Open Source SPIRAL 8.4
Summary

Franz Franchetti
Carnegie Mellon University

Tutorial based on joint work with the Spiral team at CMU, UIUC, and Drexel

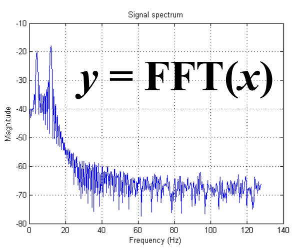
SPIRAL: AI for Performance Engineering

Given:

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

Wanted:

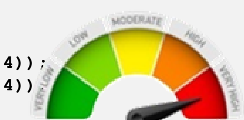
- **Very good implementation of specification on platform**
- **Proof of correctness**



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));
    ...
}
```

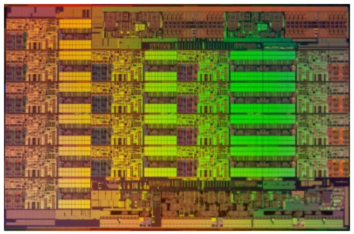


+ PROOF

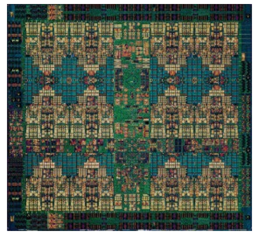
QED.

Some of SPIRAL's Targets

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

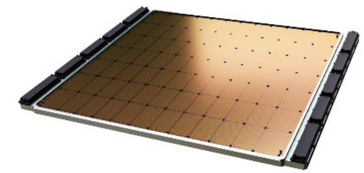


Intel Xeon 8380HL
2.5 Tflop/s, 205 W
28 cores, 2.9—4.3 GHz
2-way—16-way AVX-512



IBM POWER9
768 Gflop/s, 300 W
24 cores, 4 GHz
4-way VSX-3

Nvidia A100
9.7 Tflop/s, 400 W
6912 cores, 1.41 GHz
312 Tflop/s tensor cores

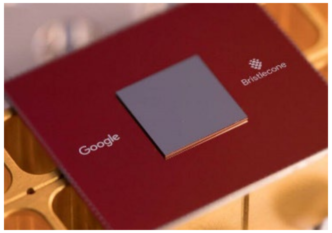


Cerebras WSE2
5.8 Pflop/s 20kW
850,000 cores

Snapdragon 835
15 Gflop/s, 2 W
8 cores, 2.3 GHz
A540 GPU, 682 DSP, NEON

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

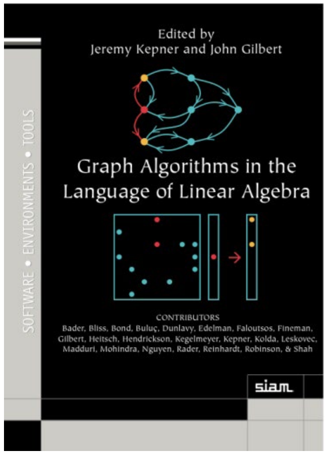
Fugaku
537 Pflop/s, 30 MW
7,630,848 cores A64FX

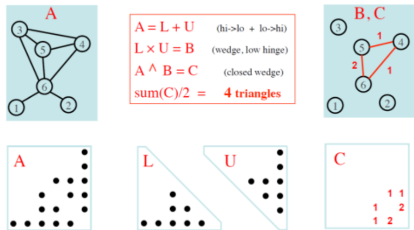


Google Bristlecone
72 qubits

Current Research Directions

SPIRAL for Graphs

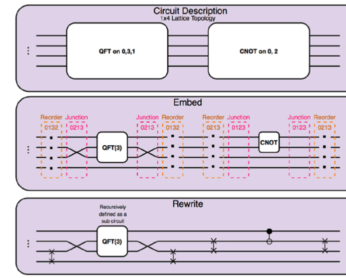


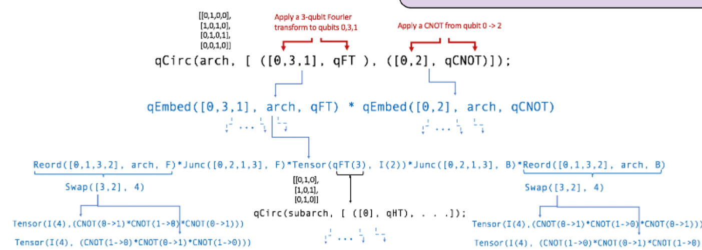


$$\Delta = \Delta + \frac{1}{2}\alpha_{10}A_{00}\alpha_{01}$$

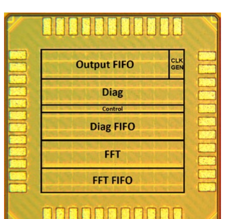
```
# OL Algorithm Specification
Accum(i4, 1, X.N-1,
  Accum_X(i6, [ i4, 0 ], i4,
    Dot([ i6, add(i4, V(1)) ],
      [ i4, add(i4, V(1)) ],
       sub(sub(X.N, i4), V(1)))
  )
)
```

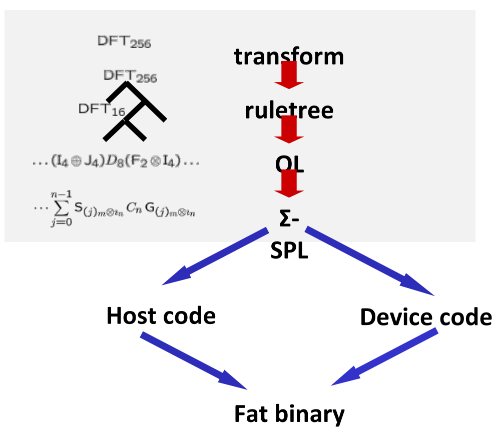
