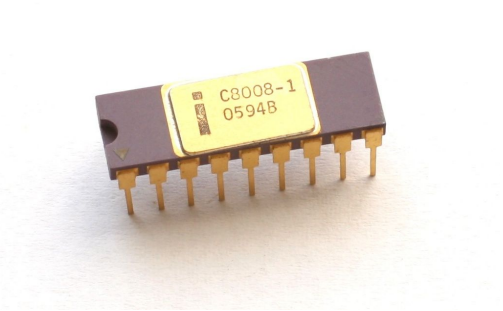
VarVE: Bring SIMD Performance to Variable-width Values

Chen Zou[1, *] Andrew A. Chien[2, 3]

Google LLC[1], University of Chicago[2,*], Argonne National Laboratory[3]

Processor datapath grows wider



Scalar Datapath	8b -> 64b
SIMD	MMX 64b -> AVX 512b

Vector-length-agnostic SIMD: Runtime VLEN

```
void sve_baxpy(int N, uint8_t a, uint8_t* output,
    const uint8_t* xdata, const uint8_t* ydata) {
  for (int i = 0; i < N; i += svcntb()) {
    svbool_t pred = svwhilelt_b8(i, N);
    svuint8_t x = svld1_u8(pred, xdata);
    svuint8_t y = svld1_u8(pred, ydata);
    svuint8_t z = svmul_m_u8(pred, x, a);
    svst1_u8(pred, output, svadd_u8_m(pred, z, y));
    xdata += svcntb();
    ydata += svcntb();
    output += svcntb();
  }
}
```

Vector-length-agnostic SIMD: Runtime VLEN

```
void sve_baxpy(int N, uint8_t a, uint8_t* output,
    const uint8_t* xdata, const uint8_t* ydata) {
  for (int i = 0; i < N; i += svcntb()) {
    svbool_t pred = svwhilelt_b8(i, N);
    svuint8_t x = svld1_u8(pred, xdata);
    svuint8_t y = svld1_u8(pred, ydata);
    svuint8_t z = svmul_m_u8(pred, x, a);
    svst1_u8(pred, output, svadd_u8_m(pred, z, y));
    xdata += svcntb();
    ydata += svcntb();
    output += svcntb();
  }
}
```

The **wider** the VLen discovered at runtime, the **longer** the strip, the **fewer** iterations.

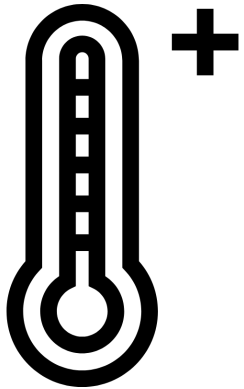
Width of each element in the vector is uniformly b, b ∈ {8, 16, 32, 64}.

