MCALIB - Measuring Sensitivity to Rounding Error with Monte Carlo Programming

Journal:	*Transactions on Programming Languages and Systems*
Manuscript ID:	TOPLAS-00035-2013.R2
Manuscript Type:	Paper
Date Submitted by the Author:	n/a
Complete List of Authors:	Frechtling, Michael; The University of Sydney, Electrical & Information Engineering Leong, Philip; The University of Sydney, Electrical & Information Engineering
Computing Classification Systems:	Programming Languages, Software Engineering, Numerical Analysis, Probability and Statistics

MCALIB - Measuring Sensitivity to Rounding Error with Monte Carlo Programming

MICHAEL FRECHTLING, The University of Sydney
PHILIP H.W. LEONG, The University of Sydney

Run-time analysis provides an effective method for measuring the sensitivity of programs to rounding errors. To date, implementations have required significant changes to source code, detracting from their widespread application. In this work we present an open source system that automates the quantitative analysis of floating point rounding errors, through the use of C-based source-to-source compilation and a Monte Carlo Arithmetic library. We demonstrate its application to the comparison of algorithms, detection of catastrophic cancellation, and determination of whether single-precision floating point provides sufficient accuracy for a given application. Methods for obtaining quantifiable measurements of sensitivity to rounding error are also detailed.

Categories and Subject Descriptors: D.3.4 [**Programming Languages**]: Processors — Compilers, Code Generation; D.2.4 [**Software Engineering**]: Software/Program Verification — Statistical Methods, Validation; G.1.0 [**Numerical Analysis**]: General — Computer Arithmetic, Error Analysis; G.3 [**Probability and Statistics**]: Probabilistic Algorithms

General Terms: Design, Reliability, Verification

Additional Key Words and Phrases: Floating Point Arithmetic, Dynamic Error Analysis, Monte Carlo Arithmetic

ACM Reference Format:
Michael Frechtling and Philip H. W. Leong, 2013. MCALIB - Measuring Sensitivity to Rounding Error with Monte Carlo Programming. *ACM Trans. Program. Lang. Syst.* X, X, Article XX (June 2014), 25 pages.
DOI:http://dx.doi.org/10.1145/0000000.0000000

1. INTRODUCTION

IEEE-754 [Floating-point Working Group 2008] has long been the standard for computing using Floating Point (FP) numbers but, as a finite precision arithmetic system, it is capable of producing anomalous results. Rounding errors can significantly reduce the accuracy of a computation, and result in errors many times larger than expected [Goldberg 1990]. FP arithmetic is now used for systems where accuracy is key, such as scientific Software (SW) and critical systems. In order to properly implement and verify numerical SW, techniques to determine the effects of such errors are required [Howden 1980; Monniaux 2008].

Monte Carlo Arithmetic (MCA) [Parker 1997] tracks rounding errors at run-time by applying randomization to input and output operands forcing the results of FP operations to behave like random variables. This turns an execution into trials of a Monte Carlo simulation allowing statistics on the effects of rounding error to

Author's addresses: M. Frechtling and P.H.W Leong, Computer Engineering Laboratory, Room 840, Building J03, University of Sydney, Camperdown, NSW 2009, Australia.

DOI:http://dx.doi.org/10.1145/0000000.0000000

be obtained over a number of executions. Statistical measurements are then used to analyse the results, sensitivity to rounding error is suspected if a high level of variance is observed between trials. As an example, consider the subtraction 11111113 - 11111111 in decimal arithmetic using 8 significant digits. In standard arithmetic, the answer is 2.0000000 and 7 digits of significance are lost due to cancellation. With MCA, the answer for a particular trial is 2.xxxxxxx where the x's are random digits. Only the most significant digit will be unchanged over a number of trials and so a large standard deviation will result.

Despite the advantages offered by MCA and similar techniques, tools for rounding error analysis are not in common usage. It is believed that one of the major barriers is that source code needs to be modified so that custom libraries are called to execute the arithmetic operations. In this work, the use of source to source compilation, supported by mixed precision libraries, is advocated. The approach allows for the implementation of a general purpose FP analysis tool that can be applied to arbitrary programs without significant changes to the source code, a technique that we refer to as Monte Carlo Programming (MCP). The implementation provides opportunities for wider adoption of runtime error analysis, and allows developers to test both the accuracy of algorithms and the suitability of different FP formats for a particular implementation. Although our tool is designed to be used with MCA, the same approach could be used in conjunction with other rounding analysis techniques. The main contributions of this work are as follows:

— An open source MCP implementation capable of performing variable precision MCA and supporting both single and double precision FP formats.
— A method for including random variables in programs and performing statistical analysis of the resulting output.
— A method for inspecting the accuracy of floating point values in existing programs.
— A method for imposing new semantics on arithmetic primitives in existing programs.

MCP can be used for the simplified implementation of several data analysis schemes, such as sensitivity analysis to measure the effect of uncertainty in input data or arithmetic operations. The effect of missing data, dirty data and inexact data can also be measured. An open source implementation of the Monte Carlo Arithmetic LIBrary (MCALIB), including CIL libraries and documentation, is available via github from https://github.com/mfrechtling/mcalib.git. The remainder of this paper is organized as follows. In Section 2, background concerning FP systems and error analysis methods is given. Section 3 gives an overview of MCA. The implementation of the library is detailed in Section 4. Methods for interpreting the results of MCA analysis are provided in Section 5. Section 6 describes test cases and methods. Results are presented in Section 7, and finally, conclusions are drawn in Section 8.

2. BACKGROUND

2.1. IEEE-754 Floating Point

The binary IEEE-754 [Floating-point Working Group 2008] FP number system $\mathbb{F}(\beta, p, e_{min}, e_{max})$ is a subset of real numbers with elements of the form:

$$x = (-1)^s m\beta^e \tag{1}$$

The system is characterized by the radix β, which is assumed to be 2 in this paper, precision p, the exponent values $e_{min} \leq e \leq e_{max}$, the sign bit $s \in \{0, 1\}$ and the significand $m \in [0, \beta)$. Normalized values are most commonly used and are represented as a non-zero $x \in \mathbb{F}$ with $|m| \in [1, \beta)$ and $e_{min} < e < e_{max}$. De-normalized numbers are

Table I. IEEE-754 Format Types

Type	Length	Precision (p)	e_{max}	e_{min}
Single	32	24	-127	128
Double	64	53	-1023	1024

Note: Format specification for single and double precision types

also supported and represent values of smaller magnitude than normalized numbers with $m \in [0, 1)$ and exponent $e = e_{min}$. Several other classes of numbers are available with special formats. Zeroes have a significand $m = 0$ and $e = 0$, both $+0$ and -0 can be represented using the sign bit s. Infinity and Not a Number (NaN) are both represented with $e = e_{max}$. The system developed for this paper supports the single and double precision IEEE-754 formats detailed in Table I.

An important concept within FP arithmetic is the distinction between **exact** and **inexact** values. FP arithmetic systems are implemented as finite precision rounded arithmetic schemes and as such are not able to represent every value within the infinite set of real numbers, $\mathbb{R}$. Instead the set of FP numbers, $\mathbb{F}$, is a finite subset of real numbers $\mathbb{F} \subset \mathbb{R}$. Real numbers that are representable in FP format are referred to as exact values, while inexact values refer to real numbers that cannot be represented and are instead rounded to the nearest exact value. Rounding of inexact values to their nearest FP approximation leads to the concept of rounding error, a subject that has been studied in several publications [Goldberg 1990; Higham 1996; Wilkinson 1994; Parker 1997] and states that the the rounded approximation, ($\hat{x} \in \mathbb{F}$), of a real number, ($x \in \mathbb{R}$), is modelled as follows:

$$\hat{x} = \mathbb{F}(x) \tag{2}$$

$$= x(1 + \delta) \tag{3}$$

The value $\delta = |\frac{x-\hat{x}}{x}|$ is the relative error of the rounded approximation and is limited by the machine epsilon of the FP system, $\delta \leq \epsilon$, where $\epsilon = 2^{-p}$ [Goldberg 1990]. In the general case errors are due to not only finite precision limitations but a number of factors including errors of measurement or estimation, quantization error, or errors propagated from earlier parts of a computation. Although the IEEE-754 standard is used in all types of applications mostly without issue, unexpected results cannot be avoided in certain situations due to the effects of finite precision and rounding error.

The primary error concerning the work in this paper is **catastrophic cancellation**. Cancellation is a phenomenon that will occur when two nearly equal values are subtracted leaving a large number of leading zeroes after the radix point. As FP utilizes normalized values, these leading zeros must be removed by shifting the result to the left and adjusting the exponent accordingly. The phenomena of catastrophic cancellation is one of the leading causes of loss of significance in FP arithmetic. As an example consider the solution to the equation $x^2 + 444x + 1 = 0$ using the quadratic formula $r = (-b \pm \sqrt{b^2 - 4ac})/(2a)$. IEEE-754 single precision format uses a 24-bit binary significand giving it a precision value $p = 24$, equivalent to $\log_{10}(2^{24}) \approx 7.225$ decimal digits. In many instances the answer will be accurate to 7 decimal places. Unfortunately, in this example, the exact result is $r_1 = -222 \pm \sqrt{49283} = -0.00225226368$ whereas IEEE-754 arithmetic gives $r_1 = 0.000000000$. This has a 100% relative error. A more formal model of catastrophic cancellation is given in [Higham 1996]. Consider the equation $\hat{x} = \hat{a} - \hat{b}$

where $\hat{a} = a(1+\delta_a)$ and $\hat{b} = b(1+\delta_b)$, in this situation the relative error of $\hat{x}$ is given by:

$$\left|\frac{x-\hat{x}}{x}\right| \le \left|\frac{(a-b)-(\hat{a}-\hat{b})}{a-b}\right| \tag{4}$$

$$\le \left|\frac{[a-a(1+\delta_a)]-[b-b(1+\delta_b)]}{a-b}\right| \tag{5}$$

$$\le \left|\frac{-a\delta_a+b\delta_b}{a-b}\right| \tag{6}$$

$$\le \max(|\delta_a|,|\delta_b|)\frac{|a|+|b|}{|a-b|} \tag{7}$$

This shows that the relative error in $\hat{x}$ is large when $|a-b| \ll |a|+|b|$

2.2. Error Analysis

The design of numerically stable algorithms must ensure that the issues reviewed do not adversely contribute to the accuracy of the solution. In the design of numerical libraries, analysts use techniques such as forward and backward error analysis to quantify the propagation of errors and understand their effect on the stability and accuracy of the algorithms [Wilkinson 1994]. Unfortunately, these techniques cannot be applied to arbitrary programs, require manual analysis and considerable expertise, and do not scale beyond small subroutines.