Spiral for Quantum Computing



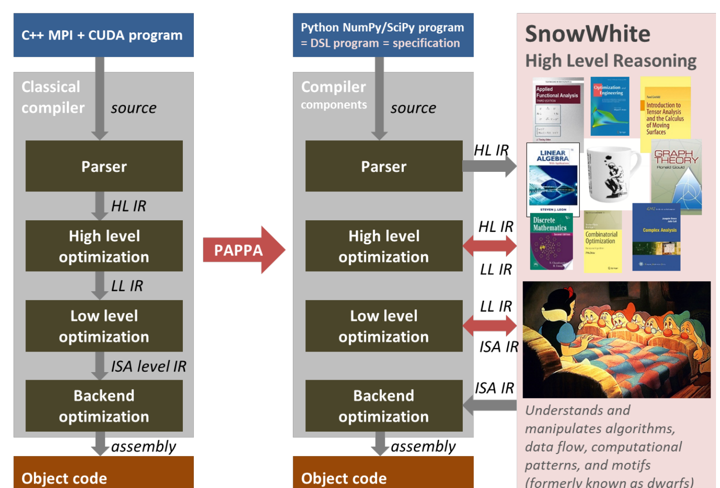


HW/SW Co-Design





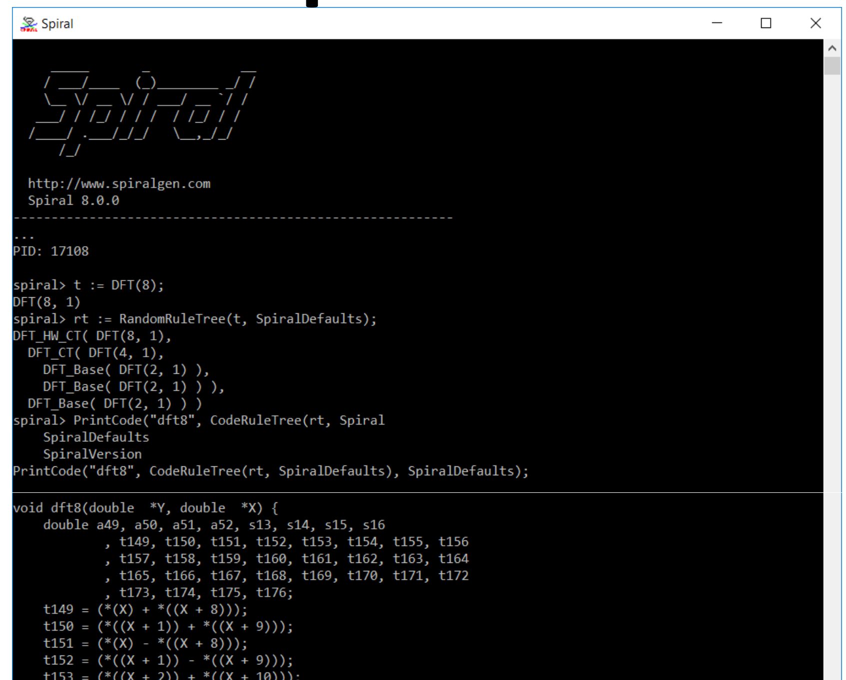
SPIRAL as AI in Compilers



SPIRAL 8.4.0: Available Under Open Source

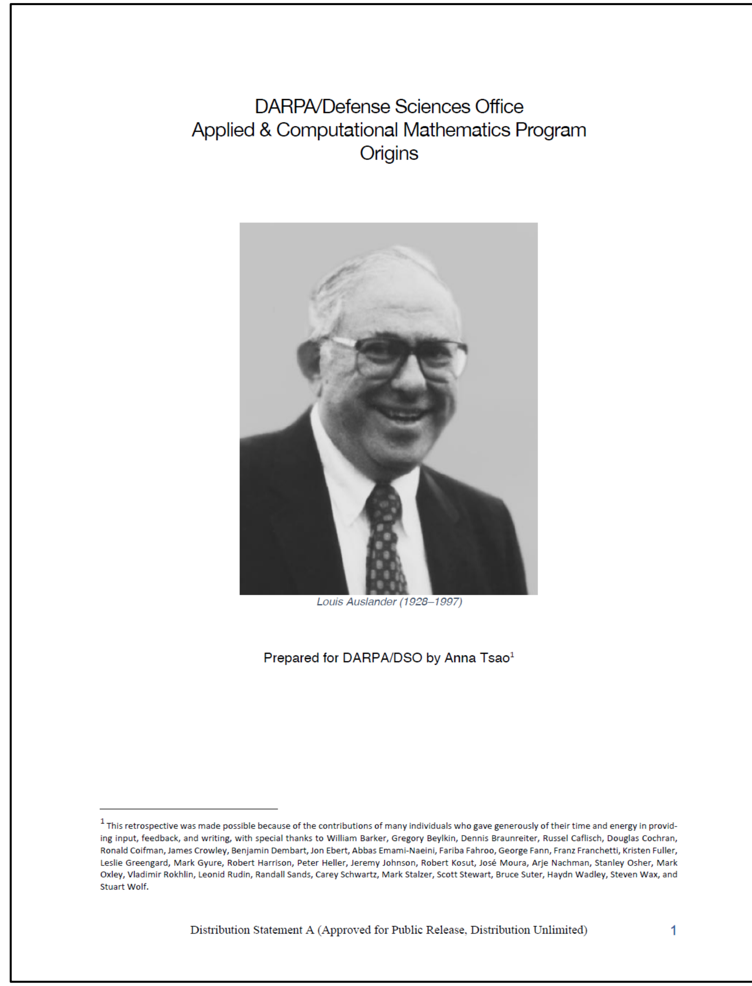
- **Open Source SPIRAL available**
  - non-viral license (BSD)
  - Initial version, effort ongoing to open source whole system
  - Commercial support via SpiralGen, Inc.
- **Developed over 20 years**
  - Funding: DARPA (OPAL, DESA, HACMS, PERFECT, BRASS), NSF, ONR, DoD HPC, JPL, DOE (ECP, XStack, SciDAC), CMU SEI, Intel, Nvidia, Mercury
  - Open sourced under DARPA PERFECT

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*

SPIRAL's History



DARPA/Defense Sciences Office
Applied & Computational Mathematics Program
Origins

Louis Auslander (1928–1997)

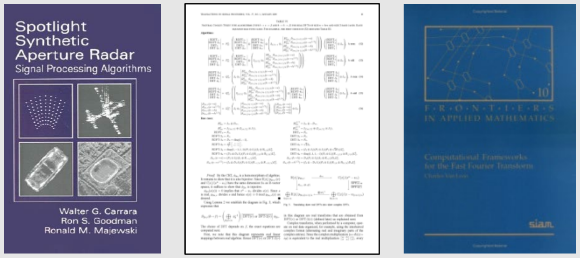
Prepared for DARPA/DSO by Anna Tsao[1]

[1] This retrospective was made possible because of the contributions of many individuals who gave generously of their time and energy in providing input, feedback, and writing, with special thanks to William Barker, Gregory Beylkin, Dennis Braunreiter, Russel Caflisch, Douglas Cochran, Ronald Coifman, James Crowley, Benjamin Dembart, Jon Ebert, Abbas Emami-Naeini, Fariba Fahroo, George Fann, Franz Franchetti, Kristen Fuller, Leslie Greengard, Mark Gyure, Robert Harrison, Peter Heller, Jeremy Johnson, Robert Kosut, José Moura, Arje Nachman, Stanley Osher, Mark Oxley, Vladimir Rokhlin, Leonid Rudin, Randall Sands, Carey Schwartz, Mark Stalzer, Scott Stewart, Bruce Suter, Haydn Wadley, Steven Wax, and Stuart Wolf.

Distribution Statement A (Approved for Public Release, Distribution Unlimited) 1

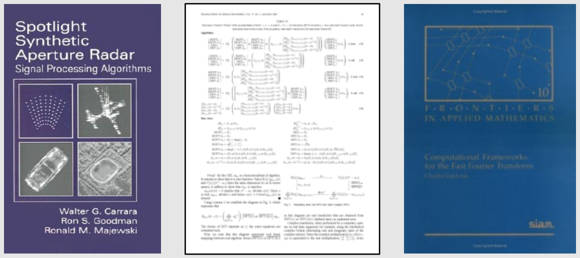
SPIRAL As AI System

Traditionally



