Bounding Variable Values and Round-off Effects using Handelman Representations

David Boland and George A. Constantinides, *Senior Member, IEEE*

***Abstract*—The precision used in an algorithm affects the error and performance of individual computations, the memory usage and the potential parallelism for a fixed hardware budget. This paper describes a new method to determine the minimum precision required to meet a given error specification for an algorithm that consists of the basic algebraic operations. Using this approach, it is possible to significantly reduce the computational word-length in comparison to existing methods, and this can lead to superior hardware designs. We demonstrate the proposed procedure on an iteration of the conjugate gradient algorithm, achieving proofs of bounds that can translate to global word-length savings ranging from a few bits to proving the existence of ranges that must otherwise be assumed to be unbounded when using competing approaches. We also achieve comparable bounds to recent literature in a small fraction of the execution time, with greater scalability.**

***Index Terms*—Numerical Analysis, Performance Analysis and Design Aids, Algorithms implemented in hardware, Optimization.**

I. Introduction

NUMBER systems are usually restricted to some finite precision, and as a result, rounding will often occur so as to represent values using the chosen number system. Whilst the error introduced by the rounding of any single value may be small, over the course of an algorithm the accumulation of these errors can cause a significant deviation from the nominal result, and this could impact issues such as the convergence of a computation.

Given that these rounding errors are dependent on the precision used in the number system, users will often use a higher precision than necessary to avoid problems resulting from the accumulation of round-off error. However, it is important to have a method to quantify the precision necessary, given that an increase in precision will imply a decrease in the performance of the hardware, an increase in the data movement and an increase in memory use. As an example, recent figures for the difference in performance, in terms of peak theoretical FLOPs, between single and double precision is approximately a factor of 1 to 2 for a CPU [1], 9 for a graphics processing unit (GPU) [2] and 14 for the IBM Cell multiprocessor [3]. As well as this performance hit, the memory use doubles when moving from single to double precision, and any data transfer will take twice as long.

For hardware platforms such as FPGAs or ASICs, the choice of precision will affect far more factors: the silicon area, clock speed, latency, memory use and data transfer. These factors, especially silicon area which has implications on the potential for parallelism, can have a great impact on performance. As it is possible to use a number system with any precision on these platforms, the effect of fine-tuning an algorithm taking into account precision has been examined in several areas ranging from DSP [4], [5] to control theory [6], with the latter paper reporting performance gains of up to a factor of 36.

Unfortunately, such fine-tuning is often difficult because simulation-based methods cannot guarantee the given error estimate and there are no known methods that can tractably calculate tight bounds for the error or range of any variable within an algorithm, given that they are affected by both input ranges and finite precision effects. Consequently, analytical tools to estimate this error tradeoff quality of bounds for execution time, and the existing methods currently lie at the extreme of this spectrum.

This paper describes a new method which we show to be capable of calculating bounds on the range or relative error for a variable in an algorithm consisting of the basic algebraic operations ($\odot \in \{+, -, *, /\}$) that are, in general, tighter in comparison to traditional methods at the cost of execution time and scalability for large algorithms, and bounds that are comparable to more recent literature in an execution time that is orders of magnitude smaller. The motivation for the focus on floating-point hardware is partially due to the relative lack of existing work discussing word-length optimisation for floating-point designs in comparison to fixed-point, and partially due to the recent trends showing that floating-point designs are becoming highly efficient in hardware [7], and the growing collection of publications of floating-point custom hardware implementations [8], [9]. We note, however, that the background theory and heuristic described in this work could easily adapted to word-length optimisation for fixed-point designs by using a different model of error.

We also note that this tool could be of value in proving whether it is safe to move to single precision for use on alternative hardware accelerators, such as GPUs, but the relative lack of control over the precision in these devices, except at a coarse level, leads us to present results here for FPGA architecture. This paper elaborates on previous work by the authors published in [10] with a more detailed description of the search heuristic, several new tests to illustrate the various contributions of this work, as well as new comparisons against existing literature, and a broader discussion of its potential for word-length optimisation and limitations. A summary of the main contributions of this work are as follows:

- the description a new method, based upon a result from real algebra, to find provable bounds for any variables within a sizeable computational kernel given input data ranges and a precision specification,
- results demonstrating that applying our approach to an

example (the conjugate gradient algorithm) generally leads to tighter bounds which translate to global word-length reductions in comparison to traditional analytical approaches,
- results that demonstrate our approach can calculate bounds which are comparable to a sophisticated existing solver for range analysis [11]–[13], in a small fraction of the compute time,
- a discussion of the limitations of our approach and its scalability relative to existing approaches.

This paper is organised as follows. It first discusses in detail the current literature regarding the methods that are used to calculate error bounds for use in word-length optimisation in Section II. The new approach, along with a description of the source of floating-point error and a general method to represent this error using polynomials, as well as some relevant background theory, is then put forward in Section III. Section IV discusses the methods to test the results, which are then shown in the following section, Section V. Finally, Section VI discusses the conclusions of this work and its limitations that must be addressed in the future.

II. Background

Due to the potential benefits on silicon area, clock speed and power that can be obtained by optimising the precision used throughout a circuit, there exists a large amount of literature focused upon word-length optimisation, summaries of which can be found in [14], [15], as well as optimisation strategies to create mixed fixed and floating point precision designs [13], and a class of work in practical tools for precision analysis [16]. One fundamental aspect of these solvers is that they require a method which can both validate that overflow will not occur for a chosen word-length and calculate the cost of the chosen word-length in terms of the error observed in the final computational result. It is interesting that despite the wealth of optimisation strategies, only a handful of methods to perform this function. For comparative purposes, these methods: simulation, interval arithmetic, affine arithmetic, and more recently, satisfiability modulo theories, will be elaborated in this section. Our approach is not based on any of these schemes, but rather on results from real algebra which will be introduced in Section III.

The most straightforward way to estimate an error is through simulation. The aim of any simulation-based approach is to find the inputs which will cause the extreme ranges of the data set. Unfortunately, the size of the search space for the inputs will generally be too large to explore exhaustively, and hence the simulation will either require random sampling of the input data space [17], statistical profiling [18] or be based upon a representative training data set [5], [6]. However, whilst the quality of the estimate can be improved by increasing the size of the training set or the search time, in either case, the estimate does not form a bound because corner cases can be missed. We also note that while it is possible to use methods to avoid precision errors *a posteriori* at run-time, at a cost of execution time [19], our goal is to calculate bounds *a priori* so as to design hardware with the minimum precision.

To calculate true bounds for general algorithms, traditionally there have only been two main analytical approaches: interval arithmetic (IA) [20] or affine arithmetic (AA) [21]. Interval arithmetic represents every value as lying within some interval $[x_1, x_2]$, where x_1 and x_2 are the lower and upper bounds respectively. The intervals are then propagated through the computation according to basic rules, given in (1), which calculate at each stage the new worst case bound. In order to create safe bounds, the intervals will also have to be outwardly rounded to the nearest number representable in the chosen number system at each stage.

$$
\begin{aligned}
&[x_1; x_2] + [y_1; y_2] = [x_1 + y_1; x_2 + y_2] \\
&[x_1; x_2] - [y_1; y_2] = [x_1 - y_2; x_2 - y_1] \\
&[x_1; x_2] \times [y_1; y_2] = [\min(x_1y_1, x_2y_1, x_1y_2, x_2y_2); \\
&\qquad\qquad \max(x_1y_1, x_2y_1, x_1y_2, x_2y_2)] \\
&\frac{[x_1; x_2]}{[y_1; y_2]} = \begin{cases} \textit{undefined} & \text{if } 0 \in [y_1; y_2] \\ [\min\left(\frac{x_1}{y_1}, \frac{x_1}{y_2}, \frac{x_2}{y_1}, \frac{x_2}{y_2}\right); & \text{otherwise} \\ \quad \max\left(\frac{x_1}{y_1}, \frac{x_1}{y_2}, \frac{x_2}{y_1}, \frac{x_2}{y_2}\right)] & \end{cases}
\end{aligned}
\tag{1}
$$

However, interval arithmetic suffers from the so-called dependency problem, where if the same variable is used twice, information is lost. A trivial example is the following: for a variable x which lies in the interval $[x_1, x_2]$, perform the operation $x - x$. The interval should be $[0, 0]$, but the result using interval arithmetic would be $[x_1 - x_2, x_2 - x_1]$. Several simple examples can demonstrate how this problem may cause bounds that are significantly wider than the tightest bounds [22]. As a result of these problems, there is an active community of researchers in *robust computing* who have developed ways to mitigate this problem. One approach is to attempt to modify algorithms to make them 'interval arithmetic friendly' at the cost of computational complexity or average case numerical robustness [23]. However, whilst such approaches are useful to obtain reliable proofs using IA, the modifications to the algorithm do not necessarily improve the true numerical stability, instead the modifications simply reduce the sensitivity to which IA bounds this error. Ideally, it is preferable to find proofs of tighter bounds for the unmodified algorithm, unless it can be proven that the numerical properties of the modified algorithm have been improved.

For this reason, some methods that reduce dependencies whilst avoiding modifying the algorithm should be mentioned. The first of these uses Taylor models with interval remainder bounds [24]. This method represents any function over a set of bounded variables by a Taylor model of order ρ, T_ρ, and an interval remainder term that bounds the remaining higher order terms in the Taylor series, I_ρ. Operations on functions in this form $(T_\rho + I_\rho)$ are initially performed symbolically, before using interval arithmetic to evaluate any of the resultant terms involving the interval remainders and using an appropriate method to bound the new terms that are of degree greater than ρ, such as the Lagrange remainder [25]. After repeating this process throughout an algorithm, the bounds for the final function could be found by applying interval arithmetic to the symbolic variables of this function and adding the interval remainder bound. This method also has the advantage of being

able to handle complex functions such as division, sine, cosine and logarithms by using Taylor formulas, but unfortunately, the complexity of computing the Taylor series approximation for these functions grows exponentially in ρ. Furthermore, whilst propagating the variables symbolically may reduce the effect of the dependency problem as it allows cancellations, the final polynomial is likely to still involve many dependencies and must still be bounded using interval arithmetic, and any terms involving the interval remainder bounds will still suffer from the dependency problem in the same way as interval arithmetic in its traditional form. To illustrate this point, one can consider that interval arithmetic in its traditional form is equivalent to Taylor models where $\rho = 0$. The second tool based upon reducing the number dependencies works by converting the interval constraints throughout an algorithm into logical propositions and attempting to remove dependencies by performing standard logical manipulations internally before finally finding the tightest bounds for which these propositions still hold according to the rules of interval arithmetic [26]. However, it is important to note that whilst both these approaches may reduce the number of dependencies, they do not necessarily remove all dependencies. This means that since both approaches then require interval arithmetic to find the final bounds, they are both still subject to the limitations of interval arithmetic, and hence they will often fail to find the tightest bounds.

Affine arithmetic is a method which mitigates the dependency problem. It works by representing every variable in an affine form given by (2) consisting of a known central value (x_0), coefficients of known value (x_i) and noise symbols (ϵ_i) that are only known to lie in the interval $[-1, 1]$:

$$x = x_0 + x_1\epsilon_1 + x_2\epsilon_2 + ... + x_n\epsilon_n. \tag{2}$$

Affine arithmetic then performs all operations on these coefficients, ensuring the result is also in affine form. The problem with affine arithmetic is that many functions, including general multiplication, are not affine and hence approximations must be made. Methods to perform these approximations can trade the size of the error for computational complexity [27], but in all cases, there will still be a widening of the derived bounds.

