

SPIRAL: AI for High Performance Code

Franz Franchetti

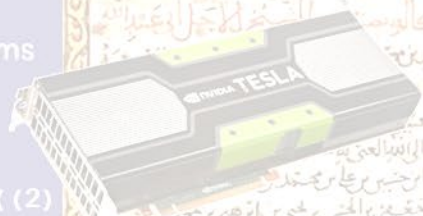
Department of Electrical and Computer Engineering
Carnegie Mellon University

www.ece.cmu.edu/~franzf

Joint work with the SPIRAL team at CMU and FFX team at CMU and LBL

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

Spotlight
Synthetic
Aperture Radar
Signal Processing Algorithms



```
pd(s5672, s5673, (0) | (2)
pd(s5672, s5673, (1) | (3)
```



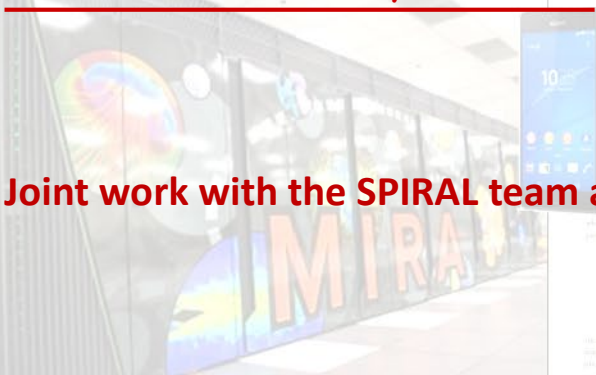
```
pd(s5678, s5679, (1) | (3)
pd(s5680, s5681)
pd(s5680, s5681)
pd(s5676, s5682)
pd(s5677, s5680)
pd(s5678, s5679)
pd(s5677, s5683)
pd(s5670, s5671)
pd(s5670, s5671)
```

Walter G. Carrara
Ron S. Goodman
J. M. Maier, Jr.
Edward A. Meehan

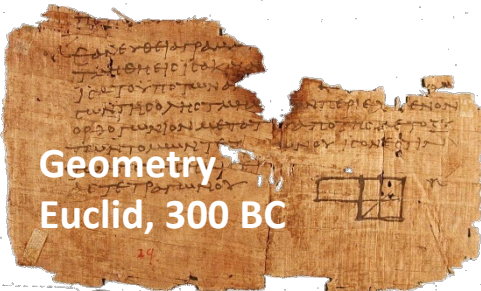


```
cast_sd(&(C22), t5735);
cast_sd(&(C22), t5736);
sub_pd(s5677, s5683);
sub_pd(s5676, s5682);
```

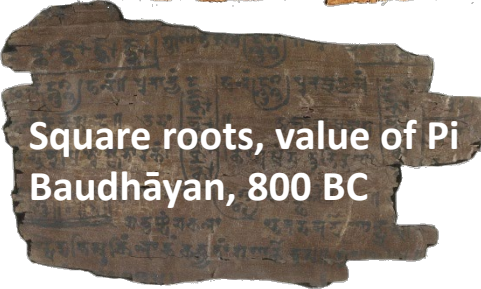
Intel
Integrated
Performance
Primitives



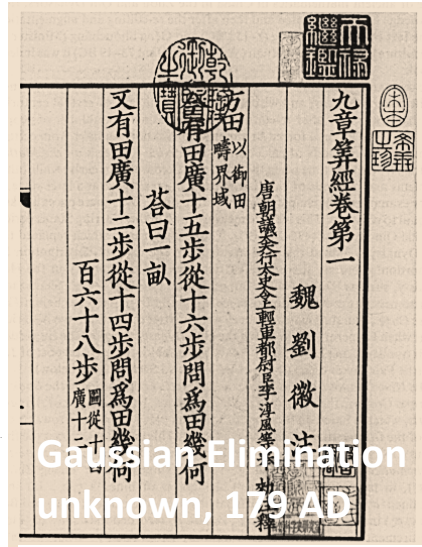
Algorithms and Mathematics: 2,500+ Years



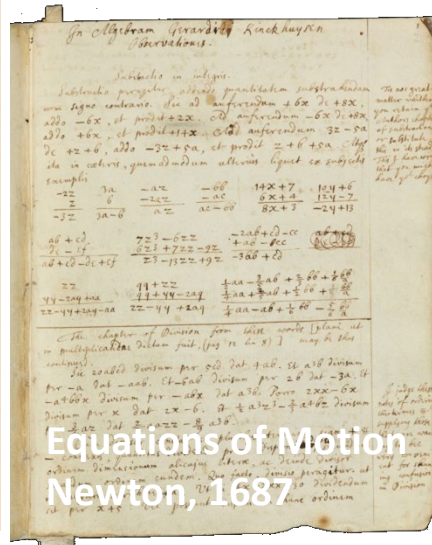
Geometry
Euclid, 300 BC



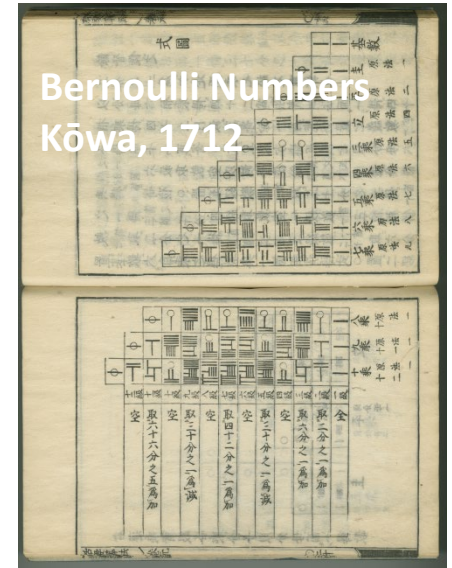
Square roots, value of Pi
Baudhāyan, 800 BC



Gaussian Elimination
unknown, 179 AD



Equations of Motion
Newton, 1687

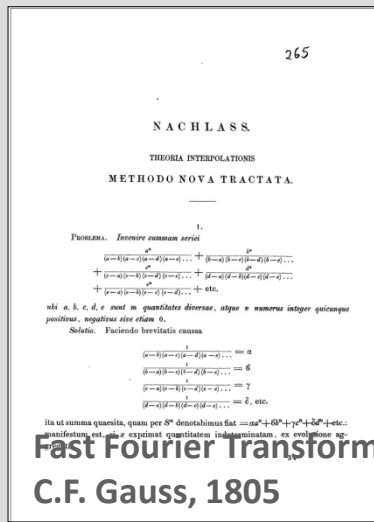


Bernoulli Numbers
Kōwa, 1712

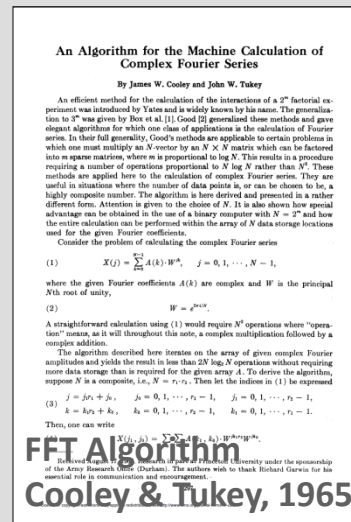


Algebra
al-Khwārizmī, 830

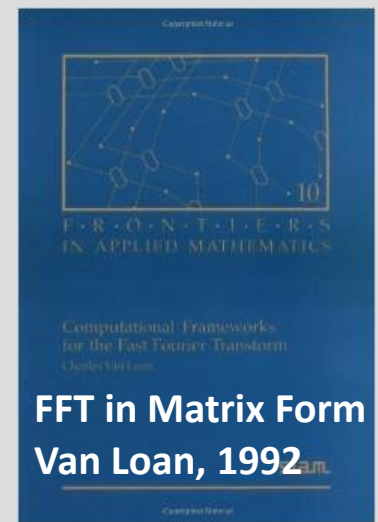
Fast Fourier Transform



Fast Fourier Transform
C.F. Gauss, 1805



FFT Algorithm
Cooley & Tukey, 1965



FFT in Matrix Form
Van Loan, 1992

Computing Platforms Over The Years

F-16A/B, C/D, E/F, IN, IQ, N, V: Flying since 1974



Compare: Desktop/workstation class CPUs/machines

Assembly code compatible !!

7



x86 binary compatible, but 500x parallelism ?!

1972

Intel 8008
0.2—0.8 MHz
Intelligent terminal

1989

IBM PC/XT compatible
8088 @ 8 MHz, 640kB RAM
360 kB FDD, 720x348 mono

1994

IBM RS/6000-390
256 MB RAM, 6GB HDD
67 MHz Power2+, AIX

2006

GeForce 8800
1.3 GHz, 128 shaders
16-way SIMT

2011

Xeon Phi
1.3 GHz, 60 cores
8/16-way SIMD

2018

Xeon Platinum 8180M
28 cores, 2.5-3.6 GHz
2/4/8/16-way SIMD

10⁷ – 10⁸ compounded performance gain over 45 years

Programming/Languages Libraries Timeline

Popular performance programming languages

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

Popular performance libraries

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

Popular productivity/scripting languages

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

2019: What \$1M Can Buy You



Dell PowerEdge R940
4.5 Tflop/s, 6 TB, 850 W
 4x 28 cores, 2.5 GHz



24U rack
7.5kW
<\$1M



OSS FSAAn-4
200 TB PCIe NVMe flash
 80 GB/s throughput



BittWare TeraBox
18M logic elements, 4.9 Tb/sec I/O
 8 FPGA cards/16 FPGAs, 2 TB DDR4



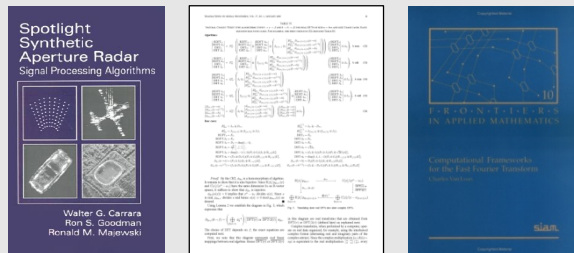
AberSAN ZXP4
90x 12TB HDD, 1 kW
 1PB raw



Nvidia DGX-1
8x Tesla V100, 3.2 kW
 170 Tflop/s, 128 GB

SPIRAL: AI for High Performance Code

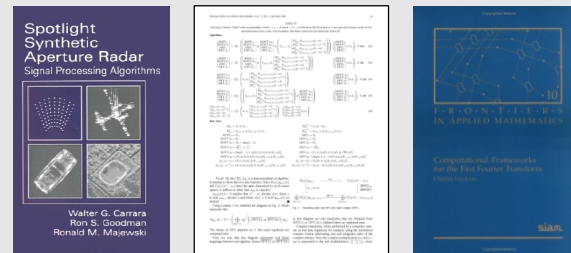
Traditionally



High performance library
optimized for given platform

*Comparable
performance*

SPIRAL Approach



SPIRAL



High performance library
optimized for given platform

Outline

- Introduction
- **Specifying computation**
- Achieving Performance Portability
- **FFTX: A Library Frontend for SPIRAL**
- Summary

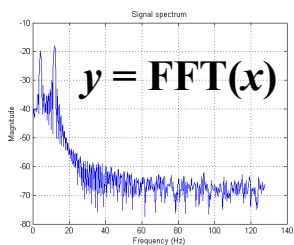
SPIRAL: AI for Performance Engineering

Given:

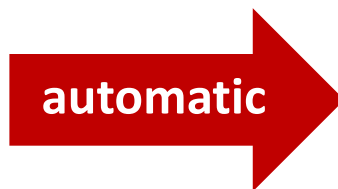
- Mathematical problem specification
core mathematics does not change
- Target computer platform
varies greatly, new platforms introduced often

Wanted:

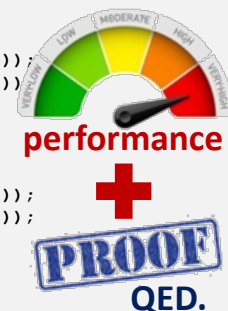
- Very good implementation of specification on platform
- Proof of correctness



on



```
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

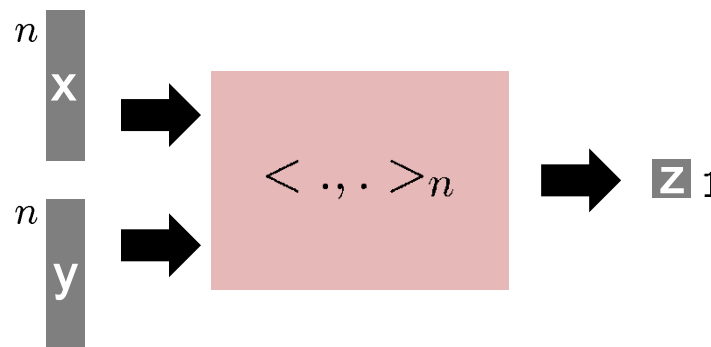
- **Operator: Multiple vectors ! Multiple vectors**
- **Stateless**
- **Higher-dimensional data is linearized**
- **Operators are potentially nonlinear**

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

Example: Scalar product

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

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



Example: Safety Distance as OL Operator

■ Passive Safety of Robots

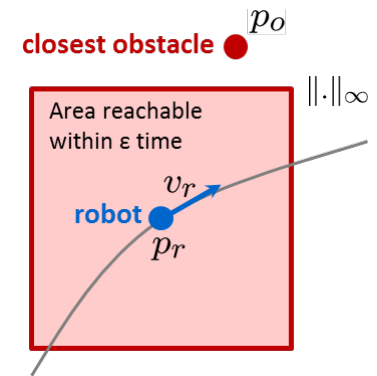
p_o : Position of closest obstacle

p_r : Position of robot

v_r : Longitudinal velocity of robot

A, b, V, ϵ : constants

$$\|p_r - p_o\|_\infty > \frac{v_r^2}{2b} + V \frac{v_r}{b} + \left(\frac{A}{b} + 1\right) \left(\frac{A}{2}\epsilon^2 + \epsilon(v_r + V)\right)$$



■ Definition as operator

$\text{SafeDist}_{V,A,b,\epsilon} : \mathbb{R} \times \mathbb{R}^2 \times \mathbb{R}^2 \rightarrow \mathbb{Z}_2$

$(v_r, p_r, p_o) \mapsto (p(v_r) < d_\infty(p_r, p_o))$ with $d_\infty(\vec{x}, \vec{y}) = \|\vec{x} - \vec{y}\|_\infty$

$$p(x) = \alpha x^2 + \beta x + \gamma$$

$$\alpha = \frac{1}{2b}$$

$$\beta = \frac{V}{b} + \epsilon \left(\frac{A}{b} + 1\right)$$

$$\gamma = \left(\frac{A}{b} + 1\right) \left(\frac{A}{2}\epsilon^2 + \epsilon V\right)$$

Formalizing Mathematical Objects in OL

■ Infinity norm

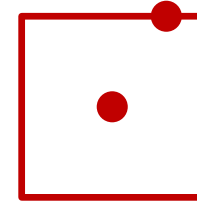
$$\|\cdot\|_{\infty}^n : \mathbb{R}^n \rightarrow \mathbb{R}$$

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

■ Chebyshev distance

$$d_{\infty}^n(\cdot, \cdot) : \mathbb{R}^n \times \mathbb{R}^n \rightarrow \mathbb{R}$$

$$(x, y) \mapsto \|x - y\|_{\infty}^n$$



■ Vector subtraction

$$(-)_n : \mathbb{R}^n \times \mathbb{R}^n \rightarrow \mathbb{R}^n$$

$$(x, y) \mapsto x - y$$

■ Pointwise comparison

$$(<)_n : \mathbb{R}^n \times \mathbb{R}^n \rightarrow \mathbb{Z}_2^n$$

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

■ Scalar product

$$< \cdot, \cdot >_n : \mathbb{R}^n \times \mathbb{R}^n \rightarrow \mathbb{R}$$

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

■ Monomial enumerator

$$(x^i)_n : \mathbb{R} \rightarrow \mathbb{R}^{n+1}$$

$$x \mapsto (x^i)_{i=0,\dots,n}$$

■ Polynomial evaluation

$$P[x, (a_0, \dots, a_n)] : \mathbb{R} \rightarrow \mathbb{R}$$

$$x \mapsto a_0 x^n + a_1 x^{n-1} + \dots + a_{n-1} x + a_n$$

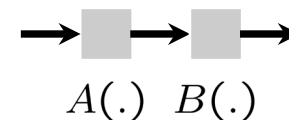
Beyond the textbook: explicit vector length, infix operators as prefix operators

Operations and Operator Expressions

■ Operations (higher-order operators)

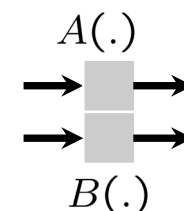
$$\circ : (D \rightarrow S) \times (S \rightarrow R) \rightarrow (D \rightarrow R)$$

$$(A, B) \mapsto B \circ A$$



$$\times : (D \rightarrow R) \times (E \rightarrow S) \rightarrow (D \times E \rightarrow R \times S)$$

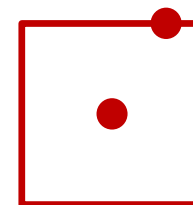
$$(A, B) \mapsto \left((x, y) \mapsto (A(x), B(y)) \right)$$



■ Operator expressions are operators

$$\|\cdot\|_{\infty}^n \circ (-)_n : \mathbb{R}^n \times \mathbb{R}^n \rightarrow \mathbb{R}$$

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



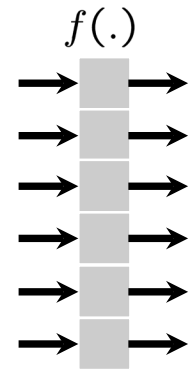
■ Short-hand notation: Infix notation

$$A(\cdot) - B(\cdot) = \left(x \mapsto A(x) - B(x) \right) \quad \text{can be expressed via} \quad (-)_n : \mathbb{R}^n \times \mathbb{R}^n \rightarrow \mathbb{R}^n$$

$$(x, y) \mapsto x - y$$

Basic OL Operators

Basic operators \approx functional programming constructs



map

$$\text{Pointwise}_{n, f_i} : \mathbb{R}^n \rightarrow \mathbb{R}^n$$

$$(x_i)_i \mapsto f_0(x_0) \oplus \cdots \oplus f_{n-1}(x_{n-1})$$

binop

$$\text{Atomic}_{f(.,.)} : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$$

$$(x, y) \mapsto f(x, y)$$

map + zip

$$\text{Pointwise}_{n \times n, f_i} : \mathbb{R}^n \times \mathbb{R}^n \rightarrow \mathbb{R}^n$$

$$((x_i)_i, (y_i)_i) \mapsto f_0(x_0, y_0) \oplus \cdots \oplus f_{n-1}(x_{n-1}, y_{n-1})$$

fold

$$\text{Reduction}_{n, f_i} : \mathbb{R}^n \rightarrow \mathbb{R}$$

$$(x_i)_i \mapsto f_{n-1}(x_{n-1}, f_{n-2}(x_{n-2}, f_{n-3}(\dots f_0(x_0, \text{id}()) \dots)))$$

unfold

$$\text{Induction}_{n, f_i} : \mathbb{R} \rightarrow \mathbb{R}^{n+1}$$

$$x \mapsto (f_n(x, f_{n-1}(\dots)), \dots, f_2(x, f_1(x, \text{id})), f_1(x, \text{id}), \text{id}())$$

Safety distance as (optimized) operator expression

$$\text{SafeDist}_{V, A, b, \varepsilon} = \text{Atomic}_{(x, y) \mapsto x < y}$$

$$\circ \left(\left(\text{Reduction}_{3, (x, y) \mapsto x + y} \circ \text{Pointwise}_{3, x \mapsto a_i x} \circ \text{Induction}_{3, (a, b) \mapsto ab, 1} \right) \right.$$

$$\left. \times \left(\text{Reduction}_{2, (x, y) \mapsto \max(|x|, |y|)} \circ \text{Pointwise}_{2 \times 2, (x, y) \mapsto x - y} \right) \right)$$

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

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

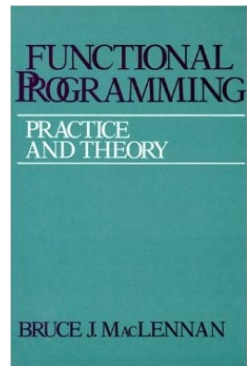
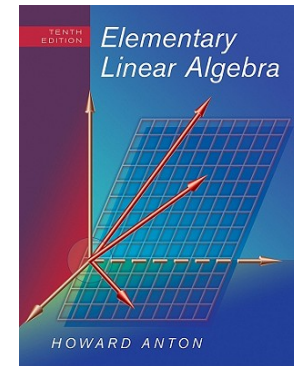
$$(\diamond)_n \rightarrow \text{Pointwise}_{n \times n, (a,b) \mapsto a \diamond b}, \quad \diamond \in \{+, -, \cdot, \wedge, \vee, \dots\}$$

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

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

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

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



Library of well-known identities expressed in OL

Loop and Code Level Rule System

Mathematical Loop Abstraction

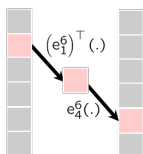
- Selection and embedding operator: *gather and scatter*

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

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

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

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

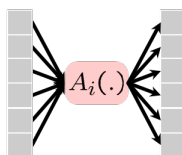


- Iterative operations: *loop*

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

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

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



- Atomic operators: *nonlinear scalar functions*

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

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



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

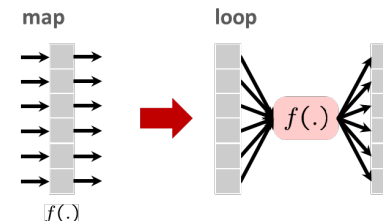
```
# Dynamic Window Monitor
let(
  i3 := var("i3", TInt), i5 := var("i5", TInt),
  w2 := var("w2", TBool), w1 := var("w1", T_Real(64)),
  s8 := var("s8", T_Real(64)), s7 := var("s7", T_Real(64)),
  s6 := var("s6", T_Real(64)), s5 := var("s5", T_Real(64)),
  s4 := var("s4", T_Real(64)), s1 := var("s1", 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(geq(w1, V(0)), w1, neg(w1))),
          assign(s1, cond(geq(s1, s6), s1, s6))
        )
      ),
      assign(w2, geq(s1, s5)),
      creturn(w2)
    )
  )
)
```

Translation and Optimization

- Translating Basic OL into Σ -OL

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

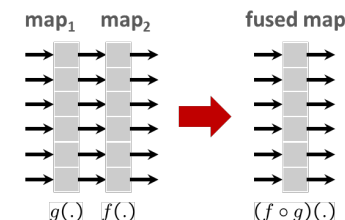
$$\text{Reduction}_{n, (a,b) \rightarrow a+b} \rightarrow \sum_{i=0}^{n-1} (e_i^n)^\top$$



- Optimizing Basic OL/ Σ -OL

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

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



Rule Based Compiler

Compilation rules: recursive descent

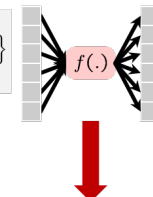
$$\text{Code}(y = (A \circ B)(x)) \rightarrow \{\text{decl}(t), \text{Code}(t = B(x)), \text{Code}(y = A(t))\}$$

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

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

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

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



Cleanup rules: term rewriting

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

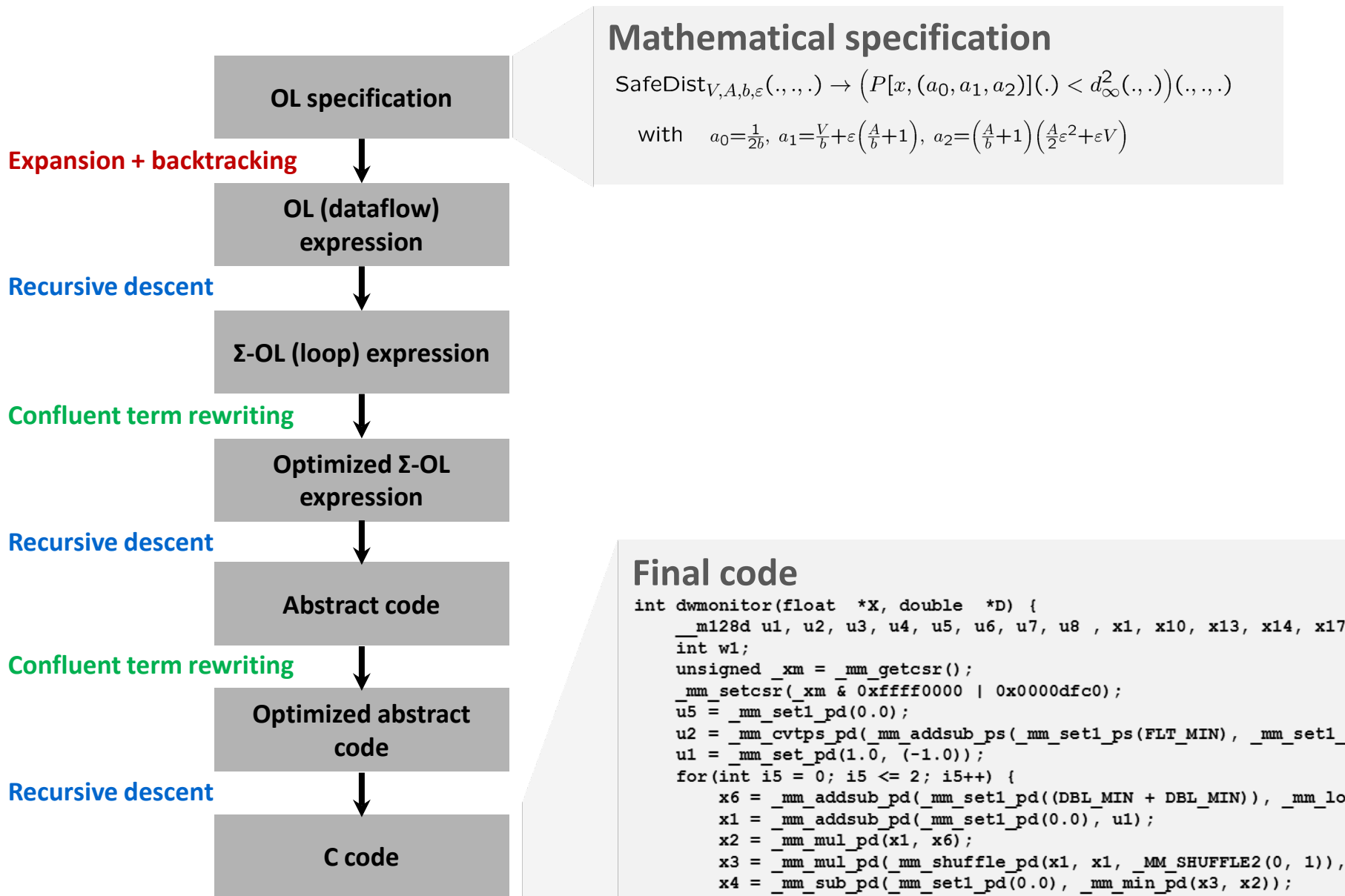
$$\text{decl}(D, \text{decl}(E, c)) \rightarrow \text{decl}([D, E], c)$$

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

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

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

Putting it Together: One Big Rule System



Final Synthesized C Code

