

A Hybrid Number Representation Scheme Based on Symmetric Level-Index Arithmetic

Xunyang Shen & Peter R. Turner
Mathematics and Computer Science,
Clarkson University, Box 5815,
Potsdam, NY 13699-5815

Abstract - *Symmetric level-index arithmetic was introduced to overcome the problems of overflow and underflow in scientific computations. A hybrid SLI-FLP number system, together with some recent improvements of SLI arithmetic can result in a sound implementation of over/underflow free computer arithmetic. The hybrid arithmetic automatically switches between FLP and SLI, in order to achieve both efficiency and robustness for the system. The number representation scheme and algorithms will be discussed briefly in this paper, followed by the description of a software implementation and its successful application to a turbulent combustion problem.*

Keywords: symmetric level-index, computer arithmetic, hybrid, overflow, underflow, turbulent combustion.

1 Introduction

Although IEEE floating point standard 754/854 [8] is satisfactory for many purposes, there are fields where much larger or much smaller quantities are required to achieve useful computational results, see [9], [7], and [2] for examples. In order to eliminate overflow/underflow problems, extensions or modifications of the floating point system, and alternative real number representations are proposed in many publications.

The level-index (LI) number representation and its algorithms for performing the basic arithmetic operations are introduced by Clenshaw and Olver in [3] and [4]. The symmetric form (SLI) was developed by Clenshaw and Turner in [6]. Using repeated logarithms, symmetric level-index arithmetic essentially allows infinite number range, with a mathematical elegance and simplicity. See the survey [5] for the detailed description.

In this paper, we focus on establishing the SLI system as a practical framework for scientific computing. To perform arithmetic operations is relatively expensive in the SLI system, and this is the major drawback of SLI number representation. A lot of work has been done towards faster algorithms and better implementations, see examples in [14], [12], [15], [16], [1], and [13]. Based on some of these techniques of SLI arithmetic, we propose a hybrid

number system, which takes both the advantages of SLI arithmetic and FLP (floating point) arithmetic.

1.1 Background

Some basic information about SLI number representation is collected here for an easier description of our scheme. The idea of level-index system is to represent a positive real number X as

$$X = \exp(\exp(\dots \exp(f))) \quad (1)$$

where $0 \leq f < 1$ and the process of exponentiation is performed l times. l and f are the *level* and *index* of X respectively. As the level increases, the LI number grows rapidly. A level 7 number could be as large as $10^{(10^{(10^{(10^{1000000}))})})}$, which would be big enough for any realistic computing applications.

$x = l + f$ is the LI image of X . The mapping function, which is called the *generalized logarithm function*, is defined formally as

$$x = \psi(X) = \begin{cases} X, & 0 \leq X < 1, \\ 1 + \psi(\ln X), & X \geq 1; \end{cases} \quad (2)$$

and the inverse mapping function, the *generalized exponential function*, is defined by

$$X = \phi(x) = \begin{cases} x, & 0 \leq x < 1, \\ e^{\phi(x-1)}, & x \geq 1. \end{cases} \quad (3)$$

The symmetric form is used to allow negative exponents, if the magnitude of X is less than 1. In other words, we first take the reciprocal of a small magnitude number, and then find the SLI image for the reciprocal. Using one bit for the reciprocal sign enables the representation of extremely small numbers, while a sign bit allows negative numbers.

Due to the mathematical nature of SLI number representation, it is only necessary to devise a fast algorithm for addition/subtraction, because other basic operations are essentially equivalent operations at different levels. The exponential function and logarithmic function are particularly simple for SLI arithmetic, which is defined by repeated logarithms.

1.2 Motivations for a Hybrid System

There are two facts that inspire us to develop a hybrid number system.

Firstly, overflow/underflow in FLP arithmetic does not happen for most computations in a double precision system. SLI arithmetic does not provide any practical benefits to such computations, since FLP arithmetic is faster in general. On the other hand, some applications have to deal with especially large or small quantities, on which SLI arithmetic could help. The purpose of a hybrid system is to take the advantages of both.

