Sandia National Laboratories

Exceptional service in the national interest

EVALUATING GATHER AND SCATTER PERFORMANCE ON CPUS AND GPUS

Or: The Spatter Benchmark Suite

Patrick Lavin, Jeffrey Young, Richard Vuduc, Jason Riedy, Aaron Vose, Daniel Ernst

Originally Presented at MEMSYS '20

Updated for MEMSYS '23

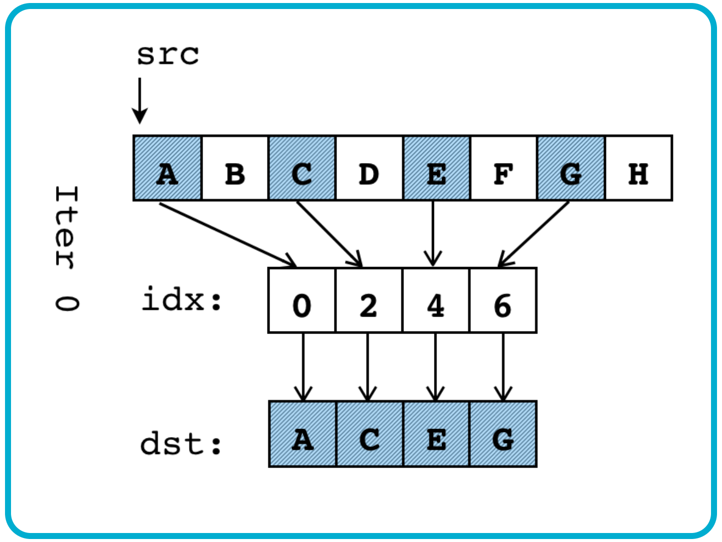
October XX, 2023, Alexandria, VA

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia LLC, a wholly owned subsidiary of Honeywell International Inc. for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

INTRODUCTION

PURPOSE

- Spatter's goal is to **represent a large class of irregular memory access patterns with a simple encoding** that can be machine generated or written by hand.
- For each input memory pattern, **Spatter reports the rate at which data was read or written**
- We compare these numbers to the STREAM-bandwidth to **understand how much of the available bandwidth is utilized** by an architecture when running different patterns



Gather Kernel

stride	Bandwidth(MB/s)
1	217498.729807
2	94488.415153
4	48114.707001
8	24105.660703

E.g. Intel 6430 (SPR)

USE CASES

- Evaluate the effect of vectorized instructions on available memory bandwidth
- Compare how different architectures handle sparse and irregular access
  - Measure how bandwidth utilization has improved across processor generations
  - Measure how CPUs and GPUs differ in their utilization
- Easily share application-derived memory access patterns

Compiler developers

Architects, system designers

System designers

IMPLEMENTATION DETAILS

IMPLEMENTATION

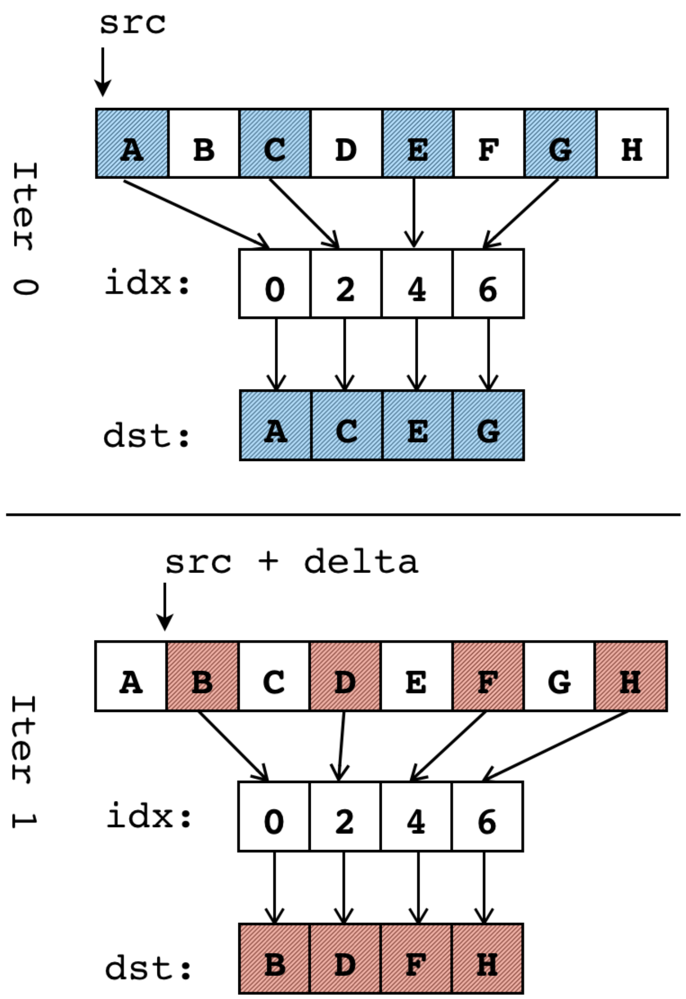
- Frontend
  - Specify inputs by hand or batch inputs with JSON

