SPIRAL, FFTX, and the Path to SpectralPACK

Franz Franchetti

Carnegie Mellon University

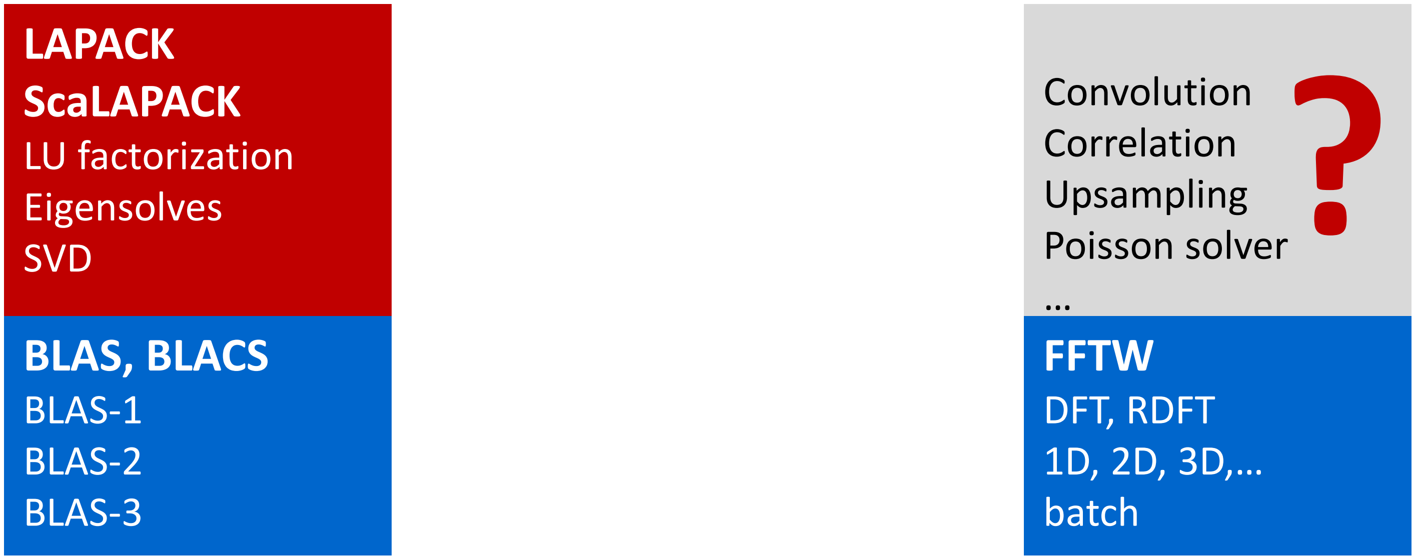
www.spiral.net

In collaboration with the SPIRAL and FFTX team @ CMU and LBL

This work was supported by DOE ECP and DARPA BRASS

EXASCALE COMPUTING PROJECT

Have You Ever Wondered About This?



Numerical Linear Algebra

LAPACK
ScaLAPACK
LU factorization
Eigensolves
SVD

BLAS, BLACS
BLAS-1
BLAS-2
BLAS-3

Spectral Algorithms

Convolution
Correlation
Upsampling
Poisson solver
…

?

FFTW
DFT, RDFT
1D, 2D, 3D,…
batch

No LAPACK equivalent for spectral methods

- **Medium size 1D FFT (1k—10k data points) is most common library call**
  applications break down 3D problems themselves and then call the 1D FFT library
- **Higher level FFT calls rarely used**
  FFTW *guru* interface is powerful but hard to used, leading to performance loss
- **Low arithmetic intensity and variation of FFT use make library approach hard**
  Algorithm specific decompositions and FFT calls intertwined with non-FFT code

It Is Worse Than It Seems

FFTW is de-facto standard interface for FFT

- **FFTW 3.X is the high performance reference implementation:** supports multicore/SMP and MPI, and Cell processor
- **Vendor libraries support the FFTW 3.X interface:** Intel MKL, IBM ESSL, AMD ACML (end-of-life), Nvidia cuFFT, Cray LibSci/CRAFFT

Issue 1: 1D FFTW call is standard kernel for many applications

- **Parallel libraries and applications reduce to 1D FFTW call** P3DFFT, QBox, PS/DNS, CPMD, HACC,…
- **Supported by modern languages and environments** Python, Matlab,…

Issue 2: FFTW is slowly becoming obsolete

- **FFTW 2.1.5 (still in use, 1997), FFTW 3 (2004) minor updates since then**
- **Development currently dormant, except for small bug fixes**
- **No native support for accelerators (GPUs, Xeon PHI, FPGAs) and SIMT**
- **Parallel/MPI version does not scale beyond 32 nodes**

