SPIRAL: AI for High Performance Code

with a side of **FFTX**

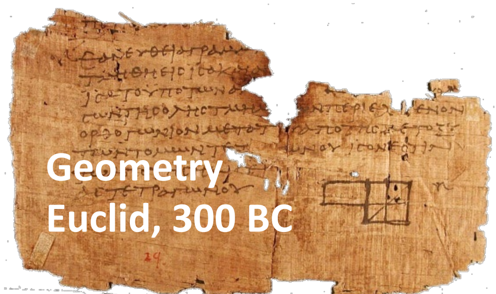
Franz Franchetti

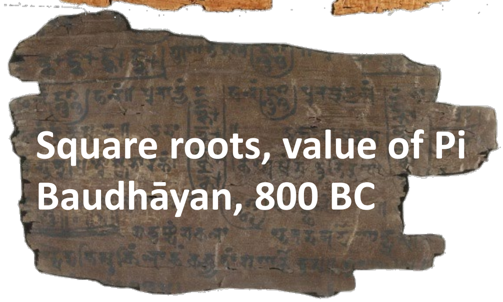
Department of Electrical and Computer Engineering
Carnegie Mellon University
www.ece.cmu.edu/~franzf

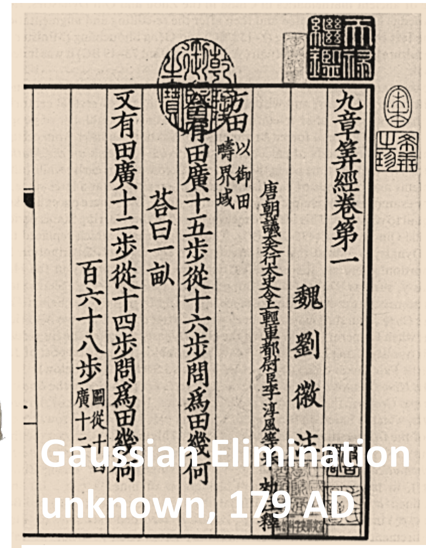
Joint work with the SPIRAL team at CMU and FFTX team at CMU and LBL

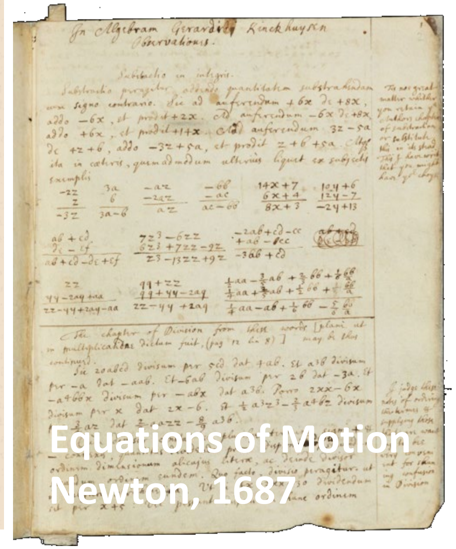
This work was supported by DARPA, DOE, ONR, NSF, Intel, Mercury, and Nvidia

Algorithms and Mathematics: 2,500+ Years

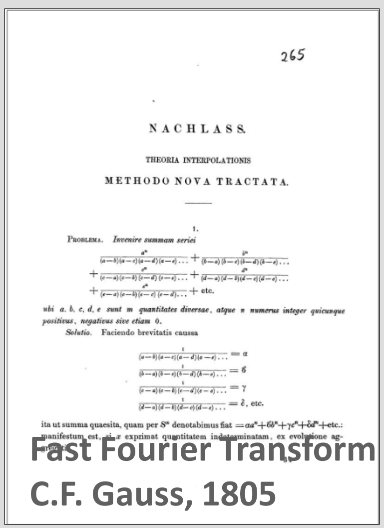


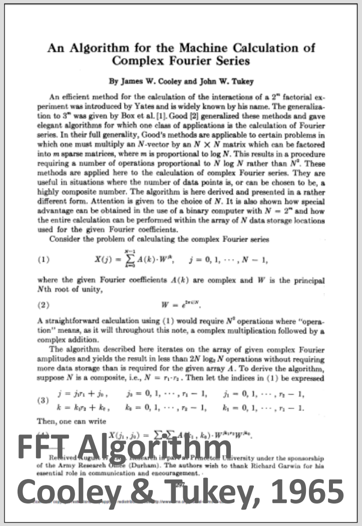


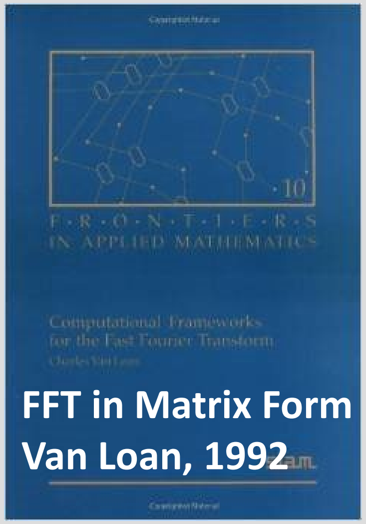




Fast Fourier Transform





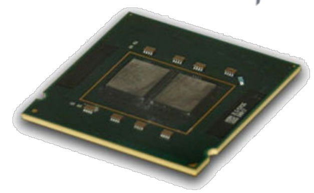


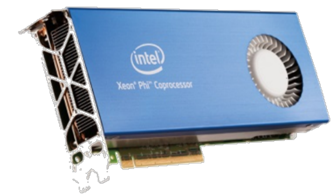
Computers I have Used: 10^{10}x Gain in 35 Years

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	1989	1994	2006	2011	2020
Intel 8008	IBM PC/XT compatible	IBM RS/6000-390	GeForce 8800	Xeon Phi	Xeon Platinum 8380HL
0.2—0.8 MHz	8088 @ 8 MHz, 640kB RAM	256 MB RAM, 6GB HDD	1.3 GHz, 128 shaders	1.3 GHz, 60 cores	28 cores, 2.9-4.3 GHz
Intelligent terminal	360 kB FDD, 720x348 mono	67 MHz Power2+, AIX	16-way SIMT	8/16-way SIMD	2/4/8/16-way SIMD

x86 ISA: hiding 10^8x 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

2020: What $1M Can Buy You

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

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

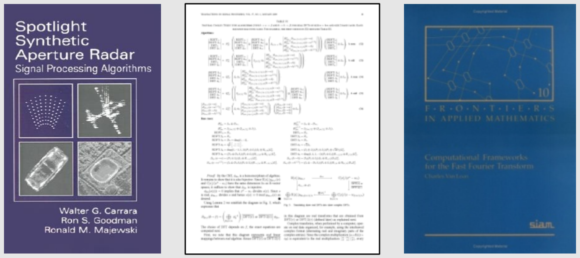
BittWare TeraBox
18M logic elements, 4.9 Tb/sec I/O
8 FPGA cards/16 FPGAs, 2 TB DDR4

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

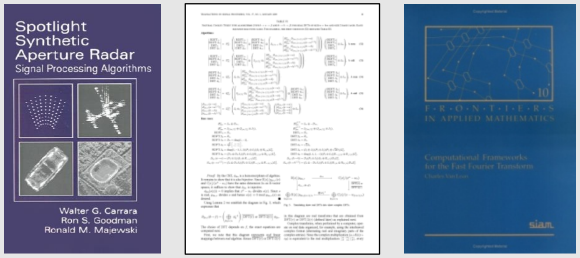
Traditionally



High performance library optimized for given platform

Comparable performance

Spiral Approach



Spiral

High performance library optimized for given platform

Outline

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

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

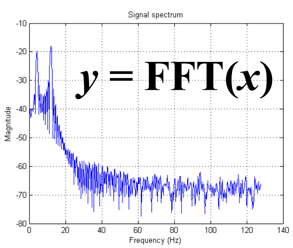
SPIRAL: AI for Performance Engineering

Given:

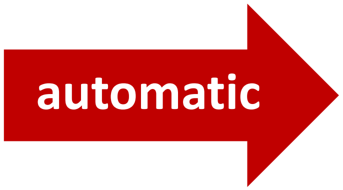
- Mathematical problem specification
  core mathematics does not change
- Target computer platform
  varies greatly, new platforms introduced often

Wanted:

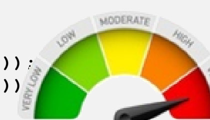
- Very good implementation of specification on platform
- Proof of correctness



on




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



+

QED.

