# Vectorized Fused Dot Product

Thomas Ferrere
Jeremy-Zolnai Lucas
Tai Li
Adedotun Adeyemo

Datapath and Formal team

**Imagination Technologies**

# Problem

Background

- AI acceleration requires dense matrix-multiplication hardware
- Multiple number formats must be supported
  - Floating-point and integer
  - Range and precision options
- Smaller number formats are gaining in popularity
  - Memory bandwidth saving
  - Increased processing

# How can we reduce the circuit area owed to arithmetic implementation?

# Previous solutions

Vector unit

- Each processing element implements a **single operation** over **multiple formats**
- Data is densely packed to maximise throughput for narrow number formats

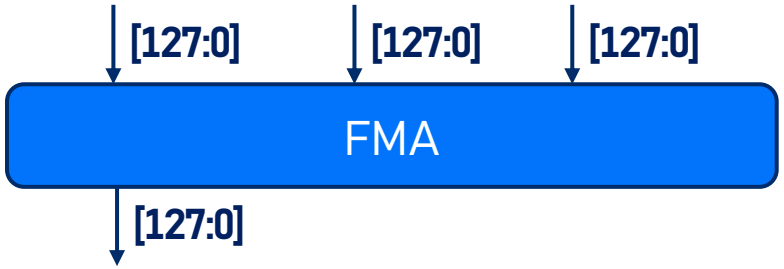
# Previous solutions

Vector unit

- Example vector layout

| FP64 | | | | FP64 | | | |
|---|---|---|---|---|---|---|---|
| FP32 | | FP32 | | FP32 | | FP32 | |
| FP16 | FP16 | FP16 | FP16 | FP16 | FP16 | FP16 | FP16 |

- Example processing element



# Previous solutions

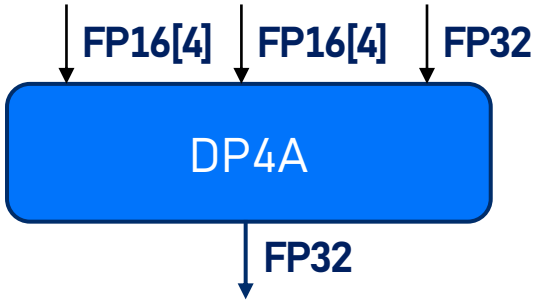
Matrix multiplication unit

- Each processing element implement **multiple operations** over a **single format** combination
- Fused arithmetic is used to maximise performance density

# Previous solution

Matrix multiplication unit

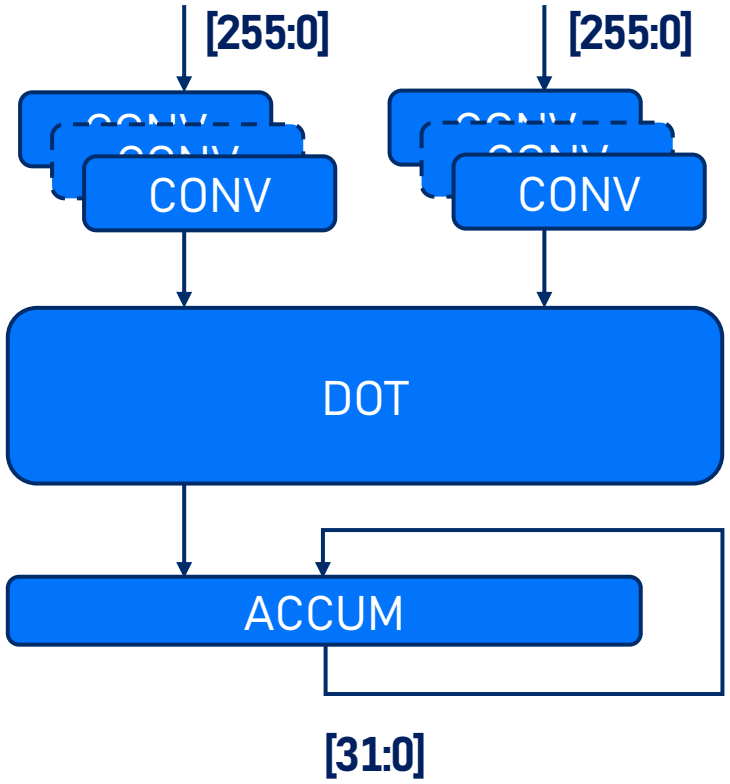
- Example operation
  - d = a0*b0 + a1*b1 + a2*b2 + a3*b3 + c
- Example processing element



# New solution

Dot product pipeline

- Single datapath with fused arithmetic for all number formats



# Vectorized fused dot product

Algorithm

- Multiplication
  - Integers and mantissas are multiplied
- Exponent maximum
  - Exponents are added into product exponents and their maximum is computed
- Mantissa alignment
  - Mantissa products are shifted by respective exponent differences to the maximum
- Summation
  - The integer and aligned mantissa products are summed
- Normalization
  - The mantissa sum is shifted according to its number of leading zeroes

