



MATLAB Simulator of Level-Index Arithmetic

Mantas Mikaitis

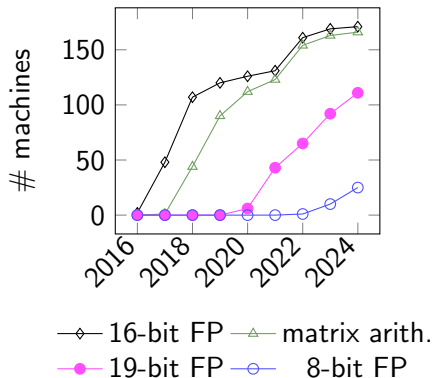
School of Computing, University of Leeds, Leeds, UK

31st IEEE International Symposium on Computer Arithmetic
Málaga, Spain, Jun. 11, 2024

Slides: <https://mmikaitis.github.io/talks>



Low-precision computer arithmetic of the TOP500



Devices counted: P100, V100, A100, H100, MI210, MI250X, MI300X, Intel Data Center GPU, from <https://www.top500.org>.

With NVIDIA B100 (block) 4/6-bit FP will appear in coming years.

Motivation for the toolbox

Low-precision and non-standard number representations are being actively considered.

Various trade-offs between hardware costs, precision, range are of interest in domain-specific architectures.

Level-index arithmetic has a wide range of representable numbers.

What you get from today's talk

- revisit symmetric level-index (SLI) representation;
- learn about a new MATLAB toolbox for simulating SLI.

Level-index (LI) number system

In the level-index representation [Clenshaw and Olver, 1984] $x \in \mathbb{R}$ is represented as

$$x = \pm e^{e^{\dots e^f}}$$

where the process of exponentiation is done l times. Here $f \in [0, 1)$.

When $l = 0$ we set $x = f$.

Example

$$0.5 = f = 0.5 \text{ (level 0)}$$

$$1 = e^0 \text{ (level 1)}$$

$$1.5 \approx e^{0.4} \text{ (level 1)}$$

$$2 \approx e^{0.69} \text{ (level 1)}$$

$$3 \approx e^{e^{0.09}} \text{ (level is now 2)}$$

Symmetric level-index (SLI) number system

Level 0 in LI is a special case that was removed in symmetric level-index system [Clenshaw and Turner, 1988].

In SLI a number $x \in \mathbb{R}$ has the form

$$x = s(x)\phi(\zeta)^{r(x)},$$

where $s(x) = \pm 1$ is the sign, $r(x) = \pm 1$ is the reciprocal sign defined by

$$r(x) = \begin{cases} +1, & \text{if } |x| \geq 1, \\ -1, & \text{if } |x| < 1, \end{cases}$$

and $\phi(\zeta)$ is the standard LI generalized exponential with $\zeta = l + f$:

$$\phi(\zeta) = \begin{cases} \zeta, & \text{if } 0 \leq \zeta < 1, \\ e^{\phi(\zeta-1)}, & \text{if } \zeta \geq 1. \end{cases}$$

Allows to preserve relative precision of numbers below 1.

To construct ζ from $x \in \mathbb{R}$:

$$\Psi(x) = \begin{cases} x, & \text{if } 0 \leq x < 1, \\ 1 + \Psi(\ln(x)), & \text{if } x \geq 1, \end{cases}$$

Symmetric level-index (SLI) number system

Numbers above 1 are represented in the same way as before.

Example

$$1 = e^0$$

$$1.5 \approx e^{0.4}$$

$$2 \approx e^{0.69}$$

$$3 \approx e^{e^{0.09}}$$

Whereas numbers below 1 are formed by reciprocating the same set of underlying ζ numbers.

Example

$$1 = e^{0^{-1}} \text{ (yes, an issue, two representations of 1)}$$

$$0.67 \approx e^{0.4^{-1}}$$

$$0.5 \approx e^{0.69^{-1}}$$

$$0.33 \approx e^{e^{0.09^{-1}}}$$

Previous results: key highlights

Notation

With $\text{sli-}k.p$ we refer to a SLI encoding with a k -bit level and a p -bit index. Total width of the encoding is $k + p + 2$.

