

# Exact Fused Dot Product Add Operators

Orégane Desrentes  
Benoît Dupont de Dinechin  
Florent de Dinechin



# Motivation

Short and biased history of floating point units

- In the 70s, adders and multipliers



# Motivation

Short and biased history of floating point units

- In the 70s, adders and multipliers
- In the 90s, FMA (fused multiply add):  $R = \circ(X \times Y + Z)$ 
  - two operations in one instruction: *faster*
  - one single rounding: *more accurate*



# Motivation

Short and biased history of floating point units

- In the 70s, adders and multipliers
- In the 90s, FMA (fused multiply add):  $R = \circ(X \times Y + Z)$
- These days:  $R \approx \sum_{i=0}^{N-1} (X_i \times Y_i) + Z$



# Motivation

Short and biased history of floating point units

- In the 70s, adders and multipliers
- In the 90s, FMA (fused multiply add):  $R = \circ(X \times Y + Z)$
- These days:  $R \approx \sum_{i=0}^{N-1} (X_i \times Y_i) + Z$
- This work:  $R = \circ(\sum_{i=0}^{N-1} (X_i \times Y_i) + Z)$



# Motivation

Short and biased history of floating point units

- In the 70s, adders and multipliers
- In the 90s, FMA (fused multiply add):  $R = \circ(X \times Y + Z)$
- These days:  $R \approx \sum_{i=0}^{N-1} (X_i \times Y_i) + Z$
- This work:  $R = \circ(\sum_{i=0}^{N-1} (X_i \times Y_i) + Z)$

In the spirit of IEEE standard: single rounding and subnormal support



# Motivation

Short and biased history of floating point units

- In the 70s, adders and multipliers
- In the 90s, FMA (fused multiply add):  $R = \circ(X \times Y + Z)$
- These days:  $R \approx \sum_{i=0}^{N-1} (X_i \times Y_i) + Z$
- This work:  $R = \circ(\sum_{i=0}^{N-1} (X_i \times Y_i) + Z)$

In the spirit of IEEE standard: single rounding and subnormal support

Multiple applications

- Matrix multiplication (neural networks, graphical applications, scientific computing, ...)
- Complex arithmetic (FFT,...)



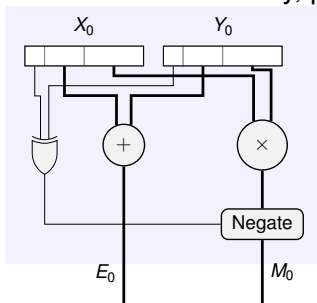
Goal : Hardware floating-point for  $\sum_{i=0}^{N-1} (X_i \times Y_i) + Z$

Low latency, parallel architecture



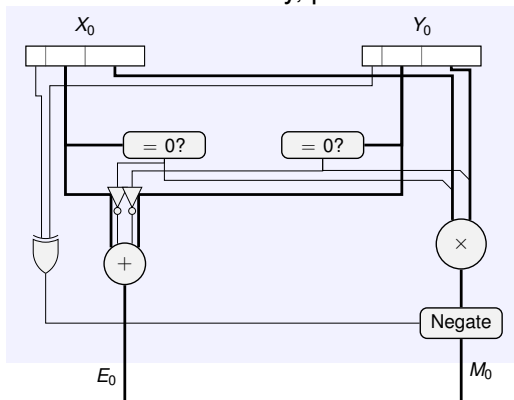
Goal : Hardware floating-point for  $\sum_{i=0}^{N-1} (X_i \times Y_i) + Z$

Low latency, parallel architecture



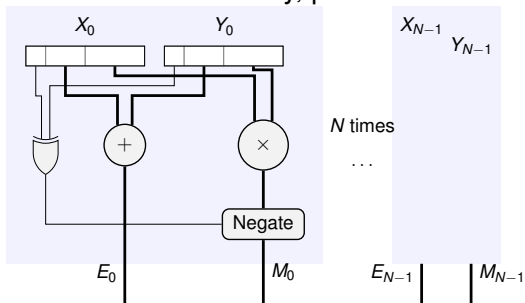
Goal : Hardware floating-point for  $\sum_{i=0}^{N-1} (X_i \times Y_i) + Z$

Low latency, parallel architecture



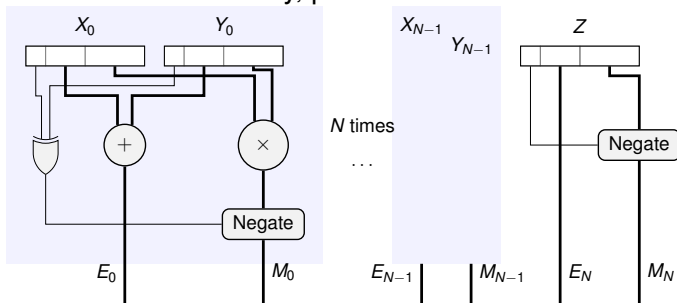
Goal : Hardware floating-point for  $\sum_{i=0}^{N-1} (X_i \times Y_i) + Z$

## Low latency, parallel architecture



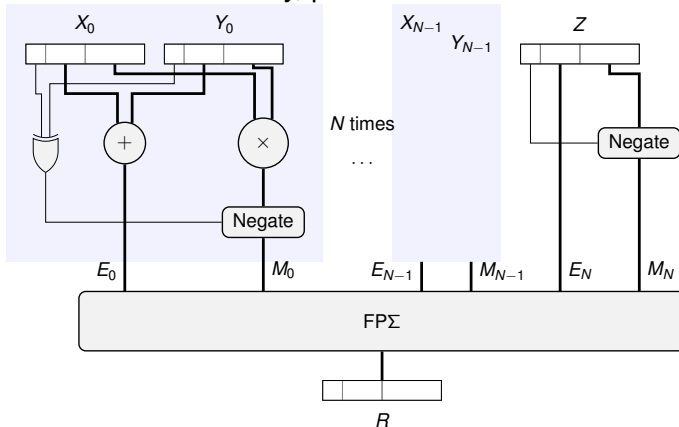
Goal : Hardware floating-point for  $\sum_{i=0}^{N-1} (X_i \times Y_i) + Z$

## Low latency, parallel architecture



Goal : Hardware floating-point for  $\sum_{i=0}^{N-1} (X_i \times Y_i) + Z$

Low latency, parallel architecture



# Usual formats

$Z$  and  $R$  have same format ( $e_{\text{out}}, m_{\text{out}}$ ): allows iterated computations.

$X_i$  and  $Y_i$  can have a format ( $e_{\text{in}}, m_{\text{in}}$ ) smaller than  $Z$ .

Using AVX 512 to determine plausible  $N$



# Usual formats

$Z$  and  $R$  have same format ( $e_{\text{out}}, m_{\text{out}}$ ): allows iterated computations.

$X_i$  and  $Y_i$  can have a format ( $e_{\text{in}}, m_{\text{in}}$ ) smaller than  $Z$ .

Using AVX 512 to determine plausible  $N$

- $\Sigma (\text{FP16} \times \text{FP16}) + \text{FP32} = \text{FP32}$ , written as **FP16  $\rightarrow$  FP32**.  
 $N \leq 16$



# Usual formats

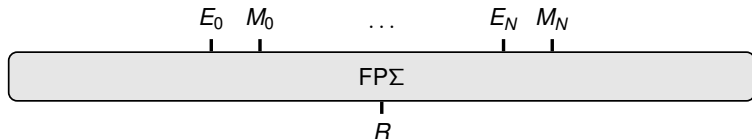
$Z$  and  $R$  have same format ( $e_{\text{out}}, m_{\text{out}}$ ): allows iterated computations.  
 $X_i$  and  $Y_i$  can have a format ( $e_{\text{in}}, m_{\text{in}}$ ) smaller than  $Z$ .

Using AVX 512 to determine plausible  $N$

- $\Sigma (\text{FP16} \times \text{FP16}) + \text{FP32} = \text{FP32}$ , written as **FP16  $\rightarrow$  FP32**.  
 $N \leq 16$
- $\text{BF16} \rightarrow \text{FP32}$ ,  $N \leq 16$
- $\text{FP32} \rightarrow \text{FP32}$ ,  $N \leq 8$
- $\text{FP32} \rightarrow \text{FP64}$ ,  $N \leq 8$
- $\text{FP64} \rightarrow \text{FP64}$ ,  $N \leq 4$
- $\text{FP8} \rightarrow \text{FP16}$ ,  $N \leq 32$
- $\text{FP8} \rightarrow \text{FP32}$ ,  $N \leq 32$



## How to add floating point numbers



# Kulisch's idea : compute the sum in fixed point

- Example:  $N = 2$ ,  
exponent bits  $e = 5$ , significand fraction bits  $m = 2$
- $M_i$  is either a significand product, or the significand of  $Z$
- Signs irrelevant here (manage using classical 2's complement)

|   |  |       |  |  |  |       |  |
|---|--|-------|--|--|--|-------|--|
|   |  | $E_0$ |  |  |  | $M_0$ |  |
| + |  | $E_1$ |  |  |  | $M_1$ |  |
| + |  | $E_2$ |  |  |  | $M_2$ |  |



# Kulisch's idea : compute the sum in fixed point

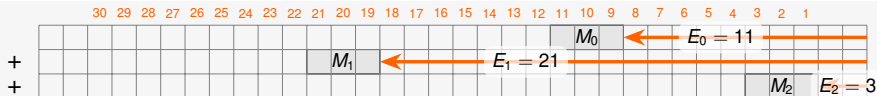
- Example:  $N = 2$ ,  
exponent bits  $e = 5$ , significand fraction bits  $m = 2$
- $M_i$  is either a significand product, or the significand of  $Z$
- Signs irrelevant here (manage using classical 2's complement)

|   |  |    |  |  |  |       |  |
|---|--|----|--|--|--|-------|--|
|   |  | 11 |  |  |  | $M_0$ |  |
| + |  | 21 |  |  |  | $M_1$ |  |
| + |  | 3  |  |  |  | $M_2$ |  |



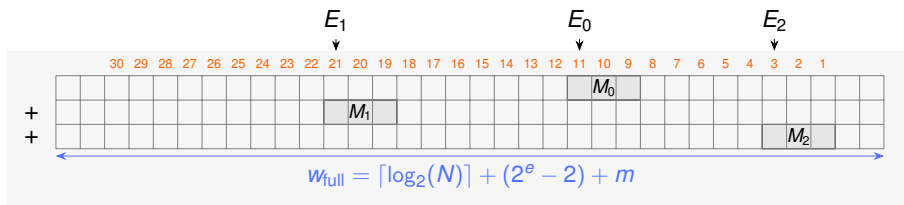
# Kulisch's idea : compute the sum in fixed point