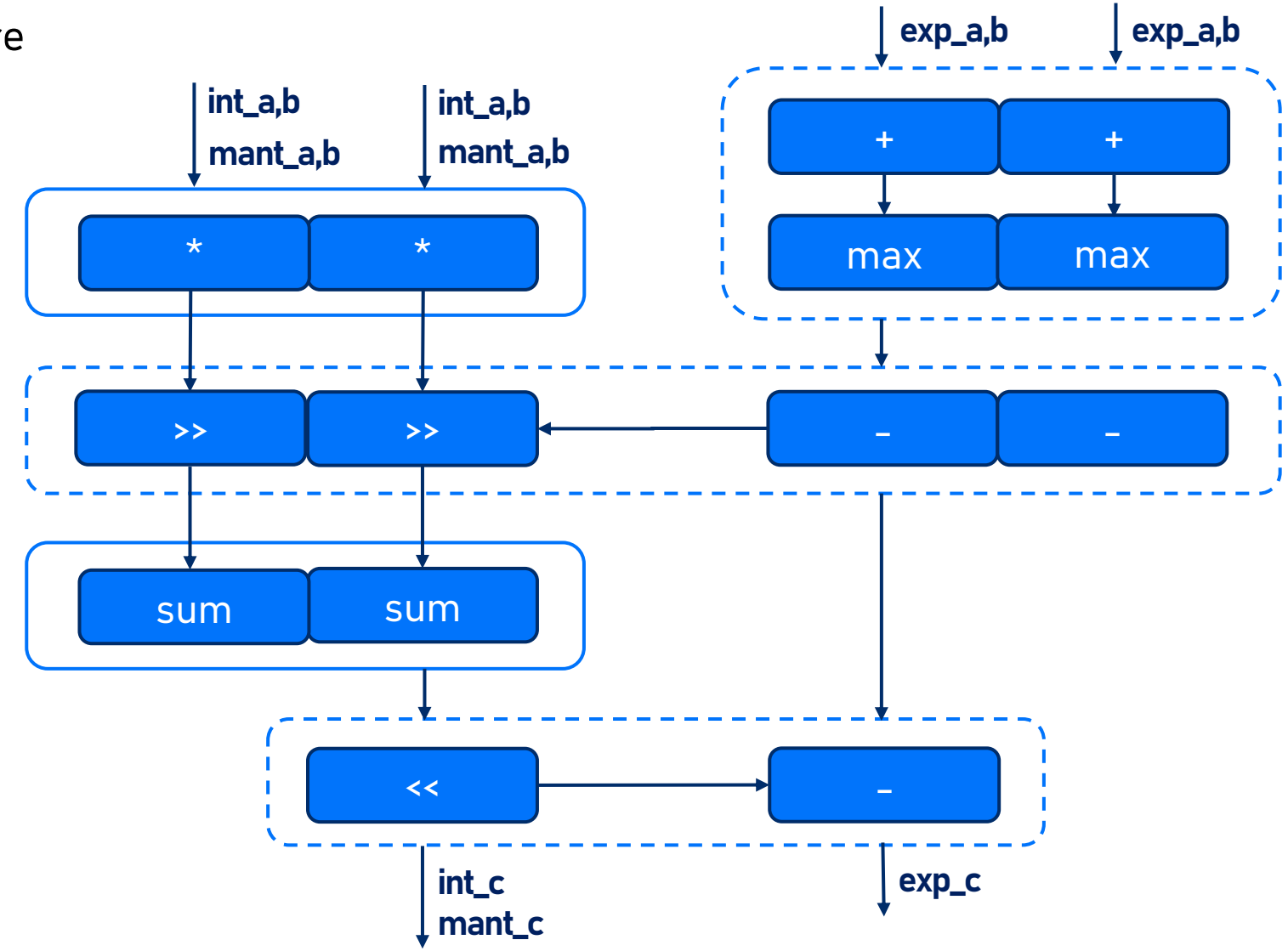
# Vectorized fused dot product

External and internal precision

| Format | BF16/FP16 | FP8 | INT8 |
|---|---|---|---|
| exp_a,b | 8 | 5 | - |
| mant_a,b, int_a,b | 11 | 4 | 8 |
| exp_p | 9 | 6 | - |
| mant_p, int_p | 22 | 8 | 16 |
| mant_q | **32** | **16** | - |
| mant_r | 36 | 21 | 21 |

