SPIRAL:
Formal Software Synthesis of Computational Kernels

Franz Franchetti

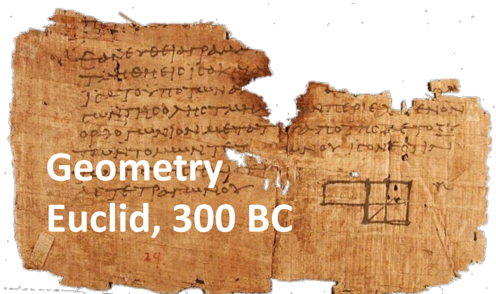
Department of Electrical and Computer Engineering
Carnegie Mellon University
www.ece.cmu.edu/~franzf

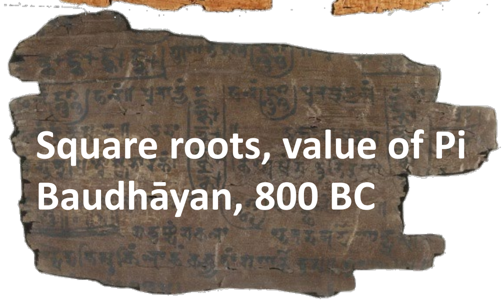
CTO and Co-Founder
SpiralGen, Inc.
www.spiralgen.com

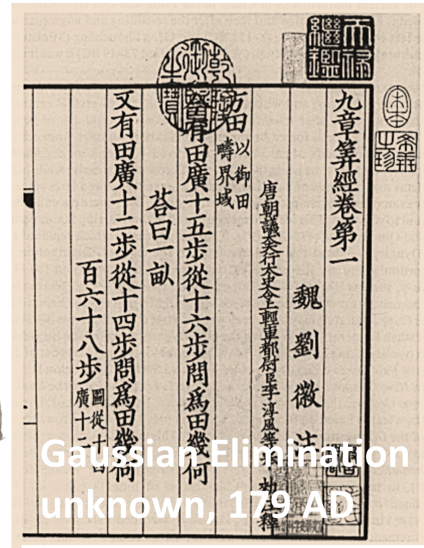
Joint work with
the SPIRAL, FFTX, DESA, PERFECT, BRASS, and HACMS groups

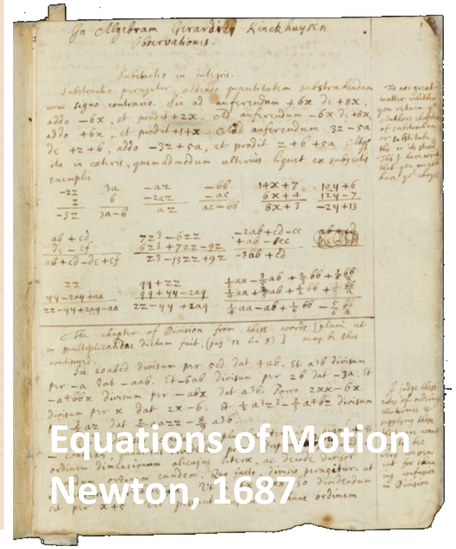
This work was supported by DARPA, DOE, ONR, NSF, Intel, Mercury, and Nvidia

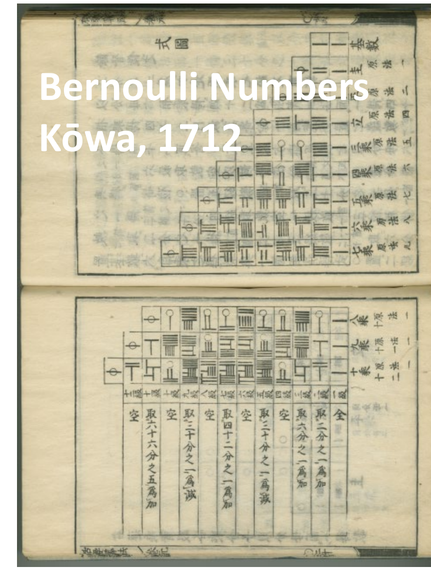
Algorithms and Mathematics: 2,500+ Years



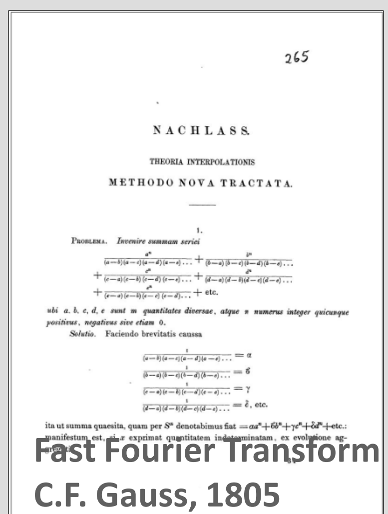


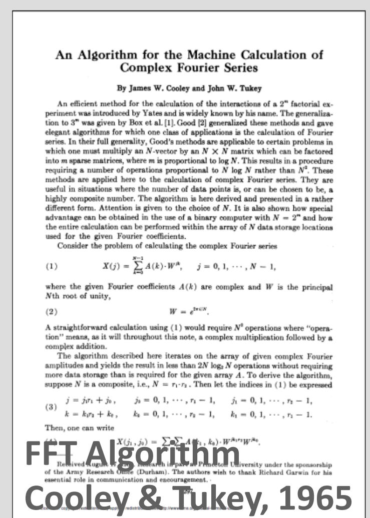


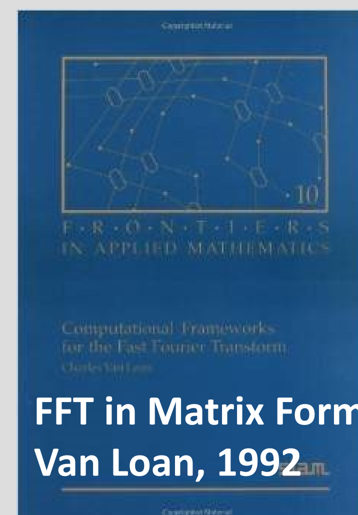




Fast Fourier Transform





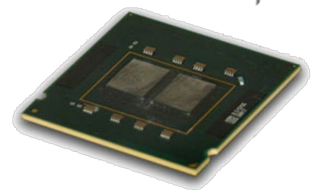


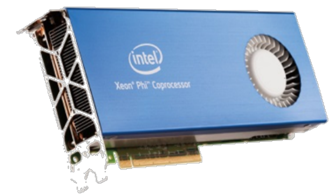
Computers I have Used: 10^7x Gain in 30 Years

The first computer I…

10 kflop/s

"My" first…





…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 2018

300 Gflop/s

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

Dell Power Edge
80 cores @ 3GHz
3 TB RAM

Dell Precision 3620
3.7 GHz Xeon Quad-core
Discrete GPU, 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

Floating-Point and Bugs (and Features)

Ariane-5

`double d > SHRT_MAX`

64-bit double cast to 16-bit integer

Landshark Robot

10˚ ≠ 10 rad

Pentium FDIV Bug

$$\frac{4{,}195{,}835}{3{,}145{,}727} \neq 1.333{\color{red}739068902037589}$$

Intel IEEE 754 Optimization

`min(-4.940656458412E-324, 0) = 0`

Since SSSE3: `_MM_DENORMALS_ZERO_ON`

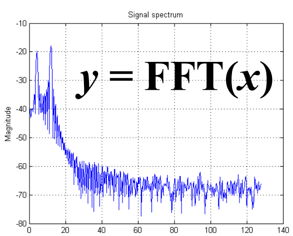
Idea: Go from Mathematics to Software

Given:

- Mathematical problem specification
  does not change
- Computer platform
  changes often

Wanted:

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



on

automatic

```
void fft64(double  *Y, double  *X) {
    ...
    s5674 = _mm256_permute2f128_pd(s5672, s5673, (0) | ((2) << 4));
    s5675 = _mm256_permute2f128_pd(s5672, s5673, (1) | ((3) << 4))
    s5676 = _mm256_unpacklo_pd(s5674, s5675);
    s5677 = _mm256_unpackhi_pd(s5674, s5675);
    s5678 = *((a3738 + 16));
    s5679 = *((a3738 + 17));
    s5680 = _mm256_permute2f128_pd(s5678, s5679, (0) | ((2) << 4));
    s5681 = _mm256_permute2f128_pd(s5678, s5679, (1) | ((3) << 4));
    s5682 = _mm256_unpacklo_pd(s5680, s5681);
    s5683 = _mm256_unpackhi_pd(s5680, s5681);
    t5735 = _mm256_add_pd(s5676, s5682);
    t5736 = _mm256_add_pd(s5677, s5683);
    t5737 = _mm256_add_pd(s5670, t5735);
    t5738 = _mm256_add_pd(s5671, t5736);
    t5739 = _mm256_sub_pd(s5670, _mm256_mul_pd(_mm_vbroadcast_sd(&(C22)), t5735));
    t5740 = _mm256_sub_pd(s5671, _mm256_mul_pd(_mm_vbroadcast_sd(&(C22)), t5736));
    t5741 = _mm256_mul_pd(_mm_vbroadcast_sd(&(C23)), _mm256_sub_pd(s5677, s5683));
    t5742 = _mm256_mul_pd(_mm_vbroadcast_sd(&(C23)), _mm256_sub_pd(s5676, s5682));
    ...
}
```

Example 1: Safety Monitor for Car/Robot

Dynamic Window Monitor

Hybrid Theorem Prover (KeYmaera):

Mathematical guarantee of passive safety:

"I will never hit anybody"

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

Synthesized software (Spiral):

guaranteed sound implementation of equation

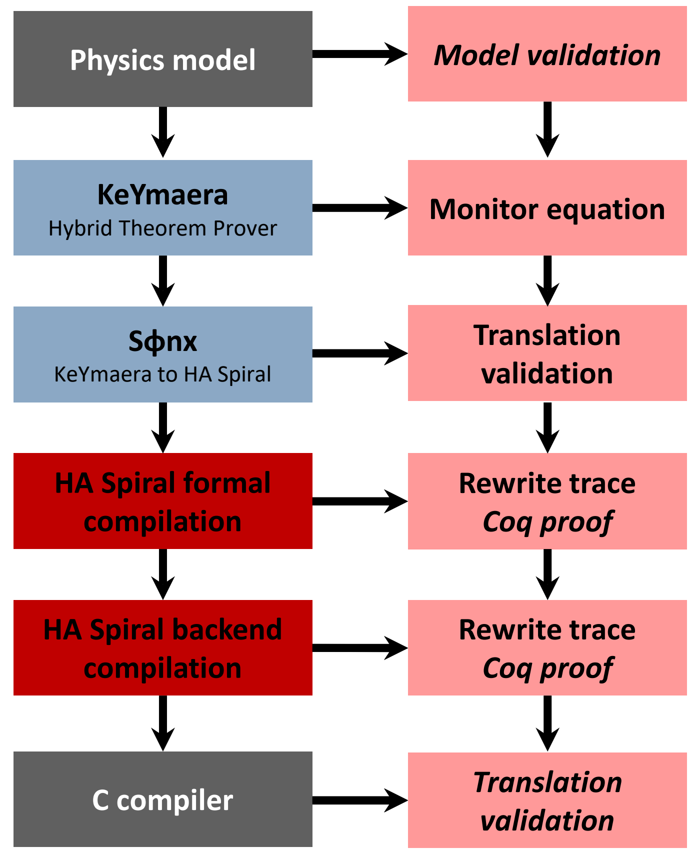
if code says "ok" then equation says "ok"

if code says "don't know" then equation says "ok" or "stop"

