PAPER

FPGA-Specific Custom VLIW Architecture for Arbitrary Precision Floating-Point Arithmetic

Yuanwu LEI[†a)], ***Student Member***, **Yong DOU**[†], ***and*** **Jie ZHOU**[†], ***Nonmembers***

SUMMARY Many scientific applications require efficient variable-precision floating-point arithmetic. This paper presents a special-purpose Very Large Instruction Word (VLIW) architecture for variable precision floating-point arithmetic (VV-Processor) on FPGA. The proposed processor uses a unified hardware structure, equipped with multiple custom variable-precision arithmetic units, to implement various variable-precision algebraic and transcendental functions. The performance is improved through the explicitly parallel technology of VLIW instruction and by dynamically varying the precision of intermediate computation. We take division and exponential function as examples to illustrate the design of variable-precision elementary algorithms in VV-Processor. Finally, we create a prototype of VV-Processor unit on a Xilinx XC6VLX760-2FF1760 FPGA chip. The experimental results show that one VV-Processor unit, running at 253 MHz, outperforms the approach of a software-based library running on an Intel Core i3 530 CPU at 2.93 GHz by a factor of 5X-37X for basic variable-precision arithmetic operations and elementary functions.

key words: *variable-precision floating-point (VP) arithmetic, Very Long Instruction Word (VLIW), elementary function, Newton's method, polynomial approximation, FPGA*

1. Introduction

Many scientific and engineering applications require efficient variable-precision floating-point (VP) arithmetic [1]. These applications range from mathematical computations to large-scale physical simulations, such as computational geometry, climate modeling and supernova simulation. It is extremely important to provide accurate results for the numerical sensitive calculations in these applications.

However, almost all recent high performance general-purpose processors do not provide hardware units for VP arithmetic operations. Most are accomplished using software approaches, such as GNU Multiple-Precision library (GMP) [2], Multiple Precision Floating-Point Reliable library (MPFR) [3], NTL [4], and so on. The main disadvantage of software approaches is their speed. Compared with 64-bit floating-point arithmetic, software approaches are at least one order of magnitude slower for quadruple precision arithmetic and 40X slower for octuple precision arithmetic [5]. For the higher precision arithmetic, the computational cost increases roughly quadratically with the precision.

Many hardware designs attempted to overcome the speed limit of software solutions, such as CADAC [6], DRAFT [7], CASCADE [8], and VPIAP [9]. E. El-Araby proposed the use of High Performance Reconfigurable Computers (HPRCs) as a promising candidate for arbitrary precision arithmetic [10]. F.T. Alexandre [11] presented the approach Host+Rconfigurable Arithmetic Coprocessor (HRAC) to implement variable long-precision computation. The above processors were designed to perform basic VP arithmetic operations.

Some studies focused on the hardware structure for variable-precision elementary functions. J. Hormigo [12] and E. Saez [13] proposed a CORDIC processor for VP elementary functions, which are evaluated by simple fixed-point adding and shifting. However, CORDIC algorithm cannot guarantee low relative error [14]. It needs higher computation bandwidth, more storage to store the elementary rotation angles, and more iterations to obtain a desired result, when the result is close to zero. At the same time, the hardware algorithms for VP division, square root, logarithm, and triangle function are proposed separately, according to the properties of these functions [15]–[17].

Field-programmable gate array (FPGA) chips, which operate at the bit level and serve as custom hardware for the different computation precisions, could potentially implement high precision scientific computing applications and provide significantly higher performance than a general-purpose CPU [10], [18], [19]. The computational capability of FPGAs is increasing rapidly. A top-level FPGA chip from Xilinx Virtex- 6 series contains 474,240 Slice LUTs, 25,920 Kbits storage and 864 DSP48E1 Slices (25×18 MAC). Many DSP48E1 and logic resources help to build more custom arithmetic units for VP arithmetic.

However, several problems still exist in using FPGA to accelerate VP scientific applications. First, there are a variety of elementary functions in scientific applications. The computation complexity of these functions usually are $O(n^{1/2}M(n))$ [20], where $M(n)$ refers to the complexity of multiplication operation and n refers to the precision of result. The consumption of hardware resources increases quadratically relative to the computation width with pipeline technology. Therefore, it is intractable to implement all of these elementary function units on the same chip, so additional communication overheads between chips are required. That will reduce performance and utilization of chips. Second, we need to redesign the function units and the entire system when they are no longer sufficiently accurate for some applications, due to the poor flexibility of cus-

Manuscript received April 25, 2011.
Manuscript revised June 30, 2011.

[†]The authors are with the National Laboratory for Parallel and Distribution Processing, National University of Defense Technology, Changsha, 410073, P. R. China.
a) E-mail: yuanwulei@nudt.edu.cn

DOI: 10.1587/transinf.E94.D.2173

tom function units. Moreover, the latency of VP elementary function is long because the higher internal working precision is required to obtain an accurate result.

In this paper, we present a special-purpose very large instruction word (VLIW) processor for variable-precision floating-point arithmetic (VV-Processor) equipped with multiple custom VP arithmetic units. This processor uses the unified hardware structure to implement various complex VP algebraic and transcendental functions. Performance is improved by using explicitly parallel technology of VLIW instruction and by dynamically varying the precision of the intermediate computation. Finally, we create a prototype of VV-Processor unit on a Xilinx Virtex-6 XC6VLX760-2FF1760 FPGA chip.

2. Background

2.1 Range Reduction

The first step for the computation of elementary function f at x is to reduce the argument range, called *range reduction*, to improve computational efficiency. It is usually divided into three steps.

- *Range reduction*: transform x into x' based on properties of f like additivity, symmetry, and periodicity.
- *Evaluation*: evaluate f at x'.
- *Reconstruction*: compute $f(x)$ from $f(x')$ using a functional identity, and normalize it to the specific format.

2.2 Methods for Elementary Function

The main methods used to evaluate high-precision elementary functions in hardware are digit-recurrence, Newton-Raphson algorithm, and polynomial approximation. Digit-recurrence methods, such as the SRT algorithm [21] and CORDIC algorithm [12], [13], [17], are linearly converge, which means fixed number bits of the result are obtained in each iteration. So the latency is very long to obtain a high-precision result. This is a serious shortcoming for high-precision computations.

Both Newton-Raphson algorithm and polynomial approximation are multiplicative approaches. Newton-Raphson algorithm [22] can be used to compute functional inverse functions, such as reciprocal, division, reciprocal square root, and square root. A table lookup, which stores the approximate value, is always followed to reduce the number of Newton-Raphson iteration. This method typically has quadratic convergence, resulting in low latency for high-precision computations. Thus, the Newton-Raphson method is a powerful tool for VP arithmetic.

Polynomial approximation is another alternative to approximate elementary functions (exponentials, logarithms, trigonometric functions, etc.) [22]. The degree of the polynomial to be used is proportional to the precision of the result. Therefore, many multiplications and additions must be performed for high-precision calculation. The Horner scheme is used to evaluate polynomials efficiently in monomial form. However, the addition and the multiplication in each iteration of Horner scheme must be executed serially.

Recently, computing-oriented FPGAs include many embedded multipliers and RAM blocks. Multiplicative approaches allow one to make the best use of these available resources. However, there are some compromises between the size of the lookup table and the number of Newton-Raphson iteration or the degree of the polynomial approximation. Choosing a good compromise may require to take into account the resources of the target FPGAs.

2.3 Custom VLIW Architecture

Figure 1 illustrates polynomial approximation algorithm and Newton-Raphson algorithm. The degree of the polynomial approximations (N_t) satisfies N_t=$prec_r$/(c*exp_i), where $prec_r$ denotes the precision of result, exp_i denotes negative exponent of x, and c is the speed of convergence, which is 1 for exponential and 2 for sine/cosine. Thus, with the aid of range reduction technology or lookup tables, the degree of polynomial is reduced quickly. The Newton-Raphson method is implemented using iteration method and the number of iteration (N_n) satisfies $N_n = \log_2(\frac{prec_r}{prec_i})$, where $prec_i$ and $prec_r$ denote the precision of initial value and result respectively. Both of polynomial approximation and Newton-Raphson method are combined with addition and multiplication operations and the elementary functions are constructed by composing basic arithmetic operations. However, the data dependences between these basic operations are varied between different elementary functions.