- Example:  $N = 2$ ,  
exponent bits  $e = 5$ , significand fraction bits  $m = 2$
- $M_i$  is either a significand product, or the significand of  $Z$
- Signs irrelevant here (manage using classical 2's complement)



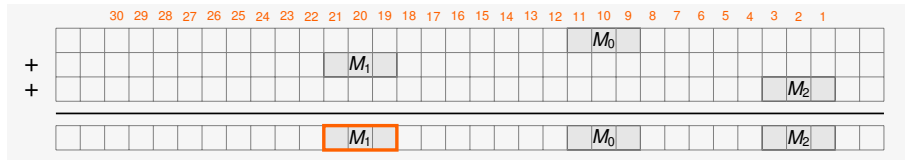
# Kulisch's idea : compute the sum in fixed point

- Example:  $N = 2$ ,  
exponent bits  $e = 5$ , significand fraction bits  $m = 2$
- $M_i$  is either a significand product, or the significand of  $Z$
- Signs irrelevant here (manage using classical 2's complement)



# Kulisch's idea : compute the sum in fixed point

- Example:  $N = 2$ ,  
exponent bits  $e = 5$ , significand fraction bits  $m = 2$
- $M_i$  is either a significand product, or the significand of  $Z$
- Signs irrelevant here (manage using classical 2's complement)

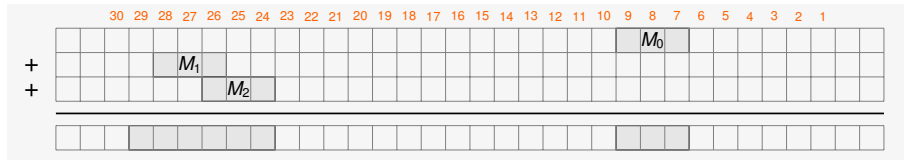


Lots of empty bits, not always in the same place



# Kulisch's idea : compute the sum in fixed point

- Example:  $N = 2$ ,  
exponent bits  $e = 5$ , significand fraction bits  $m = 2$
- $M_i$  is either a significand product, or the significand of  $Z$
- Signs irrelevant here (manage using classical 2's complement)



Lots of empty bits, not always in the same place

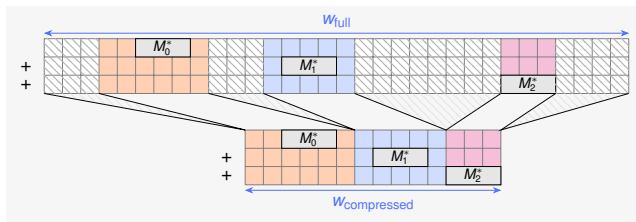


# Tao et al.'s idea : Compressing the empty bits

There's a lot of empty bits  $\rightarrow$  remove them

*Correctly Rounded Architectures for Floating-Point Multi-Operand Addition and Dot-Product Computation*, ASAP 2013, Yao Tao, Gao Deyuan, Fan Xiaoya, Jari Nurmi

Can compute  $R = \circ(\sum_{i=0}^{N-1} X_i)$  and  $R = \circ(\sum_{i=0}^{N-1} (X_i \times Y_i))$ , without subnormal support

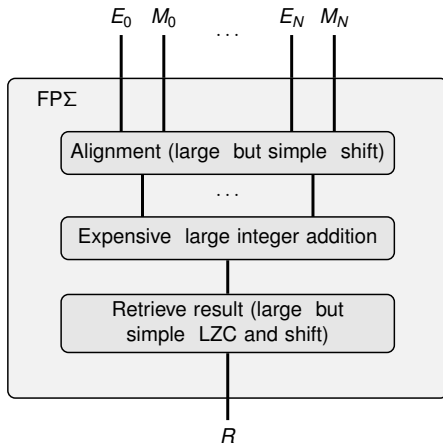


The sum is exact so correct rounding is possible.

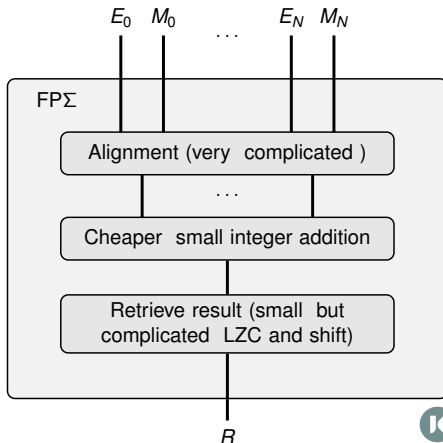


# Architecture overview

## Kulisch

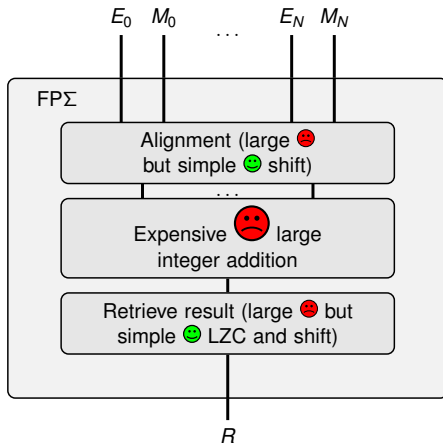


## Compressed

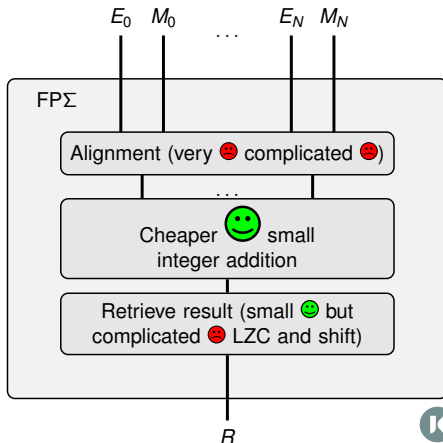


# Architecture overview

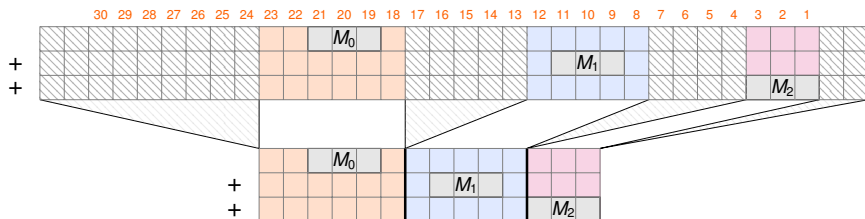
## Kulisch



## Compressed



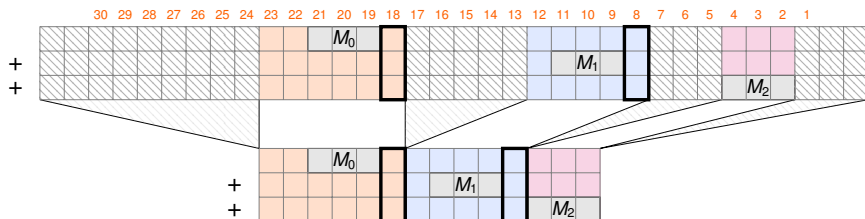
# Compressing the empty bits : Case analysis



Both show alignment for an addition with carry propagation.  
 When the significands do not overlap: 3 zones (easy case)

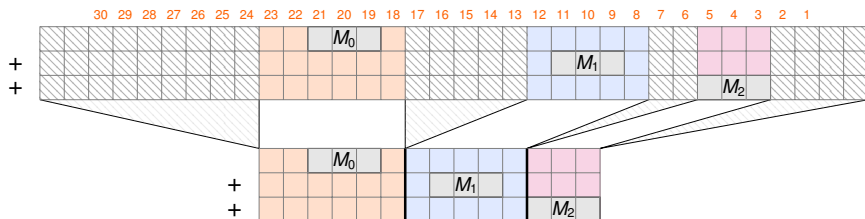


# Compressing the empty bits : Case analysis

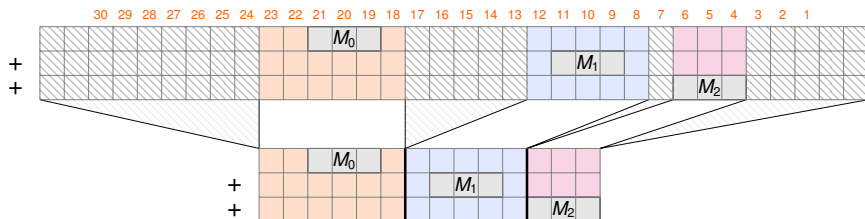


These are placeholder round bits

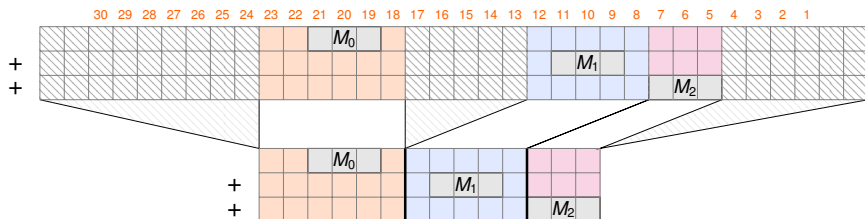
# Compressing the empty bits : Case analysis



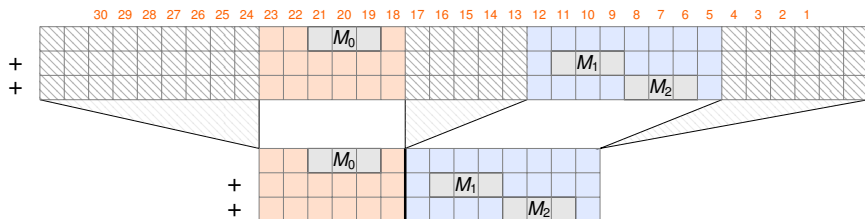
# Compressing the empty bits : Case analysis



# Compressing the empty bits : Case analysis



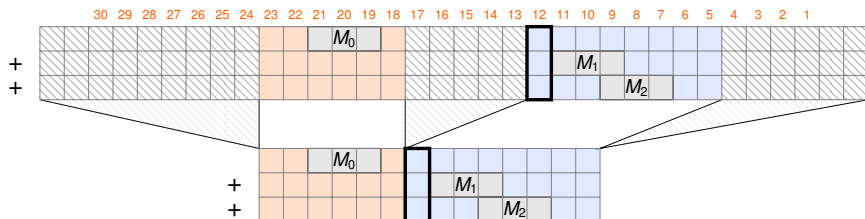
# Compressing the empty bits : Case analysis



