ソフトウェア仮想ワイドベクトルマシン

今田俊寛

背景

- ベクトルマシンとは
  - 思想

- スカラマシンでベクトルマシンっぽいコードを生成する事を考えてみる。

例)x86-64

- xmmレジスタ
  - 128bit幅
- ymmレジスタ
  - 256bit幅
- zmmレジスタ
  - 512bit幅

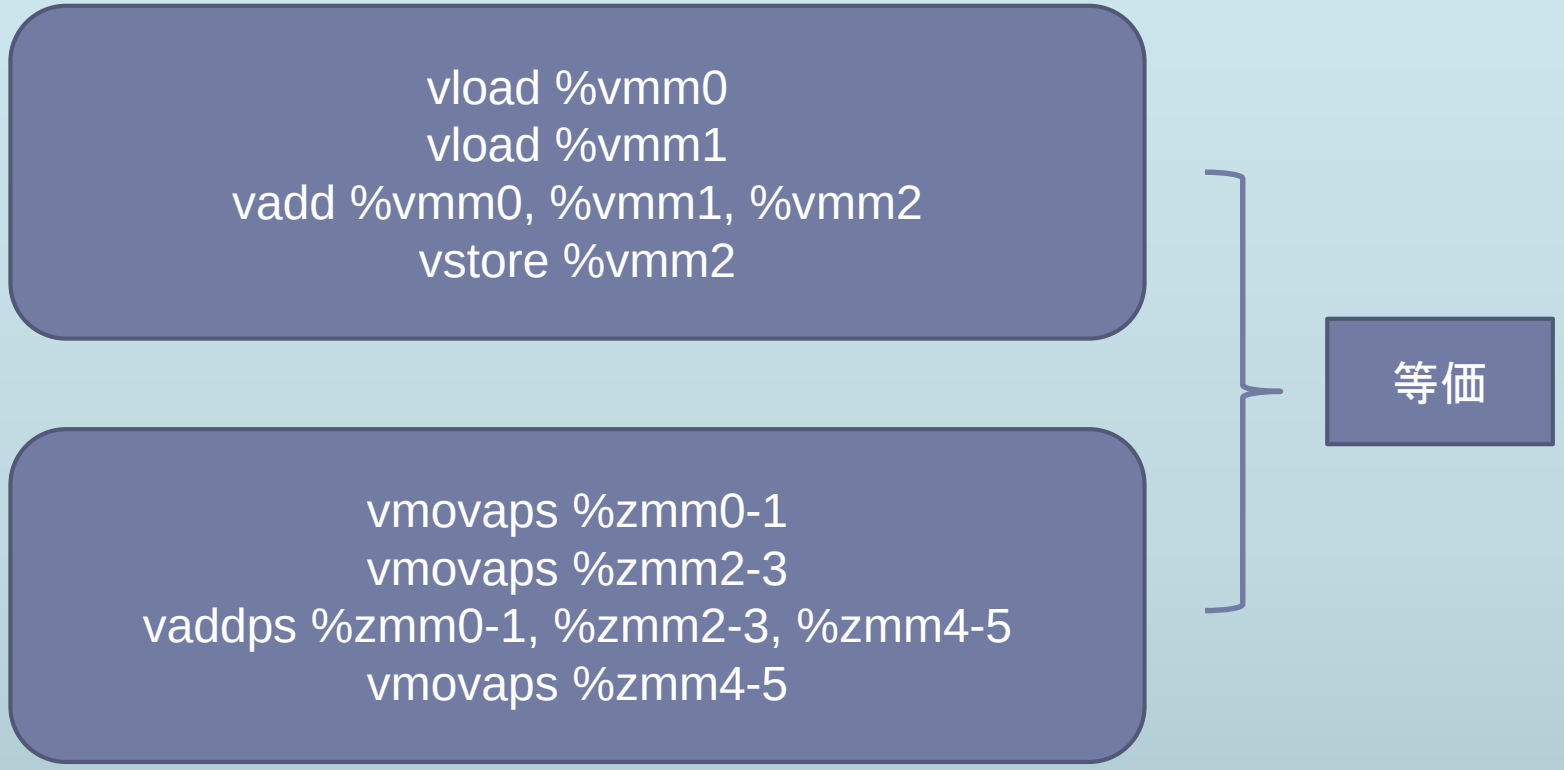
- 現状使われているのは大体この3つ。

導入

- vmmレジスタ
  - 仮想ベクトルレジスタを導入する。
- 幅は任意(実装依存)として、仮に1024bitであるとする。
- LLVMで言うと
  - v16f64 / v16i64 (64bit 16-wide SIMD)
  - v32f32 / v32i32 (32bit 32-wide SIMD)
  - v64f16 / v64i16 (16bit 64-wide SIMD)
- 辺りに相当するものとする。

