Scalable yet Rigorous Floating-Point Error Analysis

Arnab Das
University of Utah
arnabd@cs.utah.edu

Ian Briggs
University of Utah
ibriggs@cs.utah.edu

Ganesh Gopalakrishnan
University of Utah
ganesh@cs.utah.edu

Sriram Krishnamoorthy
Pacific Northwest National Laboratory
sriram@pnnl.gov

Pavel Panchekha
University of Utah
pavpan@cs.utah.edu

***Abstract*—Automated techniques for rigorous floating-point round-off error analysis are a prerequisite to placing important activities in HPC such as precision allocation, verification, and code optimization on a formal footing. Yet existing techniques cannot provide tight bounds for expressions beyond a few dozen operators—barely enough for HPC. In this work, we offer an approach embedded in a new tool called SATIRE that scales error analysis by four orders of magnitude compared to today's best-of-class tools. We explain how three key ideas underlying SATIRE helps it attain such scale: path strength reduction, bound optimization, and abstraction. SATIRE provides tight bounds and rigorous guarantees on significantly larger expressions with well over a hundred thousand operators, covering important examples including FFT, matrix multiplication, and PDE stencils.**

***Index Terms*—Floating-point arithmetic, Round-off error, Numerical Analysis, Symbolic Execution, Algorithmic Differentiation, Abstraction, Scalable Analysis**

I. Introduction

Virtually all HPC applications rely on floating-point arithmetic to realize their underlying numerical algorithms. Yet floating-point computations introduce *rounding error*, which makes computed results diverge from the mathematical truth with often negligible, but sometimes disastrous results [1], [2]. Important decisions such as precision allocation require accurately characterizing rounding error across a *range* of input values: too much precision results in excessive data movement, while too little yields erroneous behavior for demanding applications [3]. Yet most error characterization is unsound, such as measuring the round-off error for sample inputs. Unfortunately, nothing better is known today—there are no rigorous and automated methods for handling even medium-sized, everyday functions such as Partial Differential Equation (PDE) stencils, linear solvers, prefix-sums, Fast Fourier Transforms (FFT), or thousands of others.

The last few years have seen a wealth of analysis tools proposed and evaluated, including Fluctuat [4], Gappa [5], PRECiSA [6], Real2Float [7], Rosa [8], FPTaylor [9], Numerors [10], and even specialized tools for cyberphysical systems [11], [12]. Nonetheless, applying such tools to large numerical programs consisting of thousands of operators has been impossible: state-of-the-art tools are limited to programs with few dozens of operators, and otherwise time out or give woefully imprecise results. Scale is the next barrier that automated error analysis must overcome.

We present a scalable and rigorous approach to formally analyzing floating-point rounding error: SATIRE (standing for **S**calable **A**bstraction-guided **T**echnique for **I**ncremental **R**igorous analysis of round-off **E**rrors.)

SATIRE improves on the currently-best technique for rigorous error analysis: *symbolic Taylor forms* [9] which produce rigorous, precise error bounds in three steps: computing a nonlinear bounding expression based on an error model; linearizing that bounding expression by Taylor expansion; and applying global optimization to find inputs with the largest error. It introduces three key improvements to achieve its scalability. First, *path strength reduction* allows an exponential reduction in the size of the non-linear bounding expressions (§III). Second, *bound optimization* rewrites error expressions into a canonical form, ensuring that global optimization results in tighter bounds across the space of inputs (§IV). Third, *abstraction* allows for a divide-and-conquer approach to analyzing large expressions, using information-theoretic heuristics to isolate large sub-expressions and retain only their root node to summarize their error (§V). These techniques resolve weaknesses in existing tools and allow for tighter bounds and greater scale.

SATIRE achieves a precision comparable to (and often better than) the best published tools, and scales to programs four orders of magnitude larger. On a standard set of benchmarks with under 50 operators, including all benchmarks cited in prior work on symbolic Taylor forms [9], SATIRE proves tighter bounds than FPTaylor, the state of the art, and achieves an average $4.5\times$ speedup. On a small example with the 1-D heat equation (a single expression of ≈ 500 operators obtained by unrolling an iteration scheme), FPTaylor times out after eight hours while SATIRE returns results in *seconds*.

This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research under Award Number 66905. Pacific Northwest National Laboratory is operated by Battelle for DOE under Contract DE-AC05-76RL01830. This work is also supported by NSF CCF 1704715, 1817073 and 1918497.

SC20, November 9-19, 2020, Is Everywhere We Are
978-1-7281-9998-6/20/$31.00 ©2020 IEEE

On larger benchmarks such as the non-linear Lorenz equation (70 iterations with $\approx$ 1050 operators), parallel prefix sum over 4096 inputs (8K operators) and a 4096-point fast Fourier transform (FFT, $\approx$ 393K operators) are out of reach of all existing rigorous tools in this area.[1] SATIRE handles even larger examples, such as a 128x128 Matrix Multiplication (over 4M operators in under an hour) and a tensor contraction example (over 1.5M operators in under two hours). We employed shadow-value calculations in high precision at 1M randomly selected points within the input intervals to empirically check (with high confidence) that SATIRE's bounds are indeed tight.

Note that Satire-analyzed expression sizes are not "HPC-scale" in the traditional execution sense. Yet, they are many orders of magnitude larger than codes that were handled in prior rigorous analysis work. Satire's analysis is conducted across entire intervals of anticipated input values. During application development, such analysis helps a designer identify and potentially resolve lurking precision issues involving rare input combinations.

While primarily a static error analyzer, we also develop a dynamic analysis capability around Satire to obtain relative error estimates of actual applications. We show that this technique can be useful for estimating the number of bits of precision needed by many stencil-type computations and iteration schemes. Again, shadow-value testing confirms the viability of this technique.

An insight from SATIRE is that scalability is not directly tied to operator count: it also depends on the degree of non-linearity (a good example being Lorenz equations) and the degree of reconvergence within data dependence graphs (a good example being PDE solver stencils). This insight inspires SATIRE's ability to tweak tool behavior, especially abstraction, in a problem-specific manner. Recapping, SATIRE contributes:

- a path strength reduction approach that exponentially reduces the size of symbolic error expressions;
- a bound optimization method to canonicalize error expressions to best utilize global optimizers;
- an abstraction method that trades off precision and scale; and
- a scalable yet rigorous floating-point error analysis tool demonstrated on important HPC functions.

We now present background on error analysis §II and then introduce path strength reduction §III, bound optimization §IV, abstractions §V. Finally, extensive evaluations §VI demonstrate SATIRE's utility.

II. BACKGROUND

A binary floating-point number system, $\mathbb{F}$, represents a subset of real numbers in finite precision as tuples (s, e, m) representing $s \cdot m \cdot 2^e$: $s \in \{-1, 1\}$ is the sign bit, e is the exponent, and m is the mantissa represented with p bits ($p = 24$ for single-precision and $p = 53$ for double precision). For all $x \in \mathbb{R}$, then $\circ x$ denotes the element in $\mathbb{F}$ closest to x. Every real number x lying in the range of $\mathbb{F}$ can be approximated by a value $\circ x \in \mathbb{F}$ with relative error $\boldsymbol{u}$ no larger than half a *unit in the last place* 2^{1-p}. Thus, for all x in the range of $\mathbb{F}$, $\circ(x) = x(1+\delta)$ for some $|\delta| \leq \boldsymbol{u}$.

Floating-point operations are rounded as well: given two exactly-represented floating-point numbers, $\tilde{x}$ and $\tilde{y}$, each arithmetic operator $\diamond \in \{+, -, \cdot, /\}$ guarantees

$$\circ(\tilde{x} \diamond \tilde{y}) = (\tilde{x} \diamond \tilde{y})(1+\delta) = (\tilde{x} \diamond \tilde{y}) + (\tilde{x} \diamond \tilde{y})\delta, \quad |\delta| < \boldsymbol{u} \quad (1)$$

In other words, the result is equal to the exact value $\tilde{x} \diamond \tilde{y}$, plus an error term $e_1 = (\tilde{x} \diamond \tilde{y})\delta$ representing the amount of round-off error.

Now consider a straight-line program with n floating-point operations of the form

$$s_i = f_i(x_1, \ldots, x_m, s_1, s_2, \ldots, s_{i-1}), \quad (2)$$

with f ranging over arithmetic operators and elementary functions and with the program producing output s_n.

