

# SPIRAL: AI for High Performance Code

*with a side of* **FFTX**

**Franz Franchetti**

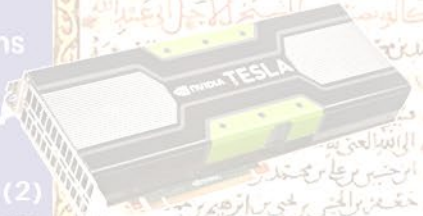
Department of Electrical and Computer Engineering  
Carnegie Mellon University

[www.ece.cmu.edu/~franzf](http://www.ece.cmu.edu/~franzf)

**Joint work with the SPIRAL team at CMU and FFTX 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



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



```
sd(s5678, s5679, (1) | (3))
sd(s5680, s5681)
```

```
sd(s5676, s5682)
sd(s5677, s5683)
```

```
sd(s5670, s5675)
sd(s5671, s5676)
```

```
sd(s5674, s5679)
sd(s5675, s5680)
```

```
sd(s5677, s5683)
sd(s5676, s5682)
```

```
sd(s5670, s5675)
sd(s5671, s5676)
```

```
sd(s5674, s5679)
sd(s5675, s5680)
```

```
sd(s5677, s5683)
sd(s5676, s5682)
```

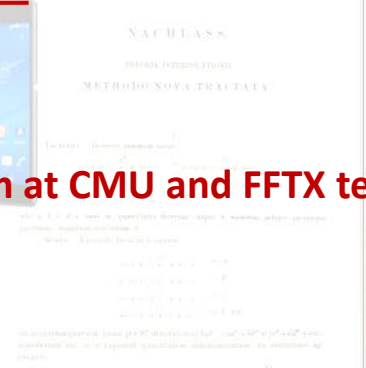
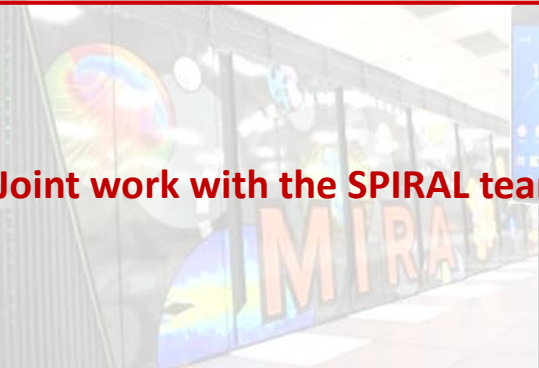
```
sd(s5670, s5675)
sd(s5671, s5676)
```

```
sd(s5674, s5679)
sd(s5675, s5680)
```

```
sd(s5677, s5683)
sd(s5676, s5682)
```

```
sd(s5670, s5675)
sd(s5671, s5676)
```

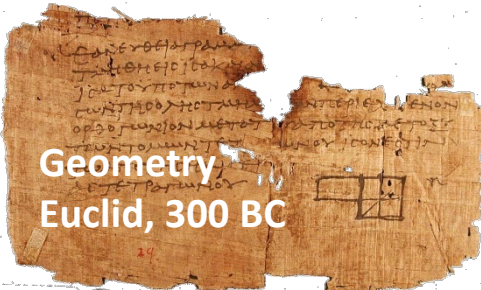
Intel  
Integrated  
Performance  
Primitives



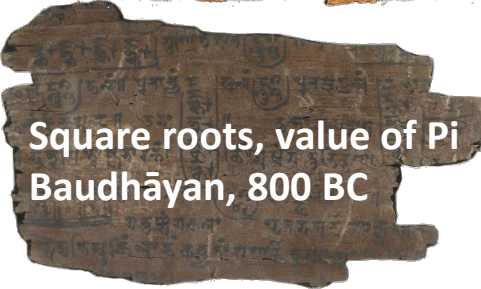
```
sd(s5670, s5675)
sd(s5671, s5676)
sd(s5674, s5679)
sd(s5675, s5680)
sd(s5677, s5683)
sd(s5676, s5682)
```

Computational  
Primitives

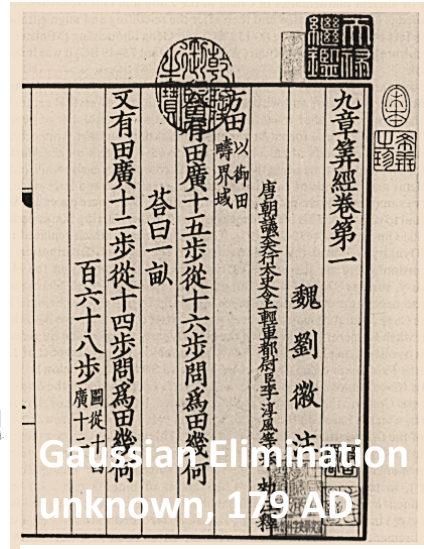
# Algorithms and Mathematics: 2,500+ Years



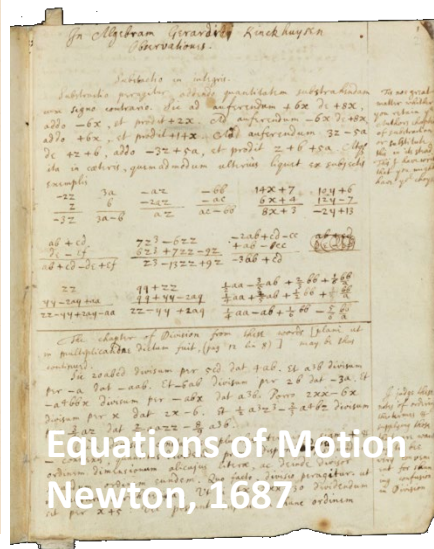
Geometry  
Euclid, 300 BC



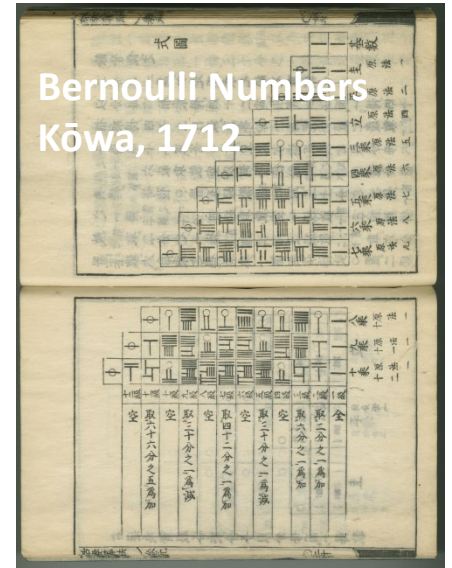
Square roots, value of Pi  
Baudhāyan, 800 BC



Gaussian Elimination  
unknown, 179 AD



Equations of Motion  
Newton, 1687

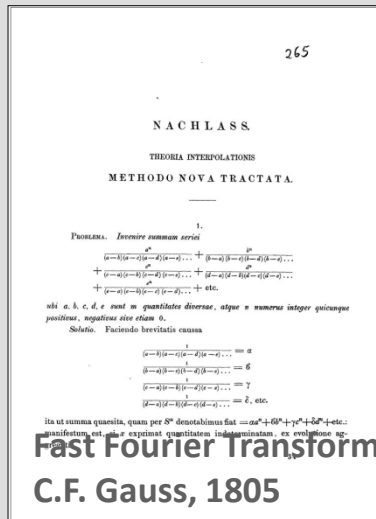


Bernoulli Numbers  
Kōwa, 1712

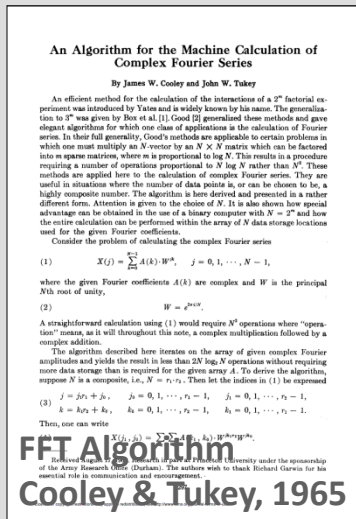


Algebra  
al-Khwārizmī, 830

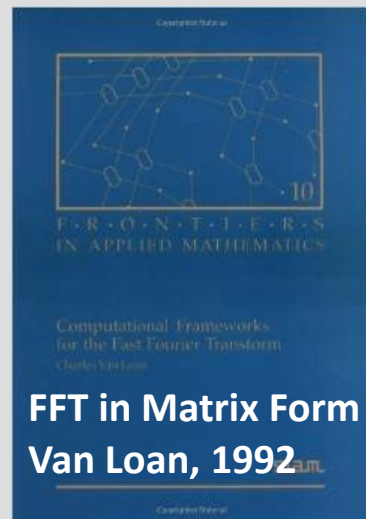
## Fast Fourier Transform



Fast Fourier Transform  
C.F. Gauss, 1805



FFT Algorithm  
Cooley & Tukey, 1965



FFT in Matrix Form  
Van Loan, 1992

# Computers I have Used: $10^{10}x$ Gain in 35 Years

The first computer I...

