Beyond 128-bit SIMD in WebAssembly

Petr Penzin

Intel Corporation

November 6, 2019

Agenda

Discuss next steps in evolution of Wasm SIMD instruction set.

Proposal summary

- Add vectors operations analogous, at lane level, to existing Wasm SIMD operations, but agnostic of the vector length
- Vector length defined by architecture and set by the runtime

Design Constraints

- Same Wasm binary to run all platforms
- Unambiguous instruction selection
- Backwards compatibility with existing Wasm SIMD instruction set

Alternatives

Longer fixed-width SIMD WebAssembly ISA

- Not universally supported in hardware
- Goes against WebAssembly's design goal of representing the common set of operations between hardware platforms
- Cross platform code generation is challenging

Proposal

We propose length-agnostic variants of operations already present in Wasm *simd128* proposal

- Loads and stores work with consecutive memory locations, like *simd128* loads and stores
- Maximum vector length is set to match the hardware by runtime at startup
- New instructions to control the width of the vector

Types and instructions

New types and instructions

- $vec. < type >$ – separate vector types for different lane types, size defaults to maximum supported by hardware
  - $i8$, $i16$, $i32$, $i64$ – integer
  - $f32$, $f64$ – floating point
- $vec. < type > .length$ – get number of elements in corresponding vector type

Types and instructions

Instructions extending existing operations in WebAssembly SIMD proposal

- $vec. < type > . < op >$ – same lane-wise operation as in *simd128* $< op >$, applied to vector of $vec. < type > .length$

  For example, $vec.f32.mul$ is identical to $f32x4.mul$ on a 4-lane vector, $vec.i32.add$ to $i32x4.add$, and so on

Example

Vector addition, $c = a + b$, sz is the size

```
(block $loop
  (block $loop_top
    (br_if $loop (i32.lt (get_local $sz) (vec.f32.length)))
    vec.f32.load (get_local $a)
    vec.f32.load (get_local $b)
    vec.f32.add
    vec.f32.store (get_local $c)
    ;; Decrement $sz and increment $a, $b, $c
    (br $loop_top)
  )
)
(block $scalar_loop ;; Finish the remaining elements
```

Code generation

- Identical to *simd128* for platforms that support only 128 bit SIMD
- Straight-forward extension to longer vectors on supporting platforms

Comparison against current SIMD proposal

- At 128-bit vector width operations are identical to current Wasm SIMD operations with sole exception of lane shuffle
- Transparent to developer and toolchain

Extension: arbitrary length

Support for size manipulation not multiple of maximum length, to educe WebAssembly and native instruction count.

Additional instructions

- $vec. < type > .set\_length$ – set number of elements in corresponding vector type

  Takes an unsigned argument, allowed use smaller number per runtime's view of the hardware

Example

Vector addition, $c = a + b$, sz is the size

```
local $len i32
(block $loop
  (block $loop_top
    (br_if $loop (i32.eq (get_local $sz) (i32.const 0)))
    (set_local $len (vec.f32.set_length (get_local $sz)))
    vec.f32.load (get_local $a)
    vec.f32.load (get_local $b)
    vec.f32.add
    vec.f32.store (get_local $c)
    ;; Decrement $sz by $len; increment $a, $b, and $c by $len
    (br $loop_top)
  )
)
```

Code generation

Relatively straightforward for vector instruction sets and SIMD predication

- Simple mask generation, code generation changes only needed for loads and stores
- Straight-forward code generation for vector instruction sets

Thank you