The Design of FFTX

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

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)

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

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

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

Poisson's equation. Δ is the Laplace operator

Solution characterization

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

$$\Phi(\vec{x}) = \frac{Q}{4\pi||\vec{x}||} + o\left(\frac{1}{||\vec{x}||}\right) \quad \text{as } ||\vec{x}|| \to \infty$$

$$Q = \int_D \rho d\vec{x}$$

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)

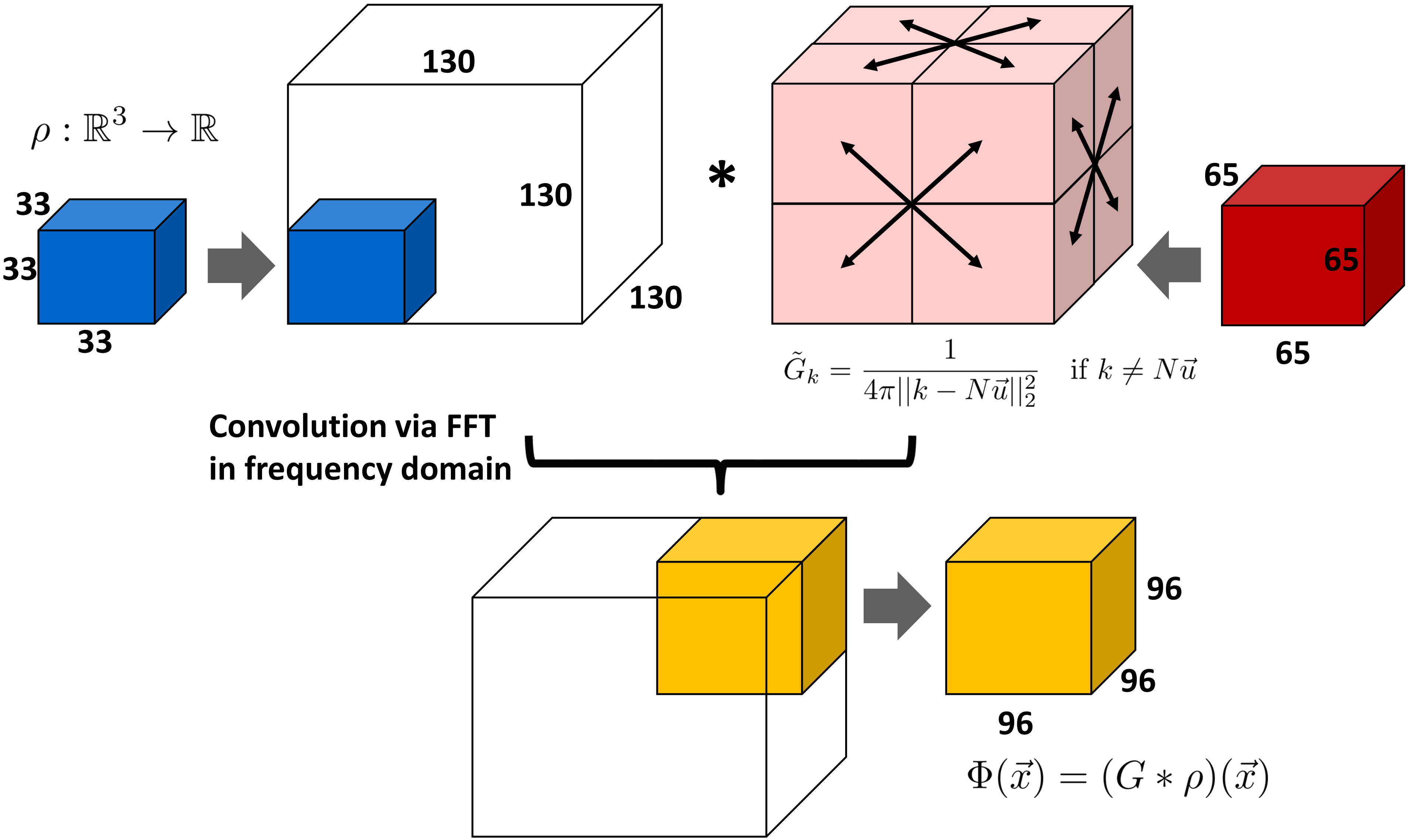
$$\tilde{G}_k = \frac{1}{4\pi||k - N\vec{u}||_2^2} \quad \text{if } k \neq N\vec{u}$$