One of the primary questions in the study of numerical analysis is not how to develop the best techniques or systems, but how to get the best techniques and systems into the hands of the developers working with real world problems. Aside from the practical considerations of ease of understanding, implementation and use, there exists the questions of what developers need or want in a numeric analysis tool. One of the best assessments of numeric analysis techniques and the current state of the art is presented in [Kahan 2006]. Kahan notes a significant problem in encouraging the adoption of numeric analysis techniques; the average developer is not interested in these techniques until **after** something has gone wrong, at which point analysis is often required for *"...an assignment of blame and the task of relieving the distress, if possible."* It is for this reason, among others, that developers often search for what Kahan calls **mindless** assessments of round-off error, essentially systems that allow for a **fire and forget** approach rather than an in depth analysis of the inner numeric workings of a piece of software. When viewed through this lens, the question of how to design systems that will be eagerly adopted by developers becomes a philosophical difference between two approaches to numeric analysis;

— How many significant digits are available in the results, or, how accurate is my program?
— What is the worst case bound on the absolute/relative error, or, how badly could my program fail?

What Kahan refers to as **mindless** assessments, often focus on the second approach, as this is the question that developers want answered after something has gone wrong (in which case the question often becomes how badly **did** my program fail?). The authors advocate the first approach for the adoption of error analysis techniques as part of the Software Development Life Cycle (SDLC). In order to achieve this we have attempted to design an analysis method that will satisfy the needs of both developers - seeking high level, automated analysis of existing programs, and numeric analysts - seeking extendible, in depth analysis of floating point formats and operators. It is this vein of thinking that MCALIB and MCP has been developed. The remainder of this section provides a more in depth overview of existing error analysis techniques and the sate of the art.

Error analysis methods for software are divided into two types, **dynamic**, which analyses the results of program execution for a specific input set, and **static**, which is performed without the need for execution. While these analysis types are intended to be complimentary and may be used to validate each others results, key differences exist. Dynamic analysis provides a higher level of flexibility and can even be performed without access to the source code in the case of automated tools, but more often requires significant modifications to the source code, and due to its data dependency must be performed using an adequate set of inputs to produce meaningful results. Conversely, by limiting analysis to individual executions of a system, dynamic analysis methods are efficient as system properties need only be checked along a single execution path. Furthermore, testing is conducted using actual operations performed by the system rather than mathematical abstractions allowing for more precise analysis. This also avoids compatibility issues being introduced from differences in arithmetic format, compilers or system architecture [Monniaux 2008]. Static analysis avoids the data dependency issue by abstracting the possible states and operators of tested software, leading to a mathematical formulation that allows all possible states of a system to be tested. An overly rigorous definition will result in a complex analysis that does not scale to large systems. Automated tools for static analysis provide the ability to pinpoint the exact locations of errors in software, often at an earlier stage in the SDLC, however, automated tools only support certain languages and static analysis becomes time consuming when performed manually.

Static analysis techniques will typically use formal methods, whereby software is analysed using mathematical techniques based on formal semantics of the programming language used. These techniques include denotational semantics, axiomatic semantics, operational semantics and abstract interpretation. Methods used for static analysis include three basic types. Model or property checking requires the creation of formal models for both the system and its specification. Model checking may then be used to determine if the system model meets all requirements of the specification model [Lam 2005]. In order to perform model checking algorithmically, it is limited to finite state systems and is typically used for the analysis of Hardware (HW) systems as the undecidability of SW limits it's effectiveness. Due to this limitation model checking is often used for analysis of SW and HW systems modelled as a Finite State Machine (FSM). Data flow analysis is a technique for generating possible sets of values for nodes in a program's Control Flow Graph (CFG), this is typically accomplished using an iterative approach that determines values for the in-states and out-states at each node in the CFG until the complete system stabilizes [Kildall 1973; Cooper et al. 2002]. Finally abstract interpretation creates partial abstractions of operations and variables in order to create a computable semantic interpretation. It is viewed as a partial execution technique for static analysis [Cousot and Cousot 1979; 1977]. The semantics created for abstract interpretation are defined as monotonic functions that relate elements of the system across ordered sets.

Systems available for static analysis of rounding errors include Fluctuat [Goubault and Putot 2006], Astree [Blanchet et al. 2002] and Polyspace [Deutsch 2003]. Fluctuat performs abstract interpretation using an abstract domain based on affine arithmetic for analysis of FP error. This tool is now being used by Airbus to automate accuracy analysis of control software [Delmas et al. 2009]. Astree is based on interval arithmetic methods and is designed for safety critical analysis, including FP error analysis [Blanchet et al. 2002]. This software is also being used by Airbus for automated software analysis [Delmas and Souyris 2007]. Polyspace is used to locate potential run-time errors including arithmetic overflow, divide by zero and buffer overrun, the software is now supplied by MathWorks and is used in several industrial applications.

Several systems have been developed for performing dynamic analysis of FP SW. Interval Arithmetic (IA) represents a value x by an interval $[x_{lo}, x_{hi}]$. Intervals are propagated through the calculation e.g. $[a_{lo}, a_{hi}] - [b_{lo}, b_{hi}] = [a_{lo} - b_{hi}, a_{hi} - b_{lo}]$. The use of an interval as opposed to a single value addresses issues resulting from inexact values and round-off error. Rather than attempting to find the nearest approximation of an inexact value, an interval consisting of two exact values is found that can be said to contain the inexact result. While IA can be used to track inexact values and rounding errors during computation it often produces overly pessimistic error bounds [Kahan 2006]. This due to the assumption that input arguments may vary independently over given intervals, an issue known as the dependency problem [Krämer 2007]. An alternative to IA is Affine Arithmetic (AA), an interval analysis method that maintains first order correlations between quantities addressing the dependency problem and producing tighter error bounds [de Figueiredo and Stolfi 2004]. The Contrôle et Estemation STochastique des Arrondis Calculs (CESTAC) technique [Vignes 1996] is an implementation of the probabilistic approach similar to MCA. Using CESTAC an execution is repeated N times with the rounding method of FP operations randomized by rounding the result up or down with 50% probability. Using this method the least significant bit of the result significand is perturbed at each arithmetic stage creating a set of N results R_N. As in MCA, statistical analysis of the result set can be used to determine the accuracy of the algorithm used. While similar to MCA, it has been noted in both [Parker 1997] and [Kahan 1996] that several issues exist with the CESTAC method such as the assumption of the normal distribution of results, and the assumption that 2-3 samples is sufficient. Parker provides a overview of these issues and the way in which they are avoided with MCA [Parker 1997].

Several SW based implementations of these methods have been published including Control of Accuracy and Debugging for Numerical Applications (CADNA) [Jézéquel and Chesneaux 2008; Asserrhine et al. 1995], an implementation of Discrete Stochastic Arithmetic (DSA) using the CESTAC method. Several SW libraries for IA are available including eXtensions for Scientific Computation (XSC), Gaol, a C++ template class available as part of the Boost library [Brönnimann et al. 2006; Goualard 2006; Klatte and Corliss 1993] and the Multiple Precision Floating-point Interval (MPFI) library, an implementation of IA that also uses the Multiple Precision Floating point Reliably (MPFR) library for mixed precision implementation. Sun micro-systems have also provided support for IA as part of their C/C++ compiler library [Walster and Chiriaev 2000]. IA has also been implemented as part of Gappa [Daumas and Melquiond 2010], a formal verification tool for fixed and FP arithmetic. Gappa utilizes forward error analysis in addition to IA and requires a bounded input in order to perform its analysis. Using this tool bounds on the outputs of an algorithms are determined in addition to proofs on these bounds that may be checked via a proof assistant [Muller et al. 2009]. In order to maintain reasonable performance, a limited number of HW implementations of IA [Amaricai et al. 2009; Schulte and Swartzlander 2000; Stine and Schulte 1998] and CESTAC [Chotin and Mehrez 2002] can be found in the literature. A SW implementation of MCA has been published by Parker [Parker 2003] along with a set of test cases, however, this implementation cannot be applied to existing source without significant modifications. A Field Programmable Gate Array (FPGA) based implementation of MCA addition and multiplication with an area penalty of less than 22% over IEEE-754 was published by Yeung et. al. [Yeung et al. 2011]. Another FPGA based implementation of MCA has also been published by the authors [Frechtling and Leong 2013]. In this case the work was aimed at increased performance of MCA operations and simplifying the design of the MCA FPU by using existing FP operations to perform MCA, however, as this work involved the development of custom HW units it required significant modifications to existing source code in order to convert existing projects to use the MCA operators.

A separate class of analysis techniques have also been developed for bit width optimisation of arithmetic operators. While primarily aimed at fixed point implementations for Digital Signal Processing (DSP) and FPGA systems, most are applicable to both fixed and FP formats. The Multiple Word Length Paradigm (MWLP) [Constantinides et al. 2001] is an analysis technique that uses perturbation and scaling analysis for fixed point arithmetic to perform error constrained word length optimization. The system uses user defined error constraints on Signal to Noise Ratio (SNR) in order to optimize FPGA based DSP systems for area use, speed or power consumption. This system has been implemented for linear [Constantinides et al. 2003; 2002] and non-linear [Constantinides 2006] DSP systems and is the basis for Right-Size, a word-length optimization system for adaptive filters [Constantinides 2003]. Bit width optimization methods have also been developed using static analysis techniques including AA and Adaptive Simulated Annealing (ASA) [Lee et al. 2006]. These are designed to use range and precision analysis of fixed point implementations in order to guarantee the absolute error bounds of the system and have been implemented in the tool MiniBit [Lee et al. 2005]. This type of analysis has also been expanded to the analysis of FP applications using Automatic Differentiation [Gaffar et al. 2002]. Mixed analysis methods for both fixed and FP systems have also been developed using mixed precision analysis for optimizing word lengths for speed, power consumption and area use in FPGA systems [Gaffar et al. 2002]. Mixed analysis tools are also available in the form of MiniBit+ [Osborne et al. 2007], and the BitSize tool [Gaffar et al. 2004]. Finally FP analysis systems have been developed using profiling techniques based on tools such as Valgrind. Using these tools FPGA based arithmetic systems for DSP implementation may be optimized for speed, power consumption and area use. They perform mixed precision analysis of FP operations in order to identify operations that may be optimized by reducing the precision of the FP operations or replacing FP operators with fixed point or dual fixed point operators [Brown et al. 2007]. This type of analysis has implemented as part of the FloatWatch tool [Brown et al. 2008].

