https://uob-hpc.github.io

Enabling next-generation processor simulations with SimEng

Simon McIntosh-Smith

University of Bristol, UK

HPC research group

https://uob-hpc.github.io/SimEng

Prof. Simon McIntosh-Smith

Mr Jack Jones

Mr Finn Wilkinson

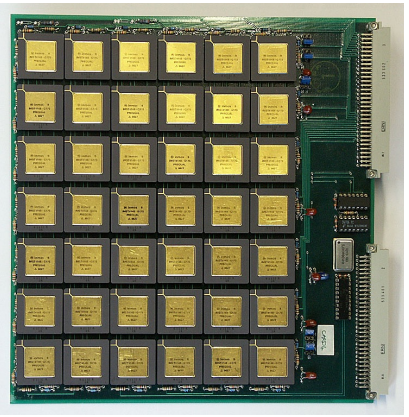
Mr Rahat Muneeb

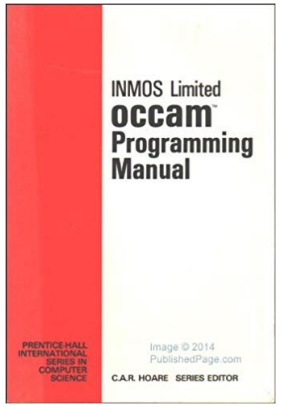
Mr Dan Weaver

University of BRISTOL

ASIMOV

Some history





- Started my career at Inmos in Bristol in 1994
  - Transputers, Occam, …
- Worked as an architect on “Chameleon” designing a SIMD instruction set for a dual-core, 64-bit, dual-issue, out-of-order CPU
- Very advanced workflow for the time
  - A single ‘master’ instruction set database drove everything
    - Documentation
    - Simulator
    - Compiler / assembler
    - Test / verification / …

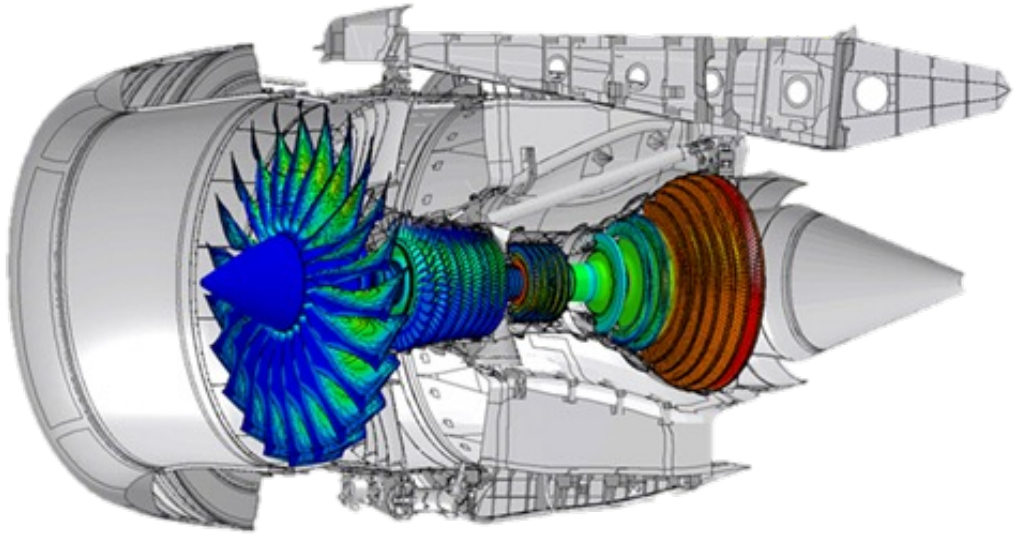
Early design space exploration

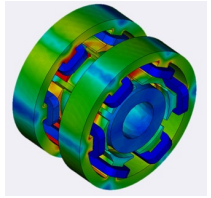
- The electronic spec-led workflow enabled rapid CPU design space exploration
- We could change most parameters about the architecture and microarchitecture, and regenerate everything quickly to try rigorous experiments
  - From the ISA to the number and spec of execution units etc.
  - Size and structure of reservation stations, memory hierarchy, …
- I rejoined academia in 2009 and wanted to try these kinds of experiments – this wasn't as straightforward as I expected…

Motivation – designing gas turbines ‘in silico’

ASiMoV 5-year project with Rolls-Royce

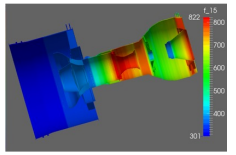
- Aiming to design new gas turbines completely in simulation
- Many different kinds of physics need to be modelled simultaneously



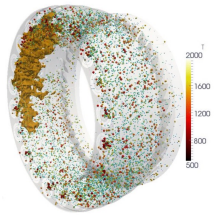


Electromagnetic

Thermo-mechanical



Contact and Friction



Computational Fluid Dynamics

Combustion

University of BRISTOL

https://sc22.supercomputing.org/presentation/?id=svs104&sess=sess277

ASIMOV

So what do we want to be able to do for ASiMoV?

Explore the design of an “optimal” processor for 5–10 years' time

- Core level
  - OoO parameters, number and width of vector units, prefetch capability…
- Co-processor level
  - Accelerators for vector–matrix math, FFTs, …
- Memory hierarchy level
- Network level?

To address these questions…

… we need a fast, easy to modify, accurate-enough simulator to support semi-automated design space exploration.

In theory, we could do this with gem5 or a number of other simulators

But we found they didn't have the specific combination of speed and accuracy to let us do the things we needed.

The "Simulation Engine" was born to investigate these issues…

SimEng design goals

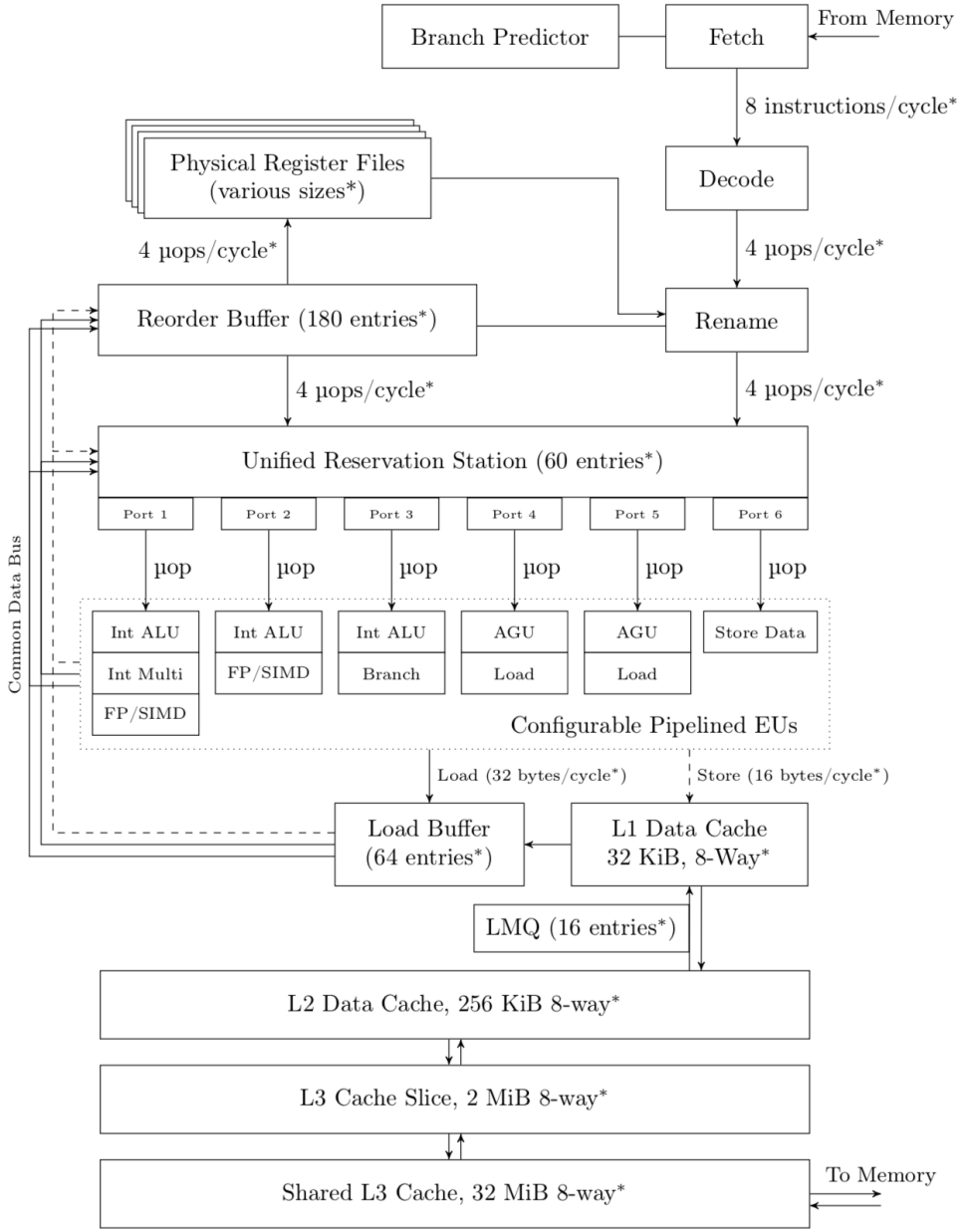
Primary goals:

- **Fast** – millions of OoO instructions per second on a single core
- **Accurate** – within 10% of real hardware
- **Easy to modify** – just hours or days to create a radically different processor model

Secondary goals:

- Use existing frameworks where possible
  - CAPSTONE for instruction decode, SST for memory hierarchy / multicore
  - Gem5-compatible tracing, checkpointing, …

SimEng initially targeted ThunderX2



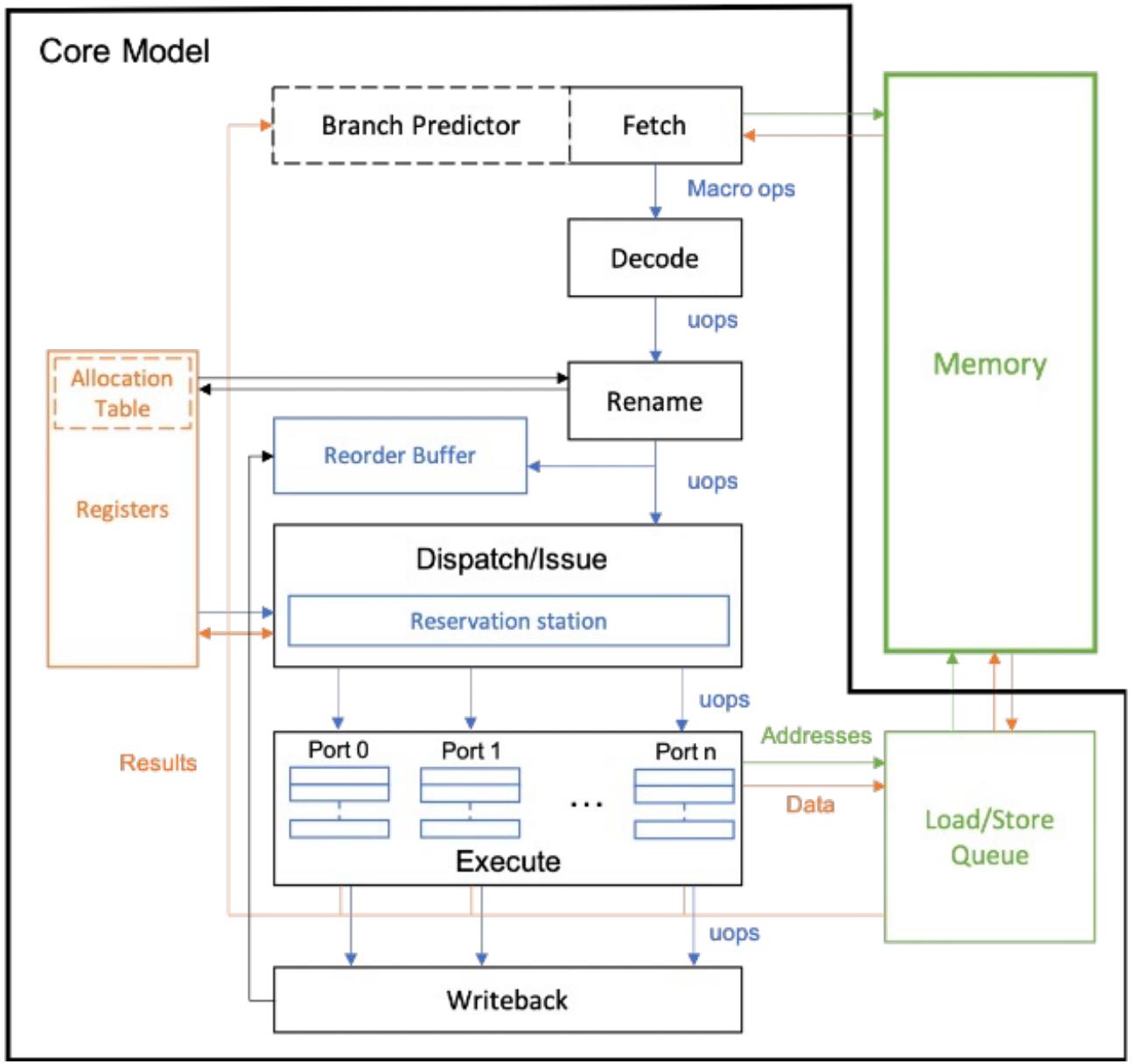
The ThunderX2 simulation was within 5-10% of the real hardware

University of BRISTOL

http://uob-hpc.github.io

ASIMOV

SimEng's Generic OoO Processor Model



http://uob-hpc.github.io

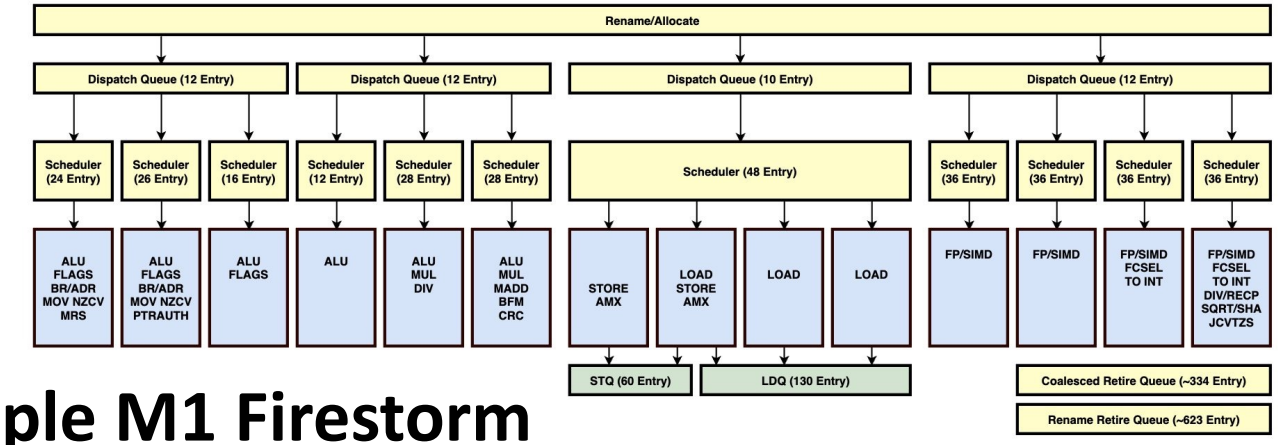
Configuration YAML Files

The generic SimEng pipeline can be parameterised to reflect existing microarchitectures, such as Marvell's ThunderX2 or Fujitsu's A64fx, or to model hypothetical core designs