High performance library optimized for given platform

Spiral Approach



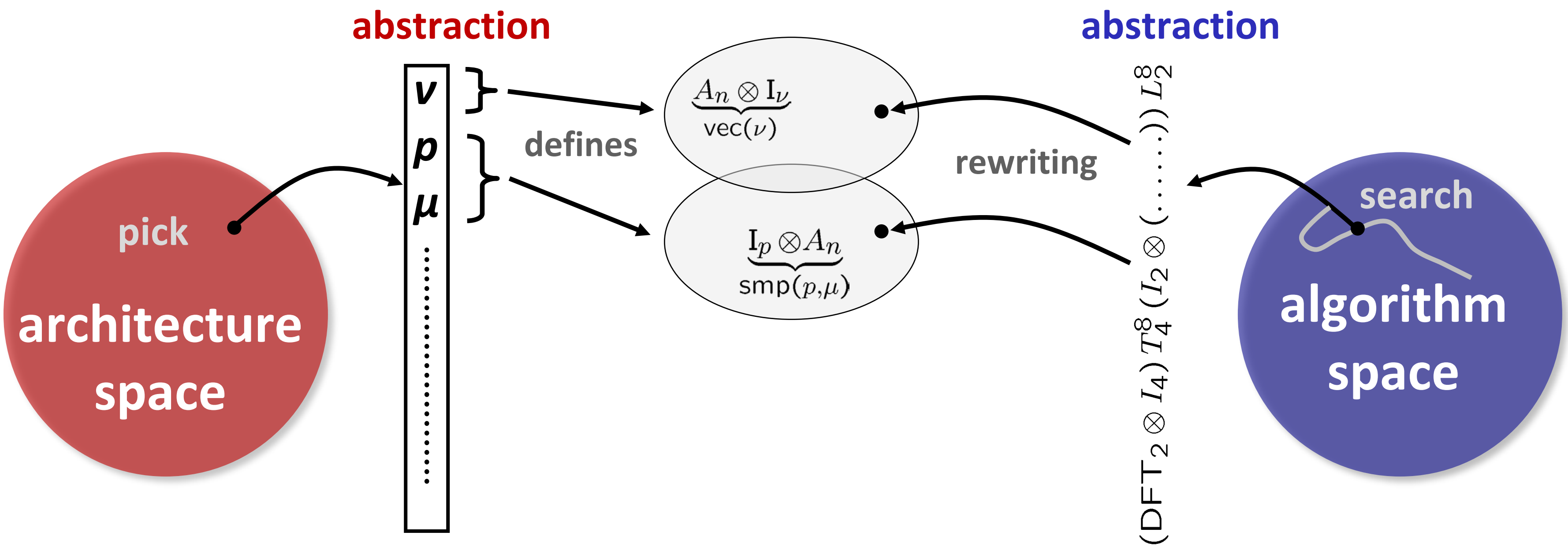
Spiral

High performance library optimized for given platform

Comparable performance

Platform-Aware Formal Program Synthesis

Model: common abstraction
= spaces of matching formulas



Architectural parameter:
Vector length,
#processors, …

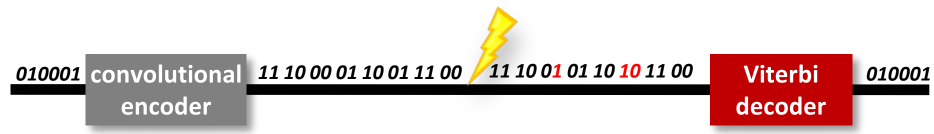
Kernel:
problem size,
algorithm choice

Some Application Domains in SPL

Linear Transforms

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

Software Defined Radio

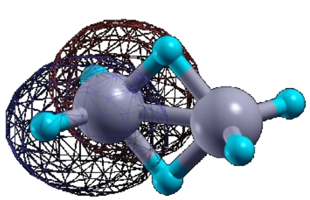


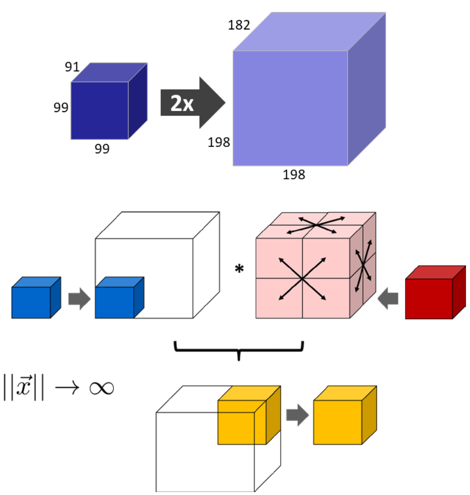
$$F_{K,F} \to \prod_{i=1}^{F} \left((I_{2^{K-2}} \otimes_j B_{F-i,j}) L^{2^{K-1}}_{2^{K-2}} \right)$$

$$\underline{\mathbf{F}_{K,F}}\ \nu \to \prod_{i=1}^{F} \left(\left(\mathrm{I}_{2^{K-2}/\nu} \otimes_{j_1} \mathrm{L}^{2\nu}_{\nu} \vec{B}^{\nu}_{F-i,j_1} \right) \left(\mathrm{L}^{2^{K-1}/\nu}_{2^{K-2}/\nu} \bar{\otimes} \mathrm{I}_\nu \right) \right)$$

$$B_{i,j} : \begin{cases} \pi_U = \min_{d_U}(\pi_A + \beta_{A\to U}, \pi_B + \beta_{B\to U}) \\ \pi_V = \min_{d_V}(\pi_A + \beta_{A\to V}, \pi_B + \beta_{B\to V}) \end{cases}$$

PDEs/HPC Simulations



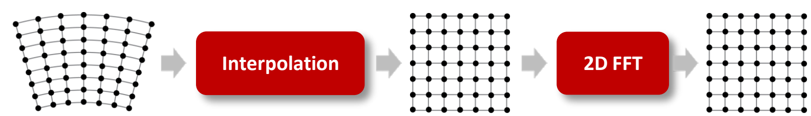


$$\Phi : \mathbb{R}^3 \to \mathbb{R}$$

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

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

Radar Image Formation (SAR, STAP)



$$\begin{aligned}
\mathrm{SAR}_{k\times m \to n\times n} &\to \mathrm{DFT}_{n\times n} \circ \mathrm{Interp}_{k\times m \to n\times n}\\
\mathrm{DFT}_{n\times n} &\to (\mathrm{DFT}_n \otimes \mathrm{I}_n) \circ (\mathrm{I}_n \otimes \mathrm{DFT}_n)\\
\mathrm{Interp}_{k\times m \to n\times n} &\to (\mathrm{Interp}_{k\to n} \otimes_i \mathrm{I}_n) \circ (\mathrm{I}_k \otimes_i \mathrm{Interp}_{m\to n})\\
