Comparing Floating-point and Logarithmic Number Representations for Reconfigurable Acceleration

Haohuan Fu, Oskar Mencer, Wayne Luk

Department of Computing, Imperial College
London, United Kingdom
{hfu,oskar,wl}@doc.ic.ac.uk

***Abstract*— We investigate floating-point and logarithmic number representations for computing with FPGAs. The key issue is to select the best number format for an application to improve performance and accuracy. Using A Stream Compiler, ASC as the hardware design and compilation tool, we develop a convenient scheme to compare the designs of both floating-point and logarithmic numbers and select the solution with the best performance and accuracy. Our contributions are: (1) optimized function evaluations for conversions between logarithmic and floating-point numbers; (2) design and implementation of logarithmic arithmetic, with optimized segmentation and polynomial degree; (3) a practical comparison case study of Monte Carlo radiative heat transfer simulation. Compared to prior work, our design supports two to six times more LNS conversion and LNS arithmetic units on one FPGA. For Monte Carlo simulation, our designs of both number systems produce 39-80% higher throughput with either a smaller area or a higher accuracy.**

I. Introduction

With the development of electronic technologies, current FPGA devices have millions of look-up tables (LUTs), registers, dedicated hardware multipliers and even microprocessors. The improvement of capabilities makes the resource-consuming floating-point and logarithmic number systems (LNS) also applicable to FPGAs.

Floating-point and LNS numbers have a similar representation range, but their arithmetic behaviors differ greatly. An application implemented with one number format has quite different accuracy and performance compared with the same one implemented with the other format [1], [2]. To improve performance and accuracy of an application, we develop a scheme to compare designs of both number formats. As shown in Figure 1, using A Stream Compiler, ASC [3] as the hardware design and compilation tool, we generate designs of the application in both number formats and make comparisons to find out the solution with best accuracy and performance.

Targeted as an approach for comparison between floating-point and LNS representations, our scheme includes basic arithmetic support for both number formats and provides an interface to build up an application with floating-point or LNS numbers. Our contributions are:

1) For conversions between floating-point and LNS numbers, we develop optimized evaluation for functions $y = \log_2(x)$ and $y = 2^x$.

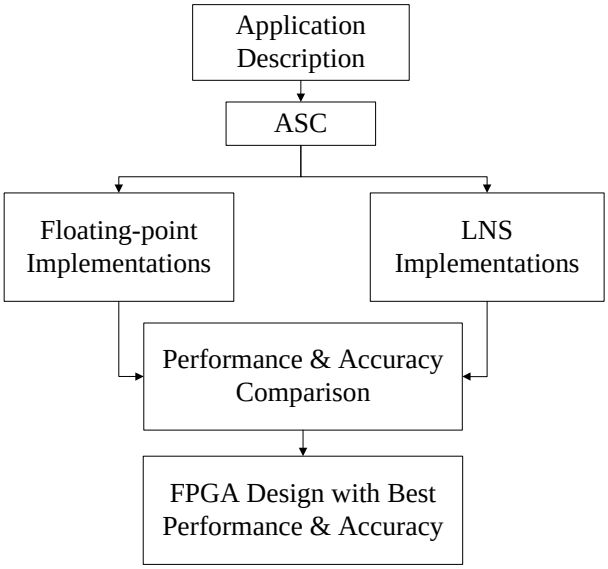

Fig. 1. General procedure of the comparison scheme.

2) In order to provide efficient LNS arithmetic support in the comparison scheme, we design and implement optimized evaluation methods using polynomial approximation for LNS addition and subtraction.
3) To demonstrate the feasibility of our scheme, we perform a practical case study of Monte Carlo radiative heat transfer simulation. The results show that our scheme can generate high-performance implementations with both number representations.

II. Background

Given the same bit-widths, floating-point and LNS numbers provide similar representation ranges. However, the arithmetic implementations of the two are quite different. For floating point numbers, addition and subtraction are simple; multiplication and division are difficult, while exponential operations are even more difficult. In contrast, multiplication, division and exponential operations are easy for LNS, but addition and subtraction are extraordinarily difficult.

