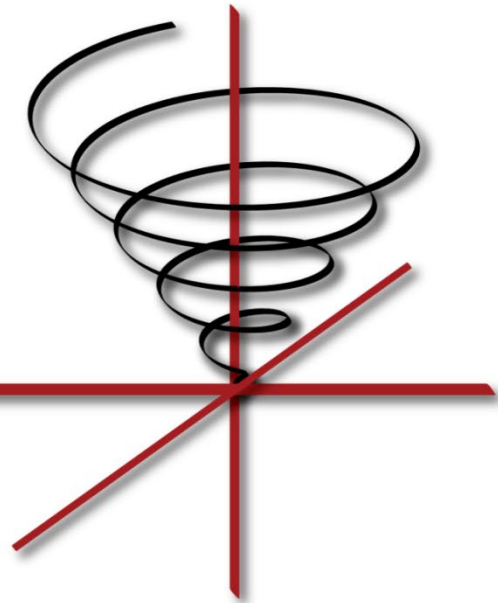


Updates on Sequential and Parallel FFTX



Franz Franchetti
Carnegie Mellon University

in collaboration with

Tze Meng Low, Het Mankad
Carnegie Mellon University

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

Mike Franusich, Patrick Broderick
SpiralGen, Inc.

This work was supported by DOE ECP project 2.2.6.04

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 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

FFTX and SpectralPACK

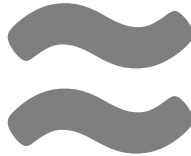
Numerical Linear Algebra

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,...
- **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)

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

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

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

Poisson's equation. Δ is the Laplace operator

Solution characterization

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

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

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

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

Method of Local Corrections (MLC)

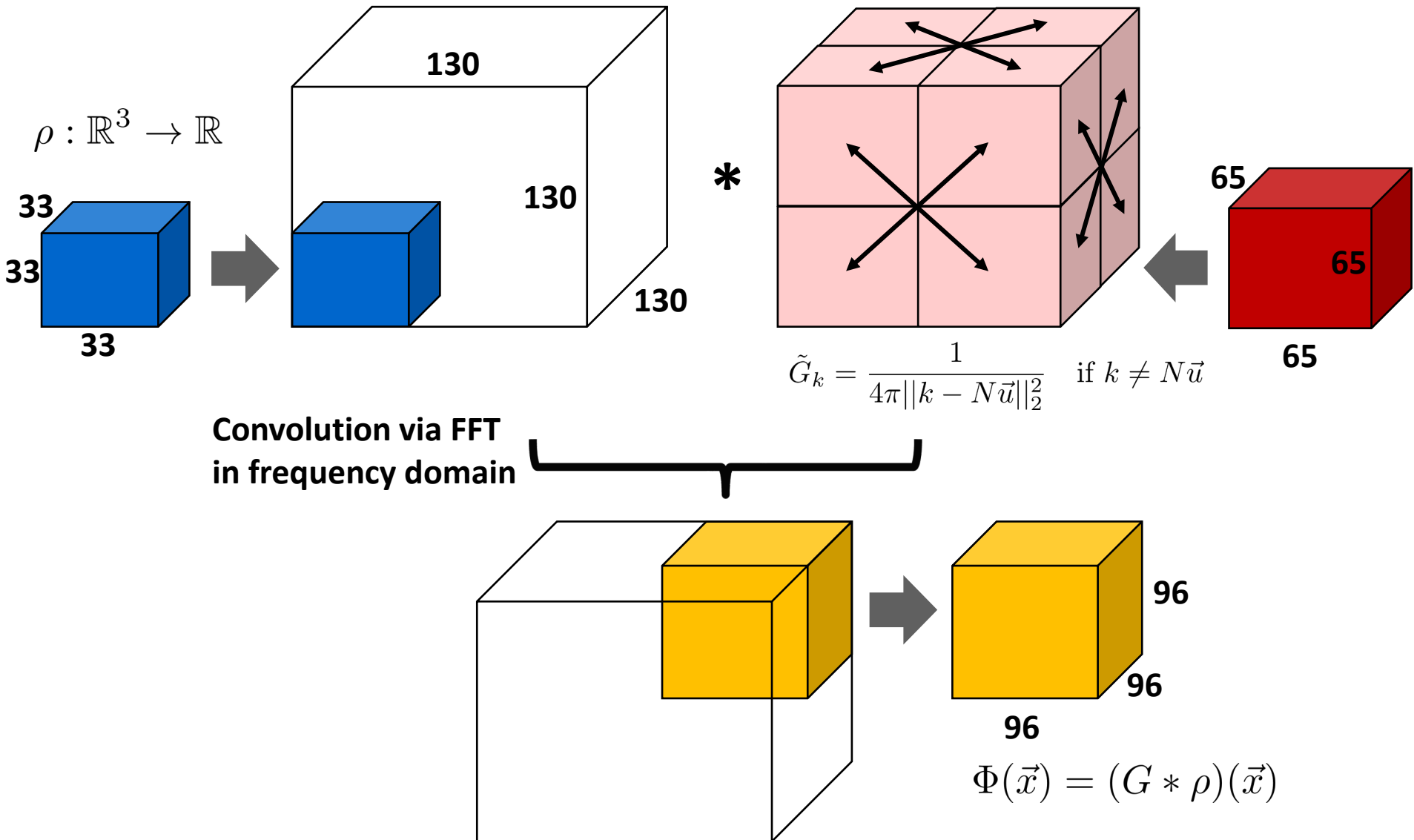
$$\tilde{G}_k = \frac{1}{4\pi\|k - N\vec{u}\|_2^2} \quad \text{if } k \neq N\vec{u}$$

