Accurately Modeling Sparse Accesses for Benchmarking and Architectural Simulation

Vincent Huang — vhuang31@gatech.edu
Jeffrey Young — jyoung9@gatech.edu
Patrick Lavin — plavin3@gatech.edu
Richard Vuduc — richie@cc.gatech.edu

Why do we need to model sparse accesses?

We are interested in sparse accesses like gather/scatter as they are a challenging type of memory access pattern found in high-performance applications.

Our approach consists of

1. Dynamic tracing to extract gather/scatter calls from an instruction stream
2. Analysis to extract the most-used patterns from an application
3. Synthesis to create patterns that represent the application's behavior

What makes a good pattern?

To recreate sparse addresses without a full memory trace requires:

1. Base address and offsets
   1. Delta values for subsequent accesses
2. Frequency of sparse accesses
   1. Some concept of how many many "regular" accesses occur in between sparse accesses of interest

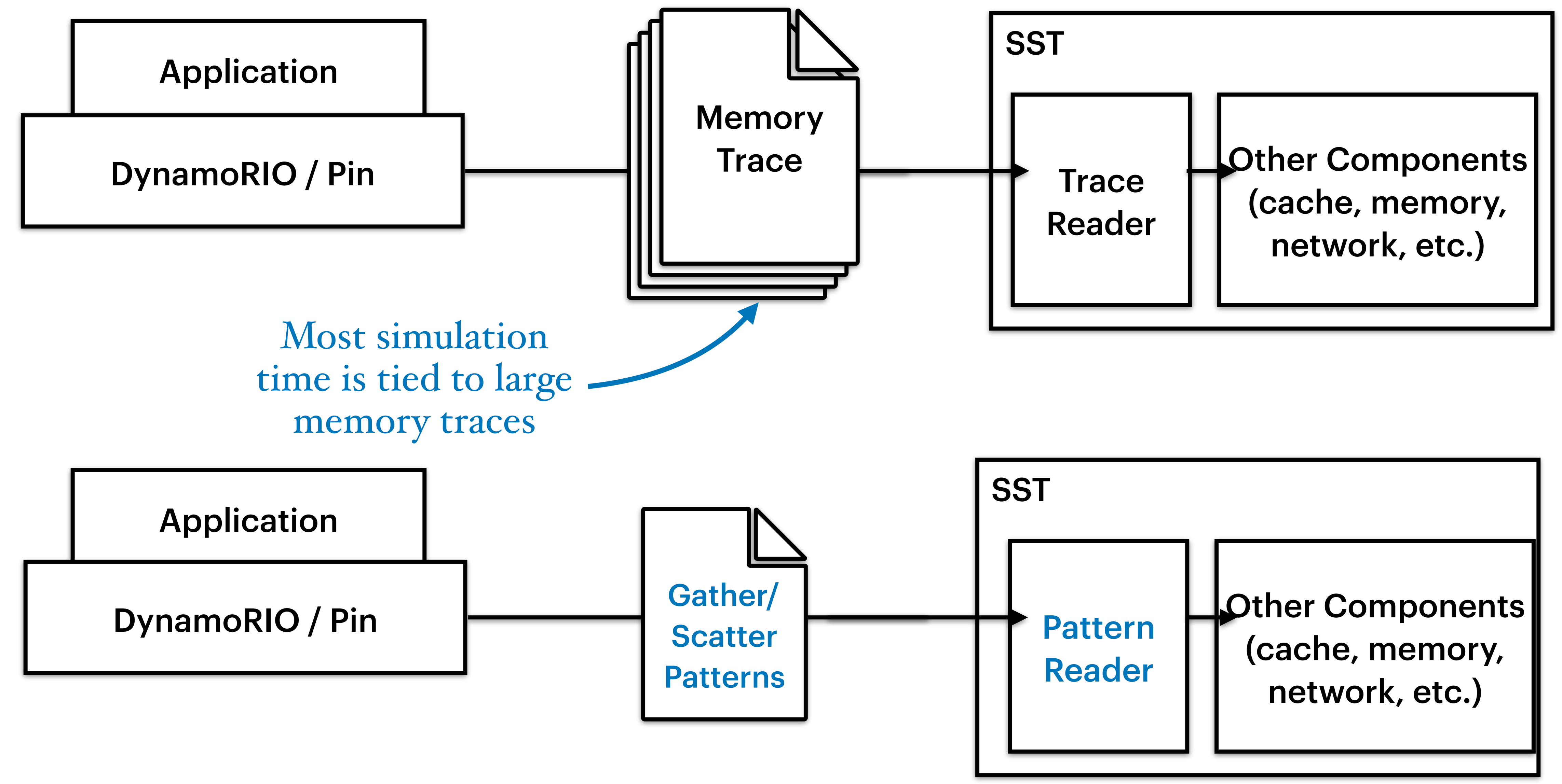
Next Steps

1. Generate more useful histogram output and test patterns against previous Spatter results
2. Handle multi-threaded tracing
3. Extend analysis to other sparse access types and aarch64
4. Test with more applications and SST

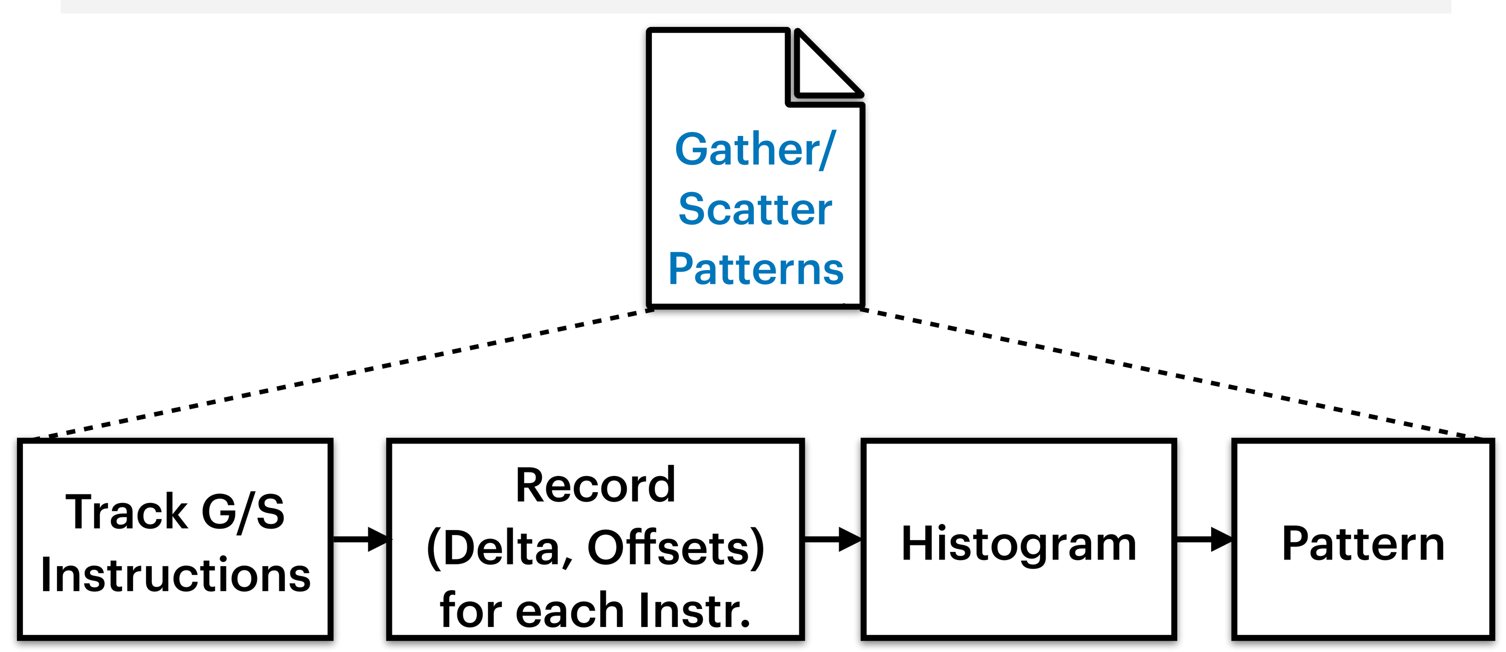
This work was funded by Sandia National Labs and the National Science Foundation

We propose a new tool for extracting useful Gather/Scatter patterns that can be replayed for simulations or benchmarking