3. MONTE CARLO ARITHMETIC

MCA is used to track information lost using finite precision arithmetic by modelling inexactness using the application of random perturbations. If x is a non-zero FP value of the form given in Equation 1 the *inexact* function is defined as [Parker 1997]:

$$\begin{aligned} \text{inexact}(x,t,\xi) &= x + 2^{e_x - t}\xi && (8) \\ &= (-1)^{s_x}(m_x + 2^{-t}\xi)2^{e_x} && (9) \end{aligned}$$

where $x \in \mathbb{R}$, $x \neq 0$, t is a positive integer representing the virtual precision, ξ is a uniformly distributed random variable in the range $(-\frac{1}{2}, \frac{1}{2})$, ($\xi \in U(-\frac{1}{2}, \frac{1}{2})$) and m_x, e_x are the significand and exponent of x. It is assumed that $0 < t \leq p$. An operation

$\circ \in \{+, -, \times, \div\}$ is implemented as:

$$x \circ y = \text{round}(\text{inexact}(\text{inexact}(x) \circ \text{inexact}(y))) \quad (10)$$

Equation 10 results in the inexact() function being applied to both operands and the final result. Adjustments to the operands are referred to as *precision bounding* and are used to detect catastrophic cancellation, while adjustments to the final result are referred to as *random rounding* and are used to detect rounding error [Parker 1997]. The system developed for this paper performs both precision bounding and random rounding by first applying a random perturbation to each input operand, performing the required operation, then applying a random perturbation to the result. The inexact() function as implemented for this paper applies random perturbations using *uniform absolute* random values. Values for ξ are uniformly distributed over $(-\frac{1}{2}, \frac{1}{2})$ with mean 0 and standard deviation $\frac{1}{\sqrt{12}}$. If x is a random variable, then $\text{inexact}(x)$ has a mean $E[x]$, (expected value of x), and a standard deviation of $\sqrt{S[x]^2 + 2^{2(e-t)}/12}$, (where $S[x]$ is the standard deviation of the input x) [Parker 1997].

The virtual precision value t is used to control the level of random perturbation applied during MCA. This, in turn, controls the accuracy of the Monte Carlo operations. A large t value will result in a smaller exponent value for the operand perturbation, increasing the accuracy of the operation. Similarly, a smaller t decreases the accuracy. In practice, t is used to determine the minimum precision required to perform a specific operation accurately. The precision is set to its lowest value then repeated computations are performed as the precision is increased. When the required number of significant figures of the result have stabilized (i.e the required accuracy has been achieved) the minimum FP precision required to perform the operation accurately using standard IEEE-754 FP operators is determined. The implementation developed for this paper thus performs *variable precision* MCA, and the value of t used by the MCA library can be modified at any time during execution.

4. MCALIB IMPLEMENTATION

4.1. Source to Source Compilation

Source to source compilation provides an effective tool for automated code transformations [Foster and Taylor 1994], and when paired with error analysis techniques allows for the implementation of automated SW verification [Nguyen and Irigoin 2005; Irigoin et al. 1991]. The C Intermediate Language (CIL) [Necula et al. 2002] is a high level language representation including a set of tools for analysis and source to source compilation of C programs. The CIL compiler **cilly** is implemented as a Perl script that performs translations to C code as defined in a set of OCaml modules provided as part of the CIL library. For the purposes of MCALIB CIL has been used for transforming C FP operations into calls to the MCALIB library. This has been done by first lowering the source code to a single statement assignment form, then converting FP operations to use MCALIB library functions. As an example the following single precision multiplication operation:

```
a = b * c;
```

would be redefined to the following function call:

```
a = _floatmul(b, c);
```

where `float _floatmul(float a, float b)` is the MCALIB function for handling single precision MCA multiplication. This process will result in all supported FP operations being replaced with function calls to the MCALIB library. It is important to note that

ALGORITHM 1: MCA Binary Operation

```
**Input**: Precision $p$ FP operands $x_f$ and $y_f$
**Output**: Precision $p$ FP result $r_f$
$x$ = extend($x_f$, $p + t$);
$y$ = extend($y_f$, $p + t$);
$r$ = extend(0.0, $p + t$);
$x$ = inexact($x$);
$y$ = inexact($y$);
$r$ = mpfr_op($x$, $y$);
$r$ = inexact($r$);
$r_f$ = round($r$, $p$);
return $r_f$
```

ALGORITHM 2: MCA Inexact Operation

```
**Input**: Precision $p + t$ MPFR_T variable $x$
**Output**: Precision $p + t$ MPFR_T variable $x$ (w. random perturbation applied)
if $x == 0$ then
    return $x$;
else
    $\xi_f$ = (rand()/RAND_MAX) - 0.5;
    $\xi$ = extend($\xi_f$, $p + t$);
    $\xi$ = mpfr_mul(pow(2, $e_x$ - ($t$ - 1)), $\xi$);
    $x$ = mpfr_add($x$, $\xi$);
    return $x$;
end
```

although operations are done in a higher precision, the storage requirements of the FP variables remain unchanged. This avoids portability issues associated with pointers and dynamic memory allocation.

4.2. Library Implementation using MPFR

MCA has been implemented within MCALIB as a set of library functions for arithmetic and comparison operations. As stated in Section 3 the primary difficulty with implementing MCA is the need to extend the precision of the FP format being tested in order to simulate infinite precision. The precision level must include p machine bits and t virtual bits, a total precision requirement of $W = p+t$, where W is the working precision of the MCA operation. The MCALIB library also implements variable precision MCA, allowing the virtual precision to vary between $1 \leq t \leq p$ at runtime. To achieve this functionality the mixed precision library MPFR [Fousse et al. 2007] is used for mixed precision arithmetic in MCALIB.

For MCA functions, FP values are converted to `mpfr_t` type variables. The `mpfr_t` type is a struct containing an arbitrary precision significand and a fixed precision exponent. The precision of the significand of any MPFR variable may be set independently at runtime to any value between `MPFR_PREC_MIN` and `MPFR_PREC_MAX`, i.e. 2 and 256 respectively. For the purposes of MCALIB, the maximum precision required is $W_{max} = p + t_{max}$ which evaluates to 106 when using double precision operators. Rounding in MPFR adheres to the C implementation of the IEEE-754 standard and the default rounding mode **round to nearest even** is used for MCALIB.

The function for implementing MCA as per Equation 10 is shown in Algorithm 1. The

Table II. MCALIB Control Symbols & Functions

Symbol	Symbol Value	Function
MCALIB_IEEE	0	Disable MCA
MCALIB_MCA	1	Enable MCA, (precision bounding and random rounding).
MCALIB_PB	2	Enable precision bounding only.
MCALIB_RR	3	Enable random rounding only.

Note: Name, values and function of MCALIB control symbols for parameter MCALIB_OP_TYPE

FP operands are first converted to `mpfr_t` with precision W, and the result variable is initialized with the same precision. The random perturbation ξ is applied to the input operands using the inexact function shown in Algorithm 2. The arithmetic operation is then performed using an MPFR operation, rounded to W bits. Random rounding is then applied to the result using the inexact function and the final result is then converted to its original format by rounding to p bits. MPFR implements correct rounding according to the IEEE-754 standard with rounding error $\delta(x) \leq \epsilon$. Rounding error will occur both during the MPFR operation, δ_W, and when rounding to the original precision, δ_p. In order to implement correct rounding while simulating infinite precision during the MCA operation we must ensure that $\delta_W \leq \frac{1}{2}\delta_p$. The worst case scenario will occur when $t = 1$, as when $t = 0$ the initial MPFR rounding stage will round to the original precision with $\delta_W \leq \epsilon$ resulting in an exact value and $\delta_p = 0$. When $t = 1$ the rounding error in the MPFR operation will be limited as follows assuming the general case $\delta \leq 2^{-p}$:

$$\delta_W \leq 2^{-(p+t)} \tag{11}$$

$$\leq \frac{1}{2}2^{-p},\ t \geq 1 \tag{12}$$

$$\leq \frac{1}{2}\delta_p \tag{13}$$

MCALIB implements the four basic arithmetic operations, $\{+, -, \times, \div\}$, unary minus, and the set of comparison operators, $\{==, != , <, >, \leq, \geq\}$ for single and double precision formats. Comparison operators do not require MCA and as such they are implemented using MPFR operations without the use of the inexact function. The library includes two global parameters for controlling an MCA execution. The integer `MCALIB_T` sets the virtual precision, t, of MCA operations while the integer `MCALIB_OP_TYPE` allows the application of MCA to be controlled using a set of pre-processor symbols defined as part of the MCALIB library. These symbols, their values and their functions are shown in Table II. Both parameters can be modified at runtime.

4.3. MCALIB Features & Work Flow

MCALIB has been designed to facilitate the following analyses;

— Detection and quantitative analysis of sensitivity to rounding error.
— Analysis of individual algorithms to determine if single or double precision floating point arithmetic is required.
— Optimization of individual algorithms for precision.
— Comparison of algorithms to determine the most suitable implementation.

Each of these features is implemented by applying MCALIB to a problem according to the following MCALIB work flow as described below.

