MANCHESTER 1824

The University of Manchester

M\cr NA
Manchester Numerical Analysis

A Trick for an Accurate $e^{-|x|}$ Function in Fixed-Point Arithmetics

Mantas Mikaitis
Department of Mathematics
The University of Manchester

`mantas.mikaitis@manchester.ac.uk`

20th International Symposium on Scientific Computing, Computer Arithmetic, and Verified Numerical Computation (SCAN2020)

September 13-17, Szeged, Hungary (Virtual)

Motivation

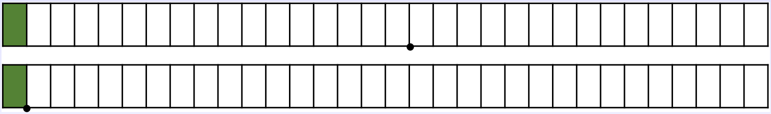
- Improve accuracy of **exponential decay**
- **Fixed-point arithmetic** on embedded CPUs and similar
- Inputs of e^x in exponential decay require **both integer and fractional parts**
- Taking e^0 as special case, **outputs only fractional**
- To optimize accuracy this means **mixed I/O format combination**
- This work: **use single-format implementation** of e^x to **get a mixed format** $e^{-|x|}$
- No change to the implementation of e^x needed
- Beneficial when single-format e^x cannot be modified

Fixed-Point Arithmetic

Consider two commonly used fixed-point data formats from the **ISO/IEC 18037:2008 standard**:

- `s16.15` (*accum*): sign, 16 integer, 15 fractional bits
- `s0.31` (*fract*): sign, zero integer, 31 fractional bits.

Here `s16.15` has a range of approximately $(-65536, 65536)$ in steps of $\varepsilon_{s16.15} = 2^{-15}$.
`s0.31` has a range of $(-1, 1)$ in steps of $\varepsilon_{s0.31} = 2^{-31}$.



Usage of two formats

Use `s16.15` for range, `s0.31` for accuracy.

Exponential in Fixed-Point Arithmetic

Assume we have an implementation (for example, in hardware) of $y_a = e^{x_a}$ with x_a and y_a `s16.15`. **Here, rounding is implicit to what format is on the left-hand side**.

The input domain is

$$x_a \in (\log_e(2^{-15}) = -10.397..., \log_e(2^{16} - 2^{-15}) = 11.09...)$$

since any x_a outside that would **under**/**overflow** output.

Mixed-format input/output domain

If we take input `s16.15` but output `s0.31`, the input domain changes to $x_a \in (\log_e(2^{-31}) = -21.487..., 0)$. Wider input range and more accurate outputs for $x_a < 0$.

Exponential Decay in Fixed-Point Arithmetic

We sometimes need only $x \leq 0$ in e^x, for example **exponential decay**.

We focus on improving the accuracy of $e^{-|x|}$ using the observations above, when we have e^x in `s16.15`. Take $y_f = e^{x_a}$ to be exponential in `s0.31`.

We use $\mathrm{int}(x)$ to get an **underlying integer representing x in fixed-point**.

If we first compute y_a and then do a basic shift to get $\mathrm{int}(y_f) = 2^{16}\mathrm{int}(y_a)$ **we would not gain any accuracy since zeros are shifted in to the bottom 16 bits**.

A Trick with Input Range Shifting

Use a property

$$2^{16}e^{x} = e^{\log_e 2^{16}}e^{x} = e^{16\times\log_e 2+x}.$$

Input range shifting

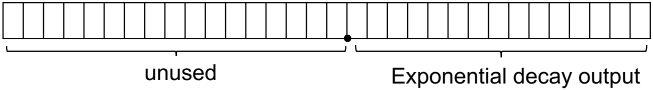
Instead of computing $\mathrm{int}(y_f)$ through first computing y_a and then shifting, we can compute

$$\mathrm{int}(y_f) = \mathrm{int}(e^{16\times\log_e 2+x_a}).$$

Note that this modification to the input **requires only an addition of a precomputed constant** and interpreting the underlying integer representation of the output from the `s16.15` exp as `s0.31`.

