Refactoring for performance optimization and porting to new architectures has become particularly challenging in the accelerator and offload computing era, where CSE software must be adapted by offloading processing to highly-parallel, exotic computational devices. The transformations entail gory details, often affecting the organization of data structures, the layout of loop nests and various code pieces. Data structures must be optimized for vectorized operations, fully or partially duplicated and moved from host to accelerator device. Loops often have to have their iteration space rearranged (tiling, fusion, etc.) and shared between host and accelerator devices. Code needs to be moved around; extracted, synchronized or repackaged for testing; specialized for optimization; generalized for reuse and cloned for cross-compilation.

Specifically, it is desirable to streamline the process of identifying and transforming computational kernels, as uninterruptedly to the code base as possible. This, we argue, requires new compilation abstractions that can be defined and directed by the CSE developer to complement the compilation toolchain.

Many tools, such as polyhedral compiler technology, stencil optimization toolkits, etc., expect the user to pre-process, isolate, or otherwise “normalize” the source code before feeding it to them. While these tools are backed by formal theory, and thus validating, the whole task can be intense and even impossible without removing offending code. Our collective experience supporting computational scientists with in-house and vendor-requested tools shows that the developer wants to transform sources with the least possible effort in order to explore an optimization path that they have worked out on paper, because they simply want to try out an idea. Productivity is not confined to just being able to describe and have a system that implements a custom refactoring.
task. When a user identifies a good refactorization, or generally a good software engineering solution, it becomes immediately highly desirable to answer the following question: which other parts of the code base can benefit from it also? Pattern-, feature- and clone-driven approaches, for instance, attempt to locate code that shares some similarity (or determine so dynamically), under the assumption that the likelihood of a refactoring being reusable is subject to the similarity metric.

A recent development in addressing these issues is based on a rather familiar to the CSE community mechanism: compiler directives. Similarly to directive APIs for parallel computing (OpenMP/OpenACC), directives are being defined for allowing the CSE developer (1) to extract characteristics from one code base or notify the compiler of which computational kernel a fragment of the CSE application resembles, (2) to construct and apply custom transformations by tagging the source code parts of interest, and (3) to combine or chain the two to facilitate custom analysis and transformation workflows ([4, 3, 2, 1, 5]).

References


