Toward Automated Precision Tuning of Weather and Climate Models: A Case Study

Jackson Vanover
University of California, Davis
Davis, USA
jdvanover@ucdavis.edu

Alper Altuntas
National Center for Atmospheric Research
Boulder, USA
altuntas@ucar.edu

Cindy Rubio-González
University of California, Davis
Davis, USA
crubio@ucdavis.edu

Abstract—**Floating-point precision tuning (FPPT) searches target programs for computations amenable to reduced-precision, thereby trading accuracy for performance. FPPT does so by searching the mixed-precision design space for program variants maximizing performance constrained by some correctness criteria. Given their computational intensity and complexity, weather and climate models present prime FPPT targets. However, past attempts at FPPT in this domain are limited by manual efforts of domain experts (tedious) and low-precision emulation (obscures speedup). *Automated* and *performance-guided* techniques are naturally of interest but have not been explored at this scale. Facilitated by a bespoke Fortran transformation tool, this paper presents a first-of-its-kind case study: based on the varied results of applying FPPT to computational hotspots in three real-world weather and climate models (MPAS-A, ADCIRC, and MOM6), we identify and discuss important lessons learned and offer insights into best practices for feasible FPPT that targets large programs in complex domains such as this.**

I. Introduction

As fabrication techniques approach the limits of transistor miniaturization and increased processor frequency, new means of performance gain are sought. Post-Moore/Dennard computing emphasizes parallelism; this is evident in the ubiquity of hardware accelerators (e.g., GPUs) and CPU instruction set architectures supporting operations on ever-wider registers (e.g., the Helium and AVX-512 vector extensions for ARM and x86 respectively). This presents an opportunity for synergy with **reduced-precision computation** [1], a performance optimization technique in which values are represented using fewer bits. Plainly put, more work can be performed with smaller data. For example, replacing 64-bit values with 32-bit values means that a single vector instruction can perform 2× the amount of computation in the same amount of time.

To optimize a program via reduced-precision is to trade excess accuracy for performance. Navigating this trade-off is the goal of **floating-point precision tuning** (FPPT). Automated dynamic-analysis-based FPPT tools implementing the cycle depicted in Figure 1 are well-represented in the literature [2], [3], [4], [5], [6], [7], [8], [9]. Provided with a target program, search space, representative input, correctness criteria, and performance metric, the tool searches the design space by generating and dynamically evaluating mixed-precision variants in order to yield one or more "optimal" variants which maximize performance subject to the correctness criteria. Crucially, however, preexisting works typically target programs that are restricted in size/complexity such as proxy applications with just a few computational hotspots that consume the majority of the runtime, e.g., LULESH [10]. This limitation hinders FPPT as a serious means of increasing performance.

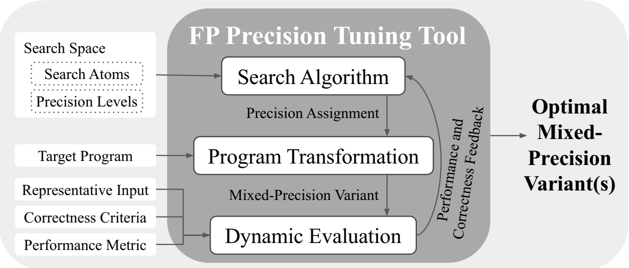

Fig. 1: An archetypal workflow for automated dynamic-analysis-based tools for floating-point precision tuning (FPPT).

By contrast, weather and climate models are orders of magnitude larger and more complex. Meteorological prediction is incredibly resource-intensive: realistic examples can consume tens of thousands of core-hours [11]. This expense is compounded since models are often run many times [12], [13]. The National Center for Atmospheric Research – host to the Derecho supercomputer – reports that 65.2% of their allocations go toward weather and climate prediction [14]. The cost of optimization is therefore amortized. Furthermore, optimization not only speeds up these models but also allows the reallocation of resources toward increased resolution and accuracy. Faster models with more predictive power improve our abilities to manage extreme weather emergencies and to draft effective environmental policy. Consequently, models are continuously under development so that they might yield more accurate results at higher resolutions at minimal cost.

FPPT studies in this domain [15], [16], [17], [18], [19], [20], [21], [22] exhibit three shortcomings. First, they require *manual* code transformation/refactoring efforts distributed across many small hotspots [23]. Second, they require a small group of domain experts at the intersection of Earth modeling systems, software engineering, numerical analysis, and Fortran (which is ubiquitous in such models). Third, many of them use low-precision emulation to transform Fortran, negating any potential performance gains [20], [21], [17], [18], [22].

There is much to gain from *automated* and *performance-guided* FPPT in this important domain. To gain initial insight

979-8-3503-5554-3/24/$31.00 ©2024 IEEE
DOI 10.1109/SCW63240.2024.00026

into the challenges, we apply the cycle depicted in Figure 1 to the MPAS-A atmosphere model [24] and the ADCIRC [25] and MOM6 [26] ocean models. Applying this cycle to any program requires making a number of choices. Below, we summarize the choices, the unique challenges presented by this domain, and our chosen approach for each. Together, these constitute the methodology for this case study:

① **Determining the search space.** n search atoms and p precision levels yield p^n possible variants. To reduce the exponential search space while maintaining tuning efficacy, we tune FP variable declarations in computationally-intensive hotspots using 32-bit and 64-bit precision (Section III-A).

② **Exploring the search space.** Brute-force searches are infeasible; to gain initial insight into challenges presented by this domain, we implement a delta-debugging-inspired algorithm for FPPT introduced in [2] and widely used in many FPPT works [7], [6], [27], [4], [28], [20] which has been shown to outperform other search algorithms with respect to the quality of the resulting variants [28] (Section III-B).

③ **Automating transformation for Fortran.** Existing tools either do not target Fortran or require an intermediate representation for which there is no robust Fortran front end. We develop a bespoke tool which uses wrappers to address Fortran's lack of implicit type conversions and which identifies, extracts, and transforms a minimal subset of the program to overcome the lack of full language support in existing compiler infrastructures targeting Fortran (Section III-C).

④ **Determining how to measure correctness.** Model correctness is often left to the best judgment of domain experts who manually inspect large amounts of multivariate time-series data. While our main contribution is a *performance*-guided case study to complement existing performance-agnostic work, we consult with a domain expert to design correctness criteria that suffice for our purposes (Section III-D).

⑤ **Determining how to measure performance.** Because existing automated dynamic-analysis-based FPPT tools target smaller programs in their entirety, the chosen metric encapsulates the time spent executing the entire program. In this case, because we target hotspots within large models, we measure the CPU time spent within the hotspot (Section III-E).

By applying the above methodology to real-world weather and climate models, we make the following contributions:

- A first-of-its-kind case study of automated, performance-guided FPPT applied to three weather and climate models: MPAS-A, ADCIRC, and MOM6.
- A variant of the MPAS-A hotspot with 1.95× speedup that incurs less error than the uniform 32-bit model.
- A discussion of lessons learned and recommendations for feasible FPPT of large programs in complex domains. Namely, we identify three criteria for a tunable hotspot and then use these criteria as the bases for recommendations toward designing/selecting targets that maximize tuning efficacy and toward statically evaluating mixed-precision variants in order to make tuning more scalable.

II. Preliminaries

A. Reduced Precision

Because reducing the precision shrinks the range of representable values and can lead to incorrect results or exceptions, it is often over-engineered using the highest precision available. However, precision reduction can also boost the performance of both memory-bound and computation-bound programs because more values can be packed into caches and vector registers. This leads to fewer expensive cache misses and increased computational throughput respectively.

Three requirements must be met to achieve performance gains from reduced precision. First is compiler/hardware synergy. Vectorization occurs only if the proper operations/-operands are supported by the hardware's ISA *and* the compiler supports that ISA. Second, the alignment and access patterns of memory must facilitate the packing of like-precision values. Irregular access patterns and poor data alignment can prevent optimizations that increase throughput. While compilers rearrange code to be optimization-friendly, programmer choices affecting control flow and data dependencies can prevent this. Third, the cost of precision conversion – i.e., **casting overhead** – must be minimized. Traditional ISA's define operations on like-precision operands. The compiler therefore generates extra conversion instructions for any mixed-precision operations in the source code.

