# FFTX: Release, Updates and Next Steps

**Sanil Rao**
Carnegie Mellon University

*in collaboration with*
**Franz Franchetti, Het Mankad, Naifeng Zhang, Tze Meng Low**
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 and X-Stack Bluestone

# FFTX Released

spiral.net/software/fftx.html

Spiral
Software/Hardware Generation for Performance

Home | Generator | Benchmarks | Publications | Software | Hardware | Grants | Team | Related | Internal

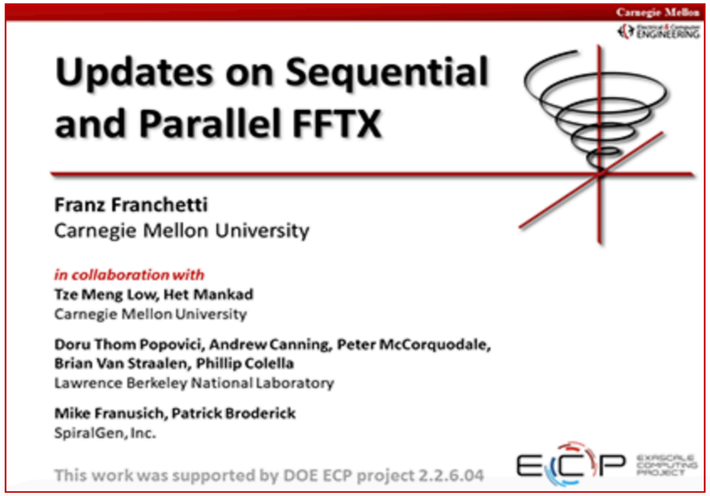
## FFTX and SPIRAL

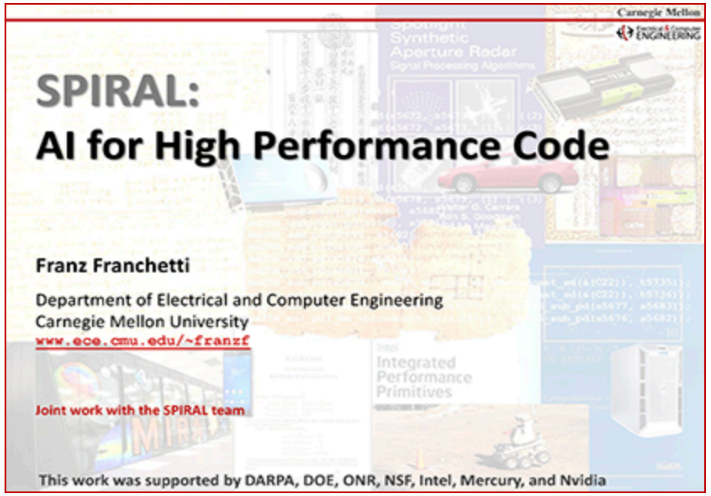
**(DE-AC02-05CH11231 — DOE FFTX)**

FFTX is the exascale follow-on to the FFTW open source discrete FFT package for executing the Fast Fourier Transform as well as higher-level operations composed of linear operations combined with DFT transforms. Though backwards compatible with FFTW, this is an entirely new work developed as a cooperative effort between Lawrence Berkeley National Laboratory, Carnegie Mellon University, and SpiralGen, Inc.

From casual look at its API, FFTX appears to be a pre-built library, but the heart of FFTX is a build-time code generator, SPIRAL, that produces very high performance kernels targeted to their specific uses and platform environments. Coupled to the platform-aware code generator is a sophisticated front end that interprets the details of the algorithms from the FFTX API, which it treats as a DSL for algorithm specification.

### Development



# 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**<br>LU factorization<br>Eigensolves<br>SVD<br>… | ≈ | **SpectralPACK**<br>Convolution<br>Correlation<br>Upsampling<br>Poisson solver<br>… |
| **BLAS**<br>BLAS-1<br>BLAS-2<br>BLAS-3 | | **FFTX**<br>DFT, RDFT<br>1D, 2D, 3D,…<br>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: Free Space Convolution in C++

## Numerical algorithm view

**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**

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

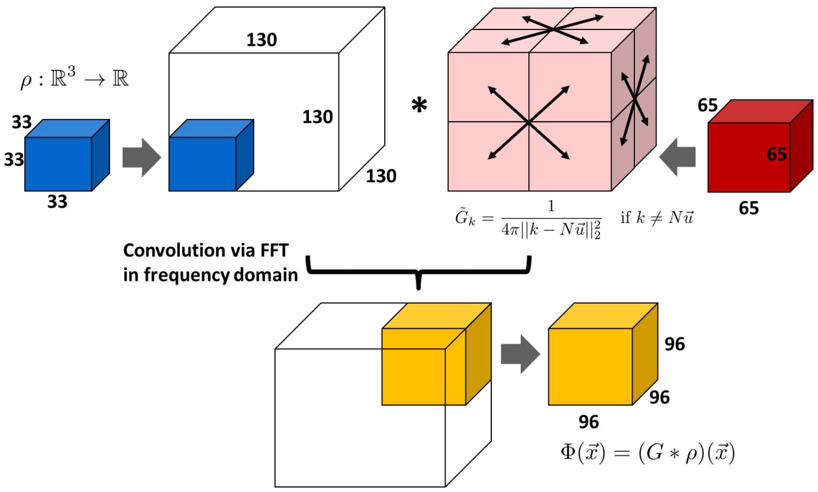
Solution: ϕ(.) = convolution of RHS ρ(.) with Green's function *G*(.). Efficient through FFTs (frequency domain)

