Designs and Applications for Variable-Precision, Interval Arithmetic Coprocessors

Michael J. Schulte and Earl E. Swartzlander, Jr.

***Abstract*—**

This paper presents hardware designs and numerical applications for coprocessors which have hardware support for variable-precision, interval arithmetic. The coprocessors give the programmer the ability to set the initial precision of the computation, determine the accuracy of the results, and recompute inaccurate results with higher precision. Special instructions and hardware features are provided for vectors, matrices, and complex numbers with either real or interval data elements. Three hardware designs with data paths of 16, 32, and 64 bits are examined. These designs are compared based on their estimated chip area, cycle time, and execution times for various numerical applications. Each coprocessor can be implemented on a single chip and has a cycle time which is comparable to existing IEEE floating point coprocessors.

I. Introduction

As noted in [1], [2], roundoff error and catastrophic cancellation in numerical computations can quickly lead to result which are completely inaccurate. Most computer systems, however, do not provide an efficient mechanism to determine the accuracy of the computation or to recompute inaccurate results with higher precision.

To improve the accuracy and reliability of numerical computations, several software tools have been developed. Early software packages, such as [3], [4], [5], supported variable-precision arithmetic, which gives the programmer the ability to specify the precision (i.e., the number of words or bits) of the computation. Later, scientific programming languages [6] were developed which provide special instructions and data types for interval arithmetic [7]. These scientific languages also support exact dot products, in which all arithmetic operations are mathematically exact and only a single rounding is performed at the very end [8]. Recently, techniques for variable-precision arithmetic and interval arithmetic have been combined in the languages PASCAL-XSC [9], C-XSC [10], ACRITH-XSC [11], and VPI [12]. These extended scientific languages provide data types and special instructions for variable-precision numbers, intervals, complex numbers, vectors, and matrices.

The main disadvantage of software tools for accurate, self-validating arithmetic is their speed. Since variable-precision, interval arithmetic operations are simulated in software, tremendous overhead occurs due to function calls, memory management, error and range checking, changing rounding modes, and exception handling. Although the IEEE 754 floating point standard [13] requires support for the directed rounding modes, changing the rounding modes on existing IEEE processors can take as long as executing six floating point additions, due to an inefficient user interface [14]. The interval arithmetic routines discussed in [15] are approximately 40 times slower than their single-precision floating point equivalents. Routines which support variable-precision, interval arithmetic (up to 56 decimal digits) are more than 1,200 times slower than the corresponding single-precision routines.

To overcome the speed limitation of existing software tools, direct hardware support is required. To improve the accuracy and performance of vector and matrix operations, processors which support exact dot products have been designed [16], [17], [18]. However, to produce accurate results when multiple operations are performed, exact dot products should be combined with variable-precision arithmetic [19]. Other processors, including CADAC [20], DRAFT [21], and CASCADE [22], have been design which provide hardware support for variable-precision arithmetic. Although these processors improve the speed of variable-precision computations, they do not provide special instructions for interval arithmetic or vector and matrix operations.

This paper presents hardware designs and numerical applications for variable-precision, interval arithmetic coprocessors (VPIACs). Section 2 gives an

The authors are with Department of Electrical and Computer Engineering at the University of Texas at Austin

overview of the number representation and hardware design of the coprocessors. In Section 3, area and delay estimates are given for three VPIACs with data paths of 16, 32, and 64 bits. Cycle counts for arithmetic and interval operations are reported in Section 4, along with execution times for dot product computation, polynomial evaluation, and interval Newton method. Conclusions are given in Section 5. The software interface and arithmetic algorithms for the VPIACs are presented in [23].

II. Hardware Design

This section gives an overview of the number representation and hardware design for the VPIACs. The hardware is designed to handle the common case quickly, while still providing correct results and acceptable performance when extremely high precision is required. Each VPIAC functions as a tightly-coupled coprocessor which receives input data and instructions from the main processor. Standard floating point arithmetic is performed by the main processor, and the VPIAC handles all variable-precision, interval computations. The hardware supports the four rounding modes specified in the IEEE 754 floating point standard: round-to-nearest even, round toward positive infinity, round toward negative infinity, and round toward zero. In the following discussion, the data path size (i.e., the number of bits per word in the significand (mantissa) of the variable-precision number) is denoted as k, and a VPIAC with a data path of k bits is referred to as a k-bit VPIAC.

