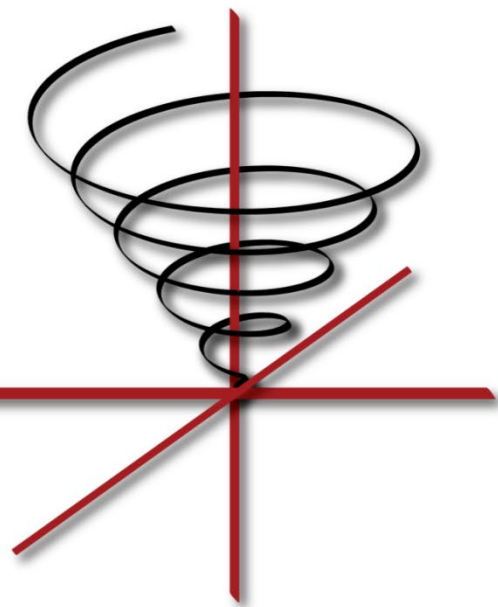


# FFTX and SpectralPACK: A First Look

---



**Franz Franchetti, 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

# It Is Worse Than It Seems

## FFTW is de-facto standard interface for FFT



- **FFTW 3.X is the high performance reference implementation:**  
supports multicore/SMP and MPI, and Cell processor
- **Vendor libraries support the FFTW 3.X interface:**  
Intel MKL, IBM ESSL, AMD ACML (end-of-life), Nvidia cuFFT, Cray LibSci/CRAFFT

## Issue 1: 1D FFTW call is standard kernel for many applications

- **Parallel libraries and applications reduce to 1D FFTW call**  
P3DFFT, QBox, PS/DNS, CPMD, HACC,...
- **Supported by modern languages and environments**  
Python, Matlab,...

## Issue 2: FFTW is slowly becoming obsolete

- FFTW 2.1.5 (still in use, 1997), FFTW 3 (2004) minor updates since then
- Development currently dormant, except for small bug fixes
- No native support for accelerators (GPUs, Xeon PHI, FPGAs) and SIMT
- Parallel/MPI version does not scale beyond 32 nodes

***Risk: loss of high performance FFT standard library***

# FFTX: The FFTW Revamp for ExaScale

## Modernized FFTW-style interface



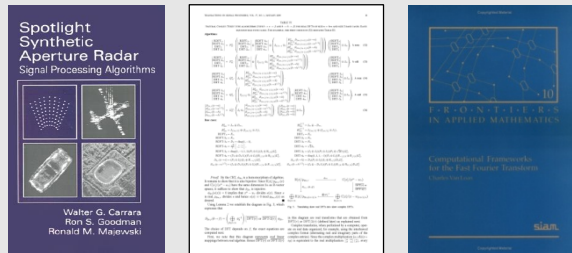
- **Backwards compatible to FFTW 2.X and 3.X**  
old code runs unmodified and gains substantially but not fully
- **Small number of new features**  
futures/delayed execution, offloading, data placement, callback kernels

## Code generation backend using **SPIRAL**

- **Library/application kernels are interpreted as specifications in DSL**  
extract semantics from source code and known library semantics
- **Compilation and advanced performance optimization**  
cross-call and cross library optimization, accelerator off-loading,...
- **Fine control over resource expenditure of optimization**  
compile-time, initialization-time, invocation time, optimization resources
- **Reference library implementation and bindings to vendor libraries**  
library-only reference implementation for ease of development

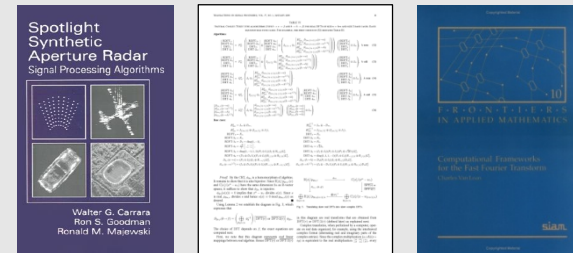
# What is Spiral? Autotuning/Code Generation

## Traditionally



High performance library  
optimized for given platform

## Spiral Approach



High performance library  
optimized for given platform

Comparable  
performance

# FFTX and SpectralPACK: Long Term Vision

## 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,...
- **Define SpectralPACK functions**  
circular convolutions, NUFFT, Poisson solvers, free space convolution,...

***FFTX and SpectralPACK solve the “spectral dwarf” long term***

# Poisson's Equation in Free Space: The Math

## 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.  $\Delta$  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

