Generating High-Performance General Size Linear Transform Libraries Using Spiral

Yevgen Voronenko Franz Franchetti Frédéric de Mesmay Markus Püschel*
Electrical and Computer Engineering
Carnegie Mellon University

May 16, 2008

Introduction

Developing numerical libraries that achieve highest performance on modern computer architectures became an extremely difficult task due to the increasingly complicated microarchitectures, deep cache hierarchies, and different forms of on-chip parallelism, such as multiple processor cores and SIMD short vector instruction sets.

The difficulty of library development led to interest in automated tools that simplify the development of high-performance libraries, without sacrificing performance. Program generator Spiral [2] is an example of such tool for the domain of linear transforms, such as the discrete Fourier transform (DFT), FIR filters, and others. Spiral automatically generates the optimized and platform-adapted implementation given only the transforms specification (e.g. $\mathbf{DFT}_{1024}$) and a high-level description of the recursive divide-and-conquer algorithm in the domain-specific language called SPL (Signal Processing Language). Spiral performs optimizations such as vectorization and parallelization using rewriting on the high-level of abstraction provided by SPL, and also lower-level representations.

To exploit the potential offered by the development automation tools, Intel Integrated Performance Primitives (IPP) library, which provides a wide number of optimized linear transform functions, starting with version 6.0, will include a special domain for the functions automatically generated by Spiral.

To date Spiral was restricted to generating code for transforms of fixed size, known at generation time. In this paper we overview our latest research results [3] that enable generating full general size libraries, for which the transform size is only known at runtime.

Library Generation

The goal of library generation is to produce a highly optimized transform implementation starting given only a transform and high-level specifications of divide-and-conquer algorithms (called *breakdown rules*) that the library should use.

*This work was supported by NSF through awards 0325687, 0702386, by DARPA through the DOI grant NBCH1050009 and the ARO grant W911NF0710416, and by an Intel grant.

For example, a typical input to the library generator is

$$\begin{aligned}\text{Transform: } & \mathbf{DFT}_n,\\ \text{Algorithms: } & \mathbf{DFT}_{km} \rightarrow (\mathbf{DFT}_k \otimes I_m)\,\mathrm{diag}(\Omega_{k,m})\\ & \qquad (I_k \otimes \mathbf{DFT}_m)L^{km}_k,\\ & \mathbf{DFT}_2 \rightarrow \left[\begin{smallmatrix}1 & 1\\ 1 & -1\end{smallmatrix}\right],\\ \text{Vectorization: } & \text{2-way SSE}\\ \text{Multithreading: } & \text{yes}\end{aligned}$$

Above we showed the well-known divide-and-conquer Cooley-Tukey fast Fourier transform (FFT) algorithm represented in SPL, which is defined in [2], however, the precise meaning of the SPL symbols and operators is not relevant here.

The output is a generated library that is

- for general input size;
- vectorized using the available SIMD vector instruction set;
- multithreaded with a fixed or variable number of threads;
- performance competitive with the best existing hand-written libraries.

Generating code for a fixed size transform is a fundamentally different problem from library generation as explained above. Fixed size code generation in Spiral works by decomposing the original transform at generation time into smaller transforms, until the base cases are reached. The full recursion tree, called *ruletree*, is known at generation time, and is first translated into SPL, then intermediate code representation (or a DAG), and finally into a program that computes the transform Generating a general size library requires compiling each breakdown rules into a recursive function, so that it can be applied dynamically at runtime. Thus a ruletree is never actually constructed at generation time.

The library generation process consists of the several steps explained next.

Compute the set of needed recursive functions. In many cases, including the Cooley-Tukey FFT, the larger transforms are decomposed into smaller transforms of the same type, and it seems that only a single recursive function is sufficient. However, in order to achieve the best possible performance different steps inside the algorithm must be merged [1], which