Excerpts from SimEng's `a64fx.yaml`:

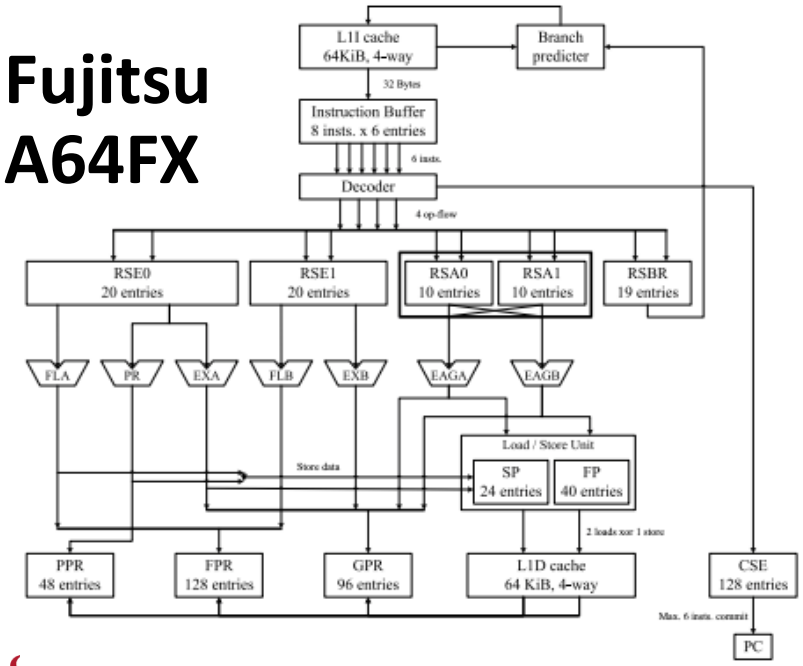
```yaml
Core:
  Simulation-Mode: outoforder
  # Clock Frequency is in GHz.
  Clock-Frequency: 1.8
  # Timer-Frequency is in MHz.
  Timer-Frequency: 100
  Fetch-Block-Size: 32
  Micro-Operations: True
  Vector-Length: 512
```

```yaml
LSQ-L1-Interface:
  Access-Latency: 5
  Exclusive: True
  Load-Bandwidth: 128
  Store-Bandwidth: 64
  Permitted-Requests-Per-Cycle: 2
  Permitted-Loads-Per-Cycle: 2
  Permitted-Stores-Per-Cycle: 1
```

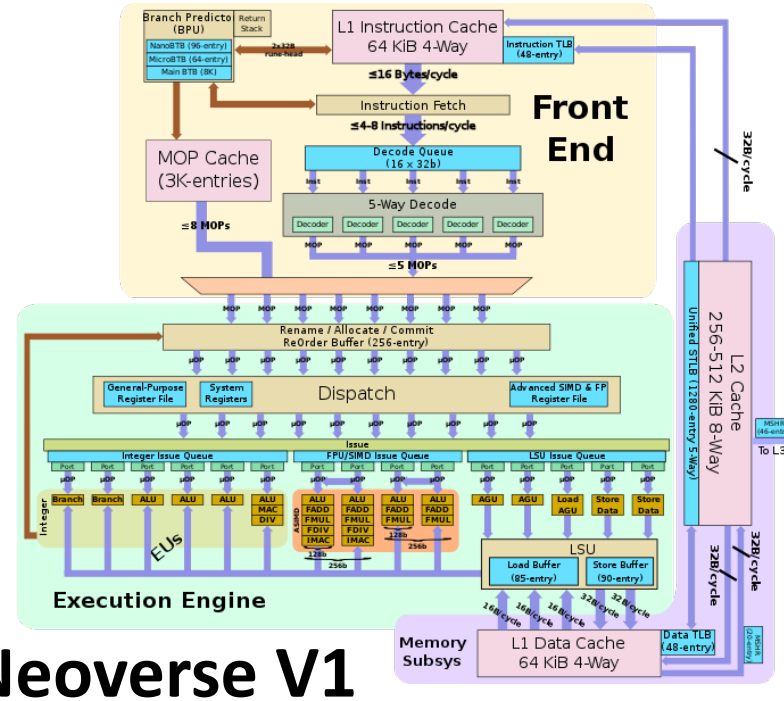


Apple M1 Firestorm

Fujitsu A64FX



Latest Models



Arm Neoverse V1

University of BRISTOL

https://uob-hpc.github.io

ASIMOV

Current Status and Work In Progress

- Primarily targeting Armv9.2-a+SVE+SME+SVE2.
  - SimEng now supports ~950 instructions, ~15% of the ISA supported by Capstone-Engine
  - Broad SVE support to match Fujitsu A64FX, Graviton 3 and other Arm cores
    - Can vary SVE widths and number of units
  - Initial RISC-V support with I (base isa), A (atomic), and M (multiply/divide) extensions
  - Initial support for load/store macro-op splitting
- Support for syscall emulation:
  - Enough to handle libc startup routines in real binaries (~50)
- Integration with the Structural Simulation Toolkit (SST) allows us to model a variety of data memory hierarchies
  - In-house base implementation is an infinite L1 cache. Looking to explore faster, simpler, alternatives to SST
  - http://sst-simulator.org

To date, a variety of workloads have been run through SimEng including C/C++/FORTRAN codes.

We've compiled codes with GNU 7-10 and ArmClang 20/22 (LLVM 9 / 13).

Current limitations:

- Must be statically compiled
  - Exploring how best to support dynamically linked binaries
- OpenMP codes supported running on a single thread
- Untested single rank MPI
- Environment variables must be passed into the simulated memory space manually; for example, OMP_NUM_THREADS=1
- Objects residing in memory but not sourced from the binary, such as the vDSO (virtual dynamic shared object), not yet officially supported

Example SimEng Tested Workloads

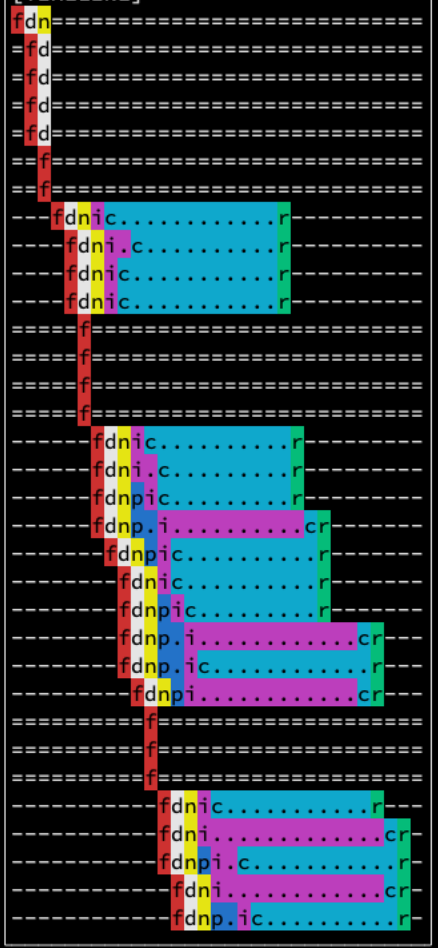
- miniBUDE - mini-app for Bristol University Docking Engine (BUDE) Chemistry code
- STREAM
- Cloverleaf (SPEChpc 2021)
- Tealeaf (SPEChpc 2021)
- FFTW

http://uob-hpc.github.io

Trace Output

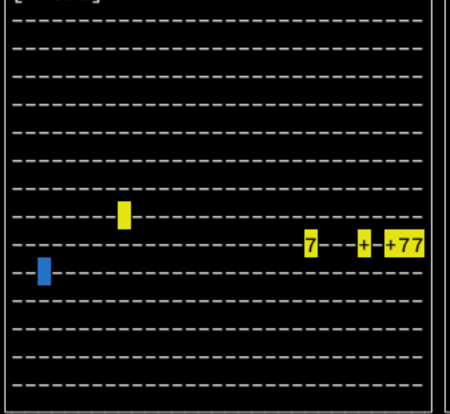
- f – fetch
- d – decode
- n – rename
- p – dispatch
- i – issue
- c – complete
- r – retire
- = - flushed

