

SPIRAL: Enabling Faster FFTs on AI Accelerators

Naifeng Zhang

Department of Electrical and Computer Engineering
Carnegie Mellon University

<https://naifeng.github.io/>

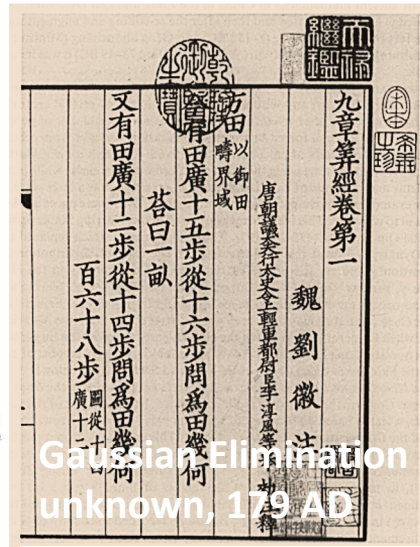
Joint work with the SPIRAL team at CMU, UIUC, Drexel, SpiralGen, Inc.,
and collaborators at LBL, ORNL, and LANL
Slides borrowed from Franz Franchetti

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

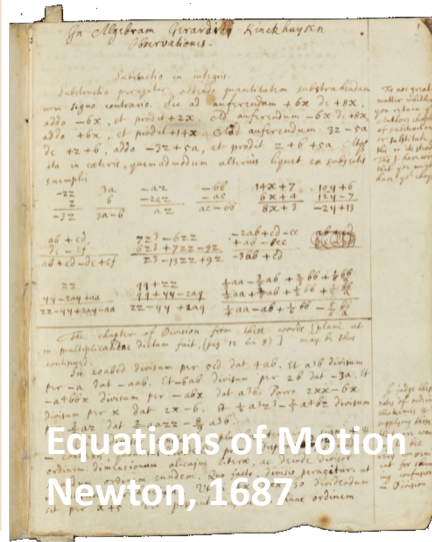
Algorithms and Mathematics: 2,500+ Years

Geometry
Euclid, 300 BC

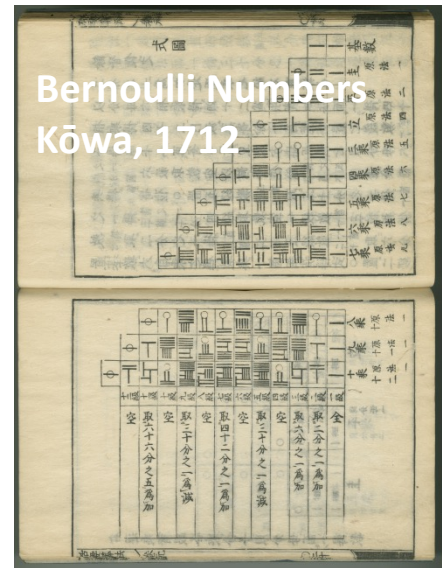
Square roots, value of Pi
Baudhāyan, 800 BC