Secondly, some recent research in [13] suggests that an approximation method based on the Taylor expansion could offer considerable improvement to the performance of SLI arithmetic operations, when one of the numbers is sufficiently large (or small). The Taylor approximation method benefits both ends of the number range, while using FLP arithmetic benefits the middle range. Combining the benefits, a hybrid system with Taylor approximation method avoids the full SLI operation algorithms for nearly all computations, and thus provides much higher efficiency.

In the following sections, the SLI-FLP hybrid number representation will be described in detail, followed by some brief explanations of its algorithms, including their performance advantages. An application will show how this hybrid system benefits real life computational problems.

2 Number Representation

Since double precision (64 bits) arithmetic is widely adopted in scientific computing and in floating point unit design, we consider only the number representation with double precision in this paper.

The idea of this hybrid system is to represent a number in floating point form, if the number is in a defined range F where overflow/underflow is not a problem for FLP arithmetic, and to represent the number in symmetric level-index form otherwise. In order to be consistent with the IEEE standards, we may define $F = [2^{-511}, 2^{511}]$, whose exponent range is roughly half exponent range of IEEE floating point representation without considering gradual underflow. This choice of F guarantees that there will be no overflow/underflow when performing the four basic arithmetic operations in IEEE standard algorithms.

To store a real number within the reduced 64-bit FLP range F , the IEEE standards are adopted in this hybrid system, except that the first two bits of exponent can now only be binary number 01 or 10, because the IEEE standards store exponents with bias $2^{10}-1$.

To store the SLI image of a real number, there are four pieces of information that need to be included -- level, index, sign, and reciprocal sign. Two bits are used for signs and three for the level [11], leaving 59 bits for the index.

In this hybrid number representation, we may use only two bits for the level, noticing that $\psi(2^{511}) \approx 4.57$, which means the level of an SLI number in this system cannot be 1, 2 or 3. The saved bit, together with the reciprocal sign bit, can be used as the indicator, telling which representation scheme the current number is using.

The number representation is visualized in the figure below. The first bit is the number sign, which is shared in both number forms. The second and third bits together are the representation indicator. 01 or 10 means the number is stored using FLP representation, 00 means an SLI number with a negative reciprocal sign, and 11 means an SLI number with a positive reciprocal sign.

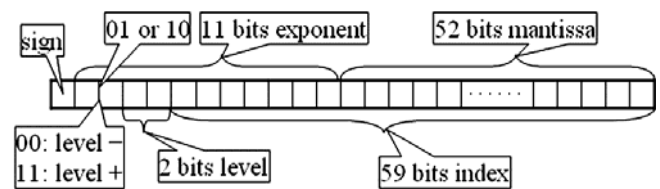


Fig. 1: Double precision SLI-FLP hybrid number representation

If the number is in FLP form, the 2nd bit through the 12th are the biased exponent part, while the last 52 bits store the mantissa. This is exactly the same format as defined in IEEE standards, so the algorithms designed for these standards preserve their effectiveness.

If the number is in SLI form, the 4th and 5th bits are used for the level. The value should be the level value reduced by 4, as long as the reciprocal sign is positive. In the case of numbers with a negative reciprocal sign, 1's complement is taken for these two bits, so that the ordering of the real number will follow the same rule as in the IEEE standards.

This way we successfully packed the hybrid number representation into a 64-bit word with very little storage waste. The efficiencies of different basic algorithms are considered and balanced in this representation scheme, in order to achieve the best overall performance. Those algorithms will be described in the next section.

3 Algorithms

Algorithms for various operations in this hybrid system have been investigated. Most of them are no more

complicated than the ones in pure SLI arithmetic. However, the performance will be improved because of the adoption of FLP number representation. We will briefly discuss some of the algorithms here, and describe a software implementation with experimental results in the next section.

3.1 Relational Operations

The design of the hybrid number representation preserves the real number ordering in FLP arithmetic. So the relational operations can be performed the same way as in FLP arithmetic, which is much faster than either a pure SLI or a mixed SLI-FLP number comparison.

Observations on the SLI addition algorithm suggest that these relational operations have some impact on the performance of arithmetic operations. A fast relational operation algorithm noticeably contributes to the overall performance of the system, and this is a very important reason to have the current representation design.

