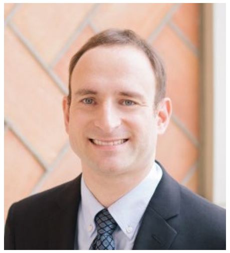
SPIRAL
Automating High Quality Software Production

Tutorial at HPEC 2019

Franz Franchetti
Tze Meng Low
Carnegie Mellon University

Mike Franusich
SpiralGen, Inc.


Franz Franchetti

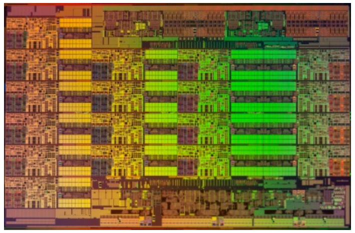
Tze Meng Low

Mike Franusich

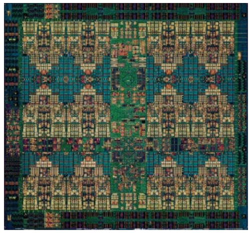
Tutorial based on joint work with the Spiral team at CMU, UIUC, and Drexel

Today’s Computing Landscape

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

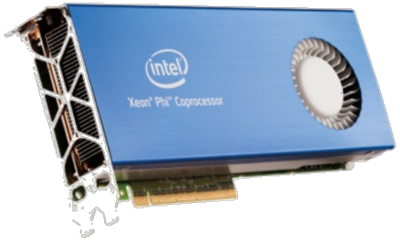


Intel Xeon 8180M
2.25 Tflop/s, 205 W
28 cores, 2.5—3.8 GHz
2-way—16-way AVX-512



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

2019: What $1M Can Buy You

Dell PowerEdge R940

4.5 Tflop/s, 6 TB, 850 W

4x 28 cores, 2.5 GHz

24U rack

7.5kW

<$1M

OSS 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 12TB HDD, 1 kW

1PB raw

Nvidia DGX-1

8x Tesla V100, 3.2 kW

170 Tflop/s, 128 GB

ISAs Longevity and Abstraction Power

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

Compare: Desktop/workstation class CPUs/machines

Assembly code compatible !!

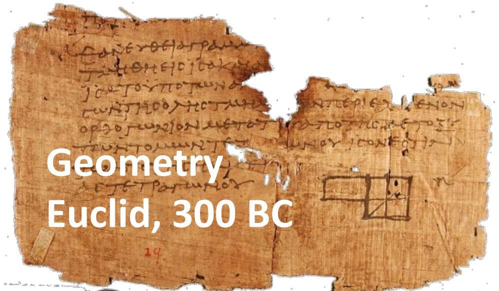
7

x86 binary compatible, but 500x parallelism ?!

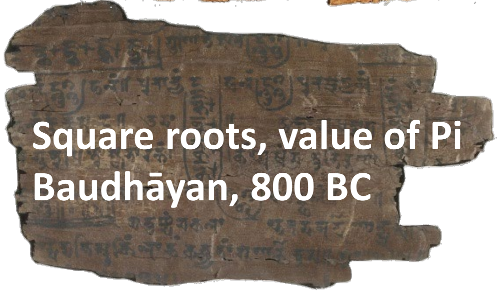
1972	1989	1994	2006	2011	2018
Intel 8008	IBM PC/XT compatible	IBM RS/6000-390	GeForce 8800	Xeon Phi	Xeon Platinum 8180M
0.2—0.8 MHz	8088 @ 8 MHz, 640kB RAM	256 MB RAM, 6GB HDD	1.3 GHz, 128 shaders	1.3 GHz, 60 cores	28 cores, 2.5-3.6 GHz
Intelligent terminal	360 kB FDD, 720x348 mono	67 MHz Power2+, AIX	16-way SIMT	8/16-way SIMD	2/4/8/16-way SIMD

10^7 – 10^8 compounded performance gain over 45 years

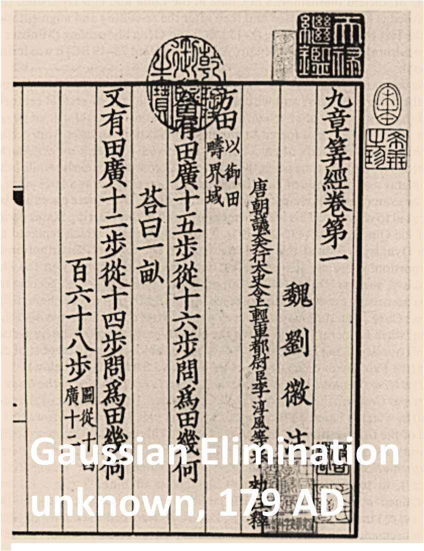
Algorithms and Mathematics: 2,500+ Years



Geometry
Euclid, 300 BC

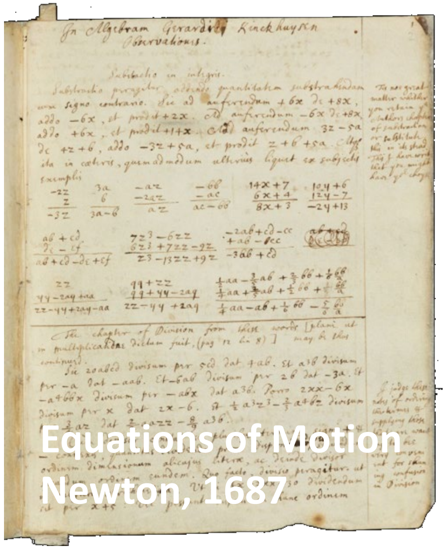


Square roots, value of Pi
Baudhāyan, 800 BC

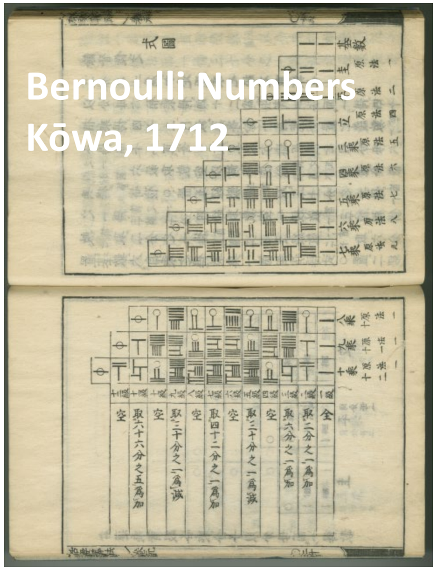


九章筭經卷第一
魏 劉 徽 注

Gaussian Elimination
unknown, 179 AD



Equations of Motion
Newton, 1687

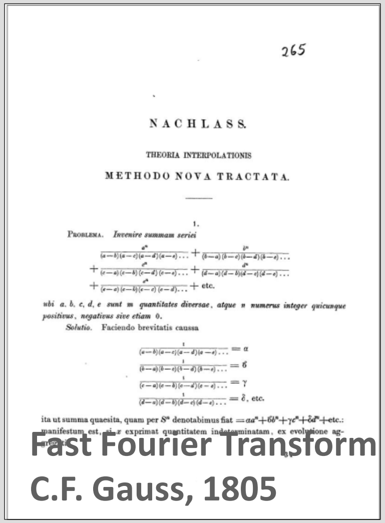


Bernoulli Numbers
Kōwa, 1712

كتاب الخوارزمي

Algebra
al-Khwārizmī, 830

Fast Fourier Transform



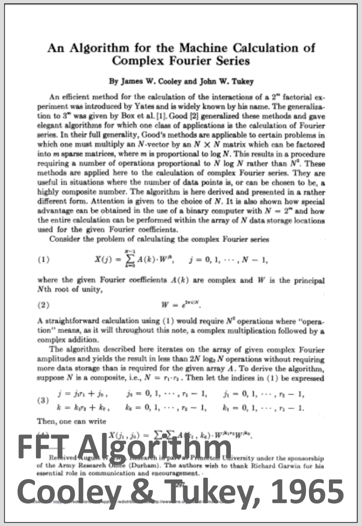
265

NACHLASS.

THEORIA INTERPOLATIONIS

METHODO NOVA TRACTATA.

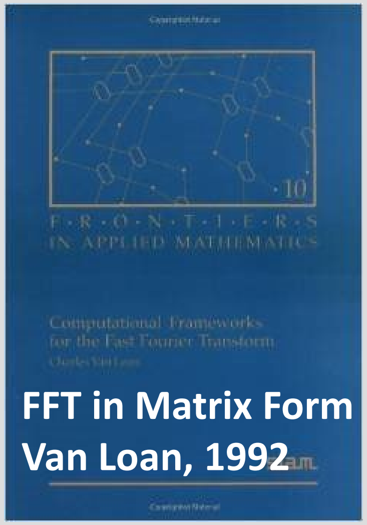
Fast Fourier Transform
C.F. Gauss, 1805



An Algorithm for the Machine Calculation of Complex Fourier Series

By James W. Cooley and John W. Tukey

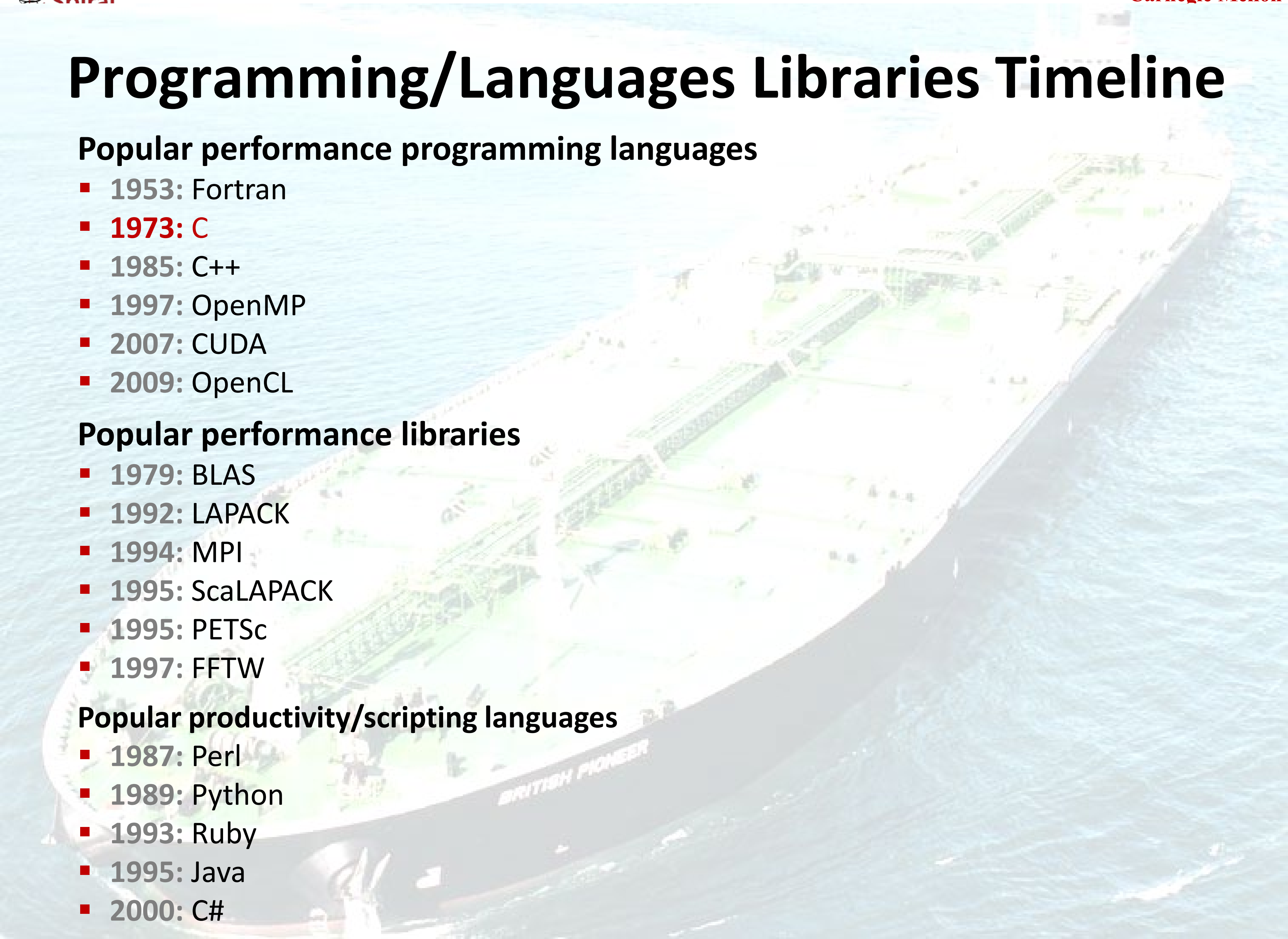
FFT Algorithm
Cooley & Tukey, 1965



10
FRONTIERS IN APPLIED MATHEMATICS
Computational Frameworks for the Fast Fourier Transform
Charles Van Loan

FFT in Matrix Form
Van Loan, 1992

Programming/Languages Libraries Timeline



Popular performance programming languages

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

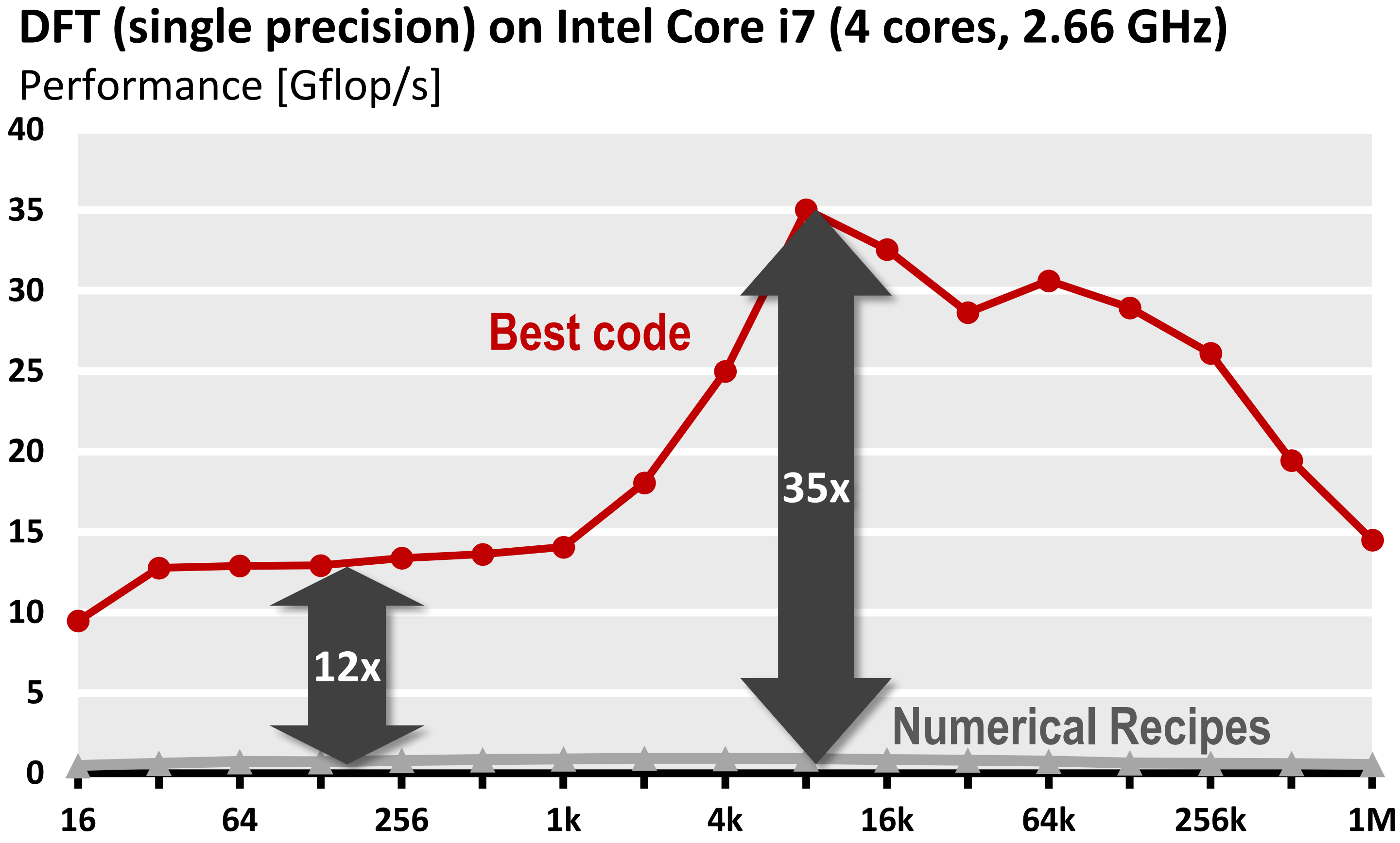
Popular performance libraries

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

Popular productivity/scripting languages

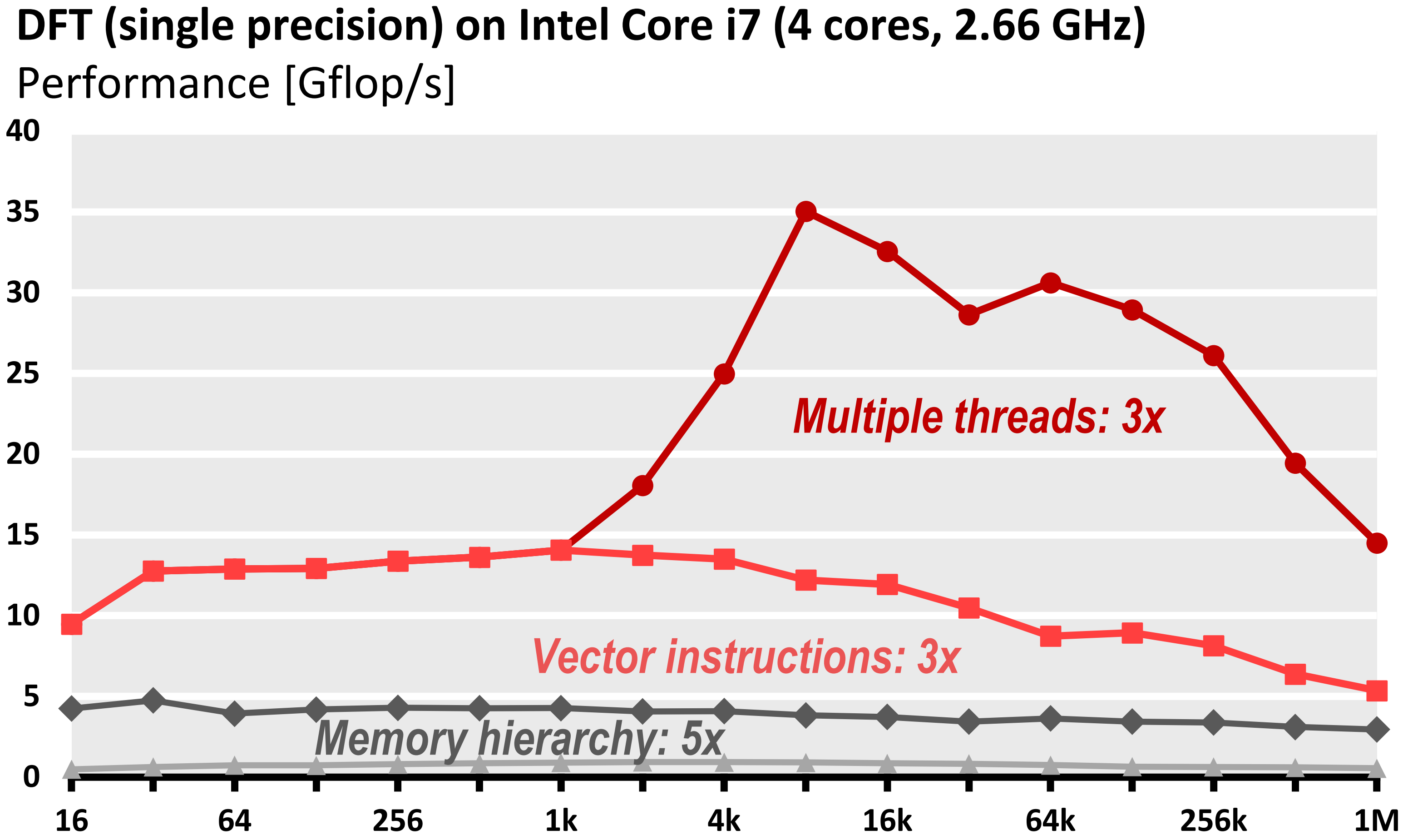
- 1987: Perl
- 1989: Python
- 1993: Ruby
- 1995: Java
- 2000: C#

The Problem: Example DFT



- Standard desktop computer, cutting edge compiler, using optimization flags
- Implementations have same operations count: ≈$4n\log_2(n)$
- ***Same plots can be shown for all mathematical functions***

DFT Plot: Analysis



High performance library development has become a nightmare

How The Generated Code Looks Like

```
void dft64(float  *Y, float  *X) {
    __m512 U912, U913, U914, U915, U916, U917, U918, U919, U920, U921, U922, U923, U924, U925,...;
    a2153 = ((__m512  *) X);  s1107 = *(a2153);
    s1108 = *((a2153 + 4));   t1323 = _mm512_add_ps(s1107,s1108);
    ...
    U926 = _mm512_swizupconv_r32(_mm512_set_1to16_ps(0.70710678118654757),_MM_SWIZ_REG_CDAB);
    s1121 = _mm512_madd231_ps(_mm512_mul_ps(_mm512_mask_or_pi(
        _mm512_set_1to16_ps(0.70710678118654757),0xAAAA,a2154,U926),t1341),
        _mm512_mask_sub_ps(_mm512_set_1to16_ps(0.70710678118654757),0x5555,a2154,U926),
        _mm512_swizupconv_r32(t1341,_MM_SWIZ_REG_CDAB));
    U927 = _mm512_swizupconv_r32(_mm512_set_16to16_ps(0.70710678118654757, (-0.70710678118654757),
        0.70710678118654757, (-0.70710678118654757), 0.70710678118654757, (-0.70710678118654757),
        0.70710678118654757, (-0.70710678118654757), 0.70710678118654757, (-0.70710678118654757),
        0.70710678118654757, (-0.70710678118654757), 0.70710678118654757, (-0.70710678118654757),
        0.70710678118654757, (-0.70710678118654757)),_MM_SWIZ_REG_CDAB);
    ...
    s1166 = _mm512_madd231_ps(_mm512_mul_ps(_mm512_mask_or_pi(_mm512_set_16to16_ps(
        0.70710678118654757, (-0.70710678118654757), 0.70710678118654757, (-0.70710678118654757),
        0.70710678118654757, (-0.70710678118654757), 0.70710678118654757, (-0.70710678118654757),
        0.70710678118654757, (-0.70710678118654757), 0.70710678118654757, (-0.70710678118654757),
        0.70710678118654757, (-0.70710678118654757), 0.70710678118654757, (-0.70710678118654757)),
        0xAAAA,a2154,U951),t1362),
        _mm512_mask_sub_ps(_mm512_set_16to16_ps(0.70710678118654757,
        (-0.70710678118654757), 0.70710678118654757, (-0.70710678118654757), 0.70710678118654757,
        (-0.70710678118654757), 0.70710678118654757, (-0.70710678118654757), 0.70710678118654757,
        (-0.70710678118654757), 0.70710678118654757, (-0.70710678118654757), 0.70710678118654757,
        (-0.70710678118654757), 0.70710678118654757, (-0.70710678118654757)),0x5555,a2154,U951),
        _mm512_swizupconv_r32(t1362,_MM_SWIZ_REG_CDAB));
    ...
}
```

D. McFarlin, F. Franchetti, and M. Püschel: **Automatic Generation of Vectorized Fast Fourier Transform Libraries for the Larrabee and AVX Instruction Set Extension.** Proceedings of the 2009 High Performance Embedded Computing (HPEC), MIT Lincoln Laboratory. **Best paper award.**

Goal: Go from Mathematics to Software

Given:

- Mathematical problem specification

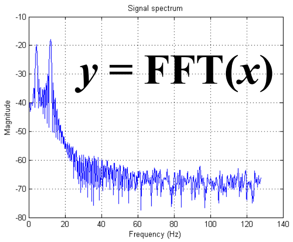
  does not change

- Computer platform

  changes often

Wanted:

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



$y = \mathrm{FFT}(x)$

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));
    ...
}
```

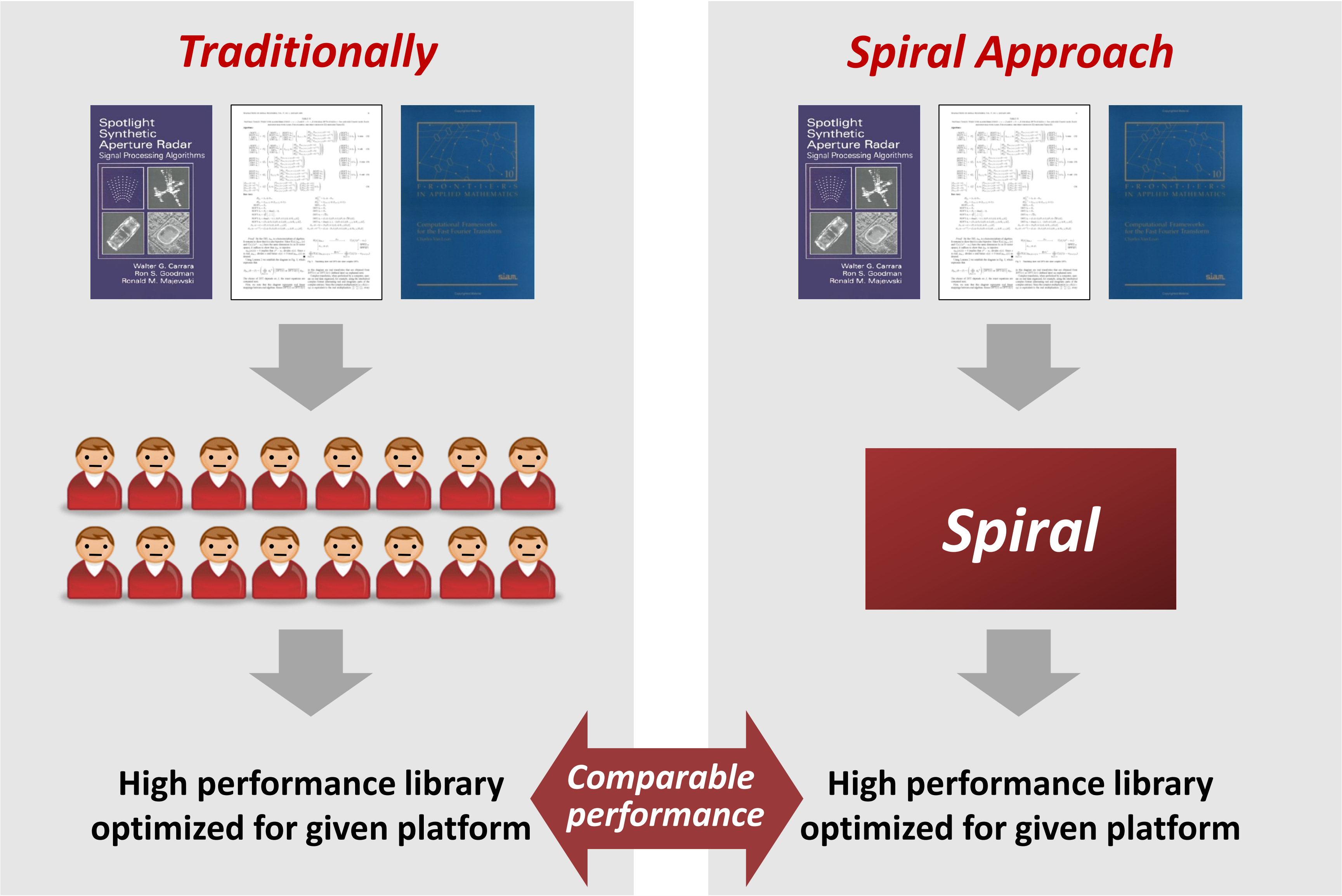
performance

+

PROOF

QED.

The Spiral System



Traditionally

High performance library optimized for given platform

Comparable performance

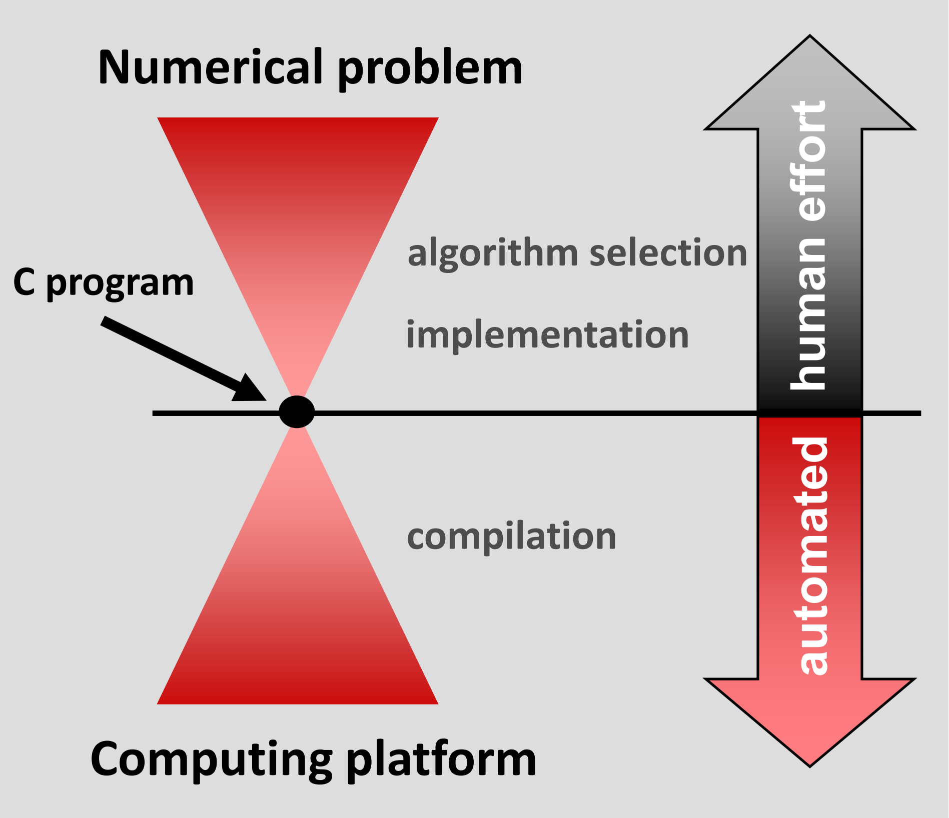
Spiral Approach

Spiral

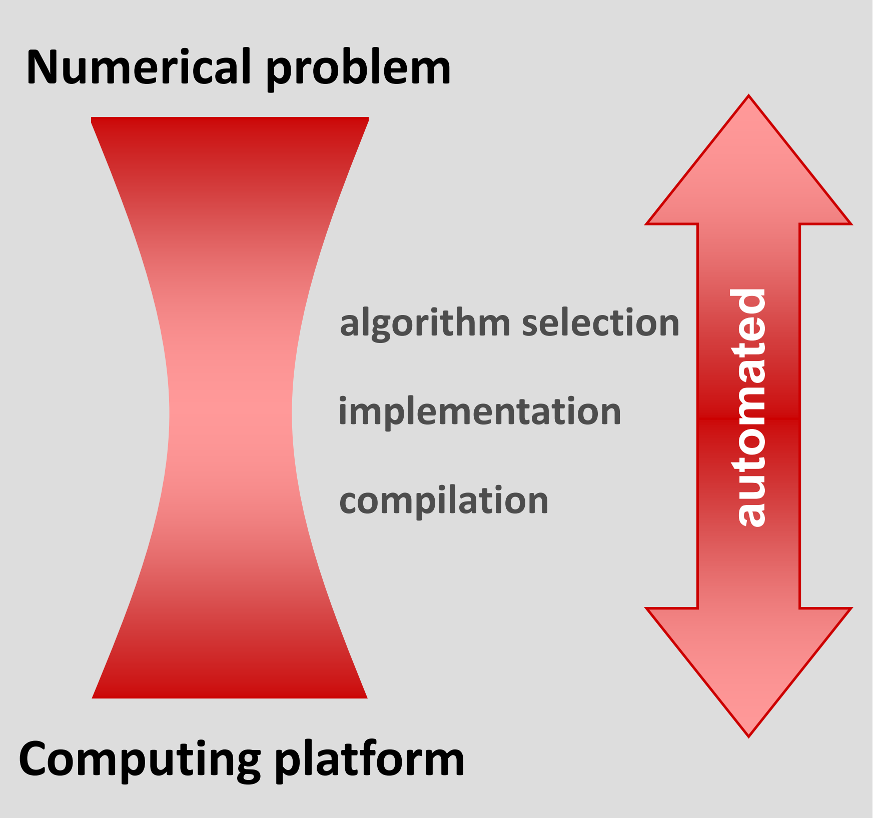
High performance library optimized for given platform

Vision Behind Spiral

Current



Future



- C code a singularity: Compiler has no access to high level information

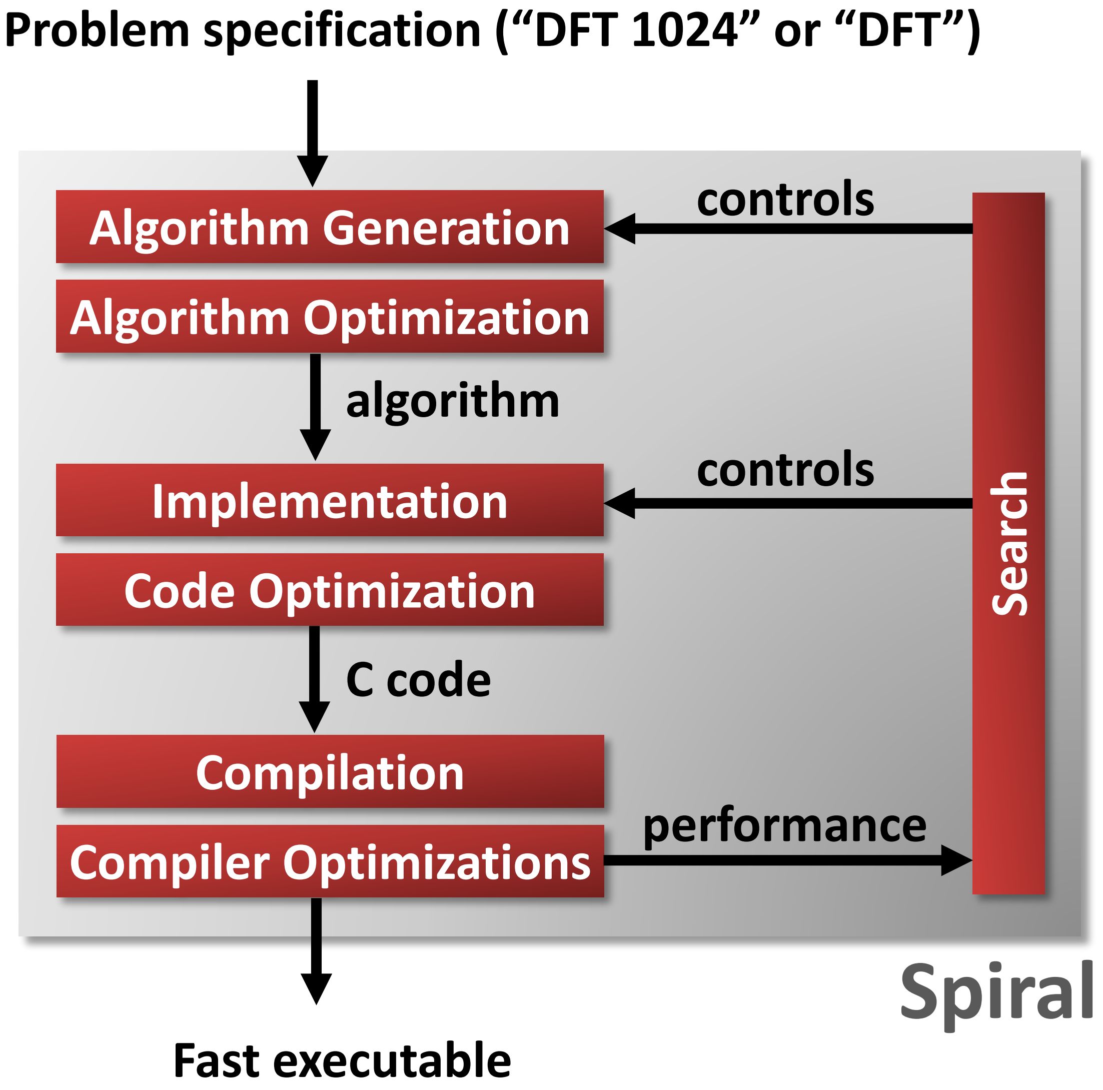
- Challenge: conquer the high abstraction level for complete automation

How Spiral Works

Complete automation of the implementation and optimization task

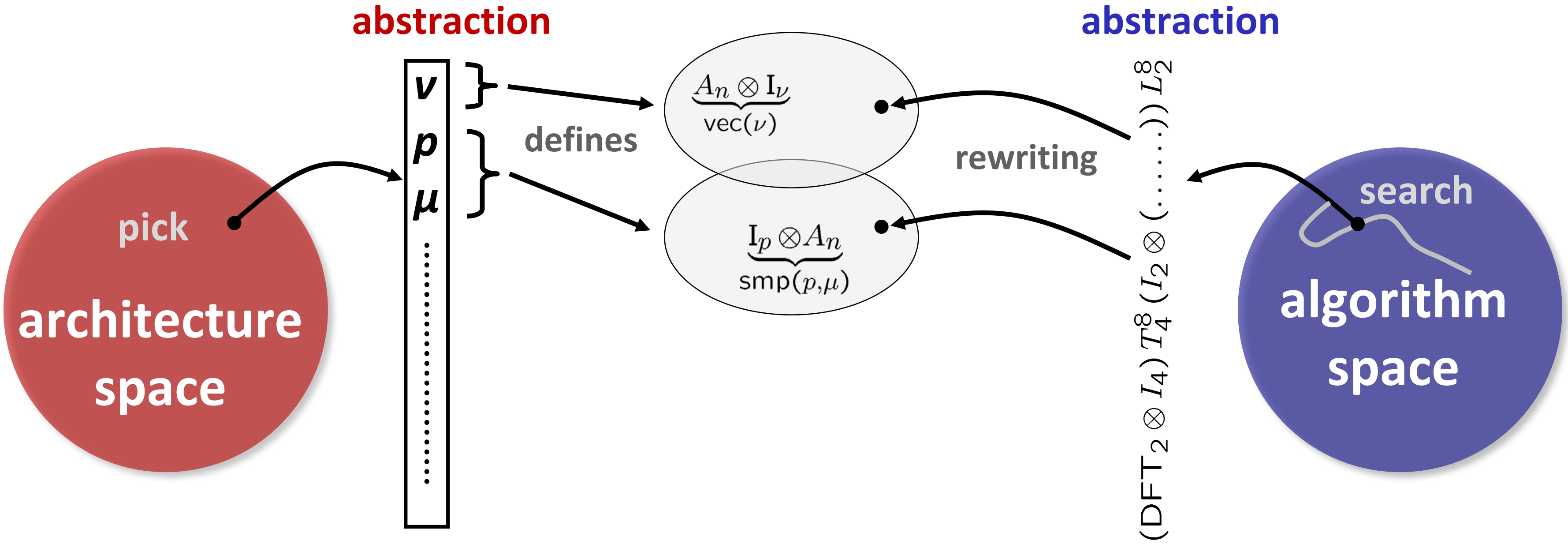
Basic ideas:

- *Declarative representation* of algorithms
- *Rewriting systems* to generate and optimize algorithms at a high level of abstraction



Spiral's Domain-Specific Program Synthesis

Model: common abstraction
= spaces of matching formulas


abstraction
abstraction
ν
p
μ
defines
$\underbrace{A_n \otimes I_\nu}_{\mathrm{vec}(\nu)}$
$\underbrace{I_p \otimes A_n}_{\mathrm{smp}(p,\mu)}$
rewriting
$(\mathrm{DFT}_2 \otimes I_4)\, T^8_4\, (I_2 \otimes (\ldots\ldots))\, L^8_2$
pick
architecture space
search
algorithm space

Architectural parameter:
Vector length,
#processors, …

optimization

Kernel:
problem size,
algorithm choice

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

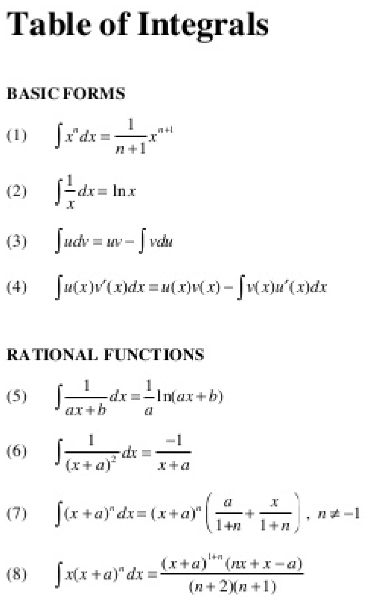


Table of Integrals

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

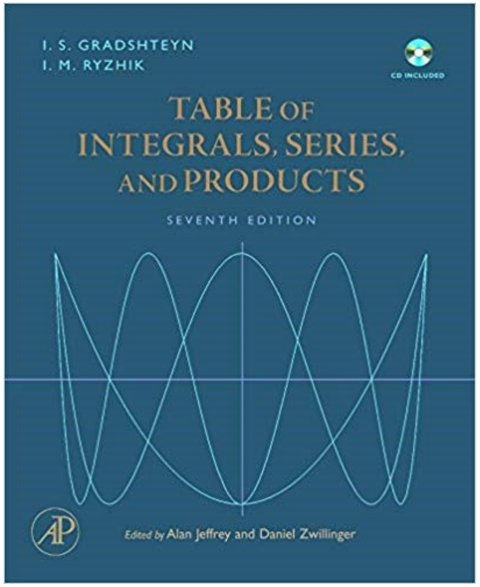
RATIONAL FUNCTIONS

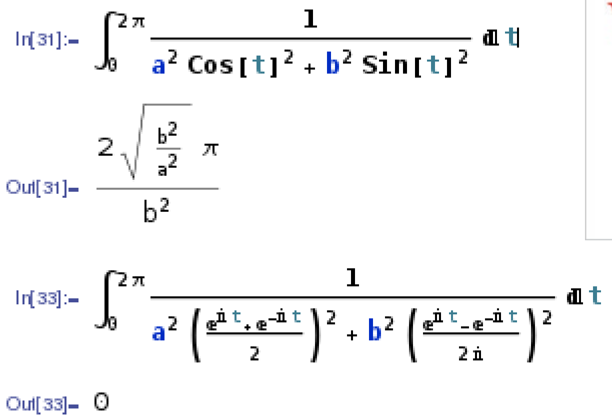
(5) $\int \frac{1}{ax+b} dx = \frac{1}{a} \ln(ax+b)$

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





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

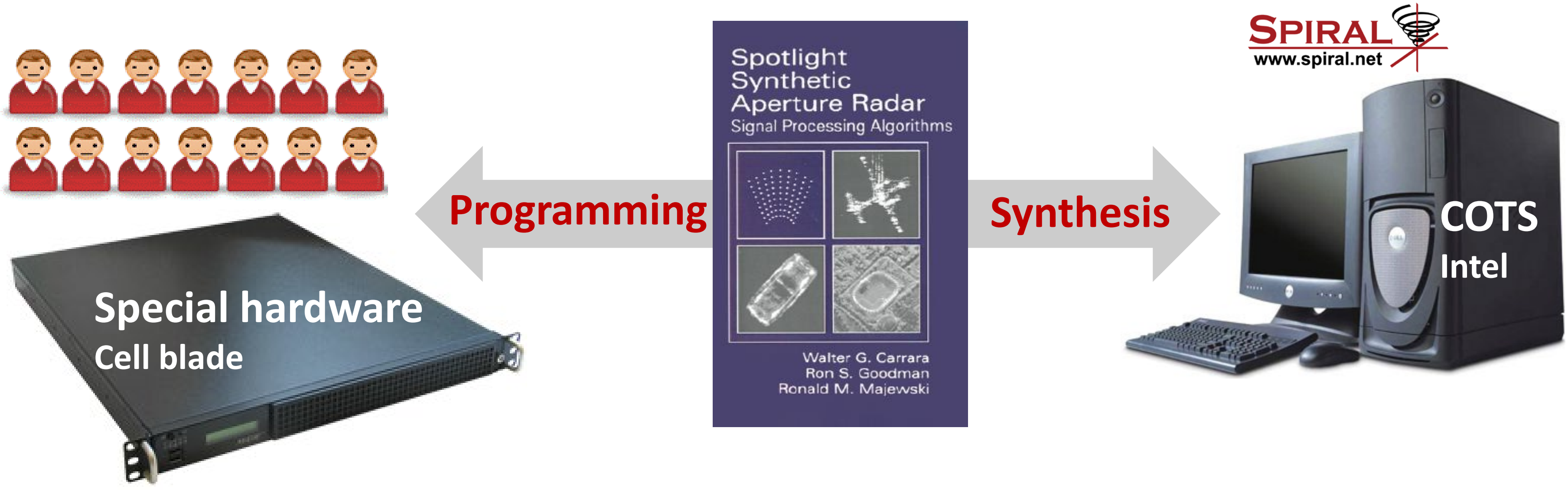
Out[31]= $\frac{2\sqrt{\frac{b^2}{a^2}}\,\pi}{b^2}$

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

Out[33]= 0

Wolfram Mathematica

Example 1: SAR for Cell BE



Special hardware
Cell blade

Programming

Spotlight Synthetic Aperture Radar
Signal Processing Algorithms

Walter G. Carrara
Ron S. Goodman
Ronald M. Majewski

Synthesis

SPIRAL
www.spiral.net

COTS
Intel

Result

Same performance, 1/10th human effort, non-expert user

Key ideas

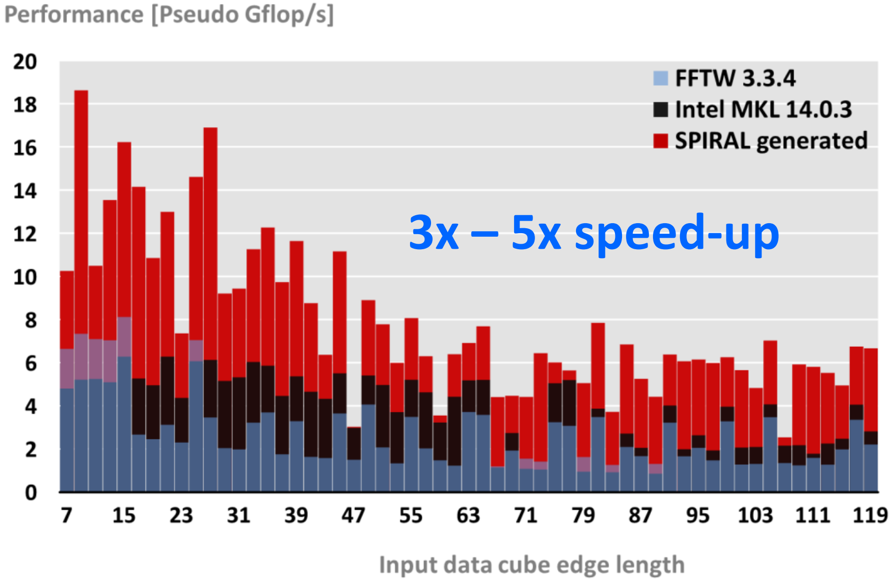
restrict domain, use mathematics, performance portability

D. McFarlin, F. Franchetti, M. Püschel, and J. M. F. Moura: **High Performance Synthetic Aperture Radar Image Formation On Commodity Multicore Architectures.** in Proceedings SPIE, 2009.

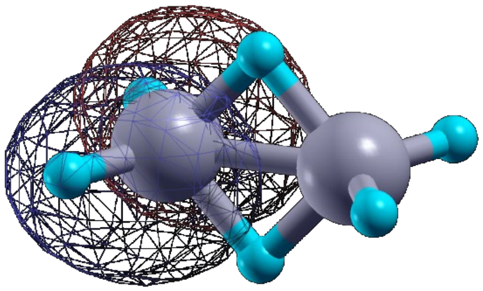
Example 2: Density Functional Theory

Performance of 2x2x2 Upsampling on Haswell

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



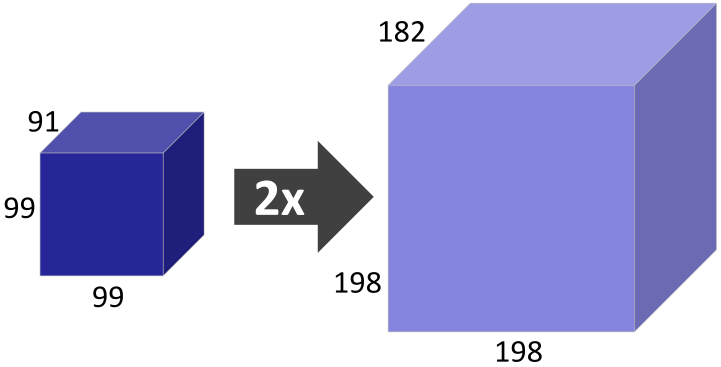
ONETEP



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.

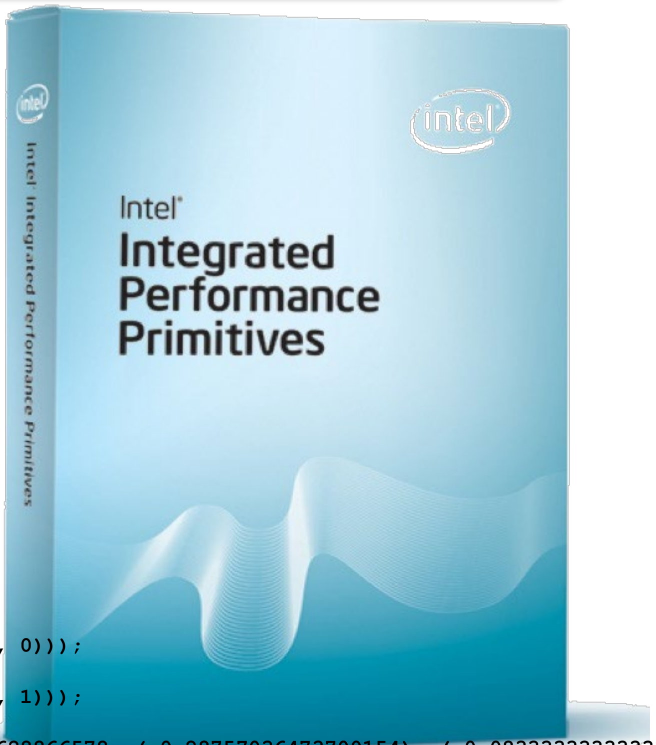
Example 3: Synthesis of Production Code

