arm Research

Performing SVE studies using the Arm Instruction Emulator

Tool hands-on and instrumentation

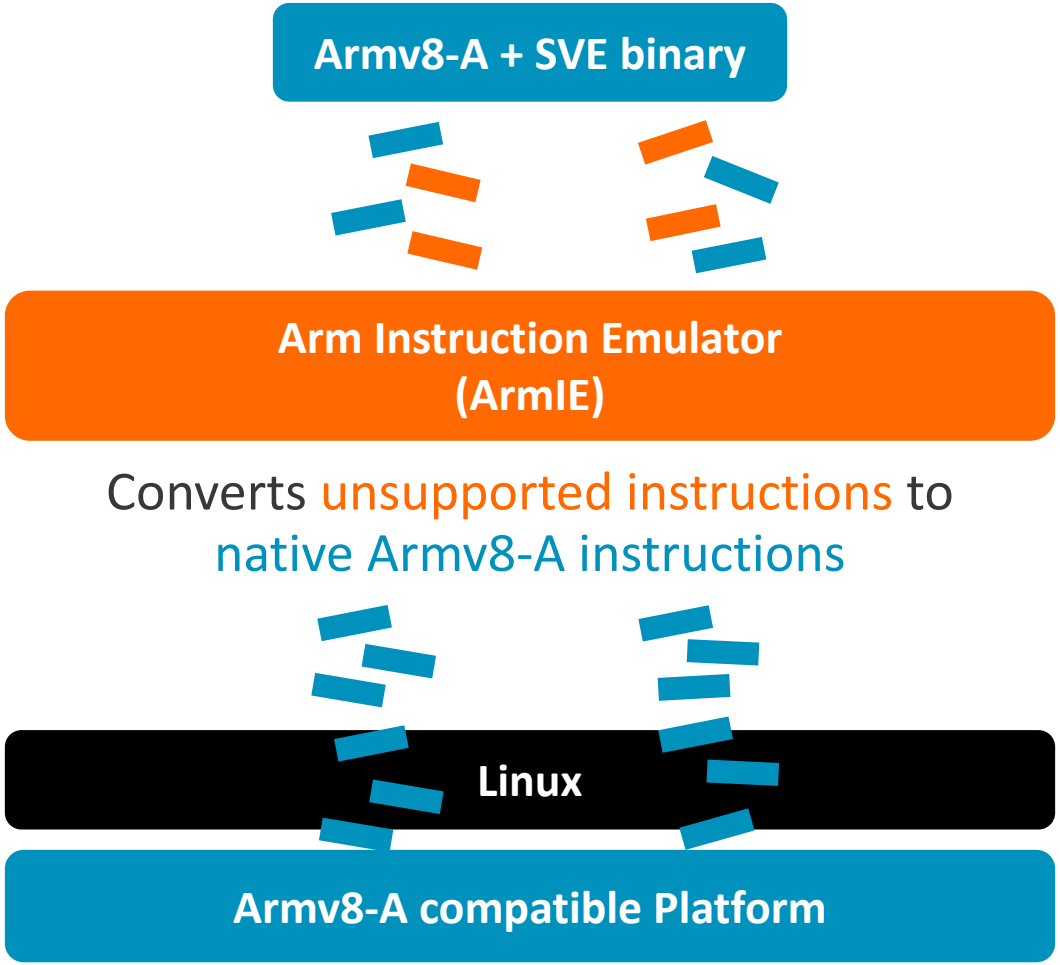
Arm SVE Hackathon

SC18

Roxana Rusitoru (on behalf of Miguel Tairum Cruz)

15 November 2018

The Arm Instruction Emulator (ArmIE)