OL Operators

Definition

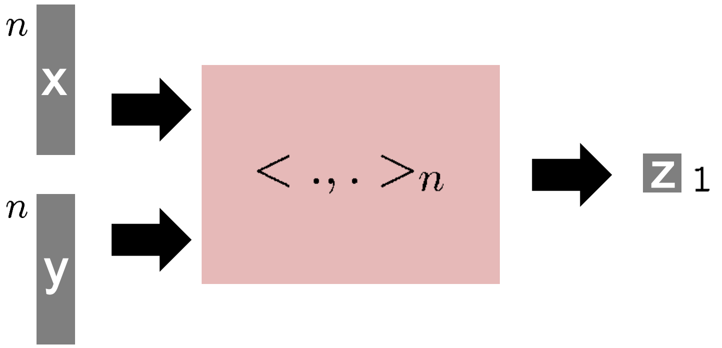
- Operator: Multiple vectors ! Multiple vectors
- Stateless
- Higher-dimensional data is linearized
- Operators are potentially nonlinear

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

Example: Scalar product

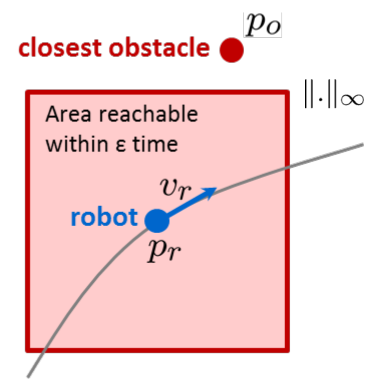
$$< .,. >_n : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$$

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



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, " : constants



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

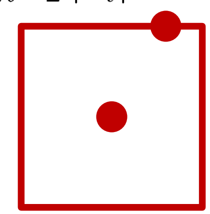
- **Definition as operator**

$$\mathsf{SafeDist}_{V,A,b,\varepsilon} : \quad \mathbb{R} \times \mathbb{R}^2 \times \mathbb{R}^2 \to \mathbb{Z}_2$$
$$(v_r, p_r, p_o) \mapsto \big(p(v_r) < d_\infty(p_r, p_o)\big) \quad \text{with} \quad d_\infty(\vec{x}, \vec{y}) = \|\vec{x} - \vec{y}\|_\infty$$

$$p(x) = \alpha x^2 + \beta x + \gamma$$
$$\alpha = \frac{1}{2b}$$
$$\beta = \frac{V}{b} + \varepsilon\left(\frac{A}{b} + 1\right)$$
$$\gamma = \left(\frac{A}{b} + 1\right)\left(\frac{A}{2}\varepsilon^2 + \varepsilon V\right)$$

Formalizing Mathematical Objects in OL

- **Infinity norm**
  $$\|.\|_\infty^n : \mathbb{R}^n \to \mathbb{R}$$
  $$(x_i)_{i=0,\ldots,n-1} \mapsto \max_{i=0,\ldots,n-1} |x_i|$$
- **Chebyshev distance**
  $$d_\infty^n(.,.) : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$$
  $$(x, y) \mapsto \|x - y\|_\infty^n$$



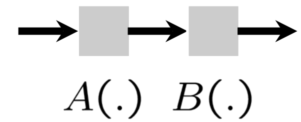
- **Vector subtraction**
  $$(-)_n : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}^n$$
  $$(x, y) \mapsto x - y$$
- **Pointwise comparison**
  $$(<)_n : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{Z}_2^n$$
  $$\big((x_i)_{i=0,\ldots,n-1}, (y_i)_{i=0,\ldots,n-1}\big) \mapsto (x_i < y_i)_{i=0,\ldots,n-1}$$
- **Scalar product**
  $$< .,. >_n : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$$
  $$\big((x_i)_{i=0,\ldots,n-1}, (y_i)_{i=0,\ldots,n-1}\big) \mapsto \sum_{i=0}^{n-1} x_i y_i$$
- **Monomial enumerator**
  $$(x^i)_n : \mathbb{R} \to \mathbb{R}^{n+1}$$
  $$x \mapsto (x^i)_{i=0,\ldots,n}$$
- **Polynomial evaluation**
  $$P[x, (a_0, \ldots, a_n)] : \mathbb{R} \to \mathbb{R}$$
  $$x \mapsto a_0 x^n + a_1 x^{n-1} + \cdots + a_{n-1} x + a_n$$

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

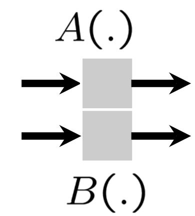
Operations and Operator Expressions

- **Operations (higher-order operators)**

$$\circ : (D \to S) \times (S \to R) \to (D \to R)$$
$$(A, B) \mapsto B \circ A$$

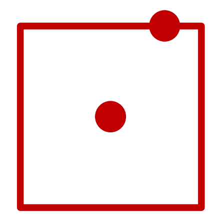


$$\times : (D \to R) \times (E \to S) \to (D \times E \to R \times S)$$
$$(A, B) \mapsto \big((x, y) \mapsto \big(A(x), B(y)\big)\big)$$



- **Operator expressions are operators**

$$\|.\|_\infty^n \circ (-)_n : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$$
$$\big((x_i)_{i=0,\dots,n-1}, (y_i)_{i=0,\dots,n-1}\big) \mapsto \max_{i=0,\dots,n-1} |x_i - y_i|$$

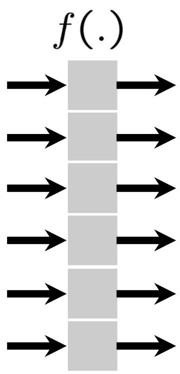


- **Short-hand notation: Infix notation**

$$A(.) - B(.) = \big(x \mapsto A(x) - B(x)\big)$$ can be expressed via $$(-)_n : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}^n$$
$$(x, y) \mapsto x - y$$

Basic OL Operators

- **Basic operators ≈ functional programming constructs**



map
$$\mathsf{Pointwise}_{n,f_i} : \mathbb{R}^n \to \mathbb{R}^n$$
$$(x_i)_i \mapsto f_0(x_0) \oplus \cdots \oplus f_{n-1}(x_{n-1})$$

binop
$$\mathsf{Atomic}_{f(.,.)} : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$$
$$(x, y) \mapsto f(x, y)$$

map + zip
$$\mathsf{Pointwise}_{n\times n,f_i} : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}^n$$
$$\big((x_i)_i, (y_i)_i\big) \mapsto f_0(x_0, y_0) \oplus \cdots \oplus f_{n-1}(x_{n-1}, y_{n-1})$$

fold
$$\mathsf{Reduction}_{n,f_i} : \mathbb{R}^n \to \mathbb{R}$$
$$(x_i)_i \mapsto f_{n-1}(x_{n-1}, f_{n-2}(x_{n-2}, f_{n-3}(\ldots f_0(x_0, \mathsf{id}()) \ldots)$$

unfold
$$\mathsf{Induction}_{n,f_i} : \mathbb{R} \to \mathbb{R}^{n+1}$$
$$x \mapsto (f_n(x, f_{n-1}(\ldots) \ldots), \ldots, f_2(x, f_1(x, \mathsf{id})), f_1(x, \mathsf{id}), \mathsf{id}())$$

- **Safety distance as (optimized) operator expression**

$$\begin{aligned}\mathsf{SafeDist}_{V,A,b,\varepsilon} = {} & \mathsf{Atomic}_{(x,y)\mapsto x<y} \\ & \circ \Big(\Big(\mathsf{Reduction}_{3,(x,y)\mapsto x+y} \circ \mathsf{Pointwise}_{3,x\mapsto a_i x} \circ \mathsf{Induction}_{3,(a,b)\mapsto ab,1}\Big) \\ & \quad \times \Big(\mathsf{Reduction}_{2,(x,y)\mapsto \max(|x|,|y|)} \circ \mathsf{Pointwise}_{2\times 2,(x,y)\mapsto x-y}\Big)\Big)\end{aligned}$$

Breaking Down Operators into Expressions

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

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

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

Problem specification: hand-developed or automatically produced

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

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

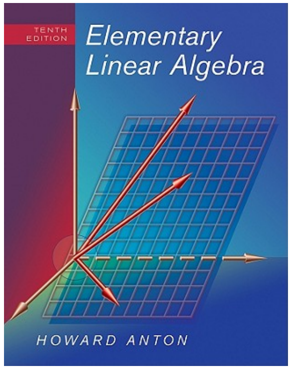
$$(\diamond)_n \to \text{Pointwise}_{n\times n,(a,b)\mapsto a\diamond b}, \quad \diamond \in \{+, -\cdot, \wedge, \vee, \ldots\}$$

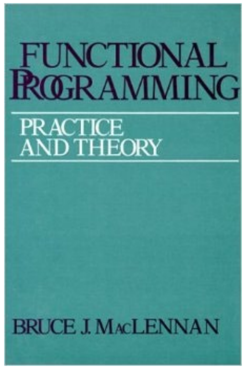
$$\|.\|_\infty^n \to \text{Reduction}_{n,(a,b)\mapsto \max(|a|,|b|)}$$

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

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

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





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
  Γ(.), distributions, Lebesgue integral
- **Semantics preserving**
  rule chain = formal proof
- **Automation**
  Mathematica, Maple

Table of Integrals

BASIC FORMS

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

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

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

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

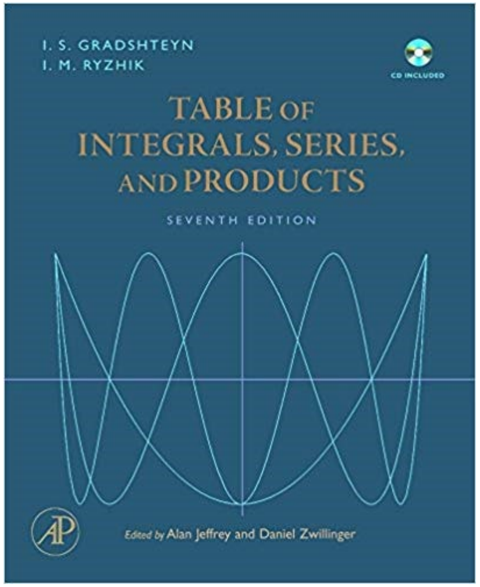
RATIONAL FUNCTIONS

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

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

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

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



Wolfram Mathematica

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

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

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

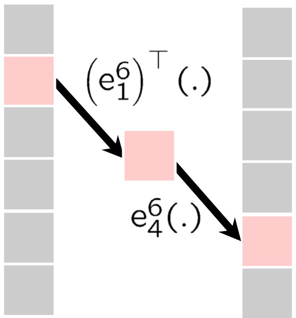
Out[33]= 0

Σ–OL: Low-Level Operator Language

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

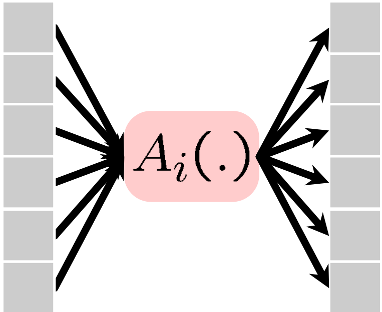
$$(\mathrm{e}_i^n)^\top(.) : \mathbb{R}^n \to \mathbb{R}^1$$
$$(x_i)_{i=0,\dots,n-1} \mapsto x_i$$