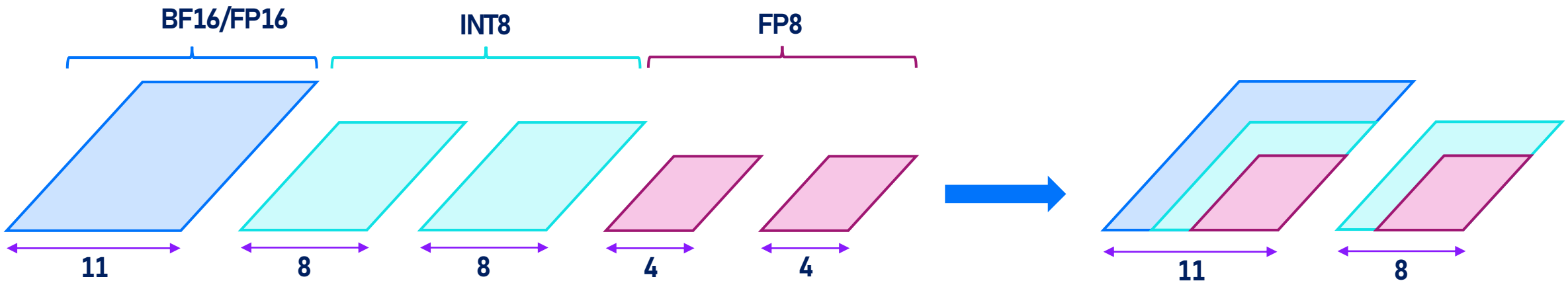
# Vectorized fused dot product

Microarchitecture



# Logic reuse

Multiplication

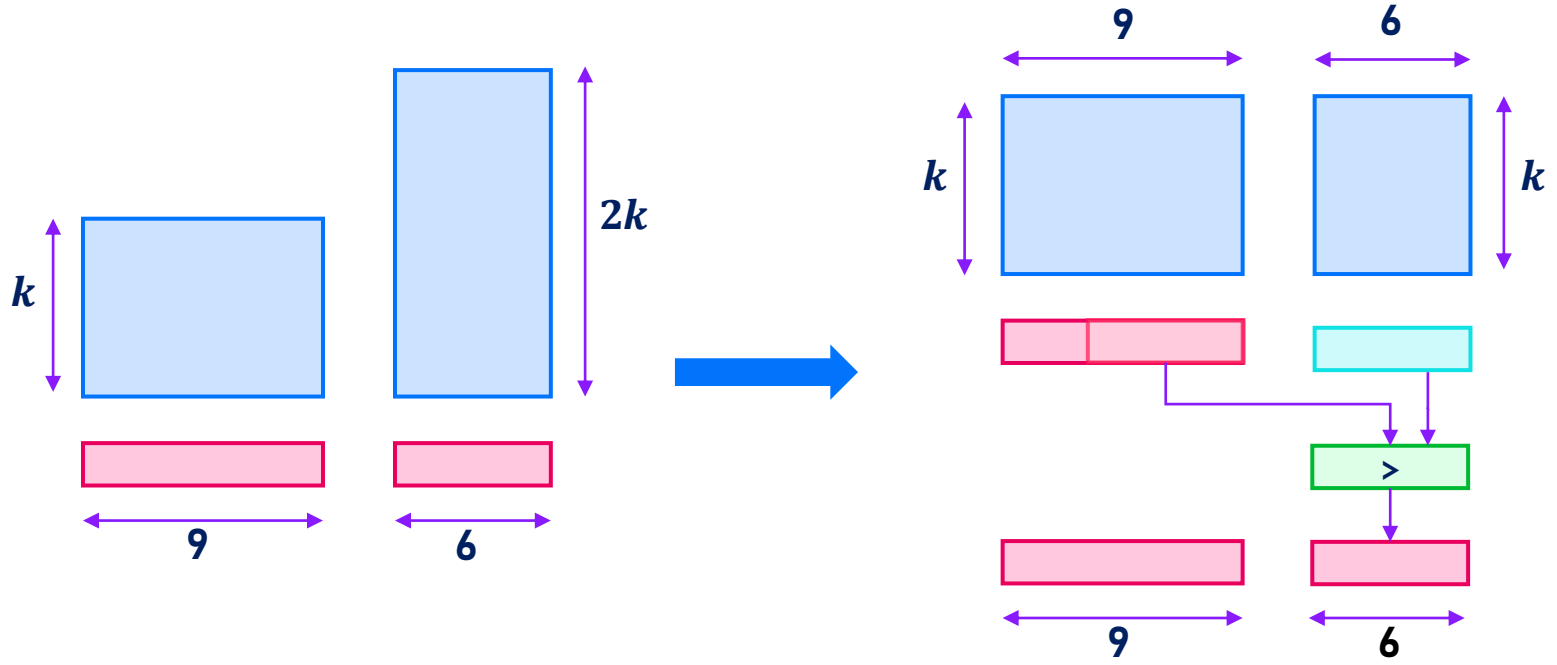



- Wider multipliers are reused for smaller widths
  - Handling of signed INT8 inputs covered when Booth recoding is used for FP16 multiplication

