arm

Asvie: A timing-agnostic SVE optimization methodology

roxana.rusitoru@arm.com

Roxana Rusitoru, Miguel Tairum Cruz, Daniel Ruiz

ProTools, 11/17/2019

Main takeaways

The things you should remember after this talk

- HPCG is a key HPC benchmark and we embarked on a journey of optimizing it for Arm and SVE.

- For this, we used ArmIE and DynamoRIO, applied our secret sauce (custom plugins and post-processing tools), and produced “Asvie: The Methodology” to achieve our optimization goals.

- You too can do this today!
  - https://github.com/ARM-software/HPCG_for_Arm
  - https://github.com/ARM-software/Methodology_for_ArmIE_SVE
- Or later this week:
  - https://pages.arm.com/sve-hackathon-sc19

arm

Asvie

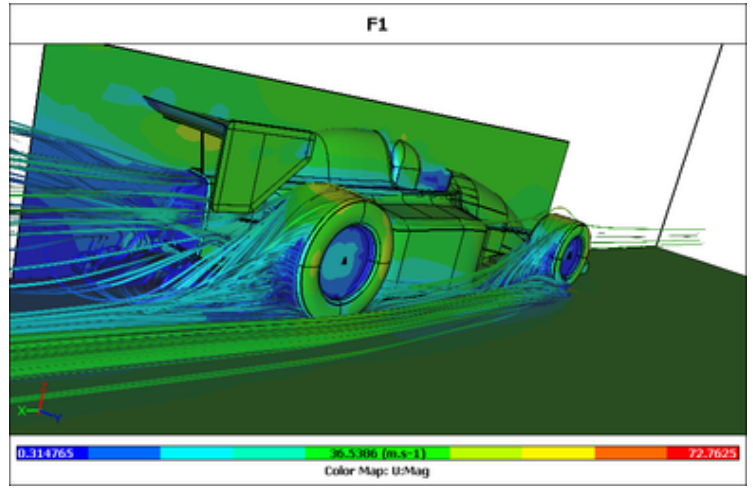
What it’s actually all about

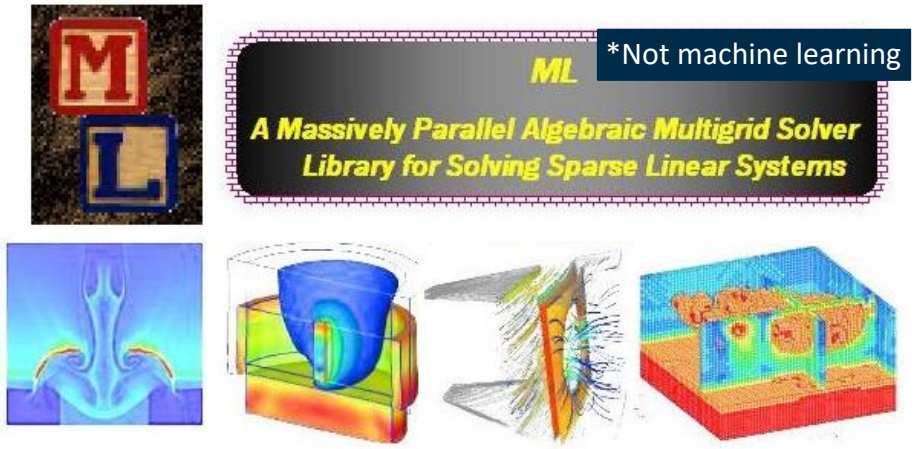
- HPCG introduction [What is this and why do we care?]
- Tools [Our hammer to help shape HPCG]
  - DynamoRIO + ArmIE + scripts
- Methodology [How the hammering takes place]
  - Optimizing for SVE using ArmIE
  - Application analysis and metrics of interest
    - Compiler auto-vectorization analysis
    - Dynamic instruction counts and SVE utilization
    - Vector lane utilization
    - Memory load and store types
    - Floating-point operations per byte
    - Cache simulator statistics

HPCG

What it is and why we all care so much about this

- Benchmark for ranking Top500 HPC systems.
- HPCG's kernels are representative of real-world scientific applications ran on HPC machines, computationally and data access pattern-wise.
  - E.g., Computational fluid dynamics (OpenFOAM), computational photography.
- Is not only about the FLOPS!
  - Don't worry, still used as Figure of Merit ☺
  - Along with the memory bandwidth





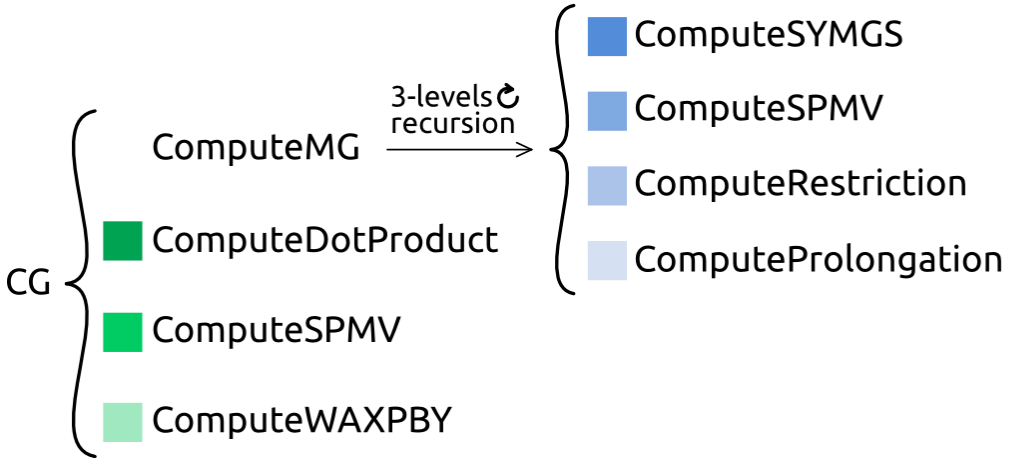
arm

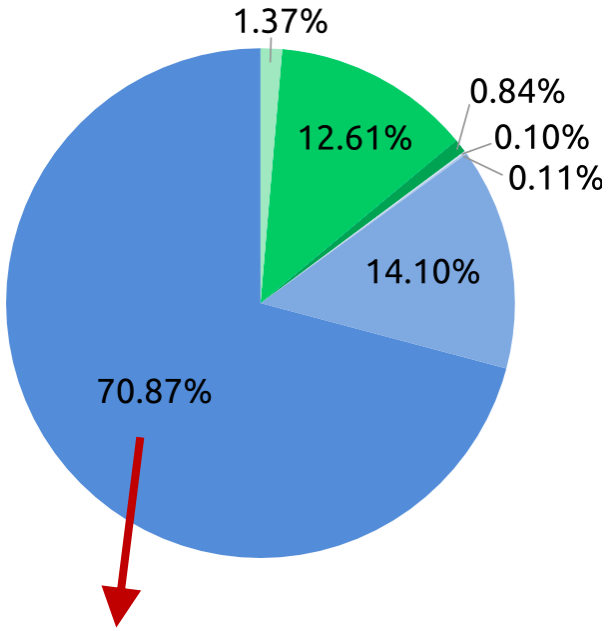
HPCG

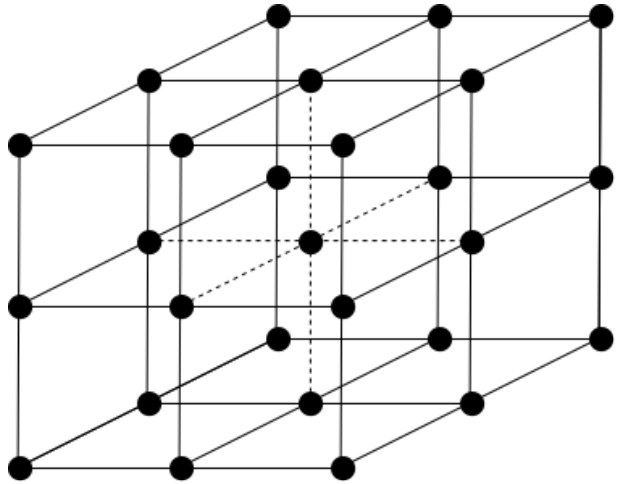
Highly-demanded application, very important for HPC, Top500 rankings

