Spatter: A Framework for Measuring Hardware Gather-Scatter Support

Patrick Lavin, Jeffrey Young, Jason Riedy, Rich Vuduc, Aaron Vose, Dan Ernst

Georgia Tech College of Computing
Computational Science and Engineering
hpcgarage
CRAY

Purpose

- Modern processors implement Indexed Vector Load and Store instructions, better known as Gather/Scatter (G/S) instructions.
    - AVX512, SVE
- Spatter aims to help application developers, compiler writers, and architects assess how well compilers and hardware support G/S.

```
Gather (indexed read):
for i in 0..vector_len:
   reg[i] = mem[idx[i]]

Scatter (indexed write):
for i in 0..vector_len:
   mem[idx[i]] = reg[i]
```

G/S Examples

- We can group SVE G/S instructions in traces based on the index buffer and the delta from the previous access
- By examining the index buffers, we can classify the types of patterns we see

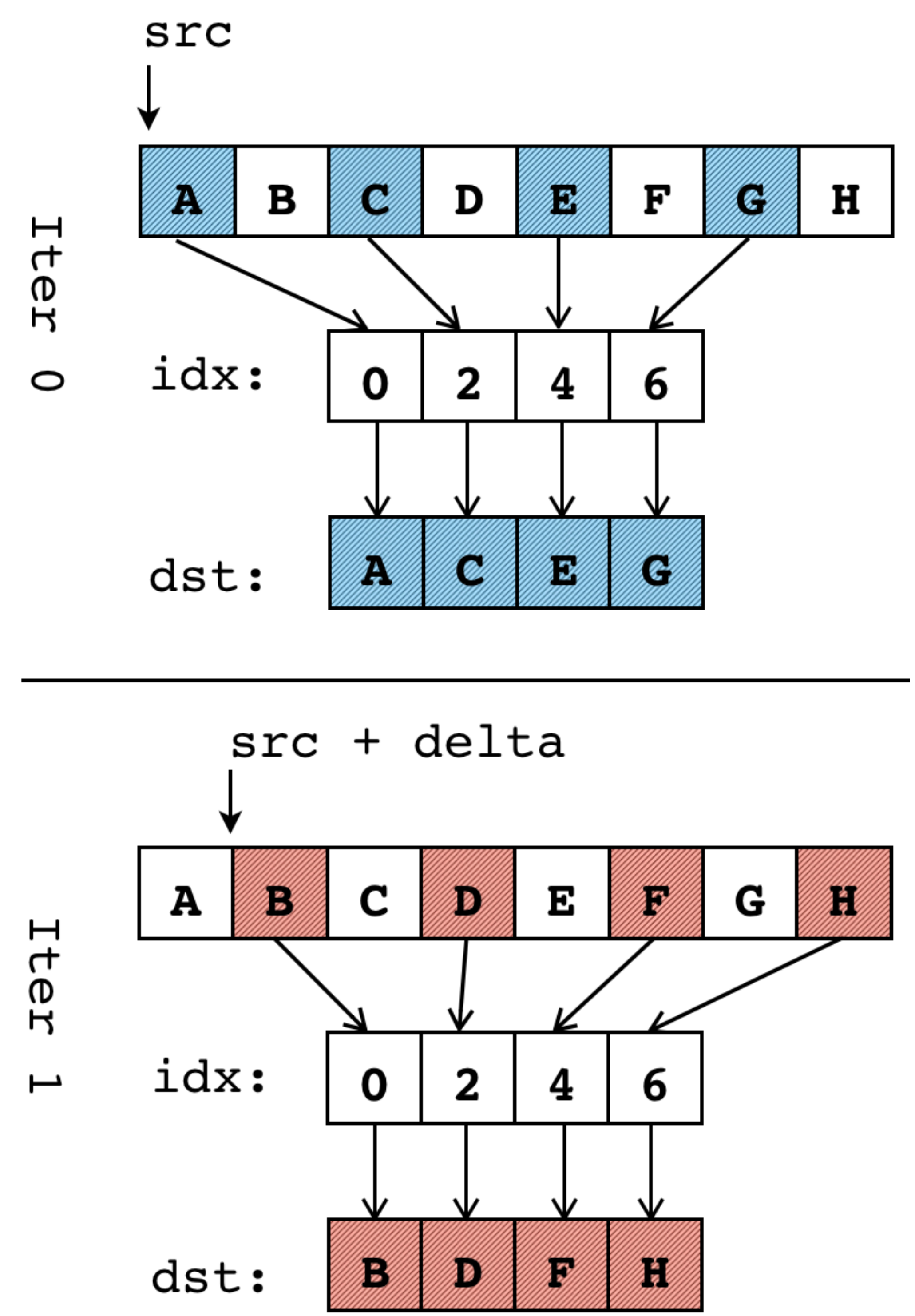
Pattern	Example	Apps
Uniform Stride	[0,4,8,12,16,20,24,28]	Nekbone, Lulesh, Pennant
Mostly Stride-1	[0,1,2,36,37,38,72,73,74]	AMG
Broadcast	[0,0,0,0,4,4,4,4]	Pennant

Spatter Kernels

- The basis of Spatter are gather and scatter kernels

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

Scatter kernel:
for i in 0..N:
   scatter(dst + delta*i, idx, reg)
```

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



Features

- Backends - Serial, OpenMP, CUDA (and SVE soon)
- Built-in common patterns (Uniform Stride, Mostly Stride-1, Laplacian stencil)
- Performance tuning
  - OpenMP Work per thread
  - CUDA block size
  - Pattern length
- Advanced scripting with JSON

Using Spatter

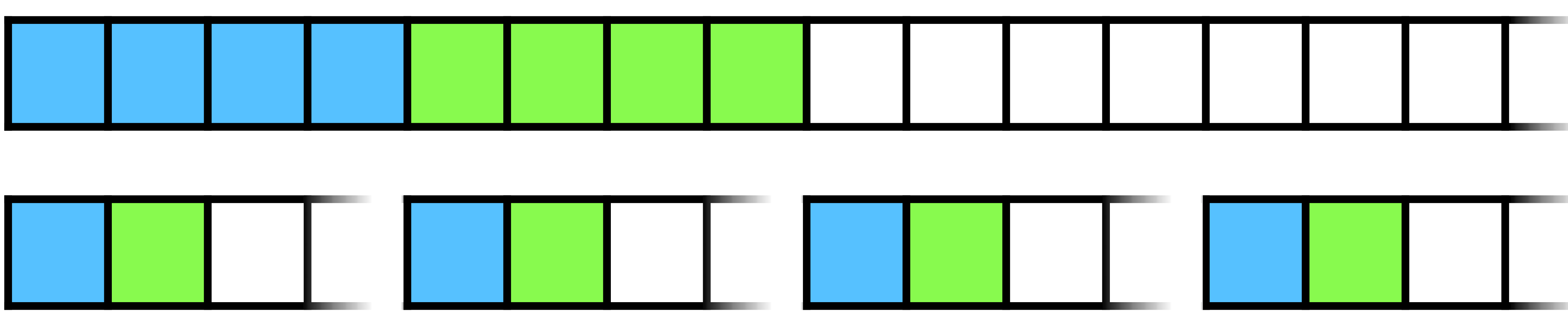
1. **Basic Usage** - Specify a pattern on the command line
2. **Advanced Usage** - specify a JSON file containing a collection of patterns

Matrix Transpose

- “Will some operation be slow if I don’t transpose the matrix first?”
- E.g. Performing a portion of an FFT across rows, when the matrix is stored in column order