Merge zones when significands overlap



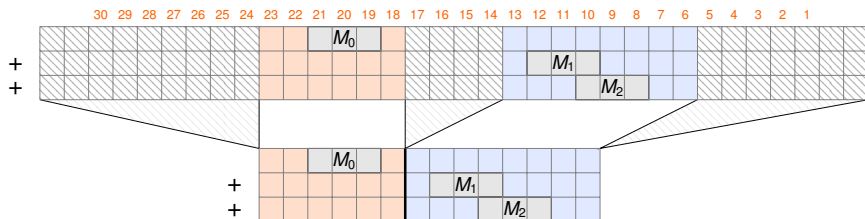
# Compressing the empty bits : Case analysis



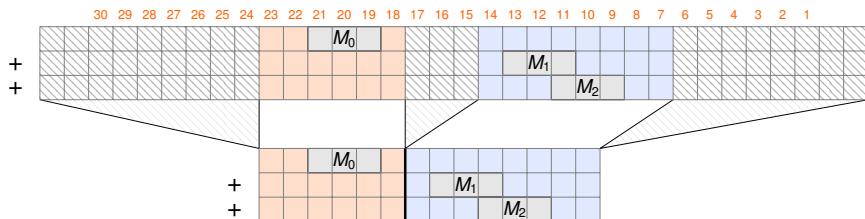
This is a protection bit in case of carry out from  $M_1 + M_2$



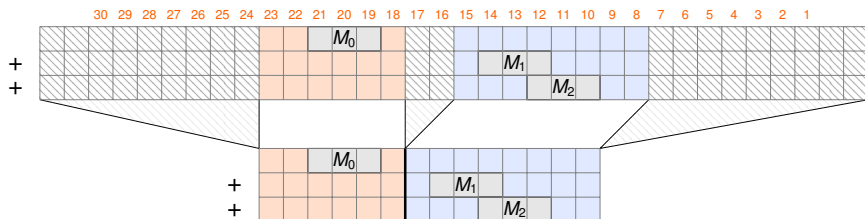
# Compressing the empty bits : Case analysis



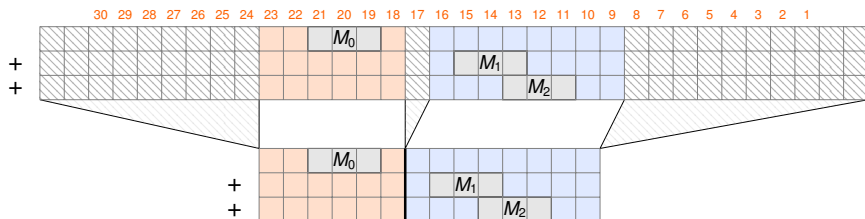
# Compressing the empty bits : Case analysis



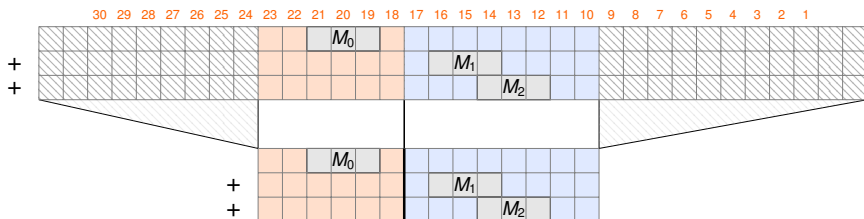
# Compressing the empty bits : Case analysis



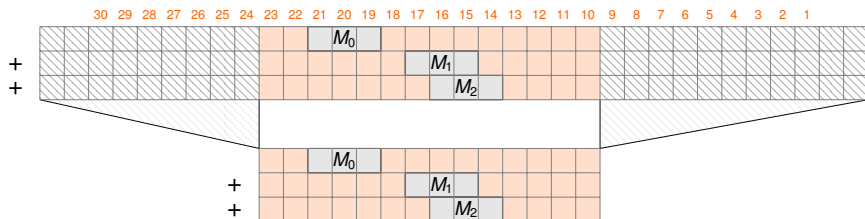
# Compressing the empty bits : Case analysis



# Compressing the empty bits : Case analysis

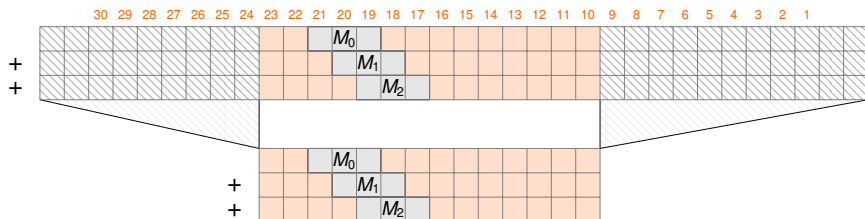


# Compressing the empty bits : Case analysis

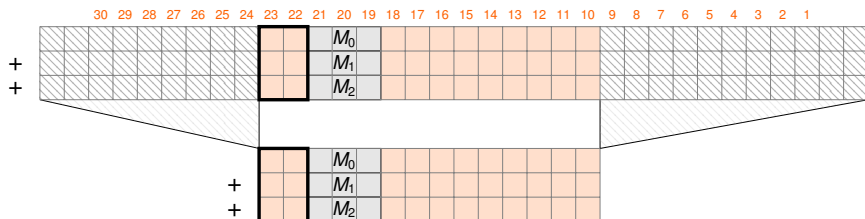


Merge zones when significands overlap

# Compressing the empty bits : Case analysis



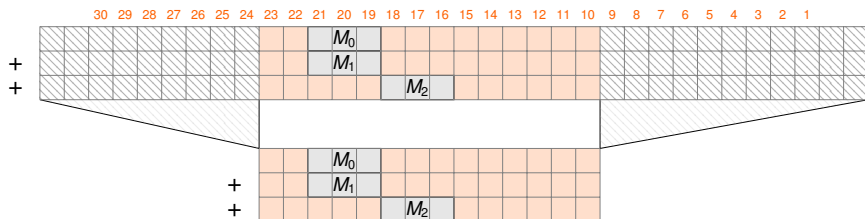
# Compressing the empty bits : Case analysis



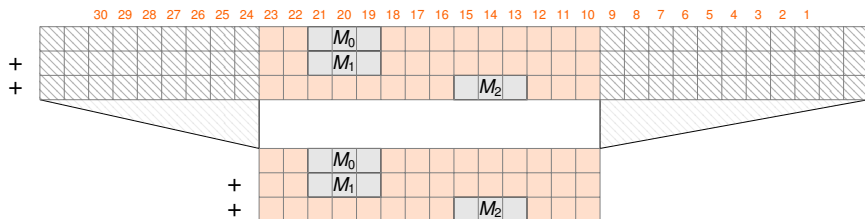
These are protection bits in case of carries from  $M_0 + M_1 + M_2$



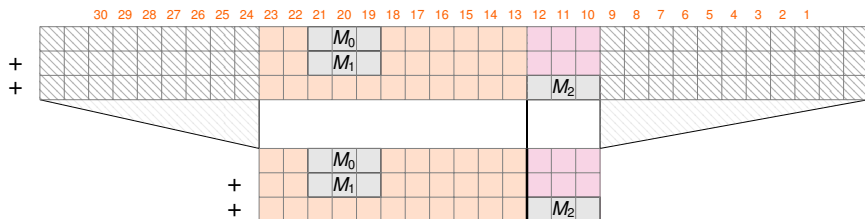
# Compressing the empty bits : Case analysis



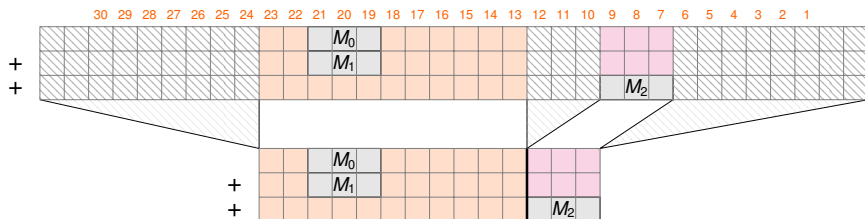
# Compressing the empty bits : Case analysis



# Compressing the empty bits : Case analysis



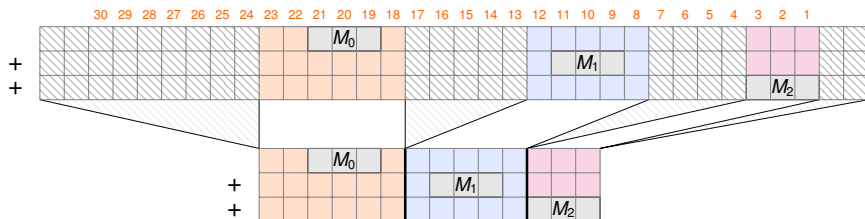
# Compressing the empty bits : Case analysis



This protects against catastrophic cancellation between  $M_0$  and  $M_1$

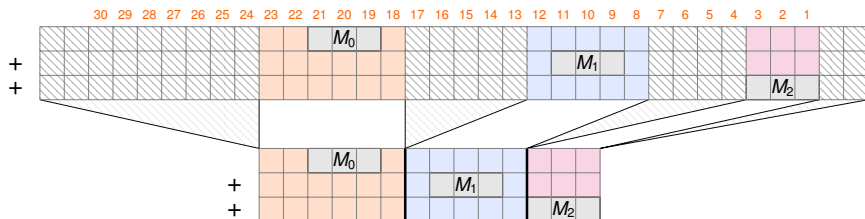


# Compressing the empty bits : Case analysis



Same size adder for all cases.