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



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

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 C Code: 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 = 3,
    batch_rank = 0,
    ...
    fftx_plan plans[5];
    fft

    tmp

    pla

    tmp
    pla

    tmp
    pla

    tmp
    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;
}
```

This is a specification dressed as a program

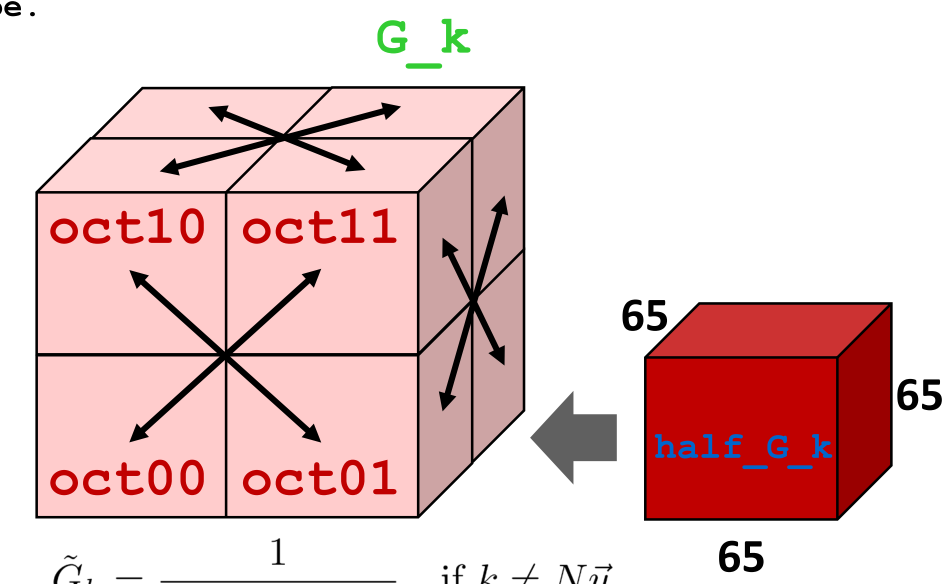
- Needs to be clean and concise
- No code level optimizations and tricks
- Don't think "performance" but "correctness"
- *For small and in-development platforms*

Looks like FFTW calls, but is a specification for SPIRAL

FFTX C Code: Describing The Hockney Symmetry

```
// FFTX data access descriptors.
// Access is to four octants of a symmetric cube.
// Cube size is N^3 and M = N/2.