- Backends
  - CPU
    - Serial
    - OpenMP
  - GPU Backend
    - CUDA

- Tuning
  - OpenMP – work per thread
  - CUDA – block size, work per thread (in progress)

IMPLEMENTATION

- Kernels
  - Spatter has two kernels, one for Gather and one for Scatter
  - The Gather kernel reads into the same buffer on each loop to avoid generating writes
    - Vice versa for Scatter
- Pattern
  - A memory access pattern is specified by:
    - Gather or Scatter
    - Index buffer
    - Delta
    - Number of gathers/scatters to perform



Gather example:
Index = [0, 2, 4, 6]
Delta = 1
Count = 2

INPUT FILE EXAMPLE

```
# amg.json
[
  {
    ”delta": 1,
    "kernel": "Gather",
    "pattern": [0, 2, 4, 6],
    “count": 2
  },
  {
    "delta": 1,
    "kernel": "Scatter",
    “pattern”: [1, 1, 5, 5],
    "count": 1
  }
]
```

OUTPUT EXAMPLE

1. Read all patterns (kernel, idx, delta) from a JSON file

2. Determine maximum memory required and allocated data

3. For each pattern:
   1. Run the specified gather or scatter kernel N times, measuring the time it took (and optionally, PAPI counters)

4. Print out the timing and bandwidth for each pattern, and stats aggregating the performance of all patterns

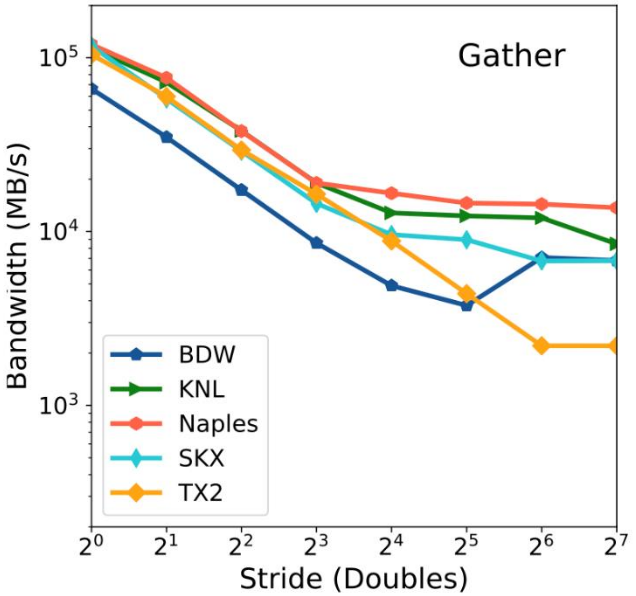
```
$ ./spatter -pFILE=amg.json

Running Spatter version 0.4
Compiler: icc ver. 19.0.0.20190206
Compiler Location: /opt/intel/bin/icc
Backend: OPENMP
Aggregate Results? YES

Run Configurations
[ {'kernel':'Gather', 'pattern':
[0,6,12,18,24,30,36,42,48,54,60,66,72,78,84,90],
  'delta':3, 'length':83333333,'threads':24},
  … (2 more omitted)]

config  time(s)      bw(MB/s)
0       0.05971      178631
1       0.184        173873
2       0.03706      172690

Min     25%     Med     75%     Max
172690  172690  173873  178631  178631
H.Mean       H.StdErr
175027       1469.4
```