Given the limitations of interval and affine arithmetic, recently, a new approach has been published which uses Satisfiability-Modulo Theories (SMT) [11]–[13] to refine the bounds given by interval or affine arithmetic by searching for a set of inputs breaking the bounds, using a Satisfiability solver. This bound is iteratively refined depending upon the results of this test, using a binary search method. The main problem with this approach is that conditions the SAT solver checks within the inner loop of this method are created by propagating constraints using interval arithmetic and splitting the input intervals to improve the bounds [28]. Though interval splitting can significantly improve bounds, using a basic approach of applying n_s splits to every variable of a problem consisting of n variables, the number of intervals to be evaluated grows $O(n_s^n)$, ensuring this approach is not scalable in its current form. This issue in general limits the use of global optimisation strategies which use interval analysis because they also rely on interval splitting to obtain tighter bounds [29].

This run-time issue limits the SMT approach to consider only the so called 'range analysis' problem which involves ensuring that over the range of input data, there is sufficient dynamic range to prevent overflow. However, when optimising word-lengths, determining the range as a result of the inputs is only a part of the problem; it is also important to perform 'precision analysis', which is typically described as ensuring that the error at the output, caused by the use of a finite precision, lies below a threshold. As such, several of the optimisation strategies do perform range analysis as a first step before precision analysis [15], but these are all targeted towards DSP systems which do not include division. However, we argue that for a word-length optimiser targeted towards a general algorithm, these two problems cannot be viewed independently because the range can be heavily affected by the errors caused by the use of finite precision. This is typically because these errors may cause a divisor to approach closer to zero than under infinite precision, resulting in a larger range; such effects will be seen in Section V-A. Combining range and precision analysis in a single word-length optimiser can therefore be seen as an added benefit of our approach because it allows us to choose the minimum exponent width that will span the desired range and ensure that the hardware will not overflow and also use less silicon area.

This work describes a new general analytical approach which can provide provable bounds. It is argued in this paper that this method can achieve significantly tighter bounds than both interval and affine arithmetic, while running significantly faster, with better scalability, than the SMT approach.

III. Proposed Algorithm

The aim of this work is, for a given floating-point precision, to find bounds on the value of any chosen variable within an algorithm, and hence bound the worst case computational error induced. The suggested method to achieve this consists of several stages. The first stage involves creating a polynomial for the variable of interest, a function of new bounded variables, each representing round-off errors introduced after a specific operation. This polynomial is then simplified into a canonical form, from which bounds for the extrema of the polynomial can be found algorithmically. An optional final stage extends the approach for polynomials to rational functions, allowing all algorithms consisting of the basic operators $\{+, -, *, /\}$ to be automatically analysed.

Notation: To formalise the discussion of the method, some simple notation is used. We consider polynomials in n variables $\delta_1, \delta_2, ..., \delta_n$. We use the notation $\boldsymbol{\delta}^{\boldsymbol{\lambda}}$ for a term, which is a product of the variables raised to some integer powers (3), where $\boldsymbol{\lambda}$ is a vector collecting the exponents (4). We denote by $|\boldsymbol{\lambda}|$ the *degree* of the term, given by (5).

$$\boldsymbol{\delta}^{\boldsymbol{\lambda}} = \delta_1^{\lambda_1}\delta_2^{\lambda_2}...\delta_n^{\lambda_n}. \tag{3}$$

$$\boldsymbol{\lambda} = (\lambda_1, ..., \lambda_n), \text{where } \lambda_i \in \mathbb{N}. \tag{4}$$

$$|\boldsymbol{\lambda}| = \lambda_1 + ... + \lambda_n. \tag{5}$$

A monomial is defined as a term multiplied by some real coefficient, *i.e.* $c\boldsymbol{\delta}^{\boldsymbol{\lambda}}$, and a polynomial is the sum of one or more monomials.

Finally, to allow a formal description in the following sections, in this work the terms in a polynomial are ordered. $\boldsymbol{\delta}^{\boldsymbol{\mu}} < \boldsymbol{\delta}^{\boldsymbol{\lambda}}$ denotes that $\boldsymbol{\delta}^{\boldsymbol{\mu}}$ precedes $\boldsymbol{\delta}^{\boldsymbol{\lambda}}$, according to degree lexicographical order, as described in (6) [30].

$$\boldsymbol{\delta}^{\boldsymbol{\mu}} < \boldsymbol{\delta}^{\boldsymbol{\lambda}} \Leftrightarrow \begin{cases} |\boldsymbol{\mu}| < |\boldsymbol{\lambda}|, \\ \text{or} \\ |\boldsymbol{\mu}| = |\boldsymbol{\lambda}| \text{ and } \exists i(\mu_i < \lambda_i \text{ and } \forall j < i(\mu_j = \lambda_j)) \end{cases} \tag{6}$$

A. Creating a Polynomial Representation of Potential Range

It can be shown that for a real value x, the closest radix-2 floating-point approximation $\hat{x}$ of x can be expressed as in (7) [31], where η is the number of mantissa bits used (referred to as the precision), provided there is no underflow (note our approach could easily be extended to support any radix by changing this equation). As mentioned in the background section, our approach can ensure underflow will not occur. It is similarly possible to specify that the radix-2 floating-point result of any scalar operation ($\odot \in \{+, -, *, /\}$) is bounded as in (8), provided the exponent is sufficiently large to span the range of the result. Operations complying with IEEE standard arithmetic exhibit this behaviour.

$$\hat{x} = x(1 + \delta_1) \qquad (|\delta_1| \leq \Delta, \text{where } \Delta = 2^{-\eta}). \tag{7}$$

$$\widehat{x \odot y} = (x \odot y)(1 + \delta_1). \tag{8}$$

In our approach, a simple compiler takes input pseudo code and applies this model of floating-point error on the result of every computation throughout an algorithm such that every output variable can be represented by a single polynomial in all the error variables, as shown for a simple example in Table I. We currently focus on straight-line code algorithms consisting of $\{+, -, *, /\}$ operators, meaning that we unroll any loops, which is a reasonable approach for real-time computation where the loop bounds are known. Any conditional statements do not directly affect our bounds procedure if they do not operate on any of the rounded variables.

TABLE I
CONSTRUCTION OF POLYNOMIALS

x, y are inputs Δ is the error bound determined by the precision, so that $\|\delta_i\| \leq \Delta$	
Pseudo Code	Polynomial Representation of Variable Value
a = x*y;	a = $xy(1 + \delta_1)$
b = a*a;	b = $(xy(1 + \delta_1))^2(1 + \delta_2)$
c = b-a;	c = $[(xy(1 + \delta_1))^2(1 + \delta_2) - xy(1 + \delta_1)](1 + \delta_3)$

B. Minimising the polynomial

Given a polynomial representing the value of a variable, as in Table I, $f(\boldsymbol{\delta})$, we want to find $\gamma_{lower} = \inf_{|\delta_i| \leq \Delta} f(\boldsymbol{\delta})$, the lower bound on the variable or function of intent, and $\gamma_{upper} = \sup_{|\delta_i| \leq \Delta} f(\boldsymbol{\delta})$. Unfortunately this is a non-convex optimisation problem, which is NP-hard, meaning traditional approaches are unsuitable. For example, calculus style approaches towards bounding a polynomial of order ρ by searching for turning points are unsuitable because finding the solutions for $f'(\boldsymbol{\delta}) = 0$ for a general multivariate polynomial is also NP-hard. Alternatively, an approach such as finding zeros by Bernstein Polynomials, has been shown to have computational complexity of $O(n\rho^{n+1})$ for a polynomial of order ρ consisting of n variables [32]. As a result, we focus on finding a computationally tractable lower bound $\hat{\gamma}_{lower} \leq \gamma_{lower}$ and upper bound $\hat{\gamma}_{upper} \geq \gamma_{upper}$.

In this work, a new approach is taken to find bounds where $\gamma_{lower} - \hat{\gamma}_{lower}$ and $\hat{\gamma}_{upper} - \gamma_{upper}$ are as small as possible. Here we first describe the background theory, which is based upon a result from real algebra discovered by Handelman, after which a new heuristic based upon this theory is proposed.

Theorem 1 ([33]). *A polynomial $p(x)$ is non-negative over the compact set of linear inequalities $g_i \geq 0$, i.e. $S = \{x \in \mathbb{R}^n | g_i(x) \geq 0\}$, if and only if p has a* Handelman representation *of the form* (9).

$$p = \sum_{\alpha \in \mathbb{N}^n} c_\alpha \prod_{i=1}^{n} g_i^{\alpha_i}, \tag{9}$$

where each c_α is a non-negative constant and $\mathbb{N}$ is the set of natural numbers.

This theorem can be applied to find lower and upper bounds to satisfy $\hat{\gamma}_{lower} \leq f(\boldsymbol{\delta}) \leq \hat{\gamma}_{upper}$ by considering that we are trying to show that the functions $f(\boldsymbol{\delta}) - \hat{\gamma}_{lower}$ and $\hat{\gamma}_{upper} - f(\boldsymbol{\delta})$ are non-negative over the compact set of inequalities specifying the bounds on $\boldsymbol{\delta}$ given in (10). By Theorem 1, this is equivalent to showing $f(\boldsymbol{\delta}) - \hat{\gamma}_{lower}$ has a Handelman representation of the form (11), or similarly satisfying (12) for the upper bound. We note the number of constants c_α in the summation in (9) is unbounded because the Handelman representation is only guaranteed to converge as the number of α vectors tends to infinity [33]. However, when using this theory to search for bounds, a practical approach to find such a representation, such as linear programming, restricts the number of constants to a finite amount [34], at the cost of potentially not finding the optimal bound.

$$S = \{\boldsymbol{\delta} \in \mathbb{R}^n | \forall i(\Delta - \delta_i \geq 0) \wedge (\Delta + \delta_i \geq 0)\}. \tag{10}$$

$$f(\boldsymbol{\delta}) - \hat{\gamma}_{lower} = \sum_{\alpha,\beta \in \mathbb{N}^n \times \mathbb{N}^n} c_{\alpha,\beta} \prod_{i=1}^{n} (\Delta - \delta_i)^{\alpha_i} (\Delta + \delta_i)^{\beta_i}. \tag{11}$$

$$\hat{\gamma}_{upper} - f(\boldsymbol{\delta}) = \sum_{\alpha,\beta \in \mathbb{N}^n \times \mathbb{N}^n} c_{\alpha,\beta} \prod_{i=1}^{n} (\Delta - \delta_i)^{\alpha_i} (\Delta + \delta_i)^{\beta_i}. \tag{12}$$

For our purposes, it will be easier to work with a generalised version of the Handelman representation that we propose, given in (13), which we refer to as a *Generalised Handelman Representation* (GHR).