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: Forward/Inverse MDDFT

```
#include "fftx3.hpp"
```

```
...  
int main  
{
```

```
    trac
```

```
    const
```

```
    box_t
```

```
    array
```

```
    array
```

```
    std::
```

```
    setIn
```

```
    setOutputs(outputs);
```

```
    openScalarDAG();
```

```
    MDDFT(domain.extents(), 1, intermediates[0], inputs);
```

```
    RCDiag(domain.extents(), 1, intermediates[1], intermediates[0]);
```

```
    IMDDFT(domain.extents(), 1, outputs, intermediates[1]);
```

```
    closeScalarDAG(intermediates, "mdconv");
```

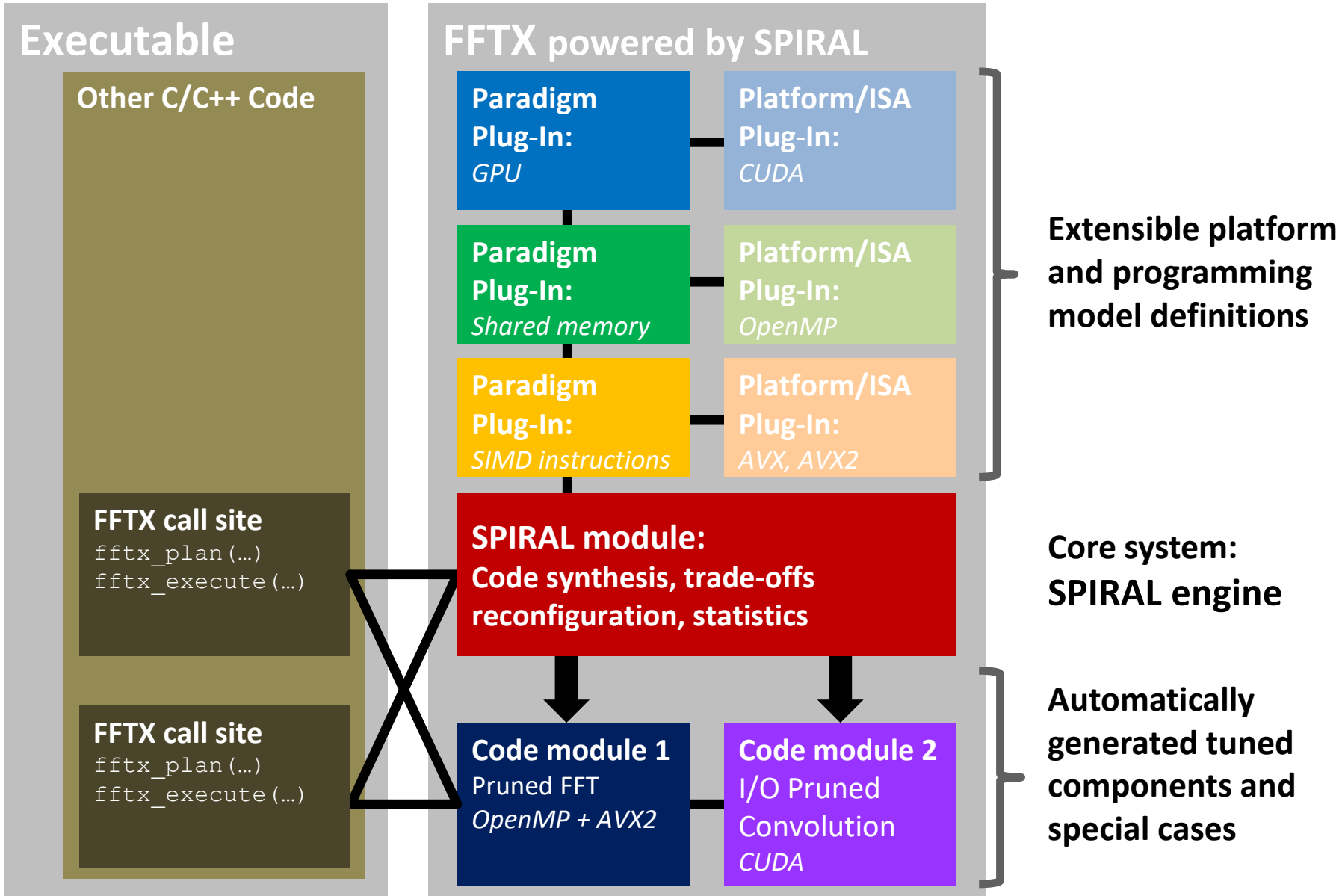
```
}
```

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*

Defines the semantics of a new library call/object

FFTX Backend: SPIRAL



Calling The Generated Transform

```
#include "mdconv.fftx.codegen.hpp"
...
mdconv::init();

fftx::box_t<3> domain(fftx::point_t<3>({{1,1,1}}),
                    fftx::point_t<3>({{80,80,80}}));

fftx::array_t<3, std::complex<double>> input(domain);
fftx::array_t<3, std::complex<double>> output(domain);
fftx::array_t<3, std::complex<double>> symbol(domain);

// initialize inputs and symbol
...
// Call transform.
mdconv::transform(input, output, symbol);
...
mdconv::destroy();
...
```

