A Logarithmic Approach to Energy-Efficient GPU Arithmetic for Mobile Devices

Miguel Lastras and **Behrooz Parhami**
Department of Electrical and Computer Engineering
University of California
Santa Barbara, CA 93106-9560, USA
parhami@ece.ucsb.edu

Abstract

Graphic processing units (GPUs) have emerged as useful components in the realization of high-performance and cost-effective digital systems for numerically intensive applications, from simple personal devices to large-scale supercomputers. In this paper, we consider GPUs that incorporate energy-efficient logarithmic arithmetic units. After analyzing numerical errors arising from such an implementation scheme, we present a simple fine-tuning of the design to reduce the worst-case error.

Keywords—Arithmetic/logic unit, Error compensation, GPU-based computing, Logarithmic arithmetic unit, Low-power design.

Introduction and Background

Graphic processing units (GPUs) have evolved from simple coprocessors, helping CPUs in handling graphics, to powerful components that dwarf, in many cases, the performance of the most powerful CPUs. The cost-effectiveness of GPUs has drawn attention of many researchers that see them as an inexpensive option to perform the computations needed for physical simulation or to manipulate huge amounts of data. This trend is known as general-purpose computing on GPUs (GPGPU) and it is currently a very active area of research and development [1, 4, 11]. As examples of nongraphic applications, there are several libraries for linear algebra [7, 2], for solving differential equations [13], or even as a load balancer for irregular applications [8].

High-performance GPUs can be just as power-hungry as top-of-the-line CPUs, but the use of GPUs in personal and mobile devices has led to the development of energy-efficient variants. Because GPUs are used for applications that are numerically intensive [14], many more transistors are devoted to arithmetic/logic units (ALUs) than in more traditional CPUs. Therefore, efficient implementation of arithmetic circuits is vital to ensure energy efficiency.

Examples of computations performed by GPUs in the graphics domain include geometric transformations, typically formulated as matrix-vector multiplications, arithmetic-heavy lighting/rendering [15], which entail vertex and pixel processing, and shading. For data on the relative arithmetic operation counts in rendering computations, the reader is referred to [16]. Very roughly speaking, multiplication and addition/subtraction are nearly equal in their frequencies, while division is about 1/6 and square-rooting 1/40 as frequent.

Logarithmic Arithmetic Unit

Because rendering is heavy on multiplication, division, and powering operations, which tend to be slower than addition/subtraction, the use of logarithmic arithmetic is a good match. With logarithmic number representation, multiplication and division are converted to additions and subtractions, respectively, but addition and subtraction become more difficult nonlinear operations. Logarithmic arithmetic also gives rise to the problems of initial conversion, from binary to logarithmic format, and final reconversion back to standard binary.

In logarithmic representation, a number $\pm x$ is represented by its sign Sx and the logarithm Lx of its magnitude x. Consistent with common practice, we assume base-2 logarithms, that is, (Sx, Lx) = $\pm \log_2 x$. Because the logarithm of a number in (0, 1) is negative, we often assume the use of a scale factor m to convert numbers in the range (0, 1) to (1, m), in order to avoid using a second sign. Arithmetic operations on logarithmic numbers are shown in Table 1.

Table 1. Arithmetic with Lx = $\log_2 x$ and Ly = $\log_2 y$.

Operation	Binary	Logarithmic
Multiplication	$x \times y$	Lx + Ly
Division	$x \div y$	Lx – Ly
Reciprocation	$1 \div x$	–Lx
Square-rooting	$\sqrt{x}$	Lx >> 1 (right shift)
Squaring	x^2	Lx << 1 (left shift)
Exponentiation	x^y	$y \times$ Lx
Addition	$x + y$	Lx + $\log_2(1 + 2^{-(Lx-Ly)})$
Subtraction	$x - y$	Lx + $\log_2(1 - 2^{-(Lx-Ly)})$

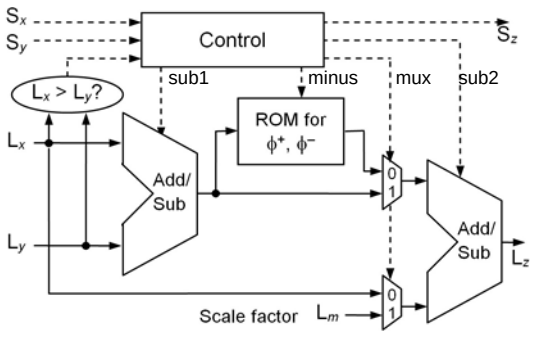

Fig. 1. A complete four-function ALU for LNS [12].

