Compilers for Precision Tuning: TAFFO Problems (and Solutions)

Daniele Cattaneo[†], Michele Chiari[†],
Nicola Fossati[†], Gabriele Magnani[†],
Stefano Cherubin[*], Giovanni Agosta[†]

[†] *DEIB, Politecnico di Milano, piazza Leonardo Da Vinci 32, 20133 Milano, Italy* [1]
[*] *Codeplay Software Limited, Argyle House Level C, EH3 9DR Edinburgh, UK* [2]

ABSTRACT

TAFFO (Tuning Assistant for Floating point to Fixed point Optimization) is an open source project that addresses the problem of precision tuning. This is a trending approximate computing technique that allows to trade off precision for performance. We present some new applications and approaches where we successfully applied the optimizations performed by TAFFO. We detail its effectiveness in improving the quality of service of such applications, and we introduce the main research and engineering challenges related to its development.

KEYWORDS: Fixed Point; Compiler Transformation; Precision Tuning

Introduction

Error-tolerating applications are increasingly common. Proposals have been made at the hardware level to take advantage of inherent perceptual limitations, redundant data, or reduced precision input [C$^+$13], and to reduce system costs or improve power efficiency [TMR$^+$18, VCRR15]. At the same time, works on floating-point to fixed-point conversion tools [L$^+$10, CAL$^+$17] allow to trade-off the algorithm exactness for a more efficient implementation. Several tools and methods for approximate computing have been proposed, ranging from the tuning of computation precision to more aggressive methods such as loop perforation [CA20].

We aim at exploiting and extending the tools for precision tuning developed as part of the H2020 FET-HPC ANTAREX project [S$^+$18a, S$^+$19, S$^+$18b]. These tools, collected in the TAFFO framework [CCC$^+$19, CDBC$^+$19] are implemented as a set of plugins for the widely-used LLVM compiler framework [LA04]. TAFFO collects programmer hints – expressed as attributes – and it performs value range analysis [M$^+$09], data type and code conversion, and static estimation of the performance impact.

[1]E-mail: {daniele.cattaneo, michele.chiari}@polimi.it – {nicola.fossati, grabriele.magnani}@mail.polimi.it – agosta@acm.org
[2]E-mail: stefano.cherubin@codeplay.com

Our efforts currently focus in two different directions. On one hand, we aim at porting the benefits of precision tuning to new classes of applications. At the same time, we are working on improving the impact of the TAFFO code transformations on the target metric (energy and/or execution time).

AI in Embedded Systems: the Fall Detection use case

AI and the IoT are bridging the gap between the human and the digital worlds. Ad hoc technologies are being developed to support the more fragile part of the population. The main avenues to improve accessibility and durability of these solutions pass through lowering costs and power consumption. Precision tuning can help to achieve these goals, as it reduces the development cost of using cheaper processing units.

We successfully applied TAFFO to an activity classification system used in wearable devices in fall detection of elder people [F$^+$20]. [3] On the k-nearest neighbors classification use case, our fixed point version obtained with TAFFO can achieve over 500% of speedup in execution time and consume about 6 times less energy to carry out the classification.

A New Paradigm for the HPC: Dynamic Precision Tuning

Most efforts in precision tuning until now have been based on a static approach, which is therefore part of the system design. In contrast, dynamic precision tuning is a recurring task which is invoked multiple times while the application is running [CCCA20]. In other terms, it is a form of *Continuous Program Optimization* [KF03], which is particularly useful where runtime conditions are constantly varying and can be hardly predicted ahead of time.

We used TAFFO, in combination with the LIBVC Dynamic compilation library [CA18], to introduce a new methodology for partially automated dynamic precision tuning. Such methodology is based on the identification of input classes where the approximation behavior is the same, and for which the same set of data types can therefore be used. We demonstrate the effectiveness of our methodology on a set of benchmarks from the AXBENCH suite, achieving a speedup between 25% and 320% on Intel and AMD server processors, at a very limited cost in terms of accuracy – less than 3% error for each benchmark, a significant improvement with respect to the traditional *static* precision tuning.

Open Challenges

Researchers have proposed several approaches to perform precision tuning [CA20]. The strategies implemented in TAFFO allow it to be exploited in a wide range of application domains, as well as to guarantee extensibility and maintainability of its codebase. TAFFO can perform precision tuning at fine-grained level, while being source-language agnostic thanks to the LLVM compiler framework. However, there are research and engineering challenges that are still open in this field, preventing us from being short on work. We invite the reader to reach us if they wish to discuss them further.

[3]No elder people were harmed during the experiment.

Complexity of Data Type Selection

As the number of feasible assignments grows exponentially with the number of variables to tune and polynomially with the number of available data types, the selection of the best type for each variable is a complex problem. When dealing with fixed point data types, the bit partitioning must also be kept into account. Whilst the traditional *trial and error* search-based approach to test all possible available data types works well in the case of tuning between single and double precision floating point, a different approach is required when the search-space of candidate data types is larger.

