SPIRAL:
AI for High Performance Code

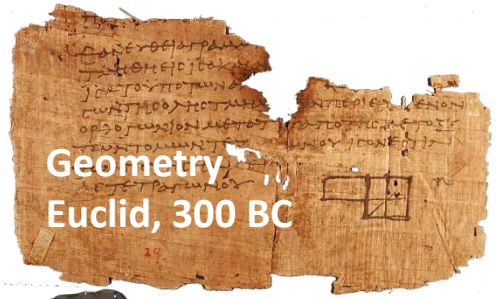
Franz Franchetti

Department of Electrical and Computer Engineering
Carnegie Mellon University
www.ece.cmu.edu/~franzf

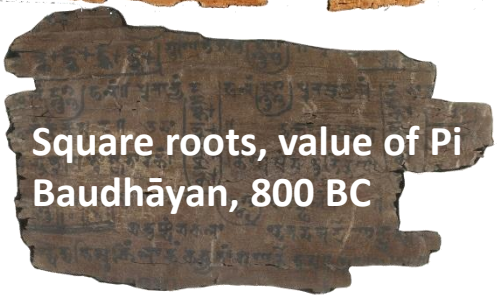
Joint work with the SPIRAL team at CMU, UIUC, Drexel, SpiralGen, Inc., and collaborators at LBL, ORNL, and LANL

This work was supported by DARPA, DOE, ONR, NSF, Intel, Mercury, SRC, and Nvidia

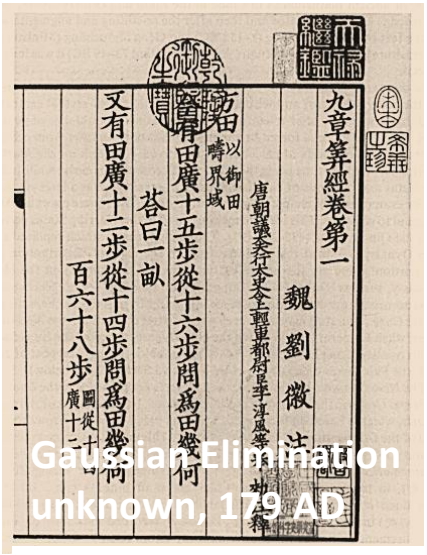
Algorithms and Mathematics: 2,500+ Years



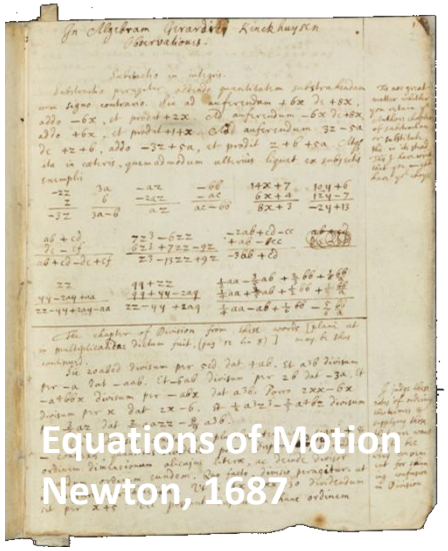
Geometry
Euclid, 300 BC



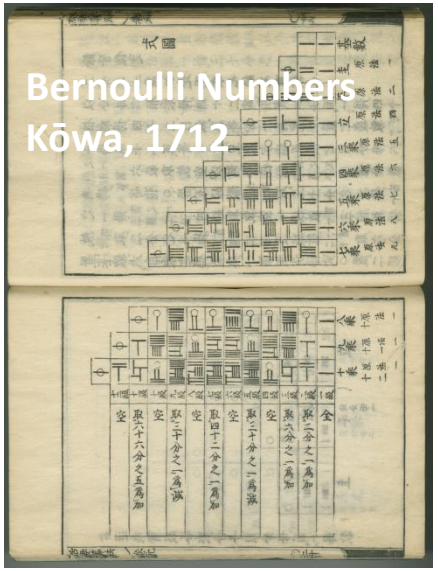
Square roots, value of Pi
Baudhāyan, 800 BC



Gaussian Elimination
unknown, 179 AD



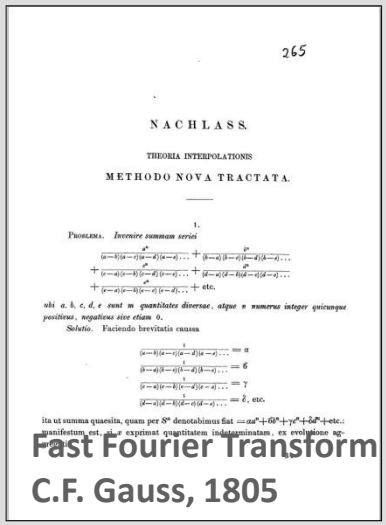
Equations of Motion
Newton, 1687



Bernoulli Numbers
Kōwa, 1712

Algebra
al-Khwārizmī, 830

Fast Fourier Transform

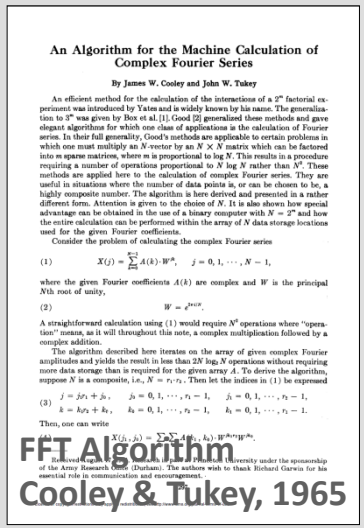


NACHLASS.

THEORIA INTERPOLATIONIS

METHODO NOVA TRACTATA.

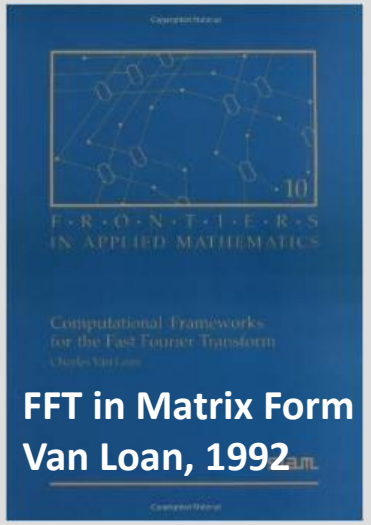
Fast Fourier Transform
C.F. Gauss, 1805



An Algorithm for the Machine Calculation of Complex Fourier Series

By James W. Cooley and John W. Tukey

FFT Algorithm
Cooley & Tukey, 1965



FFT in Matrix Form
Van Loan, 1992

Moore's Law in Practice

1 flop/s = one floating-point operation (addition or multiplication) per second
mega (M) = 10^6, giga (G) = 10^9, tera (T) = 10^{12}, peta (P) = 10^{15}, exa (E) = 10^{18}

In 2025…

Cell phone
15 Gflop/s

Laptop
120 Gflop/s

Workstation
32 Tflop/s (2xCPU)

AI Server
3.2 Pflop/s (72xGPU)

#1 supercomputer
2.7 Eflop/s

…would have been the #1 supercomputer back in…

Cray Y-MP C90
16 Gflop/s
1991

CM-5/1024
131 Gflop/s
1993

Earth Simulator
41 Tflop/s
2002

Tianhe-1A
4.7 Pflop/s
2010

But: Language Adoption is Slow

Programming languages

- 1953: Fortran
- 1973: C
- 1985: C++
- 1997: OpenMP
- 2007: CUDA
- 2009: OpenCL

Performance libraries

- 1979: BLAS
- 1992: LAPACK
- 1994: MPI
- 1995: ScaLAPACK
- 1995: PETSc
- 1997: FFTW

Productivity/scripting languages

- 1987: Perl
- 1989: Python
- 1993: Ruby
- 1995: Java
- 2000: C#

Big Data and ML Frameworks

- 2004: MapReduce
- 2005: Hadoop
- 2009: Spark
- 2015: TensorFlow
- 2016: PyTorch

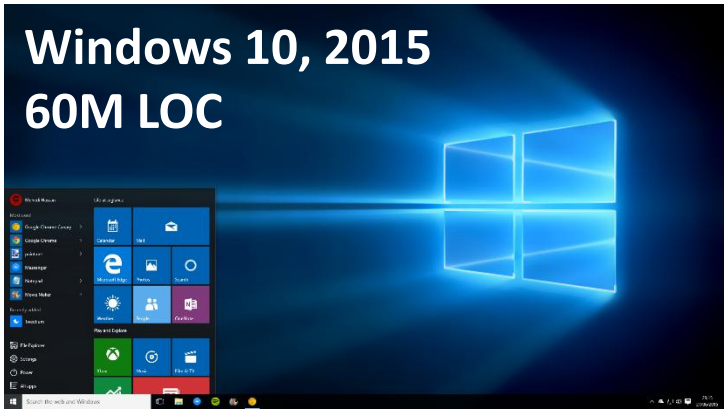
Symbolic and Numerical PSEs

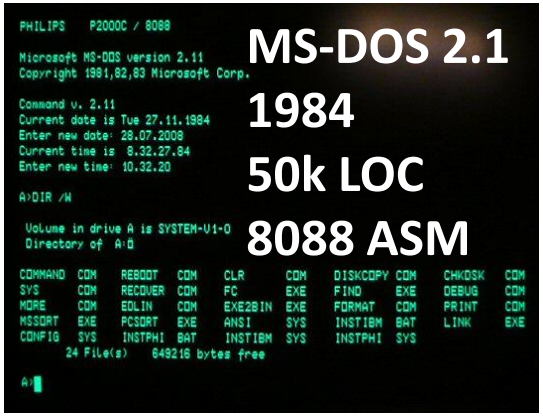
- 1958: LISP
- 1972: Prolog
- 1984: Matlab
- 1982: Maple
- 1988: Mathematica
- 1990: Haskell
- 1993: R

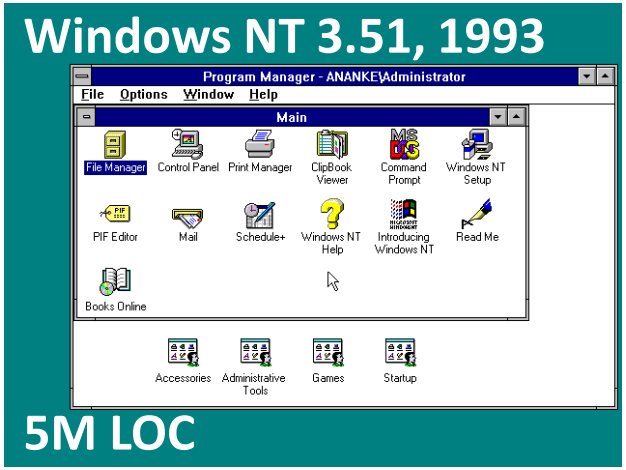
Numerical Mathematics libraries

- 1970: IMSL
- 1971: NAG
- 2001: GNU Scientific Library
- 2003: Intel Math Kernel Library

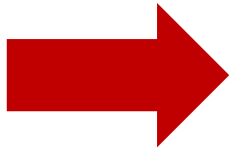
Explosive Growth of Source Code

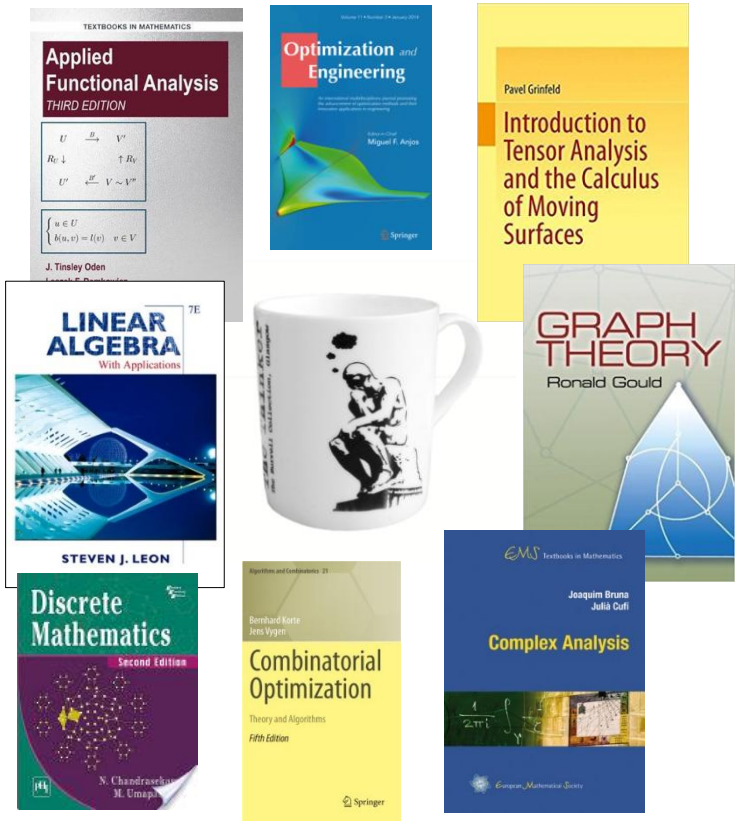






AI For Performance Engineering





How to maintain correctness? Raising abstraction is key

Example: Hockney Convolution for PDEs

Numerical algorithm view

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

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

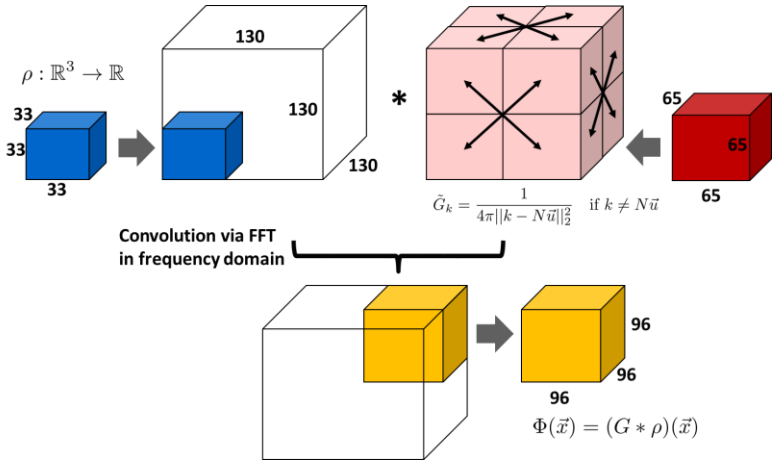
Solution: ϕ(.) = convolution of RHS ρ(.) with Green's function *G*(.). Efficient through FFTs (frequency domain)