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

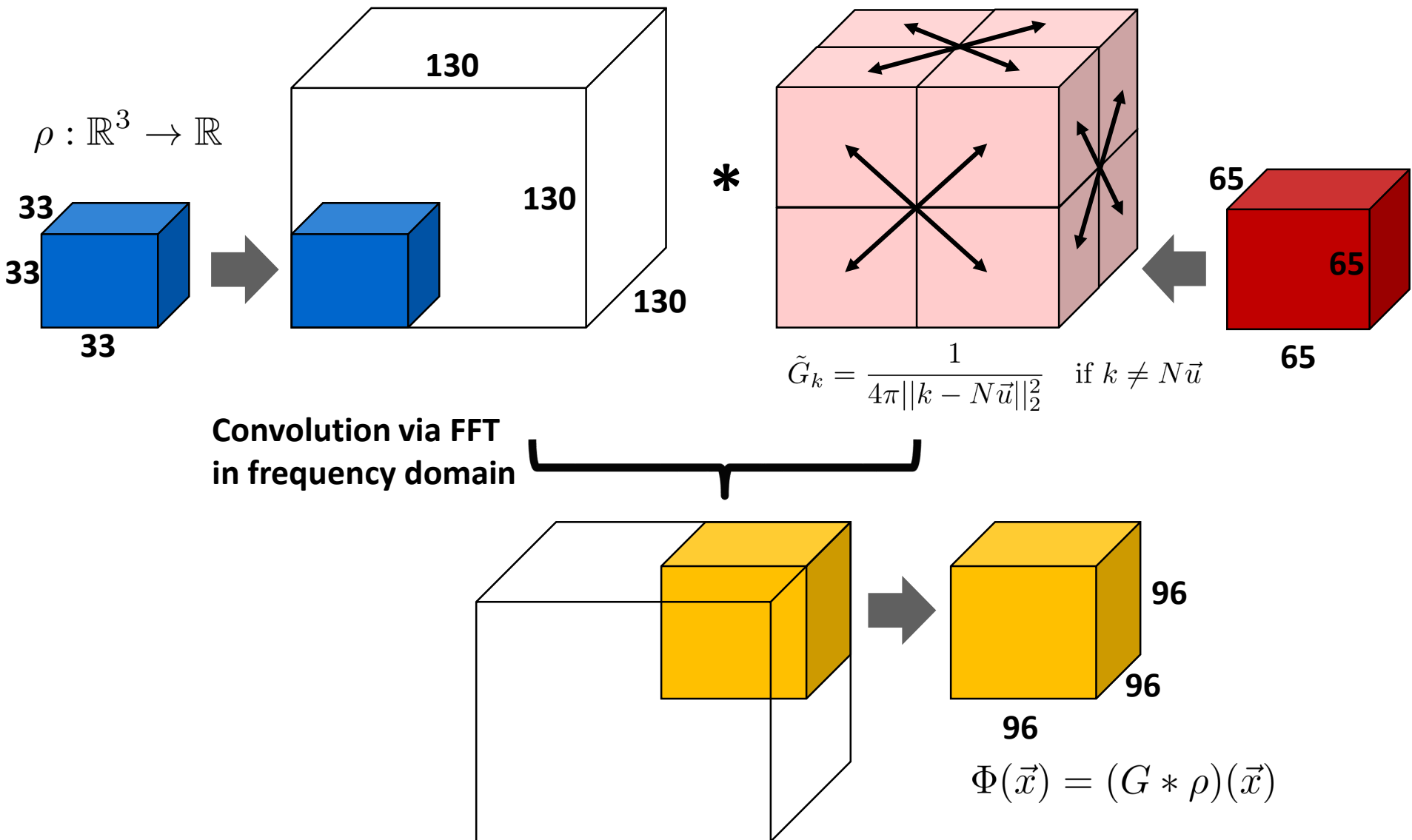
Solution:  $\phi(\cdot)$  = convolution of RHS  $\rho(\cdot)$  with Green's function  $G(\cdot)$ . Efficient through FFTs (frequency domain)

## 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

# Algorithm: Hockney Free Space Convolution



**Hockney: Convolution + problem specific zero padding and output subset**



# FFTX Example: 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];
    fftx_plan p;

    tmp1 = fftx_create_zero_temp_real(rank, &padded_dims);

    plans[0] = fftx_plan_guru_copy_real(rank, &in_dimx, in, tmp1, MY_FFTX_MODE_SUB);

    tmp2 = fftx_create_temp_complex(rank, &freq_dims);
    plans[1] = fftx_plan_guru_dft_r2c(rank, &padded_dims, batch_rank,
        &batch_dims, tmp1, tmp2, MY_FFTX_MODE_SUB);

    tmp3 = fftx_create_temp_complex(rank, &freq_dims);
    plans[2] = fftx_plan_guru_pointwise_c2c(rank, &freq_dimx, batch_rank, &batch_dimx,
        tmp2, tmp3, symbol, (fftx_callback)complex_scaling,
        MY_FFTX_MODE_SUB | FFTX_PW_POINTWISE);

