Modified FMA for exact accumulation of low precision products

RAIM 2017, Nicolas Brunie
(nbrunie@kalray.eu)

October 26th, 2017

www.kalrayinc.com

Accurate accumulation of products of small precision numbers

Goal: Assuming x_i, y_j **binary16** and S **binary32** or larger, **optimize**
$\mathbf{S = [x_0,x_1,x_2,x_3,...].[y_0,y_1,y_2,y_3,...] = x_0.y_0 + x_1.y_1 + x_2.y_2 + x_3.y_3 + ...}$

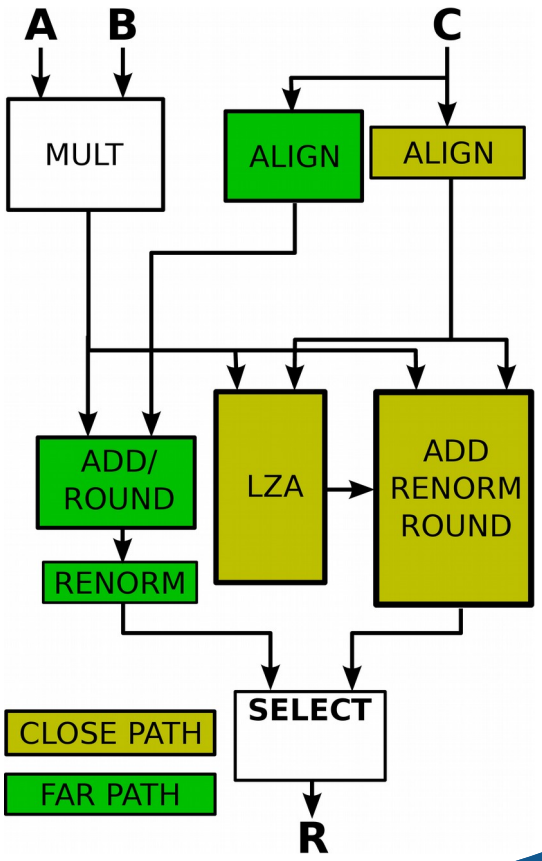
- **binary16** floating-point precision
  - Introduced in IEEE754-2008
  - As a storage format not intended for computation
  - But more and more used in computation
- Problematic :
  - Optimize accuracy
  - Optimize speed (latency and throughput)
  - Suggest a generic processor operator
- Suggestion: extend FMA to smaller precisions
  - Is there a way to exploit smaller precision ?
  - Is there a way to easily extend FMA precision ?
- Design a fast and small operator
  - How to implement low latency accumulation ?

Outline

1. Already available solutions
    1. Fused Multiply-Add
    2. Mixed Precision FMA (Generalized FP addition)
2. New design: revisiting Kulisch's accumulator
3. Metalibm and experimental results
4. Conclusion and perspectives

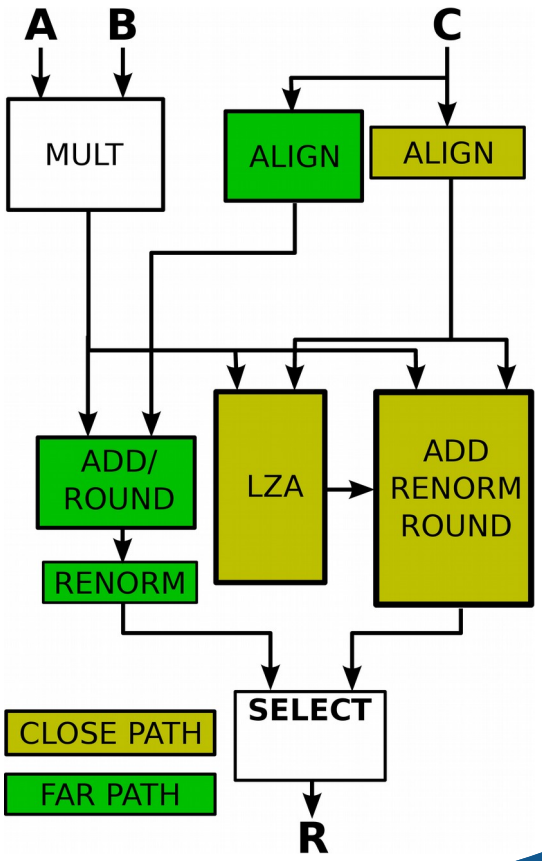
1st solution: Fused Multiply-Add

- Common operator
- Basic block for accumulaion
- Lots of literature
  - Focusing on binary32 and binary64
  - Architecture optimized for latency



1st solution: Fused Multiply-Add

- Common operator
- Basic block for accumulaion
- Lots of literature
  - Focusing on binary32 and binary64
  - Architecture optimized for latency
  - Several cycles for dependent accumulation
  - A few works on throughput optimization [2]



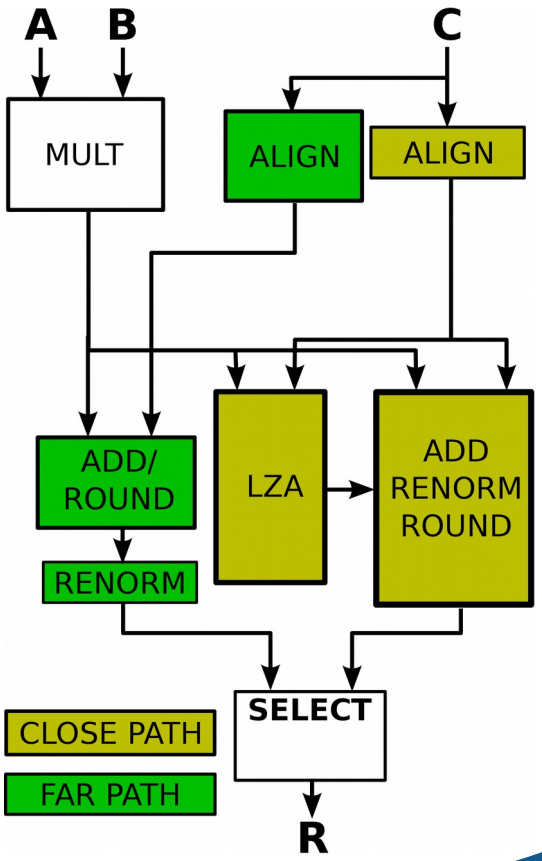
CPU	ARM A72	AMD Bulldozer	Intel Skylake
FMA latency	6/3	5	4

- [2] Fused Multiply-Add Microarchitecture Comprising Separate Early-Normalizing Multiply and Add Pipelines, David Lutz, 2011

1st solution: Fused Multiply-Add

- Common operator
- Basic block for accumulaion
- Lots of literature
  - Focusing on binary32 and binary64
  - Architecture optimized for latency
  - Several cycles for dependent accumulation
  - A few works on throughput optimization [2]
- A few drawbacks (accuracy and latency)

