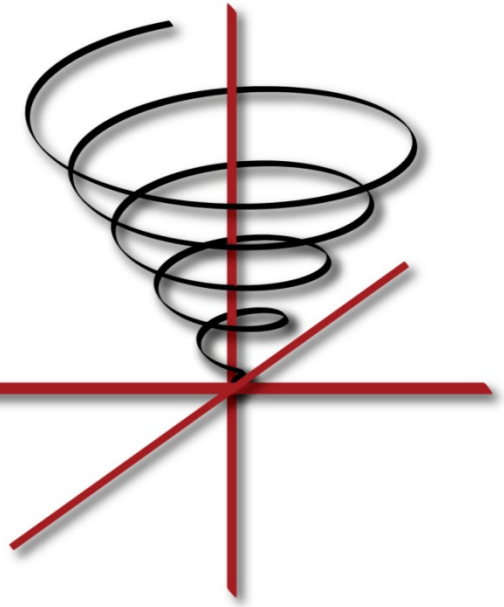
Updates on FFTX and NTTX



Het Mankad
Carnegie Mellon University

in collaboration with
Franz Franchetti, Naifeng Zhang, Tze Meng Low, Sanil Rao
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

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
…

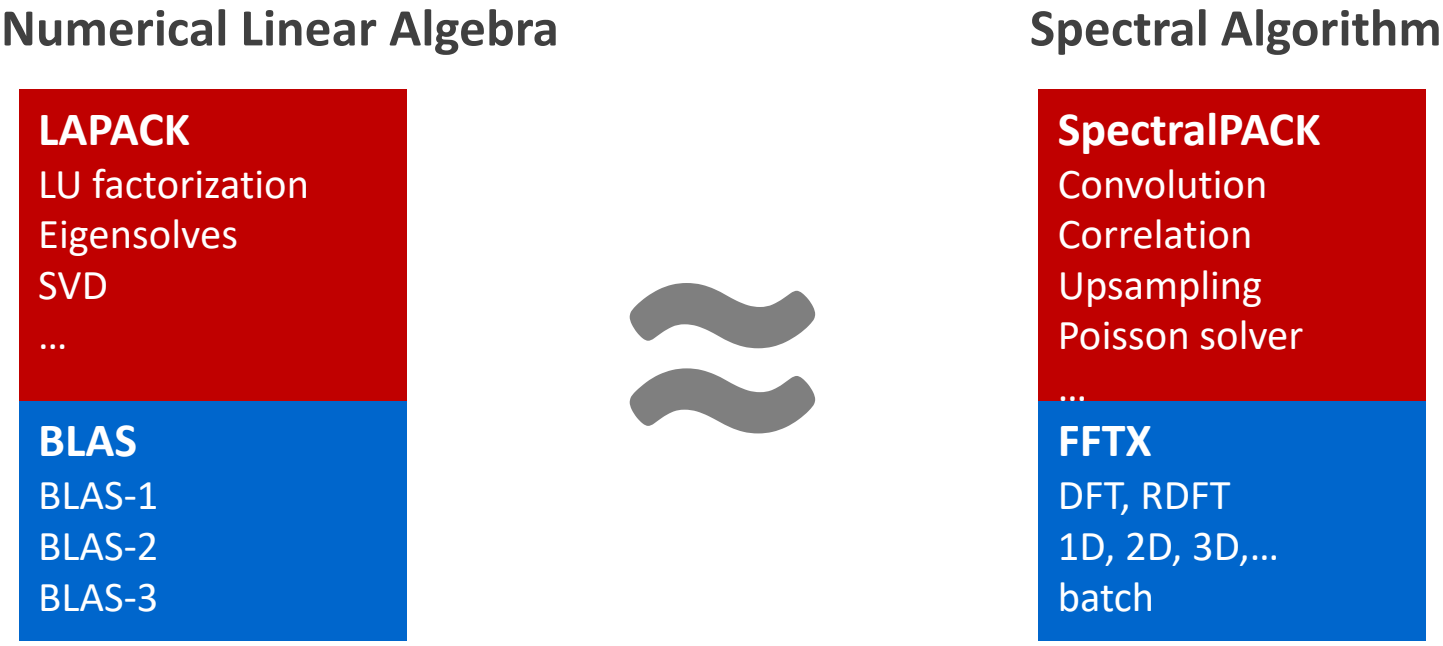
?

