(DRAFT) SVP64 Primer

Andrey Miroshnikov, Luke Kenneth Casson Leighton

June 29, 2022

1 List of Acronyms

AVX-512 Intel Advanced Vector Extensions 512-bit

CPU Central Processing Unit

DCT Discrete Cosine Transform

DSP Digital Signal Processors

FFT Fast Fourier Transform

ISA Instruction Set Architecture

MMX Intel's first SIMD implementation

RVV RISC-V Vector extension

SIMD Single Instruction Multiple Data

SWAR SIMD Within A Register (see Flynn's Taxonomy)

SV (Scalable) Simple Vectorisation or Simple-V

VLIW Very Long Instruction Word

VSX 128-bit Packed SIMD Extension to the Power ISA

2 Summary

The proposed SV is a Scalable Vector Specification for a hardware for-loop **that ONLY uses scalar instructions**.

- The Power ISA v3.1 Specification is not altered in any way. v3.1 Code-compatibility is guaranteed.
- Does not require sacrificing 32-bit Major Opcodes.
- Does not require adding duplicates of instructions (popcnt, popcntw, popcntd, vpopcntb, vpopcnth, vpopcntw, vpopcntd)
- Fully abstracted: does not create Micro-architectural dependencies (no fixed "Lane" size), one binary works across all existing *and future* implementations.
- Specifically designed to be easily implemented on top of an existing Micro-architecture (especially Superscalar Out-of-Order Multi-issue) without disruptive full architectural redesigns.
- Divided into Compliancy Levels to suit differing needs.
- At the highest Compliancy Level only requires five instructions (SVE2 requires appx 9,000. AVX-512 around 10,000. RVV around 300).
- Predication, an often-requested feature, is added cleanly (without modifying the v3.1 Power ISA)
- In-registers arbitrary-sized Matrix Multiply is achieved in three instructions (without adding any v3.1 Power ISA instructions)
- Full DCT and FFT RADIX2 Triple-loops are achieved with dramatically reduced instruction count, and power consumption expected to greatly reduce. Normally found only in high-end VLIW DSP (TI MSP, Qualcomm Hexagon)
- Fail-First Load/Store allows Vectorised high performance strncpy to be implemented in around 14 instructions (hand-optimised VSX assembler is 240).
- Inner loop of MP3 implemented in under 100 instructions (gcc produces 450 for the same function on POWER9).

All areas investigated so far consistently showed reductions in executable size, which as outlined in [1] has an indirect reduction in power consumption due to less I-Cache/TLB pressure and also Issue remaining idle for long periods. Simple-V has been specifically and carefully crafted to respect the Power ISA's Supercomputing pedigree.

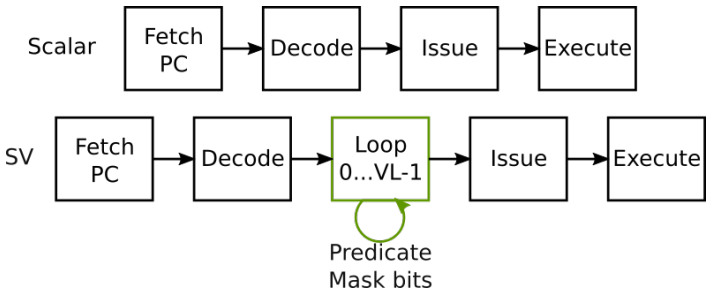

Figure 1: Showing how SV fits in between Decode and Issue

2.1 What is SIMD?

SIMD is a way of partitioning existing CPU registers of 64-bit length into smaller 8-, 16-, 32-bit pieces. [1][2] These partitions can then be operated on simultaneously, and the initial values and results being stored as entire 64-bit registers (SWAR). The SIMD instruction opcode includes the data width and the operation to perform.

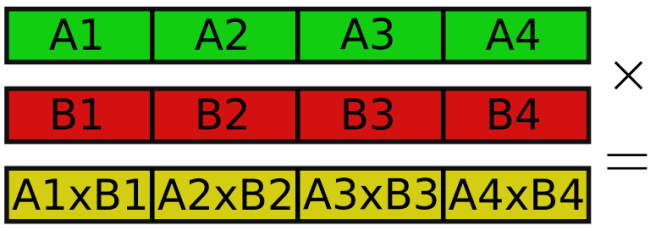

Figure 2: SIMD multiplication

This method can have a huge advantage for rapid processing of vector-type data (image/video, physics simulations, cryptography, etc.), [3], and thus on paper is very attractive compared to scalar-only instructions. *As long as the data width fits the workload, everything is fine.*

2.2 Shortfalls of SIMD

SIMD registers are of a fixed length and thus to achieve greater performance, CPU architects typically increase the width of registers (to 128-, 256-, 512-bit etc) for more partitions.

Additionally, binary compatibility is an important feature, and thus each doubling of SIMD registers also expands the instruction set. The number of instructions quickly balloons and this can be seen in for example IA-32 expanding from 80 to about 1400 instructions since the 1970s[1].

Five digit Opcode proliferation (10,000 instructions) is overwhelming. The following are just some of the reasons why SIMD is unsustainable as the number of instructions increase:

- Hardware design, ASIC routing etc.
- Compiler design
- Documentation of the ISA
- Manual coding and optimisation
- Time to support the platform
- Compilance Suite development and testing
- Protracted Variable-Length encoding (x86) severely compromises Multi-issue decoding

2.3 Scalable Vector Architectures

An older alternative exists to utilise data parallelism - vector architectures. Vector CPUs collect operands from the main memory, and store them in large, sequential vector registers.

A simple vector processor might operate on one element at a time, however as the element operations are usually independent, a processor could be made to compute all of the vector's elements simultaneously, taking advantage of multiple pipelines.