4.3.1. Algorithm Analysis. MCA is applied by first analysing the algorithm to be tested in order to determine the following;

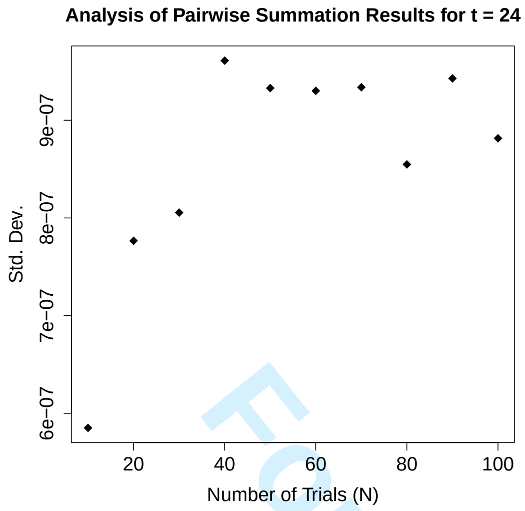

Fig. 1. Pairwise summation - comparison of standard deviation for virtual precision $t = 24$ and different number of trials, N.

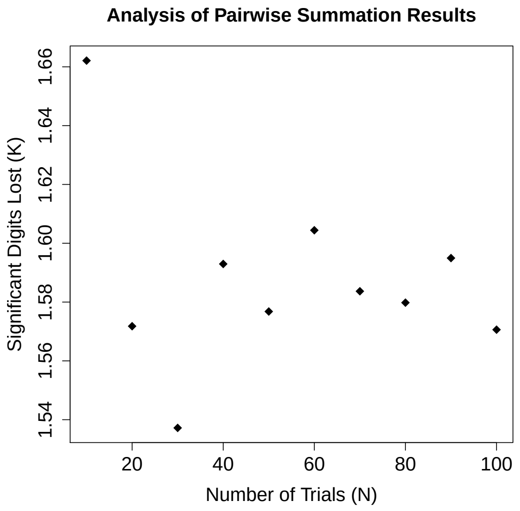

Fig. 2. Pairwise summation - comparison of significant figures lost K and different number of trials, N.

— Where should MCA be enabled, i.e. which values are exact and which are not?
— What outputs are of interest, i.e. how is accuracy in this algorithm defined?

These questions are of high importance as they will have a significant impact on the results if not answered correctly. Although MCALIB provides an automated implementation of MCA, it is still a *naive* implementation, i.e. the system does not understand the difference between exact and inexact values and must be informed of this difference by the developer. Using MCALIB, all FP operations are re-written as function calls to the MCALIB library. Determining which inputs and outputs are to be treated as exact or inexact is a decision left to the developer, and is achieved by enabling or disabling precision bounding and random rounding individually as described below. Determining what outputs are of interest is a question of determining what variables determine the overall stability of an algorithm.

4.3.2. Source Code Modification. Having determined the above, the second stage of the work flow involves modifying the source code. Implementation is a simple process and very few modifications are required. Developers need to add the MCALIB header file `mcalib.h` and modify their compilation process to utilize the cilly compiler and include the MCALIB library file `libmcalib.a`. MCA can be enabled or disabled where appropriate by setting the value of the control parameter `MCALIB_OP_TYPE` and the virtual precision can be set using the parameter `MCALIB_T`.

4.3.3. Data Collection. Once the original source code has been correctly modified the third stage of the MCALIB work flow is collection of data. In order to do this the following steps are required.

— Determine the input domain to be tested.
— Execute the required number of trials and collect data from the watched output(s).

As stated previously MCALIB is a naive implementation of MCA and as such decisions regarding the input domain are left to the developer. This is an important step, as MCA performs a dynamic error analysis and results are only relevant to the input domain tested. For example, if testing a summation algorithm using uniformly distributed inputs, $x \in U[-1, 1]$, the results of MCALIB analysis will only be relevant for this domain. Once the input domain has been determined the trials must be executed and the output data collected. An important consideration for Monte Carlo methods is the number of trials to be performed, this number being directly affected by the sampling methodology in use. For the purposes of MCALIB simplified random sampling has been implemented and it is recommended that a minimum of 100 trials be performed for any experiments. Decreasing the number of trials performed may have adverse effects on the results of analysis using techniques as shown in Section 5. As can be seen in Figures 1 and 2 decreasing the number of trials will adversely affect the results, the standard deviation and the calculated value of K do not converge until approximately $N = 50$. The recommended number of samples is based on experimental results presented in this paper, worst case sample size considerations [Hammersley and Handscomb 1964, Chapter 3], and experimental data presented in [Parker 1997]. While the recommend number of samples may appear high, it is believed that this figure can be reduced using techniques such as Quasi Monte Carlo Simulation. This will be the subject of future research.

4.3.4. Results Analysis. Having performed the required number of experiments and collected the relevant output data the next stage of the MCALIB work flow is results analysis. Using the methods described in the next section results of MCA trials may be analysed to determine the total number of digits lost to rounding error and the minimum precision required in order to avoid a total loss of significance. If no valid results are available then the virtual precision range should be widened, particularly at the top end, to collect more data at more stable precision values. If the normality tests fail consistently the developer should return to step 1 to re-analyse the algorithm and ensure that the input domain and outputs are being monitored correctly.

5. ANALYSIS OF MCA RESULTS

In previous publications [Parker 1997] the analysis of MCA results has been limited to determining the number of significant digits, and pass/fail analysis performed by comparing the mean and standard deviation of MCA results. We feel that this approach can be expanded and more formally defined in order to provide a more rigorous definition of sensitivity to rounding error in MCA results, allowing analysts to draw more meaningful conclusions from the results of MCA analysis. In this paper sensitivity to rounding error is defined using two measurements:

— The number of base-2 significant digits lost due to rounding error, K
— The minimum precision required to avoid an unexpected loss of significance, t_{min}

We must first address the ideal case for error in MCA. If relative error is defined as in Equation 2.1 then it has been noted in [Wilkinson 1994; Higham 1996; Goldberg 1990] that the relative error is limited by $\delta \leq 2^{-p}$ for binary FP systems. From [Parker 1997, page 19], the definition of relative error is used to determine the expected number of significant binary digits available from a p-digit FP system:

$$\delta \leq 2^{-p} \tag{14}$$
$$p \geq -\log_2(\delta) \tag{15}$$

These definitions may be adapted for MCA by replacing the precision of the FP system, p, with the virtual precision, t, of an MCA operation. Thus the relative error δ in a MCA operation for a virtual precision t is given by $\delta \leq 2^{-t}$, and the expected number of significant binary digits in a t-digit MCA operation is at least t. Using this definition a proof has been provided [Parker 1997, page 23] giving the total significant binary digits in a set of MCA results:

$$s' = \log_2 \frac{\mu}{\sigma} \tag{16}$$

Where μ is the mean and σ the standard deviation of the MCA results. Using the definitions in this section the total number of significant digits lost in a MCA result set, K, may be defined as follows:

$$K = t - s' \tag{17}$$
$$= t - \log_2(\frac{\mu}{\sigma}) \tag{18}$$
$$= \log_2(\Theta) + t \tag{19}$$

Where $\Theta = \frac{\sigma}{\mu} \to \mu \neq 0$ is the **Relative Standard Deviation (RSD)** of the MCA results.

As noted by Sterbenz, [Sterbenz 1974, chapter 7], in an ideal case a linear relationship exists between the precision of a FP system, p, and significant figures in the output. Using MCA, this linear relationship exists between t and $\log(\Theta)$. We identify the point of departure as when the algorithm being analysed is affected in a non-linear way by rounding error. We propose that the breakaway point in the linear model represents t_{min}; the minimum precision required to avoid an unexpected loss of significance in the results. In order to determine the the best fit of the relative error model results below, outliers are not used in the calculation of K.

5.1. Linear Regression Analysis

In order to determine the value of t_{min} and K a linear regression with a log transformed variable is used, with $\log(\Theta)$ as the dependent variable and t as the exploratory variable in the following form:

$$\log_{10}(\Theta) = \log_{10}(2^{K-t}) \tag{20}$$
$$= -\log_{10}(2)t + \log_{10}(2)K \tag{21}$$
$$= mt + c \tag{22}$$

Where $m = -\log_{10}(2) = -0.30103$ is the slope and c is the intercept such that $K = \log_2(10^c)$. Due to the requirement of detecting outlying results robust regression methods are used to evaluate the linear model. The example presented in [Fox 2002] performs robust regression using M-Estimation through the Iteratively Re-weighted Least Squares (IRLS) approach for 2-D optimization. While this approach is ideal for MCA analysis due to it's insensitivity to outliers, the approach can be simplified to a 1-D optimization problem as the slope of the linear model is already known. Given a set of MCA results for virtual precision values $t \in [1, t_{max}]$ a summary set is created by calculating Θ at each t value. It should be noted that while the samples used to calculate an individual value for Θ are Independent and Identically Distributed (IID), the complete sample set is not in general identically distributed. Given these inputs the intercept c is calculated by minimizing the following objective function using Brent's method [Brent 1973] for single variable optimization;

$$f(x) = \sum_{i=1}^{t_{max}} \gamma^{t_{max}-i} \rho_H(e_i) \tag{23}$$

where $e_i = \Theta_i - (mt_i + c)$ is the residual error, $c \in [(\Theta_{t_{max}} - mt_{max}) \pm 2m]$ is the initial search space for the intercept, $\gamma = 0.75$ and $\rho_H(e)$ is the Huber loss function [Huber

1964];

$$\rho_H(e) = \begin{cases} \frac{1}{2}e^2, & \text{for } |e| \leq k \\ k|e| - \frac{1}{2}k^2, & \text{for } |e| > k \end{cases} \tag{24}$$

where $k = 1.345\sigma$ and σ is the standard deviation of the residual error set, e. Having determined the linear model the outlying values of Θ are found by calculating a set of predicted values $P_t = mt + c$ and comparing these to the values for Θ obtained via MCA. If a value Θ_t differs from its equivalent predicted value, P_t, by more than half a binary digit it is classed as an outlier. The breakaway point, t_B is calculated by finding the highest t value where $|P_t - \Theta_t| > \log_{10}(2^{0.5})$. The value of t_{min} is then set to $t_B + 1$.

