arm

Optimizing HPCG for Arm SVE

SVE analysis and optimization methodology

Daniel Ruiz – Arm Research

Arm HPC User Group @ ISC19 - June 2019

I had the agenda all the time :D

https://developer.arm.com/solutions/hpc/resources/presentations

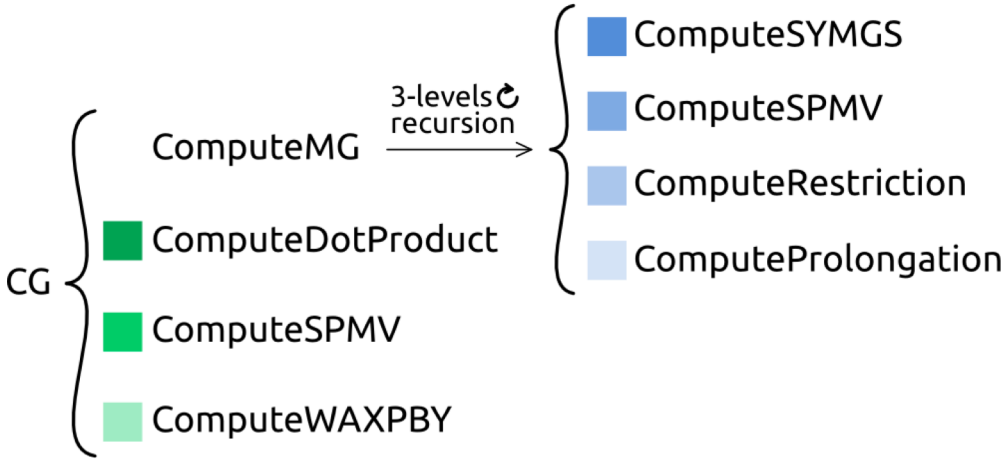
- ISC 2019
- Please notice that presentations are in order, but are time-agnostic

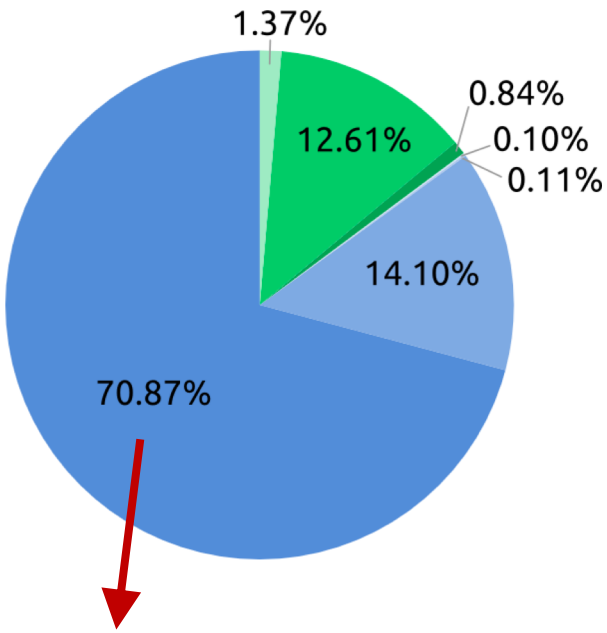
HPCG

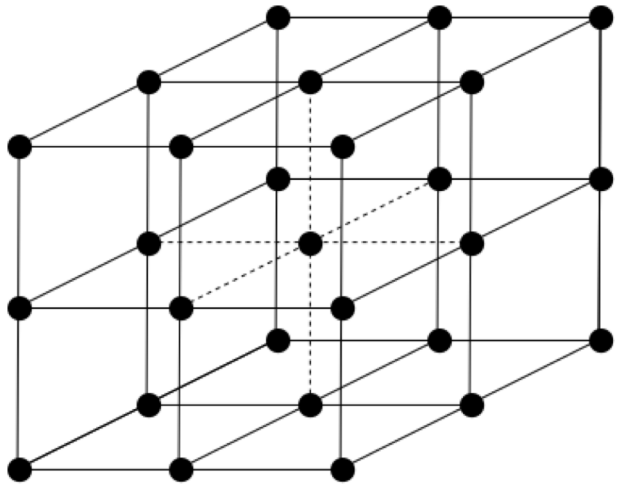
Highly-demanded application, very important for HPC, Top500 rankings

- Solves the linear system

$$A * x = r$$





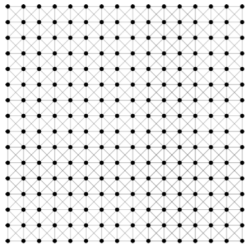


Not parallelizable!

