Reliable Computing 2 (2) (1996), pp. 133–138

Software for high radix on-line arithmetic

Thomas Lynch and Michael J. Schulte

High radix on-line arithmetic provides an efficient method for performing variable-precision arithmetic. It can be implemented on conventional microprocessors using sequences of three operand instructions. This paper presents software support for high radix on-line arithmetic. This software includes emulation modules for high radix operations, and a precision analysis program for setting the input and intermediate variable tolerances necessary for guaranteed result accuracy over a specified domain.

Программное обеспечение для компьютерной арифметики оперативного доступа с большим основанием системы счисления

Т. Линч, М. Шульте

Компьютерная арифметика оперативного доступа с большим основанием системы счисления предоставляет эффективный метод для вычислений с переменной разрядностью. Ее можно реализовать на обычных микропроцессорах с помощью последовательностей инструкций с тремя операндами. В работе представлена программная поддержка для компьютерной арифметики оперативного доступа с большим основанием системы счисления. Программное обеспечение включает в себя модули эмуляции для операций с большим основанием системы счисления, а также программу анализа разрядности, служащую для задания допусков на исходные и промежуточные переменные, обеспечивающих гарантированную точность результата на заданной области.

1. Introduction

As presented in [9], high radix on-line arithmetic is an efficient method for performing variable-precision arithmetic. With high radix on-line arithmetic, a variable-precision floating point number x is represented as a string of $n_x + 1$ signed integers $(e_x, x_0, x_1, \dots, x_{n_x-1})$, where e_x is the exponent and x_i is the i-th significand digit. The value of x is

$$x = r^{e_x} \cdot \left(\sum_{i=0}^{n_x-1} x_i \cdot r^{-i} \right) \tag{1}$$

where r is the radix of the number system. Increasing n_x increases the precision of x.

High radix on-line arithmetic can be implemented as sequences of three operand instructions, where the operands are signed integers. Hence, it is well suited for being implemented on microprocessors. With on-line arithmetic, operations are performed serially, from most significant digit to least significant digit [3, 6, 12, 13]. This is possible since the redundant signed-digit representation limits carry propagation [2, 5, 11]. Thus, high radix on-line arithmetic can be digit pipelined, since instructions that use the results of previous calculations can begin execution before the less significant digits of their operands are available.

Even without additional hardware support, high radix on-line arithmetic can be implemented on conventional microprocessors. In this paper, we discuss freely available software for performing precision analysis and high-radix on-line addition, subtraction and multiplication. The paper presents a compiler-driven approach for generating code that uses high radix variable-precision floating point numbers. This is a static approach. Other methods are possible, such as those based on lazy evaluation [4].

2. Program parsing and precision analysis

This section describes a method for taking a program and determining the precisions of input, temporary, and output variables that are sufficient to produce outputs that are accurate to within a specified error tolerance. The analysis uses four consecutive steps: program parsing, intermediate code generation, range analysis, and precision analysis. The results from a successful precision analysis are used as parameters in the constructor calls of a high radix variable-precision floating point number class that is implemented in C++.

2.1. Program parsing

The parser is a conventional LEX/YACC based program that accepts numerical programs, such as the one given in Figure 1. The first line of the program gives the program name, followed by a list of input variables and a list of output variables. The range is specified for each input. For example, a is between 4 and 8. The precision of each output variable is also specified. For example, x has a precision of four digits. The accuracy goal defaults to one unit in the last place of the specified precision.

```
ECALC (a[4, 8] b[1, 3.99] c[-2000, -1000] d[1, 100]) (x:4 y:6) {
      x = (a - b)*(c + a);
      y = x - d;
}
```

Figure 1. Initial numerical program

2.2. Intermediate code generation

Figure 2 shows how ECALC is parsed into three operand instructions. Temporary variables are added where needed. The information from the parsed program is stored in two tables; one that has information for each of the variables, and another which has information for each instruction. The variable table entries contain the variable's name, type, range, and precision. The variable's type is either input, temporary, or output. Initially, all non-input variables have a range of $[0, 0]$ and all non-output variables have a precision of 0. The instruction table entries contain the instruction being performed, and indices into the variable table for the source and destination operands, as shown in Figure 3.

```
ECALC (a[4, 8] b[1, 3.99] c[-2000, -1000] d[1, 100]) (x:4 y:6) {
      t1 = a - b;
      t2 = c + a;
      x = t1*t2;
      y = x - d;
}
```

