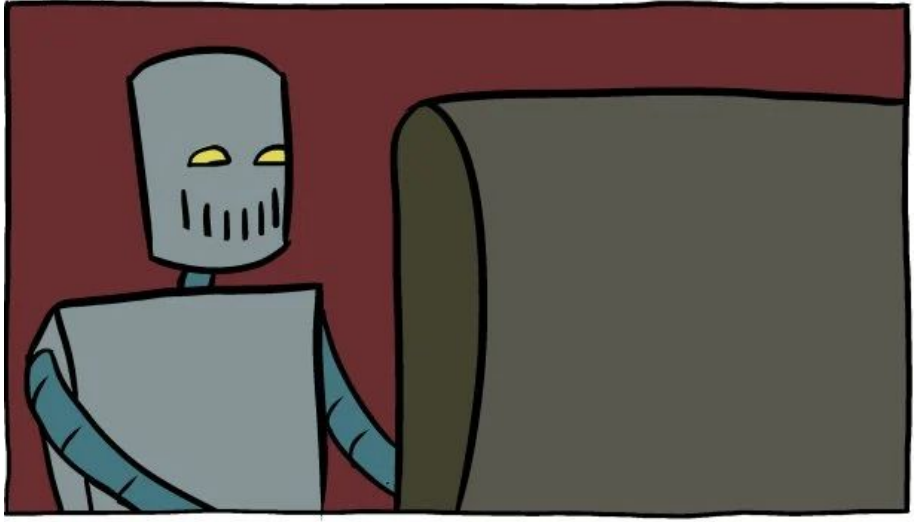
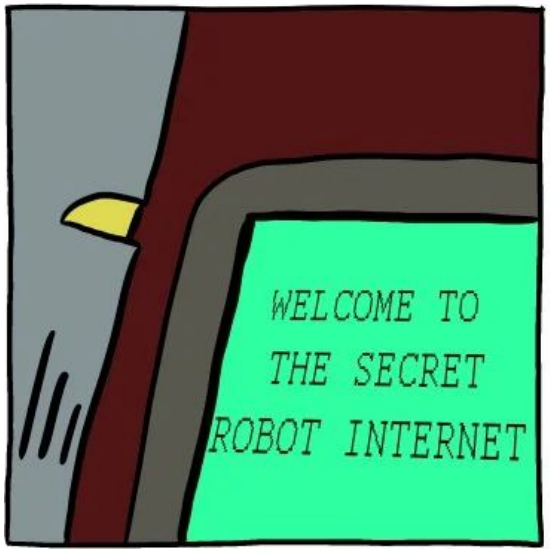
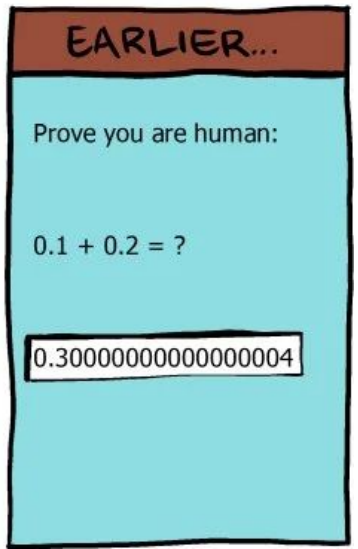
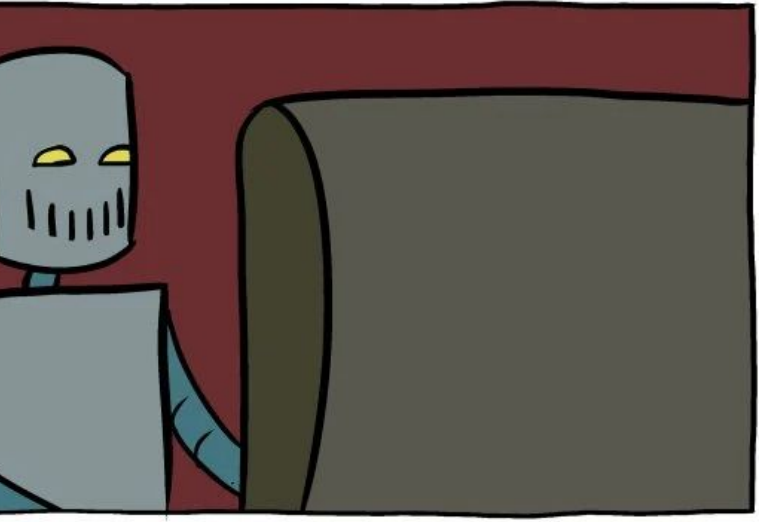
Compiler Runtime Optimization:

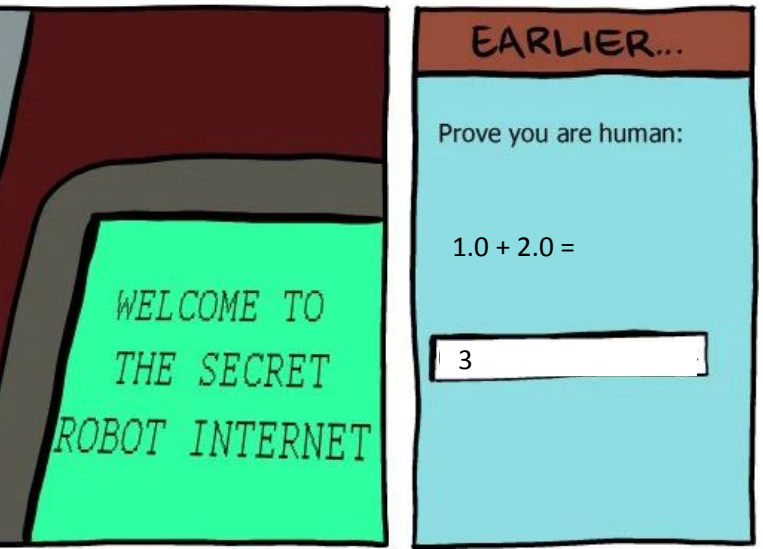
Fast `pivot` Function for MLIR's Presburger Library Through **Vectorization** and **Integer Arithmetic in FPU**



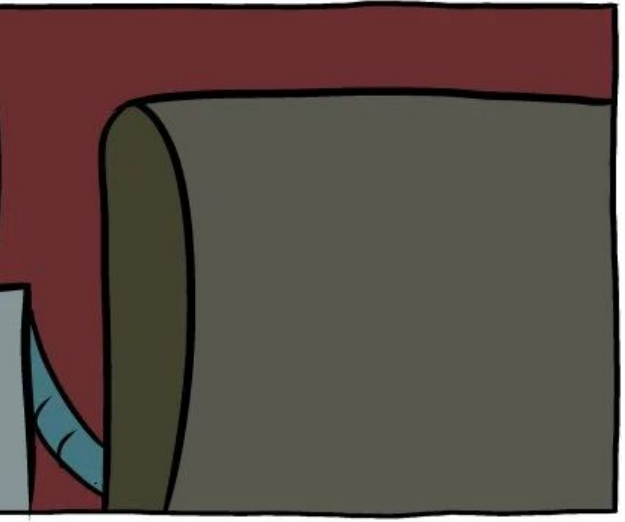

WELCOME TO
THE SECRET
ROBOT INTERNET

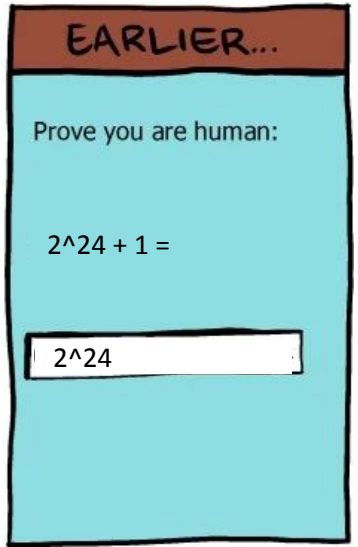

EARLIER...
Prove you are human:
0.1 + 0.2 = ?
0.30000000000000004





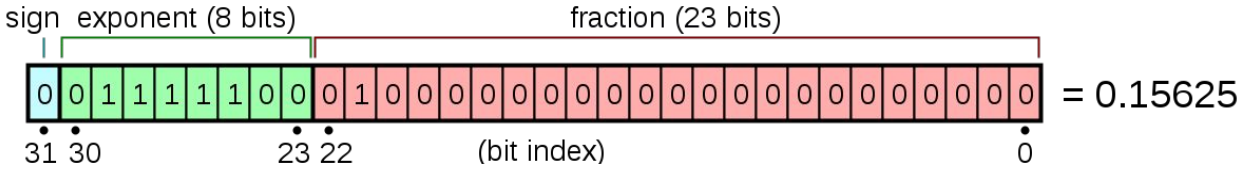

```
0.1 + 0.2 = 0.30000000000000004 ❌

1.0 + 2.0 = 3.0 ✅
11.0 + 22.0 = 33.0 ✅
111.0 + 222.0 = 333.0 ✅
```