```
...
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

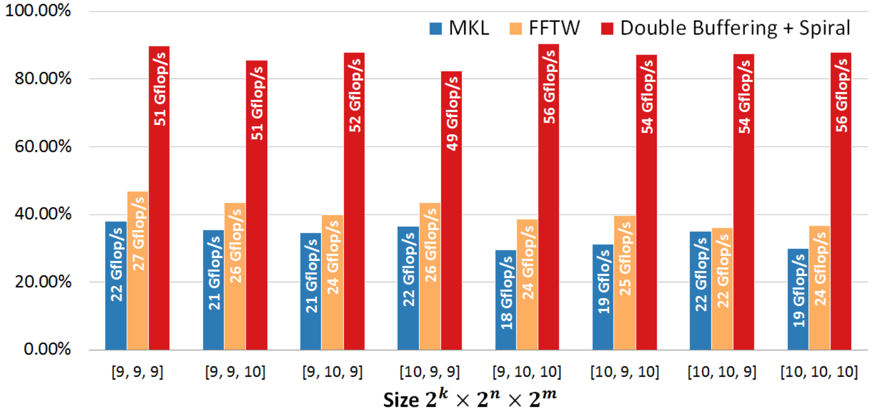
- IPP = Intel's performance primitives, part of Intel C++ Compiler suite
- Generated: 3984 C functions (signal processing) = 1M lines of code
- Full parallelism support
- Computer-generated code: Faster than what was achievable by hand

```
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), (-0.08333333333333333
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(0.2875
s3046 = _mm_sub_ps(_mm_mul_ps(_mm_set_ps((-0.25624767158293649), 0.25826039031174486, (-0.30023863596633249), 0.075902986037193879), t3799), _mm
s3047 = _mm_add_ps(_mm_mul_ps(_mm_set_ps(0.15689139105158462, (-0.15355568557954136), (-0.011599105605768193), 0.29071724147084099), t3799), _mm
```



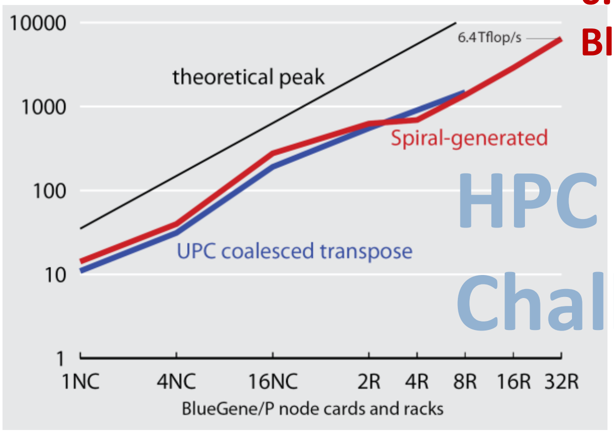
Selected Results: FFTs and Spectral Algorithms

3D FFT performance on Intel Kaby Lake 7700K
4.5 GHz, 4/8 cores/threads, double-precision, AVX
[% of achievable peak performance given STREAM bandwidth]



FFT on Multicore

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

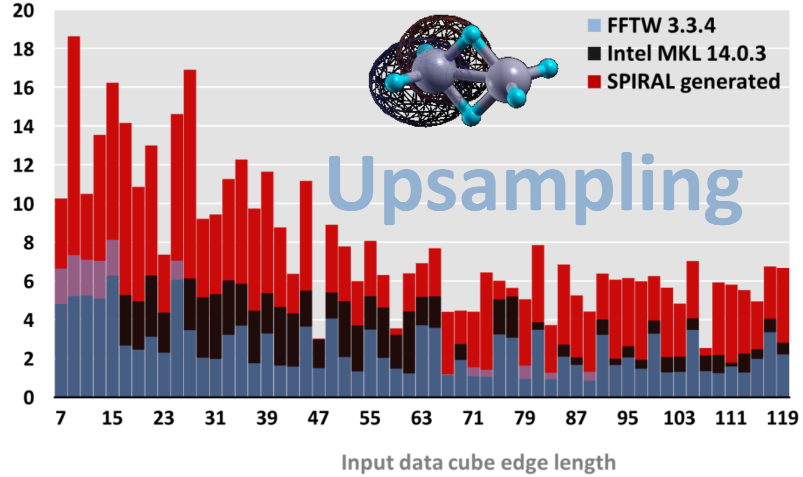


6.4 Tflop/s on BlueGene/P

HPC Challenge

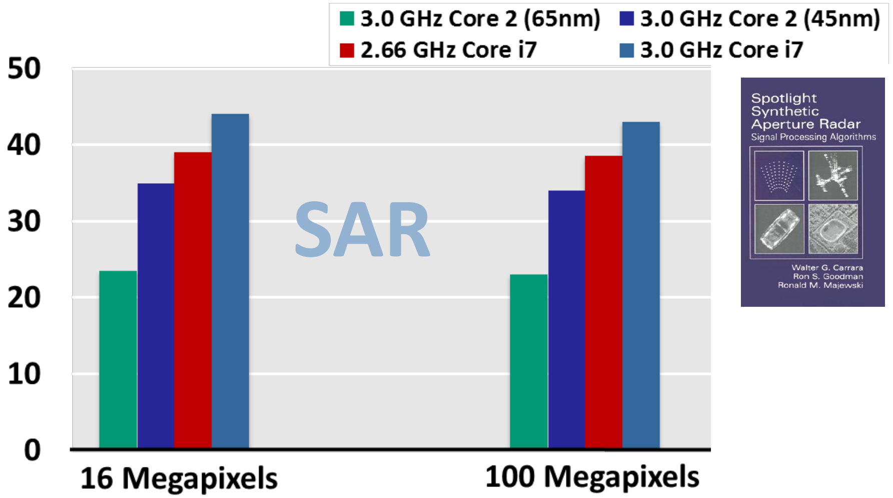
BlueGene/P at Argonne National Laboratory
128k cores (quad-core CPUs) at 850 MHz

Performance of 2x2x2 Upsampling on Haswell
3.5 GHz, AVX, double precision, interleaved input, single core
Performance [Pseudo Gflop/s]



Upsampling

PFA SAR Image Formation on Intel platforms
performance [Gflop/s]

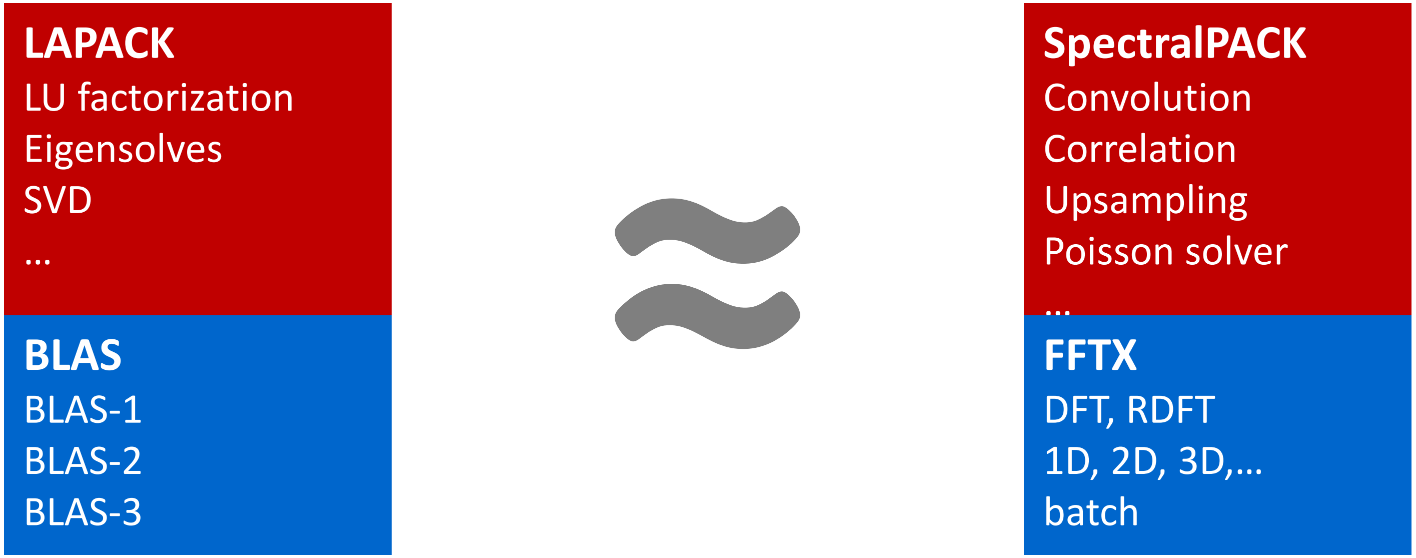


SAR

Current Work: FFTX and SpectralPACK

Numerical Linear Algebra

Spectral Algorithms



LAPACK
LU factorization
Eigensolves
SVD
…

BLAS
BLAS-1
BLAS-2
BLAS-3

≈

SpectralPACK
Convolution
Correlation
Upsampling
Poisson solver
…

FFTX
DFT, RDFT
1D, 2D, 3D,…
batch

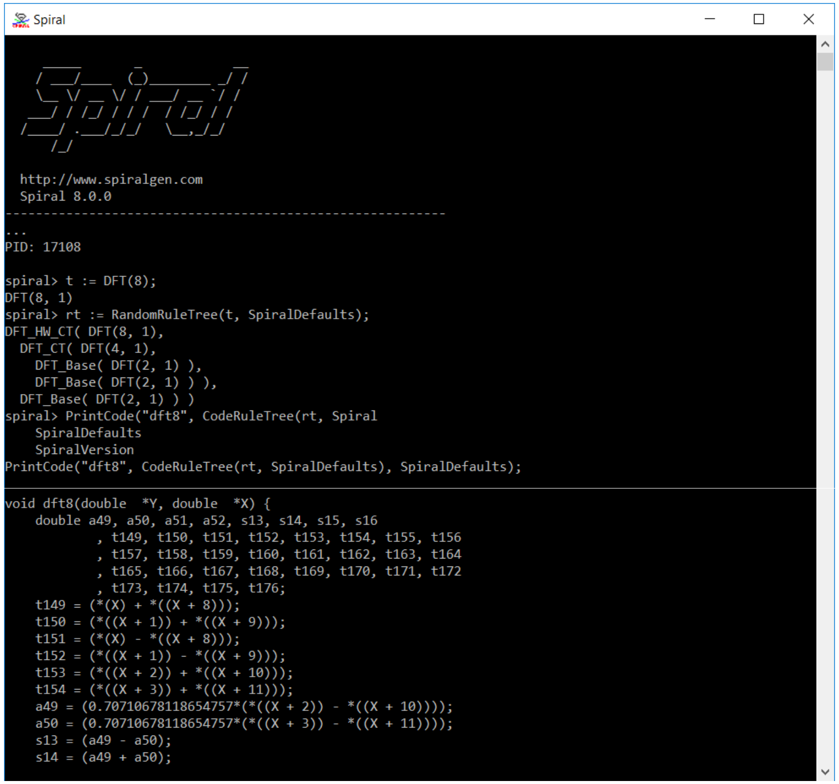
Define the LAPACK equivalent for spectral algorithms

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

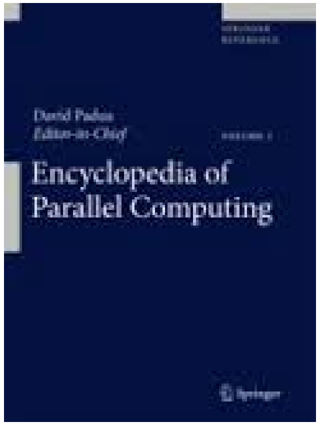
Spiral provides backend code generation and autotuning

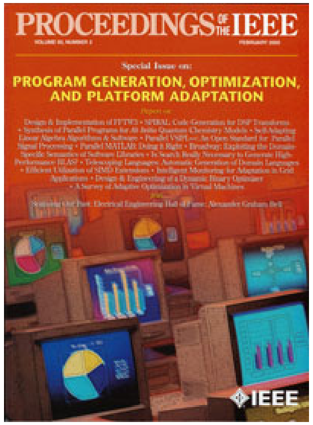
SPIRAL 8.1.0: Open Source

- **Open Source SPIRAL available**
  - non-viral license (BSD)
  - Initial version, effort ongoing to open source whole system
  - Open sourced under DARPA PERFECT
  - 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



```
   _____       _           __
  / ___/____  (_)________ _/ /
  \__ \/ __ \/ / ___/ __ `/ /
 ___/ / /_/ / / /  / /_/ / /
/____/ .___/_/_/   \__,_/_/
    /_/

  http://www.spiralgen.com
  Spiral 8.0.0
-----------------------------------------------------------
...
PID: 17108

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

void dft8(double  *Y, double  *X) {
    double a49, a50, a51, a52, s13, s14, s15, s16
            , t149, t150, t151, t152, t153, t154, t155, t156
            , t157, t158, t159, t160, t161, t162, t163, t164
            , t165, t166, t167, t168, t169, t170, t171, t172
            , t173, t174, t175, t176;
    t149 = (*(X) + *((X + 8)));
    t150 = (*((X + 1)) + *((X + 9)));
    t151 = (*(X) - *((X + 8)));
    t152 = (*((X + 1)) - *((X + 9)));
    t153 = (*((X + 2)) + *((X + 10)));
    t154 = (*((X + 3)) + *((X + 11)));
    a49 = (0.70710678118654757*(*((X + 2)) - *((X + 10))));
    a50 = (0.70710678118654757*(*((X + 3)) - *((X + 11))));
    s13 = (a49 - a50);
    s14 = (a49 + a50);
```





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*

Organization

- **SPL: Problem and algorithm specification**
- Σ-SPL: Automating high level optimization
- Rewriting: Formal parallelization
- Rewriting: Vectorization
- Verification
- Spiral as FFTX backend
- 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*

What is a (Linear) Transform?

- **Mathematically: Matrix-vector multiplication**

$$x \mapsto y = T \cdot x$$

input vector (signal)

output vector (signal)

transform = matrix

- **Example: Discrete Fourier transform (DFT)**

$$\mathrm{DFT}_n \;=\; [e^{-2k\ell\pi i/n}]_{0\le k,\ell<n}$$

Transform Algorithms: Example 4-point FFT

Cooley/Tukey fast Fourier transform (FFT):

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

Fourier transform

Diagonal matrix (twiddles)

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

Kronecker product

Identity

Permutation

- Algorithms are divide-and-conquer: Breakdown rules
- Mathematical, declarative representation: SPL (signal processing language)
- SPL describes the structure of the dataflow

Examples: Transforms

$$
\begin{aligned}
\mathbf{DCT\text{-}2}_n &= \left[\cos(k(2\ell+1)\pi/2n)\right]_{0\le k,\ell<n},\\
\mathbf{DCT\text{-}3}_n &= \mathbf{DCT\text{-}2}_n^T \quad \text{(transpose)},\\
\mathbf{DCT\text{-}4}_n &= \left[\cos((2k+1)(2\ell+1)\pi/4n)\right]_{0\le k,\ell<n},\\
\mathbf{IMDCT}_n &= \left[\cos((2k+1)(2\ell+1+n)\pi/4n)\right]_{0\le k<2n,0\le\ell<n},\\
\mathbf{RDFT}_n &= [r_{k\ell}]_{0\le k,\ell<n}, \quad r_{k\ell} = \begin{cases} \cos\frac{2\pi k\ell}{n}, & k \le \lfloor\frac{n}{2}\rfloor \\ -\sin\frac{2\pi k\ell}{n}, & k > \lfloor\frac{n}{2}\rfloor \end{cases},\\
\mathbf{WHT}_n &= \begin{bmatrix} \mathbf{WHT}_{n/2} & \mathbf{WHT}_{n/2} \\ \mathbf{WHT}_{n/2} & -\mathbf{WHT}_{n/2} \end{bmatrix}, \quad \mathbf{WHT}_2 = \mathbf{DFT}_2,\\
\mathbf{DHT} &= \left[\cos(2k\ell\pi/n)+\sin(2k\ell\pi/n)\right]_{0\le k,\ell<n}.
\end{aligned}
$$

Examples: Breakdown Rules (currently ≈220)

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

Combining these rules yields many algorithms for every given transform

Breakdown Rules (>200 for >50 Transforms)

$$\mathbf{DFT}_n \to P_{k/2,2m}^{\top}\left(\mathbf{DFT}_{2m} \oplus \left(I_{k/2-1} \otimes_i C_{2m}\,\mathbf{rDFT}_{2m}(i/k)\right)\right)\left(\mathbf{RDFT}'_k \otimes I_m\right), \quad k \text{ even},$$

$$\left|\begin{matrix}\mathbf{RDFT}_n\\ \mathbf{RDFT}'_n\\ \mathbf{DHT}_n\\ \mathbf{DHT}'_n\end{matrix}\right| \to (P_{k/2,m}^{\top} \otimes I_2)\left(\left|\begin{matrix}\mathbf{RDFT}_{2m}\\ \mathbf{RDFT}'_{2m}\\ \mathbf{DHT}_{2m}\\ \mathbf{DHT}'_{2m}\end{matrix}\right| \oplus \left(I_{k/2-1} \otimes_i D_{2m}\left|\begin{matrix}\mathbf{rDFT}_{2m}(i/k)\\ \mathbf{rDFT}_{2m}(i/k)\\ \mathbf{rDHT}_{2m}(i/k)\\ \mathbf{rDHT}_{2m}(i/k)\end{matrix}\right|\right)\right)\left(\left|\begin{matrix}\mathbf{RDFT}'_k\\ \mathbf{RDFT}'_k\\ \mathbf{DHT}'_k\\ \mathbf{DHT}'_k\end{matrix}\right| \otimes I_m\right), \quad k \text{ even},$$

$$\left|\begin{matrix}\mathbf{rDFT}_{2n}(u)\\ \mathbf{rDHT}_{2n}(u)\end{matrix}\right| \to L_m^{2n}\left(I_k \otimes_i \left|\begin{matrix}\mathbf{rDFT}_{2m}((i+u)/k)\\ \mathbf{rDHT}_{2m}((i+u)/k)\end{matrix}\right|\right)\left(\left|\begin{matrix}\mathbf{rDFT}_{2k}(u)\\ \mathbf{rDHT}_{2k}(u)\end{matrix}\right| \otimes I_m\right),$$

$$\mathbf{RDFT\text{-}3}_n \to (Q_{k/2,m}^{\top} \otimes I_2)\,(I_k \otimes_i \mathbf{rDFT}_{2m})(i+1/2)/k))\,(\mathbf{RDFT\text{-}3}_k \otimes I_m), \quad k \text{ even},$$

$$\mathbf{DCT\text{-}2}_n \to P_{k/2,2m}^{\top}\left(\mathbf{DCT\text{-}2}_{2m}\,K_2^{2m} \oplus \left(I_{k/2-1} \otimes N_{2m}\,\mathbf{RDFT\text{-}3}_{2m}^{\top}\right)\right)B_n(L_{k/2}^{n/2} \otimes I_2)(I_m \otimes \mathbf{RDFT}'_k)Q_{m/2,k},$$

$$\mathbf{DCT\text{-}3}_n \to \mathbf{DCT\text{-}2}_n^{\top},$$

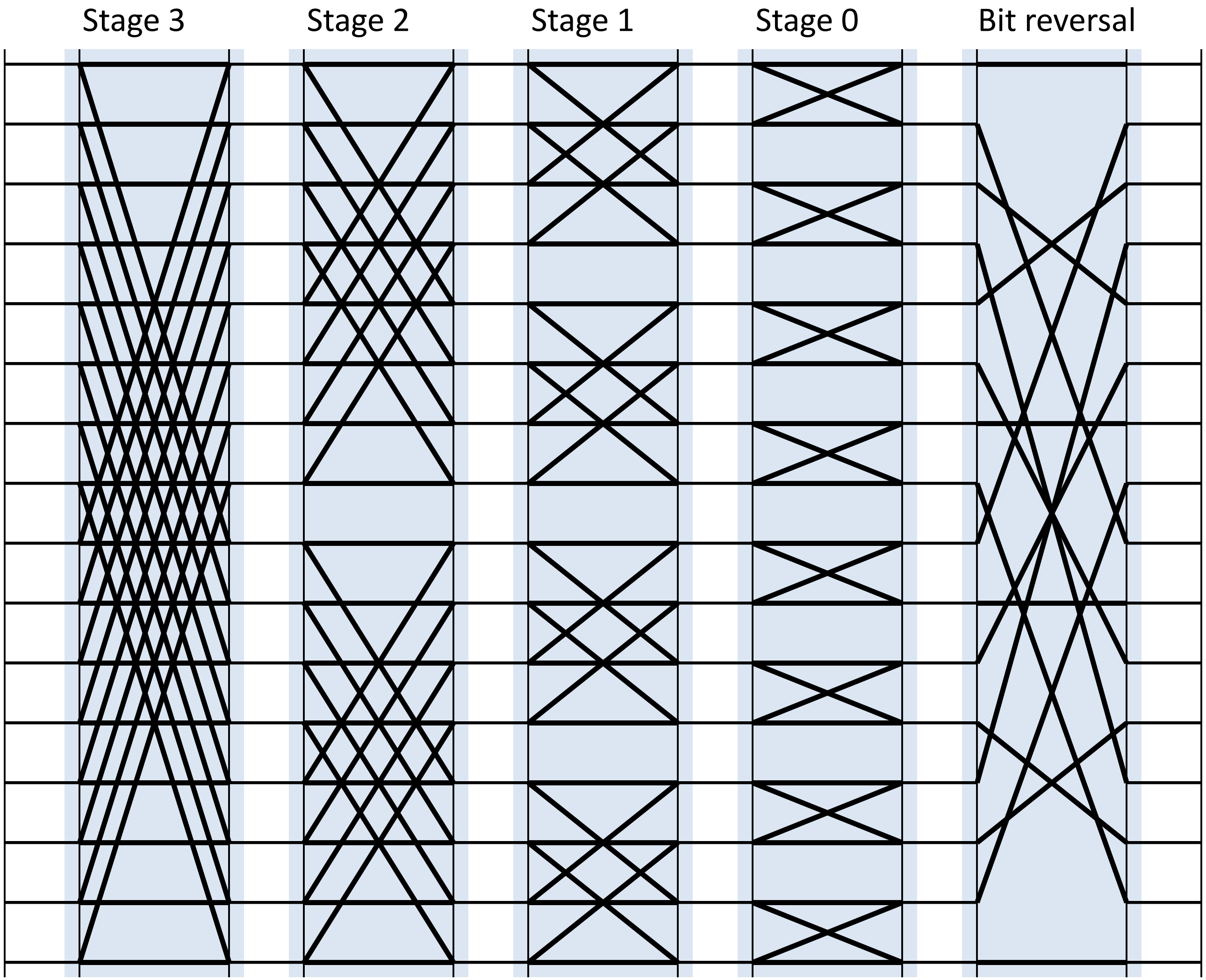
$$\mathbf{DCT\text{-}4}_n \to Q_{k/2,2m}^{\top}\left(I_{k/2} \otimes N_{2m}\,\mathbf{RDFT\text{-}3}_{2m}^{\top}\right)B'_n(L_{k/2}^{n/2} \otimes I_2)(I_m \otimes \mathbf{RDFT\text{-}3}_k)Q_{m/2,k}.$$

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

- *"Teaches" Spiral algorithm knowledge*
- *Combining these rules yields many algorithms for every given transform*

