Program Generation with Spiral: Beyond Transforms [*]

Franz Franchetti, Daniel Mcfarlin, Frédéric de Mesmay, Hao Shen, Tomasz Wlodarczyk,
Srinivas Chellappa, Marek R. Telgarsky, Peter A. Milder, Yevgen Voronenko, Qian Yu,
James C. Hoe, José M. F. Moura, Markus Püschel
{franzf, jhoe, moura, pueschel}@ece.cmu.edu
Electrical and Computer Engineering, Carnegie Mellon University

Introduction

In this paper we extend the program generation system Spiral [5, 6] beyond its problem domain. Spiral was originally developed to automate the optimization of software libraries for linear transforms like the discrete Fourier transform (DFT), filters, wavelets, and others. In previous work we enabled Spiral to generate high-performance libraries for state-of-the-art and emerging platforms, including multicore CPUs with SIMD vector instruction sets, graphics processors (GPUs), field-programmable gate arrays (FPGAs), or a CPU with FPGA acceleration [2].

Beyond transforms. In this paper we take Spiral a first step beyond the domain of linear transforms. We extend Spiral to generate parallel and/or vectorized libraries for 1) Viterbi decoding, 2) the EBCOT encoder used in JPEG2000, 3) an SAR imaging algorithm, and 4) matrix-matrix-multiplication (MMM) for small matrices. This is work in progress: our automatically generated Viterbi decoder libraries and small-size SGEMM MMM libraries are competitive with or outperform the best available hand-tuned libraries for the same functionality. For the EBCOT encoder and SAR imaging, Spiral automatically generates fast implementations but these are not yet competitive with the best available software.

Spiral. Spiral automates the generation of high-performance software libraries for the domain of linear transforms. It generates software that takes advantage of different forms of parallelism, while at the same time matching the performance of hand-written code. Spiral is based on the following key ideas: 1) A domain-specific, declarative, mathematical language to describe algorithms; and 2) the use of rewriting to parallelize and optimize algorithms at a high level of abstraction.

Related work. Spiral belongs to the field of automatic performance tuning and program generation, which includes the projects ATLAS, PHiPAC, FFTW, OSKI, TCE, and others [1, 3, 4].

Extending Spiral Beyond Transforms

Spiral uses the declarative mathematical language SPL (signal processing language) to describe the structure of signal transform algorithms. SPL is based on the Kronecker product formalism used in multi-linear algebra [5].

name	definition
basic operator	
projection	$\pi_{\mathbf{x}} : \mathbb{C}^m \times \mathbb{C}^n \to \mathbb{C}^m;\ (\mathbf{x}, \mathbf{y}) \mapsto \mathbf{x}$
linear transform	$\mathrm{M} : \mathbb{C}^n \to \mathbb{C}^m; \mathbf{x} \mapsto M\mathbf{x}$
vector sum	$\Sigma_n : \mathbb{C}^n \to \mathbb{C};\ \mathbf{x} \mapsto \sum_{i=0}^{n-1} x_i$
vector minimum	$\min_n : \mathbb{C}^n \to \mathbb{C};\ \mathbf{x} \mapsto \min(x_0, \dots, x_{n-1})$
constant vector	$\mathrm{C}_{\mathbf{c}} : \varnothing \to \mathbb{C}^n;\ () \mapsto \mathbf{c}$
operations	
addition	$(\mathrm{M} + \mathrm{N})(\mathbf{x}, \mathbf{y}) = \mathrm{M}(\mathbf{x}, \mathbf{y}) + \mathrm{N}(\mathbf{x}, \mathbf{y})$
multiplication	$(\mathrm{M} \cdot \mathrm{N})(\mathbf{x}, \mathbf{y}) = \mathrm{M}(\mathbf{x}, \mathbf{y}) \cdot \mathrm{N}(\mathbf{x}, \mathbf{y})$
cartesian product	$(\mathrm{M} \times \mathrm{N})(\mathbf{x}, \mathbf{y}, \mathbf{u}, \mathbf{v}) = \mathrm{M}(\mathbf{x}, \mathbf{y}) \times \mathrm{N}(\mathbf{u}, \mathbf{v})$
composition	$(\mathrm{M} \circ \mathrm{N})(\mathbf{x}, \mathbf{y}) = \mathrm{M}(\mathrm{N}(\mathbf{x}, \mathbf{y}))$
tensor product	$\mathrm{I} \otimes \mathrm{M},\ \mathrm{M} \otimes \mathrm{I}$

Table 1: A subset of operators and operations defined by OL.

The key idea in going beyond transforms is to extend SPL so that a larger class of algorithms can be expressed, while Spiral's rewriting system (which enables parallelization and vectorization) still remains applicable. We call this extension of SPL *operator language* (OL).

