FFTX SPIRAL Backend

Franz Franchetti
Carnegie Mellon University

in collaboration with
Daniele G. Spampinato, Anuva Kulkarni, Tze Meng Low
Carnegie Mellon University

Doru Thom Popovici, Andrew Canning, Peter McCorquodale, Brian Van Straalen, Phillip Colella
Lawrence Berkeley National Laboratory

Mike Franusich
SpiralGen, Inc.

This work was supported by DOE ECP project 2.3.3.07

ECP 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

FFTX and SpectralPACK

Numerical Linear Algebra		Spectral Algorithms
LAPACK LU factorization Eigensolves SVD …	≈	**SpectralPACK** Convolution Correlation Upsampling Poisson solver …
BLAS BLAS-1 BLAS-2 BLAS-3		**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,…
- **Library front-end, code generation and vendor library back-end**
  mirror concepts from FFTX layer

FFTX and SpectralPACK solve the "spectral motif" long term

Example: Poisson's Equation in Free Space

Partial differential equation (PDE)

$$\Delta(\Phi) = \rho$$

$$\rho : \mathbb{R}^3 \to \mathbb{R}$$

$$D = \mathrm{supp}(\rho) \subset \mathbb{R}^3$$

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: $\phi(.)$ = convolution of RHS $\rho(.)$ with Green's function $G(.)$. Efficient through FFTs (frequency domain)

Method of Local Corrections (MLC)

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

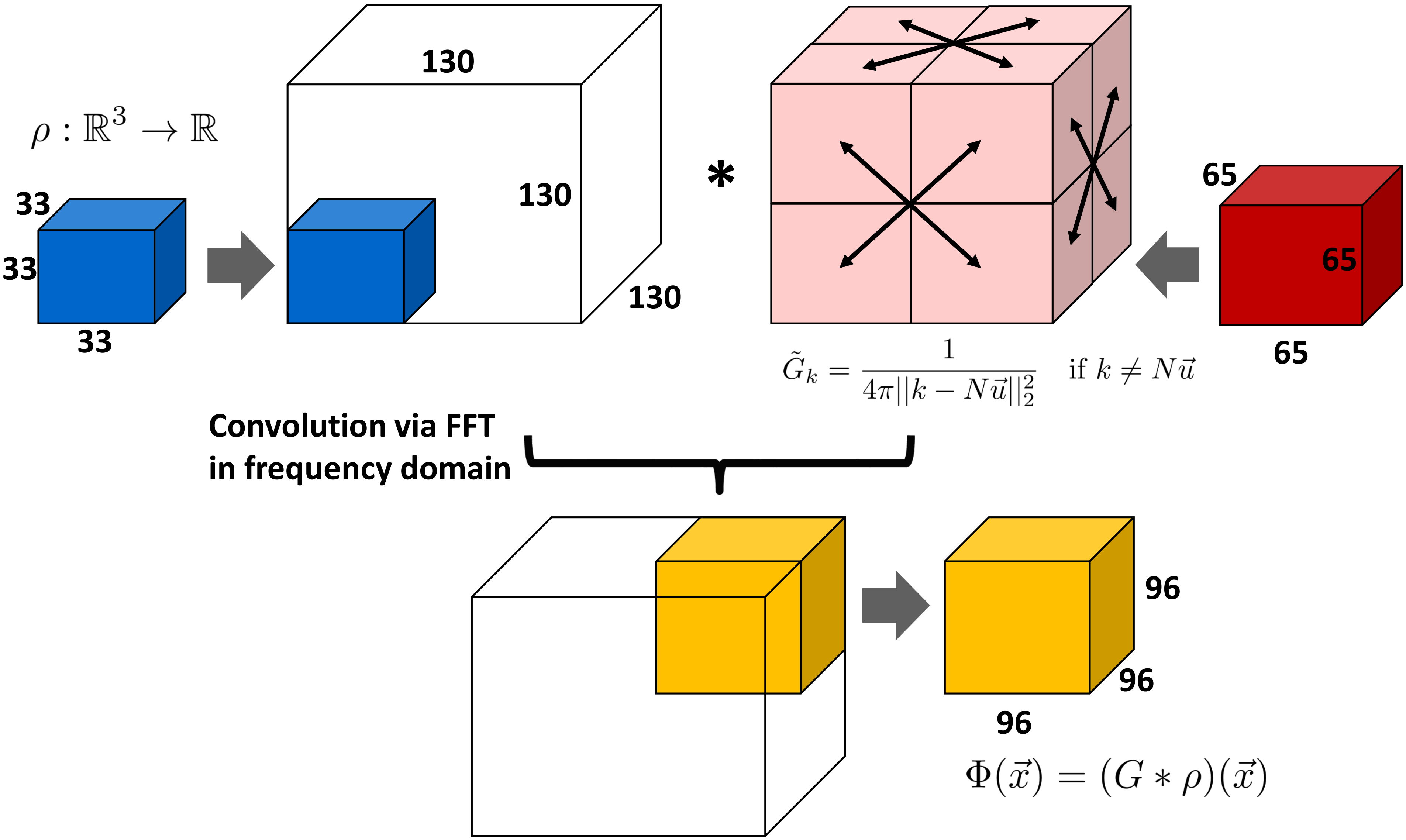
Green's function kernel in frequency domain

P. McCorquodale, P. Colella, G. T. Balls, and S. B. Baden: **A Local Corrections Algorithm for Solving Poisson's Equation in Three Dimensions.** Communications in Applied Mathematics and Computational Science Vol. 2, No. 1 (2007), pp. 57-81., 2007.

C. R. Anderson: **A method of local corrections for computing the velocity field due to a distribution of vortex blobs.** Journal of Computational Physics, vol. 62, no. 1, pp. 111–123, 1986.