- Solves the linear system

$$A * x = r$$







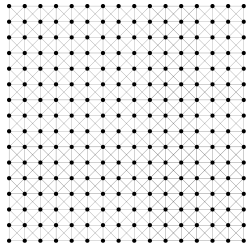
Not parallelizable!

How can we do better?

arm

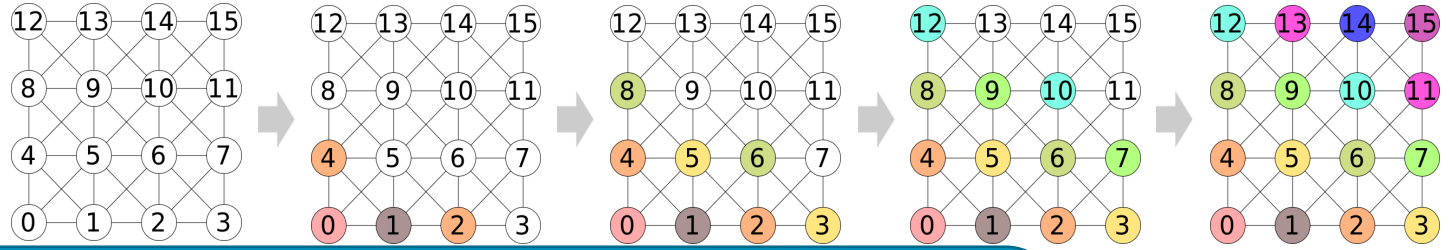
All the (parallelism) optimizations!

Merging all together



Finest level

(multi-level task dependency graph)

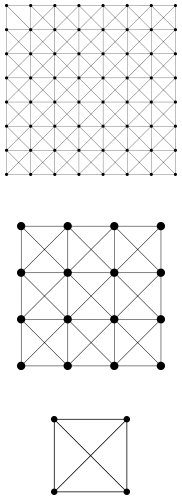


Further information about our code in the Arm blog:

http://bit.ly/2ZtstSb

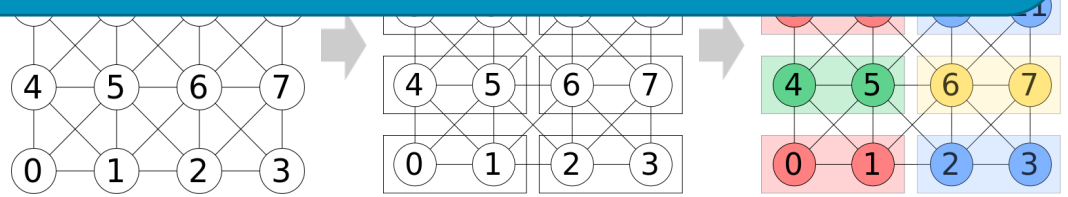
Work already presented in SC'18 HPCG BoF:

http://bit.ly/2WBrNwV



Coarser levels

(block multi-coloring)

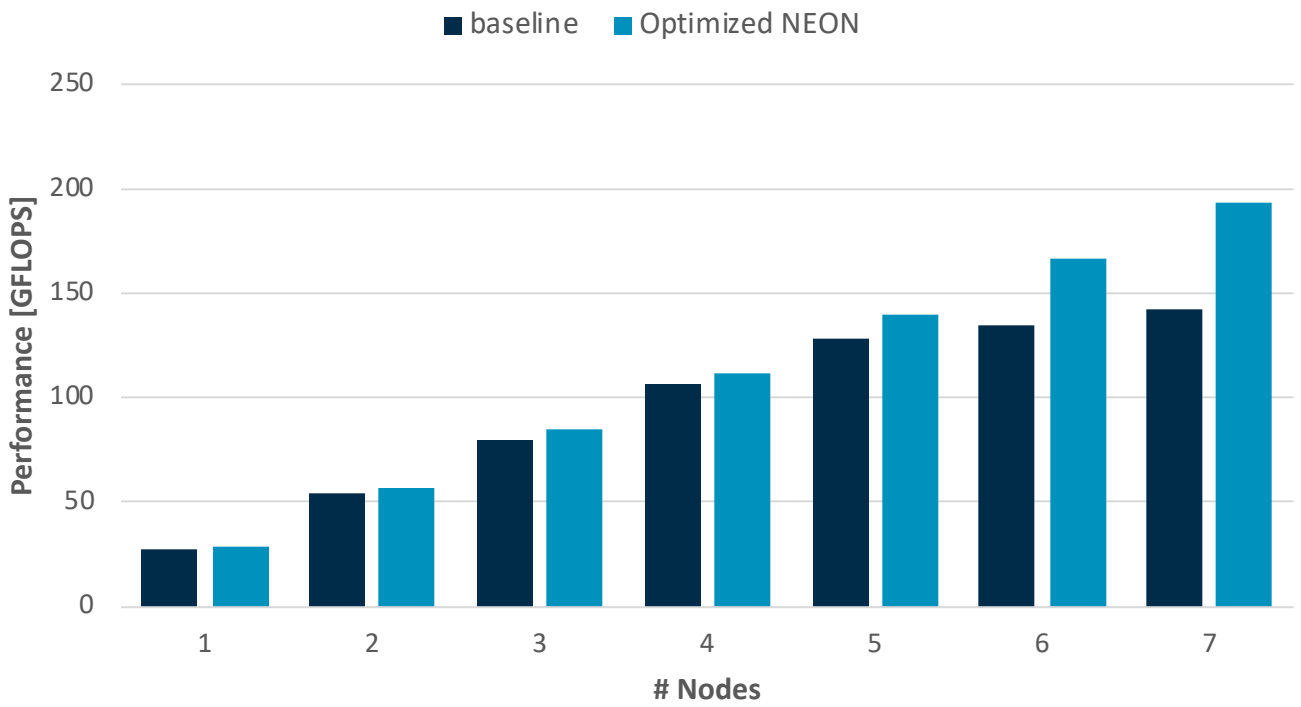


arm

Putting the optimizations to the test

HPCG performance on a ThunderX2 cluster

- Parallelizing the main kernels improves application scalability.
- Higher gap in performance expected at higher core count.
- Results presented at SC18
  - Positive feedback from the community.