Let function f_i be approximated in finite precision by $\tilde{f}_i$. Then line i applies $\tilde{f}_i$ and computes $\tilde{s}_i = s_i + e_{s_i}$, where e_{s_i} represents the round-off error. $\tilde{s}_i$ depends on both the rounded inputs $\tilde{x}_1, \ldots, \tilde{x}_m$ and earlier intermediate values $\tilde{s}_1, \ldots, \tilde{s}_{i-1}$; or we can instead think of $\tilde{s}_i$ as depending on the exact inputs and intermediates $x_1, \ldots, x_m$ and $s_1, \ldots, s_{i-1}$ *and* their rounding errors $e_{x_1}, \ldots, e_{x_m}$ and $e_{s_1}, \ldots, e_{s_{i-1}}$:

$$\begin{aligned}
\tilde{s}_i &= \tilde{f}_i(\tilde{x}_1, \ldots, \tilde{x}_m, \tilde{s}_1, \ldots, \tilde{s}_{i-1}) \\
&= \tilde{f}_i(x_1 + e_{x_1}, \ldots, x_m + e_{x_m}, s_1 + e_{s_1}, \ldots, s_{i-1} + e_{s_{i-1}}) \\
&= f_i(x_1 + e_{x_1}, \ldots, x_m + e_{x_m}, \\
&\qquad s_1 + e_{s_1}, \ldots, s_{i-1} + e_{s_{i-1}})(1+\delta_i); \quad |\delta_i| \leq \mathbf{u}
\end{aligned}$$

We can now express the error e_{s_n} of the program output as

$$\begin{aligned}
e_{s_n} = f_n(&x_1 + e_{x_1}, \ldots, x_m + e_{x_m}, \\
&s_1 + e_{s_1}, \ldots, s_{i-1} + e_{s_{n-1}})(1+\delta_n) - s_n; \quad |\delta_n| \leq \mathbf{u}
\end{aligned}$$

Symbolic Taylor forms provide an efficient way to bound this error. Trivially, e_{s_n} is zero if the error values e_{s_i}, e_{x_i}, and δ_n are all equal to zero. This justifies a Taylor expansion about zero in those error values:

$$e_{s_n} = s_n\delta_n + \sum_{j=1}^{n-1} \frac{\partial s_n}{\partial s_j} e_{s_j} + \sum_{j=1}^{m} \frac{\partial s_n}{\partial x_j} e_{x_j} + O(\mathbf{u}^2) \quad (3)$$

SATIRE's goal is to bound e_{s_n}, the *total error* of the program, which we also write $\mathcal{E}^{tr}(s_n)$. From Equation (3), we can observe that $\mathcal{E}^{tr}(s_n)$ is composed of the *local error* generated by the application of f_n, which we write $\mathcal{E}^{lr}(s_n)$, and the propagation of the incoming errors e_{x_j} and e_{s_j} [8], [14].

[1] Previous tool-based rigorous analysis for a 64-point FFT was estimated at 2K operators [13].

Given intervals $I(\mathbf{x})$ over which inputs $\mathbf{x}$ range, the **error bound** on e_{s_n}, which we write $E^{tr}(s_n)$, can be computed as follows:

$$
\begin{aligned}
E^{tr}(s_n) &\leq \max_{\mathbf{x}\in I(\mathbf{x})}(|\mathcal{E}^{tr}(s_n)|) \\
&\leq \max_{\mathbf{x}\in I(\mathbf{x})}\left(|s_n\delta_n| + |\sum_{j=1}^{n-1}\frac{\partial s_n}{\partial s_j}e_{s_j}| + |\sum_{j=1}^{m}\frac{\partial s_n}{\partial x_j}e_{x_j}|\right) + O(\mathbf{u}^2) \\
&\leq \max_{\mathbf{x}\in I(\mathbf{x})}\left(|\mathcal{E}^{lr}(s_n)| + |\sum_{j=1}^{n-1}\frac{\partial s_n}{\partial s_j}\mathcal{E}^{tr}(s_j)| + \right. \\
&\quad \left. |\sum_{j=1}^{m}\frac{\partial s_n}{\partial x_j}\mathcal{E}^{tr}(x_j)|\right) + O(\mathbf{u}^2)
\end{aligned}
\tag{4}
$$

Note the *second order* error term $O(\mathbf{u^2})$. Given that $\mathbf{u} = 2^{-53}$, n—the number of intermediate nodes in the expression tree—must approach 2^{53} for the second order error to matter, thus confirming that for most practical purposes first-order error analysis suffices.

Prior work using symbolic Taylor forms, including the FPTaylor tool, use global optimization to compute $E^{tr}(s_n)$. SATIRE introduces the following improvements: it simplifies the symbolic Taylor form using path strength reduction (§III); it uses expression canonicalization to simplify the global optimization problem (§IV); and uses abstraction to separately optimize parts of the symbolic Taylor form (§V).

III. Path Strength Reduction

We take the example of a simple unfolded 3-point stencil (Figure 1) to explain the idea of path strength reduction. Consider the problem of determining how much the local error $\mathcal{E}^{lr}(y)$ introduced at y contributes to the value at node s (location $(x, t+4)$). Expression $\mathcal{E}^{lr}(y)$ is typically a very large symbolic expression for the kinds of examples we consider, as it is obtained by performing a forward symbolic execution from primary inputs to determine the symbolic value of y, and then using Equation (1) to multiply this value by δ. FPTaylor [9] then obtains the said contribution by generating a *Symbolic Taylor Form* that ends up propagating $\mathcal{E}^{lr}(y)$ along all paths starting from y and impinging on s. In general, *there are an exponential number of such paths*—especially for examples whose dependence graphs have reconvergent paths (as Figure 1 has). This exponential behavior is encountered also when calculating the influences of nodes such as a, b, and c upon s. This directly impedes with scalability, explaining why even simple examples such as 1D-heat (§I) could not finish under FPTaylor.

A much more economic way to obtain the contribution of $\mathcal{E}^{lr}(y)$ to the value at node s *without enumerating all paths first* can be achieved by separating out the error terms from their propagation effects. In other words, we can summarize all the symbolic partial path strengths from s to y first, and then take a product with the symbolic error term generated at y, namely $\mathcal{E}^{lr}(y)$. Here, *path strength* refers to the amplification of an input change along a path.

SATIRE implements a dynamic-programming style, symbolic reverse mode algorithmic differentiation to achieve this calculation. This is also economical, since every parent node's derivative is evaluated before reaching a child node – hence derivative computation cost is shared across all nodes. To do so, as shown in Figure 1, we first compute the partial derivatives for the children of s, and then keep pushing the path strength information along their children and so on as a reduction process. In addition, symbolic expression canonicalization (discussed in §IV) is performed all through this process, keeping the expressions simplified as well as canonicalized.

For the complete path strength required at y, that is $ps_{y\to s} = ds/dy$, we compute

$$
\begin{aligned}
& ps_{y\to s} = ps_{y\to a}\cdot ps_{a\to s} + ps_{y\to b}\cdot ps_{b\to s} + ps_{y\to c}\cdot ps_{c\to s} \\
\implies & \frac{ds}{dy} = \frac{\partial a}{\partial y}\frac{\partial s}{\partial a} + \frac{\partial b}{\partial y}\frac{\partial s}{\partial b} + \frac{\partial c}{\partial y}\frac{\partial s}{\partial c}
\end{aligned}
\tag{5}
$$

where $\partial s/\partial a$, and so on, have been computed similarly and symbolically using a reverse mode backward pass beginning from the output node s. Then the total error contribution from y to s, denoted $\mathcal{E}^{tr}(s|y)$, now gets simplified by the distributive property as

$$
\mathcal{E}^{tr}(s|y) = \left|\mathcal{E}^{lr}(y)\frac{ds}{dy}\right| = \left|\mathcal{E}^{lr}(y)\left(\frac{\partial a}{\partial y}\frac{\partial s}{\partial a} + \frac{\partial b}{\partial y}\frac{\partial s}{\partial b} + \frac{\partial c}{\partial y}\frac{\partial s}{\partial c}\right)\right| \tag{6}
$$

This is done for every inner node, y_i (internal nodes and incoming error-laden inputs), in the dependence cone of s, and their symbolic partial error expressions are accumulated to obtain the global total error expression $\mathcal{E}^{tr}(s)$ as shown in equation

$$
\mathcal{E}^{tr}(s) = \sum_{y_i\in\mathbf{y}}\mathcal{E}^{tr}(s|y_i); \tag{7}
$$

where, $\mathbf{y}$ is the set of all nodes in the dependence set of s.

Notice also that path strength summaries build up proportional to the number of nodes (linearly), and not the number of paths (exponentially). This benefit directly shows up in our examples. For "1D-heat", SATIRE ends up generating and optimizing the error expression in a few seconds.

Remark: The exponentiality in Taylor-form generation is a direct consequence of generating expressions in a manner that capture higher order errors (a detailed derivation is in [15]).

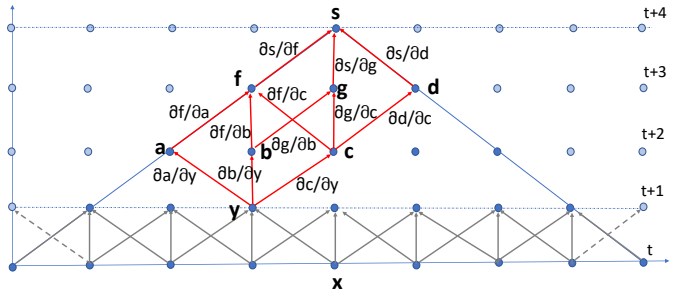

Fig. 1. Path strength information for an error propagating from **y** to **s** in an heavily interconnected network

As observed in §II, higher order error often does not matter in practice; however, in floating-point error analysis, there are almost always exceptions to every such "rule". Others [16] have observed that higher-order error analysis is quite specialized, and may require special strategies. In their work, they found that FPTaylor's own higher order analysis was not sufficiently precise for their problem.

We also tested whether FPTaylor produced a different answer—on its own benchmarks—when its second-order error estimation flag was toggled. To our surprise, we found that it emitted the same (bit-identical) results with and without second-order estimation. While this might have been due to the currently limited set of examples in FPTaylor's distribution, it still does suggest that it is difficult to make second-order error analysis matter in practice. It seems to justify our position that it is better to build a scalable tool that focuses on first-order error analysis (allowing optimizations such as path-strength reduction), leaving higher-order analysis to specialized approaches.

IV. Bound Optimization

The goal of concrete error bound calculation (Equation (4)) is to maximize the error expression over input intervals. Specifically, given our n-variable total error expression $\mathcal{E}^{tr}(s_n)$, an Interval Branch and Bound Analysis (IBBA [17]) method can search an n-dimensional box of input intervals. Each IBBA step recursively divides (modulo the number of subdivisions allowed) the initial n-dimensional box into smaller box (interval) queries to obtain the max within each subdivided interval box. This recursion bottoms out at a primitive interval library such as Gaol [18] (a component of the Gelpia optimizer we use [19]) that produces an output bound for that primitive interval. The final answer is the supremum over all n-dimensional boxes—one that produces the tighest error upper bound within a given tolerance.