Algorithm: Hockney Free Space Convolution

$\rho : \mathbb{R}^3 \to \mathbb{R}$



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

$$\Phi(\vec{x}) = (G * \rho)(\vec{x})$$

Hockney: Convolution + problem specific zero padding and output subset

FFTX C++ Code: Hockney Free Space Convolution

```
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);
```

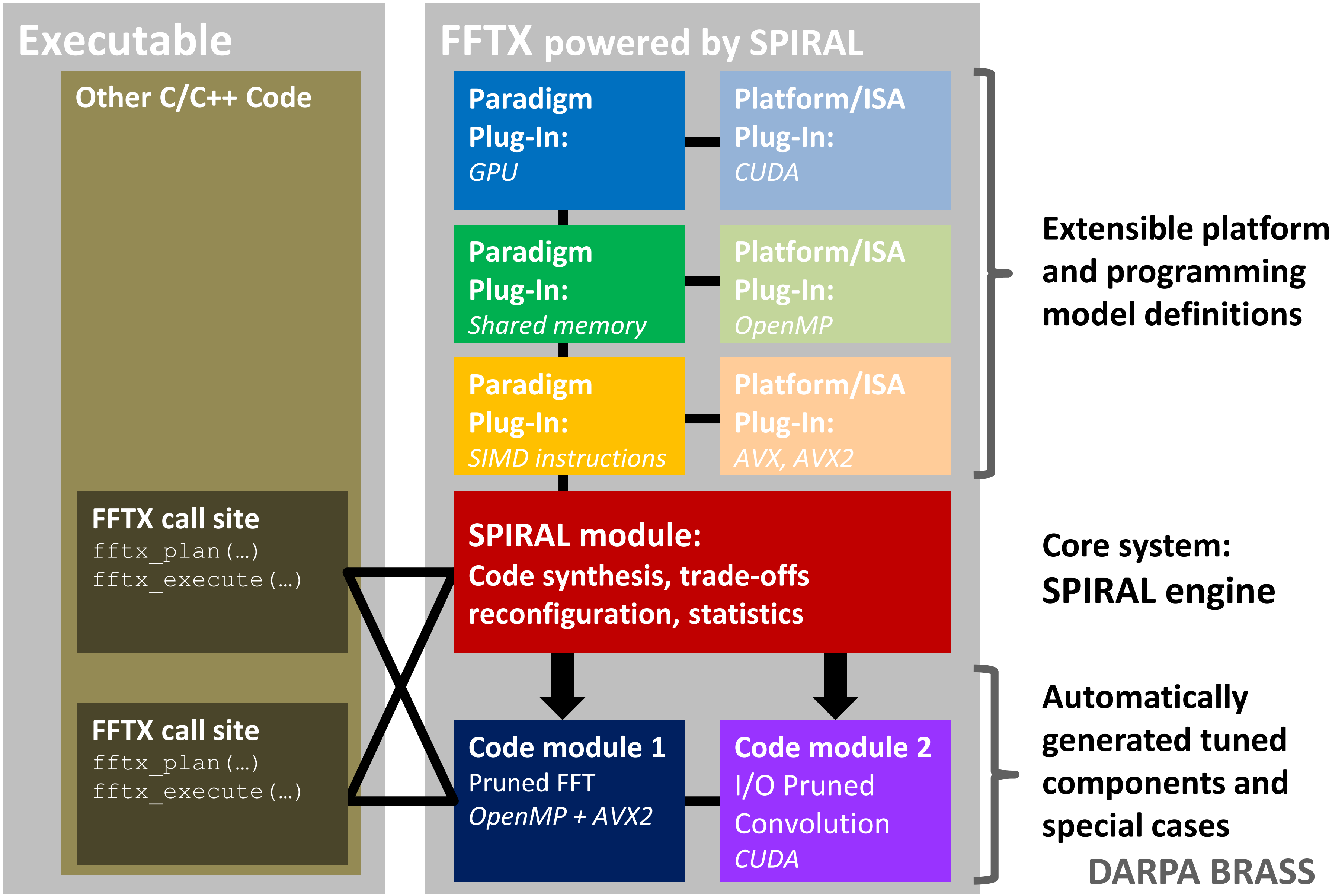
FFTX C++ Code: Hockney Free Space Convolution

```
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

auto fo
    pla

auto ke                                                                   lize);

point_t
auto in


auto so


context
context

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);
```

This is a specification dressed as a program

- Needs to be clean and concise
- No code level optimizations and tricks
- Don't think "performance" but "correctness"
- *For production code and software engineering*

FFTX Backend: SPIRAL



SPIRAL 8.2.0: 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, continuing under DOE ECP**
- **Tutorial material available online**

www.spiral.net

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*

FFTX Program Trace

```
Load(fftx);
ImportAll(fftx);

conf := FFTXGlobals.confHockneyMlcCUDADevice();
opts := FFTXGlobals.getOpts(conf);

t := let(
    n := 130, ns := 33, nd := 96,
    name := "hockney"::StringInt(n)::"_"::StringInt(nd)::"_"::StringInt(ns),
    symvar := var("symbl", TPtr(TReal)),
    TFCall(
        Compose([
            ExtractBox([n,n,n], [[n-nd..n-1],[n-nd..n-1],[n-nd..n-1]]),
            IMDPRDFT([n,n,n], 1),
            RCDiag(FDataOfs(symvar, 2*n*n*(n/2+1), 0)),
            MDPRDFT([n,n,n], -1),
            ZeroEmbedBox([n,n,n], [[0..ns-1],[0..ns-1],[0..ns-1]])]),
        rec(fname := name, params := [symvar])
    ).withTags(opts.tags)
);

c := opts.fftxGen(t);
opts.prettyPrint(c);
```

FFTX Program Trace

```
Load(fftx);
ImportAll(fftx);

conf := FFTXGlobals.confHockneyMlcCUDADevice();
op

t
        fcc(fname := name, params := [symvar])
    ).withTags(opts.tags)
);

c := opts.fftxGen(t);
opts.prettyPrint(c);
```

The whole convolution kernel is captured

- DAG with all dependencies
- User-defined call-backs
- Captures prunining, zero-padding and symmetries
- *Lifts sequence of C/C++ library calls to a specification*