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



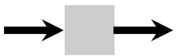
- **Iterative operations:** *loop*

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



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

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



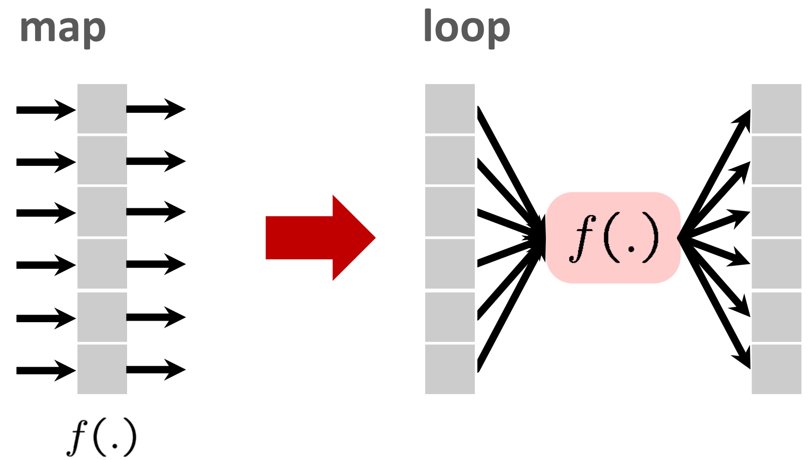
Σ-OL operator expressions = array-based programs with for loops

Rule-Based Translation and Optimization

- **Translating Basic OL into Σ-OL**

$$\mathrm{Pointwise}_{n,f_i} \rightarrow \sum_{i=0}^{n-1} \left(\mathrm{e}_i^n \circ \mathrm{Atomic}_{f_i} \circ (\mathrm{e}_i^n)^\top\right)$$

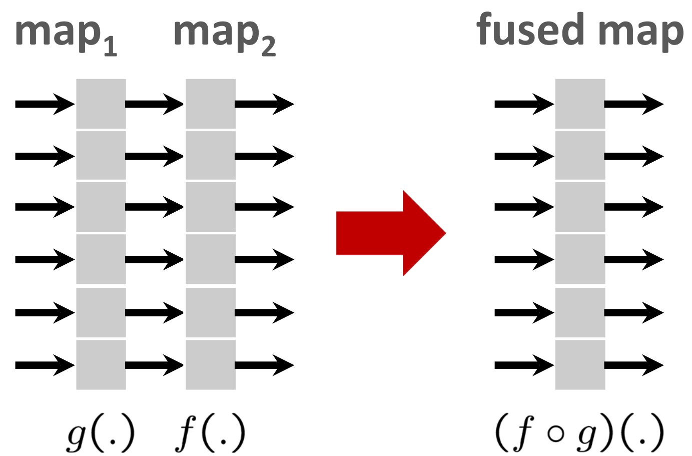
$$\mathrm{Reduction}_{n,(a,b)\mapsto a+b} \rightarrow \sum_{i=0}^{n-1} (\mathrm{e}_i^n)^\top$$



- **Optimizing Basic OL/Σ-OL**

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

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



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 Σ–OL to Abstract Code

Compilation rules: recursive descent

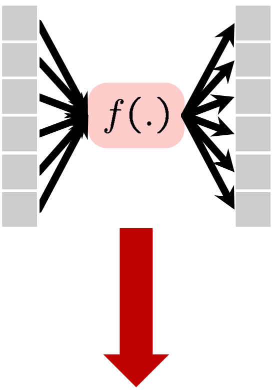
$$\mathsf{Code}\left(y = (A \circ B)(x)\right) \;\to\; \left\{\mathtt{decl}(t), \mathsf{Code}\left(t = B(x)\right), \mathsf{Code}\left(y = A(t)\right)\right\}$$

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

$$\mathsf{Code}\left(y = (\mathsf{e}_i^n)^\top(x)\right) \;\to\; y[0] := x[i]$$

$$\mathsf{Code}\left(y = \mathsf{e}_i^n(x)\right) \;\to\; \left\{y = \vec{0}, y[i] := x[0]\right\}$$

$$\mathsf{Code}\left(y = \mathsf{Atomic}_f(x)\right) \;\to\; y[0] := f(x[i])$$



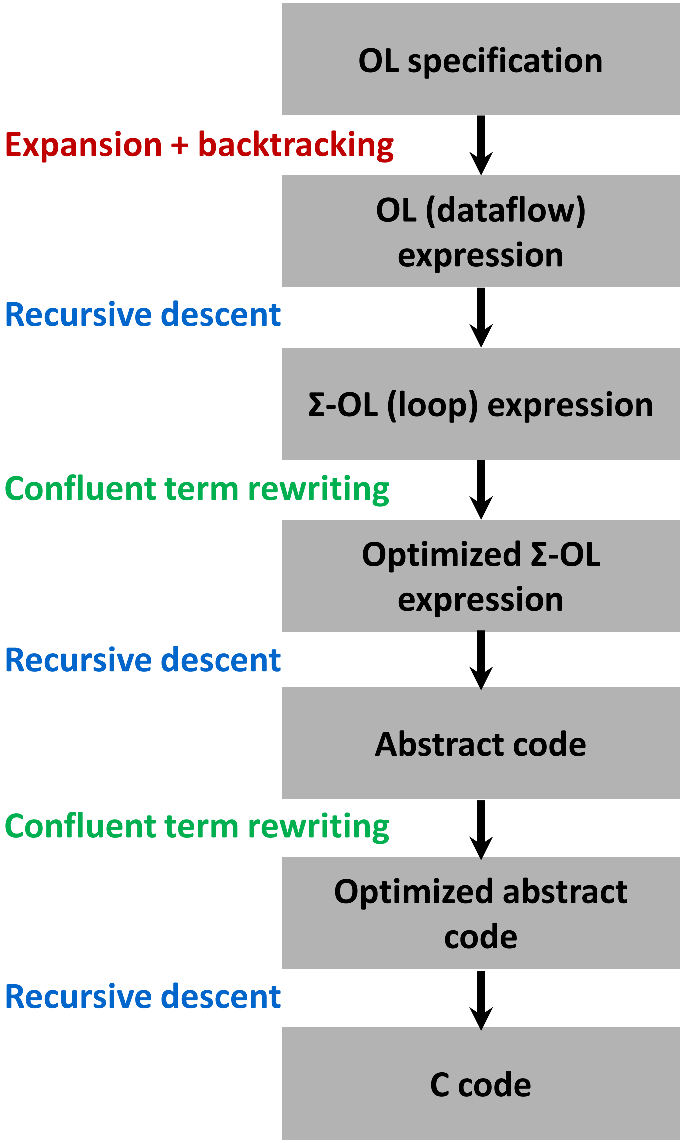

```
chain(
  assign(Y, V(0.0),
  loop(i1, [0..5],
    assign(nth(y, i1),
      f(nth(X, i1)))
  )
)
```

Cleanup rules: term rewriting

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

Rule-based code generation and backend compilation

Putting it Together: One Big Rule System



Mathematical specification

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

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

Final code

```
int dwmonitor(float  *X, double  *D) {
    __m128d u1, u2, u3, u4, u5, u6, u7, u8 , x1, x10, x13, x14, x17
    int w1;
    unsigned _xm = _mm_getcsr();
    _mm_setcsr(_xm & 0xffff0000 | 0x0000dfc0);
    u5 = _mm_set1_pd(0.0);
    u2 = _mm_cvtps_pd(_mm_addsub_ps(_mm_set1_ps(FLT_MIN), _mm_set1_
    u1 = _mm_set_pd(1.0, (-1.0));
    for(int i5 = 0; i5 <= 2; i5++) {
        x6 = _mm_addsub_pd(_mm_set1_pd((DBL_MIN + DBL_MIN)), _mm_lo
        x1 = _mm_addsub_pd(_mm_set1_pd(0.0), u1);
        x2 = _mm_mul_pd(x1, x6);
        x3 = _mm_mul_pd(_mm_shuffle_pd(x1, x1, _MM_SHUFFLE2(0, 1)),
        x4 = _mm_sub_pd(_mm_set1_pd(0.0), _mm_min_pd(x3, x2));
        u3 = _mm_add_pd(_mm_max_pd(_mm_shuffle_pd(x4, x4, _MM_SHUFF
```

