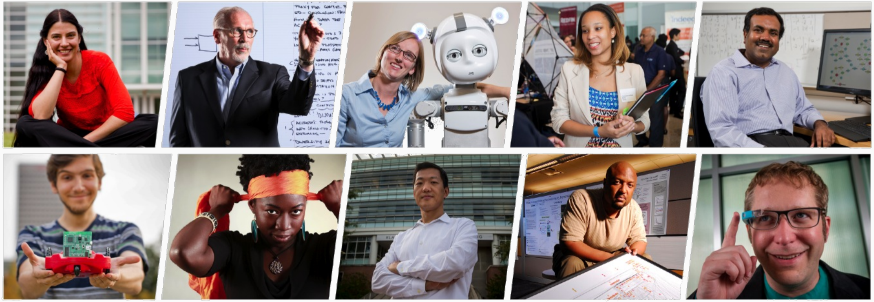

Augmented Arithmetic Operations Proposed for IEEE 754-2018

E. Jason Riedy

Georgia Institute of Technology

25th IEEE Symposium on Computer Arithmetic, 26 June 2018

Outline

History and Definitions

Quick Definitions

twoSum Two summands x and $y \Rightarrow$ sum s and error e

fastTwoSum Two summands x and y with $|x| \geq |y| \Rightarrow$ sum s and error e

twoProd Two multiplicands x and $y \Rightarrow$ product p and error e

All can overflow. Only twoProd can underflow with gradual underflow.

Otherwise, relationships are **exact** for finite x and y.

These are for **binary** floating point.

Partial History

- Møller (1965), Knuth (1965): twoSum for *quasi double precision*
- Kahan (1965): fastTwoSum for *compensated summation*
- Dekker (1971): fastTwoSum for expansions
- Shewchuk (1997): Geometric predicates
- Brigg (1998): quasi double precision implementation as *double-double*
- Hida, Li, Bailey (2001): *quad-double*
- Ogita, Rump, Oishi (2005...): Error-free transformations
- IEEE 754-2008: Punted to the future
- Ahrens, Nguyen, Demmel (2013...): ReproBLAS

“Typical” twoSum

```
void
twoSum (const double x, const double y,
          double * head, double * tail)
// Knuth, second volume of TAoCP
{
    const double s = x + y;
    const double bb = s - x;
    const double err = (x - (s - bb)) + (y - bb);
    *head = s;
    *tail = err;
}
```

Six instructions, long dependency chain.
Reducing to one or two instructions: More uses.

twoProduct

```
void
twoProduct (const double a, const double b,
            double *head, double *tail)
{
    double p = a * b;
    double t = fma (a, b, −p);
    *head = p;
    *tail = t;
}
```

Already two instructions.
With the right instruction.

twoSum, twoProd Unusual Exceptional Cases

x		y		head	tail	signal
∞	+	∞	$\Rightarrow$	∞	NaN	invalid
$-\infty$	+	$-\infty$	$\Rightarrow$	$-\infty$	NaN	invalid
x	+	y	$\Rightarrow$	∞	NaN	invalid, overflow, inexact ($x + y$ overflows)
$-x$	+	$-y$	$\Rightarrow$	$-\infty$	NaN	invalid, overflow, inexact ($-x - y$ overflows)
-0	+	-0	$\Rightarrow$	-0	$+0$	(none)
x	$\times$	y	$\Rightarrow$	∞	$-\infty$	overflow, inexact ($x \times y$ overflows)
$-x$	$\times$	y	$\Rightarrow$	$-\infty$	∞	overflow, inexact ($-x \times y$ overflows)
-0	$\times$	0	$\Rightarrow$	-0	0	(none)

Other Orders, Other Cases

- Algebraic re-orderings produce different cases
  - Different signs
  - Different NaN locations
- Similarly for twoProd and fastTwoSum.
- Hence, double-double behaves strangely...
- And reprodicibility becomes tricky.

Reproducible Linear Algebra

- Ahrens, Nguyen, Demmel: *K*-fold indexed sum
  - Base of the ReproBLAS, associative
  - Core operation: fastTwoSum!
  - **But rounding must not depend on the output value.**

  Details:
  - Demmel and Nguyen, "Fast Reproducible Floating-Point Summation," ARITH21, 2013.
  - Ahrens, Nguyen, and Demmel, "Effcient Reproducible Floating Point Summation and BLAS," UCB Tech Report, 2016.
  - `bebop.cs.berkeley.edu/reproblas/`

Note: There are other methods, *e.g.* ARM's.

Performance?

Emulating augmentedAddition as two instructions improves double-double:

Operation		Skylake	Haswell
Addition	latency	−55%	−45%
	throughput	+36%	+18%
Multiplication	latency	−3%	0%
	throughput	+11%	+16%

DDGEMM MFLOP/s:

Operation	Intel Skylake	Intel Haswell
"Typical" implementation	1732 ($\approx$ 1/37 DP)	1199 ($\approx$ 1/45 DP)
Two-insn augmentedAddition	3344 ($\approx$ 1/19 DP)	2283 ($\approx$ 1/24 DP)

Dukhan, Riedy, Vuduc. "Wanted: Floating-point add round-off error instruction," PMAA 2016.

ReproBLAS dot product: 33% rate improvement, only 2× slower than non-reproducible.

Decisions

Decisions Made for Standardization

- Time to standardize?
  - Was considered for 2008, but only use was extending precision.
  - Now more uses that can benefit.
- Names?
  - twoSum, *etc.* already exist.
  - So **augmented arithmetic operations**.
  - augmentedAddition, augmentedSubtraction, augmentedMultiplication
- Exceptional behavior?
  - My first proposal was twoSum above.
  - Generally not wise...
- Rounding...