\mathrm{Interp}_{r\to s} &\to \left(\bigoplus_{i=0}^{n-2} \mathrm{InterpSeg}_k \right) \oplus \mathrm{InterpSegPruned}_{k,\ell}\\
\mathrm{InterpSeg}_k &\to \mathrm{G}_f^{u\cdot n \to k} \circ \mathrm{iPrunedDFT}_{n\to u\cdot n} \circ \left(\frac{1}{n}\right) \circ \mathrm{DFT}_n
\end{aligned}$$

Formal Approach for all Types of Parallelism

- **Multithreading** (Multicore) $\mathrm{I}_p \otimes_{\|} A_{\mu n}, \quad \mathrm{L}_m^{mn} \bar{\otimes} \mathrm{I}_\mu$
- **Vector SIMD** (SSE, VMX/Altivec,…) $A \hat{\otimes} \mathrm{I}_\nu \quad \underbrace{\mathrm{L}_2^{2\nu}}_{\text{isa}}, \quad \underbrace{\mathrm{L}_\nu^{2\nu}}_{\text{isa}}, \quad \underbrace{\mathrm{L}_\nu^{\nu^2}}_{\text{isa}}$
- **Message Passing** (Clusters, MPP) $\mathrm{I}_p \otimes_{\|} A_n, \quad \underbrace{\mathrm{L}_p^{p^2} \bar{\otimes} \mathrm{I}_{n/p^2}}_{\text{all-to-all}}$
- **Streaming/multibuffering** (Cell) $\mathrm{I}_n \otimes_2 A_{\mu n}, \quad \mathrm{L}_m^{mn} \bar{\otimes} \mathrm{I}_\mu$
- **Graphics Processors** (GPUs) $\prod_{i=0}^{n-1} A_i, \quad A_n \hat{\otimes} \mathrm{I}_w, \quad P_n \otimes Q_w$
- **Gate-level parallelism** (FPGA) $\prod_{i=0}^{n-1}{}^{\text{ir}} A, \quad \mathrm{I}_s \tilde{\otimes} A, \quad \underbrace{\mathrm{L}_n^m}_{\text{bram}}$
- **HW/SW partitioning** (CPU + FPGA) $\underbrace{A_1}_{\text{fpga}}, \quad \underbrace{A_2}_{\text{fpga}}, \quad \underbrace{A_3}_{\text{fpga}}, \quad \underbrace{A_4}_{\text{fpga}}$

Autotuning in Constraint Solution Space

AVX 2-way
`_Complex double`

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

DFT_8

Base cases

$A^{n\times n}\vec{\otimes} I_2$

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

