FFT and Solver Libraries for Exascale: FFTX and SpectralPack

F. Franchetti, D. G. Spampinato, A. Kulkarni, T. M. Low (Carnegie Mellon University), M. Franusich (SpiralGen, Inc.),
D. T. Popovici, A. Canning, P. McCorquodale, B. Van Straalen, P. Colella (Lawrence Berkeley National Laboratory)

Our approach

Have you ever wondered about this?

LAPACK
ScaLAPACK
LU factorization
Eigensolves
SVD
…
BLAS, BLACS
BLAS-1
BLAS-2
BLAS-3
Convolution
Correlation
Upsampling
Poisson solver
…
?
FFTW
DFT, RDFT
1D, 2D, 3D,…
batch

No analogue to LAPACK 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 use, 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

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** *Risk: loss of high-performance FFT standard library*
- **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**

FFTX: FFTW revamped for Exascale

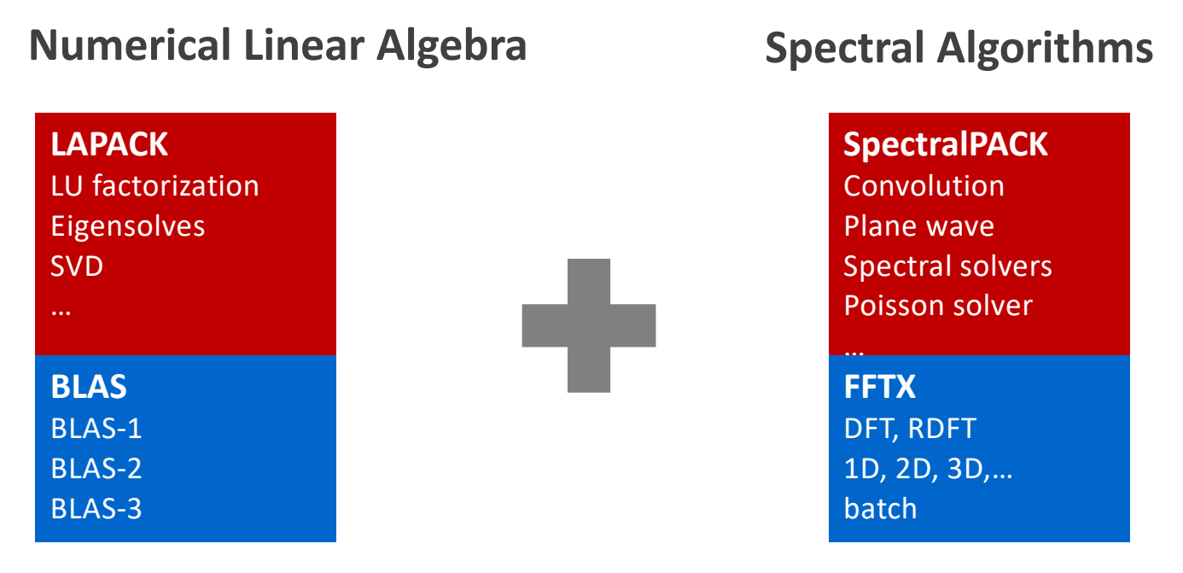
Modernized FFTW-style interface

- **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



Define the analogue to LAPACK for spectral algorithms

- **Define FFTX as the analogue to BLAS**
  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, plane wave, …

Front end

Poisson's equation in free space

Partial differential equation (PDE)

$$\Delta(\Phi) = \rho$$
$$\rho : \mathbb{R}^3 \to \mathbb{R}$$
$$D = \operatorname{supp}(\rho) \subset \mathbb{R}^3$$

Poisson's equation. Δ is the Laplace operator

Solution characterization

$$\Phi : \mathbb{R}^3 \to \mathbb{R}$$
$$\Phi(\vec{x}) = \frac{Q}{4\pi||\vec{x}||} + o\left(\frac{1}{||\vec{x}||}\right) \text{ as } ||\vec{x}|| \to \infty$$
$$Q = \int_D \rho d\vec{x}$$