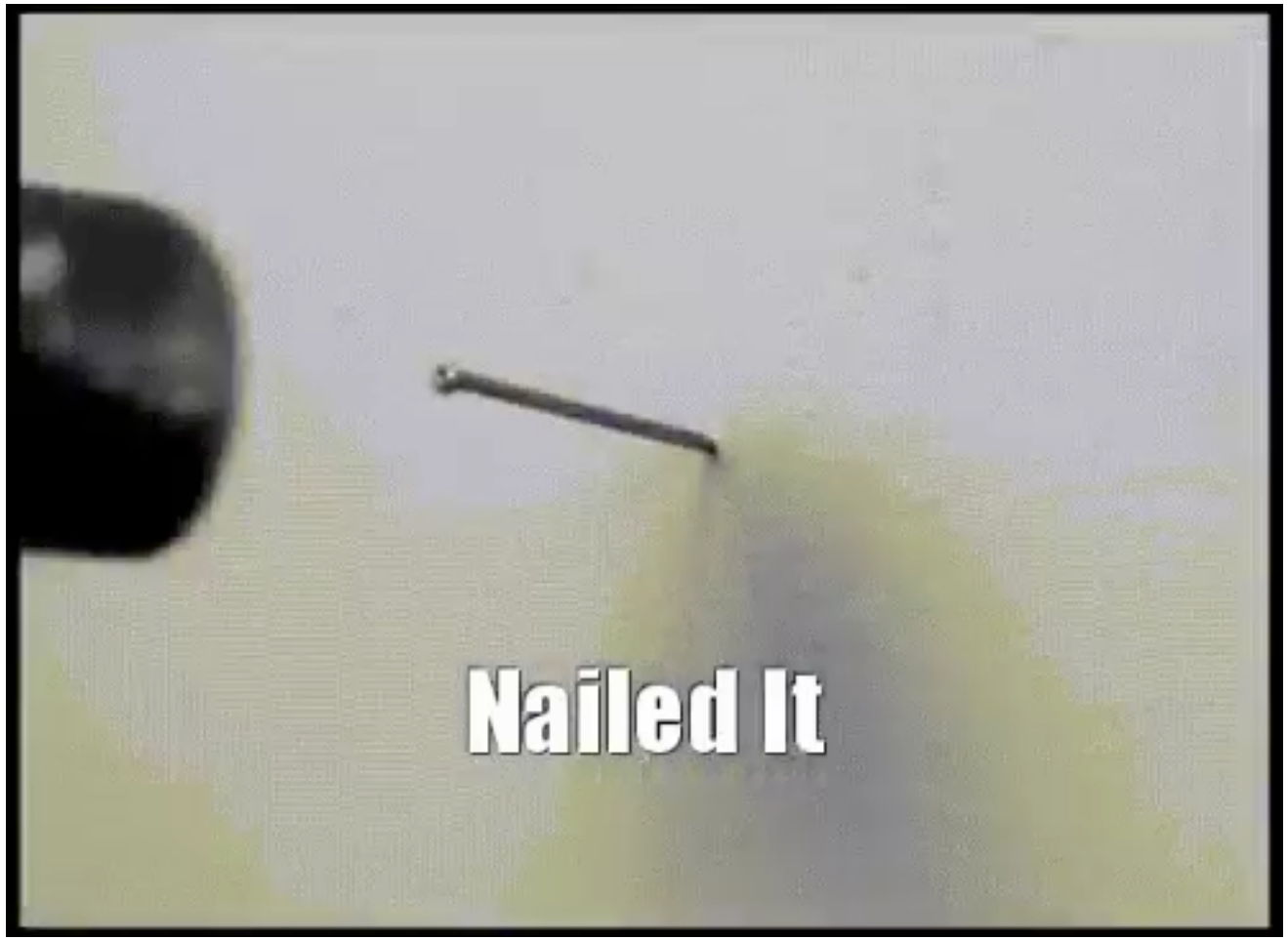
Baseline (MPI-only):
- 56 MPI ranks per node

Optimized NEON:
- 8 MPI ranks per node
- 7 OpenMP threads per MPI rank

arm

Success!

Or is it?



We increased shared memory parallelism, at the cost of single core performance.

Next steps in optimizing HPCG

Back to the drawing board

- Parallelism achieved, so then we looked towards fixing what we “broke”.
- The result?

  SVE-optimized (w/ intrinsics)
  HPCG implementation.

- But how?
  There’s no hardware, no proper performance models, and little in the way of simulators that can run big applications.

“I was wondering when you’d notice there’s lots more steps.”