```

int dwmonitor(float *X, double *D) {
  __m128d u1, u2, u3, u4, u5, u6, u7, u8 , x1, x10, x13, x14, x17, x18, x19, x2, x3, x4, x6, x7, x8, x9;
  int w1;
  unsigned _xm = __mm_getcsr();
  __mm_setcsr(_xm & 0xffff0000 | 0x0000dfc0);
  u5 = __mm_set1_pd(0.0);
  u2 = __mm_cvtps_pd(__mm_addsub_ps(__mm_set1_ps(FLT_MIN), __mm_set1_ps(X[0])));
  u1 = __mm_set_pd(1.0, (-1.0));
  for(int i5 = 0; i5 <= 2; i5++) {
    x6 = __mm_addsub_pd(__mm_set1_pd((DBL_MIN + DBL_MIN)), __mm_loadup_pd(&(D[i5])));
    x1 = __mm_addsub_pd(__mm_set1_pd(0.0), u1);
    x2 = __mm_mul_pd(x1, x6);
    x3 = __mm_mul_pd(__mm_shuffle_pd(x1, x1, _MM_SHUFFLE2(0, 1)), x6);

```

SafeDist $_{V,A,b,\varepsilon} = \text{Atomic}_{(x,y) \mapsto x < y}$

$$\circ \left(\left(\text{Reduction}_{3,(x,y) \mapsto x+y} \circ \text{Pointwise}_{3,x \mapsto a_i x} \circ \text{Induction}_{3,(a,b) \mapsto ab,1} \right) \right. \\
 \left. \times \left(\text{Reduction}_{2,(x,y) \mapsto \max(|x|,|y|)} \circ \text{Pointwise}_{2 \times 2,(x,y) \mapsto x-y} \right) \right)$$