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_loaddup_pd(&(D[i5])));
        x1 = _mm_addsub_pd(_mm_set1_pd(0.0), u1);
        x2 = _mm_mul_pd(x1, x6);
        x3 = _mm_mul_pd(_mm_shuffle_pd(x1, x1, _MM_SHUFFLE2(0, 1)), x6);
```

$$\mathsf{SafeDist}_{V,A,b,\varepsilon} = \mathsf{Atomic}_{(x,y)\mapsto x<y} \circ \Big(\big(\mathsf{Reduction}_{3,(x,y)\mapsto x+y} \circ \mathsf{Pointwise}_{3,x\mapsto a_i x} \circ \mathsf{Induction}_{3,(a,b)\mapsto ab,1} \big) \times \big(\mathsf{Reduction}_{2,(x,y)\mapsto \max(|x|,|y|)} \circ \mathsf{Pointwise}_{2\times 2,(x,y)\mapsto x-y} \big) \Big)$$

```
    }
    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;
}
```

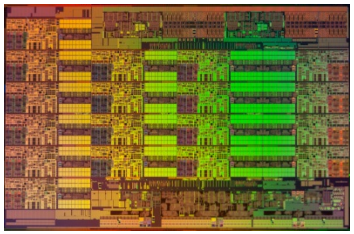
Outline

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

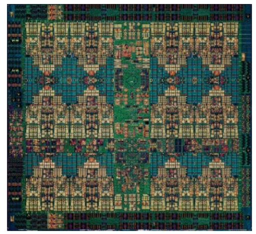
F. Franchetti, T. M. Low, D. T. Popovici, R. M. Veras, D. G. Spampinato, J. R. Johnson, M. Püschel, J. C. Hoe, J. M. F. Moura:
SPIRAL: Extreme Performance Portability, **Proceedings of the IEEE, Vol. 106, No. 11, 2018.**
Special Issue on *From High Level Specification to High Performance Code*

Today’s Computing Landscape

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



Intel Xeon 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

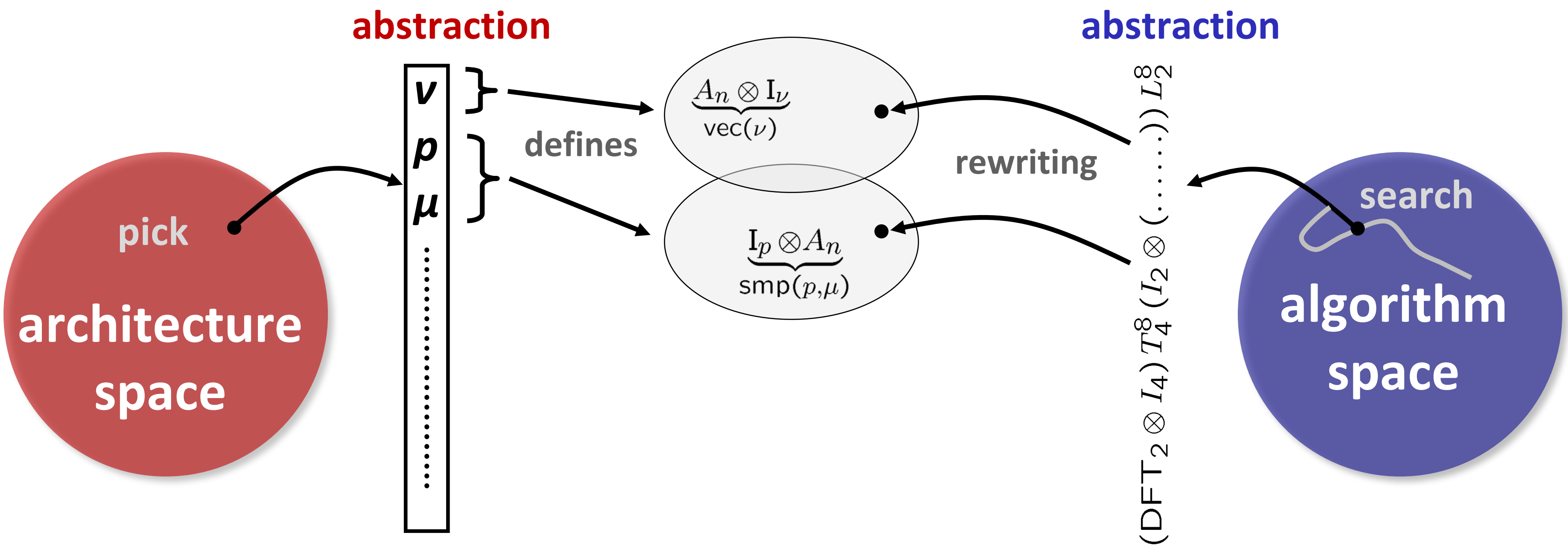
Intel Atom C3858
32 Gflop/s, 25 W
16 cores, 2.0 GHz
2-way/4-way SSSE3

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



Architectural parameter:
Vector length,
#processors, …

optimization →

Kernel:
problem size,
algorithm choice

Some Application Domains in OL

Linear Transforms

$$\mathbf{DFT}_n \rightarrow (\mathbf{DFT}_k \otimes \mathrm{I}_m)\, \mathrm{T}^n_m (\mathrm{I}_k \otimes \mathbf{DFT}_m)\, \mathrm{L}^n_k, \quad n = km$$

$$\mathbf{DFT}_n \rightarrow P_n(\mathbf{DFT}_k \otimes \mathbf{DFT}_m) Q_n, \quad n = km,\ \gcd(k,m) = 1$$

$$\mathbf{DFT}_p \rightarrow R^T_p(\mathrm{I}_1 \oplus \mathbf{DFT}_{p-1}) D_p (\mathrm{I}_1 \oplus \mathbf{DFT}_{p-1}) R_p, \quad p \text{ prime}$$

$$\mathbf{DCT\text{-}3}_n \rightarrow (\mathrm{I}_m \oplus \mathrm{J}_m)\, \mathrm{L}^n_m (\mathbf{DCT\text{-}3}_m(1/4) \oplus \mathbf{DCT\text{-}3}_m(3/4)) \cdot (\mathrm{F}_2 \otimes \mathrm{I}_m) \begin{bmatrix} \mathrm{I}_m & 0 \oplus -\mathrm{J}_{m-1} \\ & \frac{1}{\sqrt{2}}(\mathrm{I}_1 \oplus 2\,\mathrm{I}_m) \end{bmatrix}, \quad n = 2m$$

$$\mathbf{DCT\text{-}4}_n \rightarrow S_n \mathbf{DCT\text{-}2}_n \,\mathrm{diag}_{0 \le k < n}(1/(2\cos((2k+1)\pi/4n)))$$

$$\mathbf{IMDCT}_{2m} \rightarrow (\mathrm{J}_m \oplus \mathrm{I}_m \oplus \mathrm{I}_m \oplus \mathrm{J}_m) \left(\left(\begin{bmatrix} 1 \\ -1 \end{bmatrix} \otimes \mathrm{I}_m \right) \oplus \left(\begin{bmatrix} -1 \\ -1 \end{bmatrix} \otimes \mathrm{I}_m \right) \right) \mathrm{J}_{2m}\, \mathbf{DCT\text{-}4}_{2m}$$