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

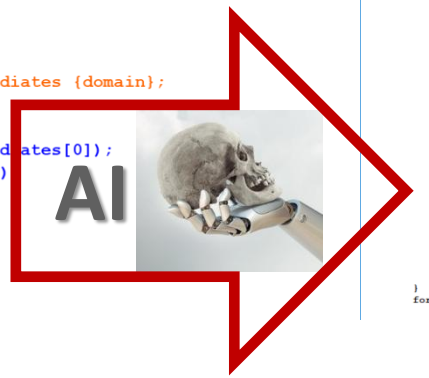
Whiteboard view



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]);
}
```

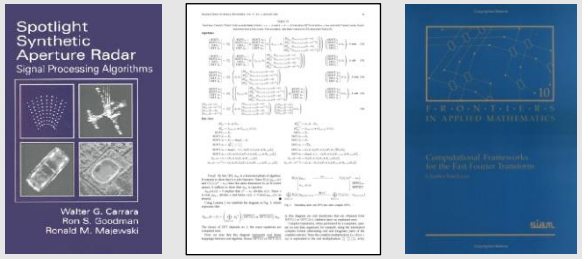
"classical compiler"



Backend program view

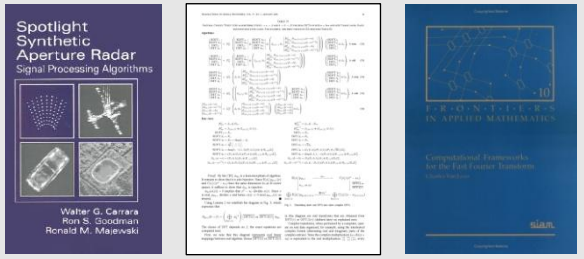
SPIRAL: AI for High Performance Code

Traditionally



High performance library optimized for given platform

SPIRAL Approach

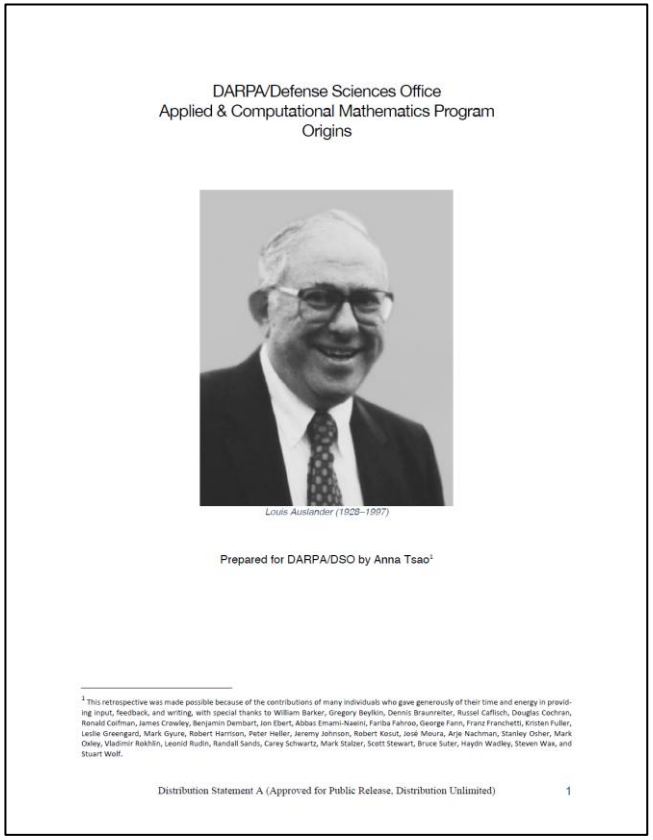


SPIRAL

High performance library optimized for given platform

Comparable performance

SPIRAL's History: The Long Arc of Math in CS



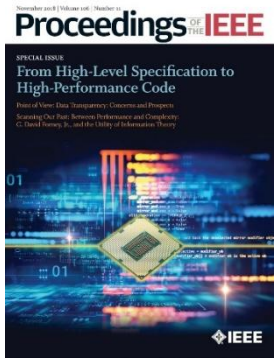
DARPA/Defense Sciences Office
Applied & Computational Mathematics Program
Origins

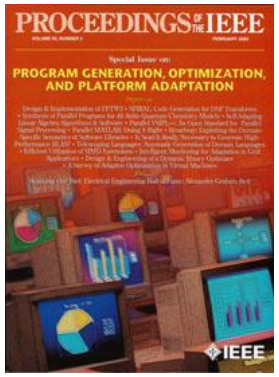
Louis Auslander (1928–1997)

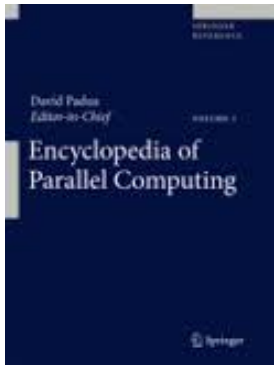
Prepared for DARPA/DSO by Anna Tsao[1]

[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







Project ongoing since 1998, core idea dates back to 1968

Outline

- Introduction
- **Specifying computation**
- Achieving Performance Portability
- SPIRAL and Generative and Agentic AI
- Other Current Work

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*

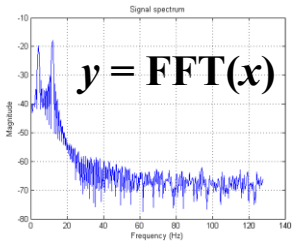
SPIRAL: AI for Performance Engineering

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));
    ...
}
```

OL Operators

Definition

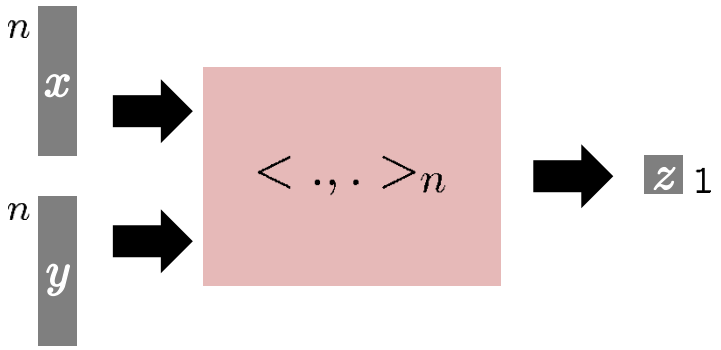
- Operator: Multiple vectors → Multiple vectors
- Stateless
- Higher-dimensional data is linearized
- Operators are potentially nonlinear

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

Example: Scalar product

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

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



Breaking Down Operators into Expressions

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

$$\text{SafeDist}_{V,A,b,\varepsilon}(.,.,.) \rightarrow \left(P[x,(a_0,a_1,a_2)](.) < d^2_\infty(.,.)\right)(.,.,.)$$

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

Problem specification: hand-developed or automatically produced

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

$$d^n_\infty(.,.) \rightarrow \|.\|^n_\infty \circ (-)_n$$

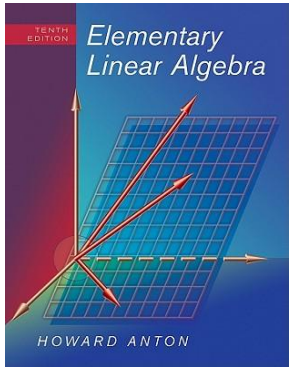
$$(\diamond)_n \rightarrow \text{Pointwise}_{n\times n,(a,b)\mapsto a\diamond b}, \quad \diamond \in \{+, -\cdot, \wedge, \vee, \dots\}$$

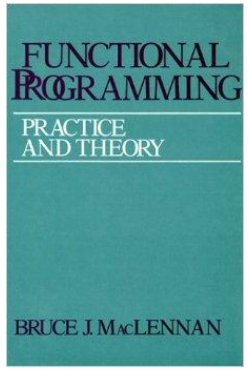
$$\|.\|^n_\infty \rightarrow \text{Reduction}_{n,(a,b)\mapsto \max(|a|,|b|)}$$

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

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

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





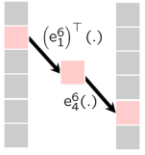
Library of well-known identities expressed in OL

Loop and Code Level Rule System

Mathematical Loop Abstraction

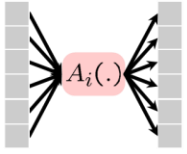
- **Selection and embedding operator:** *gather and scatter*

$$(\mathrm{e}_i^n)^\top(.) : \mathbb{R}^n \to \mathbb{R}^1$$
$$(x_i)_{i=0,\dots,n-1} \mapsto x_i$$
$$\mathrm{e}_i^n(.) : \mathbb{R}^1 \to \mathbb{R}^n$$
$$(x) \mapsto (0,\dots,0,\underbrace{x}_{i\text{th}},0,\dots,0)$$



- **Iterative operations:** *loop*

$$\bigsqcup_{i=0}^{n-1} : (D \to R)^n \to (D \to R)$$
$$A_i \mapsto (x \mapsto A_0(x) \sqcup \cdots \sqcup A_{n-1}(x))$$
$$\text{with } \sqcup \in \{\Sigma, \vee, \wedge, \prod, \min, \max, \dots\}$$



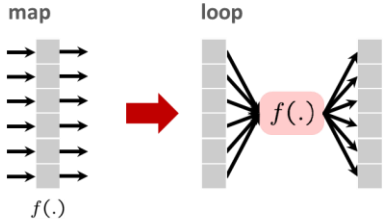
- **Atomic operators:** *nonlinear scalar functions*

$$\mathrm{Atomic}_f : \mathbb{R}^1 \to \mathbb{R}^1$$
$$(x) \mapsto (f(x))$$

Translation and Optimization

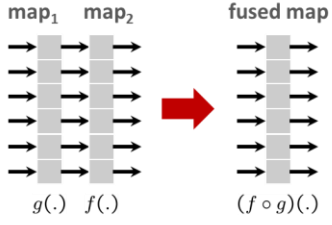
- **Translating Basic OL into Σ-OL**

$$\mathrm{Pointwise}_{n,f_i} \to \sum_{i=0}^{n-1} \left(\mathrm{e}_i^n \circ \mathrm{Atomic}_{f_i} \circ (\mathrm{e}_i^n)^\top\right)$$
$$\mathrm{Reduction}_{n,(a,b)\mapsto a+b} \to \sum_{i=0}^{n-1} (\mathrm{e}_i^n)^\top$$



- **Optimizing Basic OL/Σ-OL**

$$\mathrm{Pointwise}_{n,f_i} \circ \mathrm{Pointwise}_{n,g_i} \to \mathrm{Pointwise}_{n,f_i \circ g_i}$$
$$\mathrm{Pointwise}_{n,f_i} \circ \mathrm{e}_n^j \to \mathrm{e}_n^j \circ \mathrm{Pointwise}_{1,f_j}$$



Abstract Code

Code objects

- Values and types
- Arithmetic operations
- Logic operations
- Constants, arrays and scalar variables
- Assignments and control flow