B. Precision Tuning: A Motivating Example

Let us use a brute-force search to apply the FPPT workflow in Figure 1 to `funarc` [29]. This program performs a hard-coded arc length calculation and is often used to show the effects of precision on performance and correctness.

First, we specify the search space: search atoms are variable declarations in source code, all atoms are targeted except `result`, and we consider 64- and 32-bit precision. n search atoms at p precision levels yield p^n possible mixed-precision variants; here, our search space consists of $2^8 = 256$ variants.

Next, we parameterize the dynamic variant evaluation. Because `funarc` performs a hard-coded calculation, we do not need representative inputs. For the performance metric, we measure the elapsed time of each variant's execution. For the correctness metric, we measure the relative error of the end `result` compared to the original uniform 64-bit `funarc`.

Figure 2 plots all possible variants on a speedup-error coordinate system. One can use the optimal frontier of this plot to select a mixed-precision variant. For example, given an error threshold of 4×10^{-4}, we can see that performance is maximized with the variant yielding ∼1.3× speedup and $\sim 2 \times 10^{-4}$ relative error. This variant reduces all atoms to 32-bit except for `s1` and is almost as performant as the uniform 32-bit variant while incurring 4.5× less error. Figure 3 shows the diff between this variant and the original.

The variants on the optimal frontier show how precision reduction can trade increasing amounts of correctness for commensurate performance gains. Conversely, the variants to the left of the dotted line show how it can deteriorate both

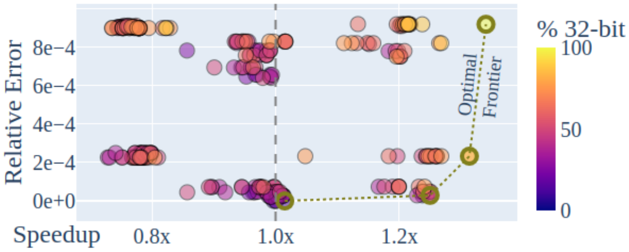

Fig. 2: Plot of `funarc` mixed-precision variants on a speedup-error coordinate system.

```
subroutine funarc(result)
-    real(kind=8) :: s1, h, t1, t2, dppi
+    real(kind=8) :: s1
+    real(kind=4) :: h, t1, t2, dppi
     ...
function fun(x) result(t1)
-    real(kind=8)  :: x, t1, d1
+    real(kind=4)  :: x, t1, d1
      ...
```

Fig. 3: The diff between the original uniform 64-bit `funarc` program and the mixed-precision variant that allows a relative error of 2×10^{-4} in exchange for a $1.3\times$ speedup.

correctness and performance: ∼67% of variants are worse than the original on both fronts despite having more 32-bit variables! This is due to excessive casting overhead.

The exponential search space presents a scalability challenge that is compounded by the cost of dynamic evaluation. The brute-force search used here is only feasible due to the program's small size, the relatively-coarse search atom granularity, the limited precision levels, and the short runtime.

C. *Precision Tuning: Weather and Climate Models*

Like many of the programs targeted by existing FPPT tools, `funarc` is idealistic. In contrast, weather and climate models contain more search atoms and longer runtimes with CPU time distributed between many hotspots [23]. The challenges brought by this are evident in the limitations of the few FPPT studies in this domain. Many do not consider performance and only demonstrate error tolerance [17], [18], [20], [22], [21]. Those that consider performance require laborious manual code transformation/refactoring by a small group of domain experts at the intersection of Earth modeling systems, software engineering, numerical analysis, and Fortran [15], [19], [16], [30], [31]. The two automated FPPT approaches we have found in this domain require unrealistic simplifications: [20] eases Fortran transformation via low-precision emulation which obscures performance; [32] reduces search space size by targeting smaller/simpler mini-apps that can be tuned in their entirety and manually clustering search atoms which requires a priori knowledge and for which no methodology is discussed.

Tuning full models is costly, yet developers undertake such efforts. This is due to both the model's high cost and the ways they are used: a single model can be rerun many times in practice, e.g., generating large ensembles with perturbations (e.g., [12]). Tuning cost is therefore naturally amortized. FPPT can offer valuable guidance by automating tedious design-space exploration and concretely measuring the performance/-correctness of hundreds of mixed-precision variants.

III. Methodology

Automated dynamic-analysis-based tools for FPPT implement the cycle shown in Figure 1. For a target program, users specify a search space, a representative input, a correctness criteria, and a performance metric. The mixed-precision design space is then systematically explored via a search algorithm which suggests precision assignments, a program transformation which generates the corresponding program variants, and a dynamic evaluation which executes variants to measure performance/correctness. When a termination criteria is reached, it returns mixed-precision variants which maximize performance subject to the correctness criteria. This cycle has not been applied to Fortran or to programs as large as weather and climate models. Accordingly, we reconsider the following.

A. *Search Space Construction*

To address the well-known scalability issues for FPPT, we adopt three means of reducing the size of the search space:

Tuning Hotspots: First, we tune computationally-intensive *hotspots*. Doing so is consistent with Amdahl's Law and a common practice for optimizing large systems [23]. Applying automated performance-guided FPPT to hotspots within full model runs is a substantial step forward compared to past manual efforts. We profile each model using the workload from the dynamic evaluation and select hotspots based on CPU-time. Our selections are corroborated by a domain expert.

Variable Declarations as Search Atoms: Second, we use FP variable declarations as the atoms of our search. This is demonstrated in Section II-B. Compared to a finer-grained approach, this offers three advantages: (1) it limits the number of search atoms and, consequently, reduces the search space magnitude, (2) it is consistent with past work on performance-guided reduced-precision computation in weather and climate models [16], [15], and (3) it results in mixed-precision code that is easily interpretable by domain expert collaborators.

Two Precision Levels: Third, we consider only 64-bit and 32-bit precision. This is the only setup that could yield speedup based on existing hardware support in supercomputer CPUs and the CPU-dominant nature of weather and climate models.

B. *Search Space Exploration*

We improve upon the $\Theta(2^n)$ complexity of a brute-force search by adopting the widely-used delta-debugging-based FPPT approach [2] which exhibits $O(nlogn)$ average case complexity and $O(n^2)$ worst case complexity [33]. This algorithm searches for a *1-minimal* variant, i.e., one possessing the smallest set of 64-bit variables for which lowering any one violates correctness or performance criteria [2], [7]. We justify our choice to reimplement this single strategy based on the goals of our study and the results of past comparisons between FPPT search algorithms. First, the goal of this case study is not to compare the efficacy of different search techniques in this domain, nor is it to propose a novel search algorithm that

```fortran
function fun_wrapper_4_to_8(x) result(output)
    real(kind=4)  :: x, output
    real(kind=8)  :: x_temp
    x_temp = x
    output = fun(x_temp)
end function fun_wrapper_4_to_8
```

Fig. 4: A simple example of the wrappers required for mixed-precision parameter-passing.

would warrant a comparison. Rather, the goal is to gain initial insight into the challenges presented by weather and climate models to FPPT in general. Accordingly, we adopt the most canonical strategy. Second, past comparisons of FPPT search algorithms show this technique is competitive [4], [28]. This is corroborated by its use in FPPT research as a baseline or as a feature of a framework [7], [6], [27], [4], [28], [20].

C. Program Transformation for Fortran

To enable dynamic evaluation, we generate mixed-precision variants via source-to-source transformations. This is consistent with past work and yields interpretable code that facilitates domain expert collaboration. Furthermore, type changes at the source level fully leverage the compiler toolchain, yielding optimizations that are missed when transforming a lower-level intermediate representation [28]. Automating source-to-source transformations of Fortran presents two challenges:

Wrappers for Mixed-Precision Parameter-Passing: The Fortran standard specifies that implicit type conversions shall occur *only via the assignment operator*, meaning that instances of mixed-precision parameter-passing require explicit wrappers. For example, while the `funarc` variant shown in Figure 3 is compilable as-is, changing the precision of `fun`'s `x` parameter to 64-bits would require the wrapper shown in Figure 4 which uses an assignment to a temporary variable with the precision required by `fun`. To accomplish this, we construct a graph whose nodes are FP variables annotated with their precisions and whose edges represent instances of parameter-passing. After applying a precision assignment, we generate wrappers to maintain the invariant that adjacent nodes should have matching annotations. For example, adding the wrapper in Figure 4 adds a node for `x_temp`, replaces "mismatching" edges between any arguments for `fun` and `x` with "matching" edges to `x_temp`, and adds a "matching" edge between `x` and `x_temp`, thus maintaining the invariant.