# Logic reuse

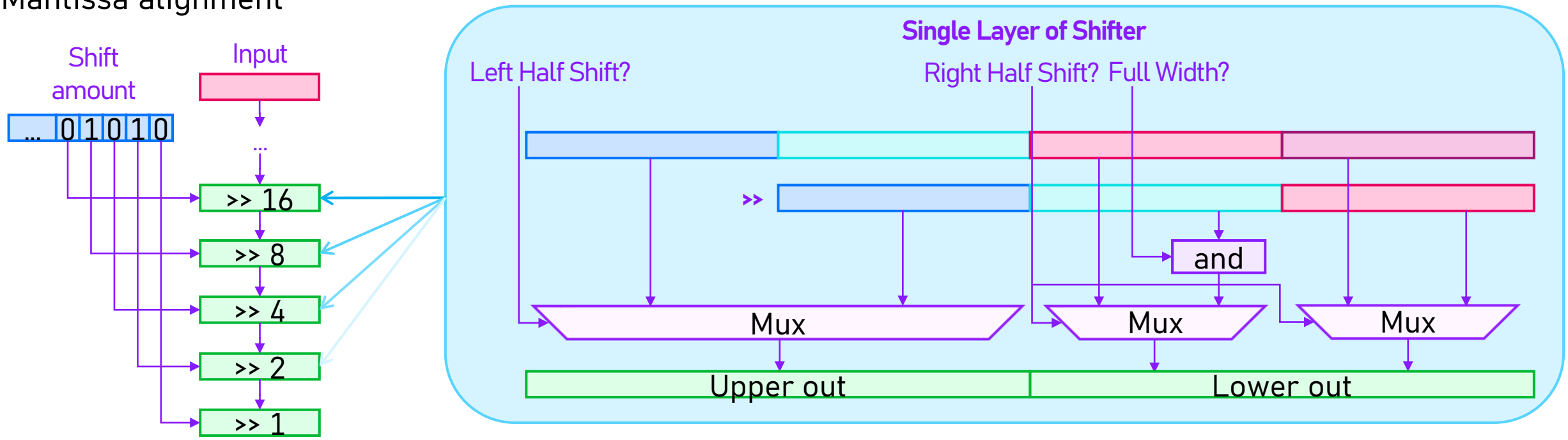
Exponent maximum

- Wider maximum reduction tree is reused for half of smaller width exponents
- Narrower reduction tree for other half of smaller width exponents



# Vectorization

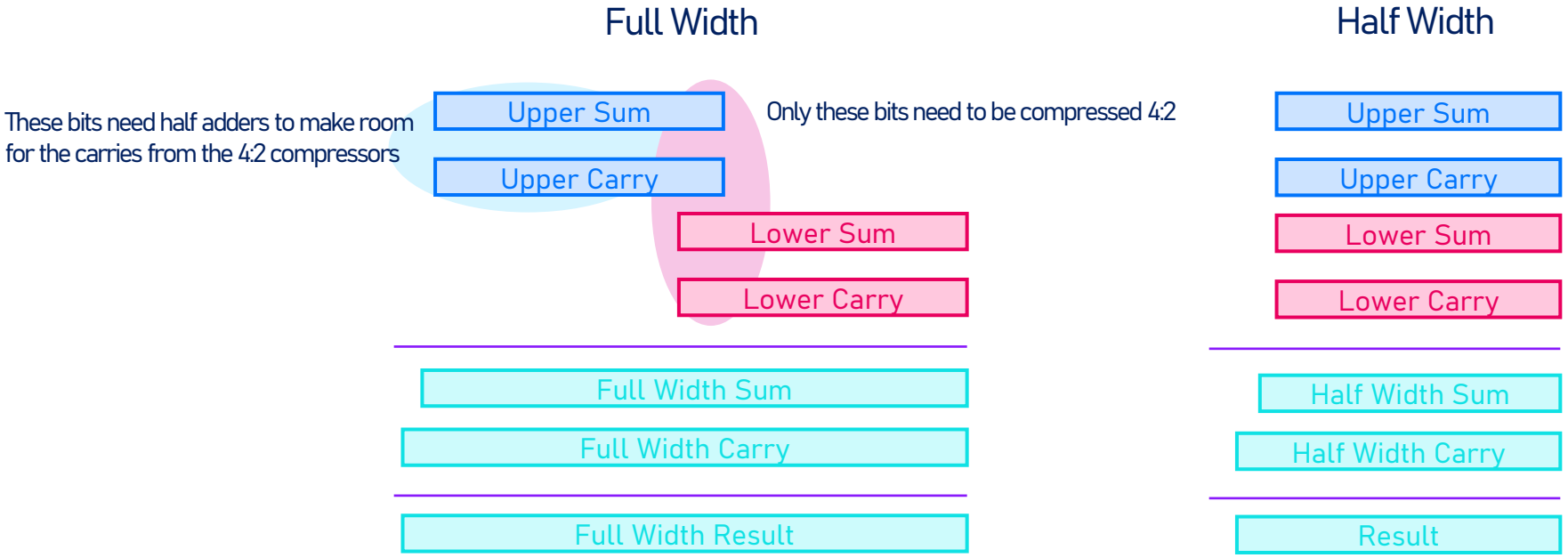
Mantissa alignment




- Wider shifter layers are split into upper and lower halves
- Upper and lower halves are reused for independent narrower terms