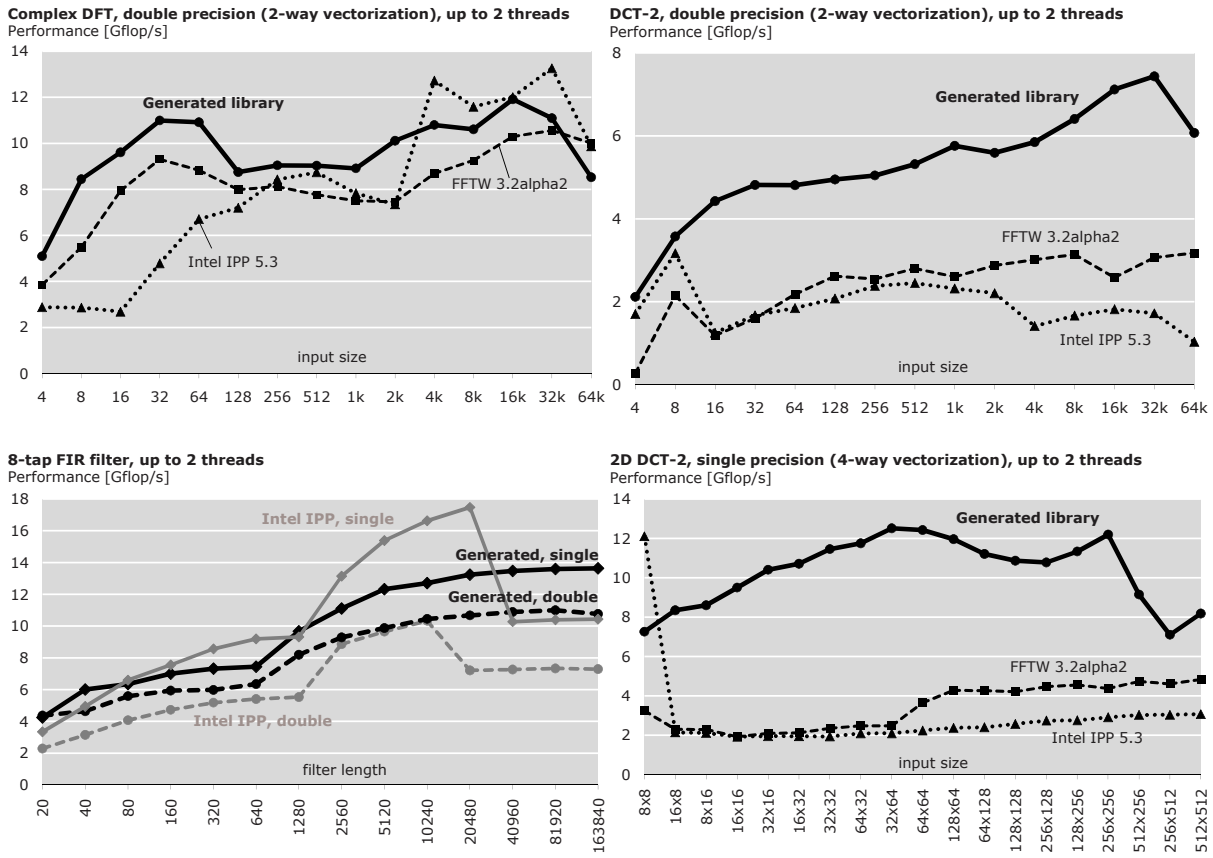

Figure 1: Performance of automatically generated libraries compared to hand-written libraries (FFTW uses generated code for small fixed size transforms). Double precision, using SSE2 and up to 2 threads. Platform: dual-core 3 GHz Intel Xeon 5160 processor with 4 MB of L2 cache running Linux. Generated libraries are in C++ and are compiled with Intel C/C++ Compiler 10.1.

leads to additional function with different interface. Implementation of these functions might require additional functions, and so forth.

Perform vectorization and parallelization. This step is done using rewriting rules applied at the SPL and lower level representations.

Hot/cold parameter partitioning. It is common for linear transform libraries to perform a number of constant precomputations and other initialization tasks, such as memory allocation, as soon as the transform size, and other parameters are known. The goal of this step is to determine which parameters are *cold*, i.e., must be provided at the initialization stage, and which parameters can be provided later.

Final code generation. The last step is to implement each of the previously derived functions in the target language. Each function will need at least one recursive general size implementation (that calls other functions), at least one base case to terminate the recursion, and the initialization code. We currently target C++, but in [3] also a Java backend is reported.

Performance

The performance of four example libraries, generated using Spiral, are shown in Fig. 1 and compared to FFTW and the Intel IPP. We observe that together with complete automation, generated libraries often achieve the highest performance. The performance of generated libraries is comparable to Intel IPP and FFTW, for the popular functionality, such as the DFT and the FIR filters. However, the less popular transforms such as the DCTs, are not as well optimized as the DFT in Intel IPP and FFTW, and consequently the generated libraries, which enjoy completely automated development and optimization, are faster.

References

[1] F. Franchetti, Y. Voronenko, and M. Püschel. Loop merging for signal transforms. In *Proc. PLDI*, pages 315–326, 2005.

[2] M. Püschel, J. M. F. Moura, J. Johnson, D. Padua, M. Veloso, B. W. Singer, J. Xiong, F. Franchetti, A. Gačić, Y. Voronenko, K. Chen, R. W. Johnson, and N. Rizzolo. SPIRAL: Code generation for DSP transforms. *Proceedings of the IEEE,* 93(2):232–275, 2005.

[3] Yevgen Voronenko. *Library Generation for Linear Transforms*. PhD thesis, Department of Electrical and Computer Eng., Carnegie Mellon University, 2008.