    tmp4 = fftx_create_temp_real(rank, &padded_dims);
    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;
}
```

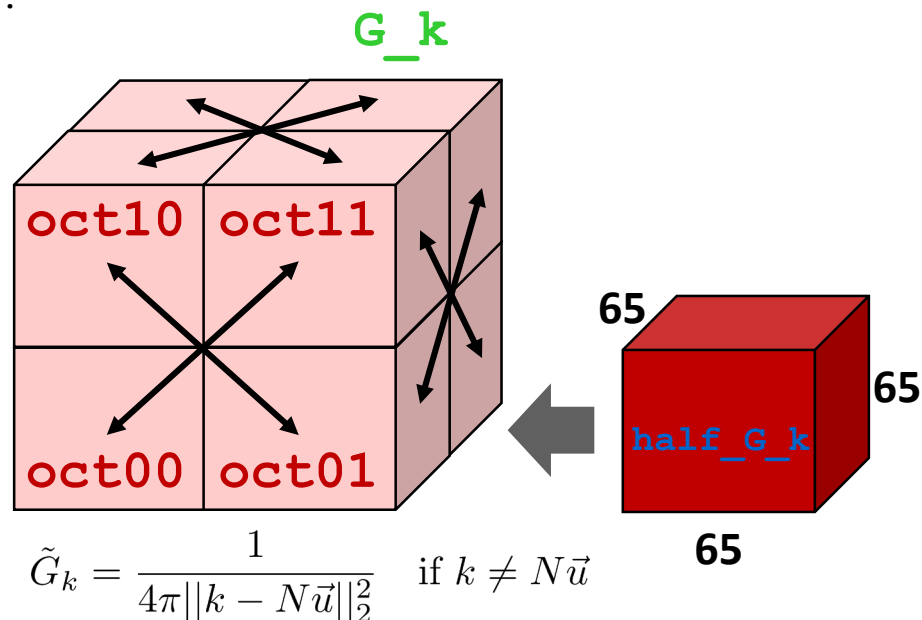
*Looks like FFTW calls, but is a specification for SPIRAL*

# FFTX Example: Describing The Hockney Symmetry

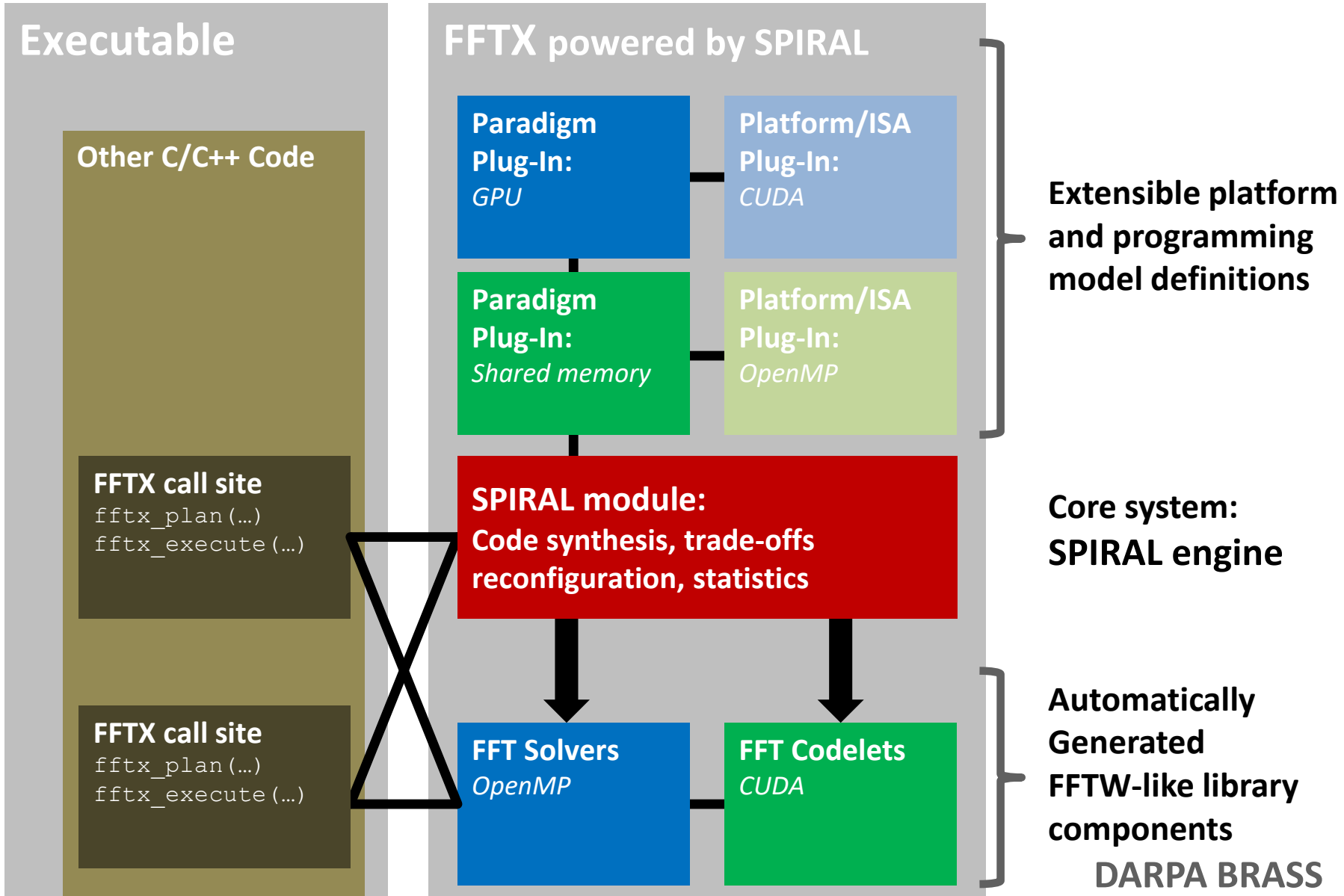
```
// FFTX data access descriptors.
// Access is to four octants of a symmetric cube.
// Cube size is N^3 and M = N/2.
```

```
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 } };
...
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);
...

```



# FFTX Backend: SPIRAL



# Generated Code For Hockney Convolution

```

void ioprunedconv_130_0_62_72_130(double *Y, double *X, double * S) {
    static double D84[260] = {65.5, 0.0, (-0.50000000000001132), (-20.686114762237267),
        (-0.5000000000000081), (-10.337014680426078), (-0.50000000000000455),
        ...
    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 = (((0.3090169943749474**((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 code, cross call optimization, etc., transparently used***

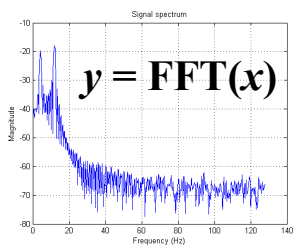
# SPIRAL: Go from Mathematics to Software

## Given:

- Mathematical problem specification  
*does not change*
- Computer platform  
*changes often*

## Wanted:

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

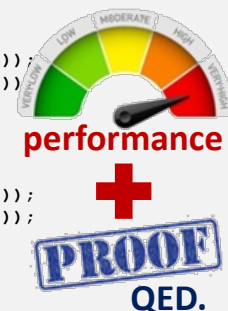


on



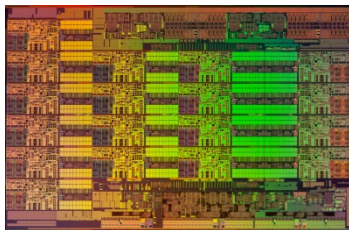
automatic

```
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));
    ...
}
```