However… Values More Diverse than 32/64b

bias

precision

range



Digitalization

Silicon thermometer: 1b-15b

37 = 6’b101001

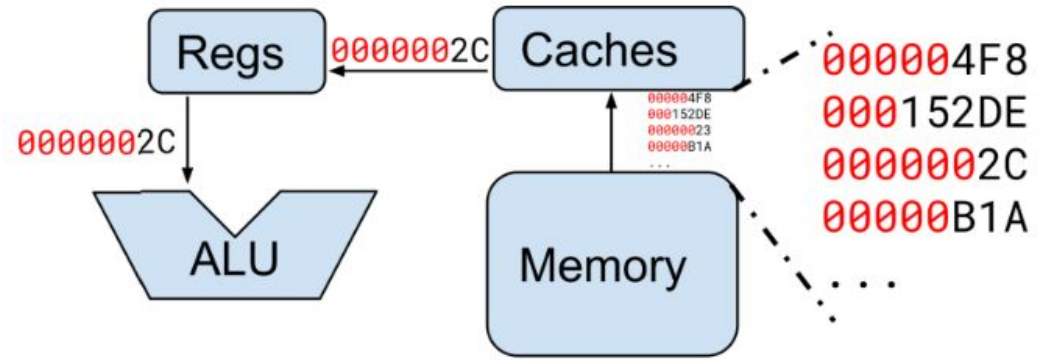
Credit card transactions: 1b-24b

42.56 = 13’b1000010100000 / 100

As a Result… Datapath Efficiency is Low

Wastes on **padding zeros**

in regs, caches, memory …

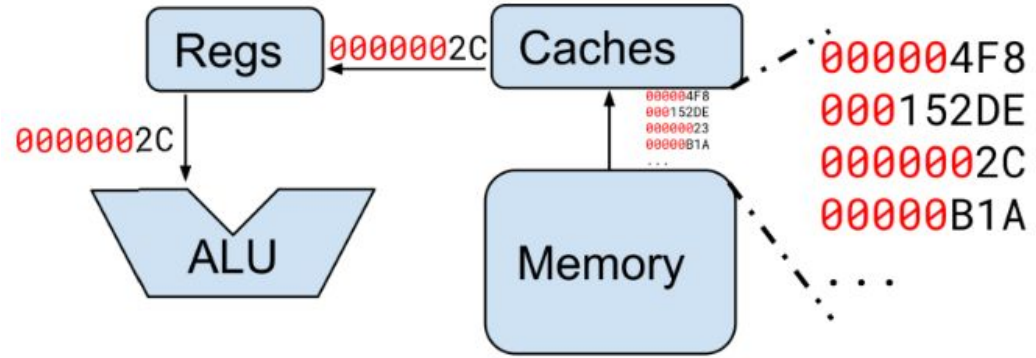


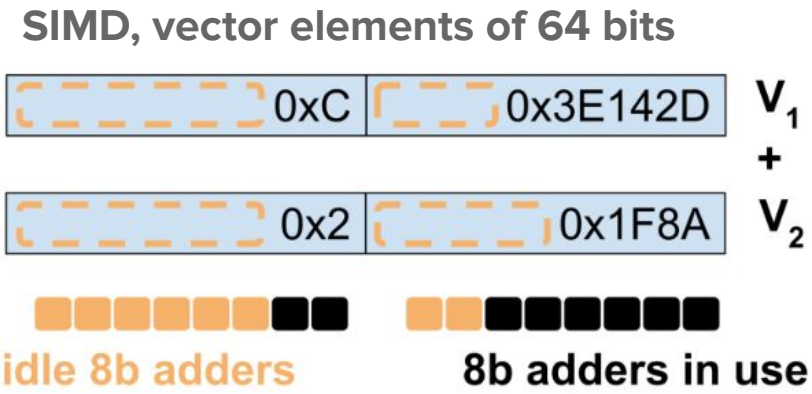
As a Result… Datapath Efficiency is Low

Wastes on **padding zeros**

in regs, caches, memory …

And ALU!





Recent work addresses a portion of issues

Bison-E, pack two vector of narrow values respectively into two scalars

Binary segmentation: Scalar multiplication === Vector IP/LC concatenated

BFloat and the other narrower FP formats make better use of bits.

Limitations:

- Cannot support general compute for diverse values. Cannot provision for outliers.
- Programmers pad to 32b/64b to be safe for outliers. Still poor datapath efficiency.

VarVE: Variable-width Value Vector Extension

1. Efficient packing/unpacking (P/UP) to load/store variable-width values
2. Efficient direct computation on variable-width values

Build on and integrate gracefully with vector-length agnostic SIMDs.

Efficiently Load/Store VarWidth Values: P/UP

// Instruction to declare a stream

```
int FRStream(char* pdata,
int width, int bits)
```

// load a vector from the stream

```
svint<w>t unpack(int sid)
```

`FWStream` and `pack` for store.

`HR/WStream` for Huffman etc.

Our capture of BisonE + UVE

Efficiently Load/Store VarWidth Values: P/UP

// Instruction to declare a stream

```
int FRStream(char* pdata,
int width, int bits)
```

// load a vector from the stream

```
svint<w>t unpack(int sid)
```

`FWStream` and `pack` for store.

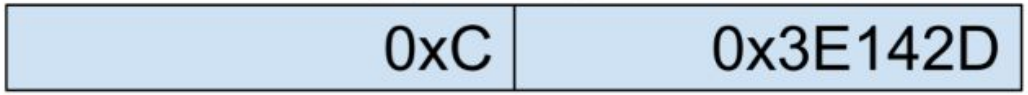
`HR/WStream` for Huffman etc.