Gaussian Elimination
unknown, 179 AD

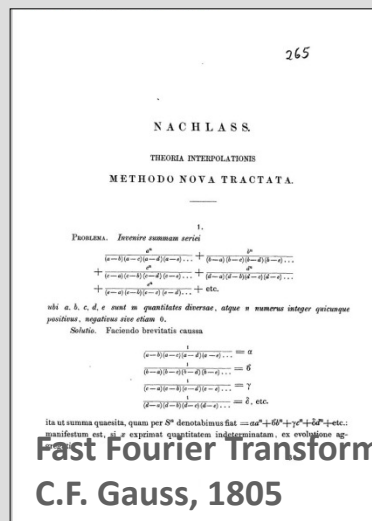


Equations of Motion
Newton, 1687

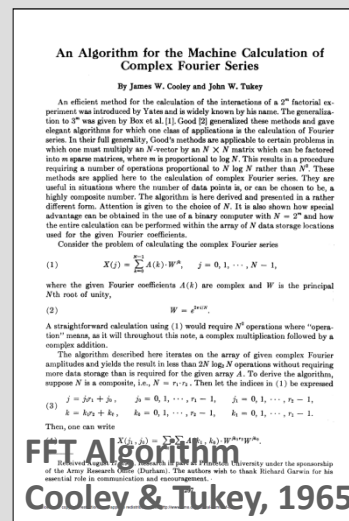


Bernoulli Numbers
Kōwa, 1712

Fast Fourier Transform



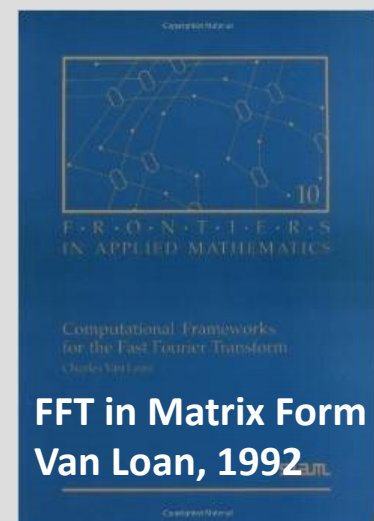
Fast Fourier Transform
C.F. Gauss, 1805



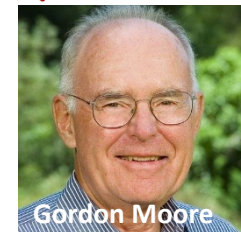
FFT Algorithm
Cooley & Tukey, 1965



Algebra
al-Khwarizmi, 830



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

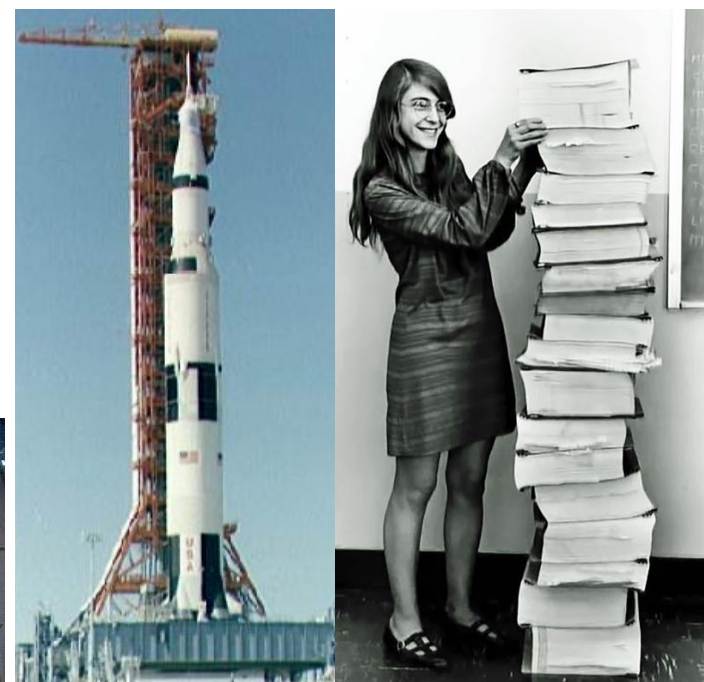
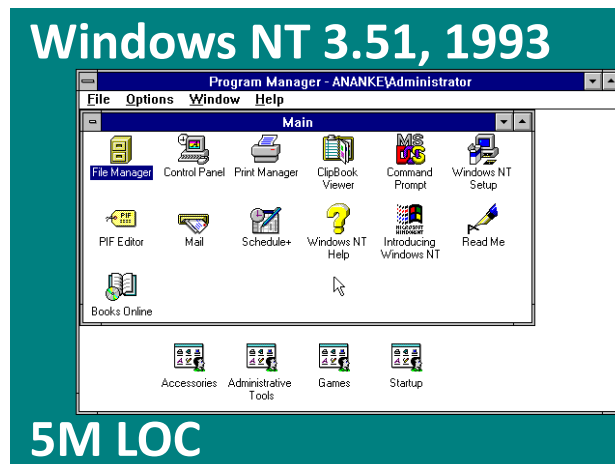
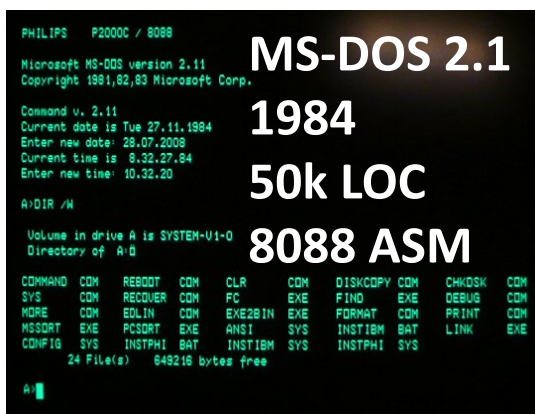
Symbolic and Numerical PSEs

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

Numerical Mathematics libraries

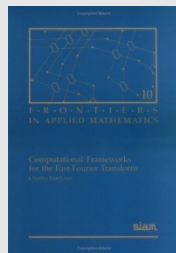
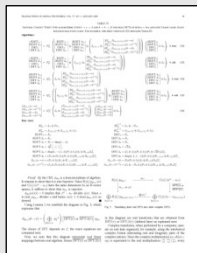
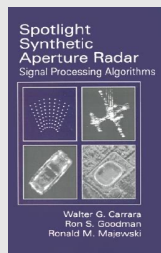
- 1970: IMSL
- 1971: NAG
- 2001: GNU Scientific Library
- 2003: Intel Math Kernel Library

Explosive Growth of Source Code



SPIRAL: AI for High Performance Code

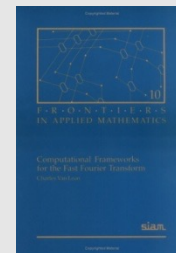
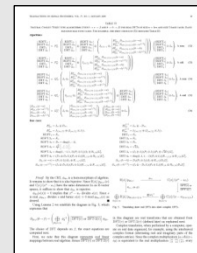
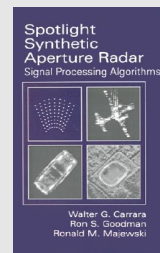
Traditionally



High performance library
optimized for given platform

*Comparable
performance*

SPIRAL Approach



High performance library
optimized for given platform

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



DARPA/Defense Sciences Office
Applied & Computational Mathematics Program
Origins



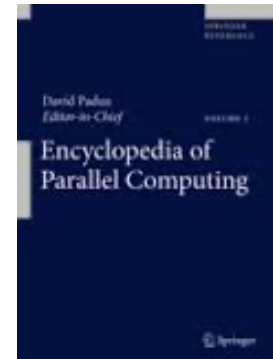
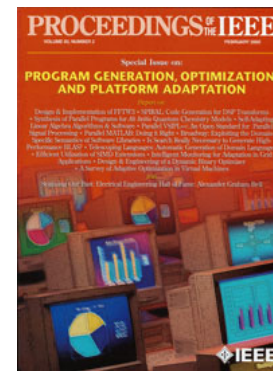
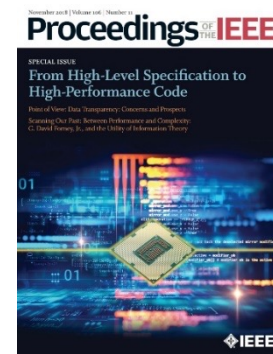
Louis Auslander (1928-1997)

Prepared for DARPA/DSO by Anna Tsao¹

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 Beylfin, Dennis Braunschneider, Russell Callach, Douglas Cochran, Ronald Colman, James Crowley, Benjamin Dembart, Jon Ebert, Abbas Emami-Naeini, Fariba Fahroo, George Fain, Franz Franchetti, Kristen Fuller, Leslie Greengard, Mark Gyure, Robert Harrison, Peter Heller, Jeremy Johnson, Robert Kosut, José Moura, Arje Nachman, Stanley Osher, Mark Odeh, Vladimir Rokhlin, Leonid Rudin, Randall Sandis, Carey Schwartz, Mark Stuber, Scott Stewart, Bruce Suter, Hayth 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
- Case Study: FFTs on AI Accelerators

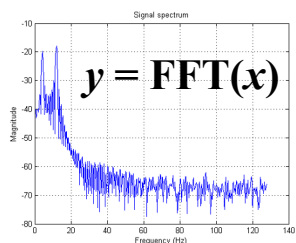
SPIRAL: AI for Performance Engineering

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

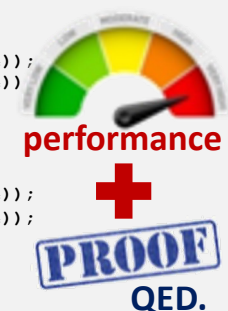


on



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

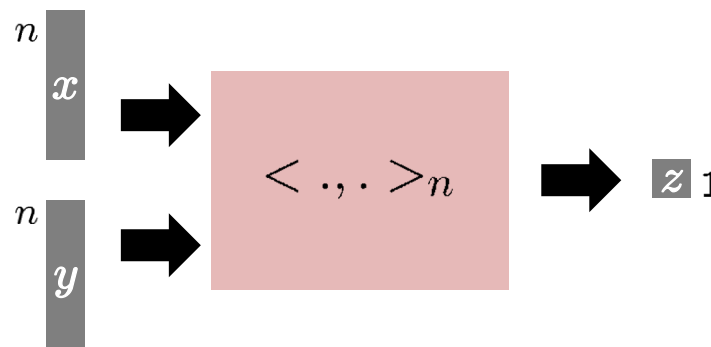
- **Operator: Multiple vectors $\rightarrow$ Multiple vectors**
- **Stateless**
- **Higher-dimensional data is linearized**
- **Operators are potentially nonlinear**

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

Example: Scalar product

$$\langle \cdot, \cdot \rangle_n: \mathbb{R}^n \times \mathbb{R}^n \rightarrow \mathbb{R}$$

