Multi-objective optimization for Floating Point Mix-Precision Tuning

Zeqing Li
Tsinghua University
Department of Computer Science
Email: zq-li17@mails.tsinghua.edu.cn

Yongwei Wu
Tsinghua University
Department of Computer Science
Email: wuyw@tsinghua.edu.cn

Youhui Zhang
Tsinghua University
Department of Computer Science
Zhongguancun Laboratory
Beijing 100094, China
Email: zyh02@tsinghua.edu.cn

***Abstract*—This paper proposes a multi-objective optimization method for mixed-precision computation. Unlike previous studies that often take mantissa length reduction as the only optimization target, our work models the actual performance and power consumption of mixed precision programs on the corresponding hardware platforms, and based on this model searches for the pareto optimal set of all precision configurations.**

Experiments show that this tool can obtain performance improvements of $15\% - 71\%$ on floating-point benchmarks while satisfying accuracy requirements. Compared to some typical counterpart-work, an average 21% improvement can be obtained in SIMD scenarios.

I. Introduction

In high-performance computing tasks, floating-point computation is a primary means of real-value computation. As computation scales increase, the efficient employment of floating-point computation has an increasingly significant impact on overall performance [1]. While higher precision can improve output accuracy, it can also increase program runtime, energy consumption, and memory-access pressure.

However, not all programs require high-precision operations throughout to guarantee output accuracy. Ideally, applications should use as little high-precision as possible to gain performance and power advantages while maintaining output precision. Developing mixed-precision codes manually is challenging, as it requires a deep understanding of the numerical behavior of the algorithm, as well as details of floating-point rounding errors, particularly for large HPC programs.

Fortunately, many studies aim to automate the development of mixed-precision versions of a given program [2]–[5]. However, all existing studies have some limitations.

Firstly, they only use the number of variable bits as the feedback signal for optimization, and do not precisely model program performance. This approach is inconsistent with reality, mainly because SIMD instructions are widely used as an effective speedup in floating-point programs, and the acceleration effect of SIMD depends on information such as the degree of vectorization and specific hardware parameters. Thus, signals such as the number of variable bits can only provide rough trends. Additionally, the additional overhead generated at the hardware level is not considered, as some reduced precision variables and their operations will not necessarily improve performance.

Secondly, existing work lacks modeling of power consumption, which should also be a key optimization goal in some scenarios.

In this paper, we propose a method to model accuracy, performance, and power consumption by sampling on the corresponding hardware platform using automatic differentiation (AD) and Gaussian processes (GP), respectively. A multi-objective Bayesian optimization algorithm then explores the search space to find the optimal precision configuration with a guaranteed precision threshold. In the vast majority of our experiments, the tool chain yields better precision configurations compared to existing work.

II. Related Work

The field of approximate computation has a range of well-established work, and we refer the reader to a detailed survey of the entire domain [6], [7], presenting here only a small part of the work most relevant to that described in this paper.

One direction of work in mixed precision is to analyze floating-point codes to find unstable numerical computations. Some work [8], [9] give tight bounds on the rounding error based on symbolic Taylor expansions, but the precision strategy obtained is too conservative. Precimonious [3], also dedicated to numerical analysis, is the first approximate computation tool that can be applied to large programs. Nhut [10] analyzes the variables of large programs in parallel for further efficiency, and can adjust the floating-point by any number of precision bits. ADAPT [11], in order to study the effect of variable precision on the output error in a more refined way, analyzes it with automatic differentiation to obtain a more radical precision configurations. We differ from these techniques in that we go beyond static analysis of the floating-point code and model the program performance power consumption by taking into account the hardware influences.

This work was supported in part by National Natural Science Foundation of China (NSFC) under Grant No. 62250006, 62072266 and 62202254, in part by Beijing National Research Center for Information Science and Technology (BNR2022RC01003), in part by Tsinghua University Initiative Scientific Research Program, and in part by Suzhou-Tsinghua Innovation Leadership Program.