CPU	ARM A72	AMD Bulldozer	Intel Skylake
FMA latency	6/3	5	4



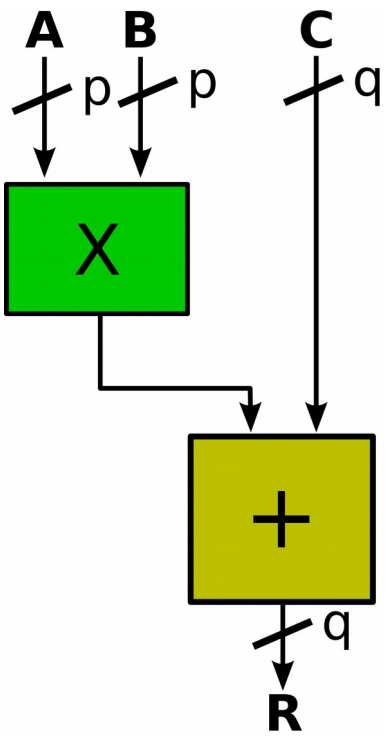
- [2] Fused Multiply-Add Microarchitecture Comprising Separate Early-Normalizing Multiply and Add Pipelines, David Lutz, 2011

2nd solution: Mixed precision FMA

2nd solution: Mixed precision FMA

- FMA with heterogeneous operands

  binary16 . binary16 + binary32 → binary32

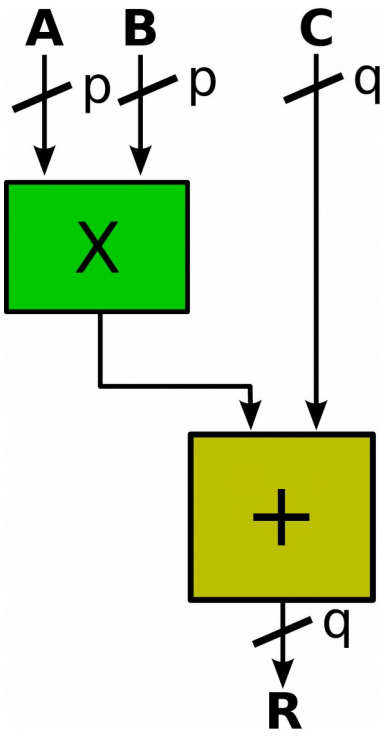


2nd solution: Mixed precision FMA

- FMA with heterogeneous operands

  binary16 . binary16 + binary32 → binary32

- Merging conversion and FMA
  - Saving conversion instructions
  - IEEE754-compliant (formatOf)
  - Compromise between large and small FMA
    - Small multiplier
    - Large alignment and adder

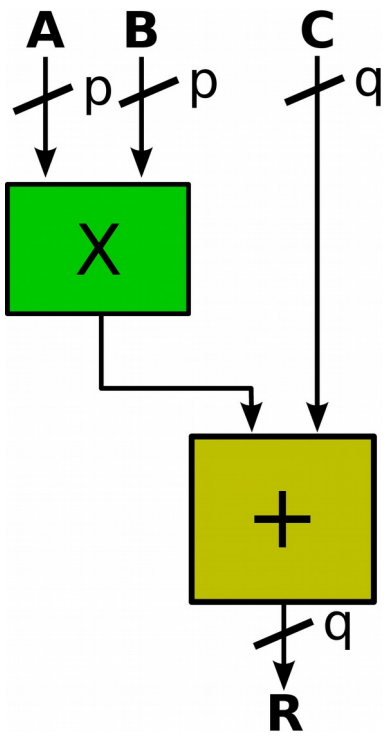


2nd solution: Mixed precision FMA

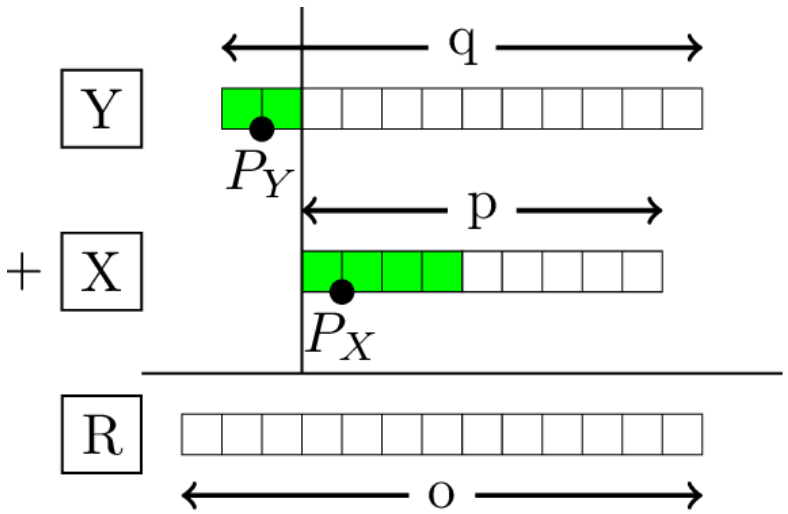
- FMA with heterogeneous operands

  binary16 . binary16 + binary32 → binary32

- Merging conversion and FMA
  - Saving conversion instructions
  - IEEE754-compliant (formatOf)
  - Compromise between large and small FMA
    - Small multiplier
    - Large alignment and adder
- Some specificities
  - Cancellation requirements
  - Datapath design

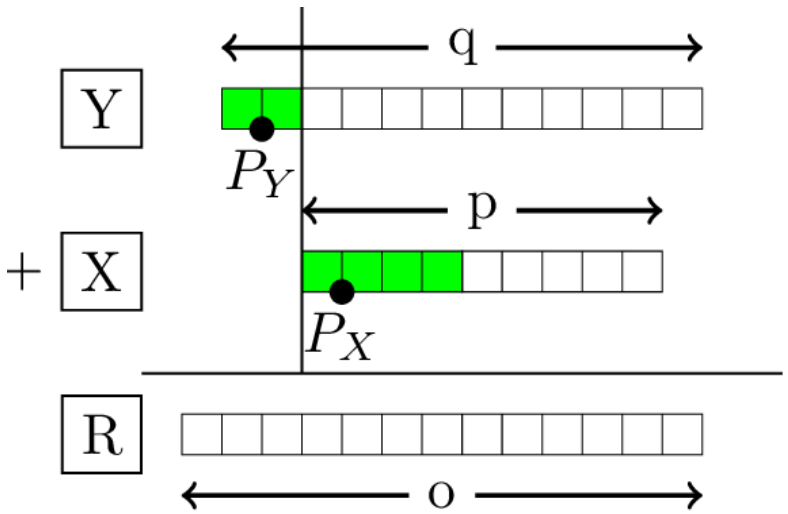


Generalized FP addition (1/4)