# Vectorization

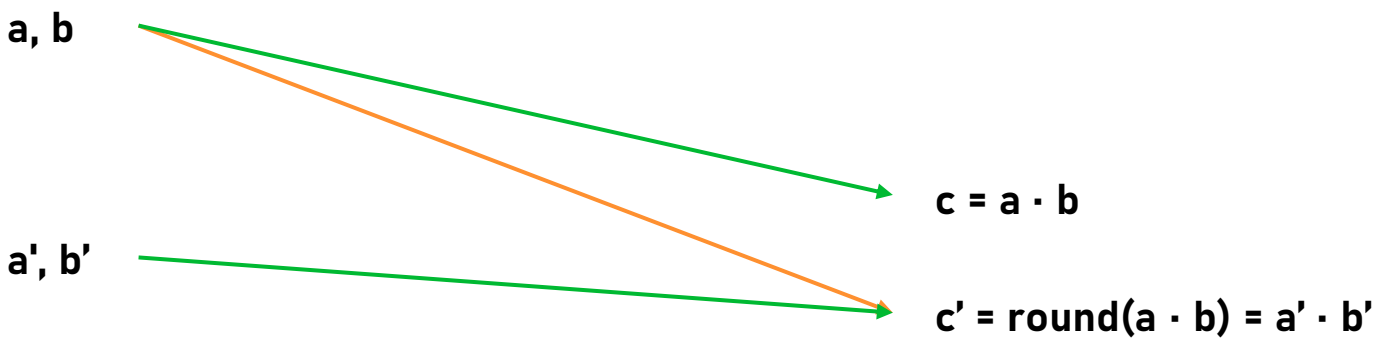
Summation




- Wider terms sum is split into upper and lower halves
- Upper and lower halves are reused for narrower terms sum

# Accuracy

Backward error analysis




- For E4M3, errors on c equivalent to error on a or b less than 0.0152 ulp
- For FP16, errors on c equivalent to error on a or b less than 0.0000144 ulp

# Accuracy

Forward error measurement

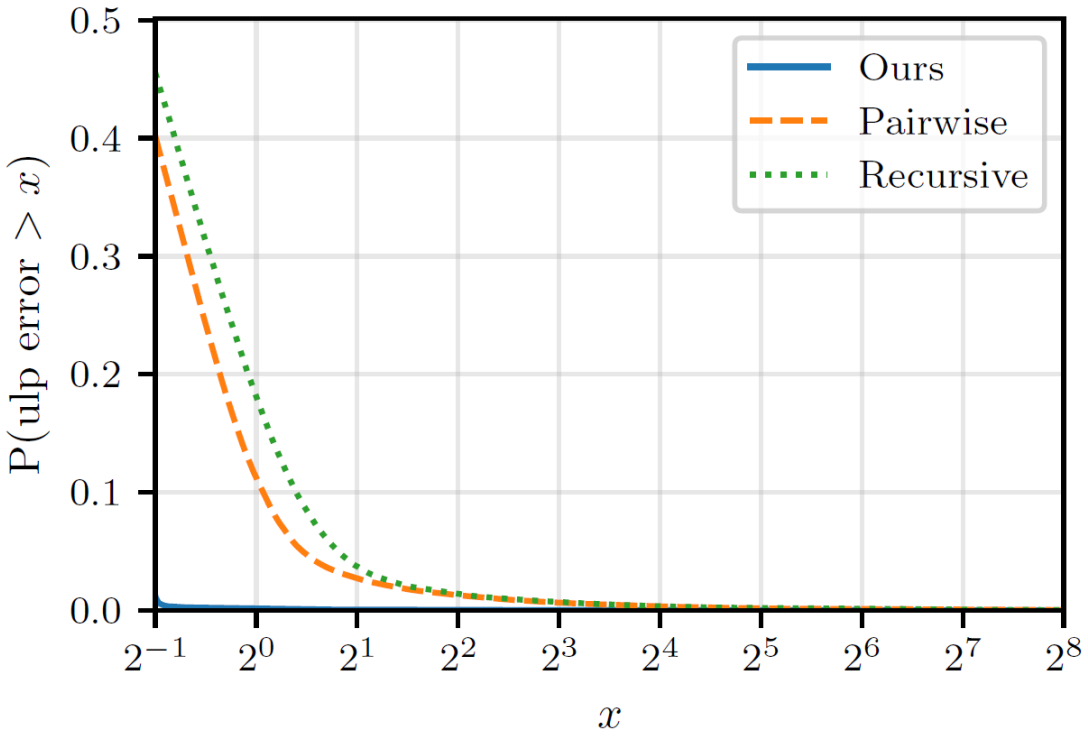
- Simulate dot products of floating points with each bit i.i.d.
- Compare the fused summation result against recursive and pairwise summation

| Multiply | Accumulate | Depth | Recursive | Pairwise | Fused | Exact |
|---|---|---|---|---|---|---|
| BF16 | FP32 | 16 | 0.186 | 0.182 | **0.145** | 0.145 |
| FP16 | FP32 | 16 | 1.373 | 1.310 | **0.259** | 0.251 |
| E5M2 | FP16 | 32 | 1.160 | 1.058 | **0.406** | 0.246 |
| E4M3 | FP16 | 32 | 2.690 | 1.744 | **0.490** | 0.250 |