"My" first...



10 kflop/s

...programmed

Commodore VIC20  
1MHz MOS 6502  
5 kB RAM, TV  
1985



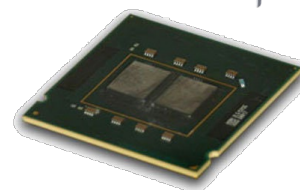
...owned

IBM PC/XT compatible  
8088 @ 8 MHz, 640kB RAM  
360 kB FDD, 720x348 mono,  
1989



...telnet'ed into

IBM RS/6000-390  
256 MB RAM, 6GB HDD  
67 MHz Power2+, AIX  
1994



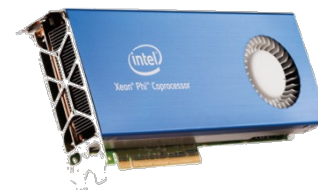
...multicore

Pentium D  
2 cores, 3.6 GHz  
2/4-way SIMD  
2005



...GPGPU

GeForce 8800  
1.3 GHz, 128 shaders  
16-way SIMT  
2006



...manycore

Xeon Phi  
1.3 GHz, 60 cores  
8/16-way SIMD  
2011

Computers I use, circa 2020

110 Tflop/s FP16 (ML)



Summit

2,282,544 cores @ 3.07 GHz  
200 Pflop/s, #1 in Top500



Dell Power Edge

80 cores @ 3GHz  
3 TB RAM



Dell Precision 3620

3.7 GHz Xeon Quad-core  
Nvidia TITAN V, 64 GB RAM



Lenovo X270

2.8 GHz Core i7 Dual-core  
Mobile GPU, 16GB RAM



Sony Xperia XZ1

2.5 GHz Octa-core  
Mobile GPU, 4GB RAM

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

# Computing Platforms Over The Years

## F-16A/B, C/D, E/F, IN, IQ, N, V



## B52 Stratofortress



## Compare: Desktop/workstation class CPUs/machines

Assembly code compatible !!



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

**2020**

Xeon Platinum 8380HL  
28 cores, 2.9-4.3 GHz  
2/4/8/16-way SIMD

**x86 ISA: hiding  $10^8$ x compounded performance gain over half a century**

# 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
- 1995: Java
- 2000: C#
- 2012: Julia

Will I adopt something new?

**NOPE**



# 2020: What \$1M Can Buy You



**Dell PowerEdge R940**  
*4.5 Tflop/s, 6 TB, 850 W*  
 4x 24 cores, 2.5 GHz



**24U rack**  
**10kW**  
**<\$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



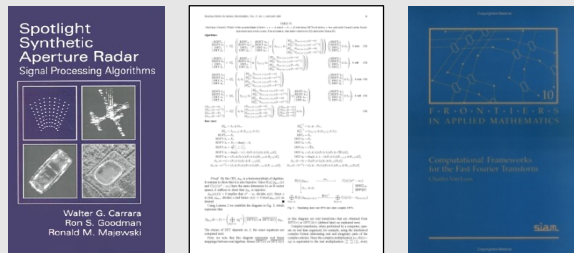
**AberSAN ZXP4**  
*90x 18TB HDD, 1 kW*  
 1.6PB raw



**Nvidia DGX-A100**  
*8x Tesla A100, 6.5kW*  
 5 Pflop/s, 320 GB

# SPIRAL: AI for High Performance Code

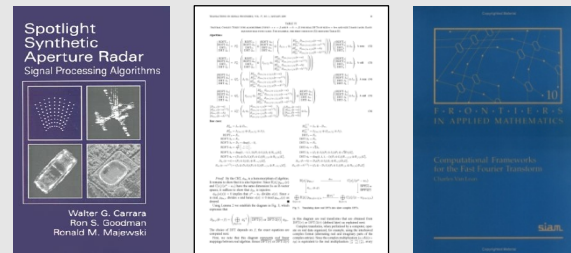
## Traditionally



High performance library  
optimized for given platform

*Comparable  
performance*

## Spiral Approach



High performance library  
optimized for given platform

# Outline

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



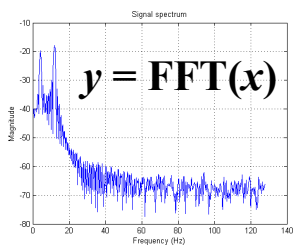
# SPIRAL: AI for Performance Engineering

## Given:

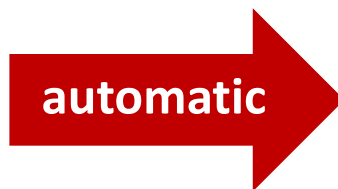
- Mathematical problem specification  
*core mathematics does not change*
- Target computer platform  
*varies greatly, new platforms introduced often*

## 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

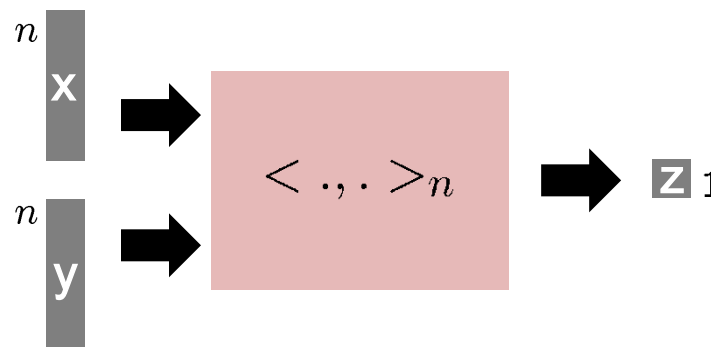
- **Operator: Multiple vectors ! Multiple vectors**
- **Stateless**
- **Higher-dimensional data is linearized**
- **Operators are potentially nonlinear**

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

## 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

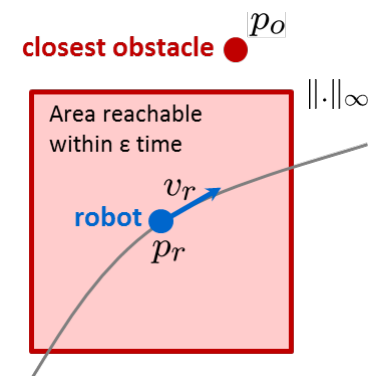
## ■ Passive Safety of Robots

$p_o$ : Position of closest obstacle

$p_r$ : Position of robot

$v_r$ : Longitudinal velocity of robot

$A, b, V, \epsilon$ : 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

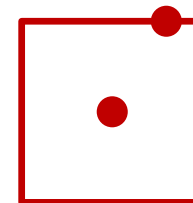
$$\|\cdot\|_{\infty}^n : \mathbb{R}^n \rightarrow \mathbb{R}$$

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

## ■ 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

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

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

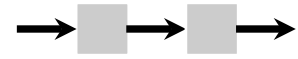
*Beyond the textbook: explicit vector length, infix operators as prefix operators*

# Operations and Operator Expressions

## ■ Operations (higher-order operators)

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

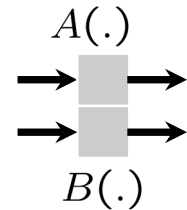
$$(A, B) \mapsto B \circ A$$



$A(\cdot)$   $B(\cdot)$

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

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

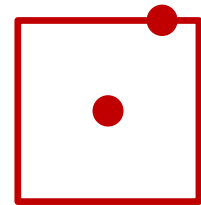


$B(\cdot)$

## ■ 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

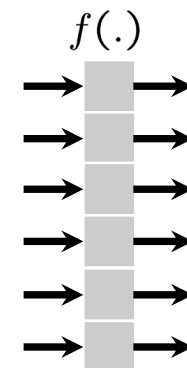
$$A(\cdot) - B(\cdot) = \left( x \mapsto A(x) - B(x) \right)$$

can be expressed via  $(-)_n : \mathbb{R}^n \times \mathbb{R}^n \rightarrow \mathbb{R}^n$