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



Breaking Down Operators into Expressions

■ Application specific: Safety Distance as Rewrite Rule

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

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

Problem specification: hand-developed or automatically produced

■ One-time effort: mathematical library

$$d_{\infty}^n(\cdot, \cdot) \rightarrow \|\cdot\|_{\infty}^n \circ (-)_n$$

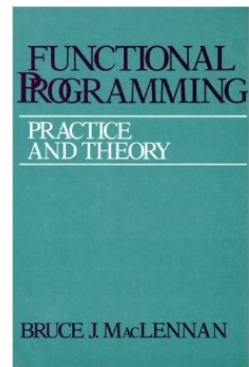
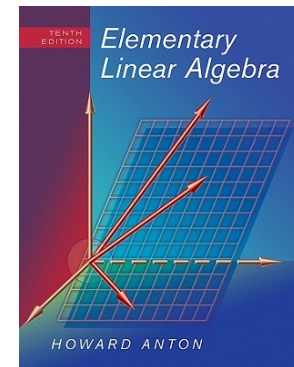
$$(\diamond)_n \rightarrow \text{Pointwise}_{n \times n, (a,b) \mapsto a \diamond b}, \quad \diamond \in \{+, -, \cdot, \wedge, \vee, \dots\}$$

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

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

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

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



Library of well-known identities expressed in OL

Loop and Code Level Rule System

Mathematical Loop Abstraction

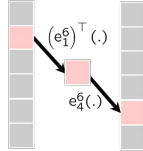
- Selection and embedding operator: *gather and scatter*

$$(e_i^n)^\top (\cdot) : \mathbb{R}^n \rightarrow \mathbb{R}^1$$

$$(x_i)_{i=0, \dots, n-1} \mapsto x_i$$

$$e_i^n (\cdot) : \mathbb{R}^1 \rightarrow \mathbb{R}^n$$

$$(x) \mapsto (0, \dots, 0, \underset{i^{\text{th}}}{x}, 0, \dots, 0)$$

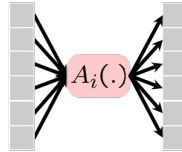


- Iterative operations: *loop*

$$\bigcup_{i=0}^{n-1} : (D \rightarrow R)^n \rightarrow (D \rightarrow R)$$

$$A_i \mapsto (x \mapsto A_0(x) \sqcup \dots \sqcup A_{n-1}(x))$$

with $\sqcup \in \{\sum, \vee, \wedge, \prod, \min, \max, \dots\}$



- Atomic operators: *nonlinear scalar functions*

$$\text{Atomic}_f : \mathbb{R}^1 \rightarrow \mathbb{R}^1$$

$$(x) \mapsto (f(x))$$

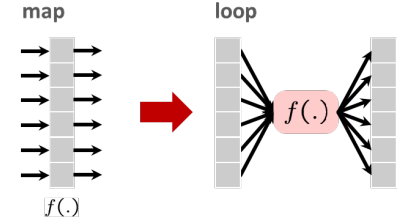


Translation and Optimization

- Translating Basic OL into Σ -OL

$$\text{Pointwise}_{n, f_i} \rightarrow \sum_{i=0}^{n-1} (e_i^n \circ \text{Atomic}_{f_i} \circ (e_i^n)^\top)$$

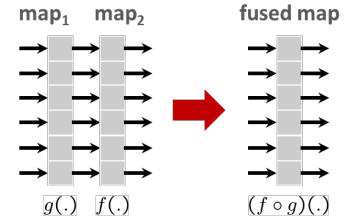
$$\text{Reduction}_{n, (a,b) \mapsto a+b} \rightarrow \sum_{i=0}^{n-1} (e_i^n)^\top$$



- Optimizing Basic OL/ Σ -OL

$$\text{Pointwise}_{n, f_i} \circ \text{Pointwise}_{n, g_i} \rightarrow \text{Pointwise}_{n, f_i \circ g_i}$$

$$\text{Pointwise}_{n, f_i} \circ e_n^j \rightarrow e_n^j \circ \text{Pointwise}_{1, f_j}$$



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, "demonitor", [ 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(qeq(w1, V(0)), w1, neg(w1))),
            assign(s1, cond(qeq(s1, s6), s1, s6))
          )
        ),
        assign(w2, qeq(s1, s5)),
        creturn(w2)
      )
    )
  )
)
```

Rule Based Compiler

Compilation rules: recursive descent

$$\text{Code}(y = (A \circ B)(x)) \rightarrow \{\text{decl}(t), \text{Code}(t = B(x)), \text{Code}(y = A(t))\}$$

$$\text{Code}\left(y = \left(\sum_{i=0}^{n-1} A_i\right)(x)\right) \rightarrow \{y := \vec{0}, \text{for}(i = 0..n-1) \text{Code}(y += A_i(x))\}$$

$$\text{Code}(y = (e_i^n)^\top(x)) \rightarrow y[0] := x[i]$$

$$\text{Code}(y = e_i^n(x)) \rightarrow \{y := \vec{0}, y[i] := x[0]\}$$

$$\text{Code}(y = \text{Atomic}_f(x)) \rightarrow y[0] := f(x[i])$$

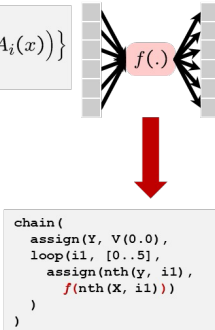
Cleanup rules: term rewriting

$$\text{chain}(a, \text{chain}(b)) \rightarrow \text{chain}([a, b])$$

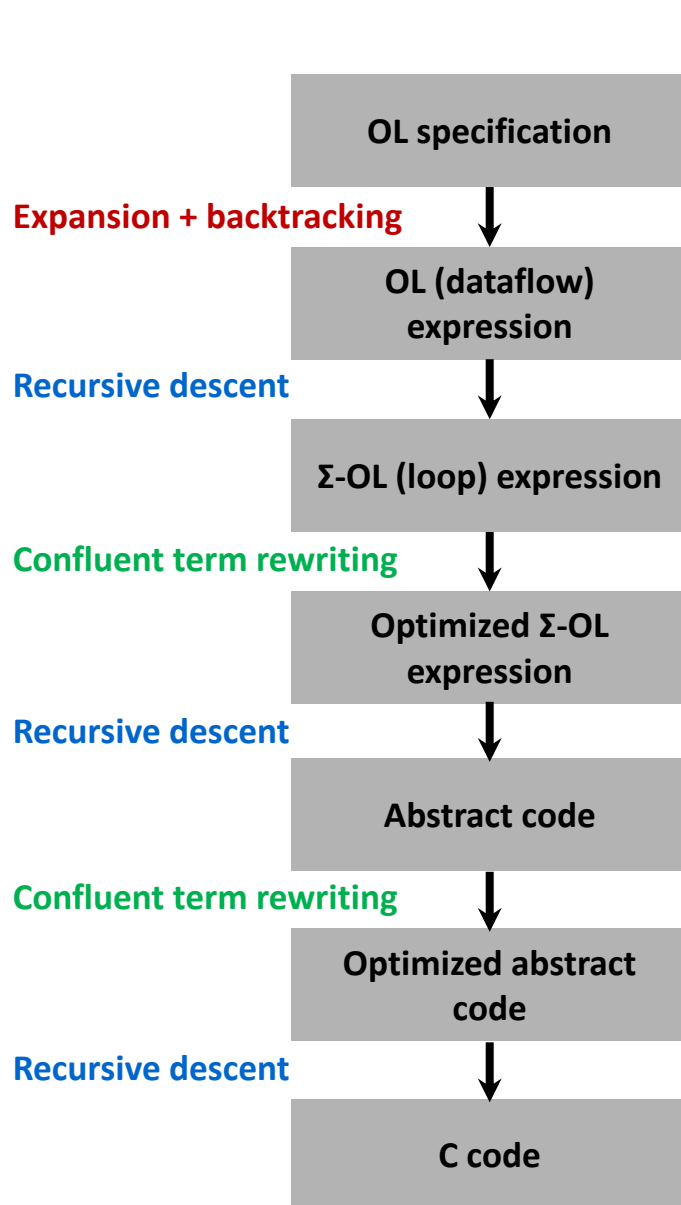
$$\text{decl}(D, \text{decl}(E, c)) \rightarrow \text{decl}([D, E], c)$$

$$\text{loop}(i, \text{decl}(D, c)) \rightarrow \text{decl}(D, \text{loop}(i, c))$$

$$\text{chain}(a, \text{decl}(D, b)) \rightarrow \text{decl}(D, \text{chain}([a, b]))$$



Putting it Together: One Big Rule System



Mathematical specification

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

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

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_ps(1.0)));
    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_load_pd(x1));
        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)), x2);
        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)), u3), x4);
    }
}