Approach: Green's function

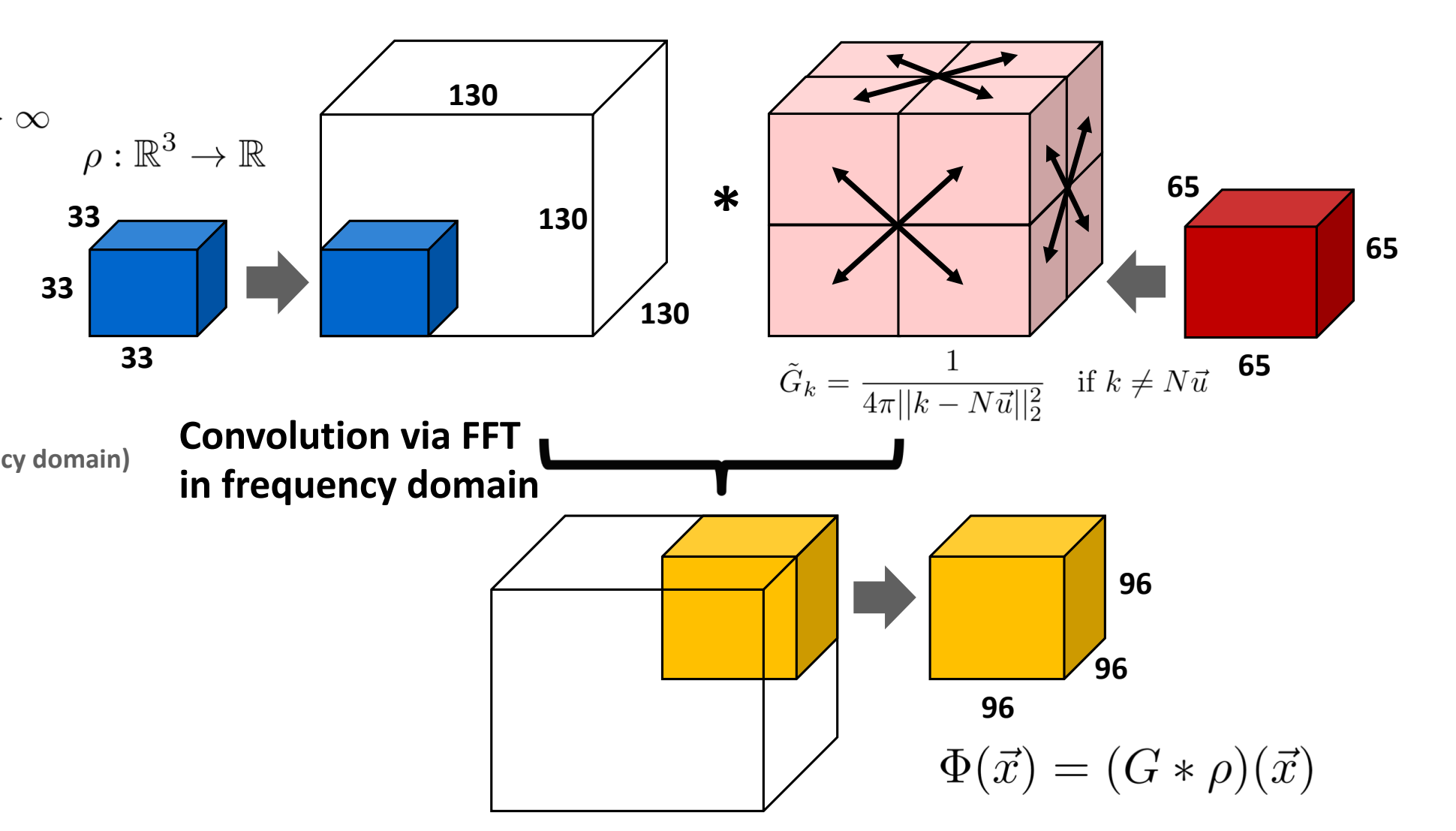
$$\Phi(\vec{x}) = \int_D G(\vec{x}-\vec{y})\rho(\vec{y})d\vec{y} \equiv (G*\rho)(\vec{x}), \quad G(\vec{x}) = \frac{1}{4\pi||\vec{x}||_2}$$

Solution: ϕ(.) = convolution of RHS *ρ*(.) with Green's function *G*(.). Efficient through FFTs (frequency domain)

$$\tilde{G}_k = \frac{1}{4\pi||k - N\vec{u}||_2^2} \quad \text{if } k \neq N\vec{u}$$

Green's function kernel in frequency domain

Hockney free-space convolution


$\rho : \mathbb{R}^3 \to \mathbb{R}$
33
33
33
130
130
130
*
65
65
65
$\tilde{G}_k = \frac{1}{4\pi||k - N\vec{u}||_2^2}$ if $k \neq N\vec{u}$
Convolution via FFT in frequency domain
96
96
96
$\Phi(\vec{x}) = (G*\rho)(\vec{x})$

