Scalable vector multimedia optimisations

RISC-V V and ARM SVE2 extensions introduction

Rémi Denis-Courmont

Remlab Tmi

Ixelles, Belgium, 4th February 2023

Outline

Attendees advisory

Disclaimer

The opinions expressed therein solely represent the personal views of the author.

Attendees advisory

Disclaimer

The opinions expressed therein solely represent the personal views of the author.

- I speak fast.
- I do not articulate well.

Attendees advisory

Disclaimer

The opinions expressed therein solely represent the personal views of the author.

- I speak fast.
- I do not articulate well.

If you did not understand. . .

Do interrupt me if needed!

Who am I?

- 16th FOSDEM attendance (since 2004)...

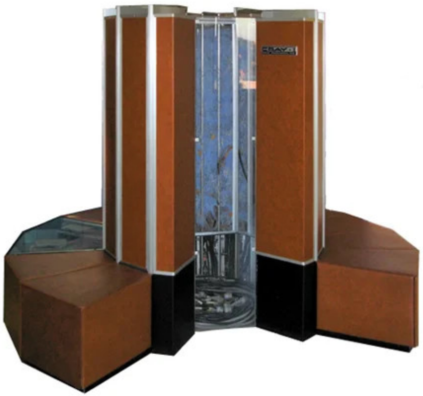
Who am I?

- 16th FOSDEM attendance (since 2004). . .
- 1st FOSDEM presentation!
- Not relevant to this presentation.

Outline

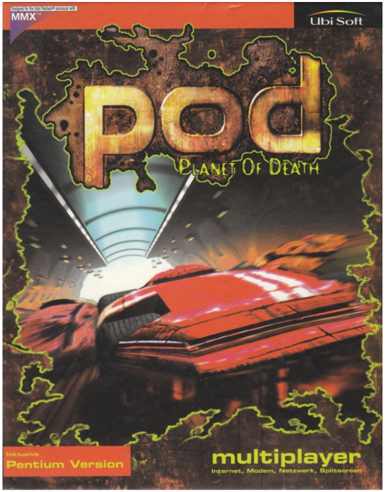
What is this?

You may know if older than me.



Planet of Death

You may know if my age.


Designed for the Intel Pentium® processor with MMX™
Ubi Soft
pod
Planet Of Death
Inklusive Pentium Version
multiplayer
Internet, Modem, Netzwerk, Splitscreen

Single Instruction Multiple Data

- x86
  - 64 bits: MMX (1997)

Single Instruction Multiple Data

- x86
  - 64 bits: MMX (1997)
  - 128 bits: SSE (1999)

Single Instruction Multiple Data

- x86
    - 64 bits: MMX (1997)
    - 128 bits: SSE (1999), SSE2 (2000)

Single Instruction Multiple Data

- x86
  - 64 bits: MMX (1997)
  - 128 bits: SSE (1999), SSE2 (2000)... AVX (2008)
  - 256 bits: AVX2 (2011)

Single Instruction Multiple Data

- x86
  - 64 bits: MMX (1997)
  - 128 bits: SSE (1999), SSE2 (2000)... AVX (2008)
  - 256 bits: AVX2 (2011)
  - 512 bits: AVX-512 (~~2013~~ 2017)

Single Instruction Multiple Data

- x86
  - 64 bits: MMX (1997)
  - 128 bits: SSE (1999), SSE2 (2000)... AVX (2008)
  - 256 bits: AVX2 (2011)
  - 512 bits: AVX-512 (~~2013~~ 2017)
- ARM
  - *32 bits*: ARMv6 SIMD (2002)

Single Instruction Multiple Data

- x86
  - 64 bits: MMX (1997)
  - 128 bits: SSE (1999), SSE2 (2000)... AVX (2008)
  - 256 bits: AVX2 (2011)
  - 512 bits: AVX-512 (~~2013~~ 2017)
- ARM
  - *32 bits*: ARMv6 SIMD (2002)
  - 128 bits: ARMv7 AdvSIMD, a.k.a. *NEON* (2005)
  - 128 bits: ARMv8 A64 AdvSIMD, also a.k.a. *NEON* (2012)
- RISC-V
  - *ENOSYS*

Need to rewrite assembler every time.

Outline

Vector length

Dear CPU, what is your vector length?