"int24_t"

Sign - **1** bit
Exponent - 8 bits => **int24_t**
Mantissa - **23** bits



(2^24 - 1) + 1 = 2^24 ✅
2^24 + 1 = 2^24 **Overflow!** ❌

Floating Point Status register

x86 Overflow-aware Multiply and Add

Floating point could be faster than integer when vectorized and overflow-checked

<table>
<tr><th></th><th>Integer</th><th>Floating points</th></tr>
<tr><td>Scalar</td><td>Clang's provides extension:
<code>bool __builtin_add_overflow (type1 x, type2 y, type3 *sum)</code>

Compiles to SETO - Set if Overflow</td><td rowspan="2">Library function feclearexcept and fetestexcept:
to read and reset floating point overflow and imprecision status register

Status register is accumulative, unless feclearexcept is called.

Single-time overhead, approx 1 ns.</td></tr>
<tr><td>Vectorized</td><td>No micro-architecture support,
require additional arithmetic:

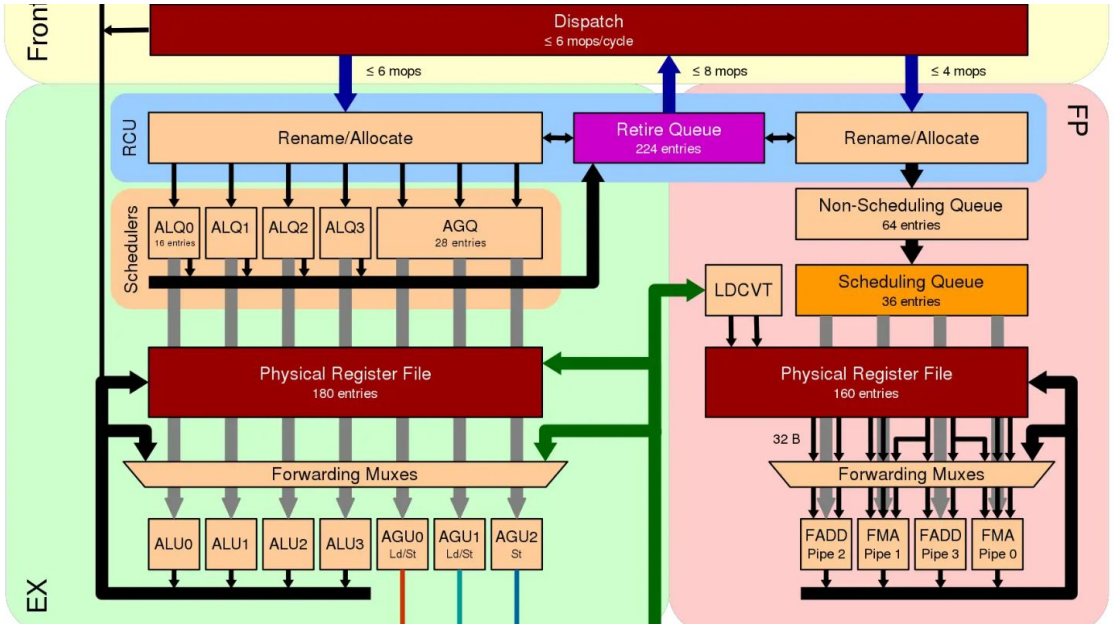
<pre>vpaddw   %zmm4,%zmm2,%zmm3 # Add
vpaddsw   %zmm2,%zmm4,%zmm2 # Add with saturation
vpcmpneqw %zmm3,%zmm2,%k1   # compare
kord   %k1,%k0,%k0   # XOR