If code says "stop" then equation says "stop"

```
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);
    ...
```

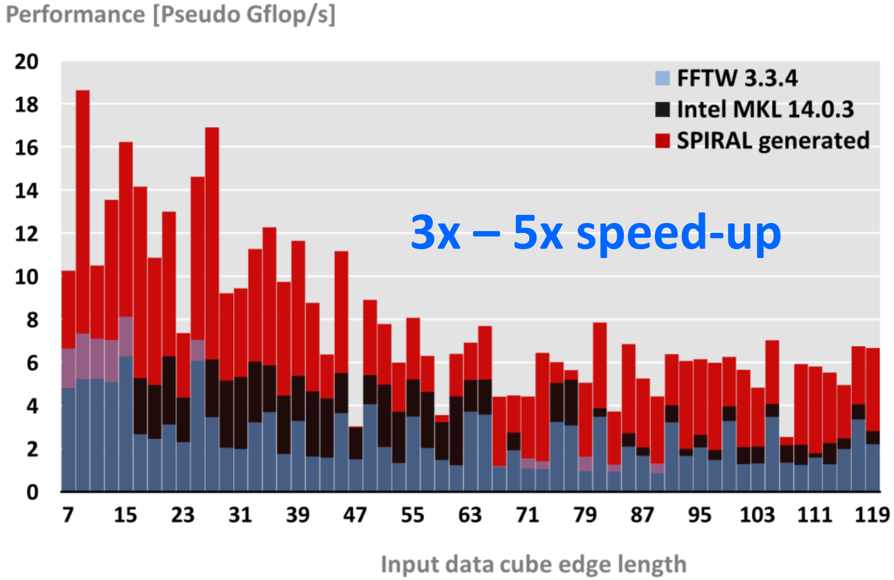
Chain of Evidence



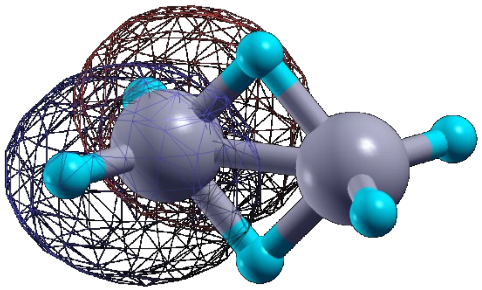
F. Franchetti, T. M. Low, S. Mitsch, J. P. Mendoza, L. Gui, A. Phaosawasdi, D. Padua, S. Kar, J. M. F. Moura, M. Franusich, J. Johnson, A. Platzer, and M. Veloso: **High-Assurance SPIRAL: End-to-End Guarantees for Robot and Car Control**, IEEE Control Systems Magazine, 2017, pages 82-103.

Example 2: Density Functional Theory

Performance of 2x2x2 Upsampling on Haswell
3.5 GHz, AVX, double precision, interleaved input, single core

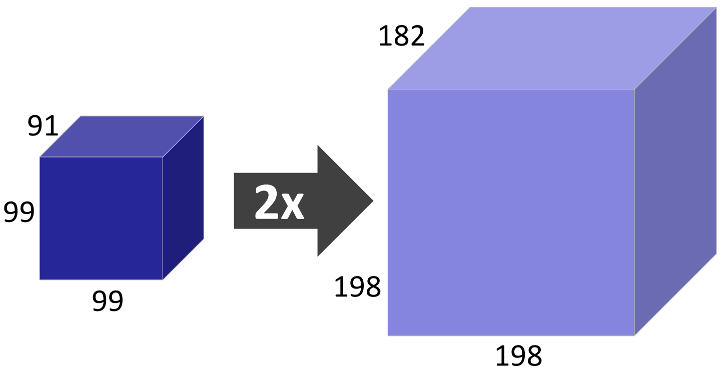


ONETEP



quantum-mechanical calculations based on density-functional theory

Core operation:
FFT-based 3D 2x2x2 upsampling



Unusual requirements:

- Odd FFT sizes
- Small sizes
- Rectangular box

ONETEP = Order-*N* Electronic Total Energy Package

P. D. Haynes, C.-K. Skylaris, A. A. Mostofi and M. C. Payne, **"ONETEP: linear-scaling density-functional theory with plane waves,"** Psi-k Newsletter 72, 78-91 (December 2005)

T. Popovici, F. Russell, K. Wilkinson, C-K. Skylaris, P. H. J. Kelly, F. Franchetti, **"Generating Optimized Fourier Interpolation Routines for Density Functional Theory Using SPIRAL,"** 29th International Parallel & Distributed Processing Symposium (IPDPS), 2015.

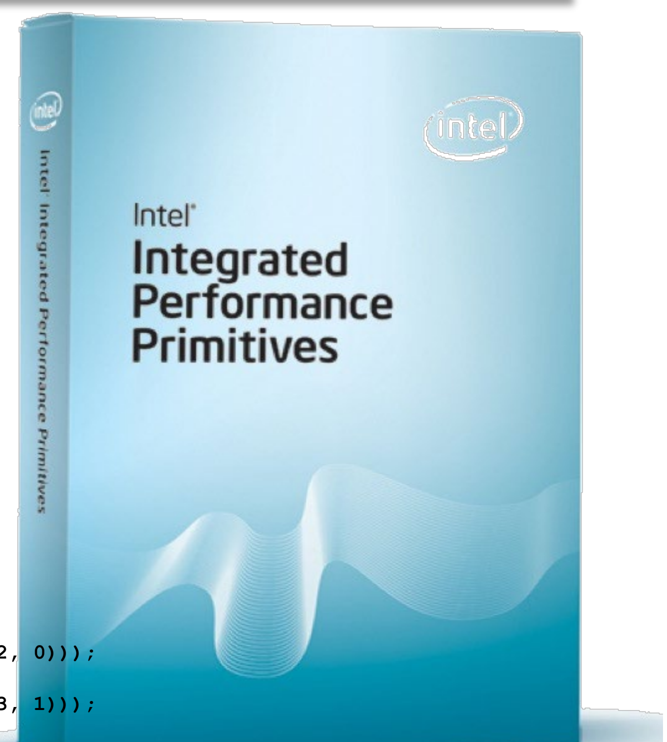
Example 3: Synthesis of Production Code

```
...
s3013 = _mm_loadl_pi(a772, ((float  *) X));
s3014 = _mm_loadh_pi(_mm_loadl_pi(a772, ((float  *) (X + 2))), ((float  *) (X + 6)));
```

Spiral-Synthesized code in Intel's Library IPP 6 and 7

- IPP = Intel's performance primitives, part of Intel C++ Compiler suite
- Generated: 3984 C functions (signal processing) = 1M lines of code
- Full parallelism support
- Computer-generated code: Faster than what was achievable by hand

```
s3017 = _mm_loadl_pi(a772, ((float  *) (X + 14)));
s3018 = _mm_loadh_pi(_mm_loadl_pi(a772, ((float  *) (X + 24))), ((float  *) (X + 20)));
s3019 = _mm_loadl_pi(a772, ((float  *) (X + 8)));
s3020 = _mm_loadh_pi(_mm_loadl_pi(a772, ((float  *) (X + 10))), ((float  *) (X + 4)));
s3021 = _mm_loadl_pi(a772, ((float  *) (X + 12)));
s3022 = _mm_shuffle_ps(s3014, s3015, _MM_SHUFFLE(2, 0, 2, 0));
s3023 = _mm_shuffle_ps(s3014, s3015, _MM_SHUFFLE(3, 1, 3, 1));
s3024 = _mm_shuffle_ps(s3016, s3017, _MM_SHUFFLE(2, 0, 2, 0));
s3025 = _mm_shuffle_ps(s3016, s3017, _MM_SHUFFLE(3, 1, 3, 1));
s3026 = _mm_shuffle_ps(s3018, s3019, _MM_SHUFFLE(2, 0, 2, 0));
s3027 = _mm_shuffle_ps(s3018, s3019, _MM_SHUFFLE(3, 1, 3, 1));
s3028 = _mm_shuffle_ps(s3020, s3021, _MM_SHUFFLE(2, 0, 2, 0));
s3029 = _mm_shuffle_ps(s3020, s3021, _MM_SHUFFLE(3, 1, 3, 1));
...
t3794 = _mm_add_ps(s3042, s3043);
t3795 = _mm_add_ps(s3038, t3793);
t3796 = _mm_add_ps(s3041, t3794);
t3797 = _mm_sub_ps(s3038, _mm_mul_ps(_mm_set1_ps(0.5), t3793));
t3798 = _mm_sub_ps(s3041, _mm_mul_ps(_mm_set1_ps(0.5), t3794));
s3044 = _mm_mul_ps(_mm_set1_ps(0.8660254037844386), _mm_sub_ps(s3042, s3043));
s3045 = _mm_mul_ps(_mm_set1_ps(0.8660254037844386), _mm_sub_ps(s3039, s3040));
t3799 = _mm_add_ps(t3797, s3044);
t3800 = _mm_sub_ps(t3798, s3045);
t3801 = _mm_sub_ps(t3797, s3044);
t3802 = _mm_add_ps(t3798, s3045);
a773 = _mm_mul_ps(_mm_set_ps(0, 0, 0, 1), _mm_shuffle_ps(s3013, a772, _MM_SHUFFLE(2, 0, 2, 0)));
t3803 = _mm_add_ps(a773, _mm_mul_ps(_mm_set_ps(0, 0, 0, 1), t3795));
a774 = _mm_mul_ps(_mm_set_ps(0, 0, 0, 1), _mm_shuffle_ps(s3013, a772, _MM_SHUFFLE(3, 1, 3, 1)));
t3804 = _mm_add_ps(a774, _mm_mul_ps(_mm_set_ps(0, 0, 0, 1), t3796));
t3805 = _mm_add_ps(a773, _mm_add_ps(_mm_mul_ps(_mm_set_ps(0.28757036473700154, 0.30046260628866578, (-0.28757036473700154), (-0.08333333333333
t3806 = _mm_add_ps(a774, _mm_add_ps(_mm_mul_ps(_mm_set_ps(0.08706930057606789, 0, 0.087069300576068001, 0), t3795), _mm_mul_ps(_mm_set_ps(0.287
s3046 = _mm_sub_ps(_mm_mul_ps(_mm_set_ps((-0.25624767158293649), 0.25826039031174486, (-0.30023863596633249), 0.07590298603719387 9), t3799), _mm
s3047 = _mm_add_ps(_mm_mul_ps(_mm_set_ps(0.15689139105158462, (-0.15355568557954136), (-0.011599105605768193), 0.29071724147084099), t3799), _mm
```