If $\mathcal{E}^{tr}(s_n)$ is syntactically expressed with distinct variable occurences, the computed bound can be excessive. As a simple example, consider computing the bound for $x \cdot x \cdot x$ using an interval library where $x \in [-1, 5]$. Gaol will perform successive interval multiplications, resulting in $[-25, 125]$ as the final answer. To avoid this bloat, Gaol encourages the use of the interface function pow that is aware of variable sharings ("the same 'x' instance is being used") when one expresses the same query as $pow(x, 3)$. This results in the much tighter bound of $[-1, 125]$.

Unfortunately, Gaol cannot, by itself, accommodate all such cases through special functions such as pow, and therefore users must step in and help. This can be achieved by externally reassociating common coefficients and grouping correlated terms before invoking Gaol.[2] Take for example, the 'direct quadrature moments method' (DQMOM) benchmark captured in Equation (8), where $I(m_i) = [-1.0, 1.0]$ and parameters $w_i, a_i \in [0.00001, 1.0]$:

$$\begin{aligned} r =& (0.0 + ((((w_2 * (0.0 - m_2)) * (-3.0 * ((1.0 * (a_2/w_2)) \\ & * (a_2/w_2)))) * 1.0) + ((((w_1 * (0.0 - m_1)) * (-3.0 * ((1.0 \\ & * (a_1/w_1)) * (a_1/w_1)))) * 1.0) + ((((w_0 * (0.0 - m_0)) \\ & * (-3.0 * ((1.0 * (a_0/w_0)) * (a_0/w_0)))) * 1.0) + 0.0)))) \end{aligned} \tag{8}$$

The above non-canonicalized expression for r yields the interval **[-4.5e+10, 4.5e+10]**. However, after canonicalization, we obtain the real-value equivalent form below which noticeably reduces the distinct occurrences of w_i and a_i:

$$r = 3.0 * (a_0^2) * m_0/w_0 + 3.0 * (a_1^2) * m_1/w_1 + 3.0 * (a_2^2) * m_2/w_2$$

The interval bound now obtained is *5 orders of magnitude tighter*, namely **[-9.0e+05, 9.0e+05]**. This allows SATIRE to report the error bound of **5.0e-10** for DQMOM while FPTaylor (which does not perform canonicalization) reported **3.45e-05** (and a shadow-value computation that resulted in **3.27e-13** as the bound, confirming that SATIRE was much tighter). Please note that canonicalization only simplifies the query expression submitted to Gaol (it does not rewrite the AST being operated upon, thus not affecting floating-point error analysis).

The combination of path strength reduction and canonicalization allows SATIRE to be much faster than FPtaylor, allowing it to often handle expressions with 10K operators *even without abstraction*. In Table I we provide a subset of the comparative study performed over a suite of 47 benchmarks exported from FPTaylor's suite (we confirmed SATIRE's empirical soundness through shadow value calculations).

Benchmarks	Absolute Error		Num OPs
	SATIRE	FPTaylor	
exp1x_32	**1.12e-06**	6.15e-06	5
doppler1	3.19e-13	**1.29e-13**	10
carbonGas	**4.13e-09**	6.08e-09	21
turbine3	**6.58e-15**	1.06e-14	23
triangle	**2.50e-14**	3.12e-14	14
rigidBody2	**1.99e-11**	3.61e-11	17
predatorPrey	**8.27e-17**	1.59e-16	12
jetEngine	**6.55e-12**	1.03e-11	35
bspline3	**2.78e-17**	7.86e-17	7
dqmom	**5.0e-10***	3.45e-05	35

TABLE I
COMPARISON OF RESULTS (**bold-face** HIGHLIGHTS BETTER RESULTS, AND * HIGHLIGHTS A DIFFERENCE OF MORE THAN AN ORDER OF MAGNITUDE)

To summarize, SATIRE obtains slightly better bounds for most benchmarks in FPTaylor's suite. Also, SATIRE retains its ability to find tight bounds on smaller benchmarks (while being on the average 4.5x faster). We will now describe abstractions (§V) that help SATIRE handle much larger examples (§VI) with similar benefits obtained.

V. Abstractions

A central strategy for scalability in SATIRE is to replace entire subexpression DAGs by its root node that summarizes

[2]We direct `SymEngine` [20] to perform such canonicalizations and also expression simplifications. Any other similar engine will suffice.

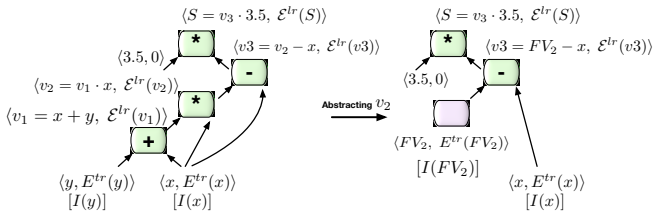

Fig. 2. Abstractions introduced in a simple expression AST

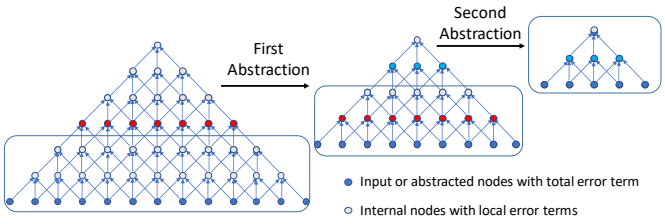

Fig. 3. Incremental error analysis using gradual abstraction

error up to that node. In Figure 2, the newly introduced input node (a "free variable" FV_2) replaces the DAG under the '*' operator. Instead of v_2 carrying the symbolic error $\mathcal{E}^{lr}(v_2)$, FV_2 carries, for the remainder of the analysis, the *concrete error* $E^{tr}(FV_2)$.[3] This has two advantages: (1) large portions of the expression graph (and their symbolic errors) do not burden the remainder of the analysis, (2) concrete errors can be treated as constants, and incorporated during further symbolic analysis, gaining even more efficiency.[4]

Using abstractions too frequently has its downside. Figure 2 illustrates how inputs such as x can flow into both the abstracted node and also "higher up." This has two disadvantages. First, variables become uncorrelated, causing the global optimizer to exaggerate error (as discussed in §IV). In our example, with $I(x) = I(y) = [-1, 1]$, abstraction obtains $FV_2 = [-2.0, 2.0]$, and the output of the '$-$' node, namely $FV_2 - x$ evaluates to $[-3, 3]$. Without abstraction, the output of the '$-$' node, namely $(x + y) * x - x$, evaluates to the tighter interval $[-2, 3]$. Second, opportunities for error cancellation (important in floating-point error analysis) are lost.

An Information-theoretic Abstraction Heuristic: We now present a practical and automated abstraction heuristic that blends a collection of competing factors. First, we recognize the fact that symbolic execution of expression DAGs with many reconvergent paths (common in FFTs, iterative solver expressions, etc.; see Figure 3) tend to blow-up expression sizes. To avoid this, we suitably incorporate the fanout of a node (number of consumers of the same expression value) into a formula (Equation (9)) detailed momentarily. Second, we recognize the fact that a global optimizer query is involved at each abstraction step, and so delayed abstractions may choke the optimizer. While this may pressure us to abstract sooner, we balance it with the risk of losing variable correlations by finding a good compromise, leading to the idea of an abstraction window that bounds the abstraction height (Equation (11)) . All this is placed on a formal footing based on ideas from Shannon's [21] work, leading to two notions: information *content* and *entropy*.

The *information content*, $I(n, h)$, of a node (n) at height h in the AST is a good relative measure for use in SATIRE (instead of the probabilistic measure used in Shannon's work; here, H is the full AST height and $0 \leq h \leq H$ measures the distance to the root). This is captured in Equation (9) which helps compute cut points for abstraction at candidate nodes n:

$$I(n, h) = -\log(h/H) \cdot (fanout(n)) \tag{9}$$

A node at the root (output node) of the AST does not needs to be abstracted and hence yields no infromation. The further away a node is from the root or larger the fanout of a node, the more fitting candidate it is for abstraction. $I(n, h)$ captures this key information quite effectively.

Next we define the cost function as an *information entropy* measure (similar to the *expectation* of information in Shannon's work). In SATIRE, we abstract all nodes at a selected height (as illustrated by Figure 3 which shows two abstraction steps at a height of 2). Let $C(h)$ define the cost in this sense, as shown in Equation (10) where $N(h)$ denotes the set of all nodes at height h. As per this formula, one can observe that h/H and $-\log(h/H)$ work in opposition, preferring h roughly "halfway":

$$C(h) = \sum_{n_i \in N(h)} \left(-\frac{h}{H} \cdot \log(h/H)\right) \cdot fanout(n_i) \tag{10}$$

In SATIRE, an *abstraction window*, define by a pair $[H_1, H_2]$, bounds candidate abstraction heights, balancing expression sizes and variable correlations, as captured by Equation (11):

$$\mathcal{H} = \underset{H_1 \leq h \leq H_2}{\arg\max} \sum_{n_i \in N(h)} \left(-\frac{h}{H_2} \cdot \log(h/H_2)\right) \cdot fanout(n_i) \tag{11}$$

Here, $\mathcal{H}$ denotes the selected abstraction height.

Last but not least, as pointed out in §I, the advantages of abstraction are far more for nonlinear benchmarks than linear benchmarks. This is an aspect not easily captured by any equation; we rely on user judgement for this aspect.

Our abstraction heuristic is designed to strike a good balance between several competing factors that affect its efficacy, while also staying fast and simple-enough to code. The efficacy of abstraction depends on expression sizes, the degree of nonlinearity, the extent of variable correlation loss, the execution time of each optimizer call, and many more such factors. The abstraction heuristic implementation is driven primarily by sub-expression sizes, node fanouts and variable correlations. We now assess the efficacy of our abstraction heuristic by comparing it against trends observed with fixed-depth abstractions.