**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

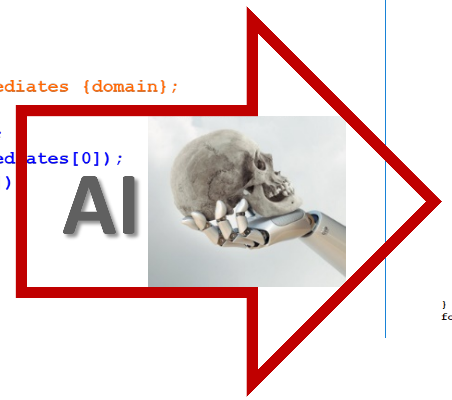
## Whiteboard view




## User program view

```
#include "fftx3.hpp"
...
int main(int argc, char* argv[])
{
  tracing=true;
  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>> inputs(domain);
  array_t<3,std::complex<double>> outputs(domain);
  std::array<array_t<3,std::complex<double>>,2> intermediates {domain};

  MDDFT(domain.extents(), 1, intermediates[0], inputs);
  RCDiag(domain.extents(), 1, intermediates[1], intermediates[0]);
  IMDDFT(domain.extents(), 1, outputs, intermediates[1]);
}
```

"classical compiler"




## Backend program view

hockney_130_33_96.c - Notepad

File Edit Format View Help

```
                a2370 = D43[(b416 + 2)];
                a2371 = D43[(b416 + 3)];
                T113[(b415 + 26)] = ((a2368*t2146) - (a2369*t2147));
                T113[(b415 + 27)] = ((a2369*t2146) + (a2368*t2147));
                T113[(b415 + 104)] = ((a2370*t2148) - (a2371*t2149));
                T113[(b415 + 105)] = ((a2371*t2148) + (a2370*t2149));
                t2150 = (s1076 - s1080);
                t2151 = (s1077 + s1081);
                t2152 = (s1076 + s1080);
                t2153 = (s1077 - s1081);
                a2372 = D43[(b416 + 4)];
                a2373 = D43[(b416 + 5)];
                a2374 = D43[(b416 + 6)];
                a2375 = D43[(b416 + 7)];
                T113[(b415 + 52)] = ((a2372*t2150) - (a2373*t2151));
                T113[(b415 + 53)] = ((a2373*t2150) + (a2372*t2151));
                T113[(b415 + 78)] = ((a2374*t2152) - (a2375*t2153));
                T113[(b415 + 79)] = ((a2375*t2152) + (a2374*t2153));
            }
        }
        for(int i68 = 0; i68 <= 64; i68++) {
            double s1094, s1095, s1096, s1097;
            int a2406, a2407;
            a2406 = (2*i68);
            s1094 = T113[a2406];
            s1095 = T113[(a2406 + 1)];
            s1096 = T113[(a2406 + 130)];
            s1097 = T113[(a2406 + 131)];
            a2407 = ((260*i18) + a2406);
            T112[a2407] = (s1094 + s1096);
            T112[(a2407 + 1)] = (s1095 + s1097);
            T112[(a2407 + 130)] = (s1094 - s1096);
            T112[(a2407 + 131)] = (s1095 - s1097);
        }
    }
    for(int i17 = 0; i17 <= 129; i17++) {
        static double T179[260];
```

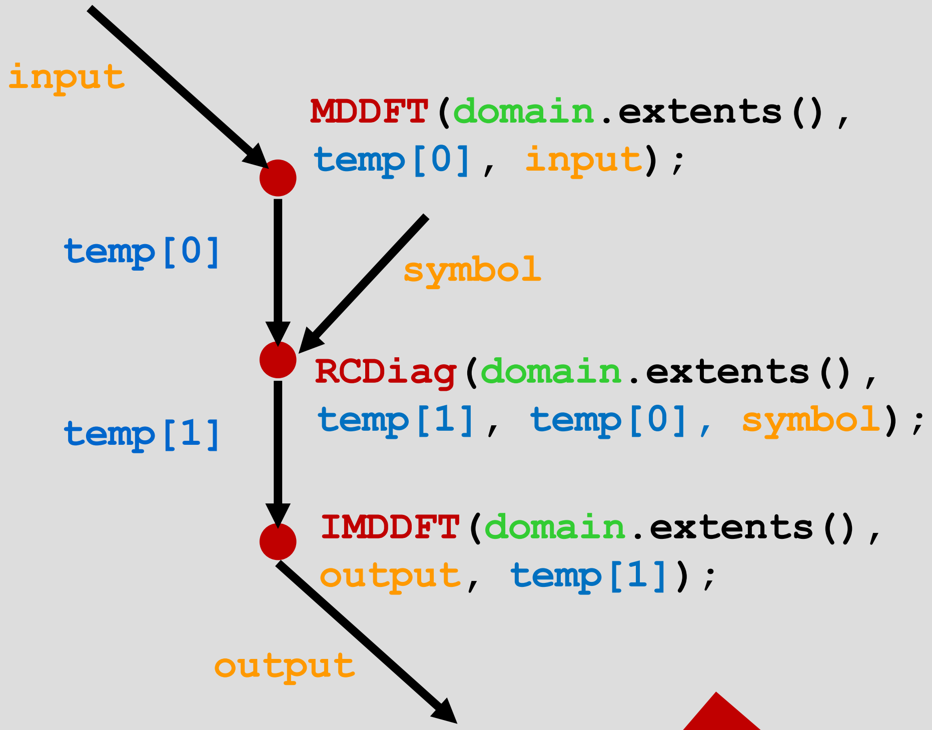
# Method 1: Delayed Execution in FFTX C++ Code

```
#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
  MDDFT(domain.extents(), output, temp[1]);
  // output now contains the correct result, but temp[] was never materialized
}
```