arm

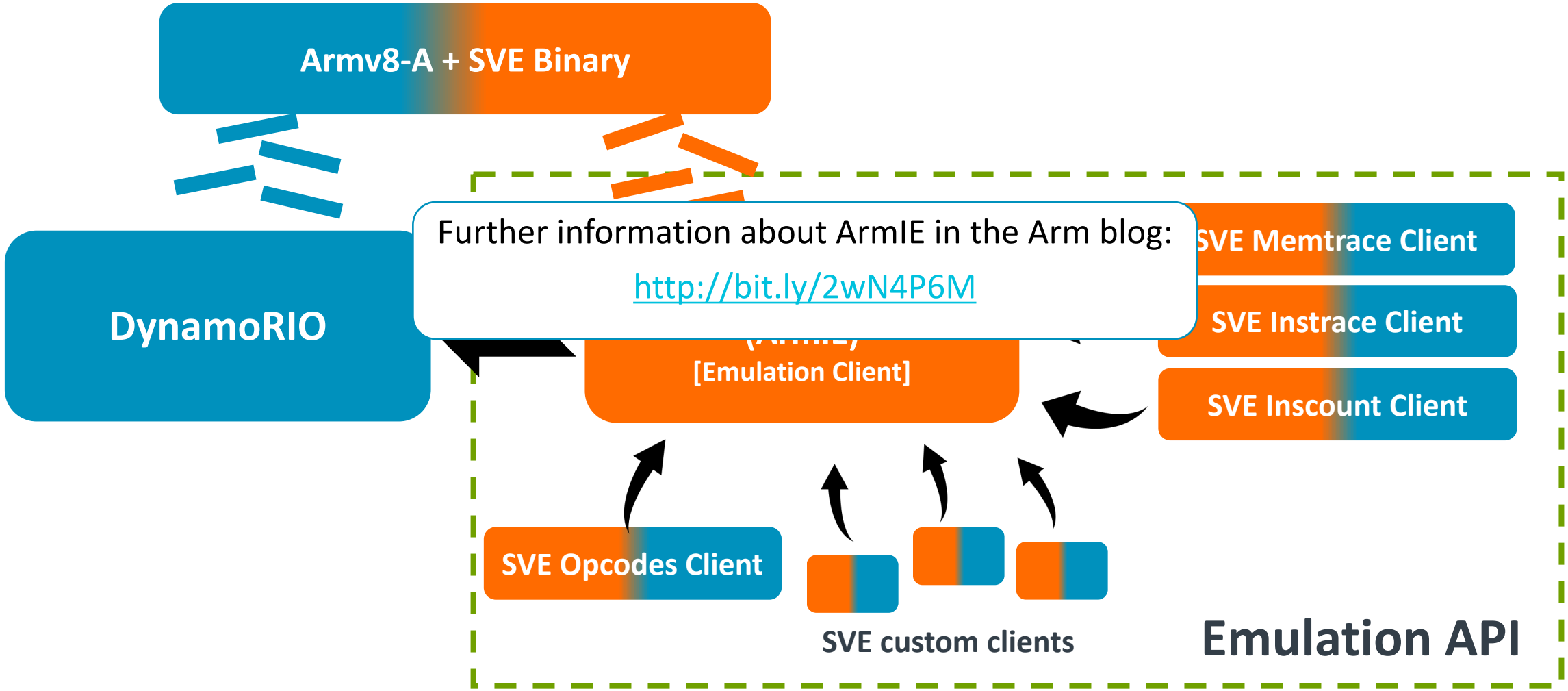
arm

Tools

DynamoRIO & ArmIE

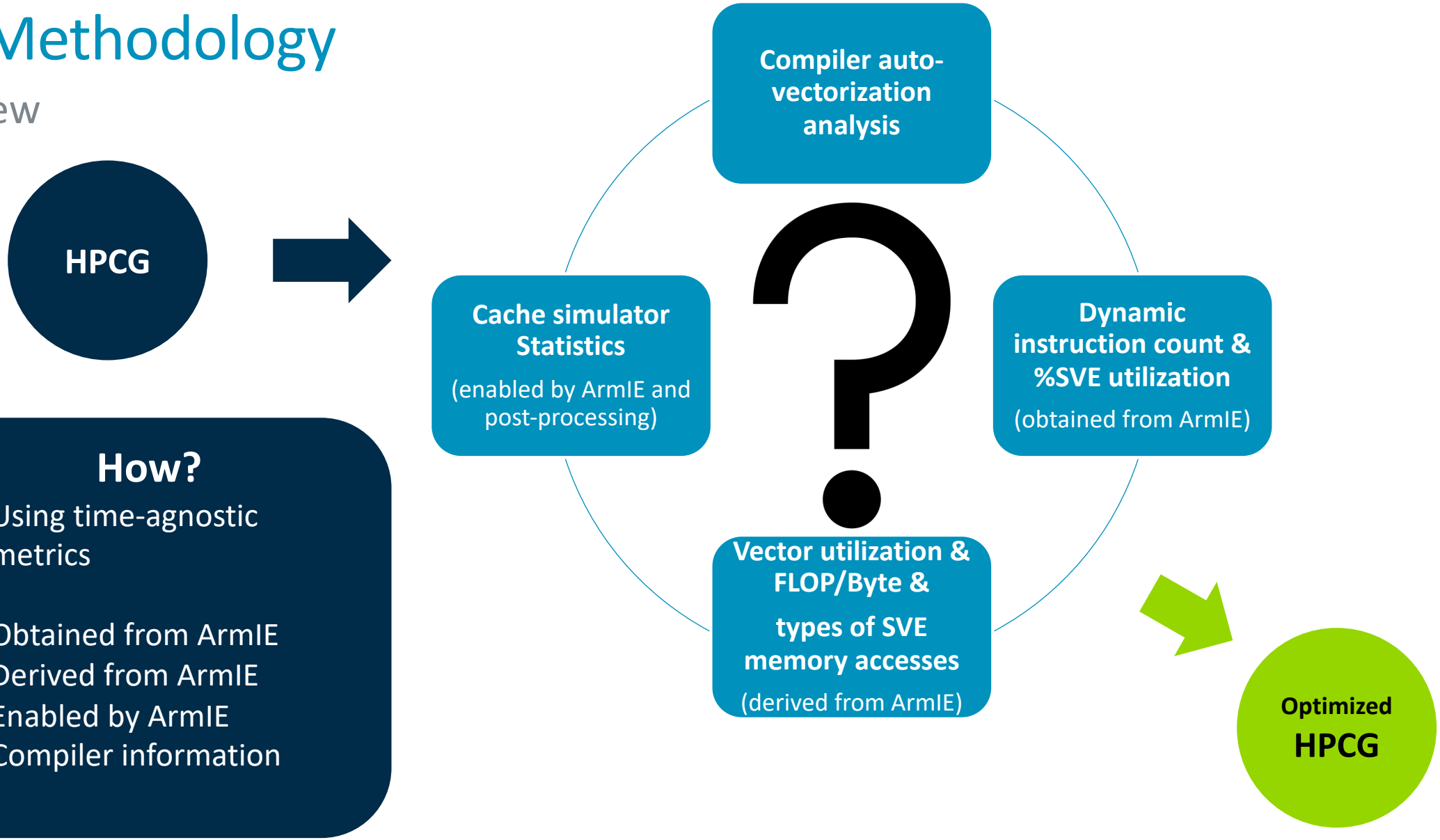
DynamoRIO & ArmIE

Or how we learned to stop worrying and love SVE



The Methodology

Overview



How?

- Using time-agnostic metrics

- Obtained from ArmIE
- Derived from ArmIE
- Enabled by ArmIE
- Compiler information

arm

Why ArmIE?

And not <insert tool here>?

Because ArmIE is:

- ✓ Fast functional emulator
  (enables apps with large inputs runs)
- ✓ Easy to use and develop
  (allows custom instrumentation and post-processing)
- ✓ Freely available
- ✓ Partly open-source
  (API to build your own instrumentation)

ArmIE is not:

- X Cycle accurate
  (no timing information)
- X A simulator
  (Requires Armv8 hardware)
- X Architecture modelling
  (Is all about the apps)

arm

arm

Methodology

Where all the magic happens

The purpose of Asvie

And how this links to the tools

- **Be able to optimize applications for SVE in the absence of a tuned performance model.**

How? Through time-agnostic metrics:

- Obtained from ArmIE
  - Dynamic instruction reduction & instruction mix [from instrumentation clients]
- Derived from ArmIE
  - Vector utilization [from memory trace analysis]
  - Floating point operations per byte [from instruction and memory trace analysis]
- Enabled by ArmIE
  - Memory system statistics (cache stats, average cycles) [obtained on a cache simulator]

What we ran

A trilogy of HPCG flavors

3 HPCG versions:

- **Baseline** – a.k.a. the reference version
  - What everyone gets to try out.

- **Optimized - No intrinsics** – the version with shared memory parallelism
  - Multi-Level Task dependency and Block coloring optimizations.
  - Other minor performance improvements (loop fusion/unroll, memory allocations, etc.).

- **Optimized - With intrinsics** – further single-thread hand-tuned optimizations
  - And now with SVE intrinsics™.
  - Fully vectorized.