Theorem 2. *A polynomial p_{ghr} is non-negative over the compact set S, defined in* (10)*, if and only if p_{ghr} can be represented by a* Generalised Handelman Representation *of the form* (13).

$$p_{ghr} = \sum_{j\in\mathbb{N}} c_j \prod_{i=1}^{n} (\Delta^{|\boldsymbol{\mu_{i,j}}|} - \boldsymbol{\delta}^{\boldsymbol{\mu_{i,j}}})^{\alpha_{i,j}} (\Delta^{|\boldsymbol{\mu_{i,j}}|} + \boldsymbol{\delta}^{\boldsymbol{\mu_{i,j}}})^{\beta_{i,j}},$$
where each $\mu_{i,j}$ is an arbitrary integer vector.
and each $c_j, \alpha_{i,j}, \beta_{i,j}$ is a non-negative constant. (13)

Proof:
$\Rightarrow$: If p_{ghr} is non-negative over the compact set S, then it has a Handelman representation; by choosing $\mu_{i,j} = i$, it also has a GHR.
$\Leftarrow$: If p_{ghr} has a GHR, then because any individual variable δ_i is bounded by $|\delta_i| \leq \Delta$, any term can also be bounded over the set S, $|\boldsymbol{\delta}^{\boldsymbol{\lambda}}| \leq \Delta^{|\boldsymbol{\lambda}|}$. This means $\Delta^{|\boldsymbol{\lambda}|} + \boldsymbol{\delta}^{\boldsymbol{\lambda}} \geq 0$ and $\Delta^{|\boldsymbol{\lambda}|} - \boldsymbol{\delta}^{\boldsymbol{\lambda}} \geq 0$, and because $c_i \geq 0$, it holds that $p_{ghr} \geq 0$, *i.e.* non-negative over the set S. ■

Using this theorem, if we can find GHRs to satisfy (14) and (15), then the left-hand side is non-negative over the set of inequalities (10), and from this a guaranteed bound follows.

$$f(\boldsymbol{\delta}) - \hat{\gamma}_{lower} = p_{lower\_ghr}. \tag{14}$$
$$\hat{\gamma}_{upper} - f(\boldsymbol{\delta}) = p_{upper\_ghr}. \tag{15}$$

1) Example: In order to demonstrate the use of this theory, we will consider the function $f(\delta_1) = \delta_1^2 - \delta_1$ over the set $|\delta_1| \leq 1/2$. For this simple function of one variable, we can first derive lower and upper bounds using familiar calculus arguments. We will then demonstrate that IA is unable to calculate the same bounds, before finally showing that GHRs can be used to prove the ideal bounds.

Using calculus to calculate bounds of $f(\delta_1) = \delta_1^2 - \delta_1$, we know that because this is a convex function, the minimum will lie where the derivative $f'(\delta_1) = 0$ and the maximum will lie at one of the extremes of the range of δ_1. Thus by differentiating $f(\delta_1)$ to get $f'(\delta_1) = 2\delta_1 - 1$, we find the minimum lies at $\delta_1 = 1/2$, leaving the maximum to be where $\delta_1 = -1/2$; this gives the range of $f(\delta_1)$ to be $[-1/4, 3/4]$. In comparison, interval arithmetic is unable to calculate the optimal lower bound, as shown in (16).

$$\begin{aligned} \delta_1 &\in [-1/2, 1/2] \\ \Rightarrow \delta_1^2 &\in [0, 1/4] \\ \Rightarrow \delta_1^2 - \delta_1 &\in [-1/2, 3/4] \end{aligned} \tag{16}$$

To find bounds using the theory presented in this section, we want to search for GHRs to satisfy (14) and (15). Two such GHRs are $p_{lower\_ghr} = (1/2 - \delta_1)^2$ and $p_{upper\_ghr} = (1/2 - \delta_1)(1/2 + \delta_1) + (1/2 + \delta_1)$. After equating these respective functions, as shown in (17) and (18), we find $\hat{\gamma}_{lower} = -1/4$ and $\hat{\gamma}_{upper} = 3/4$. Finally, by Theorem 2, we now know that $f(\delta_1) - (-1/4) \geq 0$ and $3/4 - f(\delta_1) \geq 0$ or that $-1/4 \leq f(\delta_1) \leq 3/4$, thus recovering the optimal bounds.

$$\begin{aligned} \delta_1^2 - \delta_1 - \hat{\gamma}_{lower} &= (1/2 - \delta_1)^2 \\ \delta_1^2 - \delta_1 - \hat{\gamma}_{lower} &= 1/4 - \delta_1 + \delta_1^2 \\ \hat{\gamma}_{lower} &= -1/4 \end{aligned} \tag{17}$$

$$\begin{aligned} \hat{\gamma}_{upper} - (\delta_1^2 - \delta_1) &= (1/2 - \delta_1)(1/2 + \delta_1) + (1/2 + \delta_1) \\ \hat{\gamma}_{upper} - \delta_1^2 + \delta_1 &= 1/4 - \delta_1^2 + 1/2 + \delta_1 \\ \hat{\gamma}_{upper} &= 3/4 \end{aligned} \tag{18}$$

2) Finding GHRs: Unfortunately, in general it is difficult to find GHRs to satisfy equations (14) and (15) and ensure that the calculated bounds for $\hat{\gamma}_{lower}$ and $\hat{\gamma}_{upper}$ are as tight as possible. The existing method of computing Handelman Representations is to use linear programming [34]. This is achieved by setting the objective to either $\min(\hat{\gamma}_{lower})$ or $\max(\hat{\gamma}_{upper})$, and creating a single linear constraint for each monomial in p_{lower_ghr} or p_{upper_ghr}, to match the corresponding monomial in the desired polynomial f. The linear program would then be able to calculate all the values for the variables c_j, and the desired bound γ. It is important to note at this stage that using this method, it is impossible to guarantee the global minimum will be found, since Handelman representations are only guaranteed to converge to the global optimum as the maximum order tends to infinity [34]. Therefore, in order to ensure we get a result in tractable time, we must first restrict the maximum order for p_{ghr} to some value ρ. However, whilst choosing this value of ρ will trade execution for quality of bounds, the value of ρ must be greater than or equal to the degree of f in order to satisfy (14) or (15).

Upon formulating a linear program using this method, the scalability quickly becomes a significant problem. For an arbitrary polynomial f of order ρ consisting of n variables, the number constraints is $\binom{\rho+n}{n}$ and variables is $\binom{\rho+2n}{2n}$. Clearly, whilst this is efficient for small problems, the size of the linear program quickly grows too large for existing linear programming tools. For example, consider a problem consisting of approximately $n = 30$ variables where the maximum order of f is restricted to $\rho = 6$. This would consist of almost 2 million constraints and over 90 million variables, which is unsolvable using current LP solvers.

Due to this limitation of linear programming, in this work we propose a new heuristic which is guaranteed to terminate at the same time as aiming to find practically useful bounds.

3) Our approach: The proposed algorithm to bound a polynomial is based upon finding GHRs to satisfy equations (14) and (15), similar to the earlier example. To achieve this, we first express $f(\boldsymbol{\delta})$ in a canonical form, as a sum of monomials in which each term appears at most once. In order to demonstrate this step, consider the earlier example from Table I; the polynomial representation for the variable c in this example can be expanded into the polynomial (19) when neglecting the variable δ_3, since the worst case value of this variable is trivially known to lie at the extremes.

$$\begin{aligned} f(\boldsymbol{\delta}) = -xy(xy-1) - xy(2xy-1)\delta_1 - x^2y^2\delta_2 \\ -x^2y^2\delta_1^2 - 2x^2y^2\delta_1\delta_2 - x^2y^2\delta_1^2\delta_2 \end{aligned} \tag{19}$$

After this expansion, it is possible to 'cancel' each individual monomial from the left hand side of equations (14) and (15), using a polynomial of the form (20), and we then note that the sum of polynomials created in this fashion would be a GHR. After cancelling all the monomials, we would be left with a constant from which the bound $\hat{\gamma}_{lower}$ and $\hat{\gamma}_{upper}$ could be derived in a similar fashion to the earlier example.

$$h(\boldsymbol{\delta}) = c \prod_{j=1}^{n} (\Delta^{|\boldsymbol{\mu_j}|} - \boldsymbol{\delta}^{\boldsymbol{\mu_j}})^{\alpha_j} (\Delta^{|\boldsymbol{\mu_j}|} + \boldsymbol{\delta}^{\boldsymbol{\mu_j}})^{\beta_j}. \tag{20}$$

TABLE II
EXAMPLE POLYNOMIALS OF THE FORM (13) TO CANCEL THE MONOMIAL $-x^2y^2\delta_1^2\delta_2$ FROM f.

Approach Number	1	2	3	4
Handelman Coefficients	$\mu = ([1,0],[0,1])$ $\alpha = (0,0)$ $\beta = (2,1)$ $c = x^2y^2$	$\mu = ([1,0],[0,1])$ $\alpha = (2,0)$ $\beta = (0,1)$ $c = x^2y^2$	$\mu = ([2,0],[0,1])$ $\alpha = (0,0)$ $\beta = (1,1)$ $c = x^2y^2$	$\mu = [2,1]$ $\alpha = 0$ $\beta = 1$ $c = x^2y^2$
Polynomial h	$x^2y^2(\Delta+\delta_1)^2(\Delta+\delta_2)$	$x^2y^2(\Delta-\delta_1)^2(\Delta+\delta_2)$	$x^2y^2(\Delta^2+\delta_1^2)(\Delta+\delta_2)$	$x^2y^2(\Delta^3+\delta_1^2\delta_2)$
New Polynomial $f+h$	$xy(xy-1-xy\Delta^3)$ $+xy(2xy-1-2xy\Delta^2)\delta_1$ $+x^2y^2(1-\Delta^2)\delta_2$ $+x^2y^2(1-\Delta)\delta_1^2$ $+2x^2y^2(1-\Delta)\delta_1\delta_2$ $+0\delta_1^2\delta_2$	$xy(xy-1-xy\Delta^3)$ $+xy(2xy-1+2xy\Delta^2)\delta_1$ $+x^2y^2(1-\Delta^2)\delta_2$ $+x^2y^2(1-\Delta)\delta_1^2$ $+2x^2y^2(1+\Delta)\delta_1\delta_2$ $+0\delta_1^2\delta_2$	$xy(xy-1-xy\Delta^3)$ $+xy(2xy-1)\delta_1$ $+x^2y^2(1-\Delta^2)\delta_2$ $+x^2y^2(1-\Delta)\delta_1^2$ $+2x^2y^2\delta_1\delta_2$ $+0\delta_1^2\delta_2$	$xy(xy-1-xy\Delta^3)$ $+xy(2xy-1)\delta_1$ $+x^2y^2\delta_2$ $+x^2y^2\delta_1^2$ $+2x^2y^2\delta_1\delta_2$ $+0\delta_1^2\delta_2$

The complexity of this approach is that there are many monomials in $f(\boldsymbol{\delta})$ to cancel, and many ways to cancel any given monomial using polynomials of the form (20). For example, consider again the polynomial given in (19), Table II illustrates several possible choices of these polynomials that could be used to cancel the highest order monomial $(-x^2y^2\delta_1^2\delta_2)$ from this example. Our proposed heuristic attempts to make the best choices of these polynomials. It is based on the idea that whilst canceling a high order monomial, it is possible to reduce the coefficients of lower order monomials at the same time, as shown in Table II; this will result in tighter bounds than cancelling each monomial independently. In order to ensure termination, detailed in Section III-C, the heuristic selects the highest order monomials and chooses polynomials to remove the higher order monomials in such a way that they also reduce the absolute value of the coefficient of lower order monomials. The overall heuristic is formally given in Figure 1; the rest of this section gives a high level discussion of the rationale behind the various stages of the heuristic.

