BW-BW Plot



Revisiting Codesign

Our ideal tool workflow would allow us to:

- Capture relevant memory accesses from real-world applications
- Run these patterns on bleeding edge systems
- Use the ***same patterns*** to simulate the performance of future near-memory accelerators



Spatter Integration with SST

```
Running Spatter version 1.1
Compiler: GNU ver. 11.4.0
Backend: Serial
Aggregate Results? NO

Run Configurations
[ {'id': 0, 'kernel': 'gather', 'pattern': [0, 1, 2, 3, 4, 5, 6,
     'delta': 8, 'delta-gather': 8, 'delta-scatter': 8, 'count'
  {'id': 1, 'kernel': 'scatter', 'pattern': [0, 1, 2, 3, 4, 5, 6
     'delta': 8, 'delta-gather': 8, 'delta-scatter': 8, 'count'
  {'id': 2, 'kernel': 'sg', 'pattern': [], 'pattern-gather': [0,
     'delta': 8, 'delta-gather': 8, 'delta-scatter': 8, 'count'
  {'id': 3, 'kernel': 'multiscatter', 'pattern': [0, 1, 2, 3, 4,
     'delta': 8, 'delta-gather': 8, 'delta-scatter': 8, 'count'
  {'id': 4, 'kernel': 'multigather', 'pattern': [0, 1, 2, 3, 4,
     'delta': 8, 'delta-gather': 8, 'delta-scatter': 8, 'count'

config          bytes           time(s)         bw(MB/s)
0               268435456       0.0162493       16519.8
1               268435456       0.0166213       16150
2               536870912       0.0247413       21699.4
3               268435456       0.0186002       14431.8
4               268435456       0.0195687       13717.6
```

```
Run Configurations
[ {'id': 0, 'kernel': 'gather', 'pattern': [0, 1, 2, 3, 4, 5, 6, 7], 'pattern-gather': [], 'pattern-scatter': [],
    'delta': 8, 'delta-gather': 8, 'delta-scatter': 8, 'count': 4194304, 'wrap': 1, 'threads': 1},
  {'id': 1, 'kernel': 'scatter', 'pattern': [0, 1, 2, 3, 4, 5, 6, 7], 'pattern-gather': [], 'pattern-scatter': [],
    'delta': 8, 'delta-gather': 8, 'delta-scatter': 8, 'count': 4194304, 'wrap': 1, 'threads': 1},
  {'id': 2, 'kernel': 'sg', 'pattern': [], 'pattern-gather': [0, 1, 2, 3, 4, 5, 6, 7], 'pattern-scatter': [0, 1, 2, 3, 4, 5, 6, 7],
    'delta': 8, 'delta-gather': 8, 'delta-scatter': 8, 'count': 4194304, 'wrap': 1, 'threads': 1},
  {'id': 3, 'kernel': 'multiscatter', 'pattern': [0, 1, 2, 3, 4, 5, 6, 7], 'pattern-gather': [], 'pattern-scatter': [0, 1, 2, 3, 4, 5, 6, 7],
    'delta': 8, 'delta-gather': 8, 'delta-scatter': 8, 'count': 4194304, 'wrap': 1, 'threads': 1},
  {'id': 4, 'kernel': 'multigather', 'pattern': [0, 1, 2, 3, 4, 5, 6, 7], 'pattern-gather': [0, 1, 2, 3, 4, 5, 6, 7], 'pattern-scatter': [],
    'delta': 8, 'delta-gather': 8, 'delta-scatter': 8, 'count': 4194304, 'wrap': 1, 'threads': 1} ]

config          bytes           time(s)         bw(MB/s)        cycles          time(s)/cycles
0               268435456       0.0372244       7211.27         108773542       3.4222e-10
1               268435456       0.0372244       7211.27         109804560       3.39006e-10
2               536870912       0.0744489       7211.27         112722151       6.60464e-10
3               268435456       0.0372244       7211.27         109805007       3.39005e-10
4               268435456       0.0372244       7211.27         108774550       3.42217e-10

Simulation is complete, simulated time: 818.808 ms
```

STREAM Output (i5-12400F Alder Lake vs. SST Miranda Spatterbench

Spatter can be compiled as a library and used as a new pattern input for Miranda with SST

Georgia Tech

Spatter Integration with SST

```
Running Spatter version 1.1
Compiler: GNU ver. 11.4.0
Backend: Serial
Aggregate Results? NO

Run Configurations
[ {'id': 0, 'kernel': 'scatter', 'pattern': [0, 24, 48, 72, 96, 120, 144
    'pattern-gather': [], 'pattern-scatter': [], 'delta': 0, 'delta-gath
  ....
  {'id': 11, 'kernel': 'gather', 'pattern': [0, 24, 48, 72, 96, 120, 144
    'pattern-gather': [], 'pattern-scatter': [], 'delta': 1, 'delta-gath
```