- All versions are compiled with the Arm HPC Compiler 19.0.

arm

Compiler auto-vectorization analysis

Part 1: Checking out the competition and setting expectations

Compiler		# Vectorized loops	
		baseline	no-intrinsics
SVE	GCC 8.2.0	8/8	12/17
	Arm HPC Compiler 19.0	8/8	12/17
AVX2	Intel Compiler 19.0.3	8/8	17/17

- GCC and Arm HPC Compiler:
  - Reverse loops are not vectorized.
- Intel Compiler:
  - Considers outer loops, ends up vectorizing inner loops.

```
for ( int j = 0; j < nnzInChunk; j++ ) {
  sum0 -= mtxVal[i+0][j] * xv[ mtxInd[i+0][j] ];
  sum1 -= mtxVal[i+1][j] * xv[ mtxInd[i+1][j] ];
  sum2 -= mtxVal[i+2][j] * xv[ mtxInd[i+2][j] ];
  sum3 -= mtxVal[i+3][j] * xv[ mtxInd[i+3][j] ];
}
```

Arm HPC Compiler ✓

GCC ✓

```
for ( int j = nnzInChunk-1; j >= 0; j-- ) {
  sum3 -= mtxVal[i-3][j] * xv[ mtxIndL[i-3][j] ];
  sum2 -= mtxVal[i-2][j] * xv[ mtxIndL[i-2][j] ];
  sum1 -= mtxVal[i-1][j] * xv[ mtxIndL[i-1][j] ];
  sum0 -= mtxVal[i-0][j] * xv[ mtxIndL[i-0][j] ];
}
```