Selecting Cancellation Terms. The first stage involves choosing which lower order monomials are suitable to be modified at the same time as attempting to cancel a higher order monomial. Even for the simple example given in (19), the choice of cancellation terms will always depend upon the input. If $2xy > 1$, the first approach from Table II would be the best of the four approaches as it would decrease the term δ_1 towards zero. On the other hand, if $2xy = 1$ the third approach would be the best of the four as the term δ_1 would already be zero and hence the first two approaches would create a new monomial δ_1, which is a low order monomial that would have to be later removed. Thus the heuristic searches for non-zero monomials in f whose product will equal the desired higher order term. It performs this search initially looking to build a set using lower order terms as these are the most desirable terms to reduce, and uses higher order terms if necessary.

Selecting a Subset of Cancellation Terms. If we were to create a monomial which reduces all the terms selected in the previous stage, the size of the canonical representation of the form (20) grows exponentially in the number of terms. As a result, a tuning factor has been added which checks if the number of terms found from the previous stage exceeds a user chosen maximum m. If the number of terms exceeds this maximum, a subset of m terms from the previous stage are chosen, and a single extra high order term is added to ensure the desired monomial still gets cancelled. This tuning factor allows the user to trade execution time for the quality of the bound, and is discussed in more detail in Section V-C.

Choosing Signs. Having chosen the terms that will be used to create the polynomial, the next stage chooses the signs. The signs are chosen to reduce as many of the chosen low order monomials as possible, as well as the original highest order term to ensure algorithm termination. For example, from Table II, if $2xy < 1$, the second approach would be best as it would decrease the coefficient of the monomial δ_1, whereas the first approach would increase this coefficient.

Choosing the initial multiplier. The final stage involves choosing the initial multiplier c for the polynomial h. This is a scalar chosen to be as large as possible whilst ensuring that when the polynomial is added to f, no coefficient in f changes sign. This ensures that at least one coefficient gets cancelled at every iteration of the algorithm.

C. Algorithm Termination

At each iteration of the loop in Figure 1, a polynomial h is formed which reduces the absolute value of the coefficient of the highest order monomial in f. The minimum reduction is determined by q, as defined in Figure 1, which is a function of terms present in both f and h. After the update, the term from f which determines this minimum reduction value is removed from f. The algorithm then repeats the process.

Termination is guaranteed because at each iteration of the loop, the absolute value of the coefficient of the highest order monomial in f is reduced by a factor q, which is a function of the absolute values of the coefficients of monomials in f and the cancellation polynomial h, and no higher order monomials are created. If new lower order terms are created, the value of q to remove these whilst cancelling the same high order term will be the same as the value of q which created them, so because q does does not decrease, eventually the highest order monomial will be cancelled, ensuring termination.

D. Division

Because Theorem 2 only applies to polynomials, the above method cannot be directly applied to algorithms including division. However, we note that any computation which consists of the basic operators $\{+,-,*,/\}$ can be converted into a rational function by first applying the multiplicative model of error and then performing simple algebraic manipulation, an example of such manipulation is shown in (21); we can then perform an iterative refinement to calculate a bound.

Algorithm $\gamma = BoundPoly(g, \Delta)$
// BoundPoly takes a polynomial g in the formal vector variable $\boldsymbol{\delta}$, and a real
// bound Δ on the absolute value of each element $\delta_i (1 \leq i \leq n)$.
// It returns a lower bound on $g(\delta)$ valid over $\boldsymbol{\delta} \in [-\Delta, +\Delta]^n$.

Set $f = -g$
while f is not constant
  Find greatest monomial in f: $c\boldsymbol{\delta^{\mu}}$

  // Selecting Cancellation Terms
  Set degree = 0;
  repeat
    degree++;
    Using a greedy search algorithm find a set S where each element is a monomial of the form $a_i \boldsymbol{\delta^{\lambda_i}}$ in f such that (letting $n = |S|$):
    $(\sum_{i=1}^{n} \boldsymbol{\lambda_i} = \boldsymbol{\mu}) \wedge \forall i(|\boldsymbol{\lambda_i}| \leq \text{degree})$
  until (S is nonempty)

  // Selecting a Subset of Cancellation Terms
  if $(n > m)$
    Form a subset $S' \subset S$ of the m lexicographically lowest monomials in S
    Add an extra monomial $c\boldsymbol{\delta^{\mu}} / \prod_{i=1}^{|S'|} \boldsymbol{\delta^{\lambda_i}}$ to S' to complete the cover
  else
    Let $S' = S$
  Let $n' = |S'|$

  // Choosing Signs
  Let $d\boldsymbol{\delta^{\beta}}$ be the lexicographically greatest monomial in S'.
  Modify the sign of this monomial by setting:
    $S' = S' \cup \{|d|\text{sgn}(c)\Pi_{i=1}^{n'-1}\text{sgn}(a_i)\boldsymbol{\delta^{\beta}}\} \setminus \{d\boldsymbol{\delta^{\beta}}\}$.
  for (each monomial $a_i \boldsymbol{\delta^{\lambda_i}}$ in S')
    Create a new polynomial $j_i = (\Delta^{|\boldsymbol{\lambda_i}|} - \text{sgn}(a_i)\boldsymbol{\delta^{\lambda_i}})$

  // Choosing the initial multiplier
  Create $h = \prod_{i=1}^{n'} j_i$
  Let C be the set of terms present both in f and in h.
  For each term $\boldsymbol{\delta^{\rho_i}}$ in C, let the corresponding coefficient in f and h be f_i and h_i respectively.
  Form $q = \min_i(|f_i|/|h_i|)$
  Compute $f = f + qh$
end
Set $\hat{\gamma} = -f$.

Fig. 1. Cancellation Algorithm.

$$\frac{\delta_1}{\delta_2 + \delta_3/\delta_4} = \frac{\delta_1 \delta_4}{\delta_2 \delta_4 + \delta_3}. \tag{21}$$

Assuming the rational representing the range of the chosen variable is of the form n/d, where n and d are polynomials, then we need to show $n/d \geq \hat{\gamma}_{lower}$ inside the compact set of intent. For $d > 0$, this could be re-written as $n - d\hat{\gamma}_{lower} \geq 0$, which is a polynomial inequality. The previous approach can then attempt to find a GHR for $n - d\hat{\gamma}_{lower}$; if a representation is found for a value of $\hat{\gamma}_{lower}$, we tighten the value $\hat{\gamma}_{lower}$, if it fails then one can loosen the value $\hat{\gamma}_{lower}$. A similar approach can be taken for the upper bound.

In order to minimise this search time, this is performed as a binary search with the initial range given by equation (22), where the ranges of n and d are found using the original method on the numerator and denominator polynomials alone. Using this range also has the added benefit of checking that the denominator does not include the zero value. Note that if the denominator does contain a zero not cancelled by a zero in the numerator, then no such bound exists in any case.

$$\begin{aligned}[\min(&\frac{n_{min}}{d_{max}}, \frac{n_{max}}{d_{min}}, \frac{n_{max}}{d_{max}}, \frac{n_{min}}{d_{min}}),\\ &\max(\frac{n_{min}}{d_{max}}, \frac{n_{max}}{d_{min}}, \frac{n_{max}}{d_{max}}, \frac{n_{min}}{d_{min}})].\end{aligned} \tag{22}$$

E. Combining Handelman representations with existing approaches to bound error

The main benefit of our approach is that it can take into account dependencies between variables within a polynomial when calculating bounds, unlike interval arithmetic. Since affine arithmetic (when bounding non-affine terms), Taylor series with remainder bounds (when bounding the final polynomial as well as high order terms and any actions with the remainder bounds), and global optimisation methods using interval analysis, including the inner loop of SMT (inside the HySAT solver), are all dependent upon interval arithmetic to some degree, replacing interval arithmetic by our approach within these methods could potentially improve the final bounds, albeit at the cost of increased execution time. However, as affine arithmetic and Taylor series with remainder bounds control the scalability by controlling the size of the polynomial, the size of this increase in execution time would also be controlled. Furthermore, a combined method would be able to handle non-polynomial functions such as square roots and exponentials. However, in the remainder of this paper, we have decided to focus on the algorithm described in this section so as to quantify its benefits and limitations directly with the existing methods.

IV. Testing methodology

In order to characterise the performance of this work, it has been compared over several tests using a variety methods. The test cases are those described in the comparative works [11], [12], [27] as well as an iteration of the conjugate gradient algorithm [35] applied to a 'toy' matrix of order two (the operations given in Figure 2); the interest behind the choice of the conjugate gradient algorithm is to demonstrate our approach can be applied to an example of a real algorithm that is used in finding the solution to a system of linear equations. Though it is acknowledged that for such a small matrix order, the conjugate gradient algorithm is unlikely to be used, it is large in terms of total operations compared to the results reported in [11], [12], [27], and it provides a simple real example on the effects of limited precision computations as well the highlighting the scalability issues. The inputs for the test are shown in Figure 2, these are chosen to ensure the matrix is symmetric positive definite, a property required for the convergence of the conjugate gradient algorithm, whilst the input vector is specified as an interval to demonstrate the algorithm can calculate bounds dependent upon input ranges and finite precision. As mentioned in Section II, the test cases from the comparative works are based on the range analysis problem as opposed to the effects of finite precision, but these are included to demonstrate that our approach is also applicable to this problem and performs well in comparison. We then illustrate the greater scalability of our approach relative to these comparative works by demonstrating that it can still find bounds when adding finite precision effects to these problems.

The methods to which our work is compared help to determine where our approach lies within a hierarchy of error bounds, given in Figure 3, where the higher the relative error the worse the bound.

A = $\begin{pmatrix} 10.25 & -9.75 \\ -9.75 & 10.25 \end{pmatrix}$, x = $\begin{pmatrix} 0 \\ 0 \end{pmatrix}$, b $\in \begin{pmatrix} [10.25 & 10.75] \\ [9.25 & 9.75] \end{pmatrix}$

$d_1 = b_1$	(1)	$\alpha = \delta_{new}/\alpha_{den}$	(17)
$d_2 = b_2$	(2)	$x_{t1} = \alpha d_1$	(18)
$r_1 = b_1$	(3)	$x_{t2} = \alpha d_2$	(19)
$r_2 = b_2$	(4)	$x_1 = x_1 + x_{t1}$	(20)
$\delta_{n\_t1} = r_1 r_1$	(5)	$x_2 = x_2 + x_{t2}$	(21)
$\delta_{n\_t2} = r_1 r_2$	(6)	$r_{t1} = \alpha q_1$	(22)
$\delta_{new} = \delta_{n\_t1} + \delta_{n\_t2}$	(7)	$r_{t2} = \alpha q_2$	(23)
$q_{t1} = A_{11} d_1$	(8)	$r_1 = r_1 - r_{t1}$	(24)
$q_{t2} = A_{12} d_2$	(9)	$r_2 = r_2 - r_{t2}$	(25)
$q_{t3} = A_{21} d_1$	(10)	$\delta_{old} = \delta_{new}$	(26)
$q_{t4} = A_{22} d_2$	(11)	$\delta_{n\_t1} = r_1 r_1$	(27)
$q_1 = q_{t1} + q_{t2}$	(12)	$\delta_{n\_t2} = r_2 r_2$	(28)
$q_2 = q_{t3} + q_{t4}$	(13)	$\delta_{new} = \delta_{n\_t1} + \delta_{n\_t2}$	(29)
$\alpha_{d\_t1} = d_1 * q_1$	(14)	$\beta = \delta_{new}/\delta_{old}$	(30)
$\alpha_{d\_t2} = d_2 * q_2$	(15)	$d_1 = \beta d_1$	(31)
$\alpha_{den} = \alpha_{d\_t1} + \alpha_{d\_t2}$	(16)	$d_2 = \beta d_2$	(32)

Fig. 2. Pseudo Code for one iteration of the conjugate gradient algorithm on a 2x2 matrix to solve $Ax = b$.

