Scalable Vector Extension support for AArch64 Linux

Author: Dave Martin <Dave.Martin@arm.com>

Date: 4 August 2017

This document outlines briefly the interface provided to userspace by Linux in order to support use of the ARM Scalable Vector Extension (SVE).

This is an outline of the most important features and issues only and not intended to be exhaustive.

This document does not aim to describe the SVE architecture or programmer's model. To aid understanding, a minimal description of relevant programmer's model features for SVE is included in Appendix A.

1. General

- SVE registers Z0..Z31, P0..P15 and FFR and the current vector length VL, are tracked per-thread.
- The presence of SVE is reported to userspace via HWCAP_SVE in the aux vector AT_HWCAP entry. Presence of this flag implies the presence of the SVE instructions and registers, and the Linux-specific system interfaces described in this document. SVE is reported in /proc/cpuinfo as "sve".
- Support for the execution of SVE instructions in userspace can also be detected by reading the CPU ID register ID_AA64PFR0_EL1 using an MRS instruction, and checking that the value of the SVE field is nonzero. [3]

  It does not guarantee the presence of the system interfaces described in the following sections: software that needs to verify that those interfaces are present must check for HWCAP_SVE instead.
- On hardware that supports the SVE2 extensions, HWCAP2_SVE2 will also be reported in the AT_HWCAP2 aux vector entry. In addition to this, optional extensions to SVE2 may be reported by the presence of:

  HWCAP2_SVE2 HWCAP2_SVEAES HWCAP2_SVEPMULL HWCAP2_SVEBITPERM
  HWCAP2_SVESHA3 HWCAP2_SVESM4

  This list may be extended over time as the SVE architecture evolves.

  These extensions are also reported via the CPU ID register ID_AA64ZFR0_EL1, which userspace can read using an MRS instruction. See elf_hwcaps.txt and cpu-feature-registers.txt for details.
- Debuggers should restrict themselves to interacting with the target via the NT_ARM_SVE regset. The recommended way of detecting support for this regset is to connect to a target process first and then attempt a ptrace(PTRACE_GETREGSET, pid, NT_ARM_SVE, &iov).
- Whenever SVE scalable register values (Zn, Pn, FFR) are exchanged in memory between userspace and the kernel, the register value is encoded in memory in an endianness-invariant layout, with bits [(8 * i + 7) : (8 * i)] encoded at byte offset i from the start of the memory representation. This affects for example the signal frame (struct sve_context) and ptrace interface (struct user_sve_header) and associated data.

  Beware that on big-endian systems this results in a different byte order than for the FPSIMD V-registers, which are stored as single host-endian 128-bit values, with bits [(127 - 8 * i) : (120 - 8 * i)] of the register encoded at byte offset i. (struct fpsimd_context, struct user_fpsimd_state).

2. Vector length terminology

The size of an SVE vector (Z) register is referred to as the "vector length".

To avoid confusion about the units used to express vector length, the kernel adopts the following conventions:

- Vector length (VL) = size of a Z-register in bytes
- Vector quadwords (VQ) = size of a Z-register in units of 128 bits

(So, VL = 16 * VQ.)

The VQ convention is used where the underlying granularity is important, such as in data structure definitions. In most other situations, the VL convention is used. This is consistent with the meaning of the "VL" pseudo-register in the SVE instruction set architecture.

3. System call behaviour

- On syscall, V0..V31 are preserved (as without SVE). Thus, bits [127:0] of Z0..Z31 are preserved. All other bits of Z0..Z31, and all of P0..P15 and FFR become unspecified on return from a syscall.
- The SVE registers are not used to pass arguments to or receive results from any syscall.
- In practice the affected registers/bits will be preserved or will be replaced with zeros on return from a syscall, but userspace

should not make assumptions about this. The kernel behaviour may vary on a case-by-case basis.

- All other SVE state of a thread, including the currently configured vector length, the state of the PR_SVE_VL_INHERIT flag, and the deferred vector length (if any), is preserved across all syscalls, subject to the specific exceptions for execve() described in section 6.

  In particular, on return from a fork() or clone(), the parent and new child process or thread share identical SVE configuration, matching that of the parent before the call.

