Kernel mode NEON

TL;DR summary

- Use only NEON instructions, or VFP instructions that don't rely on support code
- Isolate your NEON code in a separate compilation unit, and compile it with '-march=armv7-a -mfpu=neon -mfloat-abi=softfp'
- Put kernel_neon_begin() and kernel_neon_end() calls around the calls into your NEON code
- Don't sleep in your NEON code, and be aware that it will be executed with preemption disabled

Introduction

It is possible to use NEON instructions (and in some cases, VFP instructions) in code that runs in kernel mode. However, for performance reasons, the NEON/VFP register file is not preserved and restored at every context switch or taken exception like the normal register file is, so some manual intervention is required. Furthermore, special care is required for code that may sleep [i.e., may call schedule()], as NEON or VFP instructions will be executed in a non-preemptible section for reasons outlined below.

Lazy preserve and restore

The NEON/VFP register file is managed using lazy preserve (on UP systems) and lazy restore (on both SMP and UP systems). This means that the register file is kept 'live', and is only preserved and restored when multiple tasks are contending for the NEON/VFP unit (or, in the SMP case, when a task migrates to another core). Lazy restore is implemented by disabling the NEON/VFP unit after every context switch, resulting in a trap when subsequently a NEON/VFP instruction is issued, allowing the kernel to step in and perform the restore if necessary.

Any use of the NEON/VFP unit in kernel mode should not interfere with this, so it is required to do an 'eager' preserve of the NEON/VFP register file, and enable the NEON/VFP unit explicitly so no exceptions are generated on first subsequent use. This is handled by the function kernel_neon_begin(), which should be called before any kernel mode NEON or VFP instructions are issued. Likewise, the NEON/VFP unit should be disabled again after use to make sure user mode will hit the lazy restore trap upon next use. This is handled by the function kernel_neon_end().

Interruptions in kernel mode

For reasons of performance and simplicity, it was decided that there shall be no preserve/restore mechanism for the kernel mode NEON/VFP register contents. This implies that interruptions of a kernel mode NEON section can only be allowed if they are guaranteed not to touch the NEON/VFP registers. For this reason, the following rules and restrictions apply in the kernel: * NEON/VFP code is not allowed in interrupt context; * NEON/VFP code is not allowed to sleep; * NEON/VFP code is executed with preemption disabled.

If latency is a concern, it is possible to put back to back calls to kernel_neon_end() and kernel_neon_begin() in places in your code where none of the NEON registers are live. (Additional calls to kernel_neon_begin() should be reasonably cheap if no context switch occurred in the meantime)

VFP and support code

Earlier versions of VFP (prior to version 3) rely on software support for things like IEEE-754 compliant underflow handling etc. When the VFP unit needs such software assistance, it signals the kernel by raising an undefined instruction exception. The kernel responds by inspecting the VFP control registers and the current instruction and arguments, and emulates the instruction in software.

Such software assistance is currently not implemented for VFP instructions executed in kernel mode. If such a condition is encountered, the kernel will fail and generate an OOPS.

Separating NEON code from ordinary code

The compiler is not aware of the special significance of kernel_neon_begin() and kernel_neon_end(), i.e., that it is only allowed to issue NEON/VFP instructions between calls to these respective functions. Furthermore, GCC may generate NEON instructions of its own at -O3 level if -mfpu=neon is selected, and even if the kernel is currently compiled at -O2, future changes may result in NEON/VFP instructions appearing in unexpected places if no special care is taken.

Therefore, the recommended and only supported way of using NEON/VFP in the kernel is by adhering to the following rules:

- isolate the NEON code in a separate compilation unit and compile it with '-march=armv7-a -mfpu=neon -mfloat-abi=softfp';
- issue the calls to kernel_neon_begin(), kernel_neon_end() as well as the calls into the unit containing the NEON code from a compilation unit which is *not* built with the GCC flag '-mfpu=neon' set.

As the kernel is compiled with '-msoft-float', the above will guarantee that both NEON and VFP instructions will only ever appear in designated compilation units at any optimization level.

NEON assembler

NEON assembler is supported with no additional caveats as long as the rules above are followed.

NEON code generated by GCC

The GCC option -ftree-vectorize (implied by -O3) tries to exploit implicit parallelism, and generates NEON code from ordinary C source code. This is fully supported as long as the rules above are followed.

NEON intrinsics

NEON intrinsics are also supported. However, as code using NEON intrinsics relies on the GCC header <arm_neon.h>, (which #includes <stdint.h>), you should observe the following in addition to the rules above:

- Compile the unit containing the NEON intrinsics with '-ffreestanding' so GCC uses its builtin version of <stdint.h> (this is a C99 header which the kernel does not supply);
- Include <arm_neon.h> last, or at least after <linux/types.h>