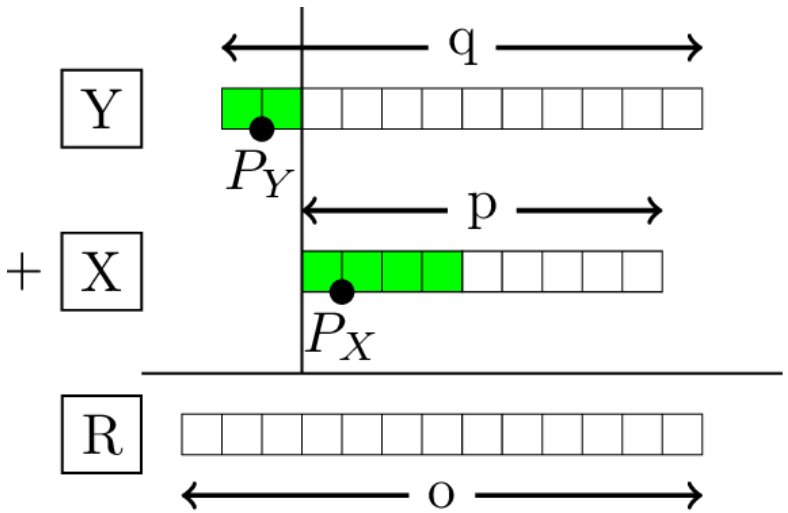
Generalized FP addition (1/4)

- Operator size related to datapath



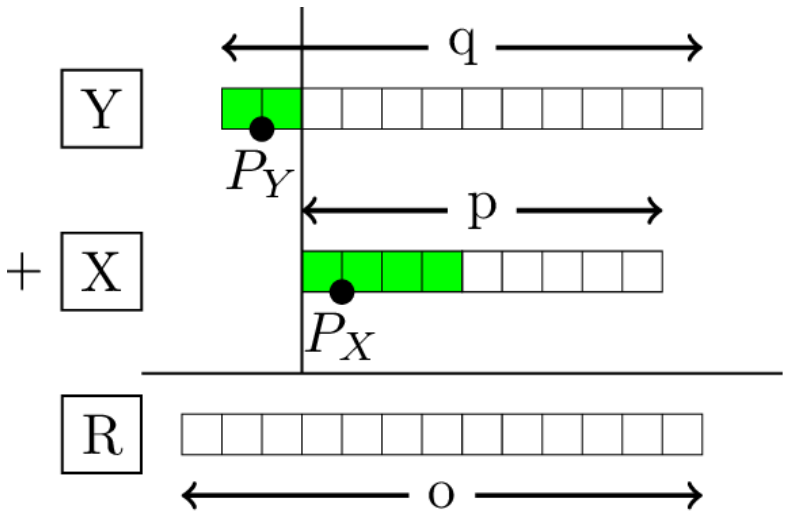
Generalized FP addition (1/4)

- Operator size related to datapath
- Computing X + Y
  - X with precision p and anchor at Px
  - Y with precision q and anchor at Py
  - Arbitrary number of leading zeros
  - Output precision o (normalized)



Generalized FP addition (1/4)

- Operator size related to datapath
- Computing X + Y
  - X with precision p and anchor at Px
  - Y with precision q and anchor at Py
  - Arbitrary number of leading zeros
  - Output precision o (normalized)
- What is the minimal datapath size ?
  - To compute R=o(X + Y) correctly rounded
  - Assuming single path
  - Assuming up to L_X leading zero(s) in X
  - Assuming up to L_Y leading zero(s) in Y



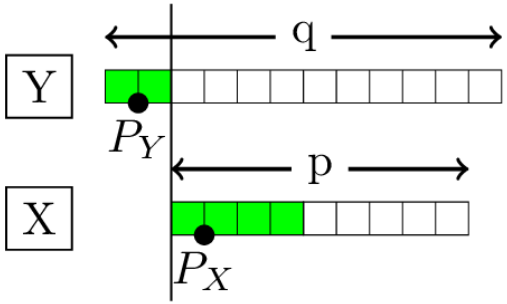
Generalized FP addition (2/4)

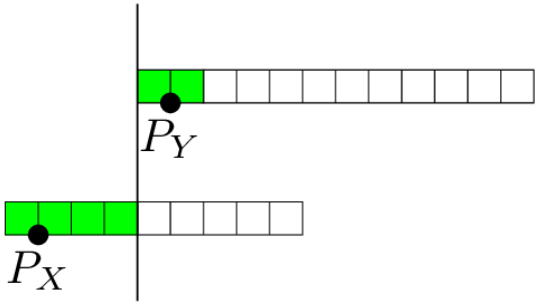
Generalized FP addition (2/4)

- 1st case: large cancellation
  - Determines the Leading Zero Count range
  - Determines the close path topology

Generalized FP addition (2/4)

- 1st case: large cancellation
  - Determines the Leading Zero Count range
  - Determines the close path topology

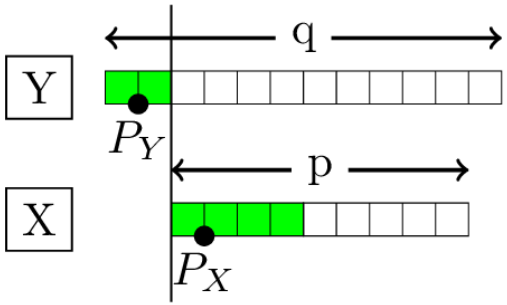


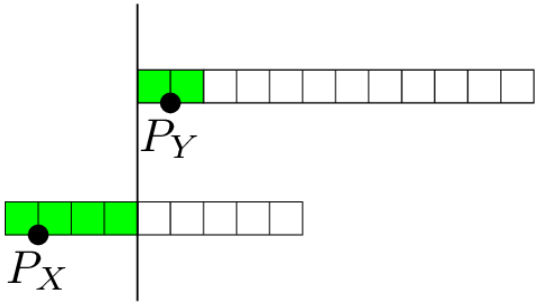


Generalized FP addition (2/4)

- 1st case: large cancellation
  - Determines the Leading Zero Count range
  - Determines the close path topology
- Cancellation occurs *iff*:

$$-(L_Y + 1) \leq \delta = e_X - e_Y \leq L_X + 1$$





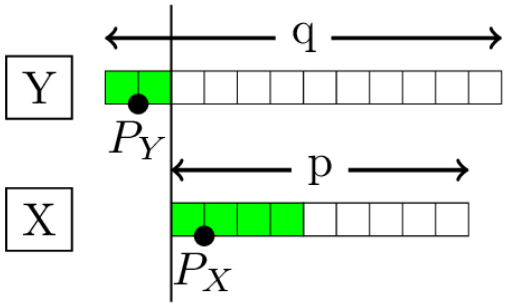
Generalized FP addition (2/4)

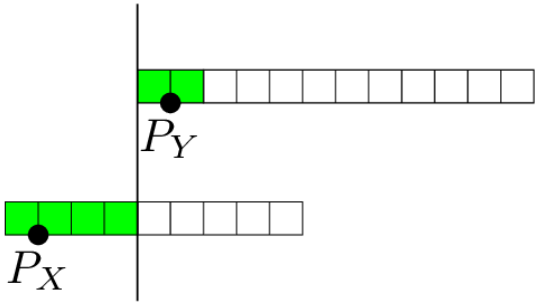
- 1st case: large cancellation
  - Determines the Leading Zero Count range
  - Determines the close path topology
- Cancellation occurs *iff*:

$$-(L_Y + 1) \leq \delta = e_X - e_Y \leq L_X + 1$$

- Leading Zero Counter requirements:

$$\max(L_X + 1 + q, L_Y + 1 + p)$$





Generalized FP addition (2/4)