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

# 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**

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

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

**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

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

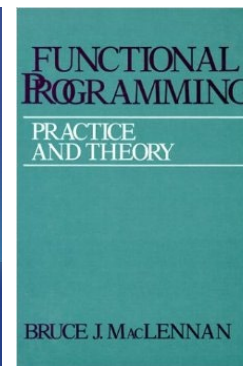
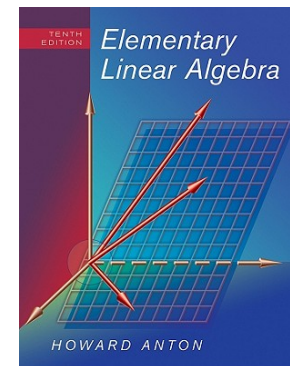
$$(\diamond)_n \rightarrow \text{Pointwise}_{n \times n, (a,b) \mapsto a \diamond b}, \quad \diamond \in \{+, -, \cdot, \wedge, \vee, \dots\}$$

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

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

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

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



*Library of well-known identities expressed in OL*

# 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

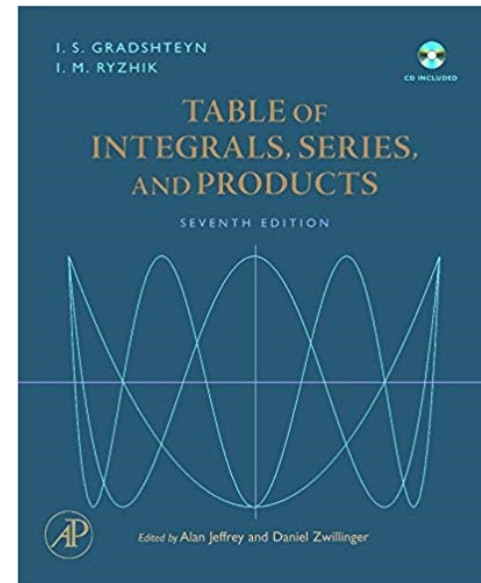
## Table of Integrals

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

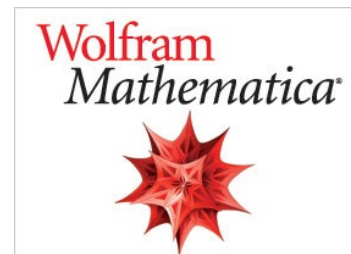


$$\text{In[31]:- } \int_0^{2\pi} \frac{1}{a^2 \cos[t]^2 + b^2 \sin[t]^2} dt$$

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

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

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





# $\Sigma$ -OL: Low-Level Operator Language

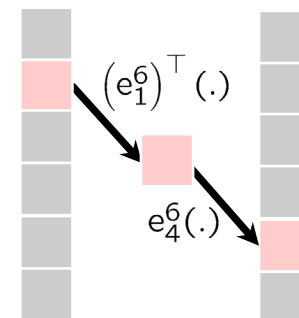
## ■ Selection and embedding operator: *gather and scatter*

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

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

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

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

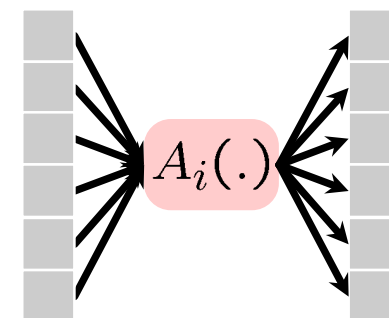


## ■ Iterative operations: *loop*

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

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

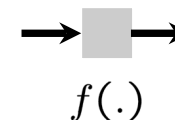
$$\text{with } \sqcup \in \{ \Sigma, \vee, \wedge, \Pi, \min, \max, \dots \}$$



## ■ Atomic operators: *nonlinear scalar functions*

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

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



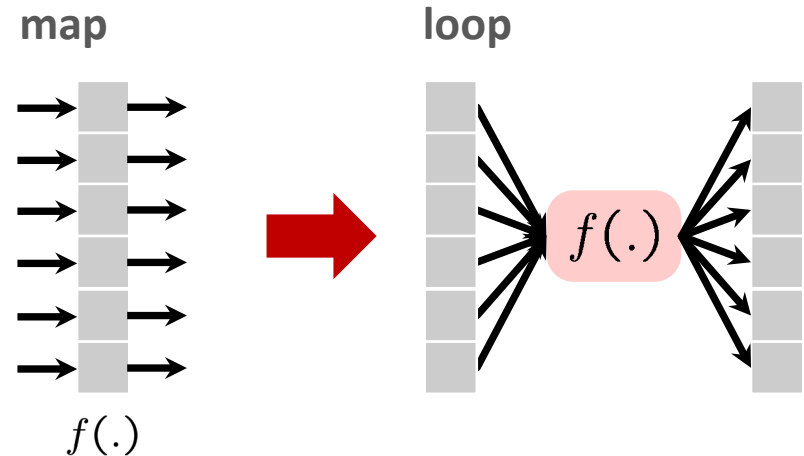
**$\Sigma$ -OL operator expressions = array-based programs with for loops**

# Rule-Based Translation and Optimization

## ■ Translating Basic OL into $\Sigma$ -OL

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

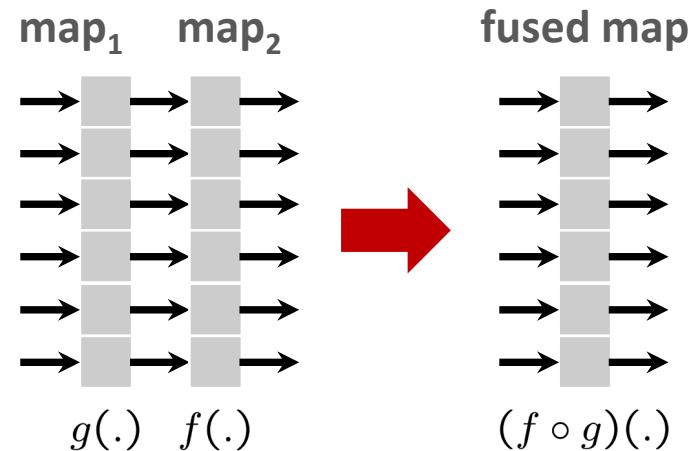
$$\text{Reduction}_{n,(a,b) \mapsto a+b} \rightarrow \sum_{i=0}^{n-1} (e_i^n)^\top$$