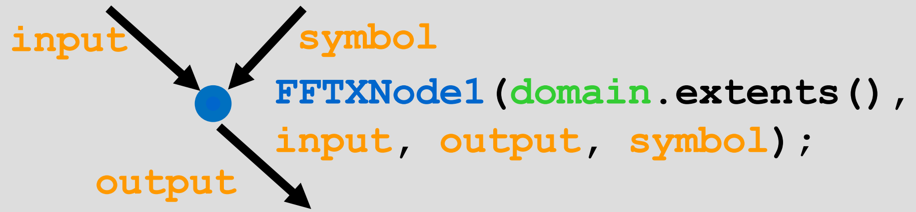
**Captures library semantics and replaces with generated high-performance kernel through runtime compilation**

# Method 2: 3D Convolution FFTX DAG Transformation

## MLIR DAG



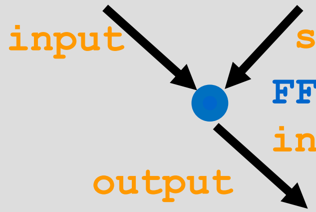

## MLIR DAG + FFTXNode Supernodes



# Method 2: FFTXNode Inserted in FFTX C++ Code/MLIR

```
#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);

  // original FFTX code
  // captured as FFTXNode1 by high level MLIR transformation
  // 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]);
  // compiler generated FFTXNode replacement code
  // injected at LLVM/MLIR level
  FFTXNode1.execEnv(DEFAULT).run(domain.extents(), input, output, symbol);
}
```

## FFTXNode Supernode




```
FFTXNode1(domain.extents(),
input, output, symbol);
```

# FFTXNode Runtime Compilation

```
// kernels as strings for the CUDA JIT
kernels[2] = { "__global__ void ker_mdprdft3d_4x4x42(double  *Y) {\n"
"    double s100, s101, s102, s97, s98, s99; \n"
"    int a191, a192; \n"
"    a191 = ((24*blockIdx.y) + (6*blockIdx.x)); \n"
"    s97 = P2[a191]; \n"
"    s98 = P2[(a191 + 4)]; \n"
"    s99 = (s97 + s98); \n"
"    s100 = (s97 - s98); \n"
"    s101 = (2.0*P2[(a191 + 2)]); \n"
"    s102 = (2.0*P2[(a191 + 3)]); \n"
"    a192 = ((16*blockIdx.y) + (4*blockIdx.x)); \n"
"    Y[a192] = (s99 + s101); \n"
"    Y[(a192 + 2)] = (s99 - s101); \n"
"    Y[(a192 + 1)] = (s100 + s102); \n"
"    Y[(a192 + 3)] = (s100 - s102); \n"
"} \n", 1, DOUBLE_PY};
```