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

Transformation rules

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

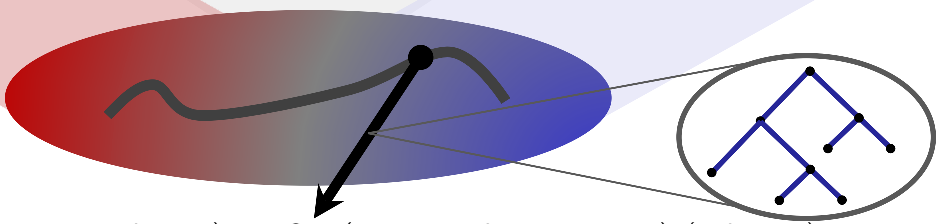
$$L^{n\nu}_\nu \to (L^n_\nu \otimes I_\nu)(I_{n/\nu}\otimes L^{\nu^2}_\nu)$$

$$A^{m\times m}\otimes I_n \to (A^{m\times m}\otimes I_{n/\nu})\otimes I_\nu$$

Breakdown rules

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

$$\mathrm{DFT}_2 \to F_2$$



$$\left((F_2\otimes I_2)T^4_2(I_2\otimes F_2)L^4_2\vec{\otimes} I_2\right)\underbrace{T^8_2}_{\mathrm{vec}(2)}\left(I_2\otimes \underbrace{L^4_2}_{\mathrm{vec}(2)}(F_2\vec{\otimes} I_2)\right)\left(L^4_2\vec{\otimes} I_2\right)$$

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

Translating an SPL Expression Into Code

Constraint Solver Input:

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

Output =

Ruletree, expanded into

SPL Expression:

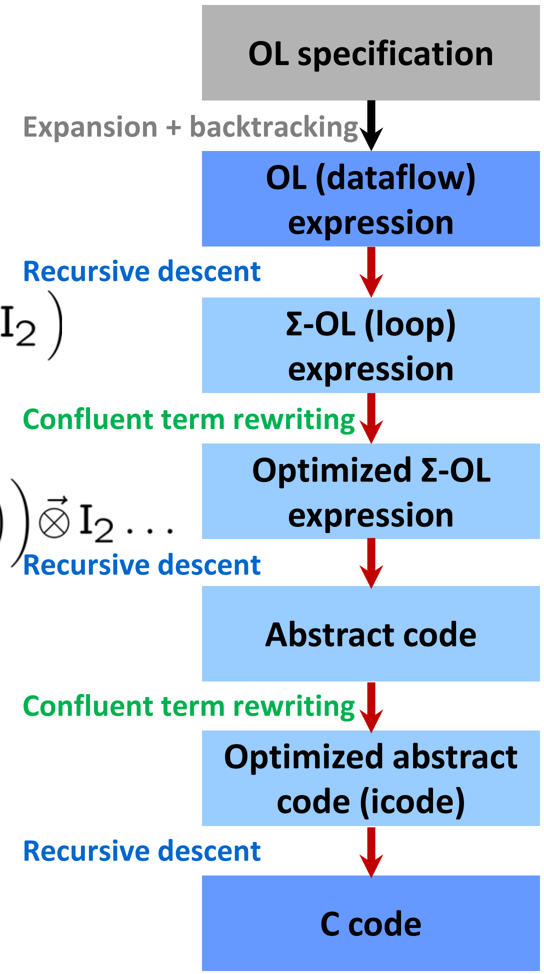
$$\big((F_2\otimes I_2)T^4_2(I_2\otimes F_2)L^4_2\vec{\otimes} I_2\big)\underbrace{T^8_2}_{\mathrm{vec}(2)}\big(I_2\otimes\underbrace{L^4_2}_{\mathrm{vec}(2)}(F_2\vec{\otimes} I_2)\big)\big(L^4_2\vec{\otimes} I_2\big)$$

∑-SPL:

$$\Big(\sum_{j=0}^{1}\big(S_{\imath_2\otimes(j)_2}F_2\,\mathrm{Map}^2_{x\mapsto\omega_4^{2i+j}x}\,G_{\imath_2\otimes(j)_2}\big)\sum_{j=0}^{1}\big(S_{(j)_2\otimes\imath_2}F_2G_{\imath_2\otimes(j)_2}\big)\Big)\vec{\otimes} I_2\ldots$$

C Code:

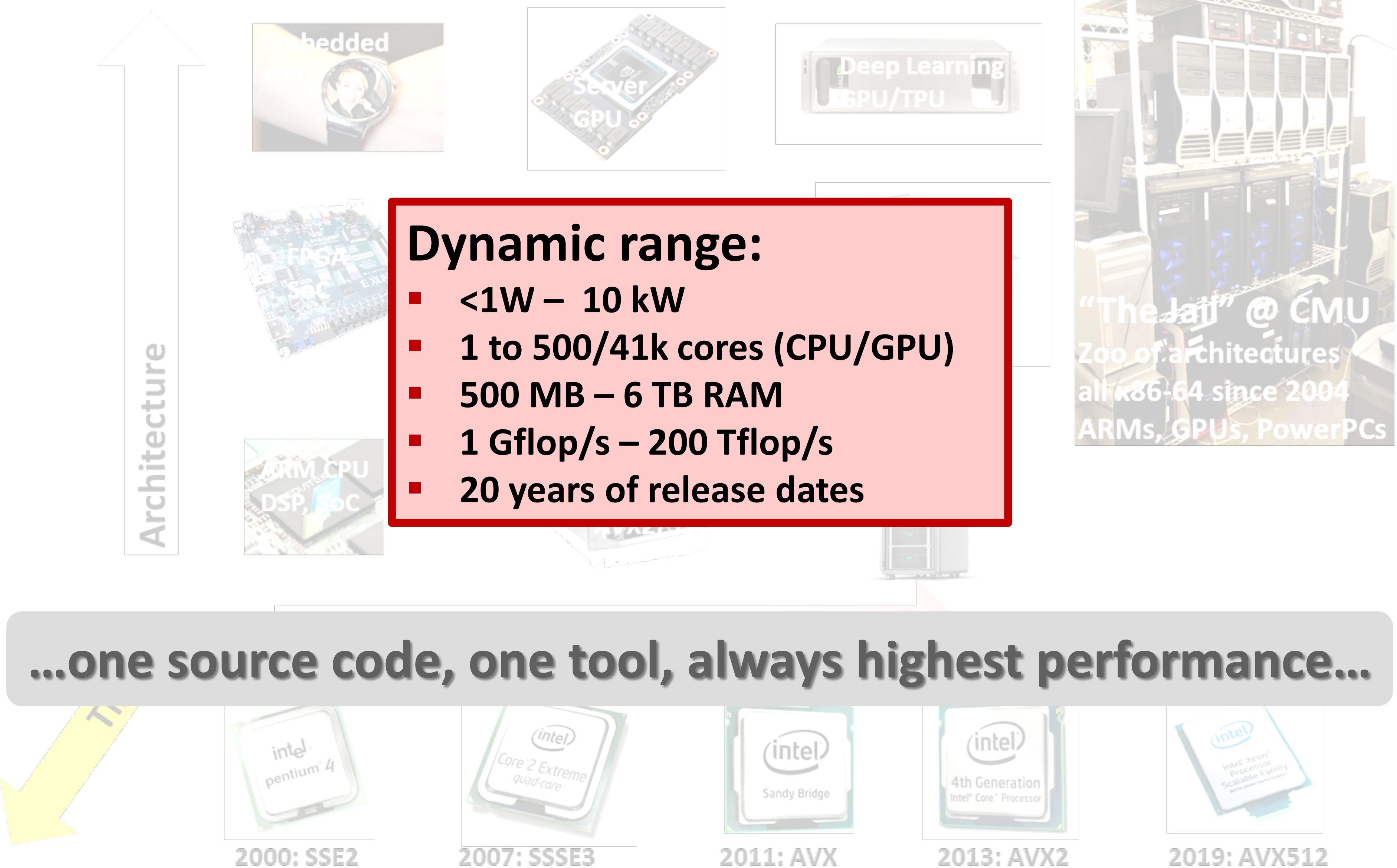
```
void dft8(_Complex double *Y, _Complex double *X) {
    __m256d s38, s39, s40, s41,...
    __m256d  *a17, *a18;
    a17 = ((__m256d  *) X);
    s38 = *(a17);
    s39 = *((a17 + 2));
    t38 = _mm256_add_pd(s38, s39);
    t39 = _mm256_sub_pd(s38, s39);
    ...
    s52 = _mm256_sub_pd(s45, s50);
 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