Invokes the new library/object as defined before

The Generated Transform

```
mddfft3d_1024x1024x1024_mpi.cu - Notepad
File Edit Format View Help
t1623 = (t1611 + t1617);
t1624 = (t1610 - t1616);
t1625 = (t1611 - t1617);
a2899 = ((8388608*blockIdx.y) + (2048*threadIdx.z) + (131072*blockIdx.x) + (2*i20) + (64*i32));
Y[a2899] = (t1618 + t1622);
Y[(a2899 + 1)] = (t1619 + t1623);
Y[(a2899 + 1024)] = (t1618 - t1622);
Y[(a2899 + 1025)] = (t1619 - t1623);
Y[(a2899 + 512)] = (t1620 + t1625);
Y[(a2899 + 513)] = (t1621 - t1624);
Y[(a2899 + 1536)] = (t1620 - t1625);
Y[(a2899 + 1537)] = (t1621 + t1624);
t1626 = (t1608 + t1615);
t1627 = (t1609 - t1614);
t1628 = (t1608 - t1615);
t1629 = (t1609 + t1614);
t1630 = (s854 + s864);
t1631 = (s855 - s865);
t1632 = (s854 - s864);
t1633 = (s855 + s865);
Y[(a2899 + 256)] = (t1626 + t1630);
Y[(a2899 + 257)] = (t1627 + t1631);
Y[(a2899 + 1280)] = (t1626 - t1630);
Y[(a2899 + 1281)] = (t1627 - t1631);
Y[(a2899 + 768)] = (t1628 + t1633);
Y[(a2899 + 769)] = (t1629 - t1632);
Y[(a2899 + 1792)] = (t1628 - t1633);
Y[(a2899 + 1793)] = (t1629 + t1632);
}
}
void mddfft3d_1024x1024x1024_mpi(double *Y, double *X) {
    dim3 b458(1, 1, 64), b459(1, 1, 1), b460(1, 1, 64), b461(1, 1, 1), b462(1, 1, 64), g1(64, 256, 1), g2(1, 1, 1), g3(64, 256, 1),
    g4(1, 1, 1), g5(64, 256, 1);
    ker_code0<<<g1, b458>>>(X);
    fftx_mpi_rcperm(Q2, Q1, 1073741824, 2, 3, 1024, 1024, 1024);
    ker_code2<<<g3, b460>>>();
    fftx_mpi_rcperm(Q2, Q1, 1073741824, 1, 3, 1024, 1024, 1024);
    ker_code4<<<g5, b462>>>(Y);
}
```

2,000 lines of code, kernel fusion, cross call data layout transforms

Distributed FFTX

```
const int nx=32;
const int ny=32;
const int nz=32;

const int p = 2;

box_t<2> grid2D(point_t<2>({{0, 0}}),
              point_t<2>({{p-1, p-1}}));

//canonical 2D global distribution
vector<FFTX_Distribution> inDist({FFTX_NO_DIST, FFTX_GRID_X, FFTX_GRID_Y});

//rotated 2D global distribution
vector<FFTX_Distribution> outDist({FFTX_NO_DIST, FFTX_GRID_X, FFTX_GRID_Y});

box_t<3> empty(point_t<3>({{1,1,1}}), point_t<3>({{0,0,0}}));
box_t<3> domain(point_t<3>({{1,1,1}}), point_t<3>({{nx,ny,nz}}));

std::array<array_t<3,std::complex<double>>,1> intermediates {{empty}};
d_array_t<3,std::complex<double>> inputs(domain, inDist);
d_array_t<3,std::complex<double>> outputs(domain, outDist);

//Describe Global Computation
openScalarDAG(grid2D);

MDDFT(domain.extents(), 1, outputs, inputs); //polymorphic by type

closeScalarDAG(intermediates, "mddft", grid2D);
```

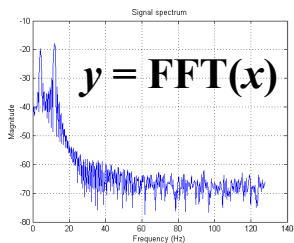
SPIRAL: Go from Mathematics to Software

Given:

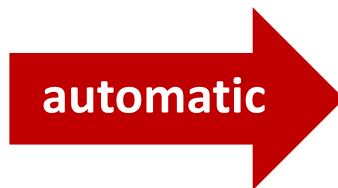
- Mathematical problem specification
core mathematics does not change
- Target computer platform
varies greatly, new platforms introduced often

Wanted:

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



on

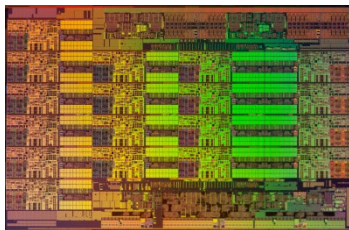


```
void fft64(double *Y, double *X) {
    ...
    s5674 = _mm256_permute2f128_pd(s5672, s5673, (0) | ((2) << 4));
    s5675 = _mm256_permute2f128_pd(s5672, s5673, (1) | ((3) << 4));
    s5676 = _mm256_unpacklo_pd(s5674, s5675);
    s5677 = _mm256_unpackhi_pd(s5674, s5675);
    s5678 = *((a3738 + 16));
    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));
    ...
}
```