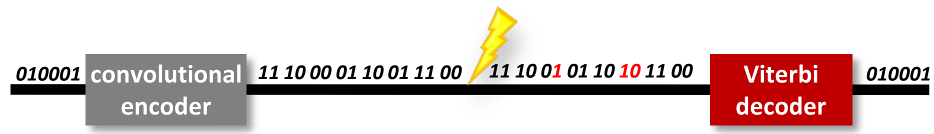
$$\mathbf{WHT}_{2^k} \rightarrow \prod_{i=1}^{t} (\mathrm{I}_{2^{k_1+\cdots+k_{i-1}}} \otimes \mathbf{WHT}_{2^{k_i}} \otimes \mathrm{I}_{2^{k_{i+1}+\cdots+k_t}}), \quad k = k_1 + \cdots + k_t$$

$$\mathbf{DFT}_2 \rightarrow \mathrm{F}_2$$

$$\mathbf{DCT\text{-}2}_2 \rightarrow \mathrm{diag}(1, 1/\sqrt{2})\, \mathrm{F}_2$$

$$\mathbf{DCT\text{-}4}_2 \rightarrow \mathrm{J}_2\, \mathrm{R}_{13\pi/8}$$

Software Defined Radio

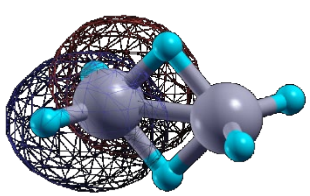


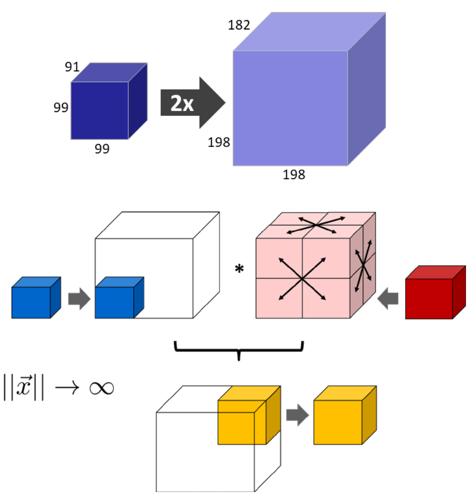
$$F_{K,F} \rightarrow \prod_{i=1}^{F} \left((I_{2^{K-2}} \otimes_j B_{F-i,j}) L^{2^{K-1}}_{2^{K-2}} \right)$$

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

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

PDEs/HPC Simulations



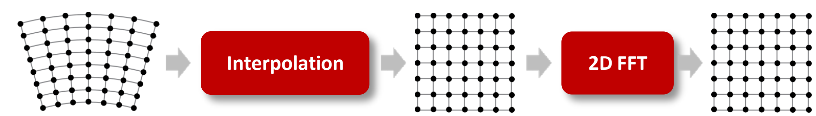


$$\Phi : \mathbb{R}^3 \rightarrow \mathbb{R}$$

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

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

Synthetic Aperture Radar (SAR)



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

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

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

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

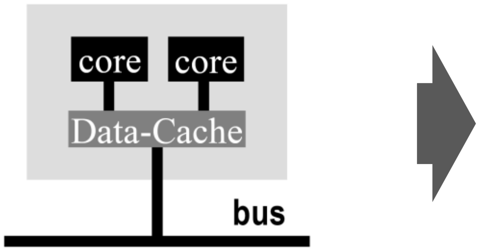
$$\mathrm{InterpSeg}_k \rightarrow \mathrm{G}^{u\cdot n\rightarrow k}_f \circ \mathrm{iPrunedDFT}_{n\rightarrow u\cdot n} \circ \left(\frac{1}{n}\right) \circ \mathrm{DFT}_n$$

Formal Approach for all Types of Parallelism

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

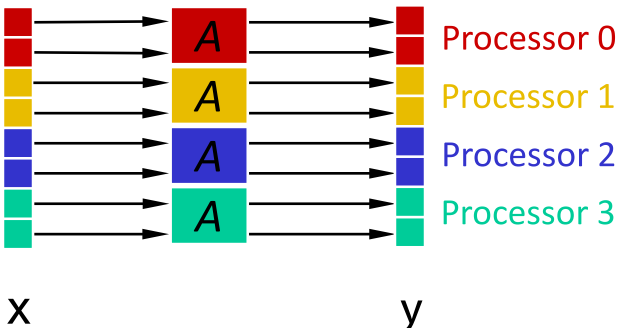
- **Hardware abstraction: shared cache with cache lines**



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

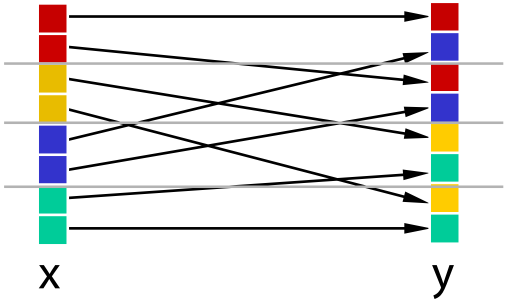
- **Tensor product: embarrassingly parallel operator**

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



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

$$y = \mathrm{L}_4^8(x)$$



Example Program Transformation Rule Set

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

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

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

Recursive rules

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

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

Base case rules

Autotuning in Constraint Solution Space

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

AVX 2-way
`_Complex double`

DFT₈ is written in the page's notation as DFT_8

Base cases

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

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