Typically, today's vector processors can execute two, four, or eight 64-bit elements per clock cycle. [1]. Vector ISAs are specifically designed to deal with (in hardware) fringe cases where an algorithm's element count is not a multiple of the underlying hardware "Lane" width. The element data width is variable (8 to 64-bit just like in SIMD) but it is the *number* of elements being variable under control of a "setvl" instruction that specifically makes Vector ISAs "Scalable"

RVV supports a VL of up to 2^{16} or 65536 bits, which can fit 1024 64-bit words. [4]. The Cray-1 had 8 Vector Registers with up to 64 elements (64-bit each). An early Draft of RVV supported overlaying the Vector Registers onto the Floating Point registers, similar to MMX.

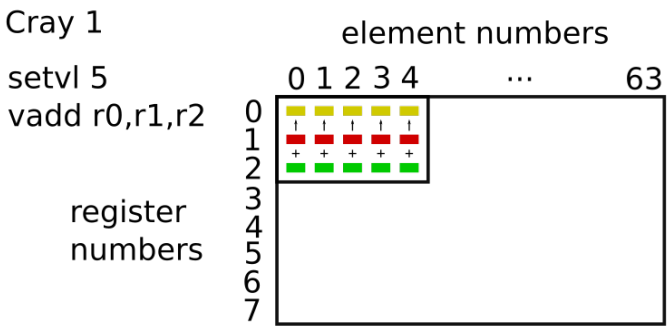

Figure 3: Cray Vector registers: 8 registers, 64 elements each

Simple-V's "Vector" Registers (a misnomer) are specifically designed to fit on top of the Scalar (GPR, FPR) register files, which are extended from the default of 32, to 128 entries in the high-end Compliancy Levels. This is a primary reason why Simple-V can be added on top of an existing Scalar ISA, and *in particular* why there is no need to add explicit Vector Registers or Vector instructions. The diagram below shows *conceptually* how a Vector's elements are sequentially and linearly mapped onto the *Scalar* register file:

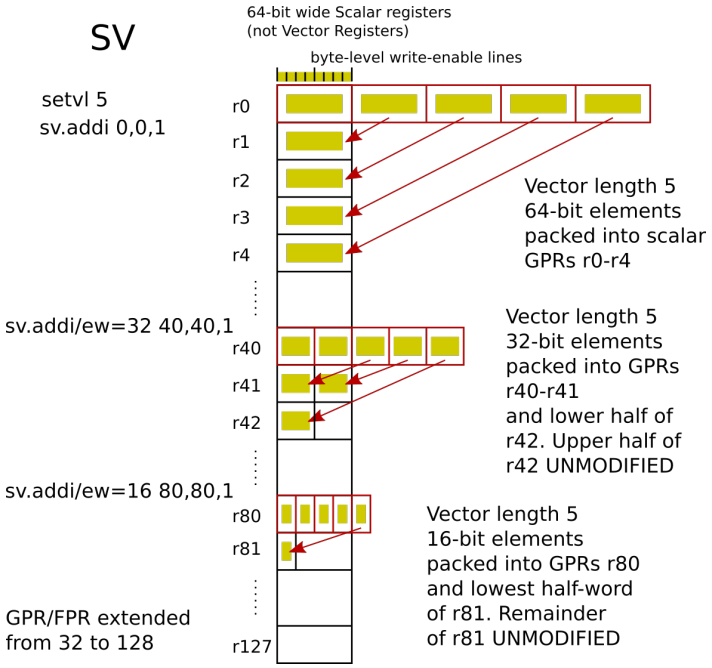

Figure 4: three instructions, same vector length, different element widths

2.4 Simple Vectorisation

SV is a Scalable Vector ISA designed for hybrid workloads (CPU, GPU, VPU, 3D?). Includes features normally found only on Cray-style Supercomputers (Cray-1, NEC SX-Aurora) and GPUs. Keeps to a strict uniform RISC paradigm, leveraging a scalar ISA by using "Prefixing". **No dedicated vector opcodes exist in SV, at all**. SVP64 uses 25% of the Power ISA v3.1 64-bit Prefix space (EXT001) to create the SV Vectorisation Context for the 32-bit Scalar Suffix.

Main design principles

- Introduce by implementing on top of existing Power ISA
- Effectively a **hardware for-loop**, pauses main PC, issues multiple scalar operations
- Strictly preserves (leverages) underlying scalar execution dependencies as if the for-loop had been expanded into actual scalar instructions ("preserving Program Order")
- Augments existing instructions by adding "tags" - provides Vectorisation "context" rather than adding new opcodes.
- Does not modify or deviate from the underlying scalar Power ISA unless there's a significant performance boost or other advantage in the vector space
- Aimed at Supercomputing: avoids creating significant *sequential dependency hazards*, allowing **high performance multi-issue superscalar microarchitectures** to be leveraged.

Advantages include:

- Easy to create first (and sometimes only) implementation as a literal for-loop in hardware, simulators, and compilers.
- Obliterates SIMD opcode proliferation ($O(N^6)$) as well as dedicated Vectorisation ISAs. No more separate vector instructions.
- Reducing maintenance overhead (no separate Vector instructions). Adding any new Scalar instruction *automatically adds a Vectorised version of the same.*
- Easier for compilers, coders, documentation

3 References

References

[1] "SIMD instructions considered harmful."
`https://www.sigarch.org/simd-instructions-considered-harmful/`.

[2] "High-performance embedded computing."
`https://www.sciencedirect.com/topics/computer-science/single-instruction-multiple-data`.

[3] "Webassembly and SIMD."
`https://medium.com/wasmer/webassembly-and-simd-13badb9bf1a8`.

[4] "Risc-v "v" vector extension."
`https://github.com/riscv/riscv-v-spec/blob/master/v-spec.adoc`.