Hockney: Convolution + problem-specific zero padding and output subset

FFTX 2.0 source code for Hockney

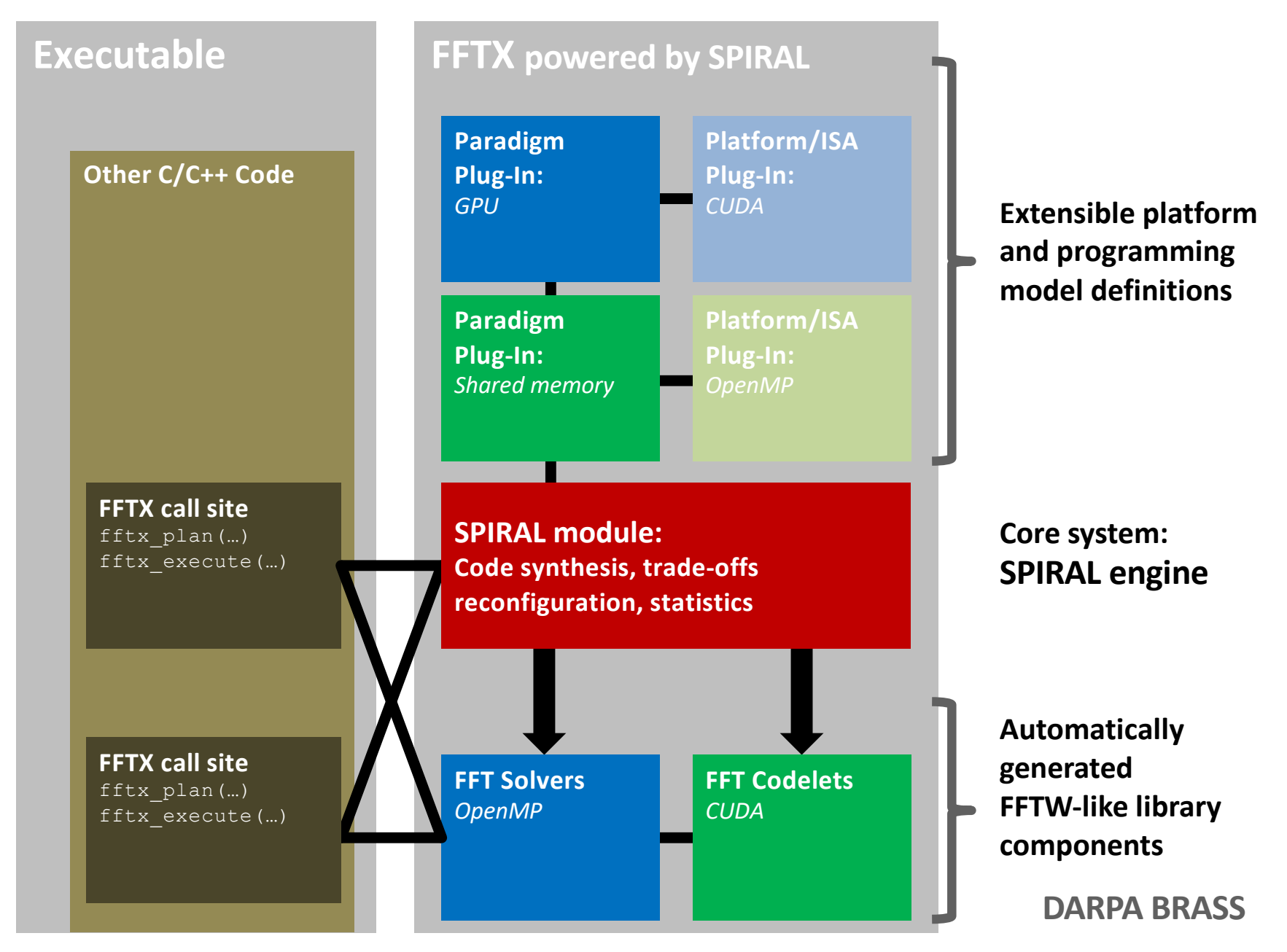
```
box_t<3> inputBox(point_t<3>({{0,0,0}}),point_t<3>({32,32,32}));
array_t<3, double> rho(inputBox);
// ... set input values.

box_t<3> transformBox(point_t<3>({{0,0,0}}),point_t<3>({{129,129,129}}));
box_t<3> outputBox(point_t<3>({33,33,33}),point_t<3>({129,129,129}));

point_t<3> kindF({{DFT,DFT,DFT}});

size_t normalize = normalization(transformBox);

auto forward_plan =
    plan_dft<3,double,std::complex<double>(kindF,inputBox,transformBox,
        transformBox);

auto kernel_plan = kernel<3,std::complex<double> >(greensFunction,
    transformBox, normalize);

point_t<3> kindI({{IDFT,IDFT,IDFT}});
auto inverse_plan = plan_dft<3, std::complex<double >, double>
                    (kindI, transformBox, outputBox, transformBox);

auto solver = chain(chain(forward_plan,kernel_plan),inverse_plan);

context_t context;
context_omp(context, 8);

std::ofstream splFile("hockney.spl");
export_spl(context, solver, splFile, "hockney33_97_130");
splFile.close();
// Offline codegen.
auto fptr = import_spl<3, double, double>("hockney33_97_130");
array_t<3, double> Phi(inputBox);
fptr(&rho, &Phi, 1);
```