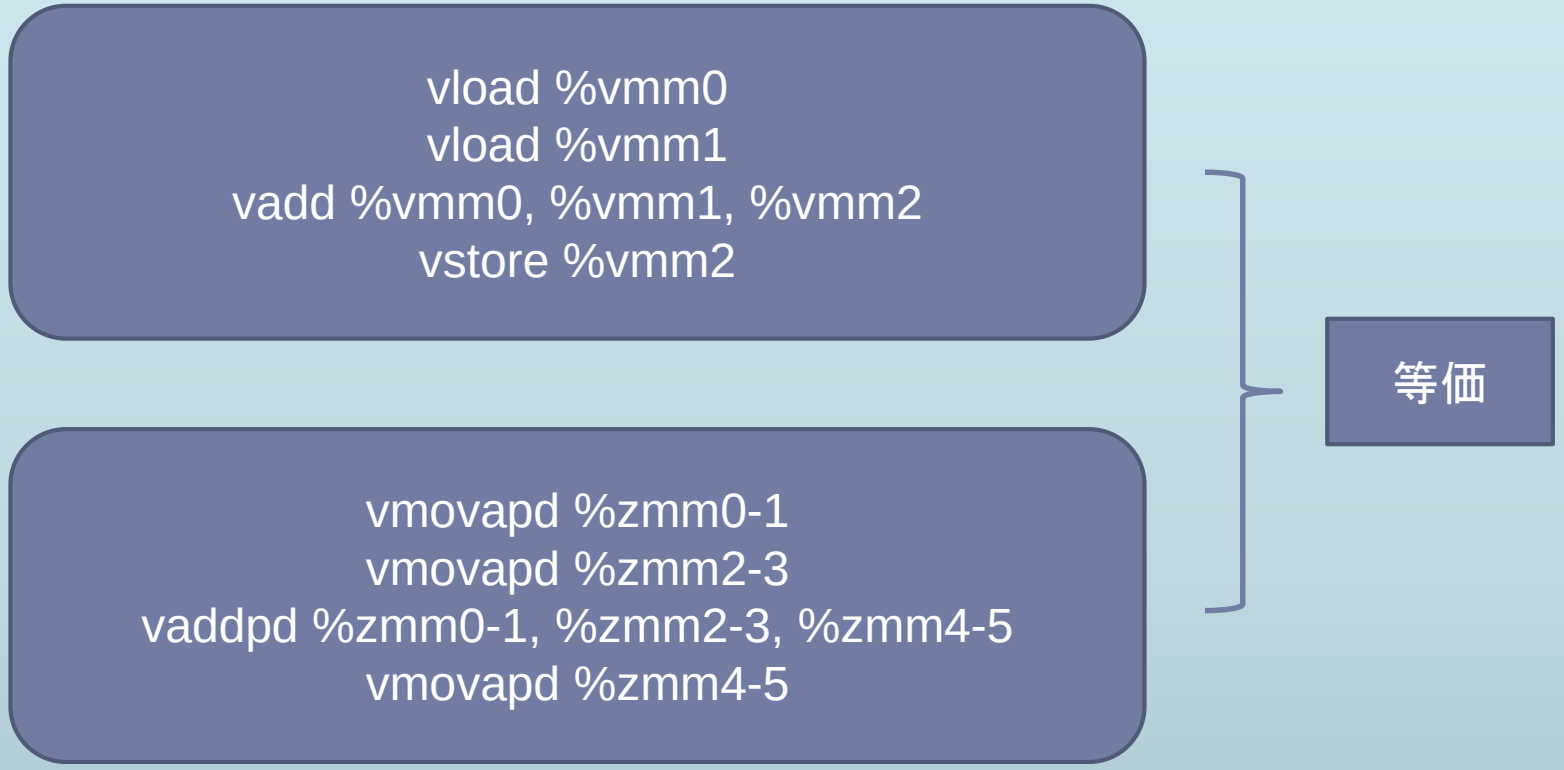
単精度ケースの例

- v32f32 を AVX512 (v16f32)で処理する事を想定



倍精度ケースの例

- v16f64 を AVX512 (v8f64)で処理する事を想定



期待出来る効果

- ベクトル処理でのソースコードの記述のポータビリティ向上
  - ベクトル幅を決め打ちで記述しても、各アーキテクチャでloweringしてやれば良い。
- コンパイラ(LLVMインフラストラクチャ等を想定)がハンドルしてやれば、ベクトル化という観点ではoptimalなコード生成が可能。
  - レジスタが足りるかどうかを判断しつつ、各処理をベクトル化可能

何が嬉しい？(その1)

こんな書き方は

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

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

と等価(ここで、予め演算子をオーバーロードしておく)。

何が嬉しい？(その2)

- 型によって、この“vadd”は単精度・倍精度・半精度演算(etc.)のどれかはコンパイラが判別可能。

終わり