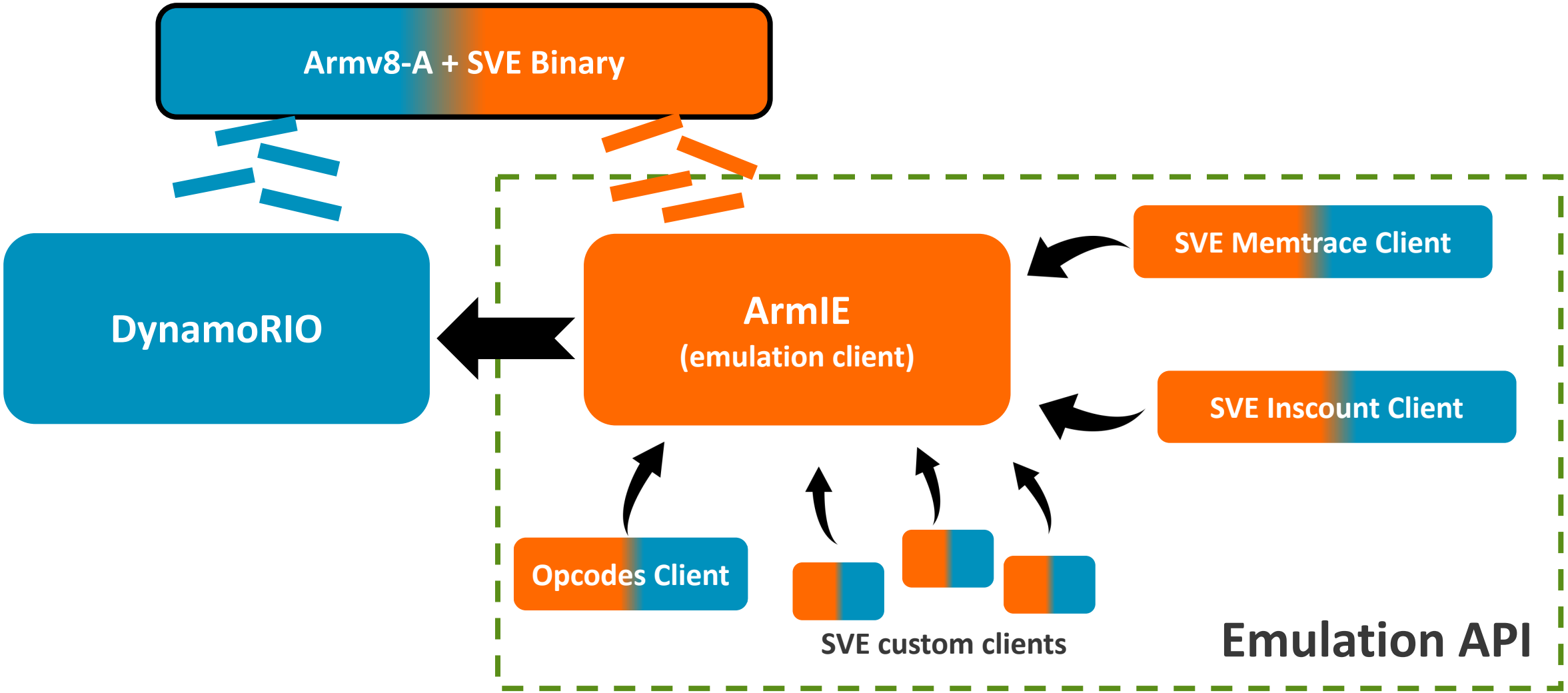


arm Research

Why ArmIE

- Emulates SVE instructions on existing Armv8 hardware
- Faster than simulators (gem5)
  - Enables the study of larger input sizes at a fraction of time
  - Simulators can complement ArmIE with timing results
- Ideal for instruction/memory tracing and dynamic binary instrumentation (DBI)
  - Helps with SVE-supported application development/optimisation
  - Helps choose adequate vector lengths

Instrumenting AArch64 and SVE



arm Research

Instrumenting AArch64 and SVE

- ArmIE integrates with DynamoRIO (DR)

- Compile application with SVE-capable compiler (e.g. Arm HPC compiler, GCC 8.2) and run it with DR SVE clients