979-8-3503-1175-4/23/$31.00 ©2023 IEEE

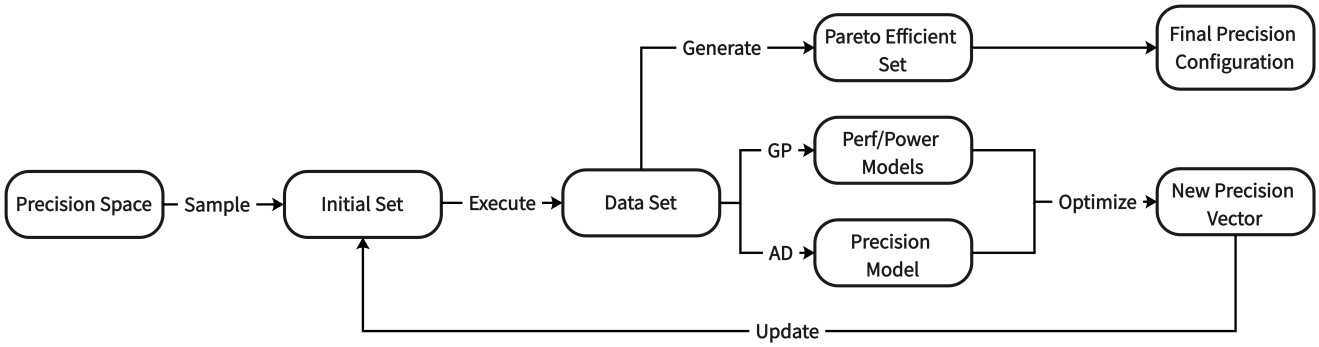

Fig. 1. Flowchart for searching mixed precision configurations. (a) Sample: A series of initial precision configurations are obtained from the precision space by a sampling algorithm. (b) Execute: These precision configurations are executed on the simulator or hardware platform to obtain the performance and power consumption corresponding to each configuration. (c) GP/AD: Modeling with Gaussian processes and automatic differentiation and using them as surrogate functions in the optimization process. (d) Optimize: Build acquisition function and use Bayesian optimization to search for better precision configurations. (e) Update: Add the new configuration to the initial set and consider it as a complete iteration after executing process b again. (f) Generate: After several iterations, a pareto efficient set is generated from the data set, and a proper precision configuration is selected according to the specific constraints.

III. Problem Formulation

Definition 1 (**Program Input Precision**). *The given program has n floating-point variables $X = \{x_1, x_2, ..., x_n\}$. The precision of each variable is inscribed by a two-tuple (exponent and mantissa), and in this paper we only consider the mantissa, such that the program precision is represented by the precision vector $P = \{p_1, p_2, ..., p_n\}$ and p_i represents the mantissa of the variable x_i.*

Definition 2 (**Output Error**). *Error of the program's output for a given precision configuration relative to the initial precision output.*

Definition 3 (**Performance**). *Performance is defined as the acceleration ratio achieved by the program in a mixed precision configuration.*

Definition 4 (**Power**). *Power consumption is defined as the sum of the power consumption of each module.*

In the present problem, there is a large correlation between output error, performance, and power consumption, and they are often not optimized simultaneously. Therefore, we introduce Pareto optimality to measure the advantages and disadvantages of the precision configuration.

Definition 5 (**Pareto Efficient Set**). *Given distinct $\boldsymbol{y}_i \in \mathbb{R}^L$ for $i = 1, \ldots, n$, we write $\boldsymbol{y}_j \succeq \boldsymbol{y}_k$ when $y_{j,l} \geq y_{k,l}$ for each $l = 1, \ldots, L$, and say " $\boldsymbol{y}_j$dominates $\boldsymbol{y}_k$ ". For the set of distinct points $\mathcal{Y} = \{\boldsymbol{y}_1, \ldots, \boldsymbol{y}_n\}$, the subset of Pareto efficient points, $\mathcal{P}(\mathcal{Y}) \subseteq \mathcal{Y}$, is defined as $\mathcal{P}(\mathcal{Y}) = \{\boldsymbol{y}_i \in \mathcal{Y} : \boldsymbol{y}_j \not\succeq \boldsymbol{y}_i \forall \boldsymbol{y}_j \in \mathcal{Y} \backslash \{\boldsymbol{y}_i\}\}$. In other words, the Pareto efficient set is the set of non-dominated points, and is always non empty.*

Problem 1 (**Floating Point Precision Tuning**). *Given a precision space $\mathcal{D}$, each precision configuration corresponds to a vector p in the space $\mathcal{D}$. Output error, performance and power consumption are the elements y in the output space Y. Precision Tuning requires finding the set of vectors $P = \{p | f(p) \in Y, y' \not\succeq f(p), \forall p \in \mathcal{D}, \forall y' \in Y\}$.*

In this article, we aim to explore the Pareto efficient set consisting of output error, performance, and power consumption under different precision configurations, and to choose the optimal configuration according to the actual output precision requirements.