```
L=$((2**24))

Transpose First:
./spatter -pUNIFORM:4:1 -d4 -l$L
⇒ 29258.5 MB/s

No Transpose:
./spatter -pUNIFORM:4:$L -d1 -l$L
⇒ 26898.5 MB/s
```

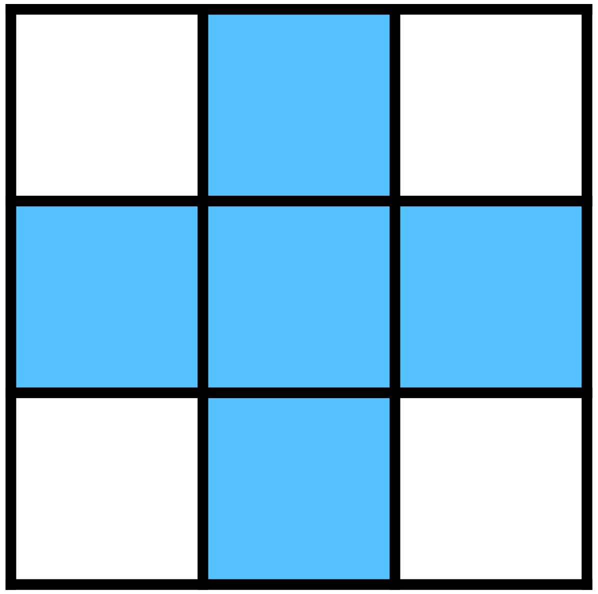
*Xeon E5-2650 v4, Skylake, 12 threads



Stencil Patterns (New for SC!)

- Spatter supports several built-in, parametrized stencils
- E.g. LAPLACIAN:2:1:100 represents this stencil on a problem of size 100x100
- Spatter will turn this into the following pattern, with a delta of one

```
[   0, 99, 100, 101, 200]
          ===
[-100, -1,   0,   1, 100]
```



```
./spatter  -pLAPLACIAN:2:1:100 -l$((2**25))
⇒ 67862.4 MB/s
```


Georgia Institute of Technology

Advanced Usage: JSON Files

- Spatter is able to optimize memory allocation and provide summarized output if all of your tests are specified in a single JSON file

ustride_simple.json

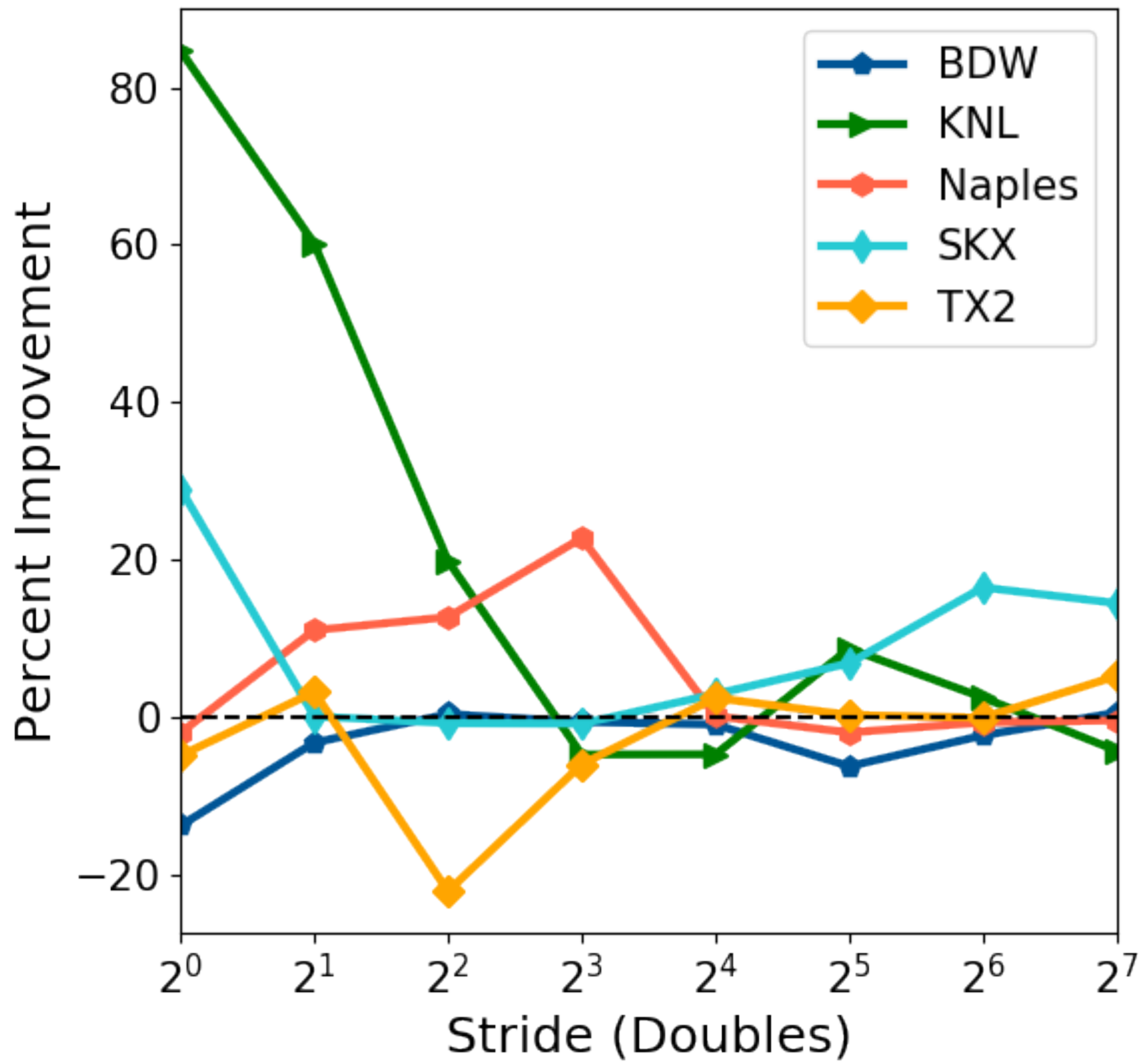
```
[ {'pattern':'UNIFORM:8:1',
   'delta':8, 'count':10000},
  {'pattern':'UNIFORM:8:2',
   'delta':16, 'count':10000},
  {'pattern':'UNIFORM:8:4',
   'delta':32, 'count':10000},
…
]
```

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

Run Configurations
[  {'name':'UNIFORM:8:1', 'delta':8,…},
   {'name':'UNIFORM:8:2', 'delta':16,…},
   {'name':'UNIFORM:8:4', 'delta':32, …},
…
]

config  time(s)      bw(MB/s)
0       0.205        78033.8
1       0.1622       49325
2       0.1705       23465.7
```


Georgia Institute of Technology

Vector vs Scalar Loads

- In previous Intel hardware generations, there was no benefit to using G/S instructions.
- More modern hardware, however, shows speedup when using these instructions for uniform stride loads.
- We provide a serial backend for this purpose