Our capture of BisonE + UVE

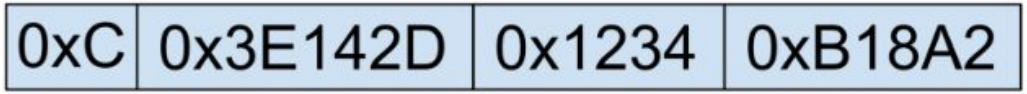
```
void sve_unpack(int w, int bits, const void *data) {
  // read 2B around each value. truncate and align
  for (int i = 0; i < bits / w; i += svcntw()) {
    svbool_t pg = svwhilelt_b32(i, bits);
    svuint32_t vid = svdup_u32(0);
    for (int j = 0; j < w; j++)
      vid = svadd_u32_m(pg, vid, svindex_u32(0, 1));
    svuint32_t vbid = svlsr_n_u32_m(pg, vid, 3);
    svint32_t vx = svld1sh_gather_offset_s32(pg, data, vbid);
    vid = svand_n_u32_m(pg, vid, 7);
    vx = svasr_s32_m(pg, vx, vid);
    vx = svand_n_s32_m(pg, vx, (1 << w) - 1);
    // vx parsed and aligned in 32b-per-element vector
  }
}
void varve_unpack(int w, int bits, const void *data) {
  int sid = frstream(data, w, bits);
  for (int i = 0; i < bits / w; i += svcntb()) {
    svint8_t vx = unpack_s8(sid);
    // vx parsed and aligned for compute
  }
}
```