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

Transformation rules

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

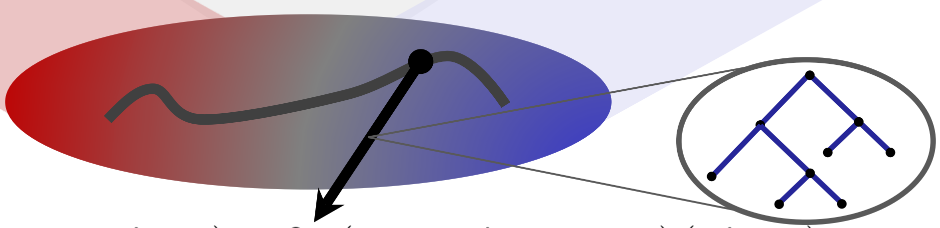
$$\mathrm{L}^{n\nu}_\nu \rightarrow (\mathrm{L}^n_\nu\otimes \mathrm{I}_\nu)(\mathrm{I}_{n/\nu}\otimes \mathrm{L}^{\nu^2}_\nu)$$

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

Breakdown rules

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

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



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

OL specification

Expansion + backtracking

OL (dataflow) expression

Recursive descent

Σ-OL (loop) expression

Confluent term rewriting

Optimized Σ-OL expression

Recursive descent

Abstract code

Confluent term rewriting

Optimized abstract code

Recursive descent

C code

Translating an OL Expression Into Code

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

Output =

Ruletree, expanded into

OL Expression:

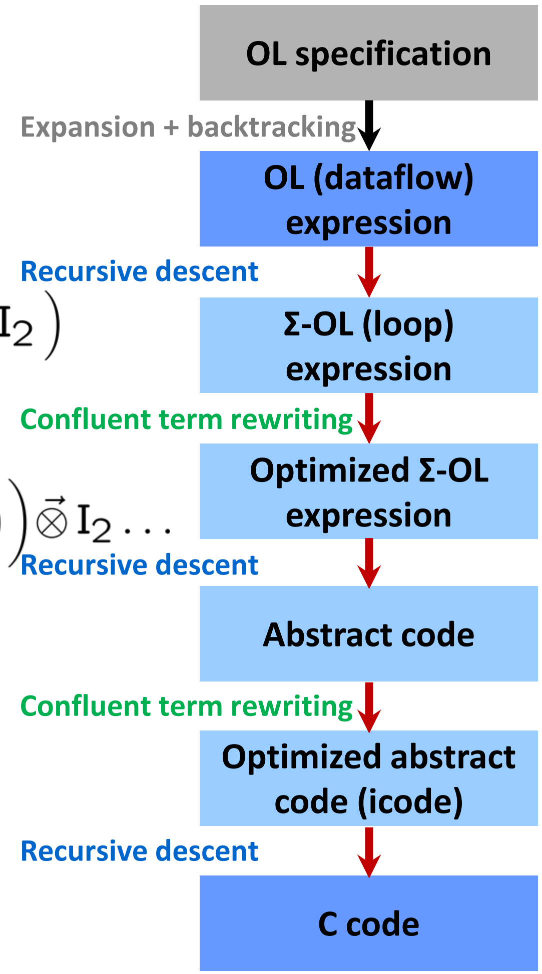
$$\left((\mathrm{F}_2\otimes \mathrm{I}_2)\mathrm{T}^4_2(\mathrm{I}_2\otimes \mathrm{F}_2)\mathrm{L}^4_2\vec{\otimes}\,\mathrm{I}_2\right)\underbrace{\mathrm{T}^8_2}_{\mathrm{vec}(2)}\left(\mathrm{I}_2\otimes\underbrace{\mathrm{L}^4_2}_{\mathrm{vec}(2)}(\mathrm{F}_2\vec{\otimes}\,\mathrm{I}_2)\right)\left(\mathrm{L}^4_2\vec{\otimes}\,\mathrm{I}_2\right)$$

∑-OL:

$$\left(\sum_{j=0}^{1}\left(\mathrm{S}_{\imath_2\otimes(j)_2}\mathrm{F}_2\mathrm{Map}^2_{x\mapsto\omega_4^{2i+j}x}\mathrm{G}_{\imath_2\otimes(j)_2}\right)\sum_{j=0}^{1}\left(\mathrm{S}_{(j)_2\otimes\imath_2}\mathrm{F}_2\mathrm{G}_{\imath_2\otimes(j)_2}\right)\right)\vec{\otimes}\,\mathrm{I}_2\dots$$

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

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

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

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

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

`DFT4([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

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

FFTX and SpectralPACK

Numerical Linear Algebra		Spectral Algorithms
LAPACK LU factorization Eigensolves SVD …	≈	**SpectralPACK** Convolution Correlation Upsampling Poisson solver …
BLAS BLAS-1 BLAS-2 BLAS-3		**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)

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

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

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

Poisson's equation. Δ is the Laplace operator

Solution characterization

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

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