Suppose a and b are the logarithmic forms of numbers A and B ($A > B$), thus $A = 2^a$ and $B = 2^b$. The basic LNS arithmetic operations of these two numbers are as follows:

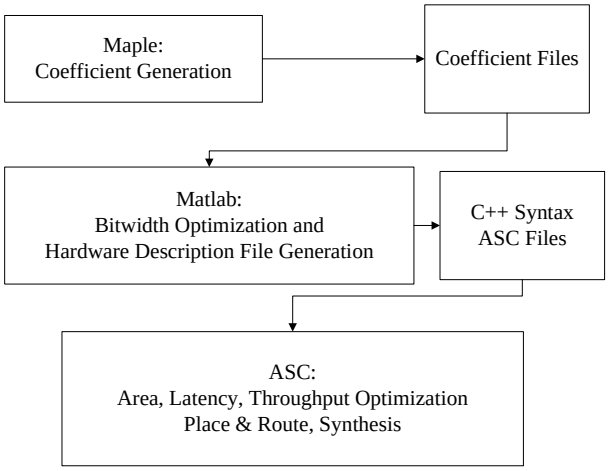

Fig. 2. Work flow: generating the polynomial approximation design.

MUL: $A \times B = 2^a \times 2^b = 2^{a+b}$.
DIV: $A \div B = 2^a \div 2^b = 2^{a-b}$.
ADD: $A + B = 2^a + 2^b = 2^{a+\log_2(1+2^{b-a})}$.
SUB: $A - B = 2^a - 2^b = 2^{b+\log_2(2^{a-b}-1)}$.

Thus, we can perform LNS MUL/DIV with fixed-point addition or subtraction, but for LNS ADD/SUB we need to evaluate $f_1(x) = \log_2(1 + 2^x)$ and $f_2(x) = \log_2(2^x - 1)$, which are non-linear and difficult to evaluate.

There are existing works on comparisons between different number representations for FPGA. Based on the arithmetic of the LNS microprocessor [4], Matousek et al. [2] present a comparison between LNS and floating-point numbers on FPGA. However, the benchmark application has only one addition and many DIV/MUL and exponential operations, which clearly favors the LNS.

J. Detrey et al. [5] provide a tool for comparison between logarithmic and floating-point FPGA designs. The tool includes libraries of parameterized arithmetic operations for the two number systems. They also show comparison examples such as a 3D transformation pipeline. However, as mentioned by the authors, the example is not a practical application on FPGA.

The most recent work is a comparison between floating-point and LNS for FPGAs [6]. It presents and compares all the basic operators for two different number representations in both 32-bit and 64-bit bit-widths. However, there is still no comparison or analysis of an FPGA application.

III. LNS Design and Implementation

The major part of the implementation for LNS is to evaluate four non-linear functions. For conversions between floating-point and LNS numbers, the logarithmic function $y = \log_2(x)$ and the exponential function $y = 2^x$ need to be evaluated. Besides, as noted in section II, for LNS ADD/SUB, f_1 and f_2 also need to be evaluated.

In our scheme, we evaluate all four functions using polynomial approximation with Maple, Matlab and ASC. We currently support a 32-bit LNS format with 8 magnitude bits and 23 fractional bits, which has a range and precision similar to single-precision floating-point.

TABLE I
Performance results of our 32-bit LNS conversion units, compared with the existing design [6]. THR refers to our design optimized for throughput, which is fully pipelined; LAT refer to our design optimized for latency. HMULs are hardware multipliers and BRAMs are block rams in the FPGA. Number of units per FPGA is calculated for the Virtex-4 XC4VLX200.

Functions	$y = \log_2(x)$			$y = 2^x$		
Designs	[6]	THR	LAT	[6]	THR	LAT
# units/FPGA	**14**	**32**	**32**	**4.8**	**32**	**32**
# slices	163	648	398	236	706	442
# 18x18 HMUL	0	3	3	20	3	3
# 18K BRAM	24	7	7	4	7	7
clk cycle(ns)	76	8.2	61.3	72	9.4	49.0
latency(cycles)	1	13	1	1	13	1
NOPS(Mhz)	13.2	122	16.3	13.9	106	20.4

A. Polynomial Approximation

Our general work flow for generating the polynomial approximation design of a function is shown in Figure 2.

We use Maple to generate the coefficients for polynomial approximations by the Minimax algorithm. With the coefficients, Matlab then generates the design for the polynomial approximation, and optimizes the bit-width of variables with an approach based on the adaptive simulated annealing (ASA) method [7]. The design described in ASC C++ syntax is then forwarded to the ASC compiler to be converted into circuit design of the function unit.