FFTX-generated SPIRAL script

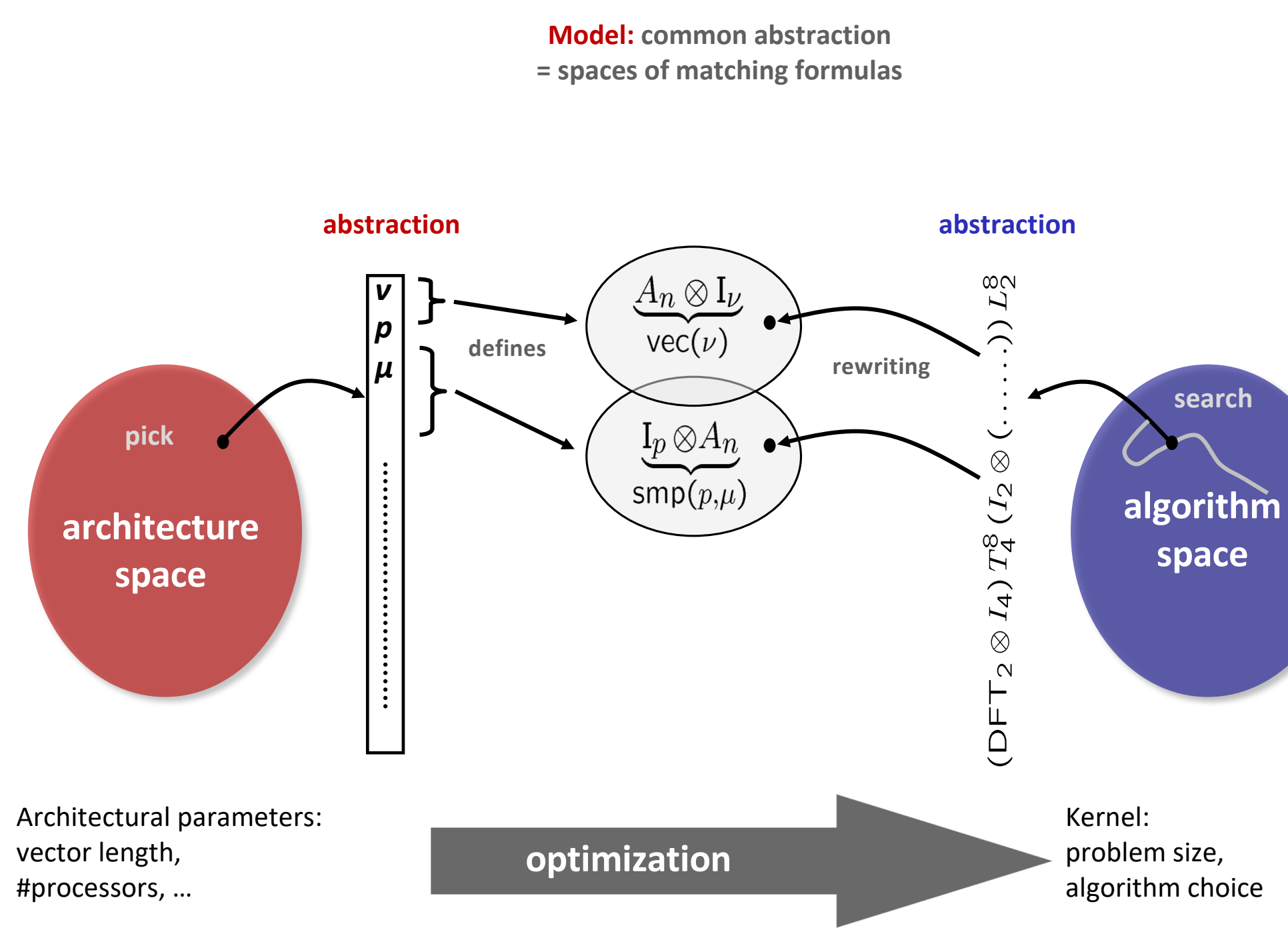
```
# Pruned 3D Real Convolution Pattern
Import(realdft);
Import(filtering);

# set up algorithms needed for multi-dimensional pruned real convolution
opts := SpiralDefaults;
opts.breakdownRules.PRDFT := [ PRDFT1_Base1, PRDFT1_Base2, PRDFT1_CT,
  PRDFT1_PF, PRDFT_PD, PRDFT_Rader];
opts.breakdownRules.IPRDFT := [ IPRDFT1_Base1, IPRDFT1_Base2,
  IPRDFT1_CT, IPRDFT_PD, IPRDFT_Rader];
opts.breakdownRules.IPRDFT2 := [ IPRDFT2_Base1, IPRDFT2_Base2, IPRDFT2_CT];
opts.breakdownRules.PRDFT3 := [ PRDFT3_Base1, PRDFT3_Base2, PRDFT3_CT,
PRDFT3_OddToPRDFT1];
opts.breakdownRules.URDFT := [ URDFT1_Base1, URDFT1_Base2, URDFT1_Base4,
URDFT1_CT ];

# specification parameters
[N, maxin, maxout, name] := Parse("fftx-trace.g");

# derived paramenters
n_freq := Int((N+3)/2);

# problem definition
sym := var.fresh_t("S", TArray(TReal, 2*n_freq));
t := IOPrunedRConv(N, sym, 1, [minout..N-1], 1, [0..maxin], true);

# generate code and autotune
rt := DP(t, opts)[1].ruletree;
c := CodeRuleTree(rt, opts);

# create files
PrintTo(name::".c", PrintCode(name, c, opts));
```

