Tessera: User-Defined Compiler Optimizations for High-Performance Scientific Computing

Jessica Cotturone
University of Illinois Urbana-Champaign
Urbana, Illinois, USA
jac18@illinois.edu

William S. Moses (advisor)
University of Illinois Urbana-Champaign
Urbana, Illinois, USA
wsmoses@illinois.edu

1 Introduction

High-performance computing has become a pivotal component of modern scientific research, by allowing scientists to model physical phenomena, test hypotheses through simulations, and make predictions at a large scale. Its success relies on the careful design of multiple stages in the computational workflow. A domain scientist writes application code that models physical phenomena; that code calls into libraries containing expert implementations of operations like matrix multiplication and PDE solvers; a compiler analyzes intermediate representations of the code and optimizes it for a given hardware target; and finally, the hardware executes the resulting binary. However, as the code is transformed from the highest to the lowest levels of abstraction, its semantic meaning is discarded at the boundaries of each of these stages. The compiler has no knowledge of domain-specific properties that it could use to further optimize the code. For instance, even though it is common knowledge to any linear algebra specialist that the inverse of the inverse of a matrix is the matrix itself, the compiler is unable to make that simplification.

Existing solutions to this problem include the design of domain-specific languages (DSLs), such as Halide for image processing [4] and TACO for sparse tensor computations [2]. Although they are highly effective in their targeted domains, DSLs require manual rewriting of application code, which can be time-consuming and error-prone, and they do not compose with each other. To address these issues, we present Tessera, a flexible framework (built on top of MLIR [3]) for specifying domain-specific optimizations and preserving them throughout the software stack so that a compiler can exploit them. Unlike DSLs, our approach aims to integrate domain-specific optimizations into a general-purpose compiler, with no major code rewrites required, and in a manner that allows the composition of optimizations from a variety of domains.

2 Approach

Tessera, our framework for integrating domain-specific optimizations into a general-purpose compiler, uses a flexible, non-intrusive mechanism that can be widely applied to libraries and applications from a variety of domains, allowing domain scientists to benefit from improved performance without needing deep compiler expertise. Domain experts define high-level properties and optimizations on functions via source-level annotations. These annotations can also be added to library functions by library developers and then imported into all codes that use the library, so that the optimizations are portable and easily distributable. After a user adds optimization rules denoting domain-specific properties in application or library code, our Clang plugin parses the annotations and lowers them into patterns that can be used by the compiler to automatically optimize the code.

For example, a developer of MPFR, a library for arbitrary-precision floating-point computation [1], could annotate the library function to indicate an arithmetic property, like that multiplying a number by -1 is equivalent to negating it – a trivial identity, but invisible to the compiler since it can only see opaque function calls to the MPFR library. Similarly, they could specify a function rewrite, indicating that the square root function in MPFR is more efficient than the pow function when raising a number to the power of 0.5. Without our framework, a general-purpose compiler cannot perform these optimizations, no matter how much program analysis its generic passes perform. Tessera is designed to close that gap by turning user-defined annotations of high-level properties into compiler rewrite patterns. This mechanism is not restricted to a single domain; it is intended to handle compositions of optimizations from multiple domains.

3 Results

We successfully applied Tessera to several benchmarks in the PETSc library [**?**] that run iterative solvers. In all of these benchmarks, by adding annotations indicating that the matrices were symmetric positive definite, we were able to automatically reroute the Generalized Minimal Residual (GMRES) solver to a Conjugate Gradient (CG) solver, which is cheaper per iteration. As a baseline, we used Clang with our custom plugin but with no Tessera annotations. Compared to this baseline, we were able to obtain up to a 10.63× speedup on a benchmark for 2D elasticity by adding just a few reusable annotations. We also support rules in MFEM, MPFR, and other libraries, and these optimizations can interoperate in applications that use multiple libraries.

4 Conclusions

We presented Tessera, a framework for user-defined optimizations that allows domain experts to benefit from high-performance code without needing compiler expertise or major rewrites to their code. Our approach is general and applicable across a variety of domains, and it is designed to bridge the gaps between the layers of the scientific software stack while preserving high-level semantic meaning.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
SC '26, Chicago, IL
© 2026 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-x-xxxx-xxxx-x/YYYY/MM
https://doi.org/XXXXXXX.XXXXXXX

References

[1] Laurent Fousse, Guillaume Hanrot, Vincent Lefèvre, Patrick Pélissier, and Paul Zimmermann. 2007. MPFR: A multiple-precision binary floating-point library with correct rounding. *ACM Trans. Math. Softw.* 33, 2 (June 2007), 13–es. doi:10.1145/1236463.1236468

[2] Fredrik Kjolstad, Shoaib Kamil, Stephen Chou, David Lugato, and Saman Amarasinghe. 2017. The tensor algebra compiler. *Proc. ACM Program. Lang.* 1, OOPSLA, Article 77 (Oct. 2017), 29 pages. doi:10.1145/3133901

[3] Chris Lattner, Mehdi Amini, Uday Bondhugula, Albert Cohen, Andy Davis, Jacques Pienaar, River Riddle, Tatiana Shpeisman, Nicolas Vasilache, and Oleksandr Zinenko. 2021. MLIR: scaling compiler infrastructure for domain specific computation. In *Proceedings of the 2021 IEEE/ACM International Symposium on Code Generation and Optimization* (Virtual Event, Republic of Korea) *(CGO '21)*. IEEE Press, 2–14. doi:10.1109/CGO51591.2021.9370308

[4] Jonathan Ragan-Kelley, Connelly Barnes, Andrew Adams, Sylvain Paris, Frédo Durand, and Saman Amarasinghe. 2013. Halide: a language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines. In *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation* (Seattle, Washington, USA) *(PLDI '13)*. Association for Computing Machinery, New York, NY, USA, 519–530. doi:10.1145/2491956.2462176