3.2 Arithmetic Operations

The performance of arithmetic operations is our major concern in this hybrid system, since the low efficiency of performing arithmetic operations is an obvious obstruction to practical uses of SLI arithmetic. The hybrid number representation gains multiple advantages on the performance of arithmetic operations, comparing to the pure SLI arithmetic.

As stated before, the overall performance can be evaluated by the performance of addition/subtraction algorithm in SLI arithmetic. There can be three different situations in the hybrid system - pure FLP additions, pure SLI additions, mixed SLI-FLP additions.

When both the addends are within the reduced FLP range $[2^{-511}, 2^{511}]$, the same algorithm can be used as in an IEEE floating point system, except that the 2nd and 3rd bits of each addends must be checked before performing the addition, and the result of addition will be examined against the reduced FLP range in order to be stored properly. The overhead will only slow down the FLP addition by little, as long as the result does not exceed the reduced FLP range.

When both the addends are beyond the reduced FLP range, the SLI addition algorithm will be performed. Refer to [13] for the details of the algorithm with Taylor approximation. The algorithm allows more parallel implementation if it is modified similarly as in [1]. In summary, the algorithm computes the a - and b -sequences defined as follows,

$$a_j = \frac{1}{\phi(x-j)}, j = l_x - 1, l_x - 2, \dots, 0; \quad (4)$$

$$b_j = \begin{cases} \frac{\phi(y-j)}{\phi(x-j)}, & (\text{large}) j = l_y - 1, l_y - 2, \dots, 0, \\ \frac{1}{\phi(y-j)}, & (\text{mixed}) j = l_y - 1, l_y - 2, \dots, 0, \\ \frac{\phi(x-j)}{\phi(y-j)}, & (\text{small}) j = l_x - 1, l_x - 2, \dots, 0; \end{cases} \quad (5)$$

and then decides if the full algorithm or the Taylor approximation method should be used. Here l_x and l_y are the levels of two addends respectively, and $X = \phi(x)^{\pm 1}$ has a greater magnitude than $Y = \phi(y)^{\pm 1}$. The large/small addition is defined as the case when both addends have positive/negative reciprocal signs, and the mixed addition is the case when two addends have different reciprocal signs. To complete the addition, the chosen method is performed. A saving in the logical structure of the full algorithm due to the hybrid number representation will be described later.

When one addend is represented in SLI form but the other is in FLP form, the algorithm could be simpler than the one for pure SLI additions. If the SLI addend has a smaller magnitude than the FLP addend, it is safe to convert the SLI addend into FLP form and do a pure FLP addition. Otherwise the large case of SLI addition applies, and b_0 can be obtained directly as the product of a_0 and the FLP addend.

The addition algorithm of the hybrid system divides additions into the above cases, and is significantly faster than the addition algorithm in a pure SLI system for the following reasons.

The pure FLP addition algorithm is used where overflow/underflow can not happen in the floating point system. This keeps the advantage of SLI arithmetic whenever it is useful, and at the same time takes benefits from the higher efficiency of FLP arithmetic.

The Taylor approximation scheme can cover most of the addition cases where the SLI number representation is used. Based on the Taylor expansion, the approximation method works when the operation changes the larger magnitude addend only a little. This is more likely to happen for high level SLI additions. Starting from the threshold picked for the SLI representation, the Taylor approximation is applicable almost everywhere except when the two addends are very close together.

The algorithm for mixed SLI-FLP addition cases is less complicated than the pure SLI addition algorithm. The time saving is considerable without generating the whole b -sequence when one of the addends is an FLP number, and this benefits both the full algorithm and the Taylor approximation.

All *flipover* cases are avoided, and thus the logical structure of the full SLI addition algorithm is simplified. A flipover addition is defined as the case where the reciprocal sign of the larger addend is changed after an SLI addition. However, any two SLI numbers with different reciprocal signs are sufficiently distant in this hybrid system to avoid flipover additions, considering the limited precision of a 64-bit number representation. In the full addition algorithm of SLI arithmetic, flipover cases are handled separately from the cases where flipover does not occur. The removal of the flipover cases makes the full algorithm simpler and thus faster.