fftx_iodimx oct00[] = {
  { M+1, 0, 0, 0, 1, 1, 1 },
  { M+1, 0, 0, 0, M+1, 2*M, 1 },
  { M+1, 0, 0, 0, (M+1)*(M+1), 4*M*M, 1} },
  oct01[] = {
  { M-1, M-1, M+1, 0, -1, 1, 1 },
  { M+1, 0, 0, 0, M+1, 2*M, 1 },
  { M+1, 0, 0, 0, (M+1)*(M+1), 4*M*M, 1} },
  oct10[] = {
  { M+1, 0, 0, 0, 1, 1, 1 },
  { M-1, M-1, M+1, 0, -(M+1), 2*M, 1 },
  { M+1, 0, 0, 0, (M+1)*(M+1), 4*M*M, 1} },
 oct11[] = {
  { M-1, M-1, M+1, 0, -1, 1, 1 },
  { M-1, M-1, M+1, 0, -(M+1), 2*M, 1 },
  { M+1, 0, 0, 0, (M+1)*(M+1), 4*M*M, 1} };
...
```

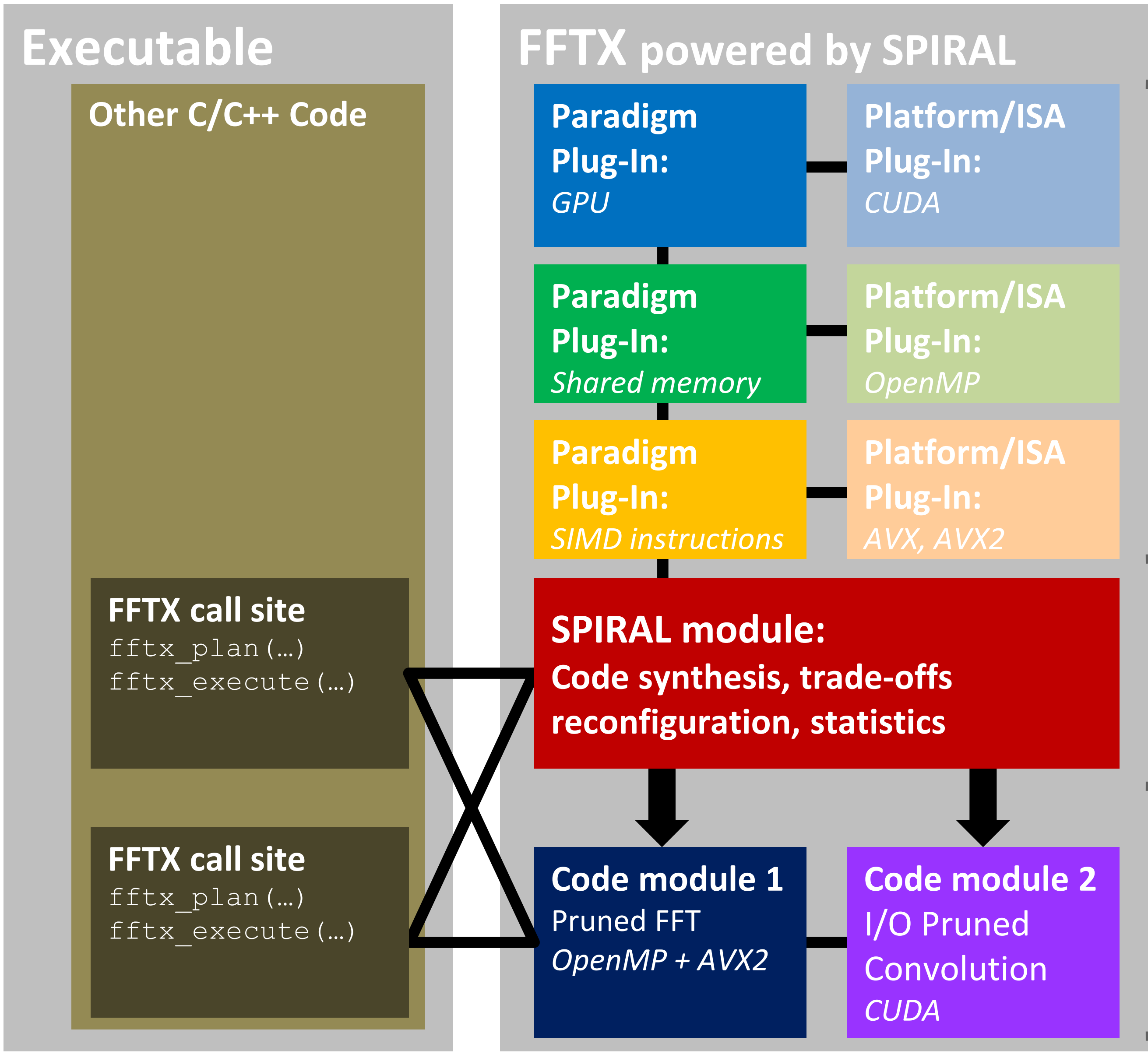


$$\tilde{G}_k = \frac{1}{4\pi||k - N\vec{u}||_2^2} \quad \text{if } k \neq N\vec{u}$$

```
fftx_temp_complex half_G_k = fftx_create_zero_temp_complex(rk, f_d);
plans[2] = fftx_plan_guru_copy_complex(rk, oct00, G_k, half_G_k, FFTX_MODE_SUB);
plans[3] = fftx_plan_guru_copy_complex(rk, oct01, G_k, half_G_k, MY_FFTX_MODE_SUB);
plans[4] = fftx_plan_guru_copy_complex(rk, oct10, G_k, half_G_k, MY_FFTX_MODE_SUB);
plans[5] = fftx_plan_guru_copy_complex(rk, oct11, G_k, half_G_k, MY_FFTX_MODE_SUB);
...
```

We are a developing higher-level more natural geometric API

FFTX Backend: SPIRAL



Extensible platform and programming model definitions

Core system: SPIRAL engine

Automatically generated tuned components and special cases

DARPA BRASS

C/C++ FFTX Program Trace