How can we do better?

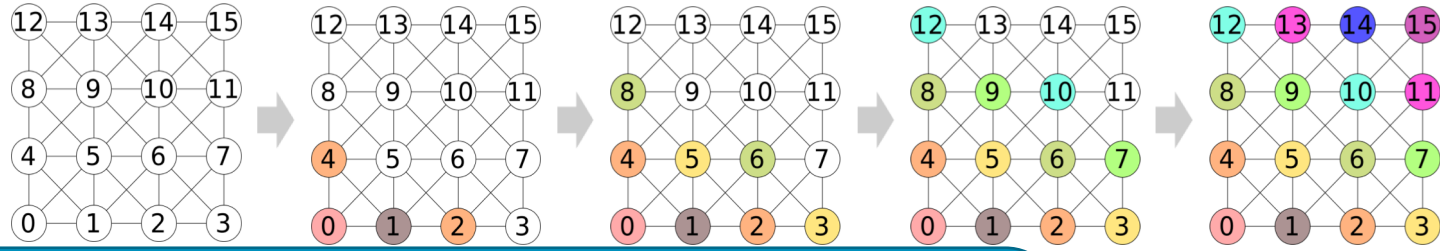
All the (parallelism) optimizations!

Merging all together



Finest level

(multi-level task dependency graph)

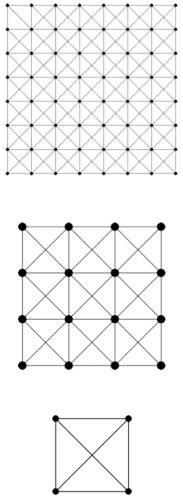


Further information about our code in the Arm blog:

http://bit.ly/2ZtstSb

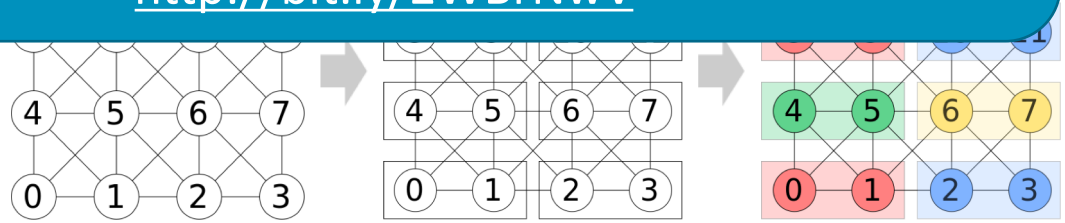
Work already presented in SC'18 HPCG BoF:

http://bit.ly/2WBrNwV



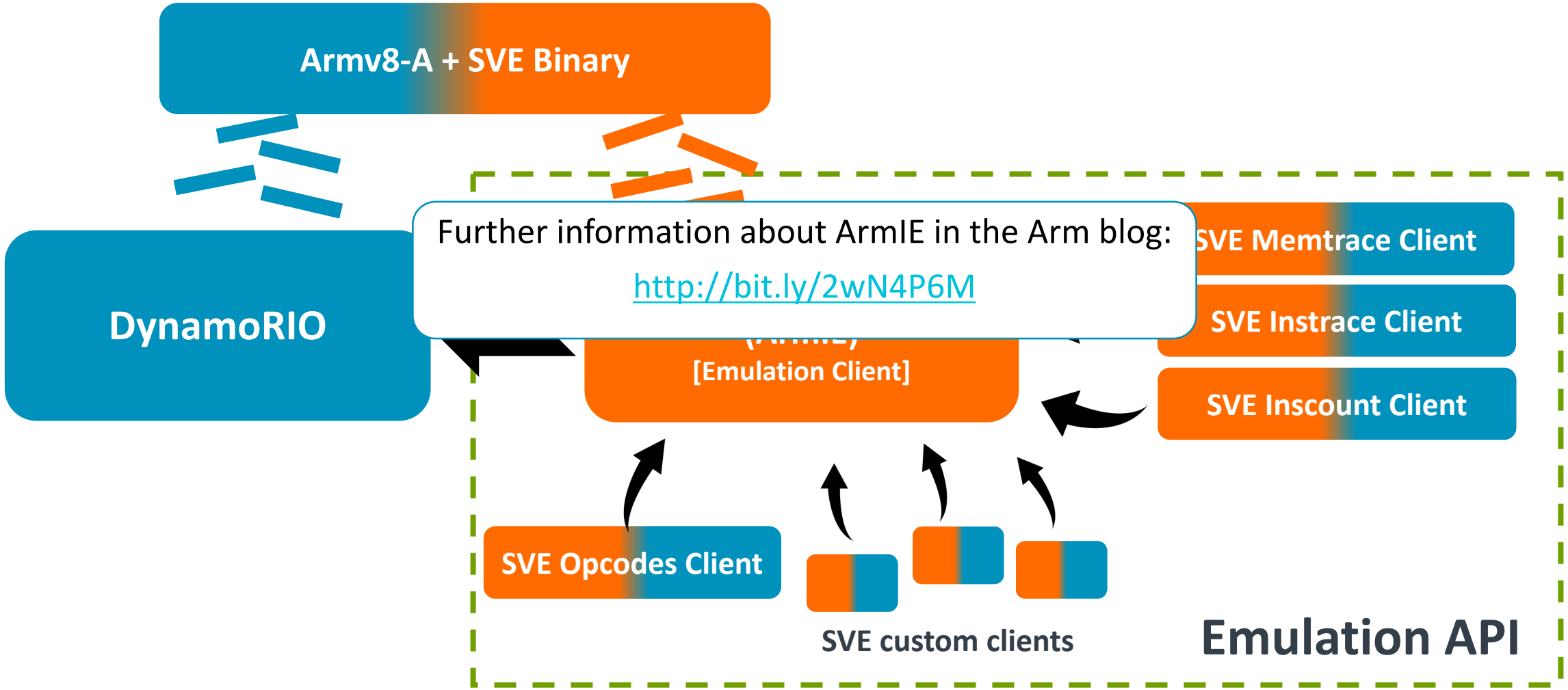
Coarser levels

(block multi-coloring)