- 1st case: large cancellation
  - Determines the Leading Zero Count range
  - Determines the close path topology
- Cancellation occurs *iff*:

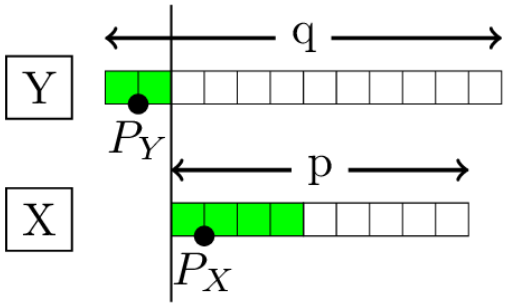
$$-(L_Y + 1) \leq \delta = e_X - e_Y \leq L_X + 1$$

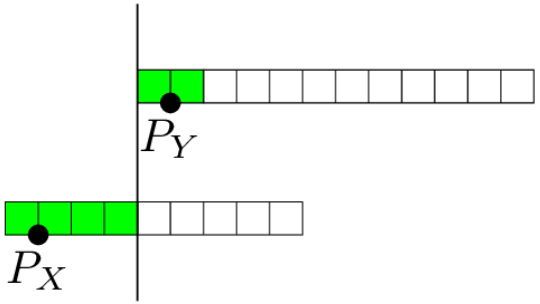
- Leading Zero Counter requirements:

$$\max(L_X + 1 + q, L_Y + 1 + p)$$

- Adder requirements:

$$\max(L_X + 1 + q, L_Y + 1 + p)$$



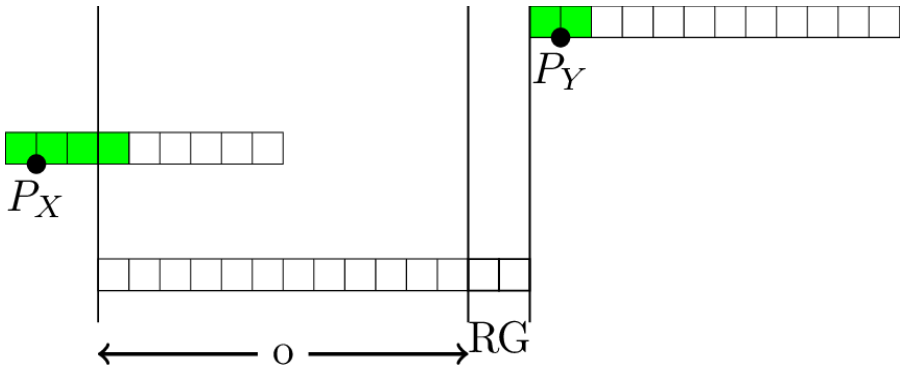


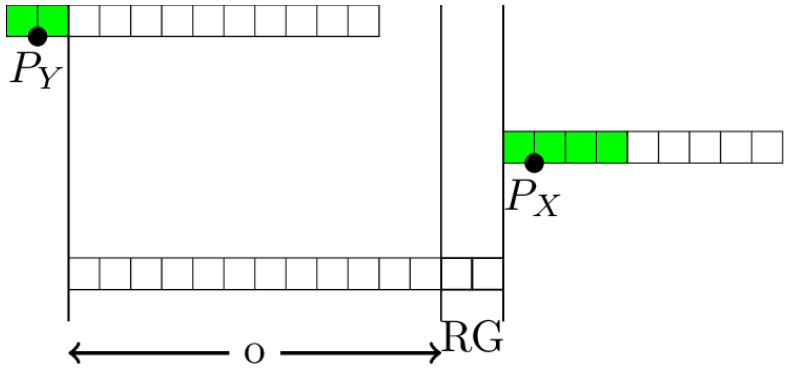
Generalized FP addition (3/4)

Generalized FP addition (3/4)

- Second case: extremal aligment
  - Determines datapath width
  - Exhibits effect of non-normalization
  - Two sub cases to be considered

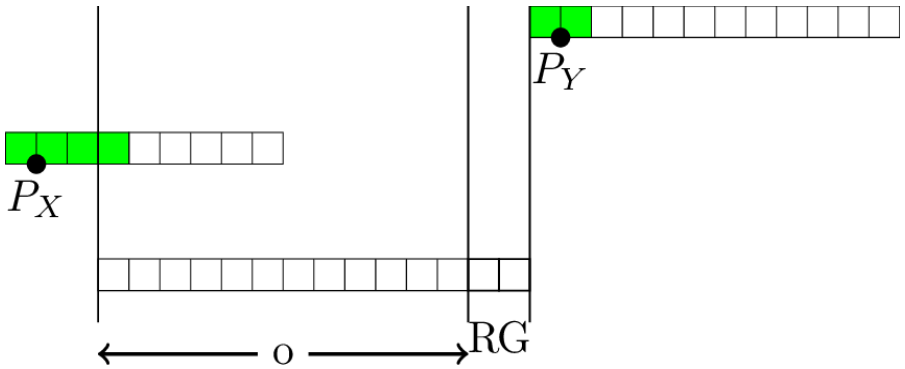
Generalized FP addition (3/4)

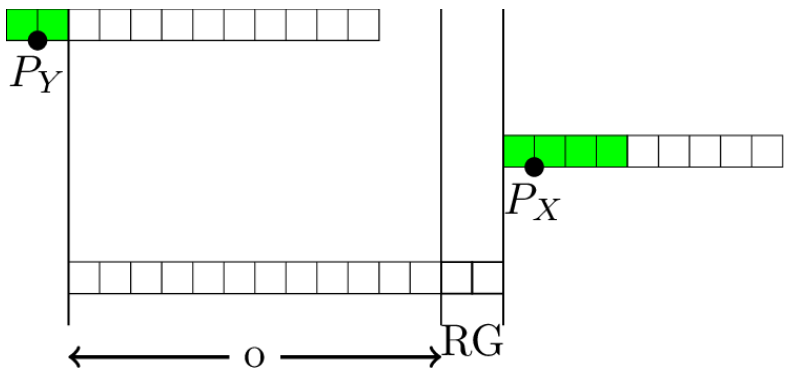




- Second case: extremal aligment
  - Determines datapath width
  - Exhibits effect of non-normalization
  - Two sub cases to be considered

Generalized FP addition (3/4)