```
fftx_session := [
  rec(op := "fftx_init", flags := IntHexString("8000000")),
  rec(op := "fftx_create_data_real", rank := 1, dims := [ rec(n := 4, is := 1, os := 1) ],
    ptr := IntHexString("000000D236DD2460")),
  ...
  rec(op := "fftx_create_zero_temp_real", rank := 1,

  re                                                                                    1,


  re



  re
```

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*

```
    callback := [
      rec(op := "call", inp := IntHexString("A000000000000001"),
        outp := IntHexString("A000000000000002"), data := IntHexString("A000000000000003")),
      rec(op := "FFTX_COMPLEX_VAR", var := IntHexString("000000D236A0FA30"),
        re := 0.000000e+00, im := 0.000000e+00),
      rec(op := "FFTX_COMPLEX_MOV", target := IntHexString("000000D236A0FA30"),
        source := IntHexString("A000000000000001")),
      rec(op := "FFTX_COMPLEX_MUL", target := IntHexString("000000D236A0FA30"),
        source := IntHexString("A000000000000003")),
      ...
```

SPIRAL Script Captures Performance Engineering

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

# set up algorithms needed for multi-dimensional pruned real convolution
opts
opts
  PR
opts
  IP
opts
opts                                                                    1];
opts

# sp
[N,

# de
n_fr

# pr
sym
t :=

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

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

Recognizes pattern and applies code generation

- Developed by
  performance engineer + application specialist
- Casts FFTX call sequence as SPIRAL non-terminal
- Does code generation and autotuning
- *Clear separation of concerns frontend/backend*

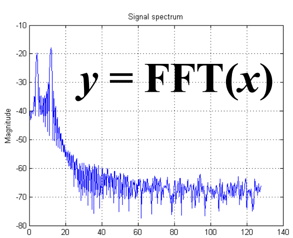
SPIRAL: Go from Mathematics to Software

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

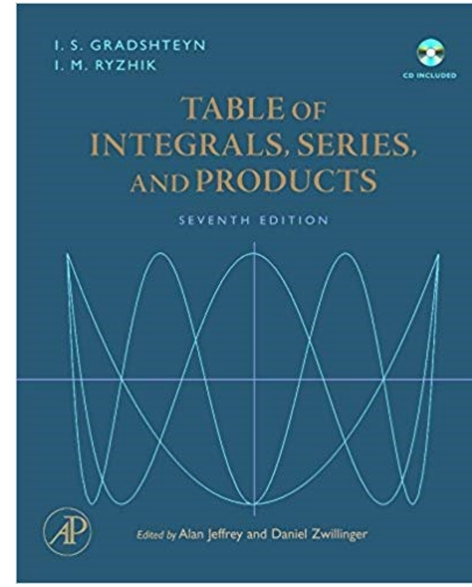
RATIONAL FUNCTIONS

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

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

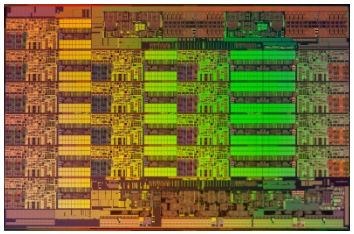
Wolfram Mathematica

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

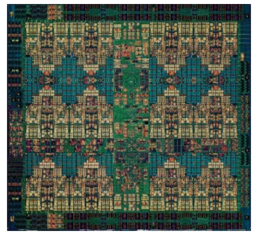
Out[33]= 0

SPIRAL’s Target Computing Landscape

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

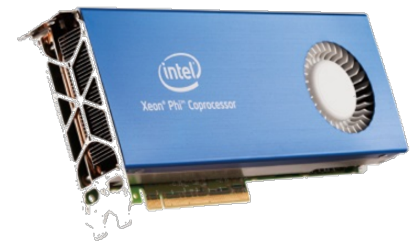


Intel Xeon 8180M
2.25 Tflop/s, 205 W
28 cores, 2.5—3.8 GHz
2-way—16-way AVX-512



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

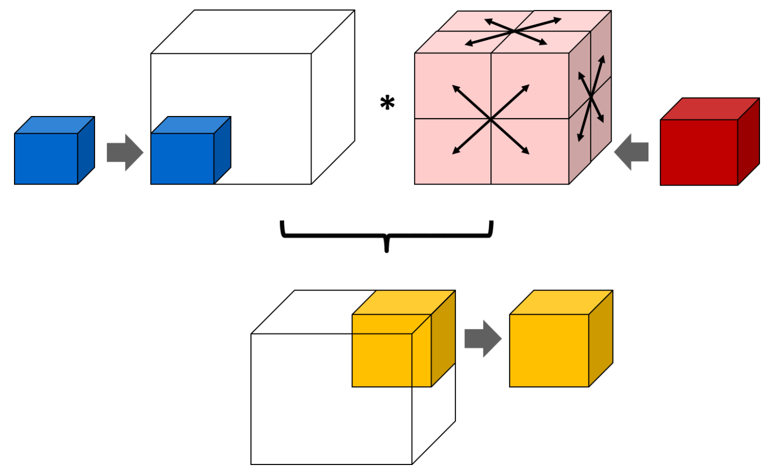
Summit
187.7 Pflop/s, 13 MW
9,216 x 22 cores POWER9
+ 27,648 V100 GPUs

Rules in Internal Domain Specific Language

Linear Transforms

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

Spectral Domain Algorithms



Hardware

- Multithreading (Multicore)
- Vector SIMD (SSE, VMX/Altivec,…)
- Message Passing (Clusters, MPP)
- Streaming/multibuffering (Cell)
- Graphics Processors (GPUs)
- Gate-level parallelism (FPGA)
- HW/SW partitioning (CPU + FPGA)

$$\mathrm{I}_p \otimes_{\|} A_{\mu n}, \quad \mathbf{L}^{mn}_m \bar{\otimes} \mathrm{I}_\mu$$