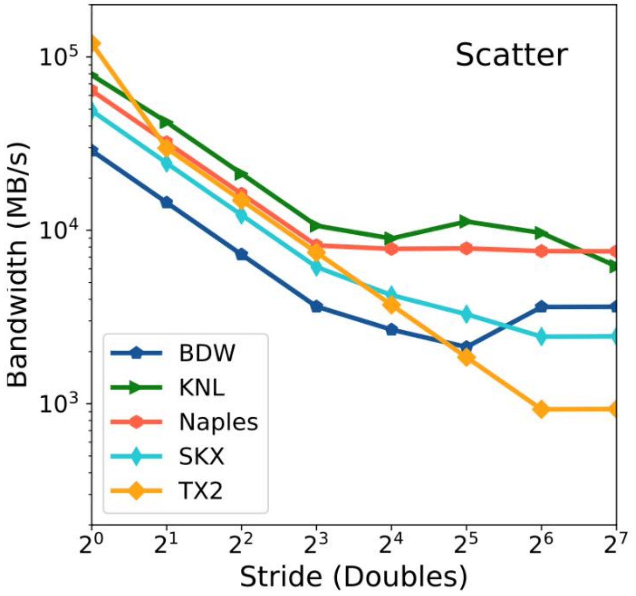
EXPERIMENTAL RESULTS

RESULTS

Uniform Stride

- We run gathers and scatters at power of 2 strides
- Utilization drops as we are not using all of the data being brought into cache
- Even past a stride of 8, where we would use one element for every cache line, bandwidth continues to drop on some architectures

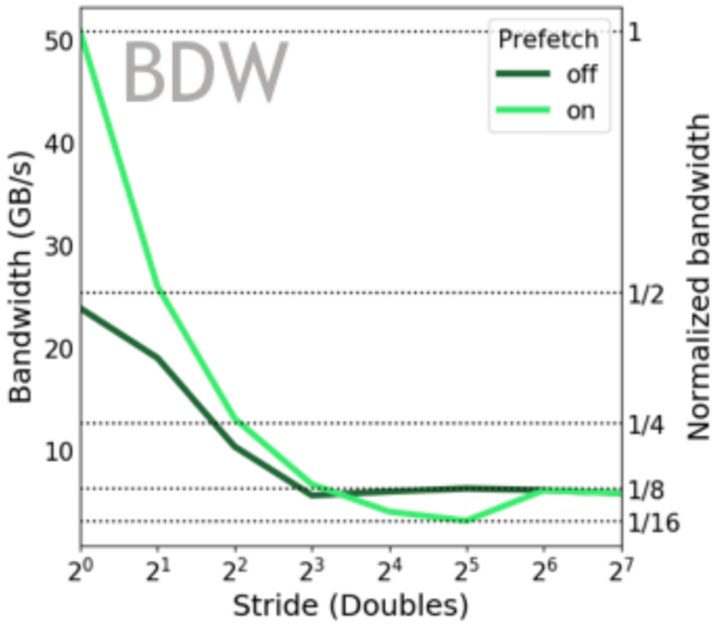


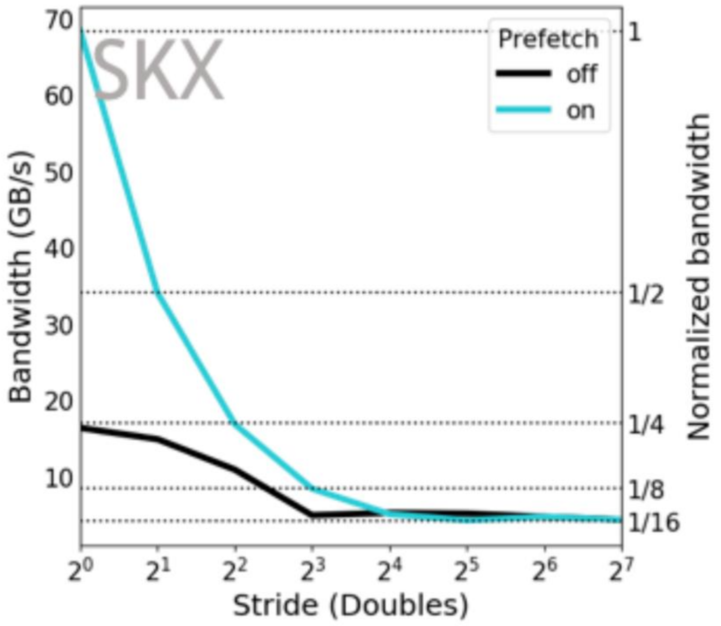


RESULTS

Uniform Stride – Broadwell vs Skylake

- Upon closer inspection, we see the prefetcher is responsible for the worse utilization on Broadwell
  - The next line is always fetches at strides lower than 128
- Skylake, however, always brings in two cache lines