Properties: at the same time

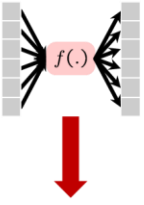
- Program = (abstract syntax) tree
- Represents program in restricted C
- OL operator over real numbers and machine numbers (floating-point)
- Pure functional interpretation
- Represents lambda expression

```
# Dynamic Window Monitor

let(
  i3 := var("i3", TInt), i5 := var("i5", TInt),
  w2 := var("w2", TBool), w1 := var("w1", T_Real(64)),
  s8 := var("s8", T_Real(64)), s7 := var("s7", T_Real(64)),
  s6 := var("s6", T_Real(64)), s5 := var("s5", T_Real(64)),
  s4 := var("s4", T_Real(64)), s1 := var("s1", T_Real(64)),
  q4 := var("q4", T_Real(64)), q3 := var("q3", T_Real(64)),
  D := var("D", TPtr(T_Real(64)).aligned([16, 0])),
  X := var("X", TPtr(T_Real(64)).aligned([16, 0])),

  func(TInt, "dwmonitor", [ X, D ],
    decl([q3, q4, s1, s4, s5, s6, s7, s8, w1, w2],
      chain(
        assign(s5, V(0.0)),
        assign(s8, nth(X, V(0))),
        assign(s7, V(1.0)),
        loop(i5, [0..2],
          chain(
            assign(s4, mul(s7, nth(D, i5))),
            assign(s5, add(s5, s4)),
            assign(s7, mul(s7, s8))
          )
        ),
        assign(s1, V(0.0)),
        loop(i3, [0..1],
          chain(
            assign(q3, nth(X, add(i3, V(1)))),
            assign(q4, nth(X, add(V(3), i3))),
            assign(w1, sub(q3, q4)),
            assign(s6, cond(geq(w1, V(0)), w1, neg(w1))),
            assign(s1, cond(geq(s1, s6), s1, s6))
          )
        ),
        assign(w2, geq(s1, s5)),
        creturn(w2)
      )
    )
  )
)
```

Rule Based Compiler

Compilation rules: recursive descent

$$\mathrm{Code}\left(y = (A \circ B)(x)\right) \to \left\{\mathtt{decl}(t), \mathrm{Code}\left(t = B(x)\right), \mathrm{Code}\left(y = A(t)\right)\right\}$$
$$\mathrm{Code}\left(y = \left(\sum_{i=0}^{n-1} A_i\right)(x)\right) \to \left\{y := \vec{0}, \mathtt{for}(i = 0..n-1)\ \mathrm{Code}\left(y += A_i(x)\right)\right\}$$
$$\mathrm{Code}\left(y = (\mathrm{e}_i^n)^\top(x)\right) \to y[0] := x[i]$$
$$\mathrm{Code}\left(y = \mathrm{e}_i^n(x)\right) \to \left\{y = \vec{0}, y[i] := x[0]\right\}$$
$$\mathrm{Code}\left(y = \mathrm{Atomic}_f(x)\right) \to y[0] := f(x[i])$$

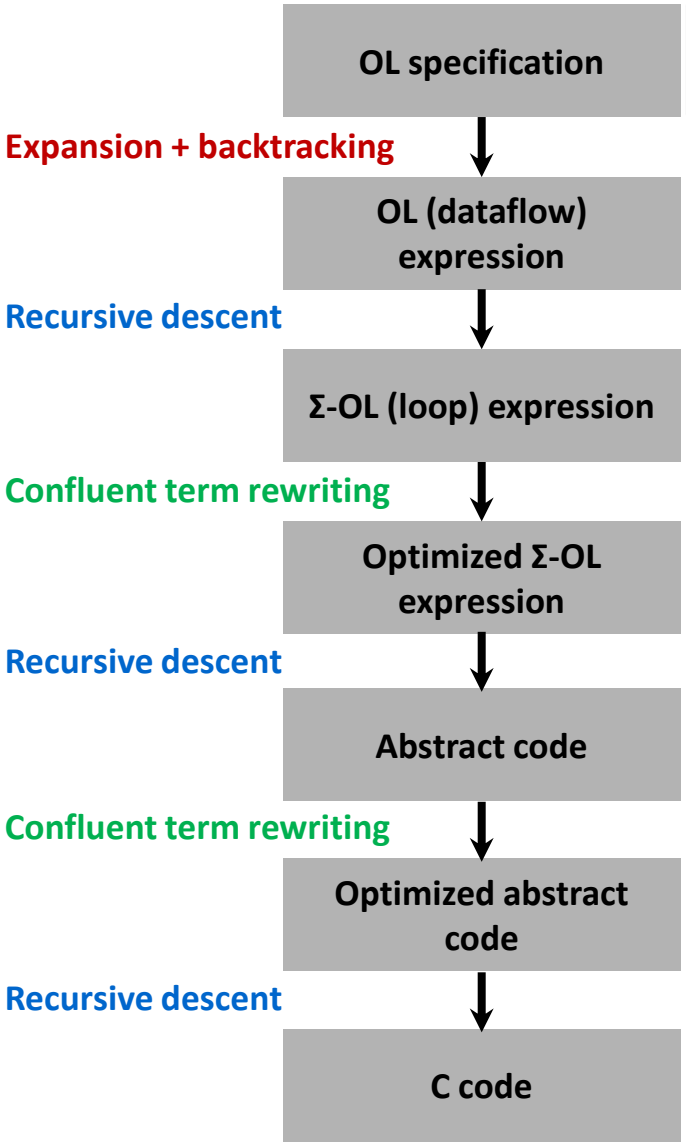



```
chain(
  assign(Y, V(0.0),
  loop(i1, [0..5],
    assign(nth(y, i1),
      f(nth(X, i1)))
  )
)
```

Cleanup rules: term rewriting

```
chain(a, chain(b))    → chain([a, b])
decl(D, decl(E, c))   → decl([D, E], c)
loop(i, decl(D, c))   → decl(D, loop(i, c))
chain(a, decl(D, b))  → decl(D, chain([a, b]))
```

Putting it Together: One Big Rule System



Mathematical specification

$$\text{SafeDist}_{V,A,b,\varepsilon}(.,.,.) \rightarrow \left(P[x,(a_0,a_1,a_2)](.) < d_\infty^2(.,.)\right)(.,.,.)$$

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

Final code

```
int dwmonitor(float  *X, double  *D) {
    __m128d u1, u2, u3, u4, u5, u6, u7, u8 , x1, x10, x13, x14, x17
    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_
    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_lo
        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)),
        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_SHUFF
```

Inspiration: Symbolic Integration

- **Rule based AI system**
  basic functions, substitution
- **May not succeed**
  not all expressions can be symbolically integrated
- **Arbitrarily extensible**
  define new functions as integrals
  Γ(.), distributions, Lebesgue integral
- **Semantics preserving**
  rule chain = formal proof
- **Automation**
  Mathematica, Maple

Table of Integrals

BASIC FORMS

(1) $\int x^n dx = \frac{1}{n+1}x^{n+1}$

(2) $\int \frac{1}{x}dx = \ln x$

(3) $\int u dv = uv - \int v du$

(4) $\int u(x)v'(x)dx = u(x)v(x) - \int v(x)u'(x)dx$

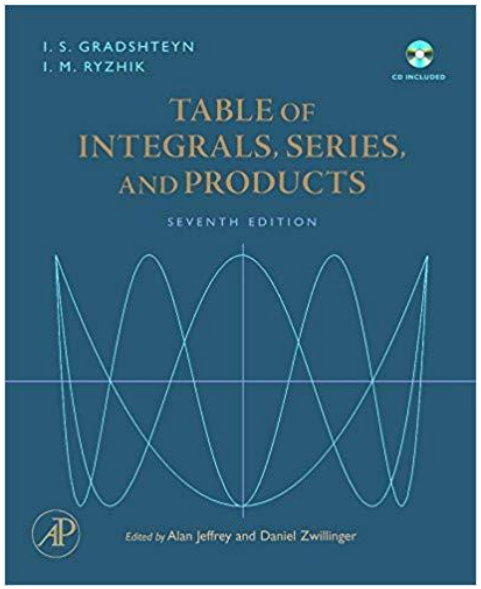
RATIONAL FUNCTIONS

(5) $\int \frac{1}{ax+b}dx = \frac{1}{a}\ln(ax+b)$

(6) $\int \frac{1}{(x+a)^2}dx = \frac{-1}{x+a}$

(7) $\int (x+a)^n dx = (x+a)^n\left(\frac{a}{1+n}+\frac{x}{1+n}\right),\ n \neq -1$

(8) $\int x(x+a)^n dx = \frac{(x+a)^{1+n}(nx+x-a)}{(n+2)(n+1)}$



In[31]:=
$$\int_0^{2\pi} \frac{1}{a^2\,\text{Cos}[t]^2 + b^2\,\text{Sin}[t]^2}\,dt$$

Out[31]=
$$\frac{2\sqrt{\frac{b^2}{a^2}}\,\pi}{b^2}$$

Wolfram Mathematica

In[33]:=
$$\int_0^{2\pi} \frac{1}{a^2\left(\frac{e^{i t}+e^{-i t}}{2}\right)^2 + b^2\left(\frac{e^{i t}-e^{-i t}}{2i}\right)^2}\,dt$$

Out[33]= 0

Outline

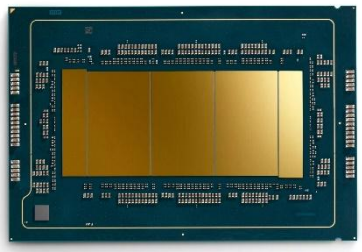
- Introduction
- Specifying computation
- **Achieving Performance Portability**
- SPIRAL and Generative and Agentic AI
- Other Current Work

M. Püschel, J. Moura, J. Johnson, D. Padua, M. Veloso, B. Singer, J. Xiong, F. Franchetti, A. Gacic, Y. Voronenko, K. Chen, R. W. Johnson, N. Rizzolo:
SPIRAL: Code Generation for DSP Transforms
Proceedings of the IEEE Special Issue on "Program Generation, Optimization, and Adaptation," Vol. 93, No. 2, 2005, pages 232-275.

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*

Today's Computing Landscape

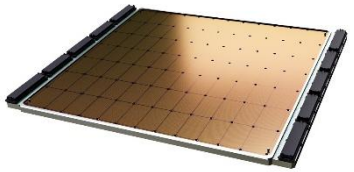
1 Gflop/s = one billion floating-point operations (additions or multiplications) per second



Intel Xeon 6980P
16 Tflop/s, 500 W
128 cores, 2 – 3.9 GHz
2-way—16-way AVX-512

IBM POWER10
7.5 Tflop/s, 130 W
30 cores, 4 GHz
4-way VSX-3, MMA

Nvidia H200
34 Tflop/s, 700 W
16,896 cores, 1.41 GHz
2 Pflop/s FP16 tensor cores

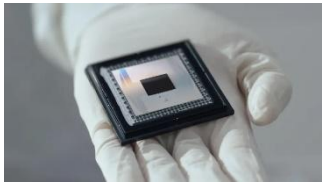


Cerebras WSE3
12.5 *Pflop/s 20kW*
900,000 cores

Snapdragon 8+ Gen1
15 Gflop/s, 2 W
8 cores, 3.2 GHz
A730 GPU, Hexagon DSP

Dell PowerEdge R960
30 Tflop/s, 8 TB, 1.5kW
4x 60 cores, 1.9 – 3.5 GHz
2-way – 16-way AVX512

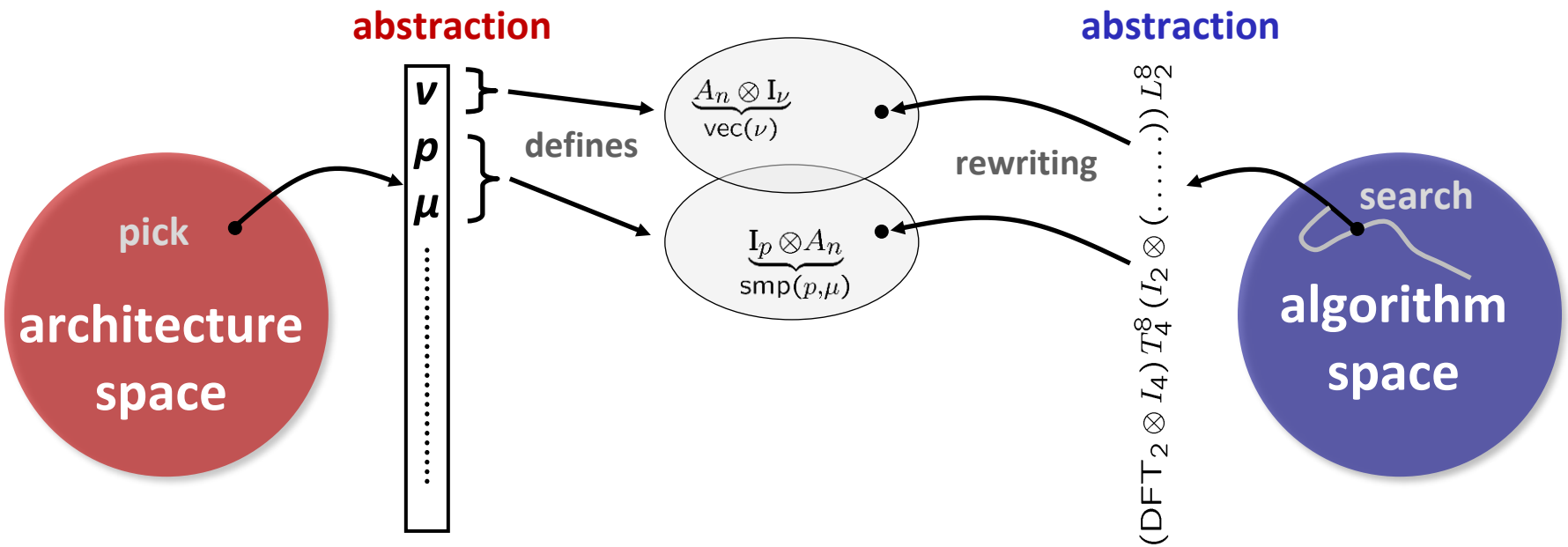
El Capitan
2.7 Eflop/s, 30MW
43k 24-core CPUs + 43k GPUs
#1 in Top500