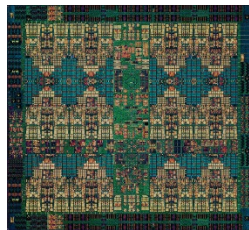


# SPIRAL's Target Computing Landscape

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



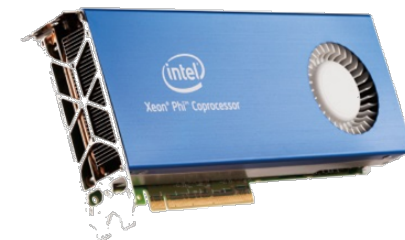
**Intel Xeon 8180M**  
*2.25 Tflop/s, 205 W*  
28 cores, 2.5—3.8 GHz  
2-way—16-way AVX-512



**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



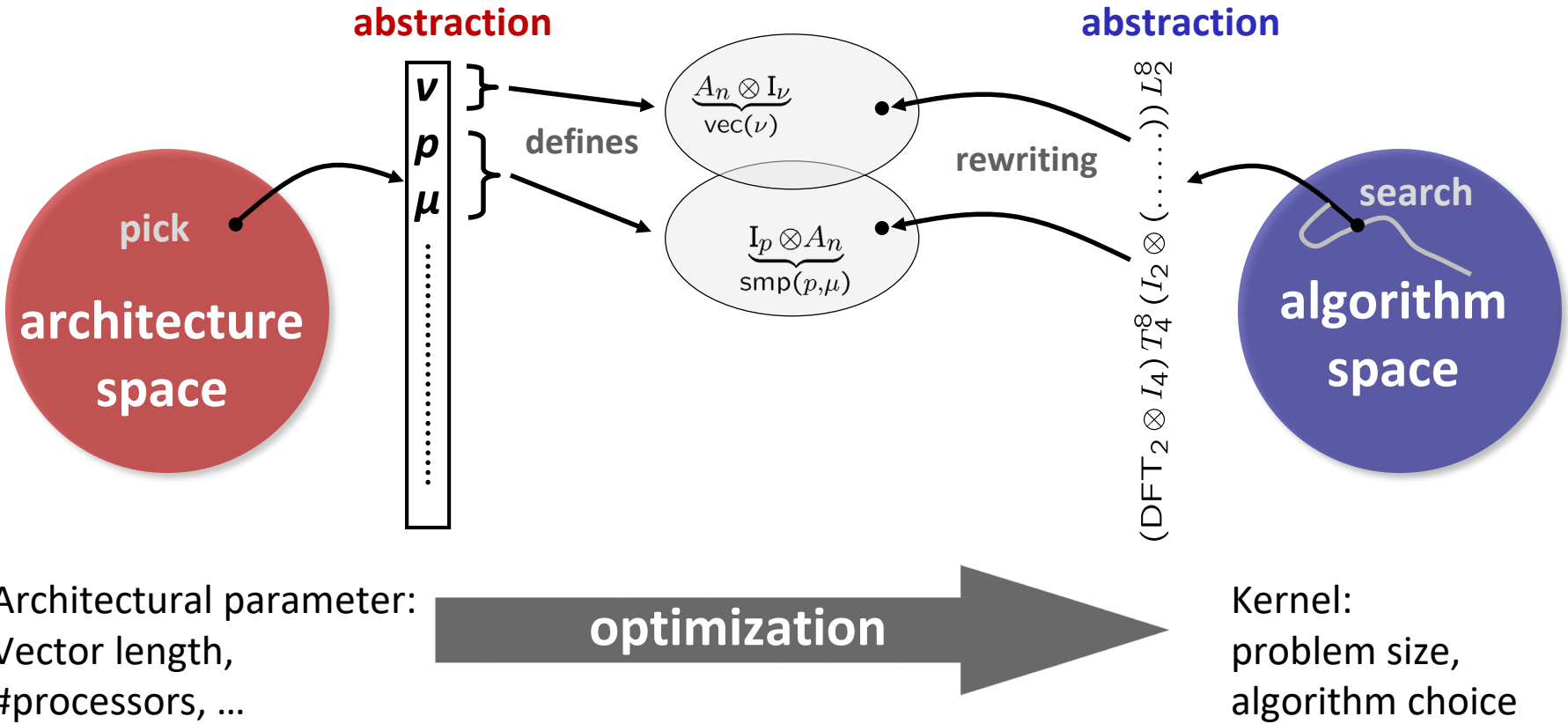
**Dell PowerEdge R940**  
*3.2 Tflop/s, 6 TB, 850 W*  
4x 24 cores, 2.1 GHz  
4-way/8-way AVX



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

# Platform-Aware Formal Program Synthesis

**Model:** common abstraction  
 = spaces of matching formulas



Architectural parameter:  
 Vector length,  
 #processors, ...

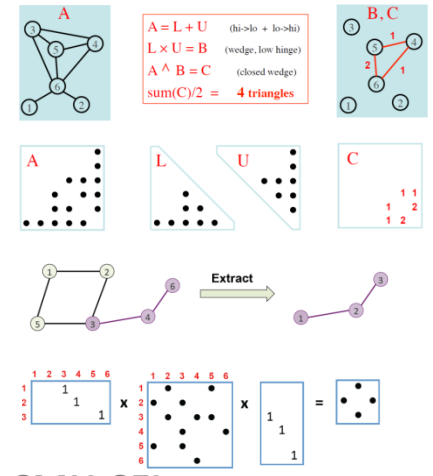
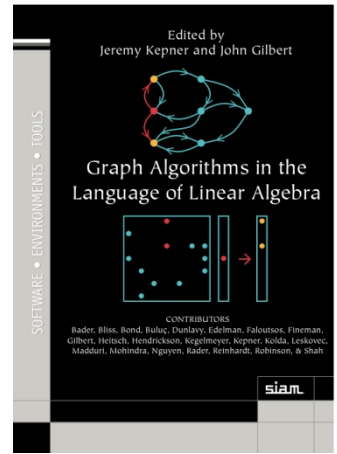
Kernel:  
 problem size,  
 algorithm choice

