Accurately Modeling Sparse Accesses for Benchmarking and Architectural Simulation

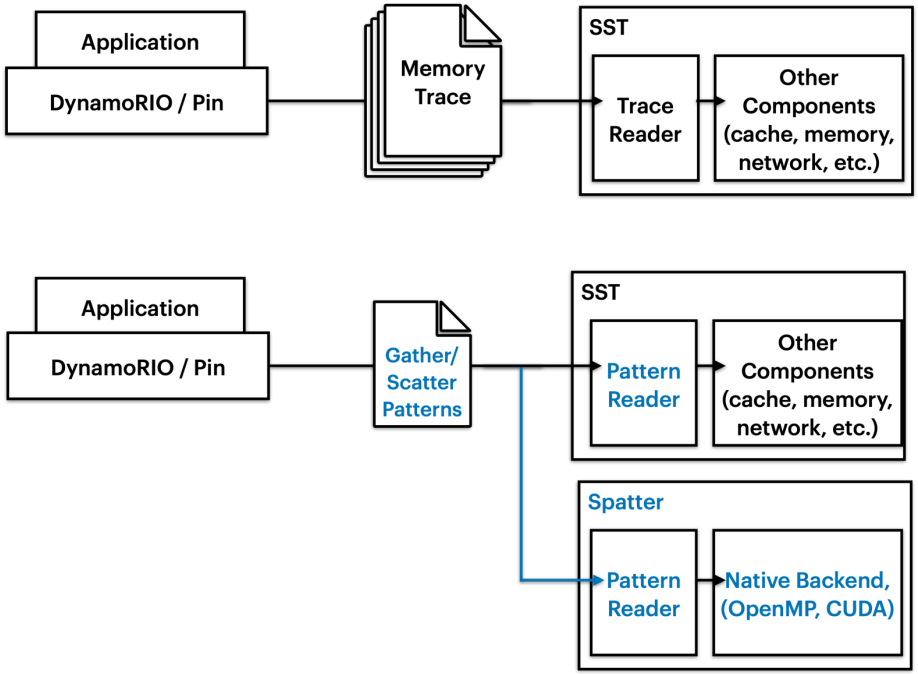
Georgia Tech Computer Science

Jeffrey Young, Vincent Huang,
Patrick Lavin, Rich Vuduc

October 7th, 2021

This work is sponsored by NSF (#1710371) and Sandia National Labs

Motivation – Benchmarking and Simulation



How do we synthesize a useful input to represent sparse accesses including gather/scatter (G/S)?

- We are interested in tracking G/S as a challenging memory access pattern for HPC applications.

What are the requirements for a good pattern?

Georgia Tech Computer Science

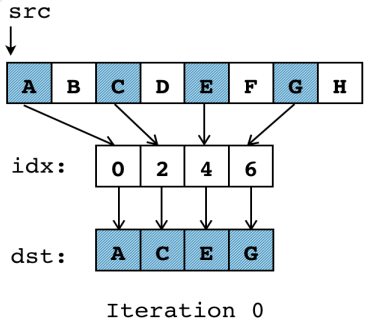
To recreate sparse accesses without requiring a full trace requires:

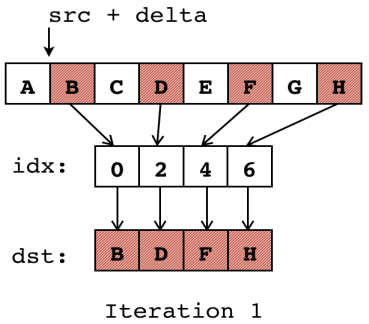
Base address and offsets

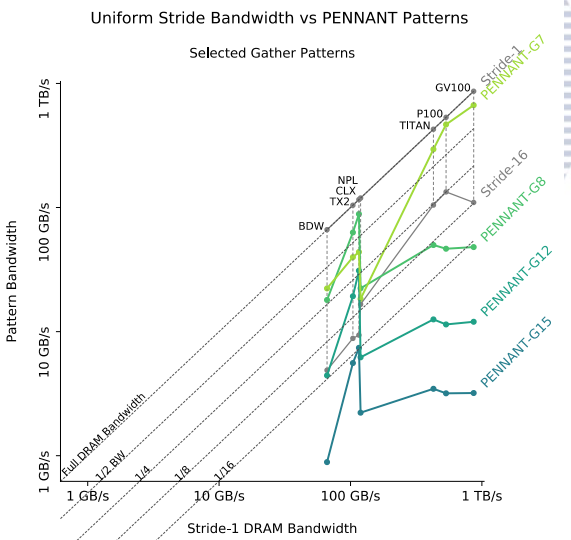
- Delta values for subsequent accesses

Frequency

- How frequently do sparse accesses occur?
- How many “regular” accesses occur in between sparse accesses of interest? How do they affect caching behavior?



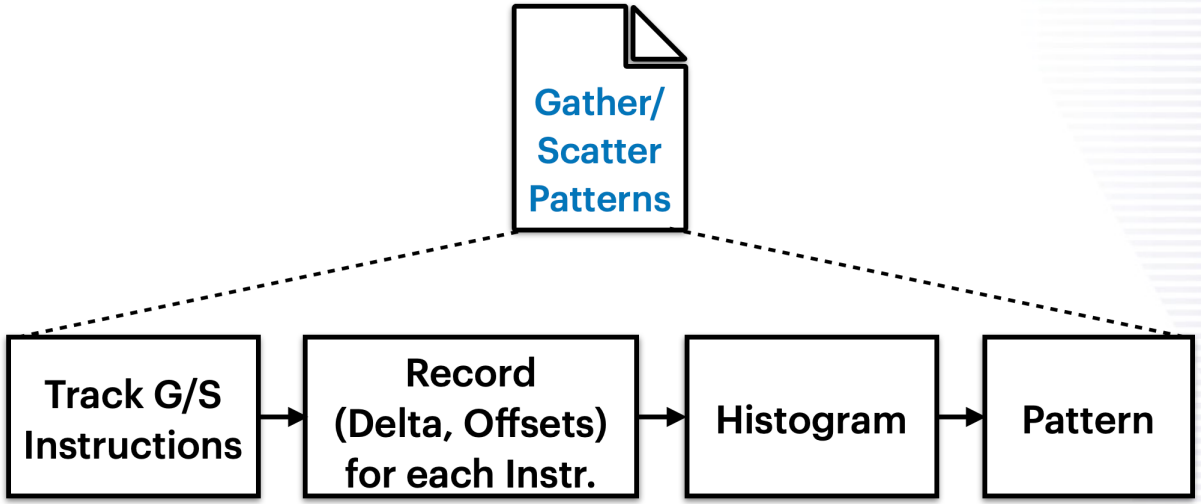




Related work:

- Patrick Lavin, et al., Evaluating Gather and Scatter Performance on CPUs and GPUs. *In Proceedings of the International Symposium on Memory Systems* (pp. 209-222). September 2020.
- Alif Ahmed, et al., Hopscotch: A micro-benchmark suite for memory performance evaluation. In *Proceedings of the International Symposium on Memory Systems* (pp. 167-172). September 2019

Building the workflow



- Using DynamoRio, we determine initial patterns and generate a trace
  - Naïve (trace entire program) or using trace delimiting to provide user-driven scoping of functions
- Determine patterns with histogram and analysis functionality (Python scripts)
- Formulate patterns that work as Spatter inputs