[3]We obtain $E^{tr}(FV_2)$ by dispatching the global optimizer, as captured in Equation (4).

[4]SATIRE is neither "interval based" nor "affine based"—it is a combined interval and "symbolic-affine" analysis approach.

Comparison with Fixed Depth Abstraction: Table II presents the error bounds and execution times at fixed abstraction depths for three examples, namely FFT, Lorenz70 and Scan. These examples rely on different (and often competing) aspects of abstraction. Scan is an example type where one can obtain the same tight error bound even with small abstraction depths, without the worry of huge variable correlation losses. For such examples, a smaller abstraction depth that leads to better overall runtime is preferred.

In contrast, FFT presents a completely different scenario: larger abstraction depths lead to tighter error bounds, but with diminishing returns beyond a point—while increasing the execution time beyond that point. Instead of forcing the user to guess this critical point, SATIRE's abstraction window mechanism allows the user to suggest a range of depths. For example, a window of (10,20) suggested by the user obtains a bound of 4.52e-13 for an execution time of merely 17 seconds. It turns out that an abstraction depth of 10 is optimal for this example (finding this value through trial-and-error is impractical).

While it is tempting to treat SATIRE's global optimizer as a black box during abstraction, doing so can severely hurt performance. An optimizer query involving 2K operators returns in 26 seconds, while with 24K operators, the runtime goes up to 4450 secs (virtually exponential growth observed). As a matter of fact non-linear benchmarks exhibit higher runtime sensitivity because their path-strength derivatives are actual expressions (for linear benchmarks, these derivatives are constants). Thus, with non-linear benchmarks, one faces expression complexity *twice*: in the forward symbolic expressions and in the path-strength expressions. The forward expression is small at small depths with the reverse derivative large; and at higher depths, the opposite is true. Therefore, abstractions exercised midway in the expression tree obtain the best results, as they tend to strike a good balance between these two.

In our studies (Table II), we find that Lorenz70 can generate error expressions that choke the optimizer merely at a depth of 19, with each query taking hours. In such situations, having the ability to suggest heuristic parameters for abstraction helps guide Satire. Note that the upper and lower bounds of the abstraction window only act as hints; Satire can instantiate the actual abstraction depths "as necessary" within such windows SATIRE can automatically adjust the bounds when faced with large expression complexity (e.g., when Lorenz70 was run with an input window (20,40), Satire automatically reduced the lower bound from 20 to 18). To summarize, simple self-adjusting abstraction mechanisms eliminate the need for users from having to commit to specific choices that may prove to be highly suboptimal.

Fixed Depth	Examples					
	FFT		Lorenz70		Scan	
	err	Exec Time	err	Exec Time	err	Exec Time
8	4.34e-13	14	1.12e-11	88	9.38e-13	24
9	3.98e-13	17	1.05e-12	45	9.38e-13	18
10	4.44e-13	18	2.58e-12	152	9.38e-13	22
11	4.60e-13	18	4.56e-12	128	9.38e-13	35
12	4.52e-13	18	5.58e-12	150	9.38e-13	49
13	4.75e-13	24	1.70e-11	168	9.38e-13	61
14	4.39e-13	24	5.91e-12	627	9.38e-13	78
15	4.36e-13	28	7.31e-12	987	9.38e-13	93
16	4.43e-13	42	7.90e-12	1220	9.28e-13	109
17	4.18e-13	41	2.43e-12	3400	9.38e-13	131
18	4.18e-13	47	7.93e-13	2041	9.38e-13	142
19	4.21e-13	70			9.38e-13	137
20	4.08e-13	71	NA		9.38e-13	140
21	4.08e-13	68			9.38e-13	138

TABLE II
ERROR BOUNDS AND EXECUTION TIMES FOR THREE EXAMPLES OBTAINED WHEN PERFORMING ABSTRCATIONS FOR A SET OF FIXED DEPTH VALUES

VI. EVALUATION

Tool Overview: Figure 4 provides an overview of the various stages of SATIRE's execution. SATIRE begins by reading-in a problem definition file in a simple syntax describing expressions and input intervals, and also optional command line arguments for abstraction-specific parameters. Each symbolic variable is derived from the *var* datatype of Symengine [20]. First, SATIRE builds the full abstract syntax tree (AST). If abstraction is disabled, SATIRE initiates Direct Solve that bypasses abstraction, attempting to directly solve for the full error expression. Otherwise, the abstraction loop is entered with the specified abstraction window (defaults to $[10, 40]$, unless the user overrides this choice). All nodes at the heuristically determined abstraction height h (§V) are then abstracted by calling Direct Solve on the set of selected candidate nodes (denoted by the dashed line in figure 4. Post abstraction, the abstracted nodes are mutated to become free variables, with concrete function and error intervals. An AST that reduces in height by h is then rebuilt. If warranted, the process of abstraction continues for the remaining AST; else Direct Solve is invoked on the AST that remains at this point.

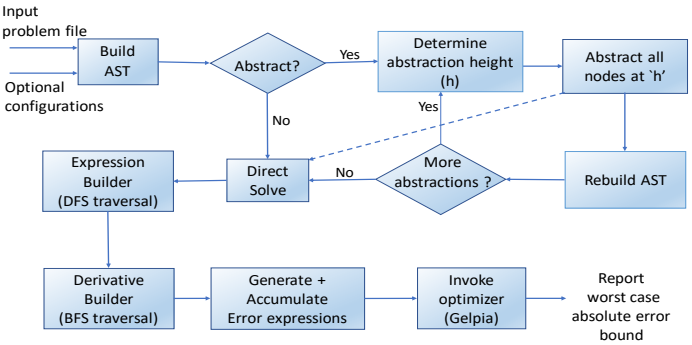

Fig. 4. Overview of SATIRE

During Direct Solve, SATIRE first invokes an expression builder to assign symbolic expression at each node, conducting expression canonicalization (§IV) in the process via the *expand* functionality of Symengine. Next, the Expression Builder performs a depth-first traversal beginning from the set of root nodes (SATIRE allows for solving multi-rooted expression

DAGs). It is followed by a breadth-first traversal to obtain the *reverse mode symbolic derivatives* (we have built this functionality as part of SATIRE).

Once both the symbolic expression and derivative are available at a node, SATIRE generates the corresponding symbolic error expression contributed by the node to each DAG output, aggregating these error expressions in an accumulator held at the outputs. The derivative evaluation and error generation are done in synchrony to avoid multiple traversals of the AST. When the output node error accumulation is finished, the results are fed to the global optimizer (Gelpia) to obtain its concrete error bound.

Experimental Setup: SATIRE is compatible with Python3, and its symbolic engine is based on `SymEngine` [20]. All benchmarks were executed with Python3.8.0 version on a dual 14-core Intel Xeon CPU E5-2680v4 2.60GHz CPUs system (total 28 processor cores) with 128GB of RAM. To arrive at an objective comparison, the core analysis algorithms were measured without any multicore parallelism (both for FPTaylor and SATIRE). The Gelpia solver (optimizer) does employ internal multithreading: we did not alter it in any way when we used either FPTaylor or SATIRE. All FPTaylor benchmarks used their specified data types.

Our main focus during evaluation will be the larger benchmarks, none of which can be rigorously analyzed by other tools. These are instantiated using the `double precision` floating-point type. For our empirical checks using shadow values, we employed GCC's `quadmath`.

Larger Benchmarks: Our larger benchmarks cover stencils (e.g., Finite-Difference Time-Domain FDTD), iterative solvers, Fast Fourier Transforms (FFT), Tensor Contractions, matrix multiplication and Lorenz system of equations (a well-known nonlinear benchmark that exhibits the "butterfly effect"). We ported the stencil kernels for heat (H*), Poisson (P*) and Convection Diffusion (C) type from [22] and unrolled them over 32 steps to be in SATIRE's input format. The data range values for these variables were obtained by scanning the input domain of their initial conditions.

For Conjugate Gradient solvers, the input matrices were obtained from [25] and analyzed over 20 iterations of the solver. Tensor contractions are extensively used in computational chemistry codes. We obtained smaller block level data for 4D tensor contractions from a CCSD [26] calculation on Uracil molecule, with the inner dimension of 35x35 for the contraction. For correctness of the contraction, the inner dimension is required to be preserved, while the outer dimensions can be divided into smaller partitions for simplifying the problem.

Table III summarizes the aforesaid large benchmarks where column Direct Solve indicates the bounds obtained without any abstractions. The "Num OPs" column shows the number of operators in a single expression tree created by unrolling the loops in these benchmarks to various sizes. The unroll factors were chosen with two objectives: (1) push SATIRE to its limits, and (2) mimic what a user of SATIRE might do in the field to understand error-buildup across iterations. We neither attempt to compute a tight loop invariant nor determine the fixed-point semantics of loops: these are known to be very difficult for floating-point computations (see discussions in §VII).

For our linear benchmarks, path strength expressions aggregate to constant factors, allowing huge scalability without abstractions. For instance, *Direct Solve* handles FDTD with $\approx 192K$ operators. However for Lorenz, the path strength expressions are quite complex symbolic expressions, and therefore abstractions were essential at around 300 operators.

Abstractions: In our experiments pertaining to abstraction, we did observe the competing forces (e.g., the effects of frequent and infrequent abstractions) mentioned in §V at play. An encouraging observation was that frequent abstractions did weaken the error bounds, but by not much—they remain in the same order of magnitude. Certain examples did suffer a lot: in FDTD for example, concretization of internal nodes during abstraction introduced larger correlation losses. Fortunately, FDTD also does not critically depend on abstractions. For such examples, correlations outweigh the need to abstract, resulting in favorable error cancellations.