Google Willow
105 qubits

Platform-Aware Formal Program Synthesis

Model: common abstraction
= spaces of matching formulas



Architectural parameter:
Vector length,
#processors, …

optimization

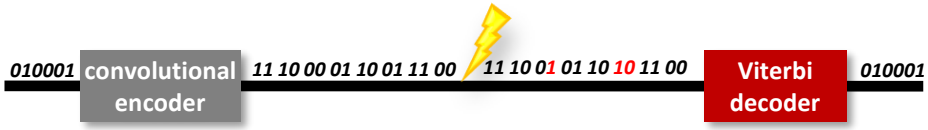
Kernel:
problem size,
algorithm choice

Some Application Domains in OL

Linear Transforms

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

Software Defined Radio

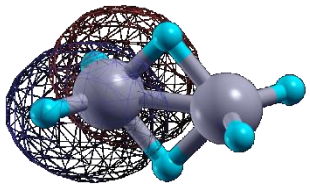


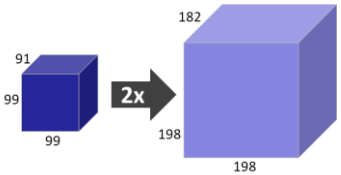
$$F_{K,F} \rightarrow \prod_{i=1}^{F} \left((I_{2^{K-2}} \otimes_j B_{F-i,j}) L^{2^{K-1}}_{2^{K-2}} \right)$$

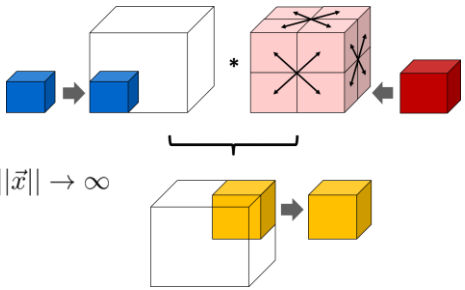
$$\underline{\mathbf{F}_{K,F}}\ \nu \rightarrow \prod_{i=1}^{F} \left(\left(\mathrm{I}_{2^{K-2}/\nu} \otimes_{j_1} \vec{\mathrm{L}}^{2\nu}_{\nu} \vec{B}^{\nu}_{F-i,j_1} \right) (\mathrm{L}^{2^{K-1}/\nu}_{2^{K-2}/\nu} \bar{\otimes} \mathrm{I}_\nu) \right)$$

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

PDEs/HPC Simulations





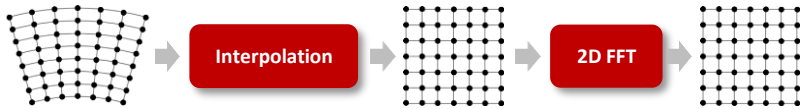


$$\Phi : \mathbb{R}^3 \rightarrow \mathbb{R}$$

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

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

Synthetic Aperture Radar (SAR)



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

Formal Approach for all Types of Parallelism

- **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)

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

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

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

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

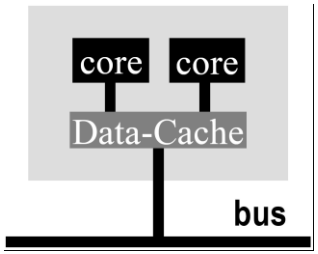
$$\prod_{i=0}^{n-1} A_i, \quad A_n \hat{\otimes} \mathrm{I}_w, \quad P_n \otimes Q_w$$

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

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

Modeling Hardware: Base Cases

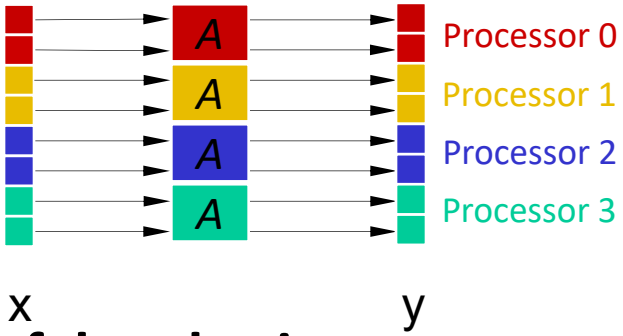
- **Hardware abstraction: shared cache with cache lines**



$$\underbrace{A_{k\times m\to n}}_{\mathrm{smp}(p,\mu)}$$

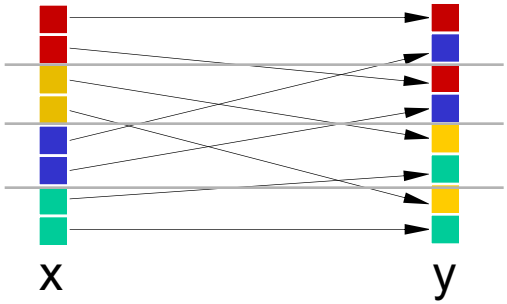
- **Tensor product: embarrassingly parallel operator**

$$y = \left(\mathrm{I}_p \otimes A\right)(x)$$



- **Permutation: problematic; may produce false sharing**

$$y = \mathrm{L}^{8}_{4}(x)$$



Example Program Transformation Rule Set

$$\underbrace{AB}_{\mathsf{smp}(p,\mu)} \rightarrow \underbrace{A}_{\mathsf{smp}(p,\mu)}\ \underbrace{B}_{\mathsf{smp}(p,\mu)}$$

$$\underbrace{A_m \otimes \mathrm{I}_n}_{\mathsf{smp}(p,\mu)} \rightarrow \underbrace{\left(\mathrm{L}^{mp}_{m} \otimes \mathrm{I}_{n/p}\right)\left(\mathrm{I}_p \otimes (A_m \otimes \mathrm{I}_{n/p})\right)\left(\mathrm{L}^{mp}_{p} \otimes \mathrm{I}_{n/p}\right)}_{\mathsf{smp}(p,\mu)}$$

$$\underbrace{\mathrm{L}^{mn}_{m}}_{\mathsf{smp}(p,\mu)} \rightarrow \begin{cases} \underbrace{\left(\mathrm{I}_p \otimes \mathrm{L}^{mn/p}_{m/p}\right)}_{\mathsf{smp}(p,\mu)}\underbrace{\left(\mathrm{L}^{pn}_{p} \otimes \mathrm{I}_{m/p}\right)}_{\mathsf{smp}(p,\mu)} \\ \underbrace{\left(\mathrm{L}^{pm}_{m} \otimes \mathrm{I}_{n/p}\right)}_{\mathsf{smp}(p,\mu)}\underbrace{\left(\mathrm{I}_p \otimes \mathrm{L}^{mn/p}_{m}\right)}_{\mathsf{smp}(p,\mu)} \end{cases}$$

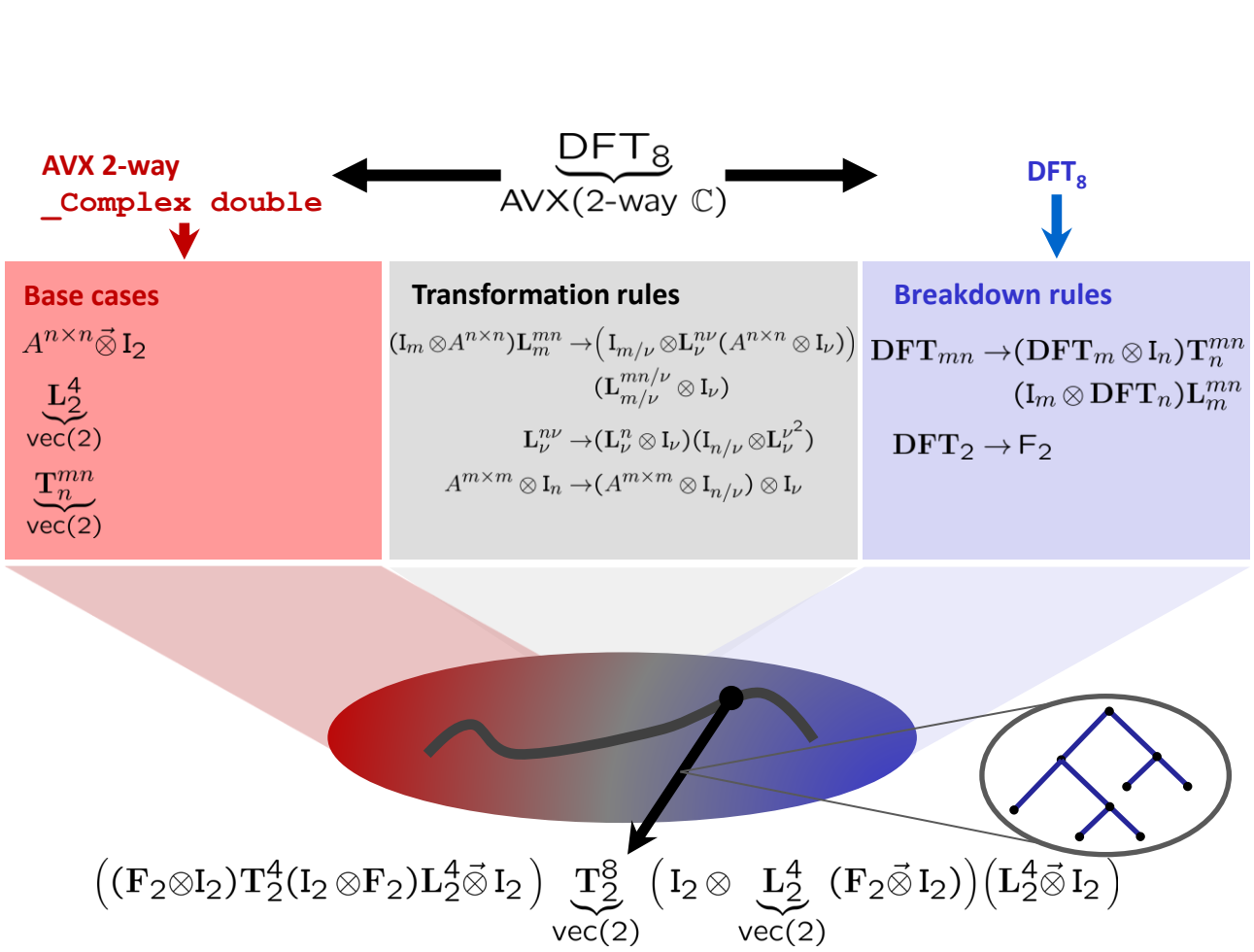
Recursive rules

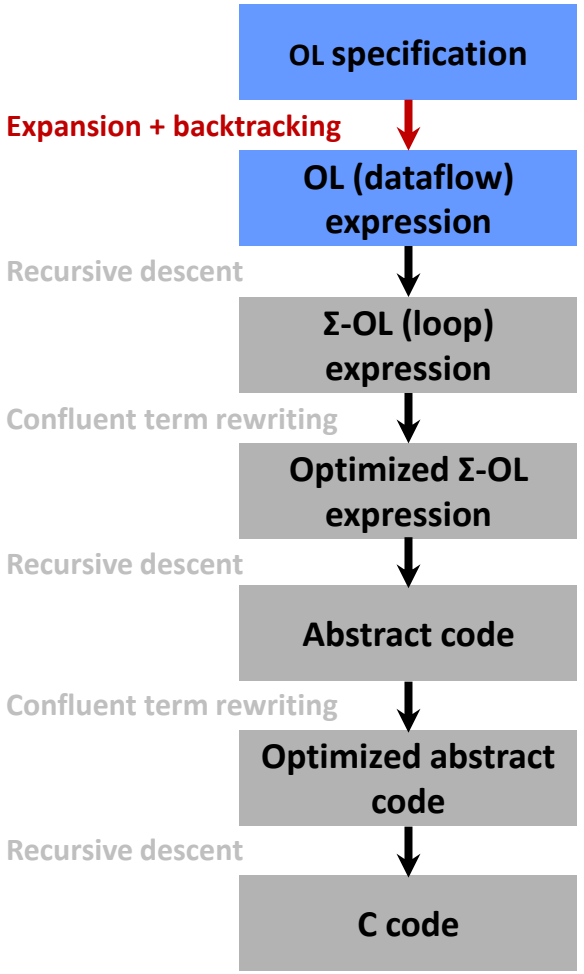
$$\underbrace{\mathrm{I}_m \otimes A_n}_{\mathsf{smp}(p,\mu)} \rightarrow \mathrm{I}_p \otimes_{\|} \left(\mathrm{I}_{m/p} \otimes A_n\right)$$

$$\underbrace{(P \otimes \mathrm{I}_n)}_{\mathsf{smp}(p,\mu)} \rightarrow \left(P \otimes \mathrm{I}_{n/\mu}\right) \overline{\otimes}\, \mathrm{I}_\mu$$

Base case rules

Autotuning in Constraint Solution Space





Translating an OL Expression Into Code

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

Output =

Ruletree, expanded into

OL Expression:

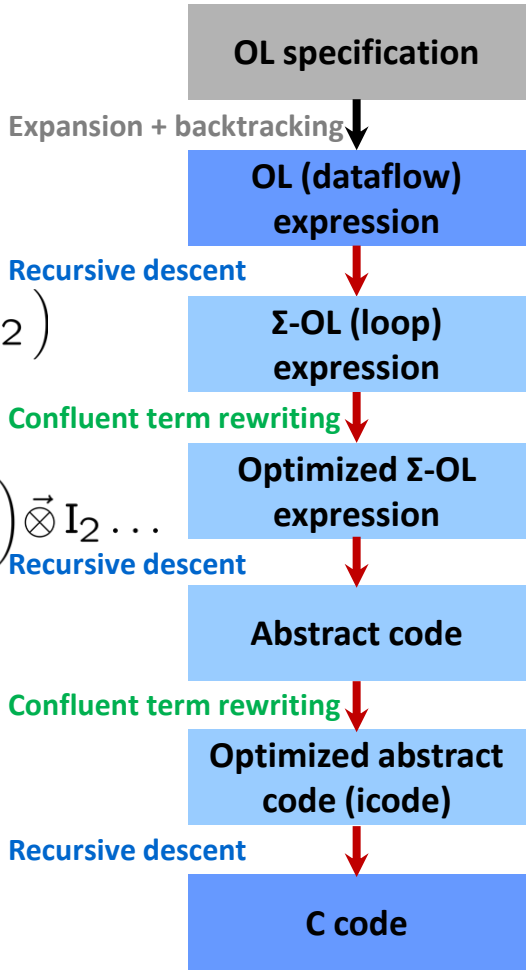
$$\left((F_2 \otimes I_2) T^4_2 (I_2 \otimes F_2) L^4_2 \vec{\otimes} I_2\right) \underbrace{T^8_2}_{vec(2)} \left(I_2 \otimes \underbrace{L^4_2}_{vec(2)} (F_2 \vec{\otimes} I_2)\right)\left(L^4_2 \vec{\otimes} I_2\right)$$

∑-OL:

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

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;
}
```



Symbolic Verification for Linear Operators

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

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

= ?

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

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

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & j & -1 & -j \\ 1 & -1 & 1 & -1 \\ 1 & -j & -1 & j \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

= ? `DFT4([0,1,0,0])`

Symbolic evaluation and symbolic execution establishes correctness

Outline

- Introduction
- Specifying computation
- Achieving Performance Portability
- **SPIRAL and Generative and Agentic AI**
- Other Current Work

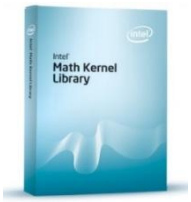
S. Rao: **LibraryX: A Framework for Cross-Library-Call Optimization**, Ph.D. Thesis, 2025

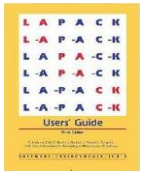
F. Franchetti, D. G. Spampinato, A. Kulkarni, D. T. Popovici, T. M. Low, M. Franusich, A. Canning, P. McCorquodale, B. Van Straalen, P. Colella: **FFTX and SpectralPack: A First Look**, Workshop on Parallel Fast Fourier Transforms (PFFT).

LLM Target: *Active* HPC Libraries as DSLs

Libraries/standards as DSL

- BLAS1/2/3, LAPACK
- GraphBLAS
- FFTW, FFTX, SuperLU
- OpenMP, MPI, OpenACC
- Boost, C++ STL, PETSc
- NumPy, SciPy wrappers





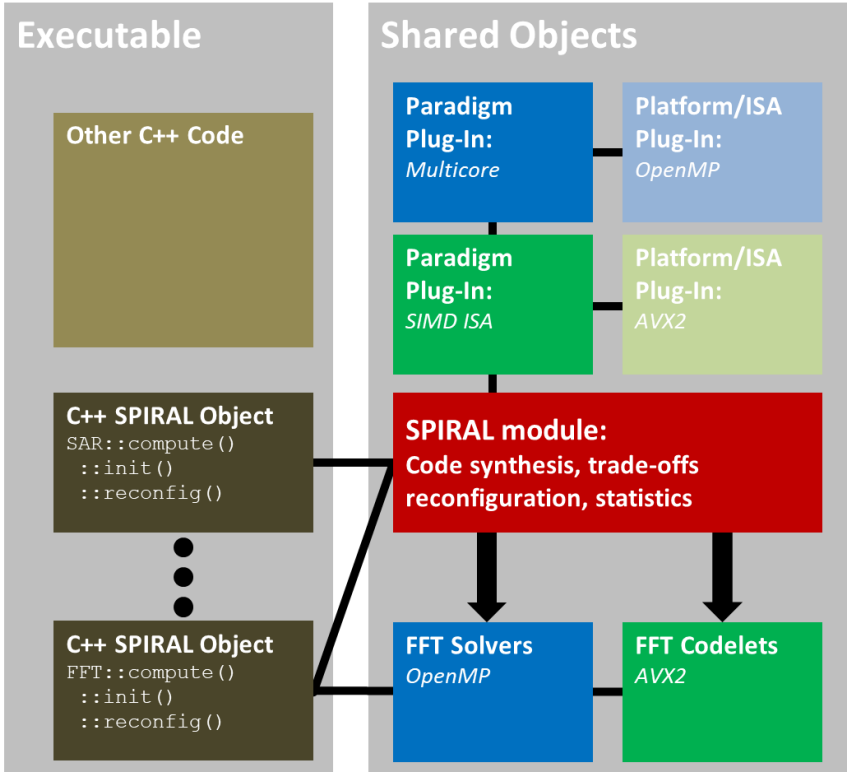
SciPy

FFTW

NumPy

OpenMP

Code Generation/JIT

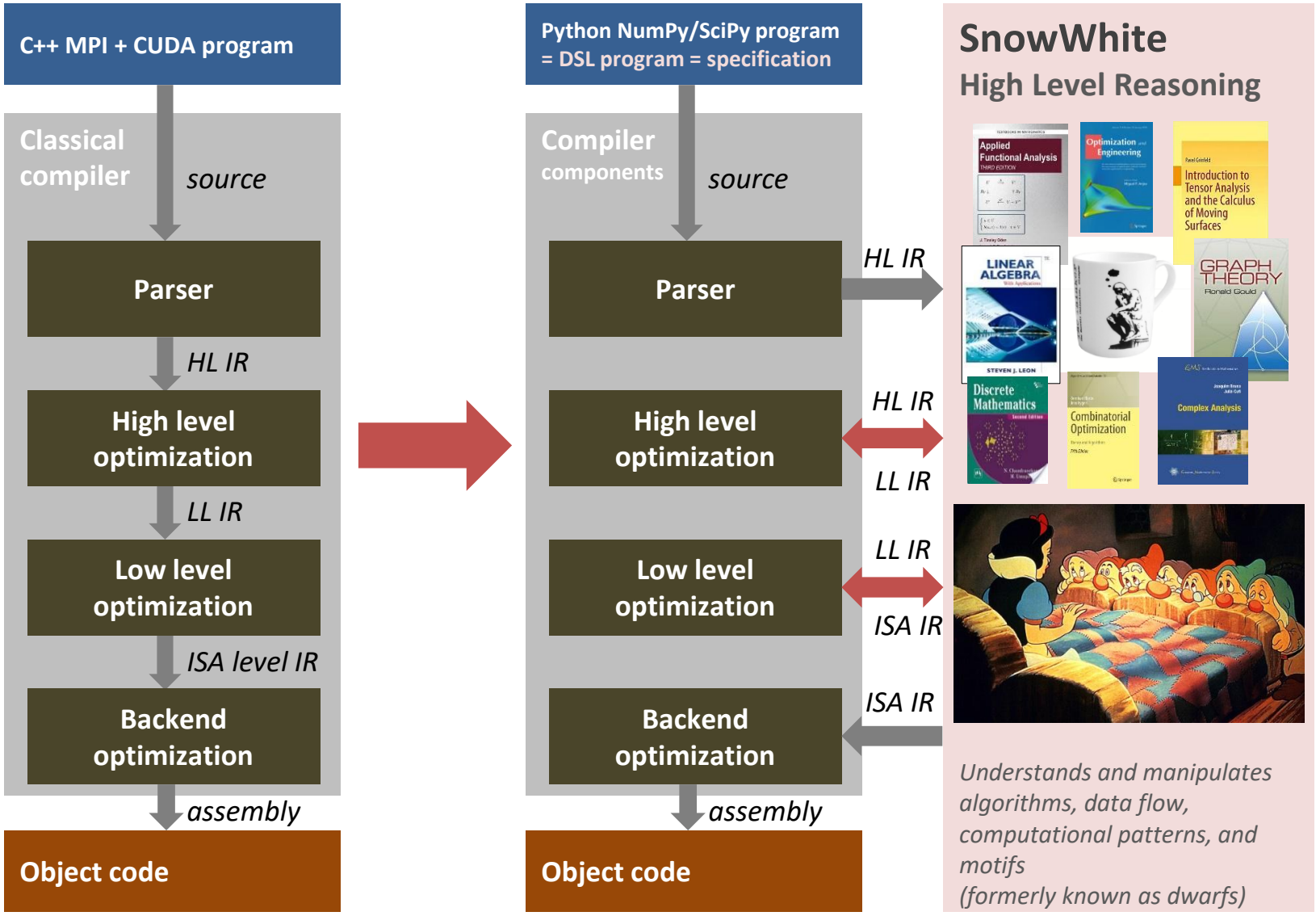


F. Franchetti, T. M. Low, D. T. Popovici, R. M. Veras, D. G. Spampinato, J. R. Johnson, M. Püschel, J. C. Hoe, J. M. F. Moura:
SPIRAL: Extreme Performance Portability, Proceedings of the IEEE, Vol. 106, No. 11, 2018.
Special Issue on *From High Level Specification to High Performance Code*

S. Rao, A. Kutuluru, P. Brouwer, S. McMillan, F. Franchetti: **GBTLX: A First Look,** IEEE High Performance Extreme Computing Conference (HPEC), 2020.

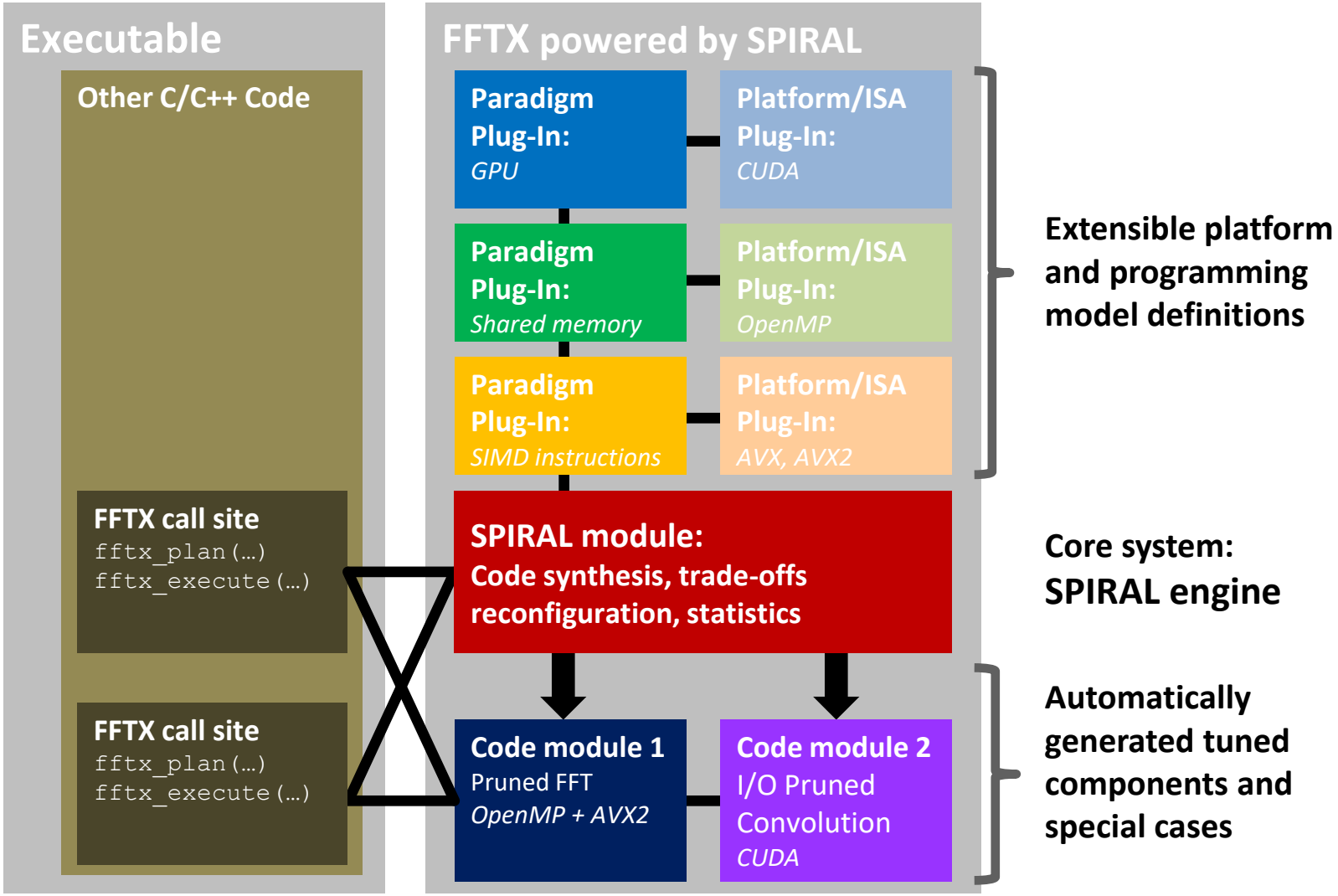
F. Franchetti, D. G. Spampinato, A. Kulkarni, D. T. Popovici, T. M. Low, M. Franusich, A. Canning, P. McCorquodale, B. Van Straalen, P. Colella: **FFTX and SpectralPack: A First Look,** Proceedings of the IEEE, Vol. 106, No. 11, 2018.

Agentic AI: High Level Reasoning in Compilers



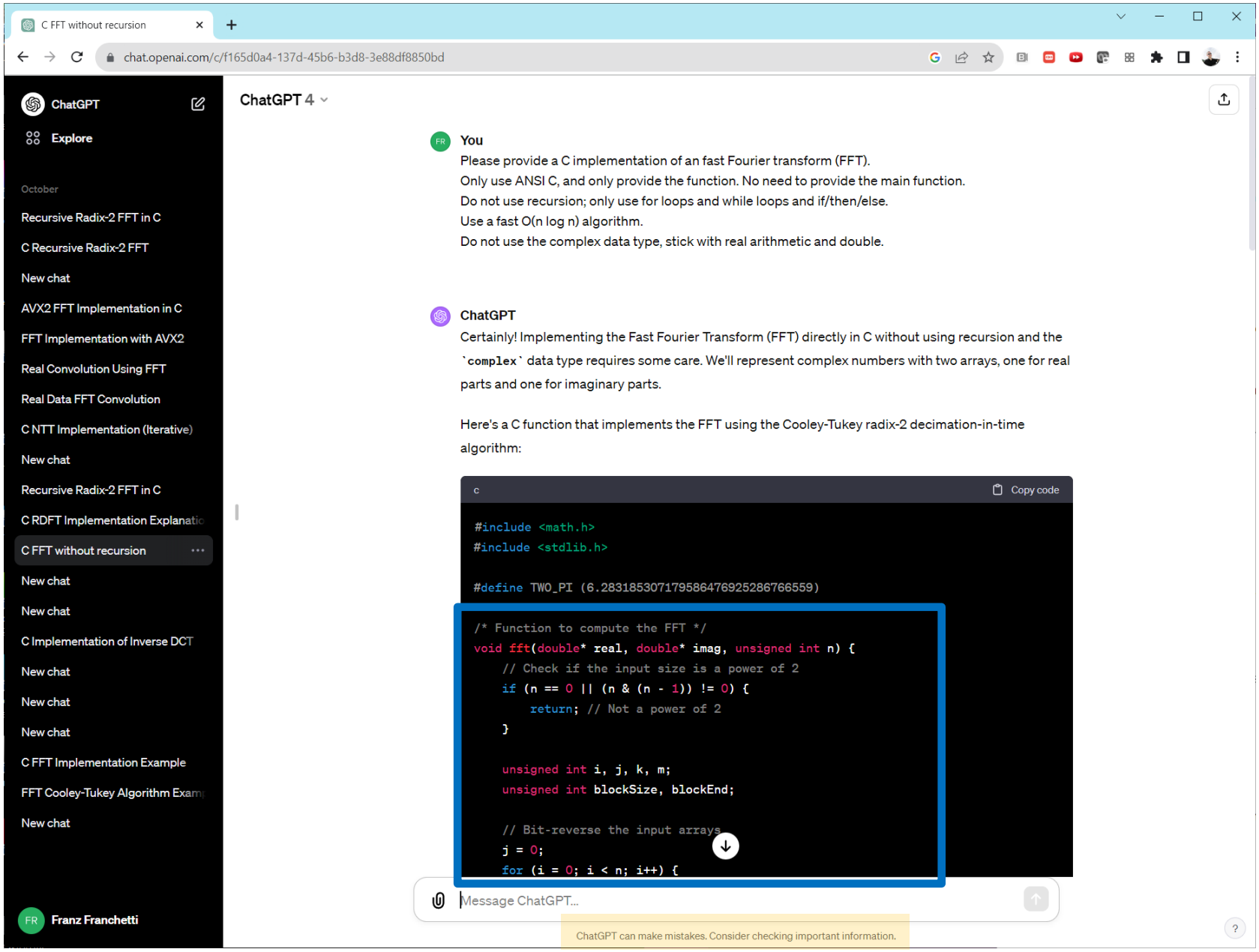
SPIRAL serves as AI Agent in connection with standard compilers

LibraryX: SPIRAL as Active Library Backend



LibraryX and C++/Python/Julia SPL API enable SPIRAL as Agent for GenAI

SPIRAL as Guardrail for Generative AI



ChatGPT, Meet SPIRAL*Prover*

Please provide a C implementation of an fast Fourier transform (FFT).
Only use ANSI C, and only provide the function. No need to provide the main function.
Do not use recursion; only use for loops and while loops and if/then/else.
Use a fast O(n log n) algorithm.
Do not use the complex data type, stick with real arithmetic and double.
Keep real and imaginary parts in one data vector.

ChatGPT

Command Prompt