Example FFT: Iterative FFT Algorithm

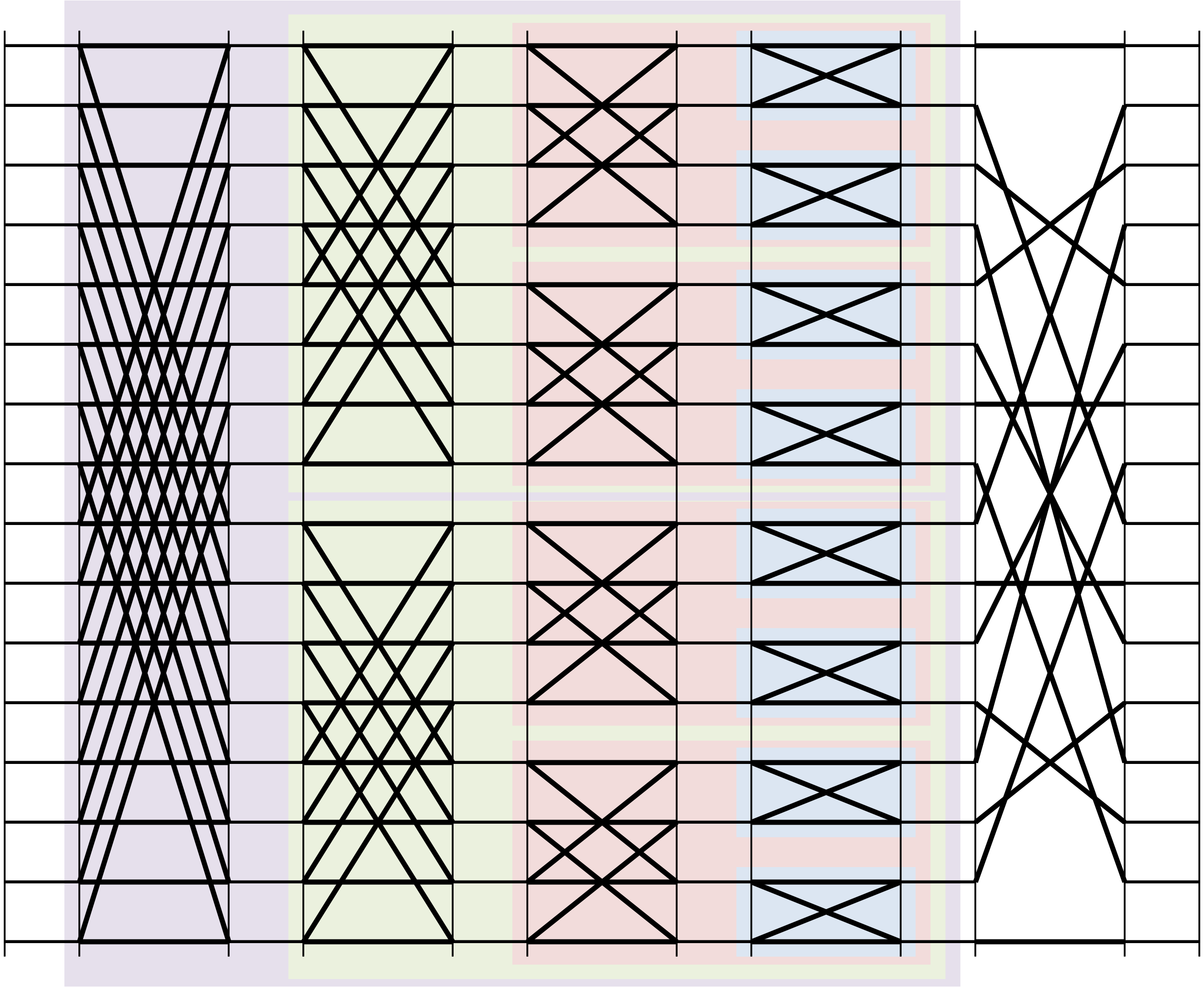
$$\mathrm{DFT}_{r^k} = \left(\prod_{i=0}^{k-1} \left(I_{r^i} \otimes \mathrm{DFT}_r \otimes I_{r^{k-i-1}} \right) D_i^{r^k} \right) R_r^{r^k}$$



$$\left((I_1 \otimes \mathrm{DFT}_2 \otimes I_8) D_0^{16} \right) \left((I_2 \otimes \mathrm{DFT}_2 \otimes I_4) D_1^{16} \right) \left((I_4 \otimes \mathrm{DFT}_2 \otimes I_2) D_2^{16} \right) \left((I_8 \otimes \mathrm{DFT}_2 \otimes I_1) D_3^{16} \right) R_2^{16}$$

Example FFT: Recursive FFT Algorithm

$$\mathrm{DFT}_{km} = (\mathrm{DFT}_k \otimes I_m) T_m^n (I_k \otimes \mathrm{DFT}_m) L_k^n$$



$$(\mathrm{DFT}_2 \otimes I_8) T_8^{16} \Big(I_2 \otimes \Big((\mathrm{DFT}_2 \otimes I_4) T_4^8 \big(I_2 \otimes ((\mathrm{DFT}_2 \otimes I_2) T_2^4 (I_2 \otimes \mathrm{DFT}_2) L_2^4) \big) L_2^8 \Big) \Big) L_2^{16}$$

SPL Compiler

SPL construct	code
$y = (A_n B_n)x$	t[0:1:n-1] = B(x[0:1:n-1]); y[0:1:n-1] = A(t[0:1:n-1];)

$y = (I_m \otimes A_n)x$	for (i=0;i<m;i++)
	y[i*n:1:i*n+n-1] =
	A(x[i*n:1:i*n+n-1])

$y = (A_m \otimes I_n)x$	for (i=0;i<m;i++)
	y[i:n:i+m-1] =
	A(x[i:n:i+m-1]);

$y = \left(\bigoplus_{i=0}^{m-1} A_n^i\right) x$	for (i=0;i<m;i++)
	y[i*n:1:i*n+n-1] =
	A(i, x[i*n:1:i*n+n-1]);

$y = D_{m,n}x$	for (i=0;i<m*n;i++)
	y[i] = Dmn[i]*x[i];

$y = L_m^{mn}x$	for (i=0;i<m;i++)
	for (j=0;j<n;j++)
	y[i+m*j]=x[n*i+j];

Example: tensor product

$$\mathrm{I}_m \otimes A_n = \begin{bmatrix} A_n & & \\ & \ddots & \\ & & A_n \end{bmatrix}$$

Works well for basic blocks. Loops and parallelization: next

Organization

- SPL: Problem and algorithm specification
- **Σ-SPL: Automating high level optimization**
- Rewriting: Formal parallelization
- Rewriting: Vectorization
- Verification
- Spiral as FFTX backend
- Summary

F. Franchetti, Y. Voronenko, M. Püschel: **Loop Merging for Signal Transforms**
Proceedings Programming Language Design and Implementation (PLDI) 2005, pages 315-326.

Problem: Fusing Permutations and Loops

$$y = (I_4 \otimes F_2)\, L^8_4\, x$$

direct mapping

Two passes over the working set
Complex index computation

```
void sub(double *y, double *x) {
   double t[8];
   for (int i=0; i<=7; i++)
      t[(i/4)+2*(i%4)] = x[i];
   for (int i=0; i<4; i++){
      y[2*i] = t[2*i] + t[2*i+1];
      y[2*i+1] = t[2*i] - t[2*i+1];
   }
}
```

C compiler cannot do this

One pass over the working set
Simple index computation

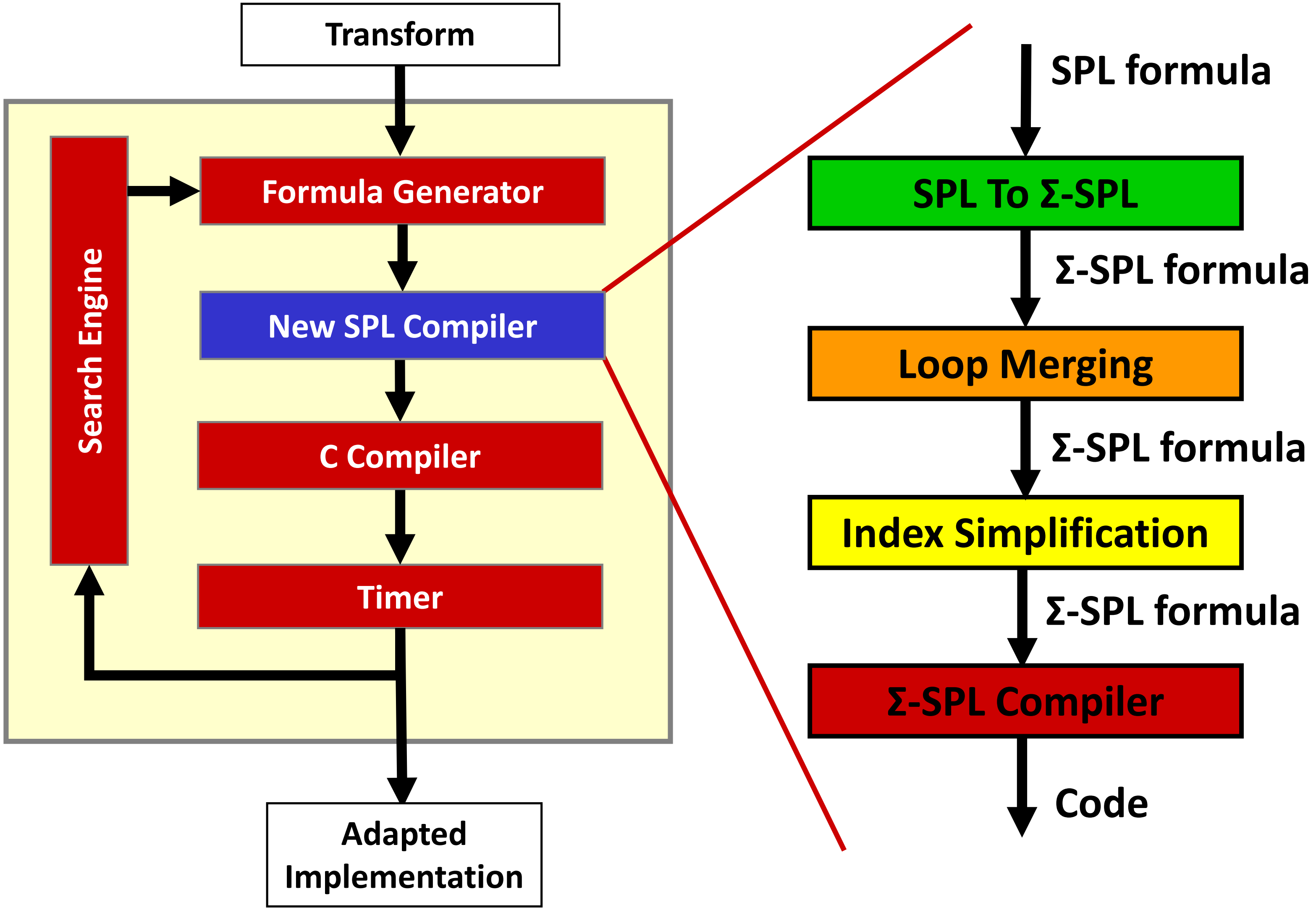
```
void sub(double *y, double *x) {
   for (int j=0; j<=3; j++){
      y[2*j] = x[j] + x[j+4];
      y[2*j+1] = x[j] - x[j+4];
   }
}
```

hardcoded special case

State-of-the-art
SPIRAL: Hardcoded with templates
FFTW: Hardcoded in the infrastructure

How does hardcoding scale?

New Approach for Loop Merging



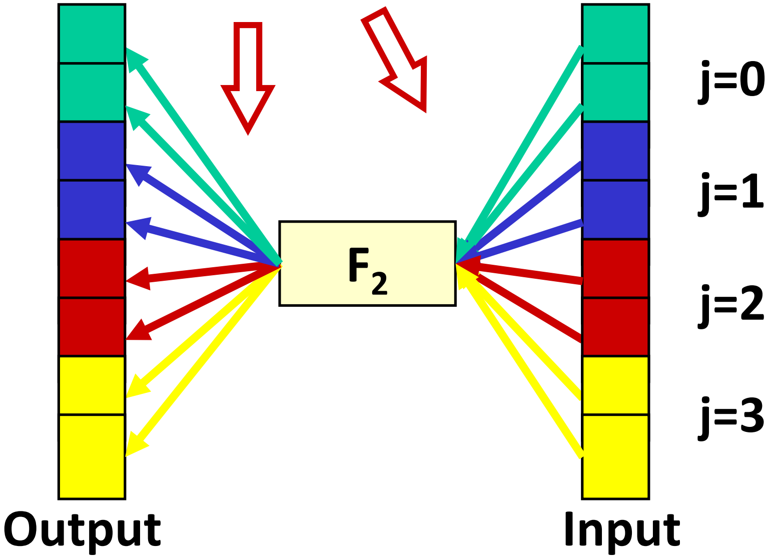
Σ-SPL

- **Four central constructs: S, G, S, Perm**
  - Σ (sum) – makes loops explicit
  - G_f (gather) – reads data using the index mapping *f*
  - S_f (scatter) – writes data using the index mapping *f*
  - Perm_f – permutes data using the index mapping *f*

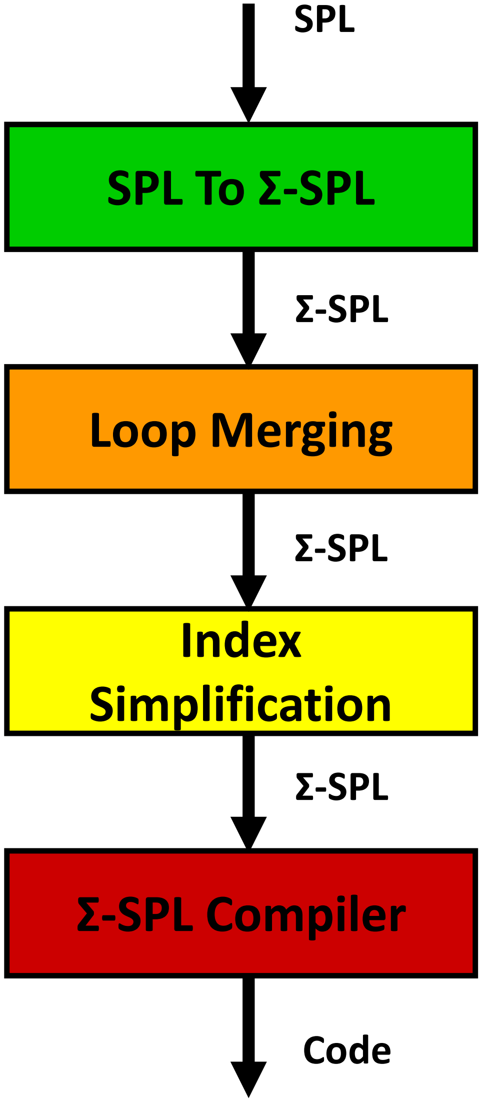
- **Every Σ-SPL formula still represents a matrix factorization**

Example: $(I_4 \otimes F_2) \rightarrow \sum_{j=0}^{3} S_{f_j} F_2 G_{f_j}$

$$\begin{bmatrix} F_2 & & & \\ & F_2 & & \\ & & F_2 & \\ & & & F_2 \end{bmatrix}$$



Loop Merging With Rewriting Rules



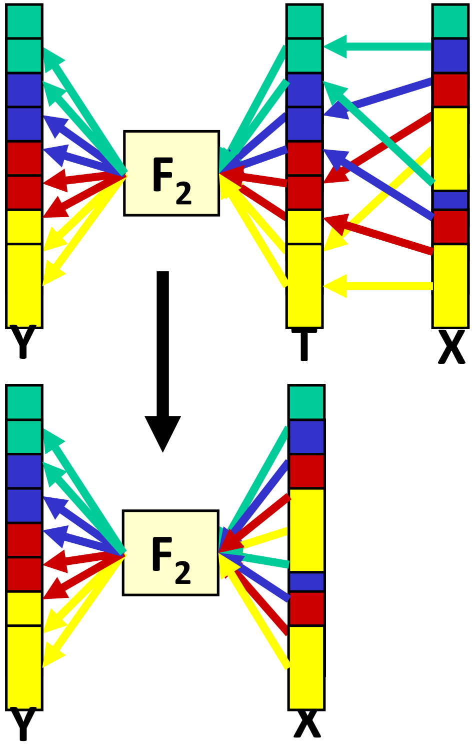
$$y = (\mathrm{I}_4 \otimes \mathrm{F}_2)\, \mathrm{L}^8_4\, x$$

$$\left(\sum_{j=0}^{3} \mathrm{S}_{(j)_4\otimes\imath_2}\, \mathrm{F}_2\, \mathrm{G}_{(j)_4\otimes\imath_2}\right) \mathrm{Perm}_{\ell^8_4}$$

$$\left(\sum_{j=0}^{3} \mathrm{S}_{(j)_4\otimes\imath_2}\, \mathrm{F}_2\, \boxed{\mathrm{G}_{\ell^8_4\circ((j)_4\otimes\imath_2)}}\right)$$

$$\left(\sum_{j=0}^{3} \mathrm{S}_{(j)_4\otimes\imath_2}\, \mathrm{F}_2\, \boxed{\mathrm{G}_{\imath_2\otimes(j)_4}}\right)$$

```
for (int j=0; j<=3; j++) {
      y[2*j]   = x[j] + x[j+4];
      y[2*j+1] = x[j] - x[j+4];
}
```



Rules:

$$\mathrm{G}_r\, \mathrm{Perm}_p \rightarrow \mathrm{G}_{p\circ r}$$

$$\ell^{mn}_m \circ \left((j)_m \otimes \imath_n\right) \rightarrow \imath_n \otimes (j)_m$$

Application: Loop Merging For FFTs

DFT breakdown rules:

Cooley-Tukey FFT $\mathbf{DFT}_{km} \rightarrow (\mathbf{DFT}_k \otimes \mathrm{I}_m)\, \mathsf{T}^{km}_m (\mathrm{I}_k \otimes \mathbf{DFT}_m)\, \mathsf{L}^{km}_k$

Prime factor FFT $\mathbf{DFT}_{km} \rightarrow \mathsf{V}^T_{k,m} (\mathbf{DFT}_k \otimes \mathrm{I}_m)(\mathrm{I}_k \otimes \mathbf{DFT}_m)\, \mathsf{V}_{k,m}$

$\mathsf{gcd}(k,m) = 1$

Rader FFT $\mathbf{DFT}_p \rightarrow \mathsf{W}^T_p (\mathrm{I}_1 \oplus \mathbf{DFT}_{p-1}) D_p (\mathrm{I}_1 \oplus \mathbf{DFT}_{p-1})\, \mathsf{W}_p$

p - prime

Index mapping functions are non-trivial:

$$\mathsf{L}^{km}_k \rightarrow \mathsf{Perm}_{\ell^{km}_k} \qquad \ell^{km}_k(i) = \left\lfloor \frac{i}{m} \right\rfloor + k(i \mod m)$$

$$\mathsf{V}_{k,m} \rightarrow \mathsf{Perm}_{v_{k,m}} \qquad v_{k,m}(i) = \left(m \left\lfloor \frac{i}{m} \right\rfloor + k(i \mod m) \right) \mod km$$

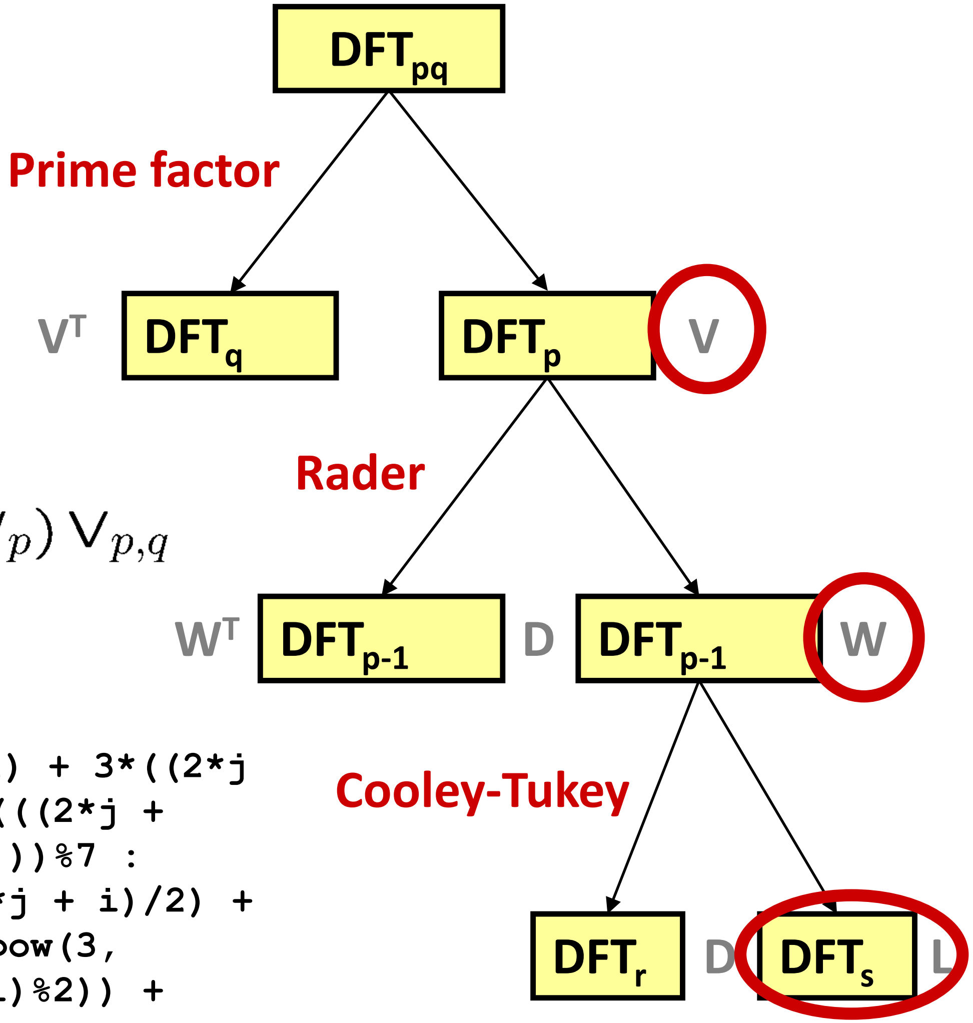
$$\mathsf{W}_p \rightarrow \mathsf{Perm}_{w^p_{1,g}} \qquad w^p_{\phi,g}(i) = \begin{cases} 0, & i = 0, \\ \phi g^i \mod p, & \text{else.} \end{cases}$$

Example

Given DFT_{pq}

p – prime

p-1 = rs



Formula fragment

$$(\mathrm{I}_p \otimes (\mathrm{I}_1 \oplus (\mathrm{I}_r \otimes \mathrm{DFT}_s)\, \mathrm{L}^{rs}_r)\, \mathrm{W}_p)\, \mathrm{V}_{p,q}$$

Code for one memory access

```
p=7; q=4; r=3; s=2;
t=x[((21*((7*k + (((((((2*j + i)/2) + 3*((2*j
     + i)%2)) + 1)) ? (5*pow(3, ((((2*j +
     i)/2) + 3*((2*j + i)%2)) + 1)))%7 :
     (0)))/7) + 8*((7*k + ((((((2*j + i)/2) +
     3*((2*j + i)%2)) + 1)) ? (5*pow(3,
     ((((2*j + i)/2) + 3*((2*j + i)%2)) +
     1)))%7 : (0)))%7))%28)];
```

Task: Index simplification

Index Simplification: Basic Idea

Example: Identity necessary for fusing successive

Rader and prime-factor step

$$\left(\varphi g^{(b+si)} \bmod N'\right) \bmod N = \left((\varphi g^{b})(g^{s})^{i}\right) \bmod N$$

$$s|N',\ \ N'|N,\ \ 0 \leq i < n$$

Performed at the Σ-SPL level through rewrite rules on function objects:

$$\overline{w}_{\phi,g}^{N' \rightarrow N} \circ \overline{h}_{b,s}^{n \rightarrow N'} \quad \longrightarrow \quad \overline{w}_{\phi g^{b}, g^{s}}^{n \rightarrow N}$$

Advantages:

no analysis necessary

efficient (or doable at all)

```
// Input: _Complex double x[28], output: y[28]
double t1[28];
for(int i5 = 0; i5 <= 27; i5++)
    t1[i5] = x[(7*3*(i5/7) + 4*2*(i5%7))%28];
for(int i1 = 0; i1 <= 3; i1++) {
    double t3[7], t4[7], t5[7];
    for(int i6 = 0; i6 <= 6; i6++)
        t5[i6] = t1[7*i1 + i6];
    for(int i8 = 0; i8 <= 6; i8++)
        t4[i8] = t5[i8 ? (5*pow(3, i8))%7 : 0];
    {
        double t7[1], t8[1];
        t8[0] = t4[0];
        t7[0] = t8[0];
        t3[0] = t7[0];
    }
    {
        double t10[6], t11[6], t12[6];
        for(int i13 = 0; i13 <= 5; i13++)
            t12[i13] = t4[i13 + 1];
        for(int i14 = 0; i14 <= 5; i14++)
            t11[i14] = t12[(i14/2) + 3*(i14%2)];
        for(int i3 = 0; i3 <= 2; i3++) {
            double t14[2], t15[2];
            for(int i15 = 0; i15 <= 1; i15++)
                t15[i15] = t11[2*i3 + i15];
            t14[0] = (t15[0] + t15[1]);
            t14[1] = (t15[0] - t15[1]);
            for(int i17 = 0; i17 <= 1; i17++)
                 t10[2*i3 + i17] = t14[i17];
        }
        for(int i19 = 0; i19 <= 5; i19++)
            t3[i19 + 1] = t10[i19];
    }
    for(int i20 = 0; i20 <= 6; i20++)
        y[7*i1 + i20] = t3[i20];
}
```

```
// Input: _Complex double x[28], output: y[28]
  int p1, b1;
  for(int j1 = 0; j1 <= 3; j1++) {
     y[7*j1] = x[(7*j1%28)];
     p1 = 1; b1 = 7*j1;
     for(int j0 = 0; j0 <= 2; j0++) {
        y[b1 + 2*j0 + 1] = x[(b1 + 4*p1)%28] +
                           x[(b1 + 24*p1)%28];
        y[b1 + 2*j0 + 2] = x[(b1 + 4*p1)%28] –
                           x[(b1 + 24*p1)%28];
        p1 = (p1*3%7);
     }
  }
```

After, 2 Loops

Before, 11 Loops

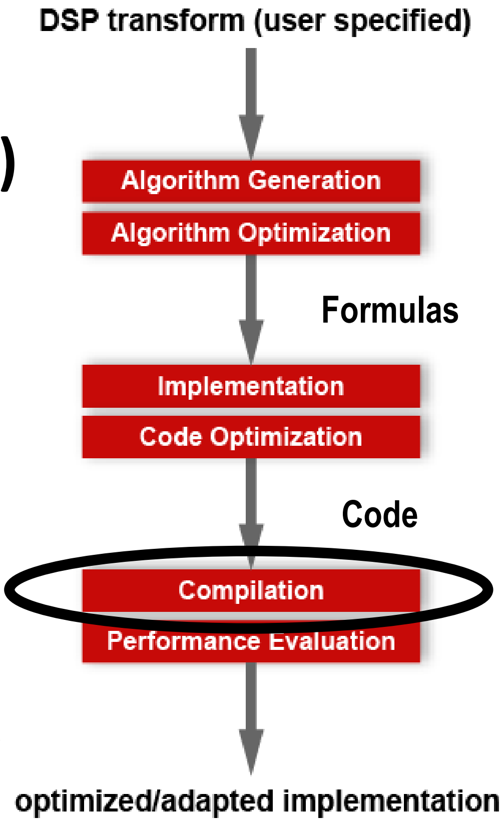
Organization

- SPL: Problem and algorithm specification
- Σ-SPL: Automating high level optimization
- **Rewriting: Formal parallelization**
- Rewriting: Vectorization
- Verification
- Spiral as FFTX backend
- Summary

F. Franchetti, Y. Voronenko, S. Chellappa, J. M. F. Moura, and M. Püschel
Discrete Fourier Transform on Multicores: Algorithms and Automatic Implementation
IEEE Signal Processing Magazine, special issue on “Signal Processing on Platforms with Multiple Cores”, 2009.