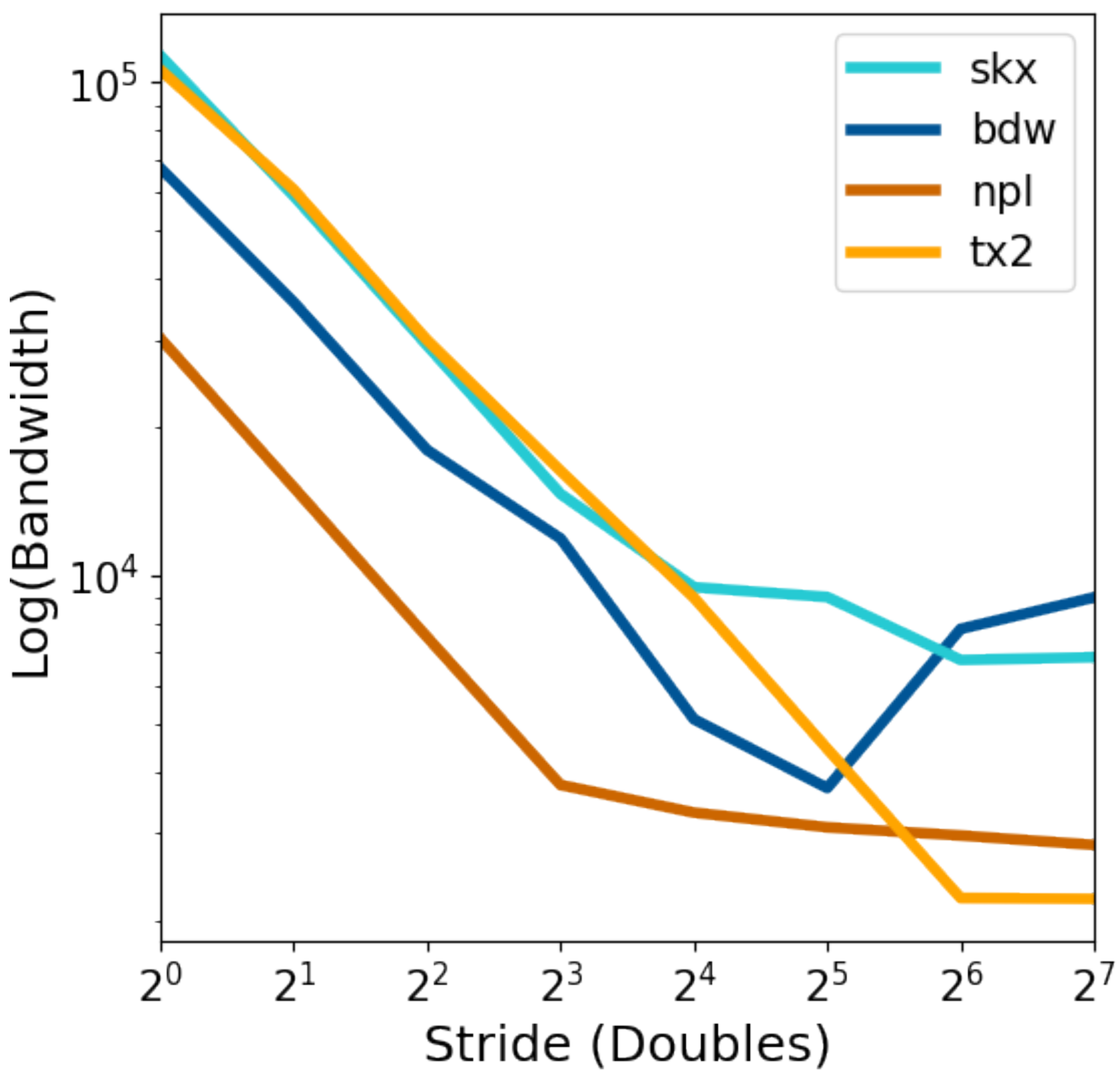


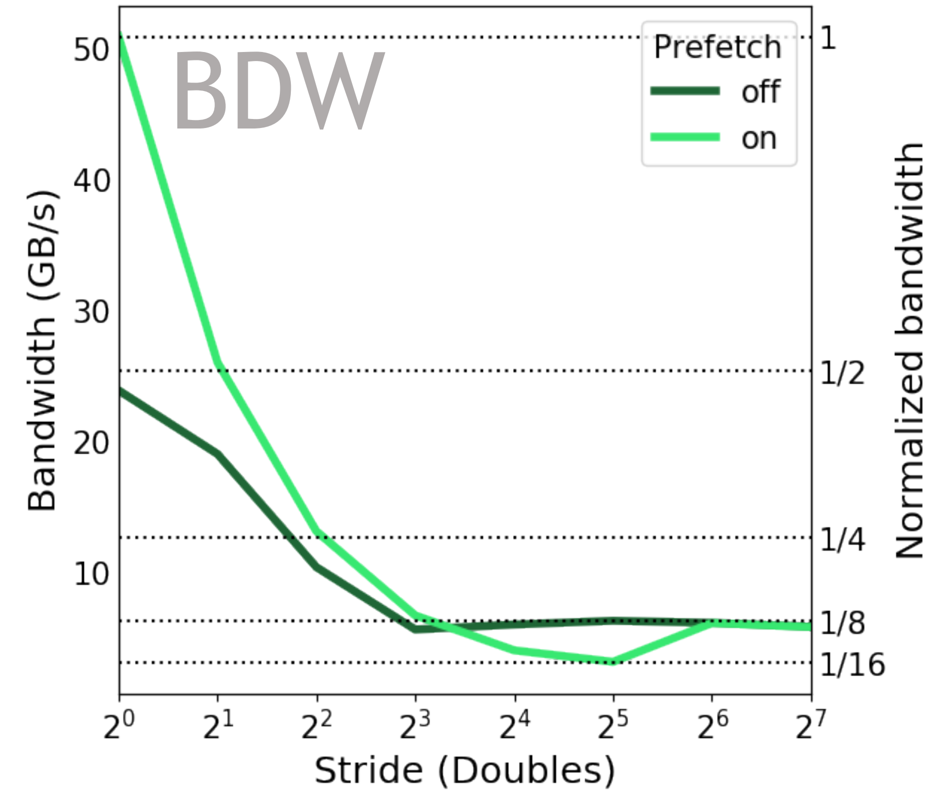

```
./spatter -pFILE=ustride_simple.json -bOPENMP
./spatter -pFILE=ustride_simple.json -bSERIAL
```

