High-Precision Anchored Accumulators for Reproducible Floating-Point Summation

ARM

David Lutz, Neil Burgess, Chris Hinds

ARITH24
July 25, 2017

Overview

- non-associativity of FP addition is a problem
- the range of FP values in a given sum can be safely reduced
- HPA - as much range as needed, 100-or-more-bit integers with a fixed *anchor*
- redundant HPA for carry-free high performance summation
- HPA on SIMD units - two approaches

Non-associativity is a problem

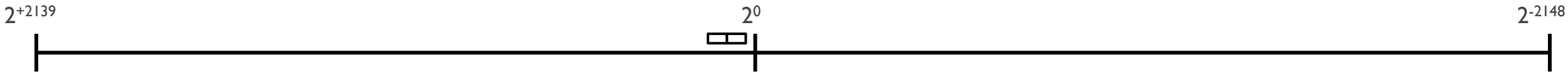
Floating-Point (FP) addition is not associative

$$
\begin{aligned}
(2^{45} + -2^{45}) + 2^{20} &= 2^{20} \\
2^{45} + (-2^{45} + 2^{20}) &= 0
\end{aligned}
$$

- methods for reproducible summation cost performance and/or complexity
- higher precision arithmetic costs power and performance
- parallelization vs reproducibility (& productivity, portability, ...)

Kulisch accumulators

- treat FP numbers and products as very wide fixed-point numbers
- use a 4288-bit accumulator!
  - 4092 possible locations for first significand bit
  - 105 fraction bits
  - extra bits so as to avoid overflow
  - 67 64-bit words
- these additions are **associative**, and the 4288-bit result is **exact**



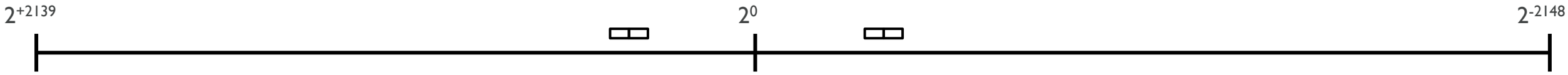
The range of FP values in a given sum can be safely reduced.

Distances in meters

- diameter of observable universe: 8.8×10^{26} m (2^{89})
- distance to nearest galaxy: 2.401×10^{22} m (2^{74})
- distance Austin to London: 7.905×10^{6} m (2^{22})
- diameter of a hydrogen atom: 1.06×10^{-10} m (2^{-34})
- diameter of a proton: 10^{-15} m (2^{-50})

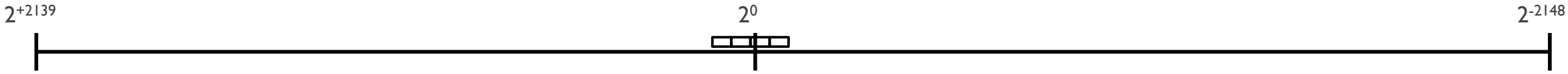
Reducing exponent range

- most problems do not require the full exponent range
  - galaxies or subatomics?
- small values may well be unimportant
- programmers can know & benefit from knowing these ranges



What is a typical range?

- "100 bits suffices for many HPC applications" (D. Bailey, 2013 ARITH keynote)
- "most problems fit in the range 10^{-25} to 10^{30}, a span of about 183 bits" (LANL)
- "128-bit integers are probably sufficient for most uses." (LANL again, SC15)
- "... in most cases we're around the 10^{-15} tolerance [2^{-50}] because of machine epilson, compiler rounding/optimization etc with results in a tighter range with lower exponents. (Sandia)



High-Precision Anchored (HPA) numbers
- as much range as needed
for a given problem