FFTX generated output sent to target runtime compilation system

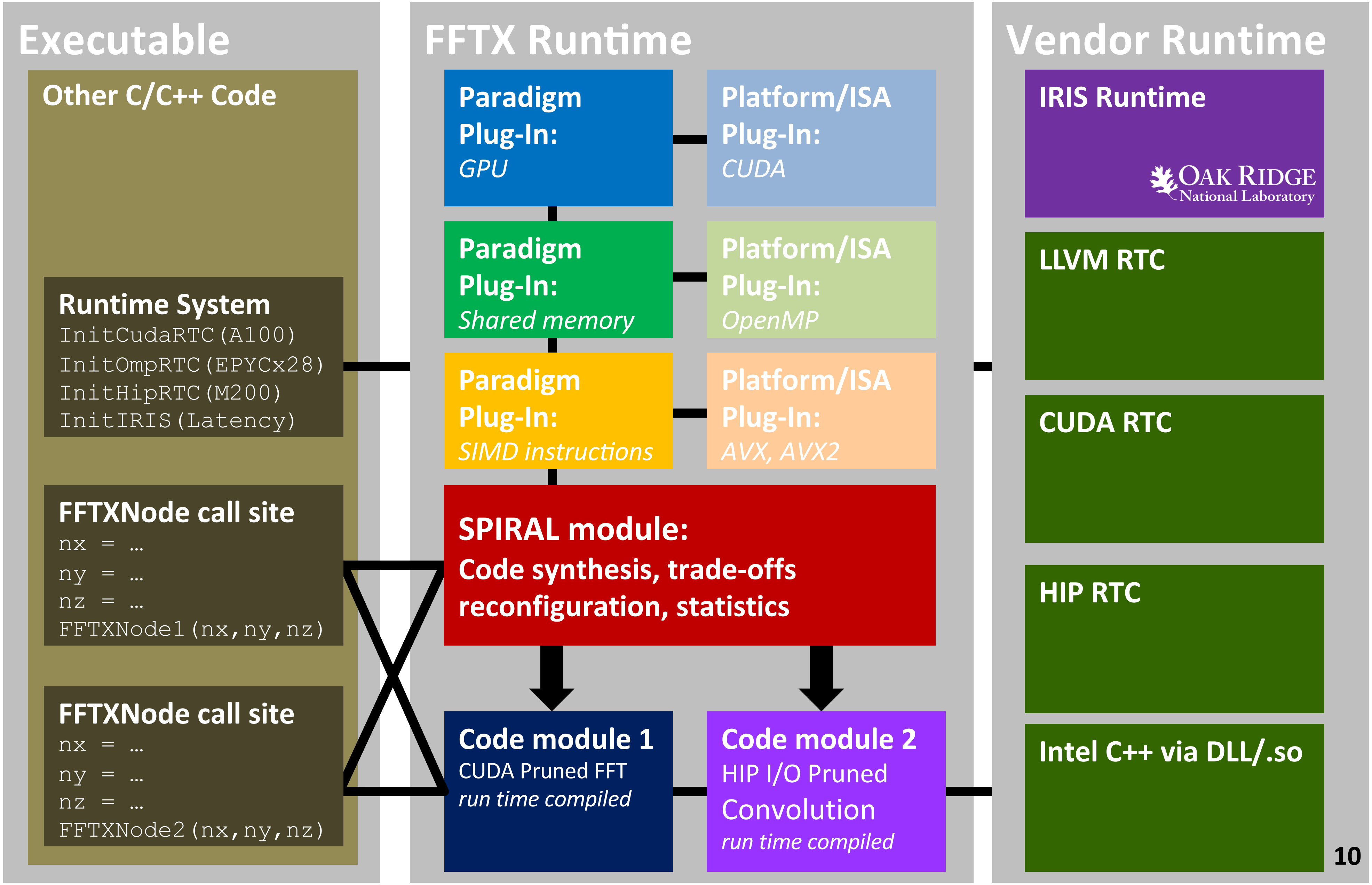
```
// main transform function encoded as integer table
transform = {Conv3D, 3, {4, 4, 4}, 3,
    { kernels[0], dim3(4, 3, 1), dim3(1, 1, 1), 1, DOUBLE_PX },
    { kernels[1], dim3(4, 3, 1), dim3(1, 1, 1), 0 },
    { kernels[2], dim3(4, 4, 1), dim3(1, 1, 1), 1, DOUBLE_PY }};
```

FFTX output for small SPIRAL interpreter

```
FFTXNode1.run(double  *Y, double  *X, double *sym) {
    // one-time JITs kernel, interprets and executes integer table
    compileAndExecute(transform, MODE_JIT_CACHE);
}
```

FFTXNode run() method invokes SPIRAL and RTC

# FFTX Runtime



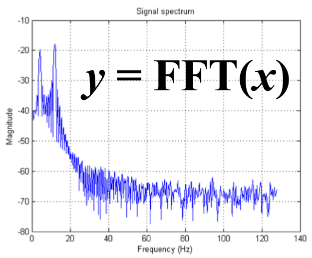
# SPIRAL: Go from Mathematics to Software