$$A \hat{\otimes} \mathrm{I}_\nu \quad \underbrace{\mathbf{L}^{2\nu}_2}_{\text{isa}}, \quad \underbrace{\mathbf{L}^{2\nu}_\nu}_{\text{isa}}, \quad \underbrace{\mathbf{L}^{\nu^2}_\nu}_{\text{isa}}$$

$$\mathrm{I}_p \otimes_{\|} A_n, \quad \underbrace{\mathbf{L}^{p^2}_p \bar{\otimes} \mathrm{I}_{n/p^2}}_{\text{all-to-all}}$$

$$\mathrm{I}_n \otimes_2 A_{\mu n}, \quad \mathbf{L}^{mn}_m \bar{\otimes} \mathrm{I}_\mu$$

$$\prod_{i=0}^{n-1} A_i, \quad A_n \hat{\otimes} \mathrm{I}_w, \quad P_n \otimes Q_w$$

$$\prod_{i=0}^{n-1}{}^{\text{ir}} A, \quad \mathrm{I}_s \tilde{\otimes} A, \quad \underbrace{\mathbf{L}^m_n}_{\text{bram}}$$

$$\underbrace{A_1}_{\text{fpga}}, \quad \underbrace{A_2}_{\text{fpga}}, \quad \underbrace{A_3}_{\text{fpga}}, \quad \underbrace{A_4}_{\text{fpga}}$$

Program Transformations

$$\underbrace{AB}_{\text{smp}(p,\mu)} \rightarrow \underbrace{A}_{\text{smp}(p,\mu)} \underbrace{B}_{\text{smp}(p,\mu)}$$

$$\underbrace{A_m \otimes \mathrm{I}_n}_{\text{smp}(p,\mu)} \rightarrow \underbrace{\left(\mathrm{L}^{mp}_m \otimes \mathrm{I}_{n/p} \right) \left(\mathrm{I}_p \otimes (A_m \otimes \mathrm{I}_{n/p}) \right) \left(\mathrm{L}^{mp}_p \otimes \mathrm{I}_{n/p} \right)}_{\text{smp}(p,\mu)}$$

$$\underbrace{\mathrm{L}^{mn}_m}_{\text{smp}(p,\mu)} \rightarrow \begin{cases} \underbrace{\left(\mathrm{I}_p \otimes \mathrm{L}^{mn/p}_{m/p} \right)}_{\text{smp}(p,\mu)} \underbrace{\left(\mathrm{L}^{pn}_p \otimes \mathrm{I}_{m/p} \right)}_{\text{smp}(p,\mu)} \\ \underbrace{\left(\mathrm{L}^{pm}_m \otimes \mathrm{I}_{n/p} \right)}_{\text{smp}(p,\mu)} \underbrace{\left(\mathrm{I}_p \otimes \mathrm{L}^{mn/p}_m \right)}_{\text{smp}(p,\mu)} \end{cases}$$

Recursive rules

$$\underbrace{\mathrm{I}_m \otimes A_n}_{\text{smp}(p,\mu)} \rightarrow \mathrm{I}_p \otimes_{\|} \left(\mathrm{I}_{m/p} \otimes A_n \right)$$

$$\underbrace{(P \otimes \mathrm{I}_n)}_{\text{smp}(p,\mu)} \rightarrow \left(P \otimes \mathrm{I}_{n/\mu} \right) \overline{\otimes}\, \mathrm{I}_\mu$$

Base case rules

Autotuning in Constraint Solution Space

AVX 2-way `_Complex double`

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

DFT_8

Base cases

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

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

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

Transformation rules

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

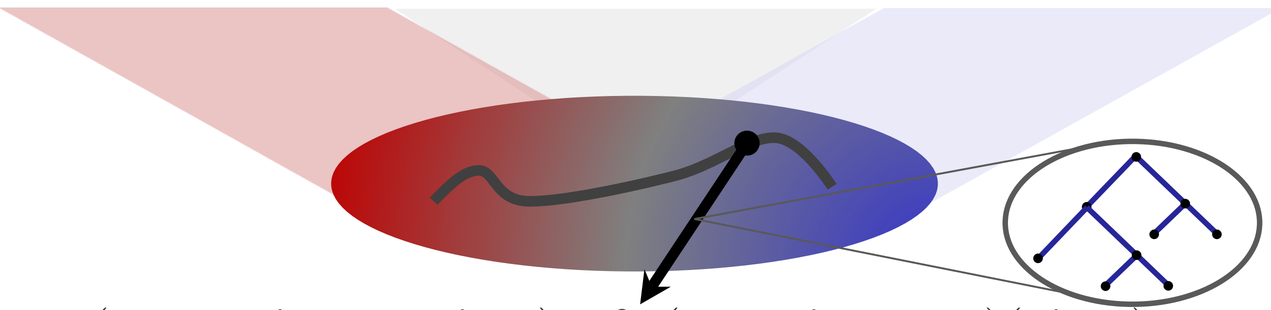
$$\mathrm{L}^{n\nu}_\nu \to (\mathrm{L}^n_\nu\otimes \mathrm{I}_\nu)(\mathrm{I}_{n/\nu}\otimes \mathrm{L}^{\nu^2}_\nu)$$

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

Breakdown rules

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

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



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

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

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:

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

∑-OL:

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

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;
}
```