- [Olver, ARITH'87] proves *closure* properties of SLI and shows that 3 bits for the level is enough for practical purposes.
- [Olver and Turner, ARITH'87] show table-based parallel algorithms for basic operations. Fast but big circuits.
- [Turner, ARITH'89]
 - Simulation of sli-3.27 in Turbo Pascal;
 - implements scalar, vector, polynomial ops;
 - sli-integer mixed-format ops.
- [Kwak and Swartzlander, 1998] show how to implement LI in hardware by using CORDIC. From $>11\text{K}$ Full Adders down to ~ 3800 .
- [Shen and Turner, 2006] show a hybrid SLI and floating point method.

Encoding of SLI numbers

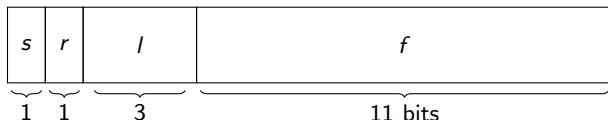
Our chosen encoding for SLI $sli-k.p$ numbers is:

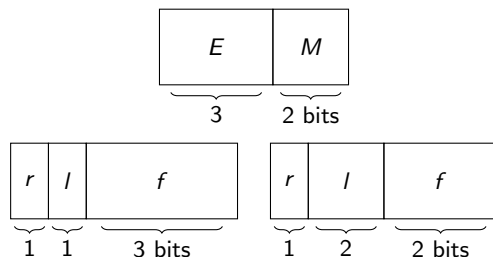
- sign bit 1 ($s(x) = -1$) or 0 ($s(x) = 1$); sign-magnitude representation.
- reciprocal bit 0 ($r(x) = -1$) or 1 ($r(x) = 1$);
- level: k -bit integer; encodes levels 1 to 2^k .
- index: p -bit fixed-point number in $[0, 1)$.

Arrangement: sign-reciprocal-level-index.

Bit patterns $00\dots$ map to numbers below 1, whilst patterns $01\dots$ to numbers 1 and higher. Analogous on negative side of SLI line.

[Turner, ARITH'89] used 1's complement when $r(x) = -1$.





Encodings of toy unsigned floating-point and SLI systems:

- top: 5-bit floating-point system following IEEE 754 rules; $e_{max} = 3$, $e_{min} = -2$.
- bottom left: sli-1.3;
- bottom right: sli-2.2.

Example: quantities encoded in toy systems

	FP	sli-1.3	sli-2.2
00000	0	<i>Zero</i>	<i>Zero</i>
00001	0.0625	$(e^{0.125})^{-1} \approx 0.8825$	$(e^{0.25})^{-1} \approx 0.7788$
00010	0.125	$(e^{0.25})^{-1} \approx 0.7788$	$(e^{0.5})^{-1} \approx 0.6065$
00011	0.1875	~ 0.6873	~ 0.4724
00100	0.25	~ 0.6065	$(e^{e^0})^{-1} \approx 0.3679$
00101	0.3125	~ 0.5353	~ 0.2769
00110	0.375	~ 0.4724	~ 0.1923
00111	0.4375	~ 0.4169	~ 0.1204
01000	0.5	$(e^{e^0})^{-1} \approx 0.3679$	$(e^{e^{e^0}})^{-1} \approx 0.06599$
01001	0.625	$(e^{e^{0.125}})^{-1} \approx 0.322$	~ 0.02702
01010	0.75	$(e^{e^{0.25}})^{-1} \approx 0.2769$	~ 0.0055
01011	0.875	~ 0.2334	$\sim 2.4 \times 10^{-4}$
01100	1	~ 0.1923	$(e^{e^{e^{e^0}}})^{-1} \approx 2.6 \times 10^{-7}$
01101	1.25	~ 0.1544	$\sim 8.4 \times 10^{-17}$
01110	1.5	~ 0.1204	$\sim 1.7 \times 10^{-79}$
01111	1.75	~ 0.0908	$\sim 10^{-1758}$

Example: quantities encoded in toy systems

	FP	sli-1.3	sli-2.2
10000	2	$(e^0)^1 = 1$	$(e^0)^1 = 1$
10001	2.5	$(e^{0.125})^1 \approx 1.1331$	$(e^{0.25})^1 \approx 1.284$
10010	3	$(e^{0.25})^1 \approx 1.284$	$(e^{0.5})^1 \approx 1.6487$
10011	3.5	~ 1.455	~ 2.117
10100	4	~ 1.6487	$(e^0)^1 \approx 2.7183$
10101	5	~ 1.8682	~ 3.6111
10110	6	~ 2.117	~ 5.2003
10111	7	~ 2.3989	~ 8.3062
11000	8	$(e^{e^0})^1 \approx 2.7183$	$(e^{e^0})^1 \approx 15.1533$
11001	10	$(e^{e^{0.125}})^1 \approx 3.1054$	~ 37.0085
11010	12	$(e^{e^{0.25}})^1 \approx 3.6111$	~ 181.3313
11011	14	~ 4.2844	~ 4048.8237
11100	$+\infty$	~ 5.2	$(e^{e^{e^0}})^1 \approx 3.8 \times 10^6$
11101	NaN	~ 6.4769	$\sim 1.18 \times 10^{16}$
11110	NaN	~ 8.306	$\sim 5.6387 \times 10^{78}$
11111	NaN	~ 11.0108	$\sim 10^{1758}$