```
spiral>

D:\franzf\Spiral\spiral-software\namespaces\packages\bluestone\nrproof\ChatGPT\fft>spiralprover fft.c
Calling Spiral

  http://www.spiral.net
  Spiral 8.5.1_Alpha
------------------------------------------------------------
```

```
            blockEnd = i + blockSize / 2;
            for (j = i; j < blockEnd; j += 1) {
                double temp;
                tReal = uReal * data[2 * j + blockSize / 2] - uImag * data[2 * j + blockSize / 2
                tImag = uReal * data[2 * j + blockSize / 2 + 1] + uImag * data[2 * j + blockSize
                data[2 * j + blockSize / 2] = data[2 * j] - tReal;
                data[2 * j + blockSize / 2 + 1] = data[2 * j + 1] - tImag;
                data[2 * j] += tReal;
                data[2 * j + 1] += tImag;
                temp = uReal * wReal - uImag * wImag;
                uImag = uReal * wImag + uImag * wReal;
                uReal = temp;
            }
        }
    }
}
spiral> spiral> spiral> spiral> spiral>
Symbolically executing fft(data, n), n = 8
spiral>
spiral> spiral> > > > > > Found sematics for function fft:
RC(
  DFT(8, 7)
)
spiral>

D:\franzf\Spiral\spiral-software\namespaces\packages\bluestone\nrproof\ChatGPT\fft>
```

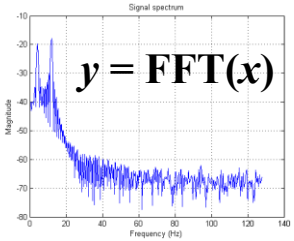
```
Symbolically executing fft(data, n), n = 8
spiral>
spiral> spiral> > > > > > Found sematics for function fft:
RC(
  DFT(8, 7)
)
```

SPIRAL found the semantics of `fft()` via *static analysis*

Also: Dusty Deck, Meet SPIRAL*Prover*

```
spiral> spiral> > > > > > Found sematics for function four1:
RC(
  DFT(16, 1)
)
spiral>
```

SPIRAL found the semantics of `four1()` via *static analysis*



$$x \mapsto y = T \cdot x$$

$$\mathrm{DFT}_n = [e^{-2k\ell\pi i/n}]_{0 \le k,\ell < n}$$

```
Command Prompt
        wpr = -2.0 * wtemp * wtemp;
        wpi = sin(theta);
        wr = 1.0;
        wi = 0.0;
        for (m = 1; m < mmax; m += 2) {
            for (i = m; i <= nn; i += istep) {
                j = i + mmax;
                tempr = wr * data[j - 1] - wi * data[j];
                tempi = wr * data[j] + wi * data[j - 1];
                data[j - 1] = data[i - 1] - tempr;
                data[j] = data[i] - tempi;
                data[i - 1] += tempr;
                data[i] += tempi;
            }
            wr = (wtemp = wr) * wpr - wi * wpi + wr;
            wi = wi * wpr + wtemp * wpi + wi;
        }
        mmax = istep;
    }
}
Symbolically executing four1(data, n, isign), n = 16, isign = 1
spiral>
spiral> spiral> > > > > > Found sematics for function four1:
RC(
  DFT(16, 1)
)
spiral>
```