SPIRAL Options Capture Performance Engineering

```
hockneyMlcCUDADeviceOpts := function(arg)
    local opts;
    opts := Copy(HockneyMlcCUDADeviceOpts);
    opts.includes := [];
    opts.breakdownRules.PRDFT := [ PRDFT1_Base2, CopyFields(PRDFT_PD, rec(maxSize := 7)), PRDFT_PD_loop];
    opts.breakdownRules.IPRDFT := [ IPRDFT1_Base1, IPRDFT1_Base2, IPRDFT_PD_loop, IPRDFT_PD ];
    opts.breakdownRules.IPRDFT2 := List([ IPRDFT2_Base1, IPRDFT2_Base2, IPRDFT2_CT], _noT);
    opts.breakdownRules.PRDFT3 := [ ];
    opts.breakdownRules.URDFT := List([ URDFT1_Base1, URDFT1_Base2, URDFT1_Base4, URDFT1_CT ], _noT);
    opts.breakdownRules.DFT := List([ DFT_Base, CopyFields(DFT_PD, rec(maxSize := 7)),  DFT_PD_loop], _noT);
    opts.breakdownRules.PrunedPRDFT := [ PrunedPRDFT_base, PrunedPRDFT_CT_rec_block ];
    opts.breakdownRules.PrunedIPRDFT := [ CopyFields(PrunedIPRDFT_base, PrunedIPRDFT_CT_rec_block ];
    opts.breakdownRules.PrunedDFT := List([ PrunedDFT_base, PrunedDFT_CT_rec_block ], _noT);
    opts.breakdownRules.IOPrunedMDRConv := List([ IOPrunedMDRConv_3D_2trip_zyx_freqdata ], _noT);
    opts.breakdownRules.MDRConv := [ MDRConv_3D_2trip_zyx_freqdata ];

    opts.formulaStrategies.postProcess := opts.formulaStrategies.postProcess
        :: opts.formulaStrategies.sigmaSpl :: [MergedRuleSet(HockneyMLC_SIMTRules, RulesSums)];


    opts.preProcess := (self, t) >> let(t1 := RulesFFTXPromoteNT(Copy(t)), RulesFFTXPromoteNT_Cleanup(t1));

    return opts;
end;
```

SPIRAL Options Capture Performance Engineering

```
hockneyMlcCUDADeviceOpts := function(arg)
    local opts;
    opts := Copy(HockneyMlcCUDADeviceOpts);
    opts.includes := [];
    opts.breakdownRules.PRDFT := [ PRDFT1_Base2, CopyFields(PRDFT_PD, rec(maxSize := 7)), PRDFT_PD_loop];
    opts.breakdownRules.IPRDFT := [ IPRDFT1_Base1, IPRDFT1_Base2, IPRDFT_PD_loop, IPRDFT_PD ];
    opts.breakdownRules.IPRDFT2 := List([ IPRDFT2_Base1, IPRDFT2_Base2, IPRDFT2_CT], _noT);
```

Configures code generation

- Developed by performance engineer + application specialist
- Lifts FFTX call sequence into SPIRAL non-terminal
- Configures all SPIRAL rewriting systems
- Does code generation and autotuning
- *Clear separation of concerns frontend/backend*

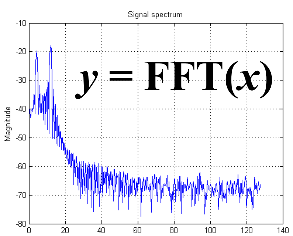
SPIRAL: Go from Mathematics to Software

Given:

- **Mathematical problem specification**
  core mathematics does not change
- **Target computer platform**
  varies greatly, new platforms introduced often

Wanted:

- **Very good implementation of specification on platform**
- **Proof of correctness**



on

automatic

```
void fft64(double  *Y, double  *X) {
    ...
    s5674 = _mm256_permute2f128_pd(s5672, s5673, (0) | ((2) << 4));
    s5675 = _mm256_permute2f128_pd(s5672, s5673, (1) | ((3) << 4));
    s5676 = _mm256_unpacklo_pd(s5674, s5675);
    s5677 = _mm256_unpackhi_pd(s5674, s5675);
    s5678 = *((a3738 + 16));
    s5679 = *((a3738 + 17));
    s5680 = _mm256_permute2f128_pd(s5678, s5679, (0) | ((2) << 4));
    s5681 = _mm256_permute2f128_pd(s5678, s5679, (1) | ((3) << 4));
    s5682 = _mm256_unpacklo_pd(s5680, s5681);
    s5683 = _mm256_unpackhi_pd(s5680, s5681);
    t5735 = _mm256_add_pd(s5676, s5682);
    t5736 = _mm256_add_pd(s5677, s5683);
    t5737 = _mm256_add_pd(s5670, t5735);
    t5738 = _mm256_add_pd(s5671, t5736);
    t5739 = _mm256_sub_pd(s5670, _mm256_mul_pd(_mm_vbroadcast_sd(&(C22)), t5735));
    t5740 = _mm256_sub_pd(s5671, _mm256_mul_pd(_mm_vbroadcast_sd(&(C22)), t5736));
    t5741 = _mm256_mul_pd(_mm_vbroadcast_sd(&(C23)), _mm256_sub_pd(s5677, s5683));
    t5742 = _mm256_mul_pd(_mm_vbroadcast_sd(&(C23)), _mm256_sub_pd(s5676, s5682));
    ...
}
```

performance

QED.

Inspiration: Symbolic Integration

- **Rule based AI system**
  basic functions, substitution
- **May not succeed**
  not all expressions can be symbolically integrated
- **Arbitrarily extensible**
  define new functions as integrals
  Γ(.), distributions, Lebesgue integral
- **Semantics preserving**
  rule chain = formal proof
- **Automation**
  Mathematica, Maple

Table of Integrals

BASIC FORMS

(1) $\int x^n dx = \frac{1}{n+1} x^{n+1}$

(2) $\int \frac{1}{x} dx = \ln x$

(3) $\int u dv = uv - \int v du$

(4) $\int u(x) v'(x) dx = u(x)v(x) - \int v(x) u'(x) dx$

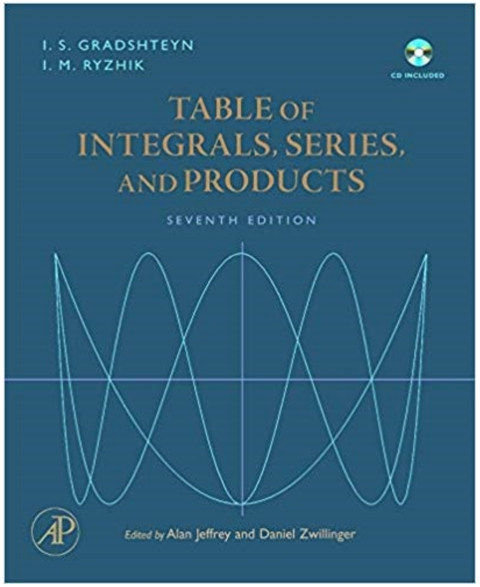
RATIONAL FUNCTIONS

(5) $\int \frac{1}{ax+b} dx = \frac{1}{a} \ln(ax+b)$

(6) $\int \frac{1}{(x+a)^2} dx = \frac{-1}{x+a}$

(7) $\int (x+a)^n dx = (x+a)^n \left(\frac{a}{1+n} + \frac{x}{1+n} \right), \; n \neq -1$