Very long instruction word (VLIW) [23]–[25] is an effective approach to achieve high levels of instruction level parallelism (ILP) by executing long instruction words composed of multiple heterogeneous operations. This is a type of multiple instruction multiple data (MIMD) processor. We propose a custom VLIW architecture for VP arithmetic, which has the following advantages:

- The unified hardware, which composed of multiple custom VP arithmetic units, is used to to evaluate a variety of VP elementary functions and the data dependence between basic operations is maintained under the control of VLIW instruction.

(A) Polynomial approximations: $f(x)=a_0+a_1x^1+\cdots+a_nx^n$.	(B) Division based on Newton's method: $z_{i+1}=z_i(1+\varepsilon)$, where $z_0\approx y^{-0.5}$, $\varepsilon=1-z_iz$. Then $z_n=y^{-0.5}$, $z=x*x_n=x/y$.
S1: Initialization. The *range reduction* is used to reduce the argument x and the degree of $f(x)$ (N_t) is fixed according to the precision of result. Set T=x, F=0;	**S1: Initialization.** Lookup table to obtain z_0 and fix the number of iterations (N_n) according to the precision of z_0 and result.
S2: Evaulation:	**S2: Evaulation:**
for(i=1..N_t)	**for**(i=0..N_n)
begin	**begin**
$S = a_i * T$; ▶Multiplication (1)	$T_1 = z_i * y$; ▶Multiplication (1)
$F = F + S$; ▶Addition (2)	$S_1 = 2 + T_1$; ▶Addition (2)
$T = T * y$; ▶Multiplication (3)	$z_{i+1} = z_i * S_1$; ▶Multiplication (3)
end	**end**
S3: Normalization.	*Result* = $x * x_n$; ▶Multiplication (4)
	S3: Normalization.

Fig. 1 Polynomial approximation and Newton-Raphson.

- The performance is improved through the explicitly parallel technology of VLIW structure and more available ILP can be exploited using the loop unrolling technology, as shown in Fig. 7 (B).
- This custom VLIW processor achieves high scalability of elementary functions. Under the control of specific VLIW instructions, the VP arithmetic units are combined to a special-purpose hardware for elementary function.

To the best of our knowledge, this paper is the first report advancing a special-purpose VLIW architecture for variable-precision floating-point arithmetic on FPGA, which can evaluate basic VP arithmetic operations and VP elementary functions.

3. Implementation of VV-Processor

This section first defines the format for VP numbers. Then, we present the organization of the special-purpose VLIW architecture on FPGA (VV-Processor), equipped with multiple custom VP arithmetic units. The performance of VV-Processor is improved by the explicitly parallel technology of the VLIW structure.

3.1 Variable-Precision Number Format

The format of variable-precision numbers is similar to the one presented in [26]. An VP number x consists of a 16-bit exponent field represented with a bias of 32,767 (BE_x, the real value of exponent is E_x=BE_x-32,767), a sign bit (S_x), a 7-bit mantissa length field (P_x), and a variable-precision mantissa field (M_x) that consists of P_x 64-bit words (M_x[0] to M_x[P_x-1], and M_x[0] is the highest word). The value of the normalized mantissa is between 0.5 and 1, i.e., $0.5 \le M_x < 1$. The precision of x is 64*P_x and the value of x is:

$$x = (-1)^{S_x} \cdot M_x \cdot 2^{BE_x - 32767} = (-1)^{S_x} \cdot M_x \cdot 2^{E_x}.$$

3.2 Hardware Organization

As shown in Fig. 2, VV-Processor is mainly composed of a parallel computation module, a control state machine module, a VLIW instruction RAM, and a decode module. The evaluation of all arithmetic functions are accomplished by the parallel computation module through explicitly parallel technology. This module consists of multiple custom VP arithmetic units, three on-chip memory modules, and control logic.

The basic arithmetic units include n_m VP multiplication (VP_Mult) units, n_a VP addition/subtraction (VP_Add/Sub) units, and a VP separator (VP_Sep) unit. The VP_Sep module performs mod one operation, which separates an VP number A into an integer I and a fraction F, satisfied A=I+F, and $|F| < 1$.

Figure 3 shows the execution cycle of different approximation algorithms illustrated in Fig. 1 with various number of VP_Mult units in VV-Processor. The polynomial approximations are implemented through the classical approach and Horner scheme. With the help of loop unrolling technology, Operation (2) in the i^{th} iteration, Operation (3) in the $(i+1)^{th}$ iteration, and Operation (1) in the $(i+2)^{th}$ iteration in Fig. 1 (A) can be executed in parallel. Thus, the execution cycle of the classical polynomial approach is smaller than that of Horner scheme when $n_m \ge 2$. The value of n_m=2 achieves the lowest latency for different algorithms. Therefore, we are equipped with two VP_Mult units and two VP_Add/Sub units in the parallel computation module to gain high performance.

Three on-chip memory modules are Lookup-Table, ROM-C, and RAM-M. The Lookup-Table, a 1024×9 ROM with a single port, stores the initial approximation for some functions based on the table lookup method. For example,

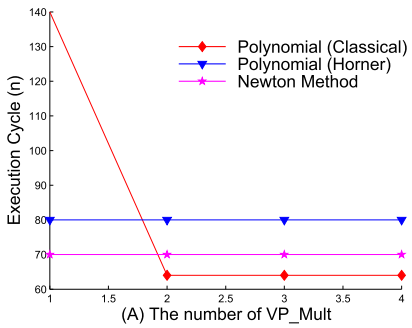


Fig. 3 The execution cycle of approximation algorithms. Given N_t=20, N_n=10 and the execution cycle for addition and multiplication operation are n and $3n$, respectively.

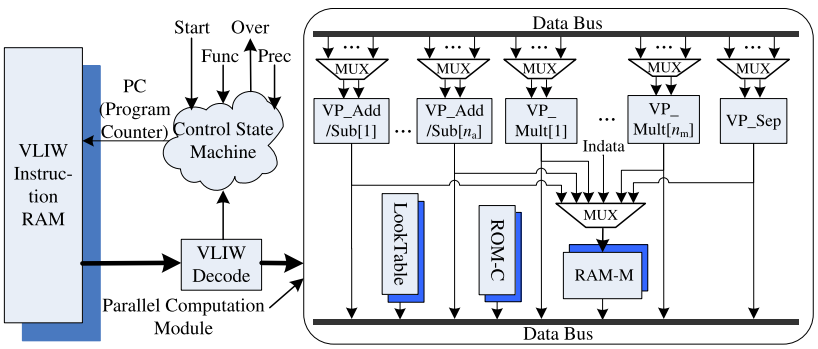


Fig. 2 Block diagram of the hardware design of VV-Processor.

it stores the initial value of reciprocal for the division operation and the approximation of reciprocal square root for the square root operation. ROM-C, a 8192×64 ROM with two read ports, is used to store constants. RAM-M, a 512×64 RAM with two ports, is used to buffer middle and final results. The port width of these memory modules is 64 bits and the maximum mantissa length is 63 64-bit words (4032 bits). A variable-precision number is stored in 64 consecutive 64-bit words and the lowest word contains the exponent field, sign bit, and mantissa length field.

VLIW instruction RAM is a 2048×120 RAM. Each word is a VLIW instruction, as shown in Fig. 4, which consists of several fields including the data selecting field (*sel_a*, *sel_b*) and precision set field (*m*, *n*, *r*) for each basic arithmetic unit, write enable field (*en_a*, *en_b*), address field (*Addr_a*, *Addr_b*) for each on-chip memory module, control field (*control*), iteration number (*num*), and so on. The decode module forms these fields from the VLIW instructions.