[TIMELINE]



[INSN_NUM]	[CYCLE]	[PC]	[μ_NUM]	[DISASM]
4261478	1301319	0x60159C	0	movprfx z16, z5
4261479	1301320	0x6015A0	0	fcadd z16.d, p0/m, z16.d, z6.d, #0x10e
4261480	1301320	0x6015A4	0	mov z16.q, q16
4261481	1301320	0x6015A8	0	st1d {z16.d}, p0, [x21, z0.d, lsl #3]
4261482	1301320	0x6015AC	0	fcadd z5.d, p0/m, z5.d, z6.d, #0x5a
4261483	1301321	0x6015B0	0	mov z5.q, q5
4261484	1301321	0x6015B4	0	st1d {z5.d}, p0, [x20, z0.d, lsl #3]
4261485	1301322	0x6016FC	0	lsl x1, x4, #6
4261486	1301323	0x601700	0	neg x3, x4, lsl #5
4261487	1301323	0x601704	0	add x10, x0, x1
4261488	1301323	0x601708	0	lsl x5, x4, #4
4261489	1301324	0x768EB0	0	cmp x19, x2
4261490	1301324	0x768EB4	0	mov x1, x22
4261491	1301324	0x768EB8	0	csel x2, x19, x2, ls
4261492	1301324	0x768EBC	0	add x22, x22, x2
4261493	1301325	0x60170C	0	add x11, x10, x3
4261494	1301325	0x601710	0	neg x6, x4, lsl #3
4261495	1301325	0x601714	0	add x8, x11, x1
4261496	1301325	0x601718	0	ld1d {z6.d}, p1/z, [x8]
4261497	1301326	0x60171C	0	add x7, x8, x5
4261498	1301327	0x601720	0	sub x6, x6, x4
4261499	1301327	0x601724	0	sub x8, x7, x1
4261500	1301327	0x601728	0	ld1d {z16.d}, p1/z, [x7]
4261501	1301327	0x60172C	0	add x7, x8, x3
4261502	1301328	0x601730	0	ld1d {z5.d}, p1/z, [x8]
4261503	1301329	0x400280	0	adrp x16, #0x496000
4261504	1301329	0x400284	0	ldr x17, [x16, #8]
4261505	1301329	0x400288	0	add x16, x16, #8
4261506	1301330	0x601734	0	add x8, x7, x1
4261507	1301330	0x601738	0	ld1d {z3.d}, p1/z, [x7]
4261508	1301330	0x60173C	0	add x7, x8, x6, lsl #3
4261509	1301331	0x601740	0	ld1d {z17.d}, p1/z, [x8, x6, lsl #3]
4261510	1301331	0x601744	0	add x6, x7, x1

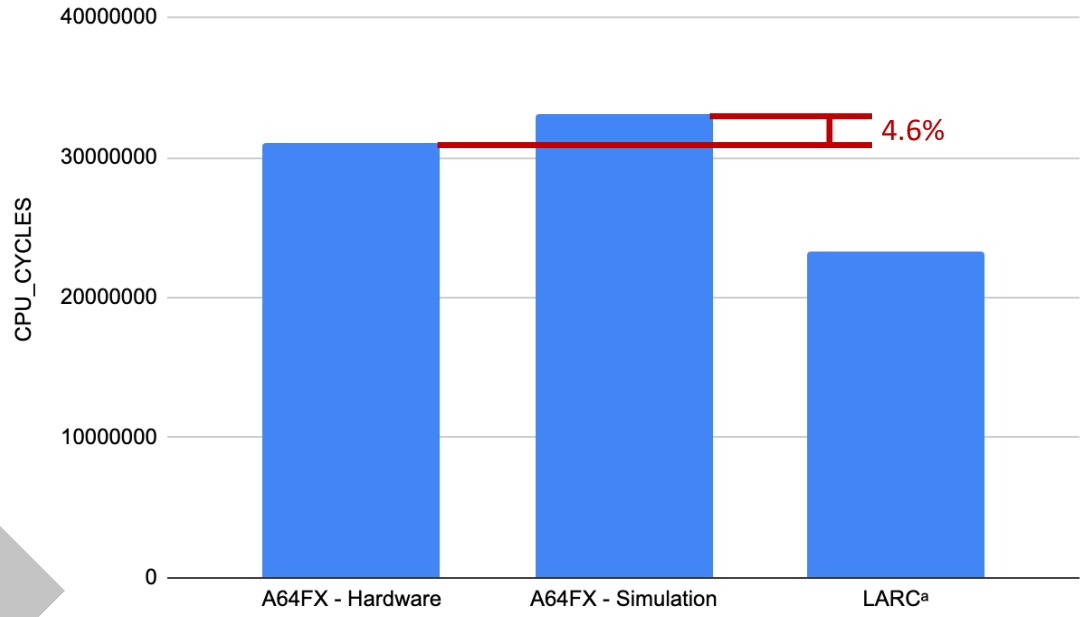
[PROBE]



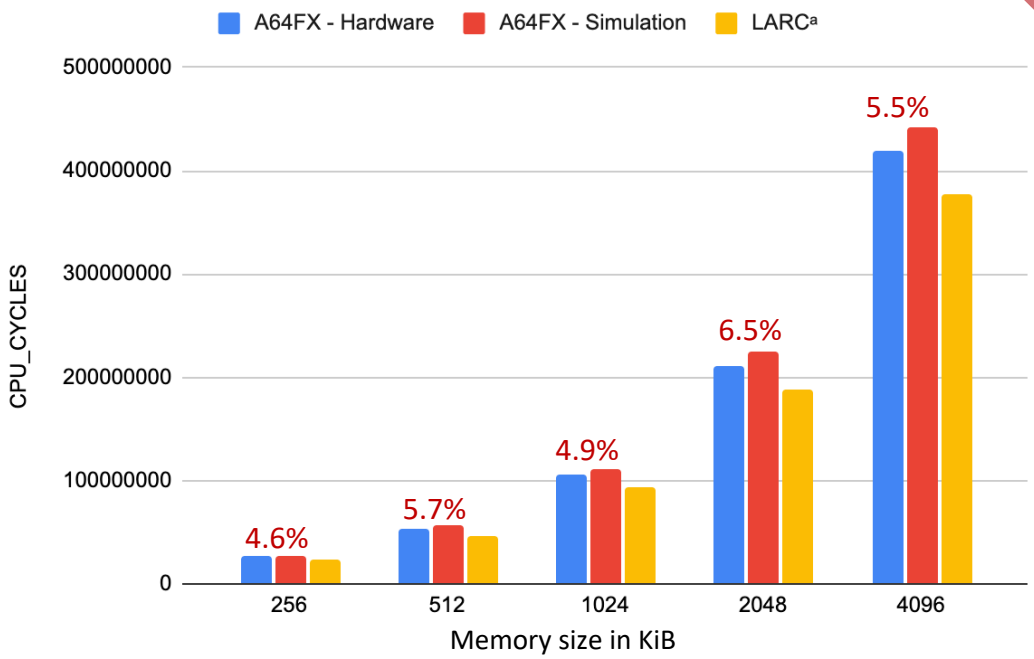
[PROBES SELECTED]	[ASSOCIATED_INSTRUCTIONS]
Stalled.fetch.instructionFetch	-
Stalled.fetch.instructionDecode	-
Stalled.rename.robFull	-
Stalled.rename.lsqFull	-
Stalled.rename.allocation	-
Stalled.dispatch.rsFull	-
Stalled.issue.portBusy	-
Stalled.issue.rsEmpty	-
Stalled.loadStoreQueue.notReady	4261496, 4261496, 4261496, 4261496, 4261496, 4261496, 4261496, 4261500, 4261500, 4261500, 4261500, 4261500, 4261500, 4261469
Branch.execute.misprediction	-
Flush.rob.storeViolation	-
Halt.fetch.programMemoryExceeded	-
Exception.rob.robCommit	-
Exception.fixedLatencyMemoryRead	-

