Modified FMA for exact accumulation of low precision products

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

October 26th, 2017
Accurate accumulation of products of small precision numbers

Goal: Assuming \(x_i \cdot y_j\) binary16 and \(S\) binary32 or larger, optimize
\[
S = [x_0, x_1, x_2, x_3, \ldots] \cdot [y_0, y_1, y_2, y_3, \ldots] = x_0 \cdot y_0 + x_1 \cdot y_1 + x_2 \cdot y_2 + x_3 \cdot y_3 + \ldots
\]

- **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**
1\textsuperscript{st} solution: Fused Multiply-Add

- Common operator
- Basic block for accumulation
- Lots of literature
  - Focusing on binary32 and binary64
  - Architecture optimized for latency
1\textsuperscript{st} solution: Fused Multiply-Add

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

<table>
<thead>
<tr>
<th>CPU</th>
<th>ARM A72</th>
<th>AMD Bulldozer</th>
<th>Intel Skylake</th>
</tr>
</thead>
<tbody>
<tr>
<td>FMA latency</td>
<td>6/3</td>
<td>5</td>
<td>4</td>
</tr>
</tbody>
</table>

1\textsuperscript{st} solution: Fused Multiply-Add

- Common operator
- Basic block for accumulation
- 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)

<table>
<thead>
<tr>
<th>CPU</th>
<th>ARM A72</th>
<th>AMD Bulldozer</th>
<th>Intel Skylake</th>
</tr>
</thead>
<tbody>
<tr>
<td>FMA latency</td>
<td>6/3</td>
<td>5</td>
<td>4</td>
</tr>
</tbody>
</table>

2\textsuperscript{nd} solution: Mixed precision FMA
2\textsuperscript{nd} solution: Mixed precision FMA

- FMA with heterogeneous operands

\[
\text{binary16} \cdot \text{binary16} + \text{binary32} \rightarrow \text{binary32}
\]
2\textsuperscript{nd} solution: Mixed precision FMA

- FMA with heterogeneous operands
  
  \[
  \text{binary16} \cdot \text{binary16} + \text{binary32} \rightarrow \text{binary32}
  \]

- Merging conversion and FMA
  - Saving conversion instructions
  - IEEE754-compliant (formatOf)
  - Compromise between large and small FMA
    - Small multiplier
    - Large alignment and adder
2<sup>nd</sup> solution: Mixed precision FMA

- **FMA with heterogeneous operands**
  
  $\text{binary16} \cdot \text{binary16} + \text{binary32} \rightarrow \text{binary32}$

- **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 alignment**
  - Determines datapath width
  - Exhibits effect of non-normalization
  - Two sub cases to be considered
Generalized FP addition (3/4)

- Second case: extremal alignment
  - Determines datapath width
  - Exhibits effect of non-normalization
  - Two sub cases to be considered
Generalized FP addition (3/4)

- Second case: extremal alignment
  - 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 alignment
  - 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 feasibility
- Applying this paradigm to FMA:
Generalized FP addition (4/4)

- **Paradigm for add-based FP blocks**
  - Evaluate datapath size
  - Evaluate feasibility

- **Applying this paradigm to FMA:**

<table>
<thead>
<tr>
<th>Operator</th>
<th>Datapath width</th>
</tr>
</thead>
<tbody>
<tr>
<td>FMA16 (p=o=q=11)</td>
<td>49</td>
</tr>
<tr>
<td>FMA32 (p=o=q=24)</td>
<td>101</td>
</tr>
<tr>
<td>MPFMA16 (p=11, q=o=24)</td>
<td>99</td>
</tr>
</tbody>
</table>
Generalized FP addition (4/4)

- **Paradigm for add-based FP blocks**
  - Evaluate datapath size
  - Evaluate feasibility

- **Applying this paradigm to FMA:**

<table>
<thead>
<tr>
<th>Operator</th>
<th>Datapath width</th>
</tr>
</thead>
<tbody>
<tr>
<td>FMA16 (p=o=q=11)</td>
<td>49</td>
</tr>
<tr>
<td>FMA32 (p=o=q=24)</td>
<td>101</td>
</tr>
<tr>
<td>MPFMA16 (p=11, q=o=24)</td>
<td>99</td>
</tr>
</tbody>
</table>

- **Mixed Precision FMA**
  - Better accuracy than FMA
  - Comparable latency
Generalized FP addition (4/4)

- Paradigm for add-based FP blocks
  - Evaluate datapath size
  - Evaluate feasibility