# Algorithms: Rules in Domain Specific Language

## Linear Transforms

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

## Graph Algorithms



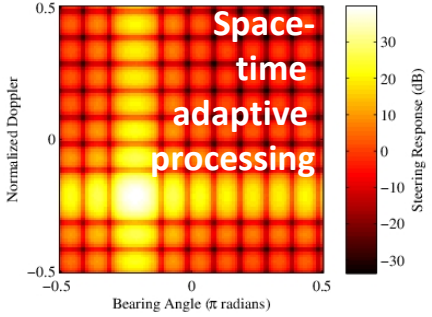
In collaboration with CMU-SEI

## Numerical Linear Algebra



$$\begin{aligned}
 \text{MMM}_{1,1,1} &\rightarrow (\cdot)_1 \\
 \text{MMM}_{m,n,k} &\rightarrow (\otimes)_{m/m_b \times 1} \otimes \text{MMM}_{m_b,n,k} \\
 \text{MMM}_{m,n,k} &\rightarrow \text{MMM}_{m,n_b,k} \otimes (\otimes)_{1 \times n/n_b} \\
 \text{MMM}_{m,n,k} &\rightarrow ((\Sigma_{k/k_b} \circ (\cdot)_{k/k_b}) \otimes \text{MMM}_{m,n,k_b}) \circ \\
 &\quad ((L_{k/k_b}^{m/k/k_b} \otimes I_{k_b}) \times I_{kn}) \\
 \text{MMM}_{m,n,k} &\rightarrow (L_m^{mn/n_b} \otimes I_{n_b}) \circ \\
 &\quad ((\otimes)_{1 \times n/n_b} \otimes \text{MMM}_{m,n_b,k}) \circ \\
 &\quad (I_{km} \times (L_{n/n_b}^{kn/n_b} \otimes I_{n_b}))
 \end{aligned}$$

## Spectral Domain Applications





# Formal Approach for all Types of Parallelism

- **Multithreading** (Multicore)

$$I_p \otimes_{\parallel} A_{\mu n}, \quad L_m^{mn} \bar{\otimes} I_{\mu}$$

- **Vector SIMD** (SSE, VMX/AltiVec,...)

$$A \hat{\otimes} I_{\nu} \quad \underbrace{L_2^{2\nu}}_{\text{isa}}, \quad \underbrace{L_{\nu}^{2\nu}}_{\text{isa}}, \quad \underbrace{L_{\nu}^{\nu^2}}_{\text{isa}}$$

- **Message Passing** (Clusters, MPP)

$$I_p \otimes_{\parallel} A_n, \quad \underbrace{L_p^{p^2} \bar{\otimes} I_{n/p^2}}_{\text{all-to-all}}$$

- **Streaming/multibuffering** (Cell)

$$I_n \otimes_2 A_{\mu n}, \quad L_m^{mn} \bar{\otimes} I_{\mu}$$

- **Graphics Processors** (GPUs)

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

- **Gate-level parallelism** (FPGA)

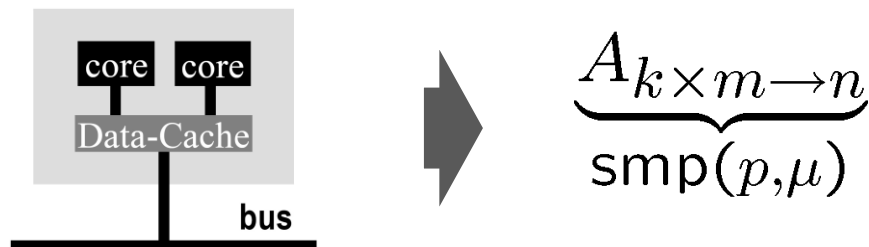
$$\prod_{i=0}^{n-1} A_i, \quad I_s \tilde{\otimes} A, \quad \underbrace{L_n^m}_{\text{bram}}$$

- **HW/SW partitioning** (CPU + FPGA)

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

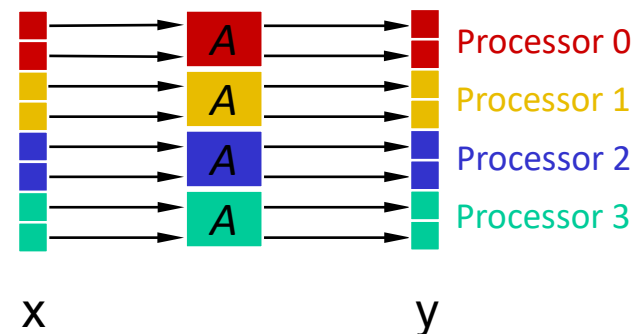
# Modeling Hardware: Base Cases

- Hardware abstraction: shared cache with cache lines



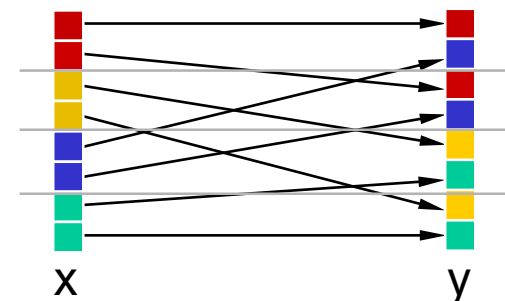
- Tensor product: embarrassingly parallel operator

