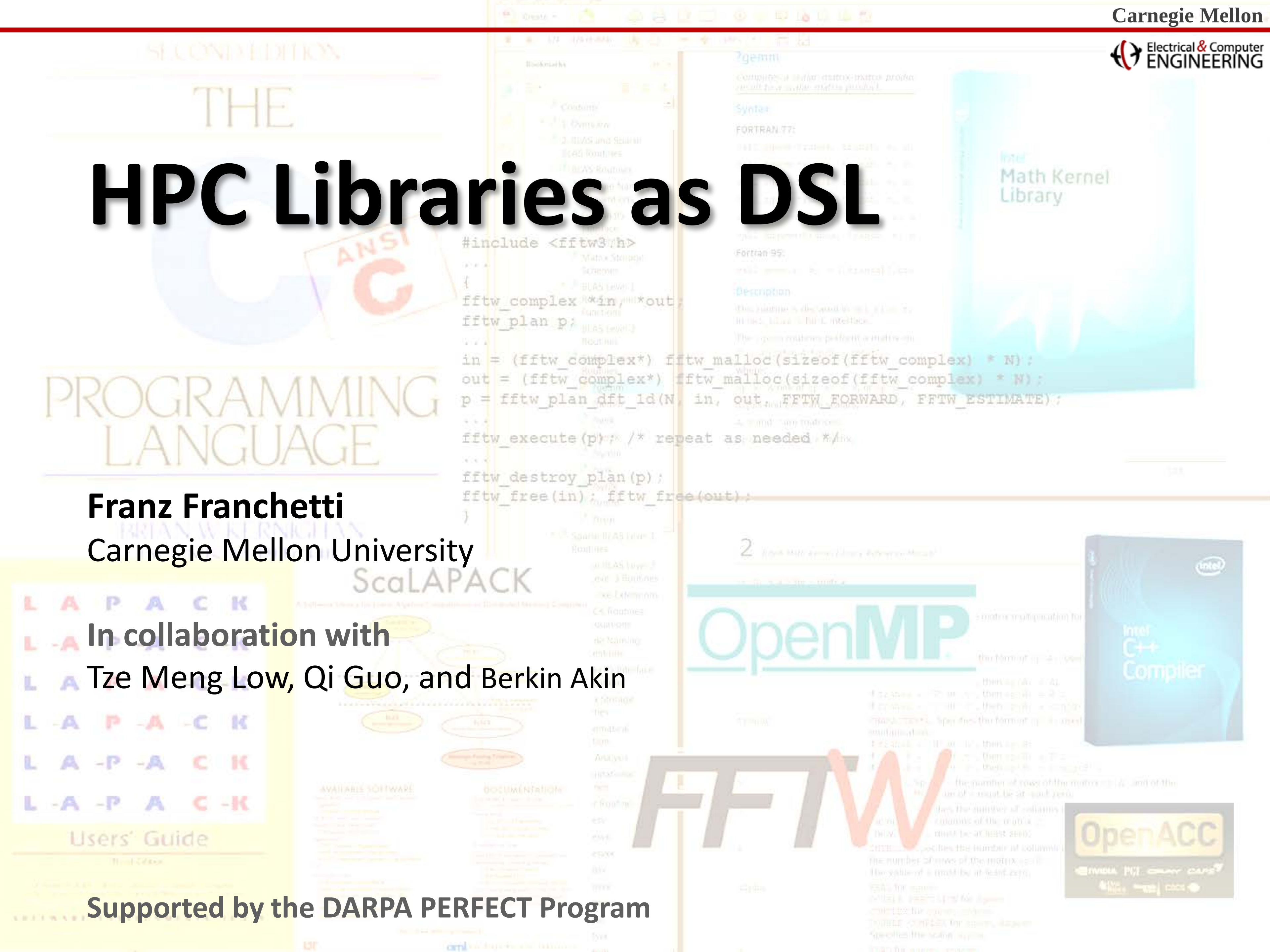
# HPC Libraries as DSL

**Franz Franchetti**
Carnegie Mellon University

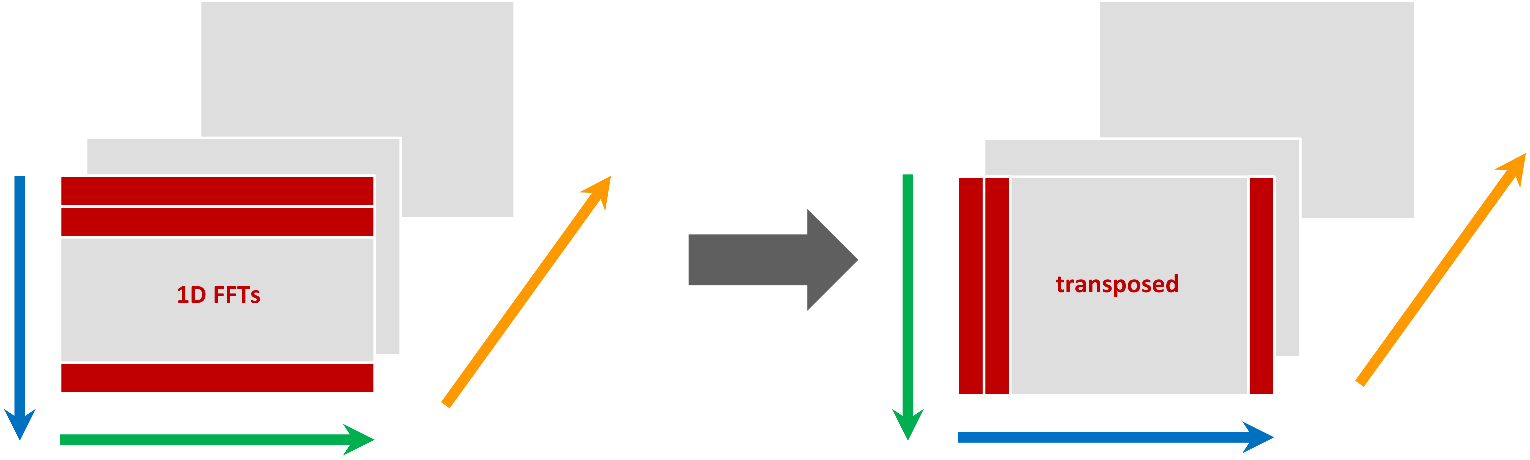
**In collaboration with**
Tze Meng Low, Qi Guo, and Berkin Akin

**Supported by the DARPA PERFECT Program**



```
#include <fftw3.h>
...
{
fftw_complex *in, *out;
fftw_plan p;
...
in = (fftw_complex*) fftw_malloc(sizeof(fftw_complex) * N);
out = (fftw_complex*) fftw_malloc(sizeof(fftw_complex) * N);
p = fftw_plan_dft_1d(N, in, out, FFTW_FORWARD, FFTW_ESTIMATE);
...
fftw_execute(p); /* repeat as needed */
...
fftw_destroy_plan(p);
fftw_free(in); fftw_free(out);
}
```

# How (Not To) Program a Computational Kernel



```
for (chan = 0; chan < N_CHAN; ++chan)
{
  for (range = 0; range < N_RANGE; ++range)
  {
    /* Apply a 1D FFT to convert to Doppler space */
    fft(
     (float *)&datacube_pulse_major_padded[chan][range][0],
     N_DOP,
     N_DOP_LOG2,
     FFT_FORWARD);
  }
}
/* Transpose from doppler major to range major*/
corner_turn_to_range_major_order
(doppler_datacube, datacube_pulse_major_padded);
```

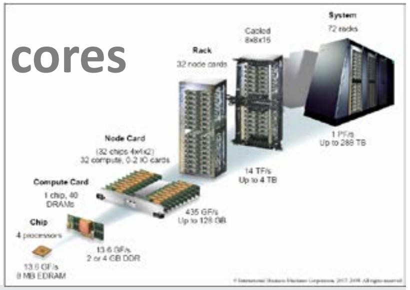
7.5x

```
/* Replace with fftw guru interface*/
plan = fftwf_plan_guru_dft(
            1, dims,
            2, howmany_dims,
            datacube_pulse_major_padded,
            doppler_datacube,
            FFTW_FORWARD,
            FFTW_WISDOM_ONLY);

fftwf_execute(plan);
fftwf_destroy_plan(plan);
```

