Spatter

Spatter.io

A Framework for Measuring Hardware Gather-Scatter Support

Patrick Lavin, Jeffrey Young (advisor) , Richard Vuduc (advisor)

Georgia Tech

Abstract

In recent years, we have seen the re-addition of vector units to CPUs. While these units easily give speedups for easily vectorized applications with dense memory access, it can be hard to characterize how different access patterns will effect the performance of vectorized code.

We have developed Spatter, a benchmark which allows us to test and investigate the gather-scatter units available on current and upcoming hardware. The information that Spatter reveals to users is of use to everyone from hardware vendors who wish to compare gather-scatter units across platforms, to compiler writers and application developers who wish to test memory access pattern performance in their vectorized code.

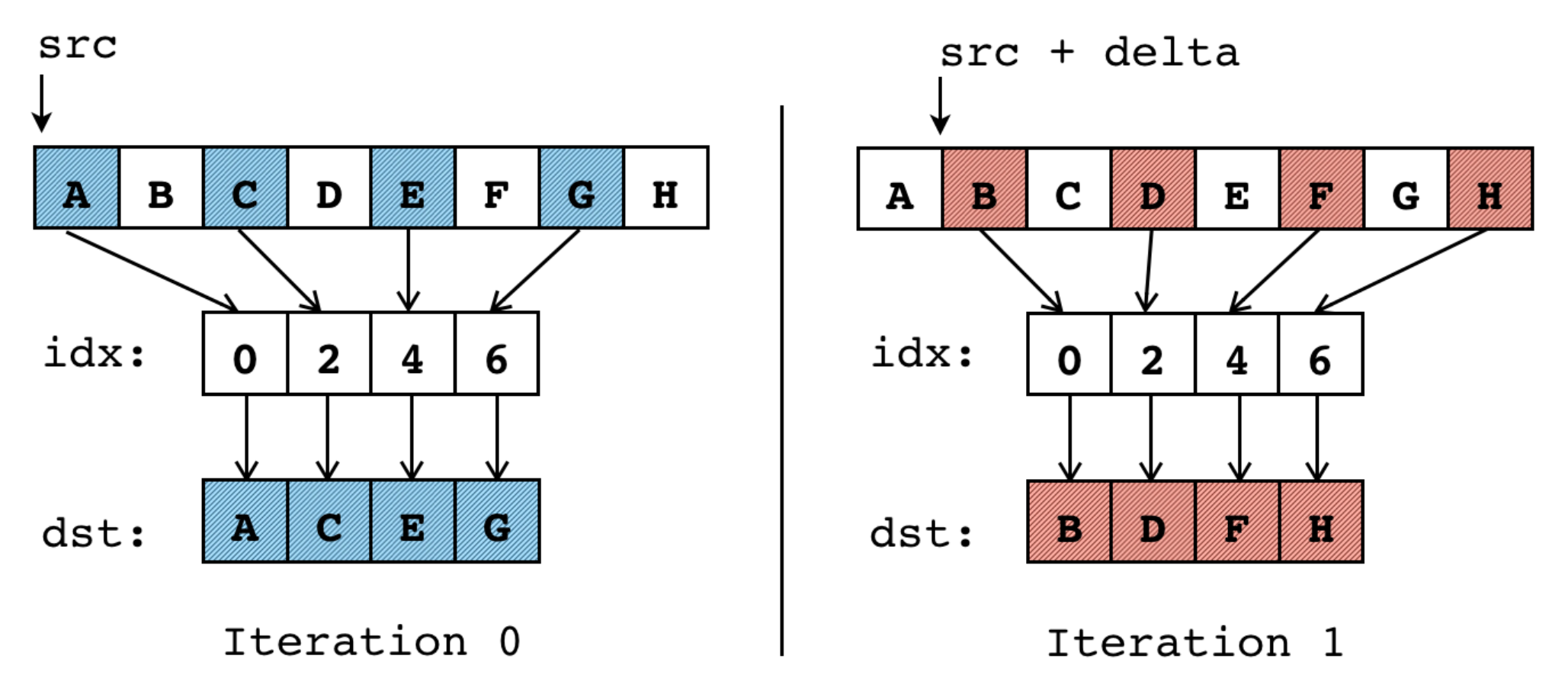
Background on Gather-Scatter

What are gather-scatter patterns?