For each function, we enumerate different polynomial degree values, experiment with both uniform and non-uniform segmentation methods to find out the optimal choices. In our scheme, we use a simplified version of the hierarchical segmentation approach [8] as the non-uniform segmentation method.

B. LNS Conversion Functions

For functions $y = \log_2(x)$ and $y = 2^x$, the approximation procedure is divided into three steps:

1) Range Reduction: reduce x into a smaller interval.
2) Function Evaluation: approximate the value of the function in the reduced interval with polynomials.
3) Range Reconstructions: map the result value in the reduced interval back into the full range of x.

Based on the functions' mathematical properties and representation formats, we reduce the input value to the interval of [1,2) for $y = \log_2(x)$ and [0,1) for $y = 2^x$. Based on extensive experiments for different polynomial degree values and segmentation methods, we evaluate the conversion functions with the following design procedures: with an approximation error requirement of 2^{-23} for 32-bit LNS numbers, $y = \log_2(x)$ is calculated by a degree-two polynomial with 128 uniform

TABLE II

PERFORMANCE RESULTS OF BASIC 32-BIT LNS ARITHMETIC UNITS. 'ADD0' AND 'SUB0' REFER TO THE EXISTING DESIGN [6]. 'ADD_T' AND 'ADD_L' REFER TO OUR ADDITION DESIGNS OPTIMIZED FOR THROUGHPUT AND LATENCY RESPECTIVELY, WITH A POLYNOMIAL DEGREE OF TWO. 'SUB2', 'SUB3' AND 'SUB4' REFER TO OUR SUBTRACTION DESIGNS WITH DIFFERENT POLYNOMIAL DEGREE OF TWO, THREE AND FOUR. HMULS ARE HARDWARE MULTIPLIERS AND BRAMS ARE BLOCK RAMS IN THE FPGA. NUMBER OF UNITS PER FPGA IS CALCULATED FOR THE VIRTEX-4 XC4VLX200.

function	addition			subtraction						
unit	add0 [6]	add_t	add_l	sub0 [6]	sub2_t	sub2_l	sub3_t	sub3_l	sub4_t	sub4_l
# units/FPGA	**4**	**16**	**16**	**4**	**12.4**	**12.4**	**12**	**12**	**8**	**8**
# slices	750	1775	932	838	2170	1168	2412	1259	2773	1427
# 18x18 HMUL	24	6	6	24	6	6	8	8	12	12
# 18K BRAM	4	7	7	8	27	27	14	14	12	12
clk cycle(ns)	91	7.9	96	95	16.0	112.0	12.5	141.1	16.6	168.3
latency(cycles)	1	25	1	1	27	1	32	1	37	1
NOPS(Mhz)	11.0	127	10.4	10.5	62.5	8.93	80.0	7.09	60.2	5.94

segments; the function $y = 2^x$ is calculated by a degree-two polynomial with 64 uniform segments.

Table I shows the performance results of our conversion units, compared with a previous design [6]. As our hardware compilation tool ASC supports different optimization options, circuit designs optimized for throughput and latency are both implemented. Our throughput-optimized designs, which are fully pipelined, consume three to four times more slices than the previous one [6], but they have a much smaller clock cycle and provide eight to nine times higher throughput. Our latency designs also show a 20-30% smaller latency than the previous one [6]. Meanwhile, the previous designs [6] consume 20 multipliers or 24 Block RAMs (BRAMs) for each unit, which greatly constrain the maximum number of units that can be supported in one FPGA. For the Virtex-4 XC4VLX200 FPGA, our designs support two to six times more units.

C. LNS Addition & Subtraction

As noted in section II, the ADD/SUB design is the most difficult in providing building blocks for LNS arithmetic. Without the mathematical properties like $y = \log_2(x)$ and $y = 2^x$, the range of f_1 and f_2 cannot be reduced and the whole interval of [0, 256) needs to be approximated.

Based on extensive experiments for different polynomial degree values and segmentation methods, we implement LNS ADD/SUB with the following design procedures: both functions use a non-uniform segmentation to reduce the number of segments to an acceptable range; degree of two is an optimal value for LNS ADD while degree of two, three and four are all good candidates for LNS SUB, as some of them consume fewer multipliers and others consume fewer BRAMs.