## ■ Optimizing Basic OL/ $\Sigma$ -OL

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

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



***Captures program optimizations that are traditionally hard to do***

# Last Step: Abstract Code

## Code objects

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

## Properties: at the same time

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

```
# Dynamic Window Monitor

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

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

# Translating $\Sigma$ -OL to Abstract Code

## Compilation rules: recursive descent

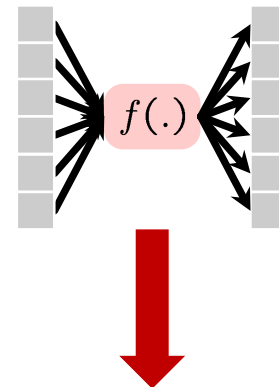
$$\text{Code}(y = (A \circ B)(x)) \rightarrow \{\text{decl}(t), \text{Code}(t = B(x)), \text{Code}(y = A(t))\}$$

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

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

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

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



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

## Cleanup rules: term rewriting

`chain(a, chain(b))` → `chain([a, b])`

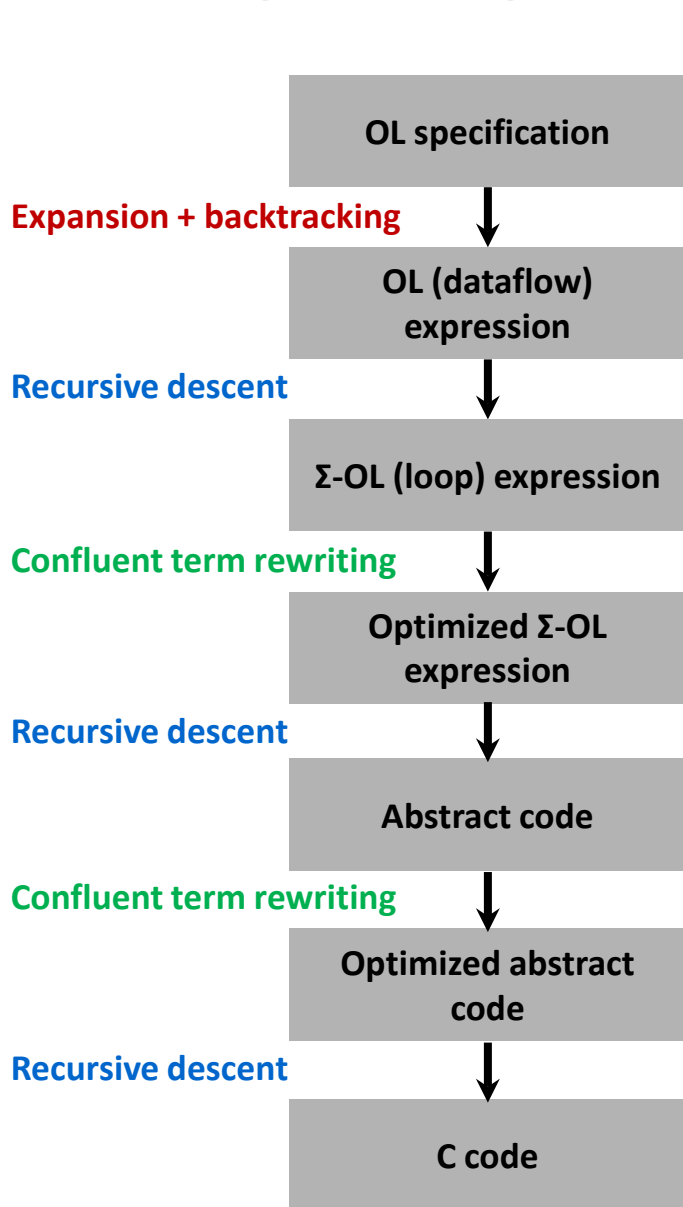
`decl(D, decl(E, c))` → `decl([D, E], c)`

`loop(i, decl(D, c))` → `decl(D, loop(i, c))`

`chain(a, decl(D, b))` → `decl(D, chain([a, b]))`

***Rule-based code generation and backend compilation***

# 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), u5));
    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_loaddb_pd(x1, x10));
        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);
    }
}
  
```

# Final Synthesized C Code

```

int dwmonitor(float *X, double *D) {
  __m128d u1, u2, u3, u4, u5, u6, u7, u8, 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;
}

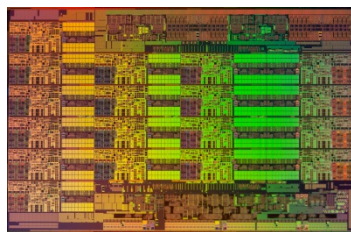
```

# Outline

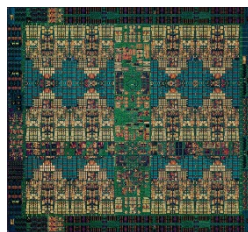
- Introduction
- Operator Language
- **Achieving Performance Portability**
- FFTX: A Library Frontend for SPIRAL
- Summary

# Today's Computing Landscape

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



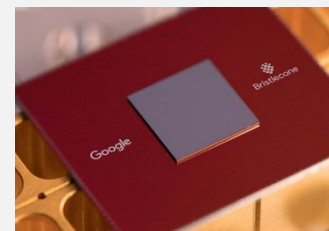
**Intel Xeon 8380HL**  
*2.5 Tflop/s, 205 W*  
 28 cores, 2.9—4.3 GHz  
 2-way—16-way AVX-512



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



**Nvidia Tesla A100**  
*9.7/19.5 Tflop/s, 400 W*  
 6912 cores, 1.4 GHz  
 32-way SIMT, tensor cores



**Google Bristlecone**  
*72 qubits*



**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

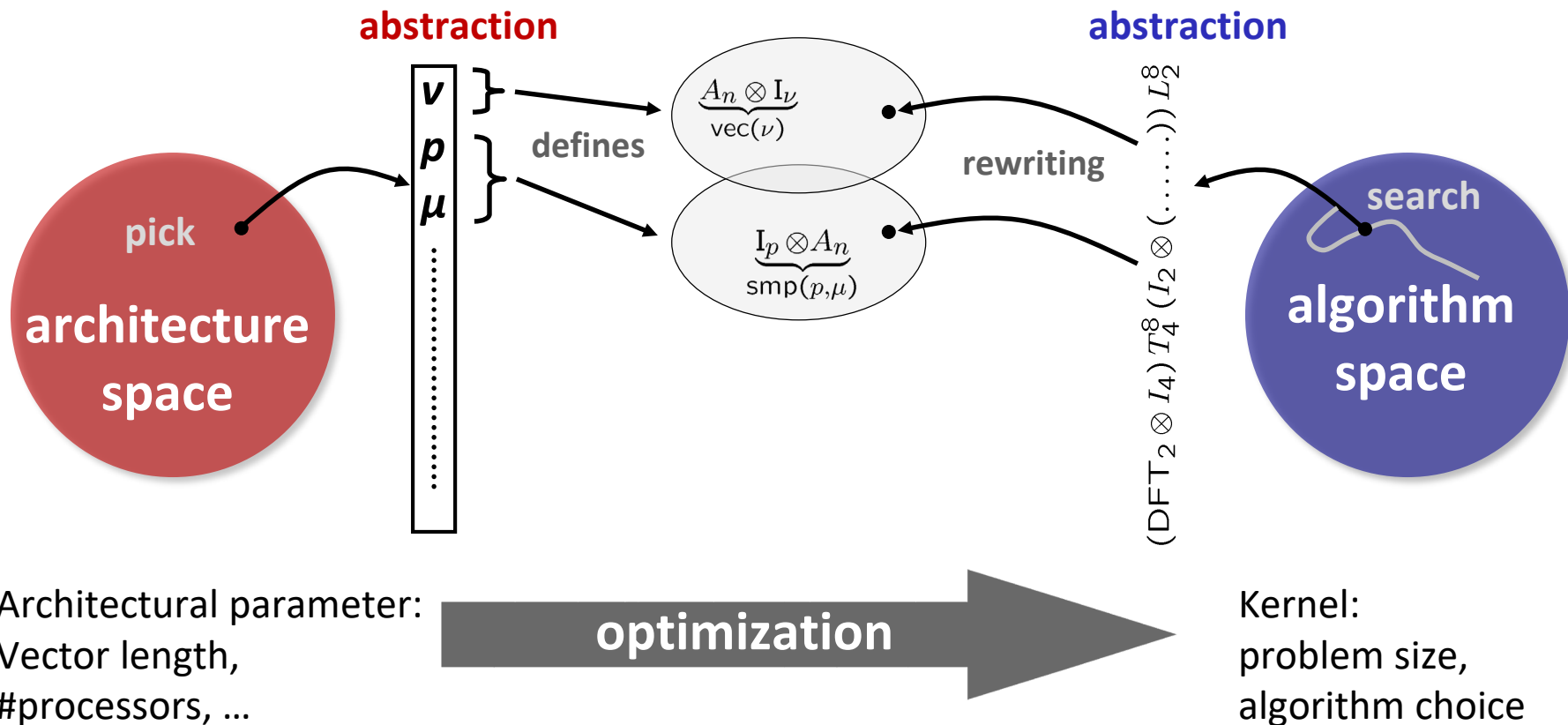


**Summit**  
*187.7 Pflop/s, 13 MW*  
 9,216 x 22 cores POWER9  
 + 27,648 V100 GPUs



# Platform-Aware Formal Program Synthesis

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



# Some Application Domains in OL

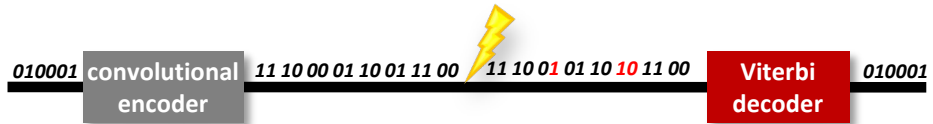
## Linear Transforms

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

## 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

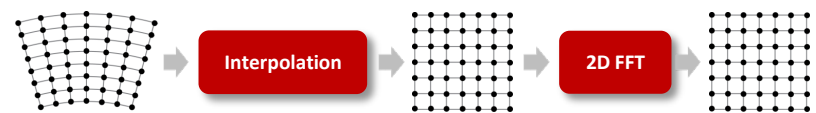


$$\mathbf{F}_{K,F} \rightarrow \prod_{i=1}^F \left( (\text{I}_{2^{K-2}} \otimes_j B_{F-i,j}) \text{L}_{2^{K-2}}^{2^{K-1}} \right)$$