vpmullw   %zmm1,%zmm3,%zmm2 # Multiply
vpmulhw   %zmm1,%zmm3,%zmm3 # Multiply high bits
vpsraw $0xf,%zmm2,%zmm5 # shift
vpcmpneqw %zmm3,%zmm5,%k1   # compare
kord   %k0,%k1,%k0   # XOR</pre></td></tr>
</table>

llvm-mca and Zen 3

```
float * src1_ptr, src2_ptr, src3_ptr, dst_ptr;
for (uint32_t i = 0; i < size; i += 1 ){
    dst_ptr[i] = src1_ptr[i] * src2_ptr[i] + src3_ptr[i];
}
```

```
$ llvm-mca-15 -mcpu=znver3 x.s
Iterations:        100
Instructions:      600
Total Cycles:      413
Total uOps:        600

Dispatch Width:    6
uOps Per Cycle:    1.45
IPC:               1.45
Block RThroughput: 3.0


Instruction Info:
[1]: #uOps
[2]: Latency
[3]: RThroughput
[4]: MayLoad
[5]: MayStore
[6]: HasSideEffects (U)
```

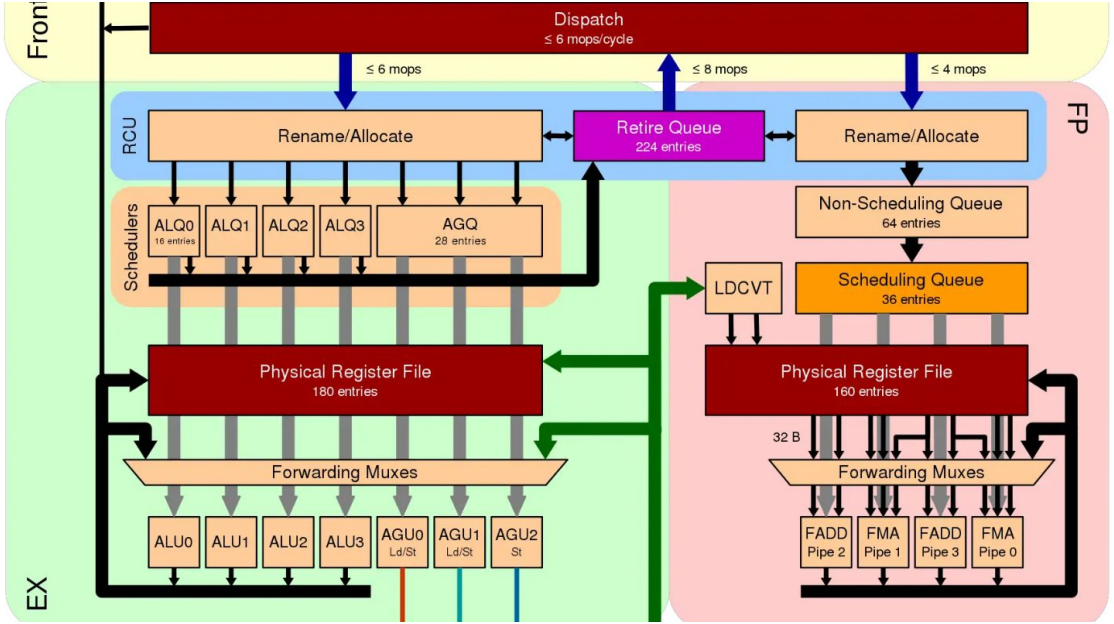



```
[1]    [2]    [3]    [4]    [5]    [6]    Instructions:
 1      8     0.50    *                    vmovups  32(%rcx,%rdi,4), %ymm1
 1      8     0.50    *                    vmovups  32(%rbx,%rdi,4), %ymm3
 1      11    0.50    *                    vfmadd213ps     (%rdx,%rdi,4), %ymm0, %ymm2
 1      11    0.50    *                    vfmadd213ps     32(%rdx,%rdi,4), %ymm1, %ymm3
 1      1     1.00           *             vmovups  %ymm2, (%rdx,%rdi,4)
 1      1     1.00           *             vmovups  %ymm3, 32(%rdx,%rdi,4)
```