```

}
u6
for
  u8 = __mm_cvtps_pd(__mm_addsub_ps(__mm_set1_ps(FLT_MIN), __mm_set1_ps(X[(i3 + 1)])));
  u7 = __mm_cvtps_pd(__mm_addsub_ps(__mm_set1_ps(FLT_MIN), __mm_set1_ps(X[(3 + i3)])));
  x14 = __mm_add_pd(u8, __mm_shuffle_pd(u7, u7, _MM_SHUFFLE2(0, 1)));
  x13 = __mm_shuffle_pd(x14, x14, _MM_SHUFFLE2(0, 1));
  u4 = __mm_shuffle_pd(__mm_min_pd(x14, x13), __mm_max_pd(x14, x13), _MM_SHUFFLE2(1, 0));
  u6 = __mm_shuffle_pd(__mm_min_pd(u6, u4), __mm_max_pd(u6, u4), _MM_SHUFFLE2(1, 0));
}
x17 = __mm_addsub_pd(__mm_set1_pd(0.0), u6);
x18 = __mm_addsub_pd(__mm_set1_pd(0.0), u5);
x19 = __mm_cmpge_pd(x17, __mm_shuffle_pd(x18, x18, _MM_SHUFFLE2(0, 1)));
w1 = (__mm_testc_si128(__mm_castpd_si128(x19), __mm_set_epi32(0xffffffff, 0xffffffff, 0xffffffff, 0xffffffff)) -
      (__mm_testnzc_si128(__mm_castpd_si128(x19), __mm_set_epi32(0xffffffff, 0xffffffff, 0xffffffff, 0xffffffff))));
__asm nop;
if (__mm_getcsr() & 0x0d) {
  __mm_setcsr(_xm);
  return -1;
}
__mm_setcsr(_xm);
return w1;
}

```

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

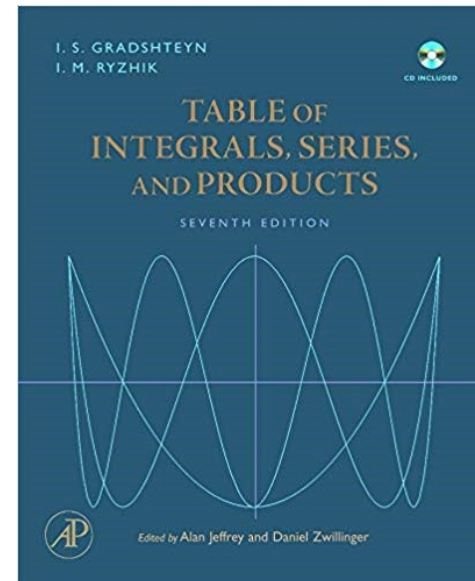
Table of Integrals

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

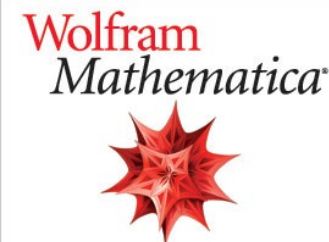


$$\text{In}[31]:= \int_0^{2\pi} \frac{1}{a^2 \cos[t]^2 + b^2 \sin[t]^2} dt$$

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

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

Out[33]= 0

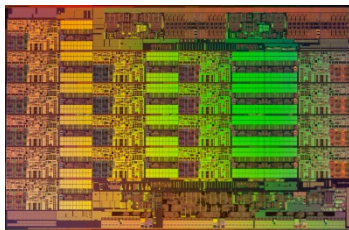


Outline

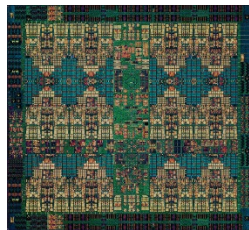
- Introduction
- Specifying computation
- **Achieving Performance Portability**
- FFTX: A Library Frontend for SPIRAL
- Summary

Today's Computing Landscape

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



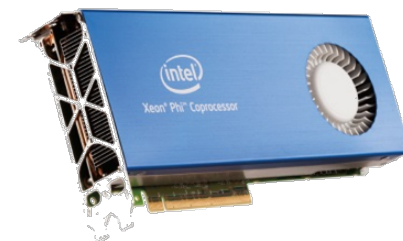
Intel Xeon 8180M
2.25 Tflop/s, 205 W
 28 cores, 2.5—3.8 GHz
 2-way—16-way AVX-512



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



Nvidia Tesla V100
7.8 Tflop/s, 300 W
 5120 cores, 1.2 GHz
 32-way SIMT



Intel Xeon Phi 7290F
1.7 Tflop/s, 260 W
 72 cores, 1.5 GHz
 8-way/16-way LRBni



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



Intel Atom C3858
32 Gflop/s, 25 W
 16 cores, 2.0 GHz
 2-way/4-way SSSE3



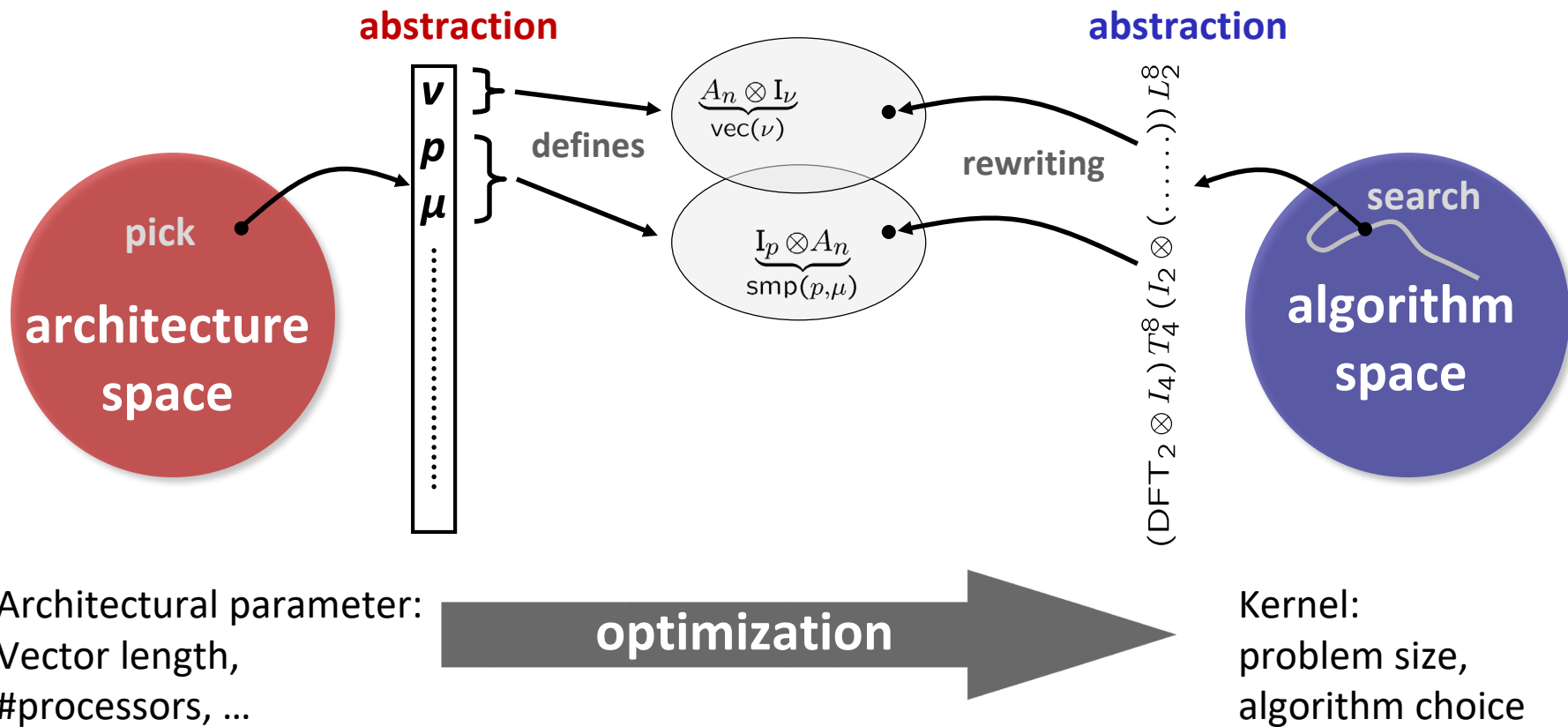
Dell PowerEdge R940
3.2 Tflop/s, 6 TB, 850 W
 4x 24 cores, 2.1 GHz
 4-way/8-way AVX



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

Platform-Aware Formal Program Synthesis

Model: common abstraction
 = spaces of matching formulas



Some Application Domains in OL

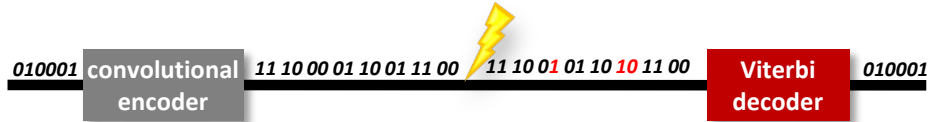
Linear Transforms

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

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

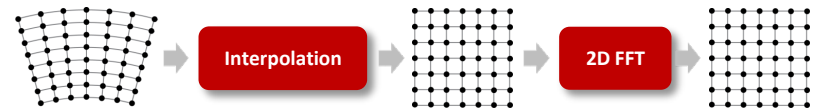


$$\mathbf{F}_{K,F} \rightarrow \prod_{i=1}^F \left((\text{I}_{2^{K-2}} \otimes_j B_{F-i,j}) \text{L}_{2^{K-2}}^{2^{K-1}} \right)$$

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

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

Synthetic Aperture Radar (SAR)



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

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

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

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

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

Formal Approach for all Types of Parallelism

- **Multithreading** (Multicore)

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

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

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

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

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

- **Streaming/multibuffering** (Cell)

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

- **Graphics Processors** (GPUs)

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

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

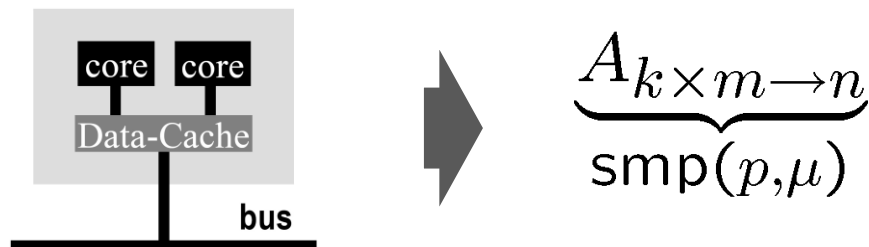
$$\prod_{i=0}^{n-1} A_i^{ir}, \quad I_s \tilde{\otimes} A, \quad \underbrace{L_n^m}_{\text{bram}}$$

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

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

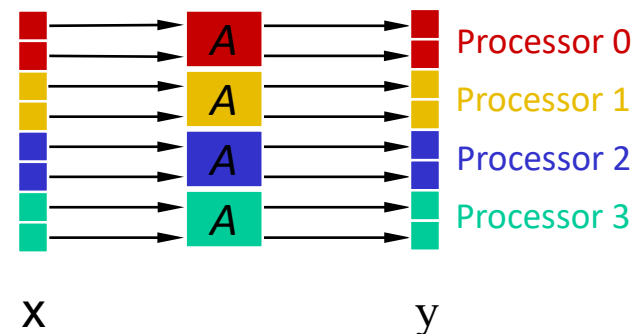
Modeling Hardware: Base Cases

- Hardware abstraction: shared cache with cache lines



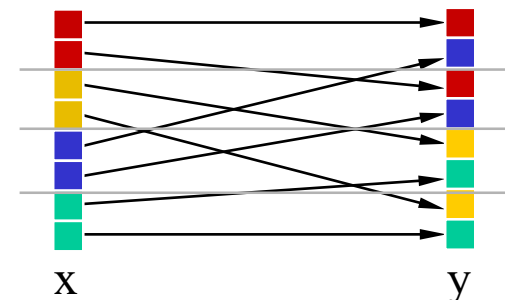
- Tensor product: embarrassingly parallel operator

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



- Permutation: problematic; may produce false sharing

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



Example Program Transformation Rule Set

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

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

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

Recursive rules

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

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

Base case rules

Autotuning in Constraint Solution Space

AVX 2-way
_Complex double

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

DFT_8

Base cases

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

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

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

Transformation rules

$(\text{I}_m \otimes A^{n \times n}) \text{L}_m^{mn} \rightarrow (\text{I}_{m/\nu} \otimes \text{L}_{\nu}^{n\nu} (A^{n \times n} \otimes \text{I}_{\nu})) (\text{L}_{m/\nu}^{mn/\nu} \otimes \text{I}_{\nu})$

$\text{L}_{\nu}^{n\nu} \rightarrow (\text{L}_{\nu}^n \otimes \text{I}_{\nu}) (\text{I}_{n/\nu} \otimes \text{L}_{\nu}^{\nu^2})$

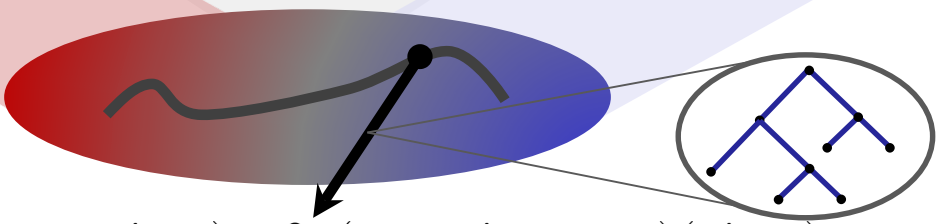
$A^{m \times m} \otimes \text{I}_n \rightarrow (A^{m \times m} \otimes \text{I}_{n/\nu}) \otimes \text{I}_{\nu}$

Breakdown rules

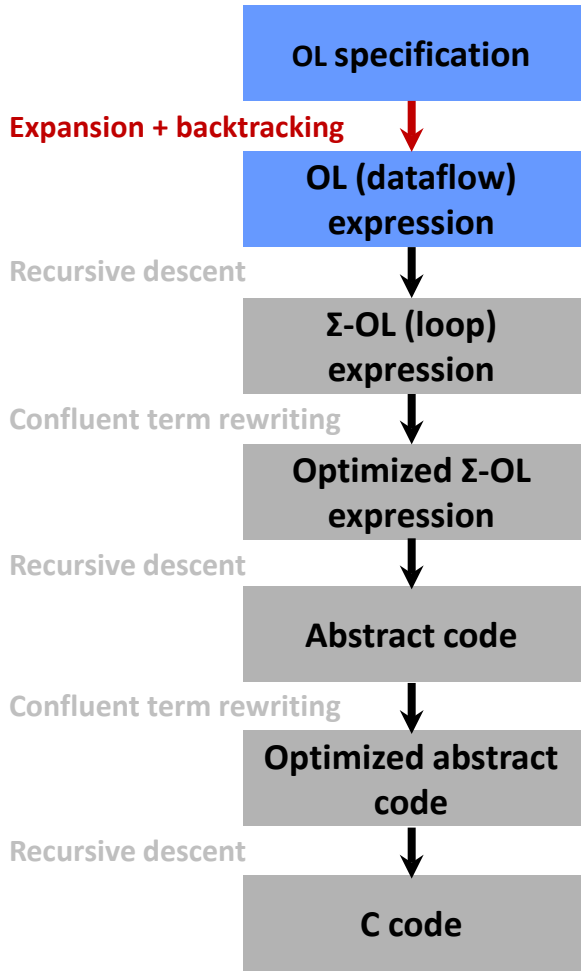
$\text{DFT}_{mn} \rightarrow (\text{DFT}_m \otimes \text{I}_n) \text{T}_n^{mn}$

$(\text{I}_m \otimes \text{DFT}_n) \text{L}_m^{mn}$

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



$((\text{F}_2 \otimes \text{I}_2) \text{T}_2^4 (\text{I}_2 \otimes \text{F}_2) \text{L}_2^4 \vec{I}_2) \underbrace{\text{T}_2^8}_{\text{vec}(2)} \left(\text{I}_2 \otimes \underbrace{\text{L}_2^4}_{\text{vec}(2)} (\text{F}_2 \vec{I}_2) \right) (\text{L}_2^4 \vec{I}_2)$



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:

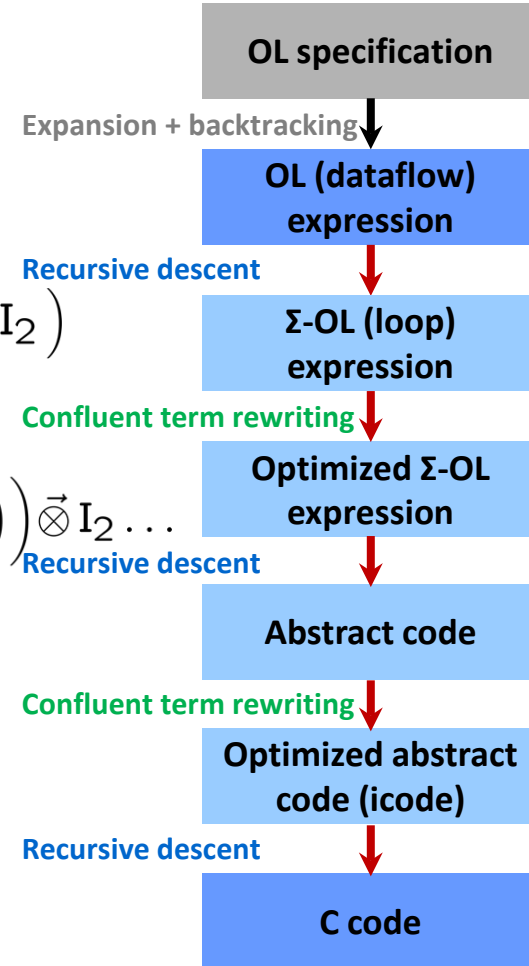
$$\left((F_2 \otimes I_2) T_2^4 (I_2 \otimes F_2) L_2^4 \vec{\otimes} I_2 \right) \underbrace{T_2^8}_{\text{vec}(2)} \left(I_2 \otimes \underbrace{L_2^4}_{\text{vec}(2)} (F_2 \vec{\otimes} I_2) \right) \left(L_2^4 \vec{\otimes} I_2 \right)$$

Σ -OL:

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

C Code:

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



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
- **FFTX: A Library Frontend for SPIRAL**
- Summary

FFTX and SpectralPACK

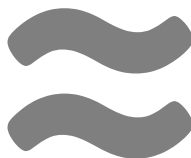
Numerical Linear Algebra

LAPACK

LU factorization
Eigensolves
SVD
...

BLAS

BLAS-1
BLAS-2
BLAS-3



Spectral Algorithms

SpectralPACK

Convolution
Correlation
Upsampling
Poisson solver
...

FFTX

DFT, RDFT
1D, 2D, 3D,...
batch

Define the LAPACK equivalent for spectral algorithms

- **Define FFTX as the BLAS equivalent**
provide user FFT functionality as well as algorithm building blocks
- **Define class of numerical algorithms to be supported by SpectralPACK**
PDE solver classes (Green's function, sparse in normal/k space,...), signal processing,...
- **Library front-end, code generation and vendor library back-end**
mirror concepts from FFTX layer

FFTX and SpectralPACK solve the “spectral motif” long term

Example: Poisson's Equation in Free Space

Partial differential equation (PDE)

$$\Delta(\Phi) = \rho$$

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

$$D = \text{supp}(\rho) \subset \mathbb{R}^3$$

Poisson's equation. Δ is the Laplace operator

Solution characterization

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

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

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

Approach: Green's function

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

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

Method of Local Corrections (MLC)

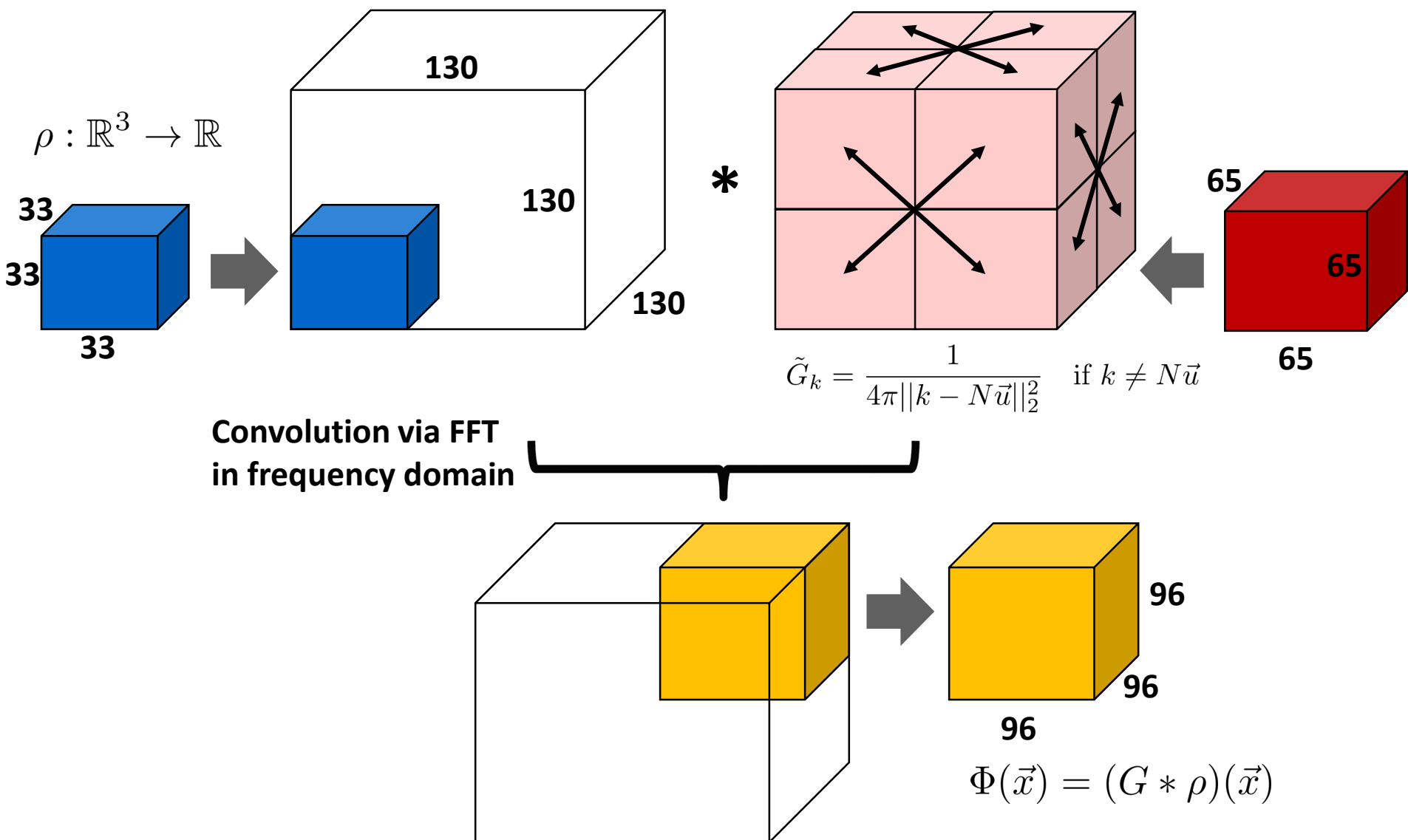
$$\tilde{G}_k = \frac{1}{4\pi\|k - N\vec{u}\|_2^2} \quad \text{if } k \neq N\vec{u}$$

Green's function kernel in frequency domain

P. McCorquodale, P. Colella, G. T. Balls, and S. B. Baden: **A Local Corrections Algorithm for Solving Poisson's Equation in Three Dimensions**. Communications in Applied Mathematics and Computational Science Vol. 2, No. 1 (2007), pp. 57-81., 2007.

C. R. Anderson: **A method of local corrections for computing the velocity field due to a distribution of vortex blobs**. Journal of Computational Physics, vol. 62, no. 1, pp. 111-123, 1986.

Algorithm: Hockney Free Space Convolution



Hockney: Convolution + problem specific zero padding and output subset

FFTX C++ Code: Hockney Free Space Convolution