4. Signal handling

- A new signal frame record sve_context encodes the SVE registers on signal delivery. [1]
- This record is supplementary to fpsimd_context. The FPSR and FPCR registers are only present in fpsimd_context. For convenience, the content of V0..V31 is duplicated between sve_context and fpsimd_context.
- The signal frame record for SVE always contains basic metadata, in particular the thread's vector length (in sve_context.vl).
- The SVE registers may or may not be included in the record, depending on whether the registers are live for the thread. The registers are present if and only if: sve_context.head.size >= SVE_SIG_CONTEXT_SIZE(sve_vq_from_vl(sve_context.vl)).
- If the registers are present, the remainder of the record has a vl-dependent size and layout. Macros SVE_SIG_* are defined [1] to facilitate access to the members.
- Each scalable register (Zn, Pn, FFR) is stored in an endianness-invariant layout, with bits [(8 * i + 7) : (8 * i)] stored at byte offset i from the start of the register's representation in memory.
- If the SVE context is too big to fit in sigcontext.__reserved[], then extra space is allocated on the stack, an extra_context record is written in __reserved[] referencing this space. sve_context is then written in the extra space. Refer to [1] for further details about this mechanism.

5. Signal return

When returning from a signal handler:

- If there is no sve_context record in the signal frame, or if the record is present but contains no register data as desribed in the previous section, then the SVE registers/bits become non-live and take unspecified values.
- If sve_context is present in the signal frame and contains full register data, the SVE registers become live and are populated with the specified data. However, for backward compatibility reasons, bits [127:0] of Z0..Z31 are always restored from the corresponding members of fpsimd_context.vregs[] and not from sve_context. The remaining bits are restored from sve_context.
- Inclusion of fpsimd_context in the signal frame remains mandatory, irrespective of whether sve_context is present or not.
- The vector length cannot be changed via signal return. If sve_context.vl in the signal frame does not match the current vector length, the signal return attempt is treated as illegal, resulting in a forced SIGSEGV.

6. prctl extensions

Some new prctl() calls are added to allow programs to manage the SVE vector length:

prctl(PR_SVE_SET_VL, unsigned long arg)

Sets the vector length of the calling thread and related flags, where arg == vl | flags. Other threads of the calling process are unaffected.

vl is the desired vector length, where sve_vl_valid(vl) must be true.

flags:

PR_SVE_VL_INHERIT

Inherit the current vector length across execve(). Otherwise, the vector length is reset to the system default at execve(). (See Section 9.)

PR_SVE_SET_VL_ONEXEC

Defer the requested vector length change until the next execve() performed by this thread.

The effect is equivalent to implicit exceution of the following call immediately after the next execve() (if any) by the thread:

prctl(PR_SVE_SET_VL, arg & ~PR_SVE_SET_VL_ONEXEC)

This allows launching of a new program with a different vector length, while avoiding runtime side effects in the caller.

Without PR_SVE_SET_VL_ONEXEC, the requested change takes effect immediately.

Return value: a nonnegative on success, or a negative value on error:

EINVAL: SVE not supported, invalid vector length requested, or
invalid flags.

On success:

- Either the calling thread's vector length or the deferred vector length to be applied at the next execve() by the thread (dependent on whether PR_SVE_SET_VL_ONEXEC is present in arg), is set to the largest value supported by the system that is less than or equal to vl. If vl == SVE_VL_MAX, the value set will be the largest value supported by the system.
- Any previously outstanding deferred vector length change in the calling thread is cancelled.
- The returned value describes the resulting configuration, encoded as for PR_SVE_GET_VL. The vector length reported in this value is the new current vector length for this thread if PR_SVE_SET_VL_ONEXEC was not present in arg; otherwise, the reported vector length is the deferred vector length that will be applied at the next execve() by the calling thread.
- Changing the vector length causes all of P0..P15, FFR and all bits of Z0..Z31 except for Z0 bits [127:0] .. Z31 bits [127:0] to become unspecified. Calling PR_SVE_SET_VL with vl equal to the thread's current vector length, or calling PR_SVE_SET_VL with the PR_SVE_SET_VL_ONEXEC flag, does not constitute a change to the vector length for this purpose.

prctl(PR_SVE_GET_VL)

Gets the vector length of the calling thread.

The following flag may be OR-ed into the result:

PR_SVE_VL_INHERIT

Vector length will be inherited across execve().

There is no way to determine whether there is an outstanding deferred vector length change (which would only normally be the case between a fork() or vfork() and the corresponding execve() in typical use).

To extract the vector length from the result, bitwise and it with PR_SVE_VL_LEN_MASK.

Return value: a nonnegative value on success, or a negative value on error:
EINVAL: SVE not supported.

7. ptrace extensions

- A new regset NT_ARM_SVE is defined for use with PTRACE_GETREGSET and PTRACE_SETREGSET.