Risk: loss of high performance FFT standard library

FFTX: The FFTW Revamp for ExaScale

EXASCALE COMPUTING PROJECT

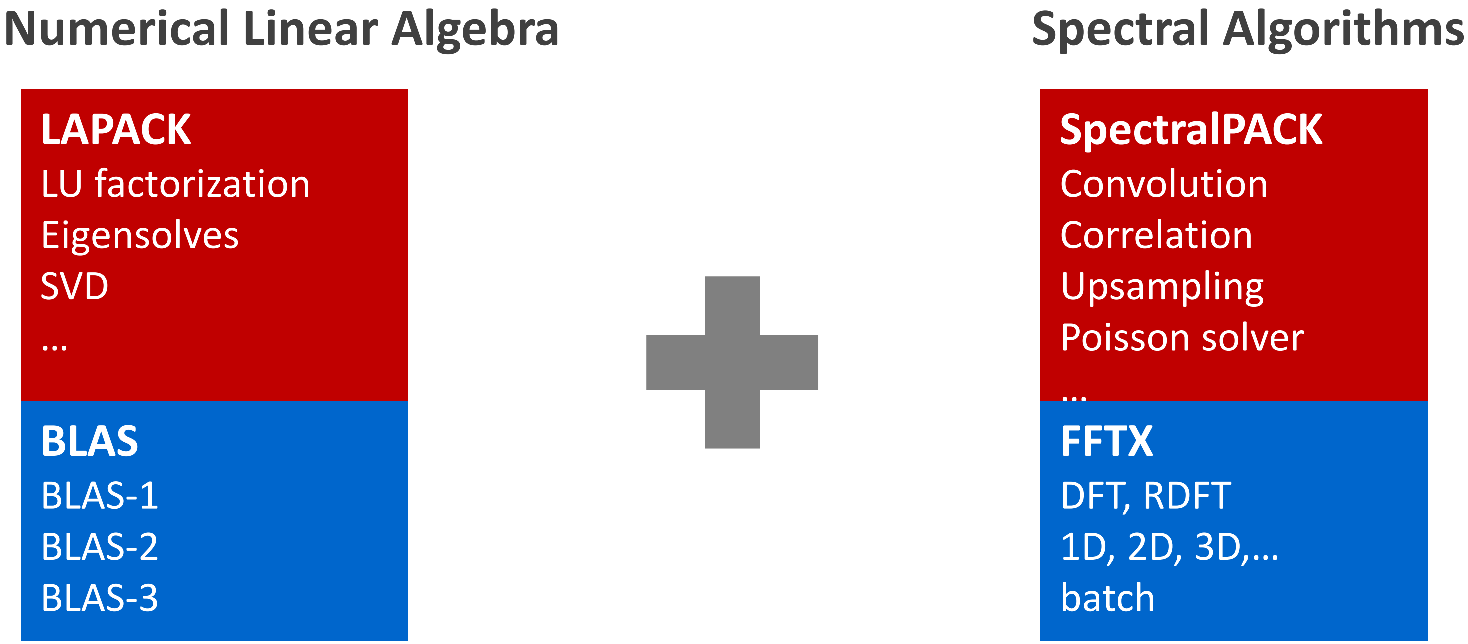
Modernized FFTW-style interface

- **Backwards compatible to FFTW 2.X and 3.X**
  old code runs unmodified and gains substantially but not fully
- **Small number of new features**
  futures/delayed execution, offloading, data placement, callback kernels

Code generation backend using SPIRAL

- **Library/application kernels are interpreted as specifications in DSL**
  extract semantics from source code and known library semantics
- **Compilation and advanced performance optimization**
  cross-call and cross library optimization, accelerator off-loading,…
- **Fine control over resource expenditure of optimization**
  compile-time, initialization-time, invocation time, optimization resources
- **Reference library implementation and bindings to vendor libraries**
  library-only reference implementation for ease of development