Some SPIRAL Generated C/AVX Code

```
int dwmonitor(float  *X, double  *D) {
    __m128d u1, u2, u3, u4, u5, u6, u7, u8 , x1, x10, x13, x14, x17, x18, x19, x2, x3, x4, x6, x7, x8, x9;
    int w1;
    unsigned _xm = _mm_getcsr();
    _mm_setcsr(_xm & 0xffff0000 | 0x0000dfc0);
    u5 = _mm_set1_pd(0.0);
    u2 = _mm_cvtps_pd(_mm_addsub_ps(_mm_set1_ps(FLT_MIN), _mm_set1_ps(X[0])));
    u1 = _mm_set_pd(1.0, (-1.0));
    for(int i5 = 0; i5 <= 2; i5++) {
        x6 = _mm_addsub_pd(_mm_set1_pd((DBL_MIN + DBL_MIN)), _mm_loaddup_pd(&(D[i5])));
        x1 = _mm_addsub_pd(_mm_set1_pd(0.0), u1);
        x2 = _mm_mul_pd(x1, x6);
        x3 = _mm_mul_pd(_mm_shuffle_pd(x1, x1, _MM_SHUFFLE2(0, 1)), x6);
        x4 = _mm_sub_pd(_mm_set1_pd(0.0), _mm_min_pd(x3, x2));
        u3 = _mm_add_pd(_mm_max_pd(_mm_shuffle_pd(x4, x4, _MM_SHUFFLE2(0, 1)), _mm_max_pd(x3, x2)), _mm_set1_pd(DBL_MIN));
        u5 = _mm_add_pd(u5, u3);
        x7 = _mm_addsub_pd(_mm_set1_pd(0.0), u1);
        x8 = _mm_mul_pd(x7, u2);
        x9 = _mm_mul_pd(_mm_shuffle_pd(x7, x7, _MM_SHUFFLE2(0, 1)), u2);
        x10 = _mm_sub_pd(_mm_set1_pd(0.0), _mm_min_pd(x9, x8));
        u1 = _mm_add_pd(_mm_max_pd(_mm_shuffle_pd(x10, x10, _MM_SHUFFLE2(0, 1)), _mm_max_pd(x9, x8)), _mm_set1_pd(DBL_MIN));
    }
    u6 = _mm_set1_pd(0.0);
    for(int i3 = 0; i3 <= 1; i3++) {
        u8 = _mm_cvtps_pd(_mm_addsub_ps(_mm_set1_ps(FLT_MIN), _mm_set1_ps(X[(i3 + 1)])));
        u7 = _mm_cvtps_pd(_mm_addsub_ps(_mm_set1_ps(FLT_MIN), _mm_set1_ps(X[(3 + i3)])));
        x14 = _mm_add_pd(u8, _mm_shuffle_pd(u7, u7, _MM_SHUFFLE2(0, 1)));
        x13 = _mm_shuffle_pd(x14, x14, _MM_SHUFFLE2(0, 1));
        u4 = _mm_shuffle_pd(_mm_min_pd(x14, x13), _mm_max_pd(x14, x13), _MM_SHUFFLE2(1, 0));
        u6 = _mm_shuffle_pd(_mm_min_pd(u6, u4), _mm_max_pd(u6, u4), _MM_SHUFFLE2(1, 0));
    }
    x17 = _mm_addsub_pd(_mm_set1_pd(0.0), u6);
    x18 = _mm_addsub_pd(_mm_set1_pd(0.0), u5);
    x19 = _mm_cmpge_pd(x17, _mm_shuffle_pd(x18, x18, _MM_SHUFFLE2(0, 1)));
    w1 = (_mm_testc_si128(_mm_castpd_si128(x19), _mm_set_epi32(0xffffffff, 0xffffffff, 0xffffffff, 0xffffffff)) –
         (_mm_testnzc_si128(_mm_castpd_si128(x19), _mm_set_epi32(0xffffffff, 0xffffffff, 0xffffffff, 0xffffffff))));
    __asm nop;
    if (_mm_getcsr() & 0x0d) {
        _mm_setcsr(_xm);
        return -1;
    }
    _mm_setcsr(_xm);
    return w1;
}
```