Experiments with SimEng – LARC from RIKEN

STREAM triad



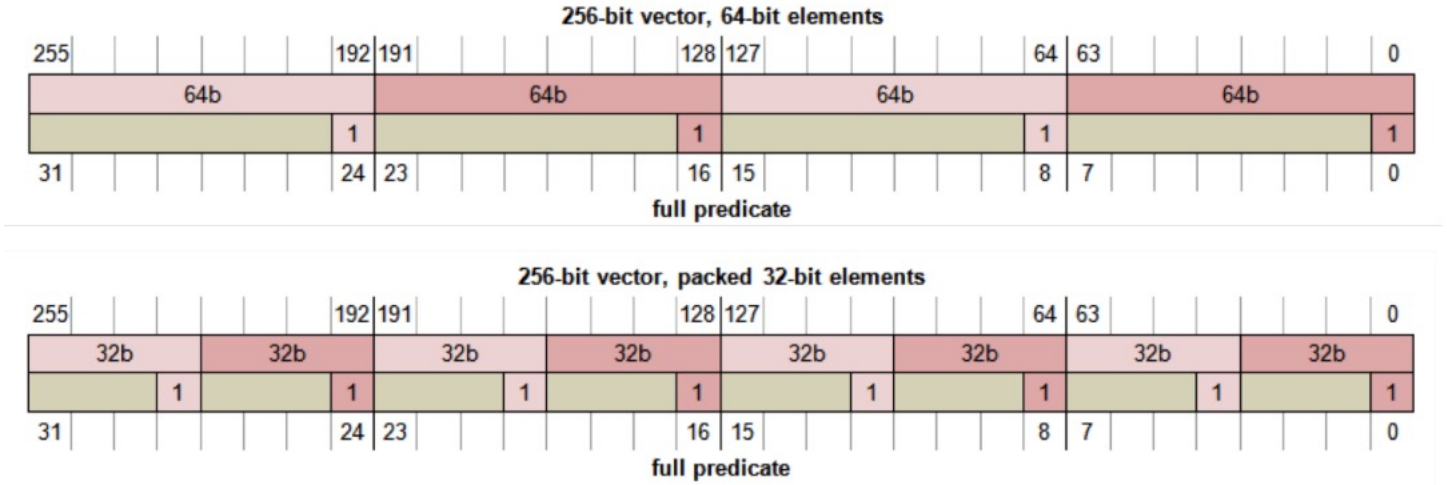
CacheBW



https://uob-hpc.github.io

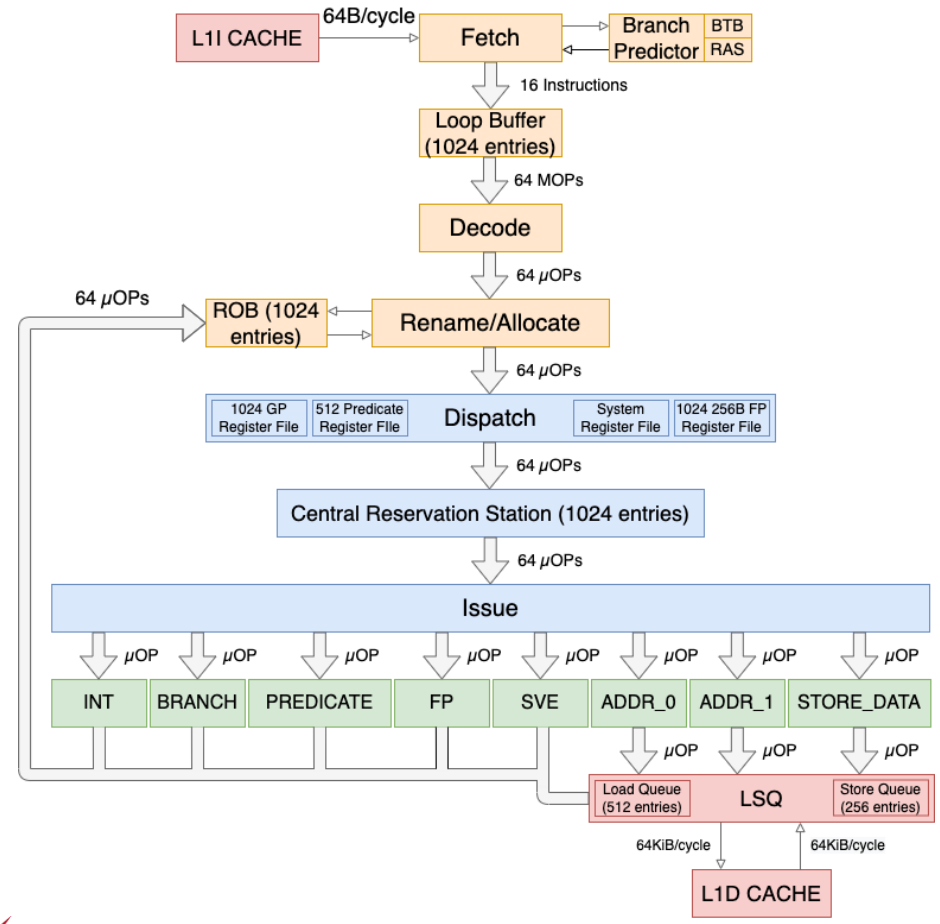
Vector unit design space exploration

- What is the optimal vector width?
- What is the optimal number of vector units?
- Focus on Arm's SVE to facilitate varying vector widths: 128-2048 bits



Source: The ARM Scalable Vector Extension paper, doi: 10.1109/MM.2017.35

Methodology



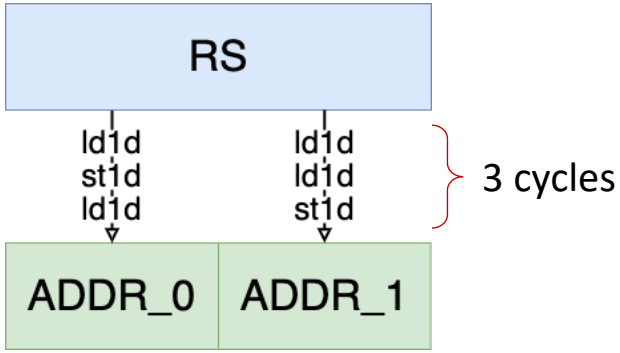
- Model a very wide frontend with large Out-of-Order resources and wide load/store to support it
- Two vector unit arrangements:
  1. Single vector unit spanning various widths (128b to 2048b)
  2. Multiple vector units totaling the same vector width (16x128b to 2x1024b)
- Arm Compiler for Linux 20.0 used to compile workloads with target `-march=armv8.4-a+sve`