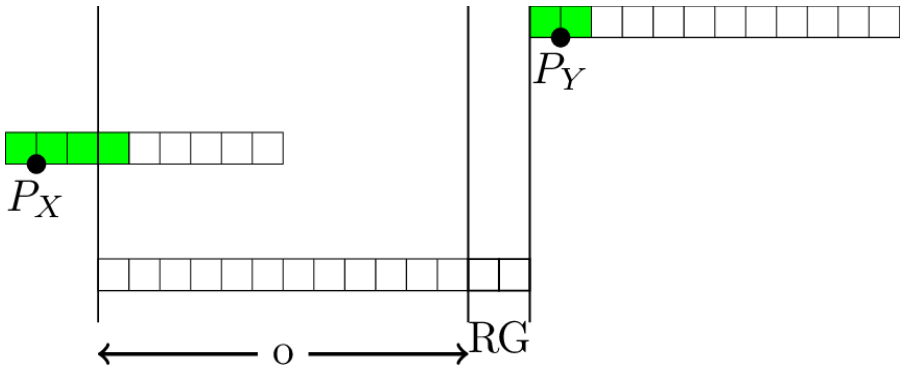


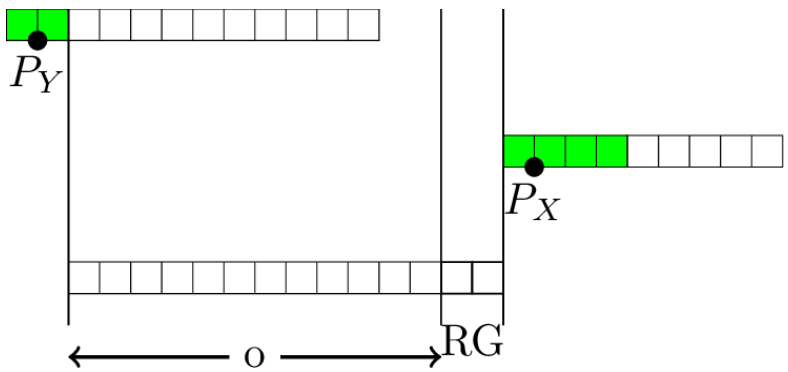


- Second case: extremal aligment
  - Determines datapath width
  - Exhibits effect of non-normalization
  - Two sub cases to be considered
- Alignment requirements:

$$\max(o+L_X,p) + \max(o+L_Y,q) + 4 + \min(p,q)$$

Generalized FP addition (3/4)





- Second case: extremal aligment
  - Determines datapath width
  - Exhibits effect of non-normalization
  - Two sub cases to be considered
- Alignment requirements:

$$\max(o+L_X,p) + \max(o+L_Y,q) + 4 + \min(p,q)$$

- Adder requirements:

$$\max(o+L_X,p) + \max(o+L_Y,q) + 5$$

Generalized FP addition (4/4)

Generalized FP addition (4/4)

- Paradigm for add-based FP blocks
  - Evaluate datapath size
  - Evaluate feasability
- Applying this paradigm to FMA:

Generalized FP addition (4/4)

- Paradigm for add-based FP blocks
  - Evaluate datapath size
  - Evaluate feasability
- Applying this paradigm to FMA:

Operator	Datapath width
FMA16 (p=o=q=11)	49
FMA32 (p=o=q=24)	101
MPFMA16 (p=11, q=o=24)	99

Generalized FP addition (4/4)

- Paradigm for add-based FP blocks
  - Evaluate datapath size
  - Evaluate feasability
- Applying this paradigm to FMA:

Operator	Datapath width
FMA16 (p=o=q=11)	49
FMA32 (p=o=q=24)	101
MPFMA16 (p=11, q=o=24)	99

- Mixed Precision FMA
  - Better accuracy than FMA
  - Comparable latency

Generalized FP addition (4/4)

- Paradigm for add-based FP blocks
  - Evaluate datapath size
  - Evaluate feasability
- Applying this paradigm to FMA:

Operator	Datapath width
FMA16 (p=o=q=11)	49
FMA32 (p=o=q=24)	101
MPFMA16 (p=11, q=o=24)	99

Operator	Cell Area (µm²)	Acc. Latency
MPFMA fp16/fp32	2690	3
FMA fp16	1840	3
FMA fp32	4790	3

- Mixed Precision FMA
  - Better accuracy than FMA
  - Comparable latency

Outline

1. Already available solutions
    1. Fused Multiply-Add
    2. Mixed Precision FMA (Generalized FP addition)
2. New design: revisiting Kulisch's accumulator
3. Metalibm and experimental results
4. Conclusion and perspectives

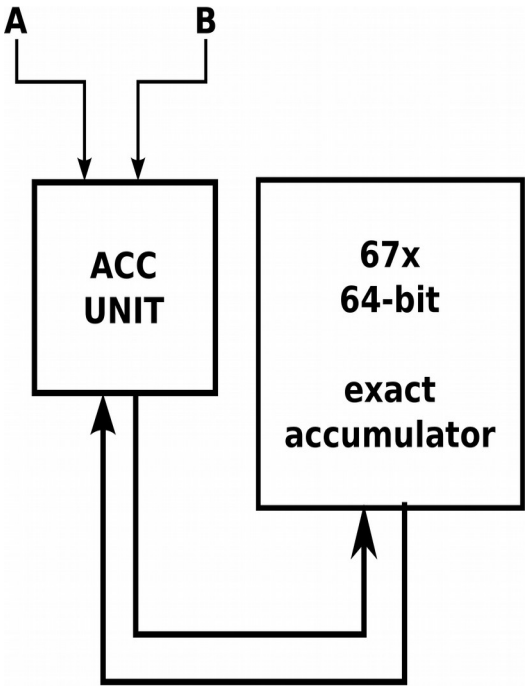
Kulisch's accumulator

Kulisch's accumulator

- Exact accumulator for FP products

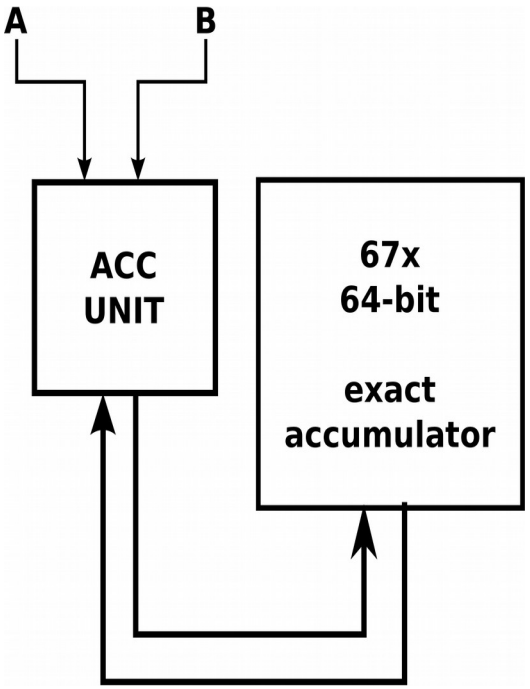
Kulisch's accumulator

- Exact accumulator for FP products
  - 554 bits for binary32
  - 4196 bits for binary64
- Kulisch design is memory-based
  - Full integration in Arithmetic Unit
  - But quite a large memory footprint
- Some drawbacks
  - Not scalable (e.g. vectorization)
  - Require heavy CPU architectural modification



Kulisch's accumulator

- Exact accumulator for FP products
  - 554 bits for binary32
  - 4196 bits for binary64
- Kulisch design is memory-based
  - Full integration in Arithmetic Unit
  - But quite a large memory footprint
- Some drawbacks
  - Not scalable (e.g. vectorization)
  - Require heavy CPU architectural modification



- [1] The Fifth Floating-Point Operation for Top-Performance Computers or Accumulation of Floating-Point Numbers and Products in Fixed-Point Arithmetic, Ulrich Kulisch, 1997
- [3] Design-space exploration for the Kulisch accumulator, Yohann Uguen et al., 2017
- [4] Reproducible and Accurate Matrix Multiplication for GPU Accelerators, Iakymchuk et al., 2015