FFTX and SpectralPACK: Long Term Vision



Numerical Linear Algebra

LAPACK
LU factorization
Eigensolves
SVD
…

BLAS
BLAS-1
BLAS-2
BLAS-3

Spectral Algorithms

SpectralPACK
Convolution
Correlation
Upsampling
Poisson solver
…

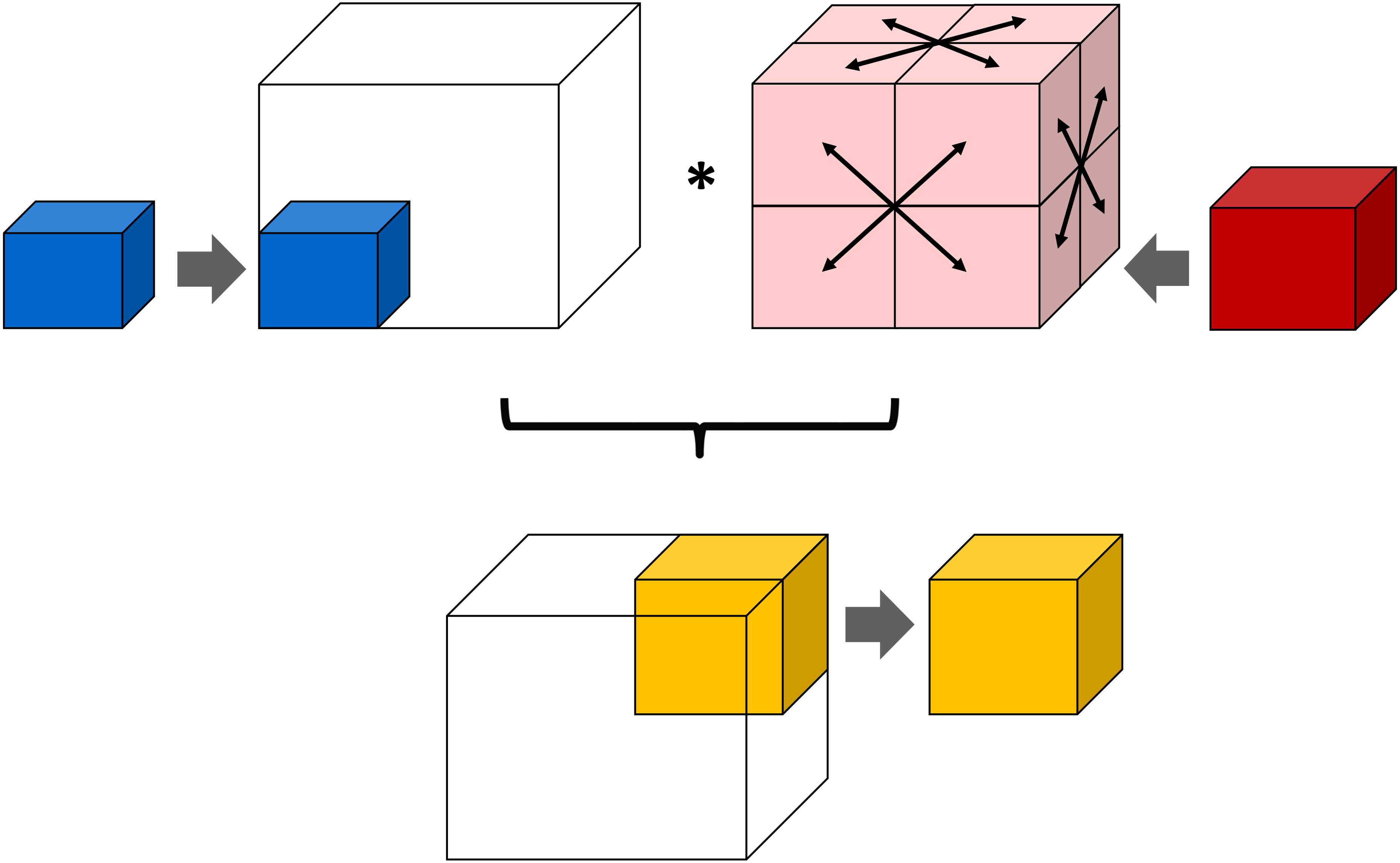
FFTX
DFT, RDFT
1D, 2D, 3D,…
batch

Define the LAPACK equivalent for spectral algorithms

- **Define FFTX as the BLAS equivalent**
  provide user FFT functionality as well as algorithm building blocks
- **Define class of numerical algorithms to be supported by SpectralPACK**
  PDE solver classes (Green's function, sparse in normal/k space,…), signal processing,…
- **Define SpectralPACK functions**
  circular convolutions, NUFFT, Poisson solvers, free space convolution,…

FFTX and SpectralPACK solve the "spectral dwarf" long term

Example: Hockney Free Space Convolution



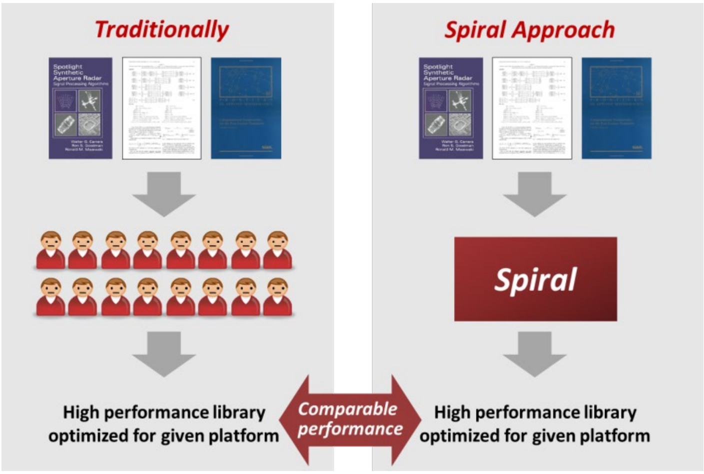
Example: Hockney Free Space Convolution

```
fftx_plan pruned_real_convolution_plan(fftx_real *in, fftx_real *out, fftx_complex *symbol,
        int n, int n_in, int n_out, int n_freq) {
    int rank = 1,
    batch_rank = 0,
    ...
    fftx_plan plans[5];
    fftx_plan p;

    tmp1 = fftx_create_zero_temp_real(rank, &padded_dims);

    plans[0] = fftx_plan_guru_copy_real(rank, &in_dimx, in, tmp1, MY_FFTX_MODE_SUB);

    tmp2 = fftx_create_temp_complex(rank, &freq_dims);
    plans[1] = fftx_plan_guru_dft_r2c(rank, &padded_dims, batch_rank,
        &batch_dims, tmp1, tmp2, MY_FFTX_MODE_SUB);

    tmp3 = fftx_create_temp_complex(rank, &freq_dims);
    plans[2] = fftx_plan_guru_pointwise_c2c(rank, &freq_dimx, batch_rank, &batch_dimx,
        tmp2, tmp3, symbol, (fftx_callback)complex_scaling,
        MY_FFTX_MODE_SUB | FFTX_PW_POINTWISE);

    tmp4 = fftx_create_temp_real(rank, &padded_dims);
    plans[3] = fftx_plan_guru_dft_c2r(rank, &padded_dims, batch_rank,
        &batch_dims, tmp3, tmp4, MY_FFTX_MODE_SUB);

    plans[4] = fftx_plan_guru_copy_real(rank, &out_dimx, tmp4, out, MY_FFTX_MODE_SUB);

    p = fftx_plan_compose(numsubplans, plans, MY_FFTX_MODE_TOP);

    return p;
}
```

Looks like FFTW calls, but is a specification for SPIRAL

Spiral Technology in a Nutshell

Library Generator

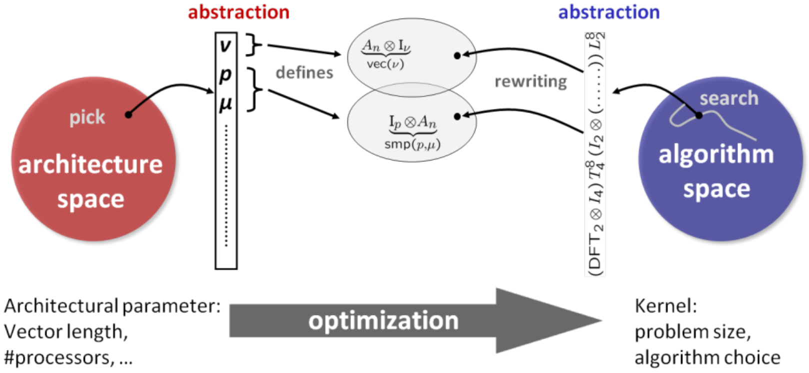


High performance library optimized for given platform

High performance library optimized for given platform

Mathematical Foundation

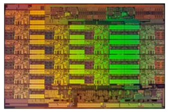
Model: common abstraction
= spaces of matching formulas


abstraction
abstraction
pick
architecture space
search
algorithm space
v
p
μ
defines
$\underbrace{A_n \otimes I_\nu}_{\text{vec}(\nu)}$
$\underbrace{I_p \otimes A_n}_{\text{smp}(p,\mu)}$
rewriting
$(\mathrm{DFT}_2 \otimes I_4)\, T^8_4\, (I_2 \otimes (\ldots\ldots))\, L^8_2$
optimization

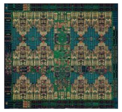
Architectural parameter:
Vector length,
#processors, …