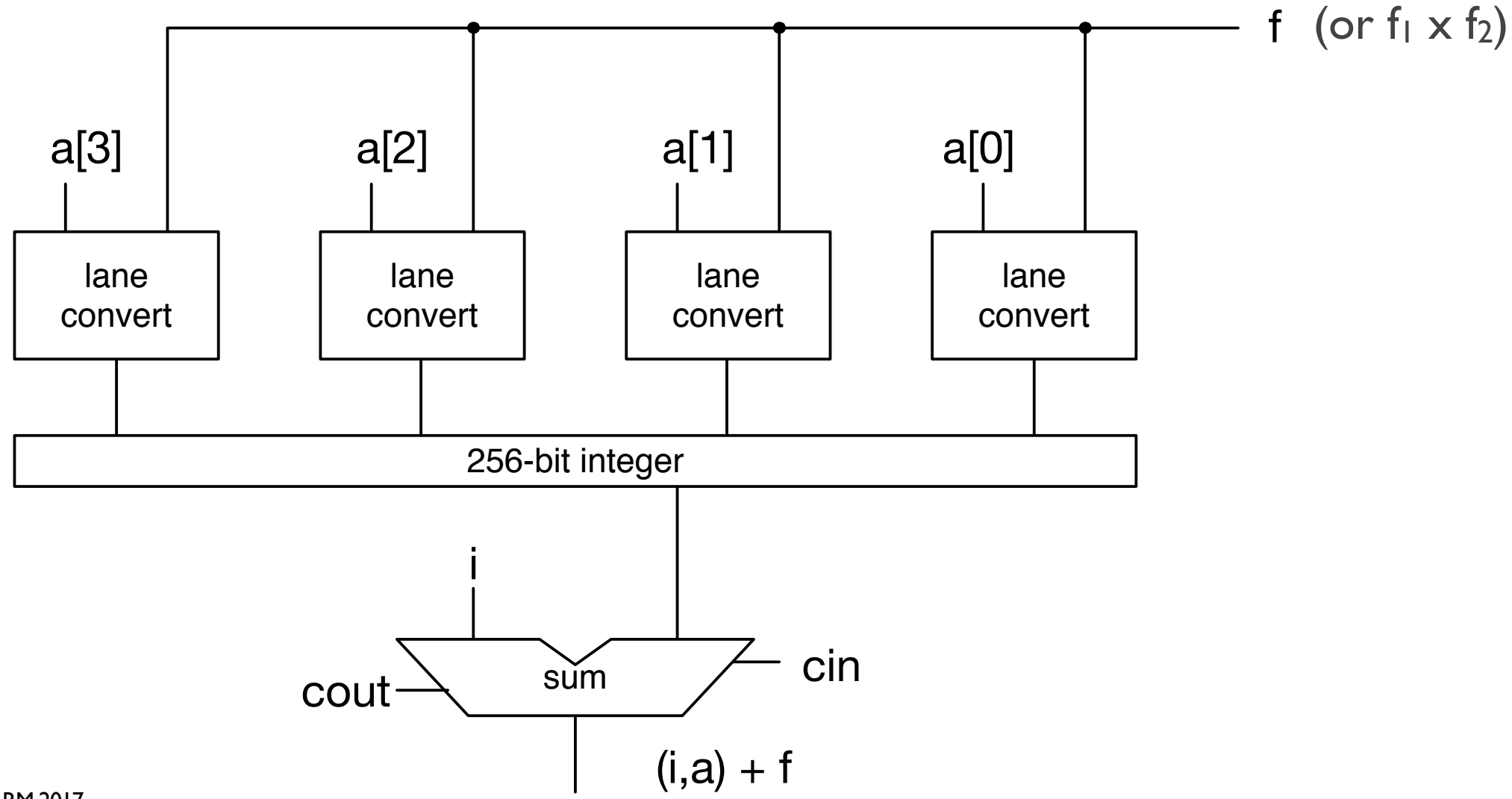
High-Precision Anchored (HPA) numbers

- An HPA number (i, a) comprises:
  - a long 2's-complement integer i, containing 100-200 (or more) bits
  - an anchor a that says how to interpret those bits

- a programmer picks the range for the application area or problem
  - anchor is analogous to a floating-point exponent, but is fixed for a given problem
  - anchor represents the least significant exponent value we are interested in
  - the length of the long integer then gives us a range for associative accumulation

- HPA accumulation is associative, reproducible, and parallelizable

Adding FP number or product to HPA value (i,a)