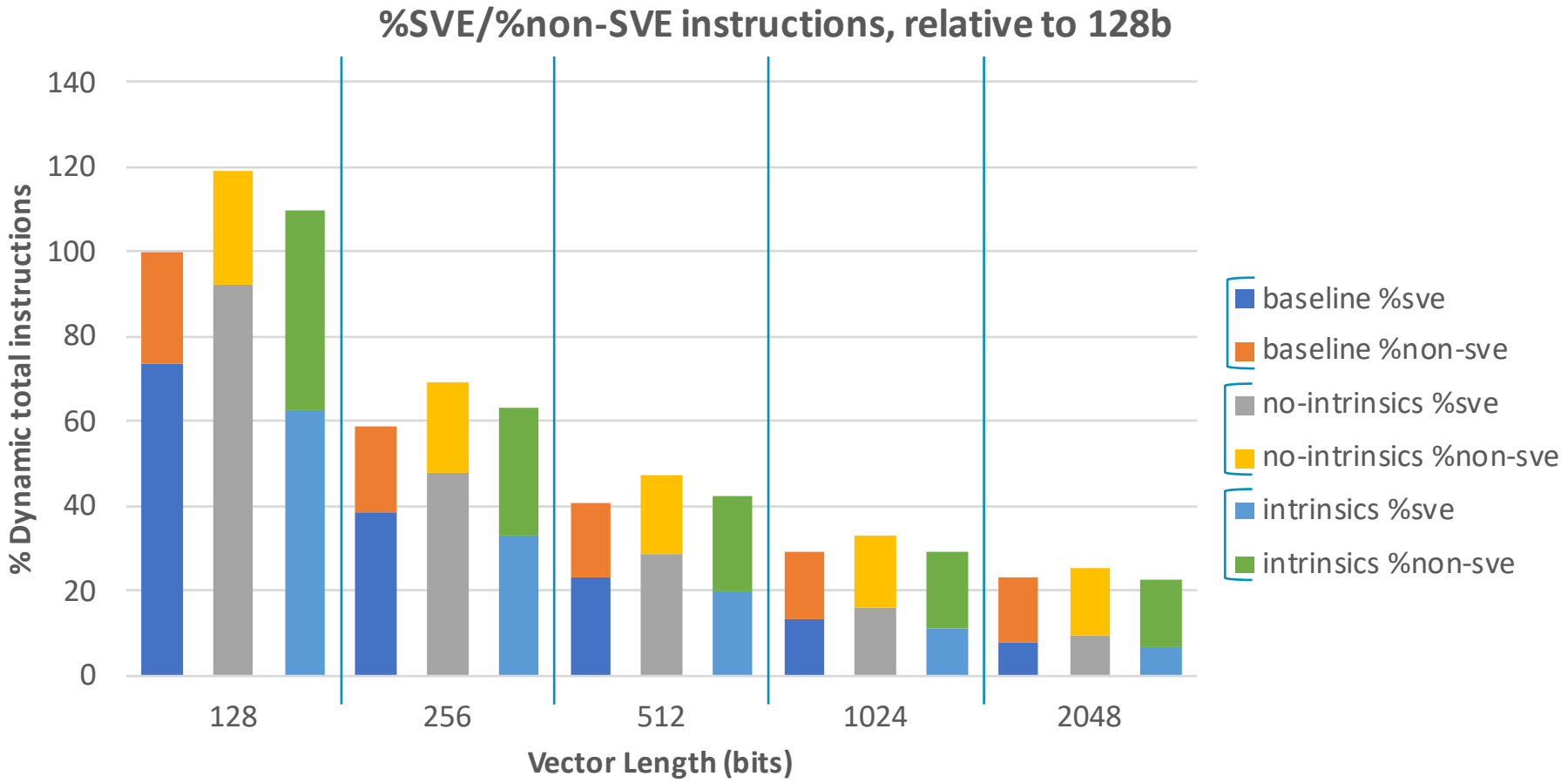
Arm HPC Compiler ✗

GCC ✗

Cue the ArmIE

Part 2: Running the applications through our instrumentation clients

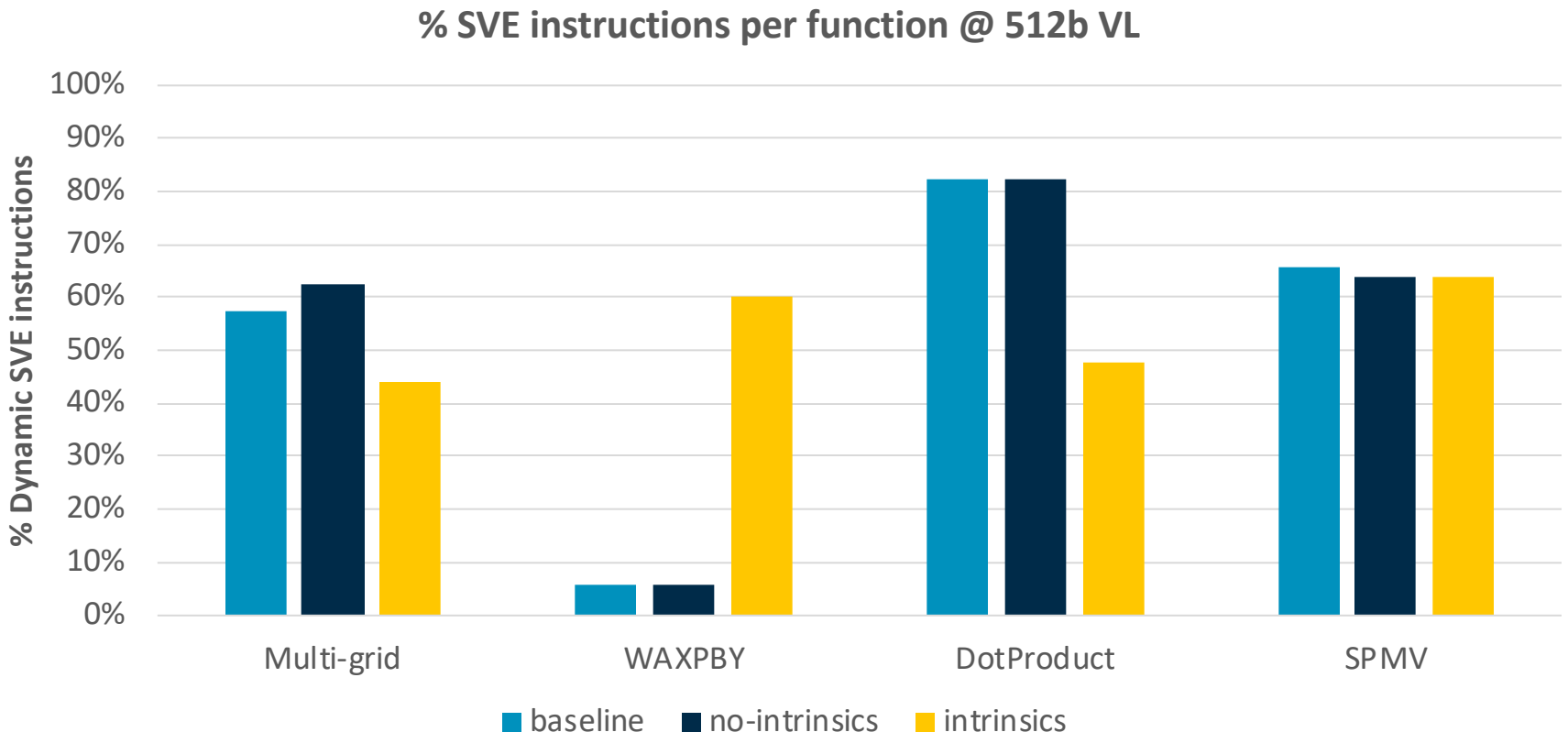
- Instruction count client



Cue the ArmIE

Part 2: Running the applications through our instrumentation clients

- Instruction count client



arm

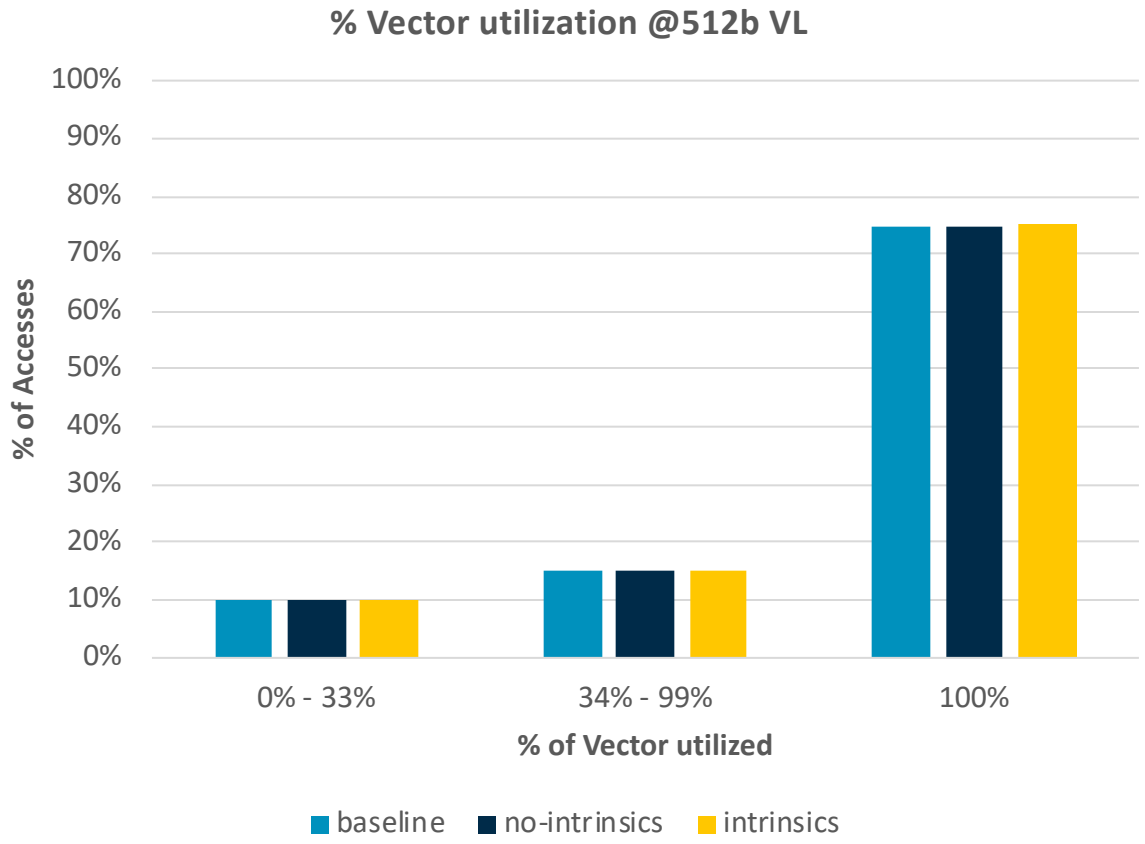

WE NEED TO GO
DEEPER
memegenerator.net

Squeeze the analysis juice

Part 3: Getting more from ArmIE with post-processing analysis

- Vector utilization [extracted from memory traces]

Version	Avg. Vector utilization
baseline	82.35%
no-intrinsics	82.39%
intrinsics	82.55%