Refer to [2] for definitions.

The regset data starts with struct user_sve_header, containing:

size

Size of the complete regset, in bytes. This depends on vl and possibly on other things in the future.

If a call to PTRACE_GETREGSET requests less data than the value of size, the caller can allocate a larger buffer and retry in order to read the complete regset.

max_size

Maximum size in bytes that the regset can grow to for the target thread. The regset won't grow bigger than this even if the target thread changes its vector length etc.

vl

Target thread's current vector length, in bytes.

max_vl

Maximum possible vector length for the target thread.

flags

either

SVE_PT_REGS_FPSIMD

SVE registers are not live (GETREGSET) or are to be made non-live (SETREGSET).

The payload is of type struct user_fpsimd_state, with the same meaning as for NT_PRFPREG, starting at offset SVE_PT_FPSIMD_OFFSET from the start of user_sve_header.

Extra data might be appended in the future: the size of the payload should be obtained using SVE_PT_FPSIMD_SIZE(vq, flags).

vq should be obtained using sve_vq_from_vl(vl).

or

SVE_PT_REGS_SVE

SVE registers are live (GETREGSET) or are to be made live (SETREGSET).

The payload contains the SVE register data, starting at offset SVE_PT_SVE_OFFSET from the start of user_sve_header, and with size SVE_PT_SVE_SIZE(vq, flags);

... OR-ed with zero or more of the following flags, which have the same meaning and behaviour as the corresponding PR_SET_VL_* flags:

SVE_PT_VL_INHERIT

SVE_PT_VL_ONEXEC (SETREGSET only).

- The effects of changing the vector length and/or flags are equivalent to those documented for PR_SVE_SET_VL.

  The caller must make a further GETREGSET call if it needs to know what VL is actually set by SETREGSET, unless is it known in advance that the requested VL is supported.

- In the SVE_PT_REGS_SVE case, the size and layout of the payload depends on the header fields. The SVE_PT_SVE_*() macros are provided to facilitate access to the members.

- In either case, for SETREGSET it is permissible to omit the payload, in which case only the vector length and flags are changed (along with any consequences of those changes).

- For SETREGSET, if an SVE_PT_REGS_SVE payload is present and the requested VL is not supported, the effect will be the same as if the payload were omitted, except that an EIO error is reported. No attempt is made to translate the payload data to the correct layout for the vector length actually set. The thread's FPSIMD state is preserved, but the remaining bits of the SVE registers become unspecified. It is up to the caller to translate the payload layout for the actual VL and retry.

- The effect of writing a partial, incomplete payload is unspecified.

8. ELF coredump extensions

- A NT_ARM_SVE note will be added to each coredump for each thread of the dumped process. The contents will be equivalent to the data that would have been read if a PTRACE_GETREGSET of NT_ARM_SVE were executed for each thread when the coredump was generated.

9. System runtime configuration

- To mitigate the ABI impact of expansion of the signal frame, a policy mechanism is provided for administrators, distro maintainers and developers to set the default vector length for userspace processes:

/proc/sys/abi/sve_default_vector_length

Writing the text representation of an integer to this file sets the system default vector length to the specified value, unless the value is greater than the maximum vector length supported by the system in which case the default vector length is set to that maximum.

The result can be determined by reopening the file and reading its contents.

At boot, the default vector length is initially set to 64 or the maximum supported vector length, whichever is smaller. This determines the initial vector length of the init process (PID 1).

Reading this file returns the current system default vector length.

- At every execve() call, the new vector length of the new process is set to the system default vector length, unless

  - PR_SVE_VL_INHERIT (or equivalently SVE_PT_VL_INHERIT) is set for the calling thread, or
  - a deferred vector length change is pending, established via the PR_SVE_SET_VL_ONEXEC flag (or SVE_PT_VL_ONEXEC).

- Modifying the system default vector length does not affect the vector length of any existing process or thread that does not make an execve() call.

Appendix A. SVE programmer's model (informative)

This section provides a minimal description of the additions made by SVE to the ARMv8-A programmer's model that are relevant to this document.

Note: This section is for information only and not intended to be complete or to replace any architectural specification.

