Asymmetric and Compressed Logarithmic Number Systems for a Multimedia Coprocessor (Invited Paper)

Mark G. Arnold (marnold@eecs.lehigh.edu)
Lehigh University; Bethlehem, PA 18015 USA

Abstract

The Logarithmic Number System (LNS) represents a real number with its sign and the logarithm of its absolute value. Inexpensive logarithmic arithmetic has shorter word length than fixed point. Conventional LNS is symmetrical: every reciprocal is representable. This paper proposes asymmetrical LNS (one sixteenth to 2048) with acceptable visual quality for MPEG decoding that saves one bit compared to conventional LNS. Novel compression saves another bit with only slight image degradation and allows 8 by 8 matrices to fit in 64 (not 128) bytes. The proposed LNS Inverse Discrete Cosine Transform (IDCT) coprocessor is compatible with conventional byte-sized packed-arithmetic SIMD architectures.

1. Introduction

The usual choice for multimedia and signal-processing applications, such as MPEG decoding, is fixed-point arithmetic, but its drawbacks include costly multiplication and long word lengths to cover the dynamic range needed by such applications. The Logarithmic Number System (LNS) is an alternative that offers less expensive multiplication and possibly much shorter word-length requirements for equivalent dynamic range. A conventional signed LNS represents a real number with the sign bit of the value and the logarithm of its absolute value. In two's-complement LNS when the absolute value is between 0.0 and 1.0, the logarithm is negative; otherwise it is positive. Except for $1...1_2$ (which represents 0.0), the reciprocals of every value can be represented in such a symmetrical two's-complement LNS.

A different approach [5] for LNS defines a scaling parameter, τ. When τ is chosen such that it is equal both to the maximum possible value and to the reciprocal of the minimum possible value, a symmetric LNS is defined with properties (word length, range, precision) similar to the two's-complement implementation, except that the logarithm is biased similar to the exponent field of IEEE-754 floating point. This makes LNS ordering identical to that of unsigned (rather than two's complement) integers. Using $\tau \neq 1.0$ introduces complications, such as the requirement that the hardware subtract $\log(\tau)$ when forming a product.

If the minimum possible value is not the reciprocal of the maximum possible value, an asymmetrical LNS can be defined. The arithmetic algorithms, including the correction by $\log(\tau)$, remain the same. For example, Coleman et al. [3] have implemented a digital filter using an asymmetric LNS (under their misleading label of "extended precision"), in which τ is chosen to be approximately the square of what would be selected for a symmetrical LNS, resulting in a system that can only represent absolute values less than one.

This paper examines a typical video decoding application in which an asymmetric range is useful. In the MPEG standard [6], inputs to the two-dimensional Inverse Discrete Cosine Transform (2D-IDCT) can be clipped to be between 2^{-4} to 2^{12}, which implies $\tau = 2^4$. This gives an asymmetrical dynamic range with truncation of intermediate results less than $1/\tau$. Since the asymmetrical dynamic range covers sixteen binades, only four bits are required for the integer part of the logarithm. Including the sign bit of the value and F bits of precision, the total word width is $F + 5$. In contrast, the conventional two's-complement approach (with an additional logarithm sign bit) [1] and the equivalent symmetrical-offset-LNS approach ($\tau = 2^{12}$) require $F + 6$ bits.

The 2D-IDCT is implemented by eight 1D-IDCTs whose results are stored in a 64-element memory so that an 8×8 matrix transpose operation can occur before eight final 1D-IDCTs occur. Experiments seem to indicate the visual quality with $(F+5)$-bit asymmetrical LNS in this memory is almost as good as $(F+6)$-bit conventional symmetrical LNS.

Additional compression can cut the word length requirement to F + 4 bits for each element of the matrix. Compression like [2] degrades the visual result more than is acceptable, but the novel asymmetrical LNS compressions proposed here achieves this word length with only slight image degradation. Since earlier experiments indicate F = 4 may be acceptable for some MPEG applications [1], the proposed approach allows the 8×8 matrix to fit snugly in 64 bytes, rather than the 128 bytes needed for a conventional 16-bit fixed-point IDCT. Since most Single Instruction Multiple Data (SIMD) processors have packed-arithmetic instructions with granularity on byte (not 9- or 10-bit) boundaries, the proposed compressed asymmetrical LNS offers a convenient way to interface an LNS-IDCT coprocessor into a conventional architecture.