Kernel:
problem size,
algorithm choice

Performance Portability



Intel Xeon 8180M
2.25 Tflop/s, 205 W
28 cores, 2.5—3.8 GHz
2-way—16-way AVX-512



IBM POWER9
768 Gflop/s, 300 W
24 cores, 4 GHz
4-way VSX-3

Nvidia Tesla V100
7.8 Tflop/s, 300 W
5120 cores, 1.2 GHz
32-way SIMT

Intel Xeon Phi 7290F
1.7 Tflop/s, 260 W
72 cores, 1.5 GHz
8-way/16-way LRBni

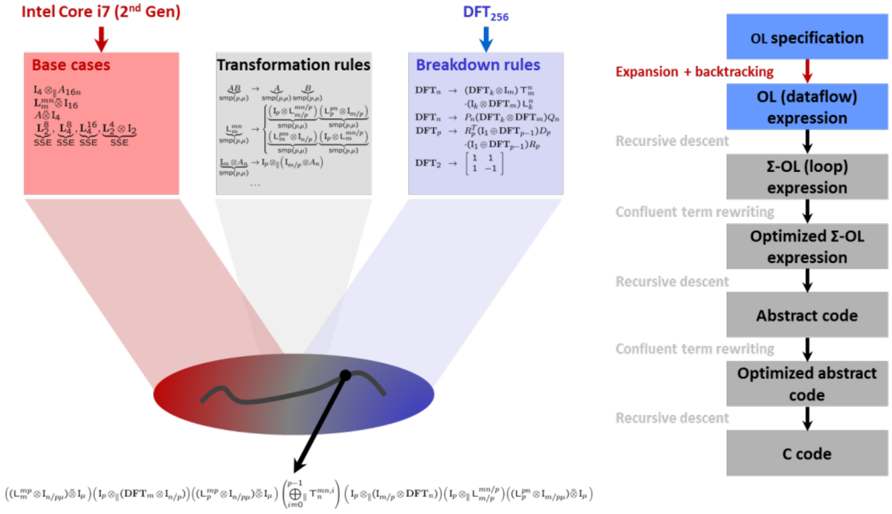
Snapdragon 835
15 Gflop/s, 2 W
8 cores, 2.3 GHz
A540 GPU, 682 DSP, NEON

Intel Atom C3858
32 Gflop/s, 25 W
16 cores, 2.0 GHz
2-way/4-way SSSE3

Dell PowerEdge R940
3.2 Tflop/s, 6 TB, 850 W
4x 24 cores, 2.1 GHz
4-way/8-way AVX

Summit
187.7 Pflop/s, 13 MW
9,216 x 22 cores POWER9
+ 27,648 V100 GPUs

Code Synthesis and Autotuning



Intel Core i7 (2nd Gen)

DFT_{256}

Base cases

$$I_4 \otimes_{\|} A_{16n}$$
$$L^{mn}_m \bar{\otimes} I_{16}$$
$$A \bar{\otimes} I_4$$
$$\underbrace{L^8_2}_{\text{SSE}}, \underbrace{L^8_4}_{\text{SSE}}, \underbrace{L^{16}_4}_{\text{SSE}}, \underbrace{L^4_2 \otimes I_2}_{\text{SSE}}$$

Transformation rules

$$\underbrace{AB}_{\text{smp}(p,\mu)} \to \underbrace{A}_{\text{smp}(p,\mu)} \underbrace{B}_{\text{smp}(p,\mu)}$$
$$\underbrace{L^{mn}_m}_{\text{smp}(p,\mu)} \to \begin{cases} \underbrace{\left(I_p \otimes L^{mn/p}_{m/p}\right)}_{\text{smp}(p,\mu)} \underbrace{\left(L^{pm}_p \otimes I_{m/p}\right)}_{\text{smp}(p,\mu)} \\ \underbrace{\left(L^{pm}_m \otimes I_{n/p}\right)}_{\text{smp}(p,\mu)} \underbrace{\left(I_p \otimes L^{mn/p}_m\right)}_{\text{smp}(p,\mu)} \end{cases}$$
$$\underbrace{I_m \otimes A_n}_{\text{smp}(p,\mu)} \to I_p \otimes_{\|} \left(I_{m/p} \otimes A_n\right)$$
$$\ldots$$

Breakdown rules

$$\mathrm{DFT}_n \to (\mathrm{DFT}_k \otimes I_m)\, T^n_m \cdot (I_k \otimes \mathrm{DFT}_m)\, L^n_k$$
$$\mathrm{DFT}_n \to P_n(\mathrm{DFT}_k \otimes \mathrm{DFT}_m) Q_n$$
$$\mathrm{DFT}_p \to R^T_p (I_1 \oplus \mathrm{DFT}_{p-1}) D_p \cdot (I_1 \oplus \mathrm{DFT}_{p-1}) R_p$$
$$\mathrm{DFT}_2 \to \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

$$\left((L^{mp}_m \otimes I_{n/p\mu}) \bar{\otimes} I_\mu\right)\left(I_p \otimes_{\|} (\mathrm{DFT}_m \otimes I_{n/p})\right)\left((L^{mp}_p \otimes I_{n/p\mu}) \bar{\otimes} I_\mu\right)\left(\bigoplus_{i=0}^{p-1}{}_{\|}\, T^{mn,i}_n\right)\left(I_p \otimes_{\|} (I_{m/p} \otimes \mathrm{DFT}_n)\right)\left(I_p \otimes_{\|} L^{mn/p}_{m/p}\right)\left((L^{pn}_p \otimes I_{m/p\mu}) \bar{\otimes} I_\mu\right)$$

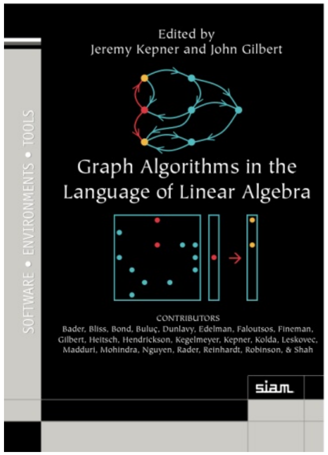
OL specification
Expansion + backtracking
OL (dataflow) expression
Recursive descent
Σ-OL (loop) expression
Confluent term rewriting
Optimized Σ-OL expression
Recursive descent
Abstract code
Confluent term rewriting
Optimized abstract code
Recursive descent
C code