IV. Methodology

A. Overview

Figure 1 illustrates the full process of the mixed precision searching algorithm. First, we sample in the precision space and optimize the sampling process using active learning and domain knowledge (IV-B). Then, it is tested in real hardware or clock-driven processor simulator to obtain the power-consumption and performance under different configurations. After obtaining the dataset, we model the accuracy using automatic differentiation (IV-C) and model the performance and power consumption with Gaussian processes (IV-D). Taking the above models as surrogate functions, we apply a multi-objective Bayesian optimization algorithm (IV-E) to search for a better precision vector and add it to the dataset. After several iterations, we find the Pareto effective set from the dataset and select the optimal precision configuration based on actual constraints.

B. Sampling

Considering the impact of sampling time and data set size on modeling speed, we aim to utilize a small but information-rich data set. Although a naive uniform sampling method exists, it is not ideal for capturing representative samples in our problem.

To address this challenge, we leverage transductive experimental design (TED), a technique that has shown significant promise in architecture design [12]. As outlined in Algorithm

1, TED constructs a distance matrix K between newly sampled vectors and unsampled spaces using a suitable distance function. We then maximize the trace of this matrix to obtain more representative sampling points [12].

To further enhance sampling performance, we incorporate domain knowledge and manually prune the sampling space. Specifically, we specify original sampling points and apply the TED algorithm to neighborhoods of these points. Our experiments demonstrate that incorporating domain knowledge can lead to a 13% reduction in the number of required samples on average (as shown in Algorithm 1).

Algorithm 1 Sampling algorithm

Input: D is the unsampled space, α is the normalization factor, N is the amount of sampled data, and C is the set of artificially introduced sampling points.
Output: S is the sampled data set with $|S| = N$.

$S \leftarrow \emptyset$
$D_i \leftarrow$ neighborhood of $c_i \in C$
for x in D_i **do**
    $S_i \leftarrow TED(x, \alpha, \lfloor N/|C| \rfloor)$
    $S \leftarrow S \cup S_i$
end for
Procedure $TED(D, \alpha, N)$
Input: K is the metric matrix.
$s \leftarrow \emptyset$
for $i = 1$ to N **do**
    $s_* \leftarrow argmaxTr[K_{Ds}(K_{ss} + \alpha I)^{-1} K_{sD}]$
    $s \leftarrow s \cup s_*$, $D \leftarrow D \backslash s_*$
    $K \leftarrow K - K_{Ds_*}(K_{s_*s_*} + \alpha I)^{-1} K_{s_*D}$
end for
return s
EndProcedure

C. Output Error Estimation Model

We construct our error model by approximating the program with a first-order Taylor series, where the output error is assumed to be linear in the rounding error. Specifically, we consider the input of the program, denoted by x, and the output, denoted by y, which together can be viewed as a function f that satisfies $y = f(x)$. Assuming a small perturbation Δx in the input caused by a change in input precision, we can estimate the output error of the program using the following equation: $\Delta y \approx \left| f'(a)^T \overline{\Delta x} \right|$.

To measure the error of a specific precision, we use the absolute error due to rounding in variable x, which is less than $|x|\epsilon$, where $\epsilon = 2^{-p}$ and p represents the number of bits in the mantissa for that precision [11]. We define a metric $\mathcal{E}$ to capture the sensitivity of any input or intermediate variable to rounding errors:

$$\mathcal{E}_x = |f'(x) \times x| \tag{1}$$

D. Performance and Power Models

Research on building reliable prediction models quickly for a given initial training set has been a popular topic in recent years. Gaussian process (GP) models have gained significant attention due to their non-parametric and robust nature, which have contributed to their success in traditional optimization problems. In architecture design, GP has been successfully applied in several studies. For instance, in the design of 64-bit prefix adders, GP was used to efficiently and effectively explore the parameter space of EDA tools [13]. Similarly, BOOM-Explorer [14] uses GP to characterize the design space in microarchitecture design. Experiments show that this method can accurately fit processor performance and further derive optimal microarchitecture parameters.