## 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**

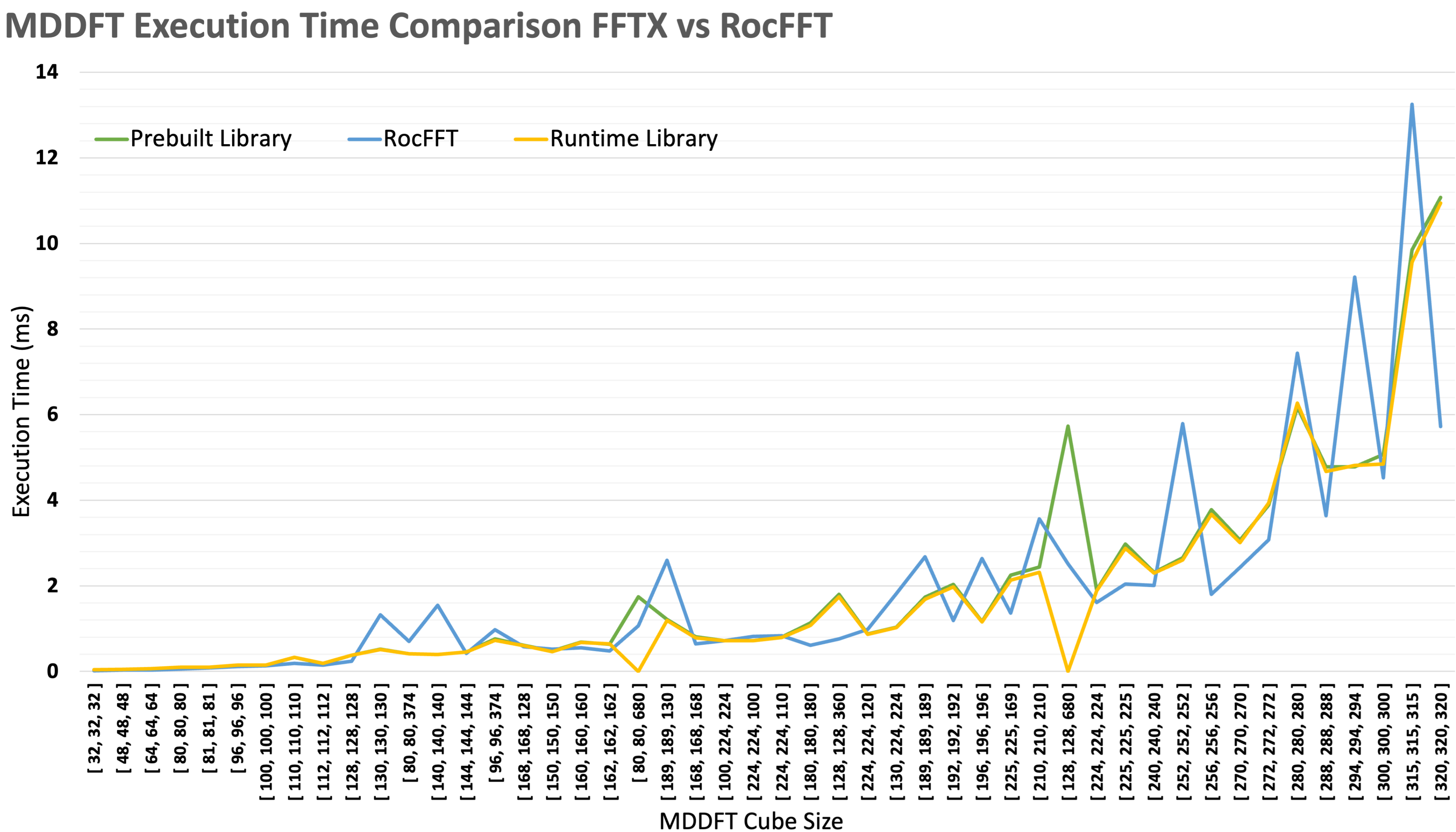
```
void fft64(double  *Y, double  *X) {
    ...
    s5674 = _mm256_permute2f128_pd(s5672, s5673, (0) | ((2) << 4));
    s5675 = _mm256_permute2f128_pd(s5672, s5673, (1) | ((3) << 4));
    s5676 = _mm256_unpacklo_pd(s5674, s5675);
    s5677 = _mm256_unpackhi_pd(s5674, s5675);
    s5678 = *((a3738 + 16));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

+

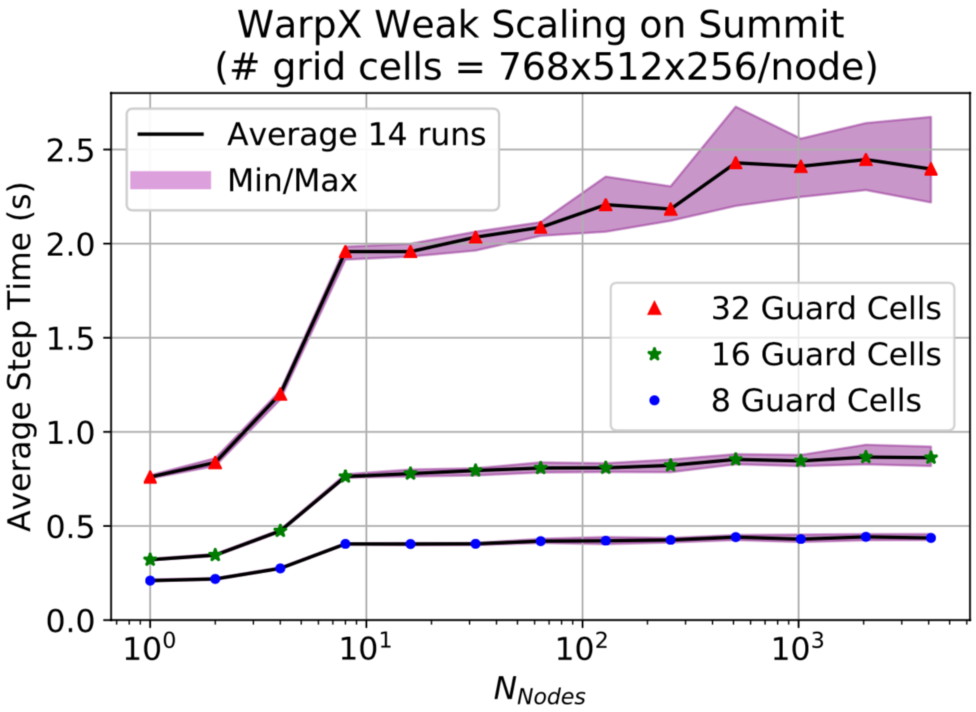

QED.

# FFTX Single Device Results on Frontier

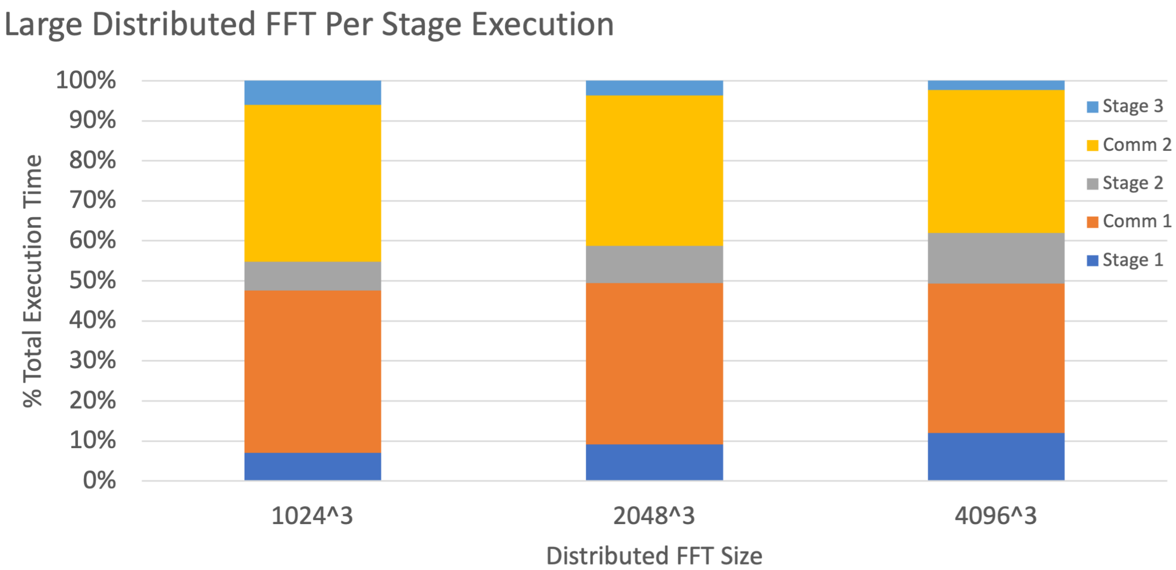



**FFTX is competitive with vendor libraries while having options of prebuilding or runtime compiling kernels**

# FFTX Distributed Results



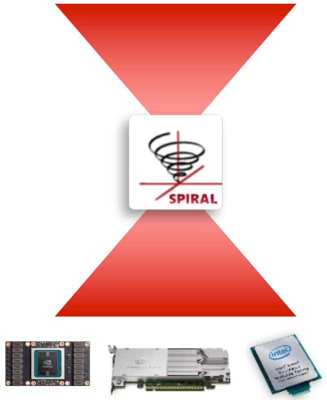