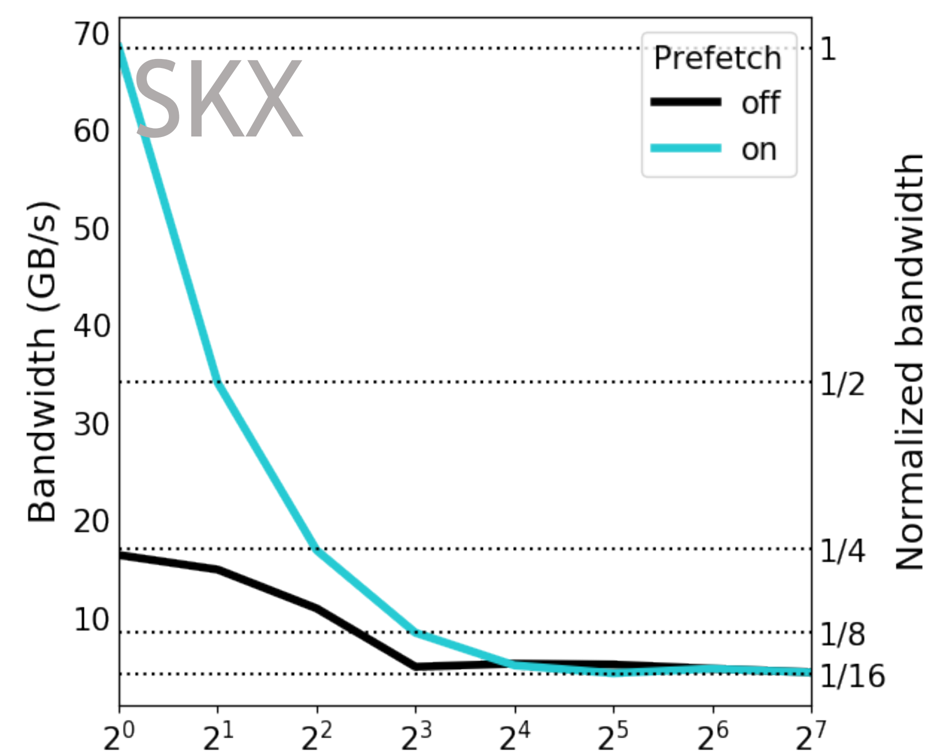

Georgia Institute of Technology

Cache Implementation Exploration: Prefetching

- Why does Broadwell bandwidth improve at large strides and why does it out-perform Skylake?







GPU: G/S Available Bandwidth Improvement

- GPU gather/scatter performance has improved in recent generations.
  - Full main memory bandwidth now available when doing gather/scatter operations (assuming a stream-like access pattern)

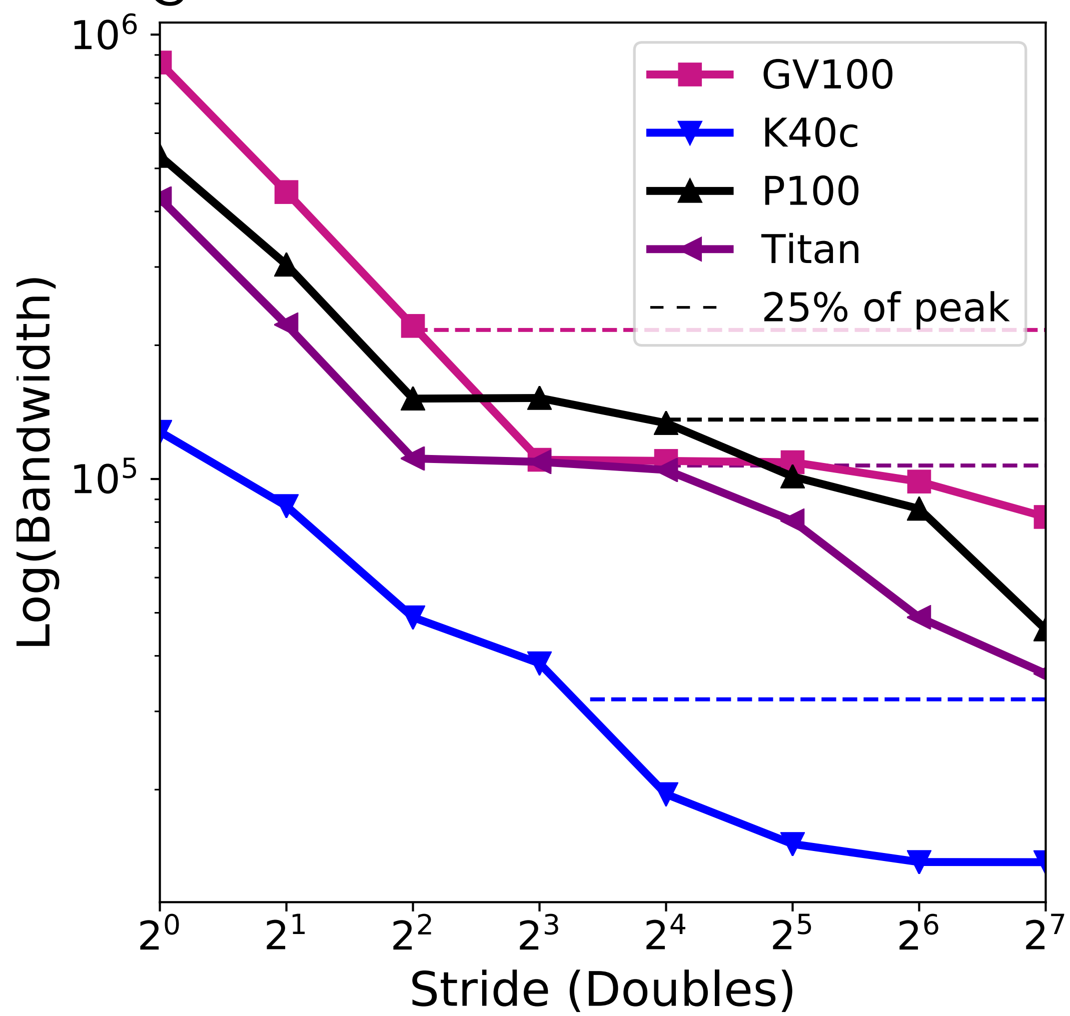
	K40c	Titan Xp	P100	GV100
Reported BW	**288**	**547.7**	**732**	**870**
Spatter Gather	**145**	**427**	**578**	**877**
Spatter Scatter	**196**	**480**	**600**	**896**