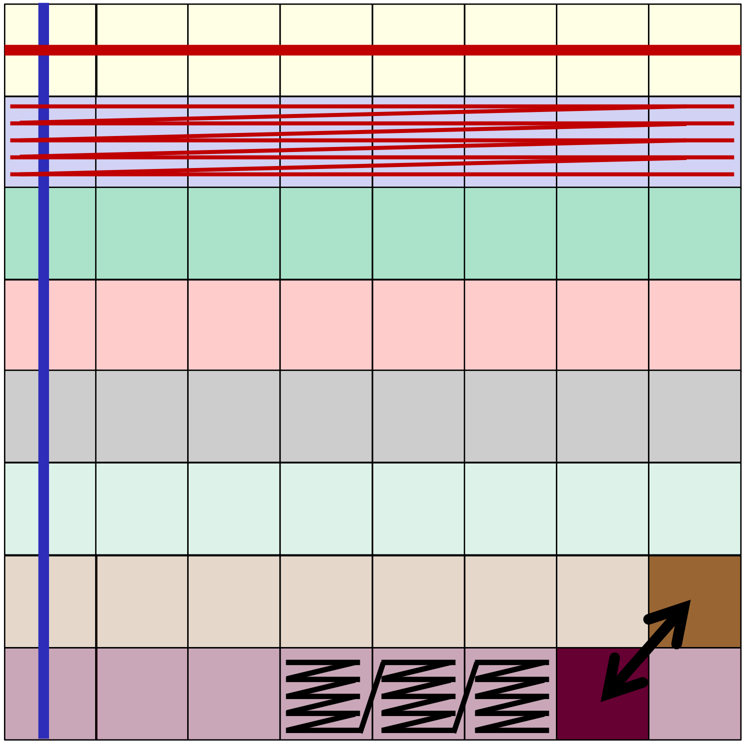
T. M. Low, Q. Guo, F. Franchetti: **Optimizing Space Time Adaptive Processing Through Accelerating Memory-Bounded Operations.** IEEE High Performance Extreme Computing Conference (HPEC), 2015.

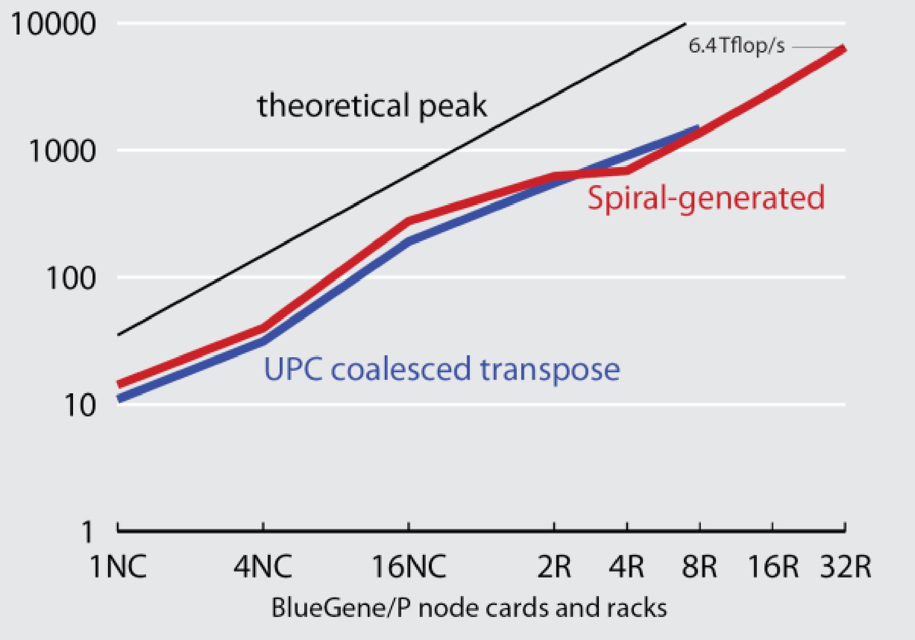
# But: Libraries Impose Incompatible Constraints

128k cores



FFTW: contiguous rows/columns





MPI: contiguous sub-blocks

30% gain from MPI-aware FFTW

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

# Using Libraries—Isn’t that best HPC practice?

## In theory, yes. But

- **Sometimes looks like overkill**

  Add a whole library for a one-page function?!

- **People often do not use them**

  dependencies, don’t know them,
  “not invented here”