The control state machine module controls the run of VV-Processor. This module changes the value of program counter (PC) to access different VLIW instructions from VLIW instruction RAM to implement the corresponding VP elementary functions.

Both Newton's algorithm and polynomial approach algorithm are implemented with iterative method, which is controlled by the control field of VILW instruction, as shown in Table 1. First, we use the barrier instruction to synchronize all basic VP units. When a barrier instruction is reached, the unit will wait until all other units have finished their works. Then, a branch instruction, which modifies the PC value depending on the outcome of the condition, is followed to judge the completion of iterative computation. This way has to stall the execution of some units. However, it exerts little influence on the total performance, since the performance of VV-processor is depended on key operations in the critical path, as illustrated in Sect. 4. This means that the most time-consuming unit, such as VP multiplier, will be the bottleneck.

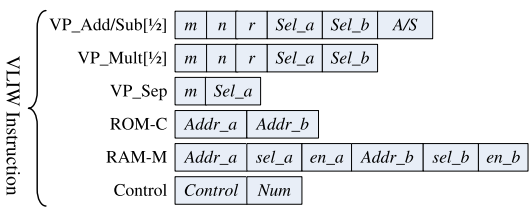


Fig. 4 The field of custom VLIW instruction.

Table 1 Partial definition of control field.

control	Description
0000	Normal instruction: Execution in order, i.e. PC=PC+1
0001	Barrier instruction: Execution the same instructions until all basic VP units are done
0010	Branch instruction: Select the VLIW instruction according to condition, i.e. if(cond) PC=Addr_Yes, else PC=Addr_No
1111	Termination instruction: Generate *over* signal and set PC=0

VV-Processor works through the following stages:

Stage 1, Start-up: After receiving the *Start* signal, the initial address to access VLIW instruction RAM is fixed according to the type of function (*Func*). At the same time, the initial data (*Indata*) is written into RAM-M.

Stage 2, Calculation: The VLIW instruction is read, the execution of computation module is controlled, and the value of PC is changed according to the control field of VLIW instruction.

Stage 3, Normalization: The *Over* signal is generated after the calculation of given function is done.

3.3 VP_Mult Module

The VP_Mult module performs VP multiplication operation, in which the mantissa of the two operands are multiplied and the exponents are added, i.e., C=($S_A \otimes S_B$)*(M_A*M_B)* $2^{E_A+E_B}$, where the mantissa lengths of A, B, C are m, n, r, respectively. Figure 5 (A) shows the VP multiplication algorithm and Fig. 5 (B) depicts the structure of the VP_Mult module.

The VP_Mult module is comprised of $2p$+1 on-chip memories (RA[1]~RA[p], RB[1]~RB[p] and RC), p fixed-point multipliers (64×64), p fixed-point accumulators (134-bit), a normalization module, and control logic. RA[1]~RA[p] and RB[1]~RB[p], which are 64×64 RAMs with one read port and one write port, are used to store the copy of the mantissa of A and B, respectively. RC, a 128×64 RAM, is used to store the result of M_A*M_B.

To multiply m words (64-bit) number and n words number, $m * n$ partial products are generated and accumulated, which is the most time-consuming part in high-precision multiplication operation. Since $m * n$ partial products can be calculated in parallel, we can use multiple 64×64 fixed-point multipliers to reduce the latency. Thus, p 64-bit fixed-point multipliers are integrated into VP_Mult module, equipped with p 134-bit fixed-point accumulator and 2*p on-chip memories (RA[1]~RA[p] and RB[1]~RB[p]), as shown in Fig. 5 (B). Each multiplier reads the mantissa of A and B from corresponding RAMs (RA and RB). The group of fixed-point multipliers are running $\lceil (m+n)/p \rceil$ sweeps.

Figure 5 (C) shows an example of the multiplication operation. In the first sweep, $A_0 \times B_0$ in multiplier [1], $A_1 \times B_0$, $A_0 \times B_1$ in multiplier [2] and $A_2 \times B_0$, $A_1 \times B_1$, $A_0 \times B_2$ in multiplier [3] are calculated simultaneously. The mantissa words of C are calculated from the least to the most significant. The least significant word is the low 64-bit result of accumulator [1] in the first sweep, i.e., RC[0]=PS_0[63:0]. The following significant word RC[1], overlapped by $A_0 \times B_0$, $A_1 \times B_0$, and $A_0 \times B_1$, is the summation of the high 64-bit result of the accumulator [1] and the low 64-bit result of the accumulator [2]. The high 64-bit result of accumulator[3] in the first sweep will be propagated to the next sweep and added into RC[4], as line 8 in Fig. 5 (A).

The execution time (T_{exe}) of the mantissa product is about $(\lceil \frac{m*n}{p} \rceil + p + 4)$. As the spacetime diagram shown

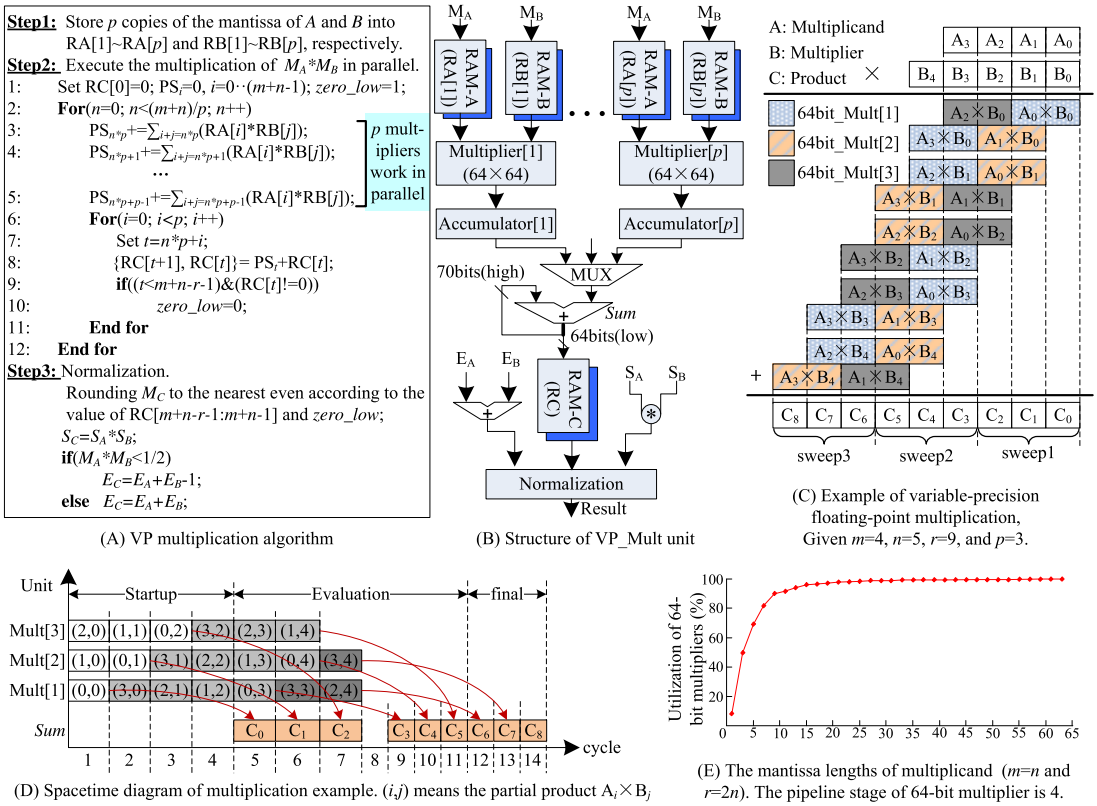


Fig. 5 Variable-precision multiplication algorithm and block diagram of hardware design.

in Fig. 5 (D), the $m * n$ partial products are generated by p 64-bit pipeline multipliers in parallel without any stalls, and the load of each multiplier is balanced. Thus, $\lceil \frac{m*n}{p} \rceil$ cycles can finish the generation of partial products. In addition, 4 cycles are required to fill the pipeline of multipliers at the start and p cycles are required to accumulate the final partial products into RC. As shown in Fig. 5 (E), the utilization of the multipliers are increasing with the mantissa length. Due to the pipeline fill time, the utilization is low for small mantissa length. However, it can reach 90% when the mantissa length is bigger than 9.