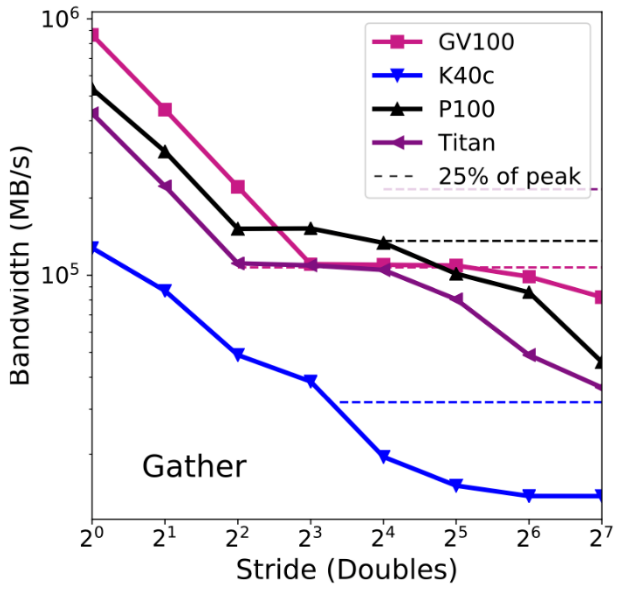


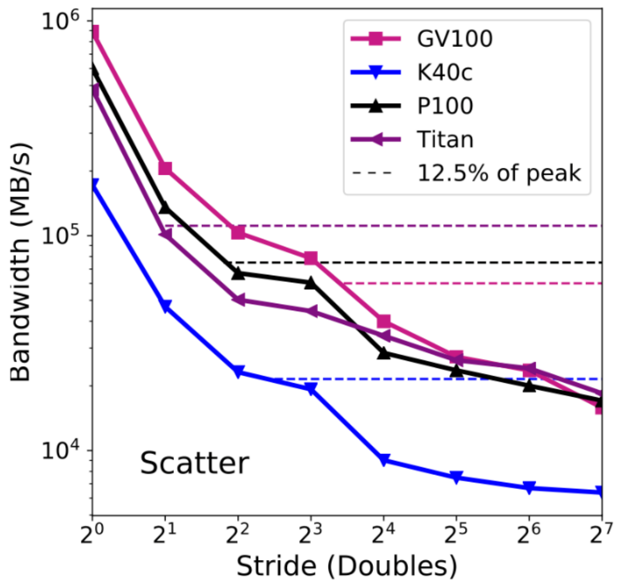


RESULTS

Uniform Stride – GPUs

- We see interesting improvements in GPU memory architecture beyond just higher bandwidth
- For intermediate strides, newer GPUs utilize a higher percentage of their bandwidth, particularly for Gather

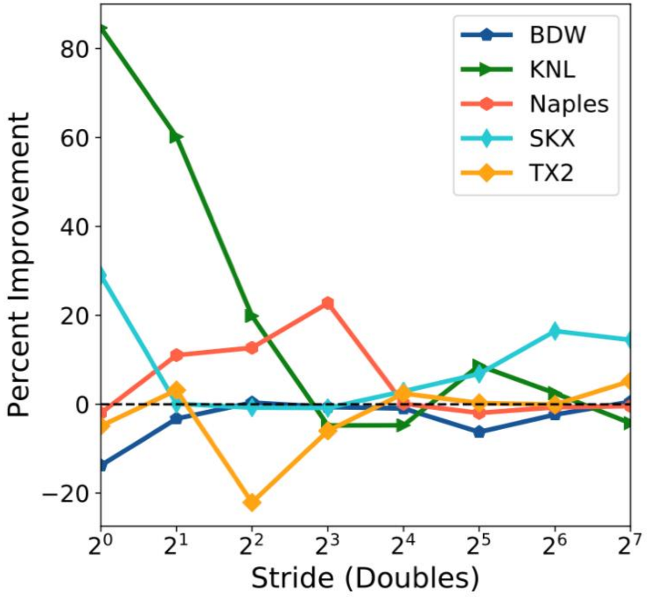




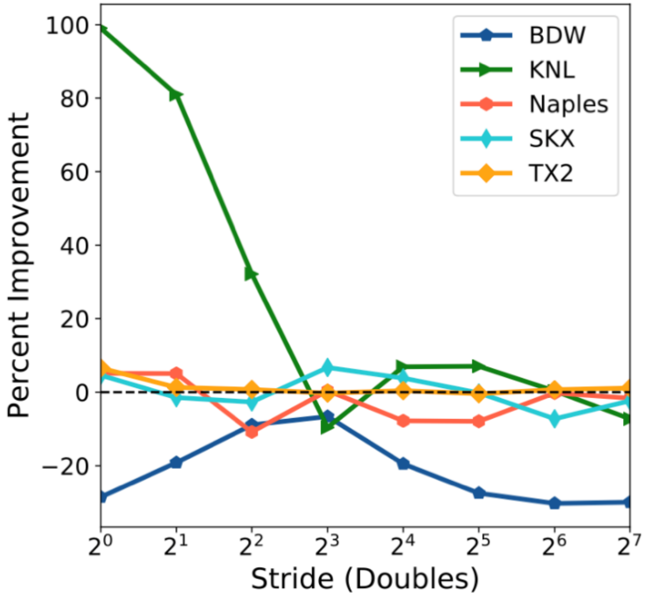
RESULTS

Uniform Stride - Serial vs OpenMP Backend

- Some architectures have more bandwidth available when using vectorized loads
- Surprisingly, some have less bandwidth available
- Broadwell had issues exposing the full bandwidth to scatter instructions