The format for variable-precision numbers is shown in Figure 1. Intervals are represented by two variable-precision numbers, which correspond to the interval endpoints. Each variable-precision number consists of a 16-bit exponent field (E), a sign bit (S), a 2-bit type field (T), a 5-bit significand length field (L), and a significand (F) which consists of $L+1$ significand words ($F[0]$ to $F[L]$) The exponent is represented with a bias of 32,768. The sign bit is zero if the number is positive and one if it is negative. The type field indicates if a number is infinite, zero, or not-a-number. The length field specifies the number of k-bit words in the significand. The words of the significand are stored from least significant $F[0]$ to most significant $F[L]$. The significand is normalized between 1 and 2, but does not use a hidden one. The value of a variable-precision floating point number VP is

$$VP = (-1)^S \times F \times 2^{E-32,768}$$

They have a maximum precision of $32k$ bits and their range is approximately

$$[2^{-32,768}, 2^{32,769}] \approx [10^{-9,864}, 10^{9,864}]$$

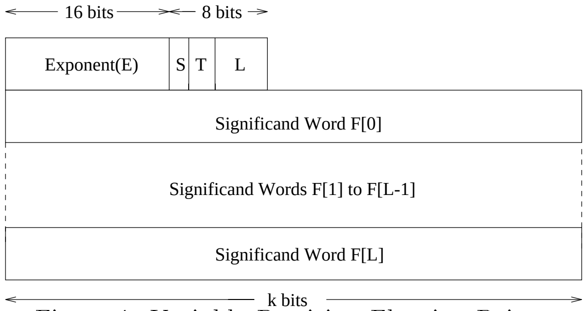

Figure 1: Variable-Precision Floating Point Format.

In comparison, IEEE double-precision floating point numbers have a maximum precision of 53 bits and their range is approximately

$$[2^{-1,022}, 2^{1,024}] \approx [10^{-307}, 10^{308}]$$

A block diagram of the hardware unit which performs variable-precision, interval arithmetic is shown in Figure 2. Control signals are shown as dashed lines. The significand and exponent data paths are depicted as bold and plain lines, respectively. The main components of the hardware unit are the register file, a k-bit by k-bit multiplier, a $2k$-bit adder, a 4 word by $2k$-bit selector, a long accumulator consisting of 64 $2k$-bit segments, two $2k$-bit shifters, and a 16-bit exponent adder.

The register file consists of two memory units: a 64-word by 32-bit header memory, and a 256-word by k-bit significand memory, as shown in Figure 3. Each header word contains the exponent, sign, type, and length of the variable-precision number, along with an index which points to the least significant word of the corresponding significand. When operations are performed on variable-precision numbers, the header words are first read. In the following cycles, the significands words are accessed based on the value of the index fields. The header and significand memories have two read ports and one write port. This allows two operand words to be read and one operand word to be written in each cycle.

Operands which are read from the register file go into either the multiplier or the selector. The selector performs comparison operations and determines which values go into the adder and the shifter. The long accumulator stores intermediate variable-precision results, as well as the partial sum of dot products.

III. Area and Delay Estimates

Tables 1 gives the hardware requirements for an IEEE double-precision coprocessor (IEEE DPC), and three VPIACs with data paths of 64, 32, and 16 bits. Here, w is the number of words and b is the number