Try it out! https://github.com/hpcgarage/dr-gather-scatter-trace



1. We have developed a DynamoRio workflow to track AVX Gather/Scatter instructions



2. We can validate our approach with gdb

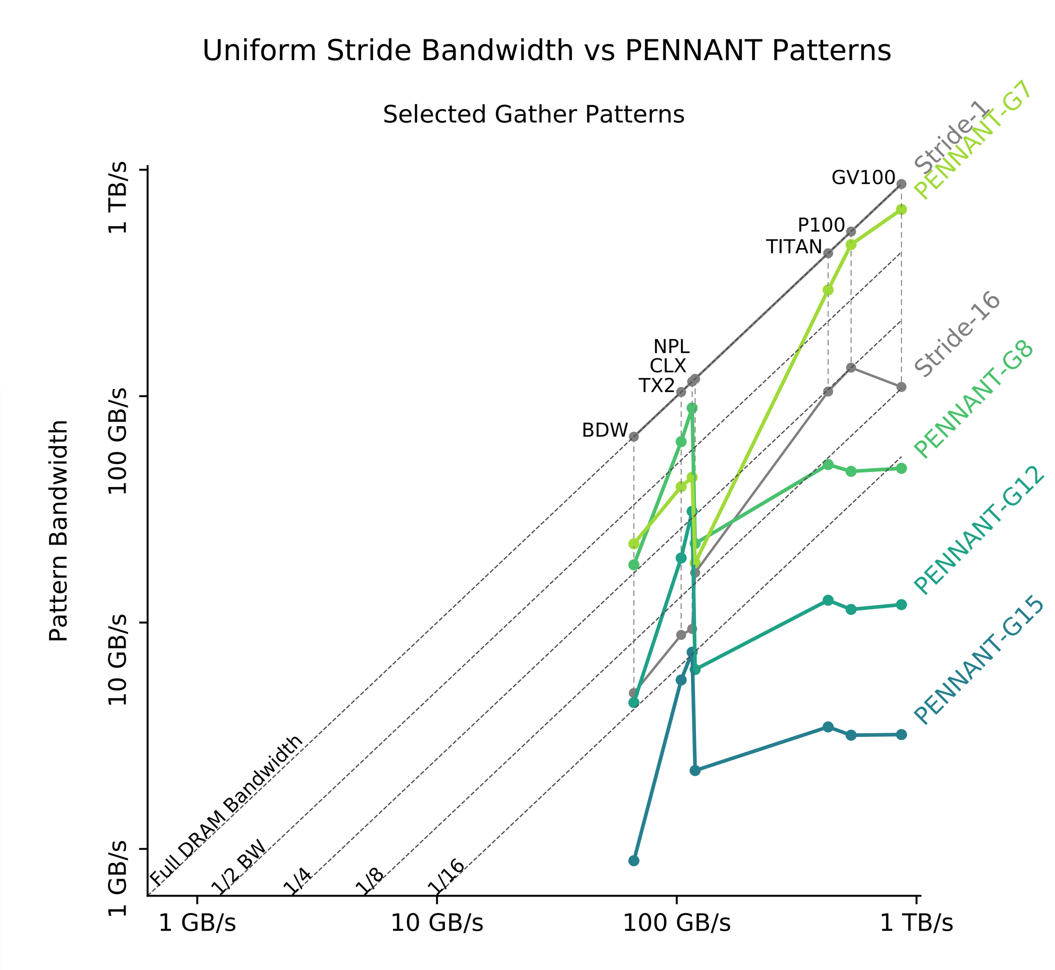
```
$ gdb ./spatter
(gdb) disassemble gather_smallbuf
[output truncated to show only gather instructions]
    0x0000000000416a0b <+389>:   vgatherqpd (%r15,%zmm2,8),%zmm3{%k1}
    0x0000000000416a4f <+457>:   vgatherqpd (%r15,%zmm3,8),%zmm4{%k1}
(gdb) b *0x0000000000416a0b
Breakpoint 1 at 0x416a0b: file ~/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
```

```
~/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
```

PENNANT Example

PENNANT has a variety of gather-focused patterns in its main kernel

```
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
```

Using our tool, we can extract a G/S trace that we can then use for analysis and pattern synthesis.