Overcoming Lack of Language Support in Frameworks for Source Code Transformation: To support wrapper generation, we use type information in the program's abstract syntax tree (AST) representation. For this, we use the ROSE compiler [34] which is the only tool we are aware of that offers partial support for source-to-source transformation of Fortran ASTs. However, ROSE often generates uncompilable source for unsupported language constructs. This is exacerbated by model code which is large, complex, and a mix of legacy and modern Fortran. To overcome this, our key insight is that transformations only require a subset of the full AST which contains: (1) the statements declaring target variables; (2) the statements passing target variables as arguments to procedure calls; (3) any statements defining symbols referenced in 1, 2, and recursively 3; (4) any import statements required to make the symbols defined in 1, 2, and 3 available for use where needed in 1, 2, and 3; and (5) any program structures – modules, procedures, derived-types, and so on – that contain any of 1, 2, 3. Consequently, we can automatically reduce the source code fed to ROSE using an approach analogous to a taint analysis: apply a taint to the target FP variables and iteratively apply propagation rules that capture 2-5 until a fixed point is reached. Tainted statements ultimately remain in the reduced program which is parsed into the AST, transformed, unparsed, and reinserted into the original model code.

D. Quantifying Correctness

Model correctness can be subjective and context-dependent and is often left to the best judgment of climate scientists and meteorologists who inspect the enormous amounts of multivariate time-series data output by these models (see, e.g., [16], [18], [15]). To move towards automated correctness checks, we look at typical practice in automated FPPT [27], [2], [6] which calculates a metric from output data which is then compared to the baseline to yield a relative error via the expression $|(out_{baseline} - out_{variant})/out_{baseline}|$. We collaborate with a domain expert to design a scalar metric from each model's output to use in the relative error calculation. Given the aforementioned complexity of ensuring model correctness, we note that the resulting criteria are not intended to provide a *guarantee*; rather they should effectively detect obviously bad variants which will suffice for this case study. Each model's correctness criteria is discussed in more detail in Section IV-A.

We emphasize that guaranteeing correctness is not the focus of this paper; our contribution centers on the incorporation of concrete performance information to complement existing works [17], [18], [20], [22], [21] that study correctness alone.

E. Quantifying Performance

The search algorithm considers the performance of past variants to inform the generation of new variants. Key considerations are what to dynamically measure for each variant and how to calculate a comparable metric from that quantity.

Collecting Hotspot CPU Time: Existing automated dynamic-analysis tools for FPPT target smaller programs in their entirety and thus measure the time spent executing the entire program. By contrast, this case study targets hotspots within large models whose CPU time is distributed across many smaller hotspots [23]. We therefore choose to measure the CPU time spent only within the hotspot in order to better emulate typical practice. We use the GPTL library [35] for this purpose. Note that we exclude non-targeted functions defined in the model source but do not exclude time spent in intrinsic or library functions. Timing incurs overhead from 1%-7%.

$$\text{Speedup} = \frac{\text{median}(T_{\text{baseline\_1}}, \ldots, T_{\text{baseline\_}n})}{\text{median}(T_{\text{variant\_1}}, \ldots, T_{\text{variant\_}n})} \tag{1}$$

A Noise-Tolerant Metric for Speedup: Excessive runtime variance in the target program can exacerbate a known issue where the search algorithm gets stuck in a local minimum [2].

TABLE I: Summary statistics for targeted hotspots.

Model	Targeted Module	% CPU Time	# FP Vars
MPAS-A	`atm_time_integration`	15%	445
ADCIRC	`itpackv`	12%	468
MOM6	`MOM_continuity_PPM`	9%	351

Accordingly, we define our speedup metric in Equation (1) to be parameterized by n which defines the size of a set of runs from which we use the median to remove outliers. We select the value of n based on the observed relative standard deviation in a set of 10 baseline runs of each targeted model. Any speedup greater than 1× represents improvement.

IV. Experimental Evaluation

We apply our methodology to three real-world weather and climate models in a first-of-its-kind case study with the goal of exploring the efficacy of automated, performance-guided FPPT in this important domain.

A. Experimental Setup

Experiments used 20 Derecho nodes, each with two 3rd Gen AMD 7763 processors (64 cores each @ 2.45 GHz) and 256 GB of DDR4 memory [36]. The transformation, compilation, and execution of variants is parallelized with each receiving a dedicated node. Each experiment uses the system's max job length of 12-hours. Each variant's timeout is 3× the time required for the 64-bit baseline. We consider the MPAS-A, ADCIRC, and MOM6 weather/climate models. We now describe the experimental setup specific to each model.

MPAS-A [24] is the atmosphere component of a family of Earth System Models collectively know as MPAS. It is the only model in this study engineered to support compilation with either 64-bit or 32-bit FP values; the 32-bit version is ∼1.4× faster than the 64-bit version. Here, we tune the 64-bit version. To exercise the model, we use a 5-day, global simulation described in recent publicly-available tutorials for the model [37] that uses 64 MPI ranks and runs in about 90 seconds. We target the work routines within the `atm_time_integration` module which consist of 445 FP variables and which constitute ∼15% of the CPU time. We quantify correctness by taking the relative error of the kinetic energy at the center of each cell in the domain, taking the most extreme error across all cells at each timestep, and then taking the L2-norm over time. We set the error threshold to 1.4×10^2 which is the relative error we observed for this metric between the double and single precision versions of the model provided by the developers. Because a 10-member ensemble of baseline model executions showed only a 1% relative standard deviation in performance, we set $n = 1$ for Equation (1).

ADCIRC [25] is an ocean model suited to high-resolution coastal modeling of inundation physics. To exercise the model, we use one of the examples on the ADCIRC website [38]: a 40-day tidal simulation off the coast of North Carolina, USA that uses 128 MPI ranks and runs in about 200 seconds. We target the `itpackv` module which consists of 468 FP variables and which constitutes ∼12% of the CPU time. We quantify correctness by calculating the relative error of the most extreme water surface elevation at each grid point over the course of the simulation and taking the L2-norm of these relative errors across the entire grid. This is a methodology supported by domain experts authoring the relevant literature [39]. Following the advice of a domain expert, we set the threshold to be 1.0×10^{-1}. Because a 10-member ensemble of baseline model executions showed only a 1% relative standard deviation in performance, we set $n = 1$ for Equation (1).

TABLE II: Summary metrics for variants explored. Columns 2-6 give the number of variants and the percentage that passed, failed correctness checks, timed out, or had a runtime error. The last column gives the speedup of the optimal variant.

Model	Total	Pass	Fail	Timeout	Error	Speedup
MPAS-A	48	37.5%	56.2%	6.3%	0%	1.95 ×
ADCIRC	74	36.4%	33.8%	0%	29.7%	1.12 ×
MOM6	858	17.2%	31.0%	0%	51.7%	1.04 ×

MOM6 [26] is an ocean model suited to larger spatio/temporal scales than ADCIRC. To exercise the model, we use a benchmark simulation [40] modified by a domain expert to more closely resemble a workhorse MOM6 configuration; it uses 128 MPI ranks and runs in about 60 seconds. We target the `MOM_continuity_PPM` module which contains 351 FP variables and which constitutes ∼9% of the CPU time. We quantify correctness by calculating the relative error of the maximum Courant-Friedrichs-Lewy (CFL) condition at each time step of the simulation and then taking the L2-norm over time. CFL is a stability criterion of numerical simulations and one of several global quantities used for MOM6 regression testing. Following the advice of a domain expert, we set the threshold to be 2.5×10^{-1}. Because a 10-member ensemble of baseline model executions showed a 9% relative standard deviation in performance, we set $n = 7$ for Equation (1).