A.1. Registers

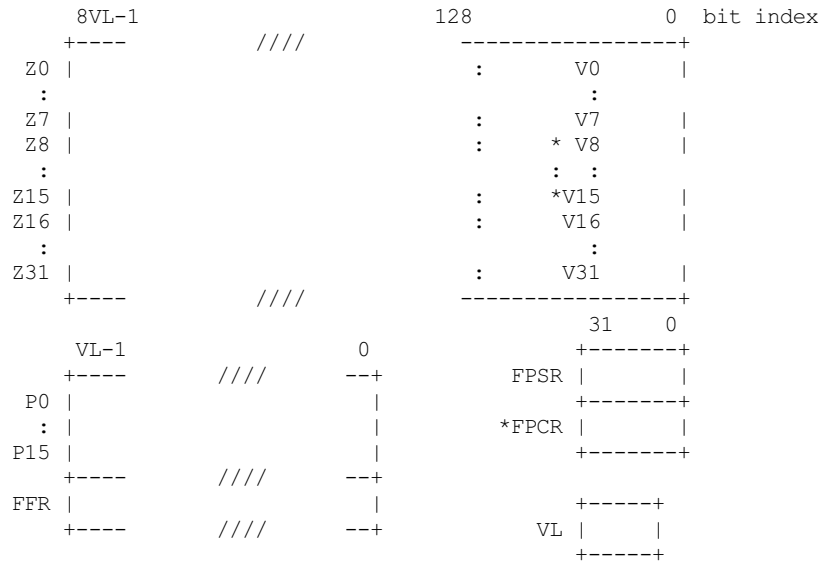
In A64 state, SVE adds the following:

- 32 8VL-bit vector registers Z0..Z31 For each Zn, Zn bits [127:0] alias the ARMv8-A vector register Vn.

  A register write using a Vn register name zeros all bits of the corresponding Zn except for bits [127:0].
- 16 VL-bit predicate registers P0..P15
- 1 VL-bit special-purpose predicate register FFR (the "first-fault register")
- a VL "pseudo-register" that determines the size of each vector register

  The SVE instruction set architecture provides no way to write VL directly. Instead, it can be modified only by EL1 and above, by writing appropriate system registers.
- The value of VL can be configured at runtime by EL1 and above: 16 <= VL <= VLmax, where VL must be a multiple of 16.
- The maximum vector length is determined by the hardware: 16 <= VLmax <= 256.

  (The SVE architecture specifies 256, but permits future architecture revisions to raise this limit.)
- FPSR and FPCR are retained from ARMv8-A, and interact with SVE floating-point operations in a similar way to the way in which they interact with ARMv8 floating-point operations:



(*) callee-save:

This only applies to bits [63:0] of Z-/V-registers. FPCR contains callee-save and caller-save bits. See [4] for details.

A.2. Procedure call standard

The ARMv8-A base procedure call standard is extended as follows with respect to the additional SVE register state:

- All SVE register bits that are not shared with FP/SIMD are caller-save.
- Z8 bits [63:0] .. Z15 bits [63:0] are callee-save.

  This follows from the way these bits are mapped to V8..V15, which are caller- save in the base procedure call standard.

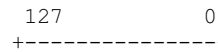
Appendix B. ARMv8-A FP/SIMD programmer's model

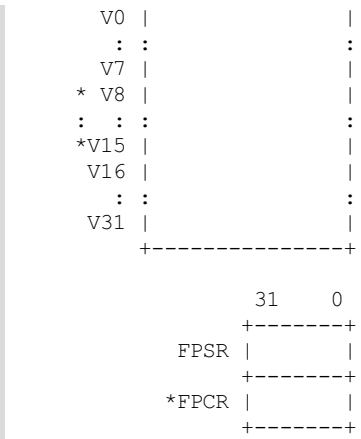
Note: This section is for information only and not intended to be complete or to replace any architectural specification.

Refer to [4] for more information.

ARMv8-A defines the following floating-point / SIMD register state:

- 32 128-bit vector registers V0..V31
- 2 32-bit status/control registers FPSR, FPCR





(*) callee-save:
This only applies to bits [63:0] of V-registers. FPCR contains a mixture of callee-save and caller-save bits.

References

[1] arch/arm64/include/uapi/asm/sigcontext.h
AArch64 Linux signal ABI definitions

[2] arch/arm64/include/uapi/asm/ptrace.h
AArch64 Linux ptrace ABI definitions

[3] Documentation/arm64/cpu-feature-registers.rst

[4] ARM IHI0055C
http://infocenter.arm.com/help/topic/com.arm.doc.ihi0055c/IHI0055C_beta_aapcs64.pdf
http://infocenter.arm.com/help/topic/com.arm.doc.subset.swdev.abi/index.html Procedure Call Standard for the ARM 64-bit Architecture (AArch64)