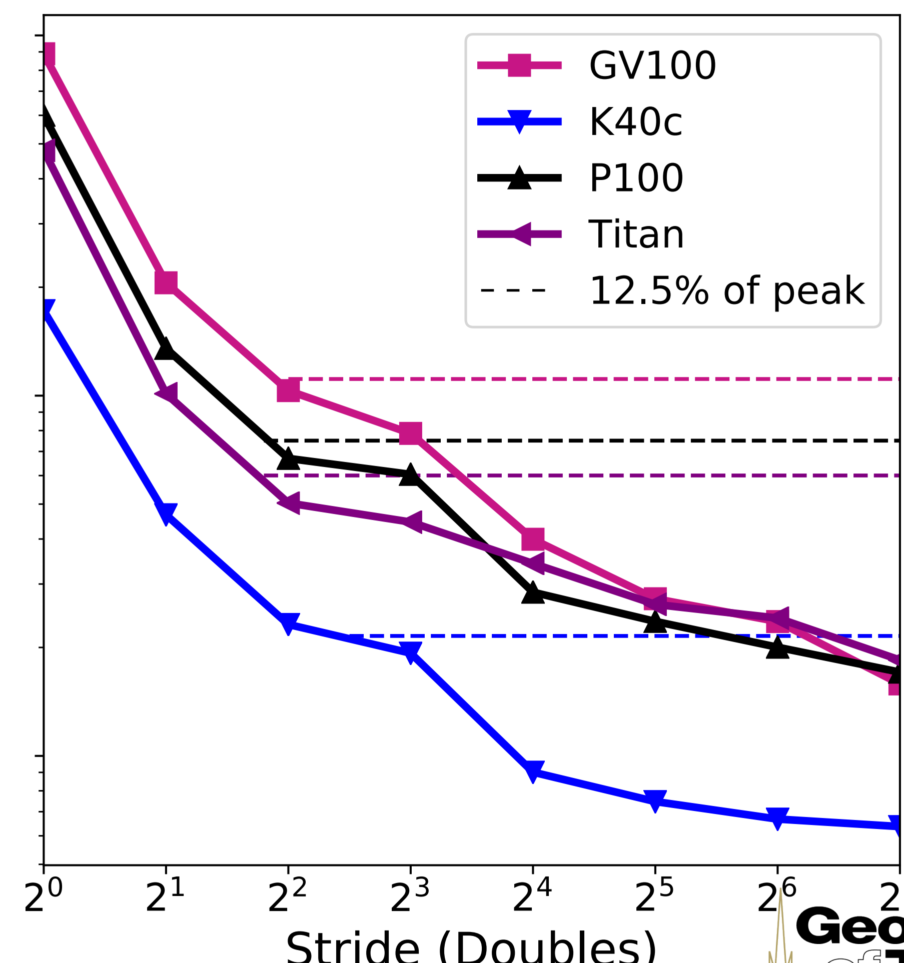

```
./spatter -pUNIFORM:256:1 -d256 -l$((2**18))
```


Georgia Institute of Technology

GPU: Uniform Stride Access Improvements

- The ability of GPU's to maintain a high percentage of peak as access stride has increase, has improved over recent generations
  - The P100 and Titan are noticeably "flatter" at intermediate strides than the K40 for gather
  - The GV100 does not flatten out until stride 8, a divergence from previous generations





Application Specific Patterns

- We have collected patterns from 4 DoE mini-apps and placed them into JSON files
- Spatter bandwidths do not correlate well with STREAM for these patterns (R value close to 1)
- GPU patterns correlate reasonably well

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

	AMG	NEKBONE	LULESH	Pennant	STREAM
BDW	123	121	20	6	43
SKL	328	308	12	35	96
CSL	234	215	9	28	94
Naples	140	323	3	11	97
TX2	270	247	232	28	241
KNL	201	190	19	4	249
R value	0.26	0.03	0.5	-0.04	
K40c	108	99	88	14	193
TitanXp	496	320	175	21	443
P100	703	673	165	19	541
GV100	1368	1395	368	20	870
R value	0.75	0.73	0.72	0.52	

Performance in GB/s

What's Next?

- Integrate our ARM SVE backend, add an AVX512 backend
    - Accurate L1/L2 Measurements
    - A64FX's Combined Gather
- Open source G/S trace generation from applications
- More kernels to express more memory access patterns, such as GUPS and Pointer Chase
- Create a standard set of configurations

More Info

- Spatter.io
  - Documentation, links to code and data repos, and a link to our ArXiv pre-print
- ArXiv Pre-print
  - Spatter: A Benchmark Suite for Evaluating Sparse Access Patterns
  - https://arxiv.org/abs/1811.03743
- ACM Student Research Competition Poster 27
  - 5:15-7:00 today
- Code
  - https://github.com/hpcgarage/spatter

Acknowledgements

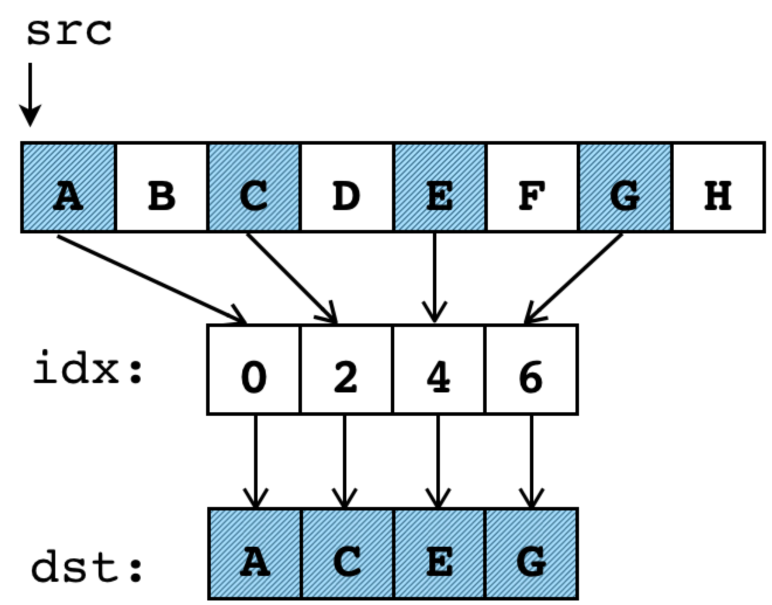
This material is based upon work supported by the National Science Foundation under Award #1710371.

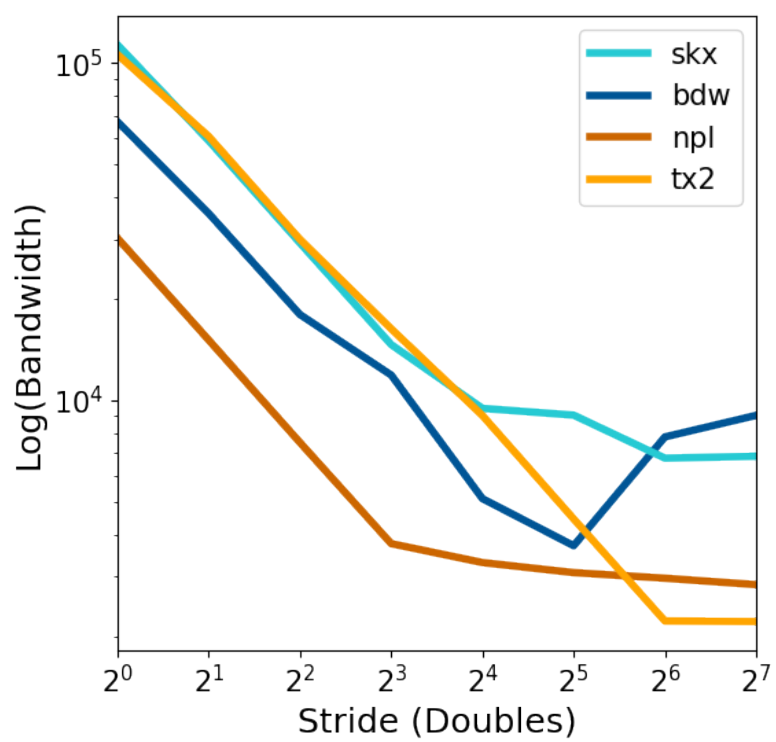
This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725