B. Weather/Climate Model Hotspot Tuning

The MPAS-A search was the most successful, discovering a 1-minimal variant of the hotspot that achieved a 1.95× speedup while incurring less relative error than the supported 32-bit version of the model. Figure 5 depicts three clusters: <30% 32-bit with ≤1× speedup, >90% 32-bit with ≥1.8× speedup, and 50-89% 32-bit with speedup from 0.7-1.8×. The first two clusters support the intuition that more low-precision applied more uniformly yields increased speedup while more mixed-precision incurs excessive casting overhead and reduced vectorization. The low performance variants of the third cluster suffer from mixed-precision interprocedural data flow to a pair of `flux` functions; due to a high-volume of calls, the casting overhead increases the hotspot's CPU time by 15-22% of the baseline. *This suggests a strategy for statically evaluating variant performance via a cost model which assigns a penalty for mixed-precision interprocedural data flow as a function of the number of calls.*

Figure 6 shows the speedup of using reduced-precision in the most expensive hotspot procedures. Furthermore, note that both `atm_recover_large_step_variables_work` and `atm_advance_acoustic_step_work` have only a few unique procedure variants; this shows how quickly

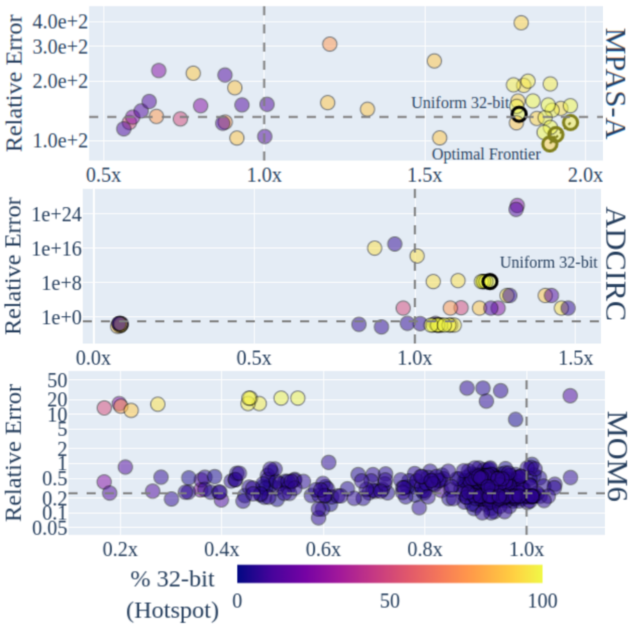

Fig. 5: Plots of mixed-precision hotspot variants on a speedup-error coordinate system. Dotted lines represent the speedup and error thresholds used by the search algorithm.

correct/performant variants were found. This contrasts with the `flux` functions and the `atm_compute_dyn_tend_work` procedure for which the large number of unique variants indicates that they presented more difficulty for the search.

Figure 6 also depicts some `flux` function variants with critical slowdown. The aforementioned casting overhead of interprocedural data flow is indirectly responsible for this; the extra conversion instructions hindered compiler optimizations by preventing function inlining. *This suggests a possible strategy for statically evaluating the performance of a variant via feedback from the compiler's optimization pass.*

For `atm_compute_dyn_tend_work`, the increased exploration ultimately led to the three variants on the optimal frontier in Figure 5 which were both more correct and more performant than uniform 32-bit; these variants lowered the precision of all variables except for either 7, 14, or 42 variables in `atm_compute_dyn_tend_work`. *This means that, with respect to our choice of performance/error metrics, FPPT was able to automatically identify a set of variables that act as a "knob" for the performance and correctness of the hotspot.*

The ADCIRC search discovered a 1-minimal variant with only one FP variable remaining in 64-bit, meeting the error threshold set by our domain expert but only achieving ∼1.1× speedup. There is no optimal frontier depicted in Figure 5 as the variants in the bottom-right quadrant are all effectively the same with respect to performance and correctness and the more performant variants in the upper-right quadrant exhibit excessive relative errors of at least 1×10^2.

Variant 42 of 74 is the first that satisfied both correctness and performance criteria. It left only 25/468 variables in high precision, all in the `jcg` procedure. While not a kernel itself, `jcg` is the driver for the hotspot's core computation and defines its key parameters. The remaining 32 variants were spent exploring `jcg` variants. Remarkably, the search ultimately identified a single parameter that must remain in 64-bit to satisfy the error threshold; otherwise, control flow substantially changes, yielding the higher speedup and intolerable error observed in the uniform 32-bit variant.

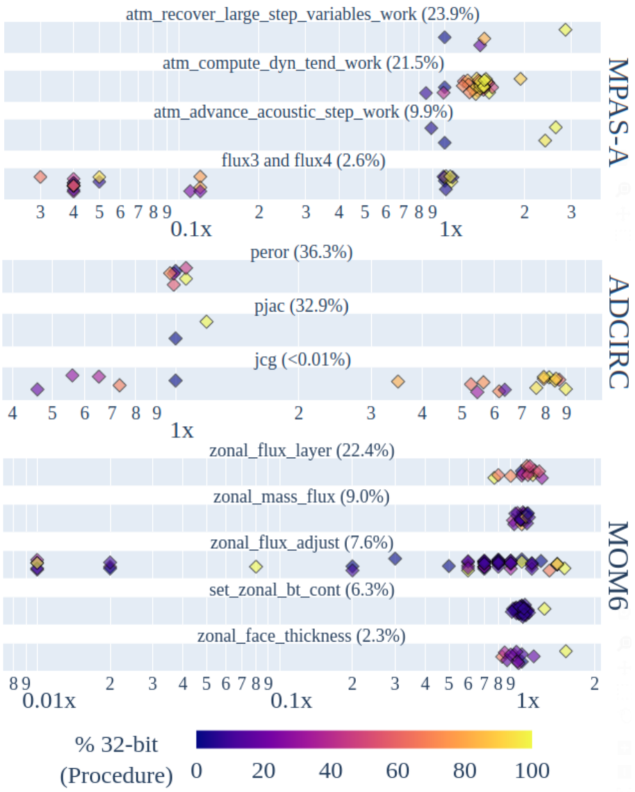

Fig. 6: Performance plots for variants of select procedures. Each marker represents a unique precision assignment of variables in that procedure. Speedup is based on the average CPU time per call and is plotted on a log axis with labeled minor ticks. Each procedure's share of the hotspot's CPU time in the 64-bit model is given in parentheses.

In Figure 6, the two most expensive procedures (`peror` and `pjac`) do not appear to benefit from reducing precision. We find two reasons for this. First, most of `peror`'s execution is spent on an `MPI_ALLREDUCE` for which many vendor implementations do not support the vectorization which is the main source of speedup when reducing precision [41]. Second, `pjac` spends its execution on a nested `for` loop which, while typically a prime candidate for vectorization, contains a data dependency that prevents this. *Together, these points suggest ensuring that targets for FPPT support vectorization to achieve the full benefit of reduced-precision.*

Lastly, note that the lowest-performing variants in Figure 5 suffered from control flow changes that led to much higher procedure execution counts, a factor not depicted in Figure 6.

The MOM6 search did not finish within the allotted

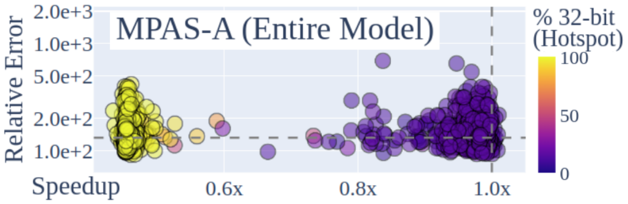

Fig. 7: Plots of the mixed-precision hotspots variants on a speedup-error coordinate system as in Figure 5. Performance of the whole model rather than hotspot is measured.