Gather



Scatter

Improvement of scatter and gather compared to scalar loads and stores

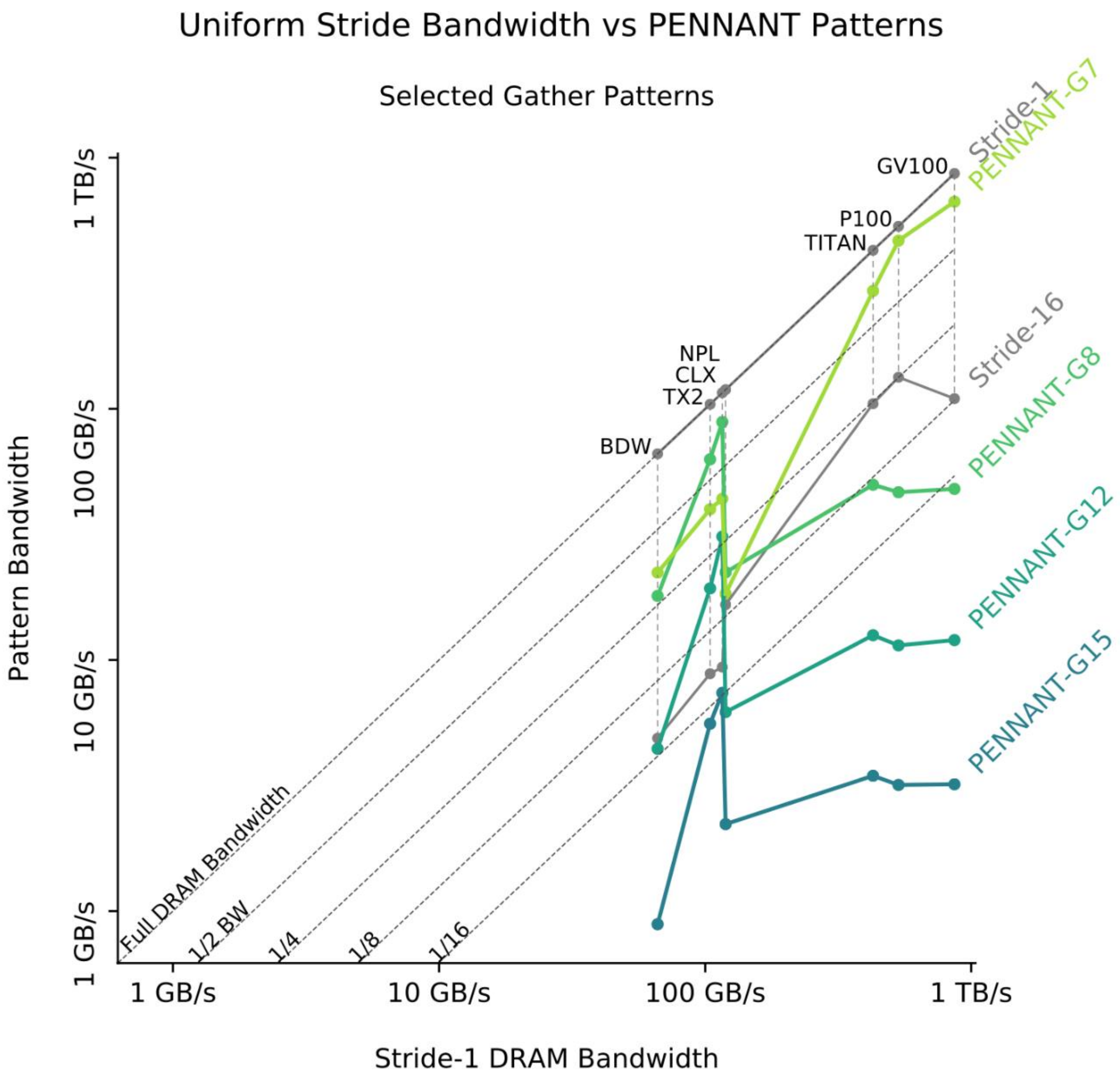
RESULTS

- We collected patterns from several mini-apps
- We compared the performance with STREAM to see if it was correlated
- For CPUs, we have a low R–value for the Pearson correlation coefficient, so they are not correlated
- For GPUs, there is some correlation
  - Pennant and Lulesh are still hard to predict