One Approach for All Types of Parallelism

- Shared Memory (Multicore)
- Vector SIMD (SSE, VMX, Double FPU…)
- Message Passing (Clusters)
- Graphics Processors (GPUs)
- FPGA
- HW/SW partitioning
- Multiple Levels of Parallelism (Cell BE)

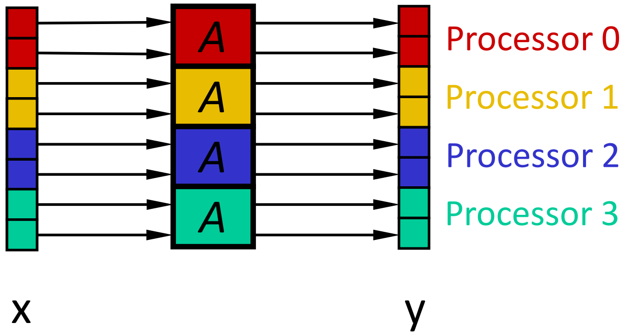


Spiral: One methodology optimizes for all types of parallelism

SPL to Shared Memory Code: Basic Idea

- **Good construct: tensor product**

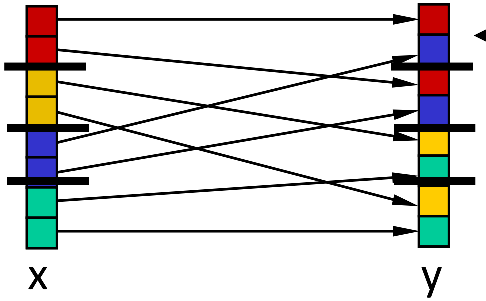
$$y = \left(\mathrm{I}_p \otimes A\right) x$$



p-way embarrassingly parallel, load-balanced

- **Problematic construct: permutations produce false sharing**

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



processors 0 and 1 write to same cache line

Task: Rewrite formulas to extract tensor product + treat cache lines as atomic

Step 1: Shared Memory Tags

- **Identify crucial hardware parameters**
  - Number of processors: *p*
  - Cache line size: *μ*

- **Introduce them as tags in SPL:**

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

This means: formula *A* is to be optimized for *p* processors and cache line size *μ*

- **Tags express hardware constraints within the rewriting system**

Step 2: Identify “Good” Formulas

- **Load balanced, avoiding false sharing**

$$y = \left(\mathrm{I}_p \otimes A\right)x \quad \text{with} \quad A \in \mathbb{C}^{m\mu \times m\mu}$$

$$y = \left(\bigoplus_{i=0}^{p-1} A_i\right)x \quad \text{with} \quad A_i \in \mathbb{C}^{m\mu \times m\mu}$$

$$y = \left(P \otimes \mathrm{I}_\mu\right)x \quad \text{with } P \text{ a permutation matrix}$$

- **Tagged operators (no further rewriting necessary)**

$$\mathrm{I}_p \otimes_{\|} A, \qquad \bigoplus_{i=0}^{p-1}{}_{\|} A_i, \qquad P \bar{\otimes} \mathrm{I}_\mu$$

- **Definition: A formula is fully optimized if it is one of the above or of the form**

$$\mathrm{I}_m \otimes A \quad \text{or} \quad AB$$

where A and B are fully optimized.

Step 3: Identify Rewriting Rules

- **Goal: Transform formulas into fully optimized formulas**
  - Formulas rewritten, tags propagated
  - There may be choices

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

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

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

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

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

Simple Rewriting Example

$$\underbrace{A_m \otimes \mathrm{I}_n}_{\mathsf{smp}(p,\mu)}$$

$$\Big\Downarrow$$

Loop tiling and scheduling
hw-conscious (knows *p* and μ)

$$\left(\mathsf{L}_m^{mp} \otimes \mathrm{I}_{n/p}\right)\left(\mathrm{I}_p \otimes_{\|} (A_m \otimes \mathrm{I}_{n/p})\right)\left(\mathsf{L}_p^{mp} \otimes \mathrm{I}_{n/p}\right)$$

fully optimized

```
parallel for (i=0; i<p; i++)
    for (j=0; j<n/p; j++)
        y[i*n/p+j:n:i*n/p+j+m-1] =
            A(x[i*n/p+j:n:i*n/p+j+m-1]);
```

Parallelization by Rewriting

$$\underbrace{\mathrm{DFT}_{mn}}_{\mathsf{smp}(p,\mu)} \rightarrow \underbrace{\Big((\mathrm{DFT}_m \otimes \mathrm{I}_n)\mathsf{T}^{mn}_n(\mathrm{I}_m \otimes \mathrm{DFT}_n)\,\mathsf{L}^{mn}_m\Big)}_{\mathsf{smp}(p,\mu)}$$

$$\ldots$$

$$\rightarrow \underbrace{\Big(\mathrm{DFT}_m \otimes \mathrm{I}_n\Big)}_{\mathsf{smp}(p,\mu)} \underbrace{\mathsf{T}^{mn}_n}_{\mathsf{smp}(p,\mu)} \underbrace{\Big(\mathrm{I}_m \otimes \mathrm{DFT}_n\Big)}_{\mathsf{smp}(p,\mu)} \underbrace{\mathsf{L}^{nm}_m}_{\mathsf{smp}(p,\mu)}$$

$$\ldots$$

$$\rightarrow \Big(\underline{(\mathsf{L}^{mp}_m \otimes \mathrm{I}_{n/p\mu})\bar{\otimes}\,\mathrm{I}_\mu}\Big)\Big(\underline{\mathrm{I}_p \otimes_{\|} (\mathrm{DFT}_m \otimes \mathrm{I}_{n/p})}\Big)\Big(\underline{(\mathsf{L}^{mp}_p \otimes \mathrm{I}_{n/p\mu})\bar{\otimes}\,\mathrm{I}_\mu}\Big)$$

$$\Big(\underline{\bigoplus_{i=0}^{p-1}{}_{\|}\ \mathsf{T}^{mn,i}_n}\Big)\Big(\underline{\mathrm{I}_p \otimes_{\|} (\mathrm{I}_{m/p} \otimes \mathrm{DFT}_n)}\Big)\Big(\underline{\mathrm{I}_p \otimes_{\|} \mathsf{L}^{mn/p}_{m/p}}\Big)\Big(\underline{(\mathsf{L}^{pn}_p \otimes \mathrm{I}_{m/p\mu})\bar{\otimes}\,\mathrm{I}_\mu}\Big)$$

**Fully optimized (load-balanced, no false sharing)
in the sense of our definition**

Same Approach for Other Parallel Paradigms

Message Passing:

$$\begin{aligned}
\underbrace{\mathbf{DFT}_{mn}}_{\mathsf{msg}(p,\mu)} &\rightarrow \underbrace{\Big((\mathbf{DFT}_m \otimes \mathrm{I}_n)\mathsf{T}^{mn}_n(\mathrm{I}_m \otimes \mathbf{DFT}_n)\,\mathsf{L}^{mn}_m\Big)}_{\mathsf{msg}(p,\mu)}\\
&\dots\\
&\rightarrow \underbrace{\Big(\mathbf{DFT}_m \otimes \mathrm{I}_n\Big)}_{\mathsf{msg}(p,\mu)}\,\underbrace{\mathsf{T}^{mn}_n}_{\mathsf{msg}(p,\mu)}\,\underbrace{\Big(\mathrm{I}_m \otimes \mathbf{DFT}_n\Big)}_{\mathsf{msg}(p,\mu)}\,\underbrace{\mathsf{L}^{nm}_m}_{\mathsf{msg}(p,\mu)}\\
&\dots\\
&\rightarrow \Big((\mathsf{L}^{mp}_m \otimes \mathrm{I}_{n/p\mu})\bar{\otimes}\,\mathrm{I}_\mu\Big)\Big(\mathrm{I}_p \otimes_{\|} (\mathbf{DFT}_m \otimes \mathrm{I}_{n/p})\Big)\Big((\mathsf{L}^{mp}_p \otimes \mathrm{I}_{n/p\mu})\bar{\otimes}\,\mathrm{I}_\mu\Big)\\
&\quad\left(\bigoplus_{i=0}^{p-1}{}_{\|}\,\mathsf{T}^{mn,i}_n\right)\Big(\mathrm{I}_p \otimes_{\|} (\mathrm{I}_{m/p} \otimes \mathbf{DFT}_n)\Big)\Big(\mathrm{I}_p \otimes_{\|} \mathsf{L}^{mn/p}_{m/p}\Big)\Big((\mathsf{L}^{pn}_p \otimes \mathrm{I}_{m/p\mu})\bar{\otimes}\,\mathrm{I}_\mu\Big)
\end{aligned}$$

Vectorization:

$$\begin{aligned}
\underbrace{\Big(\overline{\mathbf{DFT}_{mn}}\Big)}_{\mathsf{vec}(\nu)} &\rightarrow \underbrace{\overline{\Big((\mathbf{DFT}_m \otimes \mathrm{I}_n)\mathsf{T}^{mn}_n(\mathrm{I}_m \otimes \mathbf{DFT}_n)\,\mathsf{L}^{mn}_m\Big)}}_{\mathsf{vec}(\nu)}\\
&\dots\\
&\rightarrow \overleftarrow{(\underbrace{\mathsf{DFT}_m \otimes \mathrm{I}_n}_{\mathsf{vec}(\nu)})}^{\nu}\,\overline{(\underbrace{\mathsf{T}^{mn}_n}_{\mathsf{vec}(\nu)})}^{\nu}\,\overrightarrow{\underbrace{(\mathrm{I}_m \otimes \mathsf{DFT}_n)\,\mathsf{L}^{mn}_m}_{\mathsf{vec}(\nu)}}^{\nu}\\
&\dots\\
&\rightarrow (\mathrm{I}_{mn/\nu} \otimes \underbrace{\mathsf{L}^{2\nu}_\nu}_{\mathsf{sse}})(\overline{\mathsf{DFT}_m \otimes \mathrm{I}_{n/\nu}}\vec{\otimes}\,\mathrm{I}_\nu)\overline{(\underbrace{\mathsf{T}^{mn}_n}_{\mathsf{sse}})}^{\nu}\\
&\quad\Big(\mathrm{I}_{m/\nu} \otimes (\overline{\mathsf{L}^n_\nu}\vec{\otimes}\,\mathrm{I}_\nu)(\mathrm{I}_{n/\nu} \otimes (\mathsf{L}^{2\nu}_\nu \vec{\otimes}\,\mathrm{I}_\nu)(\mathrm{I}_2 \otimes \underbrace{\mathsf{L}^{\nu^2}_\nu}_{\mathsf{sse}})(\mathsf{L}^{2\nu}_2 \vec{\otimes}\,\mathrm{I}_\nu))(\overline{\mathsf{DFT}_n}\vec{\otimes}\,\mathrm{I}_\nu)\Big)\\
&\quad\Big((\mathsf{L}^{mn}_m \otimes \mathrm{I}_2)\vec{\otimes}\,\mathrm{I}_\nu\Big)(\mathrm{I}_{mn/\nu} \otimes \underbrace{\mathsf{L}^{2\nu}_2}_{\mathsf{sse}})
\end{aligned}$$

GPUs:

$$\begin{aligned}
\underbrace{\Big(\mathbf{DFT}_{r^k}\Big)}_{\mathsf{gpu}(t,c)} &\rightarrow \underbrace{\left(\prod_{i=0}^{k-1}\mathsf{L}^{r^k}_r\Big(\mathrm{I}_{r^{k-1}} \otimes \mathbf{DFT}_r\Big)\Big(\mathsf{L}^{r^k}_{r^{k-i-1}}(\mathrm{I}_{r^i} \otimes \mathsf{T}^{r^{k-i}}_{r^{k-i-1}})\,\underbrace{\mathsf{L}^{r^k}_{r^{i+1}}}_{\mathsf{vec}(c)}\Big)\right)\mathsf{R}^{r^k}_r}_{\mathsf{gpu}(t,c)}\\
&\dots\\
&\rightarrow \left(\prod_{i=0}^{k-1}(\mathsf{L}^{r^n/2}_r \vec{\otimes}\,\mathrm{I}_2)\Big(\mathrm{I}_{r^{n-1}/2} \otimes_{\times} \underbrace{(\mathbf{DFT}_r \vec{\otimes}\,\mathrm{I}_2)\,\mathsf{L}^{2r}_r}_{\mathsf{shd}(t,c)}\Big)\,\mathsf{T}_i\right)\\
&\quad(\mathsf{L}^{r^n/2}_r \vec{\otimes}\,\mathrm{I}_2)(\mathrm{I}_{r^{n-1}/2} \otimes_{\times} \underbrace{\mathsf{L}^{2r}_r}_{\mathsf{shd}(t,c)})(\mathsf{R}^{r^{n-1}}_r \vec{\otimes}\,\mathrm{I}_r)
\end{aligned}$$

Verilog for FPGAs:

$$\begin{aligned}
\underbrace{\Big(\mathbf{DFT}_{r^k}\Big)}_{\mathsf{stream}(r^s)} &\rightarrow \underbrace{\left[\prod_{i=0}^{k-1}\mathsf{L}^{r^k}_r\Big(\mathrm{I}_{r^{k-1}} \otimes \mathbf{DFT}_r\Big)\Big(\mathsf{L}^{r^k}_{r^{k-i-1}}(\mathrm{I}_{r^i} \otimes \mathsf{T}^{r^{k-i}}_{r^{k-i-1}})\,\mathsf{L}^{r^k}_{r^{i+1}}\Big)\right]\mathsf{R}^{r^k}_r}_{\mathsf{stream}(r^s)}\\
&\dots\\
&\rightarrow \left[\prod_{i=0}^{k-1}\underbrace{\mathsf{L}^{r^k}_r}_{\mathsf{stream}(r^s)}\underbrace{\Big(\mathrm{I}_{r^{k-1}} \otimes \mathbf{DFT}_r\Big)}_{\mathsf{stream}(r^s)}\underbrace{\Big(\mathsf{L}^{r^k}_{r^{k-i-1}}(\mathrm{I}_{r^i} \otimes \mathsf{T}^{r^{k-i}}_{r^{k-i-1}})\,\mathsf{L}^{r^k}_{r^{i+1}}\Big)}_{\mathsf{stream}(r^s)}\right]\underbrace{\mathsf{R}^{r^k}_r}_{\mathsf{stream}(r^s)}\\
&\dots\\
&\rightarrow \left[\prod_{i=0}^{k-1}\underbrace{\mathsf{L}^{r^k}_r}_{\mathsf{stream}(r^s)}\Big(\mathrm{I}_{r^{k-s-1}} \otimes_s (\mathrm{I}_{r^{s-1}} \otimes \mathbf{DFT}_r)\Big)\underbrace{\mathsf{T}'_i}_{\mathsf{stream}(r^s)}\right]\underbrace{\mathsf{R}^{r^k}_r}_{\mathsf{stream}(r^s)}
\end{aligned}$$

- Rigorous, correct by construction
- Overcomes compiler limitations

Autotuning in Constraint Solution Space

AVX 2-way
`_Complex double`

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

DFT_8

Base cases

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

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

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

Transformation rules

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

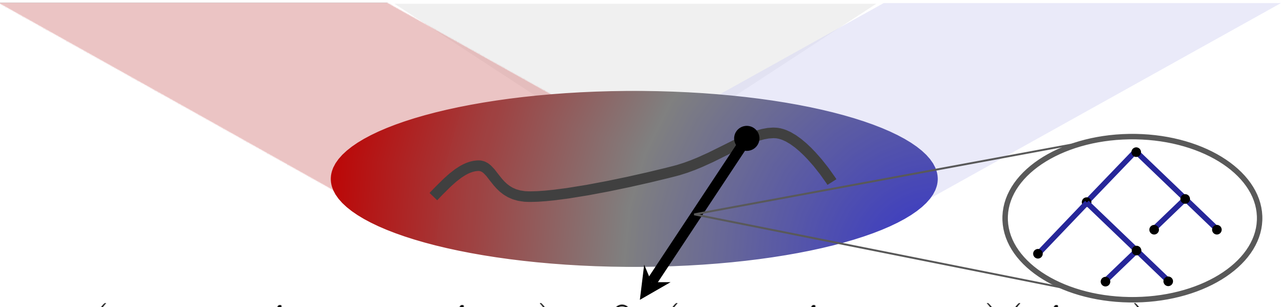
$$\mathrm{L}_\nu^{n\nu} \rightarrow (\mathrm{L}_\nu^n \otimes \mathrm{I}_\nu)(\mathrm{I}_{n/\nu} \otimes \mathrm{L}_\nu^{\nu^2})$$

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

Breakdown rules

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

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



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

SPL specification

Expansion + backtracking

SPL (dataflow) expression

Recursive descent

Σ-SPL (loop) expression

Confluent term rewriting

Optimized Σ-SPL 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

SPL Expression:

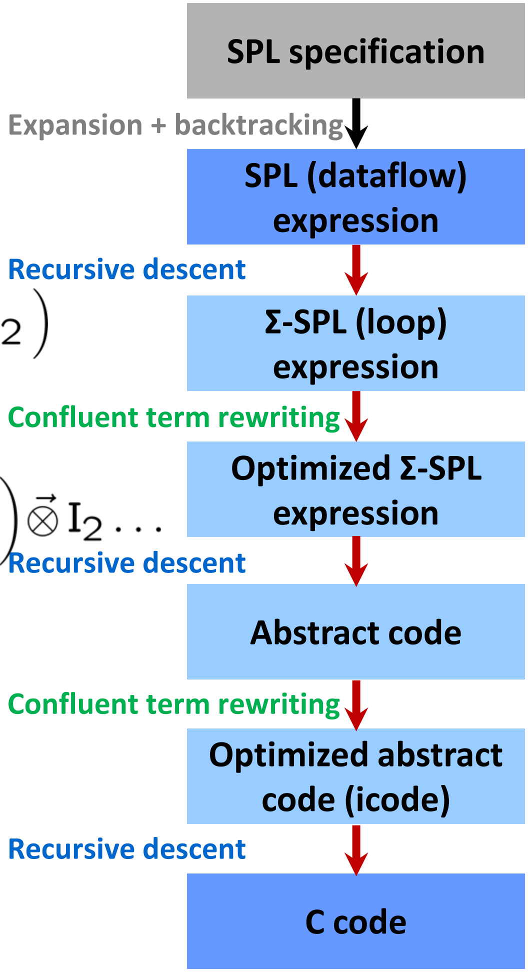
$$\Big((\mathrm{F}_2\otimes\mathrm{I}_2)\mathrm{T}^4_2(\mathrm{I}_2\otimes\mathrm{F}_2)\mathrm{L}^4_2\vec{\otimes}\,\mathrm{I}_2\Big)\ \underbrace{\mathrm{T}^8_2}_{\mathsf{vec}(2)}\ \Big(\mathrm{I}_2\otimes\ \underbrace{\mathrm{L}^4_2}_{\mathsf{vec}(2)}\ (\mathrm{F}_2\vec{\otimes}\,\mathrm{I}_2)\Big)\Big(\mathrm{L}^4_2\vec{\otimes}\,\mathrm{I}_2\Big)$$

∑-SPL:

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

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;
}
```



SPL specification

Expansion + backtracking

SPL (dataflow) expression

Recursive descent

Σ-SPL (loop) expression

Confluent term rewriting

Optimized Σ-SPL expression

Recursive descent

Abstract code

Confluent term rewriting

Optimized abstract code (icode)

Recursive descent

C code

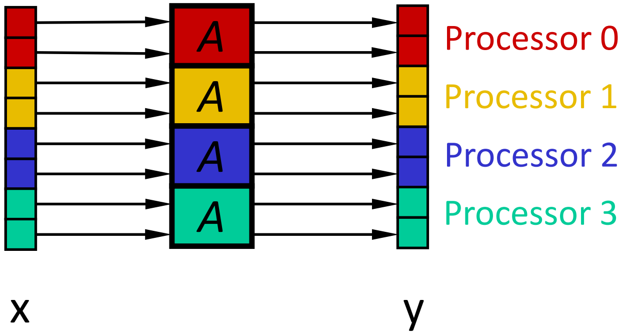
Discussion

- Parallelization at the mathematical level through rewriting and constraint programming
- Generates a space of “reasonable algorithms” that can be searched for adaptation to memory hierarchy
- Very efficient since no analysis is required
- Principled, domain-specific approach
- Applicable across transforms and parallelism types

Message Passing

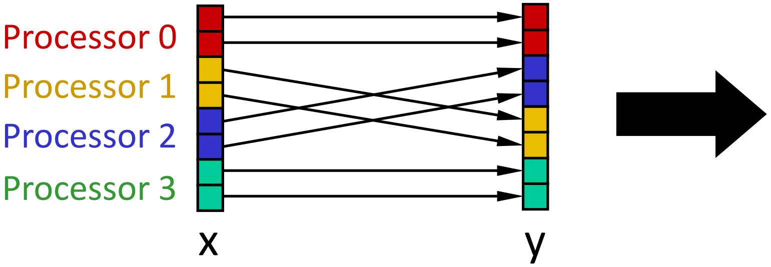
- **Good construct: tensor product**

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



Characteristics: no communication

- **Permutations are explicit communication** $y = \left(\mathrm{L}^4_2 \otimes \mathrm{I}_2\right) x$