Efficiently Compute on Variable-width Values

64b Vector Type, e.g. svint64_t



vsint_t, Variable-width Value Vector

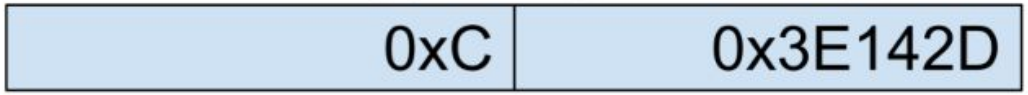


Native VarWidth Vector

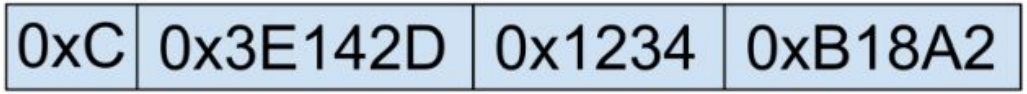
Denser packing of values

Efficiently Compute on Variable-width Values

64b Vector Type, e.g. svint64_t



vsint_t, Variable-width Value Vector



VarVE Vector Metadata: Store Bitwidths

Invisible at ISA-level

Native VarWidth Vector

Denser packing of values

Metadata to mark width of each element

Efficiently Compute on Variable-width Values

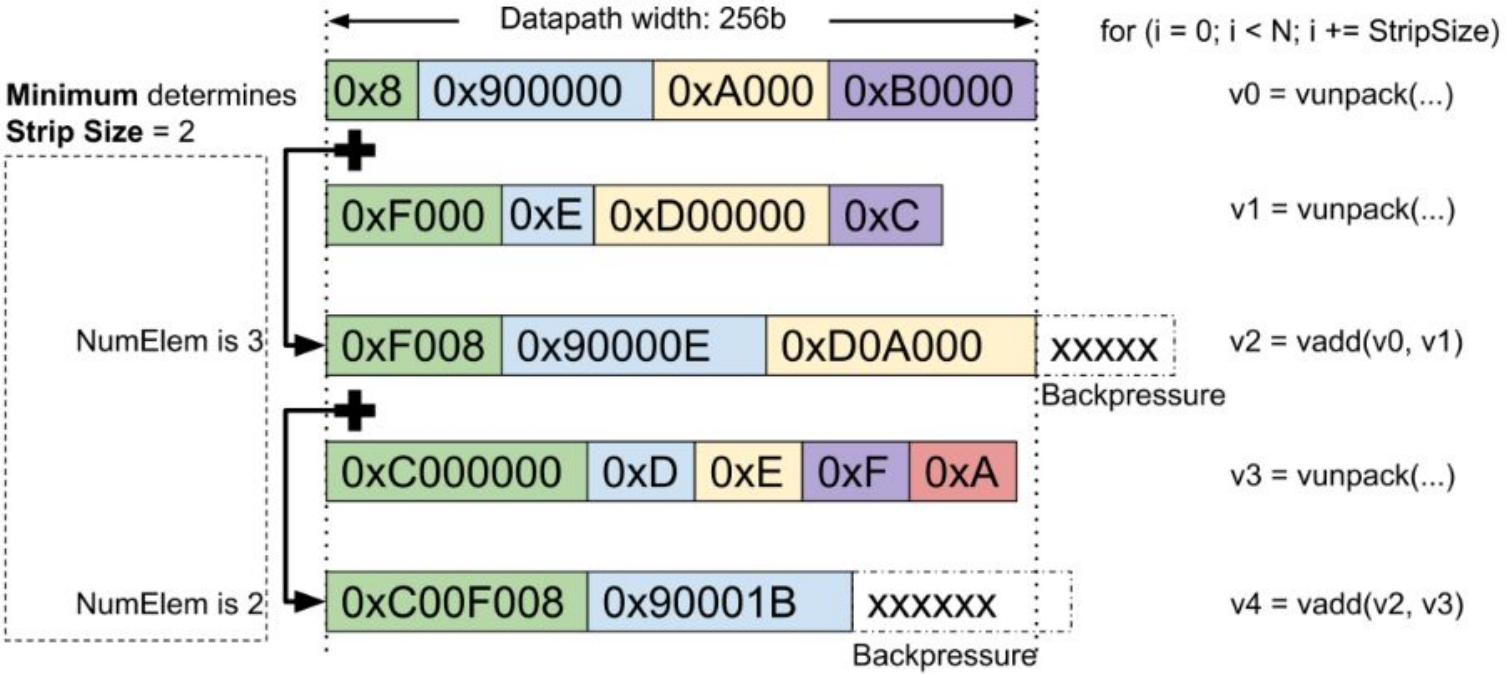
vadd, vsub, vmul, vdiv	**Arithmetic OP** Variants
vmax, vmin, vasl, vasr	Predicated Immediate
vand, vorr, veor, vbic	Predicated Vectors
vcle, vclt, vcne, vceq	Unpredicated Immediate
vcpy, vdup, vindex, vabs	Unpredicated Vectors

Element-wise operations between the native VarWidth Value Vectors

With special semantics: runtime-determined #elements produced.

$$E_{op(u,v)} <= min(E_u, E_v)$$

Dynamic Lock-step Strip Mining



StripSize is dynamically determined, by the values, throughout the iteration

Putting it together (filter from TPC-H Q03)

```
SELECT c_custkey FROM Customer WHERE c_mktsegment == BUILDING
void q03c_varve(int N, void* custkey_i, void* mktsegment_i, void* custkey_o,
                int& elems) {
  int c_custkey_s = frstream(custkey_i, 32, elems << 5);
  int c_mktsegment_s = frstream(mktsegment_i, 8 , elems << 3);
  int c_custkey_t = fwstream(custkey_o, 32);
  for (int i = 0, n = 0; i < N; i += n) {
    vsbool_t pg = vsptrue();
    vsint_t v_c_mktsegment = vunpack(pg, c_mktsegment_s);
    pg = vcmpeq(pg, v_c_mktsegment, BUILDING);
    vsint_t v_c_custkey = vunpack(pg, c_custkey_s);
    elems += vpack(pg, v_c_custkey, c_custkey_t);
    n = rcnume();
    crstream(c_custkey_s, n);
    crstream(c_mktsegment_s, n);
  }
}
```

Denser Vector

Lockstep Strip Mining

Masking optimization (predicate pushdown)

```
SELECT c_custkey FROM Customer WHERE c_mktsegment == BUILDING
void q03c_varve(int N, void* custkey_i, void* mktsegment_i, void* custkey_o,
                int& elems) {
  int c_custkey_s = frstream(custkey_i, 32, elems << 5);
  int c_mktsegment_s = frstream(mktsegment_i, 8 , elems << 3);
  int c_custkey_t = fwstream(custkey_o, 32);
  for (int i = 0, n = 0; i < N; i += n) {
    vsbool_t pg = vsptrue();
    vsint_t v_c_mktsegment = vunpack(pg, c_mktsegment_s);
    pg = vcmpeq(pg, v_c_mktsegment, BUILDING);
    vsint_t v_c_custkey = vunpack(pg, c_custkey_s);
    elems += vpack(pg, v_c_custkey, c_custkey_t);
    n = rcnume();
    crstream(c_custkey_s, n);
    crstream(c_mktsegment_s, n);
  }
}
```