Binary 16 in a nutshell

Binary 16 in a nutshell

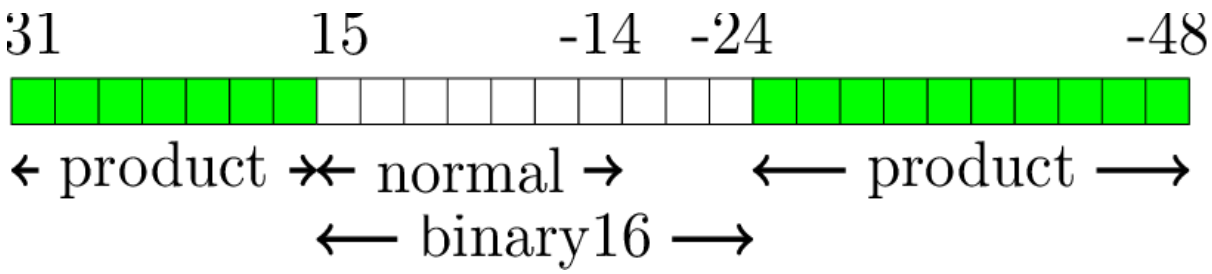
- Format with small bitfields

format	p	exp range
binary16	11	[-14,15]
binary32	24	[-126,127]
binary64	53	[-1022,1023]

Binary 16 in a nutshell

- Format with small bitfields
  - Has a very limited exponent range
    - [-14,15] for normal numbers
    - [-24,15] including subnormals
    - [-48,31] for product of any numbers

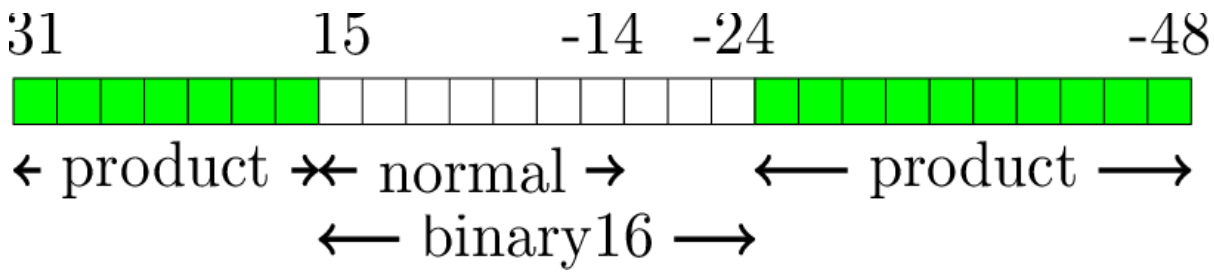
format	p	exp range
binary16	11	[-14,15]
binary32	24	[-126,127]
binary64	53	[-1022,1023]



Binary 16 in a nutshell

- Format with small bitfields
  - Has a very limited exponent range
    - [-14,15] for normal numbers
    - [-24,15] including subnormals
    - [-48,31] for product of any numbers
  - Only 80-bit required to store full product dynamic range
  - Make it suitable for in-register implementation of Kulisch's [1] accumulator

format	p	exp range
binary16	11	[-14,15]
binary32	24	[-126,127]
binary64	53	[-1022,1023]



- [1] The Fifth Floating-Point Operation for Top-Performance Computers or Accumulation of Floating-Point Numbers and Products in Fixed-Point Arithmetic, Ulrich Kulisch, 1997

Exact MPFMA16 design (1/2)

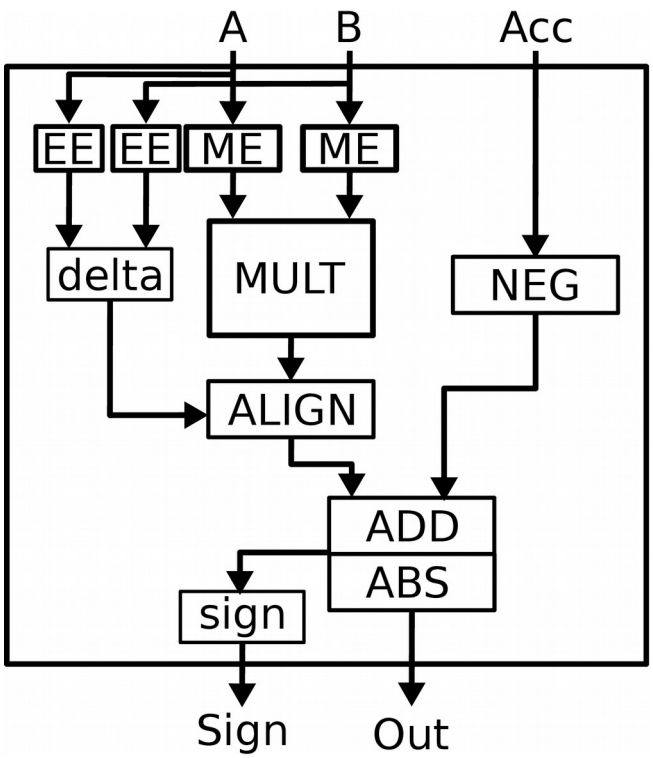
Exact MPFMA16 design (1/2)

- Implementation of Kulisch's idea
  - Using (80+ε)-bit register for accumulation
  - Fixed-point aligned to (31+ε)

Exact MPFMA16 design (1/2)

- Implementation of Kulisch's idea
  - Using (80+ε)-bit register for accumulation
  - Fixed-point aligned to (31+ε)
  - Exact (no rounding)
- Several design variations
- 1st tryout:
  - Sign-magnitude accumulator
  - Balancing acc / product path

Exact MPFMA16 design (1/2)



- Implementation of Kulisch's idea
  - Using (80+ε)-bit register for accumulation
  - Fixed-point aligned to (31+ε)
  - Exact (no rounding)
- Several design variations
- 1st tryout:
  - Sign-magnitude accumulator
  - Balancing acc / product path

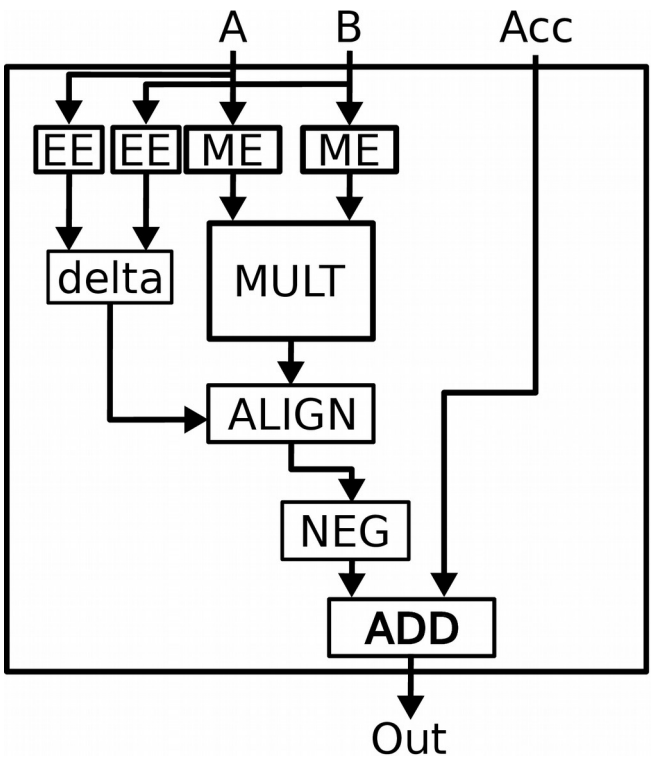
Exact MPFMA16 design (2/2)