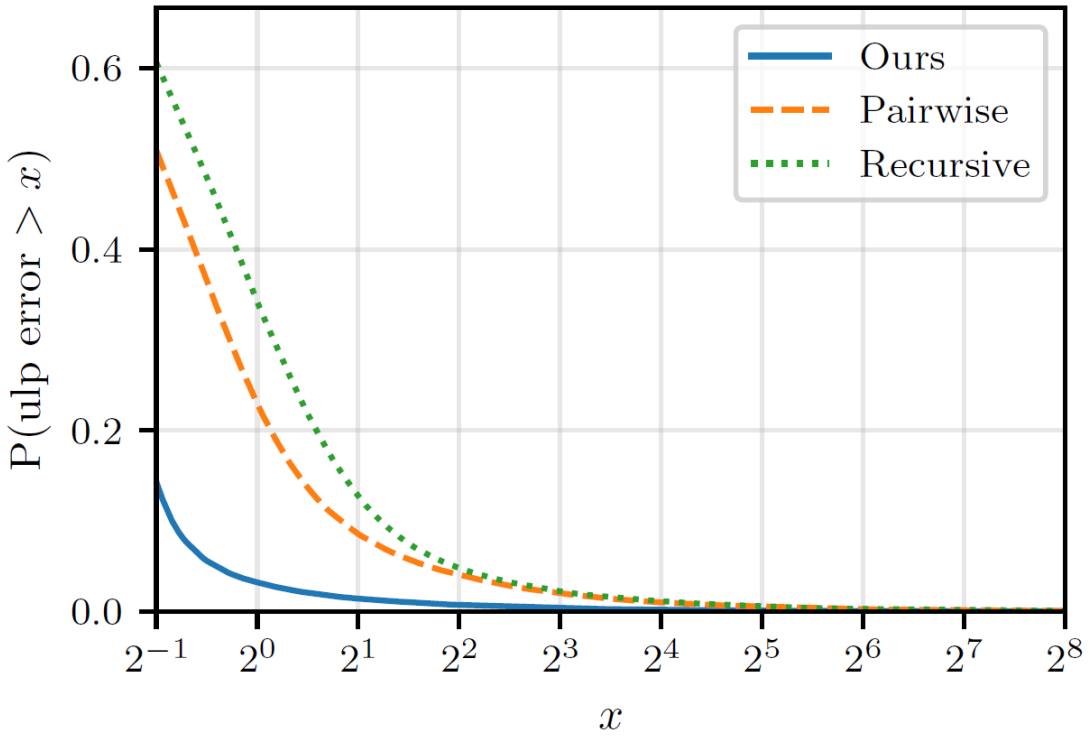
# Accuracy

Forward error measurement

- 16 FP6 multiplications with FP32 accumulation




- 32 FP8 multiplications with FP16 accumulation



# Synthesis

Area per stage of processing with/out merging and vectorisation

| | Scalar | Scalar | Merged | Scalar | Vectorised |
|---|---|---|---|---|---|
| **FP8** | **32x** | | **32x** | | **32x** |
| **INT8** | | **32x** | **32x** | | **32x** |
| **BF16/FP16** | | | | **16x** | **16x** |
| Multiplication | 77 | 506 | 506 | 407 | 673 |
| Exponent max | 138 | | 137 | 89 | 153 |
| Alignment shift | 147 | | 147 | 196 | 265 |
| Summation | 174 | 162 | 216 | 175 | 249 |
| Normalisation | 9 | | 12 | 23 | 25 |
| Other comb. | 83 | 17 | 94 | 67 | 132 |
| Registers | 209 | 204 | 370 | 215 | 493 |
| Total | 836 | 889 | 1482 | 1172 | 1990 |

- Area saving of 31.5% compared to scalar pipelines

# Perspectives

- Depth scaling enables simplified Matmul accelerator architecture
  - Other methods of scaling are possible
- Implementation compatible with OCL MX formats
- Vectorization approach tailored towards small-width formats
  - Reduction operations require novel technique
- Internal rounding errors limited to alignment stage
  - Promotes hardware-agnostic specification
  - Enables end to end formal verification

