MB2020 D4.1– SVE-enabled gem5 Simulator
Version 1.0

Document Information

Contract Number	779877
Project Website	www.montblanc-project.eu
Contractual Deadline	M6
Dissemination Level	Public
Nature	Document
Authors	Stephan Diestelhorst, Gabor Dosza, Giacomo Gabrielli, Juha Jäykkä, Javier Setoain
Contributors	
Reviewers	Adria Armejach Sanosa (BSC), Said Derradji (Bull / Atos)
Keywords	BigData and HPC applications, Architecture simulation

Notices: *This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement N° 779877.*

Change Log

Version	Description of Change
v0.1	Initial version of the deliverable
v0.2	Added most instruction descriptions and some bugfixes
v0.3	Draft complete for internal review
v1.0	Finalized report

Contents

Executive Summary

The gem5 simulator is a cycle-level, full-system simulator available under a permissive BSD license from `http://gem5.org`. This particular release (available from `https://gem5.googlesource.com/arm/gem5/+/mb2020/d4.1`) adds support for the Arm Scalable Vector Extension (SVE) ISA extension.[1]

SVE provides advanced vector instructions that decouple the width of the hardware vector registers and computation units from the ISA level. The same application binary will work with hardware vector widths ranging from 128 bits to 2048 bits.

In this deliverable, we add the majority of the instructions present in SVE; especially those that are being generated by vectorising compiler. The current development happens on a branch that tracks the main gem5 repository; we will continue pushing the SVE changes to gem5 into that mainline code base (estimated alpha release is July 2018, and stable support in September 2018). We are making our changes available to the general public under the same permissive open-source BSD license as gem5 itself.

The work for the gem5-SVE code base started under the Mont-Blanc 3 project (funded under the EU Horizon 2020 programme) and has continued under the successor project Mont-Blanc 2020 (also funded by the EU Horizon 2020 programme).

In Mont-Blanc 2020, the resulting model will be used in WP3 for helping port applications (T3.4), and deriving the right sizes for SVE implementations (T3.2); in WP4 to generate traces of applications (T4.4), and as a baseline for a power-modelling enabled simulator (D4.2). The activity factors obtained from gem5 simulation will be used in T5.8 for reliability modelling; and the traces generated (in T7.2) will be used for NoC performance testing in T5.1.

[1] Full SVE ISA documentation is available from `https://developer.arm.com/docs/ddi0584/latest/arm-architecture-reference-manual-supplement-the-scalable-vector-extension-sve-for-armv8-a`

Introduction

This work package delivers a simulation environment capable of providing realistic performance estimates of future microprocessor technologies implementing the Armv8-A Scalable Vector Extension (SVE).

The use of these extensions will be crucial to achieve desired performance levels on future supercomputers, and early work in enabling key applications to exploit them is key to adoption and efficiency of those future architectures. We base the simulation framework on the open source gem5[2]

Using the gem5 simulator provided in this deliverable allows user codes to be tested on simulated systems similar to future supercomputers, and comparison of relative performances of different system configurations by varying certain design parameters such as the vector length (VL), amount and type of caches, etc.

gem5 Simulator CPU Models

The gem5 simulator has several different CPU models, of which three are relevant to this project: *Atomic*, *Minor*, and *O3*. Earlier work done in the Mont-Blanc 3 project had implemented a subset of the SVE instructions. All models share the source code for the instruction implementation, but due to differences in the front end of the O3 model, these instructions were not usable on it. This project implements the required modifications for O3, and adds (most of) the missing SVE instructions, so all three relevant CPU models now fully support all implemented SVE instructions.

The instructions that have not been implemented are unlikely to be emitted by present day compilers (GCC with SVE support, and LLVM with SVE-support patch) and will be simple to add in the near future.

Minor CPU

This CPU model describes a CPU suitable as a LITTLE core in a big.LITTLE configuration. It is an in-order, low power core; but also relatively low performance in many cases. While the main focus of Mont-Blanc is in the O3 cores, it will be important to be able to investigate the performance of in-order Minor cores with vector extensions as well. Such cores can provide surprisingly high performance (at low power consumption) in certain workloads (e.g. xGEMM, FFT) despite their simplicity, so it is important to be able to compare their expected performance with that of O3 cores.

An existing example of such an in-order core would be the Arm Cortex-A53, but it should be noted that the Minor CPU model does not directly correspond to any Arm core.

O3

The O3 core is the out-of-order core model in gem5. We expect the mainstay of future supercomputers to feature out-of-order cores for their higher single threaded peak performance despite the higher cost in energy and silicon area: all programs, including supercomputing applications, have sections which cannot be parallelised, thus benefiting from higher peak performance. We have now enabled for the O3 model all the SVE instructions earlier available for Minor CPU.

[2]See the gem5 website `http://www.gem5.org/` for more information.

An existing example of cores with out-of-order execution would be the Arm Cortex-A73 core, but again, it should be noted that the O3 CPU model does not directly correspond to any Arm core.