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



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)

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

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

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

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)

Method of Local Corrections (MLC)

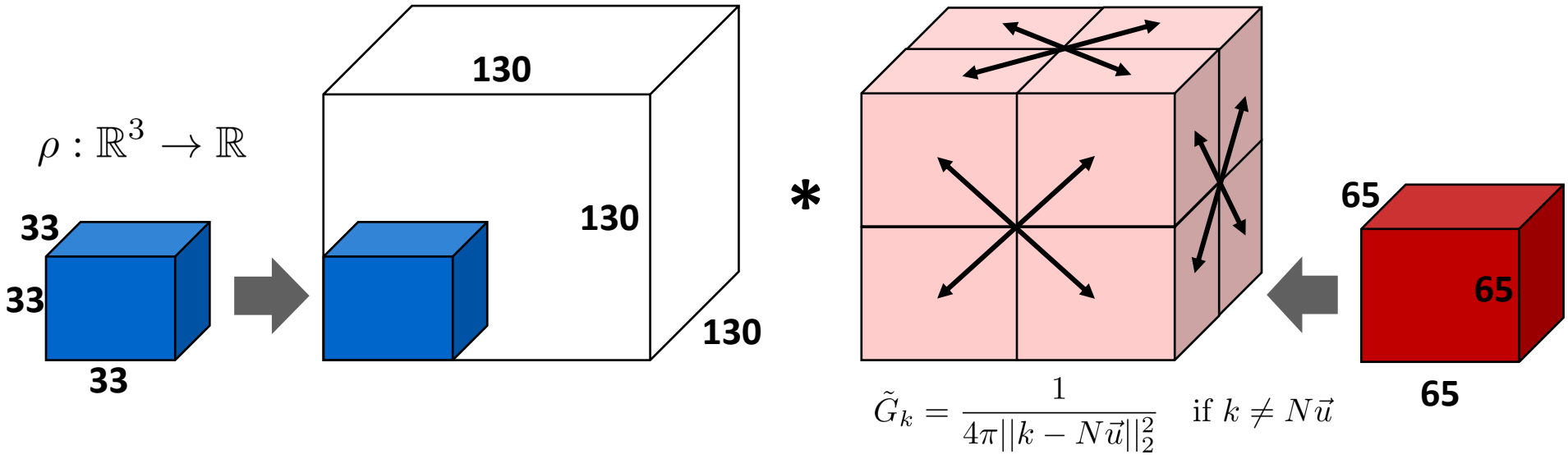
$$\tilde{G}_k = \frac{1}{4\pi||k - N\vec{u}||_2^2} \quad \text{if } k \neq N\vec{u}$$