The schematic of a logarithmic ALU is shown in Fig. 1, with the 4 bits controlling the ROM, the 2-to-1 muxes, and the two adder/subtractors defining which of the 4 operations of addition, subtraction, multiplication, or division is performed, (Table 2). Bear in mind that besides the components depicted in Fig. 1, we also need blocks for logarithm and antilogarithm computations.

Kim et al. [6] implemented an energy-efficient 32-bit fixed-point logarithmic arithmetic unit suitable for GPUs in mobile devices. The 32-bit word consists of 6 whole bits and 26 fractional bits. This format allows the representation of base-2 logarithms in the range (–32, 32), making the number representation range comparable to 32-bit binary format. The main contributions by Kim et al. actually reside in the converters to/from logarithmic numbers. To this end, they designed fairly accurate and energy-efficient logarithmic and antilogarithmic converters, which we will review in the next two sections of this paper, along with an error reduction method in the antilogarithmic converter.

Table 2. Control signals defining the ALU operations.

Operation	minus	mux	sub1	sub2
Addition	0	0	1	0
Subtraction	1	0	1	0
Multiplication	x	1	0	1
Division	x	1	1	0

Logarithmic Converter

The logarithmic converter of Kim et al. [6] uses 8-way partitioning of the interval [0, 1] for piecewise linear approximation to improve upon the very simple linear method of Mitchell [9] and a somewhat more accurate, but still crude, method of Juang et al. [5], introducing the relative errors of 5.9% and 2.9%, respectively. The 8 intervals are associated with the linear approximations

$$\log_2(1 + x) = a_i x + b_i \qquad i = 0, 1, 2, \ldots, 7$$

Optimal values for a_i and b_i are determined and values close to optimal are chosen for a and b to reduce the hardware and power requirements. Shift-and-add is used for computing $a_i x$. Table 3 shows the implementation parameters. The hardware needed for the entire computation consists of a 32-bit leading 0s counter, a variable shifter, an integer unit producing the 6-bit whole part of the result, and a fractional unit yielding the remaining 26 bits of the logarithm. The latter unit, accounting for most of the latency, uses three 26-bit carry-save adders, arranged in 3 levels (the inputs to which come from look-up tables and hardwired shifters), feeding a 26-bit carry-propagate adder.

As evident from Fig. 2, errors for this converter range from –0.190%, occurring in the first subinterval [0, 1/8), to +0.103%, occurring at 3/8.

Table 3. Parameters for a logarithmic converter.

[From	To)	a_{opt}	b_{opt}	128a	1024b
0	1/8	1.35877	0.00170531	175	0
1/8	1/4	1.21558	0.0193393	158	15
1/4	3/8	1.09970	0.0481159	142	46
3/8	1/2	1.00399	0.0838589	127	91
1/2	5/8	0.923621	0.123934	119	123
5/8	3/4	0.855165	0.166631	110	167
3/4	7/8	0.796159	0.210815	102	215
7/8	1	0.744772	0.255722	95	264

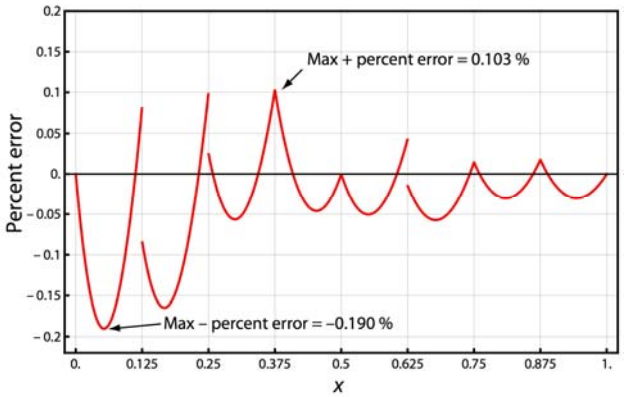

Fig. 2. Error plot for the logarithmic converter [6].

Antilogarithmic Converter

The antilogarithmic converter of Kim et al. [6] computes the function 2^f. Like their logarithmic converter, this unit also uses 8-way partitioning of the interval [0, 1] for piecewise linear approximation

$$2^x = c_i x + d_i \qquad i = 0, 1, 2, \ldots, 7$$

Optimal values for c_i and d_i are determined and values close to optimal are chosen for c and d to reduce the hardware requirements. Shift-and-add operations are used for computing $c_i x$. Table 4 shows the implementation parameters. As evident from Fig. 3, errors for this converter range from –0.070%, occurring at 3/8, to +0.082%, occurring in the sixth subinterval [5/8, 3/4).