llvm-mca and Zen 3

```
int * src1_ptr, src2_ptr, src3_ptr, dst_ptr;
for (uint32_t i = 0; i < size; i += 1 ){
    dst_ptr[i] = src1_ptr[i] * src2_ptr[i] + src3_ptr[i];
}
```

```
$ llvm-mca-15 -mcpu=znver3 x.s
Iterations:        100
Instructions:      800
Total Cycles:      409
Total uOps:        800

Dispatch Width:    6
uOps Per Cycle:    1.96
IPC:               1.96
Block RThroughput: 4.0


Instruction Info:
[1]: #uOps
[2]: Latency
[3]: RThroughput
[4]: MayLoad
[5]: MayStore
[6]: HasSideEffects (U)

[1]    [2]    [3]    [4]    [5]    [6]    Instructions:
 1      8     0.50    *                    vmovdqu  -96(%rbx,%rdx), %ymm0
 1      8     0.50    *                    vmovdqu  -64(%rbx,%rdx), %ymm1
 1      10    0.50    *                    vpmulld  -96(%rdi,%rdx), %ymm0, %ymm0
 1      10    0.50    *                    vpmulld  -64(%rdi,%rdx), %ymm1, %ymm1
 1      8     0.50    *                    vpaddd   -96(%rcx,%rdx), %ymm0, %ymm0
 1      8     0.50    *                    vpaddd   -64(%rcx,%rdx), %ymm1, %ymm1
 1      1     1.00           *             vmovdqu  %ymm0, -96(%rcx,%rdx)
```



The `pivot` function

- Linear programming solver in MLIR's Presburger Library using simplex method: Maximize or Minimize a objective function, subject to constraints

- Example:
  Maximize $Z = 40x_1 + 30x_2$
  Subject to constraints:
  $x_1 \geq 0,\ x_2 \geq 0,$
  $x_1 + 2x_2 \leq 16,$
  $x_1 + x_2 \leq 9,$

x_1	x_2	s_1	s_2	P	LHS
1	2	1	0	0	16
1	2	0	1	0	9
-40	-30	0	0	1	0

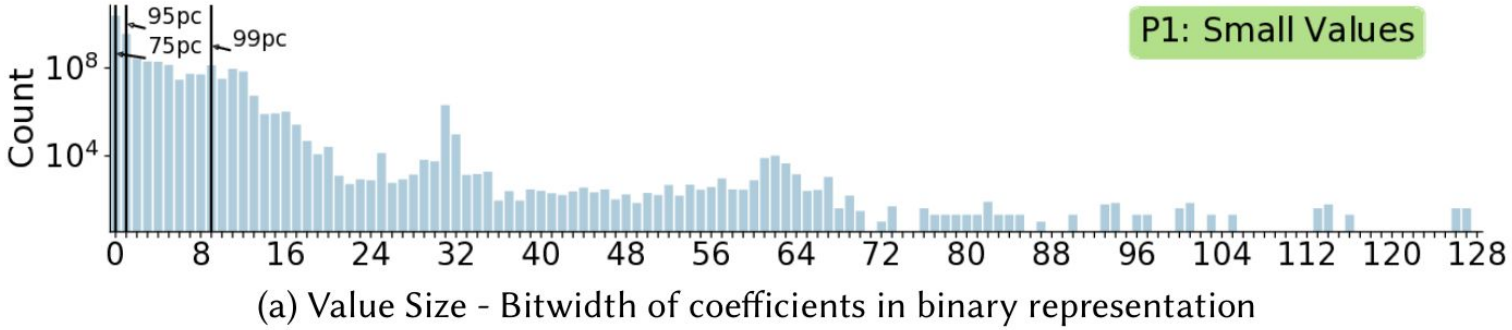
- Hot loop: multiply and add each row with the pivot row and some constant **overflow-checked** element-wise **2x mul** and **1x add**