(8) $\int x(x+a)^n dx = \frac{(x+a)^{1+n}(nx+x-a)}{(n+2)(n+1)}$



In[31]:= $\int_0^{2\pi} \frac{1}{a^2 \mathrm{Cos}[t]^2 + b^2 \mathrm{Sin}[t]^2} \, dt$

Out[31]= $\frac{2\sqrt{\frac{b^2}{a^2}}\,\pi}{b^2}$

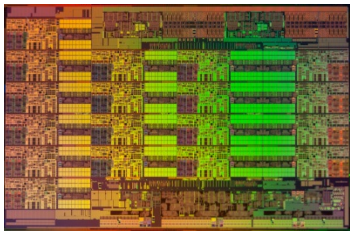
Wolfram Mathematica

In[33]:= $\int_0^{2\pi} \frac{1}{a^2 \left(\frac{e^{i t} + e^{-i t}}{2}\right)^2 + b^2 \left(\frac{e^{i t} - e^{-i t}}{2 i}\right)^2} \, dt$

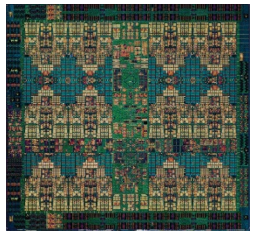
Out[33]= 0

SPIRAL’s Target Computing Landscape

1 Gflop/s = one billion floating-point operations (additions or multiplications) per second

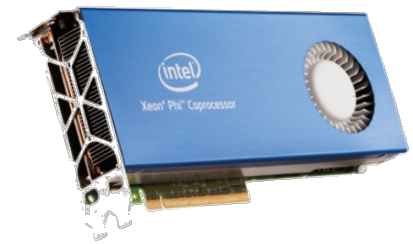


Intel Xeon 8180M
2.25 Tflop/s, 205 W
28 cores, 2.5—3.8 GHz
2-way—16-way AVX-512



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

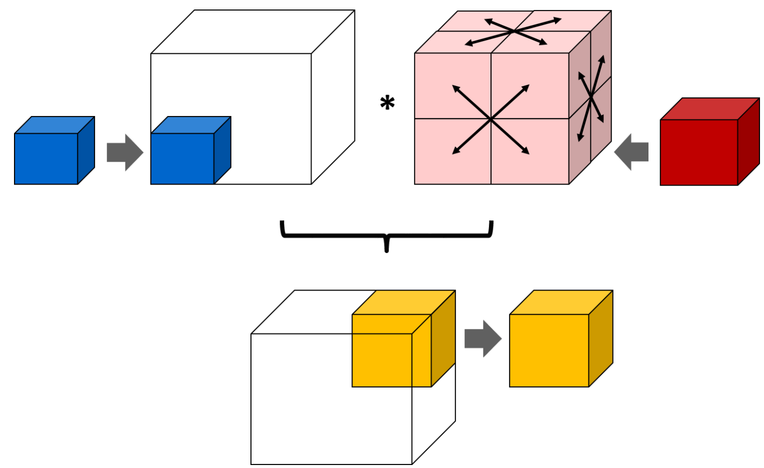
Summit
187.7 Pflop/s, 13 MW
9,216 x 22 cores POWER9
+ 27,648 V100 GPUs

Rules in Internal Domain Specific Language

Linear Transforms

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

Spectral Domain Algorithms



Hardware

- Multithreading (Multicore) — $\mathrm{I}_p \otimes_{\parallel} A_{\mu n}, \quad \mathrm{L}^{mn}_m \bar{\otimes} \mathrm{I}_\mu$
- Vector SIMD (SSE, VMX/Altivec,...) — $A \hat{\otimes} \mathrm{I}_\nu, \quad \underbrace{\mathrm{L}^{2\nu}_2}_{\text{isa}}, \quad \underbrace{\mathrm{L}^{2\nu}_\nu}_{\text{isa}}, \quad \underbrace{\mathrm{L}^{\nu^2}_\nu}_{\text{isa}}$
- Message Passing (Clusters, MPP) — $\mathrm{I}_p \otimes_{\parallel} A_n, \quad \underbrace{\mathrm{L}^{p^2}_p \bar{\otimes} \mathrm{I}_{n/p^2}}_{\text{all-to-all}}$
- Streaming/multibuffering (Cell) — $\mathrm{I}_n \otimes_2 A_{\mu n}, \quad \mathrm{L}^{mn}_m \bar{\otimes} \mathrm{I}_\mu$
- Graphics Processors (GPUs) — $\prod_{i=0}^{n-1} A_i, \quad A_n \bar{\otimes} \mathrm{I}_w, \quad P_n \otimes Q_w$
- Gate-level parallelism (FPGA) — $\prod_{i=0}^{n-1}{}^{\mathrm{ir}} A, \quad \mathrm{I}_s \tilde{\otimes} A, \quad \underbrace{\mathrm{L}^m_n}_{\text{bram}}$
- HW/SW partitioning (CPU + FPGA) — $\underbrace{A_1}_{\text{fpga}}, \quad \underbrace{A_2}_{\text{fpga}}, \quad \underbrace{A_3}_{\text{fpga}}, \quad \underbrace{A_4}_{\text{fpga}}$