Figure 2. Parsed numerical program

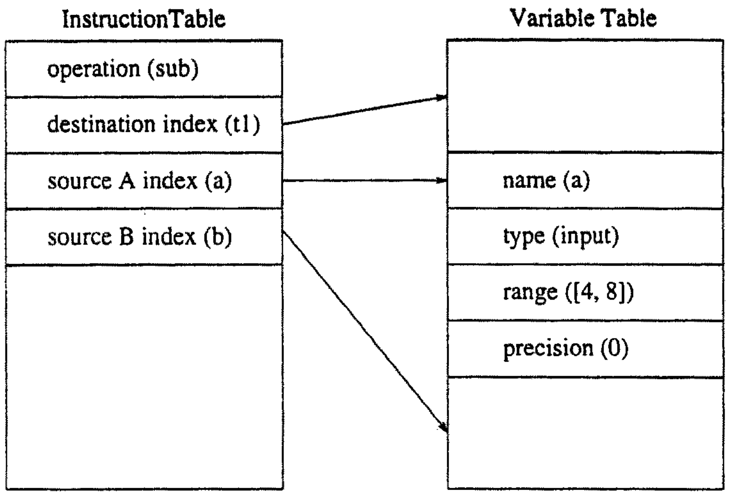

Figure 3. Instruction and variable tables

2.3. Range analysis

The range analysis program uses the ranges of the input variables to determine the ranges of all temporary and output variables. For each instruction, interval arithmetic is used to determine the range of the destination operand based on the ranges of the source operands [1, 7, 10]. Instructions are analyzed in program order (i.e., from the first instruction to the last instruction) to ensure that each operand's range is determined, before it is used as a source operand. The PROFIL/BIAS software package [8] is used to perform interval arithmetic during range analysis.

The range analysis for ECALC is shown in Figure 4. If an interval that contains zero is generated during the range analysis, an error is issued. An interval that contains zero indicates that the accuracy in one of the destination operands cannot be guaranteed, regardless of the precision of its source operands. Intervals that cross zero can occur due to undetected correlations between variables.

2.4. Precision analysis

The precision analysis uses the range information along with the variable tables to determine the precision required in input, temporary, and output variables to obtain the specified output accuracy constraint (currently this ignores approximation error). For each instruction, the

`t1 = a - b`	$[0.01, 7] = [4, 8] - [1, 3.99]$
`t2 = c + a`	$[-1996, -992] = [-2000, -1000] + [4, 8]$
`x = t1 * t2`	$[-13972, -9.92] = [0.01, 7] * [-1996, -992]$
`y = x - d`	$[-14072, -10.92] = [-13972, -9.92] - [1, 100]$

Figure 4. Range analysis

precisions of the source operands are determined from the precision of the destination operand, based on formulas that were derived from the *range expansion* formulas given in [9]. The precision equations for addition, subtraction, multiplication and division are:

$$p_x = p_{x\pm y} + \log_r\left(\left|\frac{x}{x\pm y}\right|\right) + 1, \tag{2}$$

$$p_y = p_{x\pm y} + \log_r\left(\left|\frac{y}{x\pm y}\right|\right) + 1, \tag{3}$$

$$p_x = p_y = p_{x\cdot y} + 1, \tag{4}$$

$$p_x = p_y = p_{x/y} + 1. \tag{5}$$

Here, x and y are the source operands, and r is the radix of the number system. For addition and subtraction, interval arithmetic is used to determine the maximum values of (2) and (3). For multiplication and division, the required precision of the source operands is equal to the precision of the destination operand plus one.

The precision analysis for ECALC is shown in Figure 5, for $r = 10$. The instructions are analyzed from last to first to ensure that the precision of the destination operand is set before it is used. If a variable is used more than once as a source operand, it could be assigned one of multiple precisions. In this case, the largest precision is used. For example, a is assigned a precision of 15.1 rather than 10.1, since 15.1 is larger.

`y = x - d`	$p_x = p_y + \log_r(\| \frac{x}{x-d} \|) + 1 \approx 10.2$
	$p_d = p_y + \log_r(\| \frac{d}{x-d} \|) + 1 \approx 8.0$
`x = t1 * t2`	$p_{t1} = p_x + 1 \approx 11.2$
	$p_{t2} = p_x + 1 \approx 11.2$
`t2 = c + a`	$p_c = p_{t2} + \log_r(\| \frac{c}{c+a} \|) + 1 \approx 12.5$
	$p_a = p_{t2} + \log_r(\| \frac{a}{c+a} \|) + 1 \approx 10.1$
`t1 = a - b`	$p_a = p_{t1} + \log_r(\| \frac{a}{a-b} \|) + 1 \approx 15.1$
	$p_b = p_{t1} + \log_r(\| \frac{b}{a-b} \|) + 1 \approx 14.8$