DynamoRIO & ArmIE

Or how we learned to stop worrying and love SVE

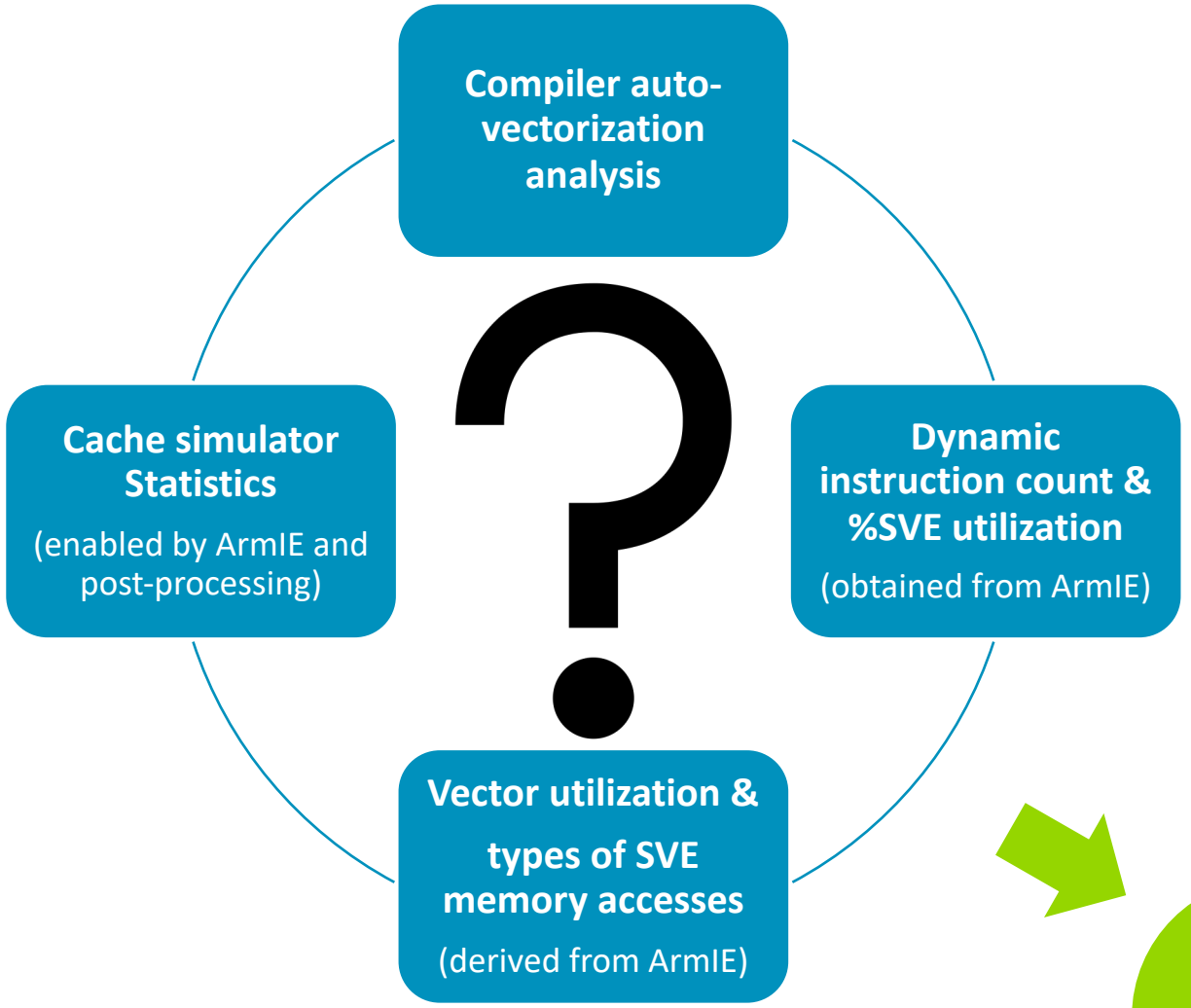


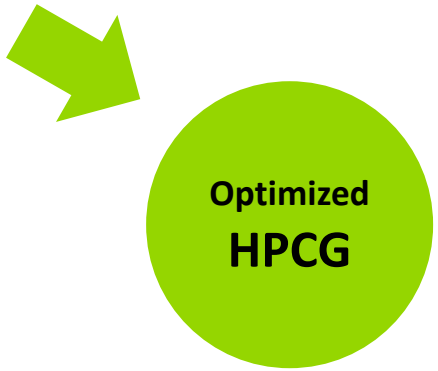
Further information about ArmIE in the Arm blog:

http://bit.ly/2wN4P6M

The Methodology

Overview