ARM

128-bit HPA example

	i[1]						i[0]					
bit[127:0]	127	126	125	...	65	64	63	62	...	2	1	0
weight	sign	76	75	...	15	14	13	12	...	-48	-49	-50

- example: 128-bit HPA number (i, -50)
  - `long long i[1:0];  // two 64-bit values`
  - anchor is -50, discard any bits smaller than 2^{-50}
  - lane anchor for i[1] is 14
- sample conversions (SP, 24 significant bits)
  - distance to Andromeda galaxy (2^{74} m)
  - distance Austin to London (2^{22} m)
  - diameter of hydrogen atom (2^{-34} m)
- add or subtract in any order: same result

Inputs outside of the selected range

- on the low end, some numbers may convert as zeros or lose accuracy
  - this should be a deliberate choice to avoid insignificant data
  - addition is still associative, parallelizable, and reproducible
- on the high end, conversion will signal overflow
  - a problem that needs to be fixed
  - add significant bits or use a higher anchor

Redundant HPA for carry-free high performance summation

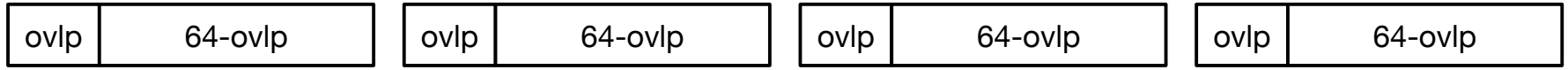
Implementation issues

- A 256-bit SIMD vector is four 64-bit values, not a 256-bit integer
  - carries between 64-bit lanes are a problem
  - negative 256-bit integers are a problem
  - “stay in the lane”

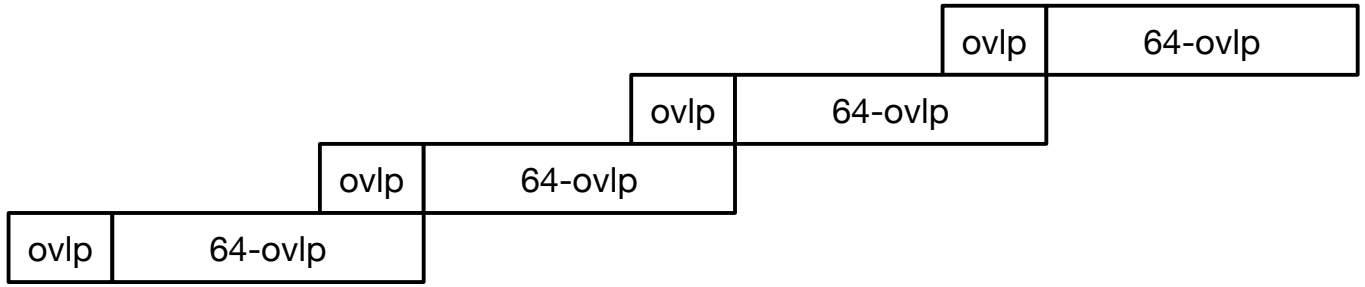
- We don’t need 256-bit integers (100-200 bits sufficient)

Redundant HPA

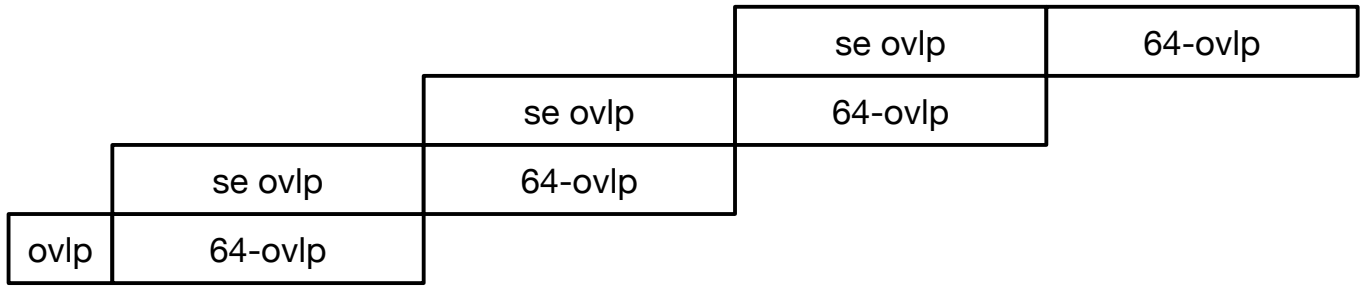
(1) 4 x 64-bit lanes, as seen by each lane