Outline

- Introduction
- **Formal Proof/Code Co-Synthesis**
- Achieving Performance Portability
- Hiding complexity from users
- Summary

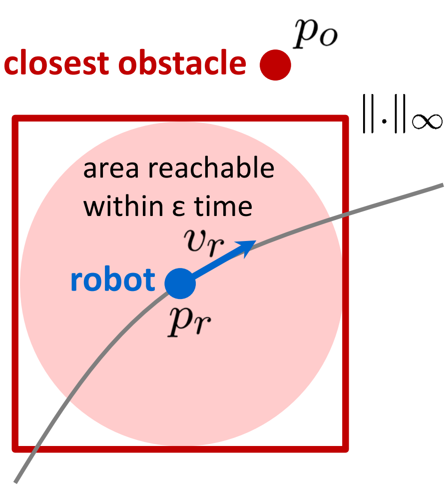
F. Franchetti, T. M. Low, S. Mitsch, J. P. Mendoza, L. Gui, A. Phaosawasdi, D. Padua, S. Kar, J. M. F. Moura, M. Franusich, J. Johnson, A. Platzer, and M. Veloso: High-Assurance SPIRAL: End-to-End Guarantees for Robot and Car Control, IEEE Control Systems Magazine, 2017, pages 82-103.

Problem Setup: Robot/Car Safety Monitor

Equations of Motion

$$v_r = \dot{x}, \quad 0 \le v_r \le V$$
$$a = \dot{v}_r, \quad -b \le a \le A$$

Safety condition



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

$$v_r = \dot{x}, \quad 0 \le v_r \le V$$
$$a = \dot{v}_r, \quad -b \le a \le A$$

KeYmaera
Hybrid Theorem Prover

QED.

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

HA Spiral
Code Synthesis

Coq
Proof Assistant

QED.

```
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);
    ...
```

performance

André Platzer and Jan-David Quesel, **"KeYmaera: A hybrid theorem prover for hybrid systems,"** *IJCAR 2008*.

Stefan Mitsch, Khalil Ghorbal, André Platzer, **"On Provably Safe Obstacle Avoidance for Autonomous Robotic Ground Vehicles,"** *RSS 2013*.

Approach

- **Develop synthesis framework**
  operator language (OL), rewriting system
- **Formalize algorithms**
  express algorithm as OL rewrite rules
- **Provide performance portability**
  optimization through rewriting
- **Correctness proof**
  rule chain = formal proof,
  symbolic evaluation and execution

Formal framework

Algorithm formalization

Platform optimization

Formal proof

OL Operators

Definition

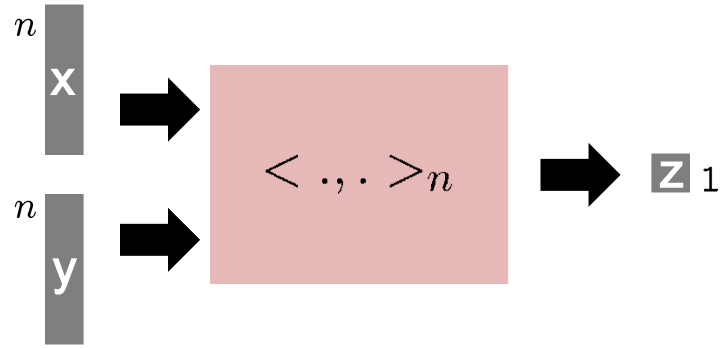
- Operator: Multiple vectors ! Multiple vectors
- Stateless
- Higher-dimensional data is linearized
- Operators are potentially nonlinear

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

Example: Scalar product

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

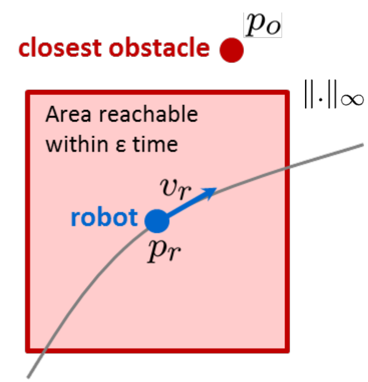
$$\big((x_i)_{i=0,\ldots,n-1}, (y_i)_{i=0,\ldots,n-1}\big) \mapsto \sum_{i=0}^{n-1} x_i y_i$$



F. Franchetti, F. de Mesmay, Daniel McFarlin, and M. Püschel, **"Operator Language: A Program Generation Framework for Fast Kernels,"** Proceedings of IFIP Working Conference on Domain Specific Languages (DSL WC), 2009. ***Best Paper Award.***

Safety Distance as OL Operator

- **Safety constraint from KeYmaera**
  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**

$$\begin{aligned} \mathsf{SafeDist}_{V,A,b,\varepsilon} : \quad & \mathbb{R} \times \mathbb{R}^2 \times \mathbb{R}^2 \to \mathbb{Z}_2 \\ & (v_r, p_r, p_o) \mapsto \big(p(v_r) < d_\infty(p_r, p_o)\big) \end{aligned}$$

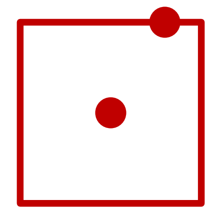
with

$$\begin{aligned} d_\infty(\vec{x}, \vec{y}) &= \|\vec{x} - \vec{y}\|_\infty \\ p(x) &= \alpha x^2 + \beta x + \gamma \\ \alpha &= \tfrac{1}{2b} \\ \beta &= \tfrac{V}{b} + \varepsilon\left(\tfrac{A}{b} + 1\right) \\ \gamma &= \left(\tfrac{A}{b} + 1\right)\left(\tfrac{A}{2}\varepsilon^2 + \varepsilon V\right) \end{aligned}$$

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**
$$(x^i)_n : \mathbb{R} \to \mathbb{R}^{n+1}$$
$$x \mapsto (x^i)_{i=0,\ldots,n}$$

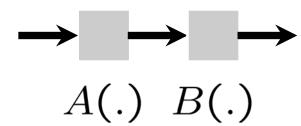
- **Polynomial evaluation**
$$P[x, (a_0, \ldots, a_n)] : \mathbb{R} \to \mathbb{R}$$
$$x \mapsto a_0 x^n + a_1 x^{n-1} + \cdots + a_{n-1} x + a_n$$

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

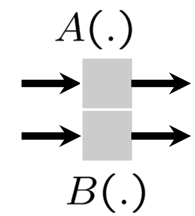
Operations and Operator Expressions

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

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

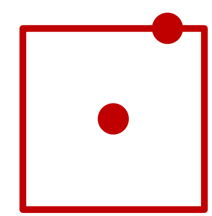


$$\times : (D \to R) \times (E \to S) \to (D \times E \to R \times S)$$
$$(A, B) \mapsto \big((x, y) \mapsto \big(A(x), B(y)\big)\big)$$



- **Operator expressions are operators**

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

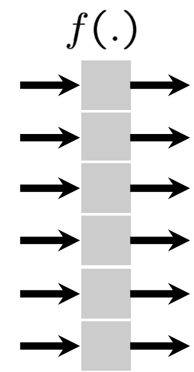


- **Short-hand notation: Infix notation**

$$A(.) - B(.) = \big(x \mapsto A(x) - B(x)\big)$$ 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**

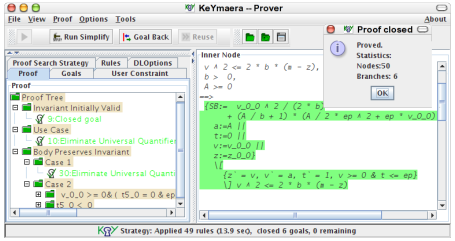
$$\begin{aligned}\mathsf{SafeDist}_{V,A,b,\varepsilon} = {} & \mathsf{Atomic}_{(x,y)\mapsto x<y} \\ & \circ \Big(\Big(\mathsf{Reduction}_{3,(x,y)\mapsto x+y} \circ \mathsf{Pointwise}_{3,x\mapsto a_i x} \circ \mathsf{Induction}_{3,(a,b)\mapsto ab,1}\Big) \\ & \quad \times \Big(\mathsf{Reduction}_{2,(x,y)\mapsto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

Breaking Down Operators into Expressions

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

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

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



Problem specification produced by KeYmaera theorem prover

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

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

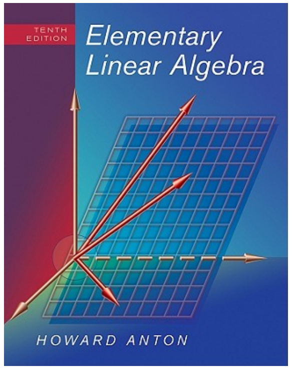
$$(\diamond)_n \to \mathsf{Pointwise}_{n\times n,(a,b)\mapsto a\diamond b},\quad \diamond \in \{+,-\cdot,\wedge,\vee,\ldots\}$$

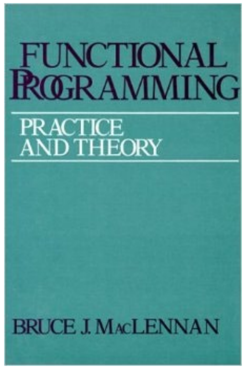
$$\|.\|_\infty^n \to \mathsf{Reduction}_{n,(a,b)\mapsto \max(|a|,|b|)}$$

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

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

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





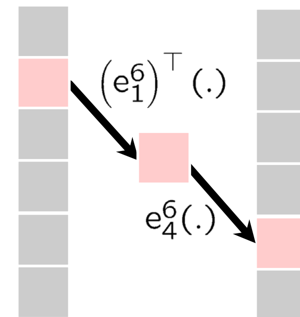
Library of well-known identities expressed in OL