The final product C will be rounded correctly in the rounding to nearest mode, extending the IEEE standard to VP arithmetic. We build one flag, called *zero_low*. The *zero_low*, which is 1 if all bits in the $RC[0]$~$RC[m+n-r-2]$ are 0, is used to indicate a special case that C is the middle of two consecutive floating-point numbers. Since the product of the mantissa is between 1/4 and 1, it may be necessary to shift M_A*M_B left one position and decrement the exponent to normalize the product C.

The execution time of VP multiplication composed of three parts. The first part (T_{init}) is the initial time required to store M_A and M_B to RA and RB, and T_{init}=max{m,n}. The second part (T_{exe}) is the execution time. The last part (T_{nor}) is the normalization time and $T_{nor}=r$. Thus, the total execution time (cycle) of VP_Mult module is ($m \geq n$):

$$T_{VP\_Mult}=\left\lceil \frac{m*n}{p} \right\rceil + p + m + r + 4.$$

3.4 VP_Add/Sub Module

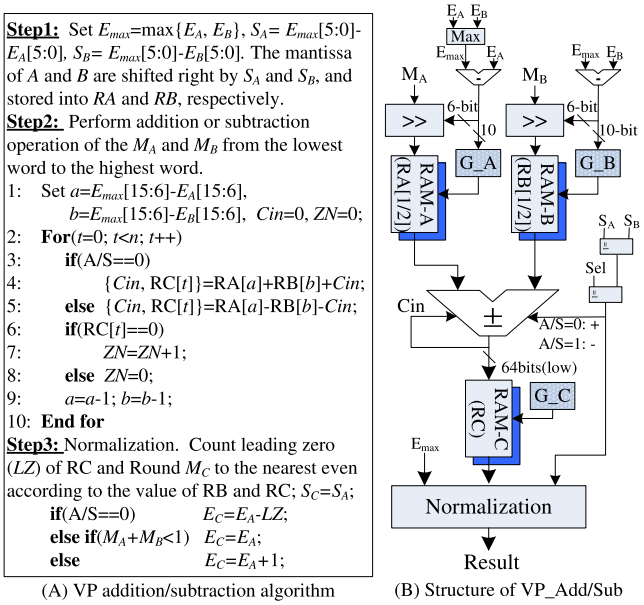
The VP_Add/Sub module performs VP addition or subtraction operation ($C=A \pm B$), where the mantissa lengths of A, B, C are m, n, r, respectively. If it is assumed that $A \geq B$, then

$$C=2^{E_A}(S_A*M_A \pm S_B*M_B*2^{E_B-E_A}).$$

Figure 6 depicts the VP addition and substraction algorithm and the structure of VP_Add/Sub module. The VP_Add/Sub module consists of three on-chip memories (RA, RB, and RC, which are 64×64 RAM with one read port and one write port), two 128-bit shifters, a 134-bit fixed-point adder, a normalization module, and three address generation modules (G_A, G_B, G_C).

The first step of VP addition or subtraction is to align the mantissas of A and B into consistent fixed-point formats, so they can be summed up with a simple fixed-point adder. We use a two-level alignment scheme to avoid using a very long shifter. In the first level, two 128-bit barrel shifters are used to align the mantissas within 64-bit word and the results store into RA and RB, respectively. In the second level, the difference between E_A and E_B is used in address generation modules (G_A and G_B) to select the appropriate words from RA and RB to align each word of mantissa, as shown in Fig. 6 (B). For $A \geq B$, the least significant words to be evaluated are $RA[0]$ and $RB[(E_A - E_B) \gg 6]$.

The addition or subtraction of the mantissa of A and B is executed from the least significant word to the most sig-



(A) VP addition/subtraction algorithm (B) Structure of VP_Add/Sub

Fig. 6 Variable-precision addition and subtraction algorithm ($A \geq B$).

nificant word, as shown in Fig. 6 (A). For the addition operation, the least mantissa part of C is RB[0]~[b-1]; for the subtraction operation, the least mantissa part of C is the two's complement of RB[0]~[b-1]. Simultaneously, ZN is calculated, which represents the number of leading zero words and is used to count the leading zero of RC quickly in the normalization.

The final result C will be rounded correctly in the rounding to nearest mode. For addition operation, since the value of $(M_A+M_B*2^{E_B-E_A})$ is between 1/2 and 2, it may be necessary to shift the RC right one position and increment the exponent. For subtraction operation, since the value of $(M_A-M_B*2^{E_B-E_A})$ is between $-1/2$ and 1, it is necessary to count leading zero of result C before normalization. First, the address (AH) of the most significant word not equal to 0 is fixed according to ON (AH=n-ON). Then, the leading zero (LZ_{AH}) of RC[AH] is counted. Thus, the number of leading zero (LZ) of C is (64*AH-LZ_{AH}) and E_C=E_A-64*AH-LZ.

Similar to VP multiplication, the total execution time of VP addition/subtraction is composed of initial time (T_{init}=n), execution time (T_{exe}=n), and normalization time (T_{nor}=r). The VP_Add/Sub unit is equipped with a ping-pong memory structure, RA[1]&RB[1] and RA[2]&RB[2]. So, the initialization and the computation can be executed in parallel and the total execution time is about n+r.

4. Variable-Precision Elementary Function Algorithm

For each VP elementary function, we need to design corresponding VLIW instructions in VV-Processor and dynamically specify the precision of the basic operations to yield the desired result and achieve high performance. First, the elementary function is decomposed into a series of basic VP

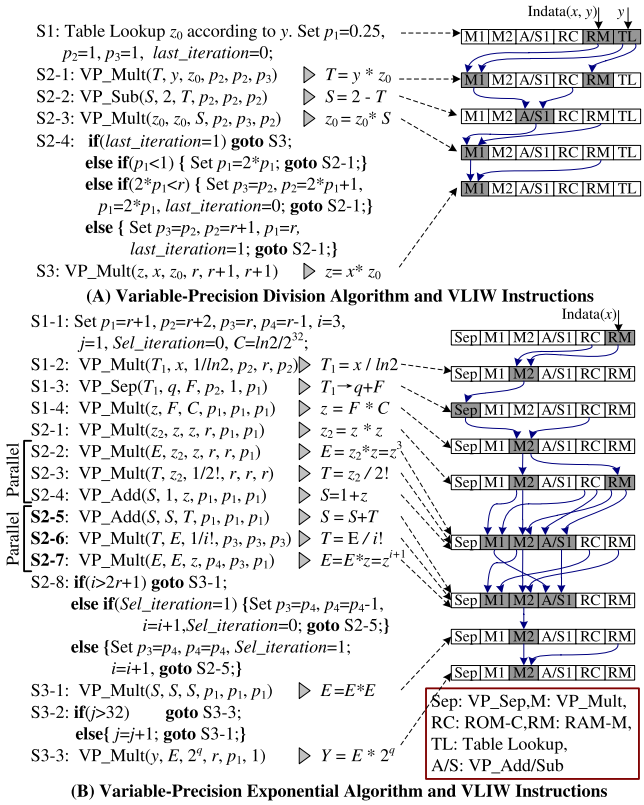


Fig. 7 VP division algorithm and exponential function algorithm.

arithmetic operations, which can be executed by the basic arithmetic units. Then, these basic operations are mapped into the custom VLIW instructions according to the data dependent between them. Finally, the corresponding VLIW instructions are executed though the explicitly parallel technology of multiple VP arithmetic units.