Back end

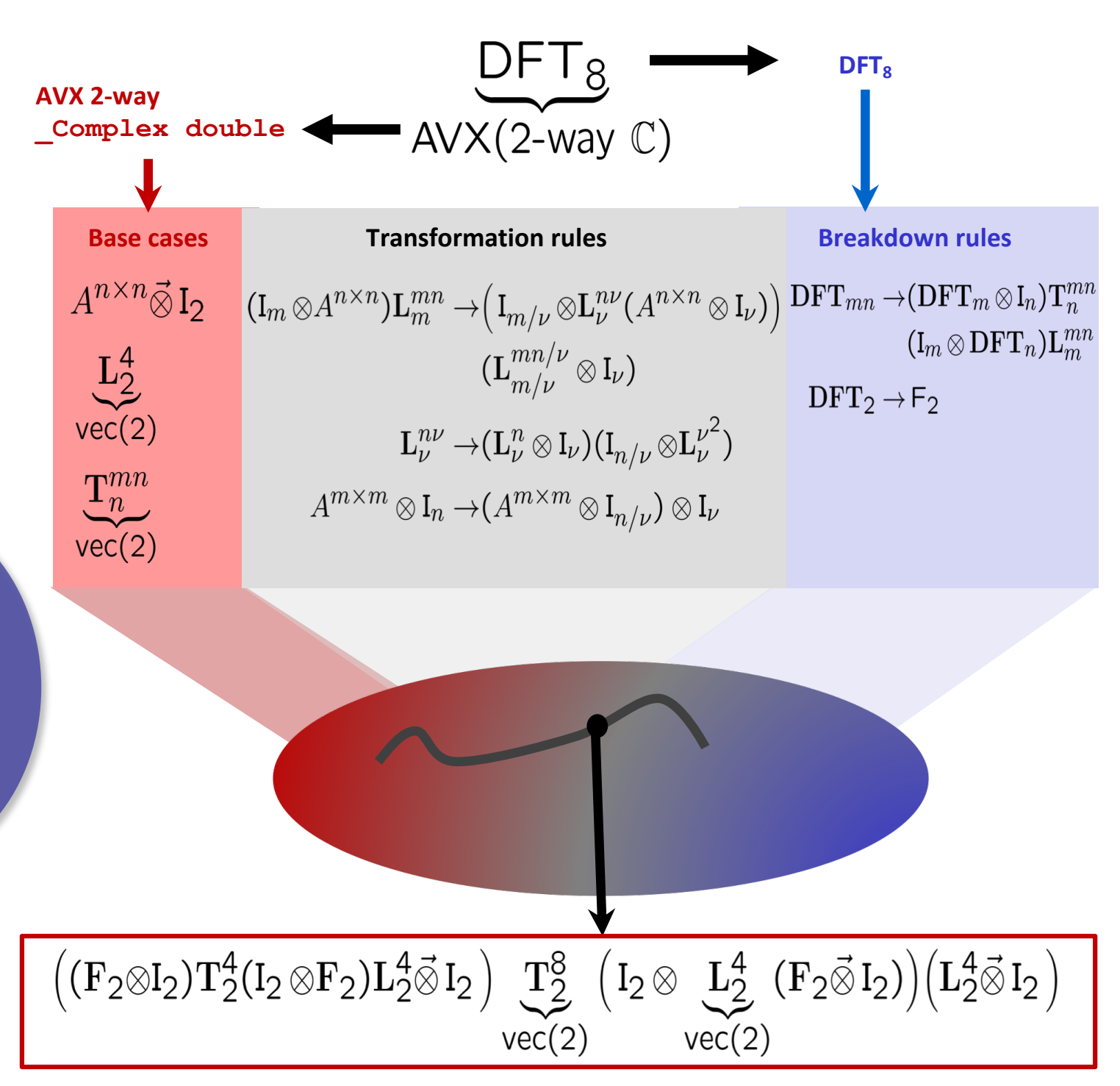
FFTX backend: SPIRAL


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
DARPA BRASS

Platform-aware formal program synthesis

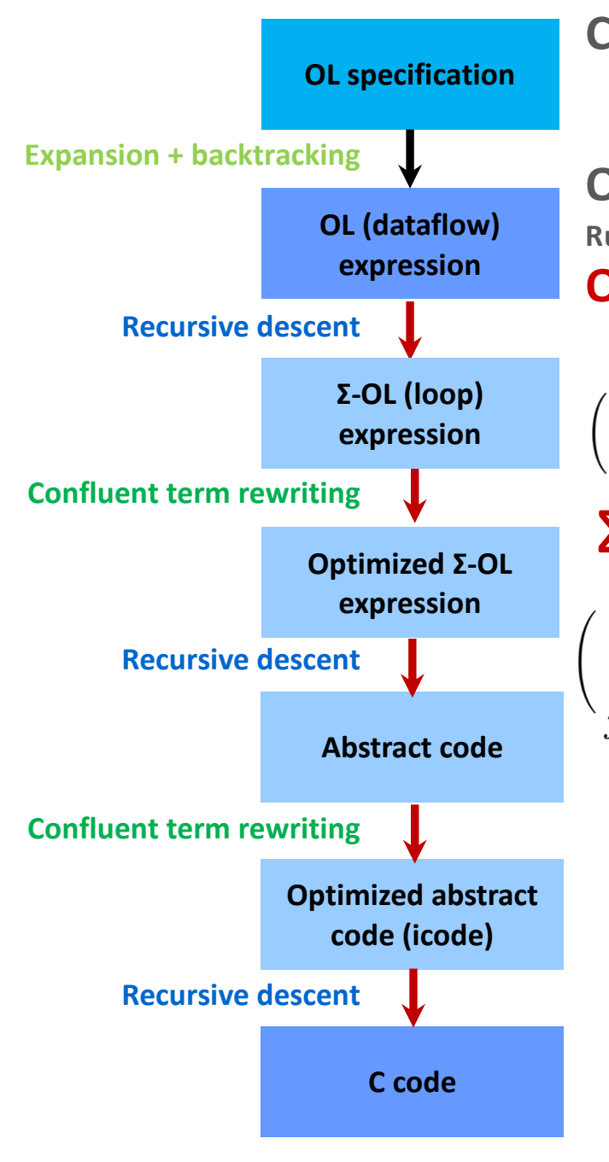

Model: common abstraction
= spaces of matching formulas
abstraction
abstraction
ν
p
μ
defines
$A_n \otimes I_\nu$
vec(ν)
$I_p \otimes A_n$
smp(p,μ)
rewriting
pick
architecture space
search
algorithm space
$(DFT_2 \otimes I_4)\, T^8_4\, (I_2 \otimes (\dots\dots))\, L^8_2$
Architectural parameters: vector length, #processors, …
optimization
Kernel: problem size, algorithm choice