# Compressing the empty bits : Case analysis



Same size adder for all cases.  
 No combinatory logic on adder.  
 Needs sorting

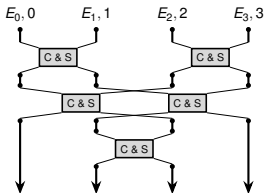
## Sorting $n$ numbers



# How to sort $n$ numbers by exponent value

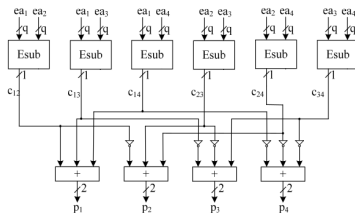
Sorting networks, based on compare and swap operators

- 4 inputs:



- 9 inputs: Parberry (1991)
- 17 inputs: Ehlers et al (2015)

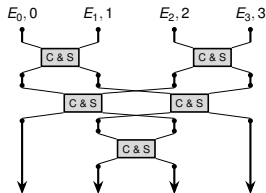
Tao et al  $\rightarrow$  parallel sort :  $\frac{n(n-1)}{2}$  comparisons and bit counting



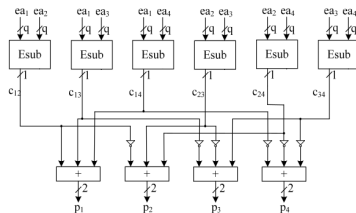
# How to sort $n$ numbers by exponent value

Sorting networks, based on compare and swap operators

- 4 inputs:



Tao et al  $\rightarrow$  parallel sort :  $\frac{n(n-1)}{2}$  comparisons and bit counting



- 9 inputs: Parberry (1991)
- 17 inputs: Ehlers et al (2015)

Tested for  $n \in \{3, 5, 9, 17\}$

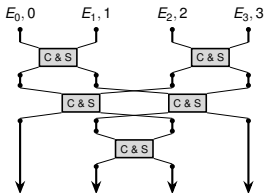
Expected: Tao faster than sorting networks but awful in area



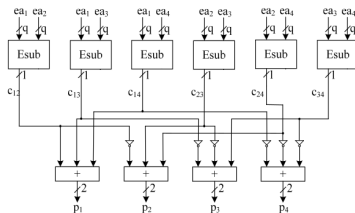
# How to sort $n$ numbers by exponent value

Sorting networks, based on compare and swap operators

- 4 inputs:



Tao et al  $\rightarrow$  parallel sort :  $\frac{n(n-1)}{2}$  comparisons and bit counting



- 9 inputs: Parberry (1991)
- 17 inputs: Ehlers et al (2015)

Tested for  $n \in \{3, 5, 9, 17\}$

Expected: Tao faster than sorting networks but awful in area

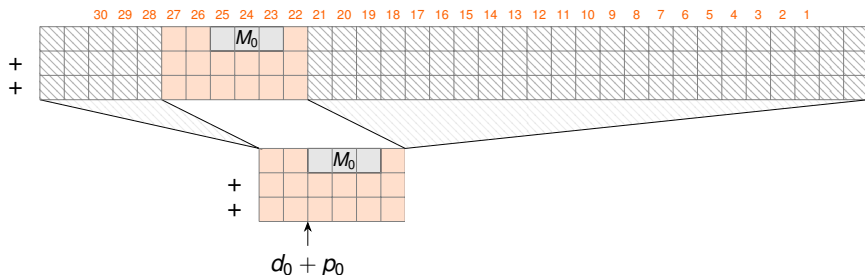
Reality: Tao faster than sorting networks and equivalent in area, especially with timing constraint. We will use this sort



## How to align the significands



# How to compute the shifts

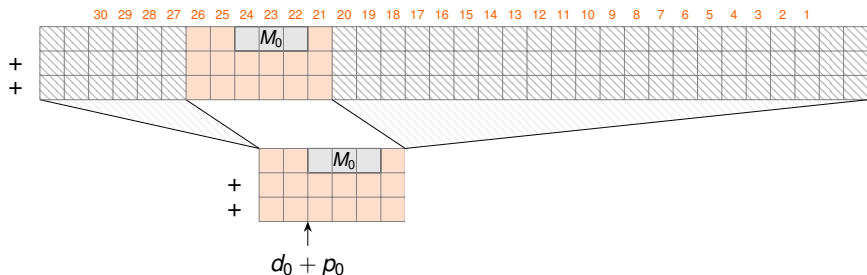


$M_0$  has a constant position

$$S_0 = d_0 + p_0$$



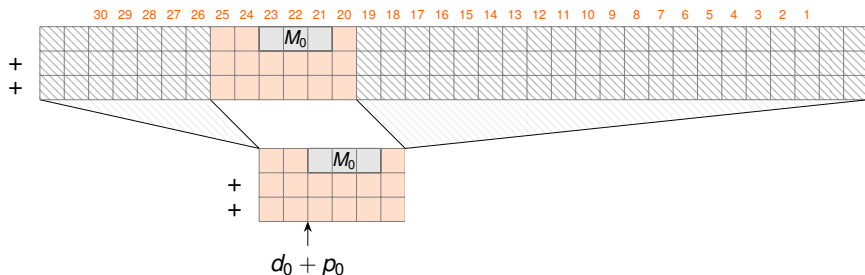
# How to compute the shifts



$M_0$  has a constant position

$$S_0 = d_0 + p_0$$

# How to compute the shifts

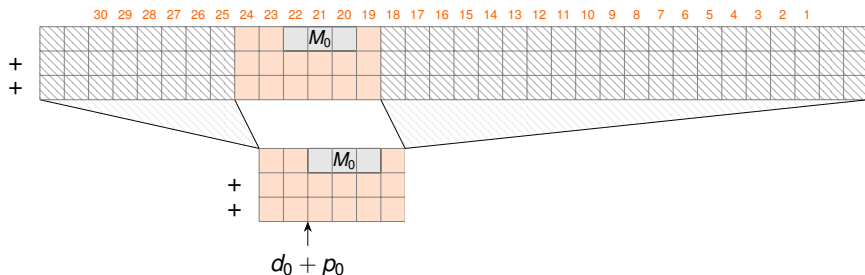


$M_0$  has a constant position

$$S_0 = d_0 + p_0$$



# How to compute the shifts

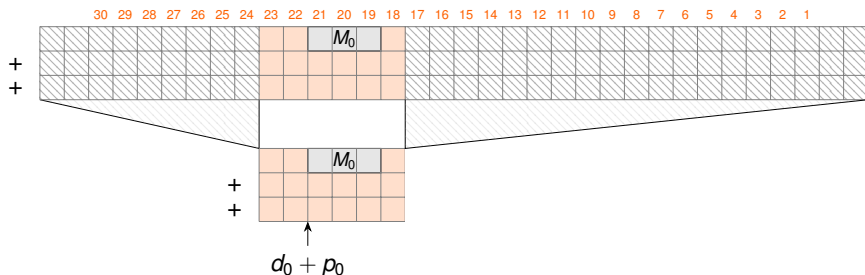


$M_0$  has a constant position

$$S_0 = d_0 + p_0$$



# How to compute the shifts

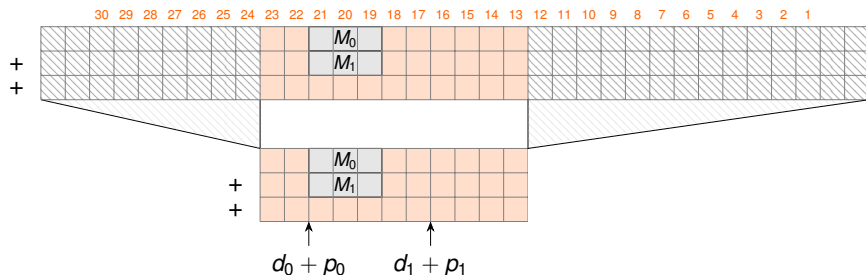


$M_0$  has a constant position

$$S_0 = d_0 + p_0$$



# How to compute the shifts



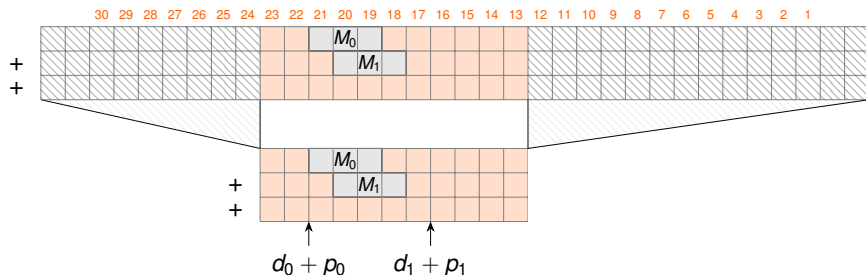
Right "shift"  $M_1$  relative to  $M_0$

$$S_0 = d_0 + p_0$$

$$S_1 = d_0 + p_0 + E_0 - E_1$$



# How to compute the shifts



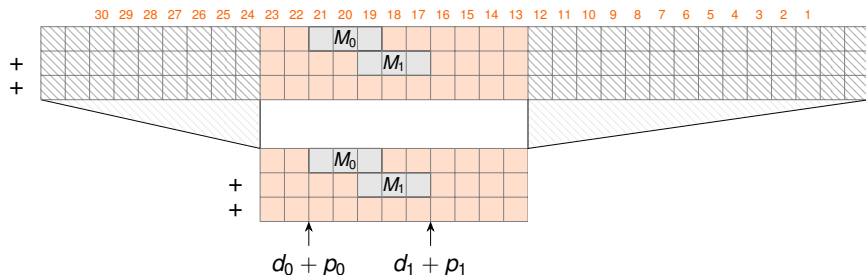
Right "shift"  $M_1$  relative to  $M_0$

$$S_0 = d_0 + p_0$$

$$S_1 = d_0 + p_0 + E_0 - E_1$$



# How to compute the shifts



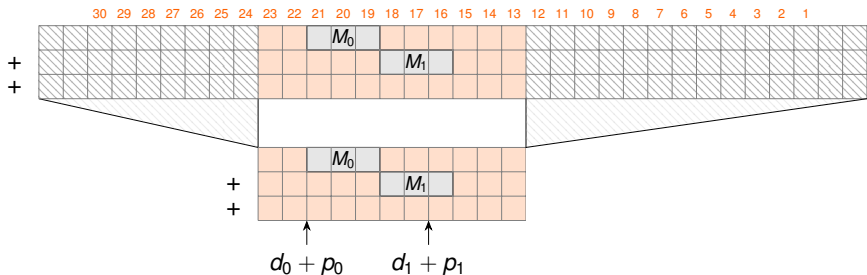
Right "shift"  $M_1$  relative to  $M_0$

$$S_0 = d_0 + p_0$$

$$S_1 = d_0 + p_0 + E_0 - E_1$$



# How to compute the shifts



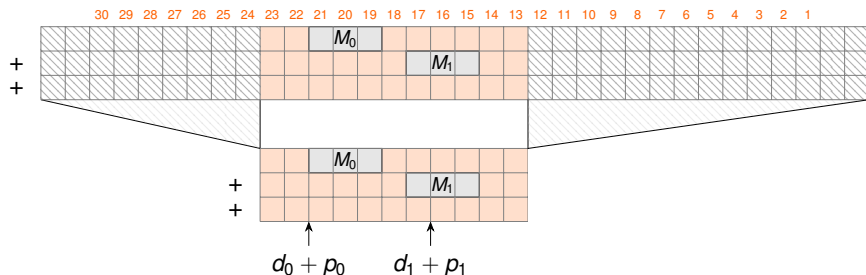
Right "shift"  $M_1$  relative to  $M_0$