```
box_t<3> inputBox(point_t<3>({{0,0,0}}),point_t<3>({32,32,32}));  
array_t<3, double> rho(inputBox);  
// ... set input values.
```

```
box_t<3> transformBox(point_t<3>({{0,0,0}}),point_t<3>({{129,129,129}}));  
box_t<3> outputBox(point_t<3>({33,33,33}),point_t<3>({129,129,129}));
```

```
point_t<3> kindF({{DFT,DFT,DFT}});
```

```
size_t
```

```
auto fo  
pla
```

```
auto ke
```

```
point_t  
auto in
```

```
auto so
```

```
context  
context
```

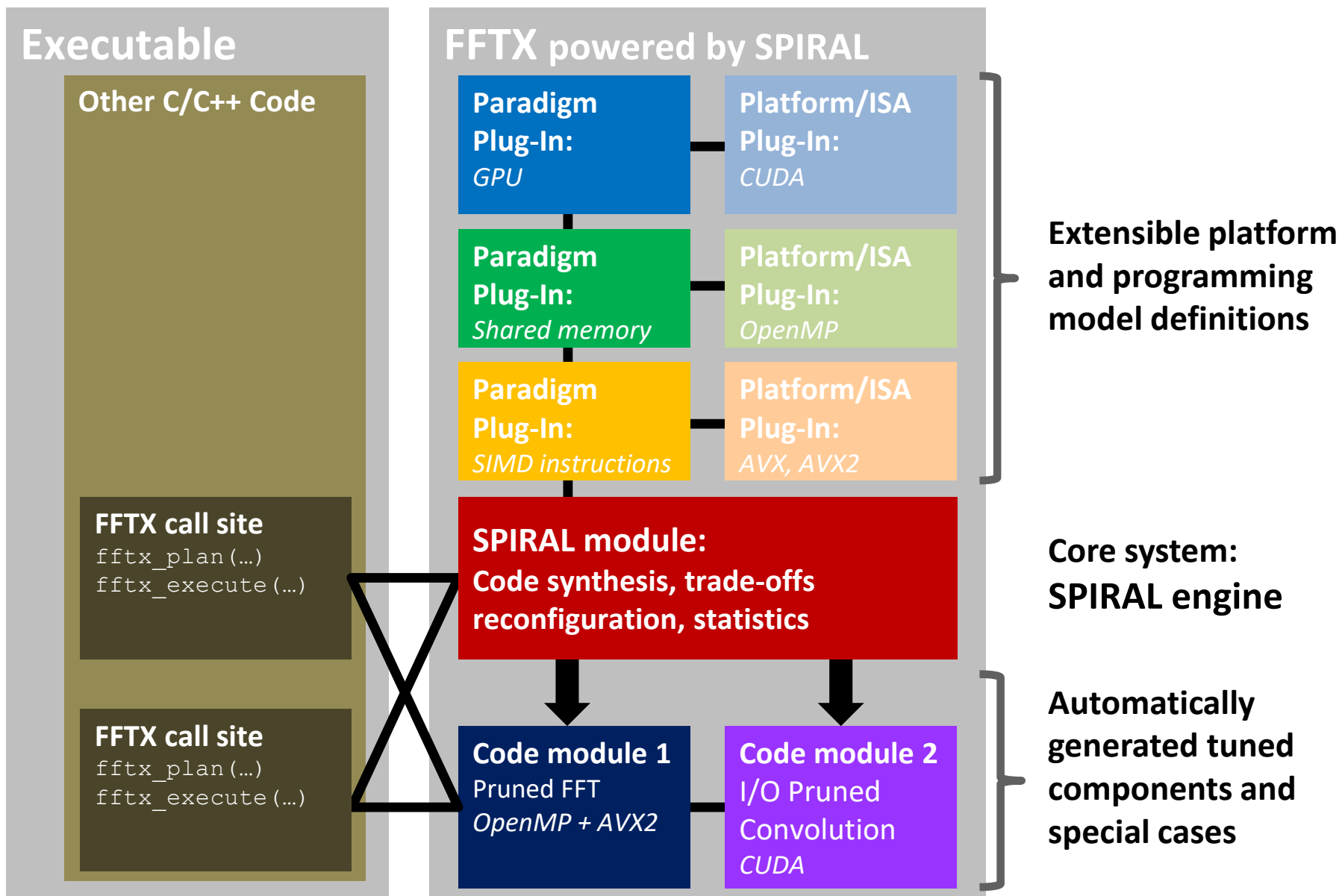
This is a specification dressed as a program

- Needs to be clean and concise
- No code level optimizations and tricks
- Don't think "performance" but "correctness"
- *For production code and software engineering*

```
std::ofstream splFile("hockney.spl");  
export_spl(context, solver, splFile, "hockney33_97_130");  
splFile.close();  
// Offline codegen.  
auto fptr = import_spl<3, double, double>("hockney33_97_130");  
array_t<3, double> Phi(inputBox);  
fptr(&rho, &Phi, 1);
```

```
lize);
```

FFTX Backend: SPIRAL



C/C++ FFTX Program Trace

```

fftx_session := [
  rec(op := "fftx_init", flags := IntHexString("8000000")),
  rec(op := "fftx_create_data_real", rank := 1, dims := [ rec(n := 4, is := 1, os := 1) ],
    ptr := IntHexString("000000D236DD2460")),
  ...
  rec(op := "fftx_create_zero_temp_real", rank := 1,
    dim
  rec(op
    data
    is
    of
  rec(op
    rank
    how
    inp
    fla
  rec(op
    rank
    ds
    dof
    inp
    data
  callback := [
    rec(op := "call", inp := IntHexString("A000000000000001"),
      outp := IntHexString("A000000000000002"), data := IntHexString("A000000000000003")),
    rec(op := "FFTX_COMPLEX_VAR", var := IntHexString("000000D236A0FA30"),
      re := 0.000000e+00, im := 0.000000e+00),
    rec(op := "FFTX_COMPLEX_MOV", target := IntHexString("000000D236A0FA30"),
      source := IntHexString("A000000000000001")),
    rec(op := "FFTX_COMPLEX_MUL", target := IntHexString("000000D236A0FA30"),
      source := IntHexString("A000000000000003")),
    ...
  ]

```

The whole convolution kernel is captured

- DAG with all dependencies
- User-defined call-backs
- Captures pruning, zero-padding and symmetries
- *Lifts sequence of C++ library calls to a specification*

SPIRAL Script Captures Performance Engineering

```
# Pruned 3D Real Convolution Pattern
```

```
Import(realdft);
```

```
Import(filtering);
```

```
# set up algorithms needed for multi-dimensional pruned real convolution
```

Recognizes pattern and applies code generation

- Developed by performance engineer + application specialist
- Casts FTX call sequence as SPIRAL non-terminal
- Does code generation and autotuning
- *Clear separation of concerns frontend/backend*

```
sym := var.fresh_t("S", TArray(TReal, 2*n_freq));
```

```
t := IOPrunedRConv(N, sym, 1, [minout..N-1], 1, [0..maxin], true);
```

```
# generate code and autotune
```

```
rt := DP(t, opts)[1].ruletree;
```

```
c := CodeRuleTree(rt, opts);
```

```
# create files
```

```
PrintTo(name:".c", PrintCode(name, c, opts));
```

Backend: SPIRAL Code Generation