Dynamic Range: Single Node/Shared Memory



Dynamic range:

- <1W – 10 kW
- 1 to 500/41k cores (CPU/GPU)
- 500 MB – 6 TB RAM
- 1 Gflop/s – 200 Tflop/s
- 20 years of release dates

…one source code, one tool, always highest performance…

Symbolic Verification

- **Transform = Matrix-vector multiplication**
  matrix fully defines the operation

$$\mathrm{DFT}_4 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & j & -1 & -j \\ 1 & -1 & 1 & -1 \\ 1 & -j & -1 & j \end{bmatrix}$$

= ?

- **Algorithm = Formula**
  represents a matrix expression, can be evaluated to a matrix

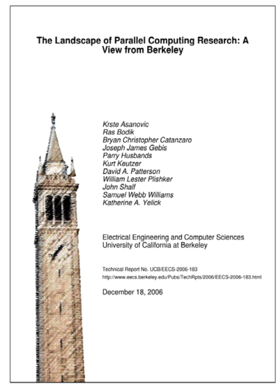
$$(\mathrm{DFT}_2 \otimes \mathrm{I}_2)\, \mathrm{T}_2^4 (\mathrm{I}_2 \otimes \mathrm{DFT}_2)\, \mathrm{L}_2^4 = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & -1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & j \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

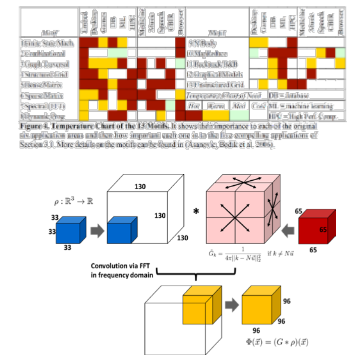
SPIRAL as DSL Frontend and Compiler Plugin

