Prof Simon McIntosh-Smith

University of Bristol

@simonmcs

Modelling Advanced Arm-based CPUs with SimEng

SimEng developers: Jack Jones, Andrei Poenaru, Harry Waugh, Ainsley Rutterford, Hal Jones, James Price
Funding: 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

University of BRISTOL

http://uob-hpc.github.io

ASIMOV

SimEng design goals

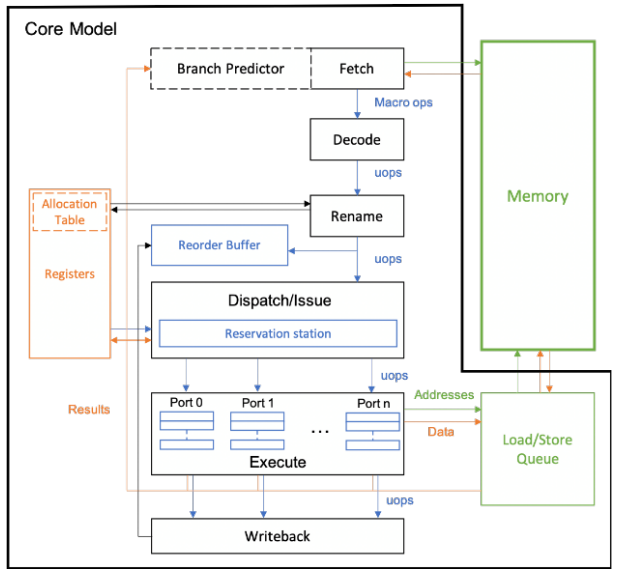
Primary goals:

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

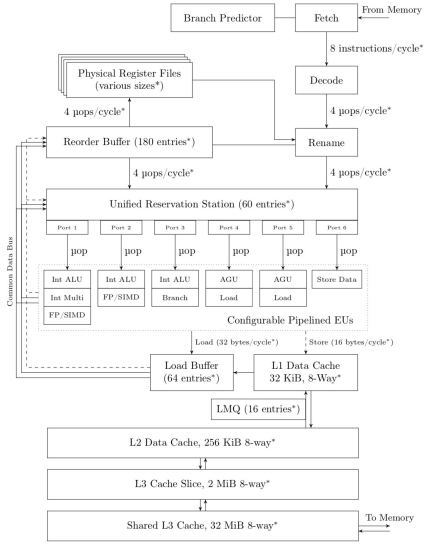
Secondary goals:

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

SimEng generic CPU model

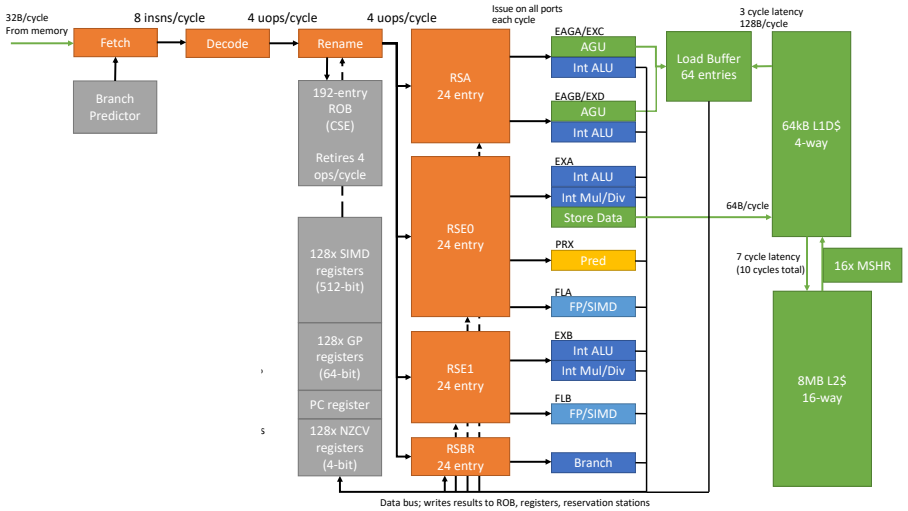


ThunderX2 model



Models of Ares, Zeus, …

A64FX model



University of BRISTOL

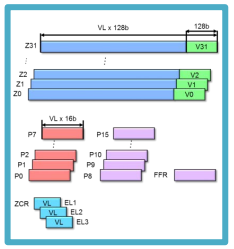
http://uob-hpc.github.io

ASIMOV

Current status and WIP

- Targeting Armv8.4+SVE. Using CAPSTONE, which also supports x86, RISC-V, POWER, …
  - SimEng now supports ~480 instructions, ~10% of the ISA
  - Includes sophisticated branch predictors (A64FX-style)
  - Partial SVE support to match A64FX
    - Can vary SVE widths and number of units
  - Single-core only (for now)
- Support for syscall emulation:
  - Enough to handle libc startup routines in real binaries (compiled from C)
  - Basic `printf` support
  - File I/O works
  - `malloc` works for most cases, but not yet complete
- Integrating with SST:
  - SimEng models up to the load/store units, will use SST's models for the memory hierarchy (SimEng includes its own infinite L1 cache model)
    - Prototype demonstrated in the summer
  - Will also use SST to enable multi-core simulations