```

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
 $\Gamma(\cdot)$, distributions, Lebesgue integral
- **Semantics preserving**
rule chain = formal proof
- **Automation**
Mathematica, Maple

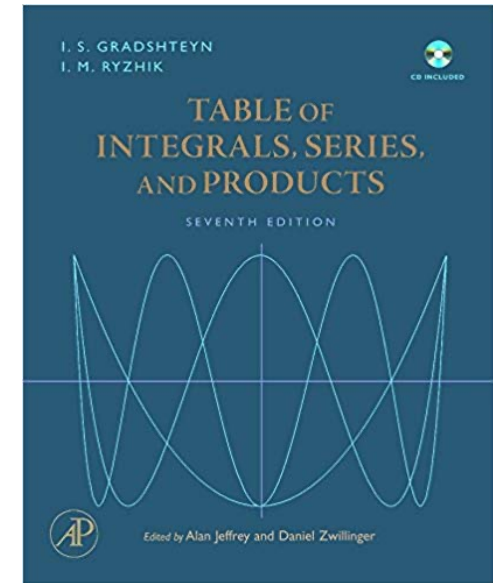
Table of Integrals

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

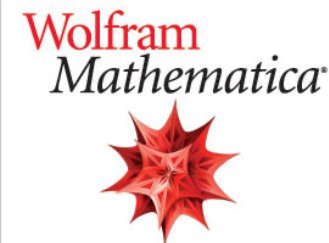


$$\text{In}[31]:= \int_0^{2\pi} \frac{1}{a^2 \cos[t]^2 + b^2 \sin[t]^2} dt$$

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

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

$$\text{Out}[33]:= 0$$



Outline

- Introduction
- Specifying Computation
- **Achieving Performance Portability**
- Case Study: FFTs on AI Accelerators

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.

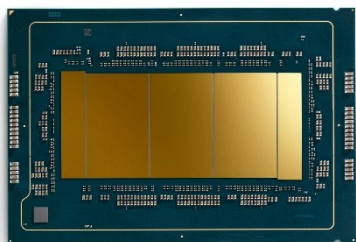
F. Franchetti, T. M. Low, D. T. Popovici, R. M. Veras, D. G. Spampinato, J. R. Johnson, M. Püschel, J. C. Hoe, J. M. F. Moura:

SPIRAL: Extreme Performance Portability, **Proceedings of the IEEE**, Vol. 106, No. 11, 2018.

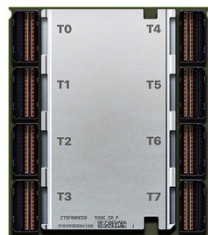
Special Issue on *From High Level Specification to High Performance Code*

Today's Computing Landscape

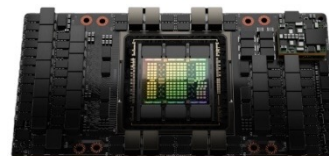
1 Gflop/s = one billion floating-point operations (additions or multiplications) per second



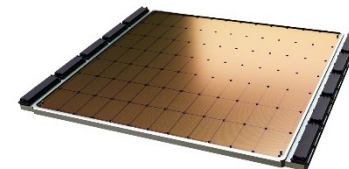
Intel Xeon 6980P
16 Tflop/s, 500 W
 128 cores, 2 – 3.9 GHz
 2-way—16-way AVX-512



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



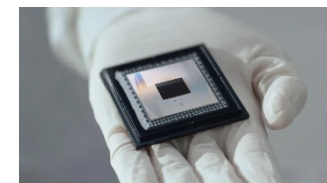
Snapdragon 8+ Gen1
15 Gflop/s, 2 W
 8 cores, 3.2 GHz
 A730 GPU, Hexagon DSP