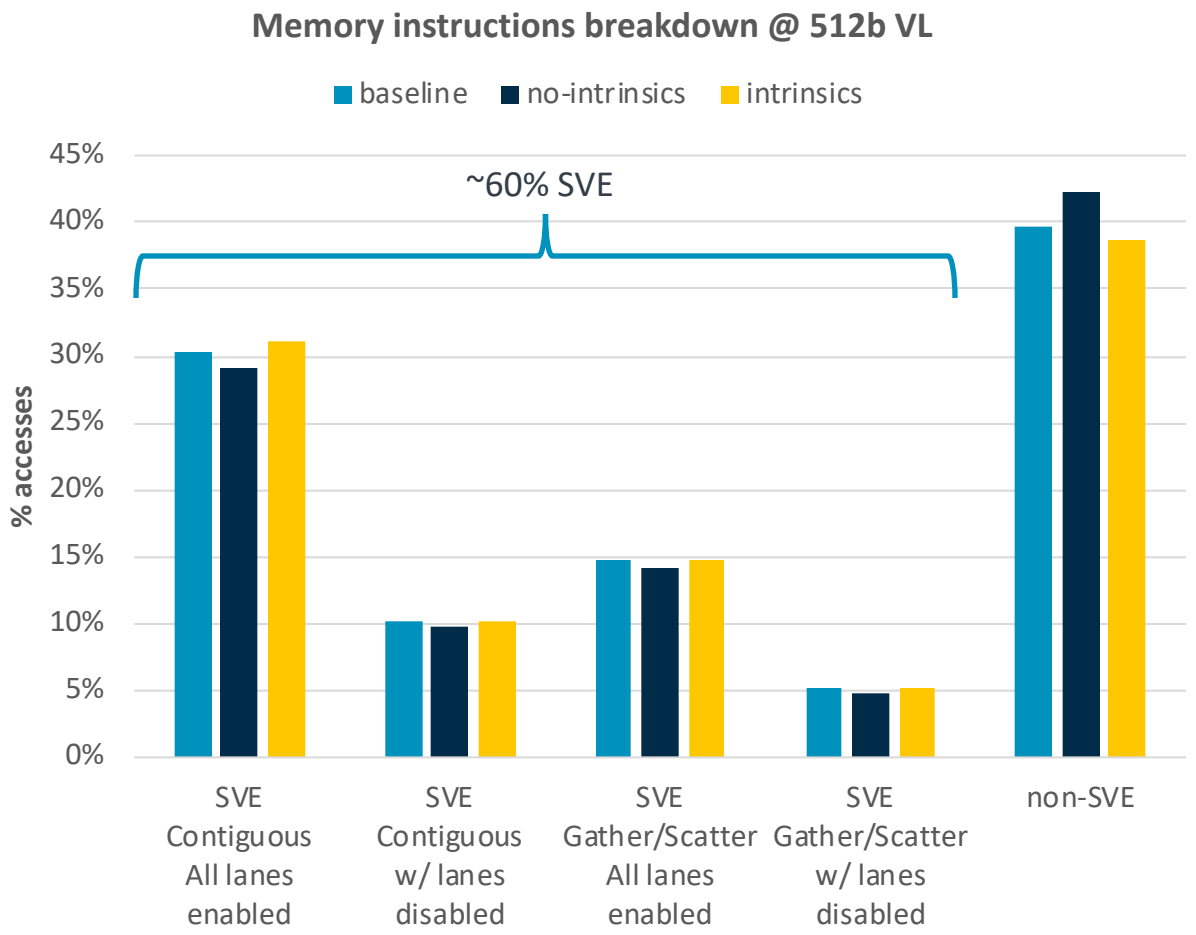
arm

Squeeze the analysis juice

Part 3: Getting more from ArmIE with post-processing analysis

- Memory accesses present similar characteristics
  - Slightly more SVE accesses in the intrinsics version.

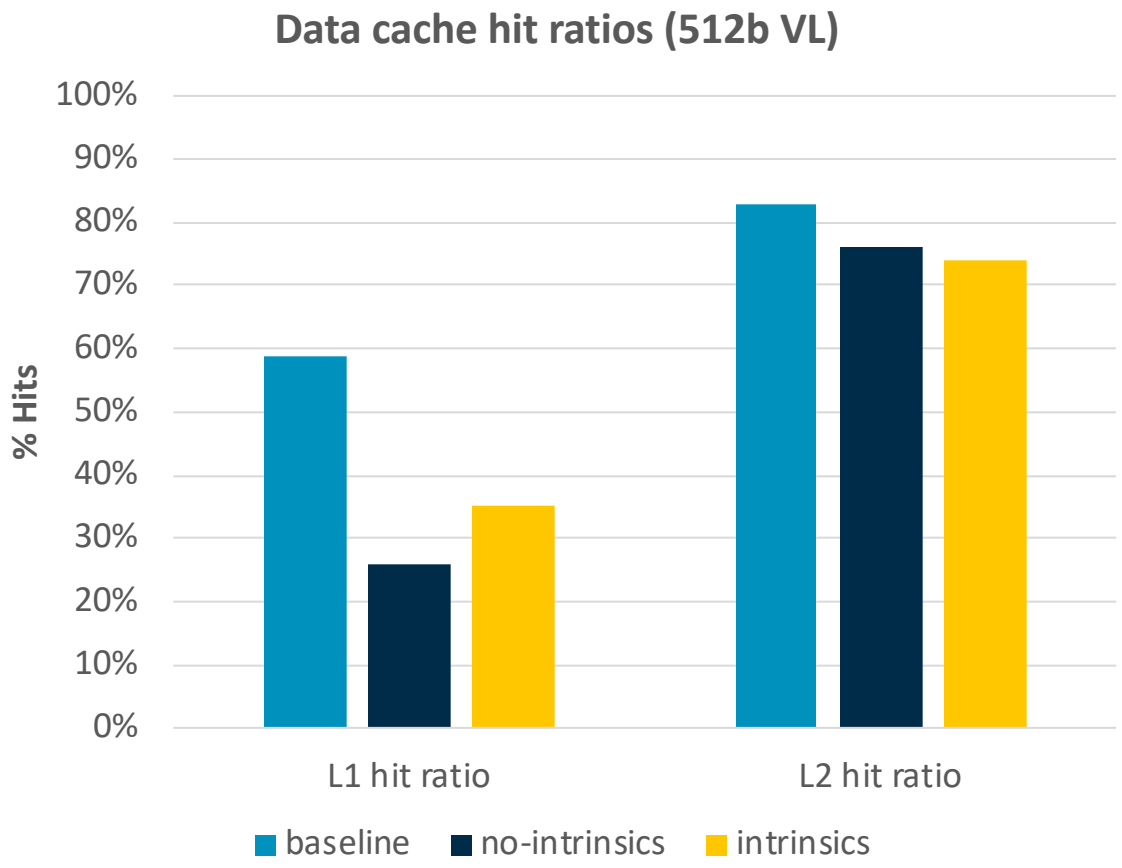
- Further work could be done to decrease the usage of gathers/scatters, replacing them with contiguous memory accesses.



arm

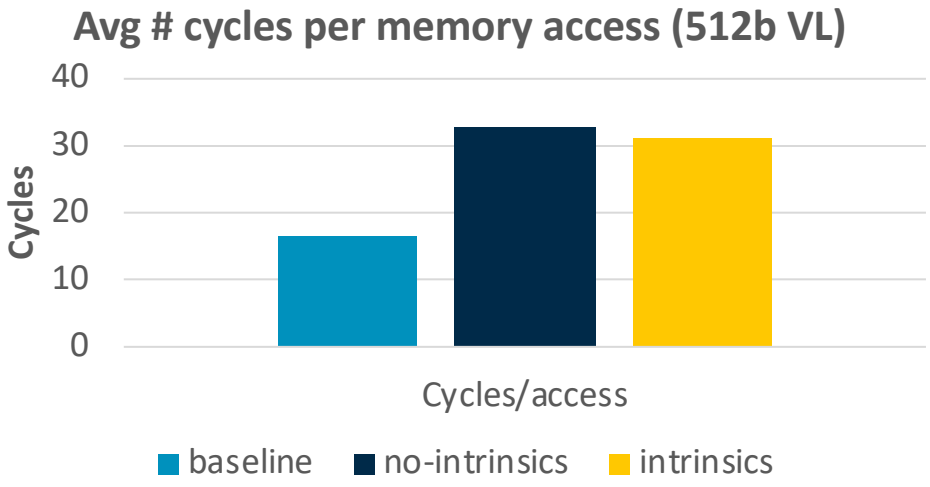
Squeeze the analysis juice

Part 4: What will the memory system hold in store for us?