We chose three candidate abstraction windows in our experiments — (10,20), (15,25) and (20,40). In a majority of the cases, smaller abstraction windows tended to estimate reasonably tight error bounds while reducing the execution time dramatically. This is especially impactful when many optimizer queries are involved. In the CCSD benchmark, SATIRE makes 26K and 12K optimizer queries for abstraction windows of [10,20] and [20,40] respectively. Although, the number of queries get halved, each query takes longer, increasing the total execution time – *while in the end obtaining the same error bound.*

We now briefly discuss whether and how semantics-preserving changes to one's code might impact abstraction. Given that such changes modify the expression DAG, round-off error bounds will also get affected correspondingly. In one simple study, a 1,024 point serial summation yielded an error bound of 2.91e-11 while a reduction-tree based summation of the same items yielded an error bound of 5.68e-13 which is two orders of magnitude tighter. In another simple study, we also compared a straightforward polynomial calculation against the Horner's method. Both Horner's method and tree reduction are known to be precision-preserving by design than their counterparts (standard polynomial evaluations and serial summation, respectively) as was confirmed by SATIRE. We also observed that for the reduction scheme, there is negligible impact due to abstraction. For Horner's method, we observe slight loss of tightness where the error bound drops from the original bound of 1.0e-13 (without abstraction) to 4.43e-13 when using an abstraction window of (10,20). In summary, these equivalence-preserving transformations appear to have the same order of magnitude of impact on the computed error bounds either with or without abstraction.

Result Reuse through Expression Hashing: We now describe a result-reuse method that saves on computations. Given that SATIRE abstracts all nodes at a selected height, for time iterative kernels such as stencils, this boils down to abstracting

Benchmarks	Num OPs	Direct Solve		Window-(10,20)		Window-(15,25)		Window-(20,40)	
		Absolute Error Bound	Execution Time (secs)	Absolute Error Bound	Execution Time (secs)	Absolute Error Bound	Execution Time (secs)	Absolute Error Bound	Execution Time (secs)
lorenz20(y)	300	NA	NA	1.25e-14	14	1.25e-14	519	1.24e-14	528
lorenz40(y)	600	NA	NA	9.11e-14	29	9.33e-14	1131	9.02e-14	1192
lorenz70(y)	1050	NA	NA	1.06e-12	50	8.14e-13	2088	8.14e-13	2150
Scan(1024pt)	2K	9.38e-13	141	9.38e-13	141	9.38e-13	141	9.38e-13	141
Scan(4096pt)	8K	4.66e-11	2822	4.66e-11	13	4.66e-11	2837	4.66e-11	2859
FFT-1024pt	81K	3.98e-13	171	4.52e-13	17	4.36e-13	28	3.98e-13	171
FFT-4096pt	393K	1.82e-12	2581	2.02e-12	68	1.935e-12	106	1.82e-12	2576
H1[a]	393K	NA	NA	1.11e-14	1322 / 274	1.11e-14	3767 / 692	1.11e-14	10579 / 2615
H2[a]	393K	NA	NA	1.94e-14	1340 / 275	1.94e-14	3792 / 694	1.94e-14	10480 / 2603
H0[a]	393K	NA	NA	5.55e-14	1334 / 280	5.55e-14	3784 / 697	5.55e-14	10472 / 2597
P1[a]	436k	NA	NA	2.26e-14	680 / 234	2.26e-14	1216 / 360	2.26e-14	2445 / 966
P2[a]	436k	NA	NA	2.26e-14	612 / 161	2.26e-14	1213 / 456	2.26e-14	2428 / 942
P0[a]	436k	NA	NA	7.55e-14	606 / 180	7.55e-14	1225 / 458	7.55e-14	2435 / 998
C1[a]	436k	NA	NA	6.25e-15	1772 / 1186	6.25e-15	4122 / 3038	6.25e-15	10875 / 7969
C2[a]	436k	NA	NA	6.25e-15	1170 / 1277	6.25e-15	4117 / 2963	6.25e-15	10766 / 7980
C0[a]	436k	NA	NA	5.56e-14	1593 / 914	5.56e-14	4164 / 2827	5.56e-14	10839 / 7946
Advection	453k	1.01e-13	3212 / 1479	1.01e-13	3218 / 1476	1.01e-13	3247 / 1623	1.01e-13	3215 / 1476
FDTD	192k	3.71e-13	8902	–	–	–	–	–	–
matmul-64x64	520K	4.76e-13	65	4.76e-13	58	4.76e-13	55	4.76e-13	54
matmul-128x128	4177K	1.86e-12	763	1.86e-12	664	1.86e-12	556	1.86e-12	527
Tensor contraction	1530K	NA	NA	2.28e-13	3676	2.28e-13	3115	2.28e-13	3142
Tensor contraction[b]	1530K	NA	NA	1.57e-19	1473	1.57e-19	953	1.57e-19	752
CG-Arc[b]	211K	3.15e-19	1206	1.23e-17	2291	1.68e-17	1032	1.54e-17	3866
CG-Pores[b]	45K	2.67e-05	9	–	–	–	–	–	–
Mol-Dyn[c]	5K	NA	NA	3.96e-14	49	4.46e-14	117	4.46e-14	117
Serial Sum	1023	2.91e-11	5407	2.91e-11	14	2.91e-11	11	2.91e-11	11
Reduction	1023	5.68e-13	32	5.68e-13	34	5.68e-13	34	5.68e-13	34
Poly-50	1325	3.26e-13	3	6.48e-13	2	6.25e-13	2	6.22e-13	1.5
Horner-50	100	1.03e-13	5	4.43e-13	2.6	2.14e-13	1.2	2.58e-13	1.3

[a] Benchmarks ported from [22] and [23]
[b] Benchmarks analyzed over degenerate intervals
[c] Benchmark realized from [24]

TABLE III
SATIRE EVALUATED WORST CASE ABSOLUTE ERROR BOUNDS AND EXECUTION TIME ON LARGER BENCHMARKS.

all nodes in the plane of the selected intermediate tile, which can be expensive in number of optimizer calls for a large tile size. However, if the analysis of such kernels is done over input ranges that are shared by multiple input variables, many of these optimizer calls will result in the same interval optimization query. This can be taken advantage of through a result-reuse method implemented through a dictionary that stores a compressed 'md5' signature of the query for each optimizer call as the key, and the previously computed result as the value. If two calls result in the same signature (modulo variable re-namings), the optimizer is bypassed and stored results are returned. For the stencil types with names H, P, C, and Advection, we report two execution times for each configuration where the numbers after the "/" indicate the execution time with this option enabled.

Degenerate intervals : SATIRE supports the ability to feed degenerate input intervals ($[m, M]$ with $m = M$) to serve three purposes all of which help improve analysis speed and/or versatility. First, this allows to propagate sparsity information by propagating 0s directly instead of interval ranges when dealing with near-sparse matrices. This greatly simplifies the analysis since the corresponding symbolic variables are replaced using constant propagation. The conjugate gradient (CG) benchmarks were executed on degenerate intervals from two sparse matrices – a computational fluid dynamics problem (Pores) and a materials problem (ARC130).

Next, it can help facilitate analysis when one encounters non-differentiable points (e.g., E_1/E_2 where E_2's interval includes 0). Given that there is no direct way to circumvent this problem, the provision for degenerate intervals allows the designer to architect selected points carefully to bypass the discontinuities. In CG, when analyzed with interval ranges, beyond two iterations, the scale factor, β, composed of the ratio of near zero residuals (from two successive iterations) encounters a zero crossing interval in the denominator reporting a divide by zero error – using degenerate intervals remove these difficulties when such situations arise.

Additionally, this facility of degenerate intervals also came in handy when studying the highly unstable Lorenz iteration scheme where we could clamp a few inputs at degenerate values and explore the remaining inputs (sensitivity analysis) across a non-degenerate interval.

Empirical consistency : We resort to extensive empirical testing to quantify the tightness and the empirical soundness of SATIRE's worst case absolute error bounds. We performed

10^6 tests across these benchmarks (except the ones subject to degenerate intervals) with random sampling over their respective input intervals. SATIRE's bounds were always found to be conservative while being on average no worse that 2 orders of magnitude. The stencil benchmarks obtain a much tighter empirical bound in the order of e-15. Prefix-sum (Scan) obtained empirical bounds only one order of magnitude tighter than the worst case bounds. The largest deviation was seen of lorenz70, where SATIRE's bound of 1.06e-12 was two orders of magnitude worse than empirically obtained maximum error of 4.1e-14.

*a) **FFT: A Case study** :* Fast Fourier transform (FFT) is an optimized algorithm for discrete Fourier transform (DFT), which converts a finite sequence of sampled points of a function into a same length sequence of an equally numbered complex-valued frequency components. It has a vast number of applications in signal processing and fast multi-precision arithmetic for large polynomial and integer multiplications.