$$y = \left( I_p \otimes A \right) (x)$$

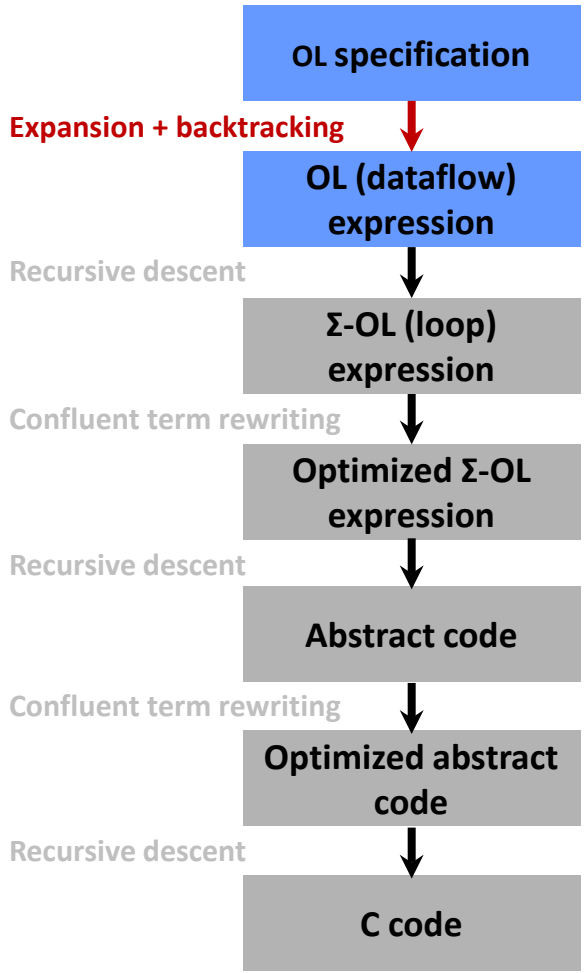
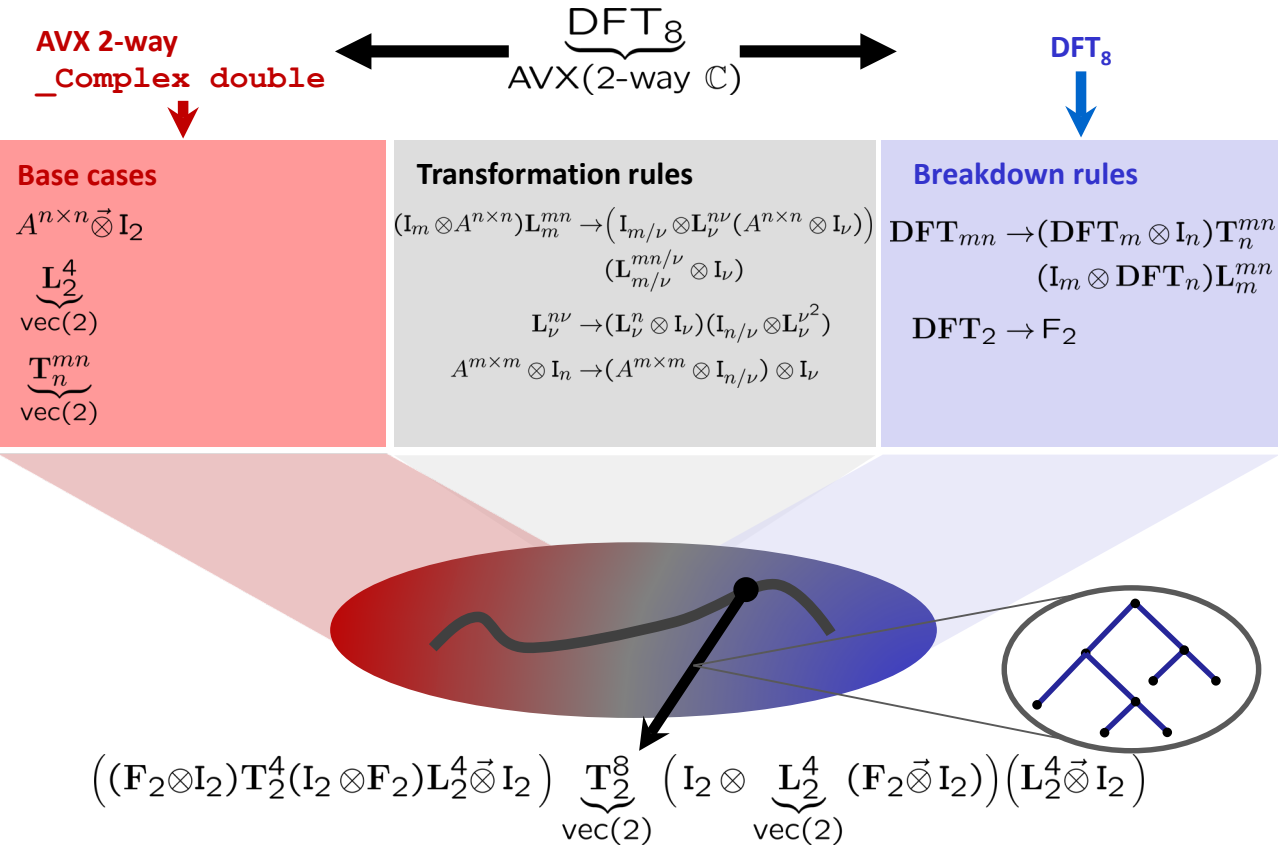