Autotuning in constraint solution space


AVX 2-way _Complex double
DFT_8
$\mathrm{AVX}(2\text{-way } \mathbb{C})$
DFT8
Base cases
Transformation rules
Breakdown rules
vec(2)
vec(2)
$\mathrm{DFT}_2 \to \mathrm{F}_2$
$\left((\mathrm{F}_2\otimes\mathrm{I}_2)\mathrm{T}^4_2(\mathrm{I}_2\otimes\mathrm{F}_2)\mathrm{L}^4_2\vec{\otimes}\mathrm{I}_2\right)\ \mathrm{T}^8_2\ \left(\mathrm{I}_2\otimes\ \mathrm{L}^4_2\ (\mathrm{F}_2\vec{\otimes}\mathrm{I}_2)\right)\left(\mathrm{L}^4_2\vec{\otimes}\mathrm{I}_2\right)$

OL expression

Translating an OL expression into code



Constraint Solver Input:

$$\underbrace{\mathrm{DFT}_8}_{\mathrm{AVX}(2\text{-way } \mathbb{C})}$$

Output =
Ruletree, expanded into
OL expression:

$$\left((\mathrm{F}_2\otimes\mathrm{I}_2)\mathrm{T}^4_2(\mathrm{I}_2\otimes\mathrm{F}_2)\mathrm{L}^4_2\vec{\otimes}\mathrm{I}_2\right)\ \underbrace{\mathrm{T}^8_2}_{\mathrm{vec}(2)}\ \left(\mathrm{I}_2\otimes\ \underbrace{\mathrm{L}^4_2}_{\mathrm{vec}(2)}\ (\mathrm{F}_2\vec{\otimes}\mathrm{I}_2)\right)\left(\mathrm{L}^4_2\vec{\otimes}\mathrm{I}_2\right)$$

Σ-OL:

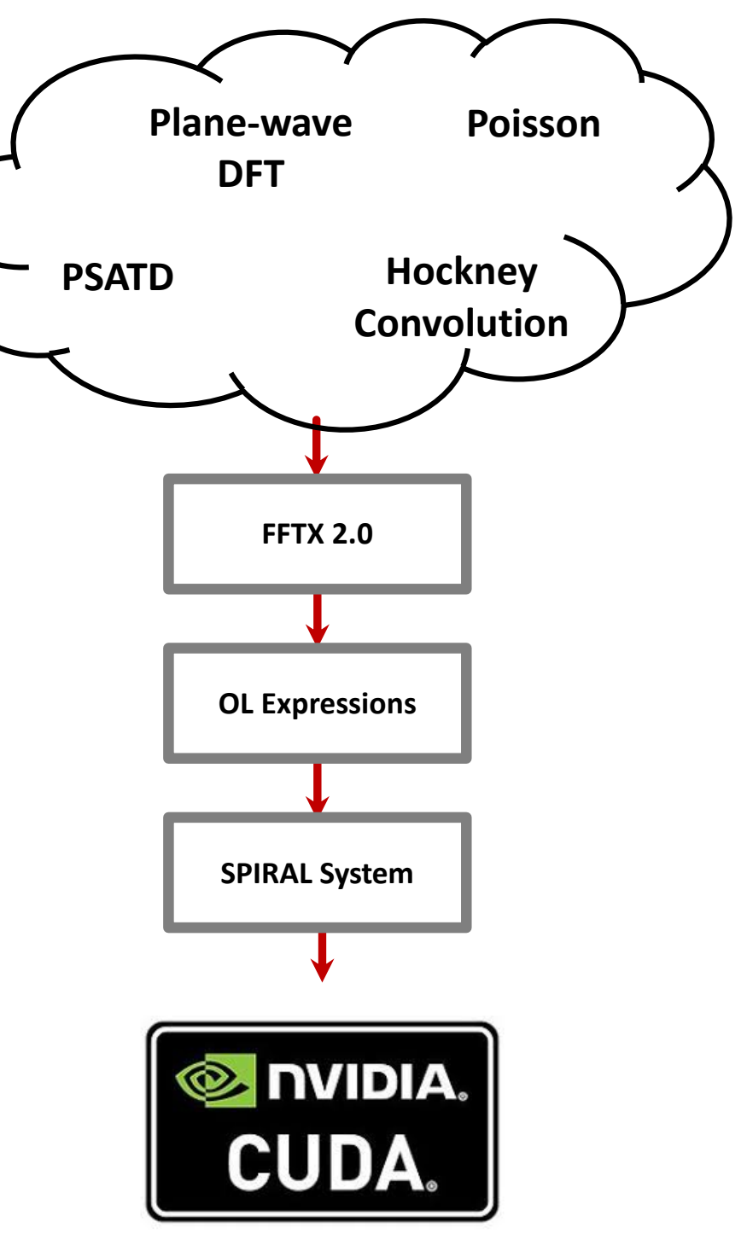
$$\left(\sum_{j=0}^{1}\left(\mathrm{S}_{\imath_2\otimes(j)_2}\mathrm{F}_2\mathrm{Map}^2_{x\mapsto\omega_4^{2i+j}x}\mathrm{G}_{\imath_2\otimes(j)_2}\right)\sum_{j=0}^{1}\left(\mathrm{S}_{(j)_2\otimes\imath_2}\mathrm{F}_2\mathrm{G}_{\imath_2\otimes(j)_2}\right)\right)\vec{\otimes}\mathrm{I}_2\dots$$