Green's function kernel in frequency domain

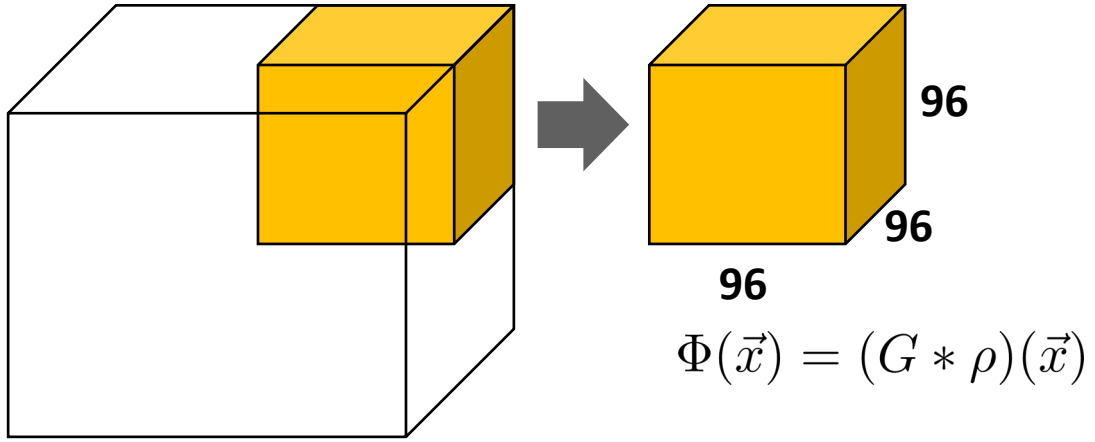
P. McCorquodale, P. Colella, G. T. Balls, and S. B. Baden: **A Local Corrections Algorithm for Solving Poisson's Equation in Three Dimensions.** Communications in Applied Mathematics and Computational Science Vol. 2, No. 1 (2007), pp. 57-81., 2007.