Program Transformations

$$
\begin{aligned}
\underbrace{AB}_{\mathrm{smp}(p,\mu)} &\to \underbrace{A}_{\mathrm{smp}(p,\mu)}\ \underbrace{B}_{\mathrm{smp}(p,\mu)}\\
\underbrace{A_m \otimes \mathrm{I}_n}_{\mathrm{smp}(p,\mu)} &\to \underbrace{\left(\mathrm{L}^{mp}_m \otimes \mathrm{I}_{n/p}\right)\left(\mathrm{I}_p \otimes (A_m \otimes \mathrm{I}_{n/p})\right)\left(\mathrm{L}^{mp}_p \otimes \mathrm{I}_{n/p}\right)}_{\mathrm{smp}(p,\mu)}\\
\underbrace{\mathrm{L}^{mn}_m}_{\mathrm{smp}(p,\mu)} &\to \begin{cases} \underbrace{\left(\mathrm{I}_p \otimes \mathrm{L}^{mn/p}_{m/p}\right)}_{\mathrm{smp}(p,\mu)} \underbrace{\left(\mathrm{L}^{pn}_p \otimes \mathrm{I}_{m/p}\right)}_{\mathrm{smp}(p,\mu)} \\ \underbrace{\left(\mathrm{L}^{pm}_m \otimes \mathrm{I}_{n/p}\right)}_{\mathrm{smp}(p,\mu)} \underbrace{\left(\mathrm{I}_p \otimes \mathrm{L}^{mn/p}_m\right)}_{\mathrm{smp}(p,\mu)} \end{cases}
\end{aligned}
$$

Recursive rules

$$
\begin{aligned}
\underbrace{\mathrm{I}_m \otimes A_n}_{\mathrm{smp}(p,\mu)} &\to \mathrm{I}_p \otimes_{\parallel} \left(\mathrm{I}_{m/p} \otimes A_n\right)\\
\underbrace{(P \otimes \mathrm{I}_n)}_{\mathrm{smp}(p,\mu)} &\to \left(P \otimes \mathrm{I}_{n/\mu}\right) \bar{\otimes} \mathrm{I}_\mu
\end{aligned}
$$

Base case rules

Autotuning in Constraint Solution Space

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

AVX 2-way
`_Complex double`

DFT_8

Base cases

$A^{n\times n}\vec{\otimes} \mathrm{I}_2$

$\underbrace{\mathrm{L}_2^4}_{\mathrm{vec}(2)}$

$\underbrace{\mathrm{T}_n^{mn}}_{\mathrm{vec}(2)}$

Transformation rules

$(\mathrm{I}_m \otimes A^{n\times n})\mathrm{L}_m^{mn} \to \left(\mathrm{I}_{m/\nu} \otimes \mathrm{L}_\nu^{n\nu}(A^{n\times n}\otimes \mathrm{I}_\nu)\right)(\mathrm{L}_{m/\nu}^{mn/\nu}\otimes \mathrm{I}_\nu)$

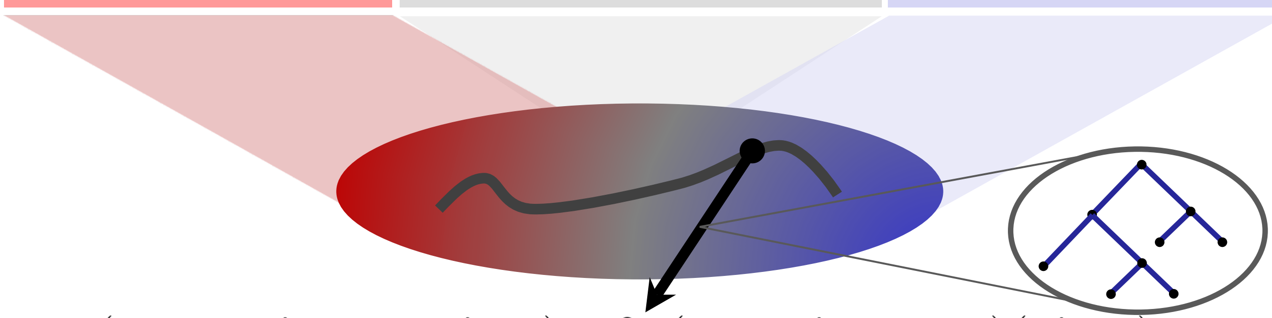
$\mathrm{L}_\nu^{n\nu} \to (\mathrm{L}_\nu^n \otimes \mathrm{I}_\nu)(\mathrm{I}_{n/\nu}\otimes \mathrm{L}_\nu^{\nu^2})$

$A^{m\times m}\otimes \mathrm{I}_n \to (A^{m\times m}\otimes \mathrm{I}_{n/\nu})\otimes \mathrm{I}_\nu$

Breakdown rules

$\mathrm{DFT}_{mn} \to (\mathrm{DFT}_m\otimes \mathrm{I}_n)\mathrm{T}_n^{mn}(\mathrm{I}_m\otimes \mathrm{DFT}_n)\mathrm{L}_m^{mn}$

$\mathrm{DFT}_2 \to \mathrm{F}_2$



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

FFTX C/C++ specification code

↓ Inspector/executor

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 module

Nonterminal Expression (tSPL)

```
conf := FFTXGlobals.confHockneyMlcCUDADevice();
opts := FFTXGlobals.getOpts(conf);

t := let(
    n := 130, ns := 33, nd := 96,
    name := "hockney"::StringInt(n)::"_"::StringInt(nd)::"_"::StringInt(ns),
    symvar := var("symbl", TPtr(TReal)),
    TFCall(
        Compose([
            ExtractBox([n,n,n], [[n-nd..n-1],[n-nd..n-1],[n-nd..n-1]]),
            IMDPRDFT([n,n,n], 1),
            RCDiag(FDataOfs(symvar, 2*n*n*(n/2+1), 0)),
            MDPRDFT([n,n,n], -1),
            ZeroEmbedBox([n,n,n], [[0..ns-1],[0..ns-1],[0..ns-1]])]),
        rec(fname := name, params := [symvar])
    ).withTags(opts.tags)
);

## API definitions
_tofAdd := (n, l) -> When(IsList(l), let(mn := Minimum(l), mx := Maximum(l),
Checked(ForAll([mn..mx], i->i in l), fAdd(n, mx-mn+1, mn))),
                          When(n <> 1, fAdd(n, 1, 0), fId(n)));
_toBox := (ns, nps)-> When(IsList(ns), fTensor(List(Zip2(ns, nps), i->ApplyFunc(_tofAdd,
i))), _tofAdd(ns, nps));

ZeroEmbedBox := (ns, nps) -> Scat(_toBox(ns, nps));
ExtractBox := (ns, nps) -> Gath(_toBox(ns, nps));
```