C code:

```
void dft8(_Complex double *Y, _Complex double *X) {
    __m256d s38, s39, s40, s41,...
    __m256d  *a17, *a18;
    a17 = ((__m256d  *) X);
    s38 = *(a17);
    s39 = *((a17 + 2));
    t38 = _mm256_add_pd(s38, s39);
    t39 = _mm256_sub_pd(s38, s39);
    ...
    s52 = _mm256_sub_pd(s45, s50);
    *((a18 + 3)) = s52;
}
```

Technology + Results

Performance Results with cuFFT backend



Poisson: scalar periodic convolution, 128^3 inputs / outputs, 200 Gflops / sec

PSATD: vector-valued convolution, 11 x 100^3 inputs , 6 x 64^3 outputs, 100^3 transform size, 100 Gflops / sec

Plane-wave: scalar transform, 128^3 inputs, 256^3 outputs / transform size, 1000 Gflops / sec

Hockney: scalar convolution, 33^3 inputs, 97^3 outputs, 130^3 transform size, 260 Gflops / sec

Algorithms: rules in domain-specific language

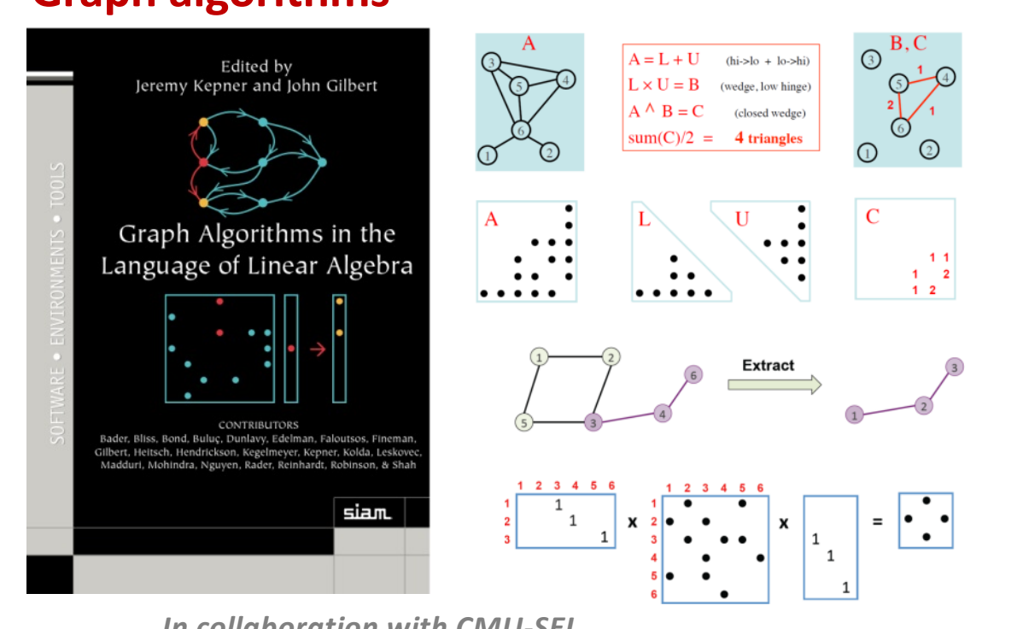
Linear transforms

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

Numerical linear algebra

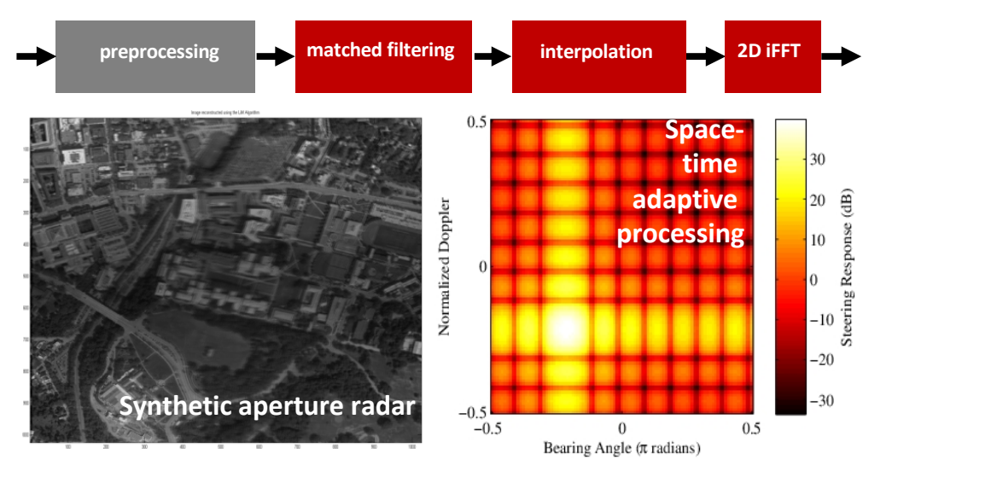
$$\begin{aligned}
\mathrm{MMM}_{1,1,1} &\to (\cdot)_1\\
\mathrm{MMM}_{m,n,k} &\to (\otimes)_{m/m_b\times 1}\otimes\mathrm{MMM}_{m_b,n,k}\\
\mathrm{MMM}_{m,n,k} &\to \mathrm{MMM}_{m,nb,k}\otimes(\otimes)_{1\times n/nb}\\
\mathrm{MMM}_{m,n,k} &\to ((\Sigma_{k/k_b}\circ(\cdot)_{k/k_b})\otimes\mathrm{MMM}_{m,n,k_b})\circ\\