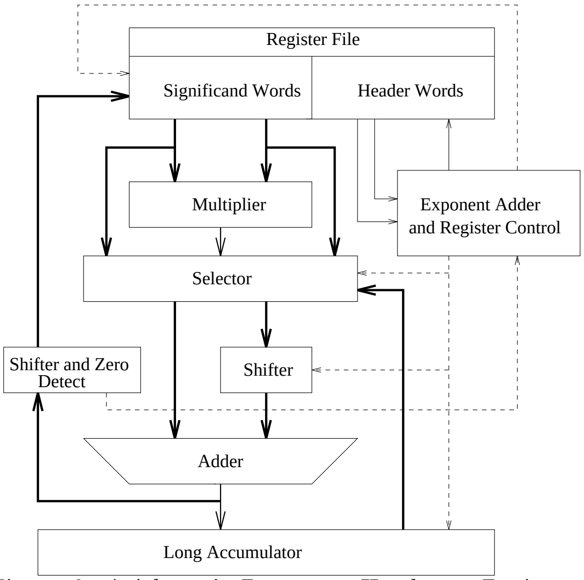

Figure 2: Arithmetic Processor Hardware Design.

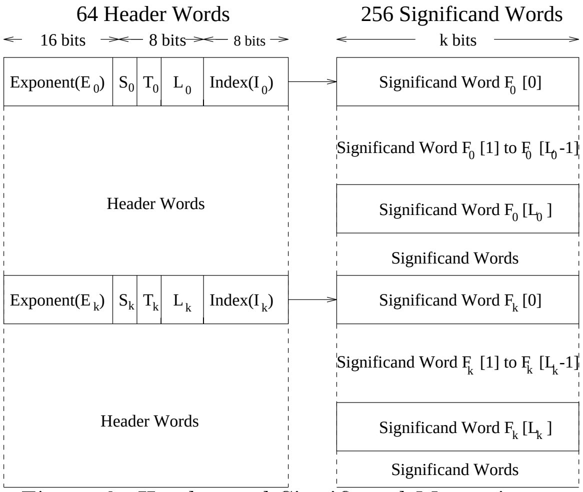

Figure 3: Header and Significand Memories.

of bits. For example, the long accumulator for the 64-bit VPIAC consists of 64 words, each of which is 128 bits long. The IEEE DPC does not use a long accumulator, header memory, or operand selector. Compared to the VPIACs, it uses a much smaller register file.

Area estimates are given in Table 2. These estimates are made based on data from a 1.0 micron CMOS standard cell library [24]. The estimates for the multiplier assume that multiplication is implemented using a Reduced Area Multiplier [25], followed by a carry look-ahead adder. The area of each component is estimated by calculating the total size of the macrocells (e.g., AND gates, full adders, half adders, etc.) which make up the component and then adding an additional 50 percent for internal wiring. The total area is estimated as the sum of the component areas plus an additional 60 percent for control logic, global routing, unused space, and pad area. The total estimated chip areas for the 64-bit, 32-bit, and 16-bit VPIACs are 227.5 mm^2, 101.9 mm^2, and 51.4 mm^2, respectively. The IEEE DPC has a total area of 100.8 mm^2, which is almost the same as that of the 32-bit VPIAC.

Delay and cycle time estimates are given in Table 3. The delay of each component is computed by taking the worst case delay of the critical path and adding 25 percent for unexpected delays and clock skew. The multipliers use two cycles. In the first cycle the partial products are reduced to two numbers. In the second cycle these numbers are added together to produce the product. The cycle time for each design is the sum of the multiplier (cycle 1) delay and the latch delay. The cycle times for the 64-bit, 32-bit, and 16-bit VPIACs are 22.0 ns, 16.0 ns, and 12.5 ns, respectively. The 64-bit, 32-bit, and 16-bit VPIACs can operate at roughly 45 MHz, 60 MHz, and 80 MHz, respectively. The IEEE DPC has cycle time of 20 ns and can operate at roughly 50 MHz.

IV. Cycle Counts and Execution Times

Table 4 shows the number of cycles required for addition/subtraction, short multiplication, multiplication, square, division, square root, and dot product on both real and interval operands. The number of cycles required to compute the midpoint of an interval, and the intersection of two intervals are also given. The number of k-bit words in each of the operands is denoted as n. Short multiplication refers to the product of a one word multiplier and an n word multiplicand. For the dot product operation, m represents the number of elements in the two vectors whose dot product is being computed. The cycle counts reported include the cycles needed for instruction fetch, instruction decode, reading the operands from the register file, performing the operation, rounding the result, and storing the rounded result back into the register file. The cycle counts given assume that the operands are already in the register file. If they must first be fetch from main memory, approximately $20 + 2w$ additional cycles are required, where w is the number of k-bit words which are read in. To improve performance, pipelining of the arithmetic operations and operand forwarding is used.