A careful inspection of Fig. 3 shows that the percentage error can be reduced via a minor modification in the d coefficients in the look-up table. Modification of the c coefficients is more delicate, since the hardware implementation is highly tuned to the values chosen. The fifth and sixth subintervals are the ones dictating the maximum positive error 0.082%. The idea is to decrease the associated value of d in these regions, until they no longer dictate the worst case. Reducing the error in these subinterval to 0.070% or less, leads to an overall symmetric error range [–0.070%, +0.070%], a 15% improvement in the maximum error.

Table 4. Parameters for an antilogarithmic converter.

[From	To)	c_{opt}	d_{opt}	128c	1024d
0	1/8	0.723972	0.999353	92	1024
1/8	1/4	0.789497	0.991115	101	1015
1/4	3/8	0.860952	0.973199	111	995
3/8	1/2	0.938875	0.943922	121	964
1/2	5/8	1.02385	0.901373	131	924
5/8	3/4	1.11652	0.843390	143	864
3/4	7/8	1.21757	0.767527	155	792
7/8	1	1.32777	0.671023	169	695

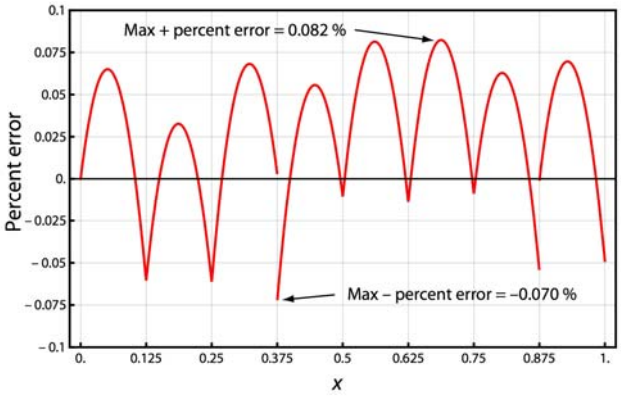

Fig. 3. Error plot for the antilogarithmic converter [6].

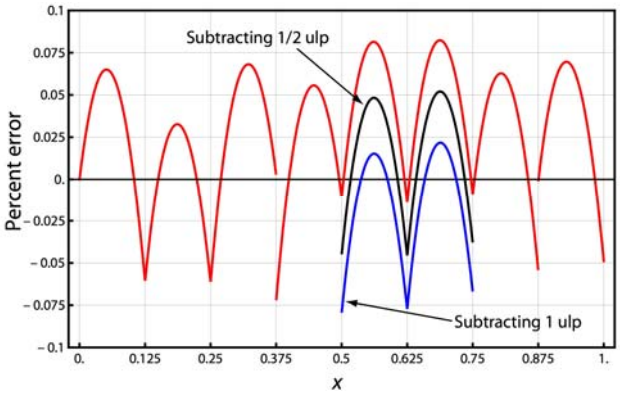

Fig. 4. Improved error plot for the antilogarithmic converter.

Subtracting 1 *ulp* (1/1024 = 2^{-10}) from the fifth and sixth d values in Table 4 would shift the curves downward by too large an amount, so that they then dictate the maximum negative error. Despite this fact, there would still be a slight improvement in the maximum error range; that is, the difference between the maximum positive error and maximum negative error would improve from 0.152 to 0.149. On the other hand, by adding one bit to the width of the lookup table for d, which is a very minor change not affecting hardware complexity and having a negligible effect on power consumption, we can reduce the fifth and sixth d coefficients by 1/2 *ulp*, leading to the aforementioned 15% improvement in error.

The reader may wonder why we did not apply the latter error reduction method to the logarithmic converter of Fig. 2. The reason has to do with the maximum negative error occurring in subinterval 1, which includes $x = 0$. Adding 1 *ulp* or 1/2 *ulp* would indeed improve the worst-case negative error, but it would also introduce an error of 1/1024 or 1/2048 in computing the logarithm of numbers very close to 1, producing very large relative errors that would be quite unacceptable in practice.

Conclusion

We have discussed the power consumption problem in high-performance graphics processors and how it would limit their use in low-power mobile devices. We reviewed an approach to power reduction that takes advantage of a logarithmic arithmetic unit, showing how a previously proposed design can be improved in terms of its error characteristics at negligible cost.

We are currently looking into other possible methods of improving both the accuracy and energy efficiency of such arithmetic units.

References

[1] D. Blythe, "Rise of the Graphics Processor," *Proc. IEEE*, Vol. 96, No. 5, pp. 761-778, May 2008.

[2] K. Fatahalian, J. Sugerman, and P. Hanrahan, "Understanding the Efficiency of GPU Algorithms for Matrix-Matrix Multiplication," *Proc. ACM SIGGRAPH/ EUROGRAPHICS Conf. Graphics Hardware*, pp. 133-137, 2004.

[3] I. Foster, *Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering*, Addison-Wesley, 1995.