In light of these successes, we aim to employ GP to construct performance and power consumption models. We assume that we have feature vectors $\boldsymbol{X}$ that correspond to a set of power or performance values $\boldsymbol{y}$. GP provides a prior over the value function f as $f(\boldsymbol{x}) \sim \mathcal{GP}(\mu, k_{\boldsymbol{\theta}})$, where μ represents the mean value and the kernel function k is parameterized by $\boldsymbol{\theta}$. Using this prior, we can construct Gaussian distributions for any collection of value functions f, as specified by the following equation:

$$\boldsymbol{f} = [f(\boldsymbol{x}_1), f(\boldsymbol{x}_2), \ldots f(\boldsymbol{x}_n)]^T \sim \mathcal{N}\left(\boldsymbol{\mu}, \mathrm{K}_{\boldsymbol{XX}|\boldsymbol{\theta}}\right) \tag{2}$$

where $\mathrm{K}_{XX|\boldsymbol{\theta}}$ is the intra-covariance matrix among all feature vectors and $\left[\mathrm{K}_{\boldsymbol{XX}|\boldsymbol{\theta}}\right]_{ij} = k_{\boldsymbol{\theta}}(\boldsymbol{x}_i, \boldsymbol{x}_j)$. Thus, given a newly sampled feature vector $\boldsymbol{x}_*$, the predictive joint distribution f_* that depends on $\boldsymbol{y}$ can be calculated according to Equation:

$$f_* \mid \boldsymbol{y} \sim \mathcal{N}\left(\begin{bmatrix} \boldsymbol{\mu} \\ \mu_* \end{bmatrix}, \begin{bmatrix} \mathrm{K}_{\boldsymbol{XX}|\boldsymbol{\theta}} + \sigma_e^2 \boldsymbol{I} & \mathrm{K}_{\boldsymbol{X}\boldsymbol{x}_*|\boldsymbol{\theta}} \\ \mathrm{K}_{\boldsymbol{x}_*\boldsymbol{X}|\boldsymbol{\theta}} & k_{\boldsymbol{x}_*\boldsymbol{x}_*|\boldsymbol{\theta}} \end{bmatrix}\right) \tag{3}$$

In general, the performance of GP depends on the expressiveness of the kernel function $k_{\boldsymbol{\theta}}$. The common kernel functions are linear kernels, polynomial kernels, and radial kernels, etc. DNN is used in BOOM-Explorer [14] to find the optimal gaussian kernel. However, in the context of this particular problem, our testing revealed that although DNN-based methods have the potential to attain higher prediction accuracy, they are not necessarily capable of generating optimal precision configurations. This can be attributed to the restricted range of variable precision options, which are limited to 16-32-64 bits. Upon careful consideration of factors such as time overhead, training data size, and prediction accuracy, we have decided to adopt the ARD Matern 5/2 kernel, which is defined as

$$\begin{gathered} k_{M52}(\boldsymbol{x}, \boldsymbol{x}') = \theta_0^2 \left(1 + \sqrt{5r^2} + \tfrac{5}{3} r^2\right) \exp\left(-\sqrt{5r^2}\right) \\ r^2 = \textstyle\sum_{d=1}^{D} (x_d - x'_d)^2 / \theta_d^2 \end{gathered} \tag{4}$$

E. Multi-objective Optimization

Next we need to use Bayesian optimization to obtain the Pareto set based on the above model. Suppose the sampled precision configurations are $X = \{x_1, x_2, ..., x_t\}$ and the model to evaluate them is $y_i = f(x_i)$. Our goal is to find a new precision configuration x_{t+1} such that $reward = f(x_{t+1}) - f(x_*)$

is maximized. This is very simple with a single objective, but in this problem a better precision configuration should not only have higher output accuracy, but also lower power consumption and runtime. There is a clear correlation between these multiple optimization objectives before, so how to generalize the single-objective optimization framework to the multi-objective case is the main problem we face.