Platform	AMG (n=36) GB/s (H-Mean)	Nekbone (n=6) GB/s (H-Mean)	Lulesh GB/s (H-Mean)	PENNANT GB/s (H-Mean)	STREAM GB/s
BDW	123	121	20	6	43
SKX	328	309	12	35	96
CLX	315	287	14	41	94
Naples	140	323	3	11	97
TX2	270	247	232	28	241
KNL	201	190	19	4	249
R-value	0.15	-0.04	0.50	-.1	
K40c	108	99	88	14	193
TitanXP	496	320	175	21	443
P100	703	673	165	19	541
R-value	0.66	0.62	0.62	0.57	

RESULTS

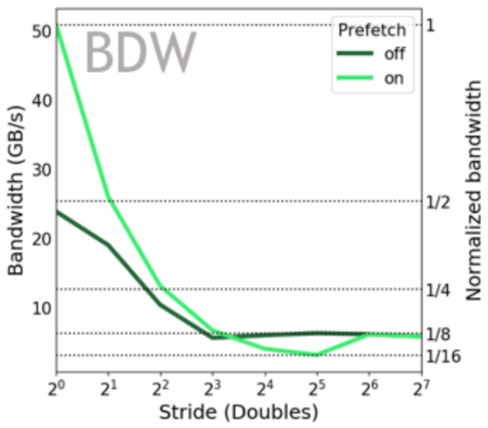
- We can rank systems in two ways
  - Absolute performance
  - Percent of bandwidth utilization
- This plot does both!
  - Pattern bandwidth as a function of maximum bandwidth
- A vertical (dashed) line represents a single system
- Trace a colored line to see how that pattern performs on different systems

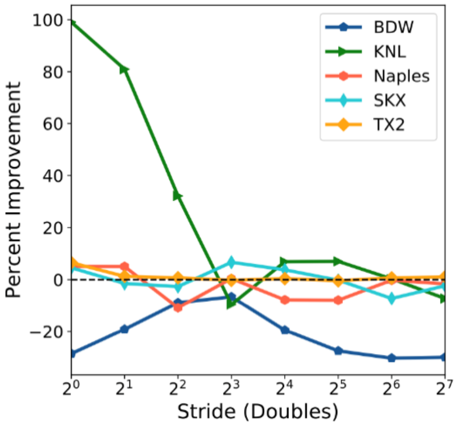


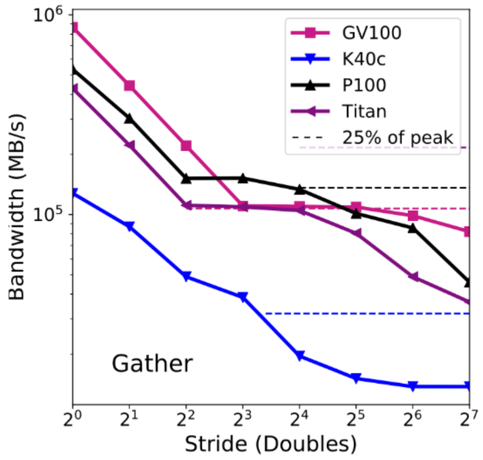
CONCLUSIONS

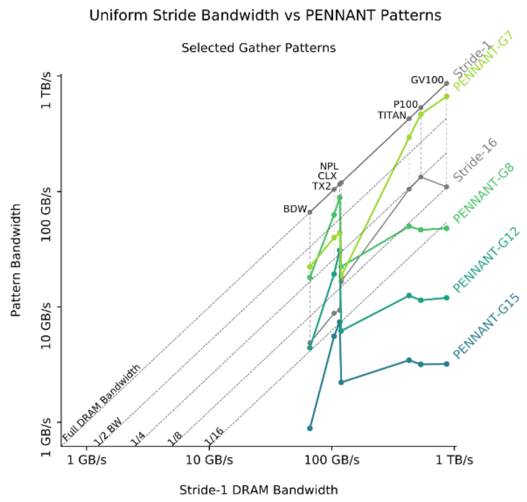
CONCLUSIONS

1. Spatter gives us a **compact representation** of a large class of memory access patterns
2. We can **compare memory systems with metrics beyond total bandwidth**, such as how different architectures handle irregular and sparse access
3. We can write patterns by hand to **investigate microarchitectural details** such as prefetcher behavior
4. System designers **easily share patterns** from real applications with vendors