Operator language (OL). OL is based on two extensions to SPL: 1) matrices are viewed as linear operators, and in addition general (potentially nonlinear) operators are allowed. 2) Matrices are viewed as operators with one input and one output vector. In OL we allow operators with multiple input and multiple output vectors. All SPL expressions can be interpreted as OL expression; for instance, the Cooley-Tukey FFT rule [5] is expressed in OL by

$$\mathrm{DFT}_{mn} \to (\mathrm{DFT}_m \otimes \mathrm{I}_n) \circ D_{m,n} \circ (\mathrm{I}_m \otimes \mathrm{DFT}_n) \circ \mathrm{L}_m^{mn}\,.$$

Here, all matrices become linear operators, and the matrix multiplication is replaced by operator composition.

Table 1 summarizes some new OL operators and operations that we use to describe algorithms outside the transform domain. Most importantly, we define a tensor product which generalizes to the non-linear case where the usual tensor product definition is not available. The definition is compatible with our parallelizing and vectorizing rewriting system.

Non-transform algorithms in OL. In this paper we express four non-transform applications in OL and generate shared memory parallel and SIMD vector code for them: 1) Viterbi decoding, 2) the EBCOT encoder used in JPEG2000, 3) a SAR imaging algorithm, and 4) matrix-matrix-multiplication (MMM) for small matrices. As example, Table 2 shows the the rewriting of OL formulas for the MMM and the Viterbi decoder, parallelizing or vectorizing

[*]This work was supported by NSF through awards 0325687, 0702386, and by DARPA through the Department of Interior grant NBCH1050009.

$$\underbrace{\mathrm{MMM}_{M,N,K}}_{\mathrm{smp}(p,\mu)} \rightarrow \left((\mathrm{L}_M^{Mp} \otimes \mathrm{I}_{N/(p\mu)})\bar{\otimes}\, \mathrm{I}_\mu\right) \circ \left(\mathrm{I}_{1\times p \to p} \otimes_{\|} \mathrm{MMM}_{M,N/p,K}\right) \circ \left((\mathrm{I}_{KM/\mu} \bar{\otimes}\, \mathrm{I}_\mu) \times ((\mathrm{L}_p^{Kp} \otimes \mathrm{I}_{N/(p\mu)})\bar{\otimes}\, \mathrm{I}_\mu)\right)$$

$$\underbrace{\mathrm{Vit}_{m,n,N}^{e,f,x}}_{\mathrm{vec}(\nu)} \rightarrow \prod_{i=0}^{n-1} \Big(\mathrm{I}_{2^{m-1}/\nu \times 2^{m-1}\times 1} \otimes_j \big((\underbrace{\mathrm{L}_\nu^{2\nu}}_{\mathrm{reg}(\nu)} \times \mathrm{I}_{2m\times nN}) \circ (\mathrm{V}_{i,4j+k}^{e,f} \bar{\otimes}_k \mathrm{I}_{\nu\times 1\times 1})\big)\Big) \circ \left((\mathrm{L}_{2^{m-1}/\nu}^{2^m} \bar{\otimes}\, \mathrm{I}_\nu) \times \mathrm{I}_{2^m n\times nN}\right)$$

Table 2: OL expressions for the shared memory parallelization of a matrix-matrix-multiplication and SIMD vectorization of a Viterbi decoder.