$$S_0 = d_0 + p_0$$

$$S_1 = d_0 + p_0 + E_0 - E_1$$



# How to compute the shifts



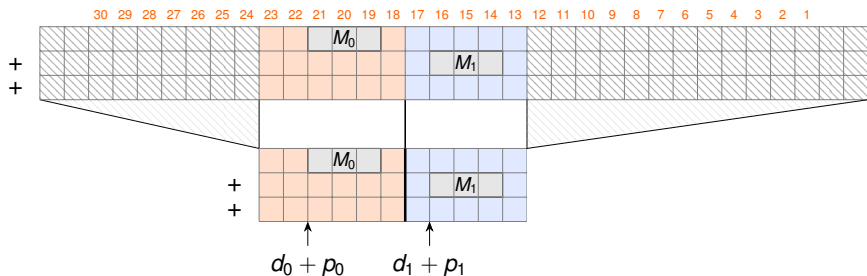
Right "shift"  $M_1$  relative to  $M_0$

$$S_0 = d_0 + p_0$$

$$S_1 = d_0 + p_0 + E_0 - E_1$$



# How to compute the shifts



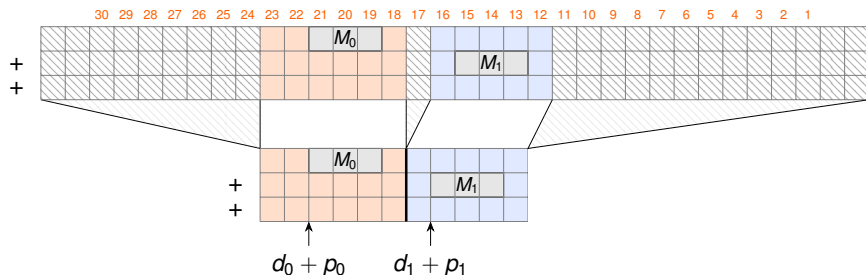
Right "shift"  $M_1$  relative to  $M_0$

$$S_0 = d_0 + p_0$$

$$S_1 = d_1 + p_1$$



# How to compute the shifts



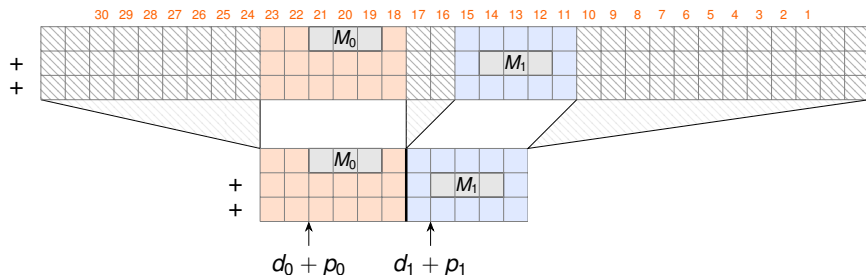
Right "shift"  $M_1$  relative to  $M_0$

$$S_0 = d_0 + p_0$$

$$S_1 = d_1 + p_1$$



# How to compute the shifts



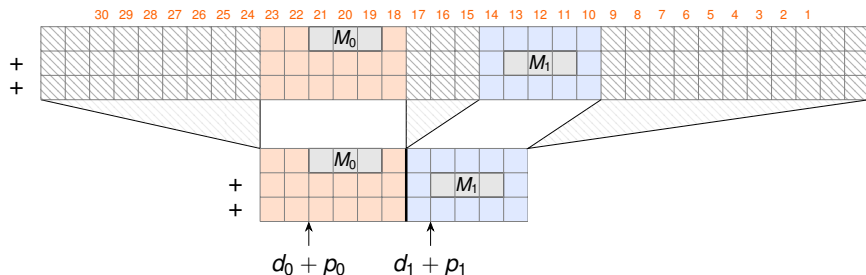
Right "shift"  $M_1$  relative to  $M_0$

$$S_0 = d_0 + p_0$$

$$S_1 = d_1 + p_1$$



# How to compute the shifts



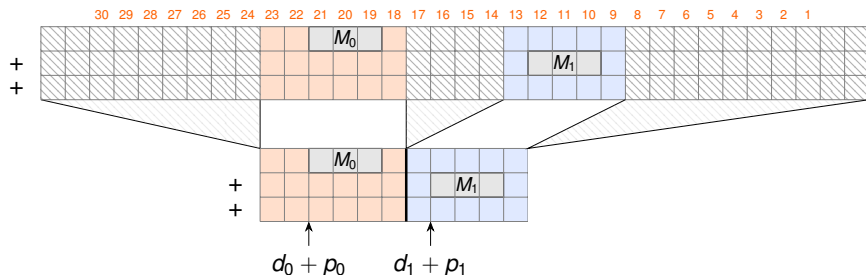
Right "shift"  $M_1$  relative to  $M_0$

$$S_0 = d_0 + p_0$$

$$S_1 = d_1 + p_1$$



# How to compute the shifts



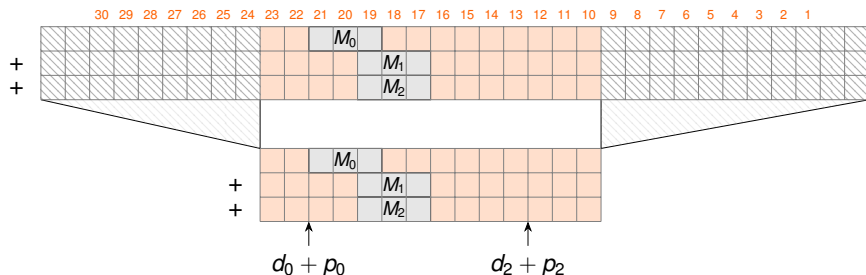
Right "shift"  $M_1$  relative to  $M_0$

