Software Virtual wide-Vector Machine

Toshihiro KONDA

Background

- **What is Vector Machine**
  - **Way of thinking**

- **Consider generating a vector machine-like code on a scalar machine.**

Ex.) x86-64

- xmm Registers
  - 128bit width
- ymm Registers
  - 256bit width
- zmm Registers
  - 512bit width

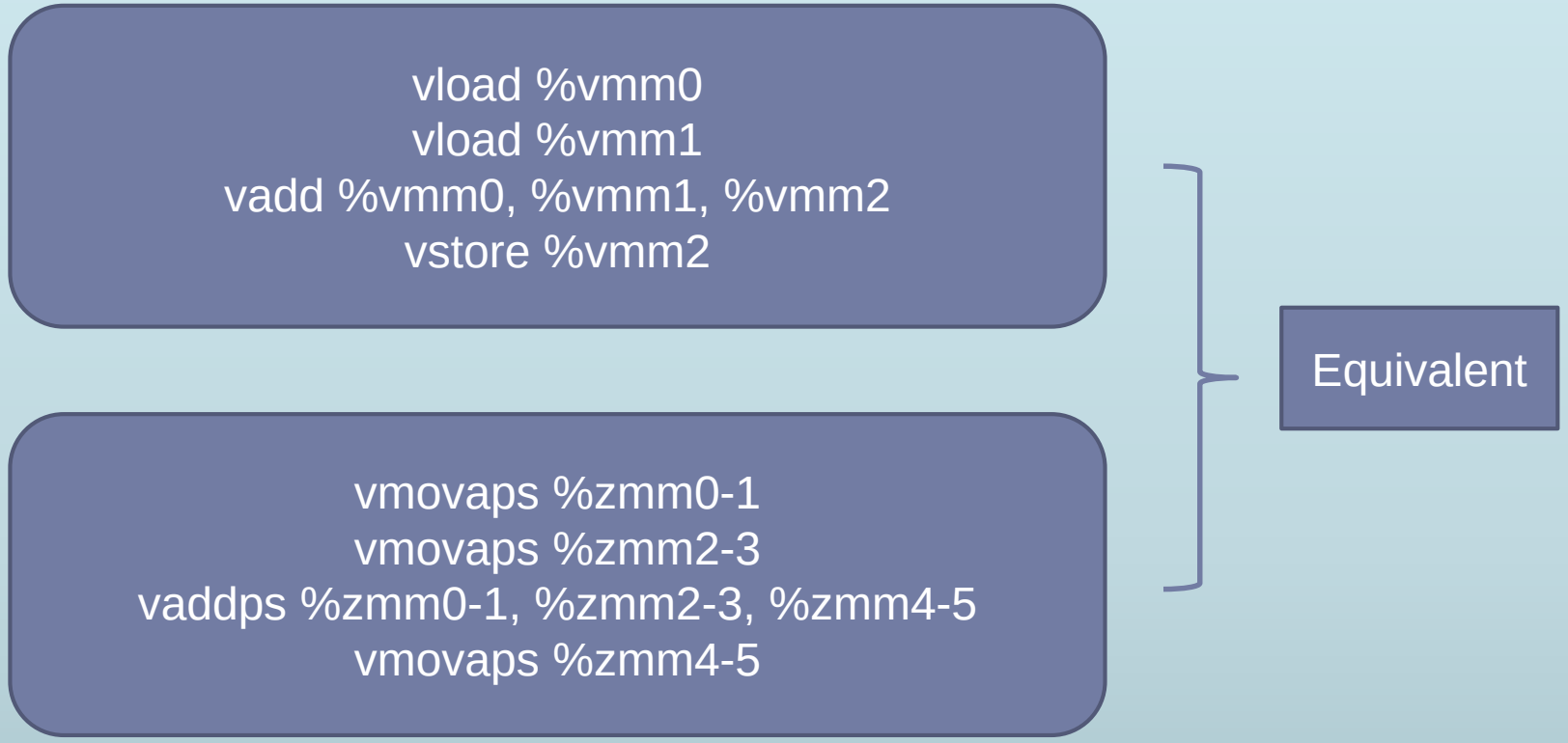
- There are roughly three of these that are currently used。

Introduction

- vmm Registers
  - Introduce a virtual vector register
- Assuming that the width is arbitrary (implementation dependent), it is assumed to be 1024 bits.
- In LLVM
  - v16f64 / v16i64 (64bit 16-wide SIMD)
  - v32f32 / v32i32 (32bit 32-wide SIMD)
  - v64f16 / v64i16 (16bit 64-wide SIMD)
- It shall be equivalent to these.

Single precision case example

- Assume to process v32f32 with AVX512 (v16f32)



Double precision case example

- Assume to process v16f64 with AVX512 (v8f64)

Expected effect

- Improve portability of description of source code in vector processing
  - Even if you describe the vector width by deciding it, you can lower it in each architecture.
- If the compiler (assuming LLVM infrastructure etc.) handles it, optimal code generation is possible from the viewpoint of vectorization.
  - Each process can be vectorized while judging whether or not the register is enough

What's nice? (Part 1)

Two ways of writing like this

```
void func(vmmf32 *vmm0,
                vmmf32 *vmm1,
                vmmf32 *vmm2) {
  *vmm2 = *vmm0 + *vmm1;
}
```

```
vload vmm0
vload vmm1
vadd vmm0, vmm1, vmm2
vstore vmm2
```

are equivalent (here, overwrite the operator in advance).

What's nice? (Part 2)

- Depending on the type, this "vadd" can be determined by the compiler for any of single precision, double precision, half precision arithmetic (etc.).

Thanks.