- Two SVE clients come pre-packaged in latest ArmIE version (18.2)
  - **SVE inscount**
  - **SVE memtrace**

SVE Instruction Count Client

- Based on the existing DR *inscount* client
  - Reports full application instruction count
- Dynamic counts of AArch64 instructions and SVE instructions are done separately
  - DR counts AArch64 instructions
  - ArmIE counts SVE instructions
- If '*-only_from_app*' flag is enabled, does not count instructions in shared libraries
- Final output reports the total instruction count

```
XX instructions executed of which YY were SVE instructions
```

SVE Instruction Count Client

Simple SVE loop code example:

```
#define N 42
int a[N], b[N], c[N];

int main(void) {
  for(int i=0; i<N; ++i){
    a[i] = b[i] + b[c[i]];
  }
}
```

Compiling with Arm HPC compiler 18.4

```
$ armclang –O3 -march=armv8-a+sve sve_example.c -o sve_example
```

SVE Instruction Count Client

- Full inscount of SVE example (512-bit vectors):

```
$ armie -msve-vector-bits=512 -i libsve_inscount.so -- ./sve_example
```

```
83971 instructions executed of which 22 were SVE instructions
```

- "`-only_from_app`" inscount (no shared libraries) of SVE example (512-bit vectors):

```
$ bin64/drrun -client lib64/release/libsve_512.so 0 "" -client
samples/bin64/libsve_inscount.so 1 "-only_from_app” -- ./sve_example
```

```
127 instructions executed of which 22 were SVE instructions
```

SVE Instruction Count Client

- 512-bit vector example:
  - Each vector holds 16 32-bit integers
  - 3 iterations of the main loop are needed and 7 SVE instructions are present
    - 21 SVE instructions + 1 extra instruction to close the loop

```
#define N 42
int a[N], b[N], c[N];

int main(void) {
  for(int i=0; i<N; ++i){
    a[i] = b[i] + b[c[i]];
  }
}
```

```
.........
400580:     e0 1f a9 25     whilelo p0.s, xzr, x9
400584:     8c f1 05 91     add     x12, x12, #380
400588:     60 41 48 a5     ld1w    {z0.s}, p0/z, [x11, x8, lsl #2]
40058c:     41 41 48 a5     ld1w    {z1.s}, p0/z, [x10, x8, lsl #2]
400590:     40 41 60 85     ld1w    {z0.s}, p0/z, [x10, z0.s, sxtw #2]
400594:     00 00 a1 04     add     z0.s, z0.s, z1.s
400598:     80 41 48 e5     st1w    {z0.s}, p0, [x12, x8, lsl #2]
40059c:     e8 e3 b0 04     incw    x8
4005a0:     00 1d a9 25     whilelo p0.s, x8, x9
.........
```


arm Research

SVE Memory Tracing Client

- Based on the existing DR *memtrace_simple* client
- AArch64 tracing done by DR
- SVE tracing is done separately by ArmIE

Trace inside a Region-of-interest (RoI) through markers in the code

```
#define __START_TRACE() { asm volatile (".inst 0x2520e020"); }
#define __STOP_TRACE()  { asm volatile (".inst 0x2520e040"); }
```

- Two trace files are generated (AArch64 and SVE trace files)
  - A shared synchronised counter is used to number ArmIE and DR traces in order to record the correct temporal sequence between the two traces
  - Merging both traces (post-process) results in the complete trace inside the RoI

arm Research

SVE Memory Tracing Client

- Memtrace format:

sequence number	Thread ID	SVE Bundle	isWrite	Data size	Data address	PC
Shared counter that ensures correct trace order		Identifies a contiguous access or a gather/scatter bundle (and the element position)	Write/read operation			

- ArmIE command example (512-bit vector):

```
$ armie -e libmemtrace_sve_512.so -i libmemtrace_simple.so -- ./sve_example
```