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



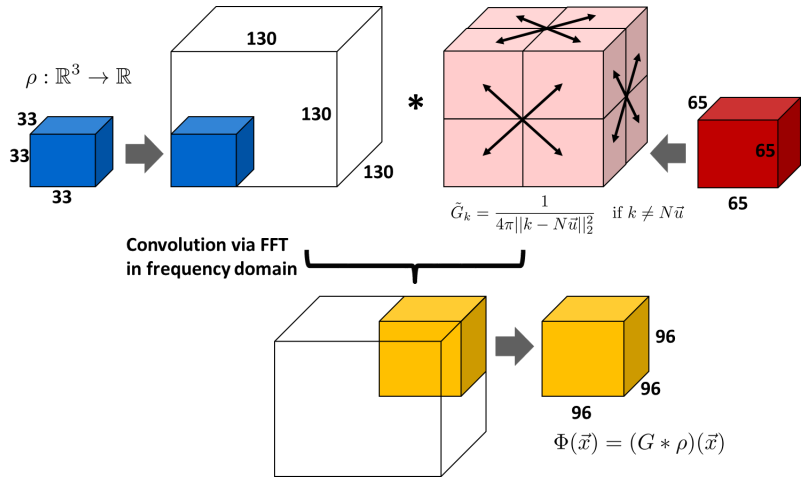
Convolution via FFT in frequency domain



Hockney: Convolution + problem specific zero padding and output subset

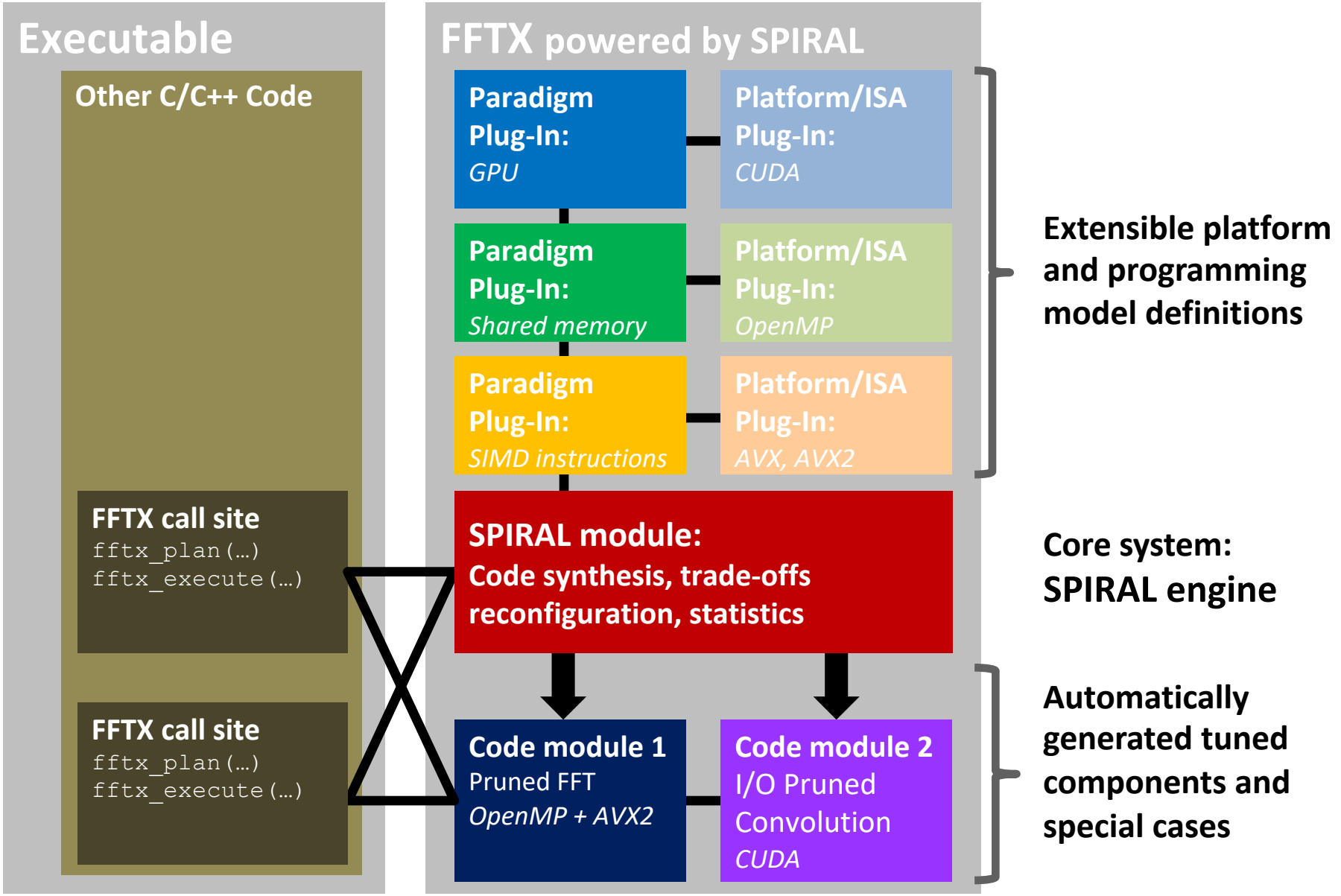
3D Convolution in FFTX/C++

```
#include "fftx3.hpp"
...
int main(int argc, char* argv[])
{
  int nx, ny, nz;
  box_t<3> domain(point_t<3>({{1,1,1}}), point_t<3>({{nx,ny,nz}}));

  array_t<3,std::complex<double>> input(domain);
  array_t<3,std::complex<double>> symbol(domain);
  array_t<3,std::complex<double>> output(domain);
  std::array<array_t<3,std::complex<double>>,2> temp {domain};

  MDDFT(domain.extents(), temp[0], input);
  RCDiag(domain.extents(), temp[1], temp[0], symbol);
  IMDDFT(domain.extents(), output, temp[1]);
}
```



Shown as source code, implemented as delayed execution in via run time compilation