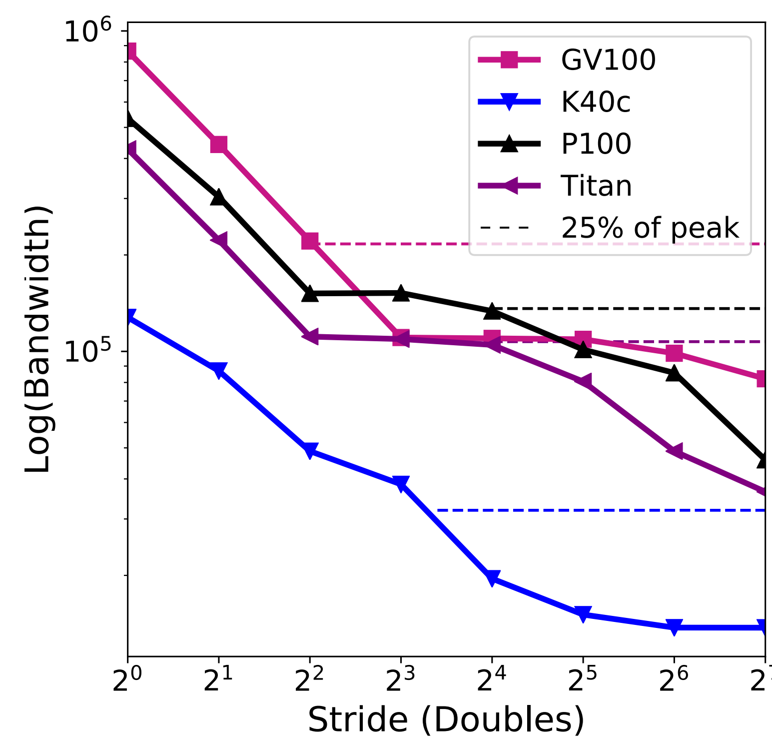
This research was supported in part by the Laboratory Directed Research and Development program at Sandia National Laboratories under contract DE-NA-0003525. *Disclaimer: The views, opinions, and/or findings contained in this document are those solely of the author(s) and should not be interpreted as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

Spatter: A Framework for Measuring Hardware Gather-Scatter Support

Example
[0,4,8,12,16,20,24,28]
[0,1,2,36,37,38,72,73,74]
[0,0,0,0,4,4,4,4]





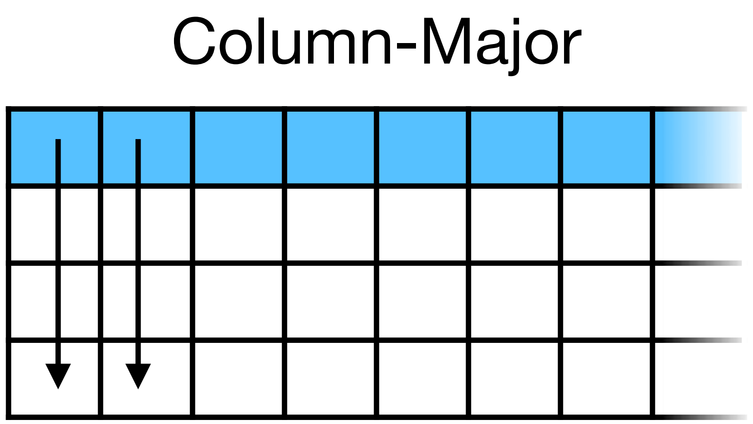


Backup Slides

Examples - Vectorization

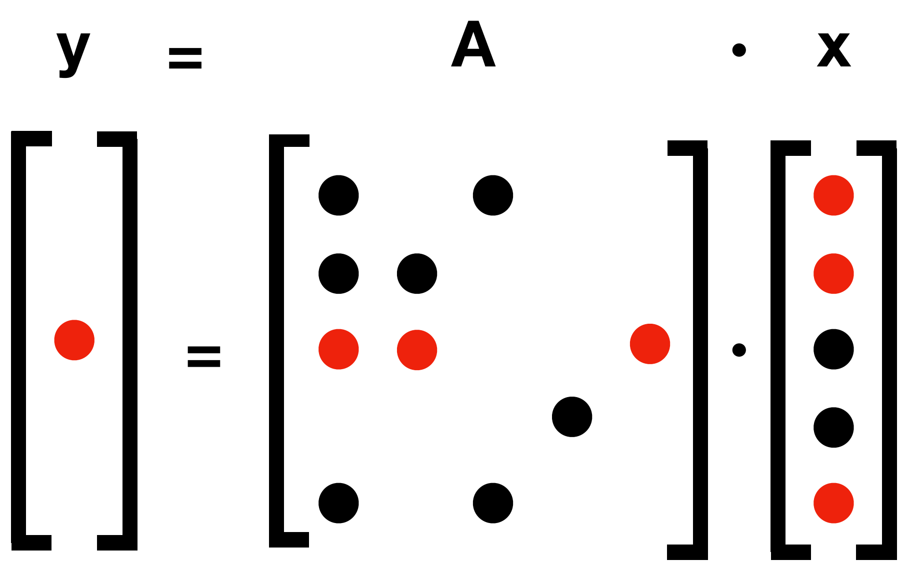
- Some forms of vectorization will naturally lead to Gather/ Scatter operations

```
Algo: SUM COLUMNS
for (j in range(N)):
    for (i in range(4)):
        out[j] += data[i,j]
```




```
Algo: Vectorized SUM COLUMNS
for (j = 0; j < N; j+=8):
    temp = 0;                       // vector of length 8
    for (i in range(4)):
        temp += gather(j+i, [0,4,8,12,16,20,24,28])
    out[j:j+8] = temp
```

Examples - CSR SpMV



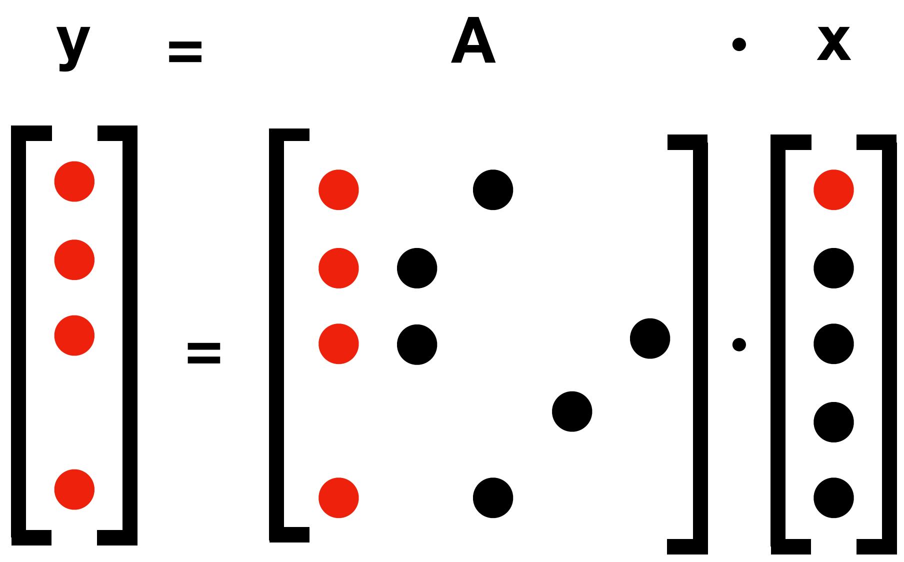
- Gathers can also represent indirection
- Gather elements of x, then do a dot product with data in A.

```
for (i in range(nrows)):
    indices ← row[i] : row[i+1]
    gather(tmp, x, col[indices])
    y[i] = dot_prod(val[indices], tmp)
```