Those improvements together offer much better performance of arithmetic operations in the hybrid system than in pure SLI arithmetic. The figure below visualizes the effectiveness of each improvement, where both coordinates stand for numbers in SLI form. Since it is justified to assume the latter addend y is always less than the former addend x for a positive addition, the region below the straight line $y = x$ represents all the possible cases of positive additions. The domain of addition is decomposed into several parts. In each part a different addition algorithm applies, and the deeper shaded the region is, the faster the algorithm will be.

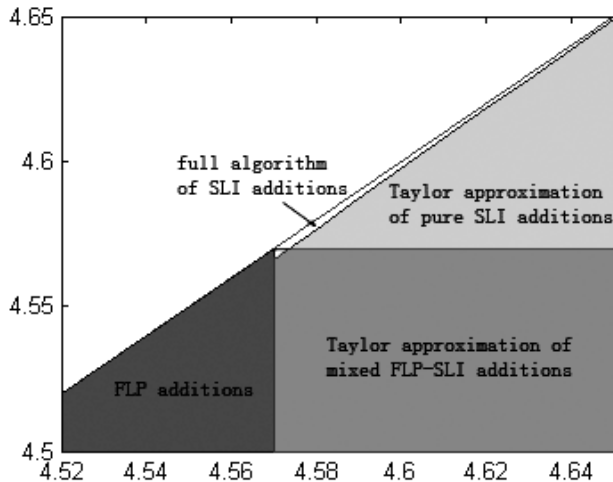


Fig. 2: Domain decomposition of additions in the SLI-FLP hybrid system

In order to make visible the smallest component, where the full algorithm needs to be performed, only a very small portion of the addition domain is included in this figure. The deepest shading represents the domain of pure FLP additions, while the other shaded regions are where the Taylor approximation is applicable, and the computation of the b -sequence is avoided in the lower region. Even when the full algorithm is required, the performance can be better than in pure SLI arithmetic because of the simplification of algorithm logical structure.

In the next section, we will describe a software implementation of this hybrid system and its application to an example of combustion problem. The computation results and performance comparisons demonstrate how this hybrid system could benefit certain computational problems.

4 Software Implementation and Applications

The software implementation we have practices most of the nice features of the hybrid system. However, due to the limitation of a high level computer language, it is not possible to implement all the aspects mentioned above.

The hybrid system was coded in C++ as a class, using FLP arithmetic operations in ANSI C++ as internal operations. The purpose of this program is not only for experiments to support theories about SLI arithmetic, but also for usages of real life scientific computations before SLI chips can be built. So the system was implemented in a way that is very easy to use.

For better performance in our software implementation, SLI numbers are not packed into 64 bits. A structure with total length of 72 bits is used instead. This prevents inefficient bit manipulations needed using a typical floating point register, but with the trade-off of losing the nice property of the same number ordering as in FLP representation.

Most platforms of C++ do not support the extended double precision FLP arithmetic. Based on the standard FLP arithmetic operations, the C++ implementation cannot have 59 bits of SLI number index, which exceed the computational capability of using 52-bit mantissa. The little loss of precision would not be a problem for most applications.

The algorithms for arithmetic operations were coded as described in the previous section. Some experiments using the software implementation were performed to check the correctness of the design, and to reflect on the efficiency of a hardware implementation, which is the ultimate goal.

The computational example we used is an aerospace science model problem in turbulent combustion, in which two chemical species (fuel and oxidizer) mix and react in a vortex field. The problem is known as the Marble problem after its originator, F.E. Marble [10]. A solution using SLI arithmetic was published by Lozier in [9]. We repeat some of the process here for a validation of the hybrid system and for time comparisons.

