OptiTrust: an Interactive Optimization Framework

Thomas Köhler
Arthur Charguéraud
Begatim Bytyqi
Damien Rouhling
Inria
France

Yann Barsamian
École Européenne de Bruxelles
Belgium

Abstract
We present OptiTrust, an interactive framework for optimizing general-purpose C code via series of programmer-guided, source-to-source transformations. Optimization steps are described in transformation scripts, expressed as OCaml programs. At every step, the programmer may interactively visualize the effect of the transformation as the difference between two pieces of human-readable C code. OptiTrust has been previously employed to optimize numerical simulation code. In this work, we showcase how to use OptiTrust to optimize matrix multiplication. We compare against TVM, which also relies on programmer guidance, but which restricts the input language and lacks easily readable feedback.

1 Motivation
Achieving high-performance on modern, parallel and heterogeneous hardware is a challenging task that is critical in many domains, e.g., numerical simulation, image processing and machine learning. It remains common industrial practice to optimize code by hand in languages such as C/C++ [24, 31] or Fortran [32].

Manual code rewriting is highly time consuming, and must be repeated numerous times to reach high performance. Several optimization paths must be empirically explored because code performance is hard to assess without benchmarking. After this process, hand optimized code is generally hard to read, maintain, adapt to other hardware, and—worst of all—may contain bugs introduced in the process.

In order to reuse optimization efforts, optimized libraries such as BLAS or MKL for linear algebra were created. However, their use remains suboptimal as state-of-the-art general-purpose compilers (e.g. Clang, GCC, ICX) fail to apply relevant optimizations across library calls. This limitation has motivated the development of specialized optimizing compilers, able to apply global optimizations on high-level algorithms written in Domain-Specific Languages (DSLs).

Examples of specialized compilers include Halide [26] and TVM [14], which have proved successful in industry for image processing and machine learning, respectively. With slightly larger application domain are compilers for array programming languages [9, 17, 20, 29], exploiting the fact that multi-dimensional arrays are a key abstraction for many performance-demanding applications. Although the success of specialized optimizing compilers is undeniable, they cannot optimize programs that fall outside of their domain. Moreover, extending these specialized compilers has proved to be challenging [5].

OptiTrust is an optimization framework that aims to support general-purpose code, avoiding domain restriction. OptiTrust operates at the level of C code, by supporting source-to-source transformations. The programmer interactively develops a transformation script. At every transformation step, the programmer may visualize the corresponding diff between pieces of human-readable C code.

Prior work [12] has demonstrated the ability of OptiTrust to reproduce a state-of-the-art optimized implementation of a numerical Particle-In-Cell simulation. In the present work, we demonstrate the ability of OptiTrust to reproduce a reference implementation of matrix multiplication generated by the specialized compiler TVM.

2 The 5 Key Components of OptiTrust

1. An internal, simplified abstract syntax tree (AST) from which readable C code can be recovered throughout transformations. A similar idea of a simplified AST has been employed in the Cetus project [16].

Manual code rewriting is highly time consuming, and must be repeated numerous times to reach high performance. Several optimization paths must be empirically explored because code performance is hard to assess without benchmarking. After this process, hand optimized code is generally hard to read, maintain, adapt to other hardware, and—worst of all—may contain bugs introduced in the process.

In order to reuse optimization efforts, optimized libraries such as BLAS or MKL for linear algebra were created. However, their use remains suboptimal as state-of-the-art general-purpose compilers (e.g. Clang, GCC, ICX) fail to apply relevant optimizations across library calls. This limitation has motivated the development of specialized optimizing compilers, able to apply global optimizations on high-level algorithms written in Domain-Specific Languages (DSLs).

Examples of specialized compilers include Halide [26] and TVM [14], which have proved successful in industry for image processing and machine learning, respectively. With slightly larger application domain are compilers for array programming languages [9, 17, 20, 29], exploiting the fact that multi-dimensional arrays are a key abstraction for many performance-demanding applications. Although the success of specialized optimizing compilers is undeniable, they cannot optimize programs that fall outside of their domain. Moreover, extending these specialized compilers has proved to be challenging [5].

OptiTrust is an optimization framework that aims to support general-purpose code, avoiding domain restriction. OptiTrust operates at the level of C code, by supporting source-to-source transformations. The programmer interactively develops a transformation script. At every transformation step, the programmer may visualize the corresponding diff between pieces of human-readable C code.