Table II shows the performance results for our ADD/SUB units, compared with a previous design [6]. Our designs optimized for latency consume 20-30% more slices, more BRAMs, but much less hardware multipliers than the previous one. And the latency values are similar. Our designs optimized for throughput are fully pipelined. They consume 2.3 to 3.3 times more slices, but provide a much smaller clock cycle with six to eleven times higher throughput. Meanwhile, same as the conversion units, the ADD/SUB units in [6] consume 24 multipliers for each unit, which greatly constrain the maximum number of units supported in one FPGA. With the Virtex-4 XC4VLX200 FPGA, our design supports two to four times more ADD/SUB units.

Generally, although consuming more resources, our design offers higher throughput, more units on a board, and a more extensive support with different settings.

IV. CASE STUDY: MONTE CARLO SIMULATION

A. Introduction to the Application

M. Gokhale et al. [9] presents an acceleration of Monte Carlo radiative heat transfer simulation on FPGA, with floating-point numbers. The simulation traces photons emitted from the surfaces of a 2-D enclosure. It records how many photons emitted from surface i are absorbed at surface j and then uses the recorded numbers to compute a heat transfer coefficient. The whole application is made up of several different levels of loops. The most inner loop, which is also the most computationally intensive part, is extracted and implemented on FPGA. The loop checks the photons for intersection of the surfaces. It performs 12 multiplications, 1 division, 3 additions and 7 subtractions in a 11-stage pipeline.

In this paper, the same application is investigated as a case study of our comparison scheme. We implement it with both single-precision floating-point and 32-bit LNS to compare the performance and accuracy.

DIV1 and DIV2 are our floating-point designs with two different division implementations: DIV1 uses general division and DIV2 uses polynomial approximation for the division, which is a reciprocal operation. P2, P3 and P4 are our LNS designs, which respectively use the degree-two, degree-three and degree-four designs of the LNS SUB operation.

B. Comparison of Performance and Accuracy

Table III shows the performance and accuracy of our Monte Carlo hardware designs, compared with the original work [9]. All the designs are mapped into the Virtex-II XC2V6000 FPGA for performance testing.

For our floating-point designs, DIV1 has a similar clock cycle to the original design [9], but it uses fewer LUTs and much fewer hardware multipliers. DIV2 has the smallest clock cycle of all the different implementations, which results

TABLE III

COMPARISON OF PERFORMANCE AND ACCURACY FOR DIFFERENT MONTE CARLO DESIGNS. DIV1 AND DIV2 REFER TO OUR FLOATING-POINT DESIGNS WITH DIFFERENT DIVISION IMPLEMENTATIONS. P2, P3 AND P4 REFER TO OUR LNS DESIGNS WITH DIFFERENT LNS SUB IMPLEMENTATIONS. HMULS ARE HARDWARE MULTIPLIERS AND BRAMS ARE BLOCK RAMS IN THE FPGA. ALL THE DESIGNS ARE MAPPED INTO THE VIRTEX-II XC2V6000.

Version	clk cycle (ns)	latency (cycles)	LUTs	18x18 HMUL	18K BRAM	absolute error		relative error	
						max($\times 10^{5}$)	mean	max($\times 10^{-4}$)	mean($\times 10^{-7}$)
[9]	29.9	41	20%	100%	8%	n/a	n/a	n/a	n/a
DIV1	29.6	114	18%	33%	8%	2.487	9.611	9.104	4.131
DIV2	16.4	74	12%	35%	10%	2.488	9.612	10.99	4.414
P2	21.3	110	31%	54%	84%	2.177	9.067	7.565	3.870
P3	26.3	120	35%	63%	45%	2.177	9.067	7.565	3.870
P4	n/a	130	36%	83%	38%	2.177	9.067	7.565	3.873

in a throughput 80% higher than the original design. DIV2 also uses the least LUTs. DIV2's BRAM usage is a little higher than DIV1 and the original design, due to the cost of polynomial approximation for reciprocal. The original design [9] uses all the multipliers on the board, and only supports one pipeline on a FPGA, while DIV1 and DIV2 consume 33-35% multipliers, and support at least two pipelines and double the throughput.

