FLInt: Exploiting Floating Point Enabled Integer Arithmetic for Efficient Random Forest Inference

Christian Hakert (TU Dortmund University), Kuan-Hsun Chen (University of Twente) and Jian-Jia Chen (TU Dortmund University)

***Abstract*—In many machine learning applications, e.g., tree-based ensembles, floating point numbers are extensively utilized due to their expressiveness. Even if floating point hardware is present in general computing systems, using integer operations instead of floating point operations promises to reduce operation overheads and improve the performance. In this paper, we provide FLInt, a full precision floating point comparison for random forests, by only using integer and logic operations. The usage of FLInt basically boils down to a one-by-one replacement of conditions: For instance, a comparison statement in C: `if(pX[3]<=(float)10.074347)` becomes `if((*(((int*)(pX))+3))<=((int)(0x41213087)))`. Experimental evaluation on X86 and ARMv8 desktop and server class systems shows that the execution time can be reduced by up to $\approx 30\%$ with our novel approach.**

Introduction: Random forests are promising machine learning models, especially for applied scenarios under resource limitations [2]. While training of the learning models usually can be done on powerful hardware, real-time inference has to be highly resource efficient in order to exploit maximal performance on the execution platform [5]. If it is required to use floating point numbers for the classification, systems need to be equipped with a hardware floating point unit or consume more energy and time for the use of software floating points [8]. In this work, we provide a new alternative. The studied **problem** is to *compute correct floating point arithmetic in random forests without the need for hardware floating point support*. By only using standard integer and logic operations and proofing the correctness of the computation, we 1) enable floating-point-based random forests on devices without floating point hardware and 2) eliminate the overheads to use the floating point unit. In order to evaluate the effectiveness of FLInt, we conduct experimental evaluation on the implementation of if-else tree based random forest ensembles. The results show that we can reduce the execution time by up to $\approx 30\%$ with the use of FLInt on desktop and server systems.

Our novel contributions:

- FLInt: A two's complement and logic operation based comparison operator for floating point numbers, where we formally prove the correctness.
- An efficient implementation in random forests with if-else tree implementations, where we resolve special case handling offline during the implementation time.
- Experimental evaluation on X86 and ARMv8 server and desktop class systems to study the reduction of execution time when using FLInt instead of floating points.

Related Work: It is reported that some CPUs internally use the same hardware unit for floating point and integer comparisons with a few additions for the floating point computation [4]. Furthermore, there are explicit considerations about the binary floating point format regarding accuracy and efficient programming [3], [9]. Efficient execution of random forests and its basic unit, decision trees, has been widely studied in the literature. The relevant techniques can be informally divided into two families of approaches: *algorithmic refinements* and *architectural optimizations*. The former targets to improve the execution algorithm of decision trees itself, e.g., by applying different representations. Kim et al. propose parallelization in the form of vectorization for decision trees to favor the usage of Intel CPUs [10]. Based on vectorization, QuickScorer performs an interleaved traversal of the trees by using logical bitwise operations for gradient boosted trees [11]. Hummingbird is proposed to compile decision trees into a small set of tensor operations and batch tensors for each tree together for tree traversal [12]. On the other hand, architectural optimizations focus on the utilization of hardware resources. Asadi et al. first introduce the concept of native trees and if-else trees as a low-level implementation [1]. Buschjäger et al. [6] use empirical probabilities for single branches within a decision tree collected on the training data set to layout the memory representation of decision trees in order to benefit cache prefetching and protect from preemption. While Chen et al. utilize the GNU binary utilities to derive exact binary sizes to optimize the memory layout [7], Prasad et al. use the MLIR infrastructure to implement tiling transformations to reduce the cost of tree walk [13].