Gather and scatter are the names given to the vector versions of indexed load and store operations on new processors. Essentially, if you want to vectorize a loop such as the one to the right, you end up with something that looks like a gather instruction. These instructions are also used to encode things like strided access, which would show up in code accessing every nth array element, for instance.

```
for (i):
  C[i] = A[B[i]]
```

In practice, the elements gathered (or scattered) are described by a base memory address, and n indices describing the offset of each array element from the base. This array of indices is what we refer to as a pattern. In Spatter, we describe a pattern with a delta and an index buffer:



Real-World G/S Patterns

Preliminary analysis of SVE gather scatter instructions from DOE mini-app traces has revealed several common patterns.

Mostly Stride-1 Observed in: AMG	[0, 1, 2, 36, 37, 38, 72, 73]
Uniform Stride Observed in: LULESH, Nekbone	[0, 6, 12, 18, 24, 30, 36, 42]
Broadcast Observed in: Pennant	[0, 0, 0, 0, 8, 8, 8]

Hardware Support for Gather/Scatter

Explicit gather-scatter instructions are currently available on Intel KNL, Skylake, and Ice lake processors, and will be enabled by SVE in upcoming ARM processors. Additionally, the Fujitsu A64FX will support pattern-dependent G/S optimizations. While GPUs do not sport instructions to perform G/S to/from memory, their support for coalescing means they perform well in these tests.

Design of the Spatter Benchmark

Input

Spatter takes many input parameters, but the most important ones are pattern and delta specifications. The pattern is applied to a buffer repeatedly to gather data into the CPU, or to scatter data into memory. The intervals at which the pattern is applied is referred to as a delta. Spatter supports the built-in patterns Uniform Stride and Mostly-Stride 1, or you can write out your own.

```
$./spatter –pUNIFORM:8:6
$./spatter –p”0,1,2,3,8,9,10,11”
```

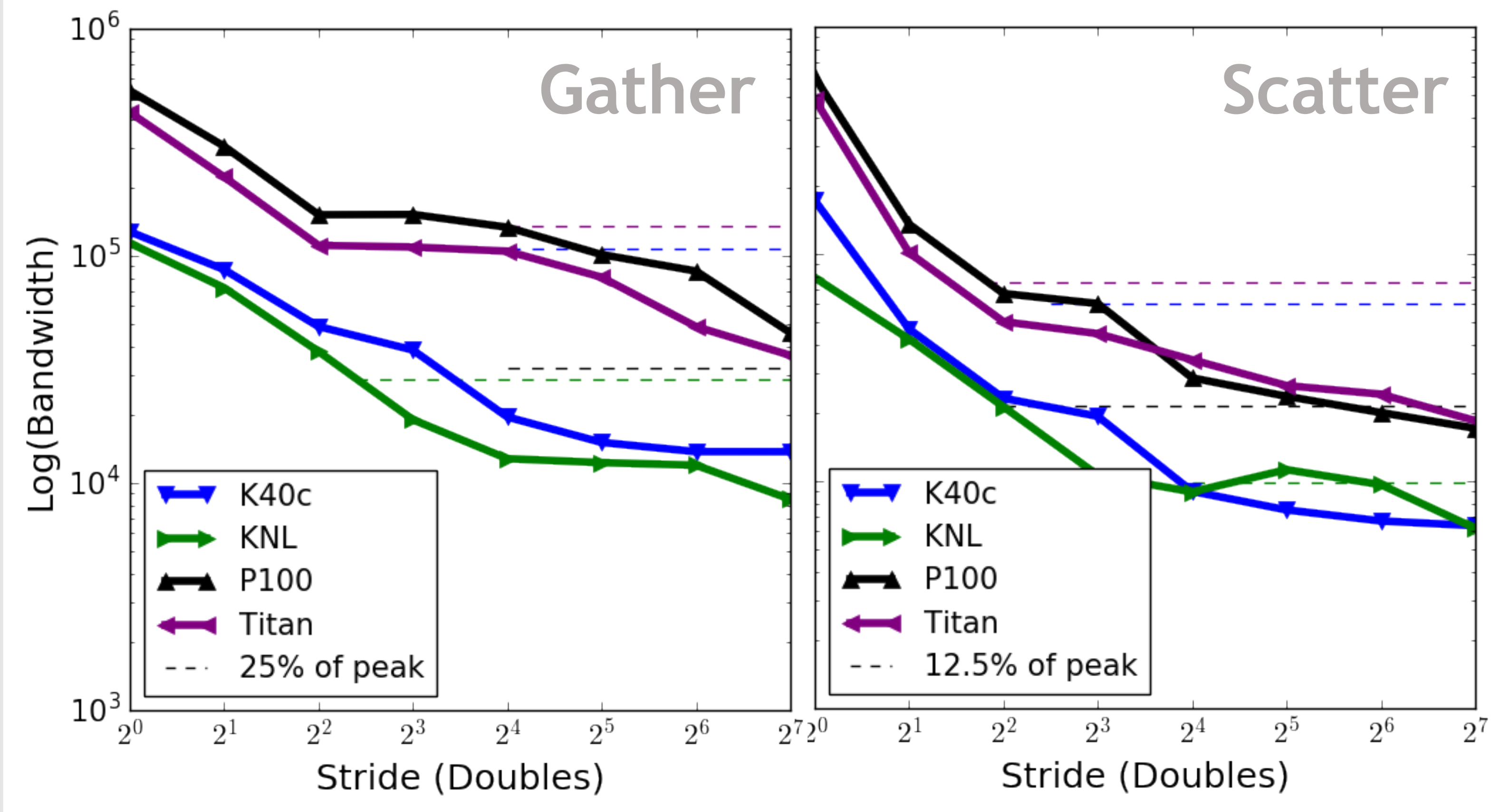
Spatter accepts a JSON file describing many configurations to allow for various optimizations and to ease scripting. We can create JSON files with a collection of patterns from an application. Spatter reports a bandwidth for each pattern.

```
$cat nekbone.json
[{"pattern":
   [0,6,12,18,24,30,36,42,48,54,60,66,72,78, 84,90],
  "delta": 3},
  … ]
```