```
// same program for all processors (SPMD)
switch (processor_num) {
    case 0: copy(y_local, x_local); break;
    case 1: SENDRECV(y_local, x_local, 2); break;
    case 2: SENDRECV(y_local, x_local, 1); break;
    case 3: copy(y_local, x_local); break;
}
```

Apply same 3-step approach:
1. ***Identify hw parameters***
2. ***Identify good formulas***
3. ***Identify rewriting rules***

Parallelization for Distributed Memory

$$\underbrace{\mathrm{DFT}_{mn}}_{\mathrm{par}(p)} \rightarrow \underbrace{(\mathrm{DFT}_m \otimes \mathrm{I}_n)}_{\mathrm{par}(p\leftarrow q)} \underbrace{\mathrm{T}^{mn}_n}_{\mathrm{par}(q)} \underbrace{(\mathrm{I}_m \otimes \mathrm{DFT}_n)}_{\mathrm{par}(q)} \underbrace{\mathrm{L}^{mn}_m}_{\mathrm{par}(q\leftarrow p)}$$

$$\ldots$$

$$\ldots$$

$$\ldots$$

$$\begin{aligned}\rightarrow\; & \underline{\left(\mathrm{I}_p \otimes_{\|} \mathrm{L}^{mn/p}_{m/p}\right)} \underbrace{\left(\mathrm{L}^{p^2}_p \otimes \mathrm{I}_{mn/p^2}\right)}_{\mathrm{comm}(p\leftarrow q)} \underline{\left(\mathrm{I}_q \otimes_{\|} (\mathrm{I}_{p/q} \otimes \mathrm{L}^n_p \otimes \mathrm{I}_{m/p})\right)\left(\mathrm{I}_q \otimes_{\|} (\mathrm{I}_{n/q} \otimes \mathrm{DFT}_m)\right)} \\ & \underline{\left(\mathrm{I}_q \otimes_{\|} \mathrm{L}^{mn/q}_{m/q}\right)} \underbrace{\left(\mathrm{L}^{q^2}_q \otimes \mathrm{I}_{mn/q^2}\right)}_{\mathrm{comm}(q)} \underline{\left(\mathrm{I}_q \otimes_{\|} (\mathrm{L}^n_q \otimes \mathrm{I}_{m/q})\right) \mathrm{T}^{mn}_n \left(\mathrm{I}_q \otimes_{\|} (\mathrm{I}_{m/q} \otimes \mathrm{DFT}_n)\right)} \\ & \underline{\left(\mathrm{I}_q \otimes_{\|} (\mathrm{I}_{p/q} \otimes \mathrm{L}^{mn/p}_{m/p})\right)} \underbrace{\left(\mathrm{L}^{p^2}_p \otimes \mathrm{I}_{mn/p^2}\right)}_{\mathrm{comm}(q\leftarrow p)} \underline{\left(\mathrm{I}_p \otimes_{\|} (\mathrm{L}^n_p \otimes \mathrm{I}_{m/p})\right)}\end{aligned}$$

***p*-way parallelized**

with intermediate computation on *q* processors

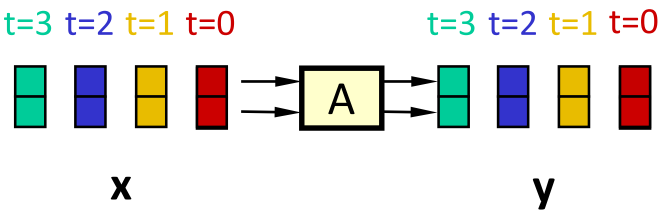
Parallelization/Streaming for FPGAs

- Data streams steadily in packets

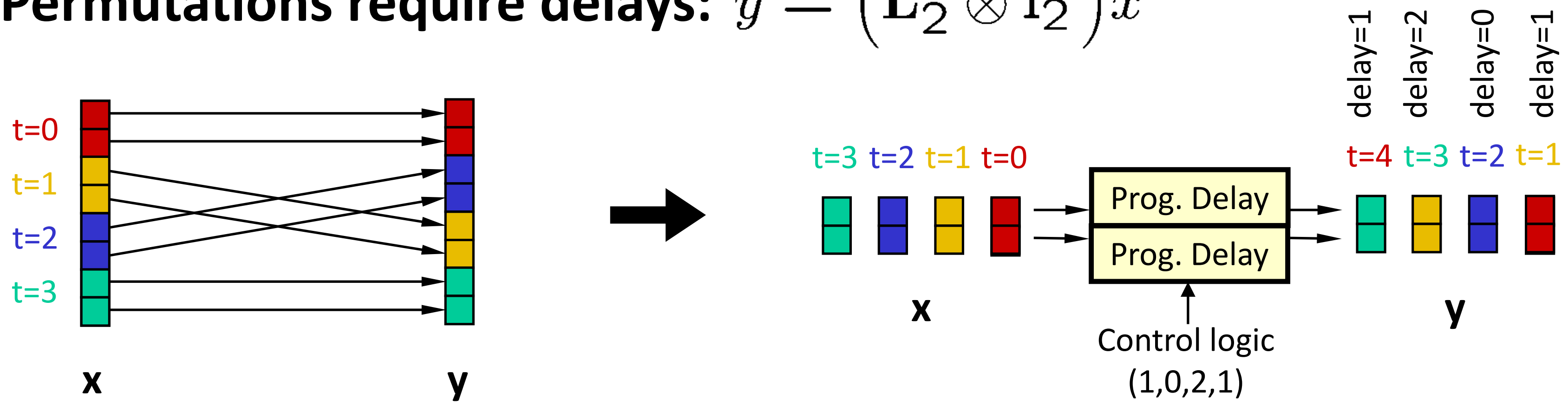
$$y = \left(\mathrm{I}_p \otimes A^{n \times n} \right) x$$

Packet size

Number of packets = time steps



- Permutations require delays: $y = \left(\mathrm{L}^4_2 \otimes \mathrm{I}_2 \right) x$



Apply same 3-step approach:

1. *Identify hw parameters*
2. *Identify good formulas*
3. *Identify rewriting rules*

Streaming by Rewriting

$$\underbrace{\left(\mathbf{DFT}_{r^k}\right)}_{\mathsf{stream}(r^s)} \;\rightarrow\; \underbrace{\left[\prod_{i=0}^{k-1} \mathsf{L}^{r^k}_{r}\left(\mathbf{I}_{r^{k-1}} \otimes \mathbf{DFT}_r\right)\left(\mathsf{L}^{r^k}_{r^{k-i-1}}(\mathbf{I}_{r^i} \otimes \mathsf{T}^{r^{k-i}}_{r^{k-i-1}})\,\mathsf{L}^{r^k}_{r^{i+1}}\right)\right]\mathsf{R}^{r^k}_{r}}_{\mathsf{stream}(r^s)}$$

$$\dots$$

$$\rightarrow\; \left[\prod_{i=0}^{k-1} \underbrace{\mathsf{L}^{r^k}_{r}}_{\mathsf{stream}(r^s)} \underbrace{\left(\mathbf{I}_{r^{k-1}} \otimes \mathbf{DFT}_r\right)}_{\mathsf{stream}(r^s)} \underbrace{\left(\mathsf{L}^{r^k}_{r^{k-i-1}}(\mathbf{I}_{r^i} \otimes \mathsf{T}^{r^{k-i}}_{r^{k-i-1}})\,\mathsf{L}^{r^k}_{r^{i+1}}\right)}_{\mathsf{stream}(r^s)}\right] \underbrace{\mathsf{R}^{r^k}_{r}}_{\mathsf{stream}(r^s)}$$

$$\dots$$

$$\rightarrow\; \left[\prod_{i=0}^{k-1} \underline{\underbrace{\mathsf{L}^{r^k}_{r}}_{\mathsf{stream}(r^s)}}\; \underline{\left(\mathbf{I}_{r^{k-s-1}} \otimes_s (\mathbf{I}_{r^{s-1}} \otimes \mathbf{DFT}_r)\right)}\; \underline{\underbrace{\mathsf{T}'_{i}}_{\mathsf{stream}(r^s)}}\right] \underline{\underbrace{\mathsf{R}^{r^k}_{r}}_{\mathsf{stream}(r^s)}}$$

streamed at r^s words per cycle

base cases

Target: Specialized Soft Processor

SPL formula $$y = (I_4 \otimes F_2)\, L^8_4\, x$$

C code generation

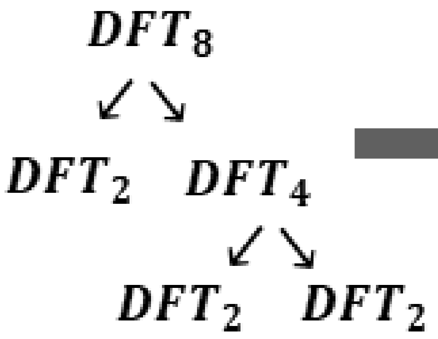
FPGA OpenCL code generation

```
// each call executes full for loop
void sub(double *y, double *x) {
   for (int j=0; j<=3; j++){
      y[2*j] = x[j] + x[j+4];
      y[2*j+1] = x[j] - x[j+4];
   }
}
```

```
// each call executes one iteration of the for loop
__kernel void sub(__global double *y,
    __global double *x, int control) {
   // reinitialize
   if (!control) { j = 0; return; }
   // for (int j=0; j<=3; j++)
   {
      static int j = 0;
      if (j<=3) {
         y[2*j] = x[j] + x[j+4];
         y[2*j+1] = x[j] - x[j+4];
         j++;
      }
   }
}
```

Hardware design issue → SPIRAL algorithm transformation + C coding style issue

OpenCL State Machine Code Generation



$$\sum_{i3=0}^{1}\left(\sum_{i6=0}^{1} S_{2,i6}A_2G_{2,i6}\cdot\sum_{i7=0}^{1}S_{1,i7}A_1G_{1,i7}\right)\cdot\sum_{i2=0}^{3}S_{0,i2}A_0G_{0,i2}$$

```
for(int i2 = 0; i2 <= 3; i2++) {
    BB(0);
}
for(int i3 = 0; i3 <= 1; i3++) {
    for(int i7 = 0; i7 <= 1; i7++) {
        BB(1);
    }
    for(int i6 = 0; i6 <= 1; i6++) {
        BB(2);
    }
}
```

```
while (true) {
    if (bben0==1 && i2<=3)  {
        // output logic
        output = f(i2);
        // next state logic
        if (i2<3) {
           i2++;
        } else if (i2==3) {
           i2=0; bben0=0;
           bben1=1;
        }
    else if (bben1==1 && i7<=1)  {
        // output logic
        // next state logic
    }
    else if (bben2==1 && i6<=1)  {
        // output logic
        // next state logic
     }
     write_channel(ouput);
}
```

Organization

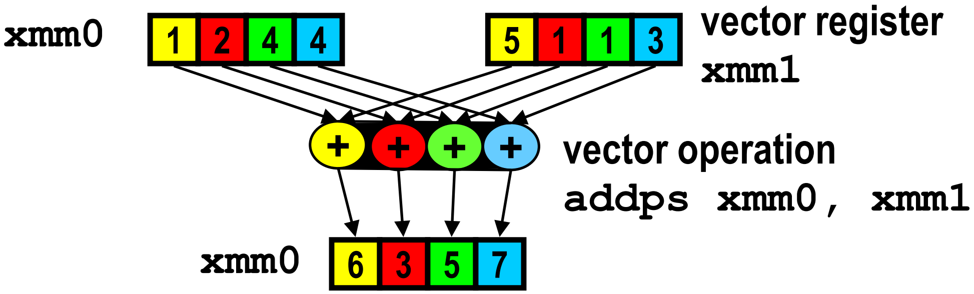
- SPL: Problem and algorithm specification
- Σ-SPL: Automating high level optimization
- Rewriting: Formal parallelization
- **Rewriting: Vectorization**
- Verification
- Spiral as FFTX backend
- Summary

F. Franchetti, M. Püschel
Short Vector Code Generation for the Discrete Fourier Transform
Proceedings of the 17th International Parallel and Distributed Processing Symposium (IPDPS '03), pages 58-67.

F. Franchetti and M. Püschel
Generating SIMD Vectorized Permutations
Proceedings of International Conference on Compiler Construction (CC) 2008

SIMD (Signal Instruction Multiple Data) Vector Instructions in a Nutshell

- **What are these instructions?**
  - Extension of the ISA. Data types and instructions for parallel computation on short **(2-way–16-way) vectors** of integers and floats



- Intel MMX
- AMD 3DNow!
- Intel SSE
- AMD Enhanced 3DNow!
- Motorola AltiVec/VMX
- AMD 3DNow! Professional
- Intel SSE2
- IBM BlueGene/L PPC440FP2
- IBM QPX
- IBM VSX
- Intel SSE3
- Intel SSSE3
- Intel SSE4, 4.1, 4.2
- Intel AVX, AVX2
- Intel AVX512

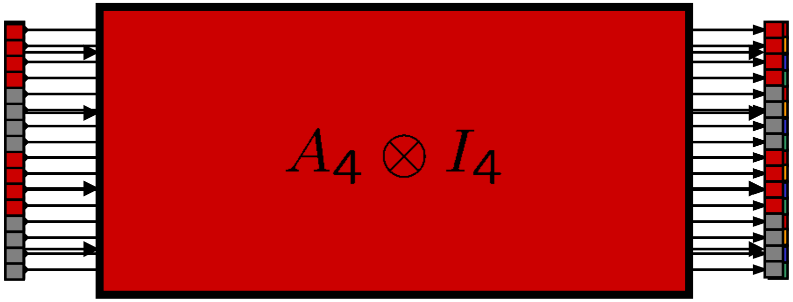
- **Problems:**
  - Not standardized
  - Compiler vectorization limited
  - Low-level issues (data alignment,…)
  - Reordering data kills runtime

One can easily slow down a program by vectorizing it

Vectorization: Basic Idea

- **Good construct: tensor product**

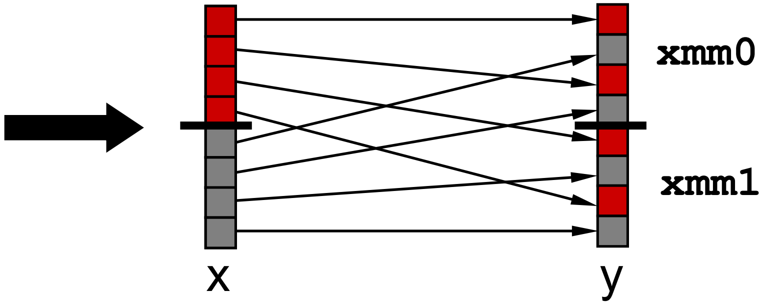
$$y = \left(A \otimes \mathrm{I}_\nu\right) x$$



Characteristics: block operation and alignment preserving

- **Problematic construct: permutations must be done in register**

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



Task: Rewrite formulas to
extract tensor product + minimize in-register shuffles

SIMD Vectorization: 3-Step Procedure

1. **Identify crucial hardware parameters** $\underbrace{A}_{\text{vec}(\nu)}$
   - **Vector length: ν**
2. **Identify good formulas**
   - **Tensor product:** $A \otimes \mathrm{I}_\nu$
   - **Base cases:** Build library from shuffle instructions **`unpacklo`**, **`unpackhi`**, **`shufps`**,…
   - **Definition:** Vectorized formula
3. **Identify rewriting rules**

$$\underbrace{\mathrm{I}_n \otimes A^{k\times m}}_{\text{vec}(\nu)} \rightarrow \underbrace{\mathrm{I}_{n/\nu} \otimes \mathrm{L}^{k\nu}_{\nu}}_{\text{vec}(\nu)} \left(A^{k\times m} \vec{\otimes}\, \mathrm{I}_\nu\right) \underbrace{\mathrm{L}^{m\nu}_{m}}_{\text{vec}(\nu)}$$

$$\underbrace{\mathrm{L}^{m\nu}_{m}}_{\text{vec}(\nu)} \rightarrow \left(\mathrm{I}_{m/\nu} \otimes \underbrace{\mathrm{L}^{\nu^2}_{\nu}}_{\text{sse}}\right)\left(\mathrm{L}^{m}_{m/\nu} \vec{\otimes}\, \mathrm{I}_\nu\right)$$

Base case for Intel SSE

Vectorization by Rewriting

$$\underbrace{\left(\overline{\mathbf{DFT}_{mn}}\right)}_{\mathrm{vec}(\nu)} \rightarrow \underbrace{\overline{\left((\mathbf{DFT}_m \otimes \mathrm{I}_n)\mathrm{T}^{mn}_n(\mathrm{I}_m \otimes \mathbf{DFT}_n)\,\mathrm{L}^{mn}_m\right)}}_{\mathrm{vec}(\nu)}$$

$$\cdots$$

$$\rightarrow \underbrace{\overleftarrow{(\mathrm{DFT}_m \otimes \mathrm{I}_n)}^{\nu}}_{\mathrm{vec}(\nu)} \underbrace{\overline{(\mathrm{T}^{mn}_n)}^{\nu}}_{\mathrm{vec}(\nu)} \underbrace{\overrightarrow{(\mathrm{I}_m \otimes \mathrm{DFT}_n)\,\mathrm{L}^{mn}_m}^{\nu}}_{\mathrm{vec}(\nu)}$$

$$\cdots$$

$$\rightarrow (\mathrm{I}_{mn/\nu} \otimes \underbrace{\mathrm{L}^{2\nu}_{\nu}}_{\mathrm{sse}}) \underline{\overline{(\mathrm{DFT}_m \otimes \mathrm{I}_{n/\nu} \vec{\otimes} \mathrm{I}_{\nu})}} \underbrace{\overline{(\mathrm{T}^{mn}_n)}^{\nu}}_{\mathrm{sse}}$$

$$\left(\mathrm{I}_{m/\nu} \otimes \underline{(\overline{\mathrm{L}^{n}_{\nu}} \vec{\otimes} \mathrm{I}_{\nu})})(\mathrm{I}_{n/\nu} \otimes \underline{(\mathrm{L}^{2\nu}_{\nu} \vec{\otimes} \mathrm{I}_{\nu})})(\mathrm{I}_2 \otimes \underbrace{\mathrm{L}^{\nu^2}_{\nu}}_{\mathrm{sse}})(\underline{\mathrm{L}^{2\nu}_2 \vec{\otimes} \mathrm{I}_{\nu})})\underline{(\overline{\mathrm{DFT}_n} \vec{\otimes} \mathrm{I}_{\nu})}\right)$$

$$\underline{\left((\mathrm{L}^{mn}_m \otimes \mathrm{I}_2) \vec{\otimes} \mathrm{I}_{\nu}\right)}(\mathrm{I}_{mn/\nu} \otimes \underbrace{\mathrm{L}^{2\nu}_2}_{\mathrm{sse}})$$

tensor products

base cases

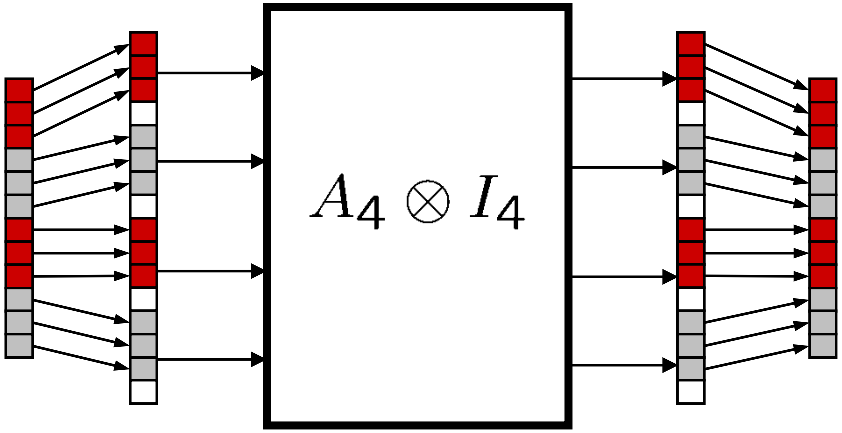
Formula is vectorized w.r.t. Definition

Vectorization of Odd Problem Sizes

Zero-padding: Load/store *km* elements into/from *m* ν-way vectors

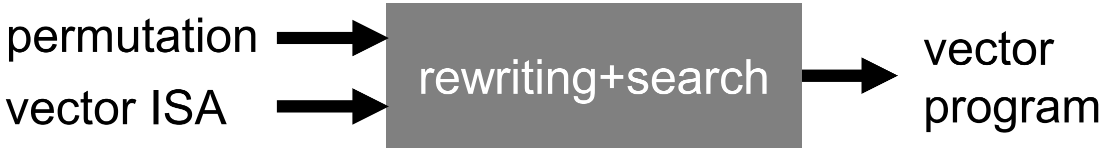
$$y = (\mathrm{I}_m \otimes \mathrm{I}_{\nu \times k})x$$

$$y = (\mathrm{I}_m \otimes \mathrm{I}_{k \times \nu})x$$



SPL formula	data type	compiled to
$\mathrm{I}_{k\times 4}$	4-way float	_mm_maskmoveu_si128 + _mm_castps_si128
$\mathrm{I}_{4\times k}$	4-way float	_mm_loadu_ps + _mm_and_i128
$\mathrm{I}_{k\times 8}$	8-way short	_mm_maskmoveu_si128
$\mathrm{I}_{8\times k}$	8-way short	_mm_loadu_si128

Automatically Deriving Vector Base Cases



- Translate SIMD vector ISA into matrix representation
- Design rule system to generate *vector matrix formulas*
- Define cost measure on matrix formulas
- Use dynamic programming with backtracking to find vector program with minimal cost

Vector matrix formula in BNF

$$\begin{aligned}
\langle\text{vmf}\rangle \;&::=\; \langle\text{vmf}\rangle\,\langle\text{vmf}\rangle \mid \mathrm{I}_m \otimes \langle\text{vmf}\rangle \mid \begin{pmatrix}\langle\text{vmf}\rangle\\ \langle\text{vmf}\rangle\end{pmatrix} \mid \langle\text{perm}\rangle \otimes \mathrm{I}_\nu \mid \\
&\quad\; \langle\text{perm}\rangle \otimes \mathrm{I}_{\nu/2} \;\text{ if }\; \mathbf{L}^4_2 \otimes \mathrm{I}_{\nu/2} \;\text{ possible} \mid M_{\mathtt{instr}} \;\text{ with }\mathtt{instr}\text{ in ISA} \\
\langle\text{perm}\rangle \;&::=\; \mathbf{L}^{mn}_m \mid \mathrm{I}_m \otimes \langle\text{perm}\rangle \mid \langle\text{perm}\rangle \otimes \mathrm{I}_m \mid \langle\text{perm}\rangle\langle\text{perm}\rangle
\end{aligned}$$

Translating Instructions into Matrices

Intel C++ Compiler Manual

```
__m128 _mm_unpackhi_ps(__m128 a, __m128 b)
r0 := a2; r1 := b2; r2 := a3; r3 := b3
```

Instruction specification (GAP code)

```
Intel_SSE2.4_x_float._mm_unpackhi_ps := rec(
   v := 4,
   semantics := (a, b, p) -> [a[2], b[2], a[3], b[3]],
   parameters := []
);
```

SSE instruction as matrix

```
__m128 t, x0, x1;
t = _mm_unpackhi_ps(x0, x1);
```

➡

$$\vec{t} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \vec{x}_0 \\ \vec{x}_1 \end{bmatrix}$$

Automatically build matrix from `semantics()` function

Example: Sequence of Two Instructions

Instruction set: Intel SSE 4-way float

```
y = _mm_unpacklo_ps(x0, x1);
```

$$\begin{bmatrix} 1&0&0&0&0&0&0&0\\ 0&0&0&0&1&0&0&0\\ 0&1&0&0&0&0&0&0\\ 0&0&0&0&0&1&0&0 \end{bmatrix}$$

```
y = _mm_shuffle_ps(x0, x1,
      _MM_SHUFFLE(1,2,1,2));
```

$$\begin{bmatrix} 1&0&0&0&0&0&0&0\\ 0&1&0&0&0&0&0&0\\ 0&0&0&0&1&0&0&0\\ 0&0&0&0&0&1&0&0 \end{bmatrix}$$

```
y = _mm_shuffle_ps(x0, x1,
      _MM_SHUFFLE(3,4,3,4));
```

$$\begin{bmatrix} 0&0&1&0&0&0&0&0\\ 0&0&0&1&0&0&0&0\\ 0&0&0&0&0&0&1&0\\ 0&0&0&0&0&0&0&1 \end{bmatrix}$$

Translating a vector matrix into a instruction sequence

$$L_2^4 \otimes I_2 = \left[\begin{array}{cccccccc} 1&0&0&0&0&0&0&0\\ 0&1&0&0&0&0&0&0\\ 0&0&0&0&1&0&0&0\\ 0&0&0&0&0&1&0&0\\ \hline 0&0&1&0&0&0&0&0\\ 0&0&0&1&0&0&0&0\\ 0&0&0&0&0&0&1&0\\ 0&0&0&0&0&0&0&1 \end{array}\right]$$

```
// __m128 *y, *x
Y[0] = _mm_shuffle_ps(x0, x1,
         _MM_SHUFFLE(1,2,1,2));
Y[1] = _mm_shuffle_ps(x0, x1,
         _MM_SHUFFLE(3,4,3,4));
```

Rule System: Recursive Matrix Factorization

- Recursively factorizes stride permutations
- “Blocking of matrix transposition” in linear memory
- Choices -> Dynamic programming with backtracking
- Trigger ISA-specific termination rules

Start: stride permutation

$$
\begin{aligned}
\boxed{\mathrm{L}_m^{mn}} &\rightarrow \mathrm{I}_1 \otimes \mathrm{L}_m^{mn} \otimes \mathrm{I}_1 \\
\mathrm{I}_\ell \otimes \mathrm{L}_n^{kmn} \otimes \mathrm{I}_r &\rightarrow \left(\mathrm{I}_\ell \otimes \mathrm{L}_n^{kn} \otimes \mathrm{I}_{mr}\right)\left(\mathrm{I}_{\ell k} \otimes \mathrm{L}_n^{mn} \otimes \mathrm{I}_r\right) \\
\mathrm{I}_\ell \otimes \mathrm{L}_n^{kmn} \otimes \mathrm{I}_r &\rightarrow \left(\mathrm{I}_\ell \otimes \mathrm{L}_{kn}^{kmn} \otimes \mathrm{I}_r\right)\left(\mathrm{I}_\ell \otimes \mathrm{L}_{mn}^{kmn} \otimes \mathrm{I}_r\right) \\
\mathrm{I}_\ell \otimes \mathrm{L}_{km}^{kmn} \otimes \mathrm{I}_r &\rightarrow \left(\mathrm{I}_{k\ell} \otimes \mathrm{L}_m^{mn} \otimes \mathrm{I}_r\right)\left(\mathrm{I}_\ell \otimes \mathrm{L}_k^{kn} \boxed{\otimes \mathrm{I}_m}\right) \\
\mathrm{I}_\ell \otimes \mathrm{L}_{km}^{\boxed{kmn}} \otimes \mathrm{I}_r &\rightarrow \left(\mathrm{I}_\ell \otimes \mathrm{L}_k^{kmn} \otimes \mathrm{I}_r\right)\left(\mathrm{I}_\ell \otimes \mathrm{L}_m^{kmn} \otimes \mathrm{I}_r\right) \\
\mathrm{I}_{k\ell} \otimes \mathrm{L}_m^{mn} \otimes \mathrm{I}_r &\rightarrow \mathrm{I}_k \otimes \boxed{\left(\mathrm{I}_\ell \otimes \mathrm{L}_m^{mn} \otimes \mathrm{I}_r\right)} \quad \text{if } \ell mnr \in \{\nu, 2\nu\}
\end{aligned}
$$

Choice: factorize kmn

Triggers termination rules

Cost Function: Weighted Instruction Count

- **Defines recursive cost function for matrix formulas**
- **Each instruction has an associated cost**
- **Vector assignments are “for free”**

$$\mathrm{Cost}_{\mathrm{ISA},\nu}(P) = \infty, \quad P \text{ not a } \langle \text{vmf} \rangle$$

$$\boxed{\begin{aligned} \mathrm{Cost}_{\mathrm{ISA},\nu}(M^{\nu}_{\mathtt{instr}}) &= c_{\mathtt{instr}} \\ \mathrm{Cost}_{\mathrm{ISA},\nu}(P \otimes \mathrm{I}_{\nu}) &= 0, \quad P \text{ permutation} \end{aligned}}$$

$$\mathrm{Cost}_{\mathrm{ISA},\nu}\left(P \otimes \mathrm{I}_{\nu/2}\right) = \lfloor n/2 \rfloor c_{\mathtt{i1}} + \lceil n/2 \rceil c_{\mathtt{i2}}, \quad P \ 2n \times 2n \text{ permutation}$$

$$\mathrm{Cost}_{\mathrm{ISA},\nu}(A\, B) = \mathrm{Cost}_{\mathrm{ISA},\nu}(A) + \mathrm{Cost}_{\mathrm{ISA},\nu}(B)$$

$$\mathrm{Cost}_{\mathrm{ISA},\nu}\left(\binom{A}{B}\right) = \mathrm{Cost}_{\mathrm{ISA},\nu}(A) + \mathrm{Cost}_{\mathrm{ISA},\nu}(B)$$

$$\mathrm{Cost}_{\mathrm{ISA},\nu}(\mathrm{I}_m \otimes A) = m\mathrm{Cost}_{\mathrm{ISA},\nu}(A)$$

Vector Program: 8-way Vectorized L^{64}_{8}

$$L^{64}_{8} = \underline{\left(I_4 \otimes (L^4_2 \otimes I_4)\right)}\left(L^8_4 \otimes I_8\right)\underline{\left(I_4 \otimes (L^8_4 \otimes I_2)\right)}\left((I_2 \otimes L^4_2) \otimes I_8\right)\underline{\left(I_4 \otimes L^{16}_8\right)}$$