In this section, we use division and exponential function as examples to illustrate the design of VP elementary function in VV-Processor. Then, the implementation algorithms of others elementary functions (square root, logarithm, triangle sine and cosine) are introduced in Appendix.

4.1 VP Division Algorithm

As shown in Fig. 1 (B) and Fig. 7 (A), the proposed method for evaluating VP division (z=x/y) is based on table lookup and Newton's method, consisting of three stages.

Stage1: Look up table for an approximation z_0 of $1/y$, which is a starting value for Newton's method.

Stage2: Newton's method is employed to calculate the reciprocal of y, and the iteration is $z_{i+1} = z_i(1 + \varepsilon_i)$, where ε_i=1-$z_i y$ and z_n=$1/y$.[22] This iteration has quadratical convergence and the number of iterations is N_n=$log_2(prec_r/prec_i)$.

Stage3: VP multiplication operation is used to obtain z= x*z_n=x/y.

This algorithm uses a 256 word table for an initial approximation z_0 of $1/y$. Each word in the table has 9 bits.

The eight highest significant bits of mantissa of y are used to access the lookup table to obtain the mantissa of z_0 (M_{z_0}). The precision of z_0 is 8-bit and the exponent of z_0 is $E_{z_0} = -1 - E_y$.

The latency of VP division is reduced through dynamically varying the precision of intermediate computation in each iteration. In stage2, p_1, representing the mantissa length of result in this iteration, is increase by a factor of 2 due to the quadratical convergence of Newton's method; p_2, representing the precision of intermediate results in this iteration, is p_1+1; p_3 represents the precision of intermediate results in the front iteration. Since the precision of z_0 is 8-bit, the precisions of result in the first and second iteration are 16-bit and 32-bit, respectively. So we can use 64-bit computation bandwidth to obtain the desired result. In the i^{th} iteration (i>2), the precision of result is 64*p_1 bits, so 64*(p_1+1) bits computation bandwidth is required.

As depicted in the Fig. 7 (A), there are data dependent among S2-1, S2-2 and S2-3. So they must be executed serially. The critical path in stage2 includes two multiplication operations and one subtraction operation. The computational complexity of addition and multiplication are O(n) and O(n^2), respectively [20]. Thus, multiplication is a key operation and the latency of VP division algorithm reduces with the performance improvement of VP_Mult.

4.2 VP Exponential Function Algorithm

As shown in Fig. 7 (B), the VP exponential algorithm (y=e^x) is based on the Taylor series and is performed through three stages as follows.

Stage 1, Range reduction: First, the argument x is reduced into interval [0, ln2). We find s and an integer q, satisfied x=q*ln2+s and $0 \le s < \ln 2$. Then $y = e^x = e^s * 2^q$. Thus, the evaluation of the exponential function on the real field is transferred into that on interval [0, ln2). To derive the value of s and q, x is multiplied by 1/ln2, and the integer part of product is q. The fraction part of product is multiplied by ln2 to gain the value of s.

To reduce the degree of Taylor series, we repeat the doubling formula ($e^{2x} = (e^x)^2$) to reduce the argument further. Given $z = s/2^{32}$, then $e^s = (e^z)^{2^{32}}$ and $|z| < 2^{-32}$. This is better since the power series converges more quickly for z. The cost is that the 32 squaring are required to reconstruct the final result from e^z.

Stage 2, Evaluation: e^z can be evaluated using the Taylor series approximation approach:

$$e^z = \sum_{n=0}^{i} \frac{z^n}{n!} = 1 + z + \frac{z^2}{2!} + \frac{z^3}{3!} + \frac{z^4}{4!} + \cdots$$

Stage 3, Reconstruction: The value of e^s is calculated 32 times using the doubling formula starting from e^z.

Since at most one bit of accuracy will lose in each multiplication operation [18], the precision of result in Stage 2 should be greater than 64r+32. The error in Stage 2 comes from two sources: approximation error (ε_a) and rounding error (ε_r). The $|z| < 2^{-32}$ and the ignored terms in the Taylor series produce the approximation error

$$\varepsilon_a = \sum_{j>i} z^j/j! < z^{i+1} < 2^{-32(i+1)} \le 2^{-(64r+32)}.$$

Thus, we need to calculate the front 2r+1 terms of the Taylor series. Rounding error occurs when calculating the Taylor series. If we use 64(r+1) bits as the internal working precision and the number of operations is smaller than 2^{32}, the rounding error ε_r will be smaller than $2^{-(64r+32)}$.

As shown in Fig. 7 (B), the Taylor series approach in stage 2 is a kind of polynomial approximation approach. S2-5 implements addition operations ($F(i\text{-}1)+a_i*x^i$) in i^{th} iteration, and S2-6 implements multiplication operations ($a_{i+1}x^{i+1}$) in $(i+1)^{th}$ iteration, and S2-7 implements multiplication operations (x^{i+2}) in $(i+2)^{th}$ iteration of the polynomial approximation approach. There is no data dependent between them, thus they can be integrated into one VLIW instruction, and executed simultaneously. Therefore, the critical path in Stage 2 includes only one VP multiplication or addition/subtraction.

5. Experiments Result

5.1 Experimental Setup

We implemented the proposed hardware design on a development board mainly consisting of two FPGAs (Virtex-5 XC5VLX50T-1FF1136 and Virtex-6 XC6VLX760-2FF1760) and two 2GB DDR2 DRAM modules, as shown in Fig. 8. The XC5VLX50T chip provides a link between the XC6VLX760 and host PC via a PCI-Express (8×) bus with bandwidth of 570 MB/s. Each DDR2 Controller runs at 200 MHz on 128 bit data width. The peak I/O bandwidth can reach 6.4 GB/s. All modules are coded in Verilog, and synthesized with Xilinx ISE 12.3. As a base for performance comparison, we applied the MPFR library, MPFR3.0.0, developed by Hanrot et al., to measure the results accuracy and delay time. This library is quite efficient and accuracy compared to other multiple-precision library.[3] The software platform includes a host PC with 2.93 GHz Intel Core i3 530 CPU and 4 GB DDR3 1333 MHz Memory.

We build a VP arithmetic accelerator (VP-Acc) to measure the performance of FPGA chip and take the communication overhead between host PC and FPGA chip into consideration. Figure 8 shows the block diagram of this accel-

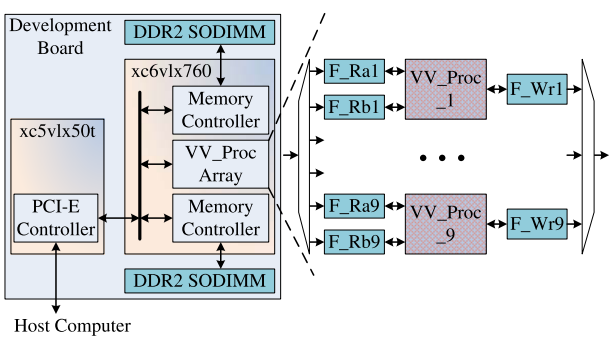


Fig. 8 VP arithmetic accelerator based-on VV-Processor.

erator, which includes a VV-Proc Array module and others control logic. The VV-Proc Array module is composed of 9 VV-Processor units and 27 FIFOs. Each VV-Processor unit reads operands A and B from corresponding FIFOs (F_Ra and F_Rb) and writes the result into FIFO (F_Wr), respectively. The running time of this accelerator includes computation time and the time for sending the operands to FPGA as well as receiving the results back to the host PC.

5.2 Resource Utilization

Table 2 shows details of the FPGA synthesis results for basic VP arithmetic units, VV-Processor unit, and VP arithmetic accelerator equipped with 9 VV-Processor units.