Figure 5. Precision analysis

3. High radix variable-precision number class

The high radix variable-precision number class is based on the high radix on-line classes `add`, `mul`, and `word`. The `add` and `mul` classes implement the high radix on-line adder and

```
main {
   vp a(16), b(15), c(13), d(8), t1(12), t2(12), x(11), y(6);
   cin >> a; cin >> b; cin >> c; cin >> d;
   sub_vp(a, b, t1);
   add_vp(a, c, t2);
   mul_vp(t1, t2, x);
   sub_vp(x, d, y);
   cout << x; cout << y;
}
```

Figure 6. Variable-precision program

multiplier, respectively, while the `word` class provides support for the internal state of the multiplier. These operator classes are described in more detail in [9].

The vp class provides software support for high radix variable-precision floating point numbers. Figure 6 shows how the vp class is used to implement the ECALC program. The first line of the program initializes variables to the precisions produced by the precision analysis program. Here, the precisions computed in Figure 5 are rounded upward to the next larger integer. In the next line, the values of the input variables are read from standard input. After this, the values of temporary and output values are computed. Last of all, the output variables are written to standard output.

4. Conclusions

High radix on-line arithmetic provides an efficient method for performing variable-precision arithmetic. Software support has been developed to facilitate the use of high radix on-line arithmetic. This software provides a method for performing precision analysis and using a high radix variable-precision floating point number class. The software described in this paper and further information on high radix on-line arithmetic can be obtained from the Internet site:

`http://devil.ece.utexas.edu/~lynch`

References

[1] Alefeld, G. and Herzberger, J. *Introduction to interval computations.* Academic Press, 1983.

[2] Avizienis, A. *Signed-digit number representations for fast parallel arithmetic.* IRE Transactions on Electronic Computers **10** (1961), pp. 389–400.

[3] Bajard, J. C., Guyot, A., Muller, J.–M., and Skaf, A. *Design of a VLSI circuit for on-line evaluation of several elementary functions using their Taylor expansions.* In: "Proceedings of the 1993 International Conference on Application Specific Array Processors", 1993, pp. 526–535.

[4] Benouamer, M. O., Jailon, P., Michelucci, D., and Moreau, J.–M. *A lazy exact arithmetic.* In: "Proceedings of the 11th Symposium on Computer Arithmetic", IEEE Computer Society Press, 1993, pp. 242–249.

[5] Ercegovac, M. D. and Lang, T. *Fast multiplication without carry-propagate addition.* IEEE Transactions on Computers **C–39** (1990), pp. 1385–1390.

[6] Ercegovac, M. D. and Lang, T. *On-line arithmetic: a design methodology and applications to digital signal processing.* IEEE Press, 1988, pp. 252–263.

[7] Herzberger, J. *Basic definitions and properties of interval arithmetic.* In: Herzberger, J. (ed.) "Topics in Validated Computations. Proceedings of IMACS–GAMM International Workshop on Validated Numerics", North Holland, 1994, pp. 1–6.

[8] Knüppel, O. *PROFIL/BIAS—a fast interval library.* Computing **53** (1994), pp. 277–288.

[9] Lynch, T. and Schulte, M. J. *A high radix on-line arithmetic for credible and accurate computing.* Journal of Universal Computer Science **1** (7) (1995), pp. 435–449.

[10] Moore, R. E. *Interval analysis.* Prentice Hall, 1966.

[11] Phatak, D. S. and Koren, I. *Hybrid signed-digit number systems: a unified framework for redundant number representations with bounded carry propagation chains.* IEEE Transactions on Computers **C–43** (8) (1994), pp. 880–891.

[12] Trivedi, K. S. and Ercegovac, M. D. *On-line algorithms for division and multiplication.* IEEE Transactions on Computers **C–26** (7) (1977), pp. 681–687.

[13] Watanuki, O. and Ercegovac, M. D. *Error analysis of certain floating-point on-line algorithms.* IEEE Transactions on Computers **C–32** (4) (1983), pp. 352–358.

Received: November 1, 1995
Revised version: December 15, 1995

T. Lynch
Advanced Architecture Development
Advanced Micro Devices
MS 615, 5204 East Ben White Blvd
Austin, TX 78741
USA
E-mail: `Tom.Lynch@amd.com`

M. J. Schulte
Department of Electrical and Computer Engineering
University of Texas at Austin
Austin, TX 78712
USA
E-mail: `schulte@pine.ece.utexas.edu`