```

__global__ void ker_code0(int *D48, double *D49, double *D50, double *D51, int *D52, double *X) {
    __shared__ double T235[260];
    ...
    if (((threadIdx.x < 13))) {
        for(int i96 = 0; i96 <= 4; i96++) {
            int a31, a32, a33, a34;
            a31 = (2*i96);
            a32 = (threadIdx.x + (13*a31));
            a33 = (threadIdx.x + (13*((a31 + 5) % 10)));
            a34 = (4*i96);
            *(((T235 + 0) + a34) + (20*threadIdx.x)) = (*(T6 + a32)) + *(T6 + a33));
            *(((1 + (T235 + 0) + a34) + (20*threadIdx.x))) = 0.0;
            *(((2 + (T235 + 0) + a34) + (20*threadIdx.x))) = (*(T6 + a32)) - *(T6 + a33));
            *(((3 + (T235 + 0) + a34) + (20*threadIdx.x))) = 0.0;
        }
        double t261, t262, t263, t264, t265, t266, t267, t268;
        int a129;
        t263 = (*( ((T235+0)+12)+(20*threadIdx.x))) + (*( ((T235+0)+8)+(20*threadIdx.x)));
        t264 = (*( ((T235+0)+12)+(20*threadIdx.x))) - (*( ((T235+0)+8)+(20*threadIdx.x)));
        ...
        *((3 + T5 + a129)) = ((0.58778525229247314*t268) - (0.95105651629515353*t266));
    }
    __syncthreads();
    if (((threadIdx.x < 1))) {
        double t305, t306, t307, t308, t309, t310, t311, t312, t313, t314, t315, t316;
        int a387;
        t305 = (*(T5 + 12)) + *(T5 + 144));
        ...
    }
}

```

FFTX/SPIRAL with
CUDA backend



**Early result:
130 Gflop/s
on par with cuFFT**

3,000 lines of code, kernel fusion, cross call data layout transforms

Outline

- Introduction
- Specifying computation
- Achieving Performance Portability
- FFTX: A Library Frontend for SPIRAL
- **Summary**

FFTX Extension For MASSIF/LANL

Convolution with Rank-4 Tensor Challenge: Fitting Into GPU Memory

MPI vpFFT: Viscoplastic Polycrystals

Periodic Boundary Conds.

Rate-sensitive approach ($n = \text{Viscoplastic exponent}$)

- $\dot{\epsilon}(x) = \dot{\gamma}_o \sum_i m^i(x) \left(\frac{m^i(x) \cdot \sigma(x)}{\tau_o^i(x)} \right)^n$
 - Schmid Tensor
 - Threshold Stress (Hardening of deformed system)
- $\sigma(x) = \sigma(x) + (L^o : \dot{\epsilon}(x) - L^o : \dot{\epsilon}(x))$
 - Stiffness of a Linear Reference Medium
- $\sigma(x) = L^o : \dot{\epsilon}(x) + (\sigma(x) - L^o : \dot{\epsilon}(x))$
 - Fluctuation (Heterogeneity Field)
- $\sigma(x) = L^o : \dot{\epsilon}(x) + \tau(x)$
 - Function of Solution
 - Requires Iterative Procedure

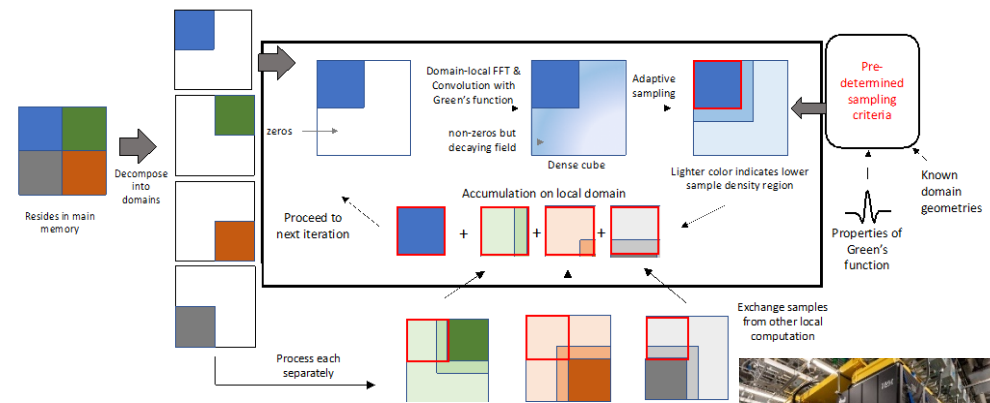
in RVE: $L^o_{ijkl} v_{k,j}(x) + \tau_{o,i}(x) - p_i(x) = 0$
 in RVE: $v_{i,k}(x) = 0$
 Equilibrium + Incompressibility
 periodic boundary conditions across RVE

Green's Function Method, see book by MURA

Slip Geometry: $m^i_j = b^i_j(n^i_j) = \mathbf{b}^i \otimes \mathbf{n}^i$

FFT code MPI parallelized via FFTW

Upon Convergence:
 Stress, Strain-Rate and Slip-Rate Fields are obtained
 Convergence provides a compromise between

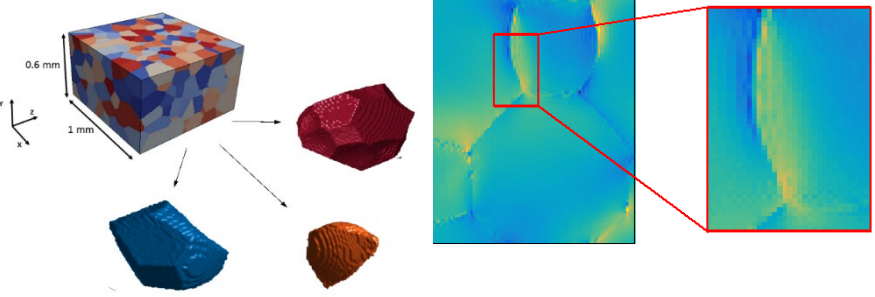


FORTRAN + MATLAB,
 reduce MPI traffic
 towards FFTX/SpectralPACK



R. Lebensohn (LANL), A.D. Rollett (CMU)

Irregular Domain Decomposition Performance Model



90224.994	32
11301.902	64

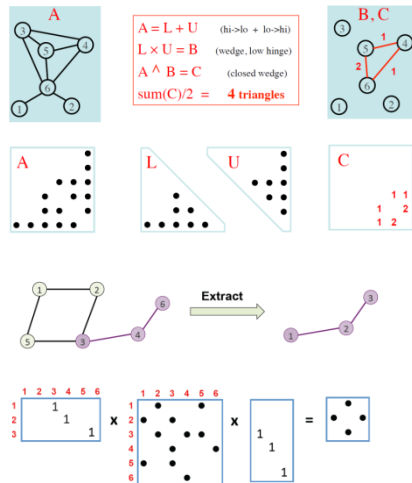
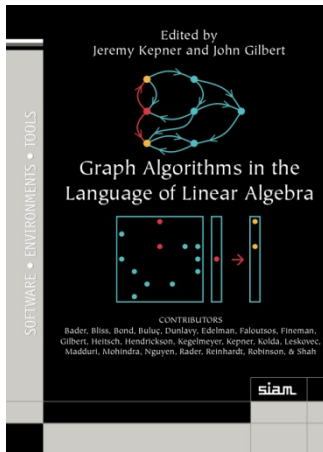
Problem size	Domain size	Compute time/domain [s]	No. of domains	Total compute time (single GPU) [Hrs]	Memory for Domain-local FFT [GB]
1024 × 1024 × 1024	128 × 128 × 128	0.292	512	0.041	1
1024 × 1024 × 1024	512 × 512 × 512	0.329	8	0.0007	4
2048 × 2048 × 2048	128 × 128 × 128	2.352	4096	2.676	4
2048 × 2048 × 2048	512 × 512 × 512	2.485	64	0.044	16
4096 × 4096 × 4096	128 × 128 × 128	19.079	32,768	173.662	16
4096 × 4096 × 4096	512 × 512 × 512	19.589	512	2.786	64
4096 × 4096 × 4096	1024 × 1024 × 1024	20.399	64	0.362	128
8192 × 8192 × 8192	64 × 64 × 64	154.882	2,097,152	90224.994	32
8192 × 8192 × 8192	128 × 128 × 128	155.208	2,62,144	11301.902	64
8192 × 8192 × 8192	512 × 512 × 512	157.272	4096	178.940	256
8192 × 8192 × 8192	1024 × 1024 × 1024	160.303	512	22.798	512

Signal processing + PDE tricks to compress

Model: 8k x 8k x 8k possible on Summit

Graph Algorithms in SPIRAL

Foundation



Triangle Counting in SPIRAL

TriangleCount ()

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

```

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

Formalization

Operation	Mathematical Description	Output	Inputs
mxm	$C \leftarrow M, z = C \odot (A^T \oplus \otimes B^T)$	C	$\neg, M, z, \odot, A, T, \oplus, \otimes, B, T$
mxv, (vxm)	$C \leftarrow m, z = c \odot (A^T \oplus \otimes b)$	c	$\neg, m, z, \odot, A, T, \oplus, \otimes, b$
eWiseMult	$C \leftarrow M, z = C \odot (A^T \otimes B^T)$	C	$\neg, M, z, \odot, A, T, \otimes, B, T$
eWiseAdd	$C \leftarrow M, z = C \odot (A^T \oplus B^T)$	C	$\neg, M, z, \odot, A, T, \oplus, B, T$
reduce (row)	$C \leftarrow m, z = c \odot [\oplus, A^T(:,j)]$	c	$\neg, m, z, \odot, A, T, \oplus$
apply	$C \leftarrow M, z = C \odot f(A^T)$	C	$\neg, M, z, \odot, A, T, f$
transpose	$C \leftarrow M, z = C \odot A^T$	C	$\neg, M, z, \odot, A (T)$
extract	$C \leftarrow M, z = C \odot A^T(i,j)$	C	$\neg, M, z, \odot, A, T, i, j$
assign	$C \leftarrow M, z (i,j) = C(i,j) \odot A^T$	C	$\neg, M, z, \odot, A, T, i, j$
build (meth.)	$C = \text{sum}(i,j, v, \odot)$	C	\odot, m, n, i, j, v
extractTuples (meth.)	$(i,j, v) = A$	i, j, v	A

Notation: i, j – index arrays, v – scalar array, m – 1D mask, other bold-*lower* – vector (column), M – 2D mask, other bold-*caps* – matrix, T – transpose, \neg – structural complement, z – clear output, \oplus mono/ldinary function, \otimes, \oplus semiring, blue – optional parameters, red – optional modifiers



Graph Challenge Champions

- Champions
- Fast Linear Algebra Based Triangle Counting with Kokkos/Keras - Michael Wolf, Mehdi Douzi, Ananthu Bary, Simon Harwood, Sravanthi Kulkarni (Intel)
 - Triangle Counting for Sparse Piv Graphs on GPUs - Zhenhua Zhang, Roger Passos (LLNL)
 - Scalable State and Dynamic Community Detection Using Graphlets - Maheshkumar Kulkarni, Han Lu (ORNL), Ananth Kulkarni (WPI), Arjun Guha (TUM)
 - Parallel Triangle Counting and Prune Identification using Graph-centric Methods - Chad Vogels, Yi-Shan Lu, Inseop Park, Kedar Pingali (TUM)
 - Static Graph Challenge on GPU - Marco Bressan, Manolis Patsis (ONVIM)

- Finalists
- Prune Decomposition on Shared Memory Parallel Systems - Shikun Smith (CMU, Intel), Ting Lin, Naoyan K. Ahmed (Intel), Anup Suresh (CMU), Fabrice Paslet (Intel), George Karypis (CMU)
 - Exploiting Optimization on Shared-memory Platforms for Parallel Triangle Counting Algorithms - Joon Seok Yim (CMU), Sravanthi Kulkarni, Naoyan K. Ahmed, Shikun Smith, Siva Kanyam, Maheshkumar Kulkarni, Braden Liu, Fabrice Paslet (Intel), George Karypis (CMU)
 - Fast Triangle Counting on Extreme Scale - Yang He, Pradyumn Kumar (GHT), Guy Sussan (Raytheon), H. Binwei Huang (GHT)

- Innovation Awards
- An Ensemble Framework for Detecting Community Changes in Dynamic Networks - Timothy La Frat, Geoffrey Denno, Christine Elmski, Yan Fei (Intel)
 - Quickly Finding a Prune in a Hypergraph - Oded Green, Zoran Fan, Ezra Kim (Georgia Tech), Federico Sussan, Nicola Bonavent (TUM), Yannis Keriell Lakatos, Siva Suresh, Shikun Smith, George Karypis, Rajgopal Kamran, Vitor Prassana (CMU), David Isac (Georgia Tech)

- Notable Mentions
- Distributed Triangle Counting in the GraphLab-Metric Math Library - Dylan Henthorn (University of Washington)
 - Fast Local Linear Algebra Based Triangle Counting without Matrix Multiplication - Ta-Ming Lou, Varun Nagaraj Rao, Matthew Lee, Dora Popovic, Franz Frennrich (CMU), Scott McMillin (SEI)
 - Scalable Stochastic Block Partitions - Abhishek Upad (GHT), Gaur Prasad (Raytheon), and H. Binwei Huang (GHT)
 - Superorder-Associative Array Architecture - Eril Debnath, Ananta Cook (Skunkworks), Srikanth Srikanta, Thomas Coates (Georgia Tech)
 - Triangle Counting Via Vectorized Set Intersection - Shikun Smith (University of Pittsburgh)
 - Collaborative GPU + GPU Algorithms for Triangle Counting and Prune Decomposition in the Muddy-Architecture - Karan Datta, Kewen Fan, Rahul Nag (UTCC), Jinyoung Young (IBM), Yun Song Kim, Woo-Min Huh (UTCC)

HPEC Graph Challenge

First Look: Linear Algebra-Based Triangle Counting without Matrix Multiplication

Ta-Ming Lou, Varun Nagaraj Rao, Matthew Lee, Dora Popovic, Franz Frennrich, Scott McMillin
 Department of Electrical and Computer Engineering, Carnegie Mellon University
 Email: lou@cmu.edu, varun@ece.cmu.edu, [mlee@popovic, ffranz@frennrich]@cmu.edu

Abstract—Linear algebra-based approaches to exact triangle counting also require sparse matrix multiplication as a primitive operation. Non-linear algebra approaches to the same problem solve where the adjacency matrix of the graph is not available. In this paper, we show that both approaches can be unified into a single approach that separates the data from the algorithm design. By not calling the triangle counting algorithm into matrix multiplication, a different algorithm that counts each triangle exactly once is possible. In addition, by choosing the appropriate sparse matrix format, we show that the same algorithm is equivalent to the computer-optimized adjacency matrix format. The adjacent matrix of the graph is not available. We show that our approach yields an initial implementation that is between 0.9 and more than 200 times faster than the reference implementation. We also show that the initial implementation can be easily parallelized on shared memory systems.