The Kernels

Spatter uses separate kernels for gather and scatter. The kernels are implemented in C, so care has to be take to ensure that gather and scatter instructions actually appear in the assembly. With a few pragmas, CCE allows us to do this. We also have an AVX512 intrinsic version that will be available soon,

GPU Results



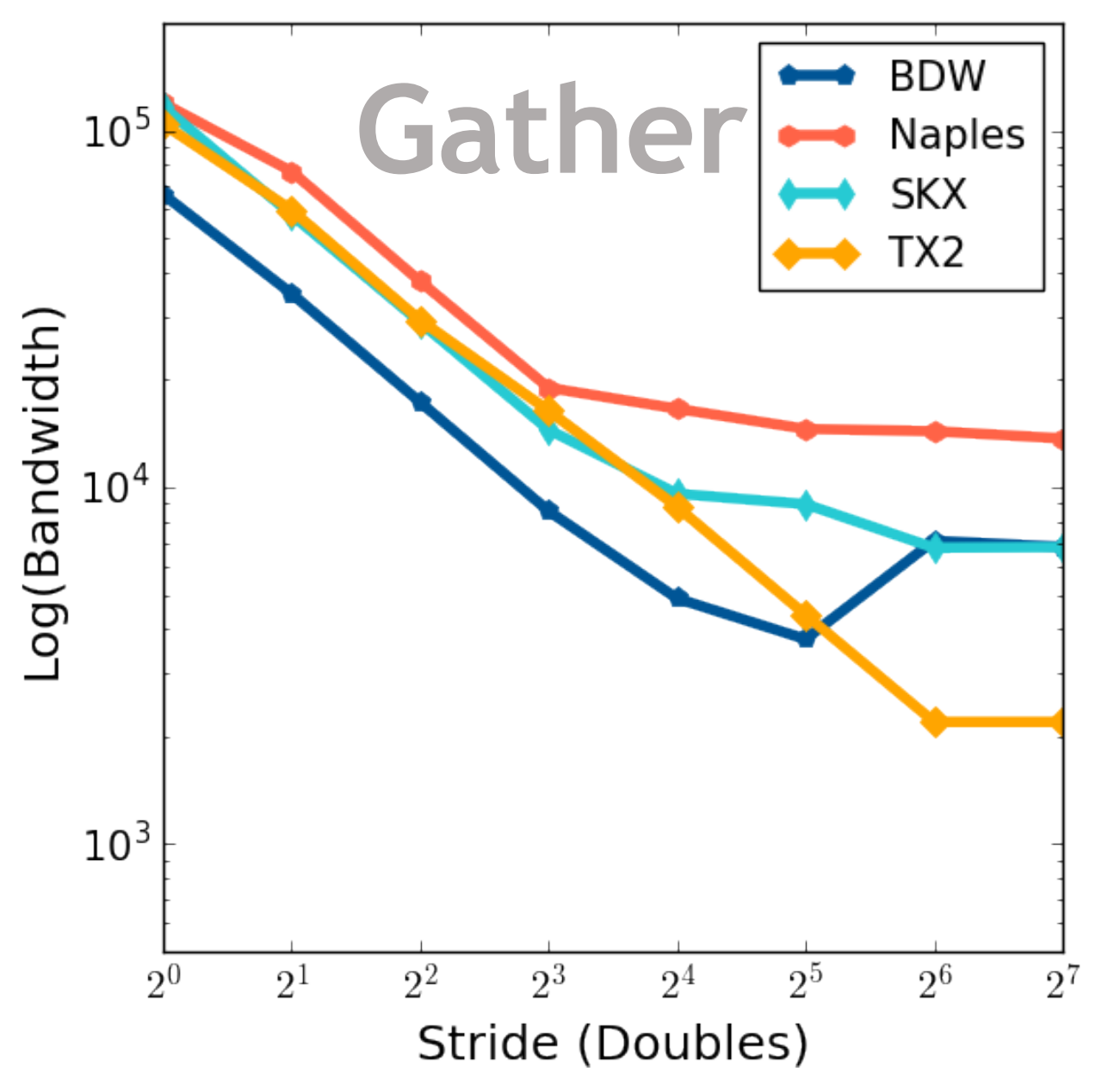
GPU Uniform Stride Tests

In these tests, we test the simplest pattern, uniform stride. We see bandwidth drop by half for stride-2 and stride-4. However, for the P100 and the Titan Xp, from stride-4 to stride-8, we see that bandwidth stays the same. This is due to the fact that **GPUs are able to coalesce some loads.** The older K40 hardware shows less ability to do so. **The KNL, despite having having MCDRAM similar to the P100's HBM, performs closer to the K40 on our tests.** However, the large number of threads on KNL may require further tuning of the OpenMP backend.