```
__m128 X[8], Y[8], t3, t4, t7, t8, t11, t12, t15, t16,
       t17, t18, t19, t20, t21, t22, t23, t24;
t3 = _mm_unpacklo_epi16(X[0], X[1]); t4 = _mm_unpackhi_epi16(X[0], X[1]);
t7 = _mm_unpacklo_epi16(X[2], X[3]); t8 = _mm_unpackhi_epi16(X[2], X[3]);
t11 = _mm_unpacklo_epi16(X[4], X[5]);t12 = _mm_unpackhi_epi16(X[4], X[5]);
t15 = _mm_unpacklo_epi16(X[6], X[7]);t16 = _mm_unpackhi_epi16(X[6], X[7]);
t17 = _mm_unpacklo_epi32(t3, t7);    t18 = _mm_unpackhi_epi32(t3, t7);
t19 = _mm_unpacklo_epi32(t4, t8);    t20 = _mm_unpackhi_epi32(t4, t8);
t21 = _mm_unpacklo_epi32(t11, t15);  t22 = _mm_unpackhi_epi32(t11, t15);
t23 = _mm_unpacklo_epi32(t12, t16);  t24 = _mm_unpackhi_epi32(t12, t16);
Y[0] = _mm_unpacklo_epi64(t17, t21); Y[1] = _mm_unpackhi_epi64(t17, t21);
Y[2] = _mm_unpacklo_epi64(t18, t22); Y[3] = _mm_unpackhi_epi64(t18, t22);
Y[4] = _mm_unpacklo_epi64(t19, t23); Y[5] = _mm_unpackhi_epi64(t19, t23);
Y[6] = _mm_unpacklo_epi64(t20, t24); Y[7] = _mm_unpackhi_epi64(t20, t24);
```

8-way vectorized transposition of 8x8 matrix

Organization

- SPL: Problem and algorithm specification
- Σ-SPL: Automating high level optimization
- Rewriting: Formal parallelization
- Rewriting: Vectorization
- **Verification**
- Spiral as FFTX backend
- Summary

Symbolic Verification

- **Transform = Matrix-vector multiplication**
  matrix fully defines the operation

$$\mathrm{DFT}_4 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & j & -1 & -j \\ 1 & -1 & 1 & -1 \\ 1 & -j & -1 & j \end{bmatrix}$$

= ?

- **Algorithm = Formula**
  represents a matrix expression, can be evaluated to a matrix

$$(\mathrm{DFT}_2 \otimes \mathrm{I}_2)\, \mathrm{T}^4_2 (\mathrm{I}_2 \otimes \mathrm{DFT}_2)\, \mathrm{L}^4_2 = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & -1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & j \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Empirical Verification

- Run program on all basis vectors,
compare to columns of transform matrix

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

= ?

```
DFT4([0,1,0,0])
```

- Compare program output on random vectors
to output of a random implementation of same kernel

```
DFT4([0.1,1.77,2.28,-55.3])
```

= ?

```
DFT4_rnd([0.1,1.77,2.28,-55.3]))
```

Verification of the Generator

- **Rule replaces left-hand side by right-hand side when preconditions match**

$$\mathrm{I}_m \otimes A_n \to \mathrm{L}^{mn}_m (A_n \otimes \mathrm{I}_m)\, \mathrm{L}^{mn}_n$$

- **Test rule by evaluating expressions before and after rule application and compare result**

$$\mathrm{I}_2 \otimes \mathrm{DFT}_2 = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & -1 \end{bmatrix}$$

= ?

$$\mathrm{L}^4_2 (\mathrm{DFT}_2 \otimes \mathrm{I}_2)\, \mathrm{L}^4_2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Organization

- SPL: Problem and algorithm specification
- Σ-SPL: Automating high level optimization
- Rewriting: Formal parallelization
- Rewriting: Vectorization
- Verification
- **Spiral as FFTX backend**
- 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

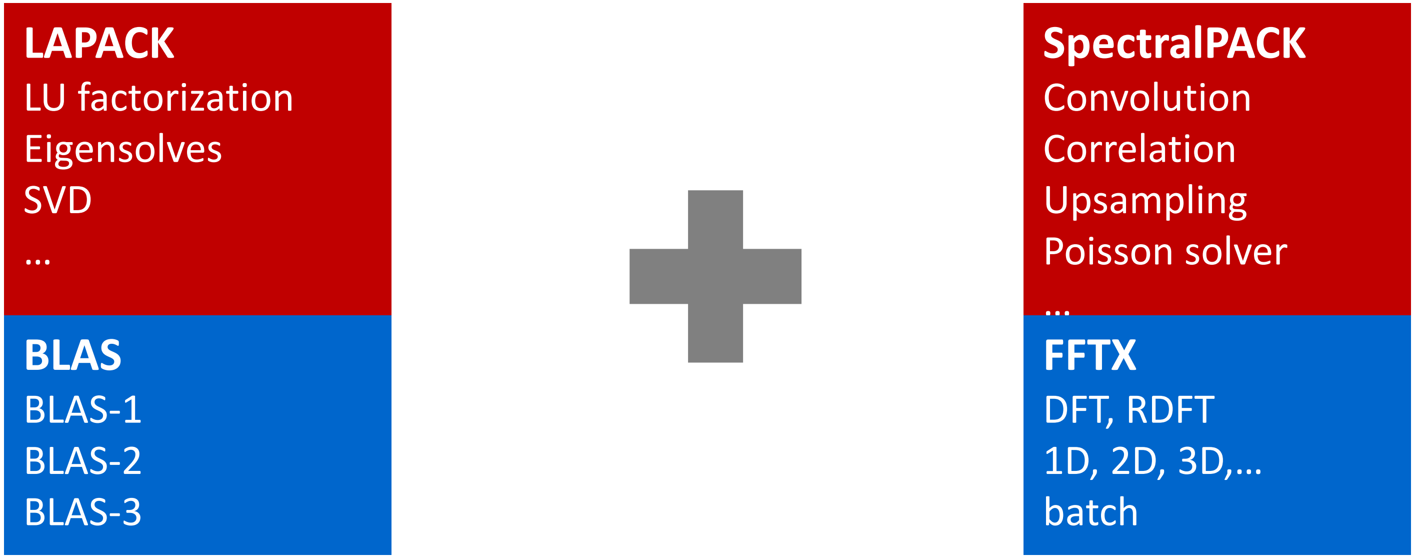
No LAPACK equivalent for spectral methods

- **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

Spectral Algorithms



LAPACK
LU factorization
Eigensolves
SVD
…

BLAS
BLAS-1
BLAS-2
BLAS-3

SpectralPACK
Convolution
Correlation
Upsampling
Poisson solver
…

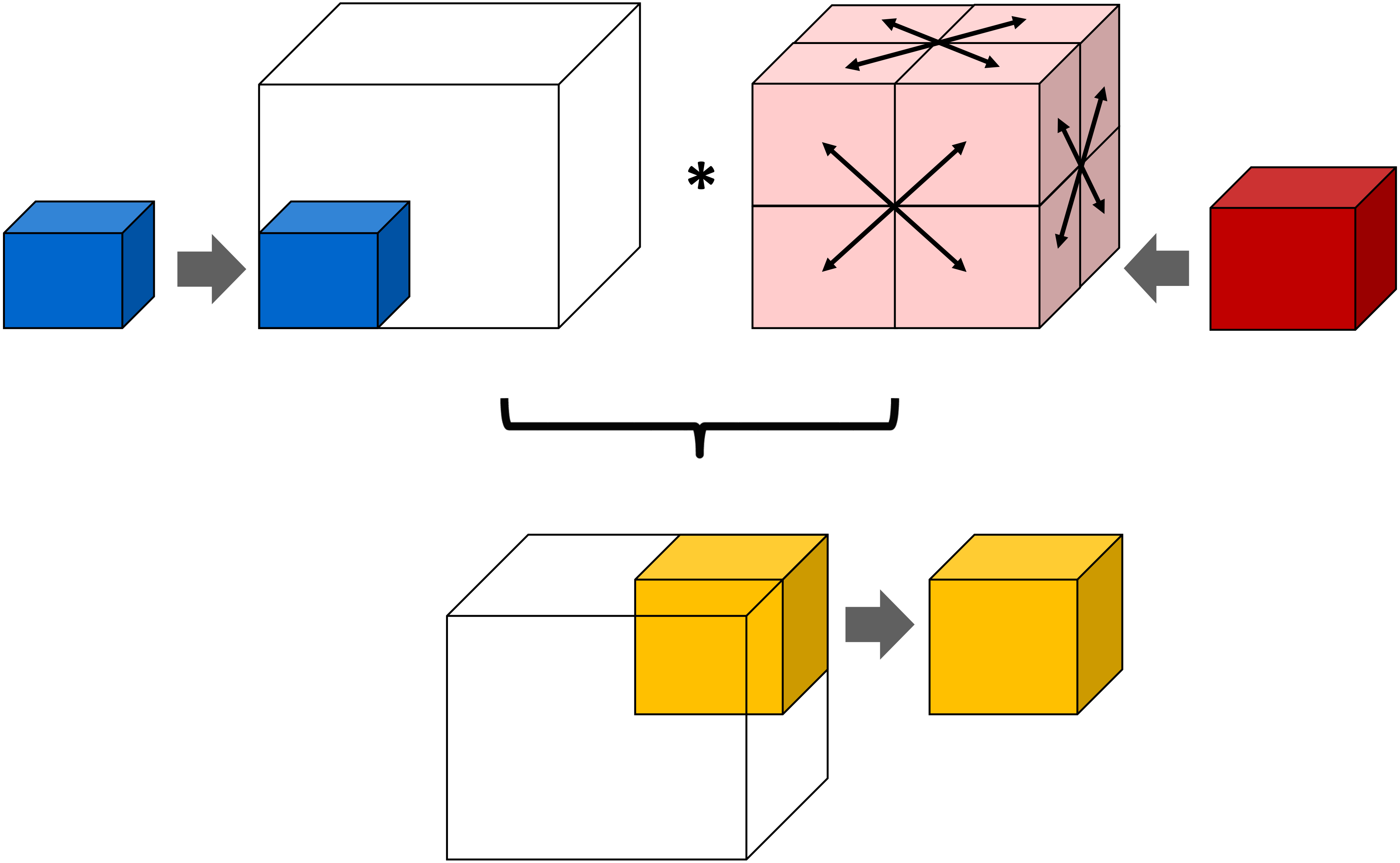
FFTX
DFT, RDFT
1D, 2D, 3D,…
batch

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

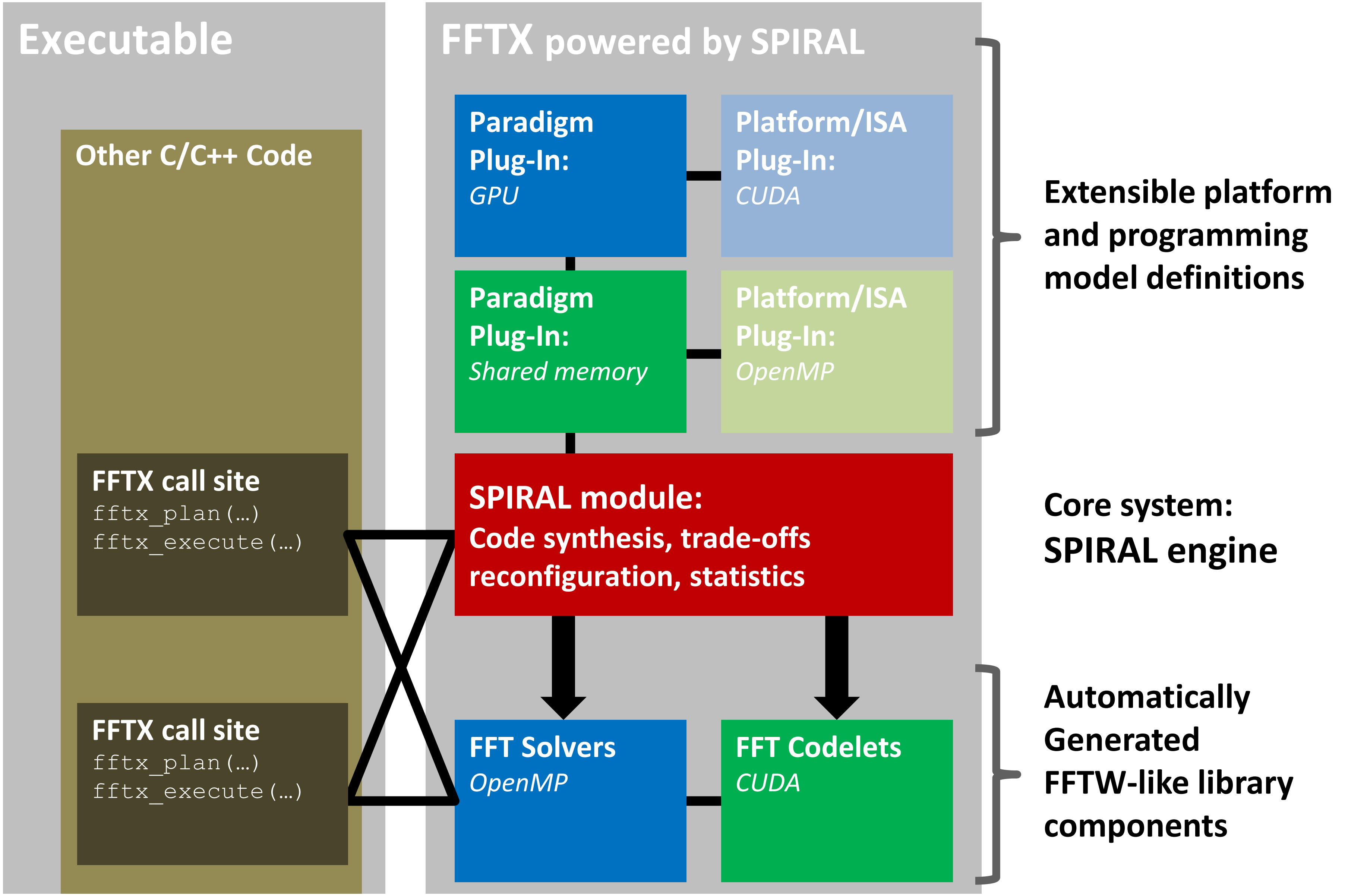


Example: Hockney Free Space Convolution

```
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



Executable

Other C/C++ Code

FFTX call site
`fftx_plan(…)`
`fftx_execute(…)`

FFTX call site
`fftx_plan(…)`
`fftx_execute(…)`

FFTX powered by SPIRAL

Paradigm Plug-In: *GPU*

Platform/ISA Plug-In: *CUDA*

Paradigm Plug-In: *Shared memory*

Platform/ISA Plug-In: *OpenMP*

SPIRAL module:
Code synthesis, trade-offs reconfiguration, statistics

FFT Solvers *OpenMP*

FFT Codelets *CUDA*

Extensible platform and programming model definitions

Core system: SPIRAL engine

Automatically Generated FFTW-like library components

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, (-0.50000000000001132), (-20.686114762237267),
      (-0.50000000000000081), (-10.337014680426078), (-0.50000000000000455),
    ...
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 = (((0.30901699437494745**((X + a9809))
            - (0.80901699437494745**((X + a9811)))) + *((X + a9807)));
        s3298 = (((0.30901699437494745**((X + a9810))
            - (0.80901699437494745**((X + a9812)))) + *((X + a9808)));
        s3299 = (((0.30901699437494745**((X + a9811))
            - (0.80901699437494745**((X + a9809)))) + *((X + a9807)));
        ...
        *((104 + Y + a12569)) = ((s3983 - s3987) + (0.80901699437494745*t6537)
            + (0.58778525229247314*t6538));
        *((105 + Y + a12569)) = (((s3984 - s3988) + (0.80901699437494745*t6538))
            - (0.58778525229247314*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**, IEEE International Conference on High Performance Computing, Data, and Analytics (HiPC), 2018

Organization

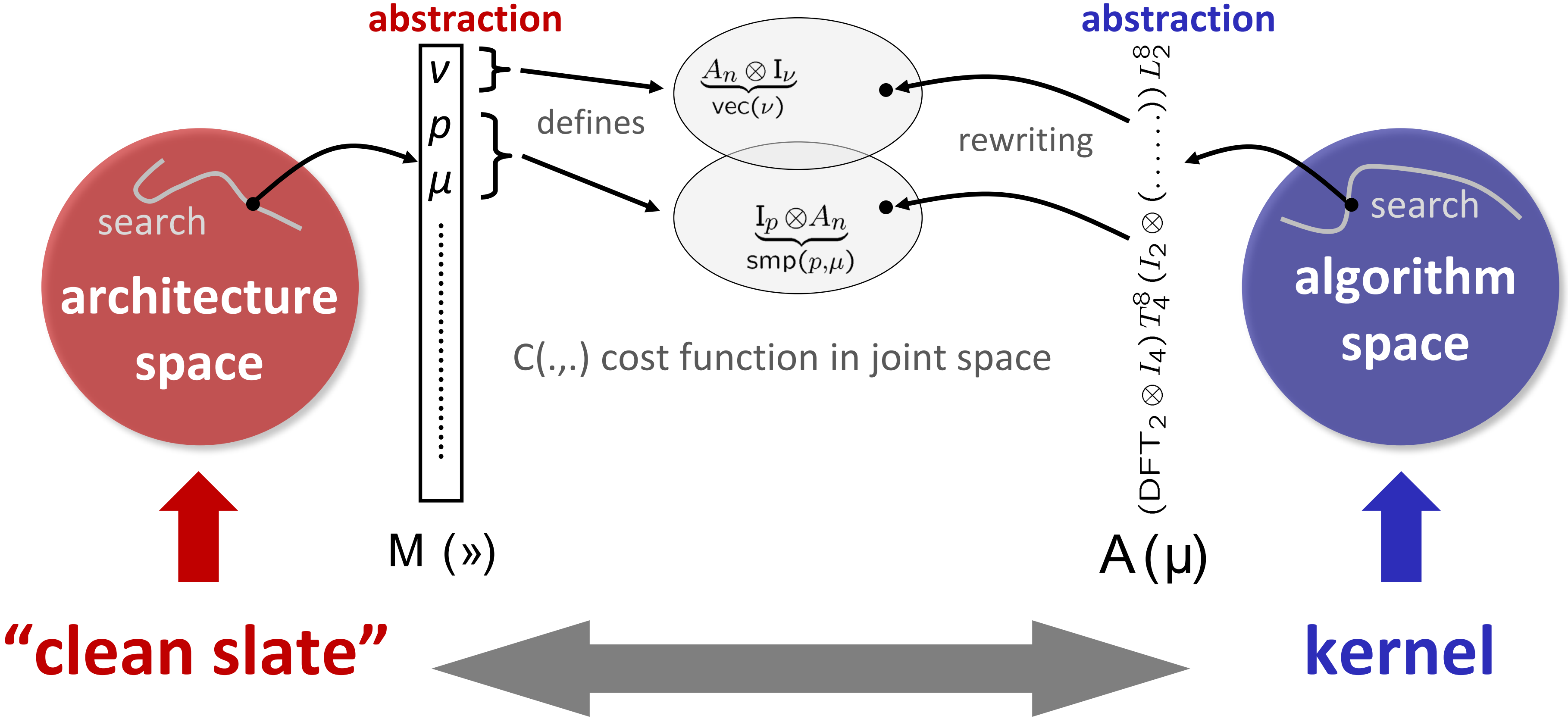
- SPL: Problem and algorithm specification
- Σ-SPL: Automating high level optimization
- Rewriting: Formal parallelization
- Rewriting: Vectorization
- Verification
- Spiral as FFTX backend
- **Summary**

Co-Optimizing Architecture and Kernel

Architectural parameter:
Vector length,
#processors, …

Model: common abstraction
= spaces of matching formulas

Kernel:
problem size,
algorithm choice



abstraction

abstraction

ν

p

μ

defines

rewriting

$\underbrace{A_n \otimes I_\nu}_{\text{vec}(\nu)}$

$\underbrace{I_p \otimes A_n}_{\text{smp}(p,\mu)}$

$(\mathrm{DFT}_2 \otimes I_4)\, T^8_4\, (I_2 \otimes (\ldots\ldots))\, L^8_2$

search

architecture space

search

algorithm space

C(.,.) cost function in joint space

M (»)

A(μ)

“clean slate”

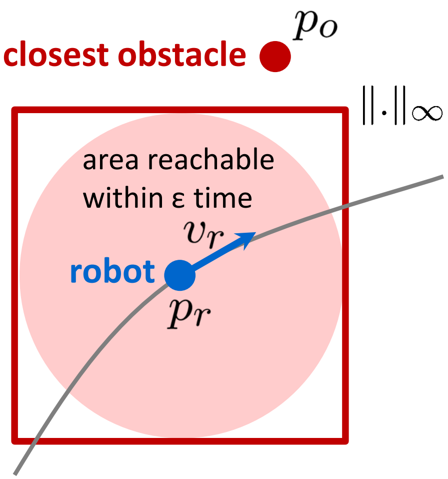
kernel

High Assurance Spiral

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



closest obstacle p_o

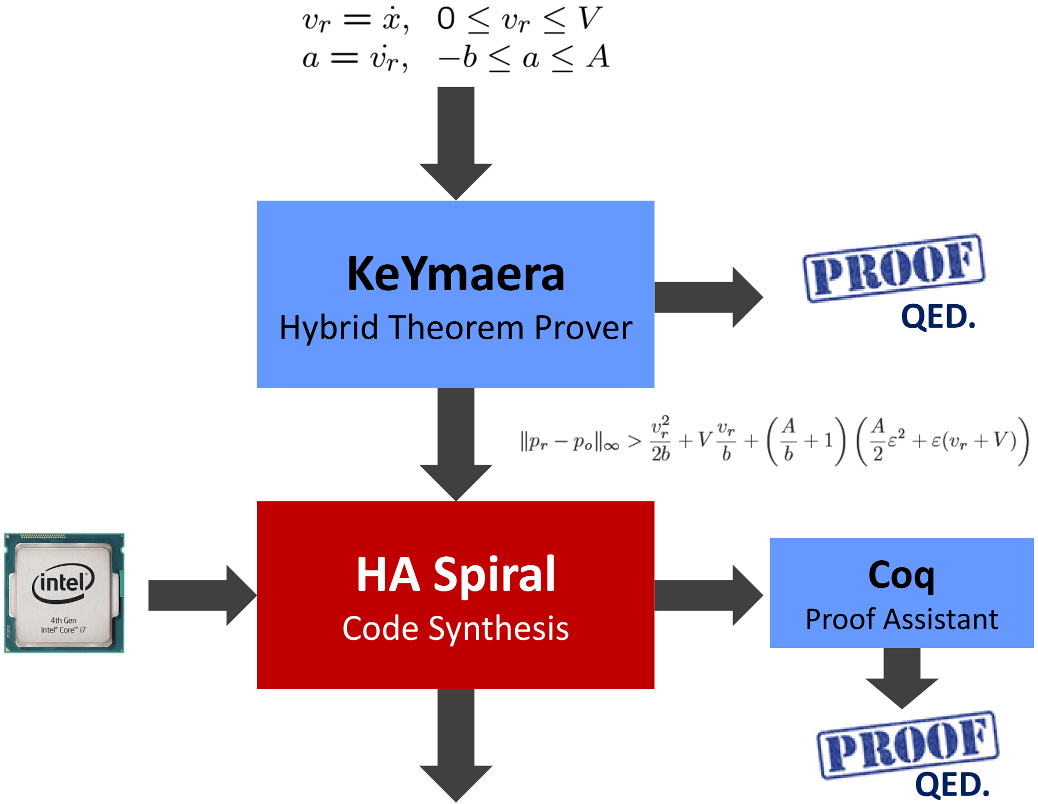
$\|\cdot\|_\infty$

area reachable within ε time

v_r

robot p_r

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



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

KeYmaera
Hybrid Theorem Prover

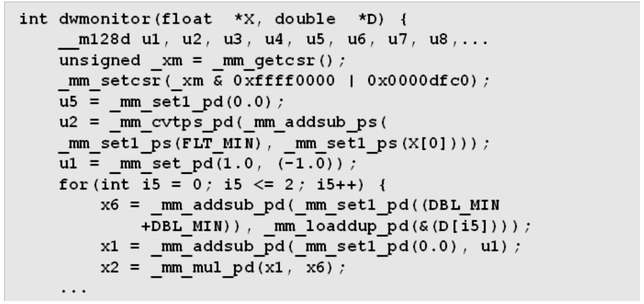
PROOF
QED.