Algorithms: Rules in Domain Specific Language

Linear Transforms

$$\begin{array}{rcl}
\mathbf{DFT}_n & \rightarrow & (\mathbf{DFT}_k \otimes \mathrm{I}_m)\, \mathsf{T}^n_m (\mathrm{I}_k \otimes \mathbf{DFT}_m)\, \mathsf{L}^n_k, \quad n = km \\
\mathbf{DFT}_n & \rightarrow & P_n(\mathbf{DFT}_k \otimes \mathbf{DFT}_m) Q_n, \quad n = km,\ \mathsf{gcd}(k,m) = 1 \\
\mathbf{DFT}_p & \rightarrow & R_p^T(\mathrm{I}_1 \oplus \mathbf{DFT}_{p-1}) D_p (\mathrm{I}_1 \oplus \mathbf{DFT}_{p-1}) R_p, \quad p \text{ prime} \\
\mathbf{DCT\text{-}3}_n & \rightarrow & (\mathrm{I}_m \oplus \mathsf{J}_m)\, \mathsf{L}^n_m (\mathbf{DCT\text{-}3}_m(1/4) \oplus \mathbf{DCT\text{-}3}_m(3/4)) \\
 & & \cdot(\mathsf{F}_2 \otimes \mathrm{I}_m) \begin{bmatrix} \mathrm{I}_m & 0 \oplus -\mathsf{J}_{m-1} \\ & \frac{1}{\sqrt{2}}(\mathrm{I}_1 \oplus 2\,\mathrm{I}_m) \end{bmatrix}, \quad n = 2m \\
\mathbf{DCT\text{-}4}_n & \rightarrow & S_n \mathbf{DCT\text{-}2}_n\, \mathsf{diag}_{0 \le k < n}(1/(2\cos((2k+1)\pi/4n))) \\
\mathbf{IMDCT}_{2m} & \rightarrow & (\mathsf{J}_m \oplus \mathrm{I}_m \oplus \mathrm{I}_m \oplus \mathsf{J}_m) \left(\left(\begin{bmatrix} 1 \\ -1 \end{bmatrix} \otimes \mathrm{I}_m \right) \oplus \left(\begin{bmatrix} -1 \\ -1 \end{bmatrix} \otimes \mathrm{I}_m \right) \right) \mathsf{J}_{2m}\, \mathbf{DCT\text{-}4}_{2m} \\
\mathbf{WHT}_{2^k} & \rightarrow & \prod_{i=1}^{t} (\mathrm{I}_{2^{k_1+\cdots+k_{i-1}}} \otimes \mathbf{WHT}_{2^{k_i}} \otimes \mathrm{I}_{2^{k_{i+1}+\cdots+k_t}}), \quad k = k_1 + \cdots + k_t \\
\mathbf{DFT}_2 & \rightarrow & \mathsf{F}_2 \\
\mathbf{DCT\text{-}2}_2 & \rightarrow & \mathsf{diag}(1, 1/\sqrt{2})\, \mathsf{F}_2 \\
\mathbf{DCT\text{-}4}_2 & \rightarrow & \mathsf{J}_2\, \mathsf{R}_{13\pi/8}
\end{array}$$

Graph Algorithms



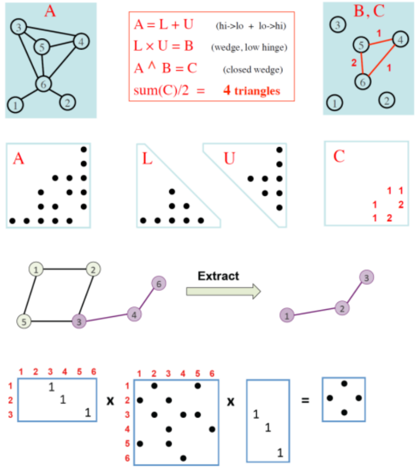
SOFTWARE • ENVIRONMENTS • TOOLS

Edited by
Jeremy Kepner and John Gilbert

Graph Algorithms in the Language of Linear Algebra

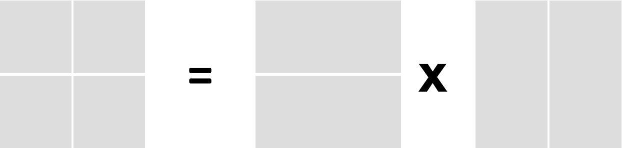
CONTRIBUTORS
Bader, Bliss, Bond, Buluç, Dunlavy, Edelman, Faloutsos, Fineman, Gilbert, Heitsch, Hendrickson, Kegelmeyer, Kepner, Kolda, Leskovec, Madduri, Mohindra, Nguyen, Rader, Reinhardt, Robinson, & Shah

siam



In collaboration with CMU-SEI

Numerical Linear Algebra



$$\mathsf{MMM}_{1,1,1} \rightarrow (\cdot)_1$$

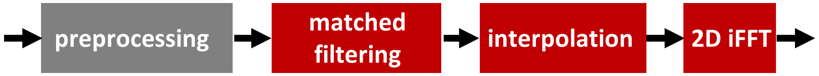
$$\mathsf{MMM}_{m,n,k} \rightarrow (\otimes)_{m/m_b \times 1} \otimes \mathsf{MMM}_{m_b,n,k}$$

$$\mathsf{MMM}_{m,n,k} \rightarrow \mathsf{MMM}_{m,nb,k} \otimes (\otimes)_{1 \times n/nb}$$

$$\begin{aligned}\mathsf{MMM}_{m,n,k} \rightarrow & ((\Sigma_{k/k_b} \circ (\cdot)_{k/k_b}) \otimes \mathsf{MMM}_{m,n,k_b}) \circ \\ & ((L^{mk/k_b}_{k/k_b} \otimes I_{k_b}) \times I_{kn})\end{aligned}$$