Previous work

FPL: Fast Presburger Arithmetic through Transprecision

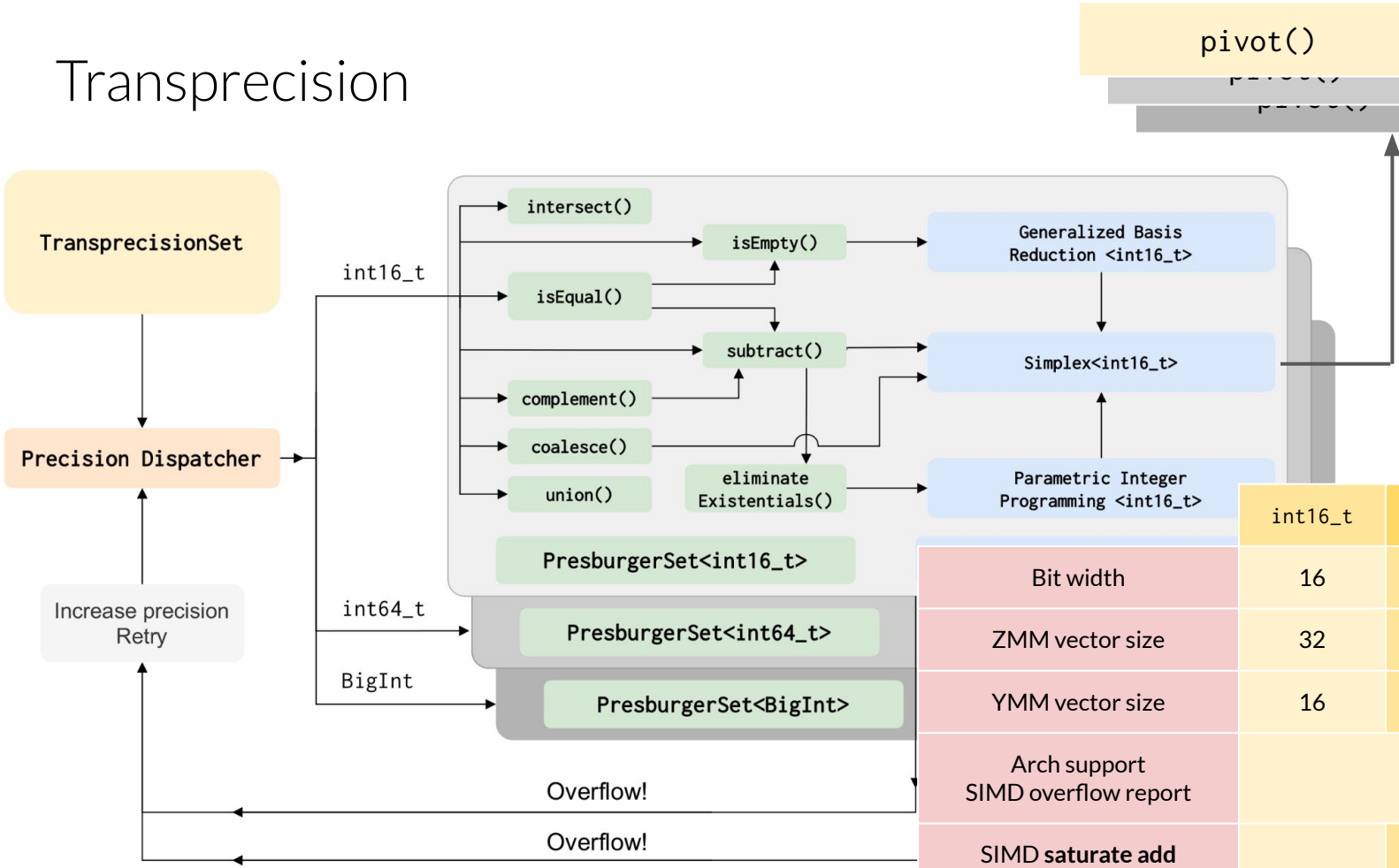
- Main performance bottleneck

- Most elements in the matrix have small value
  > 99% of the observed numbers fits inside 16-bit integers