[4] D. Geer, "Taking the Graphics Processor Beyond Graphics," *IEEE Computer*, Vol. 38, No. 9, pp.14-16, September 2005.

[5] T. B. Juang, S. H. Chen, and H. J. Cheng, "A Lower Error and ROM-Free Logarithmic Converter for Digital Signal Processing Applications," *IEEE Trans. Circuits and Systems II*, Vol. 56, No. 12, pp. 931-935, December 2009.

[6] H. Kim, B. G. Nam, J. H. Sohn, J. H. Woo, and H. J. Yoo, "A 231-MHz, 2.18-mW 32-bit Logarithmic Arithmetic Unit for Fixed-Point 3-D Graphics System," *IEEE J. Solid-State Circuits*, Vol. 41, No. 11, pp. 2373-2381, November 2006.

[7] J. Kruger and R. Westermann, "Linear Algebra Operators for GPU Implementation of Numerical Algorithms," *ACM Trans. Graphics*, Vol. 22, No. 3, pp. 908-916, 2005.

[8] M. A. Lastras-Montano, M. M. Michael, and J. A. Bivens, "Dynamic Work Scheduling for GPU Systems," in *GPUs and Scientific Applications* at Parallel Architectures and Compilation Techniques, 2010.

[9] J.N. Mitchell, "Computer Multiplication and Division Using Binary Logarithms," *IRE Trans. Electronic Computers*, Vol. 11, No. 4, pp. 512-517, 1962.

[10] NVIDIA, *Cuda Programming Guide 2.3*, NVIDIA Corp., 2009.

[11] J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Kruger, A. E. Lefohn, and T. J. Purcell, "A Survey of General-Purpose Computation on Graphics Hardware," *Computer Graphics Forum*, Vol. 26, pp. 80-113, 2007.

[12] B. Parhami, *Computer Arithmetic: Algorithms and Hardware Designs*, Oxford, 2nd ed., 2010.

[13] J. C. Thibault and I. Senocak, "CUDA Implementation of a Navier-Stokes Solver on Multi-GPU Desktop Platforms for Incompressible Flows," *Proc. 47th AIAA Aerospace Sciences Meeting*, pp. 1-15, 2009.

[14] W. Vogels, "Expanding the Cloud—Adding the Incredible Power of the Amazon EC2 Cluster GPU Instances," 2011 (document accessed on July 18, 2013). http://www.allthingsdistributed.com/2010/11/cluster_gpu_instances_amazon_ec2.html

[15] J. H. Woo, J. H. Sohn, H. Kim, and H .J. Yoo, "A 195 mW, 9.1 MVertices/s Fully Programmable 3-D Graphics Processor for Low-Power Mobile Devices," *IEEE J. Solid-State Circuits*, Vol. 43, pp. 2370-2380, 2008.

[16] K. Yoshida, T. Sakamoto, and T. Hase, "A 3D Graphics Library for 32-bit Microprocessors for Embedded Systems, *IEEE Trans. Consumer Electronics*, Vol. 44, No. 3, pp. 1107-1114, 1998.

Appendix: Sources of Energy Savings

In Section 2, we referred to the energy-efficient 32-bit logarithmic arithmetic unit implemented by Kim et al. [6]. In this appendix, we provide a brief overview of what causes logarithmic representation to have lower energy requirements compared with standard 2's-complement binary arithmetic. Essentially, we want to provide a basis and an intuitive feel for why logarithmic arithmetic fares better in terms of energy/power consumption.

The energy efficiency advantage of logarithmic number representation was known for quite some time through episodic evidence, as various designers demonstrated low-power circuits for particular applications running on associated fine-tuned hardware platforms. Subsequently, V, Paliouras and T. Stouraitis (*Proc. 15th IEEE Symp. Computer Arithmetic*, 2001, pp. 229-236) published a study of signal transitions in logarithmic arithmetic versus fixed-point arithmetic, showing significant reduction in activity with linear (uniform and correlated Gaussian) input data. Power savings from reduced activity are augmented by reduction in circuit complexity due to the fact that for LNS, multipliers, dividers, squarers, and square-rooters are replaced by the much simpler add, subtract, left-shift, and right-shift circuitry.

For many practical, compute-intensive applications, the savings from reduced activity and circuit simplifications are substantial enough to compensate for the energy requirements of conversion and reconversion processes, needed to interface the system with non-logarithmic data sources and sinks, and the somewhat greater complexity of addition and subtraction. The accrued benefits are, of course, dependent on implementation parameters such as word width, precision (location of the radix point in the logarithm field), and logarithm base.

In recent years, a variety of new applications and associated circuit realizations have confirmed the energy benefits of logarithmic arithmetic, as suggested by the preceding discussion.