# THANK YOU_

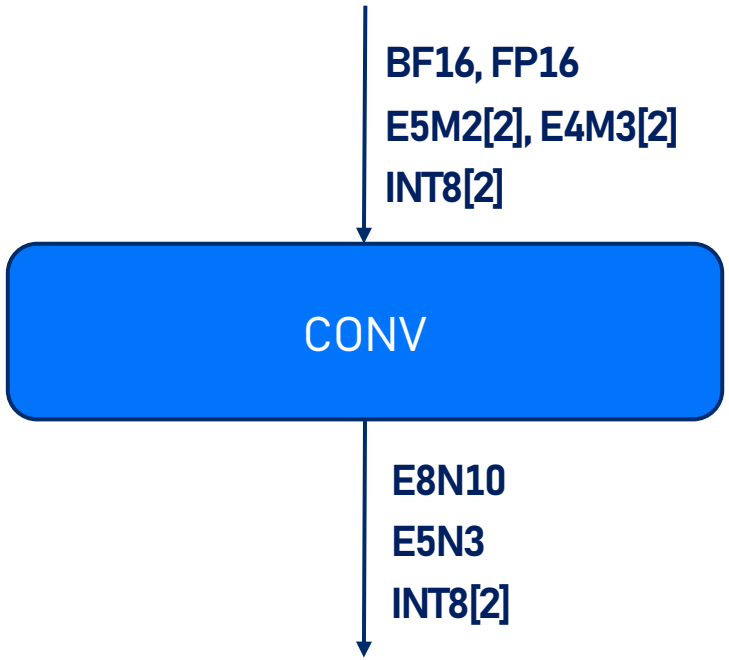
Imagination

# Appendix

# New solution

Normalized number formats

- Packed formats are unified and normalized



# Normalized number formats

16-bit floating point

| Format | Exponent | Mantissa |
|---|---|---|
| BF16 | 8 | 7 |
| FP16 | 5 | 10 |
| E8N10 | **8** | **10** |

- BF16 denormals are flushed to zero
- FP16 denormals are preserved
  - Normal and denormal values representable as normal numbers

# Normalized number formats

8-bit floating point

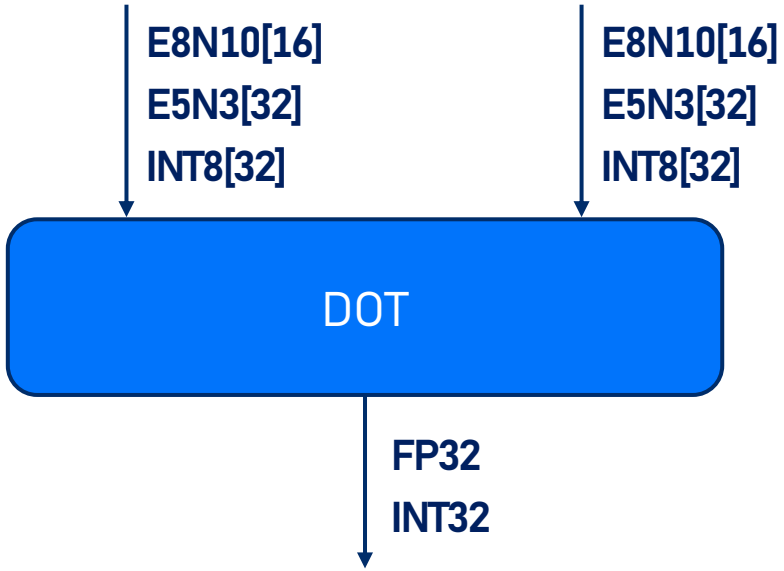
| Format | Exponent | Mantissa |
|---|---|---|
| E5M2 | 5 | 2 |
| E4M3 | 4 | 3 |
| E5N3 | **5** | **3** |

- E5M2 and E4M3 denormals are preserved
  - Nonexceptional values are all representable with bias = 16
  - Exceptional values can use separate flags