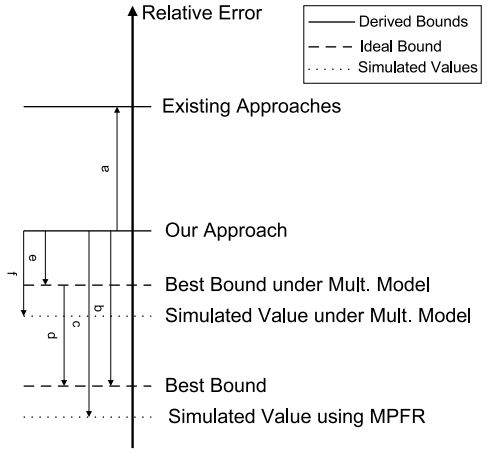


Fig. 3. Hierarchy of Error Bounds.

The initial aim of our results is to demonstrate that our approach is better than existing analytical methods that provide bounds (distance a from Figure 3). To this end, we compare our approach against Interval Arithmetic, Affine arithmetic and Taylor models with interval remainder bounds restricted to 1st, 2nd and 3rd order where possible. The reason we are restricted to these small orders is that the Taylor model for inversion is found according to (23), where $INT(f(x))$ is the range of $f(x)$ calculated using interval arithmetic, and when a large multivariate polynomial is raised to a high power when calculating $(T_\rho + I_\rho)^i$, the number of monomials grows too large to compute in a reasonable execution time.

$$\frac{1}{(T_\rho + I_\rho)} = \sum_{i=0}^{\rho} (-1)^i (T_\rho + I_\rho)^i + \frac{(-1)^{\rho+1} INT((T_\rho + I_\rho))^{\rho+1}}{INT((T_\rho + I_\rho))^{\rho+2}}. \tag{23}$$

It is also desirable to find out how far our approach lies from the ideal bound (as shown by distance b). Unfortunately, finding the ideal bound is computationally intractable, for it would involve calculating the potential values of every variable and from these values determining the worst case error loss for every precision. Instead, this is approximated by Monte Carlo simulation using the MPFR multiple precision floating library [36] (distance c). However, one must note that such an approach no longer returns a provable bound, and the difference between our approach and this bound is a function of three factors: the quality of our calculated bound, the simulation time, and the accuracy of the standard multiplicative model of floating-point error. The model of error used in this work, and throughout the numerical analysis literature, is in practice a conservative approximation because each δ_i is a function of the input variables, but this information is lost; as an example, multiplying any value by any power of two will be error free in the absence of overflow or underflow. Therefore there will exist a distance d between the ideal bound and the best any approach using this model of error could possibly achieve. As such, we are most interested to find out how far our approach differs from the best possible bound achievable using the model of floating-point error generally used in numerical analysis literature (distance e). Unfortunately, as has been discussed in Section II, finding the best bound under the multiplicative model of floating-point error involves polynomial optimisation, which is NP-hard. Therefore, this too is approximated under our test labelled 'Random sampling over poly', where we apply Monte Carlo simulation over the relevant ranges for all the variables to the polynomial we created which bounds the range or relative error of the result (distance f). It should be noted that if distance f can be shown to be small, this will imply our approach lies close to the optimal bound achievable e (since it is known *a priori* that $f \geq e \geq 0$). Finally, distance c is also reported, so as to gain an approximate estimate of the appropriateness of the multiplicative model of error. As both c and f are simulated values, it cannot be guaranteed that $c \geq f$, but if c is much larger than f, it is likely to be a result of the conservative nature of the multiplicative model.

As our approach consists of several stages, the main stages of which are creating a simplified polynomial and finding a bound for this polynomial using Handelman representations, to help quantify the contribution of each of these stages, we compare the result of bounding the polynomial in canonical form by applying interval arithmetic against our full approach. One should note that for algorithms which do not include division, the test of bounding the simplified polynomial using interval arithmetic is comparable to applying Taylor models approach without any intermediate bounding of higher order terms, which would give the tightest bound achievable using Taylor models. For algorithms which do involve division, our test which bounds the simplified polynomial using interval arithmetic simply bounds the numerator and denominator polynomial separately and performs interval arithmetic on these two results instead of using the approximation given by (23). This however remains an interesting test as it allows us to focus on quantifying how our method of applying Handelman representations to bound a rational function can find tighter bounds than any approach that is fundamentally based upon interval arithmetic.

V. Results

A. Analysis on our approach on an iteration of the 2x2 Conjugate Gradient Algorithm

1) The effects of finite precision on range: Figure 4(a) highlights the effect precision has on the conjugate gradient algorithm and the quality with which our approach can characterise this effect. It shows how the bounds on the range of the variable d_1 after an iteration of the conjugate gradient algorithm (operation 31 from Figure 2) changes as a function of the precision, for the tests mentioned in Section IV. On these graphs, the vertical dotted lines illustrate the values of precision for realisable word-lengths, *i.e.* the difference in word-length between any two adjacent dotted lines is one bit. It is clear to see there is a significant difference between the ranges computed by all the analytic approaches, as well as the simulation estimates, in comparison to interval arithmetic. In comparison to other approaches, our approach generally finds the tightest bounds, the exception being when the precision is very small where affine arithmetic and Taylor series methods can calculate bounds where our approach fails. This is because in these cases, bounds for the error variables are proportional to the bounds on the input ranges, so the first order approximation of division retains most of the information, whereas our heuristic struggles without such a simplification.

TABLE III
Resource Usage, Max Frequency and Latency of Conjugate Gradient Implementations

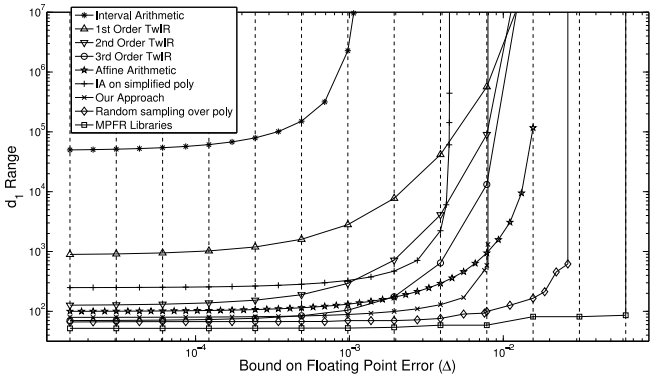
Method	Precision (# bits)	Slices	Frequency (MHz)	Latency (cycles)
Our Approach	7	1663	367	161
Affine Arithmetic	8	1710	360	165
3rd Order Taylor	9	1923	350	169
IA on Simplified Poly	9	1923	350	169
2nd Order Taylor	10	1964	314	173
IEEE Single Precision	23	5587	286	277
IEEE Double Precision	52	15672	143	425
1st Order Taylor	∞	∞	N/A	N/A
IA	∞	∞	N/A	N/A

TABLE IV
Comparison of execution times to compute range of d_1 and relative error of r_1 for a given Precision.

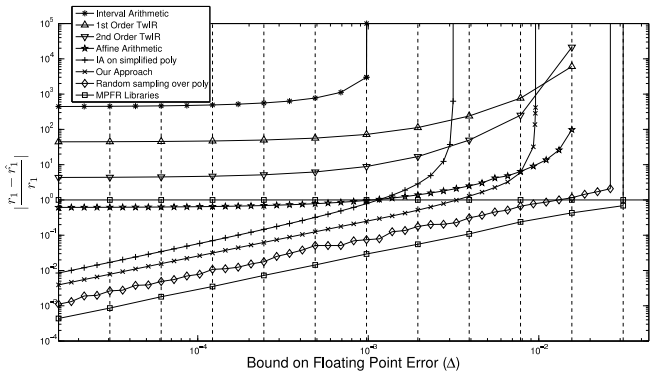
Method	Average time to compute range of d_1 (s)	Average time to compute relative error of r_1 (s)
Interval Arithmetic	0.003	0.003
Affine Arithmetic	3.7	3.5
Taylor Model of 1st Order	5.3	8.26
Taylor Model of 2nd Order	48	53
Taylor Model of 3rd Order	3700	N/A
Our approach (set-up time)	3660	150
Our approach (each iterative refinement)	2100	35

It is also interesting to note that this graph demonstrates that both stages of our approach provide significant benefits towards obtaining a better bound, for performing interval arithmetic on the simplified polynomial is significantly better than interval arithmetic in the traditional sense, whilst bounding the simplified polynomial using Handelman representations improves this bound even further.

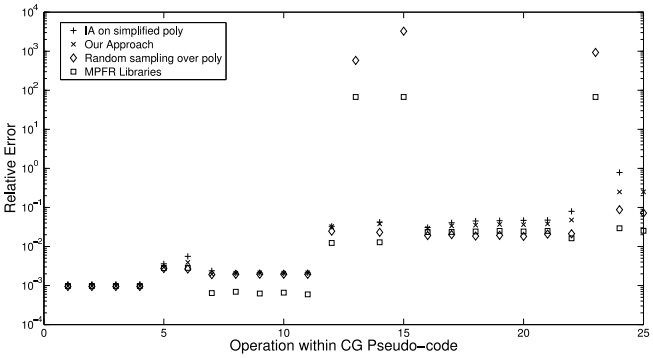
In order to view how these results translate to actual hardware savings on a FPGA, Table III shows the number of slices required, the latency, and the maximum frequency achievable when using single or double precision units, or by



(a) Bounds on the range of the variable d_1 (operation 31 from Figure 2).



(b) Bound on relative error of CG 'Residual'. r_1 (operation 24 from Figure 2) is the nominal 'residual', whilst $\hat{r}_1$ is the residual taking into account floating-point error.



(c) Growth in Relative error throughout a CG iteration. Operations correspond to pseudo code in Figure 2.

Fig. 4. Range and Relative error results for various operations in Conjugate Gradient example.

performing optimisation for a given bound on range using all approaches targeting a desired range of 600. In this test, these figures are post place and route results where the floating point components are generated using Xilinx Coregen [37]. From this table, our approach can achieve a reduction in slices in comparison to optimising the design using the other approaches, and significantly larger savings in comparison to using full IEEE single or double precision arithmetic. It also demonstrates the design would run at a faster frequency and an iteration would complete in fewer cycles by performing the optimisation.