**FFTX integrated with WarpX running at scale**

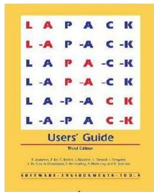


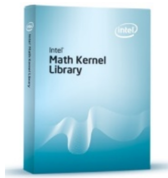

**Communication dominates local computation for large distributed FFTs**

# Proof of Concept: SnowWhite and LibraryX

## Multi-language, Multi target

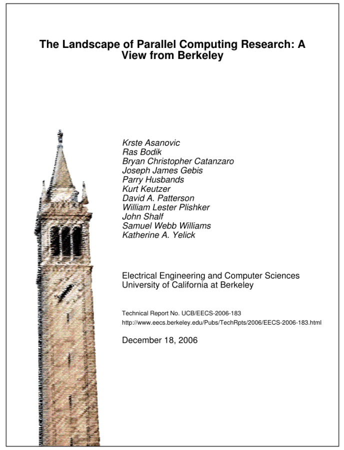



*Powered by SPIRAL*
*code generation/synthesis*

## Cross-"Dwarf" optimization

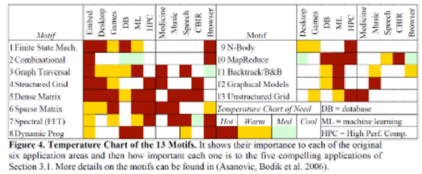

The Landscape of Parallel Computing Research: A View from Berkeley

Krste Asanovic
Ras Bodik
Bryan Christopher Catanzaro
Joseph James Gebis
Parry Husbands
Kurt Keutzer
David A. Patterson
William Lester Plishker
John Shalf
Samuel Webb Williams
Katherine A. Yelick

Electrical Engineering and Computer Sciences