NEW DEVELOPMENTS

NEW COLLABORATORS

Georgia Tech

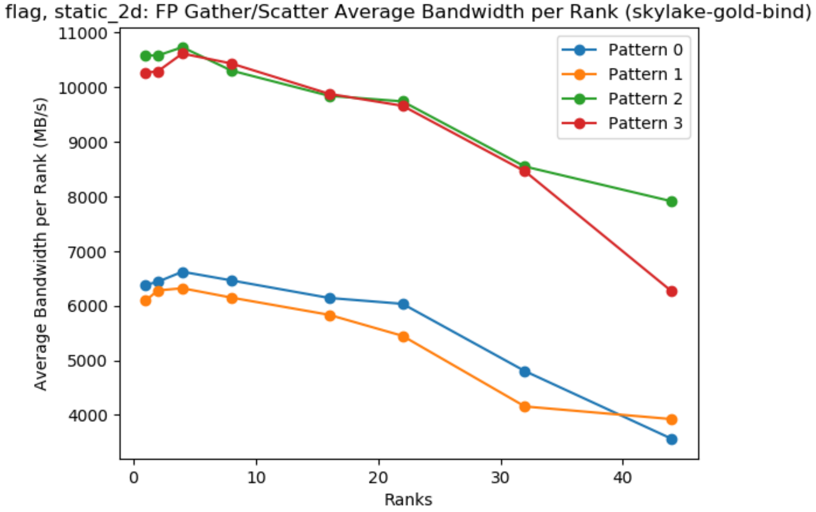
- James Wood
- Sudhanshu Agarwal
- Vincent Huang
- Julio Augustin Vaca Valverde

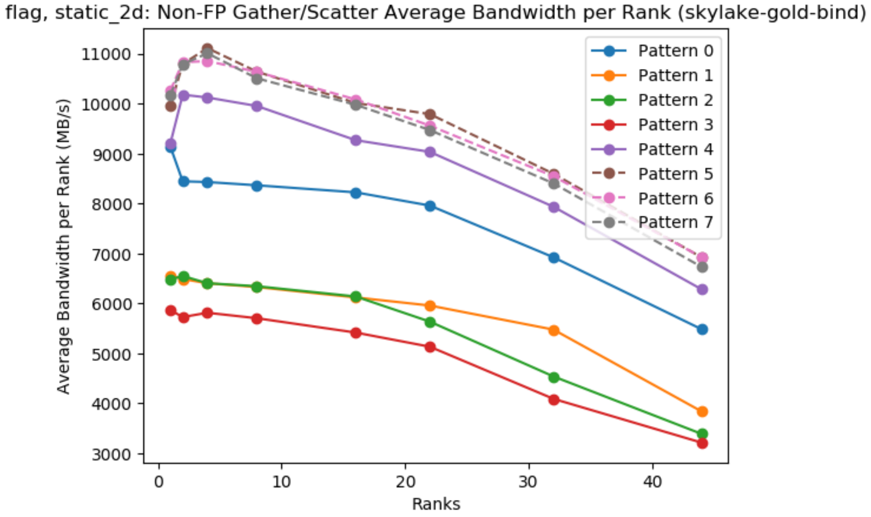
LANL

- Galen Shipman
- Jered Dominguez-Trujillo
- Kevin Sheridan

NEW FEATURES

- MPI
  - Weak scaling results[1]
- Multiple indirection
  - MultiGather, MultiScatter
    - `Target[i] = Source[idx1[idx2[i]]]`
- Concurrent Gather/Scatter
  - `Target[idx1[i] = Source[idx2[i]]`
- General usability updates
  - Binary pattern input
  - Improved testing
  - Standard suite of benchmarks
  - GettingStarted.ipynb
    - Uniform Stride and BW-BW plots
- Spatter-patterns repo (Coming soon!)
  - Share your patterns with other researchers





Patterns collected with gs_patterns (under release)
Image source: LANL[1]

[1] https://usrc.lanl.gov/emc3-project-deep-codesign-amt.php

ACKNOWLEDGEMENTS

This material is based upon work supported by the National Science Foundation under Award #1710371.

This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725

This research was supported in part by the Laboratory Directed Research and Development program at Sandia National Laboratories under contract DE-NA-0003525. Disclaimer: The views, opinions, and/or findings contained in this document are those solely of the author(s) and should not be interpreted as representing the official views or policies of any of its funding sources.