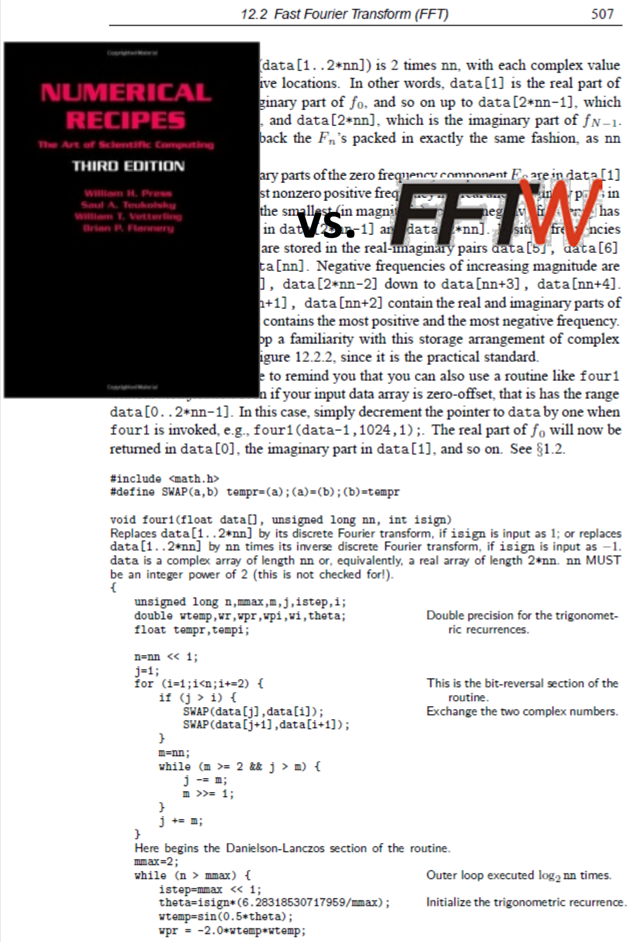
- **Need to combine many libraries**

  MPI, FFTW, LAPACK, Boost, STL,…

- **Uneven performance and coverage across functions**

  not whole FFTW interface well-supported by MKL, ESSL,…

## 1 page vs. 30 MB C code


NUMERICAL RECIPES
The Art of Scientific Computing
THIRD EDITION
William H. Press
Saul A. Teukolsky
William T. Vetterling
Brian P. Flannery
vs.
FFTW

12.2 Fast Fourier Transform (FFT) 507

(data[1..2*nn]) is 2 times nn, with each complex value
ive locations. In other words, data[1] is the real part of
ginary part of $f_0$, and so on up to data[2*nn-1], which
, and data[2*nn], which is the imaginary part of $f_{N-1}$.
back the $F_n$’s packed in exactly the same fashion, as nn

ary parts of the zero frequency component $F_0$ are in data[1]
st nonzero positive frequency has real and imaginary parts in
the smallest (in magnitude) nonzero negative frequency has
in data[2*nn-1] and data[2*nn]. Positive frequencies
are stored in the real-imaginary pairs data[5], data[6]
ta[nn]. Negative frequencies of increasing magnitude are
], data[2*nn-2] down to data[nn+3], data[nn+4].
n+1], data[nn+2] contain the real and imaginary parts of
contains the most positive and the most negative frequency.
op a familiarity with this storage arrangement of complex
igure 12.2.2, since it is the practical standard.
e to remind you that you can also use a routine like four1
n if your input data array is zero-offset, that is has the range data[0..2*nn-1]. In this case, simply decrement the pointer to data by one when four1 is invoked, e.g., four1(data-1,1024,1);. The real part of $f_0$ will now be returned in data[0], the imaginary part in data[1], and so on. See §1.2.

```
#include <math.h>
#define SWAP(a,b) tempr=(a);(a)=(b);(b)=tempr

void four1(float data[], unsigned long nn, int isign)
Replaces data[1..2*nn] by its discrete Fourier transform, if isign is input as 1; or replaces
data[1..2*nn] by nn times its inverse discrete Fourier transform, if isign is input as −1.
data is a complex array of length nn or, equivalently, a real array of length 2*nn. nn MUST
be an integer power of 2 (this is not checked for!).
{
    unsigned long n,mmax,m,j,istep,i;
    double wtemp,wr,wpr,wpi,wi,theta;          Double precision for the trigonomet-
    float tempr,tempi;                             ric recurrences.

    n=nn << 1;
    j=1;
    for (i=1;i<n;i+=2) {                       This is the bit-reversal section of the
        if (j > i) {                               routine.
            SWAP(data[j],data[i]);             Exchange the two complex numbers.
            SWAP(data[j+1],data[i+1]);
        }
        m=nn;
        while (m >= 2 && j > m) {
            j -= m;
            m >>= 1;
        }
        j += m;
    }
    Here begins the Danielson-Lanczos section of the routine.
    mmax=2;
    while (n > mmax) {                         Outer loop executed log2 nn times.
        istep=mmax << 1;
        theta=isign*(6.28318530717959/mmax);   Initialize the trigonometric recurrence.
        wtemp=sin(0.5*theta);
        wpr = -2.0*wtemp*wtemp;
```

# Idea 1: Standard HPC Libraries as Kernel DSL

## DSL definition and syntax

- **API of standard HPC libraries**
  FFTW, LAPACK, BLAS, SparseBLAS, GraphBLAS

- **Libraries and language extensions for parallelism**
  OpenMP, MPI, OpenACC

- **Subset of C/C++**
  Single threaded, side-effect free, only standard C library calls

## DSL semantics and user knowledge

- **C semantics + library semantics**
  Program can be executed
- **Use OpenMP/OpenACC annotations to communicate meta-information**
  `#pragma omp for private(…) shared(…)`