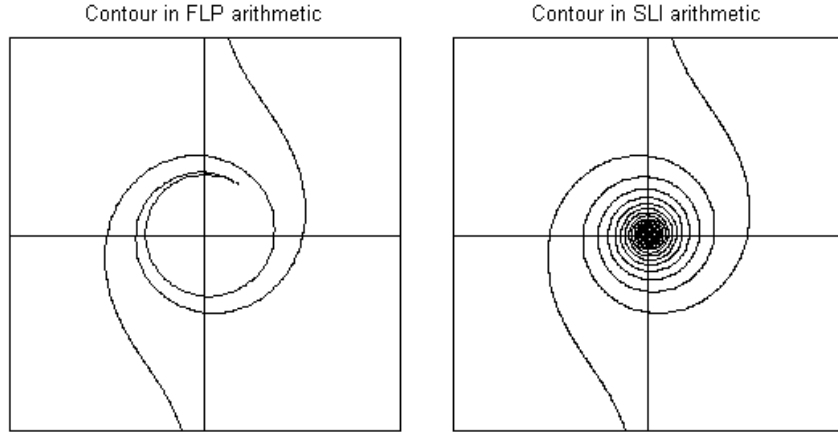


Fig. 3: Contours of the Marble solution

The Marble solution has some properties that can be shown by contour plotting. The level curves of Z exhibit radial symmetry, and the curve $Z = 1/2$ may be regarded as having two branches, joined at the origin. We compute the values of $Z-1/2$ in a C++ program, and then draw the contour plots above in MATLAB.

The left stoichiometric contour is based on the data computed in double precision FLP arithmetic. The central region is missing, and the radially symmetric spiral behavior is not observed, due to the underflow when the Z values are computed. Examinations of the data show that the central region contains a lot of zeros. In contrast, we expect $Z = 1/2$ only at the origin. The data of the right contour are generated using the C++ implementation of either the hybrid system or pure SLI arithmetic. This contour clearly displays the properties of the Marble solution as expected.

Single precision is enough to demonstrate the usefulness of SLI arithmetic for the Marble problem, as described in [9]. However, double precision is used here for more convincing performance comparisons, since double precision is now the default precision level for most scientific computations.

Table 1: Time costs of computing the Marble solution

Tolerance	10^{-308}	10^{-100}	$10^{-1,656,520}$
FLP	0.80 s	0.60 s	∞
Hybrid	2.87 s	1.18 s	231.22 s
SLI	6.95 s	4.73 s	397.33 s

The table above records the time costs to compute the Marble solution using the same algorithm, but with different arithmetic systems and different error tolerances at a certain step. All the computations are done on a desktop with a 3GHz Pentium 4 CPU and GNU Compiler in LINUX. Z values are computed on a 301 by 301 grid.

In order to obtain the right contour in Fig. 3, an error tolerance beyond the range of any commercial FLP arithmetic needs to be used. Without any analysis on the possible numerical results of the Marble problem, we chose a tolerance of $\phi(5.0)^{-1}$, which is about $10^{-1,656,520}$, as shown in the last column of the table. Since the computation in FLP arithmetic fails to reach this tolerance level, ∞ is recorded as its time cost.

The error tolerance in the first column is 10^{-308} , which is close to the threshold where underflow would occur in double precision FLP arithmetic. With this tolerance, FLP arithmetic gives almost the best results it could give for the Marble problem. The results are plotted in the left contour of Fig. 3. The second column has the tolerance of 10^{-100} , which is within the reduced FLP range of the hybrid system. Using this tolerance is to compare the performance in the cases where overflow/underflow is never a problem in computations.

The comparisons here are not very fair for the hybrid system and SLI arithmetic, because the FLP arithmetic is built in a floating point unit, but the other two are just software implementations. However, the hybrid system still shows decent efficiency. Its advantage compared to pure SLI arithmetic is obvious, since the costs are much lower for all cases. The performance of the hybrid system is acceptable even comparing with pure FLP arithmetic, when a comparison is appropriate. The hybrid system costs less than twice the FLP arithmetic does, when the computation does not go beyond the reduced FLP range in the hybrid system. The ratio of the costs looks even better than expected, since a real life problem usually spends a large portion of time on operations other than real number computations.

The Marble problem requires extremely large exponent range in computations to avoid the falsification of the contour, and hence is very difficult to solve using

techniques such as scaling. Some other problems with overflow/underflow problems may be solved by developing alternative algorithms. An example would be the condition number computation of triadiagonal matrices described in [7]. But with the help of fast overflow/underflow free arithmetic, to manually deal with overflow/underflow can be avoided.

5 Conclusions

To fully reveal the potential of SLI arithmetic is the goal of our research. The hybrid system proposed in this paper is essentially overflow/underflow free, because of the almost infinite number range of SLI representation. It also uses FLP arithmetic for computations within a reduced number range of the double precision IEEE arithmetic, so that it can have better performance than a pure SLI system by taking the advantage of the FLP arithmetic.