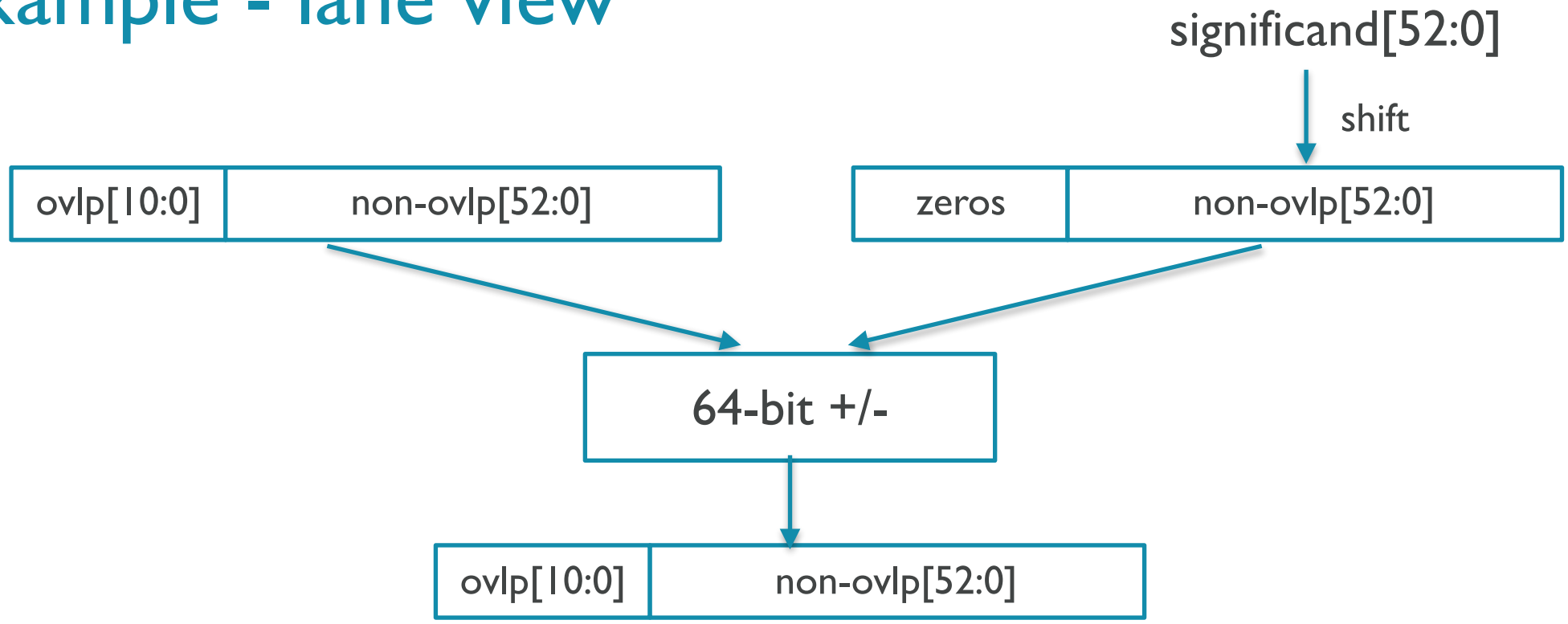
(2) (256-3*ovlp)-bit view, showing the weight of each bit



