Using the Spatter Benchmark Suite to Evaluate SVE Support for Gather/Scatter

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

Purpose

- Modern processors implement SIMD Index Load and Store instructions, better known as Gather/Scatter (G/S) instructions.
  - AVX2, AVX512, SVE (soon)
- Spatter aims to help application developers, compiler writers, and architects asses 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

Examples - Vectorization

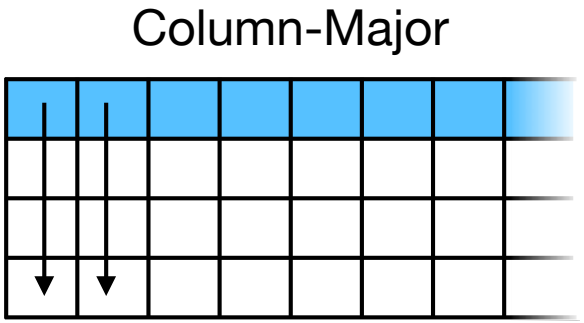
- Some forms of vectorization will naturally lead to Gather/ Scatter operations

Algo: SUM COLUMNS

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

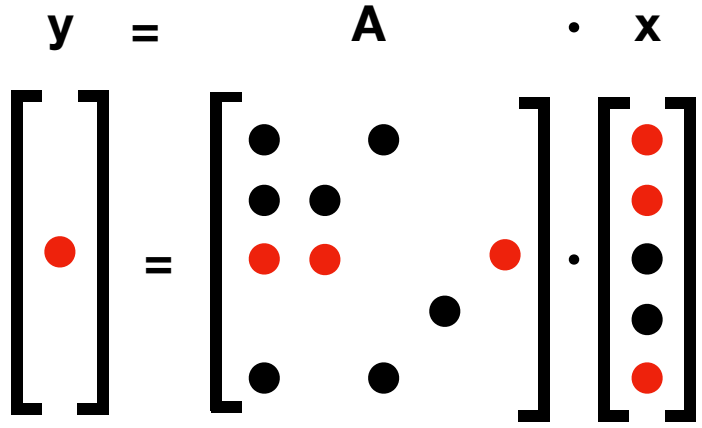
Algo: Vectorized SUM COLUMNS

```
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)
```