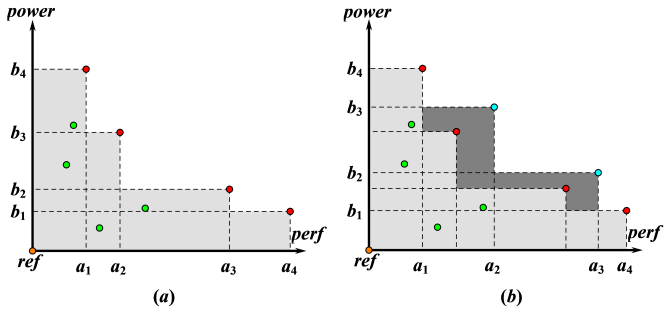

Fig. 2. Two-dimensional schematic of the Pareto hypervolume when only performance and precision are considered. (a) The four red points represent the currently obtained pareto-optimal set, and the shaded area is their PV relative to the origin. The green points included in it are the suboptimal precision configuration, dominated by the red points. (b) The EIPV is the expectation of the area of the dark region, and the blue points are candidates for a more optimal configuration. Algorithm 2 selects new sampling points by maximizing EIPV.

To solve this problem, we introduce the concept of Pareto hypervolume (PV) [15], [16]. Given a set of distinct points $\mathcal{Y} = \{\boldsymbol{y}_1, \ldots, \boldsymbol{y}_n\}$, we have defined its Pareto efficient subset, $\mathcal{P}(\mathcal{Y})$. Define a reference point, $v_{\text{ref}} \in \mathbb{R}^L$, which is dominated by each element of $\mathcal{P}(\mathcal{Y})$ i.e. $\boldsymbol{u} \succeq \boldsymbol{v}_{\text{ref}}$ for each $\boldsymbol{u} \in \mathcal{P}(\mathcal{Y})$. The Pareto hypervolume of $\mathcal{P}(\mathcal{Y})$ with respect to v_{ref} is

$$\text{Vol}_{\boldsymbol{v}_{\text{ref}}} \mathcal{P}(\mathcal{Y})) = \int_{\mathbb{R}^L} \mathbb{I}\left[\boldsymbol{y} \succeq \boldsymbol{v}_{\text{ref}}\right] \left[1 - \prod_{\boldsymbol{u} \in \mathcal{P}(\mathcal{Y})} \mathbb{I}[\boldsymbol{u} \not\succeq \boldsymbol{y}]\right] d\boldsymbol{y} \tag{5}$$

where $\mathbb{I}(.)$ is the indicator function, which outputs 1 if its argument is true and 0 otherwise.

In this way, we convert the problem of simultaneously optimizing the three objectives into optimizing the super volume of these three variables relative to the reference point. By definition a larger PV indicates a better configuration, so the optimal configuration point should maximize the total increment of PV, i.e., maximize the following expression

$$r_T = \text{Vol}_{\boldsymbol{v}_{\text{ref}}} \left(\mathcal{P}\left(\tilde{\mathcal{Y}}_T\right)\right) - \text{Vol}_{\boldsymbol{v}_{\text{ref}}} \left(\mathcal{P}\left(\mathcal{Y}^*\right)\right) \tag{6}$$

where $\mathcal{Y}^*$ is the true Pareto frontier and $\tilde{\mathcal{Y}}_T$ is the suggested Pareto frontier after T evaluations of each of the objectives.

Unfortunately, the variable proves to be computationally infeasible when T is sufficiently large. In this problem, we use the expected improvement in Pareto hypervolume introduced by [17]. In other words, our goal is to maximize the area of the dark shadows in Figure 2.

Algorithm 2 Mixed-precision Search Algorithm

Input: D is the unsampled space, α is the normalization factor, N is the amount of sampled data, and T is the maximal iteration number.

Output: Final Precision Configuration P

```
X_0 ← sampling(D, α, N)
Use gem5 to obtain corresbonding clock cycles and power
L ← X_0
U ← D\L
for x = 1 ← T do
    Establish and train GP on L
    x_* ← arg max_{x∈U} EIPV(x | U)
    Update : L, U
end for
Construct Pareto-optimal set X from L
Select precision configuration P based on constraints
return P
```

V. Evaluation

In the experimental section, we searched for the optimal configurations of different scalar and vector programs for a given accuracy constraint and compared them with previous work. In order to get power consumption conveniently, we used the cycle-driven CPU simulator, Gem5, as a test platform for most of our experiments. Further, we also did real machine tests in x86 environment to verify the correctness of the method.