The algorithms used for addition/subtraction, short multiplication, midpoint, and interval intersection are $O(n)$, and the algorithms for multiplication, division, and square root are $O(n^2)$. The algorithm for dot product computation is $O(m \cdot n^2)$. Although algorithms with better asymptotic complexities exist, they require more control logic and are slower for small and moderate precisions. Hardware support and efficient implementation of the interval operations allows them to be executed in approximately twice as many cycles as the equivalent operations on real data elements.

Tables 5 and 6 show execution times for real and interval dot products, with $m = 16$. The precision, p, is varied from 64 to 2,048 bits. Although the 16-bit and 32-bit VPIACs support maximum precisions of 512 and 1,024 bits, respectively, higher precision computations may be simulated using standard techniques. The cycle counts for real and interval dot products (with $m = 16$) are

$$CyclesDotReal = 32n^2 + 2n + 212$$

$$CyclesDotInterval = 64n^2 + 4n + 452$$

where $n = p/k$. The execution time is computed as the product of the number of cycles and the cycle time, which is given in Table 3. For example, for the 32-bit VPIAC, if the precision of the computation is 512 bits, then $n = 16$, $CyclesDotReal = 8,436$, and $CyclesDotInterval = 16,900$. Since the cycle time for the 32-bit VPIAC is 16 ns, the execution times for real and interval dot products are

$$ExecDotReal = 16 \times 8,436 = 134,976ns$$

$$ExecDotInterval = 16 \times 16,900 = 270,400ns$$

The 64-bit VPIAC has the shortest execution times, but the largest area requirements. The 16-bit VPIAC has the smallest area requirements, but the longest execution times. The 32-bit VPIAC offers a good compromise between the two designs, with fairly low area and good execution times. When the precision is relatively low (i.e., 128 bits or less), the execution times of the 32-bit VPIAC and the 64-bit VPIAC are quite close. For example, for real dot product computations with 64 bits of precision, the 32-bit VPIAC has an execution times which is only 1.7 percent greater than the execution time of the 64-bit VPIAC. This occurs because although the 64-bit VPIAC requires fewer cycles to compute the dot product, it has a longer cycle time.

Tables 7 and 8 show the execution times for real and interval polynomial evaluation, for a 20 term polynomial. Horner's rule is used, so that each term in the polynomial requires 1 addition/subtraction and one multiplication. The cycle counts for evaluating a 20 term polynomial are

$$CyclesPolyReal = 20n^2 + 60n + 400$$

$$CyclesPolyInterval = 40n^2 + 120n + 680$$

When the precision is 64 bits, the 32-bit VPIAC actually has a lower execution time than the 64-bit VPIAC, due to its shorter cycle time. As the precision increases, the n^2 term dominates and the 64-bit VPIAC has the shortest execution time. For high precisions, doubling the precision increases the execution time by nearly a factor of four.

Table 9 shows the execution times for one iteration of interval Newton method [7]. The interval Newton method takes an interval X_i which includes a zero of the function $f(x)$, and computes a tighter interval X_{i+1} which also includes a zero of $f(x)$ according to the following iterative equation:

$$X_{i+1} = (mid(X_i) - \frac{f(mid(X_i))}{f'(mid(X_i))}) \cap X_i$$

where $mid(X_i)$ is the midpoint of the interval X_i.

For the execution times shown in Table 9, $f(x)$ and its derivative $f'(x)$ were chosen as

$$f(x) = 10x^2 - 5x + 3\sqrt{x} - 17$$

$$f'(x) = 20x + \frac{3}{2\sqrt{x}} - 5$$

Each iteration of the algorithm requires six additions/subtactions, 5 short multiplies, 1 square, 2 divides, 1 square root, 1 midpoint operation, and one intersection. Each of these operations is performed on interval operands. The number of cycles per iteration is

$$CyclesNewtonInterval = 19n^2 + 81n + 338$$

For all reported interval Newton methods, the 16-bit VPIAC has the longest execution times. The 64-bit VPIAC has the shortest execution times, except when the precision is 64 bits.