For LNS designs, P4 does not have a clock cycle result, because the design contains 34 BRAMs and 120 multipliers. The multiplier site adjacent to the location of a BRAM must remain free because of resource sharing. Thus it cannot be mapped into Virtex-II XC2V6000 which only has 144 multipliers. P2 and P3 have a smaller clock cycle than the original version, which indicates 14-40% higher throughput. With the polynomial degree increasing from P2 to P4, the number of clock cycle, latency and usage of multipliers increases, but the usage of BRAM decreases. Thus, the three different versions can be used to deal with different situations.

As shown in Table III, the original work [9] gives no results about the errors of calculated values. For our implementations, DIV2 has slightly larger errors than DIV1, and the three LNS versions, P2, P3, and P4, have almost the same error values. For the comparison between floating-point and LNS implementations, LNS has better accuracy. Its mean absolute and relative errors are 6% smaller than the floating-point version, while the maximum errors are 13-17% smaller.

Therefore, for this Monte Carlo application, both the floating-point and LNS implementations generated by our scheme are efficient and have a better throughput than the original work [9]. Compared with each other, the floating-point version DIV2 has a smaller clock cycle than LNS versions, resulting in a higher throughput. The LNS versions generally consume more hardware resource than the floating-point versions, but they provide better accuracy.

V. CONCLUSIONS

In summary, we develop a semi-automated scheme to compare different hardware designs using floating-point or LNS representations, which enables us to select a solution with the best performance and accuracy. Using Maple, Matlab and ASC, our work provides an efficient design and implementation of LNS conversions and arithmetic.

Compared to prior work [6], we support two to six times more LNS conversion and LNS ADD/SUB units in one FPGA. To demonstrate the feasibility of our scheme, we test the implementation with a practical FPGA application, Monte Carlo radiative heat transfer simulation. Our scheme generates high-performance implementation with both floating-point and LNS numbers, and provides a comparison to improve performance and accuracy.

The plan for future work includes: (1) As our current LNS arithmetic units still consume a large amount of hardware resources, it is important to refine our design to improve the efficiency. (2) We will also use our scheme to make comparisons of other typical applications, such as encryption and multimedia processing procedures, to further investigate the optimization of number representations.

ACKNOWLEDGMENT

This work is supported by EPSRC grant no. EP/C509625/1.

REFERENCES

[1] E. Swartzlander, D. Chandra, H. Nagle, and S. Starks, "Sign/Logarithm Arithmetic for FFT Implementation," *IEEE Trans. Comput.*, vol. 32, no. 6, pp. 526–534, 1983.

[2] R. Matousek, M. Tichy, Z. Pohl, J. Kadlec, C. Softley, and N. Coleman, "Logarithmic Number System and Floating-point Arithmetic on FPGA," in *Proc. FPL*, 2002, pp. 627–636.

[3] O. Mencer, "ASC, A Stream Compiler for Computing with FPGAs," *IEEE Trans. Computer-Aided Design*, 2006.

[4] J. Coleman, E. Chester, C. Softley, and J. Kadlec, "Arithmetic on the European Logarithmic Microprocessor," *IEEE Trans. Comput.*, vol. 49, no. 7, pp. 702–715, July 2000.

[5] J. Detrey and F. Dinechin, "A Tool for Unbiased Comparison between Logarithmic and Floating-point Airthmetic." http://www.ens-lyon.fr/LIP/Pub/Rapports/RR/RR2004/RR2004-31.ps.gz.

[6] M. Haselman, M. Beauchamp, K. Underwood, and K. Hemmert, "A Comparison of Floating Point and Logarithmic Number Systems for FPGAs," in *Proc. FCCM*, 2005, pp. 181–190.

[7] D. Lee, A. A. Gaffar, O. Mencer, and W. Luk, "Optimizing hardware function evaluation," *IEEE Trans. Comput.*, vol. 54, no. 12, pp. 1520–1531, Dec. 2005.

[8] D. Lee, W. Luk, J. Villasenor, and P. Cheung, "Hierarchical segmentation schemes for function evaluation," in *Proc. FPT*, 2003, pp. 92–99.

[9] M. Gokhale and et al., "Monte Carlo Radiative Heat Transfer Simulation on a Reconfigurable Computer," in *Proc. FPL, LNCS 3203*, 2004, pp. 95–104.