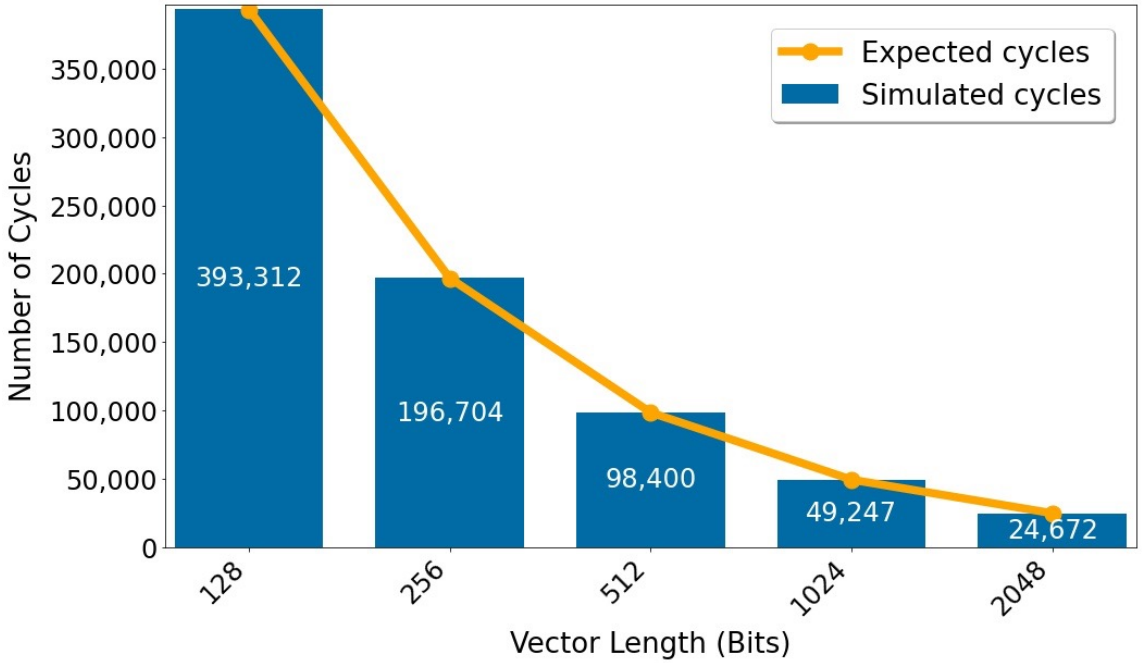
DAXPY

Problem size of 524288 elements.

$$Y := \alpha * X + Y$$

```
401e14 : ld1d {z1.d}, p0/z, [x1, x4, lsl #3]
401e18 : ld1d {z2.d}, p0/z, [x2, x4, lsl #3]
401e1c : fmla z2.d, p0/m, z0.d, z1.d
401e20 : st1d {z2.d}, p0, [x2, x4, lsl #3]
401e24 : incd x4
401e28 : whilelt p0.d, x4, x0
401e2c : b.mi 401e14
```