2. Logarithmic Number Systems

In 1975, Swartzlander and Alexopoulos [5] described the *Sign Logarithm Number System* (SLNS), which represents the magnitude of values with their base-b logarithms and a concatenated sign bit. As described in [5], the SLNS includes a constant scaling factor, τ, and a special representation for zero. The purpose of the scaling factor is to prevent the logarithm from being negative when the number being represented is in the interval between 0 and 1. The factor τ acts analogously to the offset arithmetic often used in the exponent of a floating-point number. Since Swartzlander and Alexopoulos suggest 2 as the base, the integer part of the logarithm is exactly what one would find as a floating-point exponent. The idea of this offset, and the algorithms associated with it, have reoccurred in the LNS literature several times. One could view Napier's logarithm in the framework of τ. Most recently, Coleman et al. [3] describe a similar set of algorithms to those described by Swartzlander but used for a different purpose (scaling to avoid underflow), which they mistakenly label as "extended precision." This paper will present yet another novel use of these same LNS scaling techniques to reduce the switching activity (and, it is hoped, the power consumption) by the introduction of an *asymmetrical* LNS, in which not every representation has a reciprocal.

2.1 SLNS Arithmetic

Arithmetic operations for the values X and Y requires hardware that operates on their representations:

$$x = \log_b(\tau|X|), \qquad y = \log_b(\tau|Y|) \tag{1}$$

Thus, $\log_b(\tau|XY|) = x + y - \log_b(\tau)$ and $\log_b(\tau|X/Y|) = x - y + \log_b(\tau)$. Since the values are already represented as logarithms, a simple addition or subtraction computes the product or quotient. No antilogarithm hardware is required, since values continue to be represented as logarithms. If $\tau \neq 1$, an additional subtraction or addition of a constant, $\log_b(\tau)$ is required. The sign of the product or quotient is simply the exclusive OR of the sign bits of the numbers being processed.

By making multiplication and division easy, the sign logarithm number system makes addition and subtraction more difficult than in fixed-point arithmetic. The LNS hardware needs to exploit the following:

$$\tau\Big(|X| + |Y|\Big) = \tau|X| \cdot \left(1 + \frac{\tau|Y|}{\tau|X|}\right). \tag{2}$$

To compute $X+Y$, the hardware performs the division (by subtraction of x from y), increments the quotient, and then multiplies the result by X (by addition). Of these operations, incrementation is the most difficult. This requires computation of $s_b(z) = \log_b(1 + b^z)$ for *sums.* A similar function, $d_b(z) = \log_b|1 - b^z|$, works for *differences.* A key observation in [5] is that the algorithm is no more complex when $\tau \neq 1$.

2.2 Level Index

Clenshaw and Olver [2] describe what they call the level index (LI) number system for representing very large numbers. The LI number system is to the sign logarithm number system as floating point is to fixed point. The LI number system uses iterated logarithm computation in the way a floating-point system uses iterated right shifting. The LI number system is not the same as SLNS. The representation of numbers between 1 and b in the LI number system is the same as for the SLNS, except that the LI number system would require that a level index of 1 be concatenated to the LI word. An LI-like system will be considered later in this paper.

3. MPEG decoding

The numerical heart of MPEG decoding is the Two-Dimensional Inverse Discrete Cosine Transform (2D-IDCT) [6]:

$$f(x,y) = \sum_{u=0}^{7}\sum_{v=0}^{7} F(u,v) \cdot \frac{C_u}{2} \cdot \frac{C_v}{2} \cdot \cos\left[\frac{(2x+1)u\pi}{16}\right] \cdot \cos\left[\frac{(2y+1)v\pi}{16}\right], \tag{3}$$

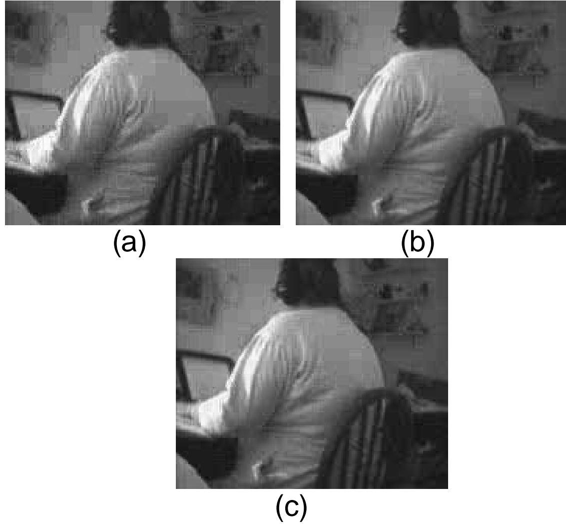

Figure 1. Effect of: (a) symmetric $F = 3$ LNS (conventional 9-bit word), (b) symmetric $F = 4$ LNS (conventional 10-bit word) and (c) asymmetric $F = 4$ LNS (novel 9-bit word).

where $F(u, v)$ is an input matrix value previously computed using the Discrete Cosine Transform (DCT) in the encoder and $f(x, y)$ is the reconstructed output matrix value.