12 hours, at which point the best speedup was negligible. The MOM6 hotspot could not tolerate much low precision: of variants that were >10% 32-bit, 95% gave runtime errors. While the search did find some executable variants that were >98% 32-bit, these variants surprisingly yielded the worst slowdowns. This contrasts with MPAS-A and ADCIRC where trading correctness generally led to increased performance. We identified two causes for this. The first is kernels that perform substantially worse with mixed-precision. Particularly, two `flux_adjust` procedures that implement an iterative algorithm can take 10-100× longer to converge (Figure 6). While these procedures can be omitted, variants would still suffer for the second reason: the overhead from mixed-precision interprocedural data flow of large array variables. For example, consider variant 58 represented by the yellow marker between the two clusters in the top left quadrant of Figure 5. This variant keeps several large arrays in 64-bit precision within the `zonal_mass_flux` procedure while its callees (all depicted in Figure 6) operate on those arrays in 32-bit precision. As a result, 40% of the CPU time is spent on casting overhead. This echoes the aforementioned issue with MPAS-A's `flux` functions except the overhead is further compounded by the number of elements in these arrays. *This suggests a strategy for static performance evaluation via a cost model which assigns a penalty for cases of mixed-precision interprocedural data flow as a function of both the number of calls and the number of array elements.*

C. Impact on Overall Model Performance

Knowing that the MPAS-A hotspot can be successfully tuned with our approach, we conduct a new search guided by the wall-clock time spent executing *the entire model*.

The 1-minimal variant discovered by the search lowers the precision of 10% of the variables yielding no appreciable speedup. Figure 7 depicts two clusters of variants: high-precision variants with 0.8-1× speedup and low-precision variants with $< 0.6\times$ speedup. This is a stark contrast to the MPAS-A low-precision/high-performance variants in Figure 5 caused by the casting overhead of passing high-precision data into low-precision hotspots vastly outweighing the speedups. When MPAS-A is compiled in uniform 32-bit, this overhead does not exist. Crucially, this includes all the input data which are generated by a separate pre-processing step. This up-front and offline overhead allows the model to realize the performance benefits of reduced-precision computation.

The challenge of realizing the performance benefits of low-precision hotspots is analogous to a well-known challenge with hardware-accelerators like GPUs: despite impressive speedups with FP arithmetic on such devices versus the CPU, care must be taken to ensure that the overhead of moving data to the accelerator does not obscure the performance gains.

V. Lessons Learned

In this case study, we observed very different performance distributions with respect to the reduced-precision hotspot variants explored in each search (Figures 5 and 7). Regarding variants that were tolerably correct with respect to the chosen metrics, MOM6 yielded massive slowdown, ADCIRC yielded minimal speedup, and MPAS-A yielded high speedup that nonetheless led to slowdown in the overall model. In Section IV-B, we describe the results of manual variant analyses to explain these varied distributions. In this section, we provide a discussion of lessons learned and insights for best practices regarding feasible FPPT for larger programs in more complex domains such as this. First, we identify three key criteria for a tunable hotspot. Then, using these criteria, we provide recommendations toward designing/selecting targets that maximize tuning efficacy and toward statically evaluating mixed-precision variants in order make tuning more scalable.

Three Key Criteria for Tunable Hotspots:

1) Source code that supports compiler auto-vectorization.
2) A low volume/frequency of FP data flow between kernels within the hotspot that require different precisions.
3) A low volume/frequency of FP data flow into the hotspot.

The results and analyses described in Section IV-B make sense when viewed through this lens: We saw that the MPAS-A hotspot succeeded with respect to (1) and (2) leading to high-performance variants, the ADCIRC hotspot suffered with respect to (1) leading to variants with minimal speedup, and the MOM6 hotspots suffered with respect to (2) leading to variants with massive slowdown. In Section IV-C, measuring overall model performance showed that the MPAS-A hotspot ultimately suffers with respect to (3) leading to overall model slowdown despite higher-performance variants.

Recommendations for Scalable FPPT: We divide the discussion of these recommendations based on when they apply: at design time, just prior to tuning, and during tuning.

Designing hotspots to synergize with FPPT: One should design software to support (1). As discussed in Section II-A, memory alignment and access patterns must support the packing of like-precision values in order to support compiler auto-vectorization. Developer choices can negatively affect this. Compiler manuals offer guidance, from the design of vectorizable loops to hints provided to the compiler via pragmas.

Selecting hotspots that are amenable to FPPT: One should select hotspots which support both (1) and (3). For (1), one should check compiler vectorization reports or check assembly code for vector instructions. For (3), one could expand beyond the source code boundaries of hotspots (e.g., procedures, modules) to also optimize the "surrounding" code that moves data into and out of these hotspots to minimize casting overhead.

Work by [27] uses a static analysis on a DAG to cluster sets of operations that minimize the ratio of casting overhead to low-precision arithmetic; while their approach does not take into account execution counts and is limited to small GPU kernels, techniques like this could be powerful in this domain.

Minimizing overhead of variant evaluation during FPPT: One should use both (1) and (2) to minimize the overhead of dynamic evaluation. For (1), one could filter out variants that have less vectorization than the baseline prior to execution by inspecting compiler vectorization reports or generated assembly. For (2), one could use the same DAG that supports (3) during target selection to filter out generated variants that pass too much FP data between kernels in different precisions.

While designing code to synergize with FPPT will always be beneficial, the limitation of selecting hotspots comes as a consequence of the high overhead of dynamic variant evaluation. Minimizing this overhead could theoretically enable tuning of large programs in their entirety. Innovations in search algorithm design which avoid evaluating bad variants is needed, such as recent work [42] that uses ML to predict the performance and accuracy of mixed-precision programs.

Support for (2) and (3) requires tools for IR manipulation/-analysis to construct a DAG based on def-use and use-def chains [43], [44], [6], [27], [4]. At the time of this writing, such tools targeting the Fortran language are still in their infancy and lack the robustness of the analogous tools used by the aforementioned works for their target languages (e.g., compare Flang+LLVM to the more mature Clang+LLVM). Still, automated FPPT can be useful to model developers by guiding the investment of expensive manual efforts.

VI. Threats to Validity

Our search atoms are variable declarations, not individual uses of variables. This is consistent with FPPT tools that transform source code. Changing the precision of variable uses in source code requires substantial engineering effort; omitting this functionality does not threaten the above insights.

We do not compare against other tools/algorithms. Our goal is not to compare techniques in this new domain, nor is it to propose a novel algorithm that would warrant a comparison. Instead, we use the most canonical strategy to explore general challenges of FPPT in this important domain (Section III-B).

The generalizability of our insights are threatened by our choices of search algorithm, of programs to tune, and of hotspots to target. Respectively, we address these by selecting a widely-used search algorithm (Section III-B), by selecting real-world models (Section IV-A), and by corroborating our hotspot selection with a domain expert (Section III-A).

Our correctness checks do not guarantee model correctness. Because there is no firm consensus for model correctness short of bit-for-bit reproducibility, we collaborate with a domain expert to design criteria that test for *necessary* conditions for model correctness, not *sufficient* conditions (Section III-D).

VII. Related Work

Section II-C discussed related FPPT efforts [23], [17], [18], [20], [22], [21], [15], [19], [16], [30], [31], [20], [32] of weather and climate models in order to motivate the methodology used in this paper. Here, we provide further discussion of FPPT approaches not mentioned thus far.

Not all tools for FPPT operate on the same search atoms. Some approaches [45], [46], [8], [47], [48] tune values in single-line expressions or calls to library functions [9]. Our objective in this case study is orthogonal to theirs.

Of the FPPT tools that target variable declarations, not all of them transform source code. ADAPT [49], TypeForge [4], [44], and Blame Analysis [5] provide analyses that guide a search but do not perform the tuning themselves. Precimonious [2], HiFPTuner [6], and GPUMixer [27] all transform variables at the IR level and CRAFT [3] at machine code level.

Of the tools that lower variable precision via source-code transformations, not all of them measure performance dynamically. Some [8], [50] use static cost models for arithmetic operations while others [5], [49], [7] disregard performance by minimizing overall bit allocation. However, as shown in this study, this does not necessarily correspond to optimal performance. Furthermore, other factors that are not captured by static performance models can effect performance, e.g., compiler toolchains [28]. Dynamic measurements therefore offer the clearest feedback for the search.

The most comparable approaches to ours are FloatSmith [4] and AMPT-GA [43]. Both target variable declarations, transform code at the source level, and dynamically evaluate performance. However, they provide support for C/C++ and CUDA respectively and are therefore not applicable to Fortran.