2) The effect of finite precision on relative error: Figure 4(b) demonstrates the bound on relative error. This figure

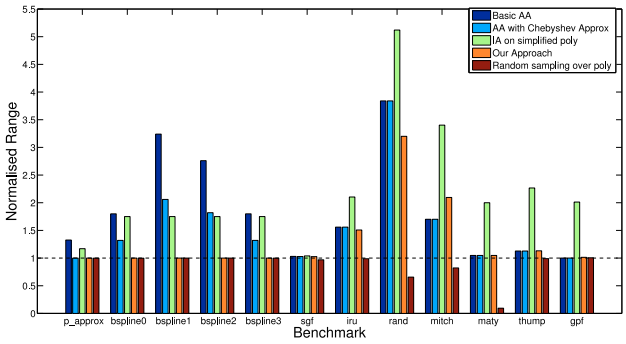
clearly demonstrates that the relative error decreases with precision, and that our algorithm is capable of tracking this relationship well, unlike all the other methods operating on the original code. The reason the other approaches cannot track this relationship is that almost all of the relative error terms will be of second order or greater because they will be a function of the input variables multiplied by some finite precision error. However interval arithmetic retains no information about the polynomial, whilst affine arithmetic arithmetic only retains first order information and approximates higher orders, and the tests using Taylor models only retain first and second order information. As a result, the bounds reported will be based on approximations of higher order terms, and these approximations must be treated independently, therefore all dependencies are lost. In contrast, by retaining all the information throughout forming the polynomial, as shown by applying interval arithmetic on the simplified polynomial, it is possible to obtain better bounds, whilst the benefit of our approach in handling dependencies within the polynomial using Handelman representations results in even tighter bounds.

3) Finite precision effects throughout an algorithm: Figure 4(c) demonstrates how the error as a result of using a chosen finite precision grows throughout the conjugate gradient algorithm. The abscissa represents the operation in the algorithm, corresponding to the line numbers given in Figure 2. This demonstrates that as the number of dependencies grow, the relative error grows, along with the difficulty to bound this error. This graph also highlights some of the deficiencies of simulation methods: at several points, the relative error is high for both the simulation methods, but undefined using the two analytical approaches. Upon further inspection, it can be shown that over the specified input range, the denominator in the relative error term $(n\hat{d})$ can legitimately include zero, as a result of input ranges. This demonstrates the limitation of the simulation approach.

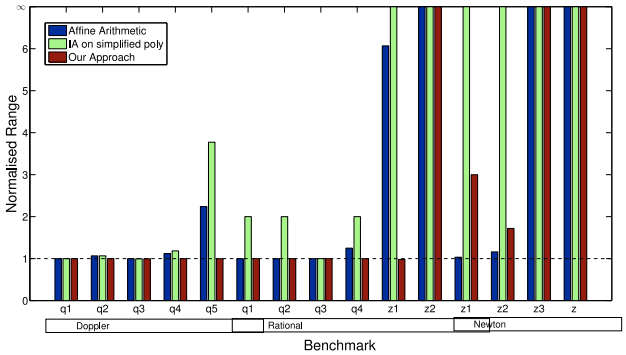
4) Execution time of our approach on Conjugate Gradient: It should however be mentioned that analysing bounds on the range of the variable d_1 or d_2 (lines 31 or 32) and the relative error for the value r_1 or r_2 (lines 24 or 25) after one iteration was as far through the conjugate gradient algorithm that our approach could calculate bounds in a reasonable amount of time. The times for all approaches to calculate the polynomial, calculated as an average over many test runs on an Intel Xeon E5345, are given in Table IV. In this table, because the conjugate gradient algorithm includes division, our approach requires the iterative refinement mentioned in Section III-D, so we have separated the computation time for our approach into two stages: set-up time - the time to calculate the initial bounds to perform the iterative refinement, and the time for each iterative refinement. As many iterative refinements are required to calculate the bounds for each point on the graph, this is as far through the algorithm that we could calculate in a reasonable time, especially given the scalability issues which will be discussed in Section V-D. In comparison to interval arithmetic and affine arithmetic our approach is significantly longer as these approaches significantly limit the number of monomials in the polynomial, allowing a faster solution, where in contrast, when finding the range of d_1, our heuristic was applied to a polynomial consisting of approximately 2 million monomials. The execution time of our approach is more comparable to Taylor models of higher orders firstly because they retain many more monomials and secondly because the time to compute the approximation for division is large.

While the execution time for computing the relative error in Table IV appears much smaller, we cannot bound a variable further through the conjugate gradient algorithm because the function representing relative error is much larger in degree and number of monomials than the associated polynomial representing range. To explain this, let us describe the nominal value of the rational function by n/d, and the rational function including floating-point error by $\hat{n}/\hat{d}$. Using this notation, for the graph in Figure 4(a), we find the bounds of the rational function $\hat{n}/\hat{d}$ for the upper and lower bounds, whereas when finding the relative error in Figure 4(b), we find bounds of the rational function $[(\hat{n}/\hat{d}) - (n/d)]/(n/d)$ or $(\hat{n}d - \hat{d}n)/(n\hat{d})$. With this larger function, the squaring operation (operation 27 in Figure 2) results in a polynomial that is too large to compute in tractable time. This additional complexity also affects other approaches, notably the computation of $1/(\hat{r}_1)$ takes too long and too much memory to compute using a 3rd order Taylor approximation. Note, however, that the size of this benchmark is still significantly greater than those reported in [11]–[13].

B. Range analysis vs other works



(a) Comparison vs AA. Range widths found for the benchmarks using our approach and AA are normalised with respect to the 'ideal' values stated in [27].



(b) Comparison vs SMT. Range widths found for the benchmarks using our approach and AA are normalised with respect to the values stated in [12], [13].

Fig. 5. Range comparison against published methods.

Figures 5(a) and 5(b) show the performance relative to the approaches given in related works, with the actual values given

TABLE V
COMPARISON OF OUR APPROACH VS AA USING CHEBYSHEV APPROXIMATIONS APPROACH [27].

	Infinite Precision											
	Basic AA		AA with Chebyshev Approx		SIA		Our		Monte Carlo		Ideal	
	Lower	Upper	Lower	Upper	Lower	Upper	Lower	Upper	Lower	Upper	Lower	Upper
Poly Approx	-0.0541	0.865	0	0.6931	0	0.8108	0	0.6932	0.0003	0.6929	0	0.6931
B - Spline 0	-0.13	0.17	-0.05	0.17	-0.125	0.1667	0	0.1667	0	0.1664	0	0.1667
B - Spline 1	-0.33	1.29	-0.05	0.98	0.0417	0.9167	0.1667	0.6667	0.1667	0.6667	0.1667	0.6667
B - Spline 2	-0.21	1.17	-0.02	0.89	0.0417	0.9167	0.1667	0.6667	0.1667	0.6667	0.1667	0.6667
B - Spline 3	-0.17	0.13	-0.17	0.05	-0.1667	0.125	-0.1667	0	-0.1666	0	-0.1667	0
sgf	-9803	9525	-9793	9487	-9821	9671	-9765	9487	-9301	8874	-9453	9303
iru	-95000	128000	-95000	128000	-148350	152450	-91390	124160	-53581	87743	-55100	87900
rand	-192	192	-192	192	-256	256	-192	128	-29.2096	36.432	-36	64
mitch	-223	881	-223	881	-1087	1121	-719	641	-7.9817	525.5058	-8	641
maty	-4800	100000	-4800	100000	-100000	100000	-4800	100000	0.2	9487.4	0	100000
thump	-60000	1000000	-60000	1000000	-1065400	1065400	-62400	1001200	0	930990	0	940000
gpf	-1.2E+08	1.19E+08	-1.2E+08	1.13E+08	-8416264	8417464	-7261200	7098000	0	1013500	3	957000
rat	-2.1E+08	3.3E+11	-2.1E+08	3.3E+11	-3.3E+11	3.34E+11	-6.1E+08	3.34E+11	0	3.32E+11	-1.03	3.3E+11

	With Finite Precision ($\Delta = 2^{-10}$)						
		Our		SIA		Monte Carlo	
		Lower	Upper	Lower	Upper	Lower	Upper
Poly Approx		-0.00331	0.814183	-1E-04	0.696197	0.000556	0.694708
B - Spline 0		-0.12598	0.167646	-2.8E-17	0.167646	3.8E-11	0.167066
B - Spline 1		0.036691	0.921643	0.16325	0.668954	0.166967	0.667602
B - Spline 2		0.036689	0.921644	0.166341	0.671402	0.16681	0.668224
B - Spline 3		-0.16683	0.125163	-0.16683	1.39E-17	-0.16645	-4.7E-15
sgf		-9930.96	9780.957	-9949.73	9608.532	-9245.45	8817.592
iru		-149425	153520.6	-91996.1	124927.5	-54465.3	87326.63
rand		-258.259	258.2588	-193.41	242.08	-33.5023	32.21554
mitch		-1095.63	1129.625	-716.476	664.7615	-7.97463	538.1909
maty		-10034.4	10034.42	-4814.08	10034.42	0.298266	9870.291
thump		-1072698	1072698	-63759	1069686	2.09488	937919.5
gpf		N/A	N/A	N/A	N/A	N/A	N/A
rat		-3.4E+11	3.37E+11	-9.4E+08	3.37E+11	2234.393	3.31E+11

TABLE VI
EXECUTION TIME OF OUR APPROACH AND NUMBER OF MONOMIALS FOR AA BENCHMARKS [27].

	Infinite Precision		
	Time using AA (ms)	Time using Our Approach (ms)	Number of Monomials
Poly Approx		232	20
B - Spline 0	85.8	176	4
B - Spline 1	94.0	143	4
B - Spline 2	95.1	149	4
B - Spline 3	83.5	173	4
sgf	1288.9	370	10
iru	1327.2.9	186	11
rand	413.9	304	9
mitch	764.8	209	10
maty	288.4	124	3
thump	627.3	287	5
gpf	2545.8	300	45
rat	1053.2	227	6

	With Finite Precision ($\Delta = 2^{-10}$)		
	Time using AA (ms)	Time using Our Approach (ms)	Number of Monomials
	N/A	2588	1280
	N/A	214	8
	N/A	722	624
	N/A	525	304
	N/A	466	128
	N/A	184527	11738
	N/A	6378	4440
	N/A	22133	4608
	N/A	9379	2786
	N/A	285	40
	N/A	551	224
	N/A	N/A	N/A
	N/A	4767	1080

TABLE VII
COMPARISON OF OUR APPROACH VS SAT MODULO THEORY APPROACH [11], [12].