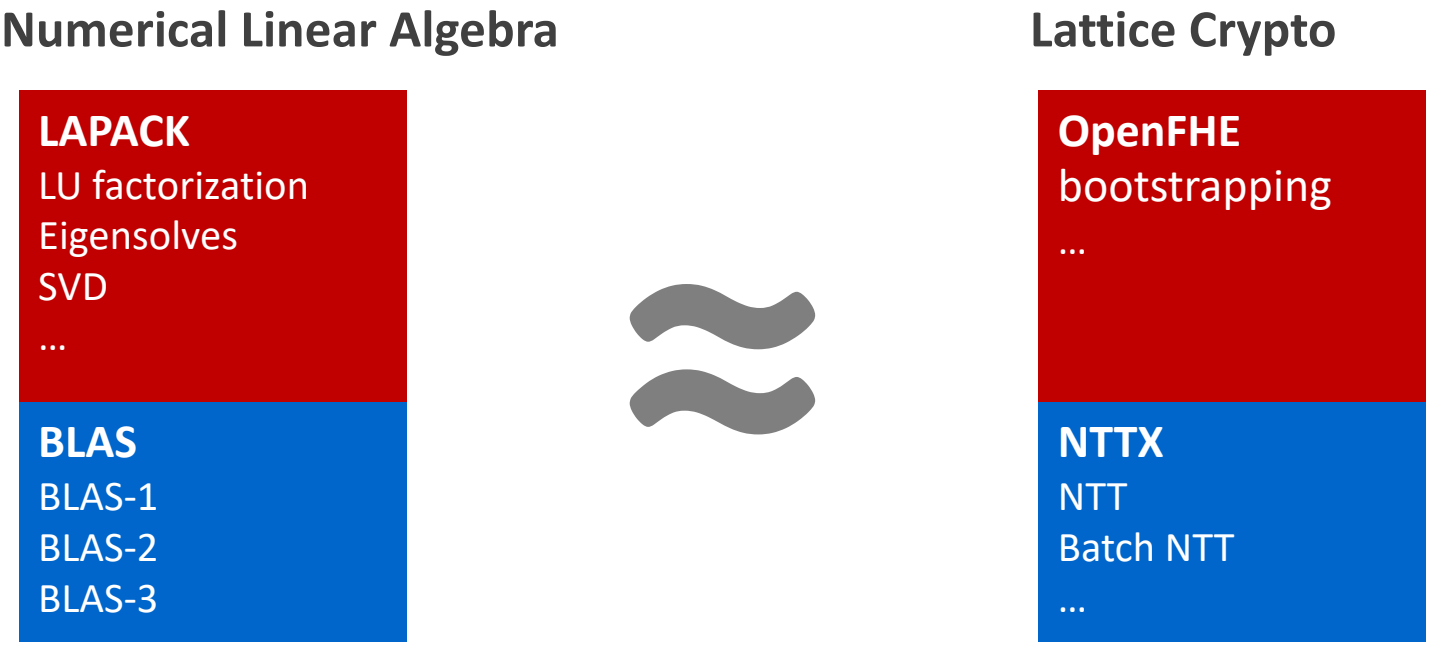
FFTX Backend: SPIRAL



In Development: Delayed Execution in FFTX C++ Code

```
#include "fftx3.hpp"
...
int main(int argc, char* argv[])
{
  int nx, ny, nz;
  box_t<3> domain(point_t<3>({{1,1,1}}), point_t<3>({{nx,ny,nz}}));

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

  // will not be materialized
  fftx::tmp_array<array_t<3,std::complex<double>>,2> temp {domain};
  // no-op, just collect calling parameters, delayed execution
  MDDFT(domain.extents(), temp[0], input);
  // no-op, just collect calling parameters, delayed execution
  RCDiag(domain.extents(), temp[1], temp[0], symbol);
  // passing in the output triggers RTC and execution of the entire delayed sequence
  IMDDFT(domain.extents(), output, temp[1]);
  // output now contains the correct result, but temp[] was never materialized
}
```