VIII. Conclusion

In this paper, we presented and applied a methodology for automated, performance-guided FPPT to three real-world weather and climate models. Based on this first-of-its-kind case study, we identify three key criteria for tunable hotspots and offer a set of recommendations for both tuning target design and selection as well as for the static evaluation of variant performance to increase FPPT's scalability. Notably, we discover an MPAS-A hotspot variant that exhibits 1.95× speedup while incurring less error than the uniform 32-bit model. We also demonstrated the need for better tool development frameworks for Fortran, the ability of automated FPPT to foster understanding of program properties that can guide the development of mixed-precision variants, and the novel challenges that require innovation in the area of automated FPPT. Source code and data are available at https://github.com/ucd-plse/PROSE.

Acknowledgments

This work was supported by the National Science Foundation under award CCF-1750983, the U.S. Department of Energy Office of Science, Advanced Scientific Computing Research under award DE-SC0020286. We also acknowledge high-performance computing support from the Derecho system [36] provided by the NSF National Center for Atmospheric Research (NCAR), sponsored by the National Science Foundation. We would also like to thank Dolores Miao for her engineering support with early prototypes of this work.

References

[1] S. Cherubin and G. Agosta, “Tools for reduced precision computation: A survey,” *ACM Comput. Surv.*, vol. 53, no. 2, pp. 33:1–33:35, 2020. [Online]. Available: https://doi.org/10.1145/3381039

[2] 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 *International Conference for High Performance Computing, Networking, Storage and Analysis, SC'13, Denver, CO, USA - November 17 - 21, 2013*, W. Gropp and S. Matsuoka, Eds. ACM, 2013, pp. 27:1–27:12. [Online]. Available: https://doi.org/10.1145/2503210.2503296

[3] M. O. Lam, J. K. Hollingsworth, B. R. de Supinski, and M. P. LeGendre, “Automatically adapting programs for mixed-precision floating-point computation,” in *International Conference on Supercomputing, ICS'13, Eugene, OR, USA - June 10 - 14, 2013*, A. D. Malony, M. Nemirovsky, and S. P. Midkiff, Eds. ACM, 2013, pp. 369–378. [Online]. Available: https://doi.org/10.1145/2464996.2465018

[4] M. O. Lam, T. Vanderbruggen, H. Menon, and M. Schordan, “Tool integration for source-level mixed precision,” in *2019 IEEE/ACM 3rd International Workshop on Software Correctness for HPC Applications (Correctness), Denver, CO, USA, November 18, 2019*, I. Laguna and C. Rubio-González, Eds. IEEE, 2019, pp. 27–35. [Online]. Available: https://doi.org/10.1109/Correctness49594.2019.00009

[5] C. Rubio-González, C. Nguyen, B. Mehne, K. Sen, J. Demmel, W. Kahan, C. Iancu, W. Lavrijsen, D. H. Bailey, and D. Hough, “Floating-point precision tuning using blame analysis,” in *Proceedings of the 38th International Conference on Software Engineering, ICSE 2016, Austin, TX, USA, May 14-22, 2016*, L. K. Dillon, W. Visser, and L. A. Williams, Eds. ACM, 2016, pp. 1074–1085. [Online]. Available: https://doi.org/10.1145/2884781.2884850

[6] H. Guo and C. Rubio-González, “Exploiting community structure for floating-point precision tuning,” in *Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 2018, Amsterdam, The Netherlands, July 16-21, 2018*, F. Tip and E. Bodden, Eds. ACM, 2018, pp. 333–343. [Online]. Available: https://doi.org/10.1145/3213846.3213862

[7] S. Graillat, F. Jézéquel, R. Picot, F. Févotte, and B. Lathuilière, “Auto-tuning for floating-point precision with discrete stochastic arithmetic,” *J. Comput. Sci.*, vol. 36, 2019. [Online]. Available: https://doi.org/10.1016/j.jocs.2019.07.004

[8] E. Darulova, E. Horn, and S. Sharma, “Sound mixed-precision optimization with rewriting,” in *Proceedings of the 9th ACM/IEEE International Conference on Cyber-Physical Systems, ICCPS 2018, Porto, Portugal, April 11-13, 2018*, C. Gill, B. Sinopoli, X. Liu, and P. Tabuada, Eds. IEEE Computer Society / ACM, 2018, pp. 208–219. [Online]. Available: https://doi.org/10.1109/ICCPS.2018.00028

[9] H. Brunie, C. Iancu, K. Z. Ibrahim, P. Brisk, and B. Cook, “Tuning floating-point precision using dynamic program information and temporal locality,” in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2020, Virtual Event / Atlanta, Georgia, USA, November 9-19, 2020*, C. Cuicchi, I. Qualters, and W. T. Kramer, Eds. IEEE/ACM, 2020, p. 50. [Online]. Available: https://doi.org/10.1109/SC41405.2020.00054

[10] I. Karlin, J. Keasler, and J. Neely, “Lulesh 2.0 updates and changes,” Lawrence Livermore National Lab.(LLNL), Livermore, CA (United States), Tech. Rep., 2013.

[11] G. Danabasoglu, J.-F. Lamarque, J. Bacmeister, D. Bailey, A. DuVivier, J. Edwards, L. Emmons, J. Fasullo, R. Garcia, A. Gettelman *et al.*, “The community earth system model version 2 (cesm2),” *Journal of Advances in Modeling Earth Systems*, vol. 12, no. 2, 2020.

[12] A. Baker, D. Hammerling, M. Levy, H. Xu, J. Dennis, B. Eaton, J. Edwards, C. Hannay, S. Mickelson, R. Neale *et al.*, “A new ensemble-based consistency test for the community earth system model (pycect v1. 0),” *Geoscientific Model Development*, vol. 8, no. 9, pp. 2829–2840, 2015.

[13] A. H. Baker, Y. Hu, D. M. Hammerling, Y.-h. Tseng, H. Xu, X. Huang, F. O. Bryan, and G. Yang, “Evaluating statistical consistency in the ocean model component of the community earth system model (pycect v2. 0),” *Geoscientific Model Development*, vol. 9, no. 7, pp. 2391–2406, 2016.

[14] R. Kelly, “Hpc at ncar for climate and weather,” https://www.hpcuserforum.com/event/hpc-user-forum-spring-2022/, 2022.

[15] F. Váňa, P. Düben, S. Lang, T. Palmer, M. Leutbecher, D. Salmond, and G. Carver, “Single precision in weather forecasting models: An evaluation with the ifs,” *Monthly Weather Review*, vol. 145, no. 2, pp. 495–502, 2017.

[16] D. J. Milroy, A. H. Baker, J. M. Dennis, and A. Gettelman, “Investigating the impact of mixed precision on correctness for a large climate code,” in *2019 IEEE/ACM 3rd International Workshop on Software Correctness for HPC Applications (Correctness), Denver, CO, USA, November 18, 2019*, I. Laguna and C. Rubio-González, Eds. IEEE, 2019, pp. 44–51. [Online]. Available: https://doi.org/10.1109/Correctness49594.2019.00011

[17] P. D. Düben, H. McNamara, and T. N. Palmer, “The use of imprecise processing to improve accuracy in weather & climate prediction,” *Journal of Computational Physics*, vol. 271, pp. 2–18, 2014.

[18] P. D. Düben, A. Subramanian, A. Dawson, and T. Palmer, “A study of reduced numerical precision to make superparameterization more competitive using a hardware emulator in the openifs model,” *Journal of Advances in Modeling Earth Systems*, vol. 9, no. 1, pp. 566–584, 2017.

[19] J. Ackmann, P. D. Dueben, T. Palmer, and P. K. Smolarkiewicz, “Mixed-precision for linear solvers in global geophysical flows,” *Journal of Advances in Modeling Earth Systems*, vol. 14, no. 9, p. e2022MS003148, 2022.

[20] O. Tintó-Prims, M. C. Acosta, A. M. Moore, M. Castrillo, K. Serradell, A. Cortés, and F. J. Doblas-Reyes, “How to use mixed precision in ocean models: exploring a potential reduction of numerical precision in nemo 4.0 and roms 3.6,” *Geoscientific Model Development*, vol. 12, no. 7, pp. 3135–3148, 2019.