Capstone
The Ultimate Disassembler

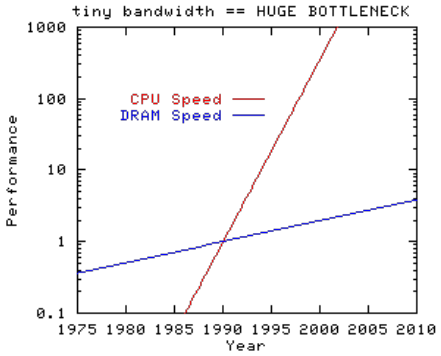


University of BRISTOL

http://uob-hpc.github.io

ASIMOV

Results



- Running McCalpin's STREAM benchmark
  - Run a problem small enough to fit in L1D cache
  - Using an out-of-order/superscalar core model, parameterized for ThunderX2 and A64FX
  - The STREAM run takes ~10ms on a real ThunderX2 core and ~13ms on a real A64FX core
- **SimEng** running on an Intel Xeon Processor E5-2603 v4 @ 1.7 GHz
  - ThunderX2 model
    - OoO takes ~94 seconds → **221 kHz / 0.50 MIPS**
    - Cycle count error is **5.3%** versus real ThunderX2 hardware
  - A64FX model
    - OoO takes ~96 seconds → **186 kHz / 0.39 MIPS**
    - Cycle count error is **7.4%** versus real A64FX hardware
- **gem5** built from Arm's sve/beta1 branch, on same Intel CPU, ThunderX2 model only
  - OoO takes ~280 seconds → **63 kHz / 0.14 MIPS** (**SimEng 3.5X / 3.6X**)
  - Cycle count error is **12.4%** versus real ThunderX2 hardware