- Permutation: problematic; may produce false sharing

$$y = L_4^{\otimes 8}(x)$$



# Autotuning in Constraint Solution Space



# Translating an OL Expression Into Code

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

Output =

Ruletree, expanded into

**OL Expression:**

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

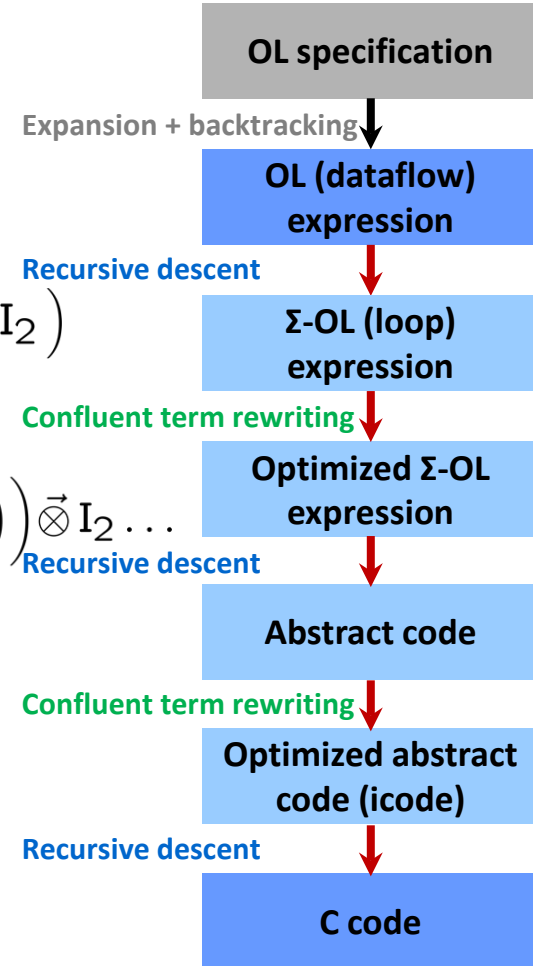
**$\Sigma$ -OL:**

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

**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;
}
```

See Figure 5



# Symbolic Verification for Linear Operators

- Linear operator = matrix-vector product  
Algorithm = matrix factorization

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & j & -1 & -j \\ 1 & -1 & 1 & -1 \\ 1 & -j & -1 & j \end{bmatrix} = \begin{bmatrix} 1 & \cdot & 1 & \cdot \\ \cdot & 1 & \cdot & 1 \\ 1 & \cdot & -1 & \cdot \\ \cdot & 1 & \cdot & -1 \end{bmatrix} \begin{bmatrix} 1 & \cdot & \cdot & \cdot \\ \cdot & 1 & \cdot & \cdot \\ \cdot & \cdot & 1 & \cdot \\ \cdot & \cdot & \cdot & j \end{bmatrix} \begin{bmatrix} 1 & 1 & \cdot & \cdot \\ 1 & -1 & \cdot & \cdot \\ \cdot & \cdot & 1 & 1 \\ \cdot & \cdot & 1 & -1 \end{bmatrix} \begin{bmatrix} 1 & \cdot & \cdot & \cdot \\ \cdot & \cdot & 1 & \cdot \\ \cdot & 1 & \cdot & \cdot \\ \cdot & \cdot & \cdot & 1 \end{bmatrix} = ?$$

$$\text{DFT}_4 = (\text{DFT}_2 \otimes \text{I}_2) \text{T}_2^4 (\text{I}_2 \otimes \text{DFT}_2) \text{L}_2^4$$

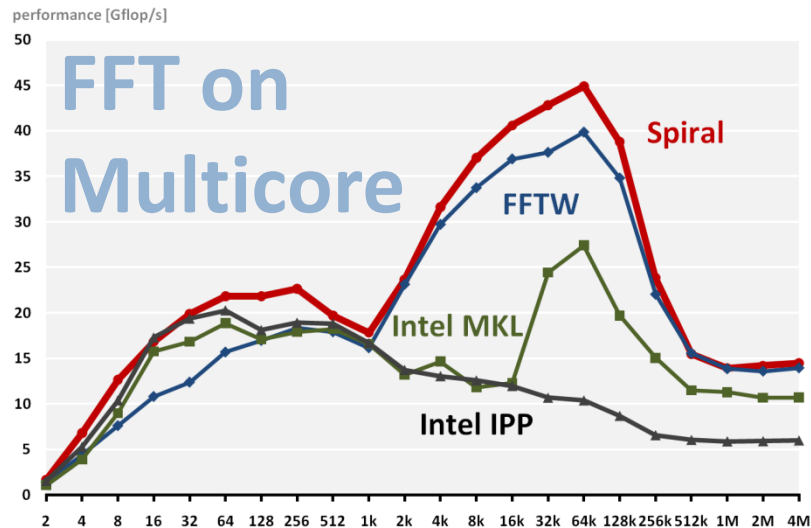
- Linear operator = matrix-vector product  
Program = matrix-vector product

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & j & -1 & -j \\ 1 & -1 & 1 & -1 \\ 1 & -j & -1 & j \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} = ? \quad \text{DFT}_4([0, 1, 0, 0])$$