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 (icode)

Recursive descent

C code module

Get Going Quickly: Hockney with OpenACC

```
void ioprunedconv_130_0_62_72_130(double  *Y, double  *X, double * S) {
    static double D84[260] = {65.5, 0.0, (-0.500000000000001132), (-20.686114762237267),
      (-0.5000000000000081), (-10.337014680426078), (-0.5000000000000455),
    ...
for(int i18899 = 0; i18899 <= 1; i18899++) {
    for(int i18912 = 0; i18912 <= 4; i18912++) {
        a9807 = ((2*i18899) + (4*i18912));
        a9808 = (a9807 + 1);
        a9809 = (a9807 + 52);
        a9810 = (a9807 + 53);
        a9811 = (a9807 + 104);
        a9812 = (a9807 + 105);
        s3295 = (*((X + a9807)) + *((X + a9809))
            + *((X + a9811)));
        s3296 = (*((X + a9808)) + *((X + a9810))
            + *((X + a9812)));
        s3297 = (((0.30901699437494745**((X + a9809))
            - (0.80901699437494745**((X + a9811))))
            + *((X + a9807)));
        ...
        *((104 + Y + a12569)) = ((s3983 - s3987)
            + (0.80901699437494745*t6537)
            + (0.58778525229247314*t6538));
        *((105 + Y + a12569)) = (((s3984 - s3988)
            + (0.80901699437494745*t6538))
            - (0.58778525229247314*t6537));
    }
}
```

FFTX/SPIRAL with OpenACC backend

Compared to cuFFT expert interface

15% faster on TITAN V

Same speed on Tesla V100

1,000s of lines of special GPU code, transparently used

Backend: SPIRAL Code Generation

```
__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)) = ((0.5877852522924731*t268) - (0.9510565162951535*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)));
        ...
```

FFTX/SPIRAL with CUDA backend

Early result:

220 Gflop/s

70% faster than cuFFT

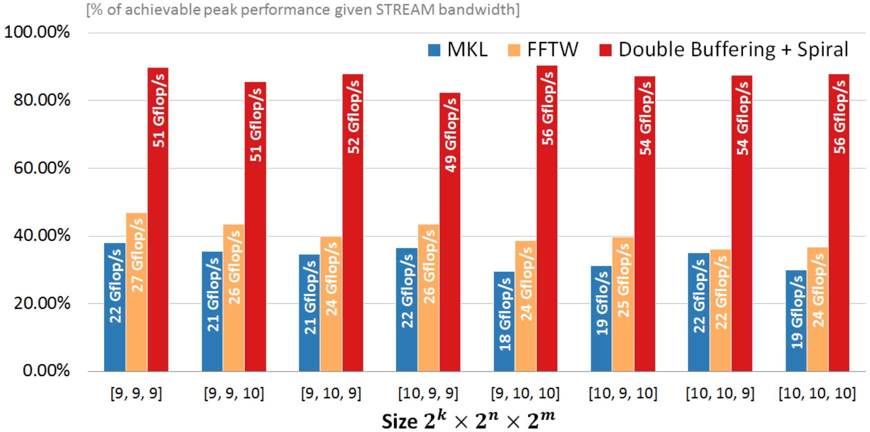
3,000 lines of code, kernel fusion, cross call data layout transforms

Selected Results: FFTs and Spectral Algorithms

3D FFT performance on Intel Kaby Lake 7700K

4.5 GHz, 4/8 cores/threads, double-precision, AVX

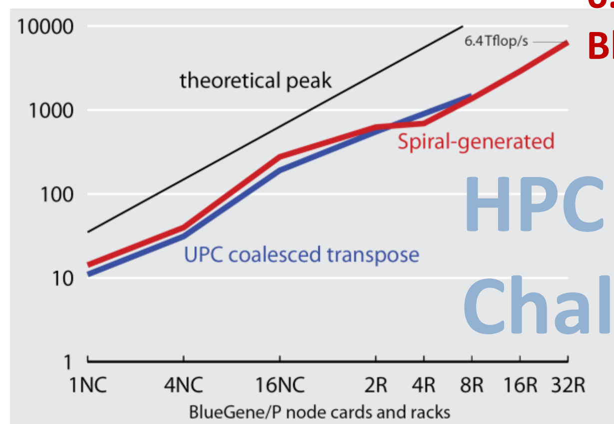
[% of achievable peak performance given STREAM bandwidth]



FFT on Multicore

Global FFT (1D FFT, HPC Challenge)

performance [Gflop/s]



6.4 Tflop/s on BlueGene/P

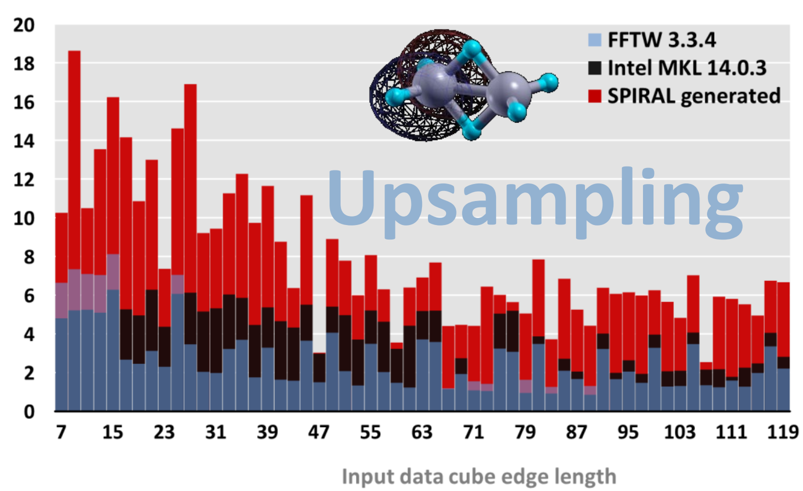
HPC Challenge

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

Performance of 2x2x2 Upsampling on Haswell

3.5 GHz, AVX, double precision, interleaved input, single core

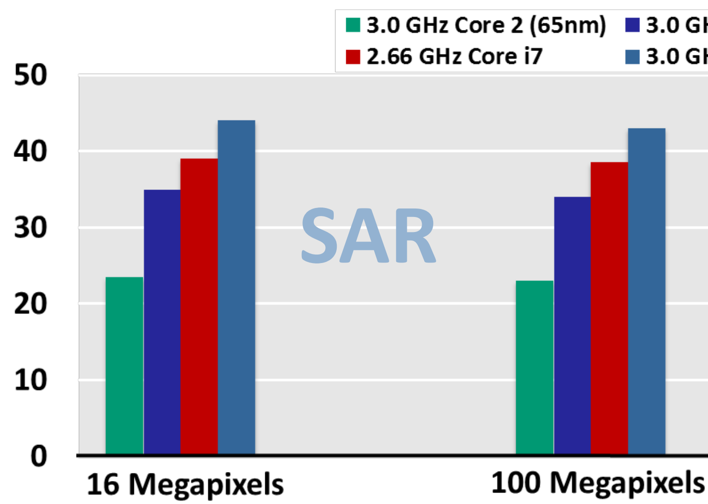
Performance [Pseudo Gflop/s]