Georgia Institute of Technology

Examples - CSC SpMV

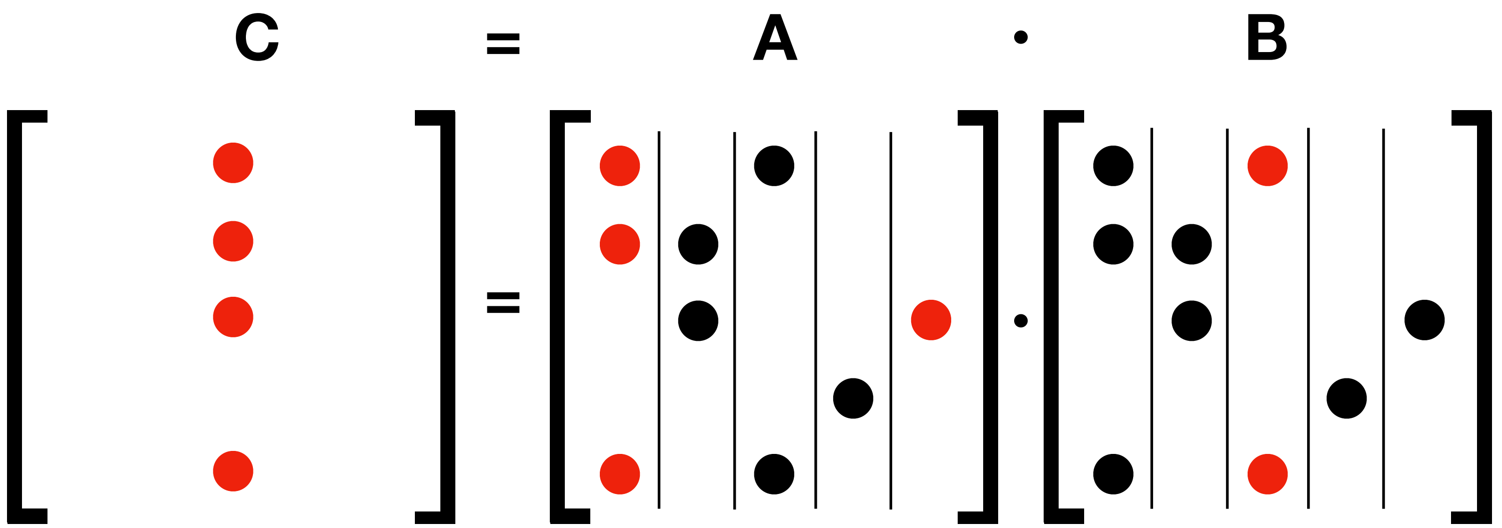
- Scale some a column of A by the value in x, then scatter-accumulate into y.

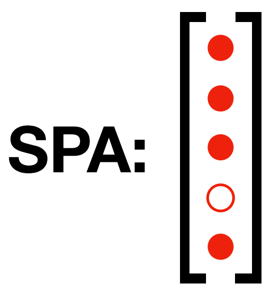



```
for (i in range(ncols)):
    indices ← col[i] : col[i+1]
    tmp ← vector_scale(val[indices], x[i])
    scatter_accum(y, row[indices], tmp)
```

Examples - SpGEMM

- Scatter-accumulate columns of A corresponding to non-zero entries in a column of B into a *dense* SPA buffer. Gather SPA into C.



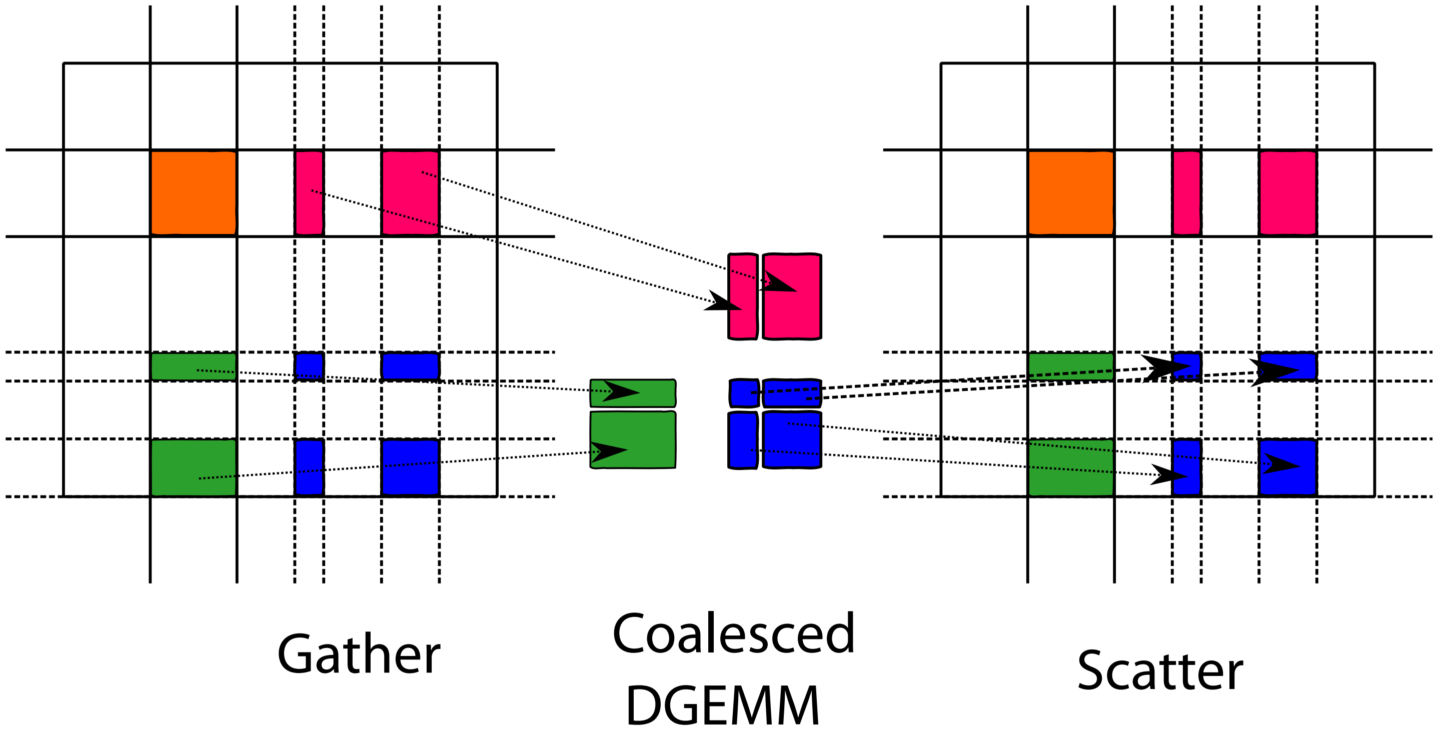



```
for (j in range(ncols) :
    SPA = 0 //dense accumulation buffer
    for non-zero B(k,j) :
        scatter_accum(SPA, A(:,k)*B(k,j))
    gather(C.val, SPA)
    gather(C.row, which(SPA))
    C.col[j+1] = C.col[j] + nnz(SPA)
```