A. Scalar Program

We compared our search results with those of ADAPT [11]. Given our output results produced precisely in-place precision vectors, we mapped the mantissa back to half-precision, single-precision, and double-precision for comparison. To obtain the program's precision configuration, we initially utilized ADAPT's test programs with equivalent input files and error metrics. As ADAPT solely supports 32-64 bit mixing, we ensured a fair comparison by setting the lowest precision to single precision in the first set of experiments. Subsequently, we extended the search logic of ADAPT to support half-precision search.

Figure 3 presents the experimental results, with each group representing a distinct test case on the horizontal coordinates. The two left columns in each group represent the data obtained by running ADAPT, while the vertical coordinates represent the speedup ratio obtained by each program relative to the full-precision version. Unlike the sampling operation, we conducted five runs of each program and calculated the average to obtain the exact running time.

Our experiments revealed that our tool could provide better precision configurations in the vast majority of cases in scalar program tests. Our tool yielded an average speedup ratio of 32.7%, which is better than ADAPT's 30.5% for $32-64$ bit configurations. Our tool also obtained an average speedup ratio of 42.5% when the precision options expanded, which is more advantageous than the existing work's 36%.

This improvement resulted from two primary reasons: firstly, a decrease in the number of bits of variables did not necessarily equate to faster speed. In some mixed-precision versions of programs, the compiler implicitly added data type conversions, increasing overhead at the hardware level, diminishing or offsetting the performance benefits of lower precision. Type conversion operations could account for $10-20\%$ of the total time in a given program, as observed in [18]. Secondly, greedy search methods risked being restricted to local optima, which became more pronounced as the search space increased, i.e., when the number of optional precision configurations per variable increased.

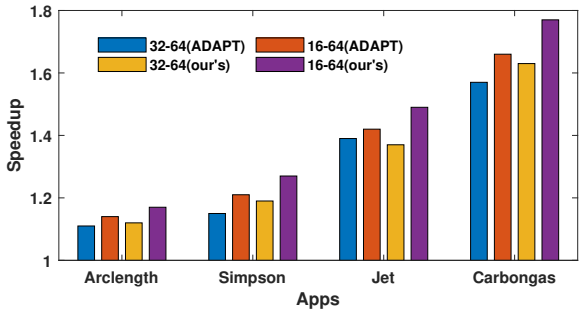

Fig. 3. Acceleration ratios obtained for each program with different precision configurations in different precision ranges

B. SIMD Program

In order to further test the performance improvement of the program by mixing precision in a vector environment, three representative programs were selected for testing, namely HPCCG, SVM and GMRES. HPCCG is an application in the mantevo benchmarking suite, and the core computational algorithm is the conjugate gradient method. SVM is the prediction stage of support vector machines, and is a representative algorithm in the field of machine learning. GMRES is the generalized minimum residual method for iteratively solving systems of linear equations and is the core algorithm for many scientific computing problems.

To maximize the impact of vectorization on the performance of the program, we mark the code segments that can be vectorized and vectorize them manually. We also keep the scalar version of the program for ADAPT analysis and accuracy model construction. To further exploit the performance and power benefits of vectorization, here we set floating-point types for each variable, i.e., half-precision, single-precision, and double-precision.

Experiments show that our tool can provide better precision configurations in a SIMD environment than ADAPT, obtaining an average performance improvement of 21%.

In the case of solving systems of linear equations (GMRES), our advantage is particularly clear. Specifically, we found that excessive reduction in the precision of the variables leads to a decrease in the convergence speed. Compared to the original precision, GMRES requires more iterations to reach convergence, and in extreme cases, the number of iterations can even increase by a factor of 2-3. Therefore, a proper search strategy should not only reduce the variable precision, but also keep the number of additional iterations within a suitable range. If the focus is only on the number of variable bits, the extra number of iterations tends to weaken the performance gain from mixing precision. Using performance rather than the number of variable bits as the optimization goal in our work can largely avoid this problem.

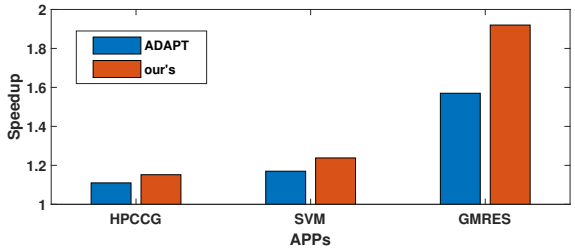

Fig. 4. The acceleration ratios obtained by each test program in the vector environment. Where blue represents the precision configuration obtained by ADAPT and orange represents the precision configuration obtained by our tool.