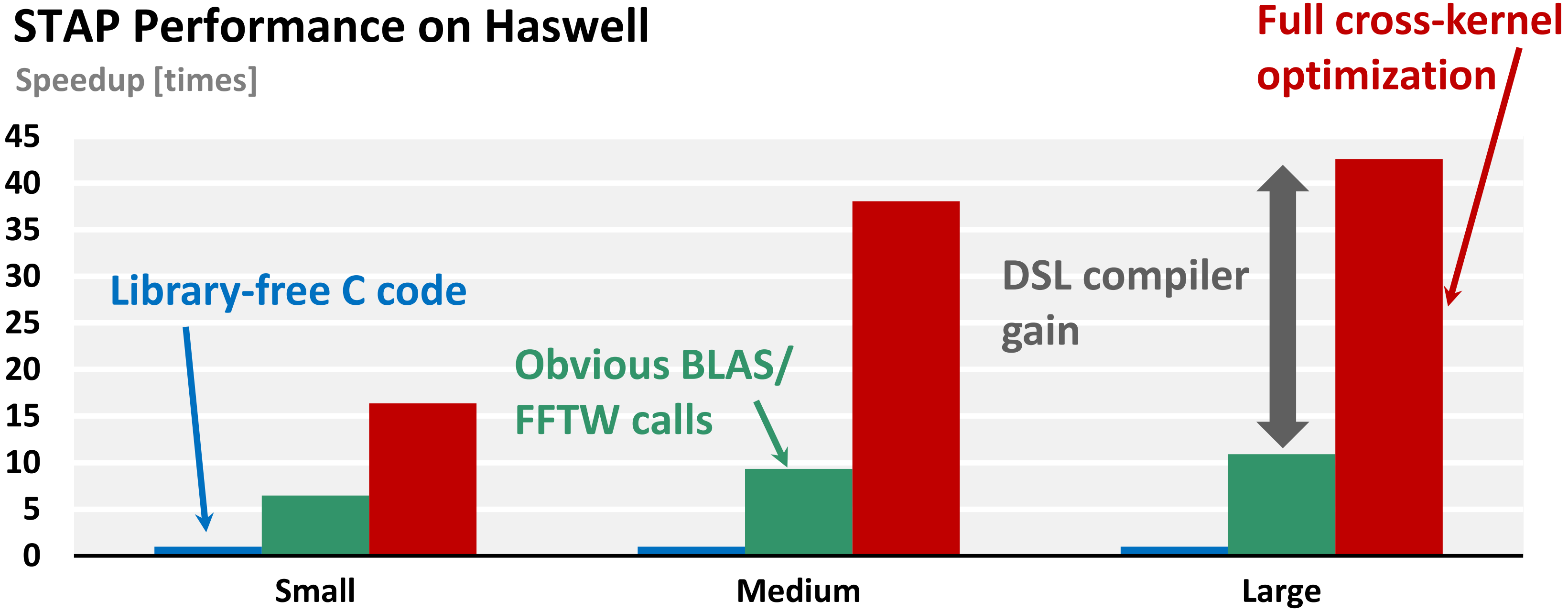
# Idea 2: Interpret Program as Specification

- **Make a *subset* of “C+OpenMP+MKL” a DSL with DSL compiler**
  Combines code synthesis, telescoping languages, and HPC best practices
- **Treat DSL program as *specification*, not as program**
  computational kernels only, thus moderate code complexity
- **Library calls are a DSL instructions, C fragments are “call-backs”**
  overcomes the usual problem of code between library calls
- **Establish side effect-freeness and parallelization opportunities**
  use OpenMP/OpenACC to express independent loops, variable visibility,…
- **Enables whole program optimization in domain specific compiler**
  data layout transformations, kernel merging, target novel accelerators,…

# Result: Performance Portability

- **Same source code, good performance across architectures**
  Intel Haswell, Intel Xeon PHI and Near Memory Accelerator
- **Parallel cross-kernel optimized library-based code is fast**
  40x speed-up over C baseline (PNNL TAV STAP benchmark)
- **Library-based code is pre-requisite for *Near Memory Accelerator***
  130x performance and 8,000x power efficiency gain on accelerator over C base line