```
1   #include <math.h>
2   #define SWAP(a,b) tempr=(a);(a)=(b);(b)=tempr
3   void four1(float data[], unsigned long nn, int isign)
4   {
5           unsigned long n, mmax, m, j, istep,i;
6           double wtemp, wr, wpr, wpi, wi, theta;
7           float tempr, tempi;
8
9           n=nn << 1;
10          j=1;
11          for (i=1;i<n;i+=2){
12                  if (j > i) {
13                          SWAP(data[j],data[i]);
14                          SWAP(data[j+1],data[i+1]);
15                  }
16                  m=n >> 1;
17                  while (m >=2 && j > m) {
18                          j -= m;
19                          m >>= 1;
20                          }
21                          j += m;
22                  }
23                  /*here begins the Danielson-Lanczos section of the routine */
24                  mmax=2;
25                  while (n > mmax) {
26                          istep = mmax << 1;
27                          theta = isign*(6.28318530717959/mmax);
28                          wtemp=sin(0.5*theta);
                            wtemp=sin(0.5*theta);
                            wpr = -2.0*wtemp*wtemp;
                            wpi = sin(theta);
                            wr=1.0;
                            wi=0.0;
                            for (m=1;m<mmax;m+=2) {
                                    for (i=m;i<=n;i+=istep) {
                                            j = i+mmax;
                                            tempr = wr*data[j]-wi*data[j+1];
                                            tempi=wr*data[j+1]+wi*data[j];
                                            data[j]=data[i]-tempr;
                                            data[j+1]=data[i+1]-tempi;
                                            data[i] += tempr;
                                            data[i+1] += tempi;
                                    }
                                    wr = (wtemp=wr)*wpr-wi*wpi+wr;
                                    wi = wi*wpr+wtemp*wtemp*wpi+wi;
                            }
                            mmax=istep;
                    }
            }
    }
```

C and Fortran code from 1986

Electrical & Computer ENGINEERING

Early Results: HPEC and POPL Workshop

Towards Automated Reasoning Chains for Verification of LLM-Generated Scientific Code

Quentin Oschatz, *Carnegie Mellon University*, Pittsburgh, USA, qoschatz@cmu.edu
Naifeng Zhang, *Carnegie Mellon University*, Pittsburgh, USA, naifengz@cmu.edu
Mike Franusich, *SpiralGen, Inc.*, Pittsburgh, USA, mike.franusich@spiralgen.com
Franz Franchetti, *Carnegie Mellon University*, Pittsburgh, USA, franzf@cmu.edu

***Abstract*—With the rise of Large Language Model (LLM) generated code, including in domains like scientific computing, ensuring not only syntactical, but also mathematical correctness, has become a critical task. Traditional formal methods approaches often struggle with the ambiguity of floating-point code, and full symbolic execution is extremely costly and limited. We propose a chain-of-reasoning approach that iteratively lifts basic semantics from code into the SPIRAL system and then establishes numerical equivalency to the desired mathematical operation. Here, we leverage the ample mathematical knowledge already formalized in SPIRAL to enable the system to recognize not just different implementations of the same algorithm but fully separate approaches to solving the given problem. The chain establishes tight error bounds on the output of given code with respect to the true continuous solution it approximates, quantifying all sources of error. We demonstrate this approach by establishing the correctness of a pseudospectral solver for a simple 1-dimensional Poisson problem.**

***Index Terms*—Numerical analysis, partial differential equations (PDEs), scientific computing, SPIRAL, large language models (LLMs)**

I. Introduction

Large Language Models (LLMs) have increasingly become a widespread tool for neural code generation, despite their tendency towards hallucinations and mistakes [1], [2]. While improvements have been made to their ability to generate semantically correct code, this is often insufficient, especially when generating code in the space of scientific computing. Here, numerical errors in algorithm implementation can be easy to make, hard to catch, and even more difficult to diagnose, especially with the added complexity of evaluating floating-point code. This can be a problem not just when attempting to generate new scientific kernel code as explored by Valero-Lara et al. [3], but also when translating code written in languages such as FORTRAN to more modern codebases as is the goal of projects such as TRACTOR [4].

Leaning on the well-known insight in mathematics and computer science that proving a solution correct is far easier than finding one, we propose a multi-step, end-to-end system to verify the numerical correctness of given code.

This material is based upon work supported by the U.S. Department of Energy, Office of Science under Award No. DE-SC0025645, by the National Science Foundation under Award No. 2431265, and by the Defense Advanced Research Projects Agency (DARPA) under Agreement No. HR00112490517. Approved for public release: distribution is unlimited.

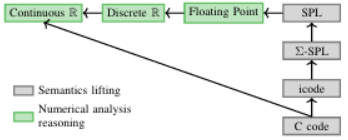


Fig. 1: Logical overview of the core steps in the pipeline connecting C code to the true, continuous solution of the given problem over the real numbers extending prior work [5].

Previous work [5] leveraged the SPIRAL system [6] to perform semantics lifting based on symbolic execution and program transformations. By restricting the problem domain to those understood by SPIRAL, Fast Fourier Transforms (FFTs) can be lifted from recursive C implementations. Reversing SPIRAL's rules-based, multi-tiered code generation system that traditionally lowers high level semantics into functioning code, the system can iteratively fuse operations into higher level functional blocks, until finally leveraging SPIRAL's database of linear transforms to recognize the components of a FFT. Due to the symbolic execution used for the fusion, minimal ambiguity is introduced by floating-point operations, and equivalency can be established. The methods introduced in this paper extend that work to support more complex mathematical algorithms consisting of a network of functional blocks.

Also utilizing the SPIRAL system, the pipeline follows a similar framework of exploiting symbolic execution and the database to recognize even higher level operations. However, complex scientific computing code comprised of numerous mathematical operations beyond ubiquitous operations like the FFT are infeasible to merely lift, as SPIRAL would need to be populated with an enormous database of all possible operations and their variations. Moreover, those operations may be implemented in mathematically distinct manners.

We propose a pipeline that can recognize and verify these more complex operations by constructing reasoning chains as follows. In Section III, the chain starts with basic C code and uses previous work to extract and verify basic

Towards Semantics Lifting for Scientific Computing: A Case Study on FFT

Naifeng Zhang, Carnegie Mellon University, Pittsburgh, USA, naifengz@cmu.edu
Sanil Rao, Carnegie Mellon University, Pittsburgh, USA, sanilr@andrew.cmu.edu
Mike Franusich, SpiralGen, Inc., Pittsburgh, USA, mike.franusich@spiralgen.com
Franz Franchetti, Carnegie Mellon University, Pittsburgh, USA, franzf@andrew.cmu.edu

Abstract

The rise of automated code generation tools, such as large language models (LLMs), has introduced new challenges in ensuring the correctness and efficiency of scientific software, particularly in complex kernels, where numerical stability, domain-specific optimizations, and precise floating-point arithmetic are critical. We propose a stepwise semantics lifting approach using an extended SPIRAL framework with symbolic execution and theorem proving to statically derive high-level code semantics from LLM-generated kernels. This method establishes a structured path for verifying the source code's correctness via a step-by-step lifting procedure to high-level specification. We conducted preliminary tests on the feasibility of this approach by successfully lifting GPT-generated fast Fourier transform code to high-level specifications.

Keywords: Semantics lifting, static analysis, scientific computing, fast Fourier transform, SPIRAL

1 Introduction

The growing adoption of neural-based code generation tools, such as large language models (LLMs), presents significant challenges in ensuring the correctness and efficiency of scientific software [10]. Although LLM-generated code may be syntactically valid, it often falls short of meeting the rigorous correctness and performance standards required for complex scientific kernels. Scientific computing demands numerical stability, domain-specific optimizations, and accurate floating-point arithmetic, which are challenging to achieve in code generated without domain expertise. By deriving the semantics of generated kernels statically (at compile time) for cases with unknown (runtime) size parameters, we can identify potential bugs, inefficiencies, and performance bottlenecks before deploying the code. This statically derived information can also be fed back into neural-based code generation tools to iteratively improve the generated code. However, scientific computing poses unique challenges for static analysis tools. Accurate handling of floating-point arithmetic requires managing rounding errors and numerical precision, while pointers, recursion, and transcendental functions like sine and cosine further complicate static analysis. To address these issues, this work proposes *stepwise semantics lifting* as an early-stage experimental solution within constrained boundaries. We develop a novel extension to the SPIRAL system [6, 16], which is equipped with symbolic execution and theorem-proving capabilities. Through an LLVM-to-SPIRAL parser, we import LLM-generated scientific kernels into SPIRAL and derive their semantics using SPIRAL's formal framework and engine.

Contributions. To summarize, this paper makes the following contributions:

1. An experimental approach, stepwise semantics lifting, for statically extracting high-level semantics from scientific kernels.
2. An end-to-end demonstration of the proposed approach by lifting GPT-generated fast Fourier transform code to its high-level specification.

2 Background

In this section, we provide background on SPIRAL, a formal code generation system, and the target scientific kernel: the fast Fourier transform (FFT).

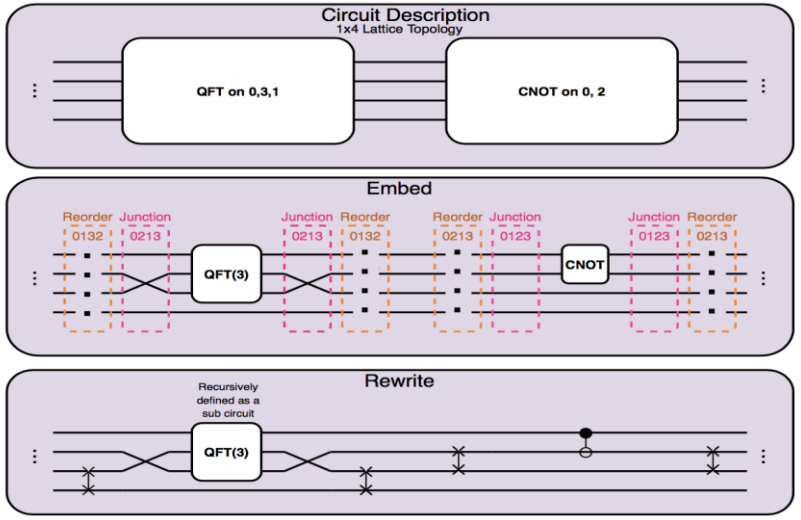
The SPIRAL system. The SPIRAL system [16] originated as an automatic performance-tuning system for signal processing algorithms, particularly focusing on FFT algorithms. This focus stemmed from the availability of a formal framework (the Kronecker product formalism [13, 19]), which enables capturing and manipulating FFT algorithms in high-level mathematical representations. Over time, this representation was generalized to encompass a broader range of algorithms [5], including both sparse and dense mathematical computations. SPIRAL has thus evolved into a comprehensive code generation system, capable of taking high-level specifications and producing optimized implementations for target platforms.

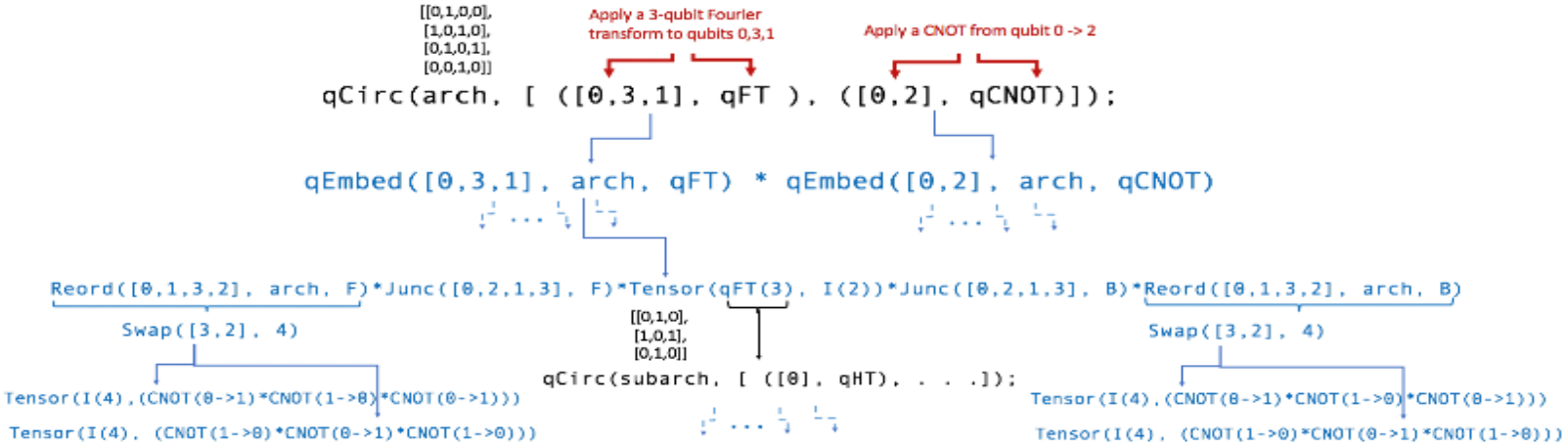
SPIRAL dialects. The SPIRAL system consists of three main components used in a stepwise code generation process: i) Signal Processing Language (SPL) [21], ii) Σ-SPL [9], and iii) internal code (icode) [6]. SPL, the top-level domain-specific language (DSL), describes the mathematical semantics of kernels and the functional data flow of the target algorithm. The lower-level DSL, Σ-SPL, captures loop abstractions, while icode serves as an abstract code representation adaptable to different code syntaxes. As shown in Figure 1, each layer of abstraction is connected by a rewrite system that applies recursive descent followed by confluent term rewriting. For further details, we refer readers to the respective citations.

Q. Oschatz, N. Zhang, M. Franusich, F. Franchetti: **Towards Automated Reasoning Chains for Verification of LLM-Generated Scientific Code**
IEEE High Performance Extreme Computing Conference (HPEC), 2025.

N. Zhang, S. Rao, M. Franusich, F. Franchetti: **Towards Semantics Lifting for Scientific Computing: A Case Study on FFT**
Theory and Practice of Static Analysis Workshop (TPSA), in conjunction with the ACM SIGPLAN Symposium on Principles of Programming Languages (POPL), 2025.

Towards SPIRAL+LLM for Quantum Computing





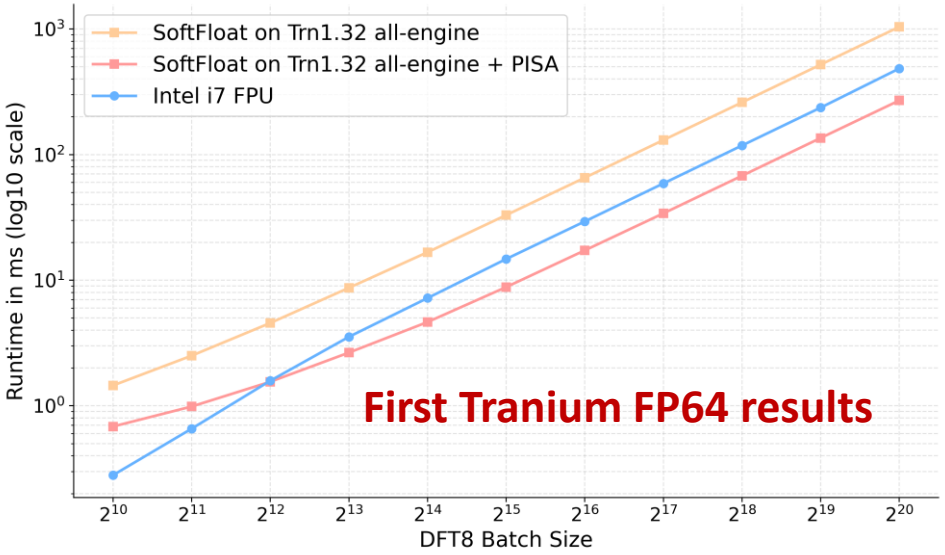
SPIRAL C++/Julia/Python SPL API + Code Llama + SPIRAL/GAP Reasoning

S. Mionis, F. Franchetti, J. Larkin: **Optimized Quantum Circuit Generation with SPIRAL**
IEEE High Performance Extreme Computing Conference (HPEC), 2021, Outstanding Student Paper Award.

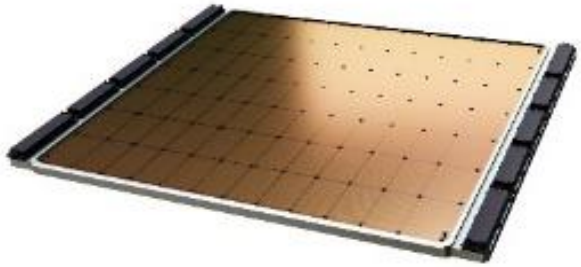
D. Sun, N. Zhang, F. Franchetti: **Optimization and Performance Analysis of Shor's Algorithm in Qiskit**
IEEE High Performance Extreme Computing Conference (HPEC), 2023.

FP64 Emulation on Non-FP64 ML Accleerators