Atomic

The Atomic CPU model is mainly used for functional emulation, but it is also very useful for debugging and since its simplicity makes it significantly faster than the more advanced models, it is also often used to bring up optimised workloads before checkpointing them at the point of interest and continuing the simulation from that point using one of the more advanced core models. It can also be used to fast-forward over uninteresting parts of the workload using the same checkpointing technique. Finally, the Atomic model is the easiest to extend model when bringing new CPU features in to the simulator.

gem5 SVE Instruction Support

We have implemented nearly all SVE instructions on all three different CPU models. A subset of the instructions were already implemented before this project for the Minor CPU model, so this list only covers the new instructions, but it should be noted that the implementation of all SVE instructions for the O3 CPU is new in this deliverable.

The new instructions can be classified into five categories: data processing instructions, structure load/store instructions, gather/scatter instructions, first-faulting/non-faulting load instructions, and prefetching instructions. The following subsections describe these classes briefly.

Data Processing Instructions

This is the biggest group by number of instructions and comprises four groups of instructions:

1. Arithmetic, bitwise and logic operations. This group includes, among others:
   - Addition, subtraction, multiplication, division: both saturated and unsaturated.
   - Minimum and maximum.
   - Absolute difference.
   - Bitwise *or*, *and*, *exclusive or* and *not*.
   - Reduction operations: both arithmetic and logic.
   - Logic and arithmetic shift.
   - Sign extension.
   - Comparisons.
   - Complex arithmetic.
   - Index generation.
2. Vector element selection and permutation. This group includes table lookup operations, interleaving and splicing, among others.
3. Predicate generation and manipulation. This group includes, among others, loop control instructions, inversion, active element count, and predicate initialisation based on several different conditions.

4. Scalar operations. Besides the vector operations, SVE also includes scalar instructions that help with the manipulation of stack frames for vector operands, count the number of elements in vectors, etc.

These instructions may come in different varieties: predicated and unpredicated (zeroing and/or merging), integer and floating point (half, single and double precision), and for all different sizes: bytes, half-words (16-bit), words (32-bits) and double words (64-bits).

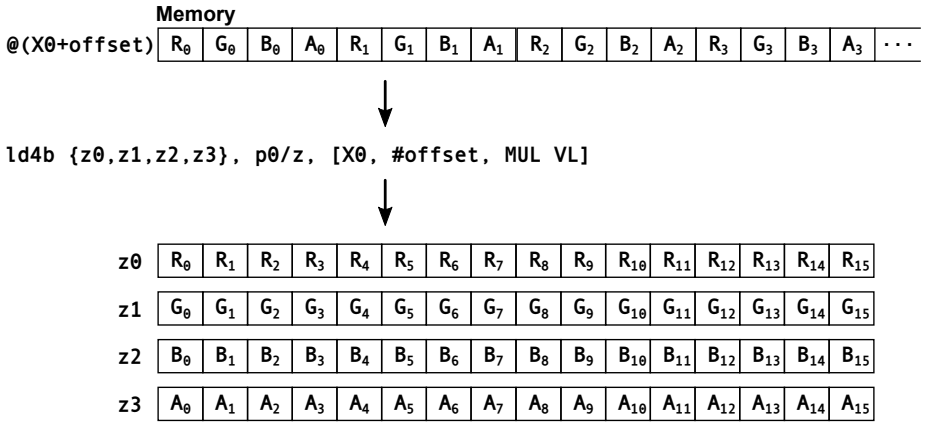
Structure Load/Store Instructions

These instructions facilitate the rearrangement of data from an array of structures (AoS) into a more vector-friendly structure of arrays (SoA). Structure Load/Store instructions can operate with structures of up to four fields of any one single size (byte, half-word, word or double word), and there are two addressing modes for each type of memory operation: scalar plus immediate and scalar plus scalar.

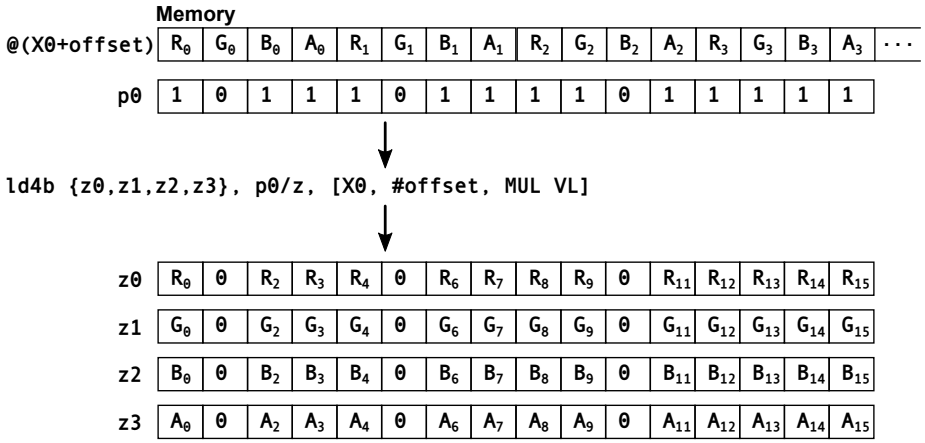
- `LDns { <registers> }, <Pg>/Z, [<Xn|SP>, #<imm>, MUL VL]`
- `LDns { <registers> }, <Pg>/Z, [<Xn|SP>, <Xm>]`
- `STns { <registers> }, <Pg>/Z, [<Xn|SP>, <Xm>]`
- `STns { <registers> }, <Pg>/Z, [<Xn|SP>, #<imm>, MUL VL]`