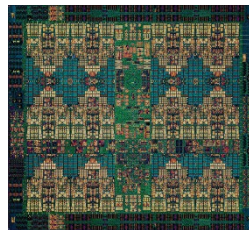


SPIRAL's Target Computing Landscape

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



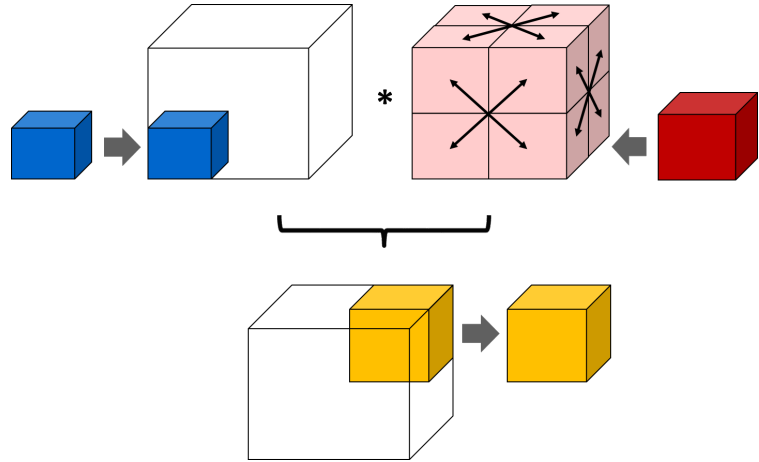
Summit
187.7 Pflop/s, 13 MW
9,216 x 22 cores POWER9
+ 27,648 V100 GPUs