An N-point FFT involves $\log_2 N$ stages, each stage having a familiar butterfly structure (see for example [27]). We are not aware of any tool supporting floating-point error analysis over complex domains barring [13] which was demonstrated on a small 64-point FFT design. However, its application to fast math multi-precision libraries necessitates precise floating point error analysis and has been the subject of multiple analytical studies [28]–[31]. These analysis methods focus on obtaining L2-norms in terms of root mean square (RMS) errors, or statistically profiled error bounds. Brisbarre et al. [27] followed up on the work from Percival's [29], [30] to report the best L2-norm bound till date of $\approx 30.99ulps$. That is, if z represents the discretized input samples, Z is the exact result and $\hat{Z}$ is the computed result, then, $\frac{||Z-\hat{Z}||_2}{||Z||_2} \leq B \approx 30.99\mathbf{u}$

To extend this result to a bound on absolute error corresponding to the L-infinity norm, we utilize two well-known relations: (1) Between the L2 norms of the input and outputs of an N-point FFT, i.e., $||Z||_2 \leq \sqrt{N} \cdot ||z||_2$, and (2) a generic relation between the L-infinity norm and L2-norm, i.e., $||Z||_2 \leq \sqrt{n}||Z||_\infty$. Using these relations, the L-infinity norm on the error (as obtained in [27]) of the computed FFT result can be obtained as

$$||Z-\hat{Z}||_\infty \leq B \cdot N\sqrt{2}||z||_\infty \tag{12}$$

Equation (12) obtains the absolute-error bound analytically. A 1024-point FFT with input samples in the interval $[0, 1]$ with an L2-norm bound of $B \approx 30.99\mathbf{u}$ obtains an absolute error bound of $78200\mathbf{u}$. For double precision data type, with $\mathbf{u} = 2^{-53}$, implying an absolute error bound of **4.98e-12**.

SATIRE partitions the real and imaginary parts of the complex operations in FFT, obtaining real expression types for the output variables guarded with rounding information at every compute stage. Two separate datapaths are generated for the real and imaginary terms, each of which is solved individually. Let E_R and E_I denote the absolute error bounds obtained by solving the real and imaginary parts, respectively. These solutions, on their own, provide the individual accuracy information of the real and imaginary expressions. Additionally, $E_T = \sqrt{E_R^2 + E_I^2}$ gives the bound on the maximum of the total absolute error. It is an upper bound on the L-infinity norm.

We show that SATIRE obtains a tighter bound than the analytical bound obtained by [27] as in Equation (12). We also select the input space in the interval $[0, 1]$. The bound obtained for the real and imaginary parts are $E^R \leq$ 3.98e-13 and $E^I \leq$ 3.80e-13. Thus the total error bound is $E_T \leq$ **5.5e-13** which is tighter than the best analytical bound obtained in [27]. We tried different input intervals for FFT, each time obtaining a tight bound in comparison to the analytical bounds achievable due to [27]. However, the optimizer faced convergence difficulties for intervals with zero crossings like $[-1, 1]$. In these cases, incrementally solving using abstraction of smaller depths allowed us to solve the problem while still obtaining tighter bounds than [27].

*b) **Lorenz equations: A Case Study**:* Lorenz equations model thermally induced fluid convection using three state variables (x, y, z). Here, x represents the fluid velocity amplitude, y models temperature difference between top and bottom membranes, while z represents a distortion from linearity of temperature [32]. The equation requires three additional parameters, a=10 called the Prandtl number, b=8/3 corresponding to the wave number for the convection, and r being the Rayleigh number proportional to the temperature difference. The recurrence relations obtained by discretizing the continuous version of the Lorenz equation are shown in Equation (13), where k represents the previous iteration and dt is the time discretization.

$$\begin{aligned} x_{k+1} &= x_k + a(x_k - y_k)dt \\ y_{k+1} &= y_k + (-x_k z_k + r x_k - y_k)dt \\ z_{k+1} &= z_k + (x_k y_k - b z_k)dt \end{aligned} \tag{13}$$

In [32], authors study the trajectories for different r values for chosen initial conditions of $(x_1, y_1, z_1, dt) = (1.2, 1.3, 1.6, 0.005)$. It shows chaotic behavior for $r \geq 22.35$ and again starts approaching equilibrium once r reaches close to 200. However, for such chaotic systems, if two initial conditions differ by a quantity of δ, the resulting difference after time t shows exponential separation in terms of $\delta \cdot e^{\lambda t}$. This becomes a critical component when evaluating such equations in finite precision since the round-off error accumulation introduces a gradual δ error building up.

We focus on two aspects of the analysis: (1) Obtaining bounds over a range of input intervals over (x, y, z), and (2) Analyze the sensitivity of initial conditions on individual inputs by using degenerate intervals on the other inputs.

Using SATIRE, we obtain tight bounds for as large as 70 iterations of the problem using abstractions. Note here the non-linearity of these equations makes it difficult to simplify expressions beyond a certain limit. The error expressions composed of the products of forward error and reverse derivative may reach a large operator count quickly within few iterations, choking `symEngine`'s simplification process. SATIRE delays further canonicalization beyond an operational count larger than a pre-defined limit, controlled by a parametric knob, $maxOpCount$, with default value of $30K$ selected over multiple experiments over a mix of non-linear and linear

systems. Using $maxOpCount$, we only allow simplification within the depth necessary for abstraction or otherwise force abstraction at a reduced height. During the next abstraction, further simplification take place.

Table IV shows the bounds obtained for varying windows of the abstraction depth and the corresponding execution time (exec time) in seconds.

Lorenz	Window of Abstraction depth : (mindepth, maxdepth)					
	(10,20)		(15,25)		(20,40)	
	err	Exec Time	err	Exec Time	err	Exec Time
lorenz20: x	5.69e-15		5.88e-15		5.64e-15	
lorenz20: y	1.25e-14	14s	1.25e-14	535s	1.24e-14	545s
lorenz20: z	4.75e-15		4.82e-15		4.72e-15	
lorenz40: x	3.78e-14		3.91e-14		3.75e-14	
lorenz40: y	9.11e-14	29s	9.33e-14	1131s	9.02e-14	1192s
lorenz40: z	7.08e-14		7.09e-14		6.95e-14	
lorenz70: x	3.02e-13		2.45e-13		2.45e-13	
lorenz70: y	1.06e-12	50s	7.93e-13	2088s	7.93e-13	2150s
lorenz70: z	1.05e-12		8.14e-13		8.14e-13	

TABLE IV
ERROR BOUNDS FOR THE THREE STATE VARIABLES IN THE LORENZ EQUATION

We perform analysis for 20, 40, and 70 iterations. The second state variable, y, shows more worst-case variation. Therefore, we studied the sensitivity of y for a larger size of 70 iterations using degenerate intervals in x and y, obtaining a bound of 5.50e-13 as opposed to 8.14e-13 in the non-degenerate case.

c) Application to Relative error profiling: After having obtained the worst case absolute error for an expression g over input intervals I using SATIRE, suppose the user wants to run their code, say instantiated at double precision (dp), with inputs i chosen from I and wants to obtain the worst case *relative error* estimates during that run. This can be useful for the user, as relative error directly (RE) informs about *precision loss*. RE is defined by $RE(g) = (|g - \tilde{g}|/g)$ where the numerator is the absolute error ("true-value - computed-value") and the denominator the "true-value." The SATIRE-obtained worst-case absolute error over I is an upper-bound for the numerator of $RE(g)$. The "true value" (denominator) can be obtained by instrumenting the code in higher precision (say quad-precision) and obtain g_{qp} as a close proxy to g, allowing us to compute an upper bound on $RE(g)$. The user can then estimate the number of bits lost as $\#bits\ lost = ceil(\log_2(RE(g)/\mathbf{u}))$ where $\mathbf{u}$ is the unit round-off.

Unfortunately, obtaining g_{qp} in this manner calls for code-instrumentation. Suppose we employ g_{dp} in lieu of g_{qp}: how much error are we introducing in our relative error estimation? Since g_{dp} is the observed value at runtime, there is no extra overhead to evaluate this quantity (unlike shadow value calculations). To determine that, we first obtain an equation characterizing the discrepancy in lost-bits estimation, calling it Q:

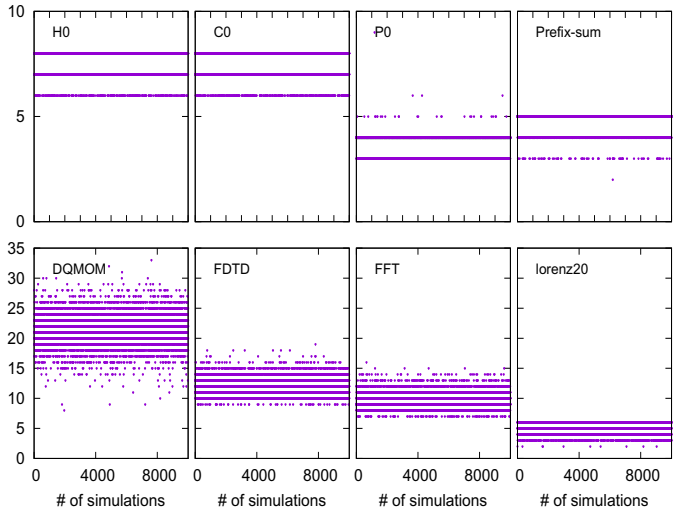

Fig. 5. x-axis: number of simulations performed for high precision shadow value evaluation; y-axis: Measured Q

$$\begin{aligned} Q &= q_{sat} - q_{shadow};\ \text{where} \\ q_{sat} &= log_2(\ \max(|g - \tilde{g}|)/(|g_{dp}| \cdot \mathbf{u})) \\ q_{shadow} &= log_2(\ \max(|g_{qp} - g_{dp}|)/(|g_{qp}| \cdot \mathbf{u})) \end{aligned} \tag{14}$$

We now estimate the spread of Q across multiple simulation runs, each time starting the simulation at an i value drawn at random from within I. For such simulations, if we plot Q, we are able to determine the spread of discrepancy in estimating the number of bits lost, shown in Figure 5, where y-axis plots the spread of Q, and the x-axis plots the simulation runs involved in generating those Qs. For this method to be (empirically) sound, Q must stay above zero. In addition, if Q stays close to 0, it would tend to confirm the tightness of the relative error estimation.