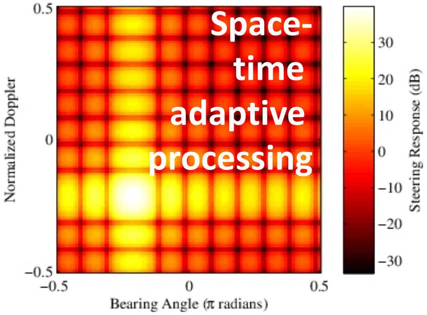
$$\begin{aligned}\mathsf{MMM}_{m,n,k} \rightarrow & (L^{mn/n_b}_m \otimes I_{n_b}) \circ \\ & ((\otimes)_{1 \times n/n_b} \otimes \mathsf{MMM}_{m,n_b,k}) \circ \\ & (I_{km} \times (L^{kn/n_b}_{n/n_b} \otimes I_{n_b}))\end{aligned}$$

Spectral Domain Applications



preprocessing → matched filtering → interpolation → 2D iFFT

Synthetic aperture radar



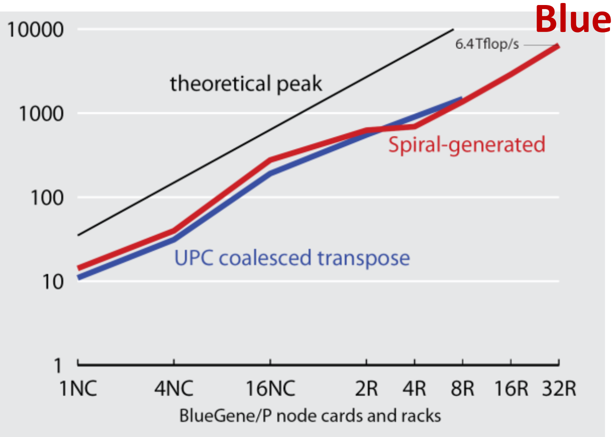
Space-time adaptive processing

SPIRAL: Success in HPC/Supercomputing

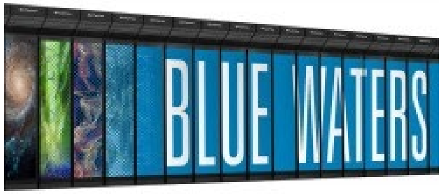
- **NCSA Blue Waters**
  PAID Program, FFTs for Blue Waters
- **RIKEN K computer**
  FFTs for the HPC-ACE ISA
- **LANL RoadRunner**
  FFTs for the Cell processor
- **PSC/XSEDE Bridges**
  Large size FFTs
- **LLNL BlueGene/L and P**
  FFTW for BlueGene/L's Double FPU
- **ANL BlueGene/Q Mira**
  Early Science Program, FFTW for BGQ QPX

Global FFT (1D FFT, HPC Challenge)
performance [Gflop/s]

6.4 Tflop/s on BlueGene/P



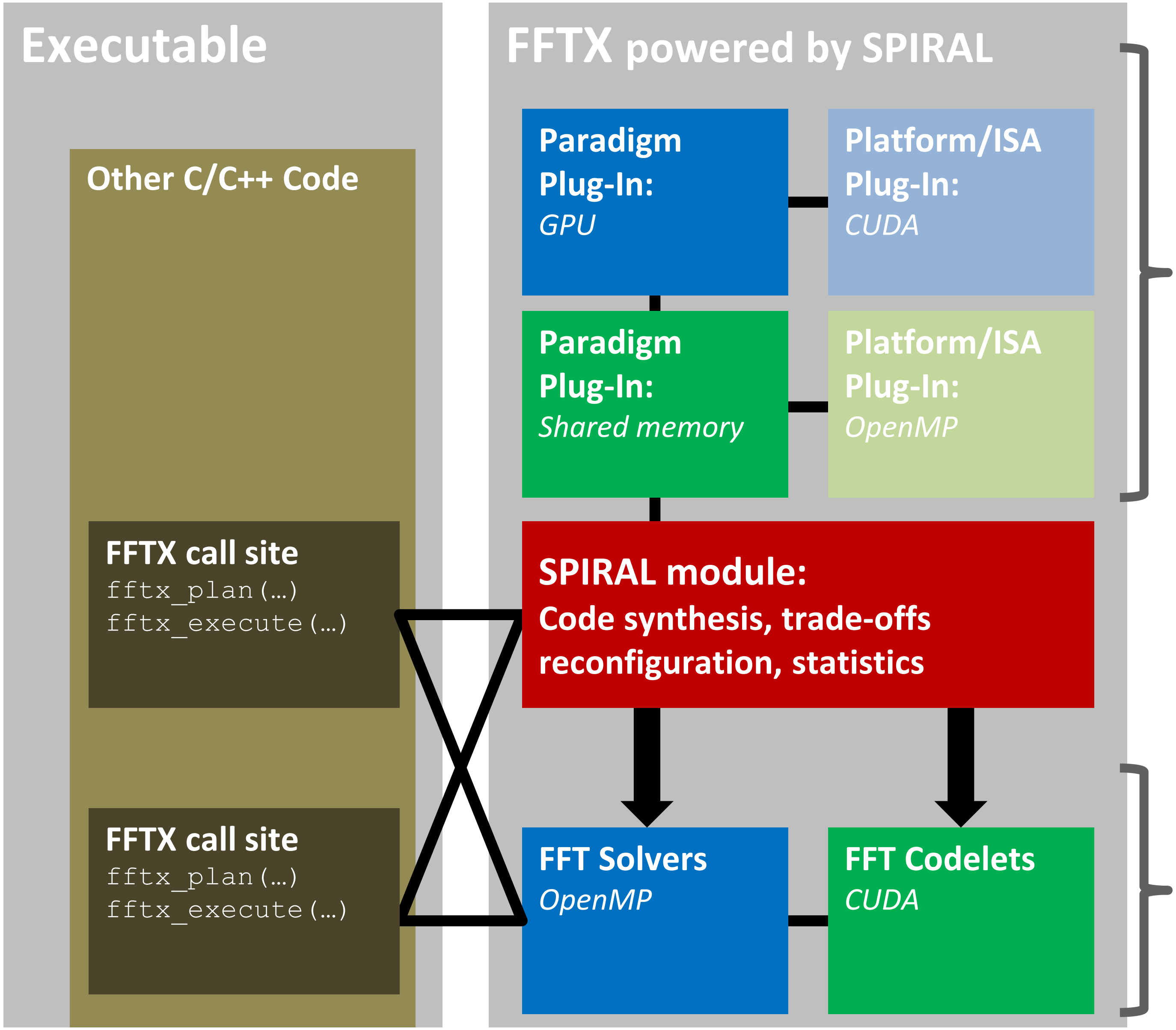
BlueGene/P at Argonne National Laboratory
128k cores (quad-core CPUs) at 850 MHz