The details of the number representation scheme and algorithms to perform arithmetic and relational operations are discussed, in order to show the performance improvements provided by the hybrid system. The number representation and the arithmetic are implemented in C++ as much as a software implementation allows.

The C++ implementation is applied on the Marble problem and good results are obtained. According to the performance comparison, we may conclude that a polished software implementation of the hybrid system is ready to be used in computational problems where speed is not crucial. If implemented in hardware, the hybrid system can be expected to have similar efficiency as a pure FLP system, as long as the computations are within the reduced FLP range. But unlike the FLP arithmetic, it will no longer suffer from overflow/underflow problems.

References

[1] M.A. Anuta, D.W. Lozier, N. Schabanel & P.R. Turner, 1996, Basic Linear Algebra Operations in SLI Arithmetic, Computing and Applied Mathematics Laboratory (Applied and Computational Mathematics Division), NIST Report, NISTIR5811, National Institute of Standards and Technology, Gaithersburg, MD.

[2] H.P. Chen, X. Sun, H.X. Chen, Z.Q. Wu & B.H. Wang, "Some Problems in Multifractal Spectrum Computation Using a Statistical Method", *New J. Physics* 6, 84, 2004.

[3] C.W. Clenshaw & F.W.J. Olver, "Beyond Floating Point", *J. ACM* 31, 319-328, 1984.

[4] C.W. Clenshaw & F.W.J. Olver, "Level-Index Arithmetic Operations", *SIAM J. Numer. Anal.* 24, 470-485, 1987.

[5] C.W. Clenshaw, F.W.J. Olver & P. R. Turner, "Level-Index Arithmetic: an Introductory Survey", Numerical Analysis and Parallel Processing (P. R. Turner, ed.), Springer-Verlag, Berlin, 95-168, 1989.

[6] C.W. Clenshaw & P.R. Turner, "The Symmetric Level-Index System", *IMA J. Numer. Anal.* 8, 517-526, 1988.

[7] I.S. Dhillon, "Reliable Computation of the Condition Number of a Triadiagonal Matrix in $O(n)$ Time", *SIAM J. Matrix Anal. APPL.* 19, 776-796, 1998.

[8] IEEE 754/854, "IEEE Standard for Floating Point Arithmetic", ANSI/IEEE Std. Vols 754/854, IEEE, New York.

[9] D.W. Lozier, "An Underflow-induced Graphics Failure Solved by SLI Arithmetic", Proc. ARITH11, IEEE Computer Society, Windsor, Canada, 10-17, 1993.

[10] F.E. Marble, "Growth of a Diffusion Flame in the Field of a Vortex", Recent Advances in Aerospace Sciences (C. Casci, ed.), 1985.

[11] F.W. Olver, "A Closed Computer Arithmetic", Proc. ARITH8, M.J. Irwin & R. Stefanelli, Eds, IEEE Computer Society, Washington, DC, 139-143, 1987.

[12] F.W. Olver & P.R. Turner, "Implementation of Level-Index Arithmetic Using Partial Table Look-up", Proc. ARITH8, IEEE Computer Society, M.J. Irwin & R. Stefanelli, Eds, Washington, DC, 144-147, 1987.

[13] X. Shen & P.R. Turner, "Taylor Approximation for Symmetric Level-Index Arithmetic Processing", *IMA J. Numer.*, 2006, (web access at <http://imanum.oxfordjournals.org/cgi/content/abstract/drl004?ijkey=JM2zizTtXl5SVhX&keytype=ref>).

[14] P.R. Turner, "Towards A Fast Implementation of Level-Index Arithmetic", *Bull IMA* 22, 188-191, 1986.

[15] P.R. Turner, "A Software Implementation of SLI Arithmetic", Proc. ARITH9, M. Ercegovac & E. Swartzlander, Eds., IEEE Computer Society, Washington DC, 18-24, 1989.

[16] P.R. Turner, "Implementation and Analysis of Extended SLI Operations", Proc. ARITH10, P. Kornerup & D.W. Matula, Eds., IEEE Computer Society, Washington DC, 118-126, 1991.