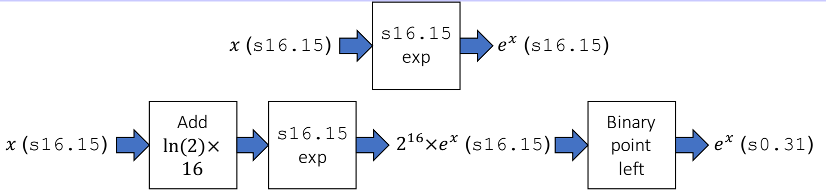
A Trick with Input Range Shifting

Decay output in accum:



- **Most bits are unused** in `s16.15` exp for $x < 0$.
- The core idea is to **shift the range of inputs** from $x < 0$ to the original input of the `s16.15` exp, compute exp using all the bits of the `s16.15` and **recover the answer in the original range**.
- The answer is placed 16 bits to the left from what it should be in `s16.15`, which allows us to interpret it as `s0.31`.

A Trick with Input Range Shifting



In summary, **the proposed modification includes following steps** (figure, top: **standard usage**, bottom: **proposed modification**).

- Add $16 \times \log_e 2 \approx 11.09$ to x_a.
- Calculate exp using the `s16.15` implementation.
- Interpret the output bits as `s0.31` instead of `s16.15`.

Example

Consider computing e^{-8}. In basic `s16.15` we get 0.000335693359375 (**error** ~ -0.00000023).

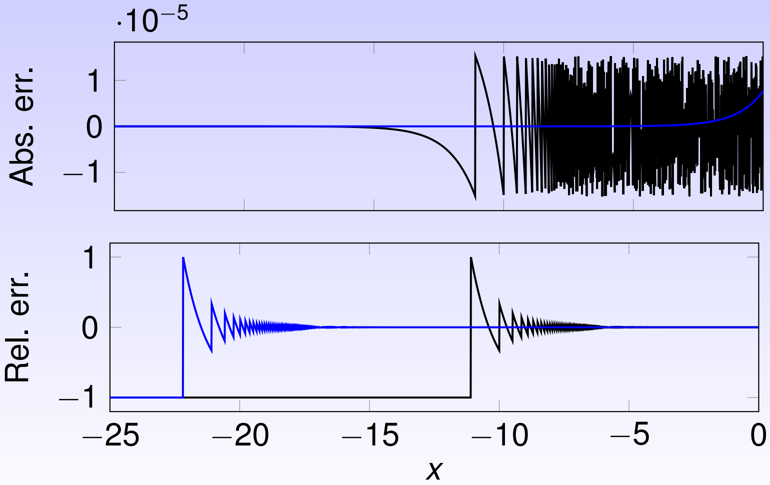
- **Compute the constant** $\log_e 2 \times 16 = 11.090362548828125$ (`s16.15`).
- **Shift range**: $-8 + 11.090362548828125 = 3.090362548828125$ (`s16.15`).
- Compute $e^{3.090362548828125} = 21.98504638671875$ (`s16.15`).
- Get the underlying int representation: $21.98504638671875 \times 2^{15} = 720506$.
- Interpret as `s0.31`: $720506/2^{31} = 0.000335511751472949981689453125$ (**error** ~ -0.000000049).

Accuracy Comparison

- To **compare the accuracy of the two approaches**, we used MATLAB's `binary64 exp`.
- We take $x \in [-25, -0.01]$ (in steps of 0.01) and for each obtain x_a by rounding to the nearest `s16.15` value.
- The constant $\log_e 2 \times 16$ is represented as `s16.15` and added to x_a (**note addition contains no error, only the rounding of the constant**).
- We compute the `binary64` exp and round it to `s16.15` to simulate the `s16.15` implementation.
- To implement the proposed mixed-format version, we round the output to `s16.15` and then divide by 2^{16} to obtain the `s0.31` answer.

Accuracy Comparison

Below: **absolute and relative errors** of the standard `s16.15` exp (black) and the **mixed-format version**.



Conclusion

- We presented **a basic input transformation** to improve the accuracy of e^x when $x < 0$.
- Tested using the `s16.15` and `s0.31` fixed-point arithmetics.
- Both the **input range** where underflow does not occur and the **accuracy** of $e^{-|x|}$ were improved.
- Future work includes **generalizing the results for parameterized precisions**, **performing error analysis**, and **finding other functions that can be similarly improved**.

Slides and code

Available `https://mmikaitis.github.io/talks/`

Acknowledgements

Funding: **EPSRC grant EP/P020720/1**.

We thank Nicholas J. Higham for his comments that improved the experiments and slides.

References I

ISO/IEC
Programming languages - C - Extensions to support embedded processors
ISO/IEC JTC 1/SC 22, 2008

M. Mikaitis
Arithmetic Accelerators for a Digital Neuromorphic Processor
University of Manchester, PhD thesis, Jul. 2020