DAG Nonterminal and Fortran Array Layout

```
Xcmj := tcast(TPtr(TColMaj(BoxND(szns, TComplex))), X);
Ycmj := tcast(TPtr(TColMaj(BoxND(szns, TComplex))), Y);

t := let(ns := szns,
    k := -1,
    name := "dft"::StringInt(Length(ns))::"d_cmj",
    TFCall(
        TDAG([
            TDAGNode(TRC(MDDFT(ns, k)), Ycmj, Xcmj)
        ]),
        rec(fname := name, params := [])).withTags(opts.tags)
);
```

Pruned Convolution Promotion Rules

```
Hockney_promote := ARule(Compose,
    [[@(6,Gath), @(8,fTensor, e->ForAll(e.children(), i->ObjId(i)=fAdd))],
     @(1,IMDPRDFT, e -> e.params[2] = 1), [@(2,RCDiag), @(4, FDataOfs, e->e.ofs = 0), @(5,I)],
     @(3,MDPRDFT, e -> e.params[2] = Product(e.params[1])-1),
     [@(7,Scat), @(9,fTensor, e->ForAll(e.children(), i->ObjId(i)=fAdd))]],
    e-> let(ii := Ind(Rows(@(2).val)), sym := @(2).val.element.var,
            symf := Lambda(ii, nth(sym,ii)),
            opat := List(@(8).val.children(), i-> _toSymList(List(i.tolist(), _unwrap))),
            ipat := List(@(9).val.children(), i-> _toSymList(List(i.tolist(), _unwrap))),
        [ IOPrunedMDRConv(@(1).val.params[1], symf, 1, opat, 1, ipat, true) ]))

Scat_Circulant_Gath__IOPrunedRConv := ARule(Compose,
    [[@(1, Gath), fAdd],  @(2,Circulant), [@(3, Scat), fAdd]],
    e-> [ IOPrunedRConv(@(2).val.params[1],
            FDataOfs(@(2).val.params[2].var, 2*(@(2).val.params[1]/2+1), 0),
            1, _toSymList(List(@(1).val.func.tolist(), _unwrap)),
            1, _toSymList(List(@(3).val.func.tolist(), _unwrap)), true)]),

MDDFT_Scat__PrunedMDDFT := ARule(Compose,
    [@(1, MDDFT), [@(2, Scat), @(3,fTensor, e->ForAll(e.children(), i->ObjId(i)=fAdd))]],
    e -> [ PrunedMDDFT(@(1).val.params[1], @(1).val.params[2], 1,
        List(@(3).val.children(), i-> _toSymList(List(i.tolist(), _unwrap))))] ),

Gath_MDDFT__PrunedIMDDFT := ARule(Compose,
    [[@(1, Gath), @(2,fTensor, e->ForAll(e.children(), i->ObjId(i)=fAdd))], @(3, MDDFT)],
    e -> [PrunedIMDDFT(@(3).val.params[1], @(3).val.params[2], 1,
        List(@(2).val.children(), i-> _toSymList(List(i.tolist(), _unwrap)))]),
```

Promoted Nonterminal

```
...
spiral> t;
TFCall(Gath(fTensor(fAdd(130, 96, 34), fAdd(130, 96, 34), fAdd(130, 96, 34))) *
  IMDPRDFT([ 130, 130, 130 ], 1) *
  RCDiag(FDataOfs(symbl, 2230800, V(0)), I(2)) *
  MDPRDFT([ 130, 130, 130 ], 2196999) *
  Scat(fTensor(fAdd(130, 33, 0), fAdd(130, 33, 0), fAdd(130, 33, 0))), rec(
  fname := "hockney130_96_33",
  params := [ symbl ] ))

spiral> tt := opts.preProcess(t);
TFCall(IOPrunedMDRConv([ 130, 130, 130 ], Lambda([ i8 ], nth(symbl, i8)), 1, [ [ 34 .. 129 ], [
34 .. 129 ], [ 34 .. 129 ] ], 1, [ [ 0 .. 32 ], [ 0 .. 32 ], [ 0 .. 32 ] ], true), rec(
  fname := "hockney130_96_33",
  params := [ symbl ] ))
```

Pruned Convolution Breakdown Rule

```
NewRulesFor(MDRConv, rec(
    MDRConv_3D_2trip_zyx_freqdata := rec(
        applicable :=  (self, nt) >> Length(nt.params[1]) = 3 and nt.params[3],
        children := nt -> let(....
            [[ PRDFT(nlist[1], -1),   # stage 1: PRDFT z
               DFT(nlist[2], -1),     # stage 2: DFT y
               CConv(nlist[3], hfunc), # stage 3+4+5: complex conv in x
               DFT(nlist[2], 1), # stage 6: iDFT in y
               IPRDFT(nlist[1], 1),    # stage 7: iPRDFT in z
            ]]),

        apply := (nt, C, cnt) -> let(...
            stage1 := L(2*nfreq*n3*n2, n3) * Tensor(I(n2), Tensor(L(2*nfreq, 2)
                          * prdft1d, I(n3))) * Tensor(L(n2*n1, n2), I(n3)),
            stage2 := Tensor(I(n3), Tensor(RC(pdft1d), I(nfreq))),
            pp := Tensor(L(n3*n2*nfreq, n2*nfreq), I(2)) * Tensor(I(n3), L(2*nfreq*n2, nfreq)),
            ppi := Tensor(I(n3), L(2*nfreq*n2, 2*n2)) * Tensor(L(n3*n2*nfreq, n3), I(2)),
            stage543 := Grp(ppi * IDirSum(i, RC(iopconv)) * pp),
            stage76 := Tensor(L(n2*n1, n1), I(n3)) * Grp(Tensor((Tensor(I(n2), iprdft1d
                 * L(2*nfreq, nfreq)) * Tensor(RC(ipdft1d), I(nfreq))), I(n3))
                 * L(2*nfreq*n3*n2, 2*nfreq*n2)),
            conv3dr := stage76 * stage543 * stage2 * stage1, conv3dr))
));
```