Arithmetic

Take LI numbers $X = l + f$, $Y = m + g$, $Z = n + h$, $X \geq Y \geq 0$.

We wish to compute $\phi(X) \pm \phi(Y) = \phi(Z)$.

Following sequences have to be computed:

$$a_{l-1} = e^{-f}, \quad a_{j-1} = e^{-1/a_j},$$

$$b_{m-1} = a_{m-1}e^g, \quad b_{j-1} = e^{-(1-b_j)/a_j} \text{ (if } m \geq 1), \text{ and}$$

$$c_0 = 1 - b_0, \quad c_j = 1 + a_j \ln(c_{j-1}).$$

Finding first $c_j < a_j$ and some additional operations provides the sum.

Multiplication and division are implemented through addition with minor pre/post processing of the inputs.

SLI arithmetic has a few modifications to take into account the reciprocal.

Version 0.1 of the package implements object `sli` that has the following properties.

- `level_bits`: number of bits assigned to the level (p_l).
- `index_bits`: number of bits assigned to the index (p_i).
- `sign`: sign bit.
- `reciprocal`: reciprocal bit.
- `level`: stored as binary64, limited to $[1, 2^{p_l}]$.
- `index`: stored as binary64, rounded to fixed point with $\varepsilon = 2^{-p_i}$.
- `value`: a binary64 image of the LI number.

All calculations for the LI arithmetic are in binary64.

We have overloaded the following MATLAB operators for `sli.m` objects:

```
plus minus uminus uplus times mtimes
rdivide ldivide lt gt le ge ne eq
transpose
```

Not yet supported:

```
mrdivide mldivide power mpower and or
not colon ctranspose horzcat vertcat
```

Examples: initializing formats

```
>> x=sli(2,12)
```

```
x =
```

```
sli with properties:
```

```
level_bits: 2
```

```
index_bits: 12
```

```
sign: []
```

```
reciprocal: []
```

```
level: []
```

```
index: []
```

```
value: []
```


Examples: setting values; approach 1

```
>> x=x.set_val(pi)
x =
sli with properties:
  level_bits: 2
  index_bits: 12
      sign: 0
reciprocal: 1
      level: 2
      index: 0.135253906250000
      value: 3.141899100868418
```

Examples: setting values; approach 2

```
>> x=x.set_sli(0, 1, 2, 0.9)
x =
sli with properties:
  level_bits: 2
  index_bits: 12
           sign: 0
reciprocal: 1
           level: 2
           index: 0.899902343750000
           value: 11.697357098484744
```

Examples: setting values; approach 2

```
>> x=x.set_sli(0, 1, 4, 0.9)
x =
sli with properties:
  level_bits: 2
  index_bits: 12
           sign: 0
reciprocal: 1
           level: 4
           index: 0.899902343750000
           value: Inf
```

Examples: plus

```
>> x=x.set_val(pi);
```

```
>> x+x
```

```
ans =
```

```
sli with properties:
```

```
level_bits: 2
```

```
index_bits: 12
```

```
sign: 0
```

```
reciprocal: 1
```

```
level: 2
```

```
index: 0.608642578125000
```

```
value: 6.283548393727487
```

Examples: minus

```
>> y=x.set_val(1);
```

```
>> x-y
```

```
ans =
```

```
sli with properties:
```

```
level_bits: 2
```

```
index_bits: 12
```

```
sign: 0
```

```
reciprocal: 1
```

```
level: 1
```

```
index: 0.7617187500000000
```

```
value: 2.141954542903604
```

Examples: uminus

```
>> x=x.set_val(pi);  
>> -x  
ans =  
sli with properties:  
  level_bits: 2  
  index_bits: 12  
    sign: 1  
  reciprocal: 1  
    level: 2  
    index: 0.135253906250000  
    value: -3.141899100868418
```