The input data to the IDCT are integers whose absolute values are less than 2^{12}. The cosines used in the IDCT are fractional values, with F bits of relative precision. Thus, the most natural approach for carrying out the complete IDCT algorithm, from input to output, would be to define a symmetrical signed LNS capable of representing the negative values $-2^{12} < X < -2^{-12}$ and the positive values $2^{-12} < X < 2^{12}$. For the IDCT there need not be a special representation of zero, as $\pm 2^{-12}$ are close enough to zero to serve this role. Here *signed* means that for every positive value that is exactly representable, the corresponding negative value is exactly representable. *Symmetrical* is used in the same sense as [2]: for every value that is exactly representable, the corresponding reciprocal is exactly representable. In an LNS context, this simply means a sign bit for the logarithm, in addition to the sign bit for the value.

This obvious approach to cover the dynamic range of the IDCT requires a signed LNS word consisting of a sign bit that reflects the sign of the input integer, a sign bit for the logarithm (known to be positive at the start since the input is integer, but possibly negative in later stages after the cosines have entered into the computation), four bits for the integer part of the logarithm (reflecting the fact that $12 = 1100_2$ requires 4 bits), and F bits for the fractional part of the logarithm. Thus, $F + 6$ bits are required. The following will consider alternative LNS representations that are more compact.

4. Asymmetric LNS Format

For the purposes of the IDCT in MPEG, a dynamic range from 2^{-4} to just less than 2^{12} will be shown to be sufficient in most cases for visually acceptable results. This implies $\tau = 2^4$. Thus 2^{-4} will be represented by a logarithm whose integer part is 0000_2; one will be represented by 0100_2; 2^{11} will be represented by a logarithm whose integer part is 1111_2. Using this approach reduces the LNS word size by one bit. With this approach, $F + 5$ bits are required. This asymmetrical dynamic range does not affect the numerical values output from the IDCT unless the computation underflows 2^{-4} or overflows 2^{12}. Overflow is impossible because of the size of the inputs and of the cosines. Underflow, which occurs rarely, is indicative of a value that will be converted to the integer zero after the IDCT has completed. This paper will give empirical evidence that for the IDCT 2^{-4} is usually as good of a representation of zero as is 2^{-12}.

For example, Figure 1 (c) shows the decoded output from `ki1.mpg` (the MPEG file studied in [1]) using an asymmetrical LNS of this type with $F = 4$, compared to the same frame using a conventional symmetrical LNS with $F = 3$ (Figure 1 (a)) and $F = 4$ (Figure 1 (b)). Visually, there is almost no difference between the two $F = 4$ representations, although the symmetrical LNS in Figure 1 (b) requires 10 bits per LNS representation whereas the asymmetrical LNS in Figure 1 (c) requires 9 bits per LNS representation. Both $F = 4$ images are clearly better than the $F = 3$ in Figure 1 (a). Thus, asymmetric LNS offers better visual performance than precision reduction.

5. Compression for Storage

The matrix data operated on by the IDCT must be fetched and stored from some kind of RAM memory. If we assume the direct form IDCT, the data memory will be read eight times as often as it is written into. Several possible architectures are possible. In some architectures, the data memory may store enough macroblocks to form several pictures. The effective capacitance of such memory may be much higher than the ALU. Thus reducing the bus width to memory might be important. This section proposes novel LNS compression techniques that do this.

Figure 2. Effect of $F = 4$ compressed LNS (novel 8-bit words): (d) type-0 and (e) type-1.

The technique described in this section requires an asymmetrical LNS ALU with a width of $F + 5$ bits as described in the last section. The ALU carries out the sum of products in this format inside a register of width $F+5$. Or alternatively, the ALU could carry out the computation in symmetrical LNS of width $F + 6$, and convert to asymmetrical LNS just prior to storage. In either option, when a 1D IDCT result has been formed (a sum of eight terms), rather than storing the LNS representation in the format used by the ALU, the proposed system compresses the representation so that the data bus width to memory is $F + 4$. The format of the LNS representation would be the usual value sign bit followed by $F + 3$ bits of the compressed-logarithm field (rather than the $F + 4$ bits of the uncompressed-asymmetrical-logarithm field assumed in the last section or the $F+5$ bits of the uncompressed-symmetrical-logarithm field assumed in [1]).