**STAP Performance on Haswell**



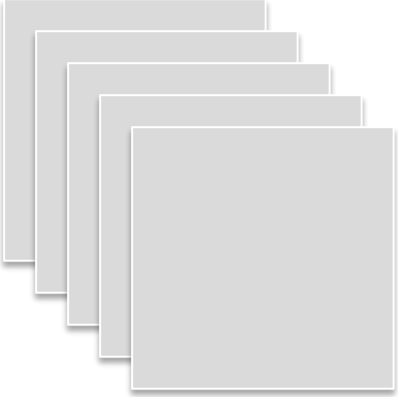
# Outline

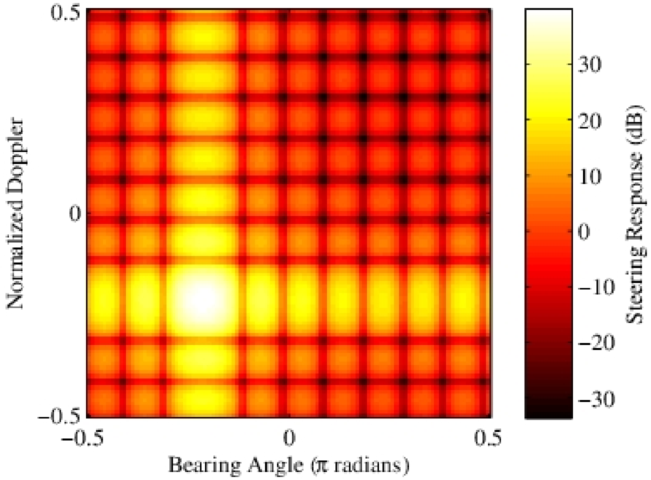
- **Example: STAP**
- **Cross-call/cross library optimization with Spiral**
- **Library-based hardware acceleration**
- **Summary**

T. M. Low, Q. Guo, F. Franchetti: **Optimizing Space Time Adaptive Processing Through Accelerating Memory-Bounded Operations.** IEEE High Performance Extreme Computing Conference (HPEC), 2015.

# Space Time Adaptive Processing (STAP)

dense data cube

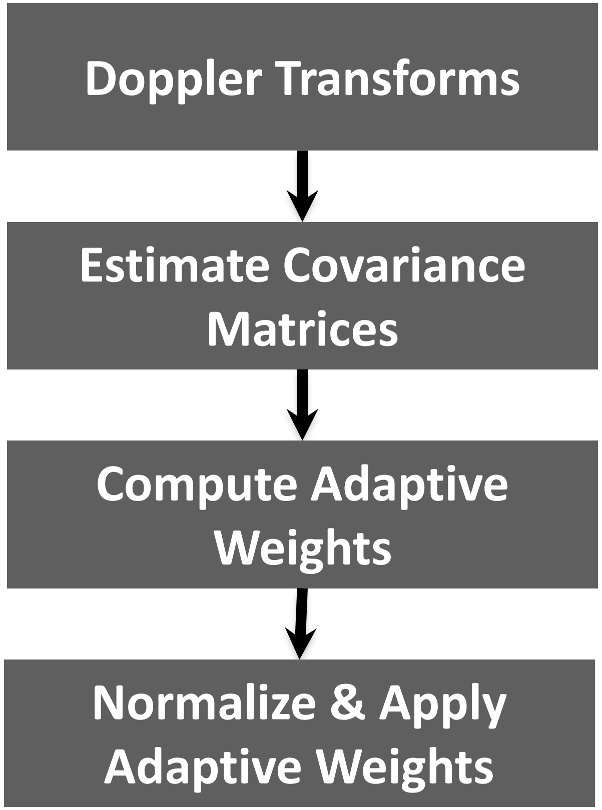




**Input: 32 MB**
**Output: 128 MB**
**Millions of Math Ops**

**SWAP requirement: high performance, low power**

# Main Computational Stages in STAP



| | | |
|---|---|---|
| Doppler Transforms | O(10k) | FFTs +<br>Data Layout Transforms |
| Estimate Covariance Matrices | O(1M) | Outer-products |
| Compute Adaptive Weights | O(1M) | Matrix Factorization<br>Triangular Solve (vector) |
| Normalize & Apply Adaptive Weights | O(10M)<br>O(100k) | Inner-products<br>Vector Scaling |