config	bytes	time(s)	bw(MB/s)
0	73959168	0.0025168	29386.1
1	29593344	0.00101654	29111.8
2	21479040	0.000900899	23841.8
3	16384256	0.00103612	15813.1
4	12334080	0.000430775	28632.3
5	12334080	0.000427073	28880.5
6	12311808	0.000426331	28878.5
7	11265408	0.000443739	25387.5
8	9829632	0.000339094	28987.9
9	9829632	0.000783912	12539.2
10	9829632	0.000334772	29362.2
11	9250560	0.000322094	28720.1

```
Run Configurations
[ {'id': 0, 'kernel': 'scatter', 'pattern': [0, 24, 48, 72, 96, 120, 144, 168, 192, 216, 240, 264, 288, 312, 336, 360],
    'pattern-gather': [], 'pattern-scatter': [], 'delta': 0, 'delta-gather': 8, 'delta-scatter': 8, 'count': 577806, 'wrap': 1, 'threads': 1},

  {'id': 11, 'kernel': 'gather', 'pattern': [0, 24, 48, 72, 96, 120, 144, 168, 192, 216, 240, 264, 288, 312, 336, 360],
    'pattern-gather': [], 'pattern-scatter': [], 'delta': 1, 'delta-gather': 8, 'delta-scatter': 8, 'count': 72270, 'wrap': 1, 'threads': 1} ]
```

config	bytes	time(s)	bw(MB/s)	cycles	time(s)/cycles
0	73959168	0.0104005	7111.11	5489157	1.89474e-09
1	29593344	0.00410376	7211.27	2300269	1.78404e-09
2	21479040	0.00297854	7211.26	1669566	1.78402e-09
3	16384256	0.00230403	7111.12	3393131	6.79029e-10
4	12334080	0.00163812	7529.40	1259456	1.30066e-09
5	12334080	0.00154176	8000.01	2530289	6.09321e-10
6	12311808	0.00153897	8000.01	2525826	6.09296e-10
7	11265408	0.0015622	7211.25	826265	1.89067e-09
8	9829632	0.00136309	7211.30	2024452	6.73312e-10
9	9829632	0.00136309	7211.28	10223443	1.3333e-10
10	9829632	0.00136308	7211.32	1212079	1.12458e-09
11	9250560	0.00128279	7211.26	678285	1.89123e-09

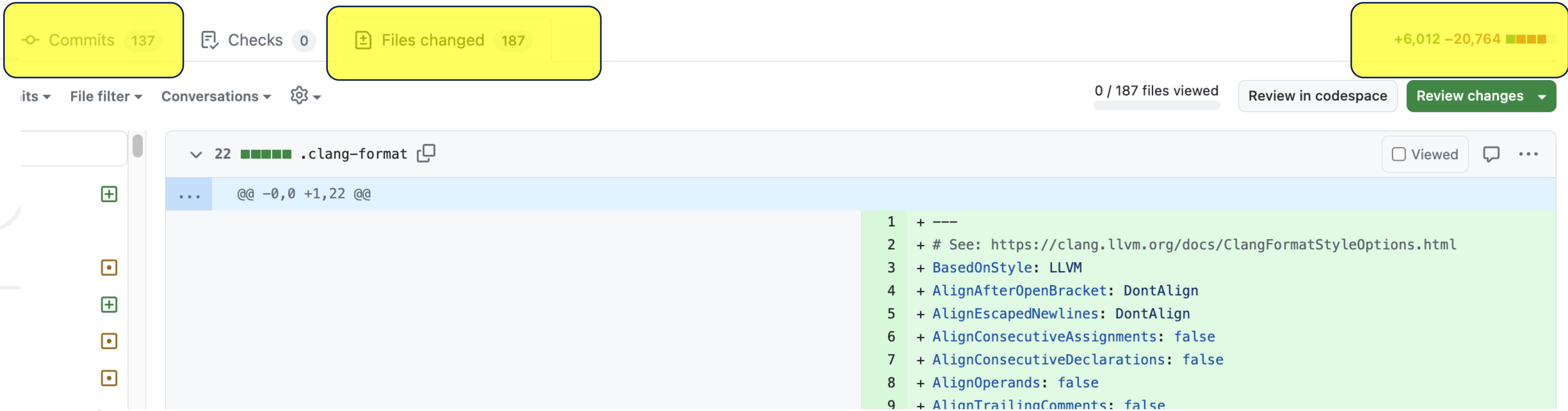
```
Simulation is complete, simulated time: 44.5121 ms
```

Lulesh Output (i5-12400F Alder Lake vs. SST Miranda Spatterbench)

Larger application patterns may benefit from either parallel simulation sweeps (e.g., gem5) or using MPI-based tests