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



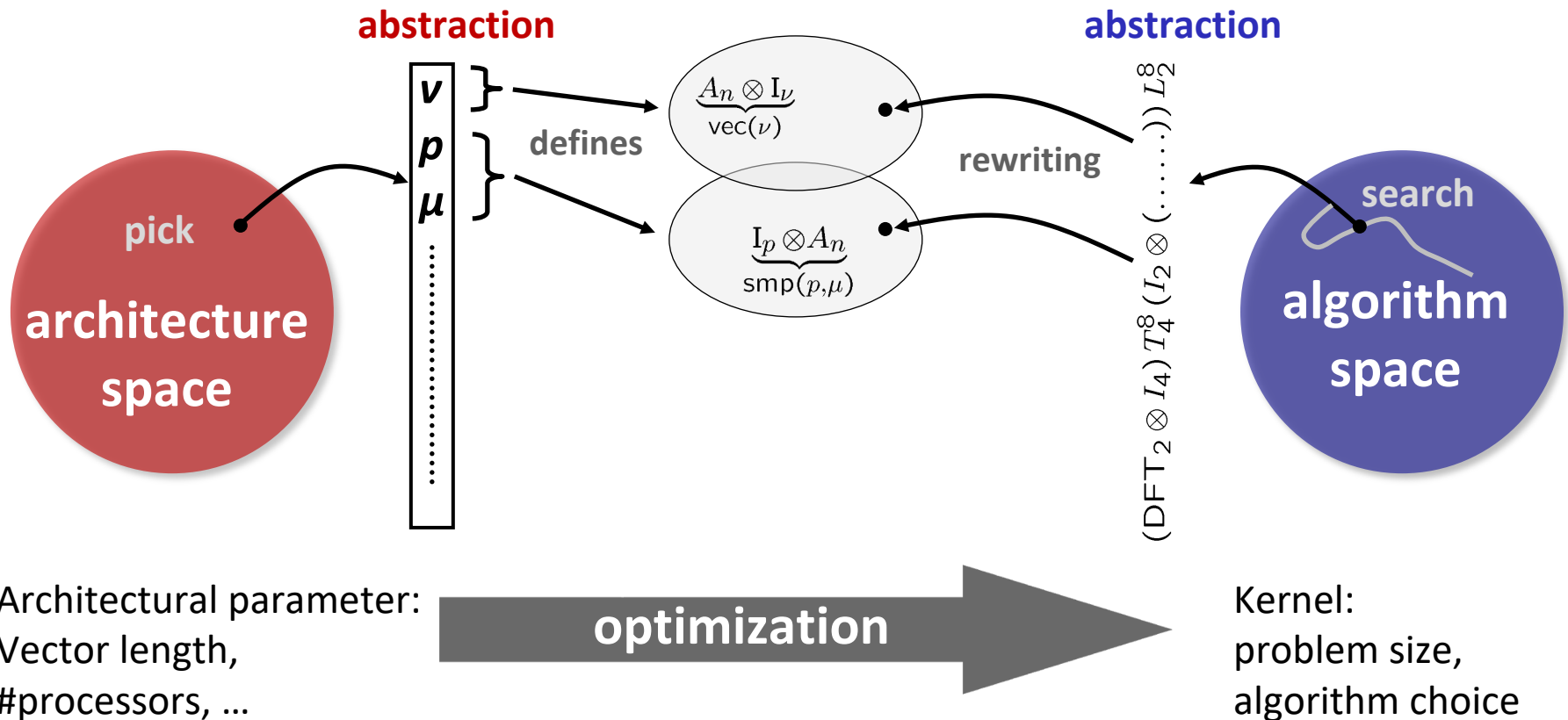
El Capitan
2.7 Eflop/s, 30MW
 43k 24-core CPUs + 43k GPUs
 #1 in Top500



Google Willow
105 qubits

Platform-Aware Formal Program Synthesis

Model: common abstraction
= spaces of matching formulas



Some Application Domains in OL

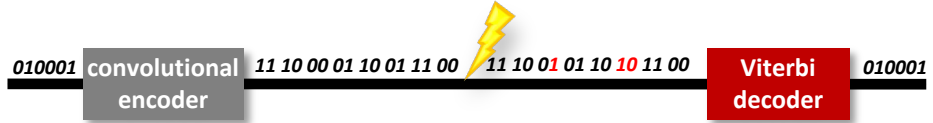
Linear Transforms

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

PDEs/HPC Simulations

$$\begin{aligned}
 \Phi : \mathbb{R}^3 &\rightarrow \mathbb{R} \\
 \Phi(\vec{x}) &= \frac{Q}{4\pi\|\vec{x}\|} + o\left(\frac{1}{\|\vec{x}\|}\right) \text{ as } \|\vec{x}\| \rightarrow \infty \\
 Q &= \int_D \rho d\vec{x}
 \end{aligned}$$

Software Defined Radio

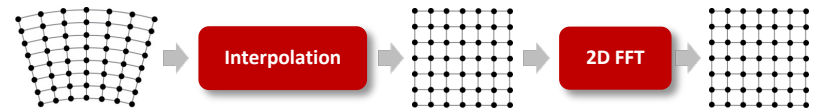


$$F_{K,F} \rightarrow \prod_{i=1}^F \left((I_{2^{K-2}} \otimes_j B_{F-i,j}) L_{2^{K-2}}^{2^{K-1}} \right)$$

$$\underline{F}_{K,F} \nu \rightarrow \prod_{i=1}^F \left((I_{2^{K-2}/\nu} \otimes_{j_1} \tilde{L}_{\nu}^{2\nu} \tilde{B}_{F-i,j_1}^{\nu}) (L_{2^{K-2}/\nu}^{2^{K-1}/\nu} \otimes I_{\nu}) \right)$$

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

Synthetic Aperture Radar (SAR)



$$\text{SAR}_{k \times m \rightarrow n \times n} \rightarrow \text{DFT}_{n \times n} \circ \text{Interp}_{k \times m \rightarrow n \times n}$$

$$\text{DFT}_{n \times n} \rightarrow (\text{DFT}_n \otimes \text{I}_n) \circ (\text{I}_n \otimes \text{DFT}_n)$$

$$\text{Interp}_{k \times m \rightarrow n \times n} \rightarrow (\text{Interp}_{k \rightarrow n} \otimes_i \text{I}_n) \circ (\text{I}_k \otimes_i \text{Interp}_{m \rightarrow n})$$

$$\text{Interp}_{r \rightarrow s} \rightarrow \left(\bigoplus_{i=0}^{n-2} \text{InterpSeg}_k \right) \oplus \text{InterpSegPruned}_{k,\ell}$$

$$\text{InterpSeg}_k \rightarrow G_f^{u \cdot n \rightarrow k} \circ \text{iPrunedDFT}_{n \rightarrow u \cdot n} \circ \left(\frac{1}{n} \right) \circ \text{DFT}_n$$

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)

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

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

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

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

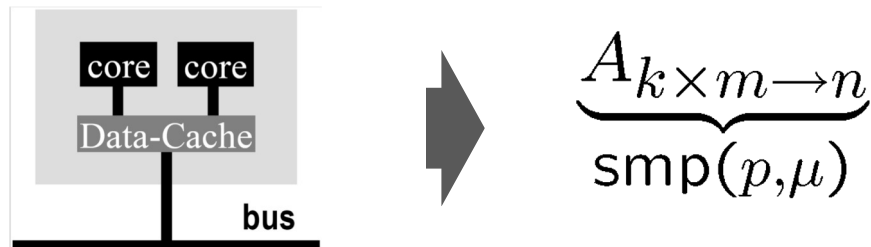
$$\prod_{i=0}^{n-1} A_i, \quad A_n \hat{\otimes} I_w, \quad P_n \otimes Q_w$$

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

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

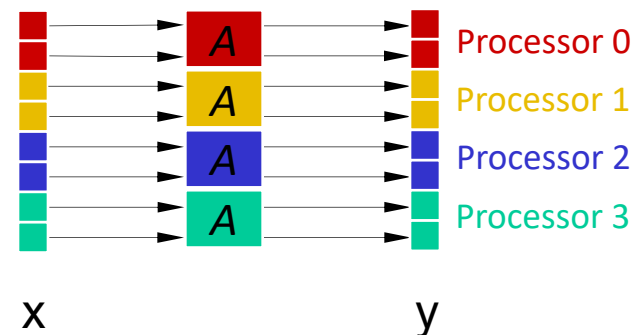
Modeling Hardware: Base Cases

- Hardware abstraction: shared cache with cache lines



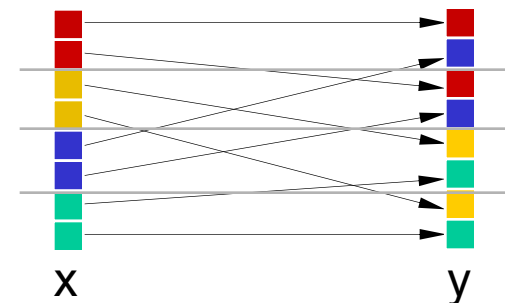
- Tensor product: embarrassingly parallel operator