Σ–OL: Low-Level Operator Language

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

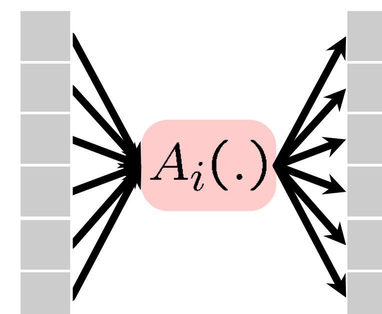
$$\left(\mathsf{e}_i^n\right)^\top(.) : \mathbb{R}^n \to \mathbb{R}^1$$
$$(x_i)_{i=0,\ldots,n-1} \mapsto x_i$$

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



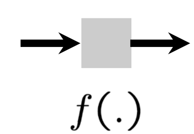
- **Iterative operations:** *loop*

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



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

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



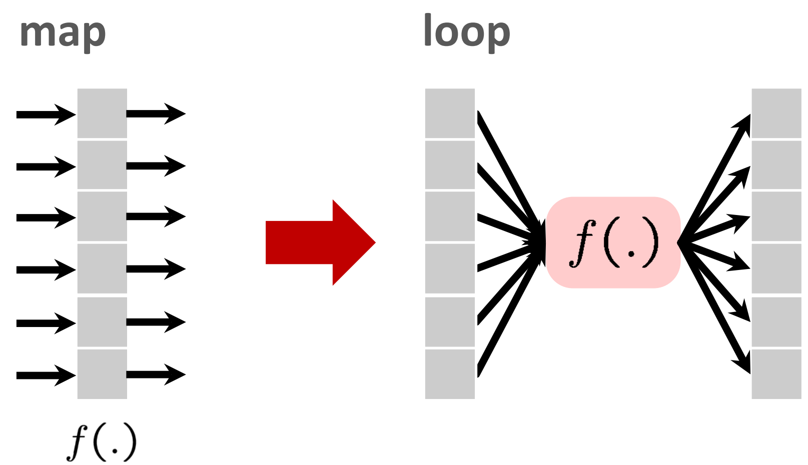
Σ-OL operator expressions = array-based programs with for loops

Rule-Based Translation and Optimization

- **Translating Basic OL into Σ-OL**

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

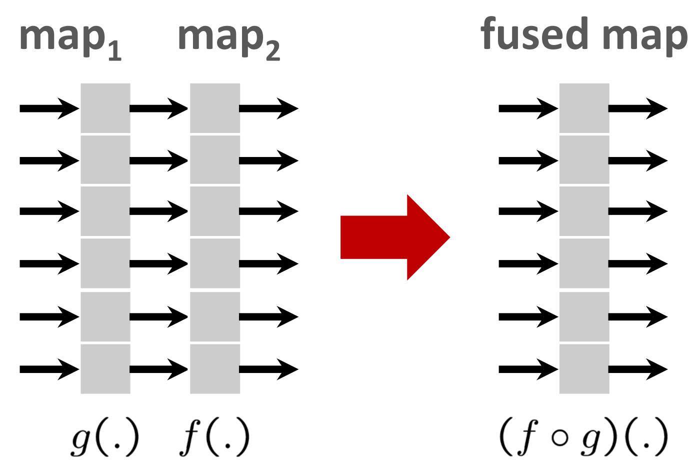
$$\mathsf{Reduction}_{n,(a,b)\mapsto a+b} \rightarrow \sum_{i=0}^{n-1} (\mathsf{e}_i^n)^\top$$



- **Optimizing Basic OL/Σ-OL**

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

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



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

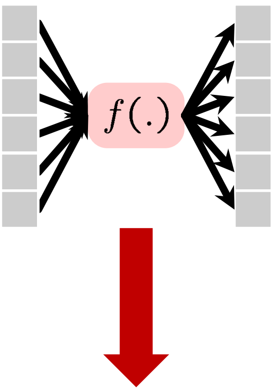
$$\mathsf{Code}\big(y = (A \circ B)(x)\big) \;\rightarrow\; \big\{\mathtt{decl}(t), \mathsf{Code}\big(t = B(x)\big), \mathsf{Code}\big(y = A(t)\big)\big\}$$

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

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

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

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




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

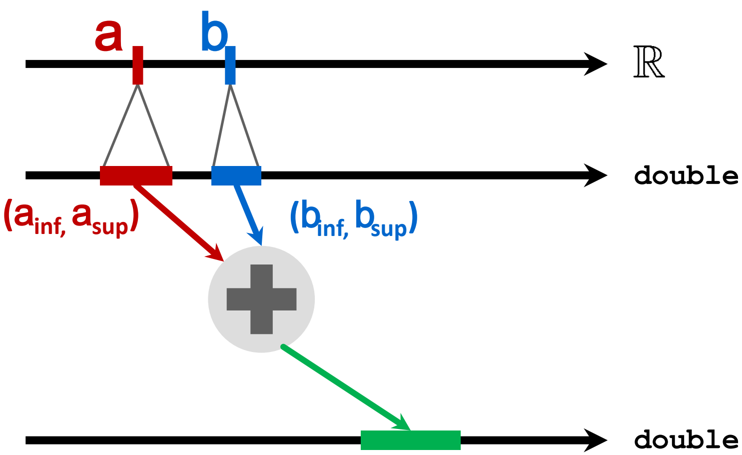
Cleanup rules: term rewriting

```
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

Sound Floating-Point Arithmetic

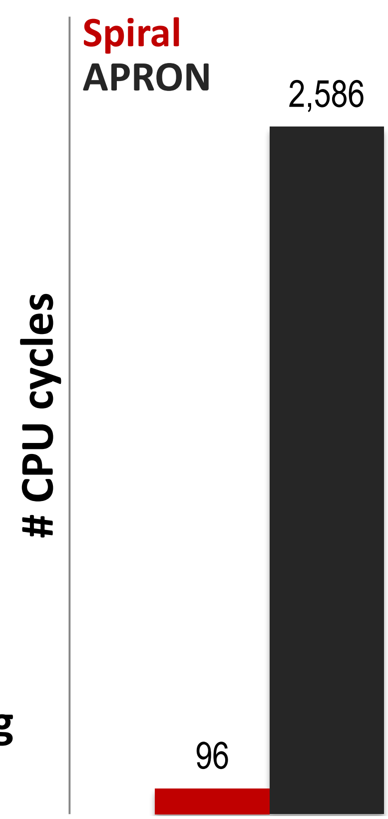
Interval Arithmetic



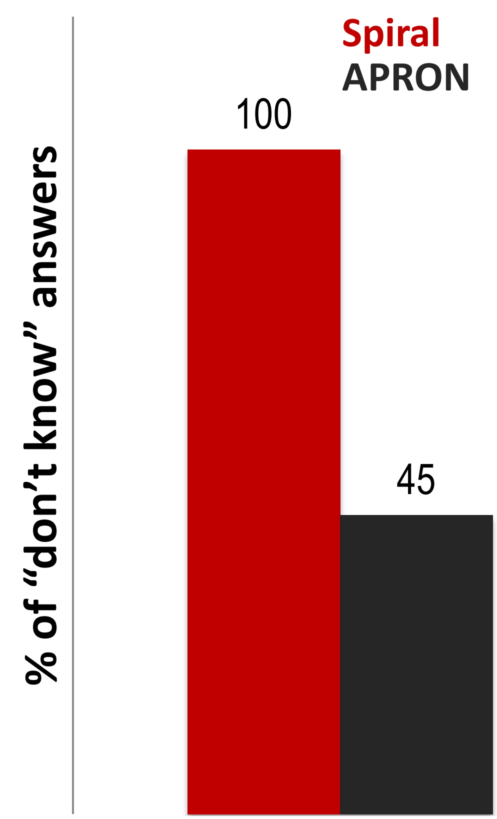
$(c_{inf}, c_{sup}) =$ (`rounddown`$(a_{inf}+b_{inf})$, `roundup`$(a_{inf}+b_{sup})$)

- **Standard implementation very slow**
  rounding mode change, IEEE754 denormals
- **Efficient implementation is challenging**
  lots of effort: 10x overhead over scalar
- **Application has headroom**
  only `float` required
- **Numerically ok for application**
  no iterations, ill-conditioning etc.

Performance



Precision at boundary



a < b ? for
$a - b \approx 10^{-15}$

SandyBridge CPU, Intel C Compiler,
SPIRAL vs. APRON Interval Arithmetic Library

Putting it Together: One Big Rule System

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

Mathematical specification

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

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

```c
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);
    }
    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;
}
```

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

Formal Proof: Specification = Code

Mathematical equivalency

- **Code over *reals* = specification**
  code as operator =
  specification as operator
- **Rules are mathematical identities**
  all objects have mathematical semantics
- **Synthesis is semantics preserving**
  chain of semantics-preserving rules

Floating-point: interval arithmetic

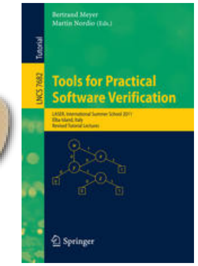
- **Numerical results are sound**
  true answer guaranteed in result interval
- **Logical answers are sound**
  conservative: true/false/unknown
- **Floating-point uncertainty**
  operators over random variables

```
<800 lines>
-------------------------------------------------------------------
RULE: eT_Pointwise
-------------------------------------------------------------------
old expression
-------------------------------------------------------------------
eT(3, i17) o
PointWise(3, Lambda([ r16, i14 ], mul(r16, nth(D, i14)))) o
Induction(3, Lambda([ r9, r10 ], mul(r9, r10)), V(1.0)) o
eT(5, 0)
-------------------------------------------------------------------
new expression
-------------------------------------------------------------------
PointWise(1, Lambda([ r16, i19 ], mul(r16, nth(D, i17)))) o
eT(3, i17) o
Induction(3, Lambda([ r9, r10 ], mul(r9, r10)), V(1.0)) o
eT(5, 0)
-------------------------------------------------------------------