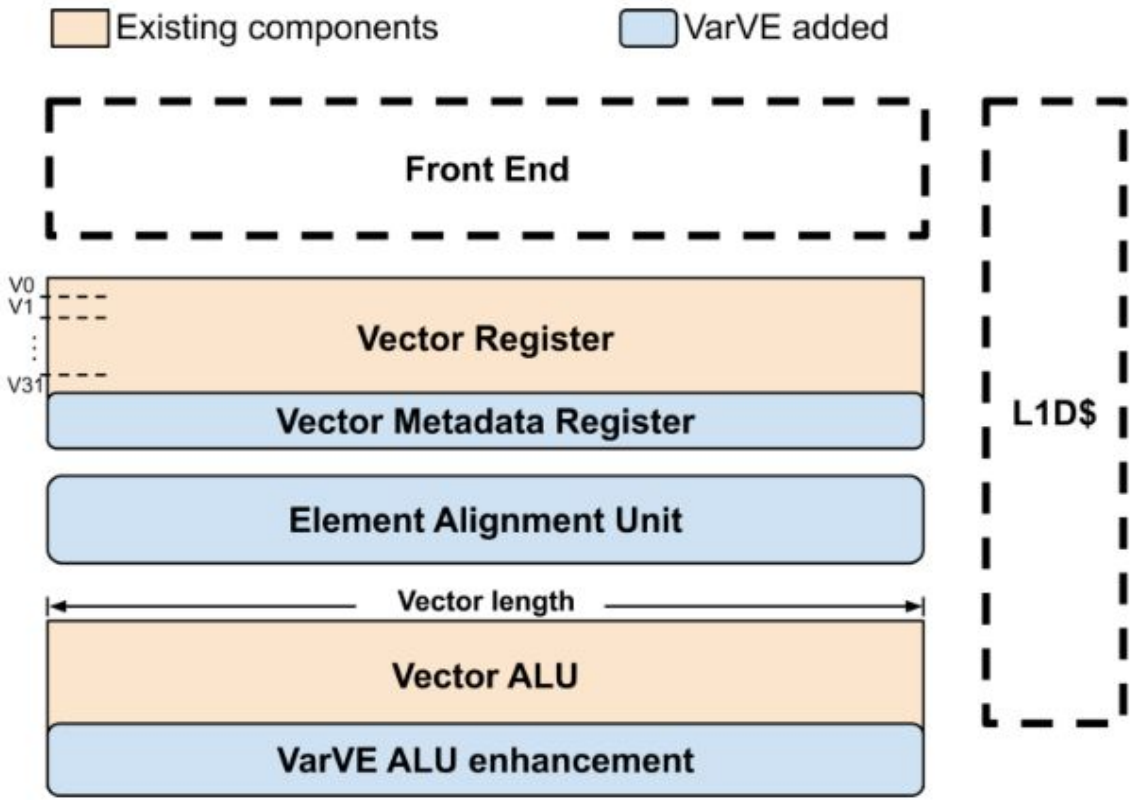
Denser Vector

Masking Optimization

Lockstep Strip Mining

predicate from fast-progressing stream can be push-down to other streams when **vunpack**