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



- Permutation: problematic; may produce false sharing

$$y = L_4^8(x)$$



Example Program Transformation Rule Set

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

$$\underbrace{A_m \otimes I_n}_{\text{smp}(p,\mu)} \rightarrow \underbrace{\left(\underbrace{L_m^{mp} \otimes I_{n/p}}_{\text{smp}(p,\mu)} \right) \left(I_p \otimes (A_m \otimes I_{n/p}) \right) \left(\underbrace{L_p^{mp} \otimes I_{n/p}}_{\text{smp}(p,\mu)} \right)}_{\text{smp}(p,\mu)}$$

$$\underbrace{L_m^{mn}}_{\text{smp}(p,\mu)} \rightarrow \begin{cases} \underbrace{\left(I_p \otimes L_{m/p}^{mn/p} \right)}_{\text{smp}(p,\mu)} \underbrace{\left(L_p^{pn} \otimes I_{m/p} \right)}_{\text{smp}(p,\mu)} \\ \underbrace{\left(L_m^{pm} \otimes I_{n/p} \right)}_{\text{smp}(p,\mu)} \underbrace{\left(I_p \otimes L_m^{mn/p} \right)}_{\text{smp}(p,\mu)} \end{cases}$$

Recursive rules

$$\underbrace{I_m \otimes A_n}_{\text{smp}(p,\mu)} \rightarrow I_p \otimes_{||} \left(I_{m/p} \otimes A_n \right)$$

$$\underbrace{(P \otimes I_n)}_{\text{smp}(p,\mu)} \rightarrow (P \otimes I_{n/\mu}) \bar{\otimes} I_\mu$$

Base case rules

Autotuning in Constraint Solution Space

AVX 2-way
_Complex double

$\overbrace{\text{DFT}_8}^{\text{AVX(2-way } \mathbb{C})}$

Base cases

$$A^{n \times n} \otimes \vec{I}_2$$

$$\underbrace{L_2^4}_{\text{vec}(2)}$$

$$\underbrace{T_n^{mn}}_{\text{vec}(2)}$$

Transformation rules

$$(I_m \otimes A^{n \times n}) L_m^{mn} \rightarrow (I_{m/\nu} \otimes L_{\nu}^{n\nu} (A^{n \times n} \otimes I_{\nu})) (L_{m/\nu}^{mn/\nu} \otimes I_{\nu})$$

$$L_{\nu}^{n\nu} \rightarrow (I_{\nu}^n \otimes I_{\nu}) (I_{n/\nu} \otimes L_{\nu}^{\nu^2})$$

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

Breakdown rules

$$\text{DFT}_{mn} \rightarrow (\text{DFT}_m \otimes I_n) T_n^{mn} (I_m \otimes \text{DFT}_n) L_m^{mn}$$

$$\text{DFT}_2 \rightarrow F_2$$

DFT_8

Expansion + backtracking

OL specification

OL (dataflow)
expression

Recursive descent

Σ -OL (loop)
expression

Confluent term rewriting

Optimized Σ -OL
expression

Recursive descent

Abstract code

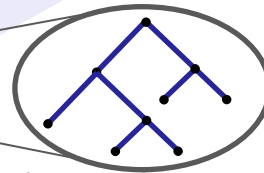
Confluent term rewriting

Optimized abstract
code

Recursive descent

C code

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



Translating an OL Expression Into Code

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

Output =

Ruletree, expanded into

OL Expression:

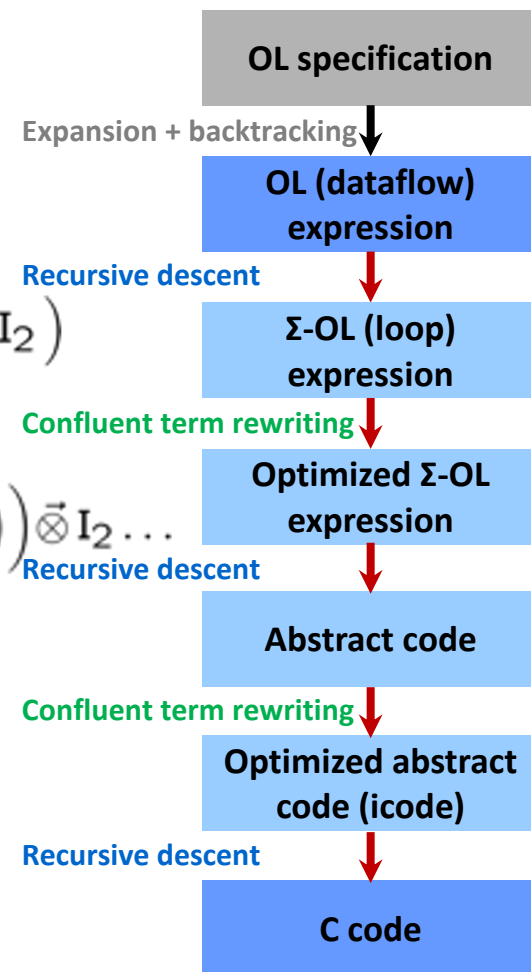
$$\left((F_2 \otimes I_2) T_2^4 (I_2 \otimes F_2) L_2^4 \vec{\otimes} I_2 \right) \underbrace{T_2^8}_{\text{vec}(2)} \left(I_2 \otimes \underbrace{L_2^4}_{\text{vec}(2)} (F_2 \vec{\otimes} I_2) \right) (L_2^4 \vec{\otimes} I_2)$$

Σ -OL:

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

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;
}
```



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

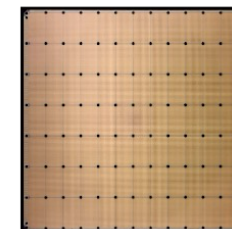
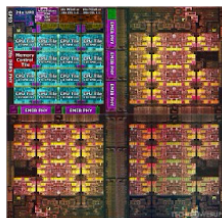
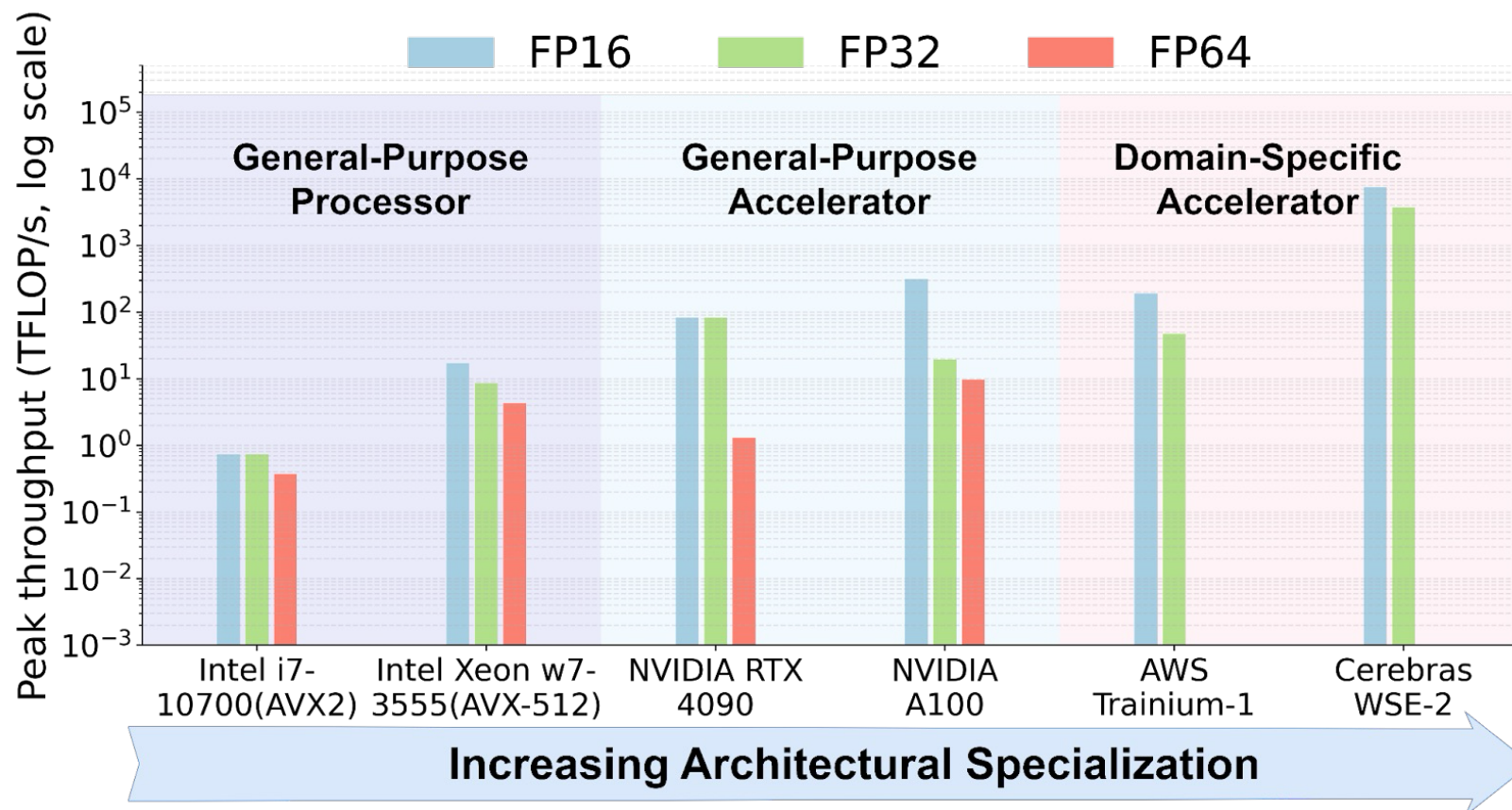
Outline

- Introduction
- Specifying Computation
- Achieving Performance Portability
- **Case Study: FFTs on AI Accelerators**

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).

Performance vs. Precision

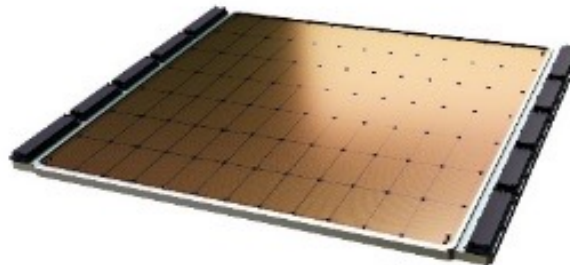


FP64 Emulation Using Integer Units