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

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

## 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)

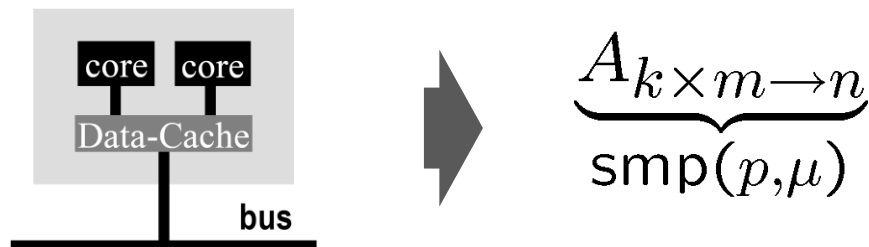
$$\prod_{i=0}^{n-1} A_i^{ir}, \quad I_s \tilde{\otimes} A, \quad \underbrace{L_n^m}_{\text{bram}}$$

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

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

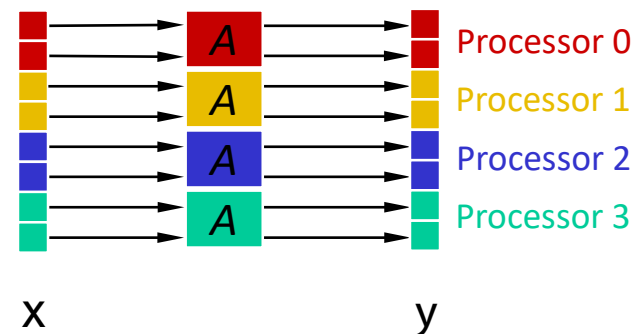
# Modeling Hardware: Base Cases

- Hardware abstraction: shared cache with cache lines



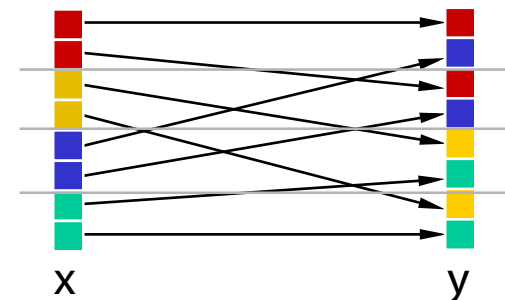
- Tensor product: embarrassingly parallel operator

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



- 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

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

$\text{DFT}_8$

**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**

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

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

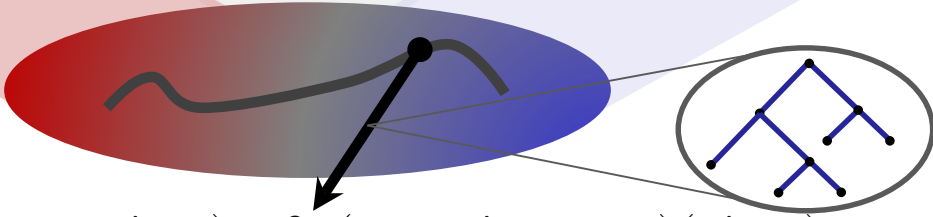
$A^{m \times m} \otimes \text{I}_n \rightarrow (A^{m \times m} \otimes \text{I}_{n/\nu}) \otimes \text{I}_{\nu}$

**Breakdown rules**

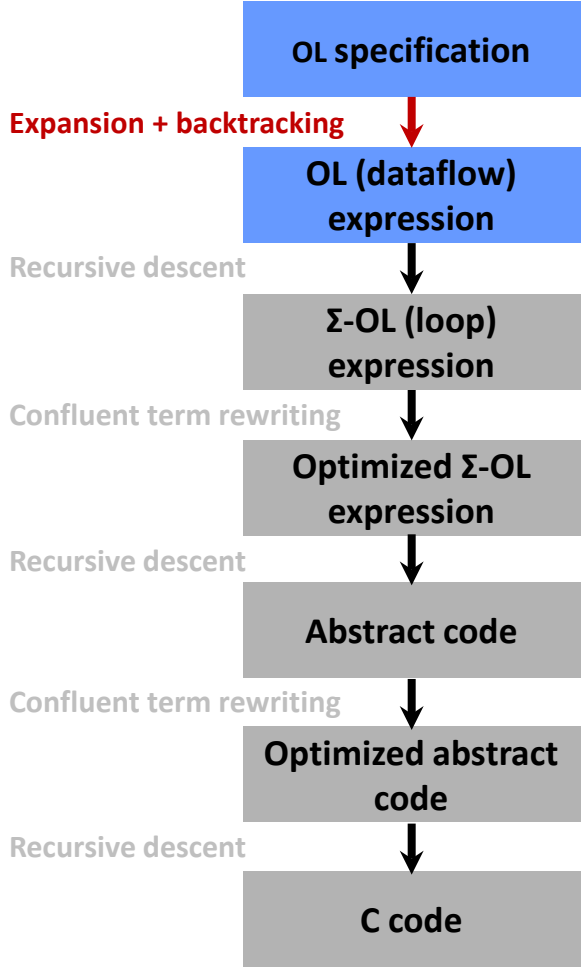
$\text{DFT}_{mn} \rightarrow (\text{DFT}_m \otimes \text{I}_n) \text{T}_n^{mn}$

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

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



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



# Translating an OL Expression Into Code

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

Output =

Ruletree, expanded into

**OL Expression:**

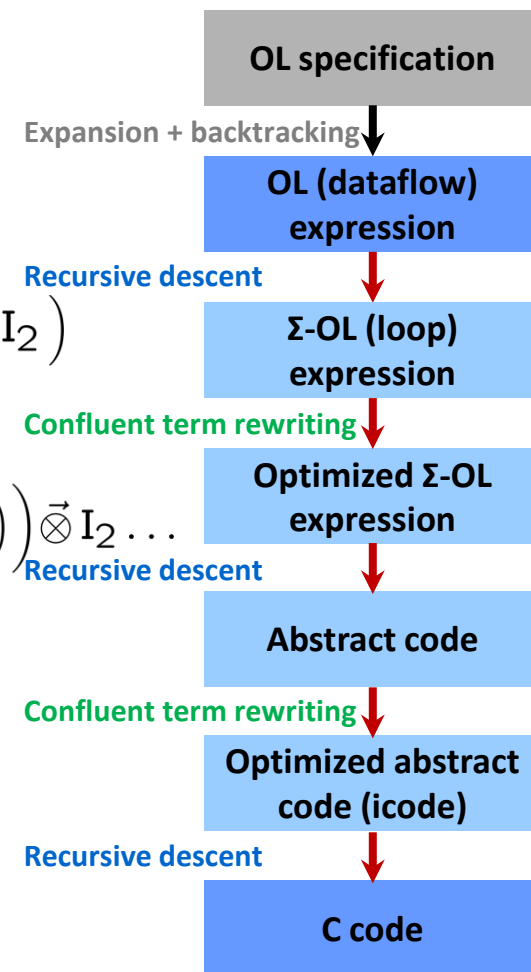
$$\left( (F_2 \otimes I_2) T_2^4 (I_2 \otimes F_2) L_2^4 \vec{\otimes} I_2 \right) \underbrace{T_2^8}_{\text{vec}(2)} \left( I_2 \otimes \underbrace{L_2^4}_{\text{vec}(2)} (F_2 \vec{\otimes} I_2) \right) \left( L_2^4 \vec{\otimes} I_2 \right)$$

