QEmu TCG Enhancements for Speeding-up the Emulation of SIMD instructions

Luc Michel, Nicolas Fournel and Frédéric Pétrot
TIMA laboratory,
CNRS/Grenoble INP/UJF, Grenoble, France.
February 11, 2011

Abstract

This contribution presents a strategy to speed-up the simulation of processors having SIMD extensions using dynamic binary translation. The idea is simple: benefit from the SIMD instructions of the host processor that is running the simulation. The realization is unfortunately not easy, as the nature of all but the simplest SIMD instructions is very different from a manufacturer to another. To solve this issue, we propose an approach based on a simple 3-addresses intermediate SIMD instruction set on which and from which mapping most existing instructions at translation time is easy. To still support complex instructions, we rely on the existing helper mechanism of QEmu. We present how this solution can be handled in QEmu’s Tiny Code Generator for the ARM Neon extension running on a MMX/SSE x86, and the gains it provide.

1 Introduction

The purpose of this contribution is to propose a solution applicable to retargetable dynamic binary translation that can make use of the host computer SIMD capabilities. As the nature of the SIMD extensions of the different instruction set architectures (ISA) are quite different, what can seem to be a trivial problem ends up as a fairly complex one, at least when targeting the ability of doing a translation between many extensions.

2 Dynamic Binary Translation of SIMD Instructions

2.1 Specificity and complexity of SIMD instruction sets

SIMD instructions perform parallel operations on multiple data (Single Instruction Multiple Data – SIMD). We can find today multiple ISA extensions providing SIMD instructions to general purpose CPUs. Among them, we specifically looked at MAX instructions for the PA-RISC, MIPS’ MDMX and DSPASE, PPC’s AltiVec, SPARC’s VIS, Intel’s MMX/SSE and ARM NEON. All these instructions set extensions share
the same characteristics, described below. For performing parallel operations on multiple data, SIMD instruction performs the operation (or sequence of instructions) on registers interpreted as array of values. This array of data can have a variable number of values of various size, for example a 128 bits wide register can be viewed as two 64 bits, four 32 bits, eight 16 bits or sixteen 8 bits values. On top of that the variety of the operations applied to the data is huge. It ranges from the classical arithmetic operation (add, sub, shift, ...) to saturated or rounding arithmetic. Among this large range of instructions, each SIMD instruction set represents a unique subset choice made by the designers. On top of these classical instructions, SIMD extensions can even integrate some exotic instructions such as the \texttt{vmul.p8} Neon instruction performing polynomial multiplication which has no equivalent in other SIMD ISA.

We have then to make a careful choice of the micro-operations to add to the intermediate representation of the DBT. The main two constraints we respect for this extension are:

- limit the number of new IR micro-operations in order to limit the burden on the code generator and the overall performances of the binary translator.

- add enough micro-operations in the IR to allow a maximum coverage of the SIMD instruction sets.

Indeed, adding too much micro-operations will tend to add one micro-operation per SIMD instruction. This will not solve the problem since the code generator (the second phase of DBT) will have an heavy work to do to guaranty the emulation of each of the micro-operations. However if we do not add enough micro-operations all SIMD instructions cannot be expressed and the translator will have a huge task to guaranty this translation. We concentrate then on an extension of the IR with a set of instructions which is close to the intersections of the studied SIMD instruction sets. The IR micro-operations will be 3-addresses operations since it is the most general case and allows to represent the 2-addresses versions easily, whereas the reverse is not true.

As opposed to scalar DBT, finding instruction equivalence in SIMD DBT has to take care the SIMD specificities: parallelism and register interpretation.

3 Realization

The proposed IR extension has been ported in the QEMU binary translator. The translator for the ARM Neon instruction set and the code generator for the Intel MMX/SSE instruction set have been implemented. Experiments on synthetic benchmarks show that a gain up to $4 \times$ can be obtained compared to the use of helpers (that are however much simpler to implement).