$$S_0 = d_0 + p_0$$

$$S_1 = \min(d_0 + p_0 + E_0 - E_1, d_1 + p_1)$$



# How to compute the shifts



If  $M_1$  in zone 0, right "shift"  $M_2$  relative to  $M_0$

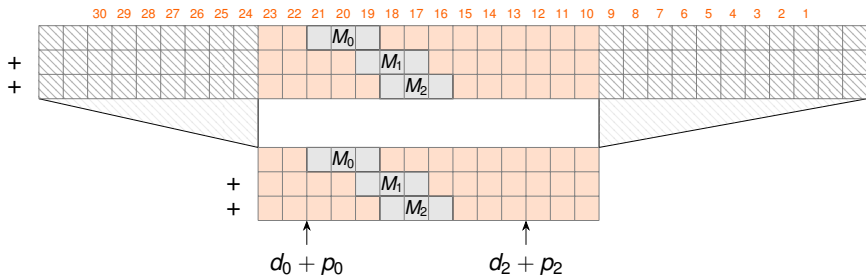
$$S_0 = d_0 + p_0$$

$$S_1 = \min(d_0 + p_0 + E_0 - E_1, d_1 + p_1)$$

$$\text{If } M_1 \text{ in zone 0: } S_2 = d_0 + p_0 + E_0 - E_2$$



# How to compute the shifts



If  $M_1$  in zone 0, right "shift"  $M_2$  relative to  $M_0$

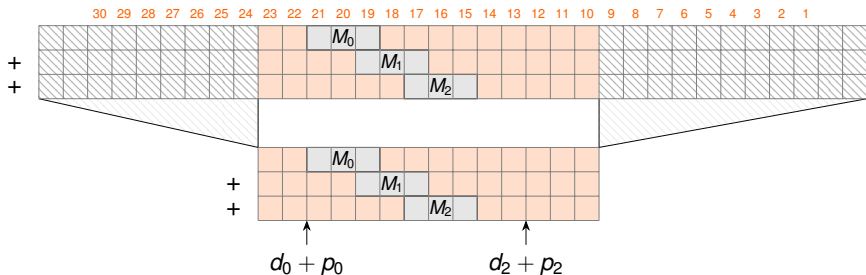
$$S_0 = d_0 + p_0$$

$$S_1 = \min(d_0 + p_0 + E_0 - E_1, d_1 + p_1)$$

$$\text{If } M_1 \text{ in zone 0: } S_2 = d_0 + p_0 + E_0 - E_2$$



# How to compute the shifts



If  $M_1$  in zone 0, right "shift"  $M_2$  relative to  $M_0$

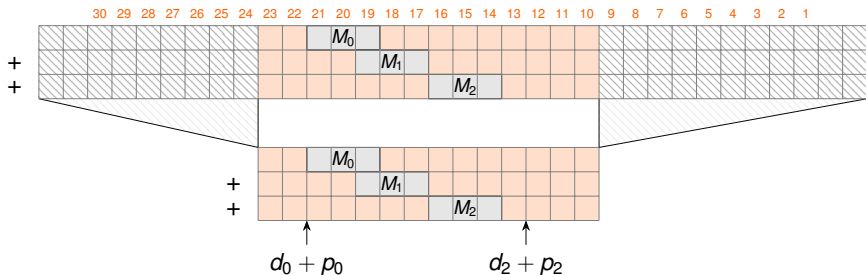
$$S_0 = d_0 + p_0$$

$$S_1 = \min(d_0 + p_0 + E_0 - E_1, d_1 + p_1)$$

$$\text{If } M_1 \text{ in zone 0: } S_2 = d_0 + p_0 + E_0 - E_2$$



# How to compute the shifts



If  $M_1$  in zone 0, right "shift"  $M_2$  relative to  $M_0$

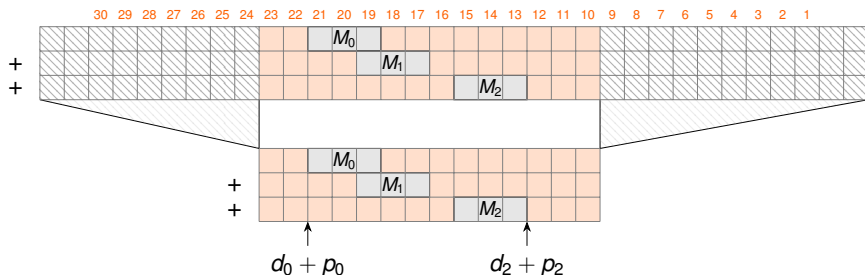
$$S_0 = d_0 + p_0$$

$$S_1 = \min(d_0 + p_0 + E_0 - E_1, d_1 + p_1)$$

$$\text{If } M_1 \text{ in zone 0: } S_2 = d_0 + p_0 + E_0 - E_2$$



# How to compute the shifts



If  $M_1$  in zone 0, right "shift"  $M_2$  relative to  $M_0$

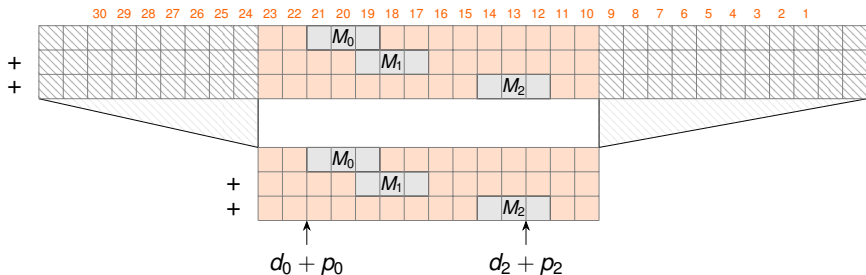
$$S_0 = d_0 + p_0$$

$$S_1 = \min(d_0 + p_0 + E_0 - E_1, d_1 + p_1)$$

$$\text{If } M_1 \text{ in zone 0: } S_2 = d_0 + p_0 + E_0 - E_2$$



# How to compute the shifts



If  $M_1$  in zone 0, right "shift"  $M_2$  relative to  $M_0$

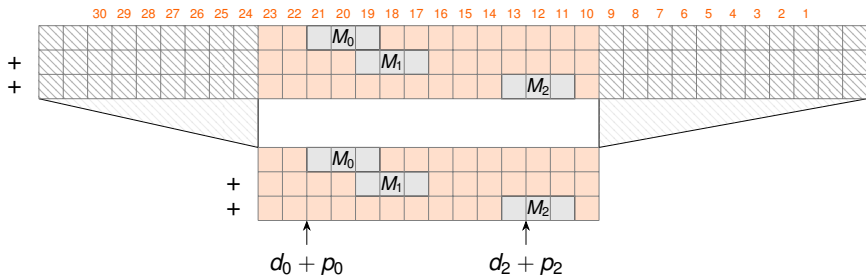
$$S_0 = d_0 + p_0$$

$$S_1 = \min(d_0 + p_0 + E_0 - E_1, d_1 + p_1)$$

$$\text{If } M_1 \text{ in zone 0: } S_2 = d_0 + p_0 + E_0 - E_2$$



# How to compute the shifts



If  $M_1$  in zone 0, right "shift"  $M_2$  relative to  $M_0$

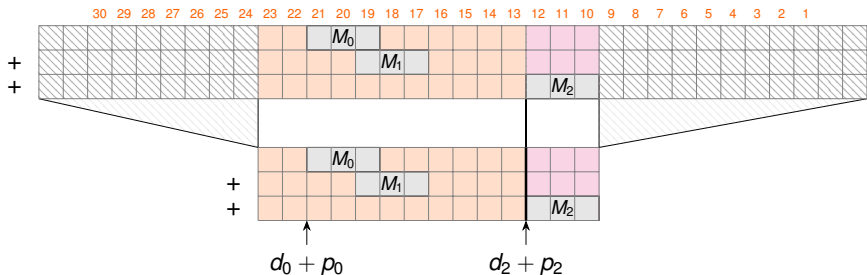
$$S_0 = d_0 + p_0$$

$$S_1 = \min(d_0 + p_0 + E_0 - E_1, d_1 + p_1)$$

$$\text{If } M_1 \text{ in zone 0: } S_2 = d_0 + p_0 + E_0 - E_2$$



# How to compute the shifts



If  $M_1$  in zone 0, right "shift"  $M_2$  relative to  $M_0$

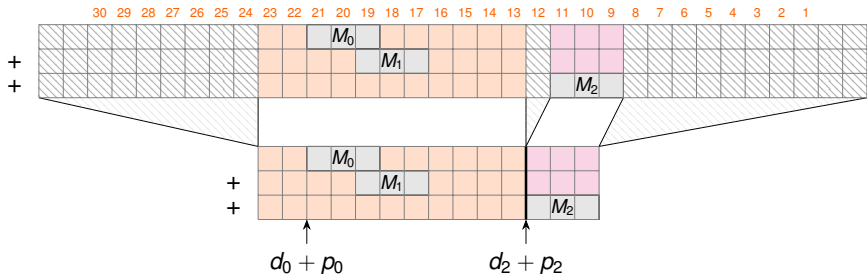
$$S_0 = d_0 + p_0$$

$$S_1 = \min(d_0 + p_0 + E_0 - E_1, d_1 + p_1)$$

If  $M_1$  in zone 0:  $S_2 = d_2 + p_2$



# How to compute the shifts



If  $M_1$  in zone 0, right "shift"  $M_2$  relative to  $M_0$

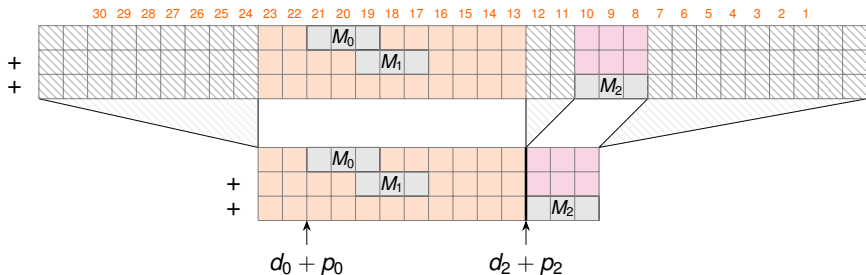
$$S_0 = d_0 + p_0$$

$$S_1 = \min(d_0 + p_0 + E_0 - E_1, d_1 + p_1)$$

If  $M_1$  in zone 0:  $S_2 = d_2 + p_2$



# How to compute the shifts



If  $M_1$  in zone 0, right "shift"  $M_2$  relative to  $M_0$

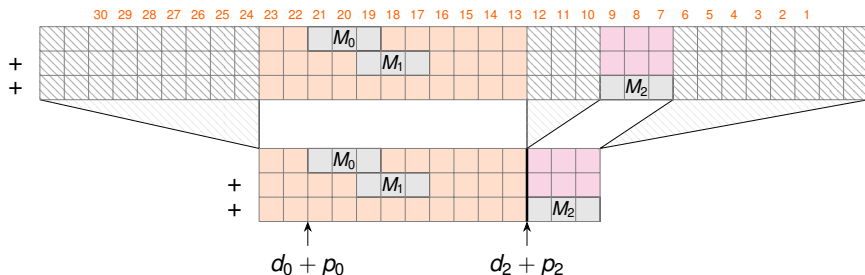
$$S_0 = d_0 + p_0$$

$$S_1 = \min(d_0 + p_0 + E_0 - E_1, d_1 + p_1)$$

If  $M_1$  in zone 0:  $S_2 = d_2 + p_2$



# How to compute the shifts



If  $M_1$  in zone 0, right "shift"  $M_2$  relative to  $M_0$

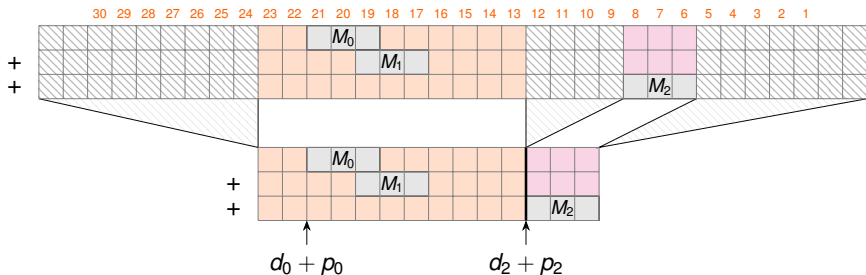
$$S_0 = d_0 + p_0$$

$$S_1 = \min(d_0 + p_0 + E_0 - E_1, d_1 + p_1)$$

If  $M_1$  in zone 0:  $S_2 = d_2 + p_2$



# How to compute the shifts



If  $M_1$  in zone 0, right "shift"  $M_2$  relative to  $M_0$

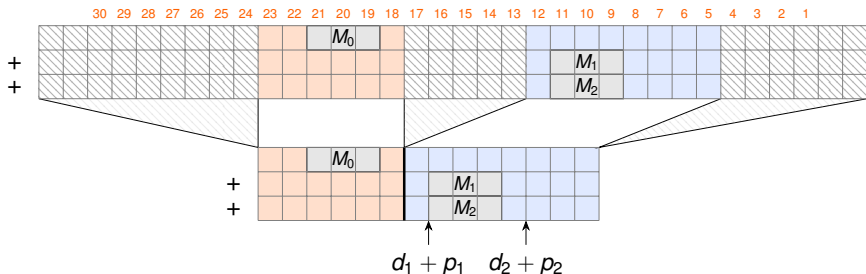
$$S_0 = d_0 + p_0$$

$$S_1 = \min(d_0 + p_0 + E_0 - E_1, d_1 + p_1)$$

$$\text{If } M_1 \text{ in zone 0: } S_2 = \min(d_0 + p_0 + E_0 - E_2, d_2 + p_2)$$



# How to compute the shifts



If  $M_1$  in zone 1, right "shift"  $M_2$  relative to  $M_1$

$$S_0 = d_0 + p_0$$

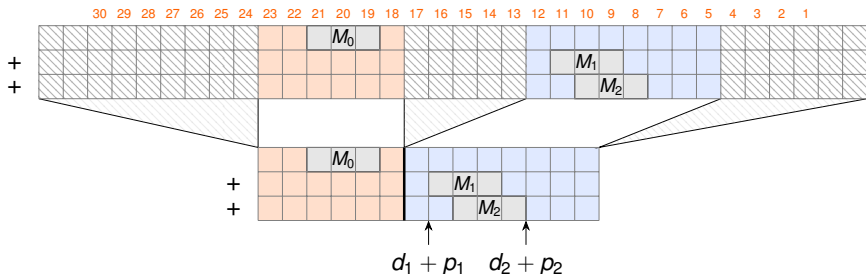
$$S_1 = \min(d_0 + p_0 + E_0 - E_1, d_1 + p_1)$$

If  $M_1$  in zone 0:  $S_2 = \min(d_0 + p_0 + E_0 - E_2, d_2 + p_2)$

Else ( $M_1$  in zone 1):  $S_2 = d_1 + p_1 + E_1 - E_2$



# How to compute the shifts



If  $M_1$  in zone 1, right "shift"  $M_2$  relative to  $M_1$

$$S_0 = d_0 + p_0$$

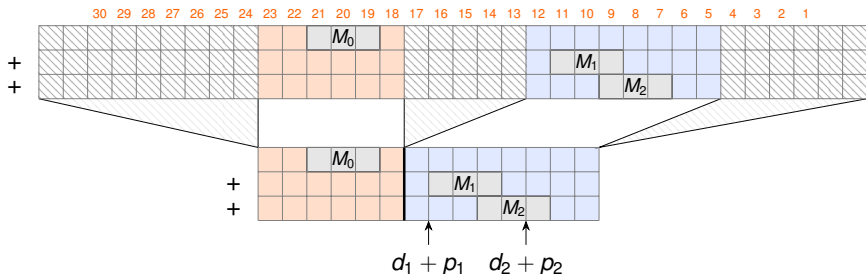
$$S_1 = \min(d_0 + p_0 + E_0 - E_1, d_1 + p_1)$$