Providing Correct Floating Point Comparisons with Integer and Logic Arithmetic: The full proof of the correctness of the FLInt operator can be found in the full paper[1]. To sketch the correctness of the FLInt operator, the relation between the binary encoding of floating points and signed integer values is graphically illustrated in Figure 1. We denote $SI(B)$ as the signed integer interpretation of a bit vector B and $FP(B)$ as the floating point interpretation, respectively. It can be seen that focusing on the negative and positive number separately, the binary ordering is order preserving between the floating point and signed integer representation. This can be also proven by explicitly considering the floating point format definition. To construct the FLInt operator, the negative space of encoded floating point numbers is inverted, such that the ascending order of the floating point interpretation is preserved for the negative and positive number space.

Theorem 1. *Given two arbitrary bit vectors $X, Y \in \{0,1\}^k$ where the positiveness of $FP(X)$ (equivalently $SI(X)$) is*

[1] https://arxiv.org/pdf/2209.04181.pdf

979-8-3503-4859-0/DATE24/© 2024 EDAA

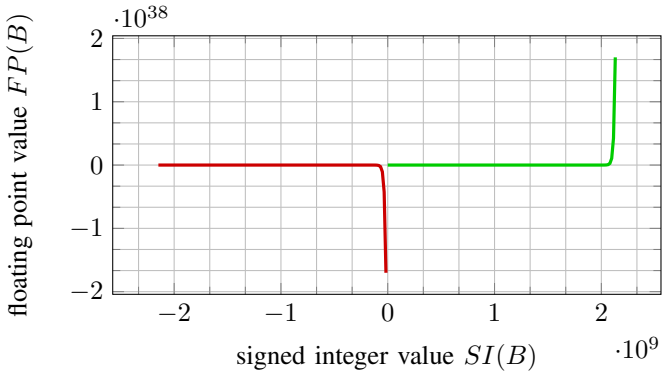

Fig. 1. Illustration of signed integer (x axis) and floating point (y axis) space for all combination of 32 bit vectors B

known a priori, the $\geq$ relation can be computed between the floating point interpretation of these bit vectors, using only two's complement signed integer arithmetic:

$$FP(X) \geq FP(Y) \Leftrightarrow \begin{cases} -1 \cdot SI(Y) \geq -1 \cdot SI(X) & \text{if } SI(X) < 0 \\ SI(X) \geq SI(Y) & \text{otherwise} \end{cases} \quad (1)$$

This operation can be applied during the implementation of a decision tree in a straight forward manner. X is chosen to be the split value, such that the negativeness can be checked at implementation time. The incoming value can be transformed to it's negative value by flipping the sign bit. According code is directly generated into the tree implementation.

Evaluation: In order to provide experimental evaluation, FLInt is implemented into C code generation of random forests. Ensembles with amounts of trees from 1 to 100 are trained. The maximal tree depth is limited to a value between 1 and 50 layers. The evaluation is conducted on X86 and ARMv8 server and desktop class systems. A baseline is provided by a standard implementation of the random forests as if else trees and normal floating point computations. FLInt is realized as a separate version, where floating point comparisons are translated to signed integer comparisons according to the proven operator. A state of the art cache aware grouping and swapping (CAGS) performance optimization technique [7] is used to assess the scale of the performance improvement. The integration of FLInt into CAGS evaluates the orthogonality of the FLInt operator.

We illustrate the average (geometric mean) normalized execution time across all data sets and ensemble sizes for a specific maximal depth of the single trees in Figure 2 for all considered systems. It can be observed that the FLInt implementation improves the execution time for almost all evaluated cases for the standard tree, as well as for the CAGS implementation.

For all systems, except the ARM server system, FLInt on its own achieves a similar or larger improvement as CAGS does. For smaller trees, the improvement is even consequently larger. Combining FLInt and cache-aware grouping and swapping can consequently provide additional performance improvement in most relevant cases.