V. Conclusions

This paper examined hardware designs for VPIACs with data path widths of 64-bits, 32-bits, and 16-bits. The 16-bit VPIAC has the shortest cycle time and uses the least amount of area, but has longest execution times for the applications examined. The 64-bit VPIAC has the shortest execution times for most applications, but uses the largest amount of area and has the longest cycle time. The 32-bit VPIAC offers a good compromise between the two design. It uses less than half the area of the 64-bit VPIAC, and has comparable execution times for low to moderate precisions.

The VPIACs give the programmer the ability to set the initial precision of the computation, determine the accuracy of the results, and recompute inaccurate results with higher precision. They can

Table 1: Hardware Requirements

Unit	64-bit VPIAC	32-bit VPIAC	16-bit VPIAC	IEEE DPC
Multiplier	64 b by 64 b	32 b by 32 b	16 b by 16 b	53 b by 53 b
Adder	128 b	64 b	32 b	106 b
Significand Memory	256 w by 64 b	256 w by 32 b	256 w by 16 b	16 w by 64 b
Header Memory	64 w by 32 b	64 w by 32 b	64 w by 32 b	none
Long Accumulator	64 w by 128 b	64 w by 64 b	64 w by 32 b	none
Shifter/Zero Detect	128 b	64 b	32 b	106 b
Shifter	128 b	64 b	32 b	106 b
Operand Selector	4 w by 128 b	4 w by 64 b	4 w by 32 b	none
Exponent Add/Sub	16 b	16 b	16 b	11 b
Latches	128 b and 64 b	64 b and 32 b	32 b and 16 b	106 b and 53 b

Table 2: Area Estimates (mm^2)

Unit	64-bit VPIAC	32-bit VPIAC	16-bit VPIAC	IEEE DPC
Multiplier	49.4	15.2	4.9	37.4
Adder	5.2	2.1	1.0	4.3
Significand Memory	35.5	17.8	8.8	4.4
Header Memory	4.4	4.4	4.4	none
Long Accumulator	17.8	8.8	4.4	none
Shifter/Zero Detect	8.5	4.2	2.1	7.2
Shifter	8.2	3.9	2.1	6.9
Operand Selector	7.8	4.1	7.8	none
Exponent Add/Sub	0.6	0.6	0.6	0.4
Latches	4.8	2.6	1.4	2.4
Control, Pads, Space etc.	82.7	38.2	19.3	37.8
Total	227.5	101.9	51.4	100.8

Table 3: Delay and Cycle Time Estimates (ns)

Unit	64-bit VPIAC	32-bit VPIAC	16-bit VPIAC	IEEE DPC
Multiplier (Reduction)	20.0	14.0	10.5	18.0
Multiplier (Final Add)	18.4	13.8	9.3	16.6
Adder	18.4	13.8	9.3	16.6
Significand Memory	8.2	7.4	7.0	6.2
Header Memory	6.8	6.8	6.8	none
Long Accumulator	7.8	7.0	6.8	none
Shifter/Zero Detect	9.2	8.5	8.0	9.0
Shifter	8.9	8.2	7.8	8.8
Operand Selector	4.2	3.5	3.2	none
Exponent Add/Sub	4.4	4.4	4.4	4.0
Latches	2.0	2.0	2.0	2.0
Cycle Time	22.0	16.0	12.5	20.0

Table 4: Cycle Counts

Operation	real operands	interval operands
Addition/Subtraction	$2n+8$	$4n+12$
Multiplication	n^2+n+12	$2n^2+2n+22$
Short Multiplication	$2n+12$	$4n+20$
Square	$n^2/2+2n+12$	$n^2+4n+22$
Division	$3n^2+4n+20$	$6n^2+8n+38$
Square Root	$3n^2+6n+26$	$6n^2+12n+48$
Dot Product	$m(2n^2+12)+2n+20$	$m(4n^2+26)+4n+36$
Midpoint	not applicable	$2n+8$
Intersection	not applicable	$3n+12$