How?

- Using time-agnostic metrics

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

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)

What we ran

A trilogy of HPCG flavors

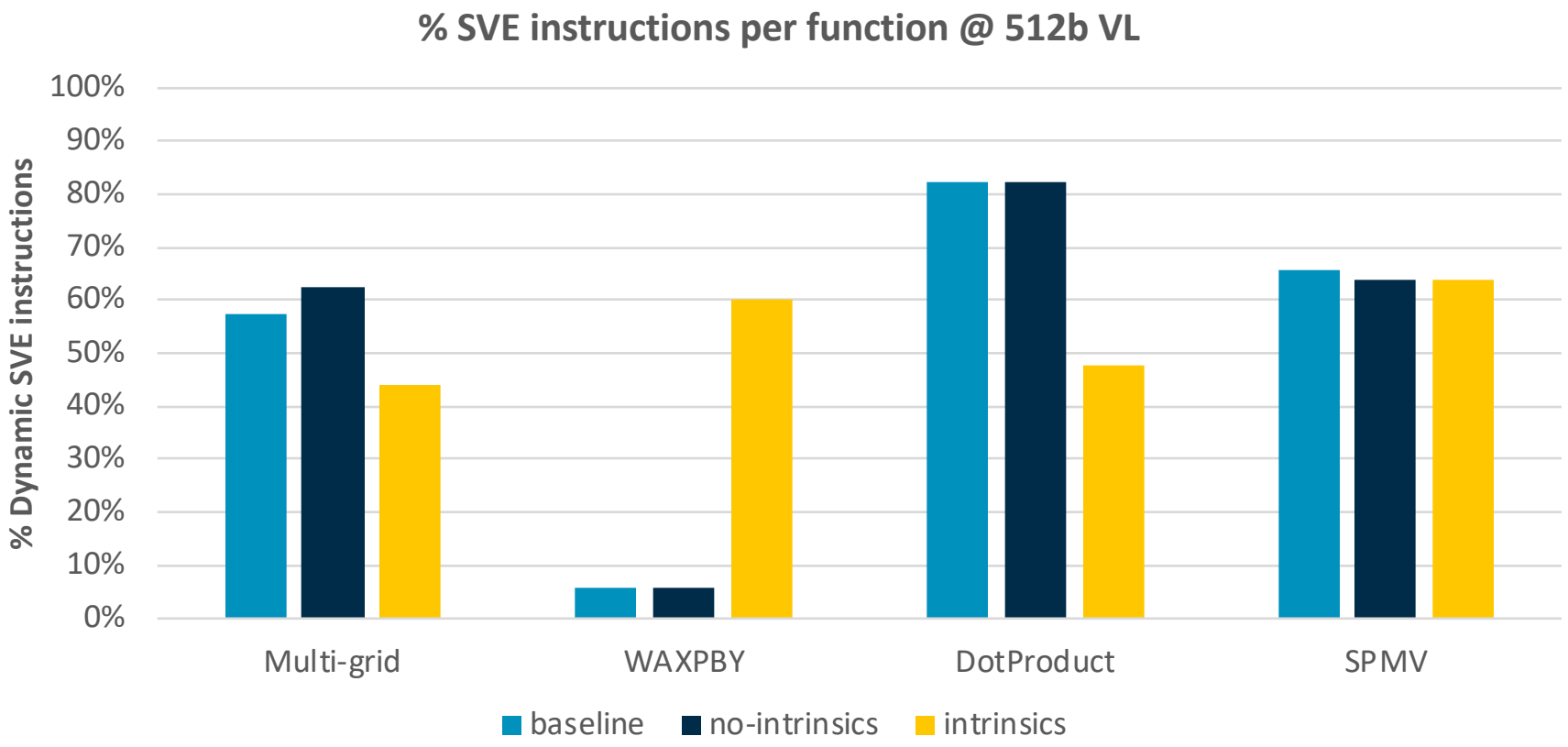
3 HPCG versions:

- **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.

Cue the ArmIE

Part 1: Running the applications through our instrumentation clients

- Instruction count client

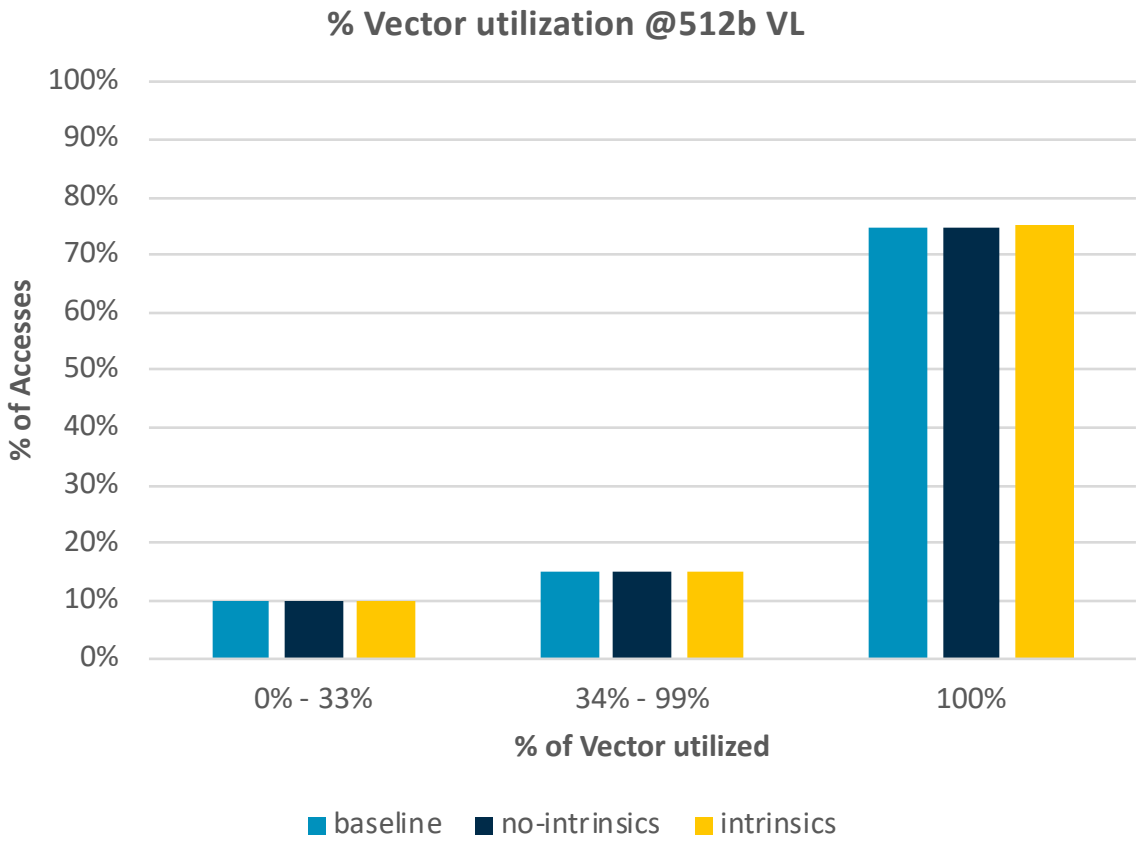


Squeeze the analysis juice

Part 2: 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%