5.2. Assumption of Normality and Conditions on Results

In order to perform analysis using the statistical methods listed in this chapter the input data set is typically assumed to be normally distributed, however, in the case of MCA no assumption of normality is made. This is explicitly stated by Parker [Parker 1997, p. 49] and is intended to allow for open ended statistical testing of MCA results. In order to provide a strong estimate on the result of K and t_{min} the normality of the sample set must first be verified for each value of t. This is determined on the raw MCA data at each t step, requiring a total of $t_{max} - t_{min}$ tests. This is done using the Anderson-Darling test to asses the goodness of fit of the frequency distribution of results to a normal distribution. If the test fails, warnings are provided on the plotted output of the calculation and the result sets that have failed the test are removed and not used for the calculation of K or t_{min}. The calculation of K and t_{min} must be done in conjunction with bounds on the input space of the function or algorithm under investigation, i.e. the results of the linear regression do not provide a guarantee of the error in an algorithm in the general case, but rather an estimate of the accuracy of the algorithm under the specific conditions tested using MCALIB.

6. TESTING & CASE STUDIES

Testing is performed by varying the virtual precision, $1 \leq t \leq p$, and performing N executions at each t value. For the tests conducted in this paper, unless stated otherwise, we use t values from 1 to 53 and number of trials at each t value $N = 100$. In this section we describe the programs used to test MCALIB.

6.1. Chebyshev Polynomials

Chebyshev polynomials [Rivlin 1990] are a series of orthogonal polynomials typically used in approximation theory. In this case we have used Chebyshev polynomials of the first kind, defined as follows:

$$T_0(z) = 1 \tag{25}$$

$$T_1(z) = z \tag{26}$$

$$T_{n+1}(z) = 2zT_n(z) - T_{n-1}(z) \tag{27}$$

Polynomials of the first kind can be represented as unique polynomials satisfying the following trigonometric definition:

$$T_n(z) = \cos(n \cos^{-1}(z)) \tag{28}$$

ALGORITHM 3: Pairwise Summation Algorithm

```
Input: Vector $X[1...n]$
Output: Sum $s$ of vector $X$
$n_{max}$ = 1;
if $n \leq n_{max}$ then
    $s$ = $X[1]$;
    for $i = 2$ to $n$ do
        $s$ = $s$ + $X[i]$;
    end
else
    $m$ = floor($n$ / 2);
    $s$ = pw($X[1...m]$) + pw($X[m+1...n]$);
end
return $s$
```

ALGORITHM 4: Kahan Summation Algorithm

```
Input: Vector $X[1...n]$
Output: Sum $s$ of vector $X$
$s$ = 0.0;
$c$ = 0.0;
for $i$ = 1 to $n$ do
    $y = X[i] - c$;
    $t = s + y$;
    $c = (t - s) - y$;
    $s = t$;
end
Return $s$
```

In particular the $T_{20}(z)$ polynomial:

$$T_{20}(z) = \cos(20\cos^{-1}(z)) \tag{29}$$

$$\begin{aligned} &= 52488z^{20} - 2621440z^{18} + 5570560z^{16} \\ &- 6553600z^{14} + 4659200z^{12} - 2050048z^{10} \\ &+ 549120z^{8} - 84480z^{6} + 6600z^{4} \\ &- 200z^{2} + 1 \end{aligned} \tag{30}$$

has been analysed by both Wilkinson [Wilkinson 1994] and Parker [Parker 1997], who note that due to catastrophic cancellation occurring among the coefficients of the expanded series the polynomial becomes ill-conditioned at the roots near $z = \pm 1$.

6.2. Summation Algorithm

FP summation is a widely used operation that sums a sequence of n FP values:

$$s = \sum_{i=1}^{n} x_i, \text{ for } n \geq 3 \tag{31}$$

Due to its widespread use in algebraic operations the accuracy of summation has been analysed in various publications and it has been shown that the relative error of the naive summation algorithm grows with order $O(\epsilon n)$ [Linz 1970; Malcolm 1971; Higham 1993]. For this paper the naive approach is compared to two alternative summation algorithms, the **Pairwise** [Higham 1993] and **Kahan** [Kahan 1965] summation algorithms, shown in Algorithms 3 and 4. Both of these algorithms have been shown to reduce numerical instability. In the case of Pairwise summation this is done using a divide and conquer strategy that reduces the relative error to order $O(\epsilon \log n)$ while not increasing the number of arithmetic operations used. Kahan summation uses a compensated sum to track round-off error during summation and reduces relative error to order $O(\epsilon)$, but significantly increases the required number of arithmetic operations.

The naive, Kahan and pairwise sum methods are compared using a set of sample values generated using the following [Sandu 2000]:

$$x_i = 10^{-p} \tag{32}$$

$$p = \lceil \log_{10}(9i + 1) - 1 \rceil \tag{33}$$

for $1 \leq i \leq 1111$.

6.3. Linear Algebra

Linear algebra subroutines are widely used in computer science and engineering. Accurate implementation of these algorithms is essential. Their implementation necessitates a large number of numeric operations and MCA is well suited for analysis of the potential effects of rounding error. For the purposes of this paper we have tested two implementations for determining the solution to a dense $n \times n$ system of linear equations $Ax = b$.

The implementations used for testing are the LINear equations software PACKage (LINPACK) benchmark [Dongarra et al. 2003], a tool which uses Gaussian Elimination with partial pivoting as an example of a general engineering problem in order to test a systems peak performance in terms of Floating Point Operations per Second (FLOPS), and a standard implementation of LU decomposition with back substitution from Numerical Recipes [Press et al. 2007]. Precision testing and error analysis have been performed using the array size $n = 100$. With the value of A and b set using the `matgen` method provided as part of the LINPACK implementation used in this test case [Toy and Menninger 1994]. Statistical measurements were performed using the Euclidean, (L^2), norm of the result vector $x[n]$, defined as follows:

$$||x|| := \sqrt{x_1^2 + ... + x_n^2} \tag{34}$$

6.4. L-BFGS Optimization

Limited memory Broyden, Fletcher, Goldfarb, Shanno (L-BFGS) optimization [Liu and Nocedal 1989] is an implementation of quasi-Newton optimization using the Broyden, Fletcher, Goldfarb, Shanno (BFGS) update method for approximation of the Hessian Matrix. L-BFGS stores a finite number of vectors to represent the approximation, unlike the original BFGS method which stores a dense $n \times n$ approximation. An important part of this algorithm is the line search method, used to determine the local minimum $x*$ of an objective function $f : \mathbb{R}^n \rightarrow \mathbb{R}$. The objective function used for testing in this paper is the Rosenbrock function [Rosenbrock 1960], a well known convex function used for performance testing of optimization systems. This function has been provided as part of the L-BFGS implementation used for this paper [Liu and Nocedal 1989], and is implemented for 10 dimensions using:

$$f(\mathbf{x}) = \sum_{i=1}^{10} [(1 - x_i)^2 + 100(x_{i+1} - x_i^2)^2], \forall x \in \mathbb{R}^n \tag{35}$$

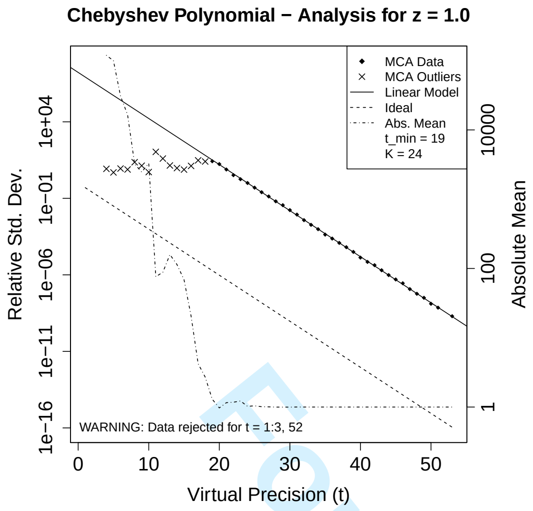

Fig. 3. Chebyshev Polynomial - Sensitivity to rounding error at $z = 1.0$

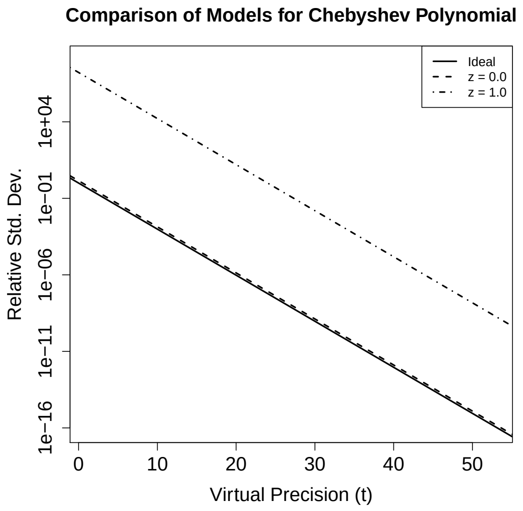

Fig. 4. Chebyshev Polynomial - Comparison of results for $z = 0.0$ and $z = 1.0$

Table III. Results - Chebyshev Polynomial

Input - z	**Min. Req. Precision** - t_{min}	**Sig. Fig. Lost** - K
0.0	5	0.5
0.2	5	5.4
0.4	11	11.5
0.6	13	15.2
0.8	18	20.0
1.0	19	24.0

Note: Full Analysis of Chebyshev Polynomial

with the input vector x defined as follows;

$$x[i] = \begin{cases} 1.2 & \text{if } i \text{ is odd} \\ 10 & \text{if } i \text{ is even} \end{cases} \tag{36}$$

for $i \in [1, 10]$. The L-BFGS implementation used for testing provides a choice between 4 different line search methods, Moore-Thuente, Armijo, Wolfe and Strong Wolfe [Moré and Thuente 1994; Dennis and Schnabel 1987] methods. Testing has been conducted for all four line search methods and statistical measurements are again performed using the Euclidean norm of the result vector.