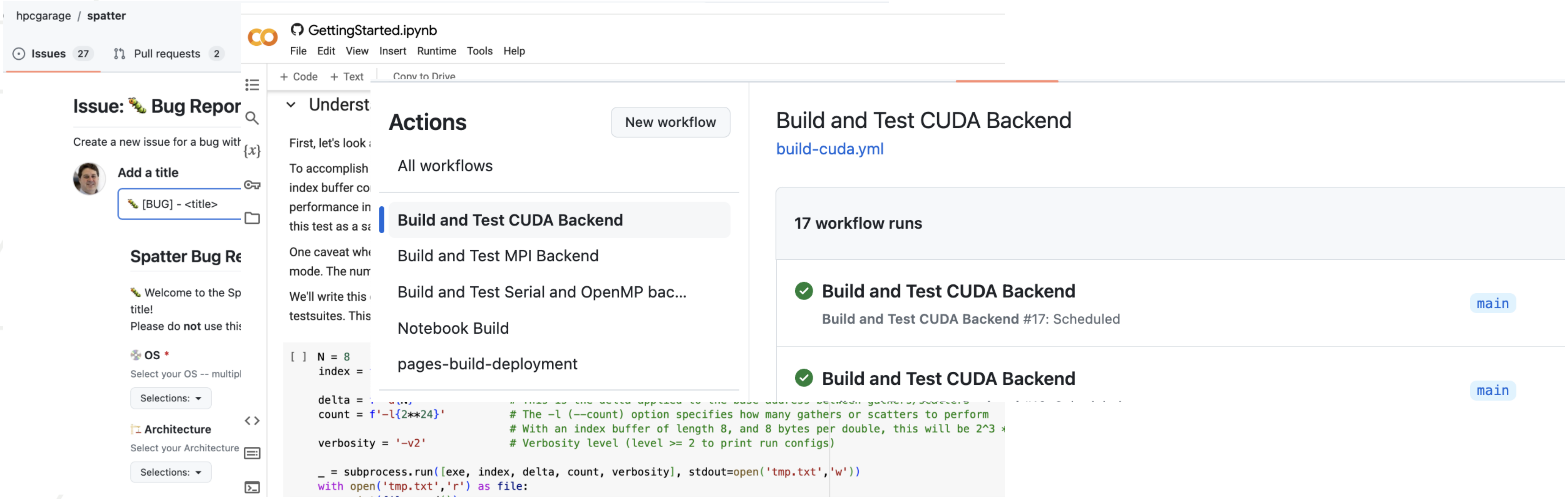
Thoughts on Software Sustainability

Unknowingly we committed several major sins with our Spatter refactor…



Thoughts on Software Sustainability

However, we've been working towards improving the overall codebase for ourselves and for new contributors



Summary

Motivating work at LANL for ATS-5 for new memory accelerator microbenchmarks led to the release of GS Patterns and Spatter 2.0

- New capabilities allow for more complete analysis and capture of real-world application patterns

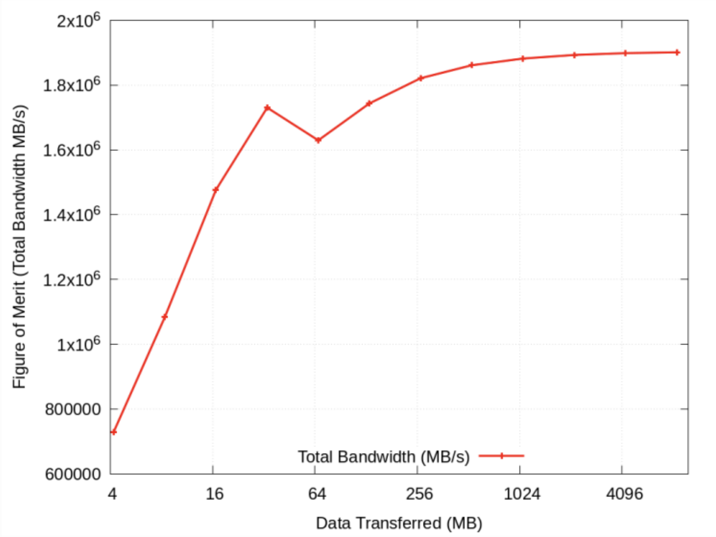


Fig. 3.14 *Spatter Throughput on H100 xRAGE Asteroid Pattern 5 on H100*

3.4.2.2. xRAGE Asteroid Spatter Pattern 9

Throughput experiment for the pattern in datafiles/xrage/asteroid/spatter9.json. Results will be found in spatter.strongscaling/H100/xrage/asteroid/spatter9/ and Figures will be found in

Improved workflow adds support for visualization of patterns as well as initial support for codesign with SST

- Much more work is needed to port to other ModSim tools and to do validation and analysis of patterns!

Spatter page from https://lanl.github.io/benchmarks/

Questions?

[GS Patterns codebase]
https://github.com/lanl/gs_patterns

[Spatter]
https://github.com/hpcgarage/spatter
ATS-5 version
https://github.com/lanl/spatter

[Spatter Patterns]
https://github.com/hpcgarage/spatter-patterns

[Spatter Miranda Extension]
PR in progress!

Georgia Tech

Acknowledgements

This work is supported by the NSF Rogues Gallery testbed grant (CNS-2016701)

This research used resources provided by the Darwin testbed at Los Alamos National Laboratory (LANL) which is funded by the Computational Systems and Software Environments subprogram of LANL’s Advanced Simulation and Computing program (NNSA/DOE).

Kevin Sheridan, Galen Shipman, and Jered Dominguez-Trujillo acknowledge support by the National Nuclear Security Administration. Los Alamos National Laboratory is operated by Triad National Security, LLC for the U.S. Department of Energy under contract 89233218CNA000001; LA-UR-24-24856.

Georgia Tech.