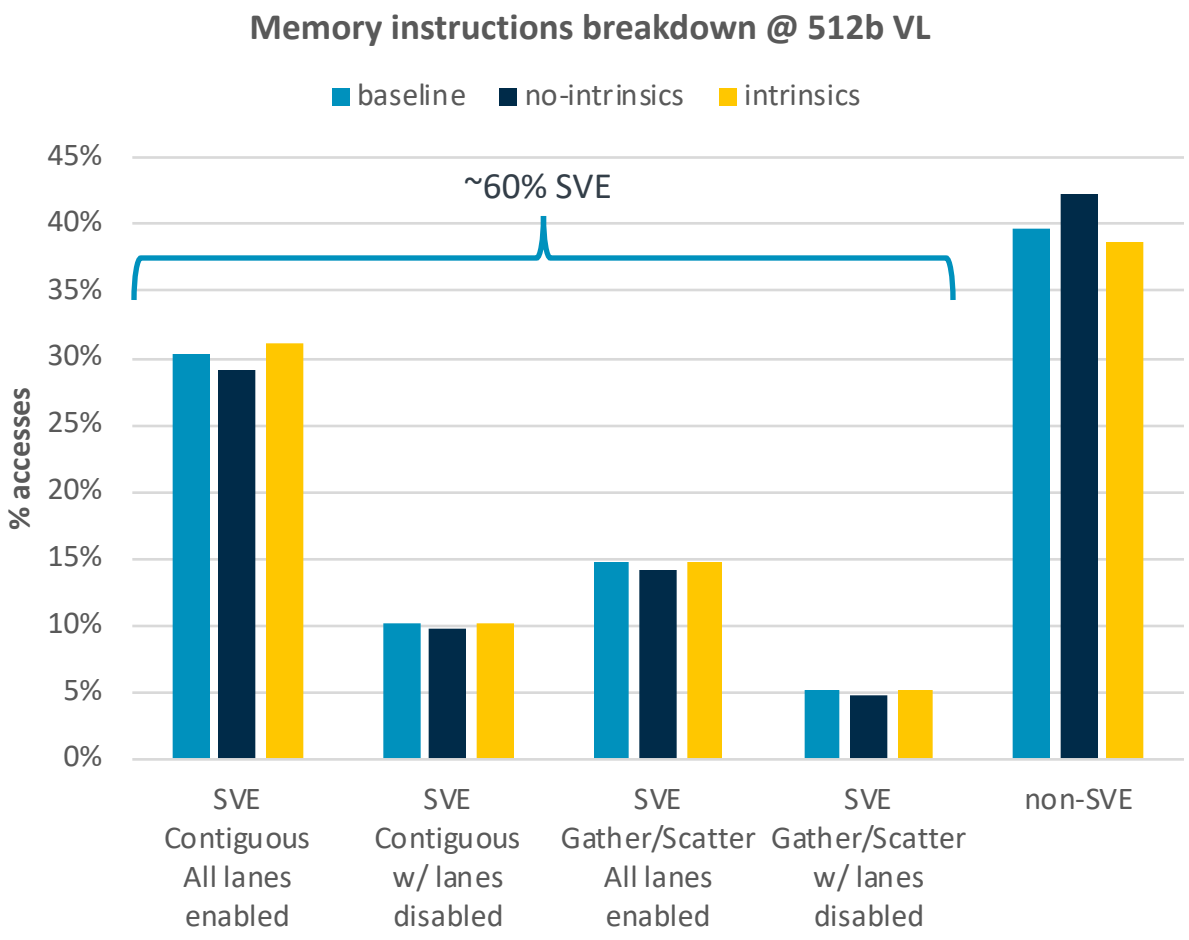


Squeeze the analysis juice

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

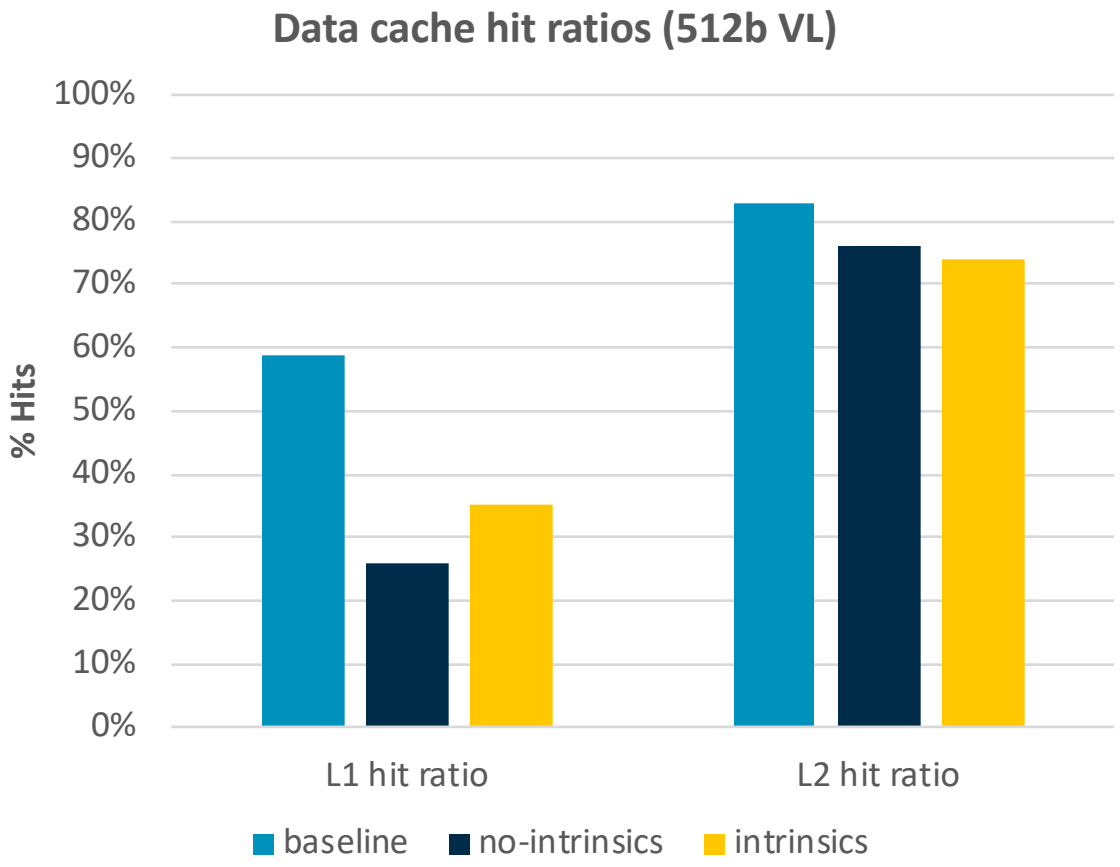
- Memory accesses present similar characteristics
  - Slightly more SVE accesses in the intrinsics version.