-------------------------------------------------------------------
RULE: eT_Induction
-------------------------------------------------------------------
old expression
-------------------------------------------------------------------
PointWise(1, Lambda([ r16, i19 ], mul(r16, nth(D, i17)))) o
eT(3, i17) o
Induction(3, Lambda([ r9, r10 ], mul(r9, r10)), V(1.0)) o
eT(5, 0)
-------------------------------------------------------------------
new expression
-------------------------------------------------------------------
PointWise(1, Lambda([ r16, i19 ], mul(r16, nth(D, i17)))) o
Inductor(3, i17, Lambda([ r9, r10 ], mul(r9, r10)), V(1.0)) o
eT(5, 0)
-------------------------------------------------------------------
<10500 lines>
```

Rewrite rule:

$$(e_n^j)^\top \circ \mathrm{Pointwise}_{n,f_i} \to \mathrm{Pointwise}_{1,f_j} \circ (e_n^j)^\top$$

Ongoing: Mechanical Proof in Coq

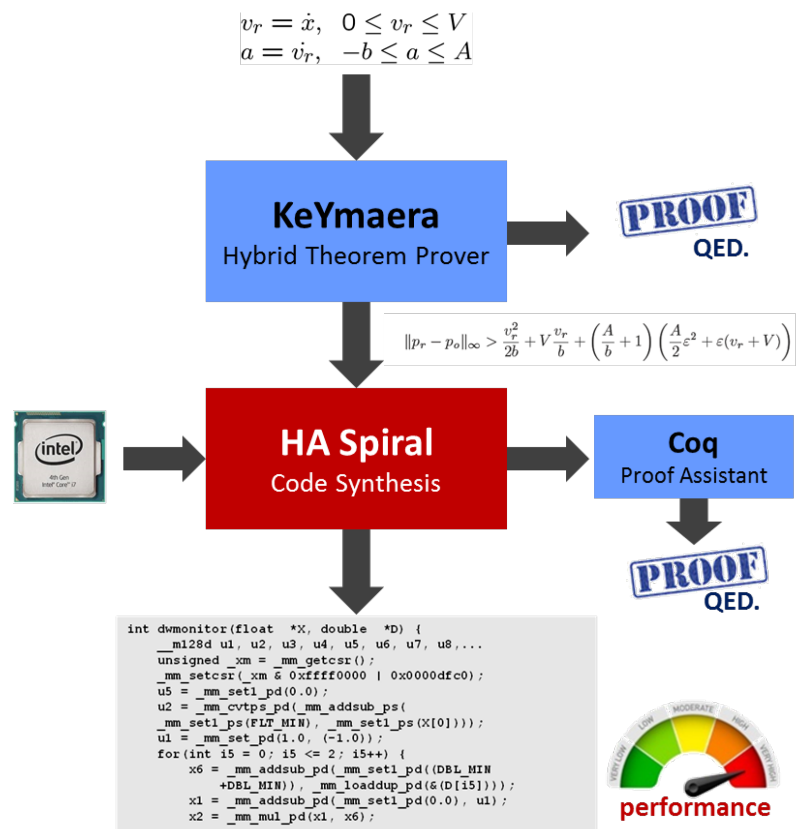


Mechanical proof of equivalency

- **Coq proof assistant**
  Calculus of Inductive Constructions
- **Rewrite rules are lemmas**
  express all objects and rules in Coq
- **Theorem: specification = code**
  prove theorem via lemmas

Backend C compiler

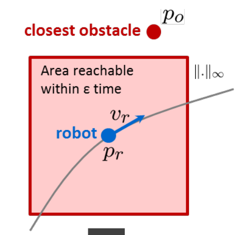
- **CompCert: certified C compiler**
  guarantees executable = source code
- **But: no SSE/AVX instructions**
  can be extended
- **Issue: performance of generated code**
  validate translation of Intel C compiler?



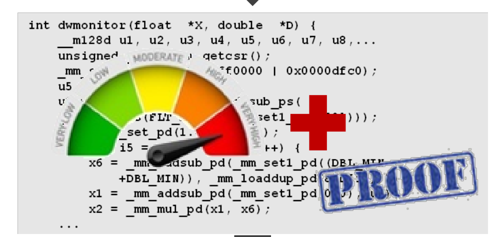
V. Zaliva, F. Franchetti: HELIX: A Case Study of a Formal Verification of High Performance Program Generation
Workshop on Functional High Performance Computing (FHPC), 2018

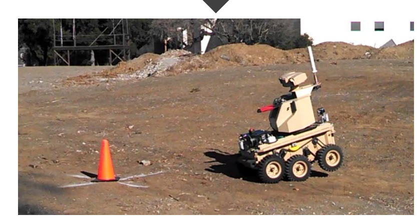
What have we achieved?

- **Formalization of mathematical objects**
  Operator Language (OL), specification with semantics
- **Formal code synthesis**
  rewriting system: OL semantics-preserving rules
- **Loop-level and code-level optimizations**
  rewriting system: abstract code, Σ-OL semantics-preserving rules
- **Floating-point: soundness of computation**
  High performance interval arithmetic
- **Formal proof of code synthesis**
  rewriting trace establishes proof of equivalency



HA Spiral
Code Synthesis





Synthesis = Sequence of mathematical identity transformations

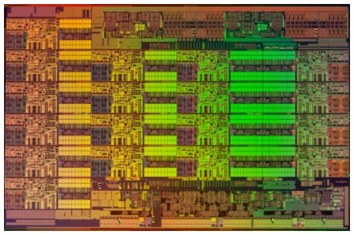
Outline

- Introduction
- Formal Proof/Code Co-Synthesis
- **Achieving Performance Portability**
- Hiding complexity from users
- Summary

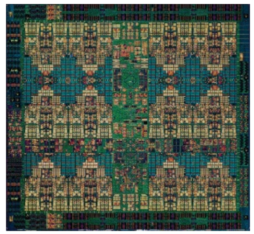
F. Franchetti, T. M. Low, D. T. Popovici, R. M. Veras, D. G. Spampinato, J. R. Johnson, M. Püschel, J. C. Hoe, J. M. F. Moura:
SPIRAL: Extreme Performance Portability, **Proceedings of the IEEE, Vol. 106, No. 11, 2018.**
Special Issue on *From High Level Specification to High Performance Code*

Today’s Computing Landscape

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

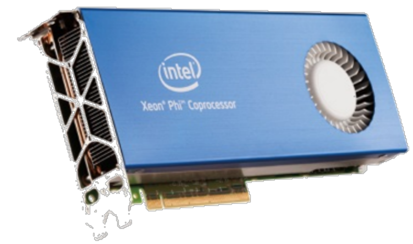


Intel Xeon 8180M
2.25 Tflop/s, 205 W
28 cores, 2.5—3.8 GHz
2-way—16-way AVX-512



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

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



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

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

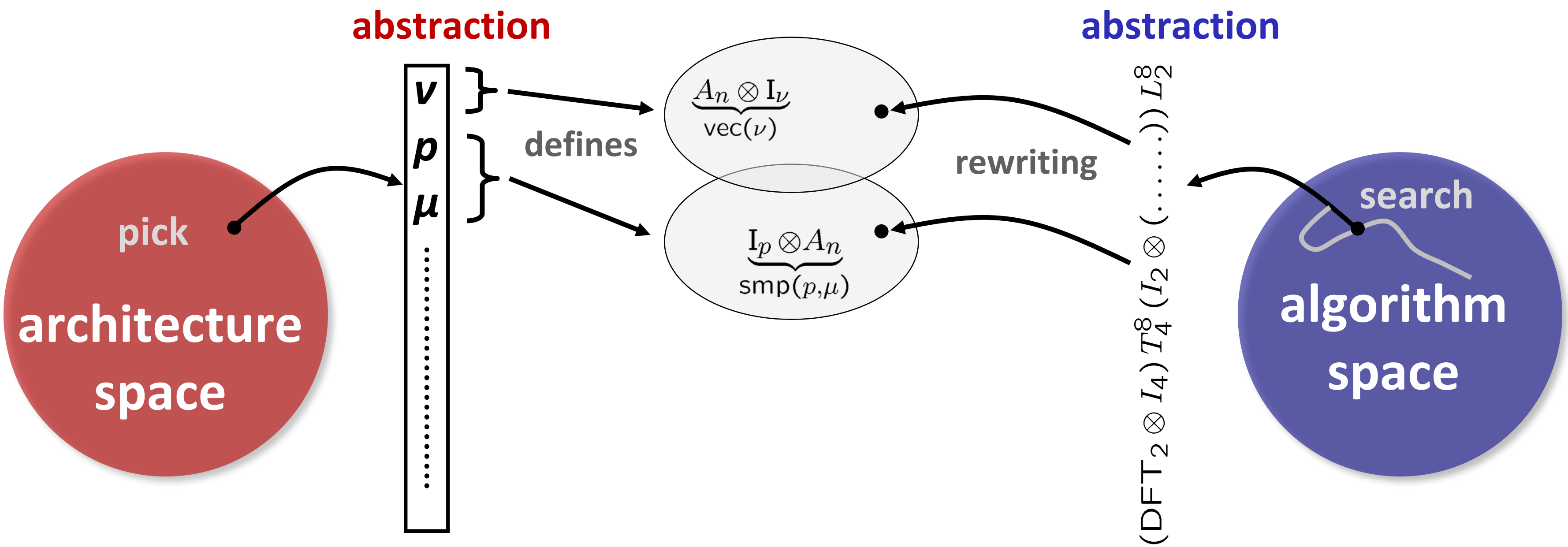
Intel Atom C3858
32 Gflop/s, 25 W
16 cores, 2.0 GHz
2-way/4-way SSSE3

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

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

Platform-Aware Formal Program Synthesis

Model: common abstraction
= spaces of matching formulas



Architectural parameter:
Vector length,
#processors, …

optimization

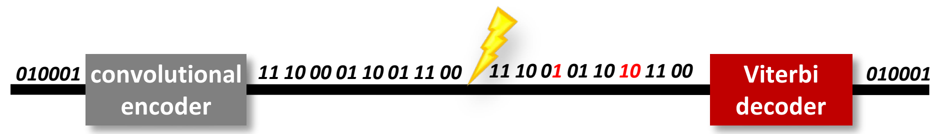
Kernel:
problem size,
algorithm choice

Some Application Domains in OL

Linear Transforms

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

Software Defined Radio



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

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

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

Matrix-Matrix Multiplication

$$\mathrm{MMM}_{1,1,1} \to (\cdot)_1$$

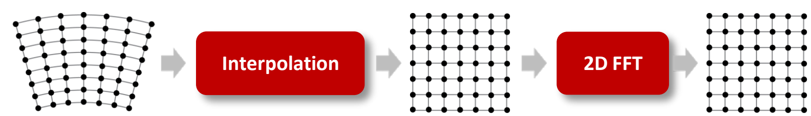
$$\mathrm{MMM}_{m,n,k} \to (\otimes)_{m/m_b \times 1} \otimes \mathrm{MMM}_{m_b,n,k}$$

$$\mathrm{MMM}_{m,n,k} \to \mathrm{MMM}_{m,nb,k} \otimes (\otimes)_{1 \times n/nb}$$

$$\mathrm{MMM}_{m,n,k} \to ((\Sigma_{k/k_b} \circ (\cdot)_{k/k_b}) \otimes \mathrm{MMM}_{m,n,k_b}) \circ ((L^{mk/k_b}_{k/k_b} \otimes I_{k_b}) \times I_{kn})$$

$$\mathrm{MMM}_{m,n,k} \to (L^{mn/n_b}_{m} \otimes I_{n_b}) \circ ((\otimes)_{1 \times n/n_b} \otimes \mathrm{MMM}_{m,n_b,k}) \circ (I_{km} \times (L^{kn/n_b}_{n/n_b} \otimes I_{n_b}))$$

Synthetic Aperture Radar (SAR)



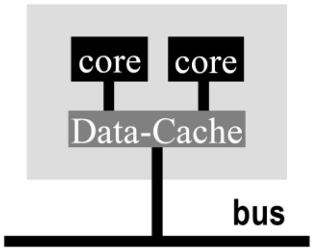
$$
\begin{array}{rcl}
\mathrm{SAR}_{k\times m \to n \times n} & \to & \mathrm{DFT}_{n\times n} \circ \mathrm{Interp}_{k \times m \to n \times n} \\
\mathrm{DFT}_{n\times n} & \to & (\mathrm{DFT}_n \otimes \mathrm{I}_n) \circ (\mathrm{I}_n \otimes \mathrm{DFT}_n) \\
\mathrm{Interp}_{k\times m \to n\times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

Formal Approach for all Types of Parallelism

- **Multithreading** (Multicore) $\quad \mathrm{I}_p \otimes_{\|} A_{\mu n}, \quad \mathbf{L}^{mn}_m \bar{\otimes} \mathrm{I}_\mu$
- **Vector SIMD** (SSE, VMX/Altivec,…) $\quad A \hat{\otimes} \mathrm{I}_\nu \quad \underbrace{\mathbf{L}^{2\nu}_2}_{\text{isa}}, \quad \underbrace{\mathbf{L}^{2\nu}_\nu}_{\text{isa}}, \quad \underbrace{\mathbf{L}^{\nu^2}_\nu}_{\text{isa}}$