7. RESULTS

In this section we present results of MCA analysis of several sample algorithms. Throughout this section results of MCA analysis are presented using plots generated via methods described in Section 5. The plots shown in Figures 3, 6, 8, and 10 provide detail on the results of the linear regression analysis. These compare the linear model with the ideal error case, ($\delta = 2^{-t}$), the experimental MCA results which were classified as outliers are clearly marked, as well as a plot of the absolute mean, $|\mu|$ to allow the mean to be checked in case it approaches zero. The plots are designed to provide a method for quick visual inspection of the MCA results. Inside the legend the magnitude of K, indicated by the distance between the linear model and the ideal case, and the

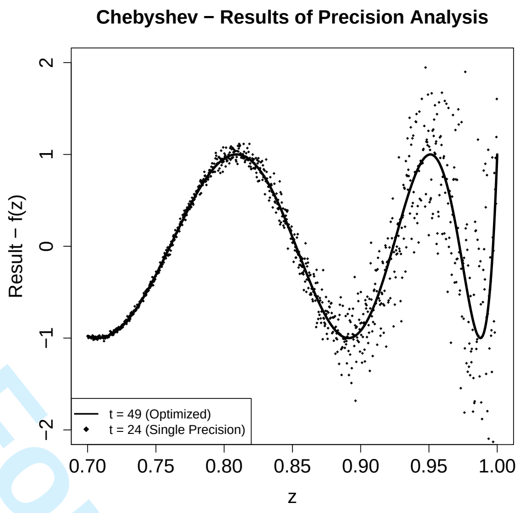

Fig. 5. Chebyshev Polynomial - Comparison of single ($t = 24$) and optimized ($t = 49$) precision.

Table IV. Results - Chebyshev Polynomial

Type	t	μ	Θ
Single	24	0.9985	1.2119e+00
Optimized	49	1.0000	3.4492e-08

Note: Comparison of Single and Optimized Precision Results for Chebyshev Polynomial (using $z = 1.0$)

value of t_{min}, indicated by the position of the outlying data points, are given. The second type of plot presented, (Figures 4, 7, 9, and 11), is designed to provide a comparison of the different algorithms being tested. These plots compare the linear models generated through analysis of the MCALIB results with the ideal error case.

7.1. Error Detection and Optimization of Sample Algorithms

One of the primary functions of MCA is to detect sensitivity to rounding error within tested algorithms, indicated by a large variance in the results of repeated executions. Using the relative error model and the methods detailed in Section 5, it is possible to determine the overall sensitivity of tested algorithms to rounding error and to optimize these algorithms by determining their minimum precision requirements.

For the Chebyshev Polynomial, testing has been conducted using input values for z between 0 and 1 in steps of 0.2, conducting $N = 100$ executions for all t values between 1 and 53 at each z step. Results for all cases are shown in Table III, results for the worst case $z = 1$ are detailed in Figure 3 and results for the $z = 0.0$ and $z = 1.0$ cases are compared in Figure 4. Initially at $z = 0$ the sensitivity to rounding error is negligible, as evidenced by a low value for t_{min} and less than 1 significant figure lost to rounding error. As z is increased to approach the root at $z = 1$ the number of significant figures decreases until, at the worst case point $z = 1$, 24 significant figures are lost to rounding error. At this point the minimum precision required to avoid an unexpected loss of significance in the results has risen to 19 bits. Having quantified the sensitivity to rounding error for input values between 0 and 1, it is possible to use the values for

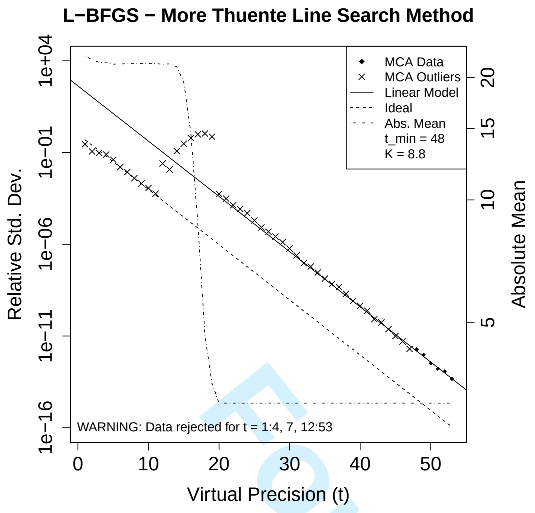

Fig. 6. L-BFGS Optimization - Analysis of More-Thuente line search method

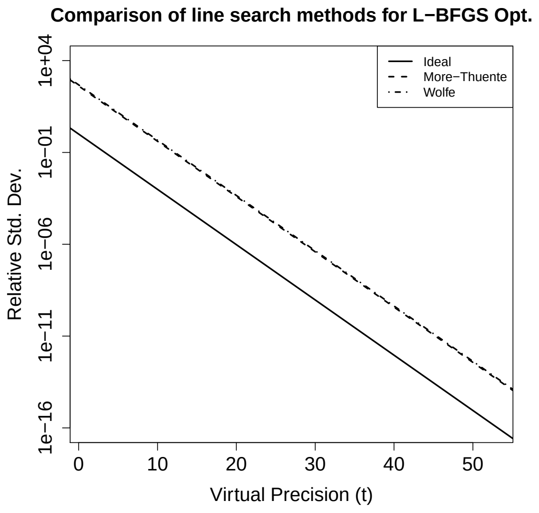

Fig. 7. L-BFGS Optimization - Comparison of More-Thuente & Wolfe line search methods

Table V. Analysis of L-BFGS Optimization

Search Type	**Min. Req. Precision** - t_{min}	**Sig. Fig. Lost** - K
More-Thuente	48	8.7
Wolfe	19	8.9
Str. Wolfe	36	8.8
Armijo	53	8.9

Note: Analysis of Line Search Methods for L-BFGS Optimization

K and t_{min} to optimize this algorithm and determine the precision level required to achieve results normally expected from single precision FP operators. Previously this was often achieved by simply switching to double precision FP operators. MCALIB allows for the effects of rounding error to be quantified and this information used to determine a required precision level. This can be done by simply adding the expected number of digits lost to the required precision level, 24 in this case, and ensuring the resulting value is greater than or equal to t_{min}. Table III shows that a precision of at least 19 bits is required, and due to the expected loss of significant figures for the worst case input, $K = 24.02$, a precision of $\lceil p + K \rceil = 49$ is required. The results of comparison testing between t values of 24 (single precision), and 49, (optimized precision), are shown in Table IV. These results have been produced using the worst case input, $z = 1$. It can be seen that the relative standard deviation is 10^8 times lower for the optimized case, and is the same order of magnitude as the maximum relative error expected from single precision arithmetic, ($\delta = 2^{-24} \approx 6 \times 10^{-8}$). Figure 5 plots the results of the Chebyshev polynomial for both single ($t = 24$) and optimized ($t = 49$) precision calculated using MCALIB. From this plot the difference between the two precision levels can be seen. A precision level of 49 results in a smooth curve, while using a level of 24 results in a random spread of points.

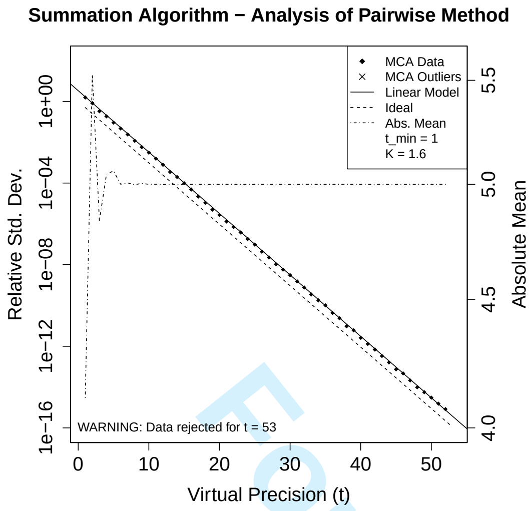

Fig. 8. Summation Algorithm - Analysis of Pairwise Summation Method

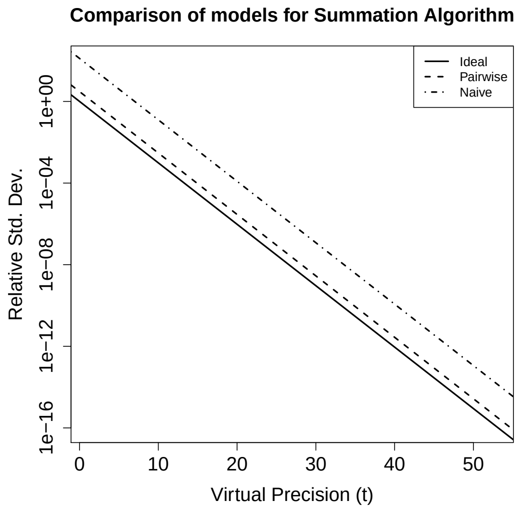

Fig. 9. Summation Algorithm - Comparison of Results for Pairwise and Naive Algorithms

Table VI. Analysis of Summation Algorithms

Algorithm Type	**Min. Req. Precision** - t_{min}	**Sig. Fig. Lost** - K
Naive	7	7
Kahan	7	7
Pairwise	1	1.6

Note: Summation Algorithm Results - Naive, Kahan & Pairwise

7.2. Comparison of Single and Double Precision Floating Point Formats

A simpler form of error analysis that may be performed with MCALIB is the comparison of single and double floating point operators. In this case an individual algorithm maybe tested in order to determine if the single precision floating point format is sufficient for the given input domain, or if double precision type operators are required. This type of analysis has been used to determine the sensitivity to rounding error of different line search algorithms as used in L-BFGS optimization of the n-dimension Rosenbrock function, allowing for both the comparison of line search methods and the selection of single or double precision operators for the tested input domain. The results of error analysis for all four line search methods are shown in Table V. The results of testing the More-Thuente line search are plotted in Figure 6 and results for the More-Thuente and Wolfe line search methods are compared in Figure 7. From the results table it can be seen that all four line search methods lose approximately 9 significant figures to rounding. This result coupled with the results for t_{min} indicates that single precision floating point operators are insufficient for this algorithm, however, it can be seen from the warning on the bottom left of Figure 6, a total of 47 data points have been rejected due to non-normality of the data set. This is most likely caused by the iterative nature of the algorithm under investigation, and the fact that the optimization process is attempting to find a solution within an error bound of 2^{-53}. Given that the virtual precision of the MCA operators is varied between 1 and 53 the error analysis method is having an adverse affect on the accuracy of the solution. For the purposes of demonstration the non-normal data points have been forcefully included in the results analysis, but in practice these results are not viable and the experimental conclusions