Repo at: https://github.com/hpcgarage/dr-gather-scatter-trace

Validation of Patterns

Georgia Tech Computer Science

```
$ gdb ./spatter
(gdb) disassemble gather_smallbuf
[output truncated to show only gather instructions]
    0x0000000000416a0b <+389>:   vgatherqpd (%r15,%zmm2,8),%zmm3{%k1}
    0x0000000000416a4f <+457>:   vgatherqpd (%r15,%zmm3,8),%zmm4{%k1}
(gdb) b *0x0000000000416a0b
Breakpoint 1 at 0x416a0b: file ~/spatter/src/openmp/openmp_kernels.c, line 44.
(gdb) b *0x0000000000416a4f
Breakpoint 2 at 0x416a4f: file ~/spatter/src/openmp/openmp_kernels.c, line 44.
(gdb) ignore 1 999999999
Will ignore next 999999999 crossings of breakpoint 1.
(gdb) ignore 2 999999999
Will ignore next 999999999 crossings of breakpoint 2.
(gdb) r -pUNIFORM:8:1 -l$((2**16)) -t1 -kGather
[output omitted]
(gdb) info breakpoints
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   0x0000000000416a0b in L_gather_smallbuf_20__par_region0_2_0
at ~/spatter/src/openmp/openmp_kernels.c:44
        breakpoint already hit 720896 times
        ignore next 995183746 hits
2       breakpoint     keep y   0x0000000000416a4f in L_gather_smallbuf_20__par_region0_2_0
at ~/spatter/src/openmp/openmp_kernels.c:44
        ignore next 999999999 hits
```

GDB with Spatter "gather_smallbuf" function

```
~/dynamorio/build/bin64/drrun -noinject -c ~/dr-gather-scatter-
trace/client/build/libcount_client.so -- ~/spatter/build_omp_intel/spatter -pUNIFORM:8:1 -
l$((2**16)) -t1 -kGather

[output truncated]
Top 15 opcode execution counts in 64-bit AMD64 mode:
    720896 : vgatherqpd
```

DynamoRio with Spatter "gather_smallbuf" function

1) Generate code with vectorized gather/scatter instructions (i.e., AVX-512 or SVE, -02 flags)
2) Disassemble and run application with gdb to observe the execution of G/S instructions
3) Run DynamoRio with our G/S tool and trace delimiting for a function to compare with code static analysis

- Counts may not be not exact due to offsets but should be within the same order of magnitude.

Repo at: https://github.com/hpcgarage/dr-gather-scatter-trace

PENNANT Example

Georgia Tech Computer Science

```
void Hydro::doCycle(const double dt) {
...
    dr_app_start();
    // Begin hydro cycle
    #pragma omp parallel for schedule(static)
    for (int pch = 0; pch < numpch; ++pch) {
...
        // 9.  compute timestep for next cycle
        calcDtHydro(zdl, zvol, zvol0, dt, zfirst, zlast);
    }  // for zch

    dr_app_stop();
} // end doCycle
```

```
#Point tools and Makefiles to your local install of DynamoRio
export DYNAMORIO_ROOT=~/DynamoRIO-Linux-8.0.18895

#Compile and link against this build of DynamoRio
icpc -O2 -I ~/DynamoRIO-Linux-8.0.18895/include -DLINUX -DX86_64 -qopenmp  -c -o
build/Hydro.o src/Hydro.cc
linking build/pennant
icpc -o build/pennant <... .o files> build/Hydro.o -qopenmp -L ~/DynamoRIO-Linux-
8.0.18895/lib64/release/ -l dynamorio

#Run with the custom dr-gs-trace tool. Currently we don't distinguish between threads
export OMP_NUM_THREADS=1
./DynamoRIO-Linux-8.0.18895/bin64/drrun -noinject -c ./dr-gather-scatter-
trace/client/build/libcount_client.so -- ./PENNANT/build/pennant
./PENNANT/test/sedovflat/sedovflat_1920.pnt &> pennant_sedovflat_1920.out
```

Next steps:

- Generate more useful histogram output and test patterns against previous results
- Handle multiple threads
- Extend analysis to other sparse access types

Repo at: https://github.com/hpcgarage/dr-gather-scatter-trace