Overall, it can be observed that the integration of FLInts into random forests can reduce the execution time in comparison to a naive implementation by up to $\approx 30\%$. Integrating FLInts into

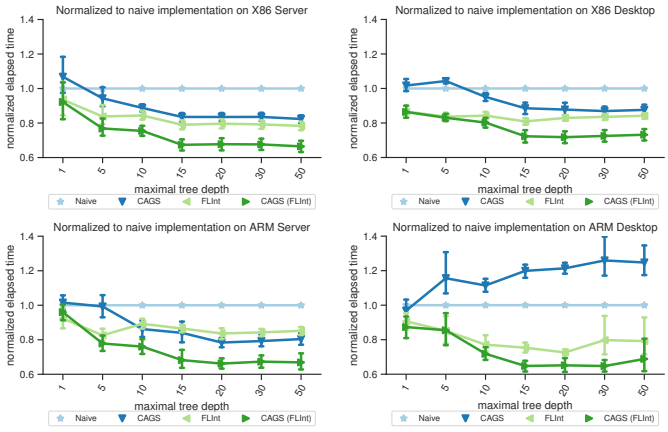

Fig. 2. Normalized execution time for increasing maximal tree depth

other existing optimization methods, even reduces the execution time by up to $\approx 35\%$. All these modifications do not impact the model output, nor the accuracy.

Acknowledgements: This paper has been supported by Deutsche Forschungsgemeinshaft (DFG), as part of the project OneMemory(405422836).

References

[1] Asadi, N., Lin, J., de Vries, A.P.: Runtime optimizations for tree-based machine learning models. IEEE Transactions on Knowledge and Data Engineering **26**(9), 2281–2292 (Sept 2014)

[2] Belgiu, M., Drgu, L.: Random forest in remote sensing: A review of applications and future directions. ISPRS Journal of Photogrammetry and Remote Sensing **114**, 24–31 (2016)

[3] Blinn, J.F.: Floating-point tricks. IEEE Computer Graphics and Applications **17**(4), 80–84 (1997)

[4] Bramley, J.: Condition Codes 4: Floating-point comparisons using VFP. https://community.arm.com/arm-community-blogs/b/architectures-and-processors-blog/posts/condition-codes-4-floating-point-comparisons-using-vfp

[5] Buschjäger, S., Morik, K.: Decision tree and random forest implementations for fast filtering of sensor data. IEEE Transactions on Circuits and Systems I: Regular Papers **PP**(99), 1–14 (2017)

[6] Buschjäger, S., Chen, K.H., Chen, J.J., Morik, K.: Realization of random forest for real-time evaluation through tree framing. In: 2018 IEEE International Conference on Data Mining (ICDM) (2018)

[7] Chen, K.H., Su, C., Hakert, C., Buschjäger, S., Lee, C.L., Lee, J.K., Morik, K., Chen, J.J.: Efficient realization of decision trees for real-time inference. Transactions on Embedded Computing Systems (2022)

[8] Cowlishaw, M.F.: Decimal floating-point: Algorism for computers. In: 16th IEEE Symposium on Computer Arithmetic. pp. 104–111 (2003)

[9] Demmel, J.W., Dhillon, I., Ren, H.: On the correctness of some bisection-like parallel eigenvalue algorithms in floating point arithmetic. Electronic Trans. Num. Anal **3**, 116–140 (1995)

[10] Kim, C., Chhugani, J., Satish, N., Sedlar, E., Nguyen, A., Kaldewey, T., Lee, V., Brandt, S., Dubey, P.: FAST: Fast architecture sensitive tree search on modern CPUs and GPUs. In: Proceedings of the International Conference on Management of data. ACM (2010)

[11] Lucchese, C., Perego, R., Nardini, F.M., Tonellotto, N., Orlando, S., Venturini, R.: Exploiting CPU SIMD extensions to speed-up document scoring with tree ensembles. In: Proceedings of the International Conference on Research and Development in Information Retrieval (2016)

[12] Nakandala, S., Saur, K., Yu, G.I., Karanasos, K., Curino, C., Weimer, M., Interlandi, M.: A tensor compiler for unified machine learning prediction serving. In: Proceedings of the 14th USENIX Conference on Operating Systems Design and Implementation. pp. 899–917 (2020)

[13] Prasad, A., Rajendra, S., Rajan, K., Govindarajan, R., Bondhugula, U.: Treebeard: An optimizing compiler for decision tree based ml inference. In: 2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO). pp. 494–511 (2022)