Where n is the number of elements in the structure (2 to 4); s is the element size (B, H, W or D for 1, 2, 4 and 8 byte elements); *<registers>* is a comma separated list of n contiguous vector registers; *<Pg>* is a predicate register; *<Xn>* and *<Xm>* are scalar registers; and *<imm>* is an optional 4-bit signed immediate value.

In this example, we load an array of pixels in RGBA format into four 128-bit SVE registers:



It's important to note that the predicate register zeroes elements in the destination registers or, in other words, a zero in the predicate register will skip a whole structure in the source array of structures:



Gather/Scatter Instructions

Gather/scatter loads/stores, aka indexed loads/stores, are one of the advanced features of SVE. They are a key tool to enable vectorization of loops with irregular memory accesses.

In this work package we have augmented gem5 to support execution of these operations across all the relevant CPU models.

It is worth noting that such complex operations are suitable to different implementation styles, depending on the specific power/performance/area targets that hardware designers aim to achieve. In this context, in order to preserve the generality of the microarchitectures modelled in gem5, we have chosen a relatively simple but representative approach to implementing such instructions: the approach chosen is based on implementing gather/scatter operations as a sequence of micro-ops, where each micro-op is responsible for the transfer of a single vector element from/to memory (gather/scatter instructions support either 32- or 64-bit wide elements)[3] The different elements in the destination vector register can be written back at the same time, thus avoiding unnecessary serialization when employing the O3 CPU model. We believe that this implementation is a representative baseline against which other, potentially more aggressive (and likely more costly) implementations can be compared against.

First-Faulting/Non-Faulting Instructions

SVE allows vectorization of loops with data dependent termination conditions by means of speculative vector load instructions. One challenge is dealing with faults for data entries that are beyond the data arrays, but are accessed due to the vectorization of the loop. A first-faulting load instruction does not generate an access fault if the memory access for the first active lane completes successfully. Instead, it generates a predicate that indicates which lanes have valid data. This fault predicate is written in the first-faulting system register (FFR) by the speculative loads. The FFR value can then be used by software to adjust the governing predicate to process only valid data lanes in further compute steps. SVE defines first-faulting variants for some gather and contiguous load variants. Furthermore, SVE also provides a non-faulting option for some contiguous load variants which always suppress access faults.

[3]For the sake of completeness, an additional micro-op is used to propagate the vector source operand used for address calculation to the other micro-ops âĂŞ- this avoids the issue of overwriting the vector source register when the same register is used as destination for the instruction.

To enable speculative vector loads in gem5, we restructured the fault handling mechanism in the load-store-queue of both Minor and O3 CPU models (in addition to implementing the instructions themselves in the Arm ISA). We also implemented the FFR system register and the instructions to read/write the FFR register.

In case of gather loads, we created an additional micro-op to compute and write the final fault predicate into the FFR system register. This extra micro-op is necessary since memory accesses for various vector lanes are executed in parallel for gathers. Contiguous loads are implemented as a single operation since the cracking of memory accesses is performed by the load-store-queue. We added first-/non-faulting support to the existing contiguous vector load operations without introducing an extra micro-op.

Prefetching Instructions

We currently treat SVE prefetch instructions as no-ops.

gem5 SVE Instructions Still Missing

Several SVE instructions are not currently emitted by compilers; for this deliverable, we chose to prioritize (1) enabling the execution of the main SVE instructions on the O3 model, and (2) adding support for those missing instructions that are being used by the compilers; and postpone the implementation of the following instructions. These will be enabled in the near future as part of normal gem5 development work, and as no known compiler emits them currently, we believe their omission will not cause problems in simulating real applications.

- Non-temporal loads/stores (`{LD,ST}NT1xx`)
- The following data-processing instructions
  - SIMD and FP scalar versions of `CLASTA` and `CLASTB`
  - `FADDA`
  - `FCADD`
  - Indexed and vector versions of `FCMLA`
  - Indexed version of `FMLA`
  - Indexed version of `FMUL`
  - SIMD and FP scalar versions of `INSR`
  - SIMD and FP scalar versions of `LASTA` and `LASTB`

Acronyms and Abbreviations

- **SVE** – Armv8-A **S**calable **V**ector **E**xtensions
- **O3** – the **O**ut-**o**f-**O**rder gem5 CPU model