Examples: Real SVE Traces

- 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

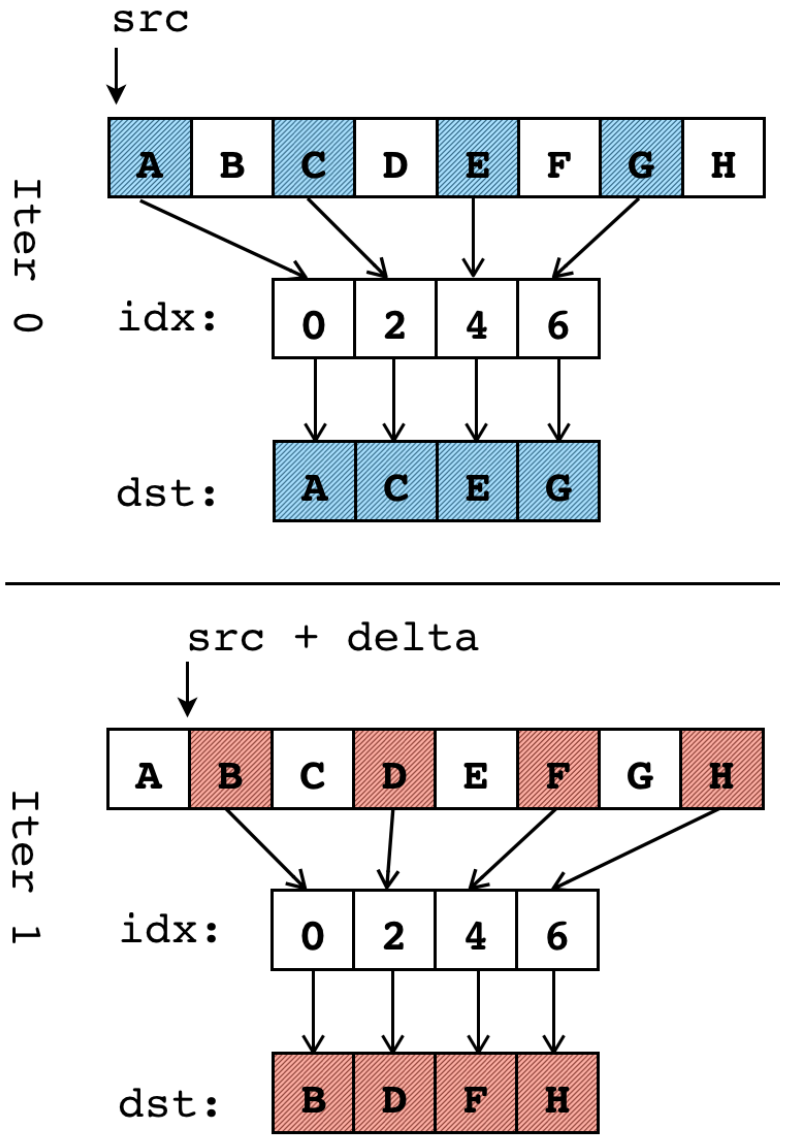
Spatter Kernels

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

```
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*.



Benchmark Design

1. Read all *patterns* (essentially [idx, delta] pairs) from a Json file

2. Determine maximum memory required and allocated data

3. For each pattern:

   - 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
```

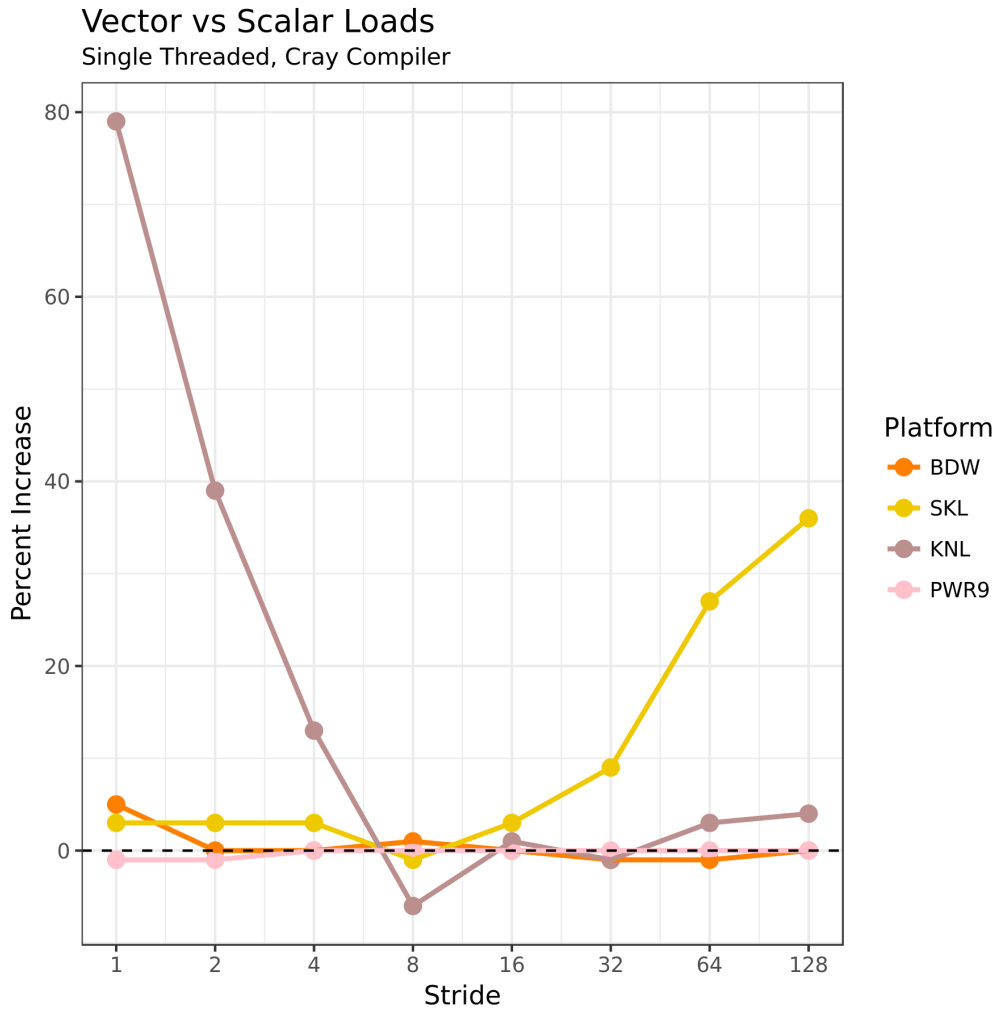
Features

- Backend - Serial, OpenMP, or CUDA
- Performance tuning
  - OpenMP Work per thread
  - CUDA block size
- Json is used as input and output