Cache simulator parameters

	L1	L2	L3
Cache Size (KB)	64	1024	2048
Line Size (#words)	16	16	16
Word Size (Bytes)	4	4	4
Set Size (n-ways)	8	8	32
Latency (cycles)	4	11	60
Memory Latency (cycles)			156
Stride prefetcher to L1			

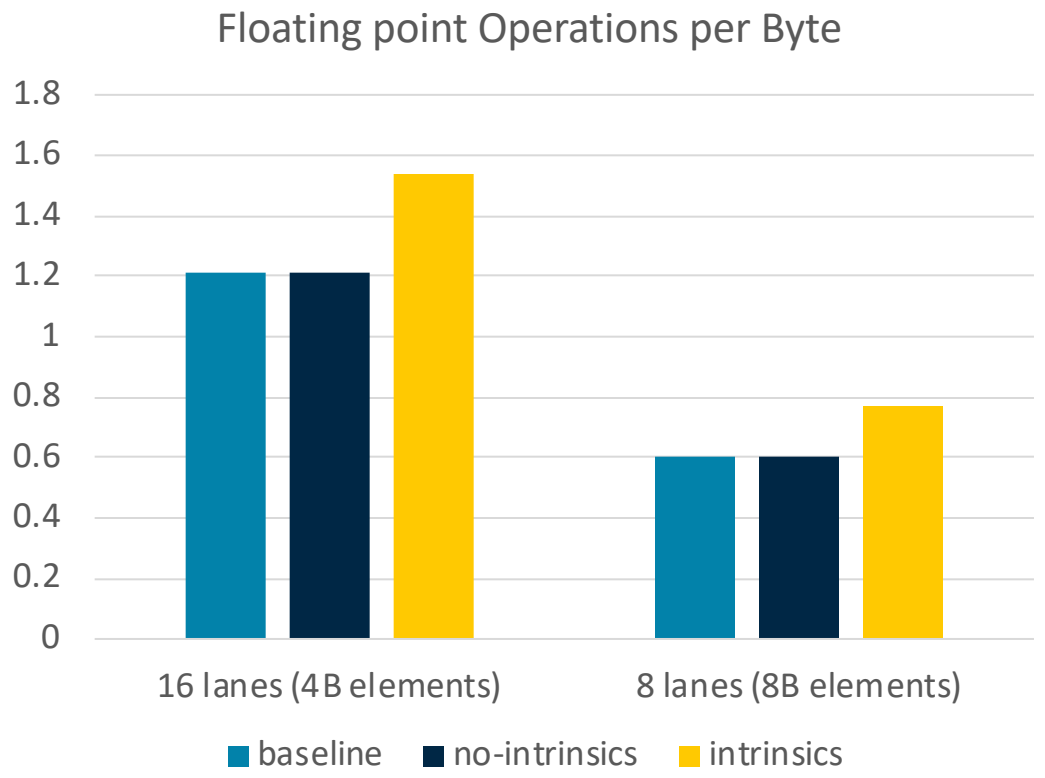


arm

Closest we can get to performance

Part 5: Ceiling FLOP/byte – need no further introduction

- Closest metric to timing-based performance estimates.
- Due to tool limitations (lack of precise vector lane information), we compute a ceiling value for the FLOP/Byte metric.
- Assumption: 100% lane utilization for FP vector operations.

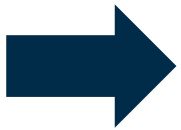


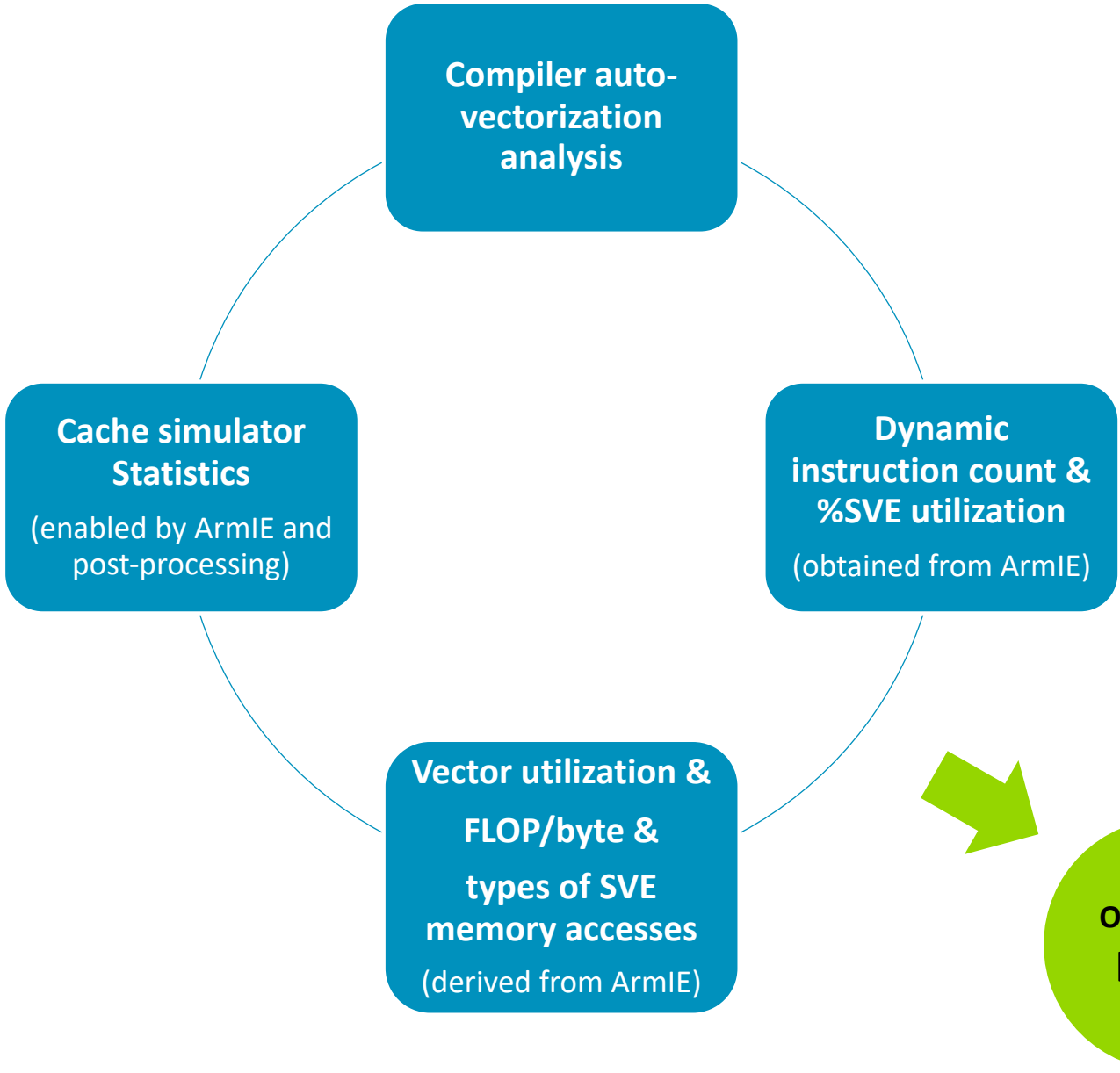
arm

Asvie

Overview

HPCG





Optimized HPCG

We improved:

- Dynamic instruction count
- Increase %SVE utilization
- Increase SVE contiguous memory accesses
- L1$ hit rate
- Decrease avg. #cycles per memory access
- Increase FLOP/byte

arm

The end of HPCG's SVE story

But Asvie is here to stay

HPCG w/ SVE (& NEON) and Asvie tools available on github!

In our successful journey to optimize HPCG we:

- Produced Asvie for optimizing applications for SVE in the absence of a tuned performance model or real hardware;
- Extended the tools (instrumentation for ArmIE) and developed scripts (post-processing, cacheSim) to enable the Methodology;
- Shared insights and helped improve other tools Arm offers (ArmPL, Arm HPC Compiler).

arm

arm

Thank You
Danke
Merci
谢谢
ありがとう
Gracias
Kiitos
감사합니다
धन्यवाद
شكرًا
תודה