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

Approach: Green's function

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

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

Method of Local Corrections (MLC)

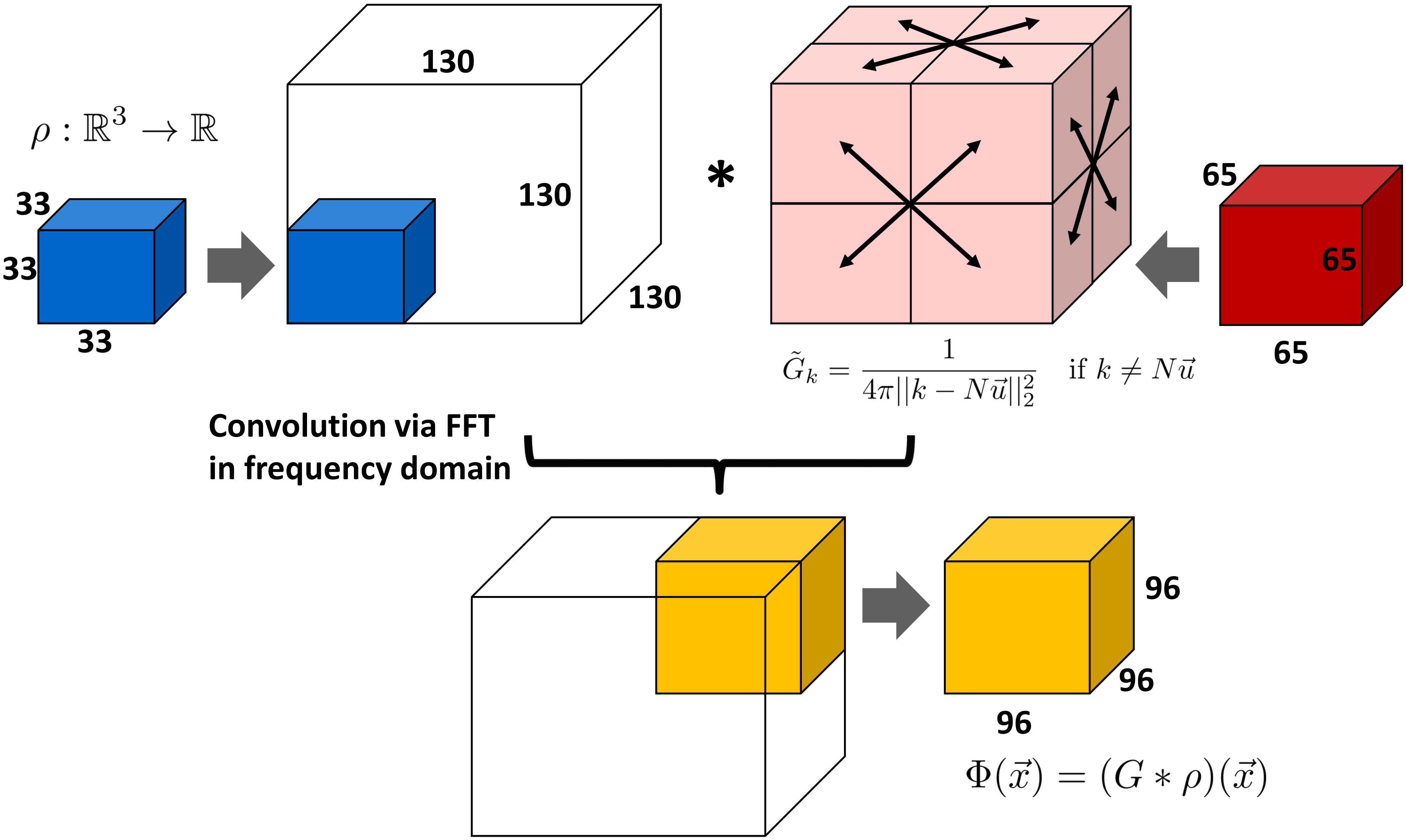
$$\tilde{G}_k = \frac{1}{4\pi||k - N\vec{u}||_2^2} \quad \text{if } k \neq N\vec{u}$$

Green's function kernel in frequency domain

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



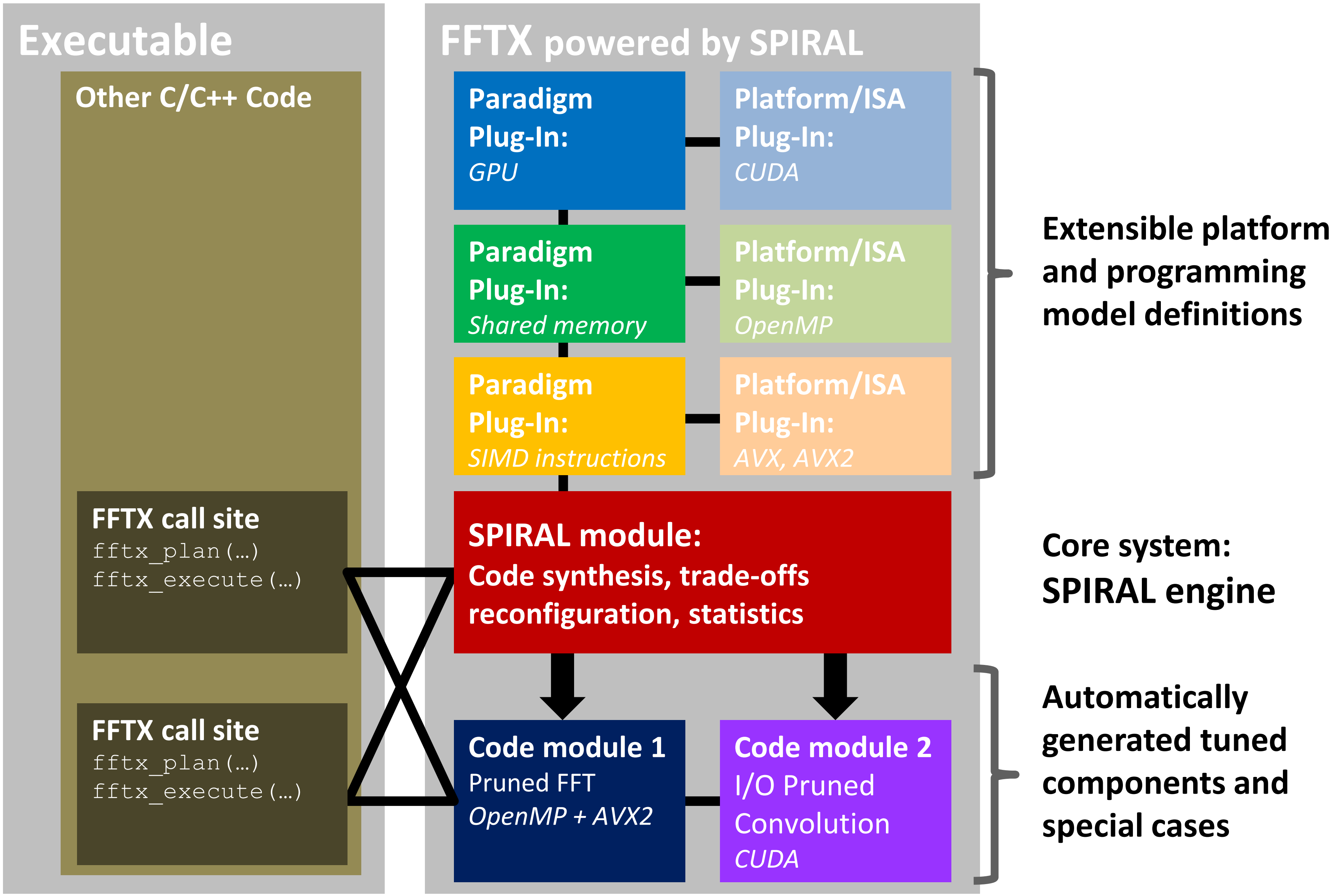
Hockney: Convolution + problem specific zero padding and output subset

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

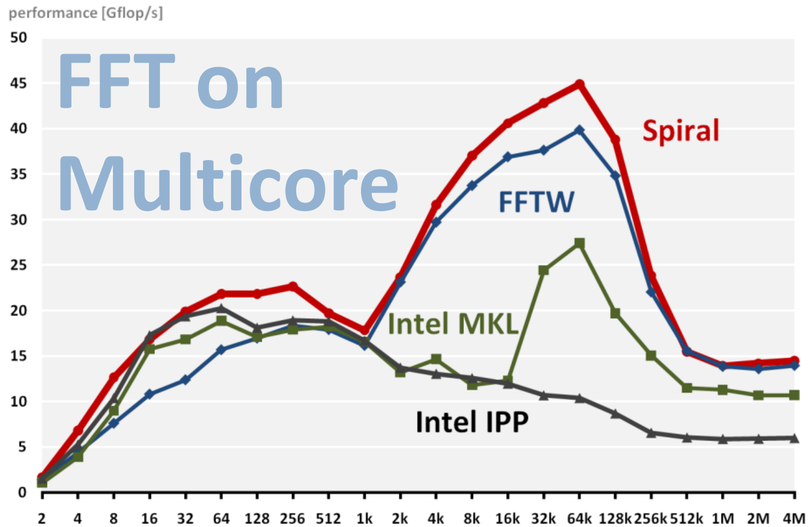


Outline

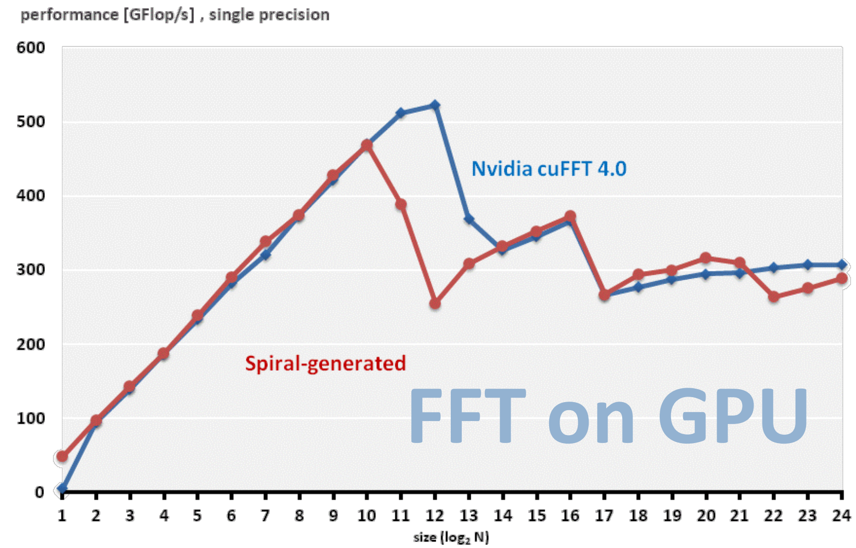
- Introduction
- Operator Language
- Achieving Performance Portability
- FFTX: A Library Frontend for SPIRAL
- **Summary**

Synthesis: FFTs and Spectral Algorithms

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

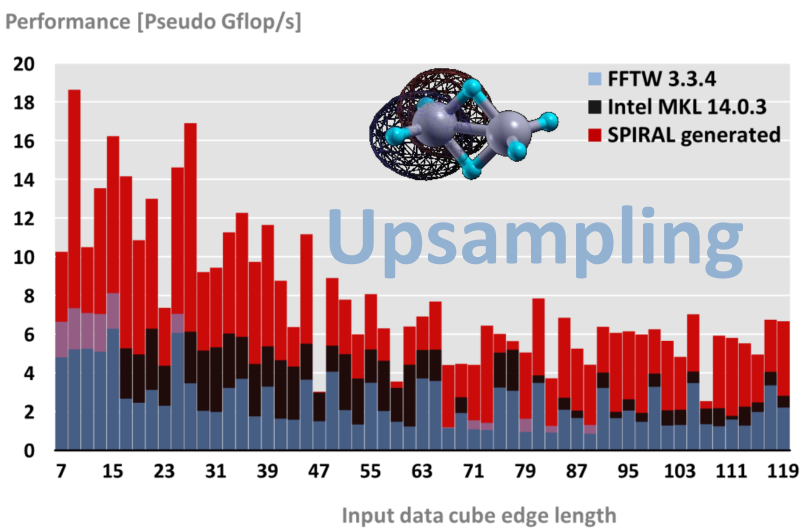


1D Batch DFT (Nvidia GTX 480)

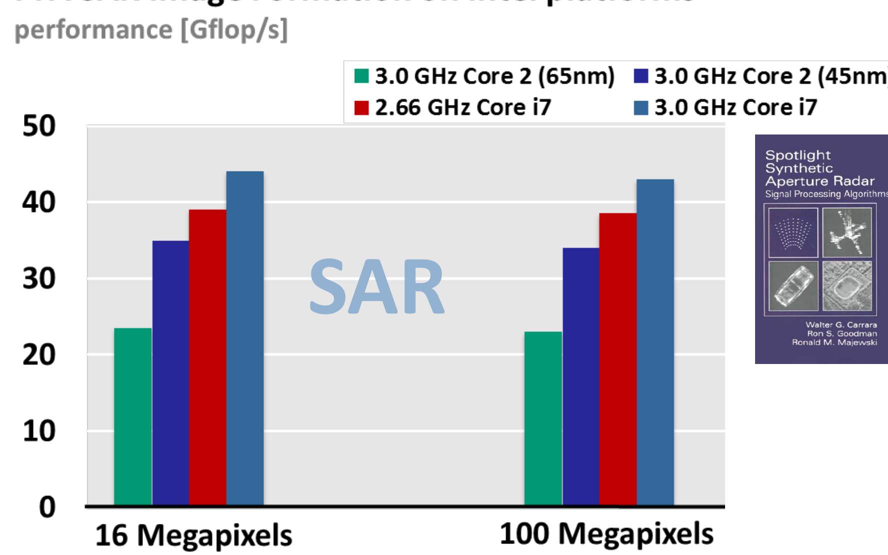


Performance of 2x2x2 Upsampling on Haswell

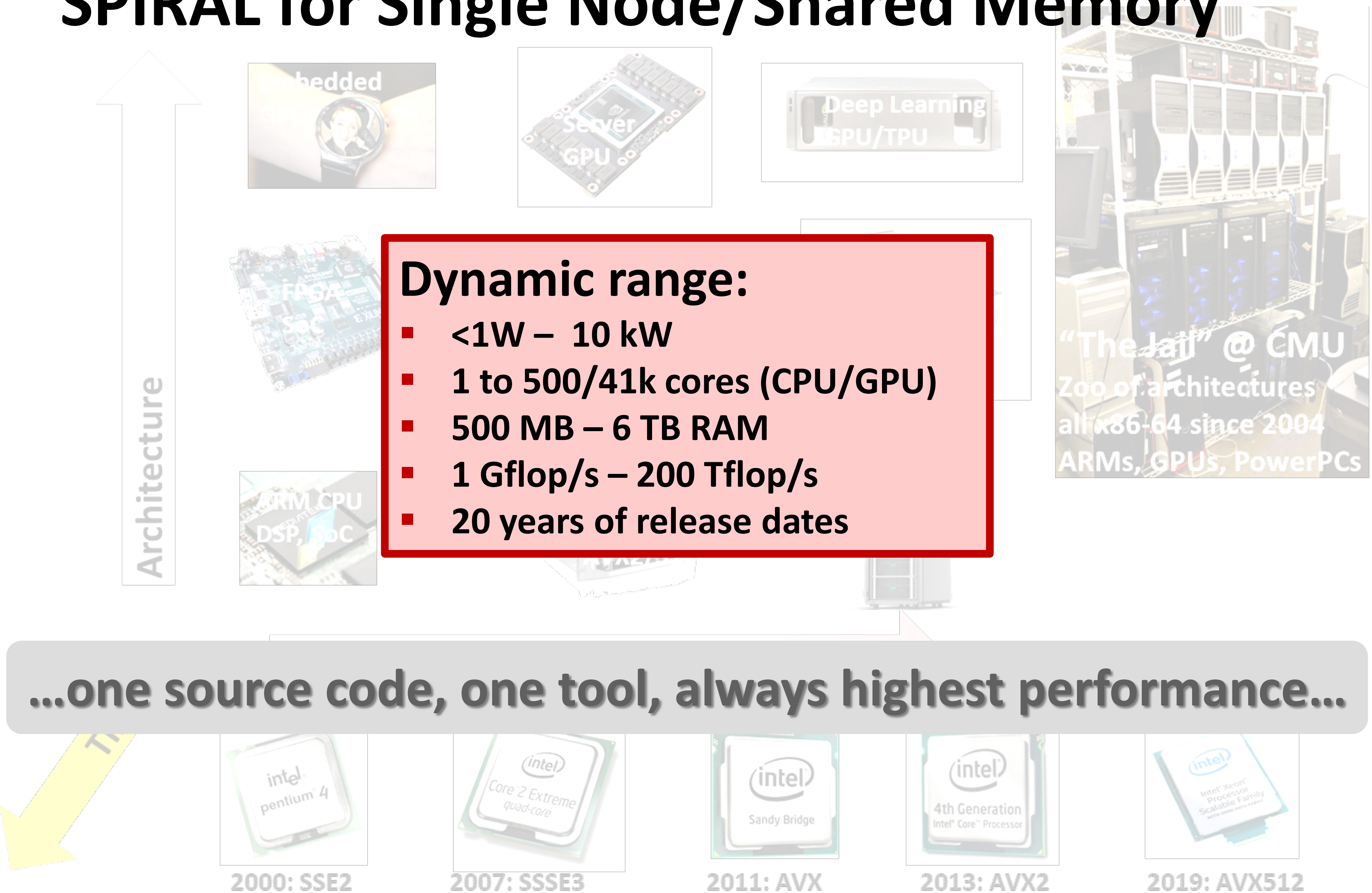
3.5 GHz, AVX, double precision, interleaved input, single core



PFA SAR Image Formation on Intel platforms



SPIRAL for Single Node/Shared Memory



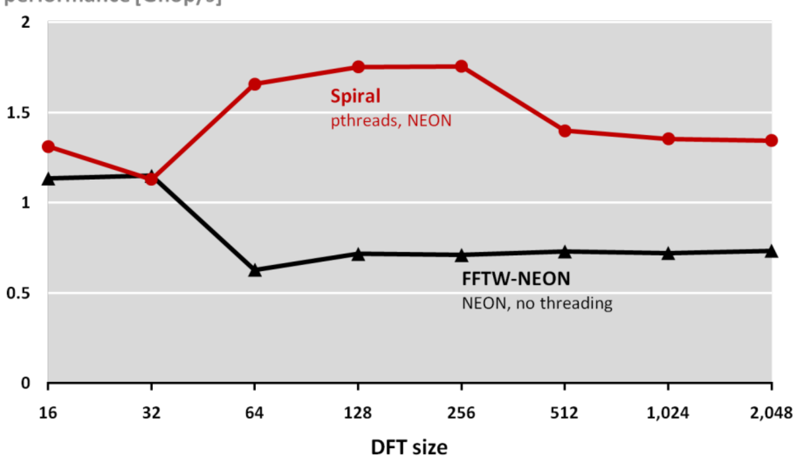
Dynamic range:

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

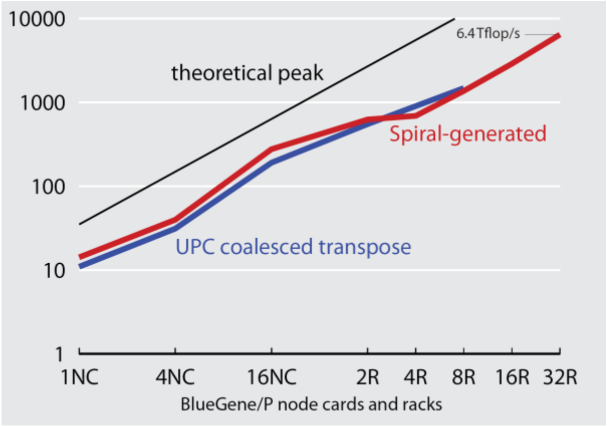
…one source code, one tool, always highest performance…

From Cell Phone To Supercomputer

DFT on Samsung Galaxy S II
Dual-core 1.2 GHz Cortex-A9 with NEON ISA



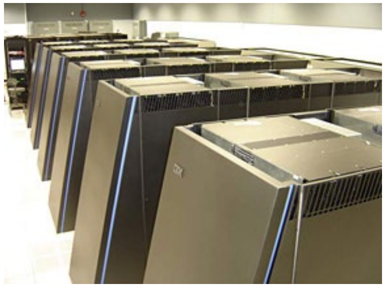
Global FFT (1D FFT, HPC Challenge)
performance [Gflop/s]



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

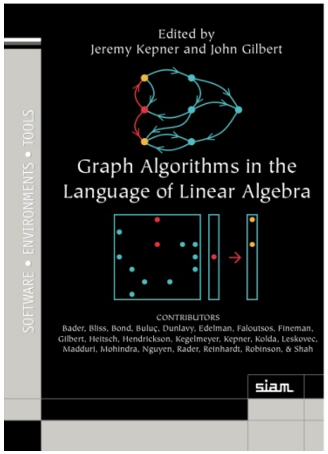


F. Gygi, E. W. Draeger, M. Schulz, B. R. de Supinski, J. A. Gunnels, V. Austel, J. C. Sexton, F. Franchetti, S. Kral, C. W. Ueberhuber, J. Lorenz, **"Large-Scale Electronic Structure Calculations of High-Z Metals on the BlueGene/L Platform,"** In Proceedings of Supercomputing, 2006. ***2006 Gordon Bell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

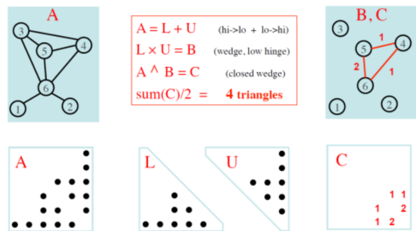
G. Almási, B. Dalton, L. L. Hu, F. Franchetti, Y. Liu, A. Sidelnik, T. Spelce, I. G. Tānase, E. Tiotto, Y. Voronenko, X. Xue, **"2010 IBM HPC Challenge Class II Submission,"** ***2010 HPC Challenge Class II Award (Most Productive System).***

Current Research Directions

SPIRAL for Graphs

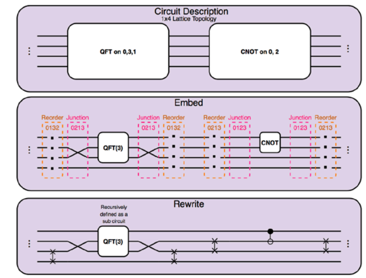


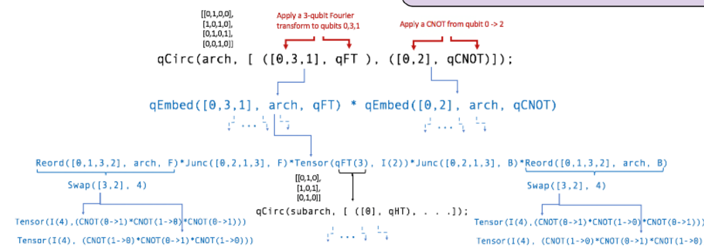
$$\Delta = \Delta + \frac{1}{2}\alpha_{10}A_{00}\alpha_{01}$$



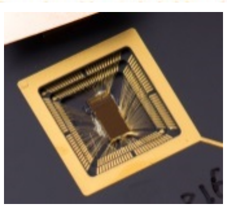