VarVE: Reasonable Speed & Cost Implementable



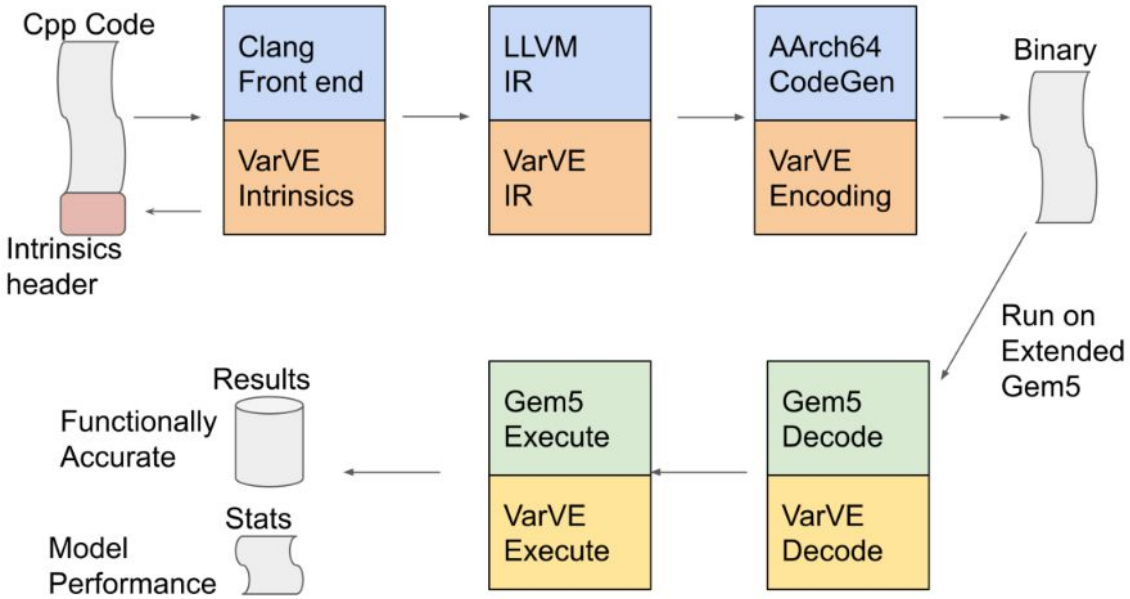
Element Alignment Unit, as one additional pipeline step, aligns corresponding elements

Dynamic VarVE ALU is dynamically partitionable, computing aligned elements

Tune in the paper for details.

VarVE is flexible in supporting hardware tradeoffs.

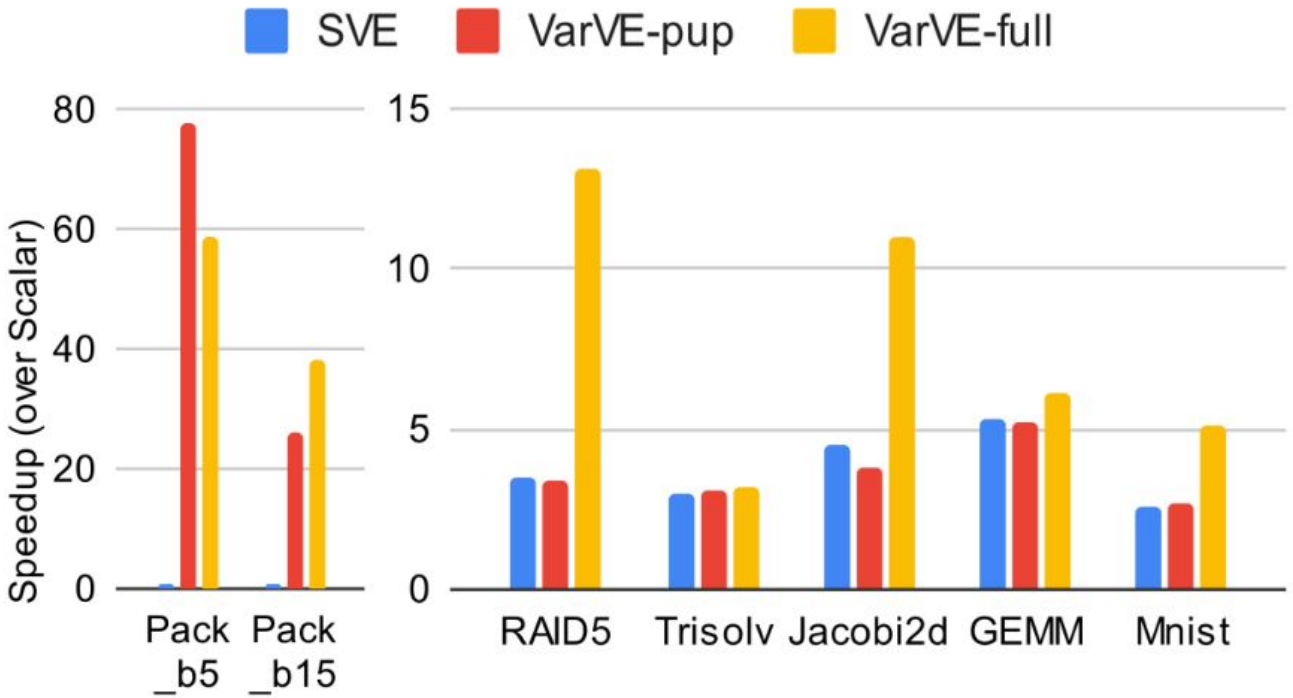
Evaluation Methodology



Extended Clang-LLVM with VarVE intrinsics on top of SVE.