[21] S. Hatfield, M. Chantry, P. Düben, and T. Palmer, “Accelerating high-resolution weather models with deep-learning hardware,” in *Proceedings of the platform for advanced scientific computing conference*, 2019, pp. 1–11.

[22] A. Dawson, P. D. Düben, D. A. MacLeod, and T. N. Palmer, “Reliable low precision simulations in land surface models,” *Climate Dynamics*, vol. 51, pp. 2657–2666, 2018.

[23] S. Zhang, H. Fu, L. Wu, Y. Li, H. Wang, Y. Zeng, X. Duan, W. Wan, L. Wang, Y. Zhuang *et al.*, “Optimizing high-resolution community earth system model on a heterogeneous many-core supercomputing platform,” *Geoscientific Model Development*, 2020.

[24] “MPAS-A (commit 09bb84c),” https://github.com/MPAS-Dev/MPAS-Model/tree/09bb84c6b239c112103758bd31b707f0f9e56c0d, 2023.

[25] R. A. Luettich, J. J. Westerink, N. W. Scheffner *et al.*, “Adcirc: an advanced three-dimensional circulation model for shelves, coasts, and estuaries. report 1, theory and methodology of adcirc-2dd1 and adcirc-3dl,” 1992.

[26] “MOM6 (commit fd68ffa),” https://github.com/mom-ocean/MOM6/tree/fd68ffa0b537fc0814b8bce73edc530bdf2f3166, 2023.

[27] I. Laguna, P. C. Wood, R. Singh, and S. Bagchi, “Gpumixer: Performance-driven floating-point tuning for GPU scientific applications,” in *High Performance Computing - 34th International Conference, ISC High Performance 2019, Frankfurt/Main, Germany, June 16-20, 2019, Proceedings*, ser. Lecture Notes in Computer Science, M. Weiland, G. Juckeland, C. Trinitis, and P. Sadayappan, Eds., vol. 11501. Springer, 2019, pp. 227–246. [Online]. Available: https://doi.org/10.1007/978-3-030-20656-7_12

[28] K. Parasyris, I. Laguna, H. Menon, M. Schordan, D. Osei-Kuffuor, G. Georgakoudis, M. O. Lam, and T. Vanderbruggen, “Hpc-mixpbench: An HPC benchmark suite for mixed-precision analysis,” in *IEEE International Symposium on Workload Characterization, IISWC 2020, Beijing, China, October 27-30, 2020*. IEEE, 2020, pp. 25–36. [Online]. Available: https://doi.org/10.1109/IISWC50251.2020.00012

[29] D. H. Bailey, “Resolving numerical anomalies in scientific computation,” 2008.

[30] N. C. Swart, J. N. S. Cole, V. V. Kharin, M. Lazare, J. F. Scinocca, N. P. Gillett, J. Anstey, V. Arora, J. R. Christian, S. Hanna, Y. Jiao, W. G. Lee, F. Majaess, O. A. Saenko, C. Seiler, C. Seinen, A. Shao, M. Sigmond, L. Solheim, K. von Salzen, D. Yang, and B. Winter, “The canadian earth system model version 5 (canesm5.0.3),” *Geoscientific Model Development*, vol. 12, no. 11, pp. 4823–4873, 2019. [Online]. Available: https://gmd.copernicus.org/articles/12/4823/2019/

[31] “Mpas-a release notes,” https://mpas-dev.github.io/atmosphere/mpas-a_release_notes.html, 2023.

[32] J. Yao and W. Xue, “Automatic search guided code optimization framework for mixed-precision scientific applications,” in *Proceedings of the SC'23 Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis*, 2023, pp. 399–403.

[33] A. Zeller and R. Hildebrandt, "Simplifying and isolating failure-inducing input," *IEEE Trans. Software Eng.*, vol. 28, no. 2, pp. 183–200, 2002. [Online]. Available: https://doi.org/10.1109/32.988498

[34] D. Quinlan and C. Liao, "The ROSE source-to-source compiler infrastructure," in *Cetus users and compiler infrastructure workshop, in conjunction with PACT*, vol. 2011. Citeseer, 2011, p. 1.

[35] E. Hartnett and J. Rosinsk, "Using standard tools to package and distribute scientific software c and fortran libraries: a demonstration with the general purpose timing library (gptl)," *Legacy*, vol. 30, p. 4807, 2019.

[36] "Derecho: Hpe cray ex system (university community computing)," doi:10.5065/qx9a-pg09, Computational and Information Systems Laboratory, Boulder, CO: NSF National Center for Atmospheric Research, 2023.

[37] "Mpas tutorial – practice session guide," https://www2.mmm.ucar.edu/projects/mpas/tutorial/Boulder2019/index.html.

[38] "Adcirc example problems," https://adcirc.org/home/documentation/example-problems/beaufort-inlet-nc-example.

[39] J. Baugh, A. Altuntas, T. Dyer, and J. Simon, "An exact reanalysis technique for storm surge and tides in a geographic region of interest," *Coastal Engineering*, vol. 97, pp. 60–77, 2015.

[40] R. Hallberg and A. Gnanadesikan, "The role of eddies in determining the structure and response of the wind-driven southern hemisphere overturning: Results from the modeling eddies in the southern ocean (meso) project," *Journal of Physical Oceanography*, vol. 36, no. 12, pp. 2232–2252, 2006.

[41] D. Zhong, Q. Cao, G. Bosilca, and J. Dongarra, "Using long vector extensions for mpi reductions," *Parallel Computing*, vol. 109, p. 102871, 2022.

[42] Y. Wang and C. Rubio-González, "Predicting performance and accuracy of mixed-precision programs for precision tuning," in *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering, ICSE 2024, Lisbon, Portugal, April 14-20, 2024*. ACM, 2024, pp. 15:1–15:13. [Online]. Available: https://doi.org/10.1145/3597503.3623338

[43] P. V. Kotipalli, R. Singh, P. Wood, I. Laguna, and S. Bagchi, "Ampt-ga: automatic mixed precision floating point tuning for gpu applications," in *Proceedings of the ACM International Conference on Supercomputing*, 2019, pp. 160–170.

[44] "TypeForge Tool," https://github.com/LLNL/typeforge, 2021.

[45] W. Chiang, M. Baranowski, I. Briggs, A. Solovyev, G. Gopalakrishnan, and Z. Rakamaric, "Rigorous floating-point mixed-precision tuning," in *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017, Paris, France, January 18-20, 2017*, G. Castagna and A. D. Gordon, Eds. ACM, 2017, pp. 300–315. [Online]. Available: http://dl.acm.org/citation.cfm?id=3009846

[46] N. Damouche and M. Martel, "Salsa: An automatic tool to improve the numerical accuracy of programs," in *Automated Formal Methods, AFM@NFM 2017, Moffett Field, CA, USA, May 19-20, 2017*, ser. Kalpa Publications in Computing, B. Dutertre and N. Shankar, Eds., vol. 5. EasyChair, 2017, pp. 63–76. [Online]. Available: http://www.easychair.org/publications/paper/x58n

[47] R. Rabe, A. Izycheva, and E. Darulova, "Regime inference for sound floating-point optimizations," *ACM Trans. Embed. Comput. Syst.*, vol. 20, no. 5s, pp. 81:1–81:23, 2021. [Online]. Available: https://doi.org/10.1145/3477012

[48] B. Saiki, O. Flatt, C. Nandi, P. Panchekha, and Z. Tatlock, "Combining precision tuning and rewriting," in *28th IEEE Symposium on Computer Arithmetic, ARITH 2021, Lyngby, Denmark, June 14-16, 2021*. IEEE, 2021, pp. 1–8. [Online]. Available: https://doi.org/10.1109/ARITH51176.2021.00013

[49] 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, SC 2018, Dallas, TX, USA, November 11-16, 2018*. IEEE / ACM, 2018, pp. 48:1–48:13. [Online]. Available: http://dl.acm.org/citation.cfm?id=3291720

[50] D. Cattaneo, M. Chiari, N. Fossati, S. Cherubin, and G. Agosta, "Architecture-aware precision tuning with multiple number representation systems," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2021, pp. 673–678.