2006 Gordon Bell Prize (Peak Performance Award) with LLNL and IBM

2010 HPC Challenge Class II Award (Most Productive System) with ANL and IBM

FFTX Backend: SPIRAL



Executable

Other C/C++ Code

FFTX call site
`fftx_plan(…)`
`fftx_execute(…)`

FFTX call site
`fftx_plan(…)`
`fftx_execute(…)`

FFTX powered by SPIRAL

Paradigm Plug-In: *GPU*

Platform/ISA Plug-In: *CUDA*

Paradigm Plug-In: *Shared memory*

Platform/ISA Plug-In: *OpenMP*

SPIRAL module:
Code synthesis, trade-offs reconfiguration, statistics

FFT Solvers *OpenMP*

FFT Codelets *CUDA*

Extensible platform and programming model definitions

Core system: SPIRAL engine

Automatically Generated FFTW-like library components

FFTX: First Results for Hockney on Volta

FFTX and SpectralPack: A First Look

Franz Franchetti, Daniele G. Spampinato, Anuva Kulkarni, Doru Thom Popovici, Tze Meng Low
Electrical and Computer Engineering Department
Carnegie Mellon University
Pittsburgh, PA, USA
{franzf, spampinato}@cmu.edu, anuvak@andrew.cmu.edu, {dpopovic, lowt}@cmu.edu

Michael Franusich
SpiralGen, Inc.
Pittsburgh, PA, USA
mike.franusich@spiralgen.com

Andrew Canning, Peter McCorquodale, Brian Van Straalen, Phillip Colella
Lawrence Berkeley National Laboratory
Berkeley, CA, USA
{acanning, pwmccorquodale, bvstraalen, pcolella}@lbl.gov

***Abstract*—We propose FFTX, a new framework for building high-performance FFT-based applications on exascale machines. Complex node architectures lead to multiple levels of parallelism and demand efficient ways of data communication. The current FFTW interface falls short in maximizing performance in such scenarios. FFTX is designed to enable application developers to leverage expert-level, automatic optimizations while navigating a familiar interface. FFTX is backwards compatible to FFTW and extends the FFTW Interface into an embedded Domain Specific Language (DSL) expressed as a library interface. By means of a SPIRAL-based back end, this enables build-time source-to-source translation and advanced performance optimizations, such as cross-library calls optimizations, targeting of accelerators through offloading, and inlining of user-provided kernels. We demonstrate the use of FFTX with the prototypical example of 1D and 3D pruned convolutions and discuss future extensions.**

***Keywords*-FFT; exasclale; code generator; high-performance;**

I. Introduction

The Discrete Fourier Transform (DFT) — and in particular its implementation using Fast Fourier Transform (FFT) [1], [2] algorithms — is a fundamental component for the design of scientific applications suitable for the emerging exascale computing ecosystem. Application domains include material science, chemistry, molecular dynamics, and cosmology. Some examples are FFT-based differential equation solvers to compute properties of materials such as stress and strain [3], simulation of incompressible flows [4], plane wave based electronic structure methods [5] [6] and particle-in-cell (PIC) codes [7].

Capturing application-specific properties of the FFTs is very important to enable high-performance execution on emerging platforms. For example, applications where a large subset of the inputs or outputs is either set to zero or not computed at all should exploit the zero structures to reduce data movements within the memory hierarchy on a node or across the network. The structure of multidimensional FFTs provide opportunities for parallelism at multiple levels — SIMD, threads, accelerators, and distributed systems. The best strategy for exploiting these opportunities strongly depends on the details of the use case and of the computing platform, as well as tradeoffs with other needs of the application in which the transforms are embedded.

Conventional FFT-based implementation approach and its limits. The implementation strategy for most of today's large science applications that depend on FFTs constists in transforming multidimensional problems into a sequence of 1D FFT calls, with the latter being performed by a library. Over the last decade or so the FFTW API became the de-facto standard FFT interface [8], [9], [10]. Vendors that provide FFT libraries like Intel, Cray, and IBM may still provide their own proprietary interface for backwards compatibility reasons, but all current vendor high-performance libraries, including Intel MKL [11], IBM ESSL [12], and Nvidia cuFFT [13], implement (at least a subset) of the FFTW interface. Thus, FFTW defines the standard FFT library interface and oftentimes is considered a key component of today's applications.