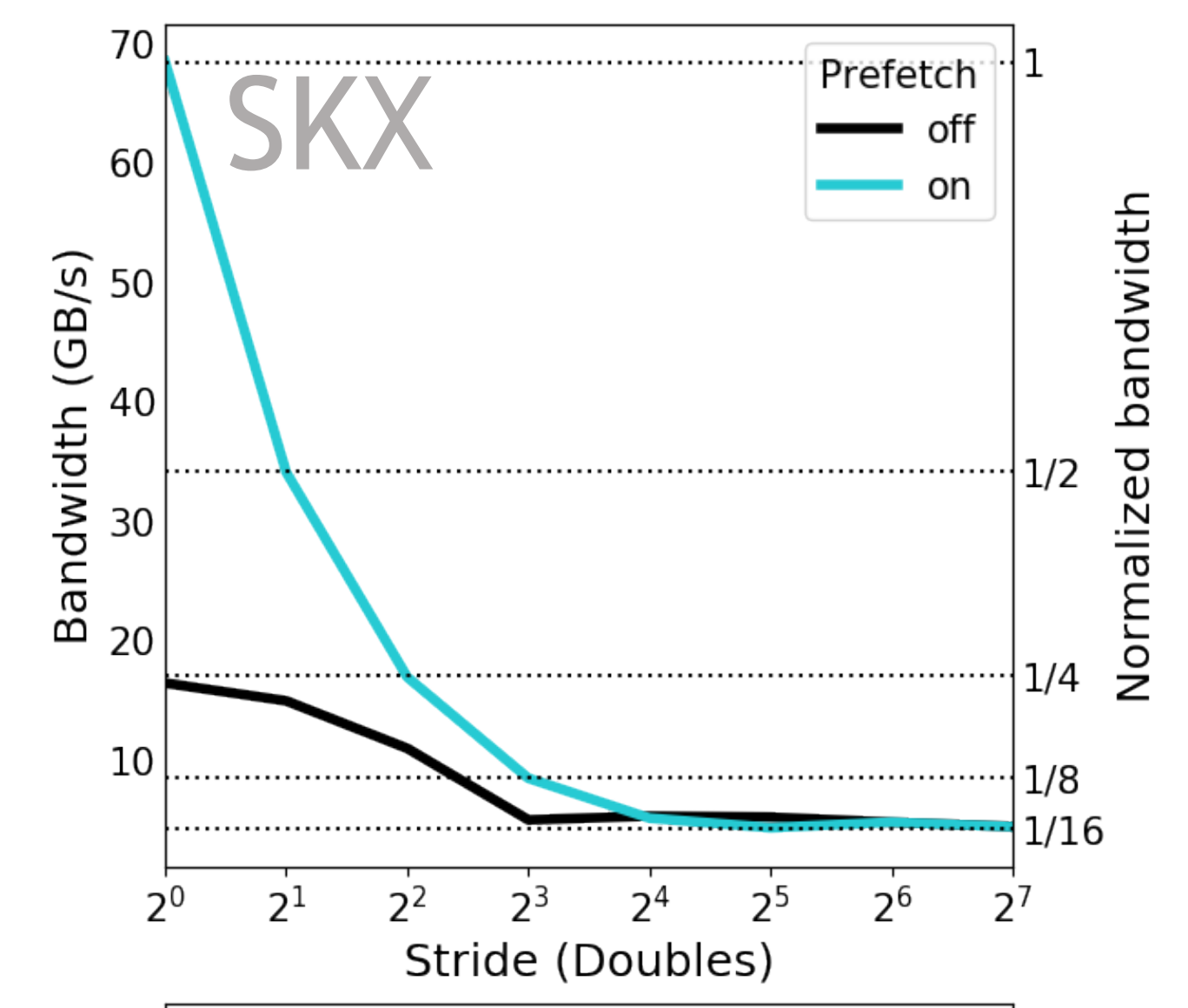
CPU Results

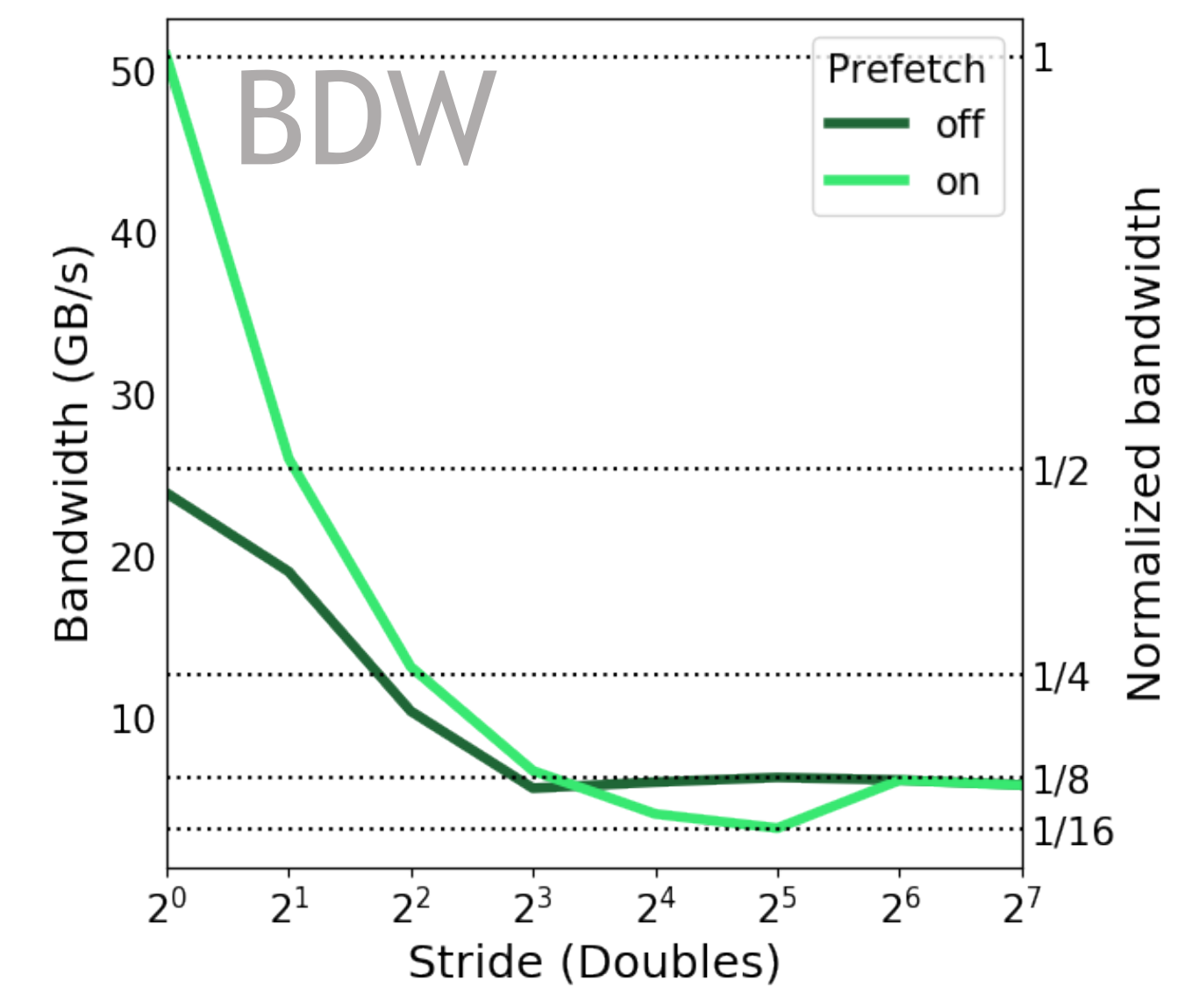
Uniform Stride Gather Tests

In the top plot, we have a platform DRAM bandwidth comparison between four CPUs. On the x-axis, we increase the stride from 1 to 128. Interestingly, Broadwell bandwidth actually increases at higher strides, even beating out Skylake.



In the BDW and SKX plots, the y-axis now reports actual bandwidth, with and without prefetching enabled. When increasing the stride, we expect bandwidth to hit 1/8 of peak, as we are using 1/8 of every cache line brought in. So why do we drop to 1/16 in some cases? **For low strides, the Broadwell prefetchers actually grab two lines at a time, but stop doing this at stride 64.** But in Skylake, the story is different. **Even with prefetching turned off, we are at 1/16 bandwidth, as the hardware always delivers 2 cache lines,** no matter what. These sorts of implementation details can be important for optimizing for sparse access.





Pattern Tests

We ran Spatter with the patterns found in 3 DOE mini-apps:

1. Lulesh shows poor performance on most CPUs because it includes a delta-0 scatter that we believe triggers cache invalidations.
2. For GPU systems, the R coefficient shows that STREAM is well correlated (close to 1) with the Spatter results. However, CPU results for the uniform stride patterns in Nekbone demonstrate how STREAM can be poorly correlated with an application that has cache-dependent gather / scatter patterns.

	AMG	NEKBONE	LULESH	STREAM
BDW	123	121	20	43
SKL	328	308	12	96
CSL	234	215	9	94
Naples	140	323	3	97
TX2	270	247	232	241
KNL	201	190	19	249
R value	0.26	0.03	0.5	
K40c	108	99	88	193
TitanXp	496	320	175	443
P100	703	673	165	541
R value	0.66	0.62	0.62	

Impact (and Next Steps)

Spatter has received development contributions from Cray and is currently being evaluated by colleagues at Intel and Arm. To help them evaluate the efficiency of gather-scatter hardware, we have three big areas to hit next. We will write:

- SVE and AVX512 kernels, to better measure L1 and L2 bandwidths
- Tracing software, to extract G-S patterns from real applications

Acknowledgement: This work is partially supported by NSF Award #1710371, and was partially completed at Cray, Inc. Thanks to Dan Ernst and Aaron Vose for their support.