(3) (256-3*ovlp)-bit view, with overlap bits sign extended



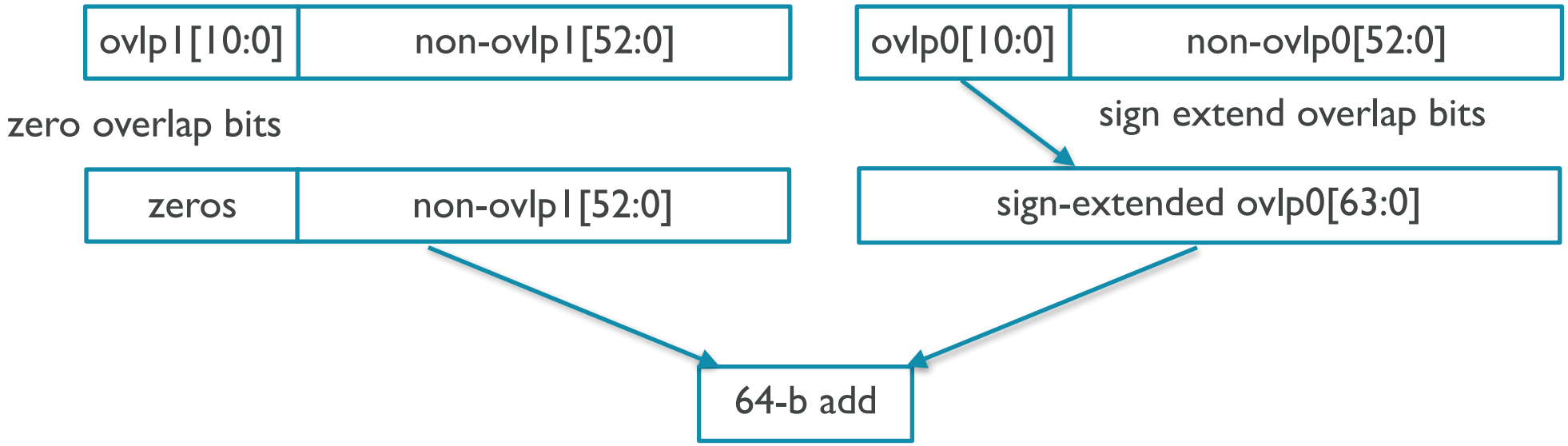
RHPA example - lane view



- compare exponent & anchor, shift significand, 64-bit 2's complement +/-
- +/- based on FP sign and desired operation
- single-cycle latency
- no lane overflow for at least 2^{10} additions/subtractions

RHPA lane arithmetic

- lanes are independent
- different lanes can have different signs
- minimally-redundant form. Every 2^{10} operations, for each lane in parallel:



- non-redundant form is similar, but must be done sequentially, low lane to high

ARM

HPA on SIMD units
two approaches

Implementation issues

- Need variable ranges (100-200 bits, possibly more)
- ARM SVE 128 to 2048 bits, only 128 bits guaranteed
- 200 bit numbers are problematic on 128-bit vectors

Long integers can be stored in 2 ways

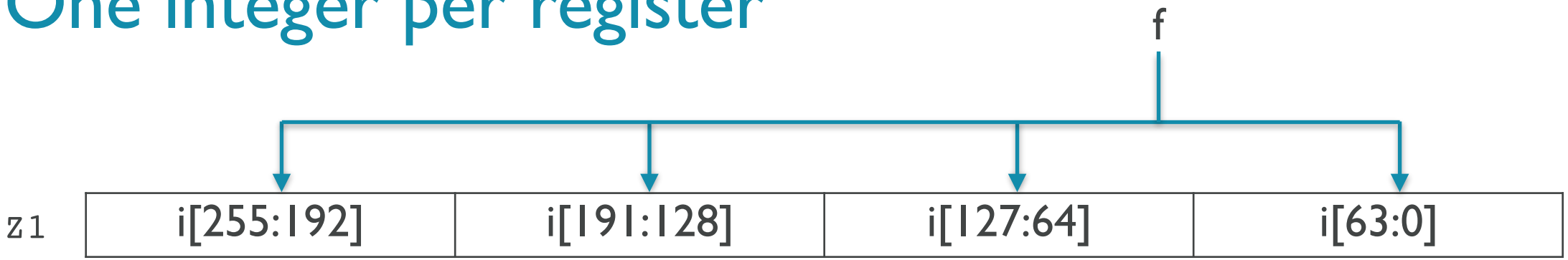
- Obvious way - one integer in per register

z1	i[255:192]	i[191:128]	i[127:64]	i[63:0]