Examples: mtimes and transpose

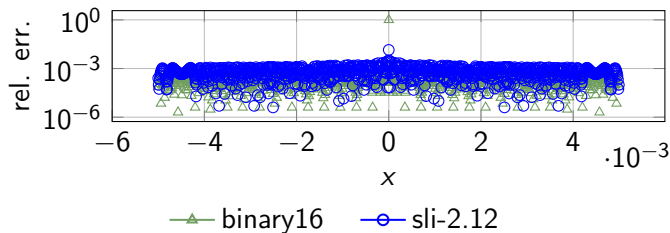
```
>> x = sli.zeros(2,3,2,12);
>> [rows, cols] = size(x);
>> for i = 1:rows
for j = 1:cols
x(i,j) = x(i,j).set_val(2);
end
end
>> y = x*x';
>> y(1,1)
ans =
    sli with properties:
    level_bits: 2
    index_bits: 12
        sign: 0
    reciprocal: 1
        level: 2
        index: 0.910400390625000
        value: 12.004930399103673
```

Examples: relation operations

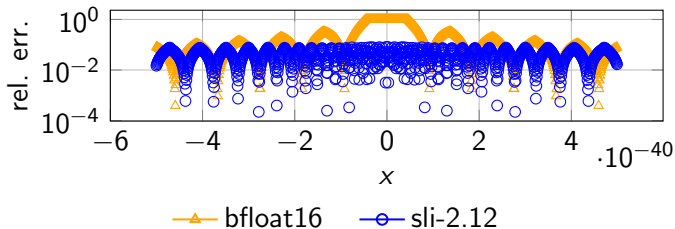
```
>> x = sli;
>> x = x.set_val(rand);
>> half = sli;
>> half = half.set_val(0.5);
>> if (x > half)
temp = x - half;
else
temp = x + half;
end
>> temp
temp =
  sli with properties:
    level_bits: 2
    index_bits: 12
           sign: 0
    reciprocal: 0
           level: 1
           index: 0.901855468750000
           value: 0.405815981871347
```


Experiments with `sli.m`

Round 1000 binary64 values around 0 with steps of 10^{-5} .

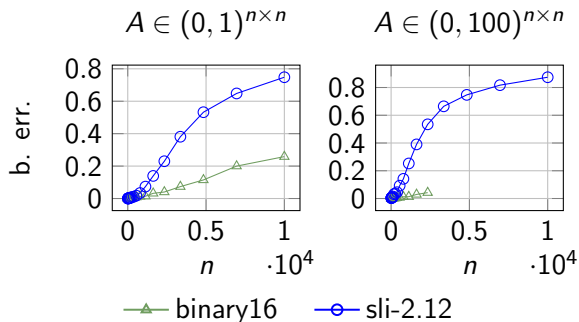


Take bfloat16 arithmetic and steps of 10^{-42} .



Experiments with sli.m

Compute the backward error of $y = Ax$ with $x \in (0, 1)^n$: $\max_i \frac{|\hat{y}_i - y_i|}{(|A||x|)_i}$.



`sli.m v0.1`

Available at <https://github.com/north-numerical-computing/level-index-simulator>

Future extensions to this toolbox:

- Decide if negative zero is needed,
- Port to GNU Octave,
- Using binary64 hardware to simulate SLI: analyse limitations.



C. W. Clenshaw and F. W. Olver

Beyond floating point

J. ACM., 31:2. Apr., 1984.



C. W. Clenshaw and P. R. Turner

The symmetric level-index system

IMA J. Numer. Anal., 8:4. Oct., 1988.



F. W. Olver

A closed computer arithmetic


IEEE 8th Symp. Comp. Arith., May 1987.





F. W. Olver and P. R. Turner

Implementation of level-index arithmetic using partial table look-up

IEEE 8th Symp. Comp. Arith., May 1987.

 P. R. Turner
A software implementation of SLI arithmetic
IEEE 9th Symp. Comp. Arith., Sep. 1989.

 J. H. Kwak and E. Swartzlander
An implementation of level-index arithmetic based on the low latency
CORDIC system
32nd Asilomar Conf. Sign. Sys. Comp., Nov. 1998.

 X. Shen and P. Turner
A hybrid number representation scheme based on symmetric
level-index arithmetic
Proc. 2006 Int. Conf. Sci. Comp. (CSC 2006), Jun. 2006.