- **Message Passing** (Clusters, MPP) $\quad \mathrm{I}_p \otimes_{\|} A_n, \quad \underbrace{\mathbf{L}^{p^2}_p \bar{\otimes} \mathrm{I}_{n/p^2}}_{\text{all-to-all}}$
- **Streaming/multibuffering** (Cell) $\quad \mathrm{I}_n \otimes_2 A_{\mu n}, \quad \mathbf{L}^{mn}_m \bar{\otimes} \mathrm{I}_\mu$
- **Graphics Processors** (GPUs) $\quad \prod_{i=0}^{n-1} A_i, \quad A_n \hat{\otimes} \mathrm{I}_w, \quad P_n \otimes Q_w$
- **Gate-level parallelism** (FPGA) $\quad \prod_{i=0}^{n-1}{}^{\text{ir}} A, \quad \mathrm{I}_s \tilde{\otimes} A, \quad \underbrace{\mathbf{L}^m_n}_{\text{bram}}$
- **HW/SW partitioning** (CPU + FPGA) $\quad \underbrace{A_1}_{\text{fpga}}, \quad \underbrace{A_2}_{\text{fpga}}, \quad \underbrace{A_3}_{\text{fpga}}, \quad \underbrace{A_4}_{\text{fpga}}$

Modeling Hardware: Base Cases

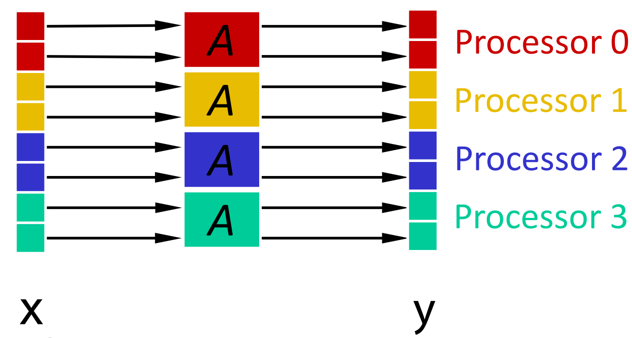
- **Hardware abstraction: shared cache with cache lines**



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

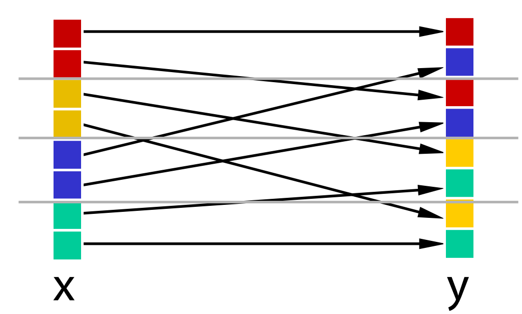
- **Tensor product: embarrassingly parallel operator**

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



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

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



Example Program Transformation Rule Set

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

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

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

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`

Base cases

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

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

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

Transformation rules

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

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

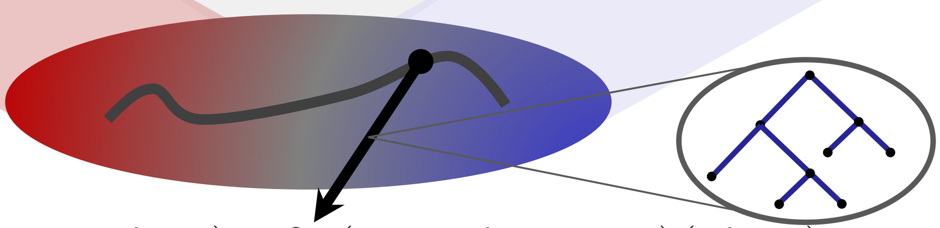
$$A^{m\times m}\otimes\mathrm{I}_n\rightarrow(A^{m\times m}\otimes\mathrm{I}_{n/\nu})\otimes\mathrm{I}_\nu$$

DFT$_8$

Breakdown rules

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

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



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

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

See Figure 5

OL Expression:

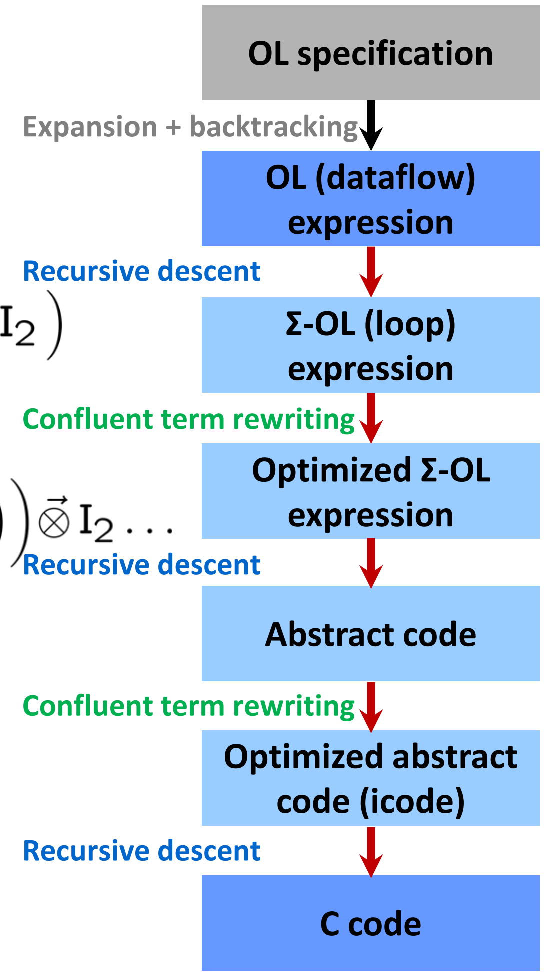
$$\big((\mathrm{F}_2\otimes \mathrm{I}_2)\mathrm{T}^4_2(\mathrm{I}_2\otimes \mathrm{F}_2)\mathrm{L}^4_2\vec{\otimes}\, \mathrm{I}_2\big)\ \underbrace{\mathrm{T}^8_2}_{\mathrm{vec}(2)}\ \big(\mathrm{I}_2\otimes\ \underbrace{\mathrm{L}^4_2}_{\mathrm{vec}(2)}\ (\mathrm{F}_2\vec{\otimes}\, \mathrm{I}_2)\big)\big(\mathrm{L}^4_2\vec{\otimes}\,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

Σ-OL:

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

C Code:

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



Symbolic Verification for Linear Operators

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

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

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

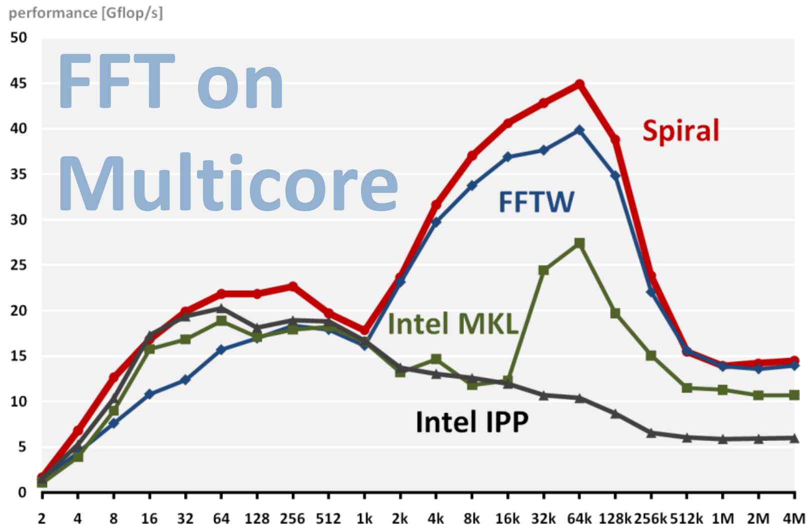
- **Linear operator = matrix-vector product**
  Program = matrix-vector product

$$\begin{bmatrix}1 & 1 & 1 & 1\\ 1 & j & -1 & -j\\ 1 & -1 & 1 & -1\\ 1 & -j & -1 & j\end{bmatrix} \cdot \begin{bmatrix}0\\ 1\\ 0\\ 0\end{bmatrix} \quad = ? \quad \texttt{DFT4([0,1,0,0])}$$

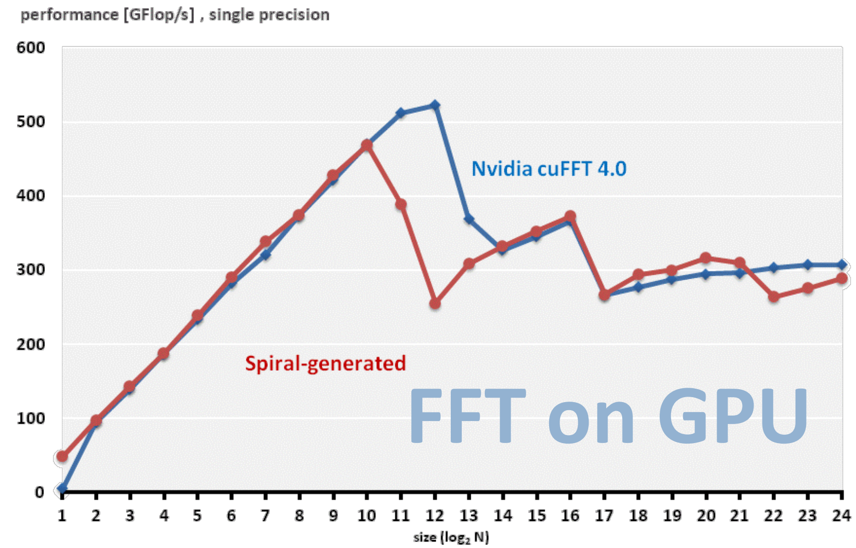
Symbolic evaluation and symbolic execution establishes correctness

Synthesis: FFTs and Spectral Algorithms

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

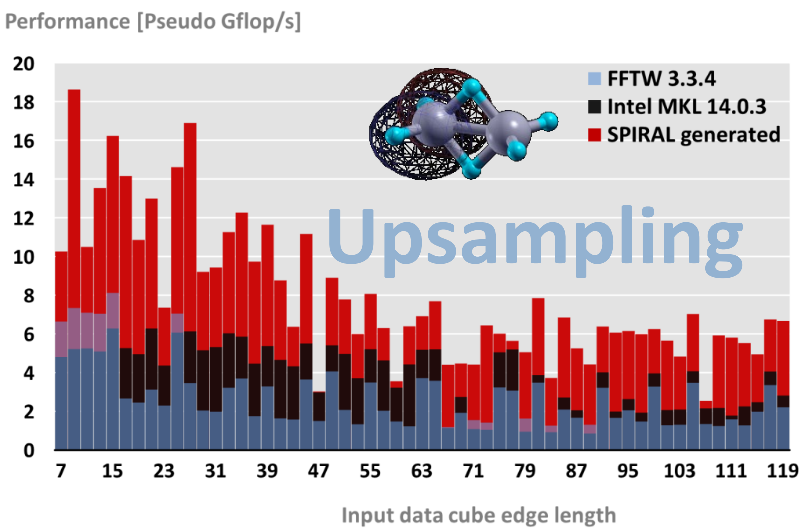


1D Batch DFT (Nvidia GTX 480)

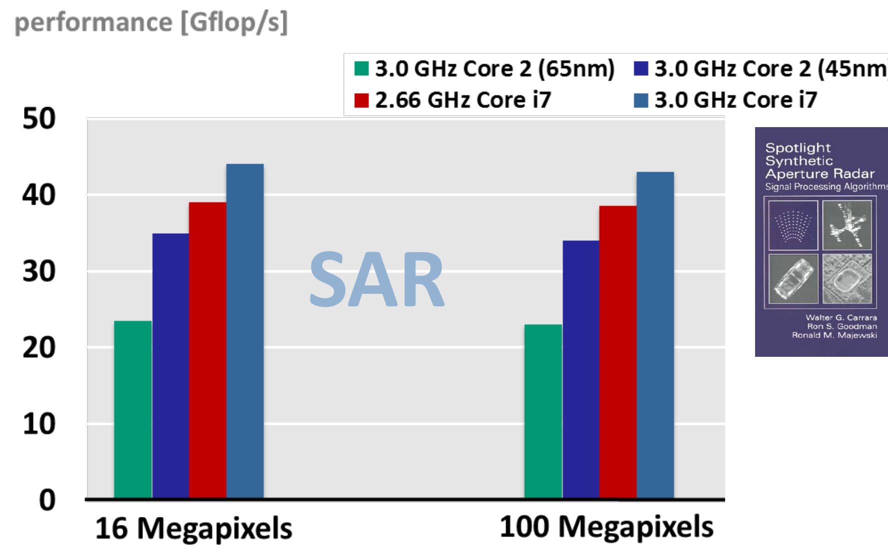


Performance of 2x2x2 Upsampling on Haswell

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

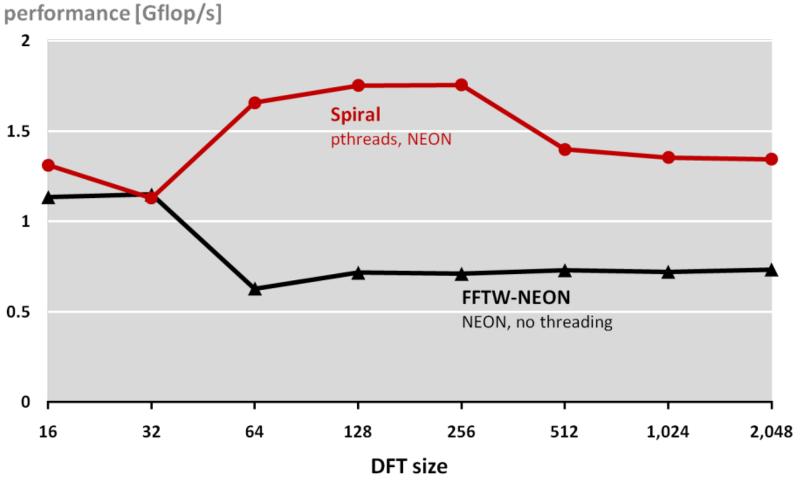


PFA SAR Image Formation on Intel platforms

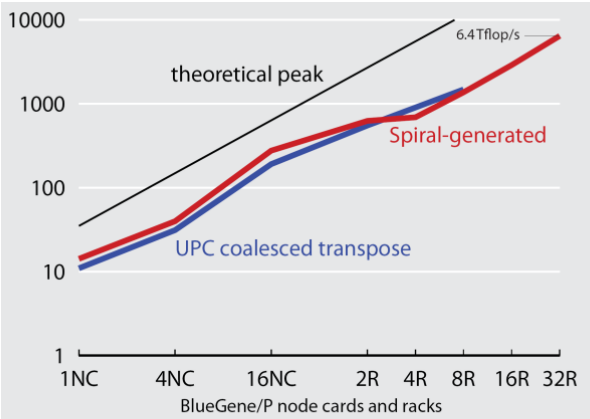


From Cell Phone To Supercomputer

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



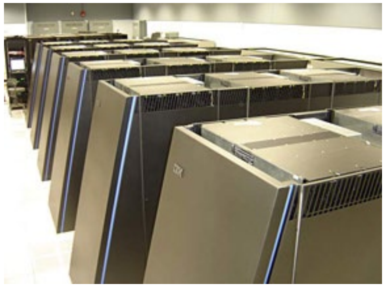
Global FFT (1D FFT, HPC Challenge)
performance [Gflop/s]



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).***

Outline

- Introduction
- Formal Proof/Code Co-Synthesis
- Achieving Performance Portability
- **Hiding complexity from users**
- Summary

Have You Ever Wondered About This?

Numerical Linear Algebra

LAPACK
ScaLAPACK
LU factorization
Eigensolves
SVD

BLAS, BLACS
BLAS-1
BLAS-2
BLAS-3

Spectral Algorithms

Convolution
Correlation
Upsampling
Poisson solver
…

?

FFTW
DFT, RDFT
1D, 2D, 3D,…
batch

No LAPACK equivalent for spectral methods

- **Medium size 1D FFT (1k—10k data points) is most common library call**
  applications break down 3D problems themselves and then call the 1D FFT library
- **Higher level FFT calls rarely used**
  FFTW *guru* interface is powerful but hard to used, leading to performance loss
- **Low arithmetic intensity and variation of FFT use make library approach hard**
  Algorithm specific decompositions and FFT calls intertwined with non-FFT code

FFTX and SpectralPACK: Long Term Vision

Numerical Linear Algebra

LAPACK
LU factorization
Eigensolves
SVD
…

BLAS
BLAS-1
BLAS-2
BLAS-3

Spectral Algorithms

SpectralPACK
Convolution
Correlation
Upsampling
Poisson solver
…

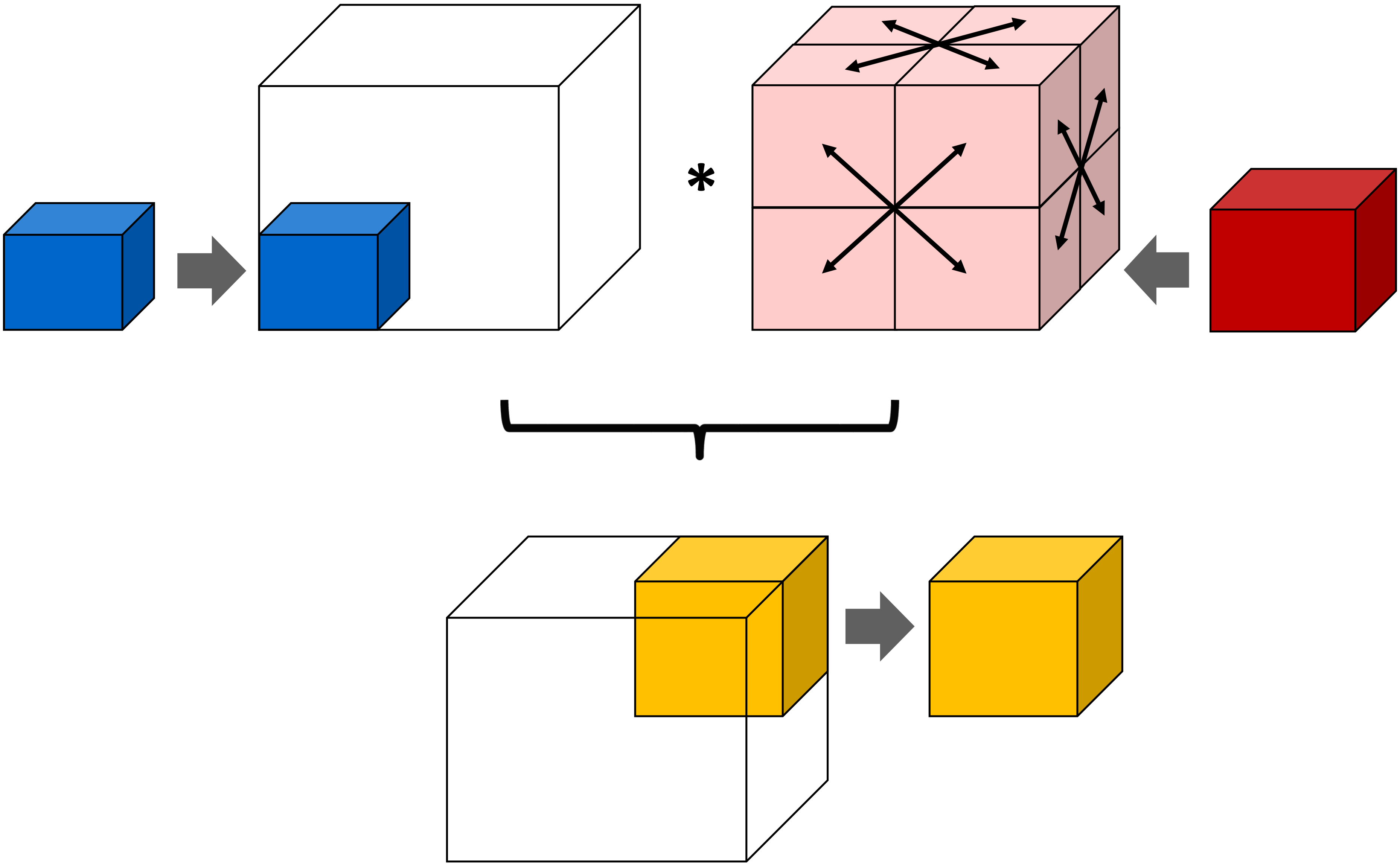
FFTX
DFT, RDFT
1D, 2D, 3D,…
batch

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,…
- **Define SpectralPACK functions**
  circular convolutions, NUFFT, Poisson solvers, free space convolution,…

FFTX and SpectralPACK solve the "spectral dwarf" long term

Example: Hockney Free Space Convolution

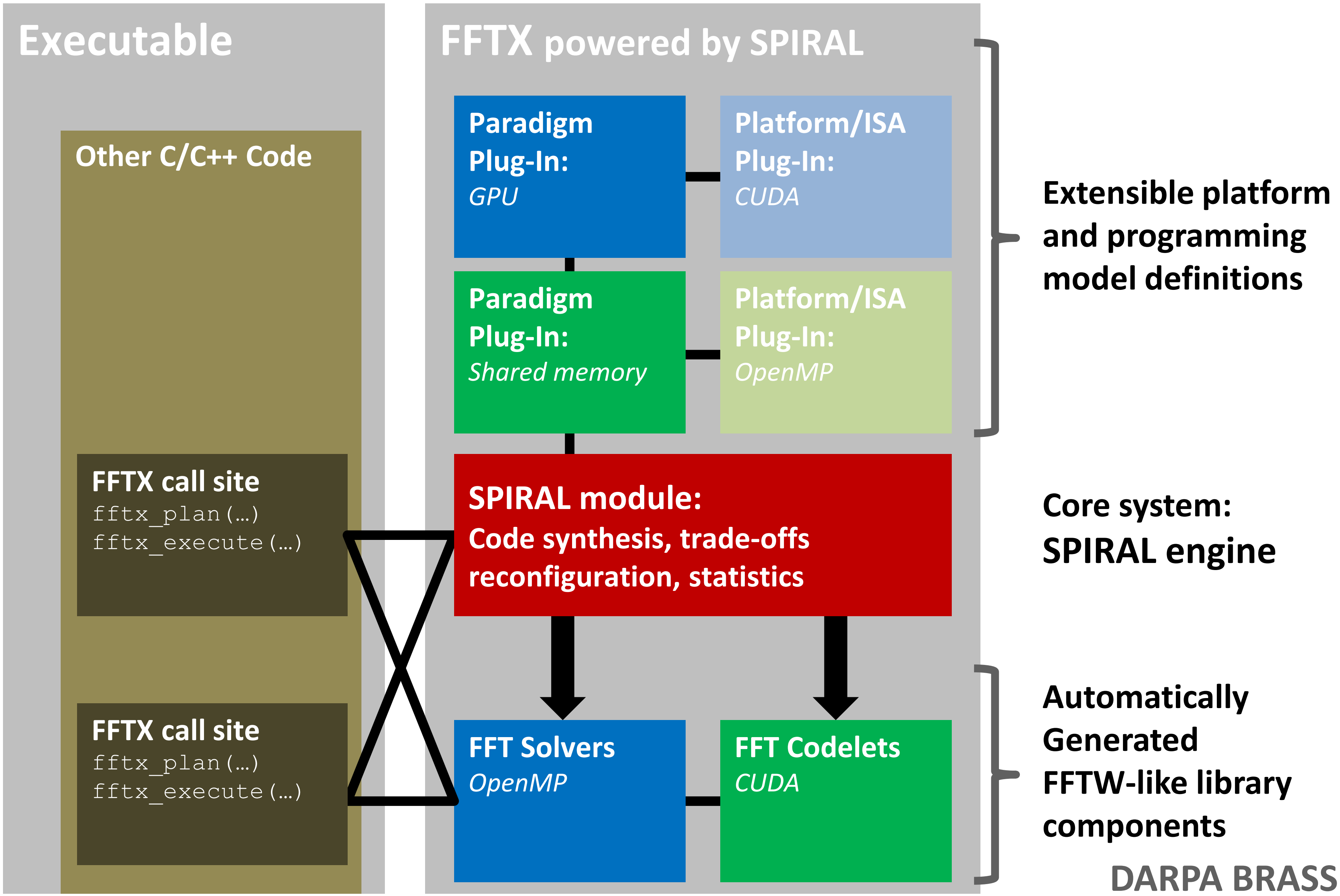


Example: Hockney Free Space Convolution