Observations about Q in our Simulations: We do find that $Q > 0$, confirming that this relative error estimation method is sound in an empirical sense. In our studies, we analyzed a subset of our benchmarks (H0, C0, P0, Prefix-sum, DQMOM, FDTD, FFT and lorenz20) as part of this case study. We chose $I = [0, 1]$, thus ensuring that it contains many binades of floating-point values (many exponent changes involved in the range $[0, 1]$). This ensures that we pick values with widely differing exponents as our candidate i, thus causing higher overall error.

Figure 5 plots the results of our empirical evaluation for 10K simulations of this subset of benchmarks. We observe that for well-conditioned problems presented in the upper half of Figure 5 (smoothing solvers such as 2-D heat—H0 in our figure), the prediction difference lies in the range of 6 to 8 bits—a good confirmation of the tightness of our estimation method. For more complex computations presented in the lower half of the figure (e.g., DQMOM, FFT, etc), the spread of Q is higher. Yet, we do see many Q values bunched up within tight bands. The lower bound of these bands indicate

the extent of tightness exhibited by our relative error profiling method (this despite our use of SATIRE's worst-case bounds in the numerator of q_{sat}).

In conclusion, the approach presented offers a promising avenue for estimating precision loss—at least in a relative sense—through the following simple steps: (1) measure the worst-case absolute error, (2) observe the runtime output value of the function being studied, and (3) apply the formula for q_{sat}. Armed with this knowledge of precision loss, a user may be able to profile their simulation to identify points at which it exhibits high relative error (e.g., as a result of the value distribution that exists at those simulation states) and mitigate the precision loss suitably—say by rewriting their function, switching to higher precision realizations, etc.

VII. Limitations, Additional Related Work

Additional Related Work: Rigorous floating-point precision analysis is central to the advancement of HPC in enabling activities such as correctness verification and precision tuning, while offering rigorous guarantees. The importance of offering formally certified bounds is well-recognized [9], [33], and scalability in this area is essential to extend the reach of certified methods. The work in [11], [12] offers another rigorous approach supported by theorem-proving. Roundoff analysis is approached using semi-definite programming in [7].

Rosa and FPTaylor are the closest to our work in their approach to extend Taylor forms [34] for rigorous floating point error estimation. Rosa propagates errors in numeric affine form and uses SMT solvers to obtain tight bounds. In FPTaylor, full symbolic Taylor forms are obtained and fed to a global optimizer, which often results in bloated error expressions, impeding scalability. In contrast, SATIRE's decoupled analysis achieves this objective albeit without this complexity, using path strength reduction and abstractions.

Fluctuat [4] is a commercially available static analyzer relying on zonotopic abstract domains. It has been primarily applied to control software where its ability to detect instability is key. While Fluctuat includes support for loops and conditionals, it is often nuanced and requires user-defined abstract domains. Their work also recognizes the difficulty of synthesizing strong loop invariants for numerical codes with roundoff incorporated, and the solutions they offer are not push-button applicable. Precisa [6] can also handle conditionals using a denotational semantics-based approach. Our emphasis in SATIRE is to create exact descriptions of finitely unrolled loops and conducting push-button analysis that yields tight error bounds across loop iterations. Eventually, a combination of loop unrolling and invariant synthesis can be effective, and a coveted future direction.

Gappa [5], while inherently an interval-based reasoning system, comes with a plethora of simplification rules of its own. Embedded into verifiers such as [35], it has provided key reasoning power to various proof-assistants [6], [36], [37]. The combined uses of static analyses [38] are popular, as demonstrated at scale in Astrée [39]. A general abstract domain for floating-point computations is described in [40].

Automatic differentiation (A/D) [41] involves chain-rule based techniques, widely used for evaluating derivatives in scientific computing. ADAPT [42] introduces a scalable approach to mixed-precision tuning for HPC applications however limited to concrete data points. It uses Codipack [43] to perform reverse mode A/D to obtain derivative values.

In contrast SATIRE implements its own symbolic library for reverse mode A/D, with analysis supported over input interval ranges for rigorously estimating the output error across the entire space of input intervals. This serves as a method for *formal specification inference* that can help future code users reliably use the code in new environments and new precision regimes—important for code adaptation that will increase in HPC. While SATIRE does not perform precision tuning, it can provide new capabilities for existing tuning assistants such as Precimonious [44], FPTuner and ADAPT which can use SATIRE to zoom into code regions and interval sub-ranges to fine-tune code accuracy in a specification-directed manner. Another interesting use of SATIRE is to direct tools such as Herbie [45] to rewrite subexpressions in a more goal-directed manner (improve precision where it lacks). The work in [46] can also derive similar benefits.

The importance of sound techniques for relative error estimation has been recognized by many. A common approach for relative error estimation is to first obtain the absolute error and then divide it by the minimum of the function interval value. A combined rigorous approach to relative error estimation is presented in [47]. Our approach for relative error estimation is meant for use in a dynamic analysis setting. It is motivated by the important pragmatic consideration of avoiding shadow-value computations on an existing piece of code. We use the observed value at runtime in combination with SATIRE's worst-case bound to obtain a relative error profile that provides insights on precision loss. Through empirical evaluation, we establish the reliability of this approach. A good contrast is also with Verificarlo [48] that has been applied at scale, but requires the provision of Monte-carlo based arithmetic operator bindings to obtain statistical precision-loss estimates. Going forward, we plan to study how close Verificarlo's relative error estimates are to ours.

Limitations Round-off analysis tools that are based on interval analysis face difficulties when given non-differentiable expressions (also pointed out in [42]). SATIRE's current solution is to rely on constant propagation based on its support for degenerate intervals, as already discussed in conjunction with our conjugate gradient and sparse matrix examples. This may be automated by employing program-level static analysis.

Higher-order errors: SATIRE's preference for first-order error analysis was discussed as directly enabling techniques such as *path strength reduction*. While this is often sufficient in practice, in our future work will explore methods to incorporate higher-order error analysis while also retaining some of the advantages of path strength reduction.

Conditionals and Loops: Conditionals can introduce control flow divergence which introduce discontinuities. Current A/D algorithms cannot handle such discontinuities and therefore

this remains an open problem for all tools in this area.

Parallelization: SATIRE's current implementation is serial. However there is ample scope for parallelization of the optimization queries. Specially, when abstracting multiple nodes, their respective queries can be potentially dispatched in parallel and/or pipelined. Furthermore, the accumulated error expression can be broken down in parallel sub-queries followed by a reduction to obtain the final error bound. Such parallelization methods, and an implementation of SATIRE in C++ or Rust can make it significantly more efficient.

VIII. Concluding Remarks

We presented SATIRE, a tool for rigorous floating-point error analysis that produces tight error bounds in practice. SATIRE is similar to many of its predecessors, but specifically emphasizes handling large expressions that arise in practice by including an information-theoretic abstraction mechanism for scalability. The effectiveness of SATIRE has been demonstrated on practical examples including FFT, parallel prefix sum, and stencils for various partial differential equation (PDE) types. Even divergent families of equations such as the Lorenz system are included in our study. SATIRE can provide insights on the loss of precision at runtime as demonstrated on a large ill-conditioned problem. We believe this variety and scale in an automated rigorous tool is unique. Our work demonstrates that scale coupled with other aspects such as nonlinearity and the DAG reconvergence structure determine scalability. Given that global optimizers are workhorses in error estimation, our studies shed light on the role of expression canonicalization. Important future directions include handling loops (enabling further scaling), improving abstractions without increasing error bounds, the use of parallelism to further speed up the analysis, and also in precision tuning.

References

[1] M. Altman, J. Gill, and M. P. McDonald, *Numerical Issues in Statistical Computing for the Social Scientist*. John Wiley & Sons, Inc., Dec. 2003. [Online]. Available: https://doi.org/10.1002/0471475769

[2] D. H. Bailey, J. M. Borwein, and V. Stodden, "Facilitating reproducibility in scientific computing: Principles and practice," in *Reproducibility: Principles, Problems, Practices, and Prospects*, H. Atmanspacher and S. Maasen, Eds., 2016, pp. 205–231.

[3] Y. Hida, X. S. Li, and D. H. Bailey, "Algorithms for quad-double precision floating point arithmetic," in *Proceedings 15th IEEE Symposium on Computer Arithmetic. ARITH-15 2001*, 2001, pp. 155–162.

[4] D. Delmas, E. Goubault, S. Putot, J. Souyris, K. Tekkal, and F. Védrine, "Towards an Industrial Use of FLUCTUAT on Safety-Critical Avionics Software," in *Formal Methods for Industrial Critical Systems, FMICS 2009*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2009, vol. 5825, pp. 53–69.

[5] M. Daumas and G. Melquiond, "Certification of Bounds on Expressions Involving Rounded Operators," *ACM Transactions on Mathematical Software*, vol. 37, no. 1, pp. 1–20, 2010.

[6] L. Titolo, M. A. Feliú, M. Moscato, and C. A. Muñoz, "An abstract interpretation framework for the round-off error analysis of floating-point programs," in *Lecture Notes in Computer Science*. Springer International Publishing, Dec. 2017, pp. 516–537. [Online]. Available: https://doi.org/10.1007/978-3-319-73721-8_24

[7] V. Magron, G. Constantinides, and A. Donaldson, "Certified roundoff error bounds using semidefinite programming," *ACM Transactions on Mathematical Software*, vol. 43, no. 4, pp. 34:1–34:31, Jan. 2017. [Online]. Available: http://doi.acm.org/10.1145/3015465

[8] E. Darulova and V. Kuncak, "Sound compilation of reals," in *Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*. ACM, 2014, pp. 235–248.