Expanded Nonterminal: RuleTree

```
spiral> rt := opts.search(tt);
TFCall_tag( TFCall(IOPrunedMDRConv([ 130, 130, 130 ], Lambda([ i8 ], nth(symbl, i8)), 1, [ Set([
34..129 ]), Set([ 34..129 ]), Set([ 34..129 ]) ], 1, [ Set([ 0..32 ]), Set([ 0.. 32 ]), Set([ 0..32 ])
], true), rec(fname := "hockney130_96_33", params := [ symbl ] )),
  IOPrunedMDRConv_3D_2trip_zyx_freqdata( IOPrunedMDRConv([ 130, 130, 130 ], Lambda([ i8 ], nth(symbl,
i8)), 1, [Set([ 34..129 ]), Set([ 34..129 ]), Set([ 34..129 ]) ], 1, [ Set([ 0..32 ]), Set([ 0.. 32 ]),
Set([ 0..32 ]) ], true),
    PrunedPRDFT_CT_rec_block( PrunedPRDFT(130, 129, 1, Set([ 0..32 ])),
      PRDFT1_Base2( PRDFT(2, 1) ),
      DFT_Base( DFT(2, 1) ),
      PrunedPRDFT_CT_rec_block( PrunedPRDFT(65, 64, 1, Set([ 0..16 ])),
        PRDFT_PD_loop( PRDFT(13, 12) ),
        DFT_PD_loop( DFT(13, 12) ),
        PrunedPRDFT_base( PrunedPRDFT(5, 4, 1, [ 0, 1 ]),
          PRDFT_PD( PRDFT(5, 4) ) ),
        PrunedPRDFT_base( PrunedPRDFT(5, 4, 1, [ 0 ]),
          PRDFT_PD( PRDFT(5, 4) ) ) ),
      PrunedPRDFT_CT_rec_block( PrunedPRDFT(65, 64, 1, Set([ 0..15 ])),
        PRDFT_PD_loop( PRDFT(13, 12) ),
        DFT_PD_loop( DFT(13, 12) ),
        PrunedPRDFT_base( PrunedPRDFT(5, 4, 1, [ 0, 1 ]),
          PRDFT_PD( PRDFT(5, 4) ) ),
        PrunedPRDFT_base( PrunedPRDFT(5, 4, 1, [ 0 ]),
          PRDFT_PD( PRDFT(5, 4) ) ) ) ),
    PrunedDFT_CT_rec_block( PrunedDFT(130, 129, 1, Set([ 0..32 ])),
      DFT_PD( DFT(5, 4) ),
      PrunedDFT_CT_rec_block( PrunedDFT(26, 25, 1, Set([ 0..6 ])),
        DFT_Base( DFT(2, 1) ),
        PrunedDFT_base( PrunedDFT(13, 12, 1, [ 0, 1, 2, 3 ]),
          DFT_PD_loop( DFT(13, 12) ) ),
        PrunedDFT_base( PrunedDFT(13, 12, 1, [ 0, 1, 2 ]),
          DFT_PD_loop( DFT(13, 12) ) ) ),
      PrunedDFT_CT_rec_block( PrunedDFT(26, 25, 1, Set([ 0, 1, 2, 3, 4, 5 ])),
        DFT_Base( DFT(2, 1) ),
        PrunedDFT_base( PrunedDFT(13, 12, 1, [ 0, 1, 2 ]),
          DFT_PD_loop( DFT(13, 12) ) ) ),
...
```

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:

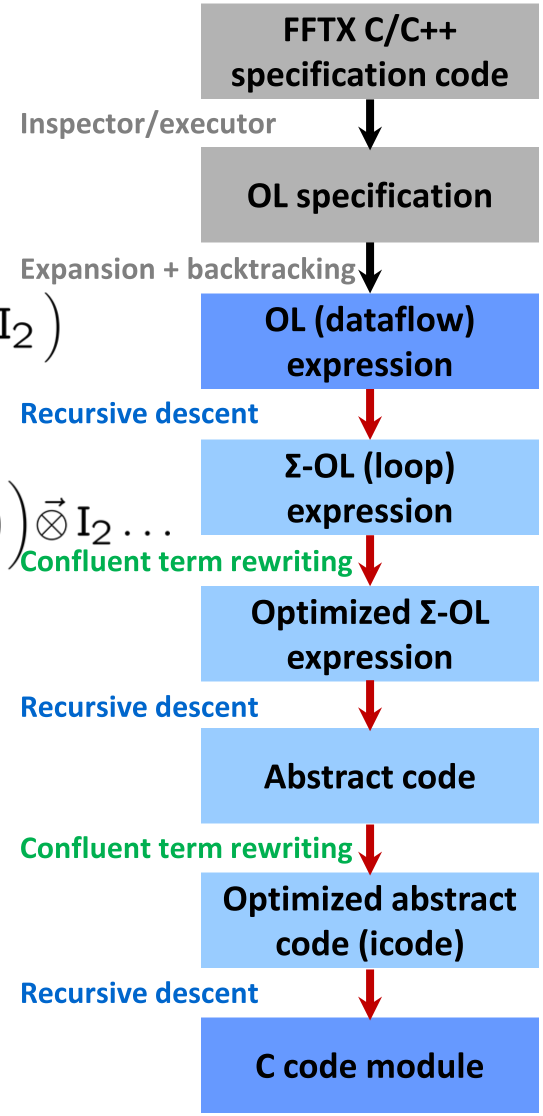
$$\left((F_2 \otimes I_2) T^4_2 (I_2 \otimes F_2) L^4_2 \vec{\otimes} I_2\right) \underbrace{T^8_2}_{\mathrm{vec}(2)} \left(I_2 \otimes \underbrace{L^4_2}_{\mathrm{vec}(2)} (F_2 \vec{\otimes} I_2)\right)\left(L^4_2 \vec{\otimes} I_2\right)$$

∑-OL:

$$\left(\sum_{j=0}^{1}\left(S_{\imath_2\otimes(j)_2} F_2 \mathrm{Map}^2_{x\mapsto\omega_4^{2i+j}x} G_{\imath_2\otimes(j)_2}\right)\sum_{j=0}^{1}\left(S_{(j)_2\otimes\imath_2} F_2 G_{\imath_2\otimes(j)_2}\right)\right)\vec{\otimes} I_2 \ldots$$

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;
}
```



Internal Representation: SIMT Σ-SPL