A. DSP resource

The DSP48E1 blocks, used to build 64-bit fixed-point multiplication module in VP_Mult, are the most constrained resource. The computational complexity of VP addition and multiplication are $O(n)$ and $O(n^2)$, respectively. Thus, the VP_Mult module is a key unit in VV-Processor. In our design, four (p=4) parallel 64-bit fixed-point multipliers are equiped to obtain the best tradeoff between the performance and resource utilization. One VV-Processor consumes 11% DSP48E available in XC6VLX760 FPGA chip. Therefore, a total of 9 VV-Processor units can be integrated into VP arithmetic accelerator.

B. Local memory resource

The local memory modules (on-chip memory), classified into distributed RAM and embedded 18 Kbits block RAM, are important in the implementation of VV-Processor. We used 29 and 14 block RAM to implement ROM-C (8192×64 bit) and VLIW instruction RAM (2048×120 bit), respectively. The more available storage resources in current FPGAs can be used to build a larger table lookup, which provide a rough approximation to the elementary function in the range reduction. Thus, the latency of evaluation is reduce further. We used distributed RAM to implement other small on-chip memory, such as RAM in basic variable-precision arithmetic units. The distributed RAM elements are mapped into LUT slices, which is an advantage for the FPGA placement and layout phase, compared with fix-positioned embedded block RAMs. However, distributed RAMs consume more LUT resources. An VV-Processor unit requires about 3% of the slice LUT resources available in XC6VLX760-2FF1760, and the VP arithmetic accelerator consumes 31% of the Slice LUTs.

C. Frequency

The achievable maximum frequency of VV-Processor unit is 253 MHz. Compared to the same circuit implemented directly in silicon (ASICs), FPGA implementation, emulated with a very large number of configurable elementary blocks and network of wires, is typically one order of magnitude slower. However, the performance of FPGAs is improved through the custom hardware for applications equipped with multiple VV-Processor units working in parallel. In VP-Acc, the final synthesis frequency can reach 245 MHz and the running frequency is 240 MHz.

Table 2 Synthesis results. 'VV-Proc' represents the VV-Processor unit, and 'VP-Acc' represents the VP arithmetic accelerator.

Type	Slice LUT	DSP48E1	BRAM	Freq(MHz)
VP_Mult	4095	48	0	262.03
VP_Add	2491	0	0	296.57
VP_Sep	986	0	0	287.55
VV-Proc	16235(3%)	96(11%)	43 (3%)	253.45
VP-Acc	147096(31%)	964(100%)	43(27%)	245.50

5.3 Performance Comparison of VV-Processor

In this subsection, we first evaluate the accuracy and performance of VV-Processor unit. Then, a VP arithmetic accelerator is built to compare the performance between FPGA chips and CPU.

A. Accuracy of VV-Processor

The accuracy of the proposed methods was tested by comparing to the accurate results produced by 4096 bits precision calculations using the MPFR library. For the elementary function, we designed corresponding variable-precision algorithms, in which the internal precision is carefully planed and guard words are used to guarantee accuracy of the result. The experiment results show that we can obtain results with correctly rounding for addition, multiplication, that ulp (unit in last place) error is not exceed 0.5, and the error is smaller than 0.55 ulp for VP elementary functions.

B. Performance comparison with a core of Core i3

Table 3 compares the performance of VV-Processor unit with the similar precision implementation of MPFR library [3]. This library runs at a single core of Core i3 processor. The speedup factor for basic arithmetic operations (addition, multiplication, division, and square root) is between 5 and 34, and that for elementary functions (exponential, sine, cosine and logarithm) is between 18 and 36.9. In the logarithm algorithm, the reconstruction, which occupies half of execution time for others transcendental functions, is simple. Therefore, the latency of VP logarithm function is smaller than that of others. However, more on-chip memory is required to store the approximation in lookup tables. The performance of FPGA implementation of VP applications is increase with the number of integrated VV-Processor units.

With the help of Intel VTune Performance Analyzer

Table 3 Timing in microsecond comparison with MPFR library, running on a core of Core i3 processor. The speed of VV-Processor is predicted based on the frequency in Table 2 and the number of cycles in Table 5.

Op	1024 bits			2048 bits		
	MPFR	Our	Speedup	MPFR	Our	Speedup
$x \pm y$	0.7	0.126	**5.6**	1.25	0.25	**5**
$x \times y$	12.9	0.41	**31.5**	32.18	1.30	**24.8**
x/y	18.6	1.95	**9.5**	64.1	5.05	**12.7**
$\sqrt{x}$	18.8	2.52	**7.5**	46.9	6.39	**7.3**
Sin(x)	458	21.0	**21.8**	1766	82.0	**21.5**
Cos(x)	405	22.2	**18.2**	1640	73.5	**22.3**
Exp(x)	420	23.0	**18.3**	1515	83.2	**18.2**
Ln(x)	579.7	15.7	**36.9**	1547	46.1	**33.6**

Table 4 Performance analysis of Vtune, where "N.I." means number of INST_RETIRED.ANY events, which counts the number of instructions that retire execution. CPI is the abbreviation of cycle per instruction.

Library		Div		Exp	
		256	2048	256	2048
MPFR	N.I.	13381	285367	210476	6646332
	CPI	0.71	0.66	0.78	0.67
	Time(μs)	3.22	6.41	56.09	1515

Table 5 Performance (cycle) comparison with VPIAP [9] and CORDIC [12].

Op	1024 bits			2048 bits		
	[9]/[12]	Our	Speedup	[9]/[12]	Our	Speedup
$x \pm y$	40	32	**1.25**	72	64	**1.13**
$x \times y$	284	104	**2.96**	1068	328	**3.33**
x/y	852	493	**1.73**	3220	1280	**2.52**
$\sqrt{x}$	1146	639	**1.79**	4314	1620	**2.66**
Sin(x)	11 K	5.2 K	**2.12**	38 K	20.3 K	**1.87**
Cos(x)	11 K	5.5 K	**2.0**	38 K	18.2 K	**2.09**
Exp(x)	11 K	5.7 K	**1.93**	38 K	20.6 K	**1.84**
Ln(x)	11 K	3.8 K	**2.89**	38 K	11.4 K	**3.33**

tools, we can explain why the performance of VV-Processor, running at 253 MHz, is higher than CPU platform running at 2.93 GHz, for exponential function. As shown in Table 4, the value of "N.I." is 6,646,332 for 2048-bit exponential function in MPFR library. Most instructions are used in function calls, memory management, error and range checking, exception handling, and so on.

C. Performance comparison with related work

In Table 5, we compare the performance of our design to two existing design, VPIAP processor [9] and CORDIC processor [12], [17]. The VPIAP processor was designed to perform the basic VP arithmetic operations. The performance of addition in our design is about the same. For multiplication operation, the speedup of 3X is obtained, since multiple fixed-point multipliers are used to generate and accumulate the partial products in parallel in the multiplication of M_A and M_B. For division and square root, Newton-Raphson iteration with table-based method is used. The precision of approximation value z_0 is 8-bit, and the number of iteration is $log_2(prec_r)$-3. The speedup over 1.7X is obtained for division and square root.

The CORDIC processor, which was designed to perform VP transcendental functions. Since the approach based on multiplication operations has high level convergence and dynamically varies the precision of intermediate computation to reduce the latency further, our proposal outperforms the CORDIC processor by a factor of 1.8X-3.3X.

Moreover, in the comparison to the work in [9] and [12], our design can implement various VP algebraic and transcendental functions in the unified hardware.

D. Performance comparison between FPGA chips and Intel Core i3 processor

Table 6 summarizes the performance of VP arithmetic accelerator. We evaluate various VP algebraic and transcendental functions on the vector X and Y. The ratio of computation time to communication time is increasing with the size (n) of X and Y and is tending towards stability when $n \geq 10\,K$, as shown in Fig. 9. Thus, there is little effect on the performance of VP-Acc for $n \geq 10\,K$. In the following, we set $n = 1\,M$. As shown in Table 6, the performance for VP addition and VP multiplication operations is limited by the I/O bandwidth of DDR2 DRAM. This leads to the low utilization of computational resource and low performance of FPGA chips. The utilization of VV-Processor units is only 18.5% for VP addition operation. The ratio of computation time to communication time is low for VP algebraic functions, due to the simple computational process. Thus, the communication overhead will take much percentage of the total running time. For VP transcendental functions, computational capability of FPGA chips is developed to the full. We can achieve the maximum speedup of 169. In the design of high-precision scientific application accelerator based on VV-Processor, the on-chip memory are used to store and reuse the data, in order to eliminate the limit of I/O bandwidth of DDR2 and PCI-Express.