**$\Sigma$ -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
- Operator Language
- Achieving Performance Portability
- **FFTX: A Library Frontend for SPIRAL**
- Summary

# FFTX and SpectralPACK

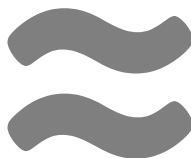
## Numerical Linear Algebra

### 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.  $\Delta$  is the Laplace operator

## Solution characterization

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

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

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

## Approach: Green's function

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

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

## Method of Local Corrections (MLC)

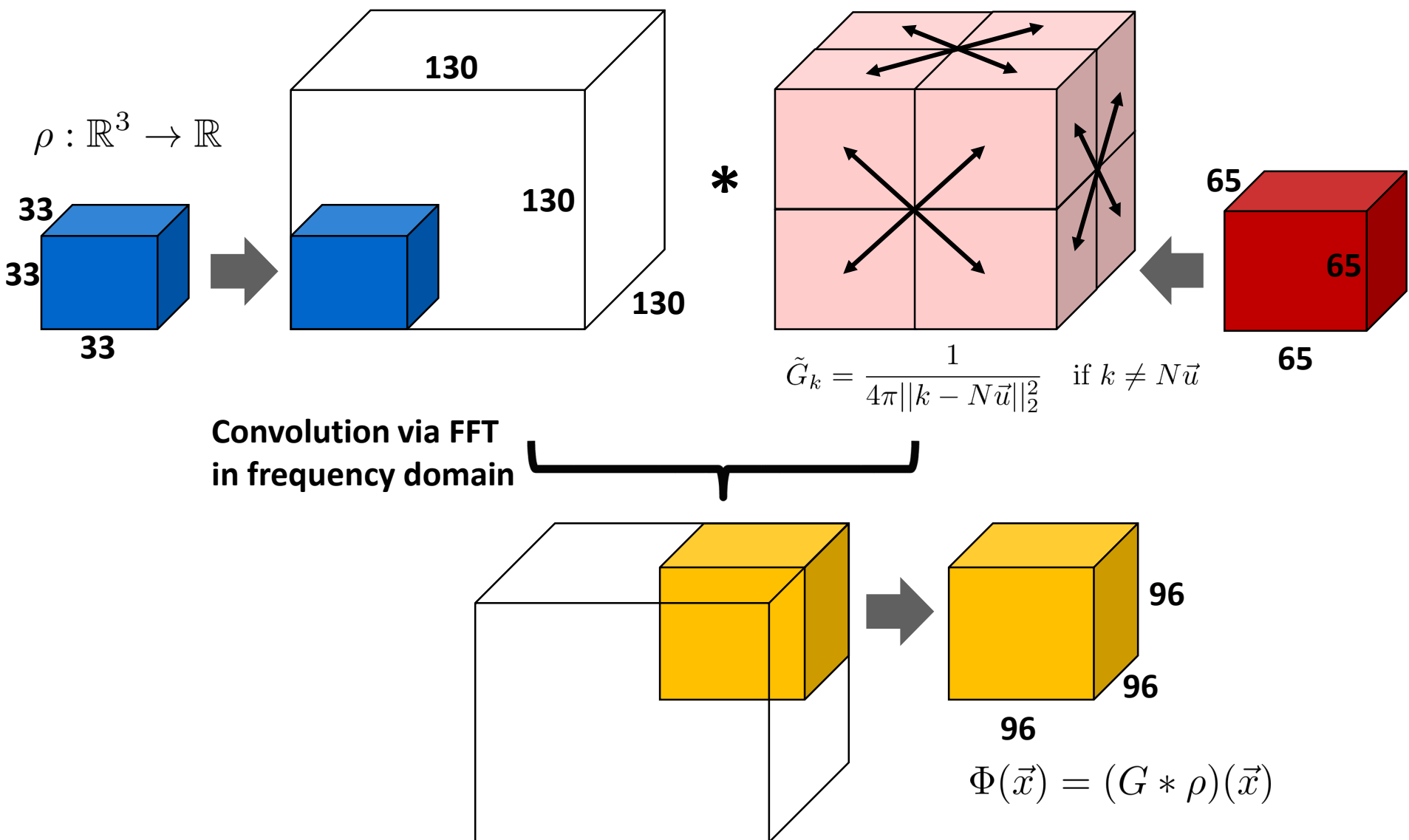
$$\tilde{G}_k = \frac{1}{4\pi\|k - N\vec{u}\|_2^2} \quad \text{if } k \neq N\vec{u}$$

Green's function kernel in frequency domain

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

```
fftx_plan pruned_real_convolution_plan(fftx_real *in, fftx_real *out, fftx_complex *symbol,
    int n, int n_in, int n_out, int n_freq) {
    int rank = 3,
    batch_rank = 0,
    ...
    fftx_plan plans[5];
    fftx_plan p;

    tmp1 = fftx_create_zero_temp_real(rank, &padded_dims);

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

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

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

    tmp4 = fftx_create_temp_real(rank, &padded_dims);
    plans[3] = fftx_plan_guru_dft_c2r(rank, &padded_dims, batch_rank,
        &batch_dims, tmp3, tmp4, MY_FFTX_MODE_SUB);

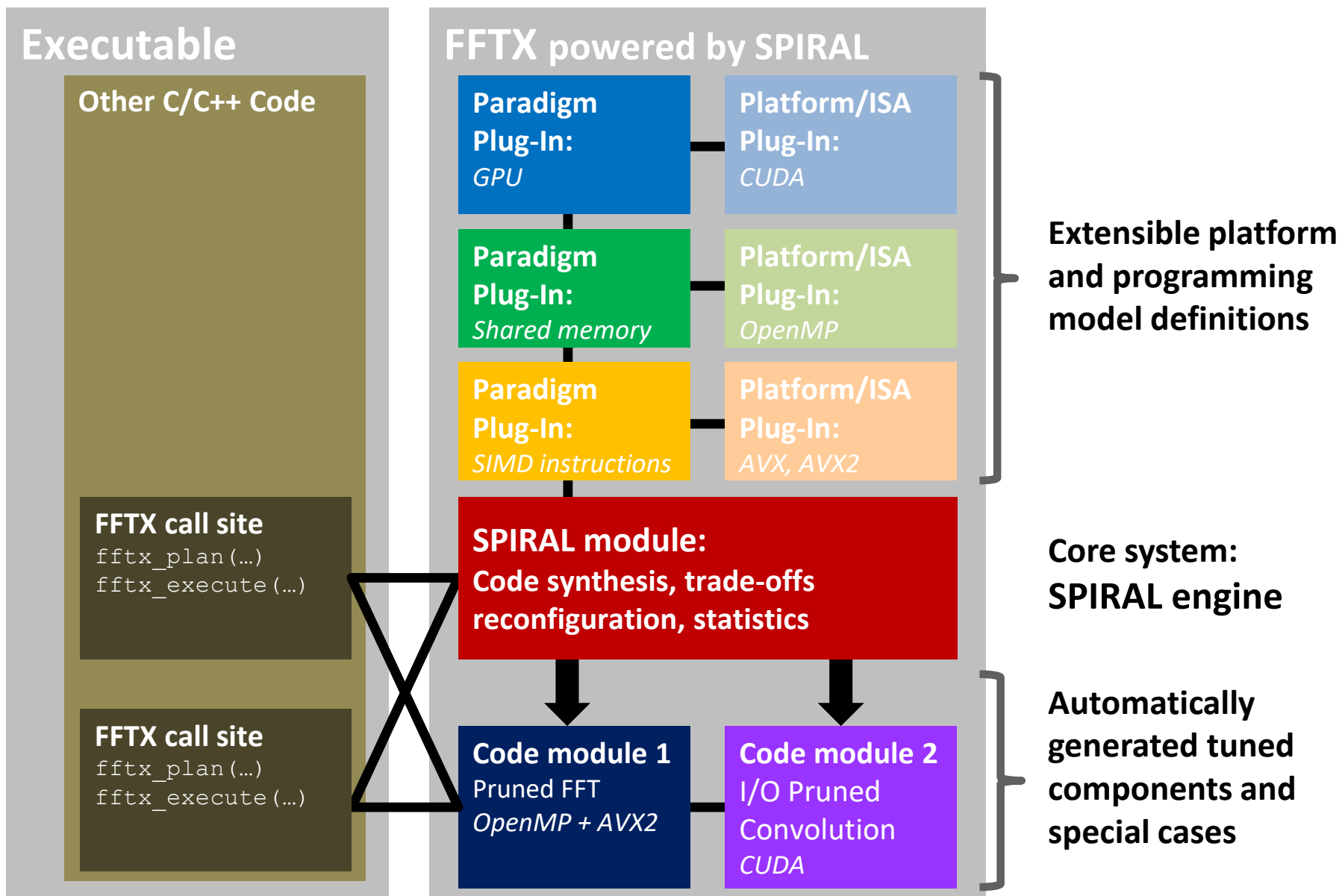
    plans[4] = fftx_plan_guru_copy_real(rank, &out_dimx, tmp4, out, MY_FFTX_MODE_SUB);

    p = fftx_plan_compose(numsubplans, plans, MY_FFTX_MODE_TOP);

    return p;
}
```