```
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 = 1,
    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



Generated Code For Hockney Convolution

```
void ioprunedconv_130_0_62_72_130(double  *Y, double  *X, double * S) {
    static double D84[260] = {65.5, 0.0, (-0.5000000000001132), (-20.686114762237267),
      (-0.500000000000081), (-10.337014680426078), (-0.5000000000000455),
    ...
for(int i18899 = 0; i18899 <= 1; i18899++) {
    for(int i18912 = 0; i18912 <= 4; i18912++) {
        a9807 = ((2*i18899) + (4*i18912));
        a9808 = (a9807 + 1);
        a9809 = (a9807 + 52);
        a9810 = (a9807 + 53);
        a9811 = (a9807 + 104);
        a9812 = (a9807 + 105);
        s3295 = (*((X + a9807)) + *((X + a9809)) + *((X + a9811)));
        s3296 = (*((X + a9808)) + *((X + a9810)) + *((X + a9812)));
        s3297 = (((0.3090169943749474**((X + a9809))
            - (0.8090169943749474**((X + a9811)))) + *((X + a9807)));
        s3298 = (((0.3090169943749474**((X + a9810))
            - (0.8090169943749474**((X + a9812)))) + *((X + a9808)));
        s3299 = (((0.3090169943749474**((X + a9811))
            - (0.8090169943749474**((X + a9809)))) + *((X + a9807)));
        ...
        *((104 + Y + a12569)) = ((s3983 - s3987) + (0.8090169943749474*t6537)
            + (0.5877852522924731*t6538));
        *((105 + Y + a12569)) = (((s3984 - s3988) + (0.8090169943749474*t6538))
            - (0.5877852522924731*t6537));
    }
}
```