Use Cases

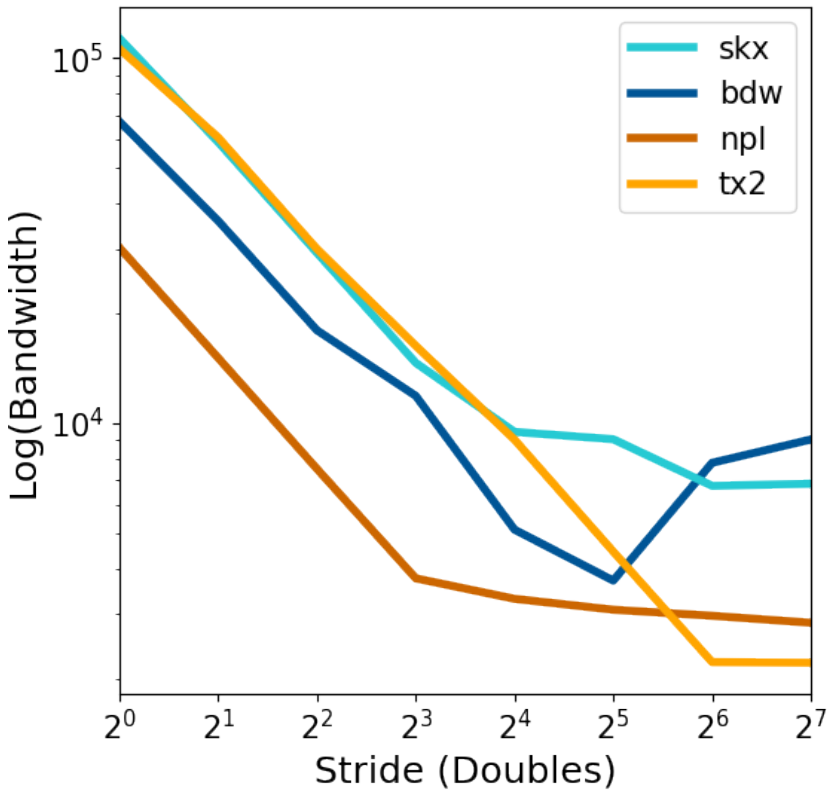
Vector vs Scalar

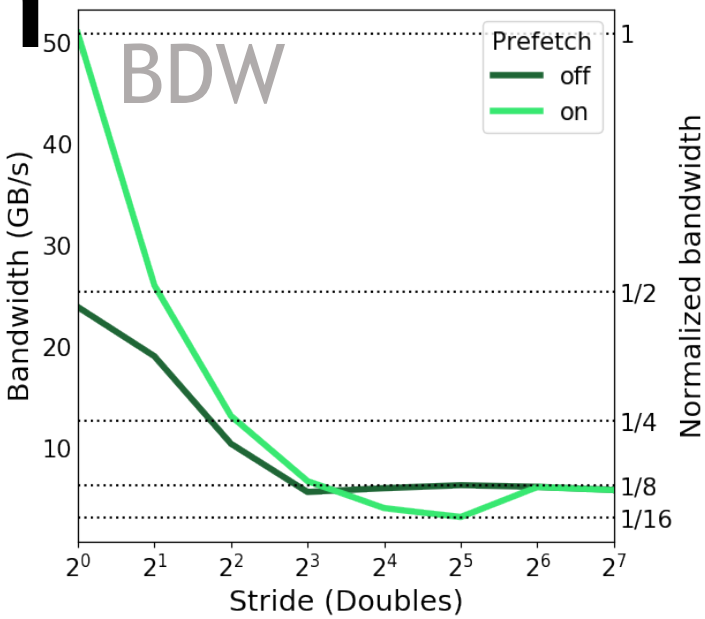
- In previous Intel hardware generations, there was no benefit to using G/S instructions.
- More modern hardware, however, shows speedup when using these instructions for uniform stride loads.