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

# FFTX Backend: SPIRAL

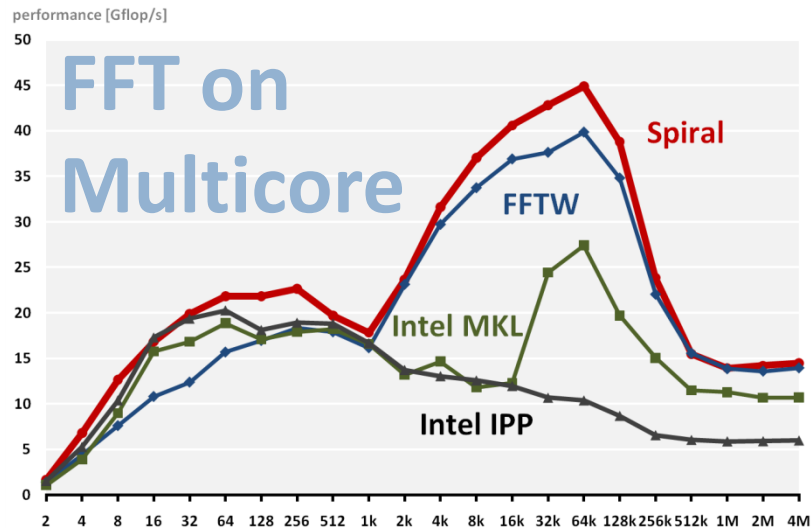


# Outline

- Introduction
- Operator Language
- Achieving Performance Portability
- FFTX: A Library Frontend for SPIRAL
- **Summary**

# Synthesis: FFTs and Spectral Algorithms

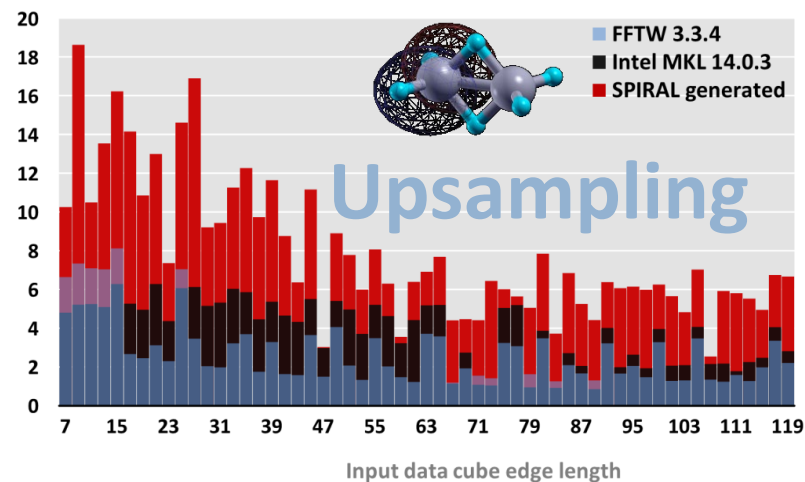
1D DFT on 3.3 GHz Sandy Bridge (4 Cores, AVX)



Performance of 2x2x2 Upsampling on Haswell

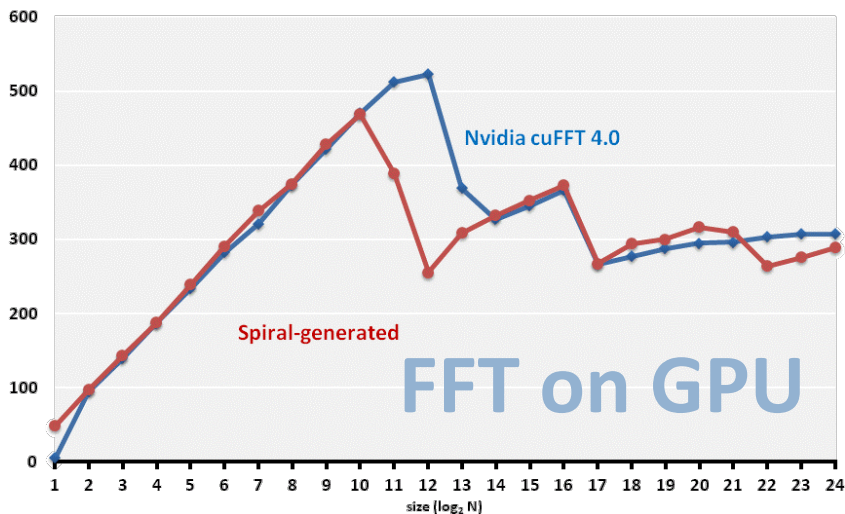
3.5 GHz, AVX, double precision, interleaved input, single core

Performance [Pseudo Gflop/s]



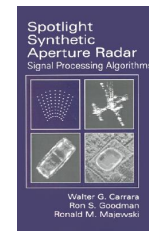
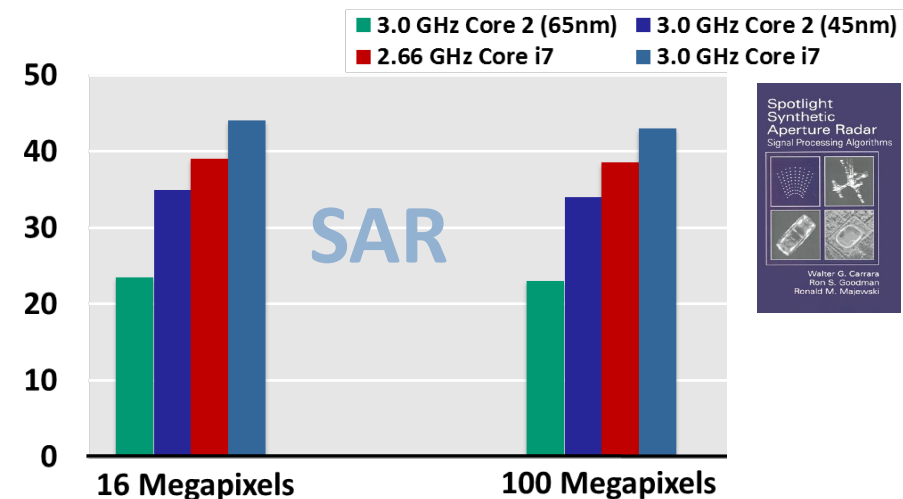
1D Batch DFT (Nvidia GTX 480)

performance [GFlop/s], single precision



PFA SAR Image Formation on Intel platforms

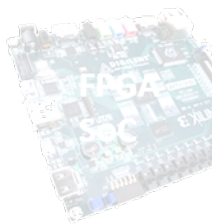
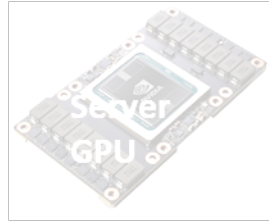
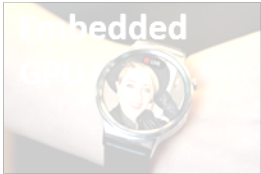
performance [Gflop/s]