If  $M_1$  in zone 0:  $S_2 = \min(d_0 + p_0 + E_0 - E_2, d_2 + p_2)$

Else ( $M_1$  in zone 1):  $S_2 = d_1 + p_1 + E_1 - E_2$



# How to compute the shifts



If  $M_1$  in zone 1, right "shift"  $M_2$  relative to  $M_1$

$$S_0 = d_0 + p_0$$

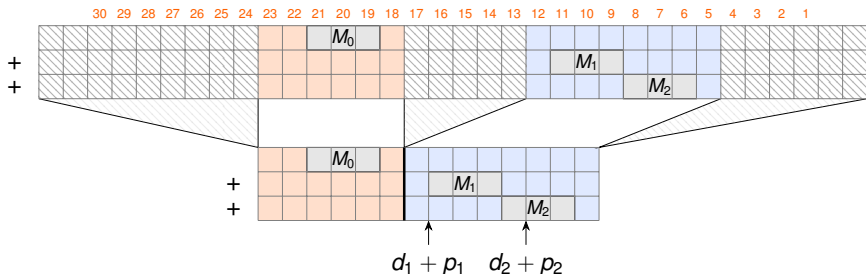
$$S_1 = \min(d_0 + p_0 + E_0 - E_1, d_1 + p_1)$$

If  $M_1$  in zone 0:  $S_2 = \min(d_0 + p_0 + E_0 - E_2, d_2 + p_2)$

Else ( $M_1$  in zone 1):  $S_2 = d_1 + p_1 + E_1 - E_2$



# How to compute the shifts



If  $M_1$  in zone 1, right "shift"  $M_2$  relative to  $M_1$

$$S_0 = d_0 + p_0$$

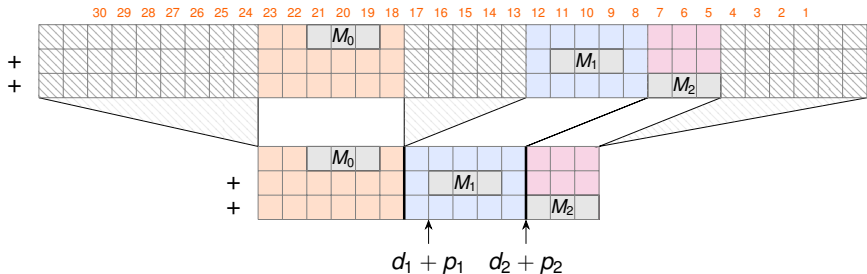
$$S_1 = \min(d_0 + p_0 + E_0 - E_1, d_1 + p_1)$$

If  $M_1$  in zone 0:  $S_2 = \min(d_0 + p_0 + E_0 - E_2, d_2 + p_2)$

Else ( $M_1$  in zone 1):  $S_2 = d_1 + p_1 + E_1 - E_2$



# How to compute the shifts



If  $M_1$  in zone 1, right "shift"  $M_2$  relative to  $M_1$

$$S_0 = d_0 + p_0$$

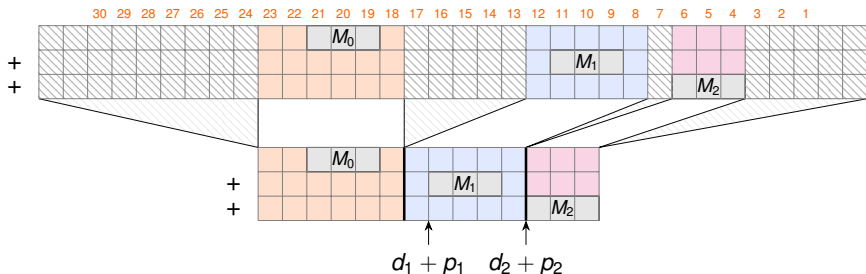
$$S_1 = \min(d_0 + p_0 + E_0 - E_1, d_1 + p_1)$$

If  $M_1$  in zone 0:  $S_2 = \min(d_0 + p_0 + E_0 - E_2, d_2 + p_2)$

Else ( $M_1$  in zone 1):  $S_2 = d_2 + p_2$



# How to compute the shifts



If  $M_1$  in zone 1, right "shift"  $M_2$  relative to  $M_1$

$$S_0 = d_0 + p_0$$

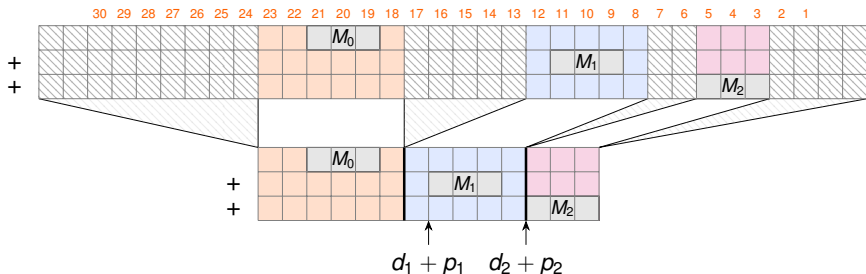
$$S_1 = \min(d_0 + p_0 + E_0 - E_1, d_1 + p_1)$$

If  $M_1$  in zone 0:  $S_2 = \min(d_0 + p_0 + E_0 - E_2, d_2 + p_2)$

Else ( $M_1$  in zone 1):  $S_2 = d_2 + p_2$



# How to compute the shifts



If  $M_1$  in zone 1, right "shift"  $M_2$  relative to  $M_1$

$$S_0 = d_0 + p_0$$

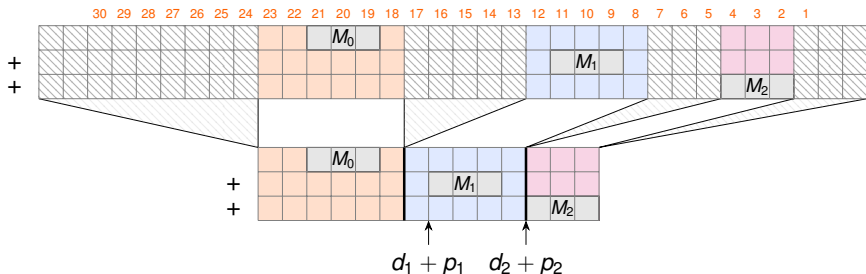
$$S_1 = \min(d_0 + p_0 + E_0 - E_1, d_1 + p_1)$$

$$\text{If } M_1 \text{ in zone 0: } S_2 = \min(d_0 + p_0 + E_0 - E_2, d_2 + p_2)$$

$$\text{Else } (M_1 \text{ in zone 1): } S_2 = d_2 + p_2$$



# How to compute the shifts



If  $M_1$  in zone 1, right "shift"  $M_2$  relative to  $M_1$

$$S_0 = d_0 + p_0$$

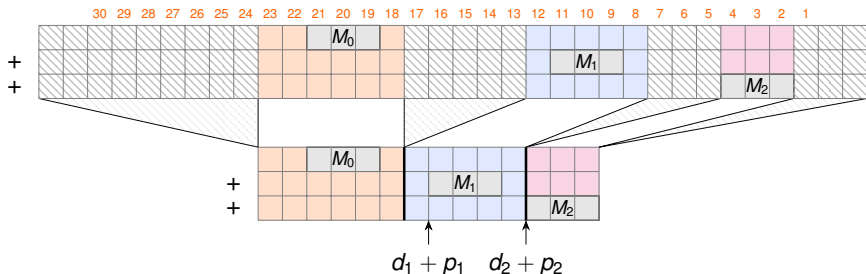
$$S_1 = \min(d_0 + p_0 + E_0 - E_1, d_1 + p_1)$$

If  $M_1$  in zone 0:  $S_2 = \min(d_0 + p_0 + E_0 - E_2, d_2 + p_2)$

Else ( $M_1$  in zone 1):  $S_2 = \min(d_1 + p_1 + E_1 - E_2, d_2 + p_2)$



# How to compute the shifts



$$S_0 = d_0 + p_0$$

$$S_1 = \min(d_0 + p_0 + E_0 - E_1, d_1 + p_1)$$

$$\text{If } M_1 \text{ in zone 0: } S_2 = \min(d_0 + p_0 + E_0 - E_2, d_2 + p_2)$$

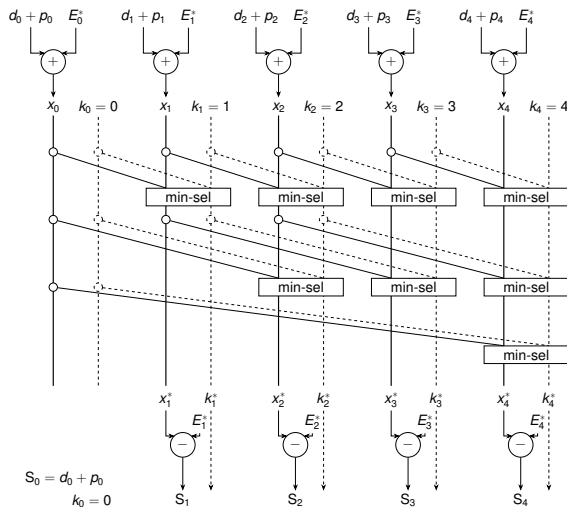
$$\text{Else } (M_1 \text{ in zone 1): } S_2 = \min(d_1 + p_1 + E_1 - E_2, d_2 + p_2)$$

...