There are two methods of compression considered here that create such an $F + 3$ bit logarithm field. These are known as level-index-like (type-0) or one's-complement (type-1) compression.

5.1 Level-index-like Compression

One option to achieve such compression is to treat the asymmetrical-logarithm field as an unsigned integer, and convert this to a floating-point representation with a two-bit exponent field and an $F + 1$ bit hidden-bit mantissa field. If the input integer is near zero, the result will be treated as an $F+1$ bit denormal similar to IEEE-754. Thus, there would be one bit for the value sign, two bits for the exponent field, and $F + 1$ for the mantissa, or a total of $F + 4$ bits per compressed-LNS representation.

Treating the $F + 4$ bit logarithm field of the uncompressed-asymmetrical-LNS representation as an integer means mapping this integer in the range 0 to $2^{F+4} - 1$ into a compressed representation. If we also think of this compressed representation as an integer in the range 0 to $2^{F+3} - 1$, the compression and decompression operations can be described by two integer functions: `compress_lns` and `decompress_lns`. Assuming `FFX`$= F + 1$ is the width of the mantissa, here is C code for LNS compression:

```
int compress_lns(int i){
  int expon,mant;
  if (i< (1<<FFX))
   {expon = 0; mant = i;}
  else
   {expon = (int)floor(log(i)/log(2.0))-FFX+1;
    mant = (i >>(expon - 1))&((1<<FFX)-1);}
  return (expon<<FFX) | mant;}
```

Such code is well known in floating-point implementation as *normalization*. This conventional floating-point normalization process is, in effect, also equivalent to Mitchell's approximation for the base-two logarithm [4], that is good to $F \approx 4$. As such, one could view this representation as similar to a Mitchell-based version of level-two in the LI number system [2]. In other words, this is approximately $\log_2(\log_2(\tau X))$. Like the LI system, this representation trades constant relative accuracy for an extended dynamic range. However, here the data-bus-width-compression usage of this well-known normalization process is unique, as it has not been used for LNS memory compression previously. The corresponding code for decompression is:

```
int decompress_lns(int i){
  int expon, mant;
  expon = i >> FFX;
  mant = i & ((1<<FFX)-1);
  if (expon == 0)
    return mant;
  else
    return ((1<<FFX)|mant)<<(expon-1);}
```

Each time a result is stored into memory with compression and later retrieved and decompressed, the operation $i_0 =$`decompress(compress(`i`))` occurs. In Table 1, i is the input integer, and i_0 is the result after type-0 storage and retrieval. The corresponding symmetrical ($\tau = 2^{12}$) LNS representation, $x = i/16 - 4$, is the representation before storage. From this, we can derive the real value, $X = 2^x$, corresponding to that representation. Table 1 shows examples of i and corresponding values of X. (Negative values, which have the high-order bit set, e.g., $i = 640$ for $X = -16.0$, are neither in the table nor code.) Similarly, $x_0 = i_0/16 - 4$ is the representation after retrieval (and decompression) and $X_0 = 2^{x_0}$ the real value corresponding to the (possibly perturbed) level-index-like representation. This type-0 mapping is lossless for small numbers but lossy for large numbers because least significant bits are discarded. For example, the real value 0.0743 is left undisturbed ($i = i_0 = 4$) but 346.6 ($i = 199$) is perturbed to become 304.4 ($i_0 = 196$).

Table 1. $F = 4$ **input (**i**), level-index-like (**i_0**) and one's-complement (**i_1**) integer mappings for real values (**X**,** X_0 **and** X_1**).**

i	i_0	i_1	X	X_0	X_1
0	0	3	0.0625	0.0625	0.0711
1	1	3	0.0652	0.0652	0.0711
2	2	3	.0681	0.0681	0.0711
3	3	3	0.0711	0.0711	0.0711
4	4	7	0.0743	0.0743	0.0846
...	...	...	...	...	...
64	64	67	1.	1.	1.13
65	64	67	1.04	1.	1.13
66	66	67	1.09	1.09	1.13
67	66	67	1.13	1.09	1.13
...	...	...	...	...	...
128	128	129	16.	16.	16.7
129	128	129	16.7	16.	16.7
130	128	131	17.4	16.	18.2
131	128	131	18.2	16.	18.2
...	...	...	...	...	...
196	196	196	304.4	304.4	304.4
197	196	197	317.9	304.4	317.9
198	196	198	331.9	304.4	331.9
199	196	199	346.6	304.4	346.6
...	...	...	...	...	...
236	236	236	1722.	1722.	1722.
237	236	237	1798.	1722.	1798.
238	236	238	1878.	1722.	1878.
239	236	239	1961.	1722.	1961.
240	240	240	2048.	2048.	2048.
...	...	...	...	...	...