- Less-obvious way - one integer per lane across several SVE registers

z2	b[63:0]	c[63:0]	d[63:0]	e[63:0]
z3	b[127:64]	c[127:64]	d[127:64]	e[127:64]
z4	b[191:128]	c[191:128]	d[191:128]	e[191:128]

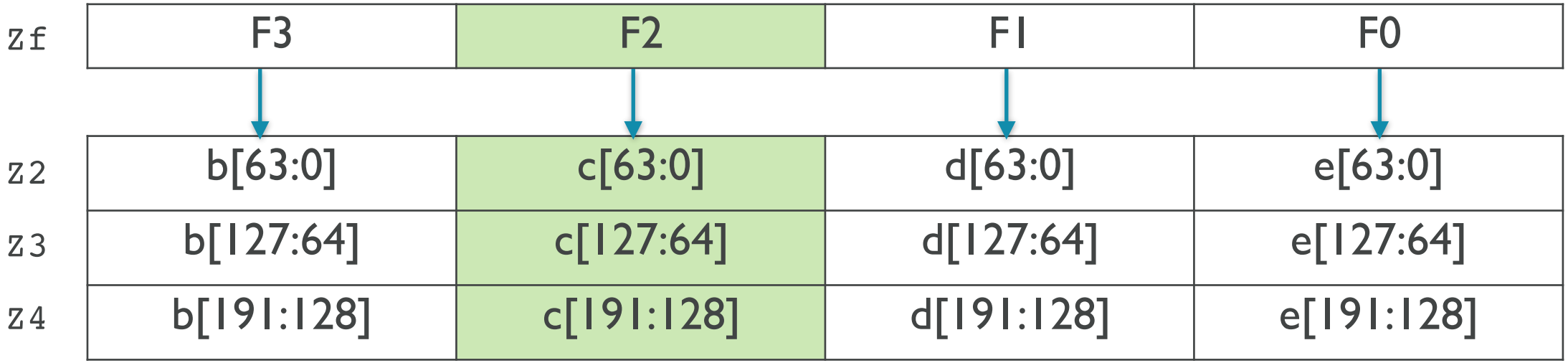
One integer per register



ADD_HPA (Z1, *a*, f)

- adds single FP number f to all lanes of Z1, with anchor *a*
  - each lane has a different lane anchor based on *a*
- f must be broadcast to all lanes (violates "stay in the lane")
- problematic if we need more bits (128-bit SVE, 200-bit HPA)
- wasteful if we need fewer bits (1024-bit SVE, 200-bit HPA)

One integer per lane



- Zf is a vector of four FP values (F3, F2, F1, F0)
- Four HPA integers b, c, d, e across three Z vectors
- Each Z vector contains one lane for four HPA values
- Each Z vector has its own lane anchor
- Computation stays in the lane
- 4 HPA values could correspond to a single accumulation

ADD_HPA (Z2, *a*2, Zf)
ADD_HPA (Z3, *a*3, Zf)
ADD_HPA (Z4, *a*4, Zf)

Performance

- adding an FP number to an HPA lane takes only one cycle
- suppose the HPA integers require 3 lanes:
  - in a 256-bit SVE unit, we can add 256/64 = 4 DP values every 3 cycles
  - could easily do twice as many SP values every 3 cycles
  - second SVE unit doubles this throughput
- for very large sums, accumulation could span multiple processors
  - sums are associative as long as we stay in the HPA format
  - adding two HPA lanes with the same anchor is a 64-bit add
  - no danger of overflow in minimally-redundant form
- conversion back to FP is multi-cycle, dependent on HPA length

Last Slide

- paper has much more information
  - hardware support for Demmel and Nguyen?

- We have ever-larger SIMD units — let's use them for large integers

- Non-associative FP addition is a problem, and one possible solution is HPA:
  - reproducible
  - parallelizable
  - reasonably fast
  - correct

THE END