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



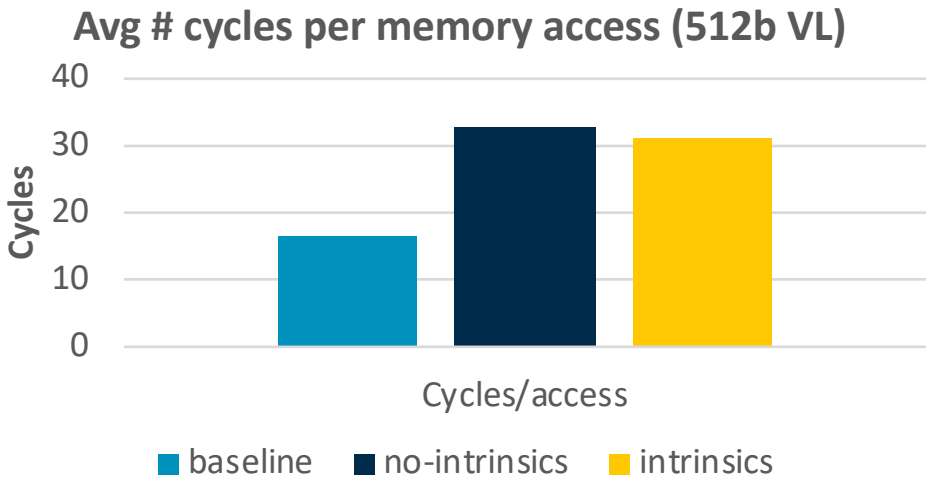
Squeeze the analysis juice

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



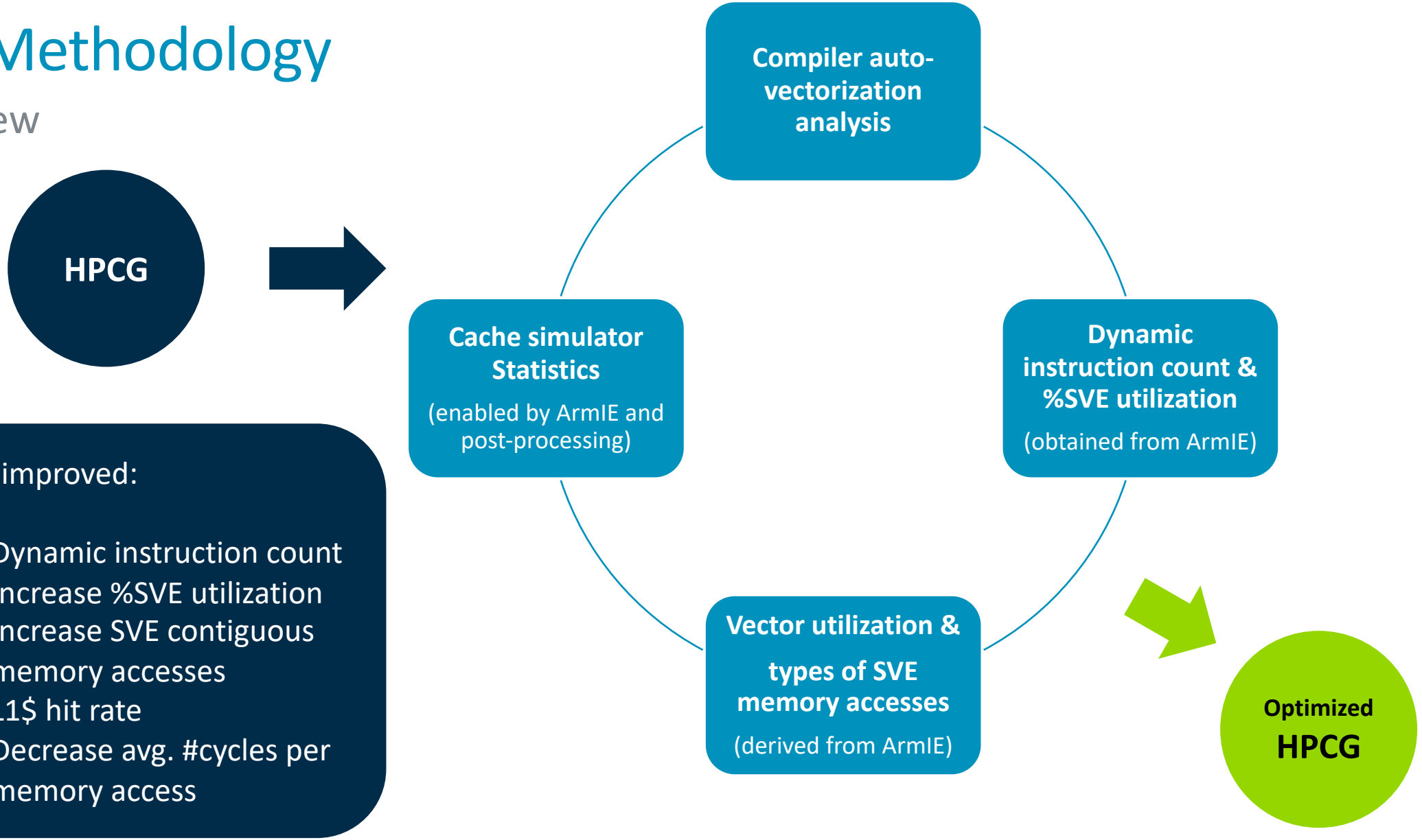
Cache simulator parameters

	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			



The Methodology

Overview



We improved:

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

The end of HPCG's SVE story

But the Methodology is here to stay

In our successful journey to optimize HPCG we:

- Produced the Methodology on optimizing applications for SVE in 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).

Methodology publicly available at the Arm Community blog:

http://bit.ly/2KZzl5y

arm

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