$\Rightarrow S_i$  has  $i$  cases depending on the position of  $M_{i-1}$



# How to compute the shifts with parallel prefix

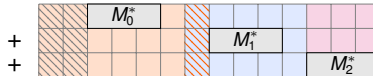


The ugly details are in the paper

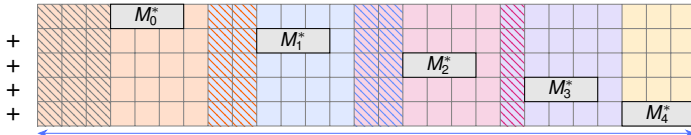


# Actual size of shifters and adders

$$N = 2$$



$$N = 4$$

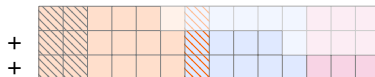


$$w_{\text{compressed}} = \sum_{i=0}^{N-1} (\lceil \log_2(N+1-i) \rceil + w + 1) + w$$

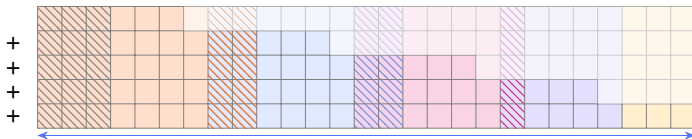


# Actual size of shifters and adders

$$N = 2$$



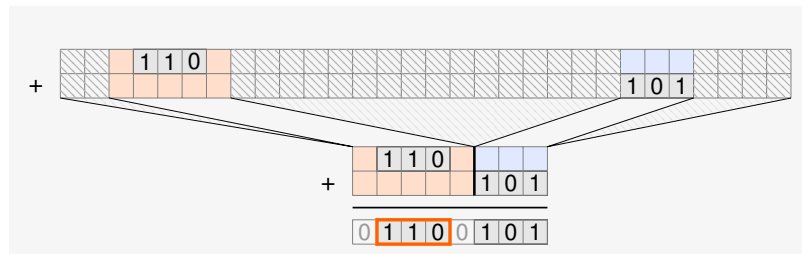
$$N = 4$$



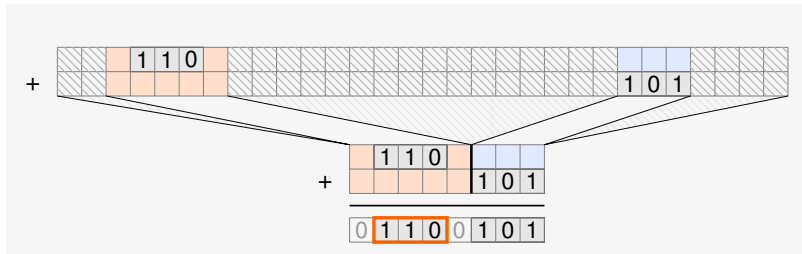
$$w_{\text{compressed}} = \sum_{i=0}^{N-1} (\lceil \log_2(N+1-i) \rceil + w + 1) + w$$



# Result retrieval and subnormal special case



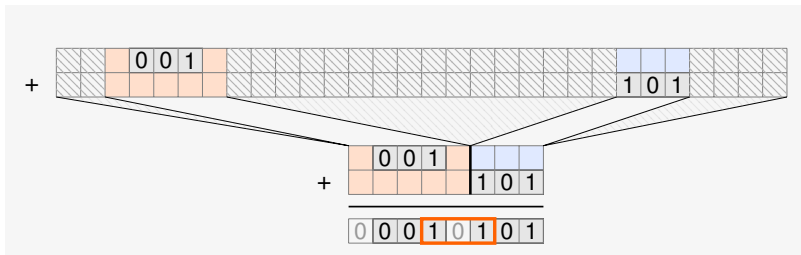
# Result retrieval and subnormal special case



Avoid at all costs :  $R$  has significand bits from two different zones



## Result retrieval and subnormal special case



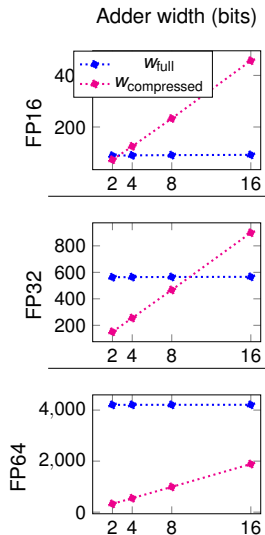
Avoid at all costs :  $R$  has significand bits from two different zones  
Subnormals: We don't know where the real first bit of  $M$  is

# Results

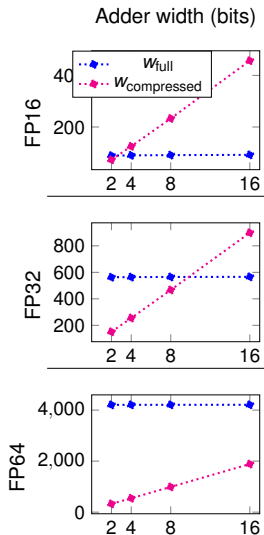


# Results

Reminder :  $w_{\text{full}} \sim 2^e$  and  
 $w_{\text{compressed}} \sim N \times m$



# Results



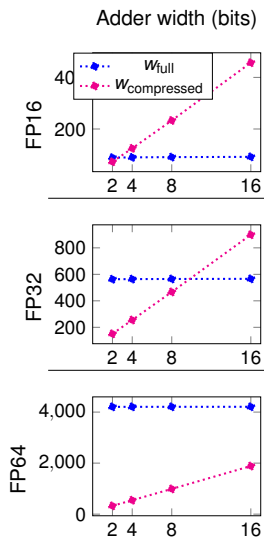
Reminder :  $w_{full} \sim 2^e$  and  
 $w_{compressed} \sim N \times m$

3 situations :

- The format has a lot of precision compared to the range,  
 $w_{compressed} > w_{full}$  even for small  $N$



# Results



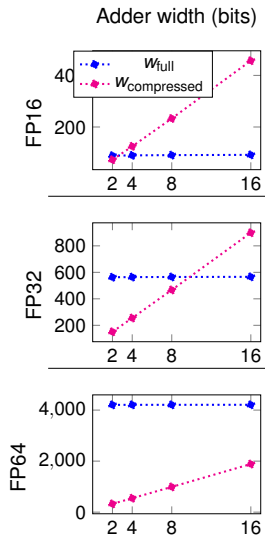
Reminder :  $w_{full} \sim 2^e$  and  
 $w_{compressed} \sim N \times m$

3 situations :

- The format has a lot of precision compared to the range,  
 $w_{compressed} > w_{full}$  even for small  $N$
- The format is more balanced,  
 $w_{compressed} < w_{full}$  for small  $N$



# Results



Reminder :  $w_{full} \sim 2^e$  and  
 $w_{compressed} \sim N \times m$

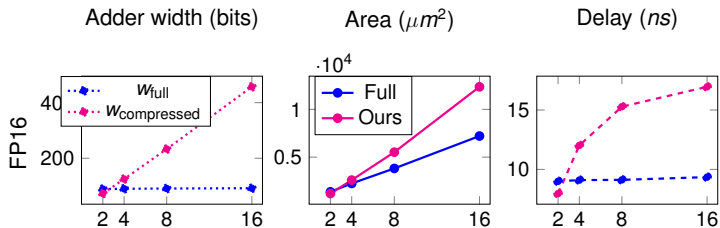
3 situations :

- The format has a lot of precision compared to the range,  
 $w_{compressed} > w_{full}$  even for small  $N$
- The format is more balanced,  
 $w_{compressed} < w_{full}$  for small  $N$
- The format has a lot of range compared to the precision  
 $w_{compressed} < w_{full}$  until larger  $N$



# Results

$$W_{\text{compressed}} > W_{\text{full}}$$



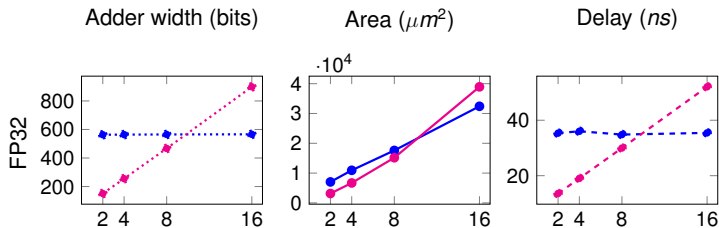
Compressing better for  $N \leq 2$

Synthesized with the Synopsys Design Compiler NXT for the TSMC 16FFC node.



# Results

$$W_{\text{compressed}} \sim W_{\text{full}}$$



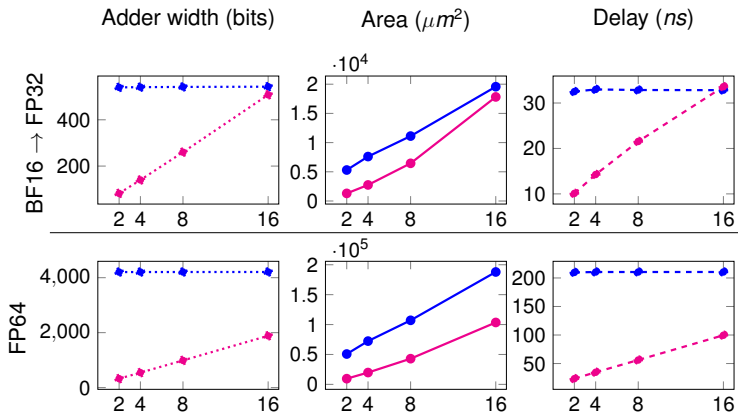
Compressing better for  $N \leq 8$

Synthesized with the Synopsys Design Compiler NXT for the TSMC 16FFC node.



# Results

$$W_{\text{compressed}} < W_{\text{full}}$$



Compressing is interesting for studied  $N$



# Conclusion

- Comparison of two different architectures



# Conclusion

- Comparison of two different architectures
- Improvements on Tao's compressed architecture
  - Parallel prefix
  - Subnormal studies



# Conclusion

- Comparison of two different architectures
- Improvements on Tao's compressed architecture
  - Parallel prefix
  - Subnormal studies
- Results that are not obvious
  - Neural networks: large  $N$ , high precision:range ratio  $\rightarrow$  Kulisch
  - Complex numbers:  $N = 2$ , FP32 or FP64  $\rightarrow$  Compressed
  - Graphic computations:  $N = 4$ ,
    - if FP16  $\rightarrow$  Kulisch
    - if FP32  $\rightarrow$  Compressed



# Conclusion

- Comparison of two different architectures
- Improvements on Tao's compressed architecture
  - Parallel prefix
  - Subnormal studies
- Results that are not obvious
  - Neural networks: large  $N$ , high precision:range ratio  $\rightarrow$  Kulisch
  - Complex numbers:  $N = 2$ , FP32 or FP64  $\rightarrow$  Compressed
  - Graphic computations:  $N = 4$ ,
    - if FP16  $\rightarrow$  Kulisch
    - if FP32  $\rightarrow$  Compressed

Some of these operators will be added in Kalray's 4th generation MPPA



# Biblio

U. W. Kulisch (2002)

Advanced Arithmetic for the Digital Computer: Design of Arithmetic Units

*Springer-Verlag*

Tao, G. Deyuan, F. Xiaoya, and J. Nurmi (2013)

Correctly rounded architectures for floating-point multi-operand addition and dot-product computation

*IEEE 24th Int. Conf. on Application-Specific Systems, Architectures and Processors*  
pp. 346–355

I. Parberry (1991)

A computer-assisted optimal depth lower bound for nine- input sorting networks

*Mathematical systems theory*, vol. 24, no. 1, pp. 101–116

T. Ehlers and M. Müller (2015)

New bounds on optimal sorting networks

*Evolving Computability: 11th Conf. on Computability in Europe*, Proceedings 11, pp. 167–176