Data Width and Memory Management

Precision tuning entails the replacement of some data types with different ones, which may or may not fit in the same data width. It follows that this process requires to consequently adjust all memory allocation operations. It also involves re-computing the data type sizes, which is non-trivial in the presence of complex and nested data structures, or arrays whose size is computed at run-time. Source-to-source compilers may work around this problem by forwarding the data type replacement in memory-size computation. From the compiler middle-end perspective, however, this fix becomes more complex, and not always feasible.

Limits of the Translation Unit

Real-world applications are modular and often rely on procedures and function calls. The scope of several tools in the state-of-the-art is limited to a single kernel function. Compilers may also analyse and transform other functions in the same translation unit. However, whichever function whose body is defined elsewhere is unknown and cannot be processed. This problems applies also to external libraries, including the standard mathematical library. In most cases, a short whitelist of special handling for the most common APIs may be sufficient to work around this problem. For example, in embedded systems, re-implementing such libraries to exploit fixed point types can provide significant performance advantages.

References

[C$^+$13] Vinay K. Chippa et al. Approximate computing: An integrated hardware approach. In *2013 Asilomar conference on signals, systems and computers*, pages 111–117. IEEE, 2013.

[CA18] Stefano Cherubin and Giovanni Agosta. libVersioningCompiler: An easy-to-use library for dynamic generation and invocation of multiple code versions. *SoftwareX*, 7:95 – 100, 2018.

[CA20] Stefano Cherubin and Giovanni Agosta. Tools for reduced precision computation: a survey. *ACM Computing Surveys*, 53(2), Apr 2020.

[CAL$^+$17] Stefano Cherubin, Giovanni Agosta, Imane Lasri, Erven Rohou, and Olivier Sentieys. Implications of Reduced-Precision Computations in HPC: Performance, Energy and Error. In *International Conference on Parallel Computing (ParCo)*, Sep 2017.

[CCC$^+$19] Stefano Cherubin, Daniele Cattaneo, Michele Chiari, Antonio Di Bello, and Giovanni Agosta. TAFFO: Tuning assistant for floating to fixed point optimization. *IEEE Embedded Systems Letters*, 2019.

[CCCA20] Stefano Cherubin, Daniele Cattaneo, Michele Chiari, and Giovanni Agosta. Dynamic precision autotuning with taffo. *ACM Trans. Archit. Code Optim.*, 17(2), May 2020.

[CDBC$^+$19] Daniele Cattaneo, Antonio Di Bello, Michele Chiari, Stefano Cherubin, and Giovanni Agosta. Fixed point exploitation via compiler analyses and transformations: Poster. In *Proceedings of the 16th ACM International Conference on Computing Frontiers*, CF '19, pages 292–294, Apr 2019.

[F$^+$20] Nicola Fossati et al. Automated precision tuning in activity classification systems: A case study. In *Proceedings of the 11th Workshop on Parallel Programming and Run-Time Management Techniques for Many-Core Architectures / 9th Workshop on Design Tools and Architectures for Multicore Embedded Computing Platforms*, PARMA-DITAM 2020, Jan 2020.

[KF03] Thomas Kistler and Michael Franz. Continuous program optimization: A case study. *ACM Trans. Program. Lang. Syst.*, 25(4):500–548, jul 2003.

[L$^+$10] Michael D Linderman et al. Towards program optimization through automated analysis of numerical precision. In *Proceedings of the 8th annual IEEE/ACM international symposium on Code generation and optimization*, pages 230–237, 2010.

[LA04] Chris Lattner and Vikram Adve. LLVM: A compilation framework for lifelong program analysis & transformation. In *Proc. Int'l Symp. on Code Generation and Optimization*, 2004.

[M$^+$09] Ramon E Moore et al. *Introduction to interval analysis*. Siam, 2009.

[S$^+$18a] Cristina Silvano et al. ANTAREX: A DSL-based approach to adaptively optimizing and enforcing extra-functional properties in high performance computing. In *21st Euromicro Conf. on Digital System Design (DSD)*, pages 600–607, 2018.

[S$^+$18b] Cristina Silvano et al. Autotuning and adaptivity in energy efficient HPC systems: The ANTAREX toolbox. In *Proceedings of the 15th ACM International Conference on Computing Frontiers*, CF '18, pages 270–275, 2018.

[S$^+$19] Cristina Silvano et al. The antarex domain specific language for high performance computing. *Microprocessors and Microsystems*, 68:58–73, 2019.

[TMR$^+$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, March 2018.

[VCRR15] Swagath Venkataramani, Srimat T Chakradhar, Kaushik Roy, and Anand Raghunathan. Approximate computing and the quest for computing efficiency. In *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2015.