Prior work [12] has demonstrated the ability of OptiTrust to reproduce a state-of-the-art optimized implementation of a numerical Particle-In-Cell simulation. In the present work, we demonstrate the ability of OptiTrust to reproduce a reference implementation of matrix multiplication generated by the specialized compiler TVM.

2 The 5 Key Components of OptiTrust

1. An internal, simplified abstract syntax tree (AST) from which readable C code can be recovered throughout transformations. A similar idea of a simplified AST has been employed in the Cetus project [16].

2. A system for targeting program points, somewhat similar to XPath [15] for XML, but specialized for an AST. This system allows describing, in a concise and robust manner, one or several program points to transform.

3. A library of general-purpose transformations, including: core data layout transformations [30], instruction-level transformations [1], control flow transformations [33].

4. A scripting language, embedded in OCaml, for describing transformations. Transformation scripts are a classic technique for programmer-guided optimization frameworks [4, 6, 7, 13, 25, 27, 34–36]. Transformation scripts provide fine-grained control, and they allow chaining large numbers of transformation steps. Other approaches include use of pragmas [10, 22, 23], schedules [3, 14, 18, 26], or rewriting strategies [2, 8, 19, 21].

5. An interactive interface allowing to visualize code diffs associated with the transformation at a given line of the script, via a key shortcut in the code editor.

3 Optimizing Matrix-Multiply in OptiTrust
We investigate how OptiTrust compares with the specialized compiler TVM for producing an optimized implementation of matrix multiplication—a standard benchmark.
void mm(float* C, float* A, float* B, int m, int n, int p) {
for (int i = 0; i < m; i++) {
for (int j = 0; j < n; j++) {
float sum = 0.0f;
for (int k = 0; k < p; k++) sum += A[i][k] * B[k][j];
C[i][j] = sum; }
}

Listing 1. Unoptimized matrix multiplication. The function mm multiplies the matrices A and B and stores the result in C. The function mm1024 specializes input sizes to 1024.

We start from the C code presented in Listing 1: a naive, unoptimized implementation of matrix multiplication. The aim is to produce similar code as the reference TVM schedule that was handwritten by an expert targeting Intel CPUs.1

We write an OptiTrust script in OCaml, as shown in Listing 2. It consists of 9 transformation steps, each one calling a function from the OptiTrust library, always providing a target describing where to apply the transformation as last argument. OptiTrust scripts are developed interactively: with the cursor on a given line of the script, we can press (e.g.) “F6” in the VSCode editor to visualize the diff associated with the transformation on that line. All intermediate versions of the code consist of human-readable, executable C code.

The final code is shown in Listing 3. It mirrors the structural optimizations described in the TVM reference schedule, including: improved data locality, parallelism, and specialization to 1024 sizes. A major difference is that we produce C code with OpenMP pragmas, whereas TVM targets directly LLVM IR. To check that the code produced using OptiTrust performs similarly to that produced by TVM, we benchmarked both codes on a 4-core Intel i7-8665U CPU with AVX2 support. Both codes have 90th percentile runtime of 9.4ms over 200 benchmark runs, corresponding to a speedup of 2. It consists of 9 transformation steps, each one calling a function from the OptiTrust library, always providing a target describing where to apply the transformation as last argument. OptiTrust scripts are developed interactively: with the cursor on a given line of the script, we can press (e.g.) “F6” in the VSCode editor to visualize the diff associated with the transformation on that line. All intermediate versions of the code consist of human-readable, executable C code.

In summary, our case study shows that the OptiTrust general-purpose optimization framework can be used to interactively develop a code competitive with that produced by a state-of-the-art specialized compiler. Unlike specialized compilers, OptiTrust takes as input C code, and produces human-readable C code at every step.

4 Contents of the Talk
During the ARRAY workshop, we will present the OptiTrust matrix multiplication case study, and compare the script to the reference TVM schedule. We will give an overview of the transformations currently supported by OptiTrust, and describe ongoing work on leveraging Separation Logic annotations [11, 28] to validate transformation correctness beyond the limit of static analyses commonly used in general-purpose compilers.

We hope to engage with the ARRAY community by presenting an approach that tackles the limitations of specialized languages and compilers. Suggestions of challenging optimizations, representative benchmarks, and compilation techniques to investigate will be especially welcomed.
References


[10] Rohit Chandra, Leo Dagum, David Kohr, Ramesh Menon, Dorr Maydan, and Jeff McDonald. 2001. Parallel programming in OpenMP.