BACKUP SLIDES

Table 5: Listing of Patterns

Gather Pattern	Index	Delta	Type
PENNANT-G0	[2,484,482,0,4,486,484,2,6,488,486,4,8,490,488,6]	2	
PENNANT-G1	[0,2,484,482,2,4,486,484,4,6,488,486,6,8,490,488]	2	
PENNANT-G2	[0,4,8,12,16,20,24,28,32,36,40,44,48,52,56,60]	2	Stride-4
PENNANT-G3	[4,8,12,0,20,24,28,16,36,40,44,32,52,56,60,48]	2	
PENNANT-G4	[0,0,0,0,1,1,1,1,2,2,2,2,3,3,3,3]	4	Broadcast
PENNANT-G5	[4,8,12,0,20,24,28,16,36,40,44,32,52,56,60,48]	4	
PENNANT-G6	[482,0,2,484,484,2,4,486,486,4,6,488,488,6,8,490]	480	
PENNANT-G7	[482,0,2,484,484,2,4,486,486,4,6,488,488,6,8,490]	482	
PENNANT-G8	[2,0,0,0,2,0,0,0,2,0,0,0,2,0,0,0]	129608	
PENNANT-G9	[0,0,0,0,1,1,1,1,2,2,2,2,3,3,3,3]	388852	Broadcast
PENNANT-G10	[0,0,0,0,1,1,1,1,2,2,2,2,3,3,3,3]	388848	Broadcast
PENNANT-G11	[0,0,0,0,1,1,1,1,2,2,2,2,3,3,3,3]	388848	Broadcast
PENNANT-G12	[6,0,2,4,14,8,10,12,22,16,18,20,30,24,26,28]	518408	
PENNANT-G13	[6,0,2,4,14,8,10,12,22,16,18,20,30,24,26,28]	518408	
PENNANT-G14	[6,0,2,4,14,8,10,12,22,16,18,20,30,24,26,28]	1036816	
PENNANT-G15	[0,0,0,0,1,1,1,1,2,2,2,2,3,3,3,3]	1882384	Broadcast
LULESH-G0	[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]	1	Stride-1
LULESH-G1	[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]	8	Stride-1
LULESH-G2	[0,8,16,24,32,40,48,56,64,72,80,88,96,104,112,120]	1	Stride-8
LULESH-G3	[0,24,48,72,96,120,144,168,192,216,240,264,288,312,336,360]	8	Stride-24
LULESH-G4	[0,24,48,72,96,120,144,168,192,216,240,264,288,312,336,360]	4	Stride-24
LULESH-G5	[0,24,48,72,96,120,144,168,192,216,240,264,288,312,336,360]	1	Stride-24
LULESH-G6	[0,24,48,72,96,120,144,168,192,216,240,264,288,312,336,360]	8	Stride-24
LULESH-G7	[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]	41	Stride-1
NEKBONE-G0	[0,6,12,18,24,30,36,42,48,54,60,66,72,78,84,90]	3	Stride-6
NEKBONE-G1	[0,6,12,18,24,30,36,42,48,54,60,66,72,78,84,90]	8	Stride-6
NEKBONE-G2	[0,6,12,18,24,30,36,42,48,54,60,66,72,78,84,90]	8	Stride-6
AMG-G0	[1333,0,1,36,37,72,73,1296,1297,1332,1368,1369,2592,2593,2628,2629]	1	Mostly Stride-1
AMG-G1	[1333,0,1,2,36,37,38,72,73,74,1296,1297,1298,1332,1334,1368]	1	Mostly Stride-1
Scatter Pattern	Index	Delta	Type
PENNANT-S0	[0,4,8,12,16,20,24,28,32,36,40,44,48,52,56,60]	1	Stride-4
LULESH-S0	[0,8,16,24,32,40,48,56,64,72,80,88,96,104,112,120]	1	Stride-8
LULESH-S1	[0,24,48,72,96,120,144,168,192,216,240,264,288,312,336,360]	8	Stride-24
LULESH-S2	[0,24,48,72,96,120,144,168,192,216,240,264,288,312,336,360]	1	Stride-24
LULESH-S3	[0,24,48,72,96,120,144,168,192,216,240,264,288,312,336,360]	0	Stride-24

PLATFORMS

System description	Abbreviation	System Type	STREAM BW (MB/s)	Power (W)	Threads/Backend
Broadwell	BDW	12-core Intel CPU	37,164	105	12 threads, OMP, OCL
Cavium ThunderX2	ThunderX2	28-core ARM CPU	120,000	175	112 threads, OMP
IBM Power8	Power8	8-core IBM CPU	25,389	190	64 threads, OMP
Kepler K40c	K40c	NVIDIA GPU	193,855	235	CUDA, OCL
Knight's Landing	KNL	Intel Xeon Phi	64,060	215	128 threads, OMP
Pascal P100	P100	NVIDIA GPU	541,835	250	CUDA, OCL
Quadro GV100	GV100	NVIDIA GPU	591,350	300	CUDA
Sandy Bridge	SNB	4-core Intel CPU	17,925	80	4 threads, OMP, OCL
Titan XP	Titan XP	NVIDIA GPU	443,533	250	CUDA, OCL