  - AArch64 output => memtrace.sve_example.26213.0000.log
  - SVE output => sve-memtrace.sve_example.26213.0000.log

Post-processing Traces

arm Research

Merging Traces

- AArch64 and SVE memory traces facilitate merging and analysis
  - Counter parameter for merging
  - RoI markers are included in the SVE trace to prune unwanted traces (same format as the trace)
    - ThreadID field is **-1** for the RoI start `19, -1, 0, 1, 0, (nil), (nil)`
    - ThreadID field is **-2** for the RoI stop `71, -2, 0, 1, 0, (nil), (nil)`

- A different separator after the counter identifies the origin of the trace
  - Colon (:) symbol for AArch64 traces `14: 0, 0, 0,  8, 0x41fde8, 0x4005fc`
  - Comma (,) symbol for SVE traces `20, 0, 0, 0, 64, 0x4200d4, 0x40058c`

arm Research

Parsing Traces

- The merged traces can be parsed to better understand the memory accesses
  - Higher verbosity / better readability

Occurrence count

Type Label

```
--- L 1.#1 ( 64B) [0x4200d4] @ 0x40058c – vec_util = 100%
--- L 2.#1 ( 64B) [0x42002c] @ 0x400590 – vec_util = 100%
=== L 3.#1 Bundle #0 started (  4B) [0x42002c] @ 0x400594
=== L 3.#16 Bundle #0 ended [0x42002c] -> 16 accesses ( 64B) – vec_util = 100%
  ^ The bundle only has contiguous accesses
  ^ Bundle stride = 4B
  ^ The following addresses were accessed repeatedly in the bundle:
    > 0x42002c, x16
--- S 1.#1 ( 64B) [0x42017c] @ 0x40059c – vec_util = 100%
```

Bundle info

Size Address PC

Load/Store

arm Research

Parsing Traces

- Parser scripts can be done to better understand the memory accesses
  - Higher verbosity / better readability

Detailed gather/scatter bundle

```
=== L 13.#17 Bundle #1 started ( 4B) [0x420070] @ 0x40061c
  . L 13.#18 Bundle #1 ( 4B) - access #2 [0x420074]
  . L 13.#19 Bundle #1 ( 4B) - access #3 [0x420078]
  . L 13.#20 Bundle #1 ( 4B) - access #4 [0x42007c]
                          ...
  . L 13.#31 Bundle #1 ( 4B) - access #15 [0x4200a8]
  . L 13.#32 Bundle #1 ( 4B) - access #16 [0x4200ac]
=== L 13.#32 Bundle #1 ended [0x4200ac] ->16 accesses ( 64B)
```

Summary statistics

```
===== Memory References:
   -> non-bundle SVE: 9 (17.65%)
   ->     bundle SVE: 42 (82.35%)
   ->        non-SVE: 0 (0.00%)
Total: 51

===== Writes:
   -> non-bundle SVE: 3 (100.00%)
Total: 3 writes
       1 unique writes [different PCs]

===== Loads:
   -> non-bundle SVE: 6 (12.50%)
   ->     bundle SVE: 42 (87.50%)
Total: 48 loads
       3 unique loads [different PCs]
```

SVE Cache Simulator

- A simple modular cache simulator for SVE memory traces
- Currently supports a single-core multi-level cache system
  - ArmIE SVE tracing has compatibility issues with multithreaded applications
- Supports prefetcher plugins
  - A simple stride prefetcher is currently available

Initial Study

arm Research

HPCG SVE Utilisation

- Inscount client counts the full application (no shared libraries)
- SVE instruction utilisation:

Vector lengths (bits)	% SVE instructions
128	69.92 %
256	60.06 %
512	50.26 %
1024	38.66 %
2048	28.40 %