Table 5: Execution Times for Real Dot Product ($m = 16$) (sec)

Precision (bits)	64-bit VPIAC	32-bit VPIAC	16-bit VPIAC
64	5.41E-6	5.50E-6	9.15E-6
128	7.57E-6	1.17E-5	2.85E-5
256	1.61E-5	3.64E-5	1.05E-4
512	5.01E-5	1.35E-4	4.13E-4
1,024	1.86E-4	5.29E-4	not supported
2,048	7.27E-4	not supported	not supported

Table 6: Execution Times for Interval Dot Product ($m = 16$) (sec)

Precision (bits)	64-bit VPIAC	32-bit VPIAC	16-bit VPIAC
64	1.14E-5	1.15E-5	1.87E-5
128	1.57E-5	2.39E-5	5.73E-5
256	3.28E-5	7.33E-5	2.11E-4
512	1.01E-4	2.70E-4	8.26E-4
1,024	3.72E-4	1.06E-3	not supported
2,048	1.45E-3	not supported	not supported

Table 7: Execution Times for Real Polynomial Evaluation (sec)

Precision (bits)	64-bit VPIAC	32-bit VPIAC	16-bit VPIAC
64	1.06E-5	9.60E-6	1.20E-5
128	1.32E-5	1.54E-5	2.70E-5
256	2.11E-5	3.45E-5	8.10E-5
512	4.75E-5	1.04E-4	2.85E-4
1,024	1.43E-4	3.65E-4	not supported
2,048	5.02E-4	not supported	not supported

Table 8: Execution Times for Interval Polynomial Evaluation (sec)

Precision (bits)	64-bit VPIAC	32-bit VPIAC	16-bit VPIAC
64	1.85E-5	1.73E-5	2.25E-5
128	2.38E-5	2.88E-5	5.25E-5
256	3.96E-5	6.72E-5	1.61E-4
512	9.24E-5	2.05E-4	5.69E-4
1,024	2.82E-4	7.28E-4	not supported
2,048	1.00E-3	not supported	not supported

Table 9: Execution Times for Interval Newton Method (sec)

Precsion (bits)	64-bit VPIAC	32-bit VPIAC	16-bit VPIAC
64	9.63E-6	9.22E-6	1.25E-5
128	1.27E-5	1.59E-5	2.75E-5
256	2.12E-5	3.52E-5	7.70E-5
512	4.84E-5	9.86E-5	2.80E-4
1,024	1.35E-4	3.58E-4	not supported
2,048	4.92E-4	not supported	not supported

also be used to evaluate the accuracy of programs before running them on a general purpose processor, or can be used to select between various programs based on their accuracy for given inputs. Direct hardware support for variable-precision, interval arithmetic greatly improves the accuracy and reliability of the computation, and is much faster than existing software techniques for controlling numerical error.

References

[1] D. Ratz, "The Effects of the Arithmetic of Vector Computers on Basic Numerical Methods," in *Contributions to Computer Arithmetic and Self-Validating Numerical Methods* (C. Ullrich, ed.), pp. 499–514, J.C. Baltzer, 1990.

[2] G. Bohlender, "What Do We Need Beyond IEEE Arithmetic?," in *Computer Arithmetic and Self-Validating Numerical Methods* (C. Ullrich, ed.), pp. 1–32, Academic Press., 1990.

[3] B. I. Blum, "An Extended Arithmetic Package," *Communications of the ACM*, vol. 8, pp. 318–320, 1965.

[4] W. T. Wyatt, D. W. Lozier, and D. J. Orser, "A Portable Extended Precision Arithmetic Package and Library with FORTRAN Precompiler," *ACM Transactions on Mathematical Software*, vol. 2, pp. 209–231, 1976.