- 2nd variation:
  - 2's complement accumulator
  - Fast accumulator path
- Very few logical levels on Acc path
  - Pushed back to final conversion
  - Product less sensible to delay
  - Bypass easy to fit in 1 cycle

Exact MPFMA16 design (2/2)

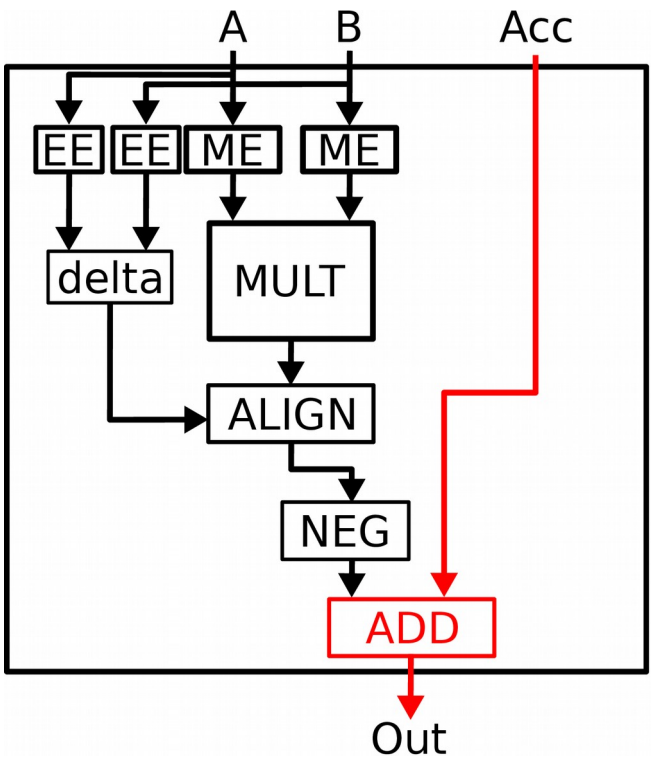
- 2nd variation:
  - 2's complement accumulator
  - Fast accumulator path
- Very few logical levels on Acc path
  - Pushed back to final conversion
  - Product less sensible to delay
  - Bypass easy to fit in 1 cycle

Exact MPFMA16 design (2/2)



- 2nd variation:
  - 2's complement accumulator
  - Fast accumulator path
- Very few logical levels on Acc path
  - Pushed back to final conversion
  - Product less sensible to delay
  - Bypass easy to fit in 1 cycle

Exact MPFMA16 design (2/2)



- 2nd variation:
  - 2's complement accumulator
  - Fast accumulator path
- Very few logical levels on Acc path
  - Pushed back to final conversion
  - Product less sensible to delay
  - Bypass easy to fit in 1 cycle

Outline

1. Already available solutions
    1. Fused Multiply-Add
    2. Mixed Precision FMA (Generalized FP addition)
2. New design: revisiting Kulisch's accumulator
3. Metalibm and experimental results
4. Conclusion and perspectives

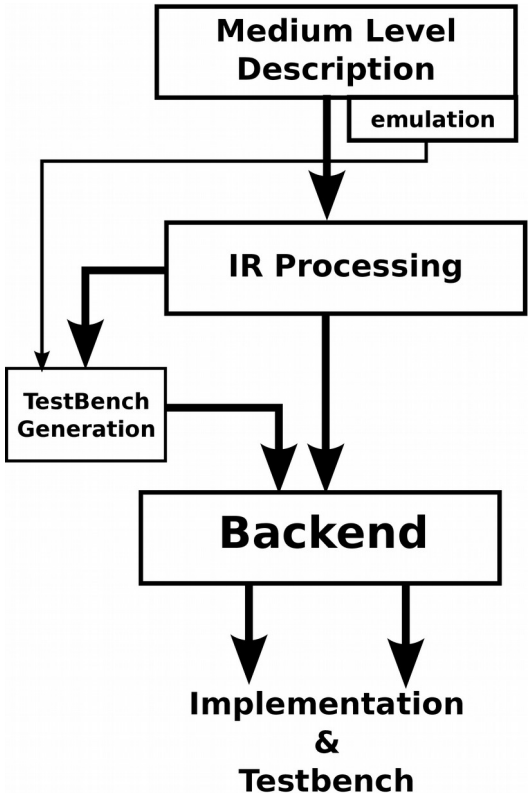
Metalibm for RTL generation

Metalibm for RTL generation

- Framework for source code generation
  - Introduced at ARITH22
  - Generate C, OpenCL-C
  - Several backends:
    - (generic x86 SSE, AVX, Kalray's K1)

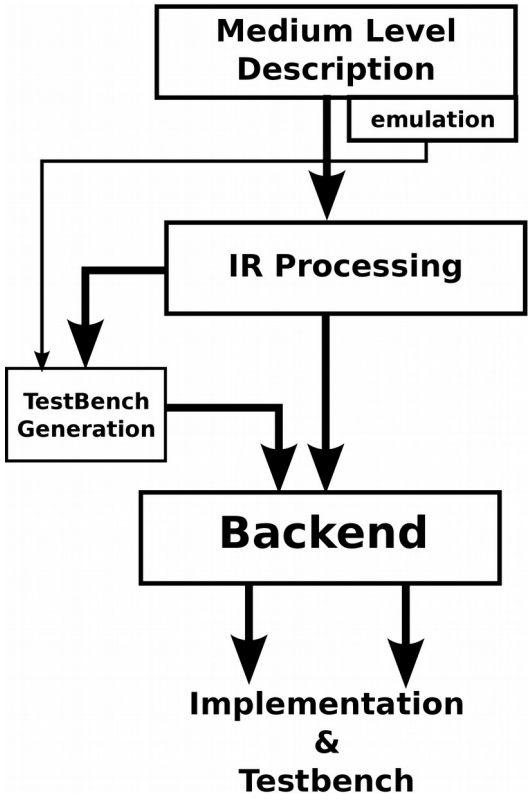
Metalibm for RTL generation

- Framework for source code generation
  - Introduced at ARITH22
  - Generate C, OpenCL-C
  - Several backends:
    - (generic x86 SSE, AVX, Kalray's K1)



Metalibm for RTL generation

- Framework for source code generation
  - Introduced at ARITH22
  - Generate C, OpenCL-C
  - Several backends:
    - (generic x86 SSE, AVX, Kalray's K1)
- Extended to generates VHDL
  - Description extension
  - IR extension
  - New VHDL backend
-