	Infinite Precision							
	AA		SIA		Our		SAT Mod	
	Lower	Upper	Lower	Upper	Lower	Upper	Lower	Upper
Doppler q1	313	362	313	362	313	362	313	362
Doppler q2	-473252	7228000	-473252	7228000	6268	7228000	6267	7228000
Doppler q3	213	462	213.4	461.4	213	461.4	213	462
Doppler q4	25363	212890	14790	212890	45539	212890	45539	212890
Doppler q5	-80	229	-32.0034	488.7892	0.0339	137.6386	0	138
Rational q1	125	250125	-249875	250125	125	250125	124	250126
Rational q2	1	10001	-9999	10001	1	10001	0	10002
Rational q3	-20000	20000	-20000	20000	-20000	20000	-20001	20001
Rational q4	-2.5E+07	1E+08	-1E+08	1E+08	1	1E+08	0	1E+08
Rational z1	-250	369	FAIL	FAIL	25.01	125	24	126
Rational z2	FAIL	FAIL	FAIL	FAIL	FAIL	FAIL	-67	67
Newton z1	-1250360	1170360	FAIL	FAIL	-3615080	3405080	-1205361	1135361
Newton z2	-5753	35769	FAIL	FAIL	-2.57E+04	3.58E+04	1	35769
Newton z3	FAIL	FAIL	FAIL	FAIL	FAIL	FAIL	-39	38
Newton z	FAIL	FAIL	FAIL	FAIL	FAIL	FAIL	-69	72

	With Finite Precision ($\Delta = 2^{-10}$)					
	SIA		Our		Monte Carlo	
	Lower	Upper	Lower	Upper	Lower	Upper
Doppler q1	313.0177	361.7823	313.0764	361.7823	313.1927	361.5274
Doppler q2	-487963	7242711	5903.514	7242711	7154.64	7183615
Doppler q3	212.5668	462.2332	212.8682	462.2332	218.0062	460.9534
Doppler q4	13809.32	213868.2	45261.52	213868.2	48330.72	208073.4
Doppler q5	-35.848	524.992	0.032124	138.6192	0.088272	127.0179
Rational q1	-250608	250858.3	124.8779	250858.3	125.0591	250134.7
Rational q2	-10018.5	10020.54	0.999023	10020.54	1.00445	9999.922
Rational q3	-20019.5	20019.53	-20019.5	20019.53	-19968.4	19980.35
Rational q4	-1E+08	1.01E+08	0.997073	1.01E+08	1.000176	99690820
Rational z1	-25.0639	25.05885	24.91244	126.8	24.94171	124.9827
Rational z2	FAIL	FAIL	FAIL	FAIL	FAIL	FAIL
Newton z1	-2487036	2507516	-2487057	2227621	-2395376	2164654
Newton z2	-292243	292307.4	-200614	292307.4	-139.109	275900.8
Newton z3	FAIL	FAIL	FAIL	FAIL	FAIL	FAIL
Newton z	FAIL	FAIL	FAIL	FAIL	FAIL	FAIL

[1]

TABLE VIII
EXECUTION TIME OF OUR APPROACH AND NUMBER OF MONOMIALS FOR SMT BENCHMARKS [11].

	Infinite Precision		
	Time using SMT Approach (ms)	Time using Our Approach (ms)	Number of Monomials
Doppler q1	∼100000	118	2
Doppler q2	∼100000	148	4
Doppler q3	∼100000	116	3
Doppler q4	∼100000	176	6
Doppler q5	∼100000	3000	10
Rational q1	∼100000	121	2
Rational q2	∼100000	124	2
Rational q3	∼100000	114	1
Rational q4	∼100000	189	3
Rational z1	∼100000	4000	4
Rational z2	∼100000	N/A	N/A
Newton z1	∼100000	171	7
Newton z2	∼100000	142	6
Newton z3	∼100000	N/A	N/A
Newton z	∼100000	N/A	N/A

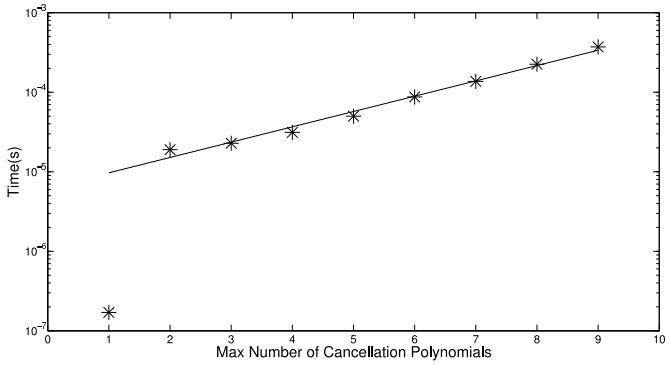
	With Finite Precision ($\Delta = 2^{-10}$)		
	Time using SMT Approach (ms)	Time using Our Approach (ms)	Number of Monomials
	N/A	165	8
	N/A	292	32
	N/A	203	18
	N/A	576	216
	N/A	31000	280
	N/A	238	10
	N/A	189	6
	N/A	145	2
	N/A	314	36
	N/A	13000	26
	N/A	N/A	N/A
	N/A	691	29
	N/A	517	8
	N/A	N/A	N/A
	N/A	N/A	N/A

[1] Exact times not reported, but all benchmark times of order 100s in [11], [12].

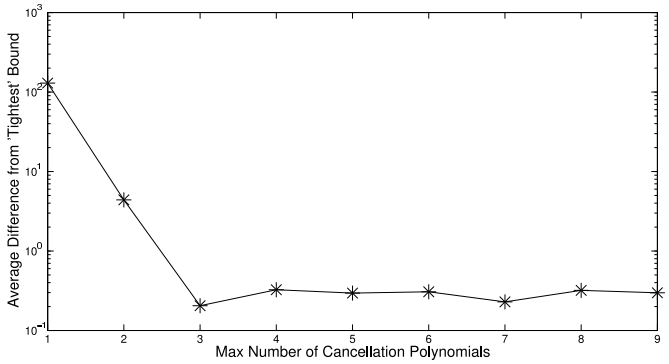
in Tables V and VII. In Figure 5(a), the ranges for the various approaches are normalised against what is quoted as the ideal range given in [27]. It is interesting to note that in many of the test cases our approach matches the ideal, and in all but one our approach is superior to the other methods. As these examples do not include any finite precision effects, this difference is not a function of the model of error, rather the fact that our approach does not perform well in this one case demonstrates that our heuristic does not always find the best Handelman representation. This is a likely to be a result of targeting our heuristic mainly towards minimising finite precision errors as opposed to range analysis. Once again, this graph also illustrates that both stages in our algorithm have significant effects on the final quality of the bound, since simply applying the first stage to create the polynomial and using interval arithmetic on this polynomial can give comparable results to the existing methods for many of the benchmarks, whilst applying the second stage improves the bound further. It is also important to note at this stage that all the test cases in Figure 5(a) and some in Figure 5(b) are only for polynomials, and hence the results for interval arithmetic on the simplified polynomial are equivalent to the best one could achieve using Taylor models without any intermediate bounding of higher order terms, and this demonstrates how our approach outperforms this method. Finally, the random sampling of values of $\boldsymbol{\delta}$ on the simplified polynomial is included to show that the difference between simulation and the best possible bound (distance (c-b) from Figure 3) exists even for these simple examples consisting of only three variables. This helps to emphasise that the bounds found in the conjugate gradient analysis are very accurate, given that they consist of up to 25 variables for the polynomial minimised in Figure 4(a).

In Figure 5(b), the various approaches are normalised against the ranges given in [11], [12], which given sufficient run time, should be optimal. Interestingly, our approach in some cases gives slightly superior results. This is likely to be a result of the fact the SMT is a refinement process which is potentially time consuming and hence the refinement stops once at a given level of accuracy. On the other hand, it is important to note that in some cases SMT does outperform our approach to the extent that in some cases our work gives unbounded results, whereas SMT solver returns bounds. Furthermore, our approach either outperforms or is equivalent to affine arithmetic in all but two of these benchmarks. Similarly to the previous example, this is a limitation of the method at finding the best Handelman representation, and whilst achieving better bounds is possible, it would require a more sophisticated search for Handelman representations, which will be time consuming. However, it should be mentioned that our solver calculates these values within a hundreds of milliseconds, as shown in Tables VI and VIII, whereas in [12], the results for each value are reported as around the order of 100 seconds on a comparable machine. Furthermore, one should also note that our approach could be used as an input to the SMT solver, which uses initial estimates based on interval or affine arithmetic, to decrease the time to find a good solution.

1) Benchmark tests in finite precision: Tables V and VII give a list of various results over the benchmarks mentioned earlier, along with the addition of finite precision effects. This is included both to provide data for future works to compare to, as well as to demonstrate how finite precision can affect such bounds. It also shows the benefit of this approach over those given in the comparative works seeing as ours is far more scalable as it can find a solution over many more variables in tractable time: most values are still calculated within hundreds of ms (on a standard desktop with an Intel Core2Duo E6850 processor), as seen in Tables VI and VIII. Though there are some values that take significantly longer (up to half a minute), these are due to the iterative refinement to compute bounds for division, with the number of iterations of the algorithm proportional to the desired precision of the bound.



(a) Average Execution Time to Compute update of 'm' Cancellation Polynomials.



(b) Average Distance From 'Tightest' Calculated Bound for Product of 'm' Cancellation Polynomials.

Fig. 6. Graphs Illustrating the effect of changing the number of Cancellation Polynomials ('m').

C. Choosing the maximum number of cancellation terms

The choice of the number of cancellation terms, the value of the variable 'm' in Figure 1, has been chosen experimentally. Figures 6(a) and 6(b) show how the execution time and error respectively change with m. This section discusses how they were used to select a value of 'm'.

The execution times plotted in Figure 6(a) are the average time to compute the variables j, h, q and f from Figure 1, as the rest of the execution time should be independent of the variable m. The growth in execution time, as can be seen in Figure 6(a), is approximately exponential in m. This is expected given that the number of monomials in $h(= \prod_{i=1}^{n'} j_i)$ is worst case exponential in m, and this means the number of

computations to choose the value q and perform the update of f will also grow exponentially. The deviations from the line of best fit are likely to be caused by the fact that $n' \leq m$, as opposed to exactly equal to m, changing the number of computations for some iterations. The computation time for $m = 1$ is so low because this is a special case in that it is equivalent to performing interval arithmetic on the expanded polynomial, which is significantly faster. Finally, one should note that the choice of m can also impact the number of iterations of the algorithm, but this is hard to quantify for practical examples as it is highly dependent upon the input polynomial.

Because it is impractical to calculate ideal bounds, in order to gain an insight into the quality of the GHR as m increases, the algorithm in Figure 1 is applied to a polynomial with various values of m and the difference is calculated between the bound for each value of m and the tightest bound out of all the tested values of m returned for that polynomial. This process is then repeated over several polynomials to obtain an average, which is plotted in Figure 6(b). From this figure, it is clear that when $m = 1$, the result is significantly worse, demonstrating again the value of applying our procedure as opposed to performing interval arithmetic on the polynomial. However, for $m \geq 3$, the quality of the best bound is unclear, as seen in Figure 6(b). This demonstrates the intricacies involved in choosing the best GHRs. Our algorithm was based upon the idea of reducing the coefficients of lower order monomials at the same time as cancelling higher order monomials. When m is larger, it will create more monomials and these can potentially reduce more lower order monomials, but whilst the algorithm is designed to attempt to choose these terms to reduce the coefficients of the monomials in f, it is not guaranteed, indeed it is even possible to create some unwanted higher order monomials that are products of the lower order monomials. However, more importantly, when m is large, many of these terms created in h will be very small as they will be multiplied by Δ raised to high powers, meaning that for $m > 3$, it is expected that there will be little quantifiable gain in quality of result, and since the execution time grows exponentially with m, the choice of $m = 3$ would appear to be ideal in practice.

D. Scalability of our approach

Figure 7 demonstrates how the execution time of our heuristic to bound the polynomial grows with the number of monomials in the polynomial g. It is clear from this graph that execution time grows at a steady rate with the number of monomials. The rate of growth is slightly super-linear, which is a result of typically creating a GHR for each term and the complexity of creating a GHR increasing for higher degree monomials and the polynomials with more monomials containing more higher degree monomials.