However, the approach of building up high-performance implementations out of calls to 1D FFTW kernels is breaking down on the current and emerging HPC platforms, for two reasons: First, node architectures have become more complex. Multiple cores and accelerators lead to multiple levels of parallelism, including threading and SIMD/SIMT. In addition, there are on-node complex memory hierarchies that are to varying extents user-controlled, and across which it is expensive to move data. This leads to more complex mappings of multidimensional FFT-based applications to the core 1D FFTs. Some of these are simply not expressible in the current FFTW interface; others can be expressed, but with significant programming effort, and often below the theoretically-predicted performance due to unexpected and opaque behavior of the FFT library software. Second, FFTW is no longer supported. The system comprises a high-level domain-specific language, a symbolic transformation/code

FFTX with SPIRAL and OpenACC: on par with cuFFT expert interface

Tesla V100 @ PSC

FFTX with SPIRAL and OpenACC: 15 % faster than cuFFT expert interface

TITAN V @ CMU

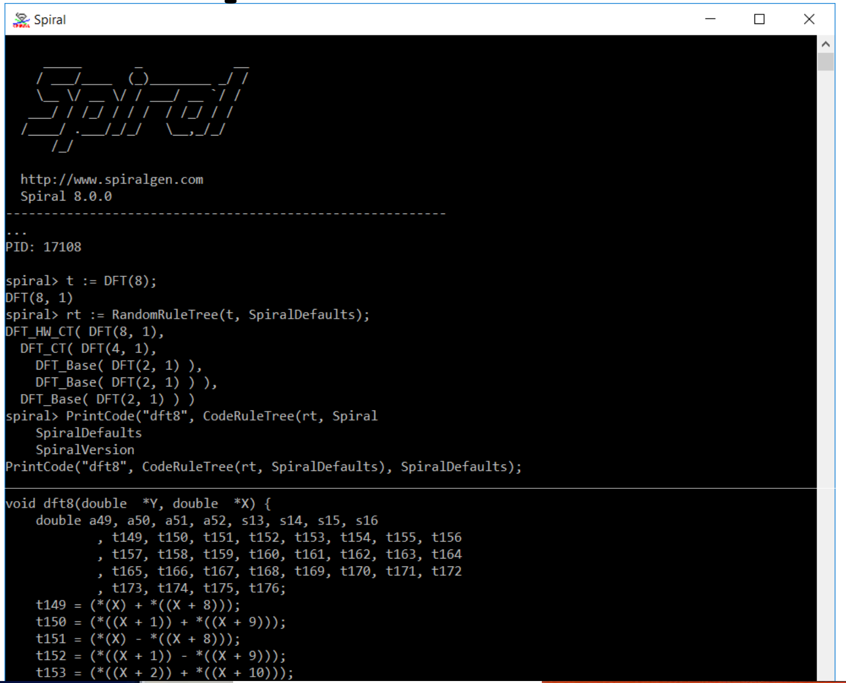
F. Franchetti, D. G. Spampinato, A. Kulkarni, D. T. Popovici, T. M. Low, M. Franusich, A. Canning, P. McCorquodale, B. Van Straalen, P. Colella:
FFTX and SpectralPack: A First Look, Workshop on Parallel Fast Fourier Transforms (PFFT), *to appear*.

http://www.spiral.net/doc/fftx

SPIRAL 8.0: Available Under Open Source

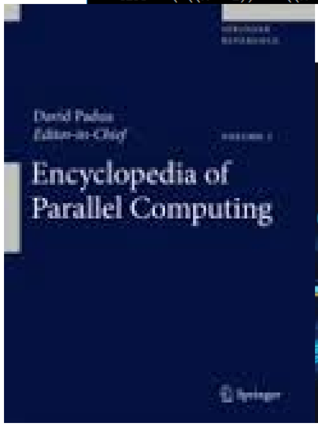
- **Open Source SPIRAL available**
  - non-viral license (BSD)
  - Initial version, effort ongoing to open source whole system
  - Commercial support via SpiralGen, Inc.
- **Developed over 20 years**
  - Funding: DARPA (OPAL, DESA, HACMS, PERFECT, BRASS), NSF, ONR, DoD HPC, JPL, DOE, CMU SEI, Intel, Nvidia, Mercury
- **Open sourced under DARPA PERFECT**

www.spiral.net

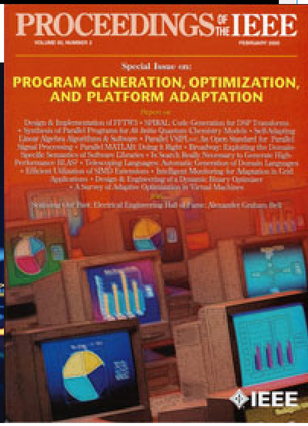


Spiral

```
   _____       _          __
  / ___/___   (_)_______ _/ /
  \__ \/ __ \/ / ___/ __ `/ /
 ___/ / /_/ / / /  / /_/ / /
/____/ .___/_/_/   \__,_/_/
    /_/

  http://www.spiralgen.com
  Spiral 8.0.0
----------------------------------------------------------
...
PID: 17108

spiral> t := DFT(8);
DFT(8, 1)
spiral> rt := RandomRuleTree(t, SpiralDefaults);
DFT_HW_CT( DFT(8, 1),
  DFT_CT( DFT(4, 1),
    DFT_Base( DFT(2, 1) ),
    DFT_Base( DFT(2, 1) ) ),
  DFT_Base( DFT(2, 1) ) )
spiral> PrintCode("dft8", CodeRuleTree(rt, Spiral
    SpiralDefaults
    SpiralVersion
PrintCode("dft8", CodeRuleTree(rt, SpiralDefaults), SpiralDefaults);

void dft8(double  *Y, double  *X) {
    double a49, a50, a51, a52, s13, s14, s15, s16
            , t149, t150, t151, t152, t153, t154, t155, t156
            , t157, t158, t159, t160, t161, t162, t163, t164
            , t165, t166, t167, t168, t169, t170, t171, t172
            , t173, t174, t175, t176;
    t149 = (*(X) + *((X + 8)));
    t150 = (*((X + 1)) + *((X + 9)));
    t151 = (*(X) - *((X + 8)));
    t152 = (*((X + 1)) - *((X + 9)));
    t153 = (*((X + 2)) + *((X + 10)));
```



November 2018 | Volume 106 | Number 11
Proceedings OF THE IEEE
SPECIAL ISSUE
From High-Level Specification to High-Performance Code
IEEE



F. Franchetti, T. M. Low, D. T. Popovici, R. M. Veras, D. G. Spampinato, J. R. Johnson, M. Püschel, J. C. Hoe, J. M. F. Moura: SPIRAL: Extreme Performance Portability, **Proceedings of the IEEE, Vol. 106, No. 11, 2018.**
Special Issue on *From High Level Specification to High Performance Code*