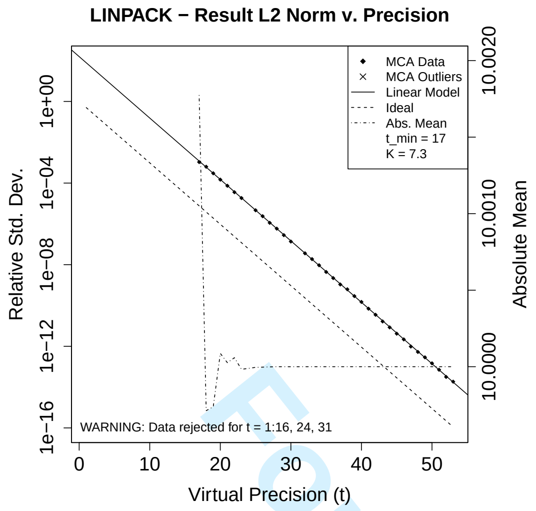

Fig. 10. Analysis of LINPACK benchmark

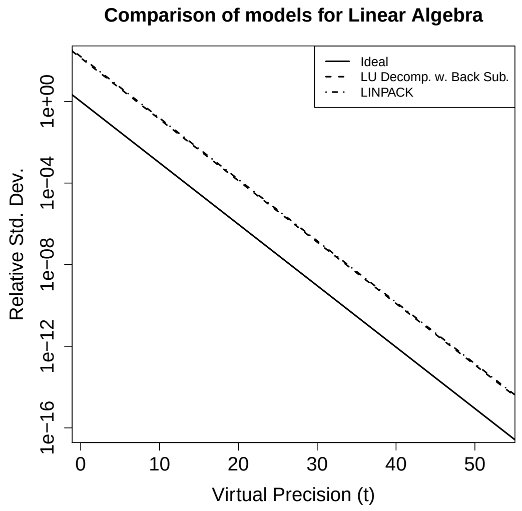

Fig. 11. Analysis of LINPACK Benchmark

Table VII. Comparison of Linear Solvers

Algorithm Type	**Min. Req. Precision** - t_{min}	**Sig. Fig. Lost** - K
LU Decomp. w. Back Sub.	17	7.1
LINPACK	17	7.3

Note: Linear Solvers - Comparison of LINPACK and LU Decomposition with Back Substitution

should be rejected. As such, while these results indicate the possibility that single precision FP is not suitable for the tested input domain, further analysis is required.

7.3. Comparison of Algorithm Implementations

In addition to performing an analysis of individual algorithms s demonstrated in the previous section, MCALIB can be used to compare competing algorithms or implementations in order to determine the best approach. The first set of algorithms tested are algorithms for FP summation, including the Naive, Kahan and Pairwise algorithms. The results of analysis for all three algorithms are shown in Table VI, the results of analysis of the Pairwise method are detailed in Figure 8 and the results for the Pairwise and Naive methods are compared in Figure 9. From these results it can be seen that all three algorithms demonstrate low sensitivity to rounding error. The Pairwise method demonstrates significantly lower sensitivity to rounding errors when compared with the alternative methods. This is evident in the lower value for t_{min}, with a result of 4 for the Pairwise algorithm versus 11 and 10 for the Kahan and Naive methods respectively. The Pairwise method is also losing less than 2 significant digits to rounding error, compared with the 7 significant digits lost for the Naive and Kahan methods. While all three methods demonstrate low sensitivity to rounding error and may be analysed using single precision operators, the Pairwise method provides the best approach for floating point summation for the tested input domain, (as detailed in Section 6.2).

This same type of analysis has also been used to compare a linear solver from Numerical Recipes [Press et al. 2007] with the one in the LINPACK benchmark [Dongarra et al. 2003]. The results for analysis of the two algorithms are shown in Table VII, the results of analysis of the LINPACK benchmark are detailed in Figure 10 and results for both implementations are compared in Figure 11. As was the case with the summation algorithms, both algorithms show a low level of sensitivity to rounding error and the result for t_{min} for both methods indicates that single precision formats are suitable for use with the tested input domain.

The error analysis results also clearly indicate a similar level of sensitivity to rounding error available in both algorithms, this being demonstrated by the approximately 7 significant figures lost to rounding error in both cases. The overall effect of rounding error on the results for the LINPACK benchmark can be seen in Figure 10. As the virtual precision is increased beyond $t = 17$ the relative standard deviation decreases exponentially forming a linear relationship with the virtual precision. These results can also be produced using single precision floating point operators if necessary. However, the values for t_{min} and K indicate that the algorithm becomes highly sensitive to rounding error if the precision is decreased below 17. Furthermore if the required significance of the results must be equivalent to single precision FP, a precision of $\lceil p + K \rceil = 32$ is recommended when using these algorithms on the tested input domain.

8. CONCLUSIONS

The MCP approach presented in this paper allows users to gain en empirical sense of the effects of rounding on the output of a program for a given input. It's application is facilitated by MCALIB, an open source tool which applies source to source compilation to rewrite FP operators to call our MCA library. Furthermore analysis techniques for better interpretation of MCA results have been presented, results being summarized by t_{min} and K. These expand the use of MCA and further demonstrate the benefit of this type of analysis for evaluating FP SW. Further work in this area will focus on investigating the use of quasi Monte Carlo techniques to reduce the required number of trials, and the use of MCA analysis to facilitate mixed precision implementations.

9. ACKNOWLEDGEMENTS

We would like to thank Dr Reiichiro Kawai at The University of Sydney School of Mathematics and Statistics for his assistance in with the results analysis in section 5. We would also like to thank the anonymous reviewers, and associate editor Dr. Michael Burke for their assistance in improving this manuscript.

REFERENCES

A. Amaricai, M. Vladutiu, and O. Boncalo. 2009. Design of Floating Point Units for Interval Arithmetic. In *Research in Microelectronics and Electronics, 2009. PRIME 2009. Ph.D.* 12 –15.

J. Asserrhine, J.M. Chesneaux, and J.L. Lamotte. 1995. Estimation of Round-Off Errors on Several Computer Architectures. *Journal of Universal Computer Science* 1, 7 (1995), 455–468.

Bruno Blanchet, Patrick Cousot, Radhia Cousot, Jérôme Feret, Laurent Mauborgne, Antoine Miné, David Monniaux, and Xavier Rival. 2002. Design and Implementation of a Special-Purpose Static Program Analyzer For Safety-Critical Real-Time Embedded Software. *The Essence of Computation* (2002), 85–108.

Richard P Brent. 1973. *Algorithms for Minimization without Derivatives*. Courier Dover Publications.

H. Brönnimann, G. Melquiond, and S. Pion. 2006. The Design of the Boost Interval Arithmetic Library. *Theoretical Computer Science* 351, 1 (2006), 111–118.

Ashley W Brown, Paul HJ Kelly, and Wayne Luk. 2007. Profiling Floating Point Value Ranges for Reconfigurable Implementation. In *Proceedings of the 1st HiPEAC Workshop on Reconfigurable Computing*. 6–16.

Ashley W Brown, Paul HJ Kelly, and Wayne Luk. 2008. Profile-Directed Speculative Optimization of Reconfigurable Floating Point Data Paths. In *Proceedings of the Workshop on Reconfigurable Computing at HiPEAC*.

R. Chotin and H. Mehrez. 2002. A Floating-Point Unit using Stochastic Arithmetic Compliant with the IEEE-754 Standard. In *Electronics, Circuits and Systems, 2002. 9th International Conference on*, Vol. 2. 603 – 606 vol.2.

George A Constantinides. 2003. Perturbation Analysis for Word-Length Optimization. In *Field-Programmable Custom Computing Machines, 2003. FCCM 2003. 11th Annual IEEE Symposium on*. IEEE, 81–90.

George A Constantinides. 2006. Word-Length Optimization for Differentiable Nonlinear Systems. *ACM Transactions on Design Automation of Electronic Systems (TODAES)* 11, 1 (2006), 26–43.

George A Constantinides, Peter YK Cheung, and Wayne Luk. 2001. The Multiple Wordlength Paradigm. In *Field-Programmable Custom Computing Machines, 2001. FCCM'01. The 9th Annual IEEE Symposium on*. IEEE, 51–60.

George A Constantinides, Peter YK Cheung, and Wayne Luk. 2002. Optimum Wordlength Allocation. In *Field-Programmable Custom Computing Machines, 2002. Proceedings. 10th Annual IEEE Symposium on*. IEEE, 219–228.

George A Constantinides, Peter YK Cheung, and Wayne Luk. 2003. Wordlength Optimization for Linear Digital Signal Processing. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on* 22, 10 (2003), 1432–1442.

Keith D Cooper, Timothy J Harvey, and Ken Kennedy. 2002. Iterative Dataflow Analysis, Revisited. *Proceedings of the (PLDI'02)* (2002).

Patrick Cousot and Radhia Cousot. 1977. Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. In *Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*. ACM, 238–252.

Patrick Cousot and Radhia Cousot. 1979. Systematic Design of Program Analysis Frameworks. In *Proceedings of the 6th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*. ACM, 269–282.

Marc Daumas and Guillaume Melquiond. 2010. Certification of bounds on expressions involving rounded operators. *ACM Transactions on Mathematical Software (TOMS)* 37, 1 (2010), 2.

L.H. de Figueiredo and J. Stolfi. 2004. Affine Arithmetic: Concepts and Applications. *Numerical Algorithms* 37, 1 (2004), 147–158.

David Delmas, Eric Goubault, Sylvie Putot, Jean Souyris, Karim Tekkal, and Franck Védrine. 2009. Towards an Industrial use of FLUCTUAT on Safety-Critical Avionics Software. *Formal Methods for Industrial Critical Systems* (2009), 53–69.