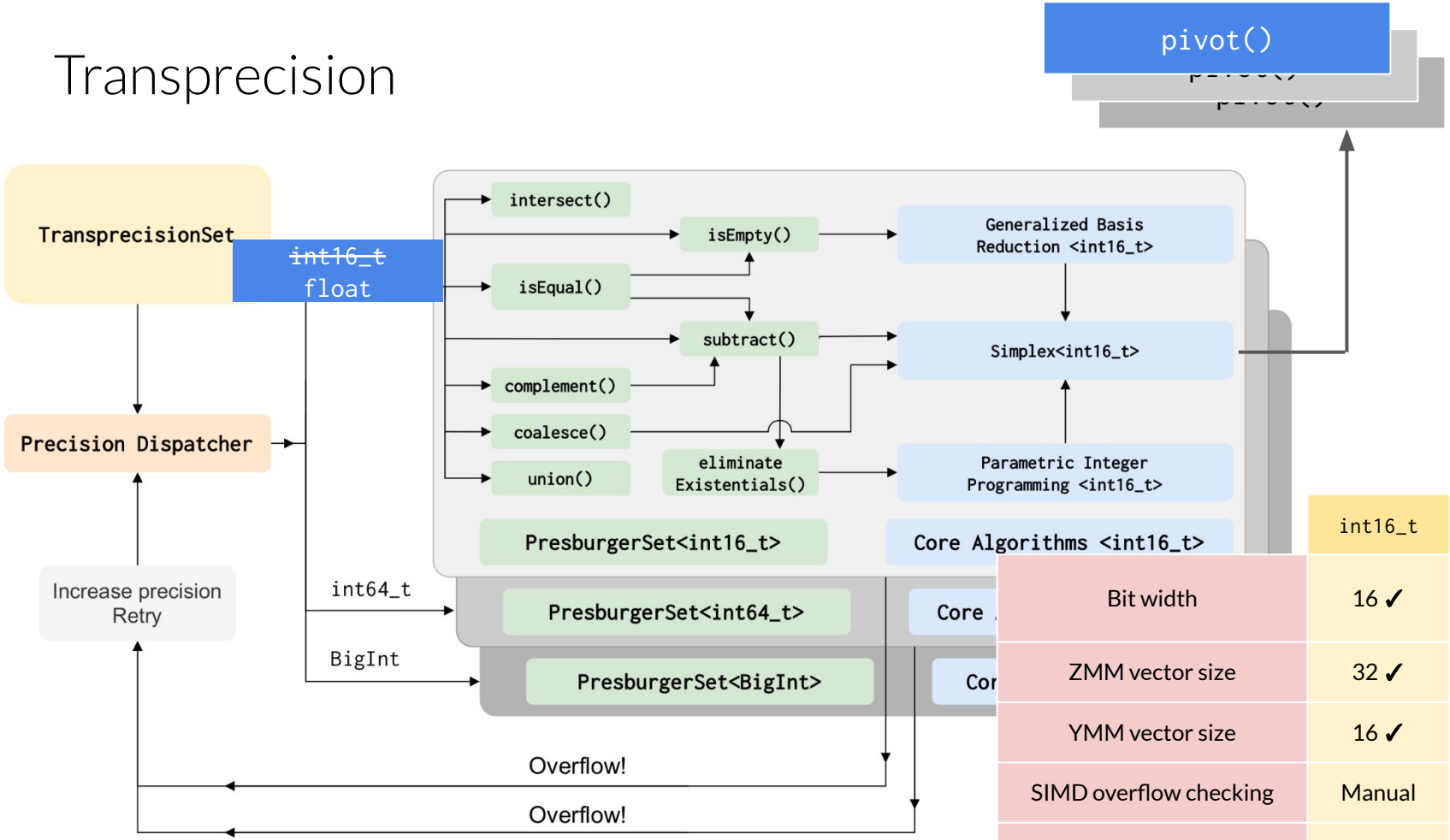
(a) Value Size - Bitwidth of coefficients in binary representation