I. INTRODUCTION

It is generally known that counting the exact number of triangles in a graph G can be described using the language of linear algebra as

$$\frac{1}{2} \text{tr}(A^3)$$

where A is the adjacency matrix of the graph G [1]. Other linear algebra approaches [2], [3] also require a sparse matrix multiplication of A or part of it as part of their computation. Alternative approaches that are not based on linear algebra leverage other forms for describing graphs such as the adjacency list to design their algorithms [4], [5].

In this paper, we show that these approaches are not mutually exclusive. Using the linear algebra approach, we describe an algorithm that computes the number of triangles exactly once. Unlike algorithms described in the language of linear algebra, this algorithm does not require a sparse matrix multiplication, and does not require a subsequent scaling of the number of triangles. This suggests that our algorithm avoids redundant computation and possibly redundant data movement.

We also show that by selecting the appropriate data format when implementing the linear algebra approach, the resulting implementation is similar to an algorithm derived using approaches starting with a description of a graph that is not the adjacency matrix.

Finally, we show that our implementation of exact triangle counting algorithm yields superior performance that is between 0.9 and more than 200 times faster than the reference implementation. Initial parallelization first yielded an additional factor of 1.2 to 1.9 improvement over the sequential implementation on various architectures.

II. TRIANGLE COUNTING ALGORITHM

Let $G = (V, E)$ be a simple undirected graph with vertex set V and edge set E . In addition, assume that V has been partitioned into two disjoint sets, V_1 and V_2 . Under these assumptions, a triangle in G , described using the triple (u, v, w) where $u, v, w \in V$, can be classified into four categories:

- Category 1: Triangles mostly in V_1 . Vertices of these triangles are from V_1 , i.e., $u, v, w \in V_1$.
- Category 2: Triangles mostly in V_2 . Vertices of these triangles are from two vertices in V_2 and one vertex in V_1 , i.e., $u, v \in V_2$, and $w \in V_1$.
- Category 3: Triangles mostly in V_1 . Vertices of these triangles are from two vertices in V_1 and one vertex in V_2 , i.e., $u, v \in V_1$, and $w \in V_2$.
- Category 4: Triangles in V_2 . Vertices of these triangles are from V_1 , i.e., $u, v, w \in V_1$.

Figure 1 describes G and the categories of triangles.

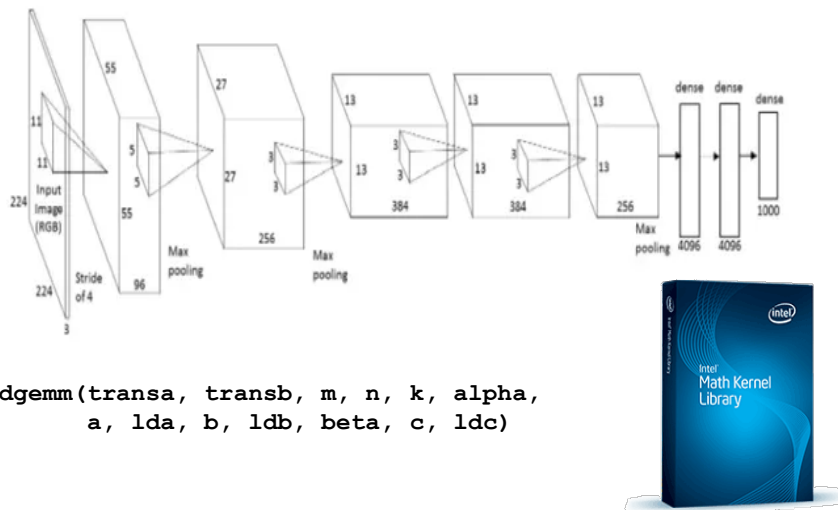
A. Notation

In this paper, we use the sets of $\{1\}$ the number of triangles whose vertices are all in V_1 , $\{2\}$ the number of triangles whose vertices are from two vertices in V_2 and one vertex in V_1 , $\{3\}$ the number of triangles whose vertices are from two vertices in V_1 and one vertex in V_2 , and $\{4\}$ the number of triangles whose vertices are from one vertex in V_1 and two vertices in V_2 . Typically, $\Delta = \{1, 2, 3, 4\}$ since we are interested in the total number of triangles. When all vertices in V_2 have been moved to V_1 , i.e., $V_1 = V$, we will have completed all triangles in G since all the triangles in G will fall in the first category, i.e. all three vertices of each triangle are in V_1 . This means that when $V_2 = V$, $\Delta = \{1\}$ contains the exact number of triangles in G .

Consider an arbitrary vertex v_1 in V_1 that had been selected to be moved to V_2 . Triangles where one of the

Towards Deep Learning in SPIRAL

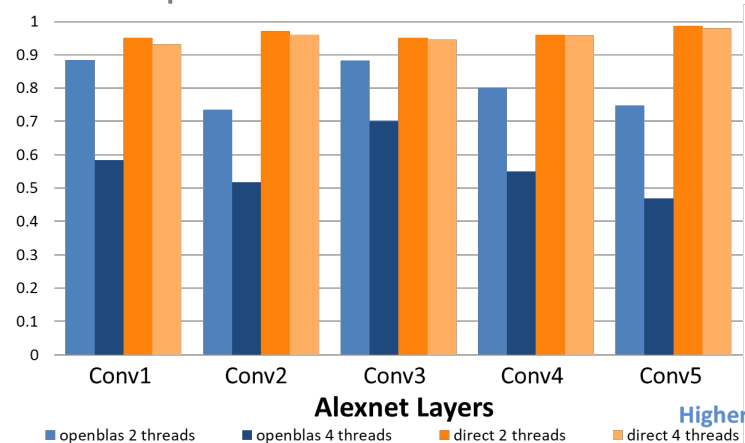
Standard: Use GEMM



Direct CNN – More efficient

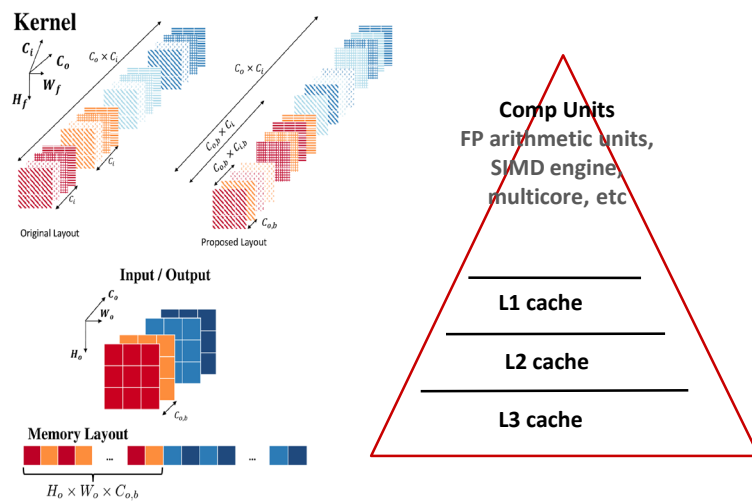
Scalability on AMD Piledriver

Normalized performance to 1 thread



Jiyuan Zhang, Franz Franchetti, Tze Meng Low, "High Performance Zero-Memory Overhead Direct Convolutions", 2018, International Conference of Machine Learning

CNN/System Friendly Layout



Towards CNNs in SPIRAL

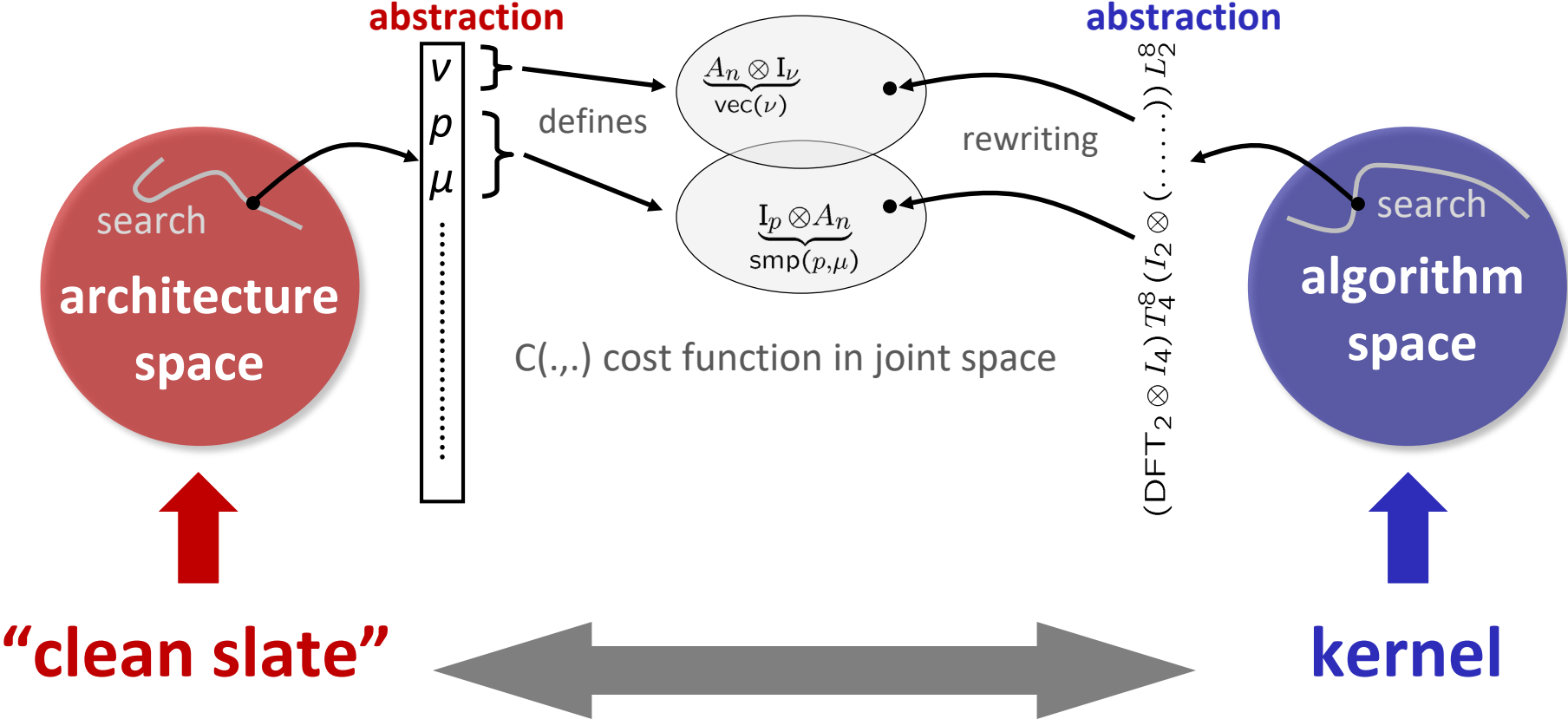
- Level 0: simple C program implements the algorithm cleanly
- Level 1: C macros plus search script use C preprocessor for meta-programming
- Level 2: scripting for code specialization text-based program generation, e.g., ATLAS
- Level 3: add compiler technology internal code representation, e.g., FFTW's genfft
- Level 4: synthesize the program from scratch high level representation, e.g., TCE and Spiral