Rules in Internal Domain Specific Language

Linear Transforms

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

Spectral Domain Algorithms



Hardware

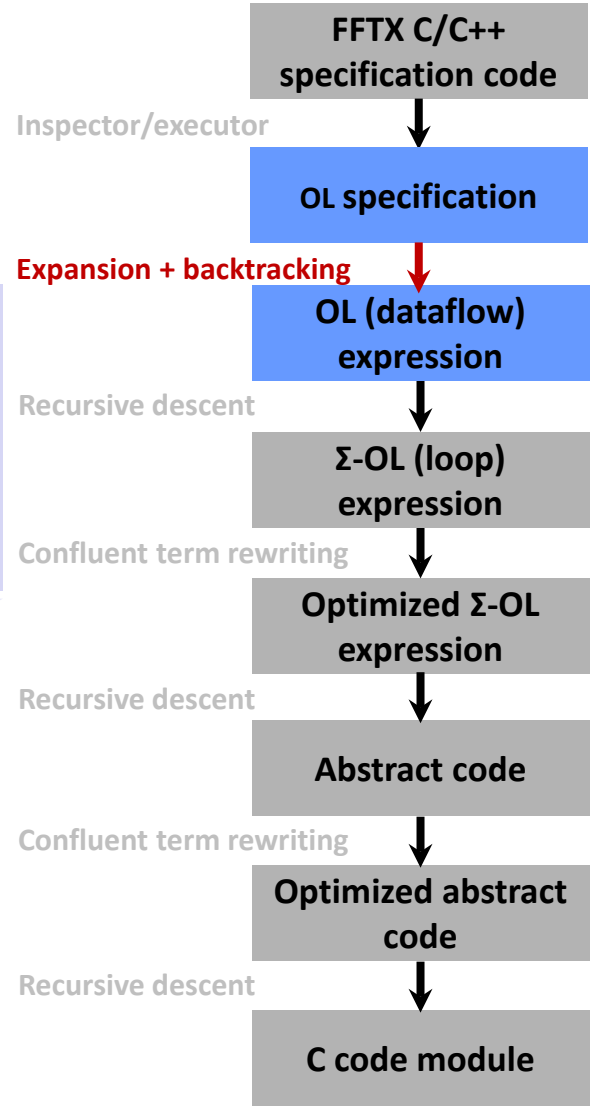
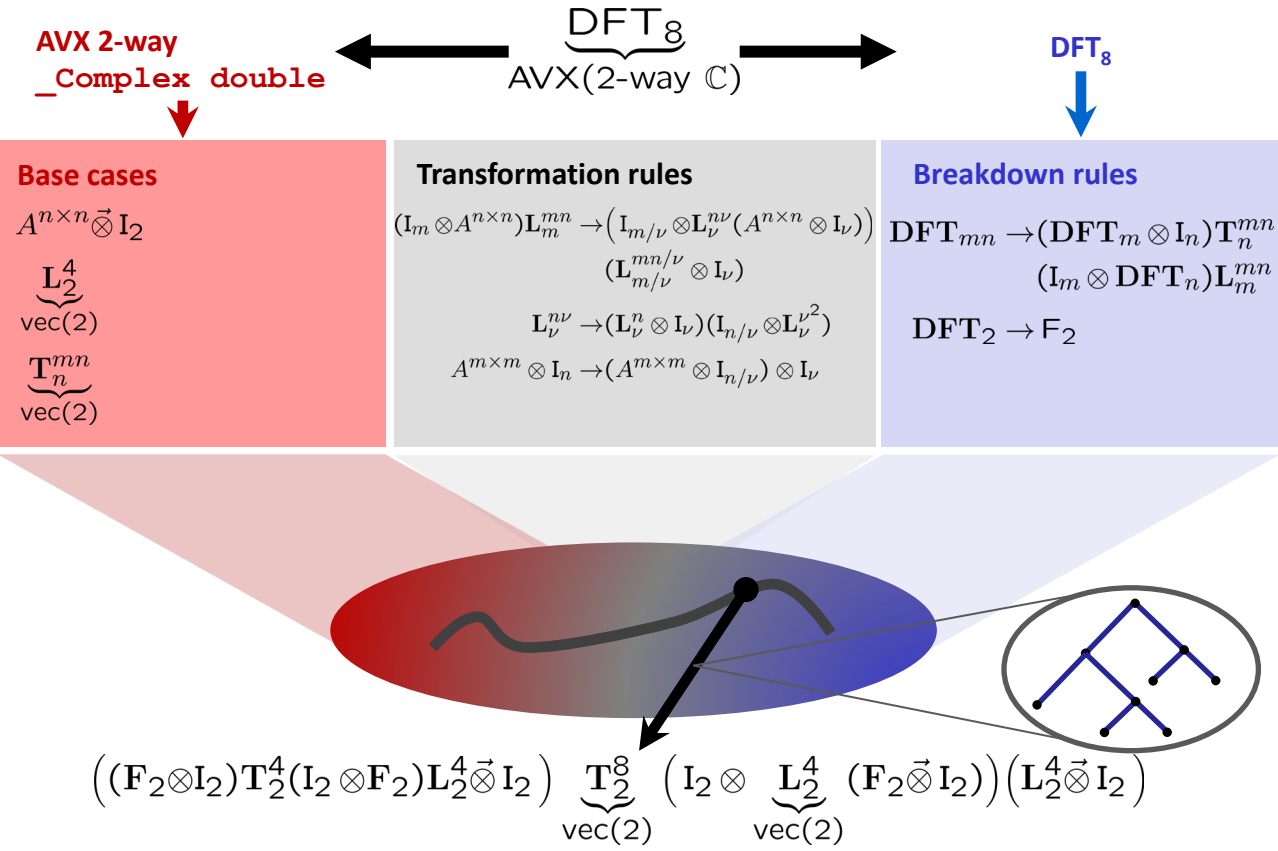
- Multithreading (Multicore)
- Vector SIMD (SSE, VMX/Altivec,...)
- Message Passing (Clusters, MPP)
- Streaming/multibuffering (Cell)
- Graphics Processors (GPUs)
- Gate-level parallelism (FPGA)
- HW/SW partitioning (CPU + FPGA)