Transprecision



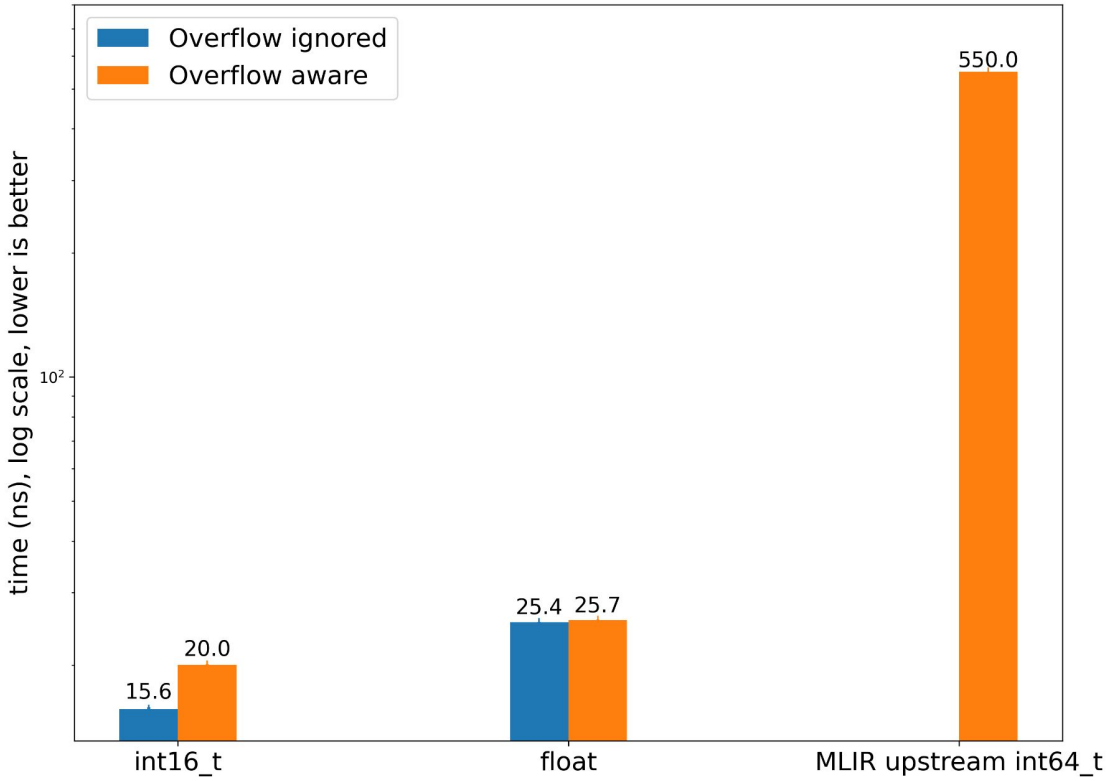
	int16_t	int32_t	int64_t
Bit width	16	32	64
ZMM vector size	32	16	8
YMM vector size	16	8	4
Arch support SIMD overflow report	✗		
SIMD **saturate add** and **mul high** to compute overflow	✓	✗ Only mul high	✗

Transprecision



	int16_t	float
Bit width	16 ✓	32, 24 effective
ZMM vector size	32 ✓	8
YMM vector size	16 ✓	4
SIMD overflow checking	Manual	Automatic ✓
Compatibility	AVX-512	AVX-2 ✓

Benchmark



- Zen 4 @ 4.5 GHz
- 30-row by 19-column matrix
- FADD units are idle
- Zen 4 is not good at register load and store

The End

Discovery: llvm-mca 🤔

```
$ llvm-mca-15 -mcpu=znver2
/dev/shm/llvm/llvm-project/llvm/test/tools/llvm-mca/X86/Znver2/resources-fma.s
Iterations:        100
Instructions:      19200
Total Cycles:      115203
Total uOps:        19200

Dispatch Width:    4
uOps Per Cycle:    0.17
IPC:               0.17
Block RThroughput: 96.0


Instruction Info:
[1]: #uOps
[2]: Latency
[3]: RThroughput
[4]: MayLoad
[5]: MayStore
[6]: HasSideEffects (U)

[1]    [2]    [3]    [4]    [5]    [6]    Instructions:
 1      5     0.50                        vfmadd132pd   %xmm0, %xmm1, %xmm2
 1      12    0.50    *                   vfmadd132pd   (%rax), %xmm1, %xmm2
```

```
$ llvm-mca-15 -mcpu=znver3
/dev/shm/llvm/llvm-project/llvm/test/tools/llvm-mca/X86/Znver3/
Iterations:        100
Instructions:      19200
Total Cycles:      105603
Total uOps:        19200

Dispatch Width:    6
uOps Per Cycle:    0.18
IPC:               0.18
Block RThroughput: 192.0


Instruction Info:
[1]: #uOps
[2]: Latency
[3]: RThroughput
[4]: MayLoad
[5]: MayStore
[6]: HasSideEffects (U)

[1]    [2]    [3]    [4]    [5]    [6]    Instructions:
 1      4     1.00                        vfmadd132pd   %xmm0,
 1      11    1.00    *                   vfmadd132pd   (%rax),
```

https://github.com/llvm/llvm-project/issues/59325