# SPIRAL for Single Node/Shared Memory

Architecture



**Dynamic range:**

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



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



2000: SSE2



2007: SSSE3



2011: AVX



2013: AVX2

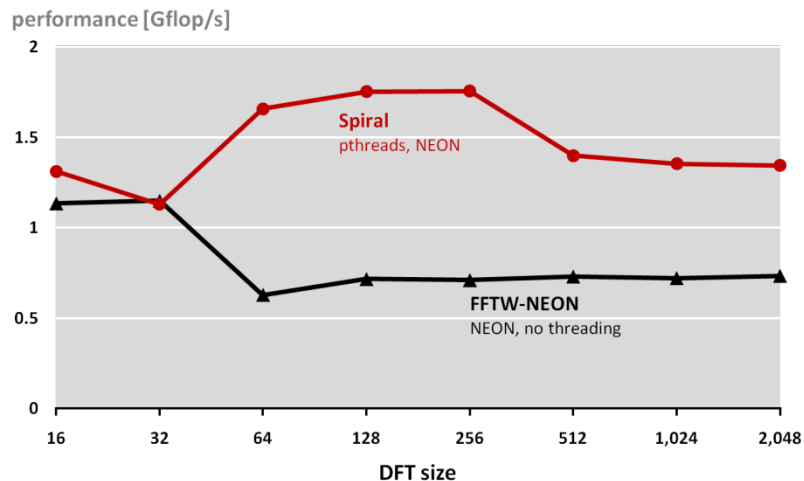


2019: AVX512

# From Cell Phone To Supercomputer

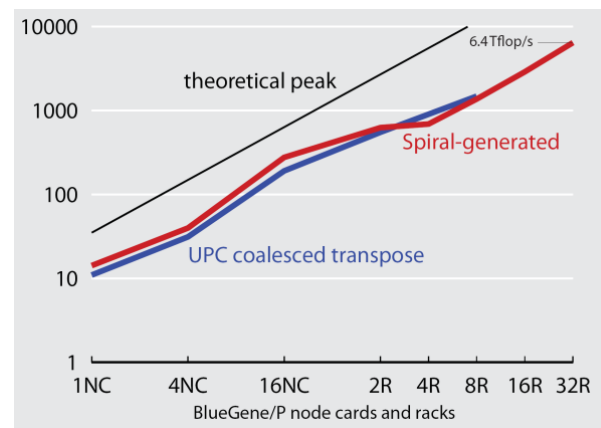
## DFT on Samsung Galaxy S II

Dual-core 1.2 GHz Cortex-A9 with NEON ISA



## Global FFT (1D FFT, HPC Challenge)

performance [Gflop/s]



**6.4 Tflop/s on BlueGene/P**

## Samsung i9100 Galaxy S II

Dual-core ARM at 1.2GHz with NEON ISA



## BlueGene/P at Argonne National Laboratory

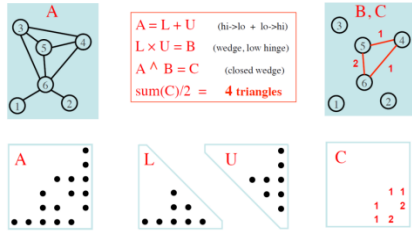
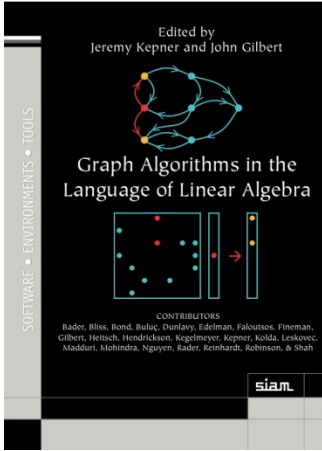
128k cores (quad-core CPUs) at 850 MHz

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).**

# Current Research Directions

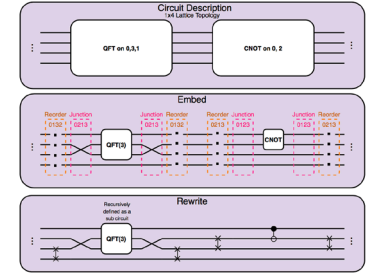
## SPIRAL for Graphs



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

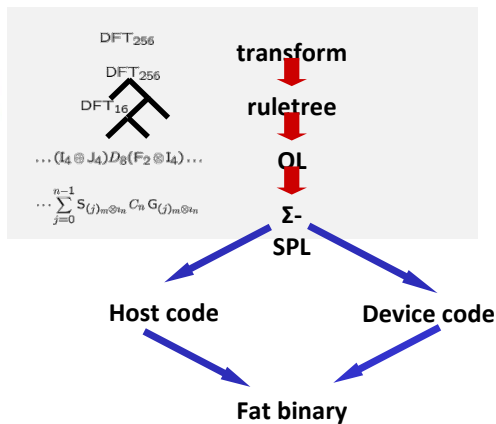
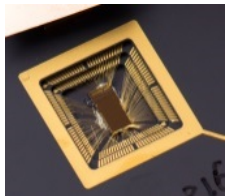
$$\Delta = \Delta + \frac{1}{2} \alpha_{10} A_{00} \alpha_{01}$$

## Spiral for Quantum Computing

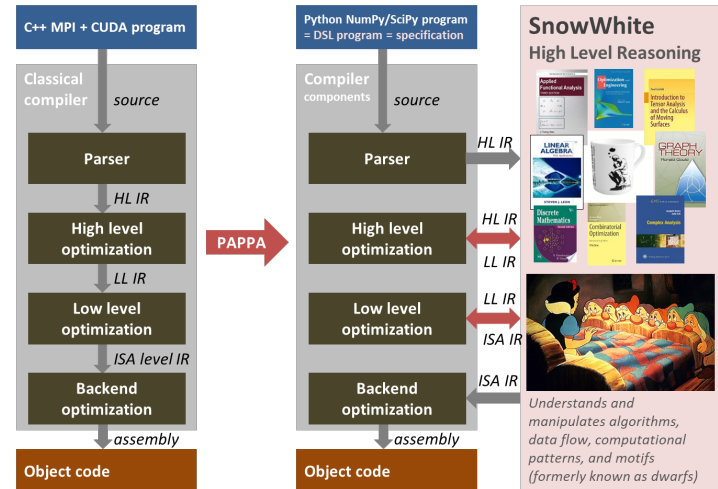


```
[[0,1,0], [1,1,0], [0,1,1], [0,1,0]]
Apply a 3-qubit Fourier transform to qubits {0,1,2}
qCirc(arch, [ ([0,3,1], qFT ), ([0,2], qCNOT) ]);
qEmbed([0,3,1], arch, qFT) * qEmbed([0,2], arch, qCNOT)
Reord([0,1,3,2], arch, F)*Junc([0,2,1,3], F)*Tensor(qFT(3), I(2))*Junc([0,2,1,3], B)*Reord([0,1,3,2], arch, B)
Swap([3,2], 4)
Tensor(I(4), (CNOT(0->1)*CNOT(1->0)*CNOT(0->1)))
Tensor(I(4), (CNOT(1->0)*CNOT(0->1)*CNOT(1->0)))
qCirc(subarch, [ ([0], qHT), ... ])
Tensor(I(4), (CNOT(0->1)*CNOT(1->0)*CNOT(0->1)))
Tensor(I(4), (CNOT(1->0)*CNOT(0->1)*CNOT(1->0)))
```

## HW/SW Co-Design

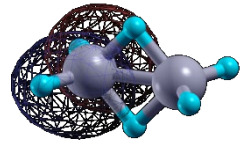
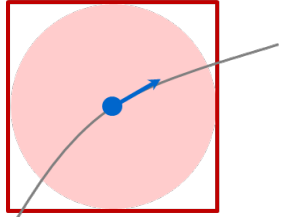
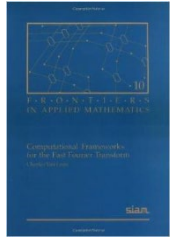


## SPIRAL as AI in Compilers



# SPIRAL: AI for High Performance Code

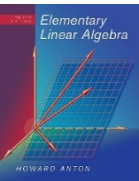
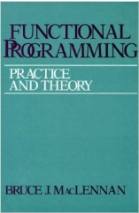
## Algorithms



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



## Correctness



## Hardware



# SPIRAL 8.2.0: 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](http://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*