```
# OL Algorithm Specification
Accum(i4, 1, X.N-1,
  Accum_X(i6, [ i4, 0 ], i4,
    Dot([ i6, add(i4, V(1)) ],
       [ i4, add(i4, V(1)) ],
        sub(sub(X.N, i4), V(1)))
    )
)
```

Spiral for Quantum Computing

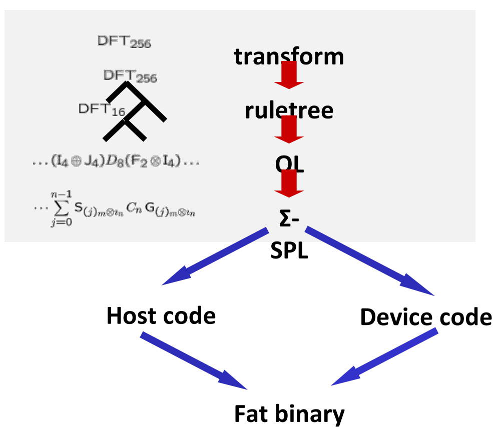




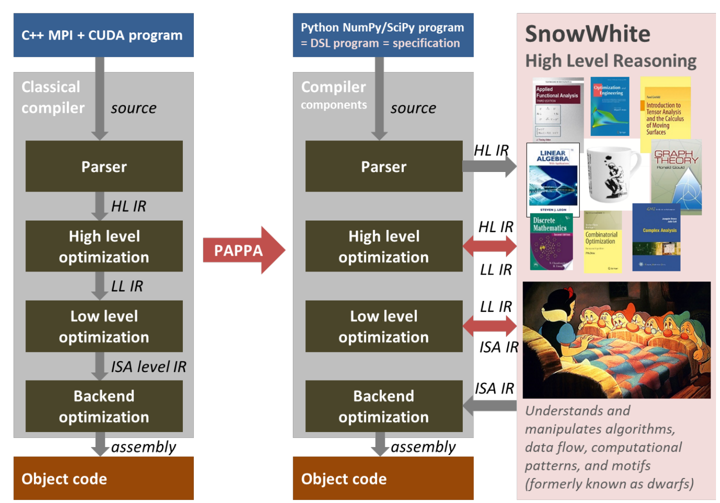
HW/SW Co-Design



RISC-V

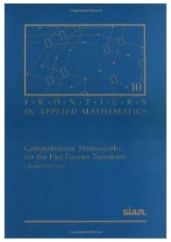


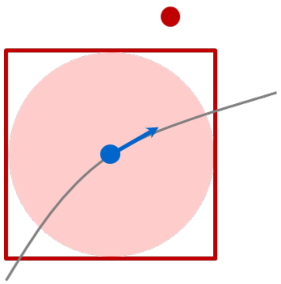
SPIRAL as AI in Compilers

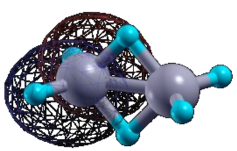


SPIRAL: AI for High Performance Code

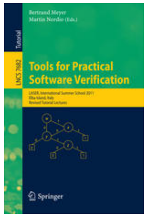
Algorithms

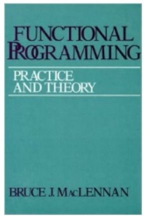


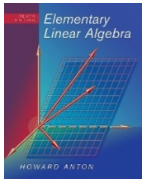




Correctness







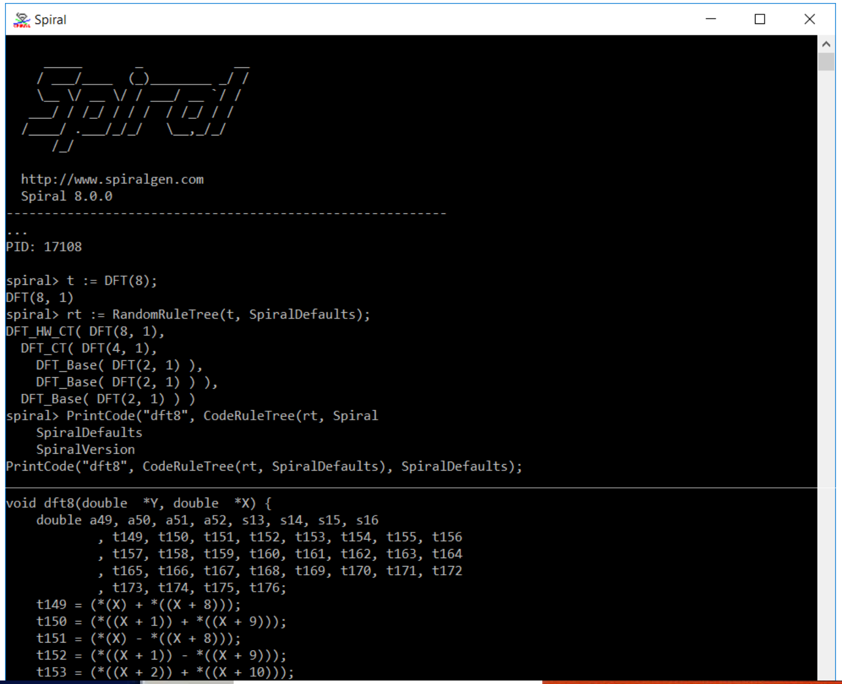

```
int dwmonitor(float  *X, double  *D) {
    __m128d u1, u2, u3, u4, u5, u6, u7, u8,...
    unsigned _xm = _mm_getcsr();
    _mm_setcsr(_xm & 0xffff0000 | 0x0000dfc0);
    u5 = _mm_set1_pd(0.0);
    u2 = _mm_cvtps_pd(_mm_addsub_ps(
    _mm_set1_ps(FLT_MIN), _mm_set1_ps(X[0])));
    u1 = _mm_set_pd(1.0, (-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 8.2.0: 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



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*