[9] A. Solovyev, M. S. Baranowski, I. Briggs, C. Jacobsen, Z. Rakamaric, and G. Gopalakrishnan, "Rigorous estimation of floating-point round-off errors with symbolic taylor expansions," *ACM Trans. Program. Lang. Syst.*, vol. 41, no. 1, pp. 2:1–2:39, 2019. [Online]. Available: https://doi.org/10.1145/3230733

[10] M. Jacquemin, S. Putot, and F. Védrine, "A reduced product of absolute and relative error bounds for floating-point analysis," in *Static Analysis - 25th International Symposium, SAS 2018, Freiburg, Germany, August 29-31, 2018, Proceedings*, ser. Lecture Notes in Computer Science, A. Podelski, Ed., vol. 11002. Springer, 2018, pp. 223–242. [Online]. Available: https://doi.org/10.1007/978-3-319-99725-4_15

[11] M. M. Moscato, L. Titolo, M. A. Feliú, and C. A. Muñoz, "Provably correct floating-point implementation of a point-in-polygon algorithm," in *Formal Methods – The Next 30 Years*, M. H. ter Beek, A. McIver, and J. N. Oliveira, Eds. Cham: Springer International Publishing, 2019, pp. 21–37.

[12] R. Salvia, L. Titolo, M. A. Feliú, M. M. Moscato, C. A. Muñoz, and Z. Rakamaric, "A mixed real and floating-point solver," in *NASA Formal Methods - 11th International Symposium, NFM 2019, Houston, TX, USA, May 7-9, 2019, Proceedings*, ser. Lecture Notes in Computer Science, J. M. Badger and K. Y. Rozier, Eds., vol. 11460. Springer, 2019, pp. 363–370. [Online]. Available: https://doi.org/10.1007/978-3-030-20652-9_25

[13] D. Boland and G. A. Constantinides, "A scalable precision analysis framework," *IEEE Transactions on Multimedia*, vol. 15, no. 2, pp. 242–256, 2013.

[14] N. J. Higham, *Accuracy and Stability of Numerical Algorithms*, 2nd ed. Society for Industrial and Applied Mathematics, 2002. [Online]. Available: https://epubs.siam.org/doi/abs/10.1137/1.9780898718027

[15] "Programming with numerical uncertainities." [Online]. Available: https://people.mpi-sws.org/~eva/papers/thesis.pdf

[16] W. Lee, R. Sharma, and A. Aiken, "On automatically proving the correctness of math.h implementations," *Proc. ACM Program. Lang.*, vol. 2, no. POPL, Dec. 2017. [Online]. Available: https://doi.org/10.1145/3158135

[17] J.-M. Alliot, N. Durand, D. Gianazza, and J.-B. Gotteland, "Finding and proving the optimum: Cooperative stochastic and deterministic search," in *Proceedings of the 20th European Conference on Artificial Intelligence (ECAI)*. ACM, 2012, pp. 55–60.

[18] F. Goualard. (2017) Gaol (not just another interval library). [Online]. Available: http://frederic.goualard.net/#research-software

[19] "Gelpia: A global optimizer for real functions," 2017. [Online]. Available: https://github.com/soarlab/gelpia

[20] "Symengine," https://github.com/symengine/symengine/.

[21] C. E. Shannon, "A mathematical theory of communication," *The Bell System Technical Journal*, vol. 27, no. 3, pp. 379–423, 7 1948. [Online]. Available: https://ieeexplore.ieee.org/document/6773024/

[22] A. Das, S. Krishnamoorthy, I. Briggs, G. Gopalakrishnan, and R. Tipireddy, "Efficient reasoning about stencil programs using selective direct evaluation," 2020.

[23] ——, "Fpdetect: Efficient reasoning about stencil programs using selective direct evaluation," *ACM Trans. Archit. Code Optim.*, vol. 17, no. 3, Aug. 2020. [Online]. Available: https://doi.org/10.1145/3402451

[24] "Molecular dynamics," https://people.sc.fsu.edu/~jburkardt/py_src/md/md.html.

[25] T. A. Davis and Y. Hu, "The university of florida sparse matrix collection," *ACM Trans. Math. Softw.*, vol. 38, no. 1, Dec. 2011. [Online]. Available: https://doi.org/10.1145/2049662.2049663

[26] J. Kim, A. Sukumaran-Rajam, C. Hong, A. Panyala, R. K. Srivastava, S. Krishnamoorthy, and P. Sadayappan, "Optimizing tensor contractions in ccsd(t) for efficient execution on gpus," in *Proceedings of the 2018 International Conference on Supercomputing*, ser. ICS '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 96–106. [Online]. Available: https://doi.org/10.1145/3205289.3205296

[27] N. Brisebarre, M. Joldes, J.-M. Muller, A.-M. Naneş, and J. Picot, "Error analysis of some operations involved in the Cooley-Tukey Fast Fourier Transform," *ACM Transactions on Mathematical Software*, pp. 1–34, 2019.

[28] G. Ramos, "Roundoff error analysis of the fast fourier transform," *Mathematics of Computation - Math. Comput.*, vol. 25, pp. 757–757, 10 1971.

[29] R. P. Brent, C. Percival, and P. Zimmermann, "Error bounds on complex floating-point multiplication," *Math. Comput.*, vol. 76, pp. 1469–1481, 2007.

[30] C. Percival, "Rapid multiplication modulo the sum and difference of highly composite numbers," *Math. Comput.*, vol. 72, no. 241, p. 387395, Jan. 2003. [Online]. Available: https://doi.org/10.1090/S0025-5718-02-01419-9

[31] E. Jr and H. Saleh, "Fft implementation with fused floating-point operations," *Computers, IEEE Transactions on*, vol. 61, pp. 284 – 288, 03 2012.

[32] J. Liang and W. Song, "Difference equation of lorenz system," *International Journal of Pure and Apllied Mathematics*, vol. 83, 02 2013.

[33] H. Becker, N. Zyuzin, R. Monat, E. Darulova, M. Myreen, and A. Fox, "A verified certificate checker for finite-precision error bounds in coq and hol4," in *FMCAD*, 10 2018, pp. 1–10.

[34] A. Neumaier, "Taylor forms—use and limits," *Reliable Computing*, vol. 9, no. 1, pp. 43–79, Feb 2003. [Online]. Available: https://doi.org/10.1023/A:1023061927787

[35] "Frama-C Software Analyzers," http://frama-c.com/index.html, 2017.

[36] G. Melquiond, "Floating-Point Arithmetic in the Coq System," *Information and Computation*, vol. 216, pp. 14–23, 2012. [Online]. Available: http://dx.doi.org/10.1016/j.ic.2011.09.005

[37] J. Harrison, "Floating-Point Verification Using Theorem Proving," in *SFM 2006*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2006, vol. 3965, pp. 211–242.

[38] E. Goubault and S. Putot, "Static Analysis of Finite Precision Computations," in *International Workshop on Verification, Model Checking, and Abstract Interpretation, VMCAI 2011*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2011, vol. 6538, pp. 232–247.

[39] P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival, "The ASTRÉE analyser," in *Proceedings of the 14th European Symposium on Programming Languages and Systems (ESOP)*, ser. Lecture Notes in Computer Science, vol. 3444. Springer, 2005, pp. 21–30.

[40] M. Martel, "Semantics of roundoff error propagation in finite precision calculations," *Higher Order Symbolic Computation*, vol. 19, no. 1, pp. 7–30, 2006. [Online]. Available: http://dx.doi.org/10.1007/s10990-006-8608-2

[41] C. Bischof, H. Buker, P. Hovland, U. Naumann, and J. Utke, Eds., *Advances in Automatic Differentiation*. Springer, 2008, iSBN : 978-3-540-68935-5.

[42] H. Menon, M. O. Lam, D. Osei-Kuffuor, M. Schordan, S. Lloyd, K. Mohror, and J. Hittinger, "Adapt: Algorithmic differentiation applied to floating-point precision tuning," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis*, ser. SC '18. IEEE Press, 2018.

[43] M. Sagebaum, T. Albring, and N. R. Gauger, "High-performance derivative computations using codipack," *CoRR*, vol. abs/1709.07229, 2017. [Online]. Available: http://arxiv.org/abs/1709.07229

[44] C. Rubio-González, C. Nguyen, H. D. Nguyen, J. Demmel, W. Kahan, K. Sen, D. H. Bailey, C. Iancu, and D. Hough, "Precimonious: Tuning assistant for floating-point precision," in *Supercomputing (SC)*, 2013, pp. 27:1–27:12, https://github.com/corvette-berkeley/precimonious.

[45] P. Panchekha, A. Sanchez-Stern, J. R. Wilcox, and Z. Tatlock, "Automatically improving accuracy for floating point expressions," in *Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2015*. ACM, 2015, pp. 1–11. [Online]. Available: http://doi.acm.org/10.1145/2737924.2737959

[46] N. Damouche, M. Martel, and A. Chapoutot, "Improving the numerical accuracy of programs by automatic transformation," *International Journal on Software Tools for Technology Transfer (STTT)*, vol. 19, no. 4, pp. 427–448, 2017. [Online]. Available: https://doi.org/10.1007/s10009-016-0435-0

[47] A. Izycheva and E. Darulova, "On sound relative error bounds for floating-point arithmetic," in *Proceedings of the 17th Conference on Formal Methods in Computer-Aided Design*, ser. FMCAD '17. Austin, Texas: FMCAD Inc, 2017, p. 15–22.

[48] C. Denis, P. de Oliveira Castro, and E. Petit, "Verificarlo: Checking floating point accuracy through monte carlo arithmetic," in *23nd IEEE Symposium on Computer Arithmetic, ARITH 2016, Silicon Valley, CA, USA, July 10-13, 2016*, P. Montuschi, M. J. Schulte, J. Hormigo, S. F. Oberman, and N. Revol, Eds. IEEE Computer Society, 2016, pp. 55–62. [Online]. Available: https://doi.org/10.1109/ARITH.2016.31