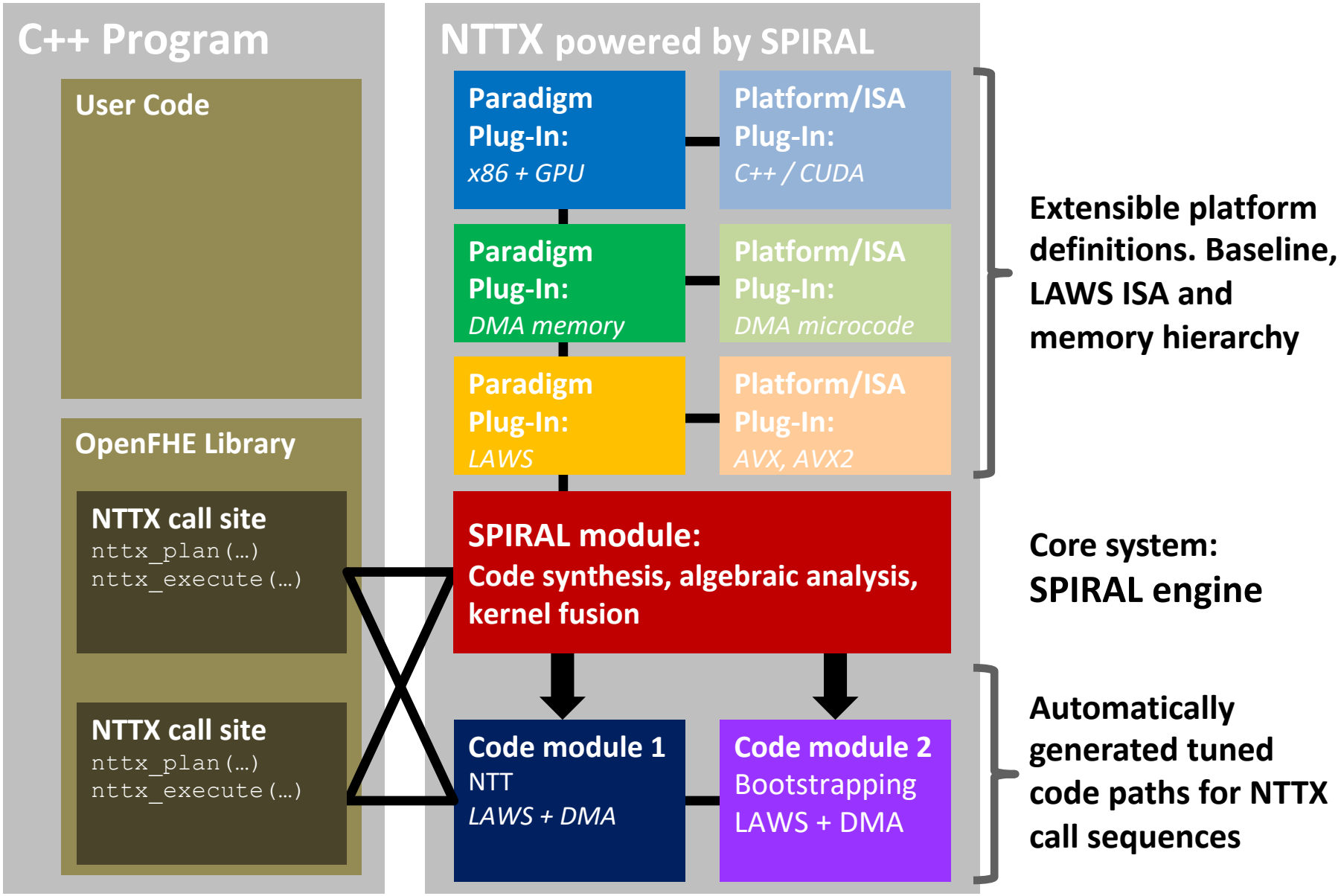
NTTX and OpenFHE



NTTX as lower layer for OpenFHE

- **Define NTTX as the BLAS/FFTW equivalent**
  platform independent high performance NTTs
- **Define higher level cross kernel operations**
  imul/imod, bootstrapping etc. expressed as composable NTTX calling sequence
- **Library front-end, SPIRAL code generation backend**
  mirror concepts from FFTX and GBTLX:
  library API + code gen = performance portability