University of California at Berkeley

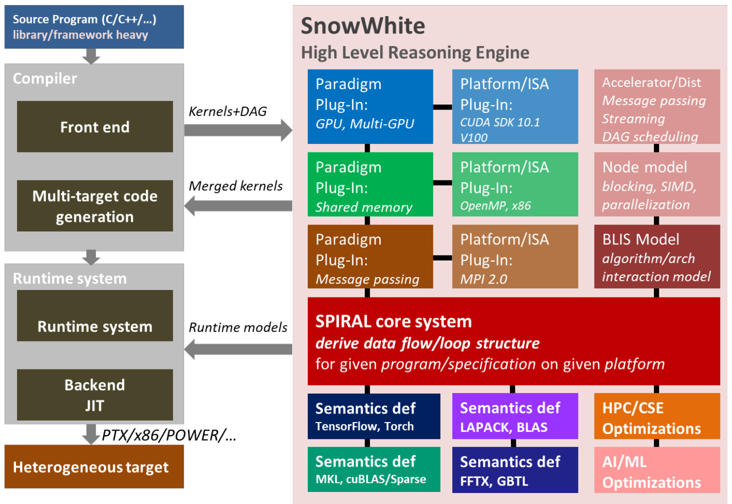
Technical Report No. UCB/EECS-2006-183
http://www.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-183.html

December 18, 2006



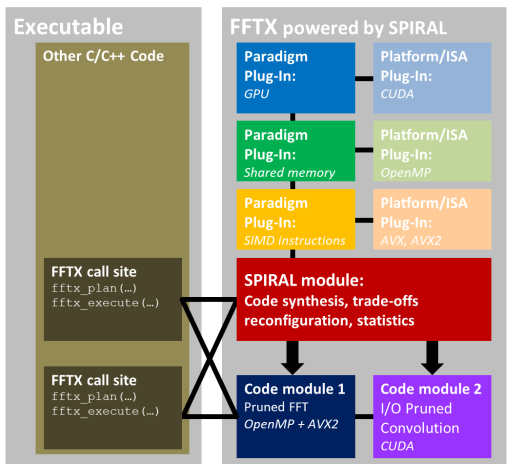
*Cross-call, cross-library, cross-motif*

## SnowWhite: SPIRAL inside compilers



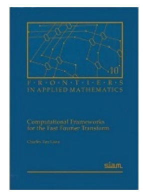

*DARPA PAPPA, X-Stack Bluestone*

## LibraryX, powered by SPIRAL





*GBTLX NTTX*



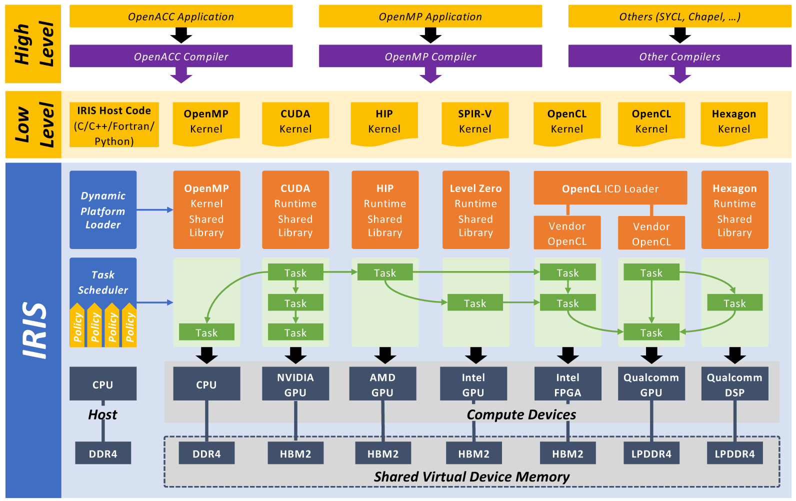
*FFTX (ECP)*

*Multiple active libraries, one infrastructure*

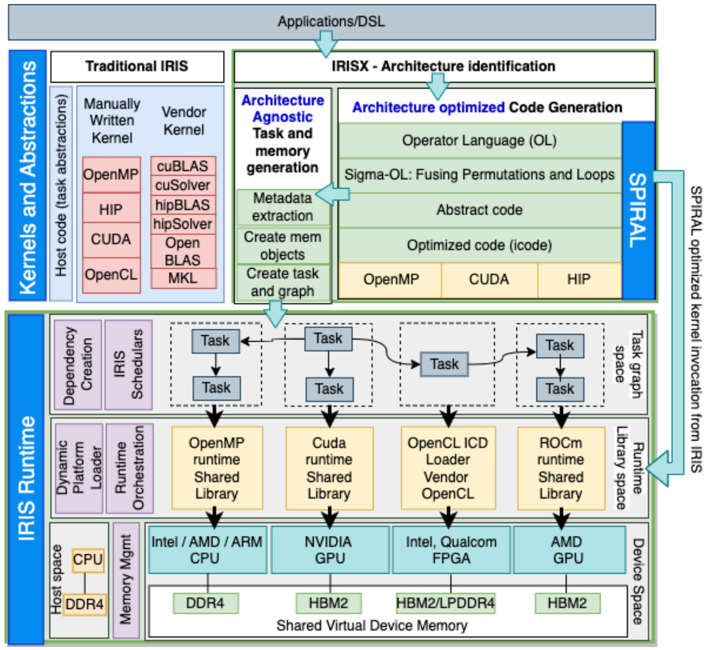
# Preview: IRISX

## Heterogenous Landscape

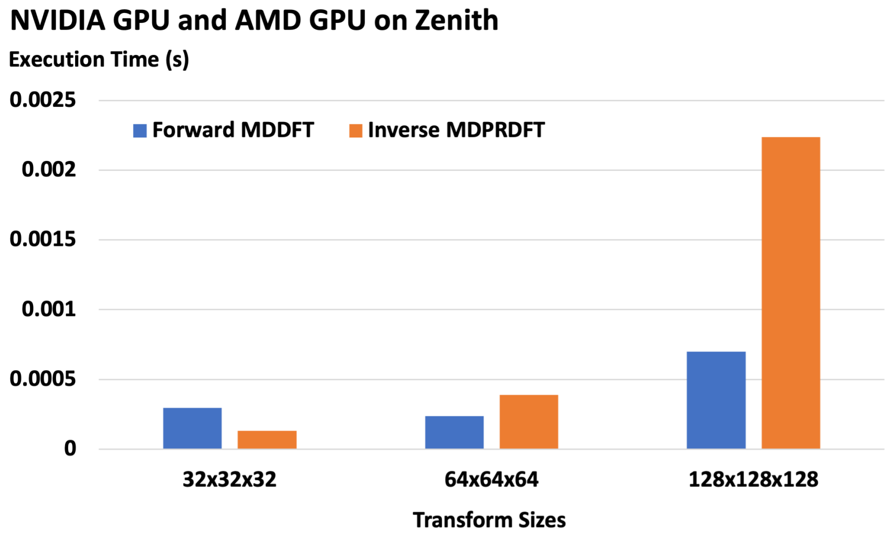