However, whilst the execution time is only slightly super-linear in the number of monomials used for the canonical polynomial representation, the number of monomials grows significantly faster. This is because in order to correctly bound the error introduced by the use of floating point precision, the

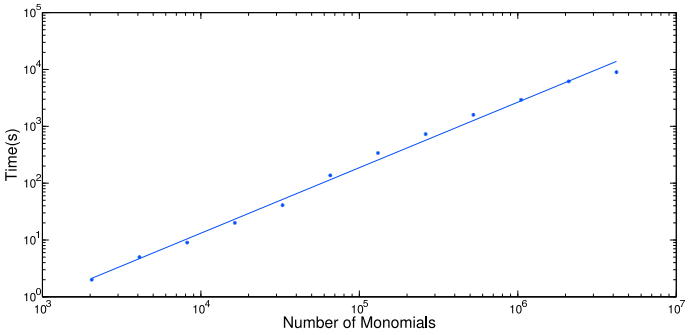


Fig. 7. Execution Time growth with Number of Monomials.

```
Algorithm y = VectorProduct(x)
y=1;
for i=1 to n
    y = y*x[i];
end
```

Fig. 8. Pseudo code to calculate the product of vector elements.

polynomial representing the result of any operation is multiplied by the function $(1 + \delta_i)$, according to the multiplicative model of error used throughout the numerical analysis literature, doubling the number of monomials in the representation. For example, the floating-point error model for an algorithm consisting of a series of floating-point multiplications, an example pseudo code for which is shown in Figure 8, will have a polynomial representation as in (24); the number of monomials in this polynomial, when expanded into canonical form, is exponential in n.

$$x_1 x_2 .... x_n (1 + \delta_1)(1 + \delta_2)...(1 + \delta_n) \tag{24}$$

However, in many algorithms, this worst case does not occur, firstly because results are often not accumulated using a single variable throughout the course of the algorithm, and secondly because it is highly likely that a large amount of cancellation will occur.

VI. Conclusion

This paper has demonstrated a heuristic, based upon a result from real algebra, that can be used to find analytical bounds for any value within an algorithm. We have demonstrated that our approach takes into account dependencies within a polynomial representing the range of a variable when calculating bounds, and this can not only lead to much better bounds than a naïve implementation of interval arithmetic, but also more sophisticated techniques such as affine arithmetic or Taylor series with remainder bounds.

Whilst this research has shown a high degree of utility, there are several limitations that would benefit from further research. These include scalability, highlighted in Section V-D, improving the heuristic to obtain better Handelman representations, discussed in Section V-B, as well as extending the method to handle non-polynomial functions such as square roots and exponentials. However, as mentioned in Section III-E, we could create a combined approach which uses polynomial simplification or polynomial models for complex functions throughout

the algorithm, such as those used for affine arithmetic or Taylor models with interval remainder bounds, which replaces any intermediate or final bounding by interval arithmetic with our approach when bounding any resultant polynomial to address these issues and improve the overall bounds. Similarly, this could be integrated within a global optimisation framework that uses interval analysis as an additional tool to improve bounds.

The Handelman representation is just one special case of 'theorems of alternatives' in which real algebraic geometry is rich. In particular, the proposed approach can be seen as a search over a particular form of Positivstellensatz refutation [38], an insight which could lead to further sophisticated algorithms developed in this field.

ACKNOWLEDGEMENTS

Supported in part by EU FP7 Project REFLECT.

REFERENCES

[1] J. Langou, J. Langou, P. Luszczek, J. Kurzak, A. Buttari, and J. Dongarra, "Exploiting the performance of 32 bit floating point arithmetic in obtaining 64 bit accuracy (revisiting iterative refinement for linear systems)," in *Proc. ACM/IEEE conference on Supercomputing*. New York, NY, USA: ACM, 2006, p. 113.

[2] Nvdia, "Tesla c1060 computing processor board," Tech. Rep., January 2007. [Online]. Available: http://www.nvidia.com/docs/IO/43395/BD-04111-001_v05.pdf

[3] J. A. Kahle, M. N. Day, H. P. Hofstee, C. R. Johns, T. R. Maeurer, and D. Shippy, "Introduction to the cell multiprocessor," *IBM J. Res. Dev.*, vol. 49, no. 4/5, pp. 589–604, 2005.

[4] G. Constantinides, P. Cheung, and W. Luk, "Wordlength optimization for linear digital signal processing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 22, no. 10, pp. 1432–1442, Oct. 2003.

[5] K.-I. Kum and W. Sung, "Combined word-length optimization and high-level synthesis of digital signal processing systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 20, no. 8, pp. 921–930, Aug 2001.

[6] A. R. Lopes, A. Shahzad, G. Constantinides, and E. Kerrigan, "More FLOPS or more precision? Accuracy parameterizable linear equations solvers for model-predictive control," in *Proc. IEEE International Symposium on Field-Programmable Custom Computing Machines*. New York, NY, USA: ACM, 2009.

[7] K. Underwood, "FPGAs vs. CPUs: trends in peak floating-point performance," in *Proc. 12th ACM/SIGDA international symposium on Field-Programmable Gate Arrays*. New York, NY, USA: ACM, 2004, pp. 171–180.

[8] D. Boland and G. Constantinides, "An FPGA-based implementation of the MINRES algorithm," in *Proc. Int. Conf. Field Programmable Logic and Applications*, Sept. 2008, pp. 379–384.

[9] A. Lopes and G. Constantinides, "A high throughput FPGA-based floating point conjugate gradient implementation," *Reconfigurable Computing: Architectures, Tools and Applications*, pp. 75–86, 2008.

[10] D. Boland and G. A. Constantinides, "Automated precision analysis: A polynomial algebraic approach," *Proc. IEEE Symp. on Field-Programmable Custom Computing Machines*, vol. 0, pp. 157–164, 2010.

[11] A. B. Kinsman and N. Nicolici., "Finite precision bit-width allocation using SAT-modulo theory," in *Proc. Int. Conf. DATE*, 2009.

[12] A. B. Kinsman and N. Nicolici, "Bit-width allocation for hardware accelerators for scientific computing using sat-modulo theory," *Trans. Comp.-Aided Des. Integ. Cir. Sys.*, vol. 29, pp. 405–413, March 2010. [Online]. Available: http://dx.doi.org/10.1109/TCAD.2010.2041839

[13] ——, "Robust design methods for hardware accelerators for iterative algorithms in scientific computing," in *Proceedings of the 47th Design Automation Conference*. New York, NY, USA: ACM, 2010, pp. 254–257.

[14] G. Constantinides, P. Cheung, and W. Luk, "Optimum wordlength allocation," in *Proc. IEEE Symp. on Field-Programmable Custom Computing Machines*, 2002, pp. 219–228.

[15] D.-U. Lee, A. Gaffar, R. Cheung, O. Mencer, W. Luk, and G. Constantinides, "Accuracy-guaranteed bit-width optimization," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 10, pp. 1990–2000, Oct. 2006.

[16] M. L. Chang and S. Hauck, "Précis: A design-time precision analysis tool," in *Proc. IEEE Symp. on Field-Programmable Custom Computing Machines*. Washington, DC, USA: IEEE Computer Society, 2002, pp. 229–238.

[17] H. Choi and W. Burleson, "Search-based wordlength optimization for VLSI/DSP synthesis," in *Proc. Workshop on VLSI Signal Processing, VII*, 1994, pp. 198–207.

[18] Z. Zhao and M. Leeser, "Precision modeling and bit-width optimization of floating-point applications," in *High Performance Embedded Computing*, 2003, pp. 141–142.

[19] D. Priest, "Algorithms for arbitrary precision floating point arithmetic," in *Proc. IEEE. Symp. Computer Arithmetic*, Jun. 1991, pp. 132–143.

[20] R. E. Moore, *Interval Analysis*. Englewood Cliff, NJ: Prentice-Hall, 1966.

[21] J. de Figueiredo, Luiz Henrique Stolfi, "Affine arithmetic: concepts and applications," *Numerical Algorithms*, vol. 37, no. 1, pp. 147–158, 2004.

[22] R. E. Moore and F. Bierbaum, *Methods and Applications of Interval Analysis*. Soc for Industrial & Applied Math, 1979.

[23] B. Einarsson, *Handbook on Accuracy and Reliability in Scientific Computation*. Soc for Industrial & Applied Math, 2005, ch. 10, pp. 195–240.

[24] M. Berz and G. Hoffstätter, "Computation and application of Taylor polynomials with interval remainder bounds," *Reliable Computing*, vol. 4, pp. 83–97, 1998.

[25] U. Abel, "On the Lagrange remainder of the Taylor formula," *The American Mathematical Monthly*, vol. 110, no. 7, pp. pp. 627–633, 2003. [Online]. Available: http://www.jstor.org/stable/3647748

[26] M. Daumas and G. Melquiond, "Certification of bounds on expressions involving rounded operators," *ACM Trans. Math. Softw.*, vol. 37, no. 1, pp. 1–20, 2010.

[27] W. Z. Linsheng Zhang, Yan Zhang, "Tradeoff between approximation accuracy and complexity for range analysis using affine arithmetic," *Journal of Signal Processing Systems*.

[28] C. Herde, A. Eggers, M. Franzle, and T. Teige, "Analysis of hybrid systems using hysat," in *Systems, 2008. ICONS 08. Third International Conference on*, 2008, pp. 196 –201.

[29] E. Hansen and G. W. Walster, *Global optimization using interval analysis*, ser. Monographs and Textbooks in Pure and Applied Mathematics. New York: Marcel Dekker Inc., 2004, vol. 264.

[30] W. W. Adams and P. Loustaunau., *An Introduction to Grobner Bases*. American Mathematical Society, 1994.

[31] N. J. Higham, *Accuracy and Stability of Numerical Algorithms*, 2nd ed. Philadelphia, PA, USA: Soc for Industrial & Applied Math, 2002.

[32] J. Garloff and A. P. Smith, "Lower bound functions for polynomials," *Proc. Int. Conf. Applied Operational Research, Tadbir Institute for Operational Research, Systems Design and Financial Services*, vol. 1, no. 1, pp. 199–211, 2008.

[33] D. Handelman, "Representing polynomials by positive linear functions on compact convex polyhedra," *Pac. J. Math*, vol. 132, no. 1, pp. 35–62, 1988.

[34] J. B. Lasserre, "Polynomial programming: LP-relaxations also converge," *SIAM J. on Optimization*, vol. 15, no. 2, pp. 383–393, 2005.

[35] M. R. Hestenes and E. Stiefel, "Methods of conjugate gradients for solving linear systems," *Journal of Research of the National Bureau of Standards*, vol. 49, pp. 409–436, Dec 1952.

[36] L. Fousse, G. Hanrot, V. Lefèvre, P. Pélissier, and P. Zimmermann, "MPFR: A multiple-precision binary floating-point library with correct rounding," *ACM Trans. Math. Softw.*, vol. 33, no. 2, June 2007.

[37] Xilinx Corp. (2010, Sept) Core generator overview. `http://www.xilinx.com/support/documentation/sw_manuals/xilinx12_2/isehelp_start.htm#cgn_c_overiew.htm`.

[38] P. A. Parrilo, "Semidefinite programming relaxations for semialgebraic problems," *Mathematical Programming Ser. B*, vol. 96, no. 2-3, pp. 293–320, 2003.