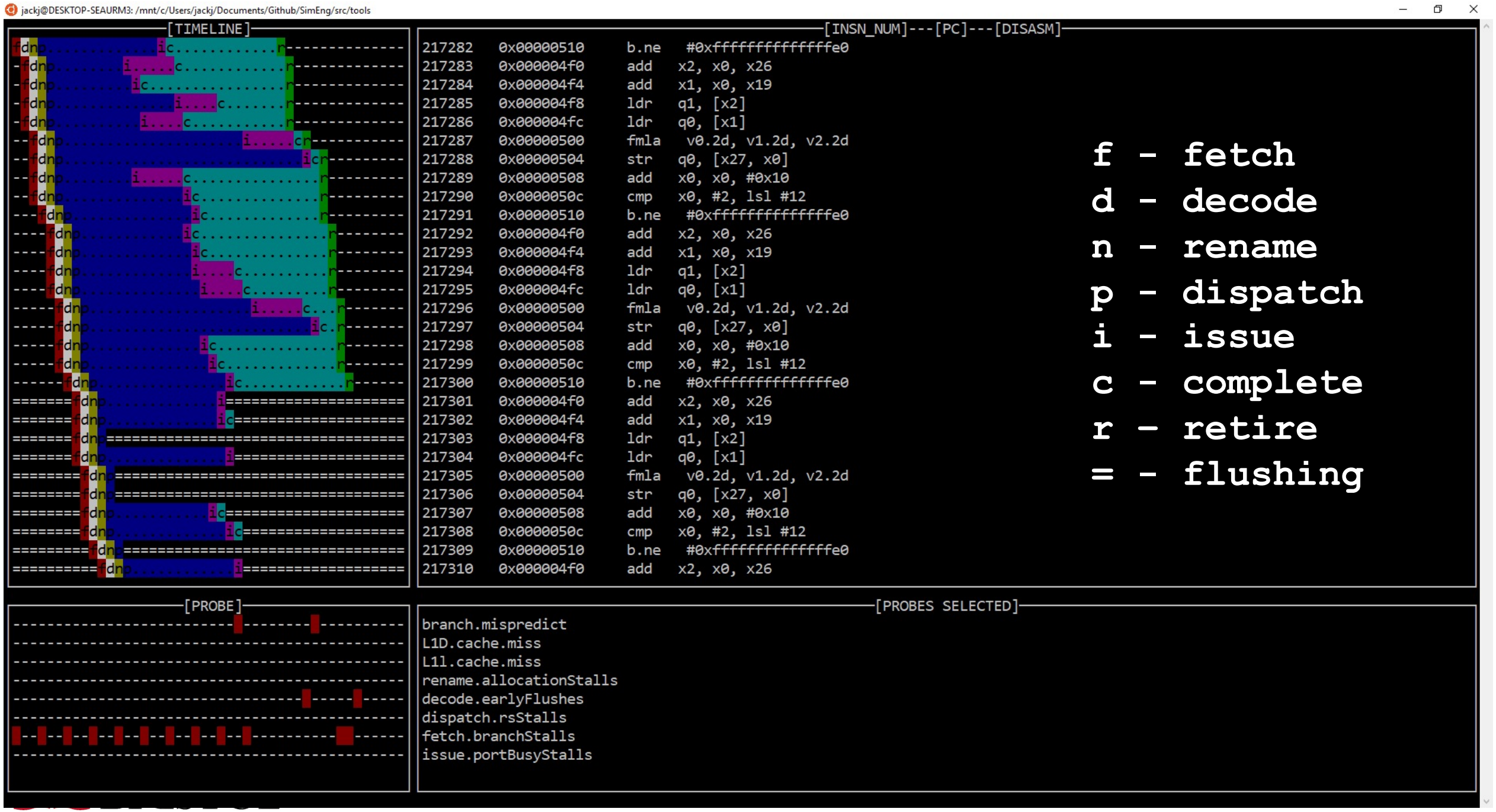
Key statistics about the project

- ~20,000 lines of simple, modern C++17
  - ~7,500 lines are specific for Armv8+SVE support
  - An additional ~10,000 lines of test code across ~350 tests
  - Can build with GCC 7 (or later), Clang (7 or 5), or Armclang 20. Intel 19 soon too.

- Includes a full Continuous Integration (CI) workflow
  - CircleCI→Jenkins, Googletest

- Supported host platforms include: Ubuntu, CentOS and macOS

- Will be released under a permissive LLVM-style Apache 2.0 license


jackj@DESKTOP-SEAURM3: /mnt/c/Users/jackj/Documents/Github/SimEng/src/tools
[TIMELINE]
[INSN_NUM]---[PC]---[DISASM]
217282 0x00000510 b.ne #0xfffffffffffffe0
217283 0x000004f0 add x2, x0, x26
217284 0x000004f4 add x1, x0, x19
217285 0x000004f8 ldr q1, [x2]
217286 0x000004fc ldr q0, [x1]
217287 0x00000500 fmla v0.2d, v1.2d, v2.2d
217288 0x00000504 str q0, [x27, x0]
217289 0x00000508 add x0, x0, #0x10
217290 0x0000050c cmp x0, #2, lsl #12
217291 0x00000510 b.ne #0xfffffffffffffe0
217292 0x000004f0 add x2, x0, x26
217293 0x000004f4 add x1, x0, x19
217294 0x000004f8 ldr q1, [x2]
217295 0x000004fc ldr q0, [x1]
217296 0x00000500 fmla v0.2d, v1.2d, v2.2d
217297 0x00000504 str q0, [x27, x0]
217298 0x00000508 add x0, x0, #0x10
217299 0x0000050c cmp x0, #2, lsl #12
217300 0x00000510 b.ne #0xfffffffffffffe0
217301 0x000004f0 add x2, x0, x26
217302 0x000004f4 add x1, x0, x19
217303 0x000004f8 ldr q1, [x2]
217304 0x000004fc ldr q0, [x1]
217305 0x00000500 fmla v0.2d, v1.2d, v2.2d
217306 0x00000504 str q0, [x27, x0]
217307 0x00000508 add x0, x0, #0x10
217308 0x0000050c cmp x0, #2, lsl #12
217309 0x00000510 b.ne #0xfffffffffffffe0
217310 0x000004f0 add x2, x0, x26
f - fetch
d - decode
n - rename
p - dispatch
i - issue
c - complete
r - retire
= - flushing
[PROBE]
[PROBES SELECTED]
branch.mispredict
L1D.cache.miss
L1l.cache.miss
rename.allocationStalls
decode.earlyFlushes
dispatch.rsStalls
fetch.branchStalls
issue.portBusyStalls

Things coming in 2021

- Support for accelerators, e.g. SME (in progress)
- More comprehensive libc support (in progress)
- Ares and Zeus models (in progress)
- Instruction fusing and micro-oping
- SST integration for the memory model and multi-core
- Other ISAs (via Capstone), e.g. RISC-V
- Integration with gem5? (Drop-in replacement for their OoO)