```
#define _fp64int64_add(_ce, _cm, _ae, _am, _be, _bm){ \
    __int16_t _diff0 = _subw((_ae), (_be));  \
    __int16_t _diff1 = _subw((_be), (_ae));  \
    _Bool _flag = _cmplsw((_ae), (_be));  \
    __int16_t _shamt = _selw(_flag, _diff1, _diff0); \
    __int64_t _sm = _selq(_flag, (_am), (_bm)); \
    __int64_t _km = _selq(_flag, (_bm), (_am));  \
    _sm = _sarq(_sm, _shamt);  \
    _sm = _selq(_cmplw(_shamt, 64), _sm, 0); \
    _Bool _cmc; __uint64_t _sum;  \
    _addqc(_sum, _cmc, _km, _sm);  \
    _Bool _ovf = _sltq(_andq(_xorq(_sm, _sum), \
         _xorq(_km, _sum)), 0);  \
    (_cm) = _selq(_ovf, _shrqdq((uint64_t)_cmc, _sum, 1), _sum);  \
    (_ce) = _caddw(_selw(_flag, (_be), (_ae)), 1, _ovf);  \
}
```



Nvidia 5090
"compatibility-only" FP64



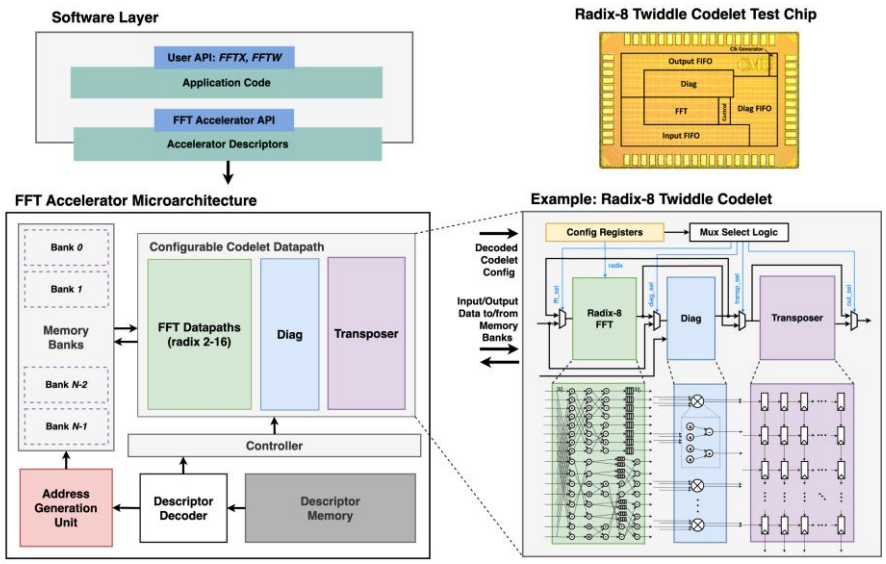
Cerebras WSE3
no FP64

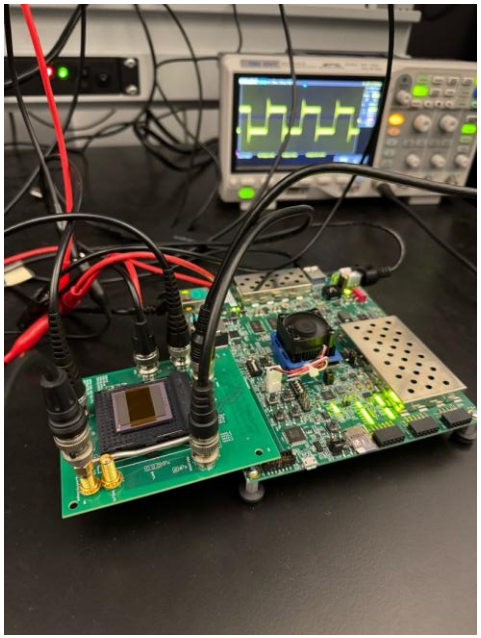
Amazon Tranium 2
no FP64

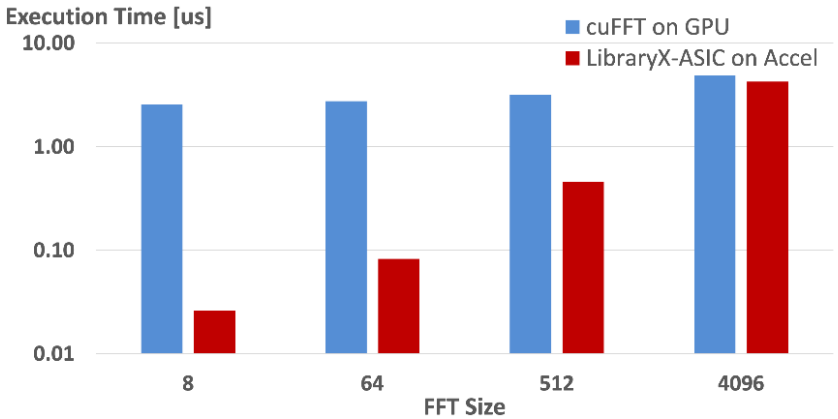
Outline

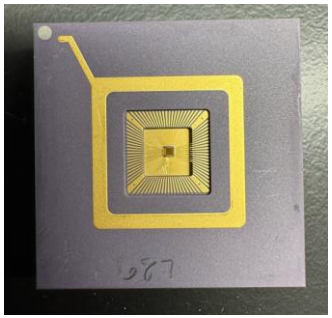
- Introduction
- Specifying computation
- Achieving Performance Portability
- SPIRAL and Generative and Agentic AI
- **Other Current Work**

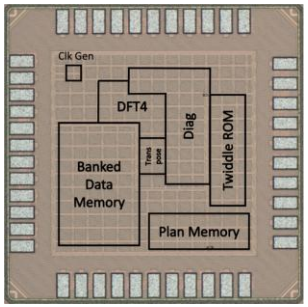
FFTW with Hardware Codelets









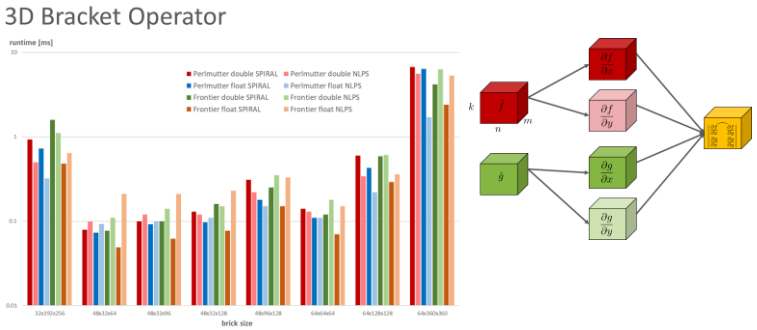


API: FFTW. Infrastructure: SPIRAL/FFTX. Performance: on chiplet/ASIC

S. Rao, L. Tang, F. Franchetti: **LibraryX-ASIC: A First Look**, International Parallel and Distributed Processing Symposium (IPDPS), Workshop on High-Level Parallel Programming Models and Supportive Environments (HIPS), 2025.

Real Science Applications + Other Motifs

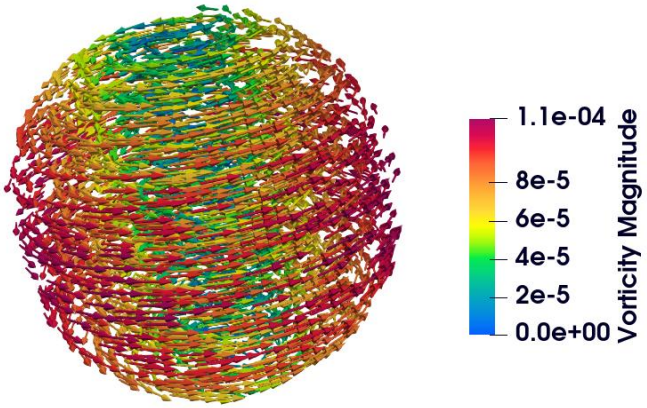
GX/Fusion



(U. of Md, CMU, LBNL)

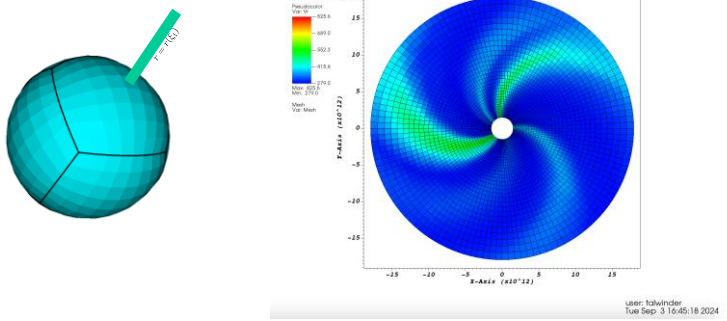
P3M Method (MLC):

Cabana / Kokkos + FFTX on a GPU



3D Hill's Vortex (ORNL, Stanford, LBNL)

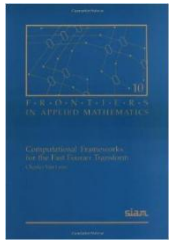
Structured Grids: Proto

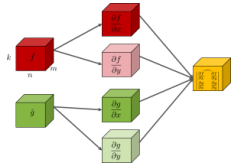


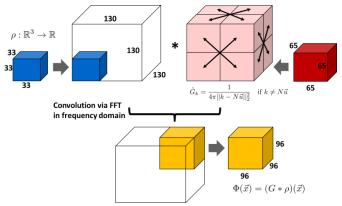
Solar wind (UAH, GSU, LBNL)

SPIRAL: AI for High Performance Code

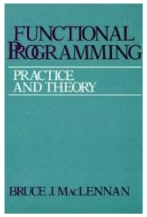
Algorithms

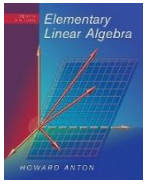






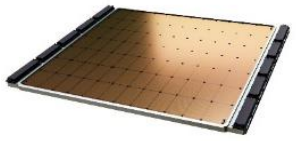
Correctness

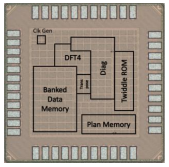





```
int dwmonitor(float  *X, double  *D) {
    __m128d u1, u2, u3, u4, u5, u6, u7, u8,...
    unsigned _xm = _mm_getcsr();
    _mm_setcsr(_xm & 0xffff0000 | 0x0000dfc0);
    u5 = _mm_set1_pd(0.0);
    u2 = _mm_cvtps_pd(_mm_addsub_ps(
    _mm_set1_ps(FLT_MIN), _mm_set1_ps(X[0])));
    u1 = _mm_set_pd(1.0, (-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);
    ...
```

Hardware

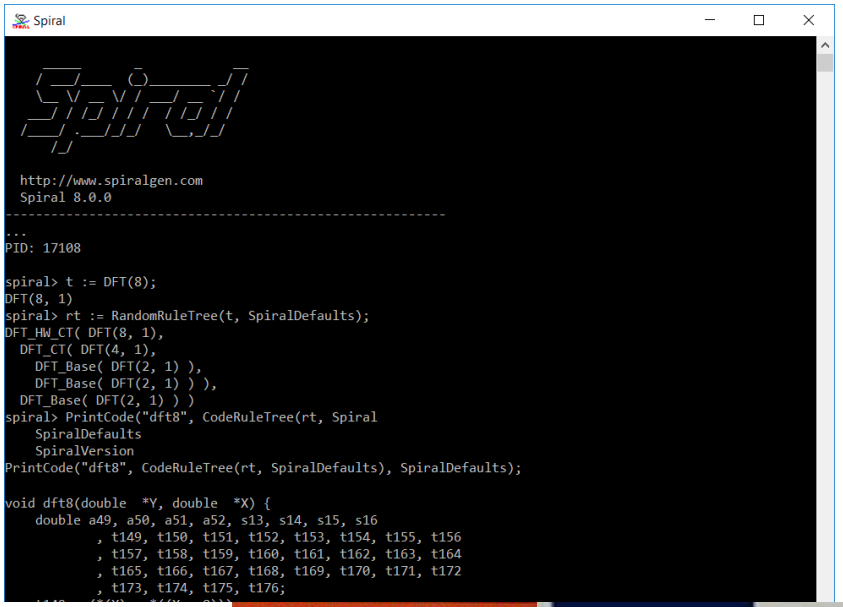


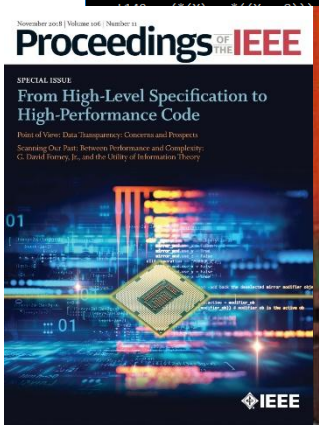


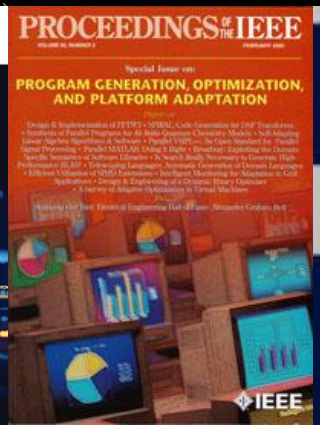
SPIRAL 8.5.3 and FFTX 1.0

ECP EXASCALE COMPUTING PROJECT

- **Open Source SPIRAL available**
  - non-viral license (BSD)
  - Commercial support via SpiralGen, Inc.
  - www.spiral.net, www.spiralgen.com
- **Developed over almost 25 years**
- **Ongoing open source development**
  DOE Ai4Science, SciDAC, Base
- **FFTX 1.0 release**
  www.spiral.net/software/fftx.html
- **SPIRAL Software Foundation**
  Next step in the evolution, ETA 2026







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*

F. Franchetti, D. G. Spampinato, A. Kulkarni, D. T. Popovici, T. M. Low, M. Franusich, A. Canning, P. McCorquodale, B. Van Straalen, P. Colella: FFTX and SpectralPack: A First Look, IEEE International Conference on High Performance Computing, Data, and Analytics, 2018