Extended Gem5 for VarVE instructions as well as performance modeling the hardware (incl. restrictions)

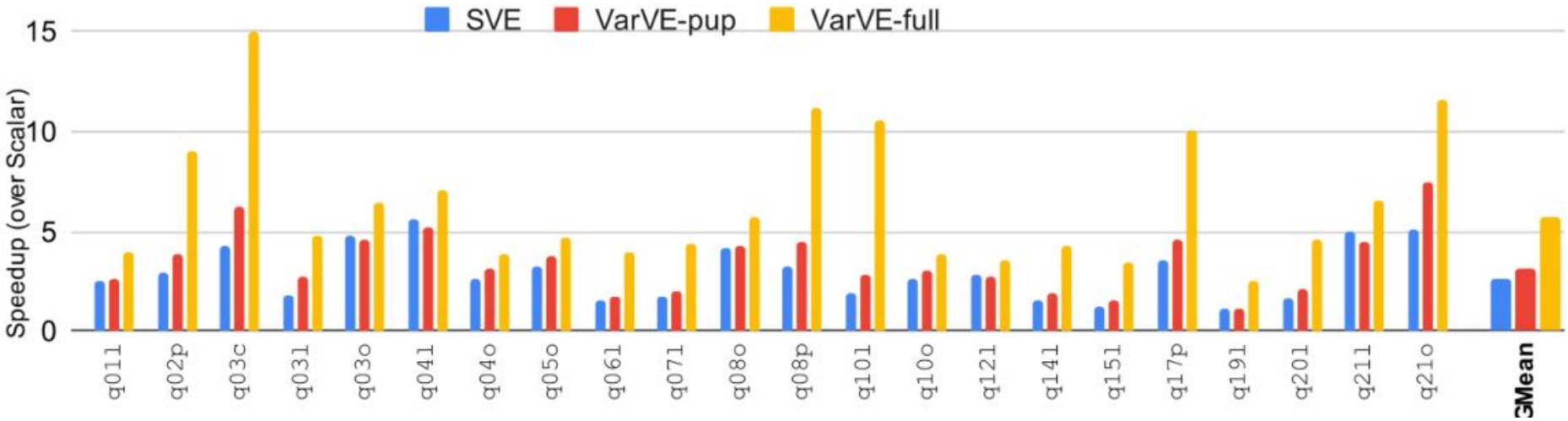
Kernel Evaluation Results



P/UP targeted workloads

Pure-compute workloads, VarVE wins (1.9x over SVE) via improving datapath efficiency

TPC-H Vectorized Filtering



Observe similar speedups, 2.1x over SVE
Masking optimization was a great contributor.

Conclusions

Sophisticated pack/unpack is not sufficient to capture the general benefits of supporting variable width values, because ALU datapath is left untouched.

VarVE provides native compute on variable-width values delivering full and general performance benefits.

Results show VarVE outperforms SVE by 1.9-2.1x on variable width data

Backup

Q&A

Masking optimization: can it be applied to SVE?

Can not apply to SVE. SVE requires fixed and full alignment of elements when loading the data into vector registers.

It cannot take advantage of the mask and load more values into the register.

Otherwise, misalignment && break ISA definition:

no longer performing element-wise operations.

Q&A

Can VarVE applies in RVV?

All features, including PUP, VarWidth registers, Dynamic Strip mining, mask optimization can apply to RVV as well.

VarVE was built unpon SVE because of the maturity of the toolchain at the time, particularly Gem5 has ARM backend with maintenance from ARM engineers.

But I have heard recent support of Gem5 on RVV 1.0.

Q&A

Extension to floating point computations.

It is somewhat a natural extension to support a dynamic E_xM_Y scheme, where x and y can be dynamically set for each value as required to preserve accuracy. But definitely complex in circuit implementation.