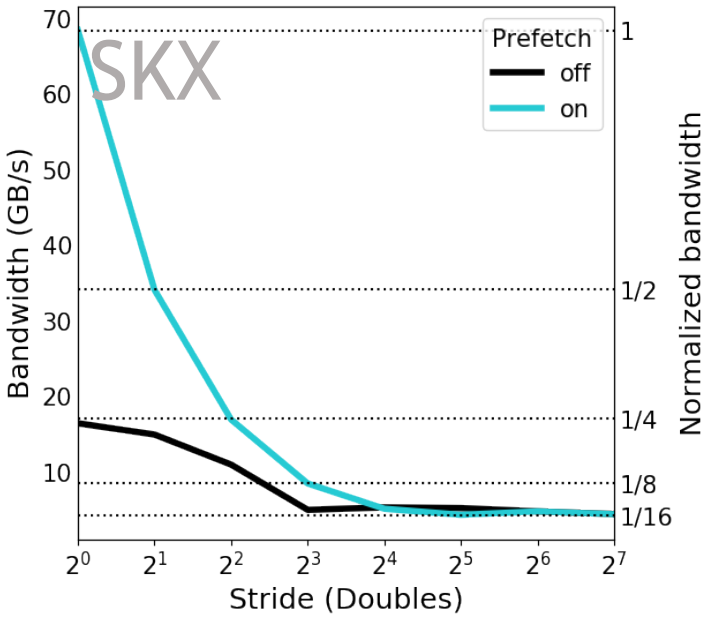


Cache Implementation Exploration

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







Application Specific Patterns

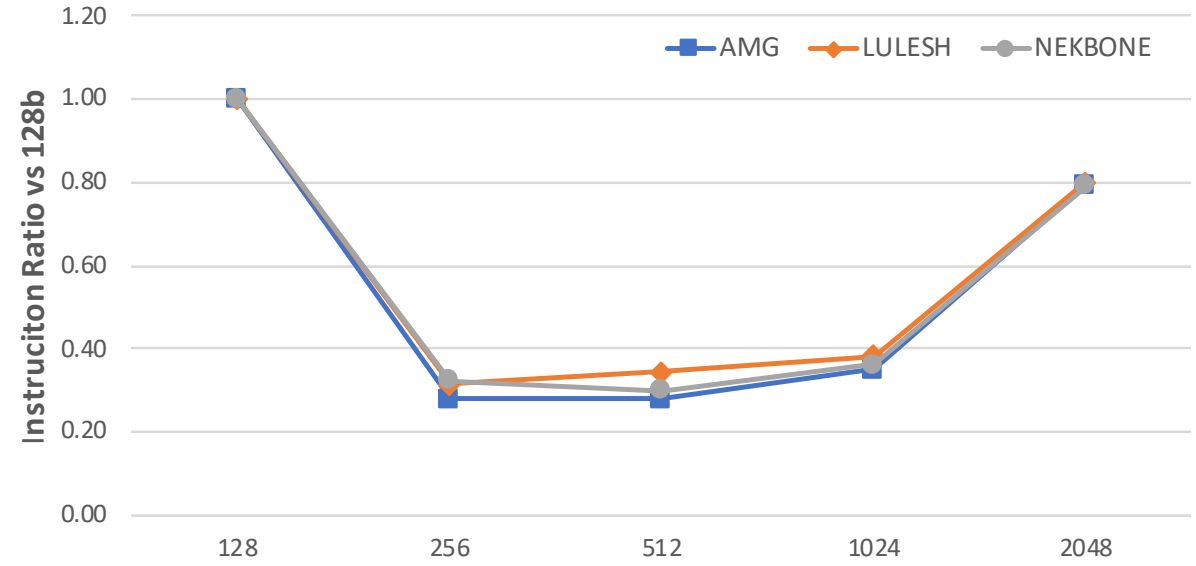
			Broadwell	Skylake	TX2
AMG	2 patterns	All gathers, Mostly Stride-1 (gap 34)	~123	318-337	~270
Nekbone	3 patterns	All gathers, uniform stride 6	118-124	280-325	235-242
Lulesh	12 patterns	Gathers and scatters, uniform stride 1, stride 8, stride 24	2-121	1-329	120-271