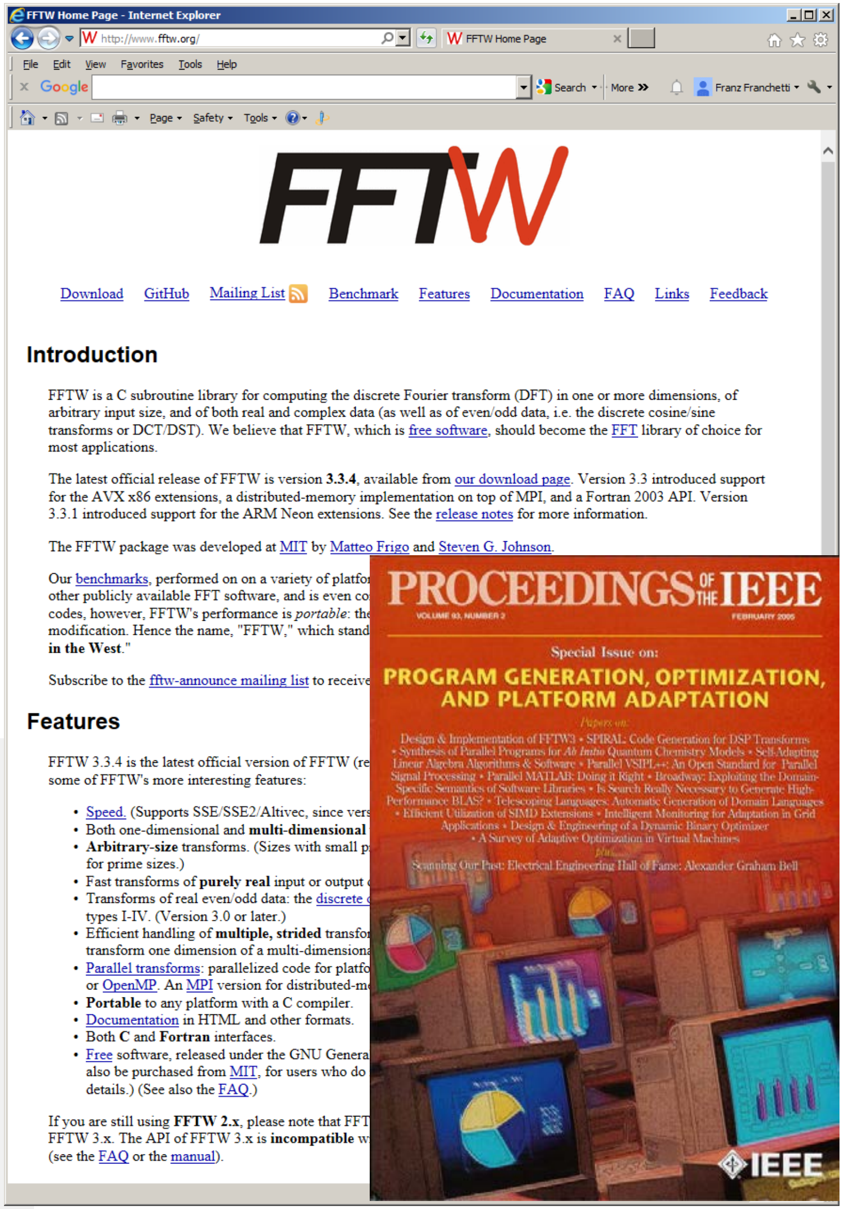
Many memory-bounded operations

Based on PNNL's Third-order Doppler STAP implementation (DARPA PERFECT STAP Benchmark)

# Background: FFTW 3

- **Latest version of FFTW**
  supports threading, SIMD, MPI
- **De-facto standard FFT library for HPC**
  distributed with Linux, installed everywhere
- **Autotuning + program generation**
  small hand-written core
- **Interface widely supported**
  Intel MKL, IBM ESSL, AMD ACML,…

```
#include <fftw3.h>
...
{
   fftw_complex *in, *out;
   fftw_plan p;
   ...
   in = (fftw_complex*) fftw_malloc(sizeof(fftw_complex) * N);
   out = (fftw_complex*) fftw_malloc(sizeof(fftw_complex) * N);
   p = fftw_plan_dft_1d(N, in, out, FFTW_FORWARD, FFTW_ESTIMATE);
   ...
   fftw_execute(p); /* repeat as needed */
   ...
   fftw_destroy_plan(p);
   fftw_free(in); fftw_free(out);}
```


FFTW Home Page - Internet Explorer
http://www.fftw.org/
FFTW Home Page
File Edit View Favorites Tools Help
Google
Search
More
Franz Franchetti
Page Safety Tools
FFTW
Download GitHub Mailing List Benchmark Features Documentation FAQ Links Feedback

## Introduction

FFTW is a C subroutine library for computing the discrete Fourier transform (DFT) in one or more dimensions, of arbitrary input size, and of both real and complex data (as well as of even/odd data, i.e. the discrete cosine/sine transforms or DCT/DST). We believe that FFTW, which is free software, should become the FFT library of choice for most applications.

The latest official release of FFTW is version **3.3.4**, available from our download page. Version 3.3 introduced support for the AVX x86 extensions, a distributed-memory implementation on top of MPI, and a Fortran 2003 API. Version 3.3.1 introduced support for the ARM Neon extensions. See the release notes for more information.

The FFTW package was developed at MIT by Matteo Frigo and Steven G. Johnson.

Our benchmarks, performed on on a variety of platfo
other publicly available FFT software, and is even co
codes, however, FFTW's performance is *portable*: the
modification. Hence the name, "FFTW," which stand
**in the West**."