[5] R. P. Brent, "A FORTRAN Multiprecision Arithmetic Package," *ACM Transactions on Mathematical Software*, vol. 4, pp. 57–70, 1978.

[6] C. Ullrich, "Programming Languages for Enclosure Methods," in *Computer Arithmetic and Self-Validating Numerical Methods* (C. Ullrich, ed.), pp. 115–136, Academic Press, 1990.

[7] R. E. Moore, *Interval Analysis*. Prentice Hall, 1966.

[8] U. W. Kulisch and W. L. Miranker, *Computer Arithmetic in Theory and in Practice*. Academic Press, 1981.

[9] R. Hammer, M. Neaga, and D. Ratz, "Pascal-XSC New Concepts for Scientific Computation and Numerical Data Processing," in *Scientific Computing with Automatic Result Verification* (E. Adams and U. Kulisch, eds.), pp. 15–44, Academic Press, 1993.

[10] C. Lawo, "C-XSC New Concepts for Scientific Computation and Numerical Data Processing," in *Scientific Computing with Automatic Result Verification* (E. Adams and U. Kulisch, eds.), pp. 71–86, Academic Press, 1993.

[11] W. V. Walter, "Acrith-XSC A Fortran-like Language for Verified Scientific Computing," in *Scientific Computing with Automatic Result Verification* (E. Adams and U. Kulisch, eds.), pp. 45–70, Academic Press, 1993.

[12] J. S. Ely, "The VPI Software Package for Variable Precision Interval Arithmetic," *Interval Computations*, vol. 2, pp. 135–153, 1993.

[13] *IEEE Standard 754 for Binary Floating Point Arithmetic*. American National Standards Institute, 1985.

[14] A. Knofel, "Hardware Kernel for Scientific/Engineering Compuations," in *Scientific Computing with Automatic Result Verification* (E. Adams and U. Kulisch, eds.), pp. 549–570, Academic Press, 1993.

[15] E. K. Reuter *et al.*, "Some Experiments Using Interval Arithmetic," in *Proceedings of the 4th Symposium on Computer Arithmetic*, pp. 75–81, 1978.

[16] A. Knofel, "Fast Hardware Units for the Computation of Accurate Dot Products," in *Proceedings of the 10th Symposium on Computer Arithmetic*, pp. 70–75, 1991.

[17] K. Hafner, "Chips for High Precision Arithmetic," in *Computer Arithmetic and Self-Validating Numerical Methods* (C. Ullrich, ed.), pp. 33–54, Academic Press, 1990.

[18] J. Kernhof *et al.*, "A CMOS Floating-Point Processing Chip for Verified Exact Vector Arithmetic," in *ESSIRC 94*, 1994.

[19] W. M. Kahan and E. LeBlanc, "Anomalies in the IBM ACRITH Package," in *Proceedings of the 7th Symposium on Computer Arithmetic*, pp. 322–331, 1985.

[20] M. S. Cohen, T. E. Hull, and V. C. Hamarcher, "CADAC: A Controlled-Precision Decimal Arithmetic Unit," *IEEE Transactions on Computers*, vol. C-32, pp. 370–377, 1983.

[21] D. M. Chiarulli, W. G. Rudd, and D. A. Buell, "DRAFT: A Dynamically Reconfigurable Processor for Integer Arithmetic," in *Proceedings of the 7th Symposium on Computer Arithmetic*, pp. 309–318, 1985.

[22] T. M. Carter, "Cascade: Hardware for High/Variable Precision Arithmetic," in *Proceedings of the 9th Symposium on Computer Arithmetic*, pp. 184–191, 1989.

[23] M. J. Schulte and E. E. Swartzlander, "Software Interface and Hardware Design for Variable-Precision Interval Arithmetic," *Interval Computations (in press)*, 1995.

[24] *LSI Logic 1.0 Micron Cell-Based Products Databook*. LSI Logic Corporation, 1991.

[25] K. C. Bickerstaff, M. J. Schulte, and E. E. Swartzlander, "Reduced Area Multipliers," in *Proceedings of the 1993 International Conference on Application Specific Array Processors*, pp. 478–489, 1993.