A Framework for Measuring Hardware Gather-Scatter Support

Patrick Lavin
plavin3@gatech.edu
Georgia Institute of Technology

Jeffrey Young (Advisor)
jyoung9@gatech.edu
Georgia Institute of Technology

Richard Vuduc (Advisor)
richie@cc.gatech.edu
Georgia Institute of Technology

ABSTRACT

This poster describes a new benchmark tool, Spatter, for assessing memory system architectures in the context of indexed accesses. This type of memory operation is often used to easily express sparse and irregular data patterns. Scatters and gathers have widespread utility in many modern HPC applications, including traditional scientific simulations, data mining and analysis computations, and graph processing.

Spatter specifically measures gather / scatter variations for multiple platforms, with several tunable backends, and provides comparison metrics for different sparse access patterns. It also allows for benchmarking with trace-driven "proxy patterns," which we demonstrate for multiple DoE mini-apps to show how pattern-based benchmarking can be used to improve system insights.

CCS CONCEPTS

• **General and reference → Measurement**.

KEYWORDS

Memory Architecture, Gather/Scatter, Microbenchmarks

ACM Reference Format:
Patrick Lavin, Jeffrey Young (Advisor), and Richard Vuduc (Advisor). 2019. A Framework for Measuring Hardware Gather-Scatter Support. In *SC19, November 18–21, 2019, Denver, CO.* ACM, New York, NY, USA, 2 pages.

1 MOTIVATION

New CPU architectures have begun to incorporate advanced vector functionality like AVX-512 and the Scalable Vector Extension (SVE) for improved SIMD application performance. In addition, many of these new vector specifications include explicit support for *indexed accesses* like gather and scatter (G/S). These types of memory operations involve a load or store through a level of indirection, such as `reg` ← `base[idx[k]]`, and they appear commonly in scientific and data analysis applications.

While many memory-focused microbenchmarks [1] are available today, a gap exists in the evaluation of indexed accesses including gather and scatter. We are motivated to design a benchmarking tool that assesses system performance on gather / scatter workloads for three different types of users: (1) vendors and hardware architects might wonder how new hardware features like AVX-512 actually impact the memory system and whether new intrinsic operations actually help to reduce the costs of data movement, (2) application developers may consider how the data structures they choose impact the G/S instructions their code compiles to, and (3) compiler writers might require better data on real-world memory access patterns to decide whether to implement a specific vectorization optimization for sparse accesses.

In considering these needs, we have formulated one such tool, called Spatter. It evaluates indexed access patterns based on gather and scatter operations, which represent a variety of applications across different language and architecture platforms. More importantly, we believe Spatter can help to answer a variety of system, application, and tool evaluation questions, some of which include: (1) What application gather / scatter patterns exist in the real world, and how do they impact memory system performance? (2) How does prefetching affect performance of indexed accesses on modern CPU platforms? (3) How does the performance of G/S change when dealing with sparse data on CPUs and GPUs?

2 SPATTER DESIGN

We show in this work that the Spatter tool suite can address these questions with the following key features. At a basic level, Spatter provides **tunable gather and scatter implementations**. These include **CUDA and OpenMP backends** with knobs for adjusting thread block size and ILP on GPUs and work-per-thread on CPUs.

Finally, Spatter includes support for running built-in, parameterized memory access patterns, or custom patterns. We show, for instance, how one can collect G/S traces from Department of Energy (DoE) mini-apps to gain insights or make rough predictions about performance for hot kernels that depend on indexed accesses.

3 EXPERIMENTAL RESULTS

Results from Spatter show that newer GPU architectures perform best for both gather and scatter operations in part due to memory coalescing and faster memories. AMD Naples performs best of all the CPU-based platforms (Broadwell, Skylake, TX2) for strided accesses. A study of prefetching with Spatter further shows how gather / scatter benefits from modern prefetching across Broadwell and Skylake CPUs. Spatter's scalar backend is also used to demonstrate how compiler vectorization can improve gather / scatter with large improvements for both Skylake and Knight's Landing. Experiments for three DoE mini-apps show G/S performance improvements enabled by caching on CPU systems and by fast HBM memory on GPUs. Surprisingly, these parameterized access pattern studies also show that STREAM bandwidth does not correlate well with specific mini-apps (Nekbone) that are cache-dependent, whereas Spatter is able to capture some of this cache behavior.

4 CONCLUSION

We have used Spatter's simple memory pattern model, consisting of a pattern and a delta, to explore features of CPU and GPU memory architecture and to model memory access patterns found in mini-apps. In the future, we will continue this effort by extending

Spatter to work on more platforms, and by generalizing our model to descriptive of a larger class of patterns.

ACKNOWLEDGMENT

This work is partially supported by NSF Award #1710371, and portions were completed while the first author was employed by Cray, Inc. as an intern. This research used resources of the Oak Ridge Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC05-00OR22725.

REFERENCES

[1] J. McCalpin, “Notes on “non-temporal” (aka “streaming”) stores.” http://sites.utexas.edu/jdm4372/tag/cache/, 2018.