$$\begin{aligned}
 &\text{I}_p \otimes_{\parallel} A_{\mu n}, \quad \text{L}_m^{mn} \otimes \text{I}_{\mu} \\
 &A \otimes \text{I}_\nu, \quad \text{L}_{\text{isa}}^{2\nu}, \quad \text{L}_{\text{isa}}^{2\nu}, \quad \text{L}_{\text{isa}}^{2\nu} \\
 &\text{I}_p \otimes_{\parallel} A_n, \quad \text{L}_p^{p^2} \otimes \text{I}_{n/p^2} \\
 &\text{I}_n \otimes_2 A_{\mu n}, \quad \text{L}_m^{mn} \otimes \text{I}_{\mu} \\
 &\prod_{i=0}^{n-1} A_i, \quad A_n \otimes \text{I}_w, \quad P_n \otimes Q_w \\
 &\prod_{i=0}^{n-1} A_i, \quad \text{I}_s \otimes A_i, \quad \text{L}_{\text{bram}}^m \\
 &\text{A}_1, \quad \text{A}_2, \quad \text{A}_3, \quad \text{A}_4 \\
 &\text{fpga}, \quad \text{fpga}, \quad \text{fpga}, \quad \text{fpga}
 \end{aligned}$$

Program Transformations

$$\begin{aligned}
 \underbrace{AB}_{\text{smp}(p,\mu)} &\rightarrow \underbrace{A}_{\text{smp}(p,\mu)} \underbrace{B}_{\text{smp}(p,\mu)} \\
 \underbrace{A_m \otimes \text{I}_n}_{\text{smp}(p,\mu)} &\rightarrow \underbrace{(\text{L}_m^{mp} \otimes \text{I}_{n/p}) (\text{I}_p \otimes (A_m \otimes \text{I}_{n/p})) (\text{L}_p^{mp} \otimes \text{I}_{n/p})}_{\text{smp}(p,\mu)} \\
 \underbrace{\text{L}_m^{mn}}_{\text{smp}(p,\mu)} &\rightarrow \begin{cases} (\text{I}_p \otimes \text{L}_m^{mn/p}) (\text{L}_p^{pn} \otimes \text{I}_{m/p}) \\ \text{smp}(p,\mu) & \text{smp}(p,\mu) \\ (\text{L}_m^{pm} \otimes \text{I}_{n/p}) (\text{I}_p \otimes \text{L}_m^{mn/p}) \\ \text{smp}(p,\mu) & \text{smp}(p,\mu) \end{cases} \quad \text{Recursive rules} \\
 \underbrace{\text{I}_m \otimes A_n}_{\text{smp}(p,\mu)} &\rightarrow \text{I}_p \otimes_{\parallel} (\text{I}_{m/p} \otimes A_n) \\
 \underbrace{(P \otimes \text{I}_n)}_{\text{smp}(p,\mu)} &\rightarrow (P \otimes \text{I}_{n/\mu}) \otimes \text{I}_{\mu} \quad \text{Base case rules}
 \end{aligned}$$

Autotuning in Constraint Solution Space



Translating an OL Expression Into Code

Constraint Solver Input: $\underbrace{\text{DFT}}_8$
AVX(2-way C)

Output =

Ruletree, expanded into

OL Expression:

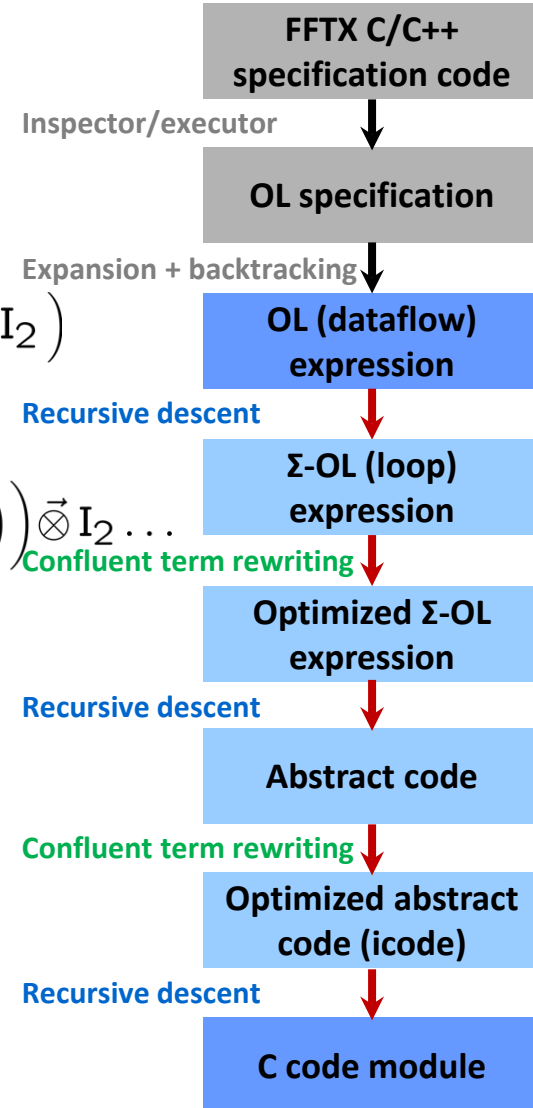
$$\left((F_2 \otimes I_2) T_2^4 (I_2 \otimes F_2) L_2^4 \vec{\otimes} I_2 \right) \underbrace{T_2^8}_{\text{vec}(2)} \left(I_2 \otimes \underbrace{L_2^4}_{\text{vec}(2)} (F_2 \vec{\otimes} I_2) \right) (L_2^4 \vec{\otimes} I_2)$$

Σ -OL:

$$\left(\sum_{j=0}^1 \left(S_{i_2 \otimes (j)_2} F_2 \text{Map}_{x \mapsto \omega_4^{2i+j_x}} G_{i_2 \otimes (j)_2} \right) \sum_{j=0}^1 \left(S_{(j)_2 \otimes i_2} F_2 G_{i_2 \otimes (j)_2} \right) \right) \vec{\otimes} I_2 \dots$$

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;
}
```



SPIRAL 8.4.0: Available Under Open Source

- Release of FFTX 1.0.0
- Open Source SPIRAL available
 - non-viral license (BSD)
 - Initial version, effort ongoing to open source whole system
 - Commercial support via SpiralGen, Inc.
- Tutorial material available online

```

Spiral

http://www.spiralgen.com
Spiral 8.0.0

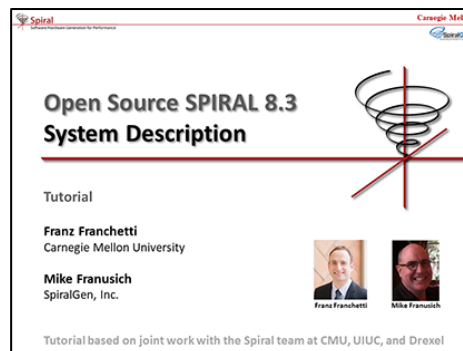
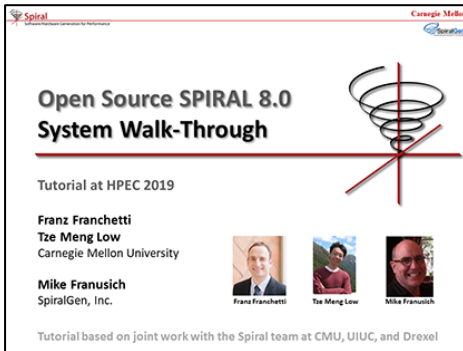
...
PID: 17108

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
SpiralDefault
SpiralVer
PrintCode("dft8", CodeRuleTree(rt, SpiralDefault, SpiralDefault));

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

```

www.spiral.net



■ FFX and SPIRAL available via GitHub

spiral-software	8.4.0-release	https://github.com/spiral-software/spiral-software
spiral-package-fftx	1.1.0-release	https://github.com/spiral-software/spiral-package-fftx
spiral-package-simt	1.1.0-release	https://github.com/spiral-software/spiral-package-simt
spiral-package-mpi	1.0.0-release	https://github.com/spiral-software/spiral-package-mpi
FFTX	1.0.0-release	https://github.com/spiral-software/fftx