Table 6 Performance of VP arithmetic accelerator comparison with MPFR library. 'Ratio' represents the ratio of communication time to the total running time. 'Util' represents the utilization of VV-processor units.

Op	1024 bits				
	MPFR[1](s)	VP-Acc[2](s)	Speedup	Ratio	Util
$x \pm y$	0.36	0.52	**0.7**	84.7%	18.5%
$x \times y$	6.06	0.52	**11.6**	84.7%	60.2%
x/y	9.78	0.67	**14.6**	65.9%	100%
$\sqrt{x}$	12.53	0.52	**24.3**	42.8%	100%
Sin(x)	220.79	2.69	**82.2**	8.2%	100%
Cos(x)	188.8	2.83	**66.8**	7.8%	100%
Exp(x)	204.02	2.92	**69.8**	7.6%	100%
Ln(x)	322.06	1.98	**163.1**	11.2%	100%

1. MPFR: Intel Core i3-530 (2 cores and 4 threads) 2.93 GHz CPU (32 nm), 4 MB Intel Smart Cache, Parallel implementation based-on OpenMP
2. VP-Acc: Virtex-6 XC6VLX760-2FF1760 FPGA (40 nm), equipped with 9 VV-Processor units, running at 240 MHz

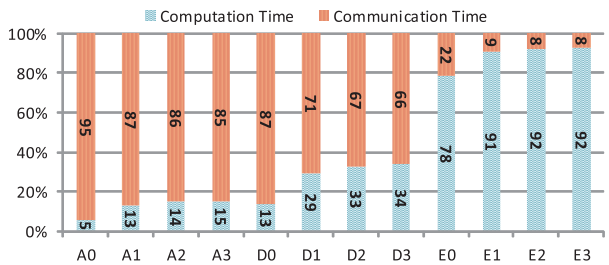


Fig. 9 The ratio of communication time to computation time, where A, D, E refer to addition, division, exponential operations respectively. 0, 1, 2, 3 represent that the size of X and Y is 1 K, 10 K, 100 K, 1 M respectively.

6. Conclusion and Future Work

We presented a custom processor for variable-precision floating-point arithmetic processor based on VLIW structure. It used the explicitly parallel technology of multiple basic arithmetic units to improve the performance. Several VP elementary function algorithms are designed in VV-Processor. The experimental results show this processor

could achieve 5X-37X performance speedup compared with MPFR library. Moreover, our hardware design can achieve better performance than the related works.

In future, we will explore methods to implement compound VP functions in VV-Processor, such as $\ln(1 + e^x)$. Further, we will apply the VV-Processor unit for accelerating large-scale scientific applications to exhibit the potential capability of FPGA chips to implement high-precision floating-point scientific applications.

Acknowledgment

This work is partially supported by NSFC (60833004) and 863 (2008AA01A201). We would like to thank the reviewers for their helpful comments.

References

[1] D.H. Bailey, "High-precision floating-point arithmetic in scientific computation," Computing in Science and Engineering, vol.7, no.3, pp.54–61, Jan. 2005.
[2] GNU Multiple-Precision arithmetic library, http://www.swox.com/gmp.
[3] L. Fousse, G. Hanrot, V. Lefevre, P. Pelissier, and P. Zimmermann, "MPFR: A multiple-precision binary floating-point library with correct rounding," Trans. Mathematical Software, vol.33, no.2, pp.1–15, 2007.
[4] Ntl: A library for doing number theory, http://www.shoup.net/ntl/.
[5] J. Fujimoto, T. Ishikawa, and D. Perret-Gallix, "High precision numerical computations—a case for an happy design," ACPP IRG note, ACPP-N-1: KEK-CP-164, May 2005.
[6] M.S. Cohen, T.E. Hull, and V.C. Hamarcher, "Cadac: A controlled-precision decimal arithmetic unit," IEEE Trans. Comput., vol.C-32, no.4, pp.370–377, 1983.
[7] D.M. Chiarulli, W.G. Ruaa, and D.A. Buell, "Draft: A dynamically reconfigurable processor for integer arithmetic," Proc. 7th Symposium on Computer Arithmetic, pp.309–318, 1985.
[8] T.M. Carter, "Cascade: Hardware for high/variable precision arithmetic," Proc. 9th Symposium on Computer Arithmetic, pp.184–191, 1989.
[9] M.J. Schulte and E.E.S. Jr, "A family of variable-precision, interval arithmetic processors," IEEE Trans. Comput., vol.49, no.5, pp.387–397, May 2000.
[10] E. El-Araby, I. Gonzalez, and T. El-Ghazawi, "Bringing high-performance reconfigurable computing to exact computations," Proc. FPL 2007, pp.79–85, Aug. 2007.
[11] A.F. Tenca and M.D. Ercegovac, "A variable long-precision arithmetic unit design for reconfigurable coprocessor architectures," Proc. FCCM '98, 1998.
[12] J. Hormigo, J. Villalba, and E.L. Zapata, "Cordic processor for variable-precision interval arithmetic," J. VLSI Signal Processing, vol.37, pp.21–39, 2004.
[13] E. Saez, J. Villalba, J. Hormigo, F.J. Quiles, J.I. Benavides, and E.L. Zapata, "FPGA implementation of a variable precision CORDIC processor," Proc. 13th Conf. on Design of Circuits and Integrated Systems (DCIS'98), pp.604–609, Nov. 1998.
[14] J. Zhou, Y. Dou, Y. Lei, J. Xu, and Y. Dong, "Double precision hybrid-mode floating-point fpga cordic coprocessor," Proc. HPCC2008, pp.182–189, 2008.
[15] J. Hormigo and J. Villalba, "A hardware algorithm for variable-precision division," Proc. 4th Conference on Real Numbers and Computers, pp.1–7, April 2000.
[16] J. Hormigo, J. Villalba, and M. Schulte, "A hardware algorithm for variable-precision logarithm," Proc. ASAP2000, pp.215–224, July 2000.
[17] J. Hormigo, J. Villalba, and E.L. Zapata, "Interval sine and cosine functions computation based on variable-precision cordic algorithm," Proc. 14th Symposium on Computer Arithmetic, pp.186–193, April 1999.
[18] Y. Dou, Y. Lei, and G. Wu, "FPGA accelerating double/quad-double high precision floating-point application for exascale computing," Proc. ICS2010, pp.325–336, 2010.
[19] K. Underwood, "FPGAs vs. CPUs: Trends in peak floating-point performance," Proc. FPGA2004, pp.171–180, 2004.
[20] Computational complexity of mathematical operations, http://en.wikipedia.org/wiki/Computational_complexity_of_mathematical_operations
[21] K.K. Parhi and H.R. Srinivas, "A fast radix-4 division algorithm and its architecture," IEEE Trans. Comput., vol.44, no.6, pp.826–831, 1995.
[22] R.P. Brent and P. Zimmermann, Modern computer arithmetic, Cambridge University Press, 2010.
[23] J.A. Fisher, "Very long instruction word architectures and the ELI-512," Proc. 10th Annual International Symposium on Computer Architecture, pp.140–150, 1983.
[24] A.K. Jones, R. Hoare, and D. Kusic, "An FPGA-based VLIW processor with custom hardware execution," Proc. FPGA2005, pp.107–117, 2005.
[25] R. Seshasayana and S.K. Srivatsa, "Implementation of novel pipeline VLIW architecture in FPGA," International Journal of Computer Science and Network Security, vol.7, no.7, pp.264–268, 2007.
[26] M.J. Schulte and E.E.S. Jr, "Hardware design and arithmetic algorithms for a variable-precision, interval arithmetic coprocessor," Proc. 12th Symposium on Computer Arithmetic, pp.222–228, 1995.