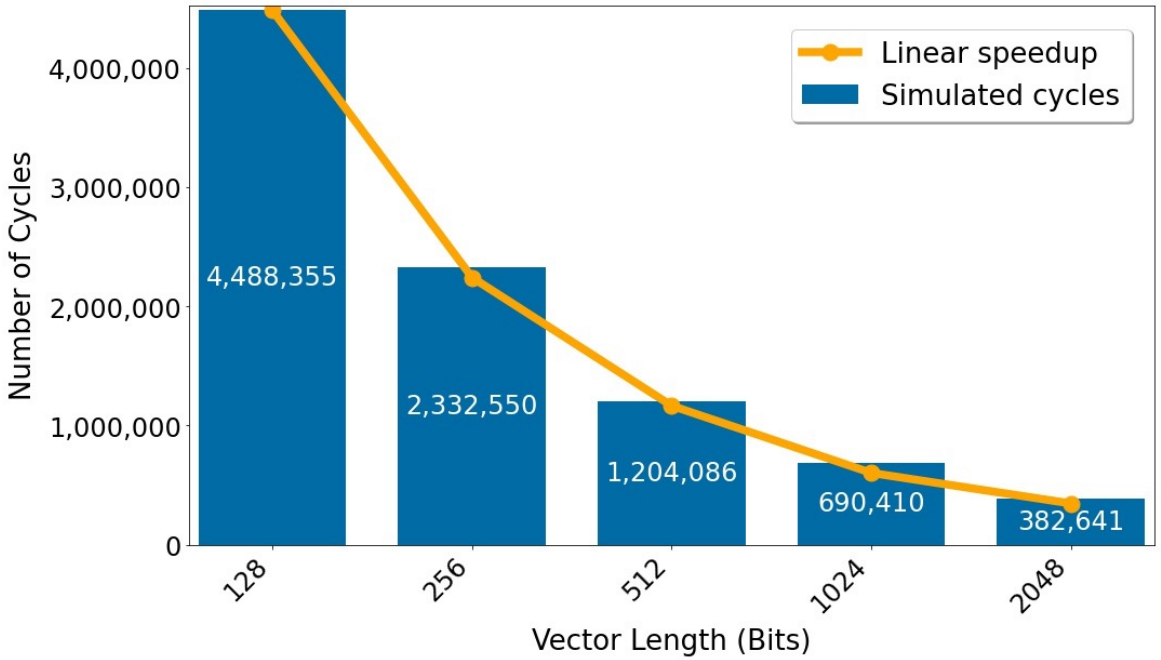


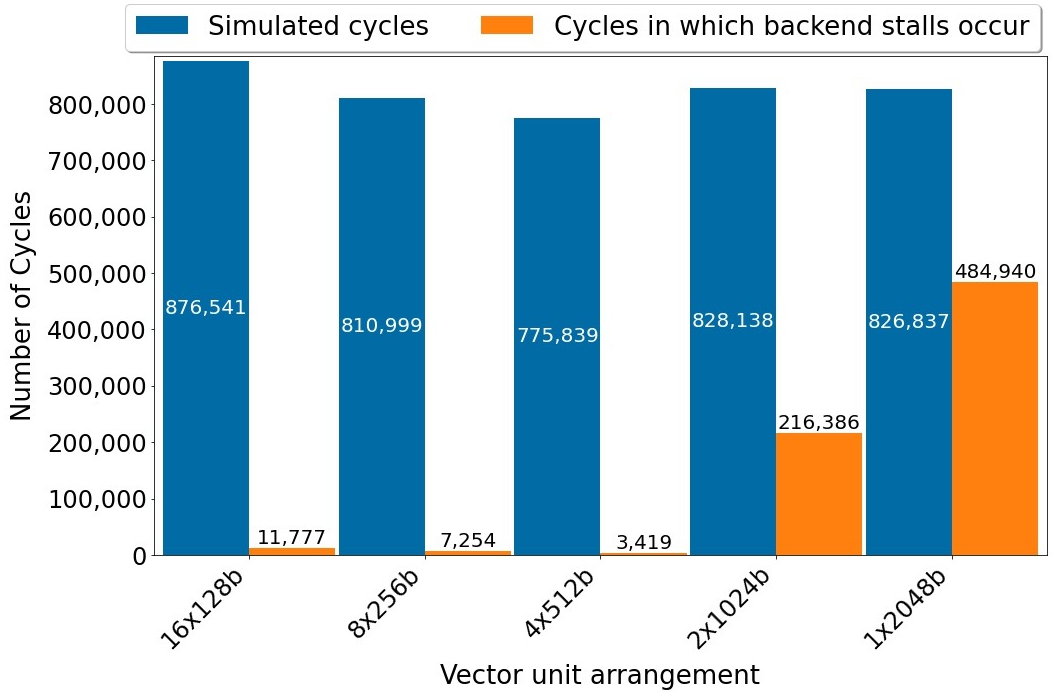


DGEMM

Problem size of M=256 N=128 K=256

$$C := \alpha * A * B + \beta * C$$





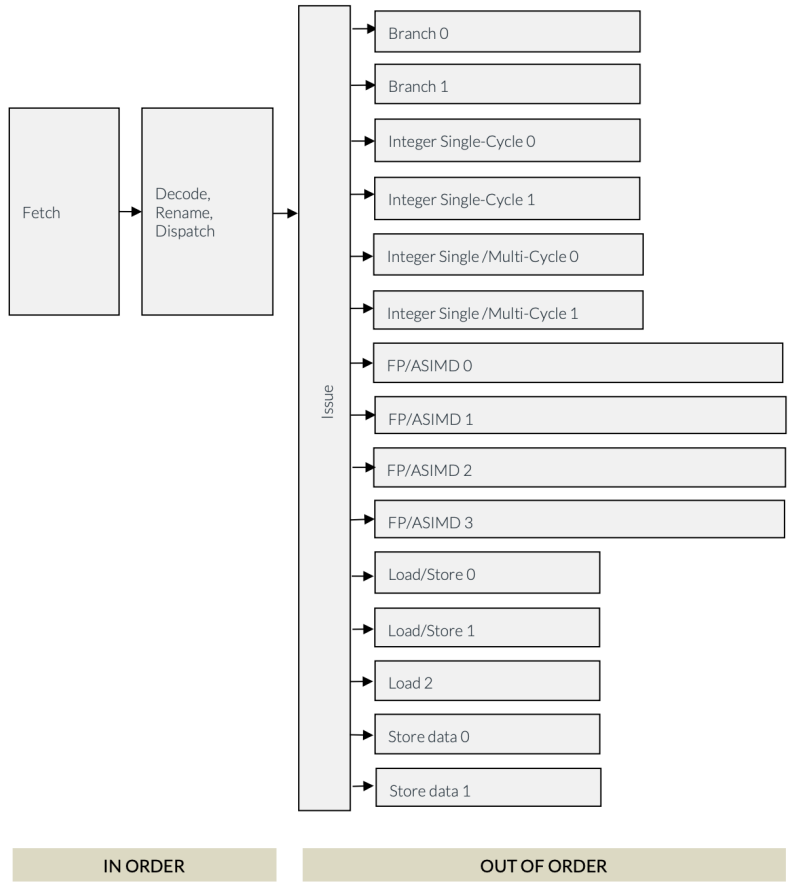
University of BRISTOL

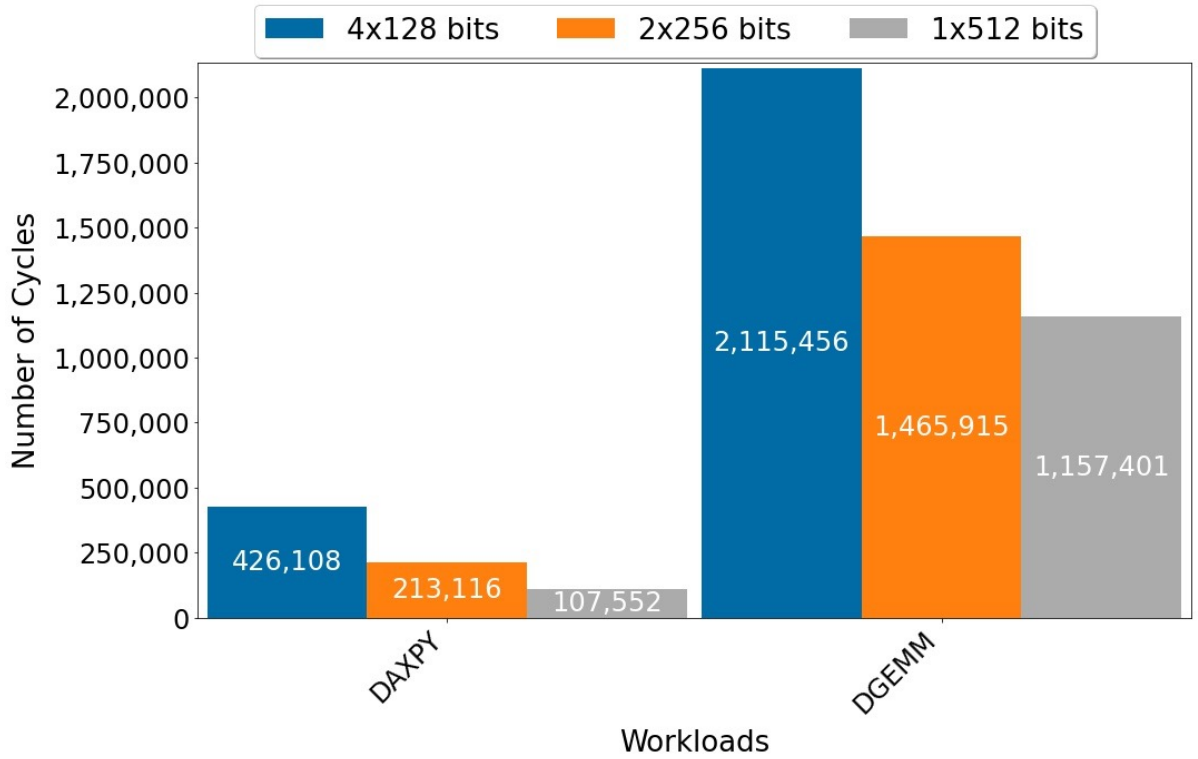
https://uob-hpc.github.io

ASIMOV

Arm Neoverse V1 model

Source: Arm Neoverse V1 Software Optimization Guide



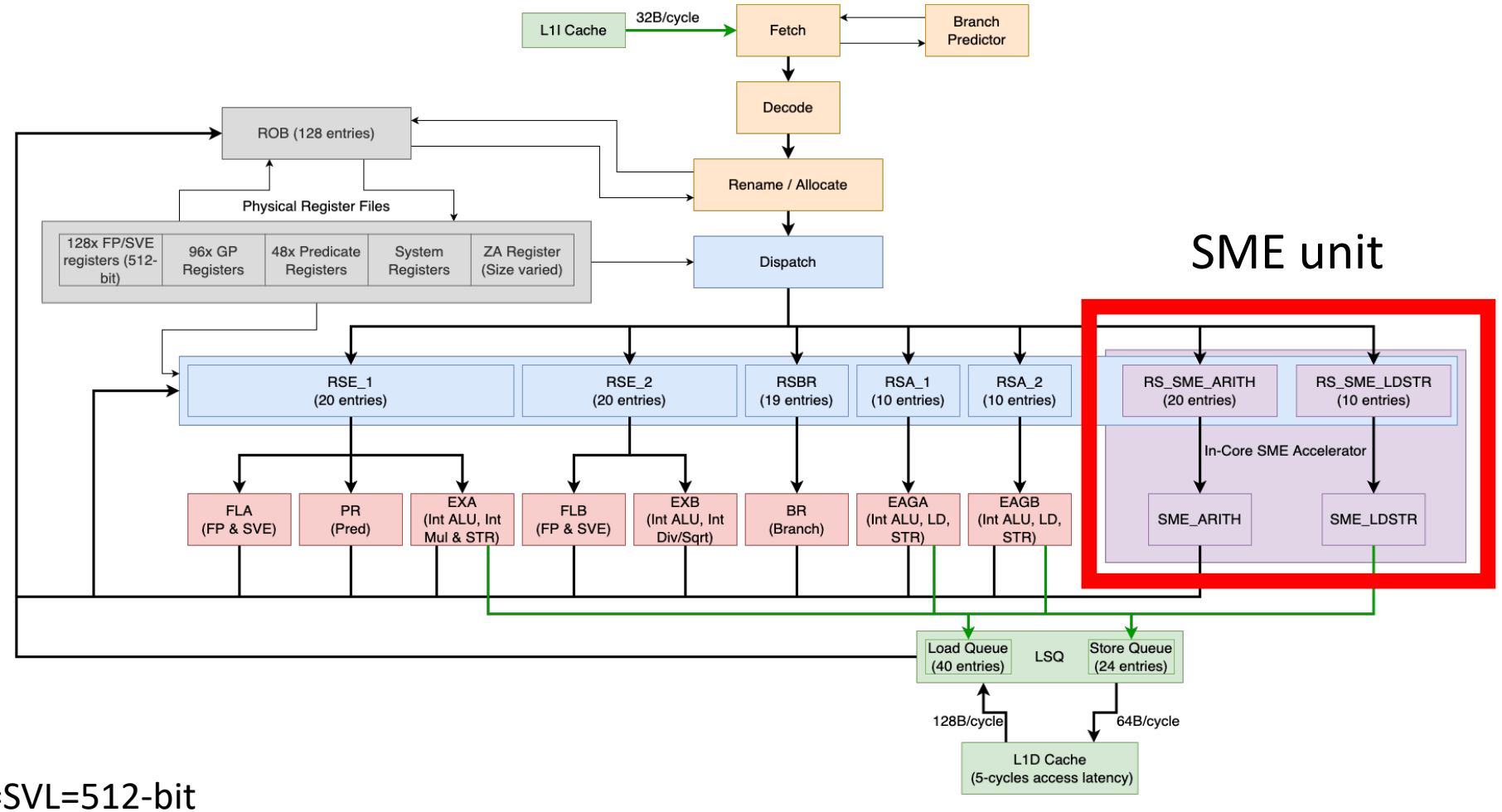


For both the DAXPY and DGEMM workloads, the 1x512b V1 variant was the most optimal.

Scalable Matrix Exctension (SME) Support

- Implemented initial support in SimEng, along with SVE2
  - SVCR sys_reg & *SVE Streaming Mode*
- Currently supports 7 instructions
  - LD1W{h,v}, ST1W{h,v}, ZERO, FMOPA, PSEL
- Started initial evaluation of SME performance vs. SVE
- Modelling in-core SME accelerator…

Hypothetical A64fx-like Core with SME



VL=SVL=512-bit

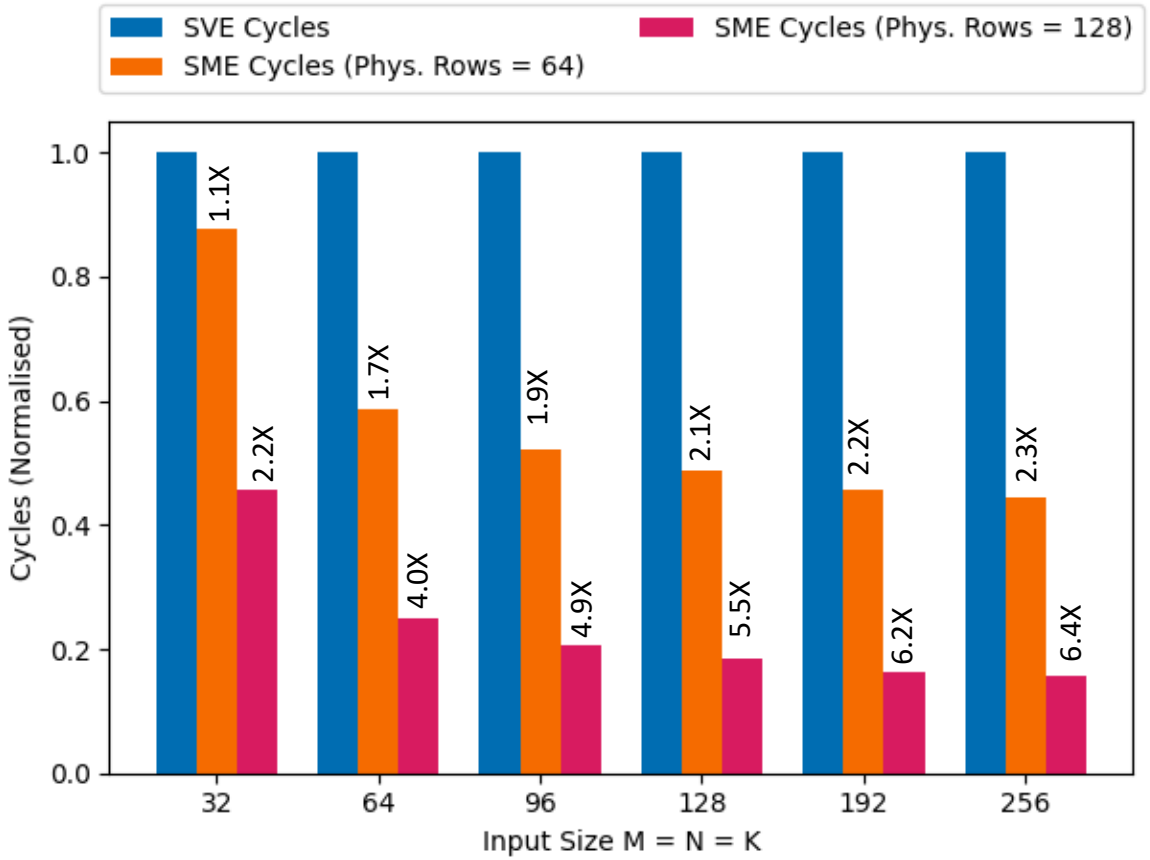
University of BRISTOL

https://uob-hpc.github.io

ASIMOV

SME MatMul Kernel Results vs. A64FX

A64FX SVE on SimEng vs. Hypothetical Core (SVL=512-bits)



SME ResNet-50 Kernel Results vs. A64FX

	Instructions	**Cycles**	**Time (Seconds)**	**GFLOPs/s**
	64x64x64 GEMM, 100 iterations			
Hardware	5,939,076	4,135,983	0.001202	43.6
Simulation	5,854,947	3,111,429	0.001010	51.9
Simulation w/ SME	1,942,164 (**3.0x**)	2,580,216 (**1.2x**)	0.000467	112.2

	1024x1024x1024 GEMM 100 iterations			
Hardware	14,095,511,071	5,888,145,390	3.00	71.5
Simulation	14,095,426,848	4,336,526,197	2.23	96.4