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

HA Spiral
Code Synthesis

Coq
Proof Assistant

PROOF
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);
    ...
```

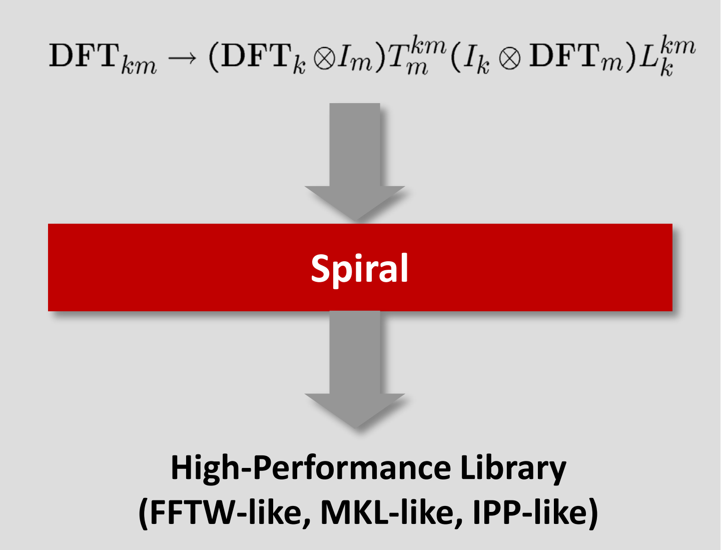
performance

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.

General Size Library Generation

Input:

- **Transform**: DFT_n
- **Algorithms**: $\mathrm{DFT}_{km} \rightarrow (\mathrm{DFT}_k \otimes I_m) T^{km}_m (I_k \otimes \mathrm{DFT}_m) L^{km}_k$
  $\mathrm{DFT}_2 \rightarrow \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$
- **Vectorization**: 2-way SSE
- **Threading**: Yes



$\mathrm{DFT}_{km} \rightarrow (\mathrm{DFT}_k \otimes I_m) T^{km}_m (I_k \otimes \mathrm{DFT}_m) L^{km}_k$

Spiral

High-Performance Library
(FFTW-like, MKL-like, IPP-like)

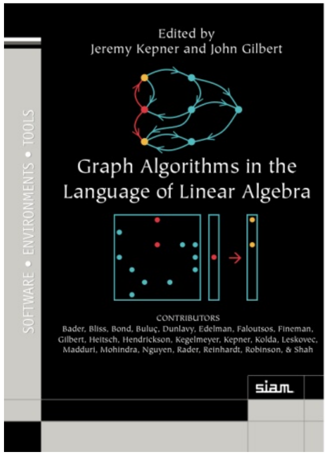
Output:

- Optimized library (10,000 lines of C++)
- For general input size
  (**not** collection of fixed sizes)
- Vectorized
- Multithreaded
- With runtime adaptation mechanism
- Performance competitive with hand-written code

Yevgen Voronenko, Frédéric de Mesmay and Markus Püschel
Computer Generation of General Size Linear Transform Libraries
Proc. International Symposium on Code Generation and Optimization (CGO), pp. 102-113, 2009

Graph Algorithms in SPIRAL

Foundation



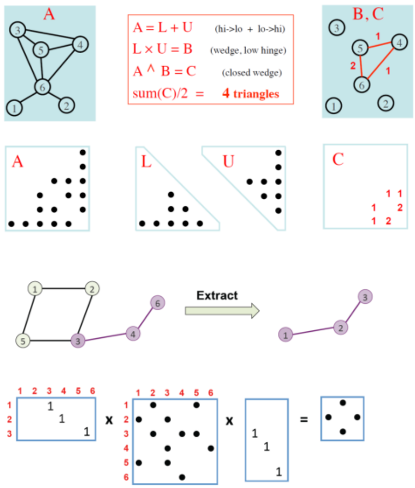
Edited by
Jeremy Kepner and John Gilbert

Graph Algorithms in the Language of Linear Algebra

CONTRIBUTORS
Bader, Bliss, Bond, Buluç, Dunlavy, Edelman, Faloutsos, Fineman, Gilbert, Heitsch, Hendrickson, Kegelmeyer, Kepner, Kolda, Leskovec, Madduri, Mohindra, Nguyen, Rader, Reinhardt, Robinson, & Shah

SOFTWARE • ENVIRONMENTS • TOOLS

siam



Context

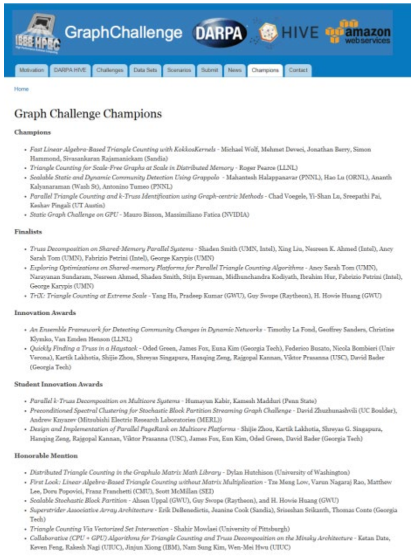
- Kepner & Gilbert et al recast graph algorithms as linear algebra operations
- GraphBLAS Forum and reference implementations
- IBM, Intel, national labs etc. on board

Formalization

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

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

HIVE Graph Challenge



IEEE HPEC GraphChallenge DARPA HIVE amazon web services

Motivation | DARPA HIVE | Challenges | Data Sets | Scenarios | Submit | News | Champions | Contact

Home

Graph Challenge Champions

Champions

- *Fast Linear Algebra-Based Triangle Counting with KokkosKernels* - Michael Wolf, Mehmet Deveci, Jonathan Berry, Simon Hammond, Sivasankaran Rajamanickam (Sandia)
- *Triangle Counting for Scale-Free Graphs at Scale in Distributed Memory* - Roger Pearce (LLNL)
- *Scalable Static and Dynamic Community Detection Using Grappolo* - Mahantesh Halappanavar (PNNL), Hao Lu (ORNL), Ananth Kalyanaraman (Wash St), Antonino Tumeo (PNNL)
- *Parallel Triangle Counting and k-Truss Identification using Graph-centric Methods* - Chad Voegele, Yi-Shan Lu, Sreepathi Pai, Keshav Pingali (UT Austin)
- *Static Graph Challenge on GPU* - Mauro Bisson, Massimiliano Fatica (NVIDIA)

Finalists

- *Truss Decomposition on Shared-Memory Parallel Systems* - Shaden Smith (UMN, Intel), Xing Liu, Nesreen K. Ahmed (Intel), Ancy Sarah Tom (UMN), Fabrizio Petrini (Intel), George Karypis (UMN)
- *Exploring Optimizations on Shared-memory Platforms for Parallel Triangle Counting Algorithms* - Ancy Sarah Tom (UMN), Narayanan Sundaram, Nesreen Ahmed, Shaden Smith, Stijn Eyerman, Midhunchandra Kodiyath, Ibrahim Hur, Fabrizio Petrini (Intel), George Karypis (UMN)
- *TriX: Triangle Counting at Extreme Scale* - Yang Hu, Pradeep Kumar (GWU), Guy Swope (Raytheon), H. Howie Huang (GWU)

Innovation Awards

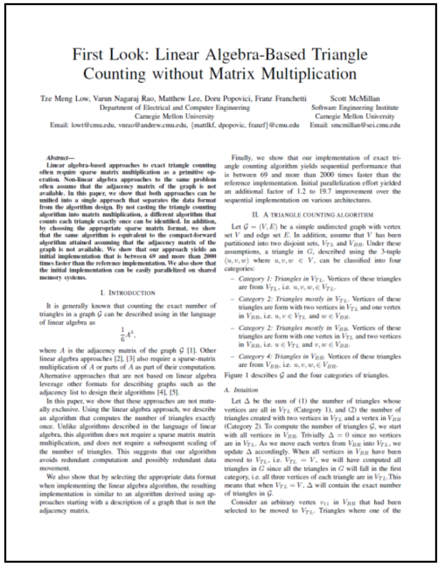
- *An Ensemble Framework for Detecting Community Changes in Dynamic Networks* - Timothy La Fond, Geoffrey Sanders, Christine Klymko, Van Emden Henson (LLNL)
- *Quickly Finding a Truss in a Haystack* - Oded Green, James Fox, Euna Kim (Georgia Tech), Federico Busato, Nicola Bombieri (Univ Verona), Kartik Lakhotia, Shijie Zhou, Shreyas Singapura, Hanqing Zeng, Rajgopal Kannan, Viktor Prasanna (USC), David Bader (Georgia Tech)

Student Innovation Awards

- *Parallel k-Truss Decomposition on Multicore Systems* - Humayun Kabir, Kamesh Madduri (Penn State)
- *Preconditioned Spectral Clustering for Stochastic Block Partition Streaming Graph Challenge* - David Zhuzhunashvili (UC Boulder), Andrew Knyazev (Mitsubishi Electric Research Laboratories (MERL))
- *Design and Implementation of Parallel PageRank on Multicore Platforms* - Shijie Zhou, Kartik Lakhotia, Shreyas G. Singapura, Hanqing Zeng, Rajgopal Kannan, Viktor Prasanna (USC), James Fox, Eun Kim, Oded Green, David Bader (Georgia Tech)

Honorable Mention

- *Distributed Triangle Counting in the Graphulo Matrix Math Library* - Dylan Hutchison (University of Washington)
- *First Look: Linear Algebra-Based Triangle Counting without Matrix Multiplication* - Tze Meng Low, Varun Nagaraj Rao, Matthew Lee, Doru Popovici, Franz Franchetti (CMU), Scott McMillan (SEI)
- *Scalable Stochastic Block Partition* - Ahsen Uppal (GWU), Guy Swope (Raytheon), and H. Howie Huang (GWU)
- *Superstrider Associative Array Architecture* - Erik DeBenedictis, Jeanine Cook (Sandia), Sriseshan Srikanth, Thomas Conte (Georgia Tech)
- *Triangle Counting Via Vectorized Set Intersection* - Shahir Mowlaei (University of Pittsburgh)
- *Collaborative (CPU + GPU) Algorithms for Triangle Counting and Truss Decomposition on the Minsky Architecture* - Ketan Date, Keven Feng, Rakesh Nagi (UIUC), Jinjun Xiong (IBM), Nam Sung Kim, Wen-Mei Hwu (UIUC)



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

Tze Meng Low, Varun Nagaraj Rao, Matthew Lee, Doru Popovici, Franz Franchetti
Department of Electrical and Computer Engineering
Carnegie Mellon University
Email: lowt@cmu.edu, vnrao@andrew.cmu.edu, {mattlkf, dpopovic, franzf}@cmu.edu

Scott McMillan
Software Engineering Institute
Carnegie Mellon University
Email: smcmillan@sei.cmu.edu

Abstract—**Linear algebra-based approaches to exact triangle counting often require sparse matrix multiplication as a primitive operation. Non-linear algebra approaches to the same problem often assume that the adjacency matrix of the graph is not available. In this paper, we show that both approaches can be unified into a single approach that separates the data format from the algorithm design. By not casting the triangle counting algorithm into matrix multiplication, a different algorithm that counts each triangle exactly once can be identified. In addition, by choosing the appropriate sparse matrix format, we show that the same algorithm is equivalent to the compact-forward algorithm attained assuming that the adjacency matrix of the graph is not available. We show that our approach yields an initial implementation that is between 69 and more than 2000 times faster than the reference implementation. We also show that the initial implementation can be easily parallelized on shared memory systems.**

I. Introduction

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

$$\frac{1}{6}A^3,$$

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

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

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

Finally, we show that our implementation of exact triangle counting algorithm yields sequential performance that is between 69 and more than 2000 times faster than the reference implementation. Initial parallelization effort yielded an additional factor of 1.2 to 19.7 improvement over the sequential implementation on various architectures.

II. A Triangle Counting Algorithm

Let $\mathcal{G} = (V, E)$ be a simple undirected graph with vertex set V and edge set E. In addition, assume that V has been partitioned into two disjoint sets, V_{TL} and V_{BR}. Under these assumptions, a triangle in G, described using the 3-tuple (u, v, w) where $u, v, w \in V$, can be classified into four categories:

- *Category 1: Triangles in V_{TL}.* Vertices of these triangles are from V_{TL}, i.e. $u, v, w, \in V_{TL}$.
- *Category 2: Triangles mostly in V_{TL}.* Vertices of these triangles are form with two vertices in V_{TL} and one vertex in V_{BR}, i.e. $u, v \in V_{TL}$ and $w \in V_{BR}$.
- *Category 2: Triangles mostly in V_{BR}.* Vertices of these triangles are form with one vertex in V_{TL} and two vertices in V_{BR}, i.e. $u \in V_{TL}$ and $v, w \in V_{BR}$.
- *Category 4: Triangles in V_{BR}.* Vertices of these triangles are from V_{BR}, i.e. $u, v, w, \in V_{BR}$.

Figure 1 describes $\mathcal{G}$ and the four categories of triangles.

A. Intuition

Let Δ be the sum of (1) the number of triangles whose vertices are all in V_{TL} (Category 1), and (2) the number of triangles created with two vertices in V_{TL} and a vertex in V_{BR} (Category 2). To compute the number of triangles $\mathcal{G}$, we start with all vertices in V_{BR}. Trivially $\Delta = 0$ since no vertices are in V_{TL}. As we move each vertex from V_{BR} into V_{TL}, we update Δ accordingly. When all vertices in V_{BR} have been moved to V_{TL}, i.e. $V_{TL} = V$, we will have computed all triangles in G since all the triangles in G will fall in the first category, i.e. all three vertices of each triangle are in V_{TL}.This means that when $V_{TL} = V$, Δ will contain the exact number of triangles in $\mathcal{G}$.

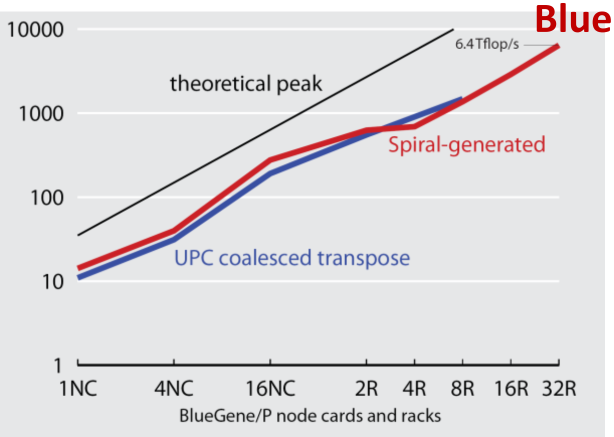
Consider an arbitrary vertex v_{11} in V_{BR} that had been selected to be moved to V_{TL}. Triangles where one of the

In collaboration with CMU-SEI

SPIRAL FFTs in HPC/Supercomputing

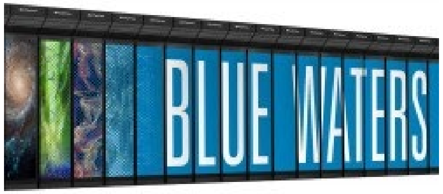
- **NCSA Blue Waters**
  PAID Program, FFTs for Blue Waters
- **RIKEN K computer**
  FFTs for the HPC-ACE ISA
- **LANL RoadRunner**
  FFTs for the Cell processor
- **PSC/XSEDE Bridges**
  Large size FFTs
- **LLNL BlueGene/L and P**
  FFTW for BlueGene/L's Double FPU
- **ANL BlueGene/Q Mira**
  Early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

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



6.4 Tflop/s on BlueGene/P

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

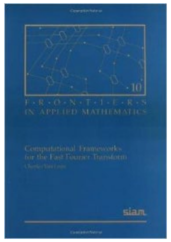


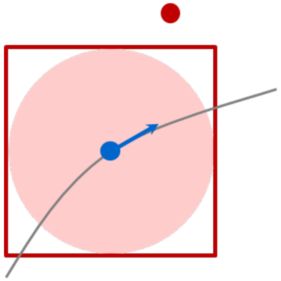
2006 Gordon Bell Prize (Peak Performance Award) with LLNL and IBM

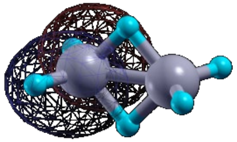
2010 HPC Challenge Class II Award (Most Productive System) with ANL and IBM

Summary: Formal Code Synthesis

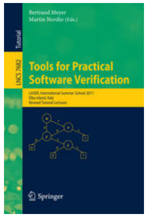
Algorithms

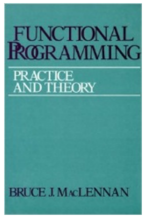


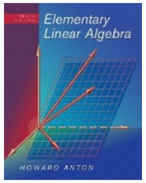


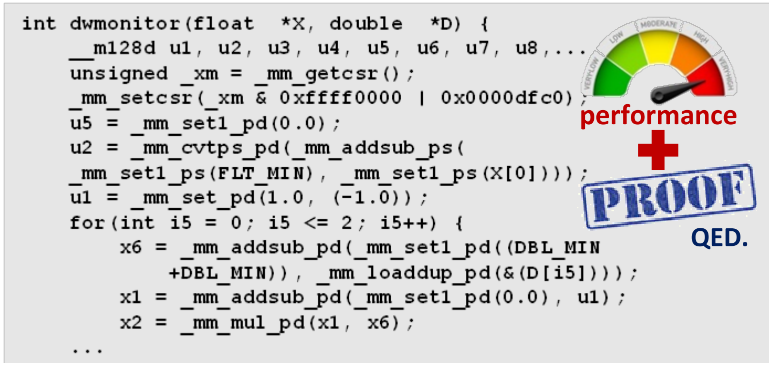


Correctness









```
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

More Information:

www.spiral.net

www.spiralgen.com

References

Overview Papers

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*

M. Püschel, F. Franchetti, Y. Voronenko: **Spiral.** Encyclopedia of Parallel Computing, D. A. Padua (Editor).

M. Püschel, J.M.F. Moura, J. Johnson, D. Padua, M. Veloso, B. Singer, J. Xiong, F. Franchetti, A. Gacic, Y. Voronenko, K. Chen, R.W. Johnson, and N. Rizzolo: **SPIRAL: Code Generation for DSP Transforms.** Special issue, Proceedings of the IEEE 93(2), 2005.

F. Franchetti, Y. Voronenko, S. Chellappa, J. M. F. Moura, and M. Püschel: **Discrete Fourier Transform on Multicores: Algorithms and Automatic Implementation.** IEEE Signal Processing Magazine, special issue on “Signal Processing on Platforms with Multiple Cores”, 2009.

Core Technology Papers

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.

Y. Voronenko, F. de Mesmay and M. Püschel: **Computer Generation of General Size Linear Transform Libraries.**

Proc. International Symposium on Code Generation and Optimization (CGO), pp. 102-113, 2009.

F. Franchetti, Y. Voronenko, M. Püschel: **Loop Merging for Signal Transforms.** Proceedings Programming Language Design and Implementation (PLDI) 2005, pages 315-326.

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.

References

SIMD Vectorization

F. Franchetti, M. Püschel: **Short Vector Code Generation for the Discrete Fourier Transform.** Proceedings of the 17th International Parallel and Distributed Processing Symposium (IPDPS '03), pages 58-67.

F. Franchetti, Y. Voronenko, M. Püschel: **A Rewriting System for the Vectorization of Signal Transforms.** Proceedings High Performance Computing for Computational Science (VECPAR) 2006, LNCS 4395, pages 363-377.

F. Franchetti and M. Püschel: **SIMD Vectorization of Non-Two-Power Sized FFTs.** Proceedings of International Conference on Acoustics, Speech, and Signal Processing (ICASSP) 07.

F. Franchetti and M. Püschel: **Generating SIMD Vectorized Permutations.** Proceedings of International Conference on Compiler Construction (CC) 2008.

D. S. McFarlin, V. Arbatov, F. Franchetti, M. Püschel: **Automatic SIMD Vectorization of Fast Fourier Transforms for the Larrabee and AVX Instruction Sets.** Proceedings of International Conference on Supercomputing (ICS), 2011.

Multicore and Distributed Memory

F. Franchetti, Y. Voronenko, and M. Püschel: **FFT Program Generation for Shared Memory: SMP and Multicore.** Proceedings Supercomputing 2006.

A. Bonelli, F. Franchetti, J. Lorenz, M. Püschel, and C. W. Ueberhuber: **Automatic Performance Optimization of the Discrete Fourier Transform on Distributed Memory Computers.** Proceedings of ISPA 06. Lecture Notes in Computer Science, Volume 4330, 2006, Pages 818 – 832.

S. Chellappa, F. Franchetti and M. Püschel: **Computer Generation of Fast FFTs for the Cell Broadband Engine.** Proceedings of International Conference on Supercomputing (ICS), 2009.

F. Franchetti, Y. Voronenko, and G. Almasi: **Automatic Generation of the HPC Challenges Global FFT Benchmark for BlueGene/P.** In Proceedings of High Performance Computing for Computational Science (VECPAR) 2012.

References

FPGA and Energy

P. A. Milder, F. Franchetti, J. C. Hoe, and M. Püschel: **Computer Generation of Hardware for Linear Digital Signal Processing Transforms.** ACM Transactions on Design Automation of Electronic Systems,17(2), Article 15, 2012.

P. A. Milder, F. Franchetti, J. C. Hoe, and M. Püschel: **Formal Datapath Representation and Manipulation for Implementing DSP Transforms.** Proceedings of Design Automation Conference (DAC), 2008.

P. A. Milder, F. Franchetti, J. C. Hoe, and M. Püschel: **Hardware Implementation of the Discrete Fourier Transform With Non-Power-Of-Two Problem Size.** Proceedings of International Conference on Acoustics, Speech, and Signal Processing (ICASSP), 2010.

P. D'Alberto, F. Franchetti, P. A. Milder, A. Sandryhaila, J. C. Hoe, J. M. F. Moura, and M. Püschel: **Generating FPGA Accelerated DFT Libraries.** Proceedings of Field-Programmable Custom Computing Machines (FCCM) 2007.

B. Akin, P.A. Milder, F. Franchetti, and J. Hoe: **Memory Bandwidth Efficient Two-Dimensional Fast Fourier Transform Algorithm and Implementation for Large Problem Sizes.** IEEE International Symposium on Field-Programmable Custom Computing Machines (FCCM), 188-191, 2012.

B. Akin, F. Franchetti, J. Hoe: **FFTs with Near-Optimal Memory Access Through Block Data Layouts.** ICASSP 2014.

P. D'Alberto, M. Püschel, and F. Franchetti: **Performance/Energy Optimization of DSP Transforms on the XScale Processor.** Proceedings of International Conference on High Performance Embedded Architectures & Compilers (HiPEAC) 2007.

References

Applications (SAR and Software-Defined Radio)

D. McFarlin, F. Franchetti, M. Püschel, and J. M. F. Moura: **High Performance Synthetic Aperture Radar Image Formation On Commodity Multicore Architectures.** in Proceedings SPIE, 2009.

F. de Mesmay, S. Chellappa, F. Franchetti and M. Püschel: **Computer Generation of Efficient Software Viterbi Decoders.** Proceedings of International Conference on High-Performance Embedded Architectures and Compilers (HIPEAC), 2010.

Y. Voronenko, V. Arbatov, C. Berger, R. Peng, M. Püschel, and F. Franchetti: **Computer Generation of Platform-Adapted Physical Layer Software.** Proceedings of Software Defined Radio (SDR), 2010.

C. R. Berger, V. Arbatov, Y. Voronenko, F. Franchetti, M. Püschel: **Real-Time Software Implementation of an IEEE 802.11a Baseband Receiver on Intel Multicore.** Proceedings of International Conference on Acoustics, Speech, and Signal Processing (ICASSP), 2011.

FFT and FIR Algorithms

F. Franchetti and M. Püschel: **Generating High-Performance Pruned FFT Implementations.** Proceedings of International Conference on Acoustics, Speech, and Signal Processing (ICASSP) 09.

LC Meng, J. Johnson, F. Franchetti, Y. Voronenko, M.M. Maza and Y. Xie: **Spiral-Generated Modular FFT Algorithms.** Proc. Parallel Symbolic Computation (PASCO), pp. 169-170, 2010.

Yevgen Voronenko and Markus Püschel: **Algebraic Derivation of General Radix Cooley-Tukey Algorithms for the Real Discrete Fourier Transform.** Proc. International Conference on Acoustics, Speech, and Signal Processing (ICASSP), Vol. 3, 2006

Aca Gacic, Markus Püschel and José M. F. Moura: **Fast Automatic Implementations of FIR Filters.** Proc. International Conference on Acoustics, Speech, and Signal Processing (ICASSP), Vol. 2, pp. 541-544, 2003

References

FFTX and SpectralPACK

F. Franchetti, D. G. Spampinato, A. Kulkarni, D. T. Popovici, T. M. Low, M. Franusich, A. Canning, P. McCorquodale, B. Van Straalen, P. Colella: **FFTX and SpectralPack: A First Look**, IEEE International Conference on High Performance Computing, Data, and Analytics (HiPC), 2018