Appendix: Artifact Description

I. Overview of Contributions and Artifacts

A. Paper's Main Contributions

C_1 A case study of automated, performance-guided floating-point precision tuning applied to three climate and weather models.

C_2 A bespoke tool for precision tuning that facilitates the case study by applying the widely-used adaptation of the delta-debugging algorithm for precision tuning to weather and climate models written in Fortran.

B. Computational Artifacts

A_1 https://github.com/ucd-plse/PROSE[†]

Artifact ID	Contributions Supported	Related Paper Elements
A_1	C_1, C_2	Table 2 Figures 5-7

II. Artifact Identification

A. Computational Artifact A_1

Relation To Contributions

This artifact supports the paper's contributions by using the bespoke Fortran tool C_2 on the Derecho system (doi:10.5065/qx9a-pg09) to conduct the same precision tuning experiments that constitute the case study C_1. (Note that for convenience, we also provide a Docker-based artifact for partial reproduction which generates the paper's figures from the raw data and runs the tool on the `funarc` motivating example. See the repo README for more details.)

Expected Results

- Three interactive HTML visualizations reproducing Figure 5. This figure plots all explored mixed-precision variants from the delta-debugging precision tuning searches of the MPAS-A, ADCIRC, and MOM6 models on speedup-error coordinate axes.
- Three interactive HTML visualizations reproducing Figure 6. This figure plots the performance of unique mixed-precision variants of select procedures from the MPAS-A, ADCIRC, and MOM6 models that were explored in the above-mentioned precision-tuning searches.
- An interactive HTML visualization reproducing Figure 7. This figure plots the all explored mixed-precision variants from the delta-debugging precision tuning search of MPAS-A which is guided by the performance of the whole model rather than the tuned hotspot.

Expected Reproduction Time (in Minutes)

- Artifact Setup (5 minutes): After a user account/core-hour allocation is acquired for the Derecho system (doi:10.5065/qx9a-pg09), setup should take five minutes.
- Artifact Execution (12 hours): Experiments are executed in parallel. Barring non-deterministic wait time for each job submission to start running, no experiment will last longer than 12 hours.
- Artifact Analysis (30 minutes)

Artifact Setup (incl. Inputs)

Hardware: Each experiment runs on a set of 20 dedicated nodes; executing all experiments in the case study in parallel therefore requires $4 \times 20 = 80$ nodes.

Software: Note that all the listed software after Artifact A_1 are included as submodules in A_1 and therefore do not need to be acquired separately.

- Artifact A_1 (https://github.com/ucd-plse/precimonious-w-rose/tree/SC24_artifact*)
- MPAS-A + Section IV-B Experiment (https://github.com/ucd-plse/MPAS-tuning/tree/figures_5_and_6*)
- MPAS-A + Section IV-C Experiment (https://github.com/ucd-plse/MPAS-tuning/tree/figure_7*)
- MOM6 + Section IV-B Experiment (https://github.com/ucd-plse/MOM6-tuning/tree/figures_5_and_6*)
- ADCIRC + Section IV-B Experiment (https://github.com/ucd-plse/ADCIRC-tuning/tree/figures_5_and_6*)
- ROSE Compiler (https://github.com/ucd-plse/rose/tree/precimonious-w-rose*)

Datasets / Inputs: All inputs and datasets are bundled with the software artifact.

Installation and Deployment: Note that all of the below should be available on the Derecho system either as Lmod modules, as pre-installed libraries, or within a python virtual environment; everything will be loaded automatically by scripts in the software artifact.

General Dependencies

- PBS job scheduler (2021.1.3.20220217134230)
- craype (2.7.20)
- ncarenv (23.06)
- gptl (8.1.1)

Python Dependencies

- python (3.8)
- plotly (5.18.0)
- numpy (1.24)
- pandas (2.0.3)
- networkx (3.1)
- xarray (2023.1.0)
- scipy (1.10.1)

Weather/Climate Model Dependencies

- gcc (12.2.0)
- ifort (2021.8.0)

[†]Upon this paper's acceptance, a DOI will be assigned.

- netcdf (4.9.2)
- parallel-netcdf (1.12.3)
- PIO (2.6.1)
- cray-mpich (8.1.25)
- craype (2.7.20)
- hdf5 (1.12.2)

ROSE Compiler Dependencies

- flex (2.6.4)
- boost (1.67.0)
- JDK (1.8.0_241)
- gcc (7.4.0)

Artifact Execution

Automated Precision Tuning Workflow

T_0 One-time preprocessing of target: search space creation (Section III-A), construction of interprocedural floating-point flow (Section III-C), taint analysis to find minimal program to be parsed into AST for transformation (Section III-C)

T_1 The Delta-Debugging search algorithm generates a batch of potential precision assignments for the targeted floating-point variables.

T_2 In parallel, each precision assignment is applied to the model source code in order to yield mixed-precision variants.

T_3 In parallel, each mixed-precision variant is compiled, executed, and evaluated to measure performance and correctness.

T_4 Performance and correctness measurements from the batch of mixed-precision variants are fed back to the Delta-Debugging search algorithm in order to provide guidance for the next batch.

Each experiment begins by executing T_0 which takes on the order of 1% of the total time of the experiment. Then, execution iterates over the cycle $T_1 \rightarrow T_2 \rightarrow T_3 \rightarrow T_4 \rightarrow T_1 \rightarrow$ ... until the termination criteria for the Delta-Debugging algorithm is met. Specifically, this is when a *1-minimal* variant is found, i.e., a mixed-precision variant for which lowering the precision of any one variable violates the specified correctness criteria or results in a variant that is less-performant than the baseline.

For this artifact, four different instances of this automated precision tuning workflow are applied in parallel. These generate the three plots in Figures 5 and 6 as well as the single plot in Figure 7.

Additional experimental parameters specified for each experiment include what variables in the model to target for tuning, what representative input to use to exercise the model (i.e., what simulation should be run), the correctness criteria, the performance metric to be collected, and the number of times to run each mixed-precision variant. Choices for each and the corresponding justifications are provided in the Methodology (Section III) and the Experimental Setup (Section IV-A). Note that none of these need be manually-specified when executing this artifact in order to reproduce the results of the case study; we mention them here for the sake of completeness.

Artifact Analysis (incl. Outputs)

Because of the inherent non-determinism of a performance-guided search, one cannot expect bit-for-bit reproduciblity. Instead, the results of each experiment should be validated by visual inspection of generated plots and ensuring that they possess the following properties:

MPAS-A + Section IV-B

- Best speedup of $\sim 1.9\times$
- Most variants that are <30% 32-bit have $\leq 1\times$ speedup
- Most variants that are >90% 32-bit have $\geq 1.8\times$ speedup
- Most variants that are 50-89% 32-bit have 0.7-$1.8\times$ speedup
- In Figure 5, these three groups should be identifiable
- In Figure 6, many more procedure variants plotted for `atm_compute_dyn_tend_work` and the `flux` procedures compared to the `atm_recover_large_step_variables_work` and `atm_advance_acoustic_step_work` procedures
- In Figure 6, some variants of the `flux` procedures exibiting large slowdowns on the order of 0.03-$0.1\times$

ADCIRC + Section IV-B

- Best speedup of $\sim 1.1\times$
- In Figure 6, speedup on the order of 1.1-$1.2\times$ for the best `peror` and `pjac` variants
- In Figure 6, bimodal distribution for speedups of variants of the `jcg` procedure: one with $\leq 1\times$ speedup and the other with 3-$10\times$ speedup.

MOM6 + Section IV-B

- Best speedup of $< 1.1\times$
- All of the executable variants with $> 98\%$ 32-bit variables exhibit slowdowns on the order of 0.2-$0.6\times$
- In Figure 6, some variants of the `zonal_flux_adjust` procedure exhibit slowdowns on the order of 0.01-$0.1\times$

MPAS-A + Section IV-C

- Best speedup of $< 1.1\times$
- Most variants that are >90% 32-bit have <$0.6\times$ speedup
- Most variants that are <50% 32-bit have 0.8-$1\times$ speedup
- In Figure 7, these two clusters should be visible