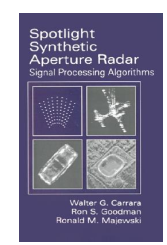
Upsampling

PFA SAR Image Formation on Intel platforms

performance [Gflop/s]



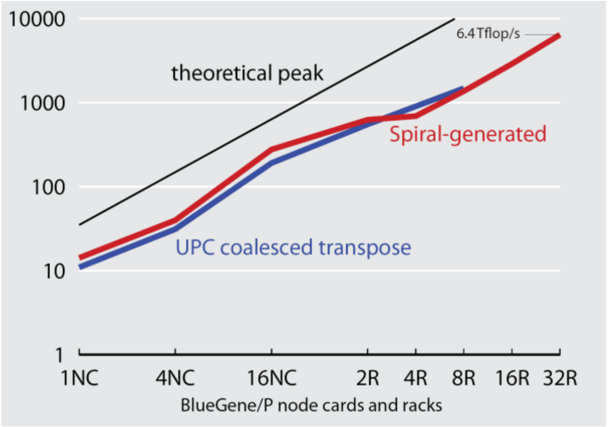
SAR



SPIRAL: Success in HPC/Supercomputing

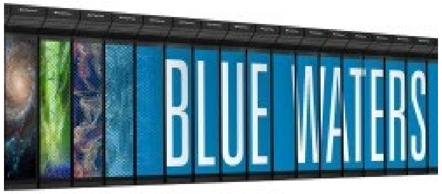
- **NCSA Blue Waters**
  PAID Program, FFTs for Blue Waters
- **RIKEN K computer**
  FFTs for the HPC-ACE ISA
- **LANL RoadRunner**
  FFTs for the Cell processor
- **PSC/XSEDE Bridges**
  Large size FFTs
- **LLNL BlueGene/L and P**
  FFTW for BlueGene/L's Double FPU
- **ANL BlueGene/Q Mira**
  Early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

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



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





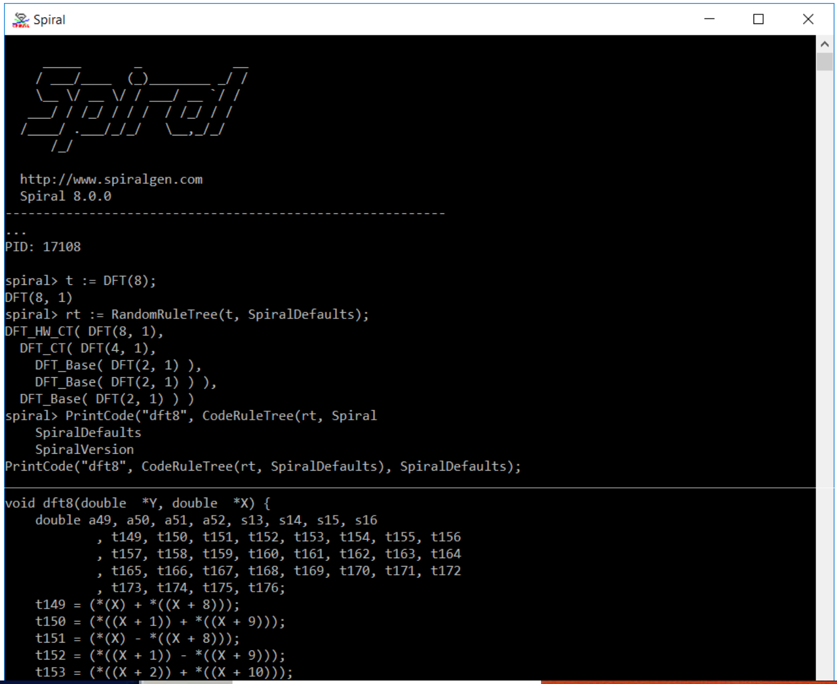
2006 Gordon Bell Prize (Peak Performance Award) with LLNL and IBM

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

SPIRAL 8.1.0: 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*