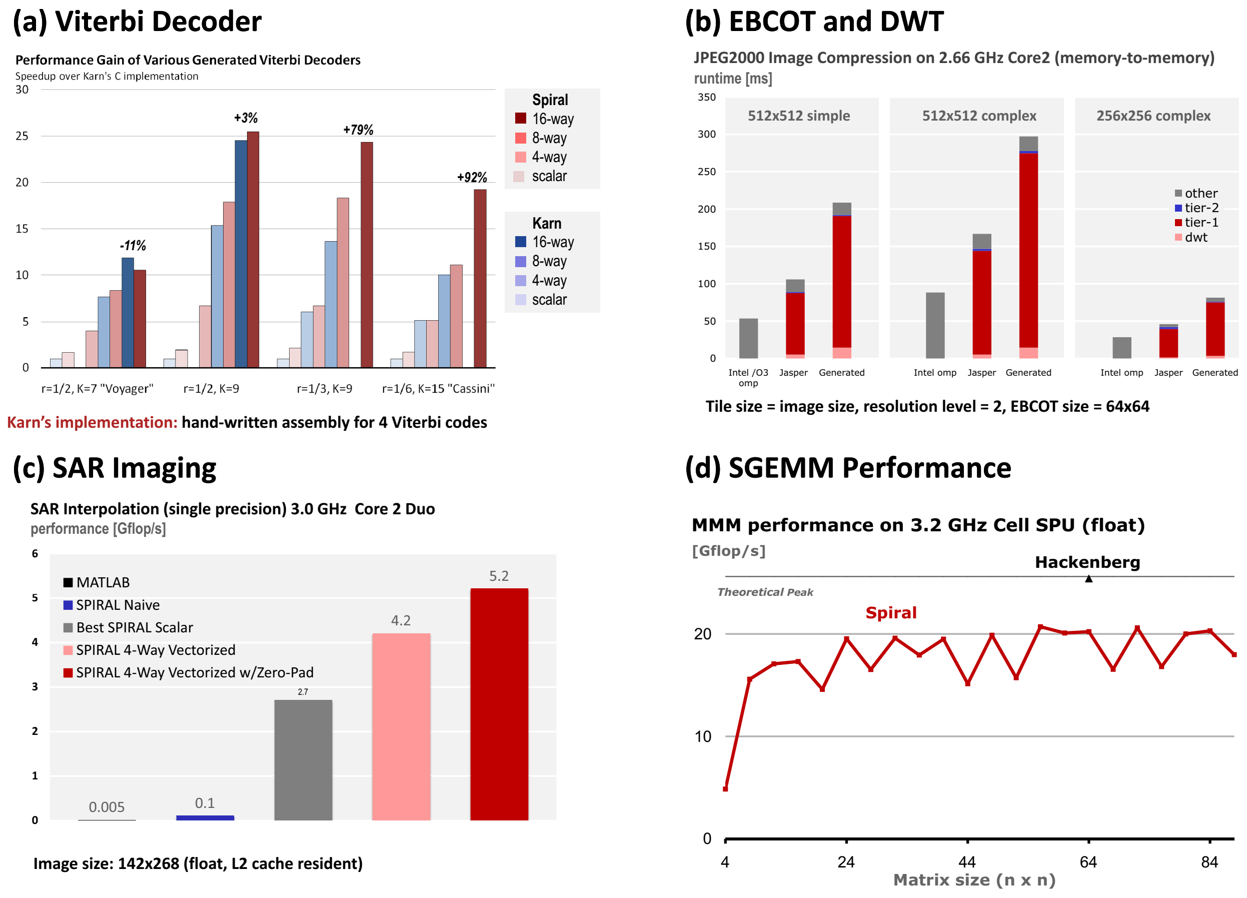

Figure 1: Experimental results. **(a)** SIMD vectorized Spiral-generated Viterbi decoder libraries perform comparably or outperform the Viterbi library by Phil Karn. **(b)** Spiral-generated JPEG2000 decoder (EBCOT and wavelet transform) performs within 2x of the C implementation Jasper and within 4x of the hand-tuned Intel IPP library. **(c)** A Spiral-generated SAR imaging implementation is sped up by SIMD vectorization and loop optimizations. **(d)** Spiral-generated matrix-matrix-multiplication on a Cell SPU (compared to Hackenberg's implementation) provides many more sizes and reaches high performance.

the respective application. Due to space limitations we omit the formulas for EBCOT and SAR imaging.

Experimental results. We summarize our experimental results in Figure 1, providing results for all four applications. We show that for the Viterbi decoder and for matrix-matrix-multiplication our generated libraries are competitive with or outperform the best available code. For JPEG2000 we show the current status (it is still work-in-progress): Our generated library is within 2x of the performance of Jasper and 4x of the Intel IPP. For SAR imaging we show how SIMD vectorization and loop optimization speeds up a reference implementation; all versions (except MATLAB) are generated by Spiral.

References

[1] Jim Demmel, Jack Dongarra, Victor Eijkhout, Erika Fuentes, Antoine Petitet, Rich Vuduc, Clint Whaley, and Katherine Yelick. Self adapting linear algebra algorithms and software. *Proceedings of the IEEE*, 93(2):293–312, 2005. Special issue on "Program Generation, Optimization, and Adaptation".

[2] Franz Franchetti, Yevgen Voronenko, Peter A. Milder, Srinivas Chellappa, Marek Telgarsky, Hao Shen, Paolo D'Alberto, Frédéric de Mesmay, James C. Hoe, José M. F. Moura, and Markus Püschel. Domain-specific library generation for parallel software and hardware platforms. In *NSF Next Generation Software Program Workshop (NSFNGS) colocated with IPDPS*, 2008.

[3] Matteo Frigo and Steven G. Johnson. The design and implementation of FFTW3. *Proceedings of the IEEE*, 93(2):216–231, 2005. Special issue on "Program Generation, Optimization, and Adaptation".

[4] José M. F. Moura, Markus Püschel, David Padua, and Jack Dongarra, editors. *Special Issue on Program Generation, Optimization, and Platform Adaptation*, volume 93(2) of *Proceedings of the IEEE*, 2005.

[5] Markus Püschel, José M. F. Moura, Jeremy Johnson, David Padua, Manuela Veloso, Bryan W. Singer, Jianxin Xiong, Franz Franchetti, Aca Gačić, Yevgen Voronenko, Kang Chen, Robert W. Johnson, and Nick Rizzolo. SPIRAL: Code generation for DSP transforms. *Proc. of the IEEE*, 93(2):232–275, 2005. Special issue on *Program Generation, Optimization, and Adaptation*.

[6] Spiral web site. www.spiral.net.