NTTX and SPIRAL



NTT in NTTX/C++

```
// C/C++ NTTX API example: compute a singel NTT
#include "nttx.h"
nttx_plan *p;
p = nttx_plan_ntt(in, out, n, modulus, NTTX_FORWARD);
nttx_execute(p);
nttx_free(p);
```

```
#include <rpu.h>
void _ntt1024x512_b1() {
    enter(OP_DEFAULT);
    _vload_512x128i(REG_V60, REG_A1, 0);
    _vload_512x128i(REG_V20, REG_A1, 8192);
    _vbroadcast_512x128i(REG_V19, REG_A3, 1, 1);
    _vimulmod_512x128i(REG_V59, REG_V20, REG_V19, REG_M1);
    _vaddmod_512x128i(REG_V58, REG_V60, REG_V59, REG_M1);
    _vsubmod_512x128i(REG_V57, REG_V60, REG_V59, REG_M1);
    _vunpacklo_512x128i(REG_V56, REG_V58, REG_V57);
    _vunpackhi_512x128i(REG_V18, REG_V58, REG_V57);
    _vbroadcast_512x128i(REG_V17, REG_A3, 2, 2);
    _vimulmod_512x128i(REG_V55, REG_V18, REG_V17, REG_M1);
    ...
    _vstores_512x128i(REG_A2, 16, REG_V21, 2);
    leave(OP_DEFAULT);
}
```

Radix-2 1,024-point NTT code using shuffle instructions for ASIC

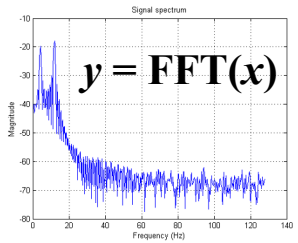
SPIRAL: Go from Mathematics to Software

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

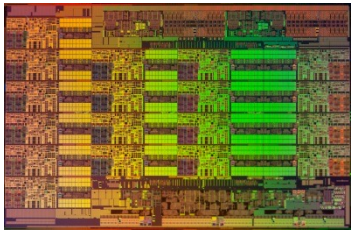
+

PROOF

QED.

SPIRAL's Target Computing Landscape

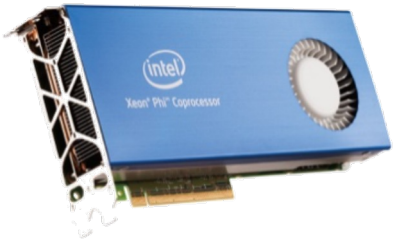
1 Gflop/s = one billion floating-point operations (additions or multiplications) per second



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

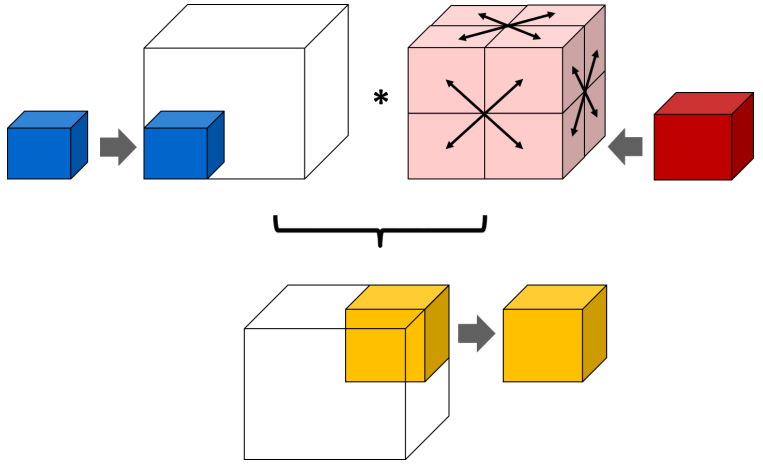
Summit
187.7 Pflop/s, 13 MW
9,216 x 22 cores POWER9
+ 27,648 V100 GPUs

Rules in Internal Domain Specific Language

Linear Transforms

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

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)

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

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

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

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

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

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

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

Program Transformations

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

Recursive rules

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

Base case rules

Translating an OL Expression Into Code

Constraint Solver Input: $\underbrace{DFT_8}_{AVX(2\text{-way } \mathbb{C})}$

Output =

Ruletree, expanded into

OL Expression:

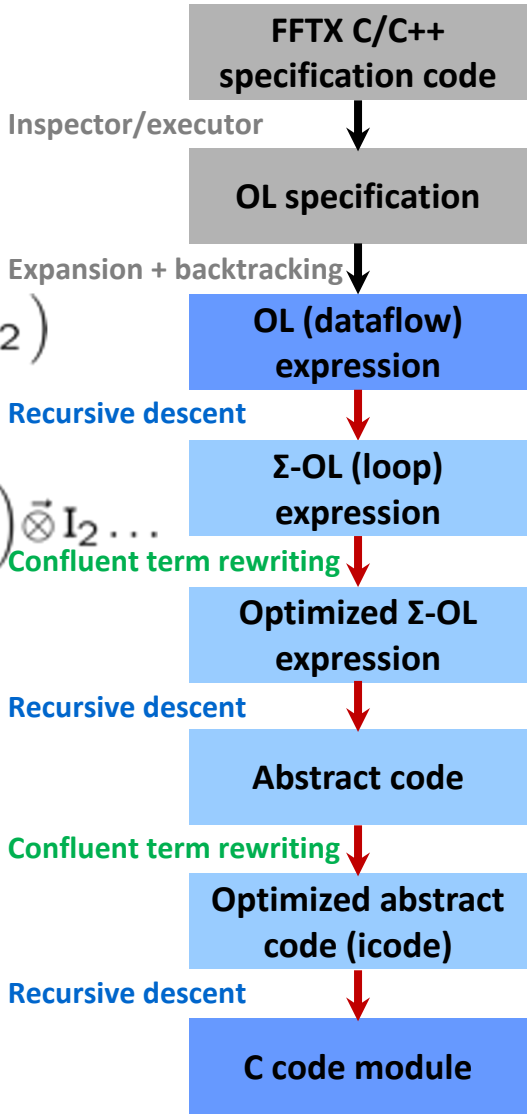
$$\left((F_2 \otimes I_2) T_2^4 (I_2 \otimes F_2) L_2^4 \vec{\otimes} I_2\right) \underbrace{T_2^8}_{vec(2)} \left(I_2 \otimes \underbrace{L_2^4}_{vec(2)} (F_2 \vec{\otimes} I_2)\right)\left(L_2^4 \vec{\otimes} I_2\right)$$

∑-OL:

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

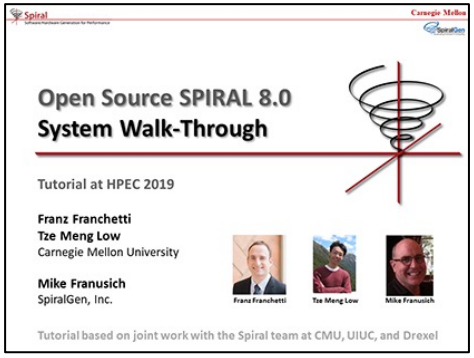
C Code:

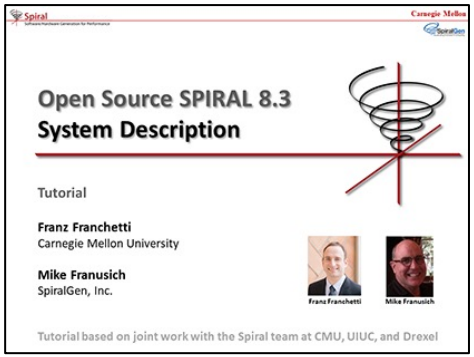
```
void dft8(_Complex double *Y, _Complex double *X) {
    __m256d s38, s39, s40, s41,...
    __m256d  *a17, *a18;
    a17 = ((__m256d  *) X);
    s38 = *(a17);
    s39 = *((a17 + 2));
    t38 = _mm256_add_pd(s38, s39);
    t39 = _mm256_sub_pd(s38, s39);
    ...
    s52 = _mm256_sub_pd(s45, s50);
 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



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**





- **FFTX 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