```
... * SIMTSUM( ASIMTLoopDim(),
   SIMTISum(ASIMTLoopDim(), i81, 3,
     SIMTISum(ASIMTLoopDim(), i215, 13, BB(
       Scat(fTensor(fCompose(H(130, 78, 0, 1), fTensor(fBase(i81), fId(2), fBase(i215))), fId(2))) *
       Blk([ [ V(1), V(0), V(1), V(0) ], [ V(0), V(1), V(0), V(1) ],
             [ V(1), V(0), V(-1), V(0) ], [ V(0), V(1), V(0), V(-1) ] ]) *
       RCDiag(FDataOfs(D62, 4, mul(V(4), i215)), I(2)) * Gath(fTensor(fId(2), fBase(i215), fId(2))) )
     ) *
     SIMTSUM( ASIMTLoopDim(),
       SIMTSUM( ASIMTLoopDim(),
         ISumAcc(i445, 13, BB(
           ScatAcc(fTensor(fCompose(H(26, 13, 0, 1), fBase(13, V(0))), fId(2))) *
           Blk([ [ V(1), V(0) ], [ V(0), V(1) ] ]) ) *
           COND(eq(i445, V(0)),
             BB(Gath(fTensor(fAdd(13, 1, 0), fId(2)))),
             BB(Gath(fTensor(fAdd(13, 1, i445), fId(2))))
           ) ),
         SIMTISum(ASIMTLoopDim(), i431, 6,
           ISumAcc(i433, 7, BB(
             ScatAcc(fTensor(fCompose(H(26, 13, 0, 1), Lambda([ i437 ], nth(D38, i437)).setRange(13), H(13, 12, 1, 1),
               fTensor(fId(2), fBase(i431))), fId(2))) *
               Blk([ [ V(1), V(0), V(1), V(0) ], [ V(0), V(1), V(0), V(1) ],
                     [ V(1), V(0), V(-1), V(0) ], [ V(0), V(1), V(0), V(-1) ] ]) ) *
             COND(eq(i433, V(0)),
               BB(Blk([ [ V(1), V(0) ], [ V(0), V(1) ], [ V(0), V(0) ], [ V(0), V(0) ] ]) *
                 Gath(fTensor(fCompose(Lambda([ i440 ], nth(D39, i440)).setRange(13), fAdd(13, 1, 0)), fId(2)))),
               BB(SIMTSUM( ASIMTLoopDim(),
                   Scat(fTensor(fBase(2, 0), fId(2))) *
                   Blk([[nth(D37, add(mul(V(6), i431), sub(i433, V(1)))), V(0.0) ],
                       [ V(0.0), nth(D37, add(mul(V(6), i431), sub(i433, V(1)))) ] ]) *
                   Gath(fTensor(fBase(2, 0), fId(2))),
                   Scat(fTensor(fBase(2, 1), fId(2))) *
                   RCDiag(RCData(fConst(TReal, 1, E(4))), I(2)) *
                   Blk([ [ nth(D37, add(V(36), mul(V(6), i431), sub(i433, V(1)))), V(0.0) ],
                         [ V(0.0), nth(D37, add(V(36), mul(V(6), i431), sub(i433, V(1)))) ] ]) *
                   Gath(fTensor(fBase(2, 1), fId(2)))) *
                 Blk([ [ V(1), V(0), V(1), V(0) ], [ V(0), V(1), V(0), V(1) ],
                       [ V(1), V(0), V(-1), V(0) ], [ V(0), V(1), V(0), V(-1) ] ]) *
                 Gath(fTensor(fCompose(Lambda([ i440 ], nth(D39, i440)).setRange(13), fAdd(13, 12, 1),
                      fTensor(fId(2), fBase(6, sub(i433, V(1))))), fId(2)))
           ) ) ) ) ) * ...
```

Internal Representation: CUDA iCode

```
func(TVoid, "transform", [ hY, hX ],
    chain(
        decl([ b664, g1 ],
            chain(
                cu_memcpy(X, hX, 35937, cudaMemcpyHostToDevice),
                    decl([ ms1, start1, stop1 ],
                        chain(
                            cu_event_create(addrof(start1)),
                            cu_event_create(addrof(stop1)),
                            cu_check_errors(cu_event_record(start1, V(false))),
                            ker_code0<<<g1,b664>>>(X, Y),
                            cu_check_errors(cu_event_record(stop1, V(false))),
                            cu_check_errors(cu_event_synchronize(stop1, V(false))),
                            assign(ms1, V(0)),
                            cu_event_elapsed_time(addrof(ms1), start1, stop1),
                            cu_event_destroy(start1),
                            cu_event_destroy(stop1)
                         )
                     ),
                     cu_memcpy(hY, Y, 884736, cudaMemcpyDeviceToHost)
                  )
               )
            )
         ),
         func(TVoid, "destroy", Set([  ]),
            chain(
               cu_free(Y),
               cu_free(X)
            )
         )
```

SPIRAL Generated CUDA Code

```
__global__ void ker_code0(int *D48, double *D49, double *D50, double *D51, int *D52, double  *X) {
    __shared__ double T235[260];
    ...
    if (((threadIdx.x < 13))) {
        for(int i96 = 0; i96 <= 4; i96++) {
            int a31, a32, a33, a34;
            a31 = (2*i96);
            a32 = (threadIdx.x + (13*a31));
            a33 = (threadIdx.x + (13*((a31 + 5) % 10)));
            a34 = (4*i96);
            *((((T235 + 0) + a34) + (20*threadIdx.x))) = (*((T6 + a32)) + *((T6 + a33)));
            *(((1 + (T235 + 0) + a34) + (20*threadIdx.x))) = 0.0;
            *(((2 + (T235 + 0) + a34) + (20*threadIdx.x))) = (*((T6 + a32)) - *((T6 + a33)));
            *(((3 + (T235 + 0) + a34) + (20*threadIdx.x))) = 0.0;
        }
        double t261, t262, t263, t264, t265, t266, t267, t268;
        int a129;
        t263 = (*((((T235+0)+12)+(20*threadIdx.x)))+*((((T235+0)+8)+(20*threadIdx.x))));
        t264 = (*((((T235+0)+12)+(20*threadIdx.x)))-*((((T235+0)+8)+(20*threadIdx.x))));
        ...
        *((3 + T5 + a129)) = ((0.58778525229247314*t268) - (0.95105651629515353*t266));
    }
    __syncwarp();
    if (((threadIdx.x < 1))) {
        double t305, t306, t307, t308, t309, t310, t311, t312, t313, t314, t315, t316;
        int a387;
        t305 = (*((T5 + 12)) + *((T5 + 144)));
        ...
```

3,000 lines of code, kernel fusion, cross call data layout transforms