David Delmas and Jean Souyris. 2007. Astrée: From Research to Industry. *Static Analysis* (2007), 437–451.

J.E. Dennis and R.B. Schnabel. 1987. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. Vol. 16. Society for Industrial Mathematics.

Alain Deutsch. 2003. Static Verification of Dynamic Properties. *PolySpace White Paper* (2003).

J.J. Dongarra, P. Luszczek, and A. Petitet. 2003. The LINPACK Benchmark: Past, Present and Future. *Concurrency and Computation: Practice and Experience* 15, 9 (2003), 803–820.

Floating-point Working Group. 2008. IEEE Standard for Floating-Point Arithmetic. *IEEE Std 754-2008* (2008), 1 –58.

Ian Foster and Stephen Taylor. 1994. A compiler approach to scalable concurrent-program design. *ACM Transactions on Programming Languages and Systems (TOPLAS)* 16, 3 (1994), 577–604.

L. Fousse, G. Hanrot, V. Lefèvre, P. Pélissier, and P. Zimmermann. 2007. MPFR: A Multiple-precision Binary Floating-point Library with Correct Rounding. *ACM Transactions on Mathematical Software (TOMS)* 33, 2 (2007), 13.

John Fox. 2002. Robust Regression. *An R and S-Plus companion to applied regression* (2002).

Michael Frechtling and Philip H.W. Leong. 2013. An FPGA-based floating point unit for rounding error analysis. In *Transforming Reconfigurable Systems*, Wayne Luk and George Constantinides (Eds.). Imperial College Press, Imperial College, London. (To appear), http://www.ee.usyd.edu.au/people/philip.leong/UserFiles/File/papers/mca_icpress13.pdf.

Altaf Abdul Gaffar, Wayne Luk, Peter YK Cheung, Nabeel Shirazi, and James Hwang. 2002. Automating Customisation of Floating-Point Designs. In *Field-Programmable Logic and Applications: Reconfigurable Computing Is Going Mainstream*. Springer, 523–533.

Altaf Abdul Gaffar, Oskar Mencer, and Wayne Luk. 2004. Unifying Bit-Width Optimisation for Fixed-Point and Floating-Point Designs. In *Field-Programmable Custom Computing Machines, 2004. FCCM 2004. 12th Annual IEEE Symposium on*. IEEE, 79–88.

A Abdul Gaffar, Oskar Mencer, Wayne Luk, Peter YK Cheung, and Nabeel Shirazi. 2002. Floating-Point Bitwidth Analysis via Automatic Differentiation. In *Field-Programmable Technology, 2002.(FPT). Proceedings. 2002 IEEE International Conference on*. IEEE, 158–165.

David Goldberg. 1990. Computer Arithmetic. *Computer Architecture: A Quantitative Approach, David Patterson and John L. Hennessy, Eds. Morgan Kaufmann, Los Altos, Calif., Appendix A* (1990).

F. Goualard. 2006. GAOL: Not Just Another Interval Arithmetic Library. (2006).

E. Goubault and S. Putot. 2006. Static Analysis of Numerical Algorithms. *Static Analysis* (2006), 18–34.

John Michael Hammersley and David Christopher Handscomb. 1964. *Monte carlo methods*. Vol. 1. Springer.

N.J. Higham. 1993. The Accuracy of Floating Point Summation. *SIAM Journal on Scientific Computing* 14, 4 (1993), 783–799.

Nicholas J Higham. 1996. *Accuracy and Stability of Numerical Algorithms*. Number 48. Siam.

William E. Howden. 1980. Applicability of software validation techniques to scientific programs. *ACM Transactions on Programming Languages and Systems (TOPLAS)* 2, 3 (1980), 307–320.

Peter J Huber. 1964. Robust Estimation of a Location Parameter. *The Annals of Mathematical Statistics* 35, 1 (1964), 73–101.

François Irigoin, Pierre Jouvelot, and Rémi Triolet. 1991. Semantical interprocedural parallelization: An overview of the PIPS project. In *Proceedings of the 5th international conference on Supercomputing*. ACM, 244–251.

F. Jézéquel and J.M. Chesneaux. 2008. CADNA: A Library for Estimating Round-off Error Propagation. *Computer Physics Communications* 178, 12 (2008), 933–955.

W. Kahan. 1965. Pracniques: Further Remarks on Reducing Truncation Errors. *Commun. ACM* 8, 1 (1965), 40.

William Kahan. 1996. The improbability of probabilistic error analyses for numerical computations. In *UCB Statistics Colloquium, evans hall edition*. 20.

W Kahan. 2006. *How Futile are Mindless Assessments of Round-off in Floating Point Computation*. http://www.cs.berkeley.edu/∼wkahan/Mindless.pdf.

Gary A Kildall. 1973. A Unified Approach to Global Program Optimization. In *Proceedings of the 1st annual ACM SIGACT-SIGPLAN symposium on Principles of programming languages*. ACM, 194–206.

R. Klatte and G.F. Corliss. 1993. *C-XSC: A C++ Class Library for Extended Scientific Computing*. Springer-Verlag.

W. Krämer. 2007. Generalized Intervals and the Dependency Problem. *PAMM* 6, 1 (2007), 683–684.

William K Lam. 2005. *Hardware Design Verification: Simulation and Formal Method-Based Approaches (Prentice Hall Modern Semiconductor Design Series)*. Prentice Hall PTR.

D-U Lee, A Abdul Gaffar, Ray CC Cheung, Oskar Mencer, Wayne Luk, and George A Constantinides. 2006. Accuracy-Guaranteed Bit-Width Optimization. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on* 25, 10 (2006), 1990–2000.

Dong-U Lee, Altaf Abdul Gaffar, Oskar Mencer, and Wayne Luk. 2005. MiniBit: Bit-Width Optimization via Affine Arithmetic. In *Proceedings of the 42nd annual Design Automation Conference*. ACM, 837–840.

P. Linz. 1970. Accurate Floating-Point Summation. *Commun. ACM* 13, 6 (1970), 361–362.

D.C. Liu and J. Nocedal. 1989. On the Limited Memory BFGS Method for Large Scale Optimization. *Mathematical programming* 45, 1 (1989), 503–528.

M.A. Malcolm. 1971. On Accurate Floating-Point Summation. *Commun. ACM* 14, 11 (1971), 731–736.

David Monniaux. 2008. The pitfalls of verifying floating-point computations. *ACM Transactions on Programming Languages and Systems (TOPLAS)* 30, 3 (2008), 12.

J.J. Moré and D.J. Thuente. 1994. Line Search Algorithms with Guaranteed Sufficient Decrease. *ACM Transactions on Mathematical Software (TOMS)* 20, 3 (1994), 286–307.

Jean-Michel Muller, Nicolas Brisebarre, Florent de Dinechin, Claude-Pierre Jeannerod, Vincent Lefevre, Guillaume Melquiond, Nathalie Revol, and Damien Stehle. 2009. *Handbook of Floating-Point Arithmetic*.

G. Necula, S. McPeak, S. Rahul, and W. Weimer. 2002. CIL: Intermediate Language and Tools for Analysis and Transformation of C Programs. In *Compiler Construction*. Springer, 209–265.

Thi Viet Nga Nguyen and François Irigoin. 2005. Efficient and effective array bound checking. *ACM Transactions on Programming Languages and Systems (TOPLAS)* 27, 3 (2005), 527–570.

William G Osborne, Ray CC Cheung, José Gabriel F Coutinho, Wayne Luk, and Oskar Mencer. 2007. Automatic Accuracy-Guaranteed Bit-Width Optimization for Fixed and Floating-Point Systems. In *Field Programmable Logic and Applications, 2007. FPL 2007. International Conference on*. IEEE, 617–620.

Douglass Stott Parker. 1997. *Monte Carlo Arithmetic: exploiting randomness in floating-point arithmetic*. Computer Science Department, University of California.

Douglass Stott Parker. 2003. Monte Carlo Arithmetic. (Oct. 2003). http://www.cs.ucla.edu/∼stott/mca/.

William H Press, Saul A Teukolsky, William T Vetterling, and Brian P Flannery. 2007. *Numerical recipes 3rd edition: The art of scientific computing*. Cambridge university press.

Theodore J Rivlin. 1990. Chebyshev Polynomials. From Approximation Theory to Algebra and Number Theory. *Pure Appl. Math.(NY)* (1990).

Howard H Rosenbrock. 1960. An Automatic Method for Finding the Greatest or Least Value of a Function. *Comput. J.* 3, 3 (1960), 175–184.

A Sandu. November 2000. *CSE-690 Home Project 2*. www.cs.vt.edu/∼asandu/Courses/MTU/CSE690/proj-2.ps.

M.J. Schulte and Jr. Swartzlander, E.E. 2000. A Family of Variable-precision Interval Arithmetic Processors. *Computers, IEEE Transactions on* 49, 5 (may 2000), 387 –397.

Pat H Sterbenz. 1974. *Floating-Point Computation*. Vol. 26. Prentice-Hall Englewood Cliffs, NJ.

J.E. Stine and M.J. Schulte. 1998. A Combined Interval and Floating Point Multiplier. In *VLSI, 1998. Proceedings of the 8th Great Lakes Symposium on*. 208 –215.

Bonnie Toy and Will Menninger. 1994. C Implementation of the LINPACK Benchmark. (Feb. 1994). http://www.netlib.org/benchmark/linpackc.new

J. Vignes. 1996. A Stochastic Approach to the Analysis of Round-off Error Propagation. A Survey of the CESTAC Method. In *Proc. 2nd Real Numbers and Computers conference*. 233–251.

GW Walster and D. Chiriaev. 2000. Interval Arithmetic Programming Reference: Forte TM Workshop 6 Update 1 C++. *Sun Microsystems Inc* (2000).

James H. Wilkinson. 1994. *Rounding Errors in Algebraic Processes*. Dover Publications, Incorporated.

JHC Yeung, EFY Young, and PHW Leong. 2011. A Monte-Carlo Floating-Point Unit for Self-validating Arithmetic. In *Proceedings of the 19th ACM/SIGDA International Symposium on Field Programmable Gate Arrays*. 199–208.

Received July 2013; revised March 2014; accepted June 2014