Regarding power consumption, our program achieves an average power reduction of 17% in a vector environment through mixed precision. However, similar to performance, low-precision operations may be affected by certain operations, such as type conversion, that may negate the advantages of mixed precision. Nevertheless, after vectorization, there is a reduction of $5-7\%$ in power consumption, primarily due to the significant reduction in floating-point operations and memory accesses (up to 40% in SVM) enabled by SIMD.

To validate our approach on real hardware, we limited the precision levels to 32 and 64 bits. For scalar programs, we achieved the same precision configuration as the existing tool for arclength and Simpson, and observed a 5% and 6.2% performance gain, respectively, due to better precision configurations in the other two programs. For vector programs, we obtained different precision configurations and an average additional performance gain of 21%, consistent with the simulator tests. Notably, we found that using the precision configuration obtained from the simulator as an initial point and fine-tuning it by hardware sampling could further optimize the search overhead. This insight suggests that future work can employ a two-step approach to first coarse-grain the search on the fast model and then fine-tune it on the accurate model.

C. Tuning Overhead

While our tool offers more accurate precision configurations than existing tools such as ADAPT [11] and Precimonious [3], it requires sampling, leading to a larger search overhead. The search time can range from tens of seconds to tens of minutes, with a significant portion of the time devoted to obtaining performance and power consumption through a clock-driven processor simulator. On average, our tool requires 10-20 iterations to achieve the final precision configuration, depending on the program and the initial training set distribution.

TABLE I
ERROR THRESHOLDS, OUTPUT ERRORS, AND ESTIMATION ERRORS FOR VARIOUS APPLICATIONS

Apps	Error Threshold	Output Error	Estimate Error
Arclength	1e-12	2.3e-13	2.5e-13
Simpson	1e-12	4.8e-14	4.3e-14
Jet	1e-13	2.2e-13	8.9e-14
Carbongas	1e-10	6.5e-11	3.7e-11
HPCCG	1e-9	2.3e-10	4.3e-10
SVM	1e-10	7.2e-12	3.1e-12
GERES	1e-9	5.9e-11	1.2e-10

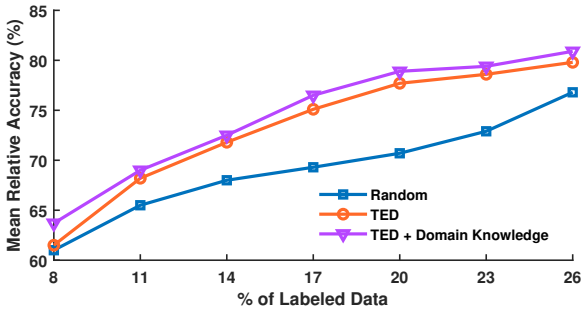

Fig. 5. The performance of MRA varies with changes in the number of samples under different sampling strategies.

To reduce the search time, we introduce domain knowledge and active learning ideas in the initial sampling process. This reduces the search time by an average of 31%. We show in Figure 5 that using the transductive experimental design algorithm individually can improve the prediction accuracy compared to random sampling. Furthermore, a better initial set with the introduction of domain knowledge can further reduce the number of samples needed, optimizing the search time.

However, there is still room for optimization of the search process, such as parallelizing fragments of the search process as done in [10], and sampling with faster simulators or building hardware models to predict program performance more quickly. These are left for future work.

VI. CONCLUSION

This paper presents an algorithm for solving mixed precision configurations of programs. The algorithm is modeled by sampling and uses multi-objective Bayesian optimization to search for the precision configuration with optimal performance and power consumption. Its novelty is that it uses information about the program's running on hardware to guide the search, implicitly including the impact of operations such as data type conversion and SIMD on program performance. Experiments show that it yields precision configurations that are better than those given by existing work in most cases. For programs with a higher degree of vectorization, the performance advantage we achieve will be even more apparent.