`csrr t0, vlenb` /* Vector LENgth in Bytes */

Vector length

Dear CPU, what is your vector length?

```
csrr t0, vlenb /* Vector LENgth in Bytes */
```

Dear CPU, how many elements can you process?

```
csrr t0, vlenb
slri t0, t0, #2 /* 32-bit elements */
```

Vector length

Dear CPU, what is your vector length?

```
csrr t0, vlenb /* Vector LENgth in Bytes */
```

Dear CPU, how many elements can you process?

```
csrr t0, vlenb
slri t0, t0, #2 /* 32-bit elements */
```

1. Write main loop.
2. Unroll main loop.
3. Deal with edges.

That is how Clang vectorisation does it. . .

Vector length

Possible answers

- A power of two!

[1] except *embedded* RISC-V

Vector length

Possible answers

- A power of two!
- 128 bits: guaranteed minimum[1].
- 256, 512 bits: silicon designs announced, yet to ship.
- 1024 bits, even 4096 proposed in (RISC-V) simulations.
- 65536 bits: syntactic maximum (RISC-V).

[1]except *embedded* RISC-V

Predication

- Not *completely* new concept
- Essential to variable vector length programming model

Predication

- Not *completely* new concept
- Essential to variable vector length programming model
- Vector of boolean
- Selects loaded/modified/stored elements

ARMv9 example

```
        MOV     x10, xzr
        B       2f
1:
        ...
2:      WHILELT p0.s, x10, x0
        B.FIRST 1b
```

Unrolling

- Ill fit with predication
- Vector processing $\neq$ SIMD
- Just don't unroll. . .

Unrolling

- Ill fit with predication
- Vector processing $\neq$ SIMD
- Just don't unroll...
- ARM: "*SVE streaming mode*"
    - Higher latency
    - Larger vectors (potentially)
    - Higher throughput
- No over-alignment required! Yay!

Outline

SVE

- Original SVE pretty useless for multimedia.

SVE

- Original SVE pretty useless for multimedia.
- SVE2 copies most NEON mnemonics.
- Just insert the predicate register operand!
- Famous last words.

SVE

Pick:

1. 1 of 10 `WHILExx` instruction: `WHILELT`, `WHILELO`, ...
2. a predicate register,
3. the element size: B, H, S or D.
4. a branch condition: `B.FIRST`, `B.LAST`...

SVE

Pick:

1. 1 of 10 `WHILExx` instruction: `WHILELT`, `WHILELO`, ...
2. a predicate register,
3. the element size: *B*, *H*, *S* or *D*.
4. a branch condition: `B.FIRST`, `B.LAST`...

- Remaining elements → Predicate register
- Predicate register → Condition flags
- Subtracted count → Output GP register

SVE

Pick:

1. 1 of 10 `WHILExx` instruction: `WHILELT`, `WHILELO`, ...
2. a predicate register,
3. the element size: *B*, *H*, *S* or *D*.
4. a branch condition: `B.FIRST`, `B.LAST`...

- Remaining elements → Predicate register
- Predicate register → Condition flags
- Subtracted count → Output GP register

Stop pretending AArch64 is a RISC.

Processor feature detection

It would be too easy without it.

- Preprocessor: `defined(__ARM_FEATURE_SVE2)`
- Bare metal: `ID_AA64*_EL1` register fields

Processor feature detection

It would be too easy without it.

- Preprocessor: `defined(__ARM_FEATURE_SVE2)`
- Bare metal: `ID_AA64*_EL1` register fields
- Linux: bits from `AT_HWCAP2` auxillary vector entry
    - `HWCAP2_SVE2` is probably what you want
    - `HWCAP2_SVEPMULL`
    - `HWCAP2_SVEBITPERM`
    - `HWCAP2_SVE2P1`

Examples

```
#include <sys/auxv.h>
(getauxval(AT_HWCAP2) & HWCAP2_SVE2)
```

Processor feature detection

It would be too easy without it.

- Preprocessor: `defined(__ARM_FEATURE_SVE2)`
- Bare metal: `ID_AA64*_EL1` register fields
- Linux: bits from `AT_HWCAP2` auxillary vector entry
  - `HWCAP2_SVE2` is probably what you want
  - `HWCAP2_SVEPMULL`
  - `HWCAP2_SVEBITPERM`
  - `HWCAP2_SVE2P1`

Examples

```
#include <sys/auxv.h>
(getauxval(AT_HWCAP2) & HWCAP2_SVE2)
```

- Other OSes: lol

Availability

- Specifications
  - SVE (2016). . .