FFTX/SPIRAL with OpenACC backend:
15 % faster than cuFFT expert interface

TITAN V

1,000s of lines of code, cross call optimization, etc., transparently used

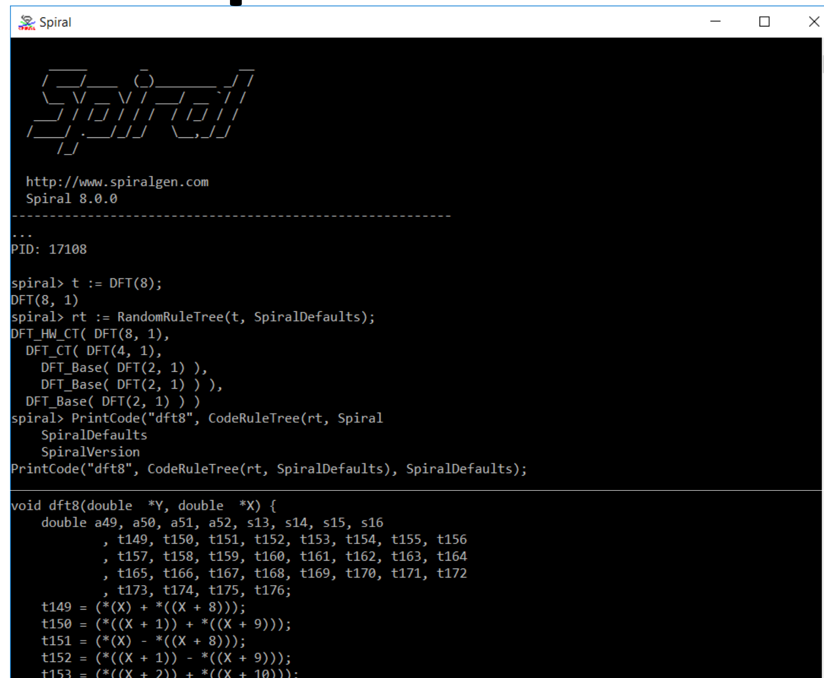
F. Franchetti, D. G. Spampinato, A. Kulkarni, D. T. Popovici, T. M. Low, M. Franusich, A. Canning, P. McCorquodale, B. Van Straalen, P. Colella: **FFTX and SpectralPack: A First Look**, Workshop on Parallel Fast Fourier Transforms (PFFT), *to appear*.

Outline

- Introduction
- Formal Proof/Code Co-Synthesis
- Achieving Performance Portability
- Hiding complexity from users
- **Summary**

SPIRAL 8.0: 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**

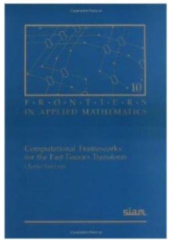


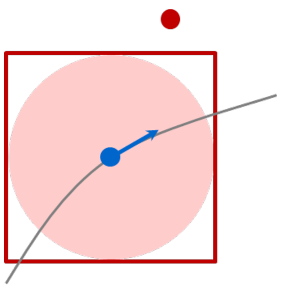
www.spiral.net

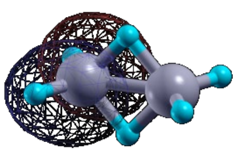
F. Franchetti, T. M. Low, D. T. Popovici, R. M. Veras, D. G. Spampinato, J. R. Johnson, M. Püschel, J. C. Hoe, J. M. F. Moura: SPIRAL: Extreme Performance Portability, **Proceedings of the IEEE, Vol. 106, No. 11, 2018.**
Special Issue on *From High Level Specification to High Performance Code*

Summary: Formal Code Synthesis

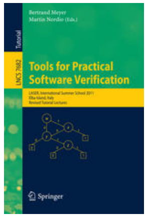
Algorithms

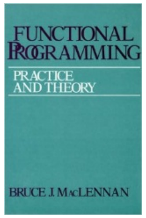


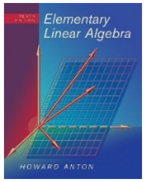




Correctness








```
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