Algorithm from Buluç and Gilbert: *Parallel Sparse Matrix-Matrix Multiplication and Indexing: Implementation and Experiments*
https://doi.org/10.1137/110848244

Example - SuperLU

- SuperLU spends a large portion its runtime on just scattering data



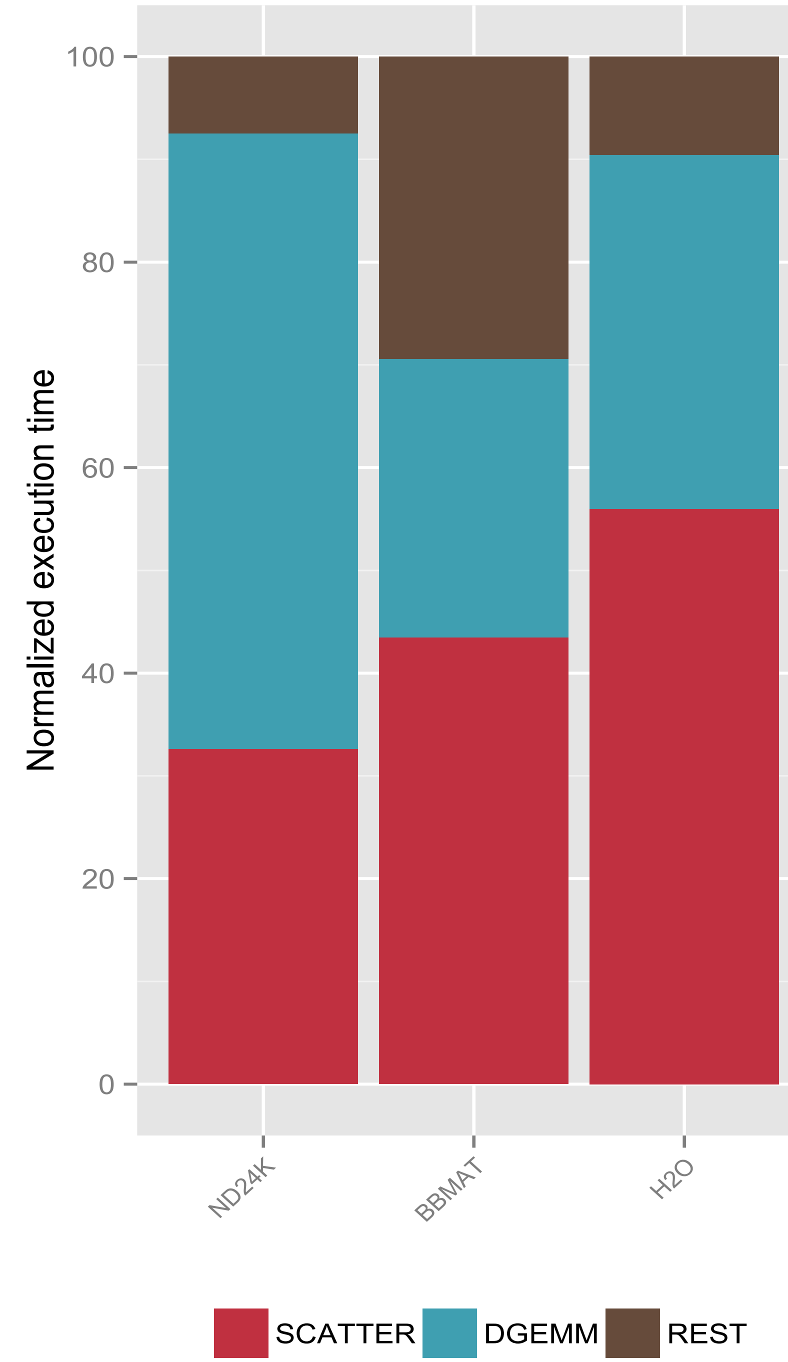


Chart credits: Piyush Sao

Georgia Institute of Technology

Platforms

TABLE III: Experimental Parameters and Systems (OMP Denotes OpenMP, and OCL Denotes OpenCL).

System description	Abbreviation	System Type	STREAM (MB/s)	Threads, Backends
Knight's Landing (cache mode)	KNL	Intel Xeon Phi	249,313	272 threads, OMP
Broadwell	BDW	32-core Intel CPU (E5-2695 v4)	43,885	16 threads, OMP
Skylake	SKX	32-core Intel CPU (Platinum 8160)	97,163	16 threads, OMP
Cascade Lake	CSX	24-core Intel CPU (Platinum 8260L)	66,661	12 threads, OMP
ThunderX2	TX2	28-core ARM CPU	120,000	112 threads, OMP
Kepler K40c	K40c	NVIDIA GPU	193,855	CUDA
Titan XP	Titan XP	NVIDIA GPU	443,533	CUDA
Pascal P100	P100	NVIDIA GPU	541,835	CUDA
Broadwell (ICC)	BDW2	12-core CPU (E5-2650)	85,750	24 threads, OMP
Skylake (ICC)	SKX2	6-core CPU (Gold 6128)	66,661	12 threads, OMP

Application Patterns

TABLE I: High-Level Characterization of Application G/S Patterns.

Application (Extracted Patterns) `Selected Kernels`	Gathers	Scatters	G/S MB (%)
AMG (partially stride-1)			
`hypre_CSRMatrixMatvecOutOfPlace`	1,696,875	0	217 (17.8)
LULESH (fixed-stride)			
`IntegrateStressForElems`	828,168	382,656	155 (22.4)
`InitStressTermsForElems`	1,121,844	1,153,827	291 (67.6)
Nekbone (fixed-stride)			
`ax_e`	2,948,940	0	377 (33.3)
PENNANT (fixed-stride, partially stride-0, complex strides)			
`Hydro::doCycle`	728,814	0	93 (13.9)
`Mesh::calcSurfVecs`	324,064	0	41 (39.5)
`QCS::setForce`	891,066	0	114 (45.5)
`QCS::setQCnForce`	1,214,318	323,800	197 (64.5)

TABLE II: Details for Selected Applications and Kernels Used for G/S Pattern Extraction.

Application – Version	Problem Size / Changes	Kernel Notes
AMG – `github.com/LLNL/AMG` commit `09fe8a7`	Arguments `-problem 1 -n 36 36 36 -P 4 4 4`, also `mg_max_iter` in `amg.c` set to 5 to limit iterations.	Entirety of each of the functions listed in Table I.
LULESH – 2.0.3	Arguments `-i 2 -s 40`, also modifications to vectorize the outer loop of the first loop-nest in `IntegrateStressForElems`.	The first loop-nest in `IntegrateStressForElems`. Arrays `[xyz]_local[8]` as well as `B[3][8]` give stride-8 and stride-24. Also, the entirety of the `InitStressTermsForElems` function.
Nekbone – 2.3.5	Set `ldim = 3, ifbrick = true, iel0 = 32, ielN = 32, nx0 = 16, nxN = 16, stride = 1`, internal `np` and `nelt` distribution. Also, `niter` in `driver.f` set to 30 to limit CG iterations.	First loop in `ax` (essentially a wrapped call to `ax_e`) contains the observed stride-6.
PENNANT – 0.9	Config file `sedovflat.pnt` with `meshparams 1920 2160 1.0 1.125` and `cstop 5`.	Entirety of each of the functions listed in Table I.