Thus, like the LI system, small numbers are represented with greater relative precision than large numbers. For general-purpose computation, such an LI scheme is attractive as a way to avoid the catastrophic effect of overflow. However, in MPEG we know in advance than no result will overflow the rather modest limit of 2,047. Thus, the theoretical advantage of such a level-index-like scheme is minimal for MPEG.

The more pertinent issue is what the visual effects are for this representation. Figure 2 (d) shows `ki1.mpg` using the level-index-like (type-0) compressed $F = 4$ LNS described above. This can be compared against the uncompressed $F = 3$ LNS and $F = 4$ LNS in Figure 1 (a) and (b). It is apparent that using this level-index-like approach introduces significant artifacts that degrade the quality of the image. In fact, it could be argued that the image quality degrades more than it does with $F = 3$ LNS. Since the compressed $F = 4$ LNS and uncompressed $F = 3$ LNS require the same total number of bits per word, it appears the level-index-like approach is not cost effective.

5.2 Novel One's-Complement Compression

Instead of thinking of this problem in the domain of real numbers, let us return to the integer-to-integer mapping, such as illustrated in Table 1. The small numbers play little role in the visual result of MPEG files. It was pointless to preserve them. In fact, with its quantization and oddification steps, the MPEG standard does not respect the integrity of small numbers very much. In contrast, the *DC* component in the MPEG standard is usually a large number which MPEG passes through unmodified (neither oddified nor quantized). The large numbers in Table 1 are the ones we ought to preserve in a lossless way, which the level-index mapping described above fails to do. For example, using 1722 as the *DC* component instead of the correct value of 1961 seems likely to alter the appearance of the macroblock.

We need a different mapping from the uncompressed integer (0 to $2^{F+4} - 1$) into a compressed integer in the range 0 to $2^{F+3} - 1$ that preserves large numbers at the expense of small numbers. The novel compression proposed is to take the one's complement before compression and undo this after decompression. This mapping is illustrated by i_1 in Table 1. For example, here 1961 is preserved and similar large numbers are preserved, at the expense of perturbing numbers like 0.0625 to become 0.0711. It seems reasonable to assume that such perturbation of small numbers will have less effect on the visual quality of the MPEG output. To illustrate this, Figure 2 (e) shows `ki1.mpg` using one's-complement (type-1) $F = 4$ compression. This can be compared against uncompressed $F = 3$ and $F = 4$ LNS in Figure 1 (a) and (b). The artifacts from type-1 compression that result are hardly noticeable, and the image quality is clearly better than $F = 3$. Thus, type-1 compression achieves the goal of reducing the width of the memory bus to $F+4$ bits. In other words, for the $F = 4$ precision that this and earlier experiments indicate may be sufficient for some MPEG applications, it is possible to store the intermediate results from the first 1D-IDCT in a byte-wide format, rather than the double-byte-wide format required by fixed point.

References

[1] M. G. Arnold, "Reduced Power Consumption for MPEG Decoding with LNS," *ASAP Conf.*, IEEE, San Jose, CA, pp. 65-75, Jul. 2002.

[2] C. W. Clenshaw and F. W. Olver, "Beyond Floating Point," *J. ACM*, vol. 31, pp. 319-328, 1984.

[3] J. N. Coleman and J. Kadlec, "Extended Precision Logarithmic Arithmetic," *34th Asilomar Conf. on Signals, Systems and Computers*, Oct. 2000.

[4] J. N. Mitchell, "Computer Multiplication and Division using Binary Logarithms," *IEEE Trans. Comput.*, vol. EC-11, pp. 512-517, Aug. 1962.

[5] E. E. Swartzlander and A. G. Alexopoulos, "The Sign/Logarithm Number System," *IEEE Trans. Comput.*, vol. C-24, pp. 1238–1242, Dec 1975.

[6] J. Watkinson, *The MPEG Handbook*, Focal Press, Oxford, 2001.