- Applying this paradigm to FMA:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Datapath width</th>
</tr>
</thead>
<tbody>
<tr>
<td>FMA16 (p=o=q=11)</td>
<td>49</td>
</tr>
<tr>
<td>FMA32 (p=o=q=24)</td>
<td>101</td>
</tr>
<tr>
<td>MPFMA16 (p=11, q=o=24)</td>
<td>99</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Operator</th>
<th>Cell Area (µm²)</th>
<th>Acc. Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPFMA fp16/fp32</td>
<td>2690</td>
<td>3</td>
</tr>
<tr>
<td>FMA fp16</td>
<td>1840</td>
<td>3</td>
</tr>
<tr>
<td>FMA fp32</td>
<td>4790</td>
<td>3</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>format</th>
<th>p</th>
<th>exp range</th>
</tr>
</thead>
<tbody>
<tr>
<td>binary16</td>
<td>11</td>
<td>[-14,15]</td>
</tr>
<tr>
<td>binary32</td>
<td>24</td>
<td>[-126,127]</td>
</tr>
<tr>
<td>binary64</td>
<td>53</td>
<td>[-1022,1023]</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>format</th>
<th>p</th>
<th>exp range</th>
</tr>
</thead>
<tbody>
<tr>
<td>binary16</td>
<td>11</td>
<td>[-14,15]</td>
</tr>
<tr>
<td>binary32</td>
<td>24</td>
<td>[-126,127]</td>
</tr>
<tr>
<td>binary64</td>
<td>53</td>
<td>[-1022,1023]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>31</th>
<th>15</th>
<th>-14</th>
<th>-24</th>
<th>-48</th>
</tr>
</thead>
<tbody>
<tr>
<td>← product ← normal →</td>
<td>← product →</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>← binary16 →</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>format</th>
<th>p</th>
<th>exp range</th>
</tr>
</thead>
<tbody>
<tr>
<td>binary16</td>
<td>11</td>
<td>[-14,15]</td>
</tr>
<tr>
<td>binary32</td>
<td>24</td>
<td>[-126,127]</td>
</tr>
<tr>
<td>binary64</td>
<td>53</td>
<td>[-1022,1023]</td>
</tr>
</tbody>
</table>

[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 + \(\varepsilon\))-bit register for accumulation
  - Fixed-point aligned to (31 + \(\varepsilon\))
  - 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

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

<table>
<thead>
<tr>
<th>Operator</th>
<th>Cell Area (μm²)</th>
<th>Acc. latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>FMA fp16</td>
<td>1840</td>
<td>3</td>
</tr>
<tr>
<td>FMA fp32</td>
<td>4790</td>
<td>3</td>
</tr>
<tr>
<td>MPFMA fp16/fp32</td>
<td>2690</td>
<td>3</td>
</tr>
<tr>
<td>Fixed MPFMA Sign Magnitude</td>
<td>2195</td>
<td>1</td>
</tr>
<tr>
<td>Fixed MPFMA 2's complement</td>
<td>1950</td>
<td>1</td>
</tr>
</tbody>
</table>
## Experimental results

<table>
<thead>
<tr>
<th>Operator</th>
<th>Cell Area (μm²)</th>
<th>Acc. latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>FMA fp16</td>
<td>1840</td>
<td>3</td>
</tr>
<tr>
<td>FMA fp32</td>
<td>4790</td>
<td>3</td>
</tr>
<tr>
<td>MPFMA fp16/fp32</td>
<td>2690</td>
<td>3</td>
</tr>
<tr>
<td>Fixed MPFMA Sign Magnitude</td>
<td>2195</td>
<td>1</td>
</tr>
<tr>
<td>Fixed MPFMA 2's complement</td>
<td>1950</td>
<td>1</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Operator</th>
<th>Cell Area (μm²)</th>
<th>Acc. latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>FMA fp16</td>
<td>1840</td>
<td>3</td>
</tr>
<tr>
<td>FMA fp32</td>
<td>4790</td>
<td>3</td>
</tr>
<tr>
<td>MPFMA fp16/fp32</td>
<td>2690</td>
<td>3</td>
</tr>
<tr>
<td>Fixed MPFMA Sign Magnitude</td>
<td>2195</td>
<td>1</td>
</tr>
<tr>
<td>Fixed MPFMA 2’s complement</td>
<td>1950</td>
<td>1</td>
</tr>
</tbody>
</table>

- 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 architectures:
  - 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
  - Straightforward 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
- [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
Accurate accumulation of products of small precision numbers

**Goal:** Assuming $S$ binary32 or larger and $x_i, y_j$ binary16 optimize

$S = [x_0, x_1, x_2, x_3, ...][y_0, y_1, y_2, y_3, ...] = x_0 \cdot y_0 + x_1 \cdot y_1 + x_2 \cdot y_2 + x_3 \cdot y_3 + ...$

- **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?