```
#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

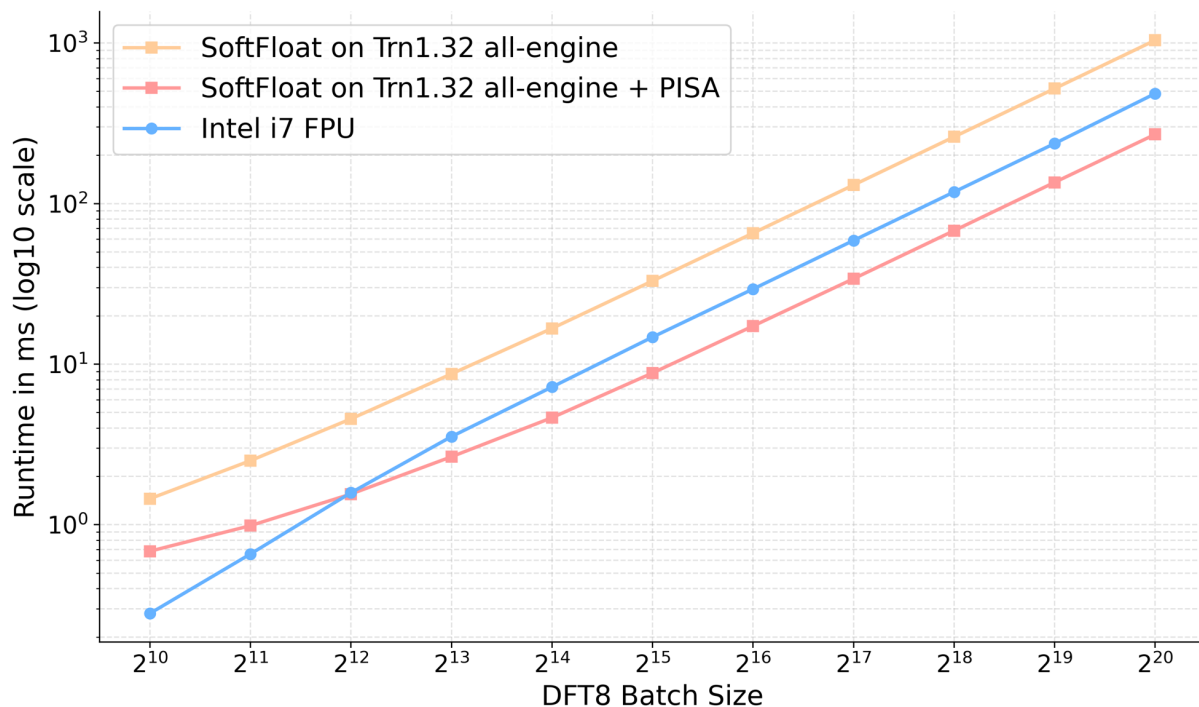


Cerebras WSE3
no FP64



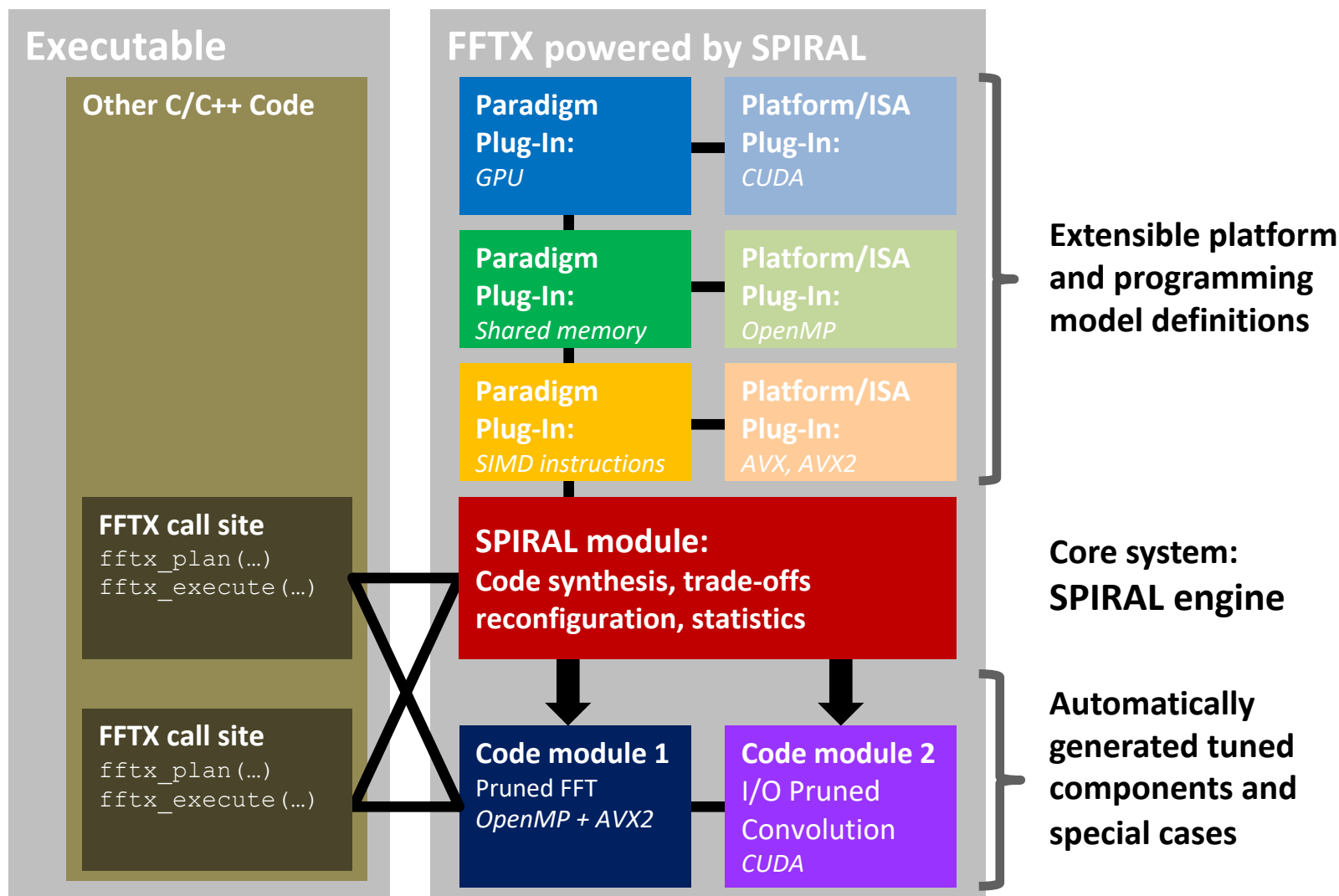
Amazon Trainium
no FP64

First Amazon Trainium FP64 Results



Amazon Trainium
no FP64

LibraryX: SPIRAL as Active Library Backend

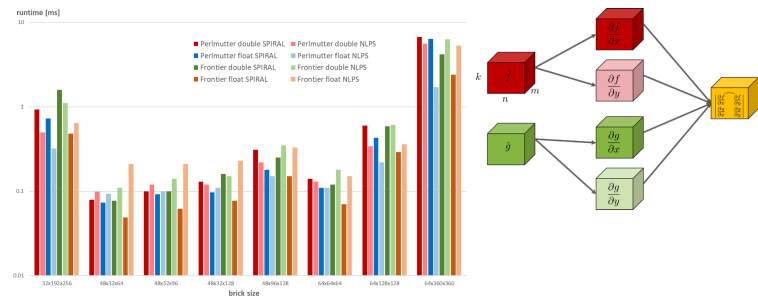


LibraryX and C++/Python/Julia SPL API enable SPIRAL as Agent for GenAI

Real Science Applications + Other Motifs

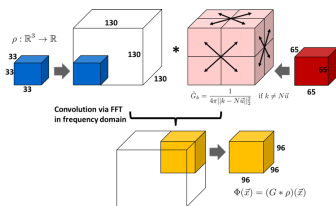
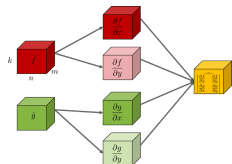
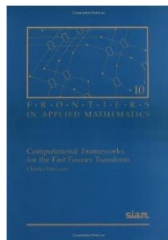
GX/Fusion

3D Bracket Operator

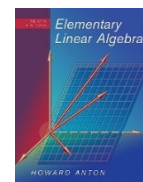
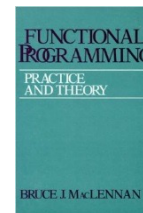
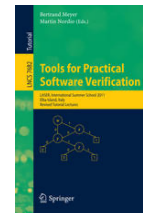


SPIRAL: AI for High Performance Code

Algorithms



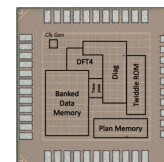
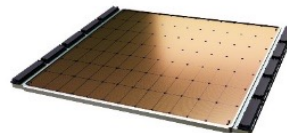
Correctness



```
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_loadup_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



- **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

```

Spiral

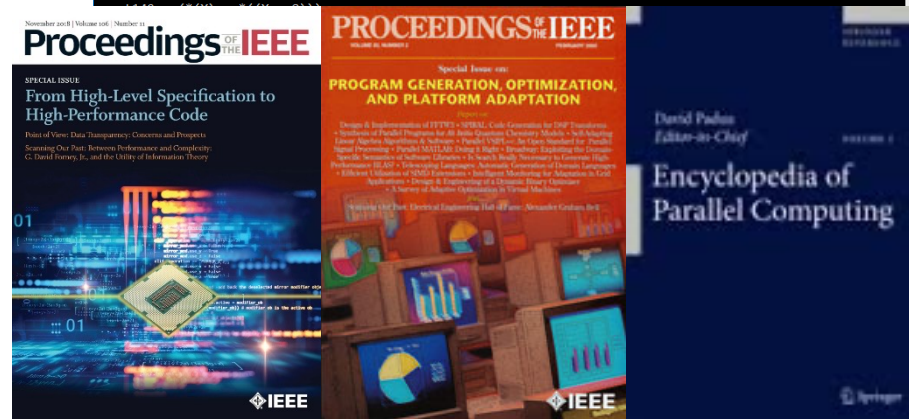
http://www.spiralgen.com
Spiral 8.0.0

...
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
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;

```



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