- Memtrace client considers only the RoI
- Average Vector Utilisation in memory accesses (512-bit vectors):
- **69.6% vector use** (44.54B out of 64B vectors)

arm Research

HPCG benchmark

- We use an optimised version of HPCG
  - Multiblock colour reordering to parallelize the Symmetric Gauss-Seidel
  - https://gitlab.com/arm-hpc/benchmarks/hpcg/tree/dynamic_slice_coloring
  - Compiled with Arm HPC compiler 18.4
  - Flags: `-O3 -march=armv8-a+sve -ffast-math -DHPCG_NO_MPI -DHPCG_NO_OPENMP -DHPCG_CONTIGUOUS_ARRAYS`

- To reduce the memory trace size for larger inputs, the used RoI corresponds to a single iteration of the optimised *CG* kernel
- Input size = 32^3

SVE Cache Simulator

- 2-level cache model
- L2 is non-inclusive & non-exclusive of L1
- Memory latency refers to the round-trip to memory

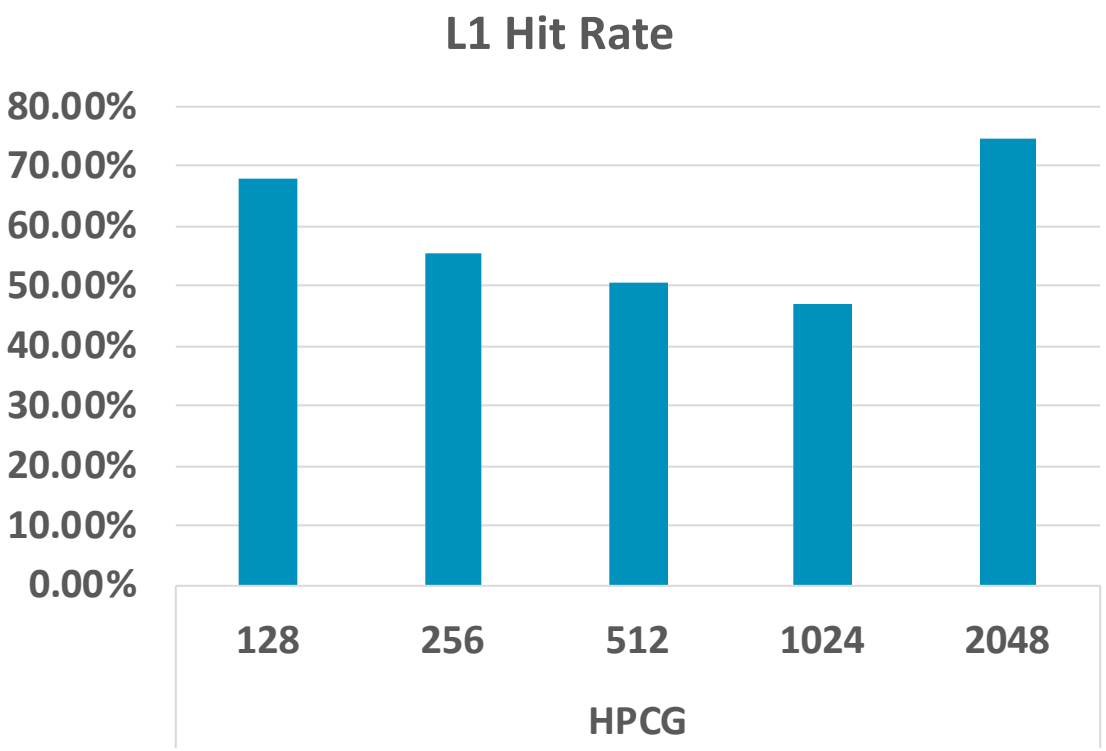
- Stride prefetcher (optional)
  - History of last 100 accesses
  - Fetches to L2

Model Parameters	L1 cache	L2 cache
Cache Size (B)	32K	4M
Line Size (B)	64	64
Set Size (ways)	8	8
Word Size (B)	4	4
Latency (cycles)	4	8
Mem_latency (cycles)	100	100