# New solution

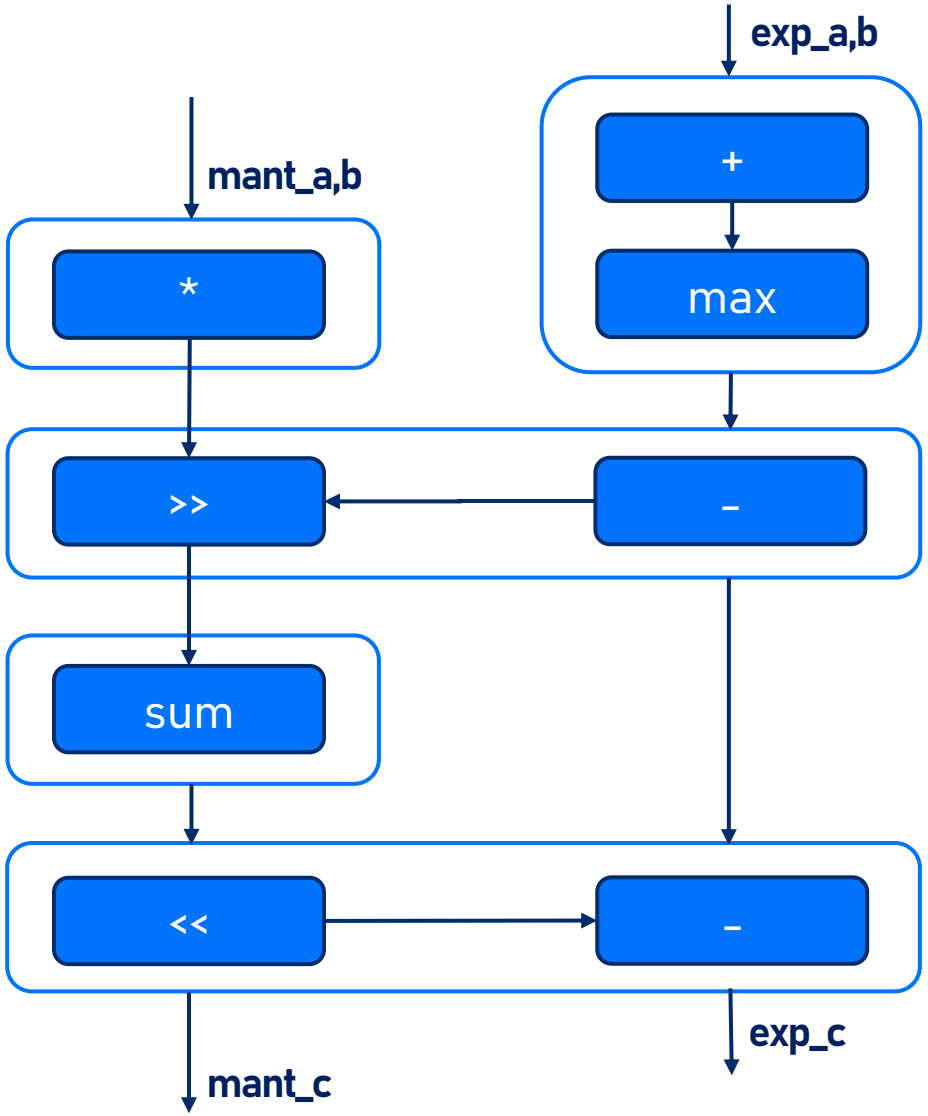
Vectorized fused dot product

- A single processing element fuses multiple operations and number formats



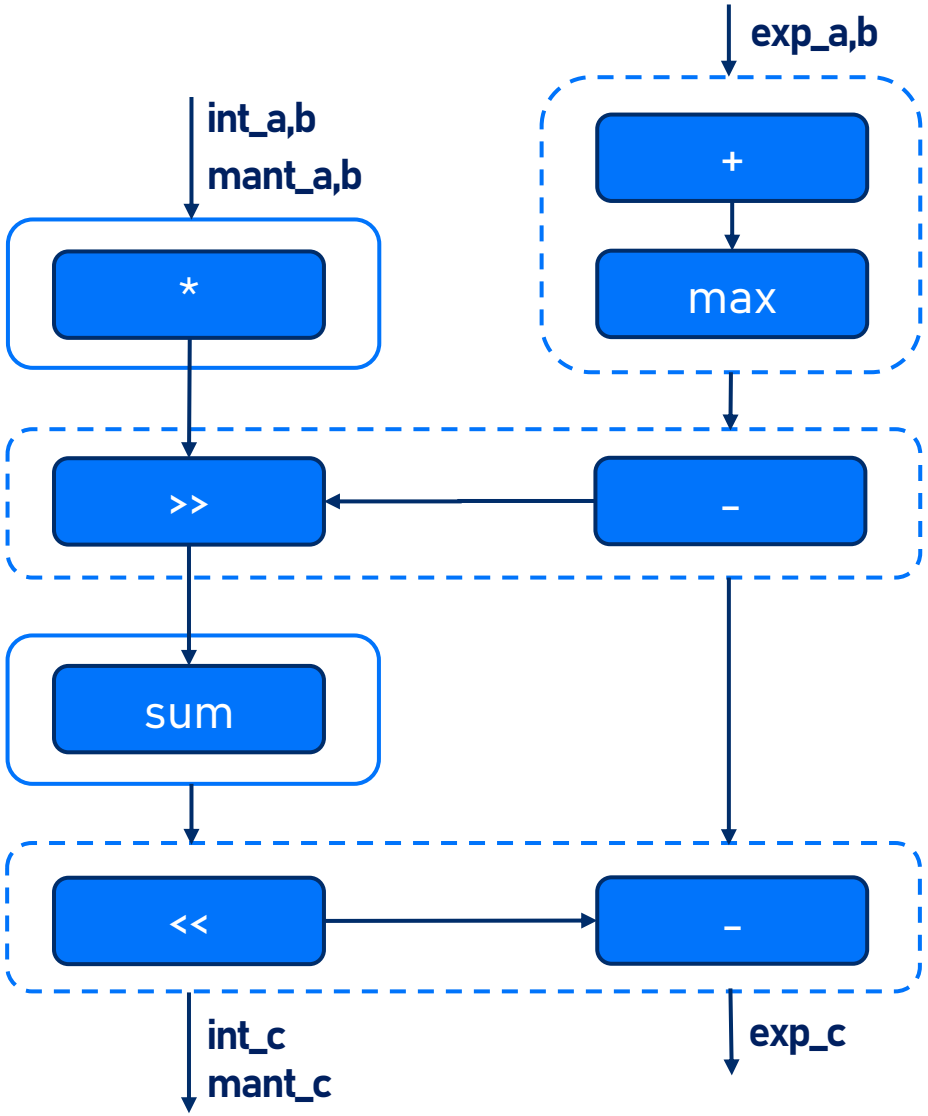
# Vectorized fused dot product

Floating point operation



# Vectorized fused dot product

Integer operation



# New solution

Shared accumulation stage

- Supports addition in integer, floating point and mixed precision