Multi-language, Multi target

Powered by SPIRAL
code generation/synthesis

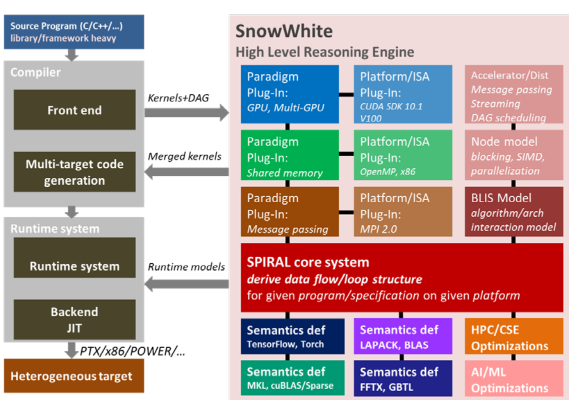
Cross-motif optimization





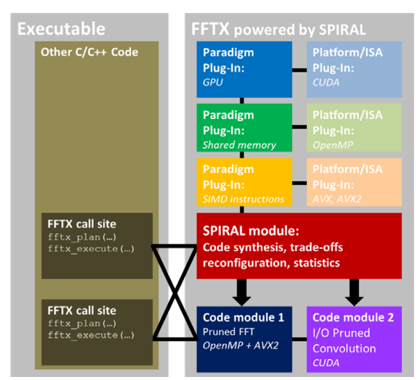
Cross-call, cross-library, cross-motif

SnowWhite: SPIRAL inside compilers



DARPA PAPPA, X-Stack Bluestone

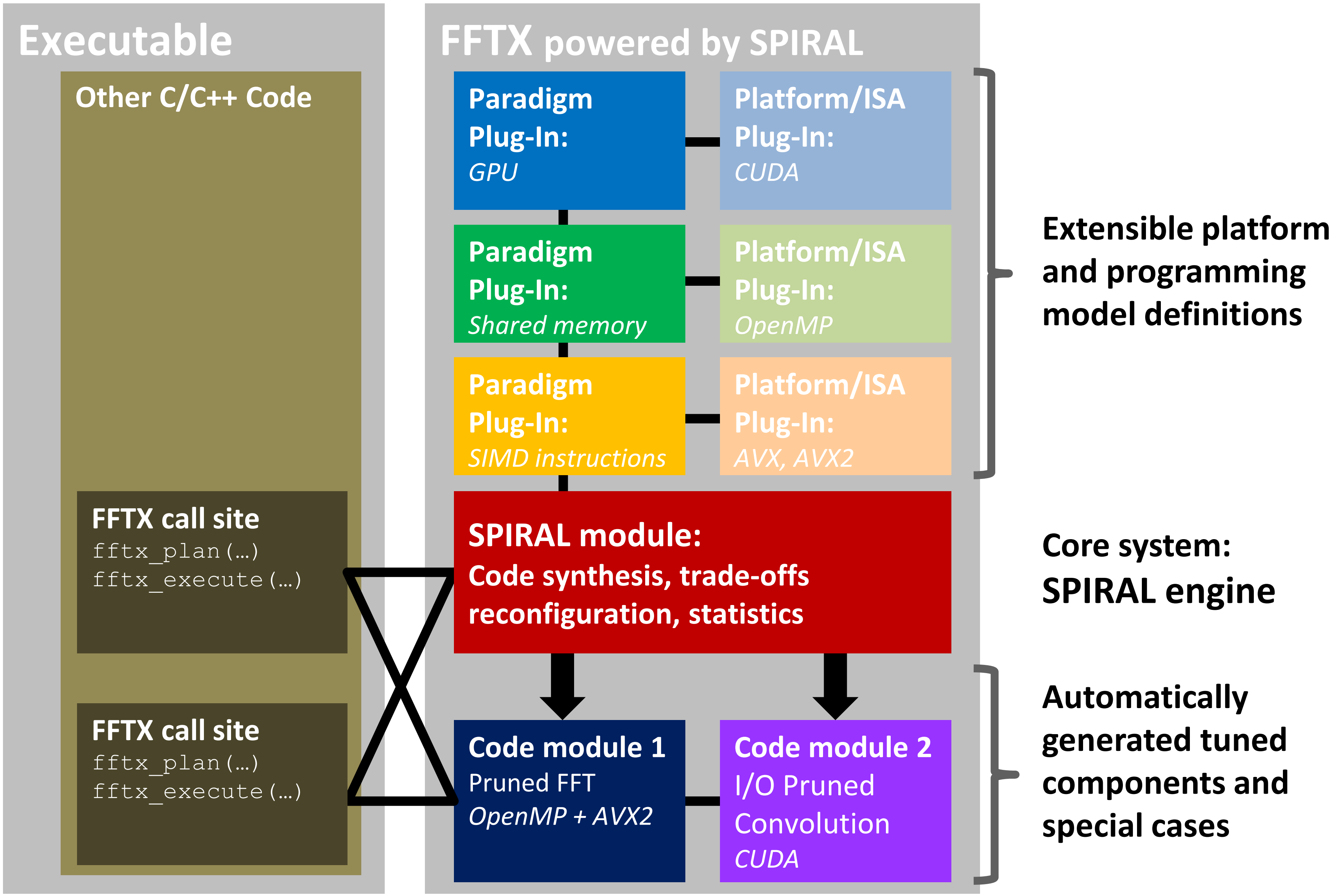
LibraryX, powered by SPIRAL



One instance of a family of active libraries

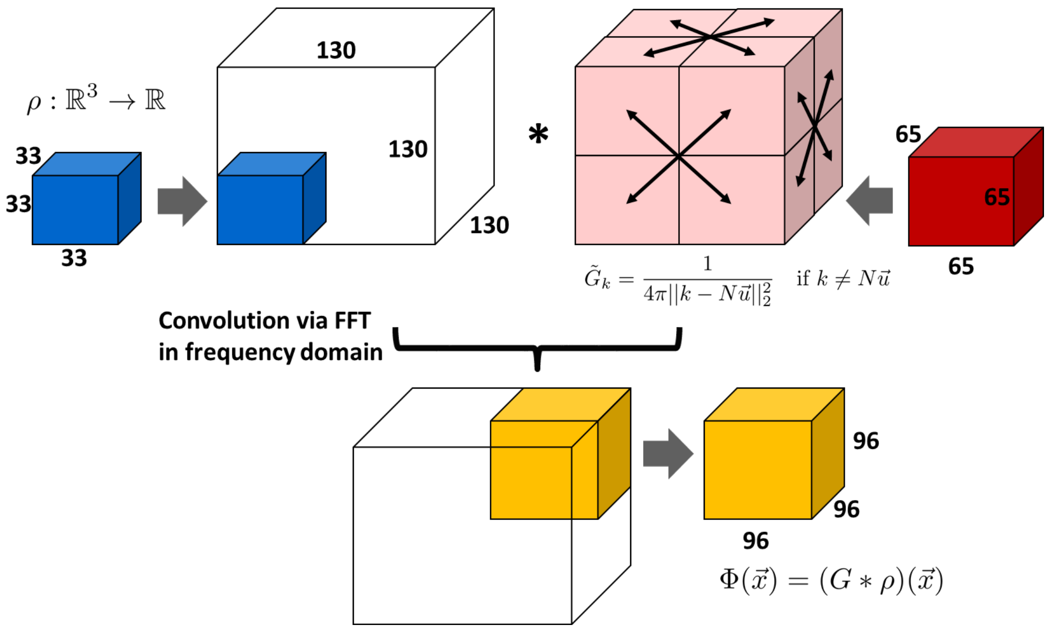
Multiple active libraries, one infrastructure

SPIRAL as Core Engine in LibraryX



In Development: Python API and Beyond

```
embIn = numpy.zeros((N, N, N))
embIn[0:Ns, 0:Ns, 0:Ns] = In
FI = numpy.fft.rfftn(embIn)
C = FI * S
embOut = numpy.fft.irfftn(C)
Out = embOut[N-Nd:N, N-Nd:N, N-Nd:N]
```



Hockney Free Space Convolution translated from C++/FFTX to Python/NumPy

Beyond Linear: Operator Language

Definition

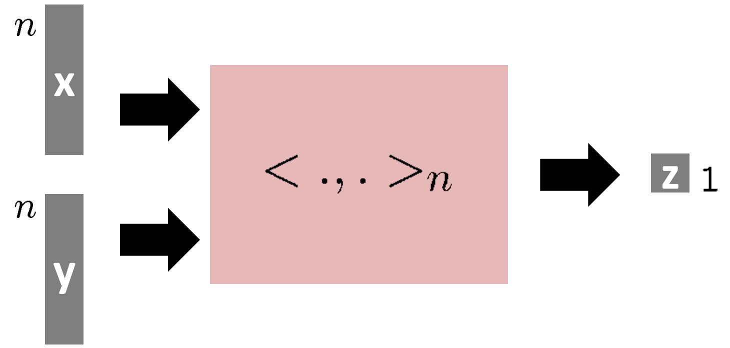
- Operator: Multiple vectors -> Multiple vectors
- Stateless
- Higher-dimensional data is linearized
- Operators are potentially nonlinear

$$\mathrm{M} : \begin{cases} \mathbb{C}^{n_0} \times \cdots \times \mathbb{C}^{n_{k-1}} \to \mathbb{C}^{N_0} \times \cdots \times \mathbb{C}^{N_{\ell-1}} \\ (\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{k-1}) \mapsto \mathrm{M}(\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{k-1}) \end{cases}$$

Example: Scalar product

$$< .,. >_n : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$$

$$\big((x_i)_{i=0,\ldots,n-1}, (y_i)_{i=0,\ldots,n-1}\big) \mapsto \sum_{i=0}^{n-1} x_i y_i$$



OL: Linear Algebra + Functional Programming

- **Application specific:** Safety Distance as Rewrite Rule

$$\mathsf{SafeDist}_{V,A,b,\varepsilon}(.,.,.) \to \Big(P[x,(a_0,a_1,a_2)](.) < d_\infty^2(.,.)\Big)(.,.,.)$$

with $a_0=\frac{1}{2b},\ a_1=\frac{V}{b}+\varepsilon\Big(\frac{A}{b}+1\Big),\ a_2=\Big(\frac{A}{b}+1\Big)\Big(\frac{A}{2}\varepsilon^2+\varepsilon V\Big)$

Problem specification: hand-developed or automatically produced

- **One-time effort:** mathematical library

$$d_\infty^n(.,.) \to ||.||_\infty^n \circ (-)_n$$

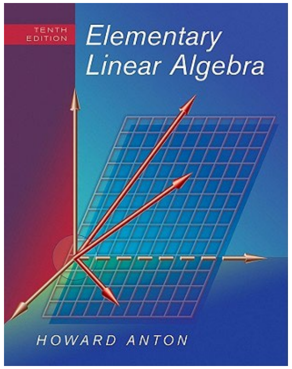
$$(\diamond)_n \to \mathsf{Pointwise}_{n\times n,(a,b)\mapsto a\diamond b}, \quad \diamond \in \{+,\ -\cdot,\ \wedge,\ \vee, \dots\}$$

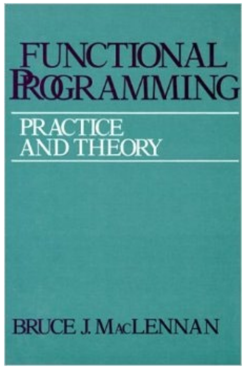
$$||.||_\infty^n \to \mathsf{Reduction}_{n,(a,b)\mapsto \mathsf{max}(|a|,|b|)}$$

$$< .,. >_n \to \mathsf{Reduction}_{n,(a,b)\mapsto a+b} \circ \mathsf{Pointwise}_{n\times n,(a,b)\mapsto ab}$$

$$P[x,(a_0,\dots,a_n)] \to < (a_0,\dots,a_n),. > \circ (x^i)_n$$

$$(x^i)_n \to \mathsf{Induction}_{n,(a,b)\mapsto ab,1}$$





Library of well-known identities expressed in OL

More Information:

www.spiral.net

www.spiralgen.com