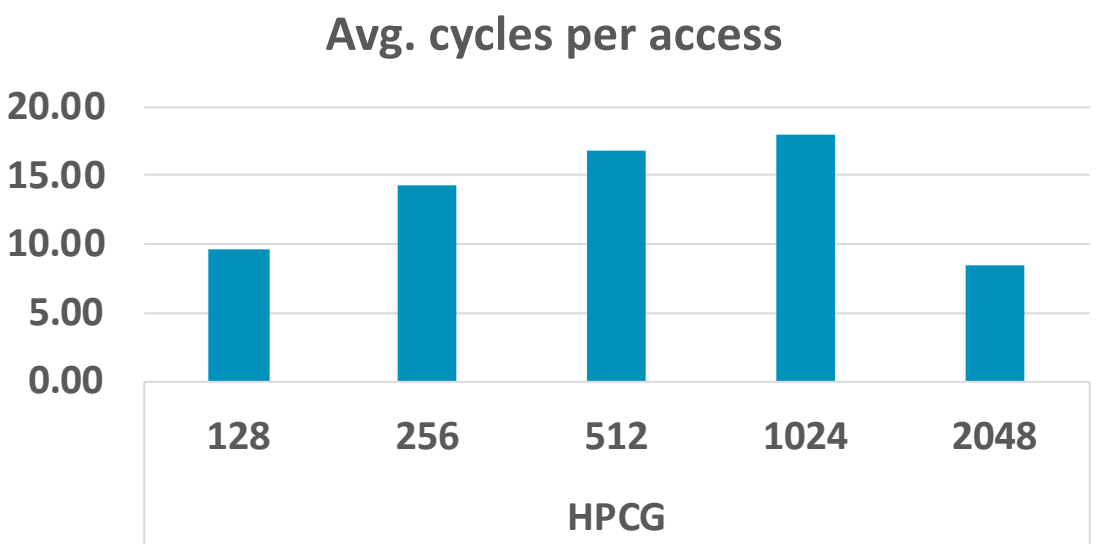
arm Research

SVE Memory Impact

- CacheSim HPCG Results (512-bit vectors)



Base Parameters	L1 cache	L2 cache
Cache Size (B)	64K	2M
Line Size (B)	16	16
Set Size (ways)	2	8
Word Size (B)	4	4
Latency (cycles)	4	11
Mem_latency (cycles)	100	100



arm Research

What do the numbers mean?

- In app performance numbers (e.g. clock_gettime())
  - Counting cost of emulation, no real indication of performance!
- Numbers based on the cache simulator
  - These are estimates, as the cache simulator is a reasonable approximation of real HW, but it's by no means 100% accurate.
- % SVE vectorization
  - Reminder: if SVE used, the wider the unit, the lower the total number of executed instructions (and potentially %SVE vectorization).
- How do they relate to code quality and performance numbers?
  - Let's have a conversation about this!

arm Research

ArmIE Roadmap

- ArmIE users are able to write their own instrumentation clients separate from emulation clients using an emulation API

- ArmIE instrumentation is fully compatible with single-threaded apps and, as of the latest version (18.4), there is multi-threaded support too.

- SVE DR CacheSim/open-source internal cache simulator planned for future version.

Final Remarks

- ArmIE enables SVE studies in preparation for upcoming silicon
  - It enables vector length considerations for future micro-architectures and application optimisations
  - As an emulator, it allows for faster application runs and quicker development turnaround
- We include an instruction counter and a memory tracing clients with ArmIE 18.4
  - An emulation API is scheduled for the next version of ArmIE (18.3)
  - People will be able to easily build their own clients and perform their own studies
- Post-processing traces extend application analysis (scripts, cache simulator, etc.)
  - Anyone can write their own scripts (we are exploring ways to share some of our post-processing tools with Partners)
- ArmIE guide: https://bit.ly/2zzf53n
- HPCG used: https://bit.ly/2yXP0eU

Thank You!
Danke!
Merci!
谢谢!
ありがとう!
Gracias!
Kiitos!
Obrigado!

arm Research