*Symbolic evaluation and symbolic execution establishes correctness*

# Results: FFTs and Spectral Algorithms

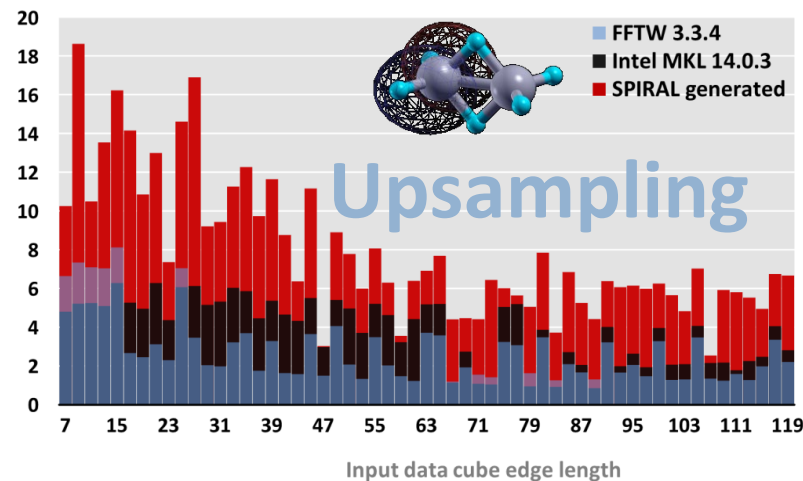
1D DFT on 3.3 GHz Sandy Bridge (4 Cores, AVX)



Performance of 2x2x2 Upsampling on Haswell

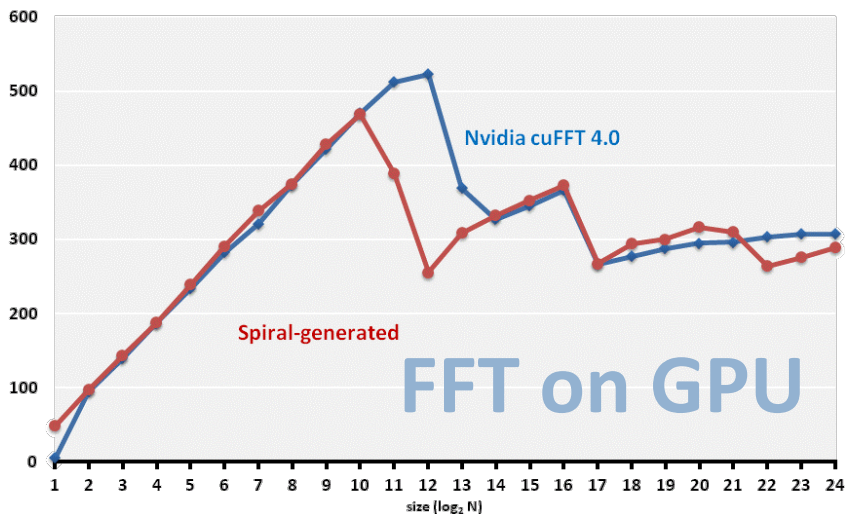
3.5 GHz, AVX, double precision, interleaved input, single core

Performance [Pseudo Gflop/s]



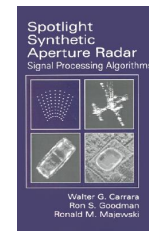
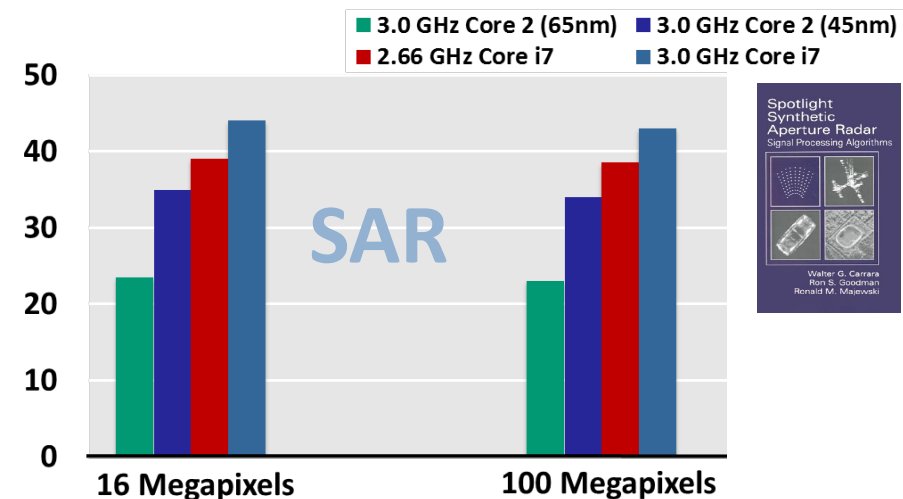
1D Batch DFT (Nvidia GTX 480)

performance [GFlop/s], single precision



PFA SAR Image Formation on Intel platforms

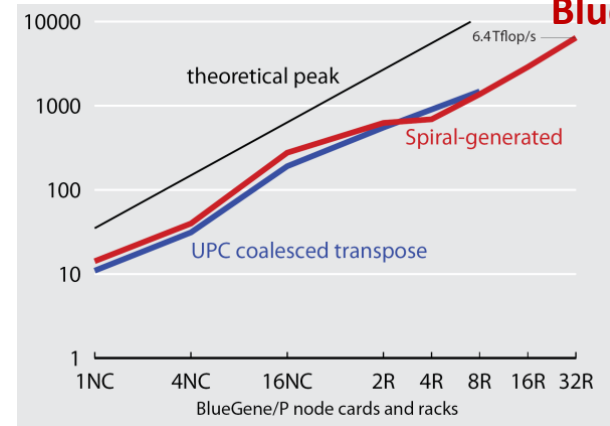
performance [Gflop/s]



# SPIRAL: Success in HPC/Supercomputing

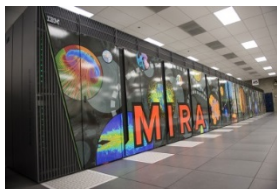
- **NCSA Blue Waters**  
PAID Program, FFTs for Blue Waters
- **RIKEN K computer**  
FFTs for the HPC-ACE ISA
- **LANL RoadRunner**  
FFTs for the Cell processor
- **PSC/XSEDE Bridges**  
Large size FFTs
- **LLNL BlueGene/L and P**  
FFTW for BlueGene/L's Double FPU
- **ANL BlueGene/Q Mira**  
Early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

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



6.4 Tflop/s on  
BlueGene/P

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 8.0: 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**

```

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
  SpiralDefaults
  SpiralVersion
PrintCode("dft8", CodeRuleTree(rt, SpiralDefaults), SpiralDefaults);

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
    , t165, t166, t167, t168, t169, t170, t171, t172
    , t173, t174, t175, t176;
  t149 = *(X) + *((X + 8));
  t150 = *((X + 1)) + *((X + 9));
  t151 = *(X) - *((X + 8));
  t152 = *((X + 1)) - *((X + 9));
  t153 = *((X + 2)) + *((X + 10));

```

[www.spiral.net](http://www.spiral.net)