Appendix: VP Elementary Function Algorithms

(1) Variable-Precision Square Root Algorithm

As shown in Fig. A·1 (A), a similar scheme in division algorithm, which employs table lookup and Newton's method, can be used to evaluate square root($y = \sqrt{x}$), working as following:

Stage1: Look table for an approximation z_0 of $x^{-1/2}$.

Stage2: Use Newton's method to evaluate the reciprocal square root of x. The iteration is $z_{i+1} = z_i(1 + \frac{1}{2}\varepsilon_i)$, where $\varepsilon_i = 1 - z_i^2 x$ and $z_n \approx 1/\sqrt{x}$. This iteration has quartical convergence as the same to division algorithm.

Stage3: VP multiplication operation is used to obtain $y = x * z_n = \sqrt{x}$.

(2) Variable-Precision Sine Algorithm

As shown in Fig. A·1 (B), the approach based on Taylor series to calculate VP sine function (y=sin(x)) works as following stages.

Stage1, Range reduction: First, the argument x is reduced into interval $[0, 2\pi)$. We find s and an integer q, satisfied that x=q*2π+s, with the similar scheme of computation in stage1 in exponential function. Then y=sin(q*2π+s)=sin(s). Tripling formula ($\sin(3x) = 3\sin(x) - 4\sin^3(x)$) is used to reduce the argument further. Since $2\pi * 3^{-12} \le 2^{-16} < 2\pi * 3^{-11}$, then $z = s * 3^{-12}$ and $|z| < 2^{-16}$.

Stage2, Evaluation: The value of sin(z) can be evaluated using Taylor series approximation approach:

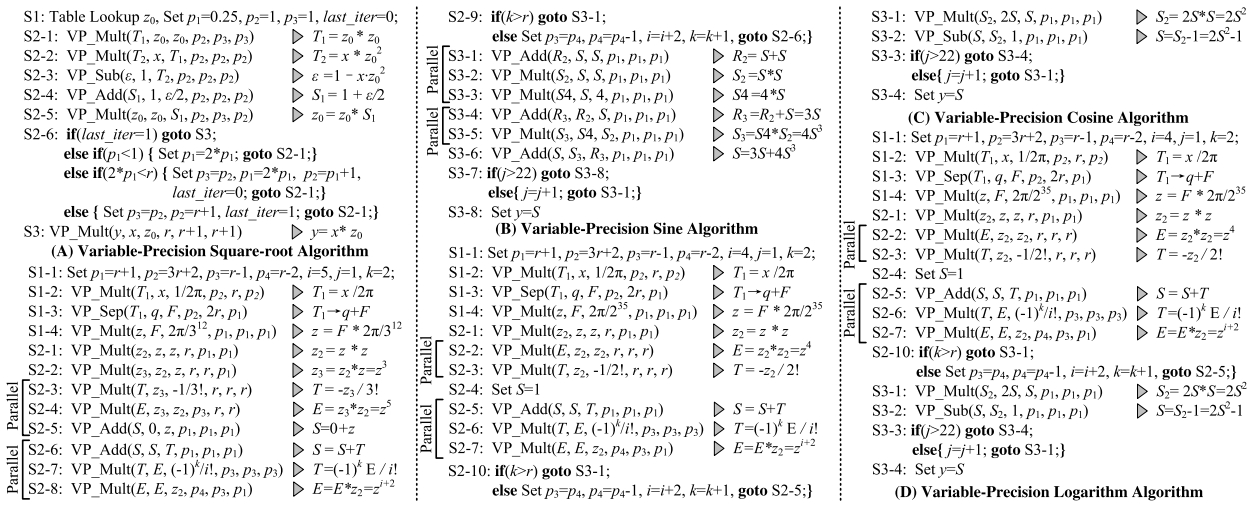


Fig. A·1 Variable-precision elementary function algorithms.

$$\sin(z) = \sum_{n=0}^{i} \frac{(-1)^n z^{2n+1}}{(2n+1)!} = z - \frac{z^3}{3!} + \frac{z^5}{5!} - \frac{z^7}{7!} + \cdots.$$

Stage3, Reconstruction: The value of $\sin(s)$ is calculated using tripling formula 12 times starting from $\sin(z)$.

(3) Variable-Precision Cosine Algorithm

As shown in Fig. A·1 (C), a similar scheme in sine function can be used for cosine function. The VP cosine algorithm for evaluating y=cos(x) works as following stages.

Stage1, Range reduction: First, reduce the argument x into interval $[0, 2\pi)$. Then y= $\cos(q*2\pi+s)$=$\cos(s)$. Formula $(\cos(2x) = 2\cos^2(x)-1)$ is used to reduce the argument further. Since $2\pi * 3^{-19} < 2^{-16}$, then $z = s * 2^{-19} \in [0, 2^{-16})$ and $|z| < 2^{-16}$.

Stage2, Evaluation: The value of $\cos(z)$ can be evaluated using Taylor series approximation approach:

$$\cos(z) = \sum_{n=0}^{i} \frac{(-1)^n z^{2n}}{(2n)!} = 1 - \frac{z^2}{2!} + \frac{z^4}{4!} - \frac{z^6}{6!} + \cdots.$$

Stage3, Reconstruction: The value of $\cos(s)$ is calculated using the recurrence formula $(\cos(2x) = 2\cos^2(x) - 1)$ 19 times starting from $\cos(z)$.

(4) Variable-Precision Logarithm Algorithm

As shown in Fig. A·1 (D), the VP logarithm algorithm uses a combined method of table lookup and polynomial approximation as described in [16] and is computed as

$$y = \ln(x) = \ln(x') + (E_x - 1) \cdot \ln(2)$$

where $x' = M_x$ and $1 \le x' < 2$. The computation of $\ln(x')$ works as following:

Stage1, Range reduction: The function $\ln(x')$ is evaluated using the iterative equation $x_{i+1} = x_i \cdot r_i$, where $x_0 = x'$, r_i is the i^{th} scale factors, and $0 \le i \le 7$. Thus, $\ln(x') = \ln(x_8) - \ln(r_0) - \cdots - \ln(r_7)$. The value of r_i and $\ln(r_i)$ is obtained from the lookup table according to the (2*i) most significant bits of x_i.

Stage2, Evaluation: The value of $\ln(x_8)$ is evaluated using a power series [22]

$$\ln(x_8) = 2(z + \frac{z^3}{3} + \cdots + \frac{z^{2i+1}}{2i+1} + \cdots)$$

where $z=(x_8-1)/(x_8+1)$. Compared to the general Taylor series expansion of $\ln(x_8)$, this $z < x_8/2$ if $x_8 > 1$.

Stage3, Reconstruction: The value of $\ln(x_8)$ and $\ln(r_i)$ $(0 \le i \le 7)$ are added to $(E_x - 1) \cdot \ln(2)$ to yield y.

Yuanwu Lei was born in 1982. He received his M.S. degree in Computer Science and Technology at National University of Defense Technology in 2007, and now he is a Ph.D. candidate at National University of Defense Technology. His research interests include high performance computer architecture.

Yong Dou was born in 1966, professor, Ph.D. supervisor, senior membership of China Computer Federation (E200009248). He received his B.S., M.S., and Ph.D. degrees in Computer Science and Technology at National University of Defense Technology in 1995. His research interests include high performance computer architecture, high performance embedded microprocessor, reconfigurable computing, and bioinformatics. He is a member of the IEEE and the ACM.

Jie Zhou was born in 1980. He received his M.S. degree in Computer Science and Technology at National University of Defense Technology in 2005, and now he is a Ph.D. candidate at National University of Defense Technology. His research interests include high performance computer architecture.