Standard Behavior augmentedAddition (I)

x	y	head	tail	signal
NaN	NaN	NaN	NaN	invalid on sNaN
$\pm\infty$	NaN	NaN	NaN	invalid on sNaN
NaN	$\pm\infty$	NaN	NaN	invalid on sNaN
∞	∞	∞	∞	(none)
∞	$-\infty$	NaN	NaN	invalid
$-\infty$	∞	NaN	NaN	invalid
$-\infty$	$-\infty$	$-\infty$	$-\infty$	(none)
x	y	∞	∞	overrflow, inexact ($x + y$ overflows)
$-x$	$-y$	$-\infty$	$-\infty$	overrflow, inexact ($-x - y$ overflows)

Standard Behavior augmentedAddition (II)

x	y	head	tail	signal
$+0$	$+0$	$+0$	$+0$	(none)
$+0$	-0	$+0$	$+0$	(none)
-0	$+0$	$+0$	$+0$	(none)
-0	-0	-0	-0	(none)

Double-double, *etc.* look more like IEEE 754.

Rounding

- Reproducibility cannot round ties to nearest even.
    - Depends on the output value.
- Leaves rounding ties away...
    - Already exists for binary.
    - Surveyed users.
    - Shewchuck: `triangle` could not use it.
    - Rump, *et al.*: Proofs need reworked.
- So rounding ties toward zero.
    - roundTiesToZero from decimal
    - Only these operations.
    - sigh.

Testing

Testing augmentedAddition

- Assume p bits of precision, and augmentedAddition$(x, y) \Rightarrow (h, t)$.
- Typical cases are fine, only need special cases to test for roundTiesToZero.
- Generate x, y such that y immediately follows x and causes a tie that rounds to a value easy to check.
  - Params: s sign, T significand, E exponent
  - $x = (-1)^{1-s} \cdot T \cdot 2^E$ with T odd
  - $y = (-1)^{1-s} \cdot 2^{E-p-1}$
- Results:
  - roundTiesToZero: (x, y)
  - Away or to even: $(x + 2y, -y)$

Testing augmentedMultiplication: Rounding

- Similarly need special cases to test for roundTiesToZero and underflow.
- Generate x, y such that xy requires $p+1$ bits and has 11 as the final bits.
- $x_m \times y_m = (2^{p+1} + 4M + 3) \cdot 2^E$ with $0 \leq M < 2^{p-1}$ and $emin \leq E + p - 1 < emax$
    - Sample M
    - Factor the significand $2^{p+1} + 4M + 3$
    - Sample random exponents E_{x_m} and E_{y_m} with $emin \leq E_{x_m} + E_{y_m} < emax$
- But also need underflow...

Testing augmentedMultiplication: Underflow

- Testing underflow: Product needs the tail chopped off by underflow.
- Let $0 \leq k < p - 1$ be the number of additional bits we want below the hard underflow threshold.
- Sample:
  - $0 \leq M < 2^{p-1} - 1$
  - $0 \leq N < 2^{p-k-1}$
  - $0 \leq T < 2^k$

$$x_m \times y_m = \left(\underbrace{(2^{p-1} + M) \cdot 2^p}_{\text{head}} + \underbrace{N \cdot 2^k}_{\text{tail}} + \underbrace{2T + 1}_{\substack{\text{below} \\ \text{underflow}}} \right) \cdot 2^{emin-p-k}$$

- This case must signal inexact & underflow.

Reference Code

Not done yet, but...

1. Handle special cases.
2. If no ties (far case), twoSum.
3. If ties (near case), split and take care.
    - Tools from the next talk.

Should be slow but clear.

Anyone feel like adding this to the RISC-V Rocket core?

Summary

Summary

- Three “new” recommended ops in draft IEEE 754-2018
  - augmentedAddition, augmentedSubtraction, augmentedMultiplication
- Specified to support extending precision (double double), reproducible linear algebra
- Two instruction implementation can provide good performance benefits.
- Quite easily testible
- Future: Decimal? Hardware?

fastTwoSum

```
void
fastTwoSum (const double x, const double y,
                  double * head, double * tail)
/* Assumes that |x| ≤|y| */
{
    const double s = x + y;
    const double bb = s − x;
    const double err = y − bb;
    *head = s;
    *tail = err;
}
```

Standard Behavior augmentedMultiplication (I)

x	y	head	tail	signal
NaN	NaN	NaN	NaN	invalid on sNaN
$\pm\infty$	NaN	NaN	NaN	invalid on sNaN
NaN	$\pm\infty$	NaN	NaN	invalid on sNaN
x	y	∞	∞	overflow, inexact ($x \times y$ overflows)
$-x$	y	$-\infty$	$-\infty$	overflow, inexact ($-x \times y$ overflows)
x	$-y$	$-\infty$	$-\infty$	overflow, inexact ($x \times -y$ overflows)
$-x$	$-y$	∞	∞	overflow, inexact ($-x \times -y$ overflows)

Standard Behavior augmentedMultiplication (II)

x	y	head	tail	signal
∞	∞	∞	∞	(none)
∞	$-\infty$	$-\infty$	$-\infty$	(none)
$-\infty$	∞	$-\infty$	$-\infty$	(none)
$-\infty$	$-\infty$	∞	∞	(none)
$+0$	$+0$	$+0$	$+0$	(none)
$+0$	-0	-0	-0	(none)
-0	$+0$	-0	-0	(none)
-0	-0	$+0$	$+0$	(none)

Double-double, *etc.* look more like IEEE 754.