Availability

- Specifications
    - SVE (2016). . . explicitly not intended for multimedia payloads
    - SVE2 (2019)
    - SME / Scalable Matrix Extension (2021)
    - Streaming SVE

Availability

- Specifications
  - SVE (2016)... explicitly not intended for multimedia payloads
  - SVE2 (2019)
  - SME / Scalable Matrix Extension (2021)
  - Streaming SVE
- Hardware
  - Cortex-X2, Cortex-A510, Cortex-A710
  - Arm DynamIQ-110 cluster (2022)

Availability

- Specifications
  - SVE (2016)... explicitly not intended for multimedia payloads
  - SVE2 (2019)
  - SME / Scalable Matrix Extension (2021)
  - Streaming SVE
- Hardware
  - Cortex-X2, Cortex-A510, Cortex-A710
  - Arm DynamIQ-110 cluster (2022)
  - Samsung Exynos 2200
  - Qualcomm SM8450 Snapdragon 8 Gen 1

Outline

Predication

Not sure if simpler or more intricate

Vector configuration

```
vsetvli t0, a4, e16, m1, ta, ma
```

- `a4` = available elements (input)
- Output operand: `t0` = vector length (output)
- Element size: `e16` $\leftrightarrow$ 16 bits
- Group size: `m1` $\leftrightarrow$ 1 vector $\Leftrightarrow$ no grouping
- Tail mode: `ta` agnostic $\Leftrightarrow$ don't care
- Mask mode: `ma` agnostic $\Leftrightarrow$ don't care

Registers

- Prefer greatest power-of-two multiple-numbered vectors
  ↰ grouping and segmentation require aligned numbers
- FP registers $\neq$ Vectors
  ↳ more registers for hybrid scalar/vector functions

Warning

Mind the FP calling conventions!

Bit shuffling

- Segmented loads & stores up to 8 structures (ARM can do up to 4 only)
- GP register-strided loads & stores
- . . . including negative strides.

Example

# Load a *column* of 16-bit samples
# at [*a*0] with pitch *a*4 in vector *v*8.

```
vlse16.v v8, (a0), a4
```

Bit shuffling

- Segmented loads & stores up to 8 structures (ARM can do up to 4 only)
- GP register-strided loads & stores
- . . . including negative strides.

Example

# Load a *column* of 16-bit samples
# at [*a*0] with pitch *a*4 in vector *v*8.

```
vlse16.v v8, (a0), a4
```

- But. . . no vector↔vector transpose/zip

Processor feature detection

- Preprocessor:
  - Element size: `__riscv_v_elen_fp` = 32 or 64
  - `__riscv_vector` ⇒ *elen* $\geq$ 64 bits
  - Vector length: `__riscv_zvl`{32,64,128,...}`b`
  - `__riscv_vector` ⇒ *VL* $\geq$ 128 bits
- Hardware:

Processor feature detection

- Preprocessor:
  - Element size: `__riscv_v_elen_fp` = 32 or 64
  - `__riscv_vector` ⇒ *elen* ≥ 64 bits
  - Vector length: `__riscv_zvl`{32,64,128,...}`b`
  - `__riscv_vector` ⇒ *VL* ≥ 128 bits
- Hardware: DeviceTree cpu node property
- Linux: bit 21 from `AT_HWCAP` auxillary vector entry

Examples

```
#include <sys/auxv.h>
(getauxval(AT_HWCAP) & (1U << ('V' - 'A')))
```

Availability

- Specifications
  - RISC-V "V" Vector extension version 1.0 (ratified 2021)
  - Not integrated in RISC-V unprivileged specificaton yet

Availability

- Specifications
  - RISC-V "V" Vector extension version 1.0 (ratified 2021)
  - Not integrated in RISC-V unprivileged specificaton yet
- Hardware
  - Open-source designs exist (but. . .)
  - T-Head (Alibaba): draft version 0.7.1 only so far
  - SiFive: several IPs announced, not sold yet
  - Andes: AX45, not sold yet

Further references

- Arm Architecture Reference Manual, ARMv8-A
- Arm SVE supplement
- Arm SME supplement
- RISC-V Vector extension version 1.0.
- FFmpeg source code.

Any questions?