Simulation w/ SME	1,589,378,697 (**8.9x**)	2,407,339,782 (**1.8x**)	1.08	198.0

Planned Development

New processor models

- Continued development of Apple M1
- NVIDIA Grace / Neoverse V2

Multi-core and SMT support

Improved **memory hierarchy**

Further set of supported codes to cover more of the simulated µarch

- SPEC_CPU 2017, SPEChpc 2021, MLPerf, RIKEN Kernels, BERT, ResNet-50,…

More compiler support

- Cray, Fujitsu, Nvidia

Support for advanced branch prediction, cache prefetching

RISC-V support

- Included in next release with floating point

Improved CI testing

- Testing on multiple OS's

Improved tooling for design space exploration

https://uob-hpc.github.io

ASIMOV

Conclusions

- **SimEng** is letting us explore how fast we can make a microarchitecture level simulator
- We can now easily make major changes to a microarchitecture to enable rapid design space exploration
- SimEng achieving **>4-5X speedup over gem5** while being **more accurate** and **much, much simpler to use and modify**
- SST works well for adding memory hierarchy models, soon multicore
- We now have a fast, accurate, stand-alone, single-core model in O(10,000) lines of code – **what can you use this for**?

Acknowledgments

- Key development team in Bristol:
  - SimEng started by Hal Jones, James Price, Andrei Poenaru

- Funders:
  - EPSRC ASiMoV project (Advanced Simulation and Modelling of Virtual systems) - EP/S005072/1
  - Arm via a Centre of Excellence in HPC at University of Bristol