Co-Optimizing Architecture and Kernel

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

Model: common abstraction
= spaces of matching formulas

Kernel:
problem size,
algorithm choice



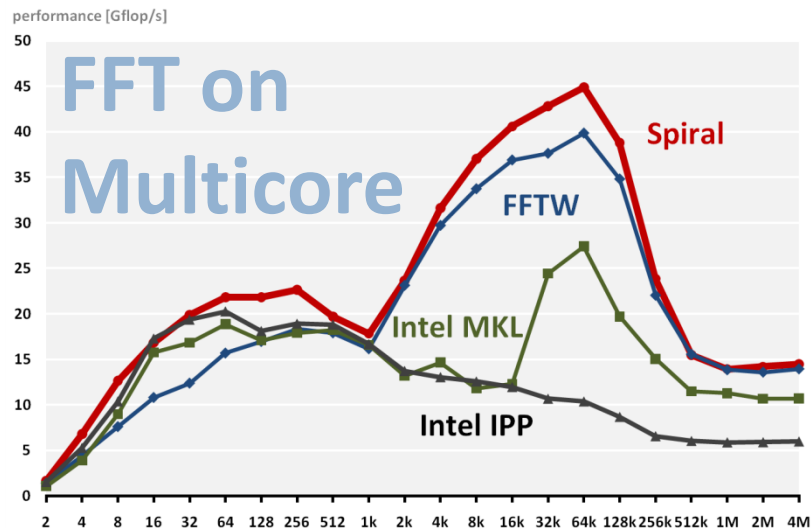
“clean slate”

kernel

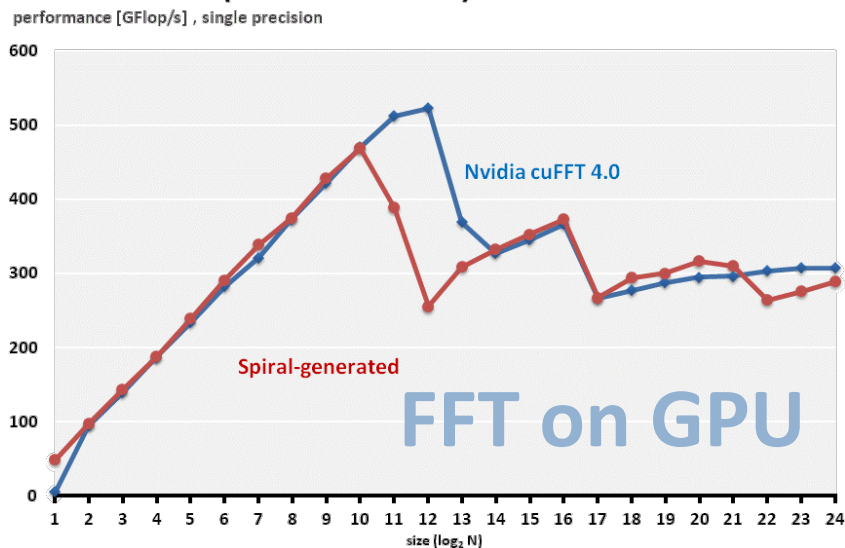
Goal: SPIRAL co-designed RISC-V accelerator chip, taped out

Some Results: FFTs and Spectral Algorithms

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

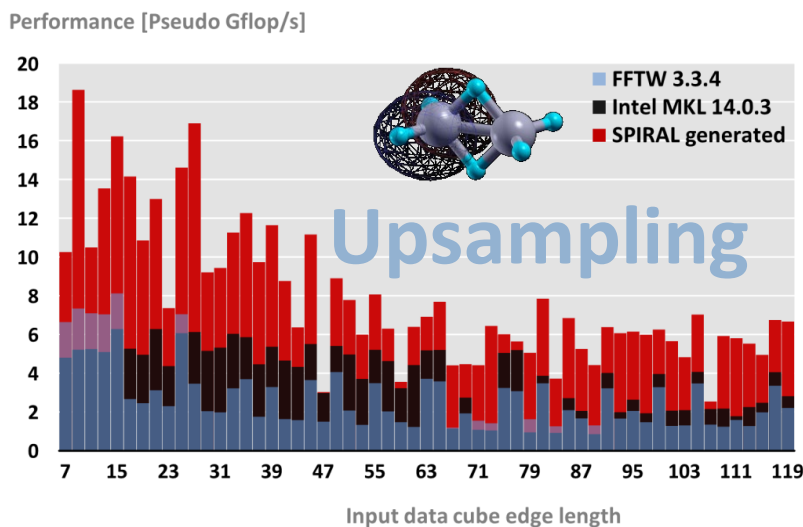


1D Batch DFT (Nvidia GTX 480)



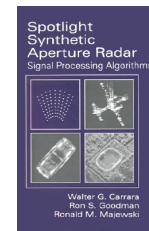
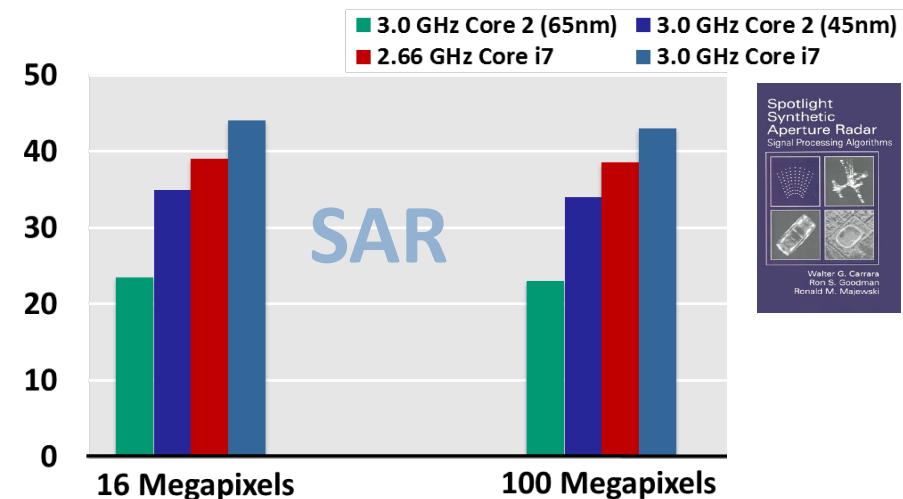
Performance of 2x2x2 Upsampling on Haswell

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



PFA SAR Image Formation on Intel platforms

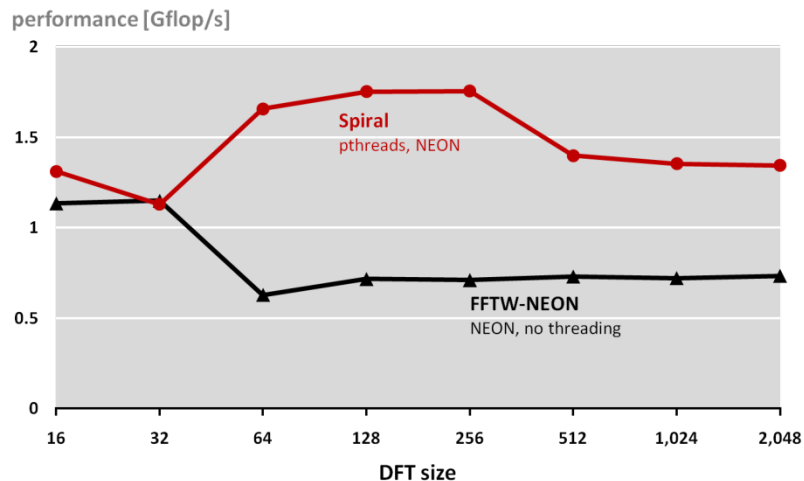
performance [Gflop/s]



From Cell Phone To Supercomputer

DFT on Samsung Galaxy S II

Dual-core 1.2 GHz Cortex-A9 with NEON ISA

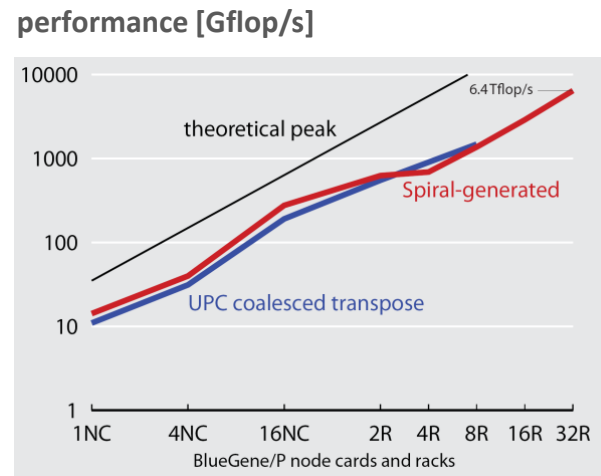


Samsung i9100 Galaxy S II

Dual-core ARM at 1.2GHz with NEON ISA



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



6.4 Tflop/s on BlueGene/P

BlueGene/P at Argonne National Laboratory

128k cores (quad-core CPUs) at 850 MHz



F. Gygi, E. W. Draeger, M. Schulz, B. R. de Supinski, J. A. Gunnels, V. Austel, J. C. Sexton, F. Franchetti, S. Kral, C. W. Ueberhuber, J. Lorenz, "Large-Scale Electronic Structure Calculations of High-Z Metals on the BlueGene/L Platform," In Proceedings of Supercomputing, 2006. **2006 Gordon Bell Prize (Peak Performance Award).**

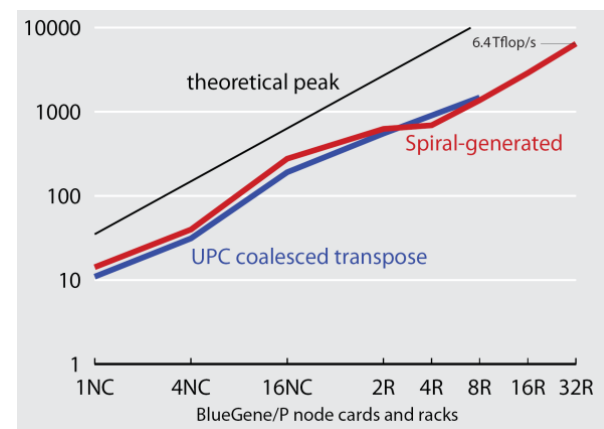
G. Almási, B. Dalton, L. L. Hu, F. Franchetti, Y. Liu, A. Sidelnik, T. Spelce, I. G. Tánase, E. Tiotto, Y. Voronenko, X. Xue, "2010 IBM HPC Challenge Class II Submission," **2010 HPC Challenge Class II Award (Most Productive System).**

SPIRAL: Success in HPC/Supercomputing

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



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



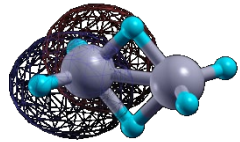
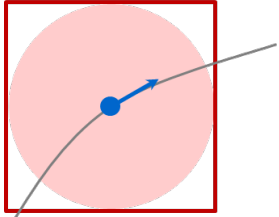
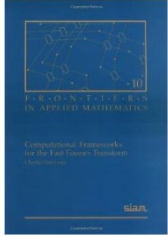
BlueGene/P at Argonne National Laboratory
128k cores (quad-core CPUs) at 850 MHz

2006 Gordon Bell Prize (Peak Performance Award) with LLNL and IBM

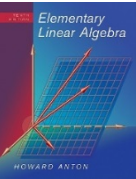
2010 HPC Challenge Class II Award (Most Productive System) with ANL and IBM

SPIRAL: 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 8.1.0: Available Under Open Source

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

www.spiral.net

```

Spiral

http://www.spiralgen.com
Spiral 8.0.0

...
PID: 17108

spiral> t := DFT(8);
DFT(8, 1)
spiral> rt := RandomRuleTree(t, SpiralDefaults);
DFT HW CT( DFT(8, 1),
  DFT_CT( DFT(4, 1),
    DFT_Base( DFT(2, 1) ),
    DFT_Base( DFT(2, 1) ),
    DFT_Base( DFT(2, 1) ) )
spiral> PrintCode("dft8", CodeRuleTree(rt, Spiral
  SpiralDefaults
  SpiralVersion
PrintCode("dft8", CodeRuleTree(rt, SpiralDefaults), SpiralDefaults);

void dft8(double *Y, double *X) {
  double a49, a50, a51, a52, s13, s14, s15, s16
    , t149, t150, t151, t152, t153, t154, t155, t156
    , t157, t158, t159, t160, t161, t162, t163, t164
    , t165, t166, t167, t168, t169, t170, t171, t172
    , t173, t174, t175, t176;
  t149 = *(X) + (*(X + 8));
  t150 = (*(X + 1)) + (*(X + 9));
  t151 = *(X) - (*(X + 8));
  t152 = (*(X + 1)) - (*(X + 9));
  t153 = (*(X + 2)) + (*(X + 10));

```



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*