Experimental results

Experimental results

- Used metalibm to generate RTL
  - From parametric description
  - With associated testbench

Experimental results

Operator	Cell Area (μm^2)	Acc. latency
FMA fp16	1840	3
FMA fp32	4790	3
MPFMA fp16/fp32	2690	3
Fixed MPFMA Sign Magnitude	2195	1
Fixed MPFMA 2's complement	1950	1

- Used metalibm to generate RTL
  - From parametric description
  - With associated testbench

Experimental results

Operator	Cell Area (μm^2)	Acc. latency
FMA fp16	1840	3
FMA fp32	4790	3
MPFMA fp16/fp32	2690	3
Fixed MPFMA Sign Magnitude	2195	1
Fixed MPFMA 2's complement	1950	1

- Used metalibm to generate RTL
  - From parametric description
  - With associated testbench
- Fixed MPFMA more expensive than FMA
  - Larger shifter and adder

Experimental results

Operator	Cell Area (μm^2)	Acc. latency
FMA fp16	1840	3
FMA fp32	4790	3
MPFMA fp16/fp32	2690	3
Fixed MPFMA Sign Magnitude	2195	1
Fixed MPFMA 2's complement	1950	1

- Used metalibm to generate RTL
  - From parametric description
  - With associated testbench
- Fixed MPFMA more expensive than FMA
  - Larger shifter and adder
- Much more accurate
  - Fixed MPFMA is exact

Outline

1. Already available solutions
    1. Fused Multiply-Add
    2. Mixed Precision FMA (Generalized FP addition)
2. New design: revisiting Kulisch's accumulator
3. Metalibm and experimental results
4. Conclusion and perspectives

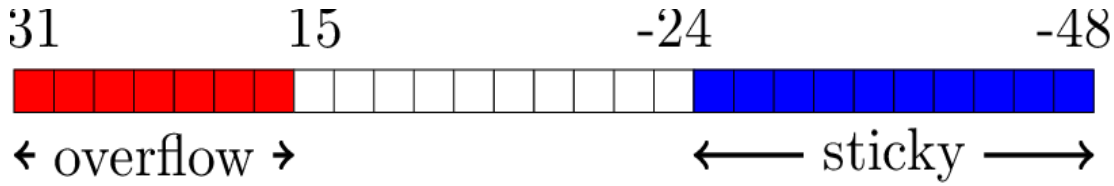
Conclusion and perspectives

- New operator archictectures:
  - MPFMA applied to binary16
  - Fixed-Point MPFMA
- Next directions:
  - Get rid of a troubling architectural state
    - e.g. 80-bit hard to save when switching context
  - Fast conversion to binary32
  - Useful for larger dimensional dot product
    - Very low overhead to add more than one product
  - Push forward 3-operand ADD

Thank you for your attention.

Converting back

- Converting back to fp32 is hard
  - Around 80-bit Leading Zero Count
  - Around 80-bit Shifter
  - 24-bit Incrementer for rounding
- Converting back to fp16 is much easier
  - exp > 14 implies overflow
  - exp < -24 implies dump into sticky
  - Straighforward subnormal output (when detected)



- [1] The Fifth Floating-Point Operation for Top-Performance Computers or Accumulation of Floating-Point Numbers and Products in Fixed-Point Arithmetic, Ulrich Kulisch, 1997

Extended bibliography

- [1] The Fifth Floating-Point Operation for Top-Performance Computers or Accumulation of Floating-Point Numbers and Products in Fixed-Point Arithmetic, Ulrich Kulisch, 1997
- [2] Fused Multiply-Add Microarchitecture Comprising Separate Early-Normalizing Multiply and Add Pipelines, David Lutz, 2011(ARITH20)
- [3] Design-space exploration for the Kulisch accumulator, Yohann Uguen et al., 2017
- [4] Reproducible and Accurate Matrix Multiplication for GPU Accelerators, Iakymchuk et al., 2015

Binary16 and Kulisch-like accumulator

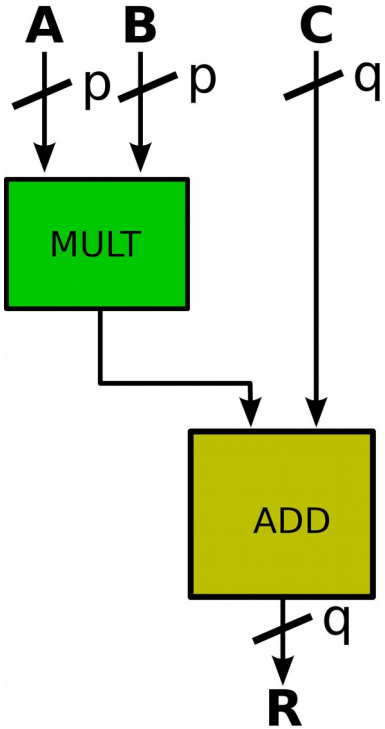
- Kulisch conceived a full precision accumulator for any formats
- Allow exact accumulation of products
- As-is hard to implement in hardware
- Require large amount of memory
- Binary16 exponent range is very reduced
- [-14,15] for normal numbers
- [-24,15] for all numbers including subnormal
- [-48,31] for product of any numbers
- Kulisch scheme can be applied to binary16
- Swapping memory accumulator for a “large” fixed-point register
-
-

Binary16 and Kulisch-like accumulator

- Binary16 exponent range is very reduced
- [-14,15] for normal numbers
-
- [-24,15] for all numbers including subnormal
- [-48,31] for product of any numbers
- Kulisch scheme can be applied to binary16
- Swapping memory accumulator for a “large” fixed-point register
-
-

Mixed Precision FMA

- Heterogeneous precision FMA
- Fuse conversions and FMA operations
- Save conversion instruction
- Keep IEEE-compliant semantic (formatOf)
- Reduce Hardware cost of FMA
- Sometimes FMA operates on heterogeneous precision
- Presented at ASILOMAR 2011
- Conversion is easy to do
- Different bias to consider when working with exponent
- Mantissa extension on the least significant side
- Fused conversions and FMA
- So why no fuse it with the FMA ?
- Save extra conversion instructions
- Keep IEEE semantic (formatOf)
- Allow high precision accumulation of small precision product
- Denormal number management changes a little
- The assumption that not both product operands are subnormal no longer holds
-



KALRAY

Accurate accumulation of products of small precision numbers

Goal: Assuming S binary32 or larger and x_i,y_j binary16 **optimize**
$\mathbf{S = [x_0,x_1,x_2,x_3,...].[y_0,y_1,y_2,y_3,...] = x_0.y_0 + x_1.y_1 + x_2.y_2 + x_3.y_3 + ...}$

- **binary16** floating-point precision
  - Introduced in IEEE754-2008
  - As a storage format not intended for computation
  - But more and more used in computation
- Problematic :
  - Optimize accuracy
  - Optimize speed (latency and throughput)
  - Suggest a generic processor operator
- Suggestion: extend FMA to smaller precisions
  - Is there a way to exploit smaller precision ?
  - Is there a way to easily extend FMA precision ?
- Design fast and small operators
  - How to implement low latency accumulation ?