REFERENCES

[1] Ganesh Gopalakrishnan, Paul D. Hovland, Costin Iancu, Sriram Krishnamoorthy, Ignacio Laguna, Richard A. Lethin, Koushik Sen, Stephen F. Siegel, and Armando Solar-Lezama. Report of the hpc correctness summit, jan 25–26, 2017, washington, dc, 2017.

[2] Michael O Lam, Jeffrey K Hollingsworth, Bronis R de Supinski, and Matthew P LeGendre. Automatically adapting programs for mixed-precision floating-point computation. In *Proceedings of the 27th international ACM conference on International conference on supercomputing*, pages 369–378, 2013.

[3] Cindy Rubio-González, Cuong Nguyen, Hong Diep Nguyen, James Demmel, William Kahan, Koushik Sen, David H Bailey, Costin Iancu, and David Hough. Precimonious: Tuning assistant for floating-point precision. In *SC'13: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, pages 1–12. IEEE, 2013.

[4] Alexey Solovyev, Marek S Baranowski, Ian Briggs, Charles Jacobsen, Zvonimir Rakamarić, and Ganesh Gopalakrishnan. Rigorous estimation of floating-point round-off errors with symbolic taylor expansions. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 41(1):1–39, 2018.

[5] Eva Darulova, Einar Horn, and Saksham Sharma. Sound mixed-precision optimization with rewriting. In *2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems (ICCPS)*, pages 208–219. IEEE, 2018.

[6] Qiang Xu, Todd Mytkowicz, and Nam Sung Kim. Approximate computing: A survey. *IEEE Design Test*, 33(1):8–22, 2016.

[7] Gennaro Rodrigues, Fernanda Lima Kastensmidt, and Alberto Bosio. Survey on approximate computing and its intrinsic fault tolerance. *Electronics*, 9(4), 2020.

[8] Thierry Braconnier and Philippe Langlois. From rounding error estimation to automatic correction with automatic differentiation. In *Automatic differentiation of algorithms*, pages 351–357. Springer, 2002.

[9] Alexey Solovyev, Marek S Baranowski, Ian Briggs, Charles Jacobsen, Zvonimir Rakamarić, and Ganesh Gopalakrishnan. Rigorous estimation of floating-point round-off errors with symbolic taylor expansions. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 41(1):1–39, 2018.

[10] Nhut-Minh Ho, Elavarasi Manogaran, Weng-Fai Wong, and Asha Anoosheh. Efficient floating point precision tuning for approximate computing. In *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 63–68. IEEE, 2017.

[11] Harshitha Menon, Michael O Lam, Daniel Osei-Kuffuor, Markus Schordan, Scott Lloyd, Kathryn Mohror, and Jeffrey Hittinger. Adapt: Algorithmic differentiation applied to floating-point precision tuning. In *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 614–626. IEEE, 2018.

[12] Kai Yu, Jinbo Bi, and Volker Tresp. Active learning via transductive experimental design. In *Proceedings of the 23rd international conference on Machine learning*, pages 1081–1088, 2006.

[13] Yuzhe Ma, Ziyang Yu, and Bei Yu. Cad tool design space exploration via bayesian optimization. In *2019 ACM/IEEE 1st Workshop on Machine Learning for CAD (MLCAD)*, pages 1–6, 2019.

[14] Chen Bai, Qi Sun, Jianwang Zhai, Yuzhe Ma, Bei Yu, and Martin DF Wong. Boom-explorer: Risc-v boom microarchitecture design space exploration framework. In *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pages 1–9. IEEE, 2021.

[15] Samuel Daulton, Maximilian Balandat, and Eytan Bakshy. Differentiable expected hypervolume improvement for parallel multi-objective bayesian optimization. *Advances in Neural Information Processing Systems*, 33:9851–9864, 2020.

[16] Amar Shah and Zoubin Ghahramani. Pareto frontier learning with expensive correlated objectives. In *International conference on machine learning*, pages 1919–1927. PMLR, 2016.

[17] Michael Emmerich. *Single-and multi-objective evolutionary design optimization assisted by gaussian random field metamodels*. PhD thesis, Dortmund, Univ., Diss., 2005, 2005.

[18] Giuseppe Tagliavini, Stefan Mach, Davide Rossi, Andrea Marongiu, and Luca Benini. A transprecision floating-point platform for ultra-low power computing. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1051–1056. IEEE, 2018.