&\qquad((L^{mk/k_b}_{k/k_b}\otimes I_{k_b})\times I_{kn})\\
\mathrm{MMM}_{m,n,k} &\to (L^{mn/n_b}_m\otimes I_{n_b})\circ\\
&\qquad((\otimes)_{1\times n/n_b}\otimes\mathrm{MMM}_{m,n_b,k})\circ\\
&\qquad(I_{km}\times(L^{kn/n_b}_{n/n_b}\otimes I_{n_b}))
\end{aligned}$$

Graph algorithms


Edited by Jeremy Kepner and John Gilbert
Graph Algorithms in the Language of Linear Algebra
siam
A
A = L + U
L × U = B
A ^ B = C
sum(C)/2 = 4 triangles
B, C
L
U
C
Extract

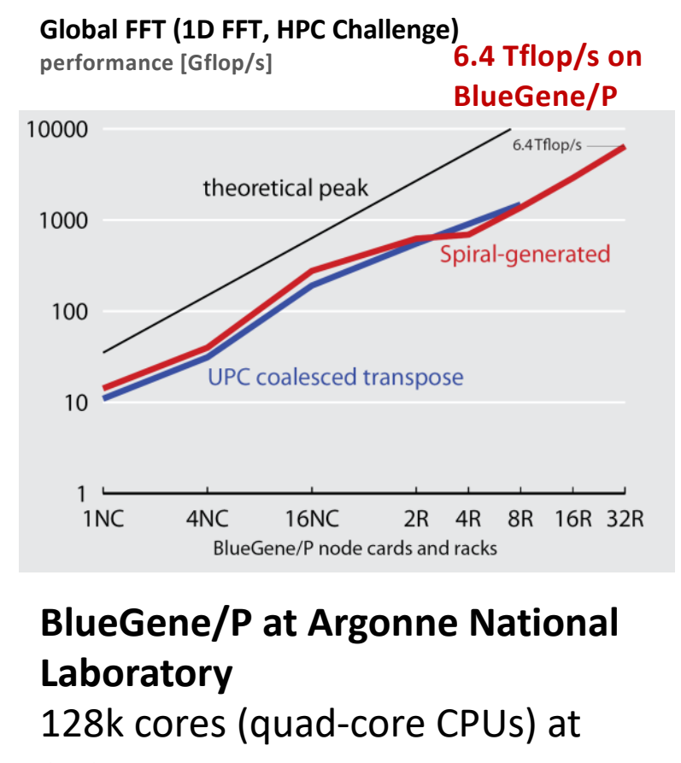
In collaboration with CMU-SEI

Spectral domain applications

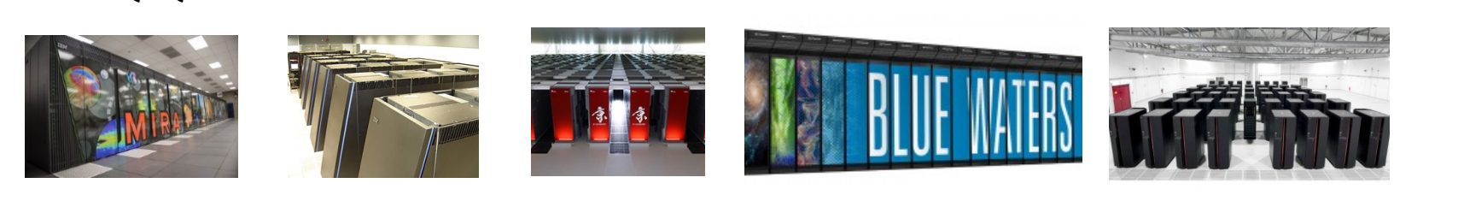


SPIRAL: success in HPC/supercomputing

- **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



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

SPIRAL 8.1.1: 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**

Spiral
Encyclopedia of Parallel Computing
Proceedings IEEE
From High-Level Specification to High-Performance Code
PROCEEDINGS IEEE
PROGRAM GENERATION, OPTIMIZATION, AND PLATFORM ADAPTATION

```
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
```

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*

www.spiral.net

Carnegie Mellon
SpiralGen Accelerating Innovation in Computing
BERKELEY LAB Lawrence Berkeley National Laboratory
ECP EXASCALE COMPUTING PROJECT