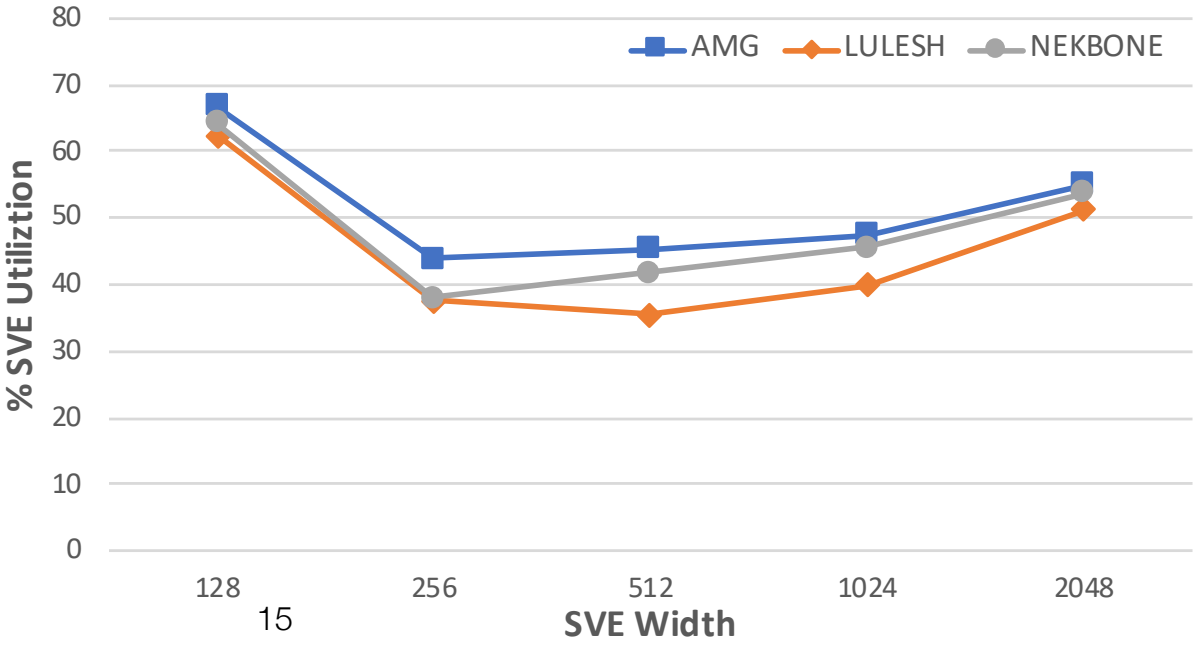
Performance in GB/s

ARMIE Experiments

- Spatter is run with mini-app patterns using ARMIE 19.1 to evaluate the *relative* performanc e of using different SVE vector widths.

- Performance will likely increase as SVE utilization increases. However, a lower instruction count with a slightly lower SVE utilization (e.g., with 256,512,1024) may actually perform better on a real-world system.





What's Next?

- The benchmark in its current state is good at measuring how a single instruction will perform when operating on large buffer sizes, but is not as accurate when the data fits in L1. This requires kernels written with intrinsics or assembly.

- Currently, we can't express orderings between G/S instructions. We will use SVE memory traces generated from ARMIE or other simulation tools to work on memory stream approximations that can be used to increase the fidelity of Spatter.

More Info

- Spatter.io
  - Documentation, and link to source code, link to data
- ArXiv Pre-print (Out of date, but expect a new submission soon)
  - Spatter: A Benchmark Suite for Evaluating Sparse Access Patterns
  - https://arxiv.org/abs/1811.03743
- SC19 Poster: A Framework for Measuring Hardware Gather-Scatter Support
- Code
  - https://github.com/hpcgarage/spatter

Acknowledgements

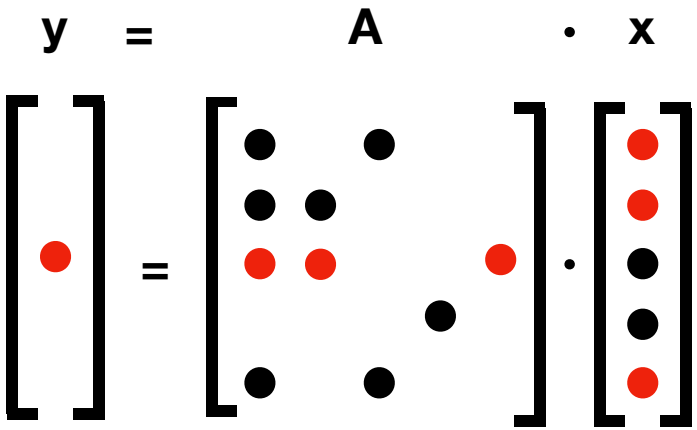


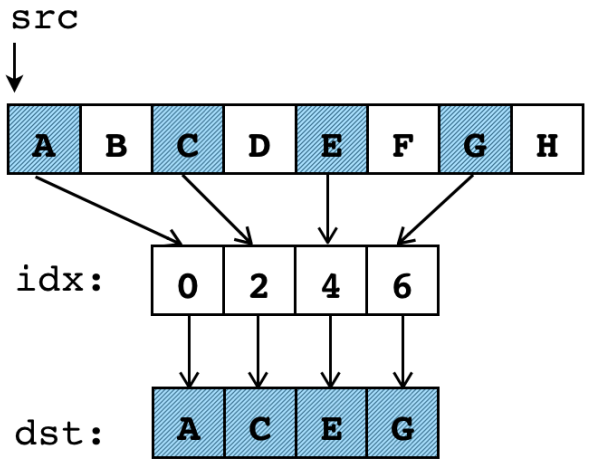
This material is based upon work supported by the National Science Foundation under Award #1710371.

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 also 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.*

Using the Spatter Benchmark Suite to Evaluate SVE Support for Gather/Scatter





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]`