## IRIS: Intelligent Task Scheduling




## IRISX: LibraryX + IRIS




## Automatic Portability and Heterogeneity

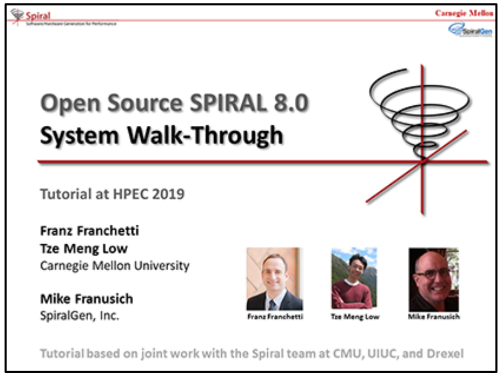


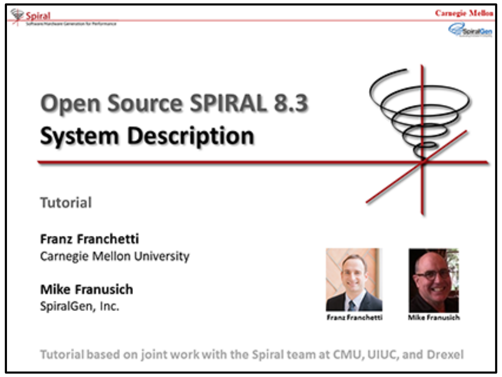

Sanil Rao, Mohammad Alaul Haque Monil, Het Mankad, Jeffrey Vetter, and Franz Franchetti. 2023. FFTX-IRIS: Towards Performance Portability and Heterogeneity for SPIRAL Generated Code. In Proceedings of the SC '23 Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis (SC-W '23).

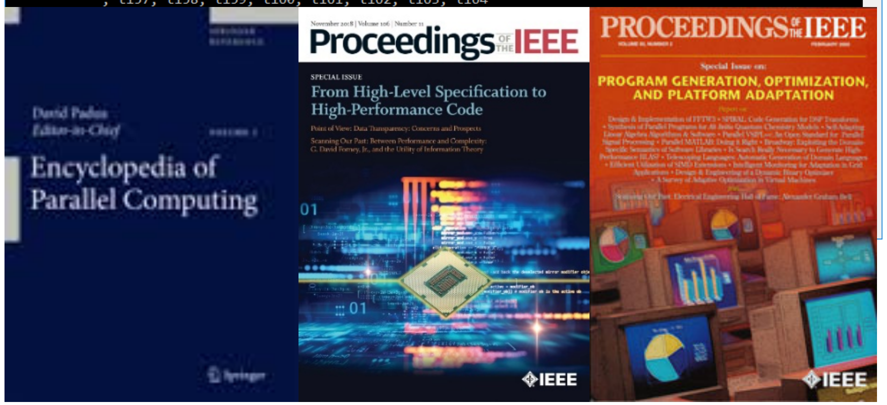
# SPIRAL 8.5.0: Available Under Open Source

- **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 | 8.5.0-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-jit | 1.0.0-release | https://github.com/spiral-software/spiral-package-jit |
| FFTX | 1.0.0-release | https://github.com/spiral-software/fftx |