accepted in WOLFHPC 2011 : First International Workshop on Domain-Specific Languages and High-Level Frameworks for High Performance Computing

How a Domain-Specific Language Enables the Automation of Optimized Code for Dense Linear Algebra

Bryan Marker * Don Batory * Jack Poulson †
Robert van de Geijn *†

Ideally, a *domain-specific language (DSL)* allows one to code at the same level of abstraction as one reasons about a domain problem. For *dense linear algebra (DLA)*, we demonstrate how an appropriately chosen DSL can automate that reasoning. Elemental is a modern DLA C++ API for distributed-memory architectures that is a successor of ScaLAPACK. It embodies a DSL for the DLA domain and is representative of the FLAME API, which can be viewed as a more general DSL for DLA. We use Elemental to demonstrate the power of a well-structured DSL.

When an expert approaches a DLA algorithm to implement in code, she (implicitly or explicitly) chooses an initial sequential algorithm then, step-by-step, parallelizes and optimizes the algorithm and corresponding code to reach a final, optimized version. This process is very systematic and is repeated for most algorithms in the domain. In fact the process is now sufficiently well-understood that it can be (and has been) automated.

With Elemental, we demonstrate automated reasoning by embracing ideas from *model-driven engineering (MDE)*. With MDE we encode knowledge about the operations and algorithms in the DSL and the target architecture. With that knowledge we automate the transformation from algorithm to highly-optimized code.

We report on a prototype that takes a high-level DLA algorithm, applies transformations to it, and outputs optimized code for that algorithm in the Elemental DSL. We show that modularity and abstraction afforded by Elemental enabled us to encode knowledge about the language's constructs (e.g. computation operations). We also encode knowledge about the target architecture and the common parallelization methods for it. Together, knowledge about the architecture and the DSL operations enable our system to generate many (sometimes thousands) implementations for an input algorithm. The system then uses cost predictions of the operations to choose the most efficient implementations. Early results for a handful of case studies are output codes that are the same as those hand-produced by an expert. Sometimes, the resulting code is better because the human expert is limited by time, effort, and complexity.

We present the DSL used for this project, the prototype system we developed, and how Elemental's layering enabled our success. We also explain that our work is an example of an approach that extends to the more general FLAME DSL for DLA.

Our work also illustrates next-generation libraries: they should not be developed as instantiations in code. Rather, they should exist as an encoding of algorithms, knowledge about algorithms, and knowledge about target architectures. Optimizing transformations are then applied at a higher-level of abstraction than compilers, so more information about the algorithm is used. Instantiations are a final step that produces executable code. Other examples of this approach can be found in the Spiral and FENICS projects.

*Department of Computer Science, The University of Texas at Austin
†Institute for Computational Engineering and Sciences, The University of Texas at Austin