Subscribe to the fftw-announce mailing list to receive

## Features

FFTW 3.3.4 is the latest official version of FFTW (re
some of FFTW's more interesting features:

- Speed. (Supports SSE/SSE2/Altivec, since vers
- Both one-dimensional and **multi-dimensional**
- **Arbitrary-size** transforms. (Sizes with small p
  for prime sizes.)
- Fast transforms of **purely real** input or output
- Transforms of real even/odd data: the discrete
  types I-IV. (Version 3.0 or later.)
- Efficient handling of **multiple, strided** transfo
  transform one dimension of a multi-dimensiona
- Parallel transforms: parallelized code for platfo
  or OpenMP. An MPI version for distributed-m
- **Portable** to any platform with a C compiler.
- Documentation in HTML and other formats.
- Both **C** and **Fortran** interfaces.
- Free software, released under the GNU Genera
  also be purchased from MIT, for users who do
  details.) (See also the FAQ.)

If you are still using **FFTW 2.x**, please note that FFT
FFTW 3.x. The API of FFTW 3.x is **incompatible** w
(see the FAQ or the manual).

PROCEEDINGS OF THE IEEE

VOLUME 93, NUMBER 2 FEBRUARY 2005

Special Issue on:

PROGRAM GENERATION, OPTIMIZATION, AND PLATFORM ADAPTATION

*Papers on:*

Design & Implementation of FFTW3 • SPIRAL: Code Generation for DSP Transforms • Synthesis of Parallel Programs for *Ab Initio* Quantum Chemistry Models • Self-Adapting Linear Algebra Algorithms & Software • Parallel VSIPL++: An Open Standard for Parallel Signal Processing • Parallel MATLAB: Doing it Right • Broadway: Exploiting the Domain-Specific Semantics of Software Libraries • Is Search Really Necessary to Generate High-Performance BLAS? • Telescoping Languages: Automatic Generation of Domain Languages • Efficient Utilization of SIMD Extensions • Intelligent Monitoring for Adaptation in Grid Applications • Design & Engineering of a Dynamic Binary Optimizer • A Survey of Adaptive Optimization in Virtual Machines

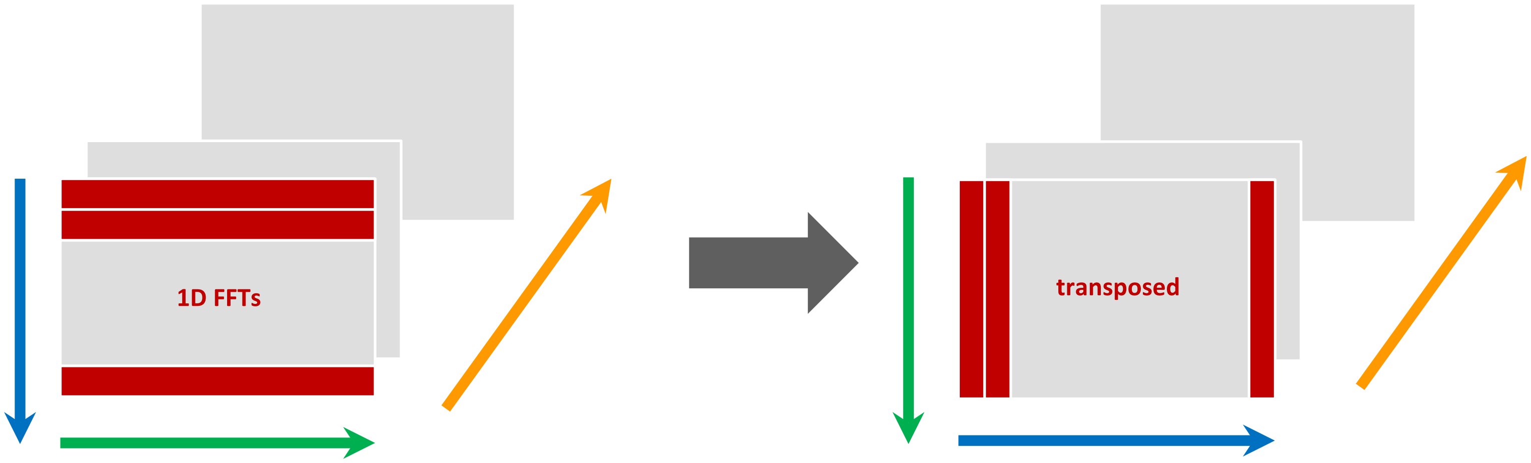
*plus...*

Scanning Our Past: Electrical Engineering Hall of Fame: Alexander Graham Bell

IEEE

Matteo Frigo and Steven G. Johnson: **The Design and Implementation of FFTW3.** Proceedings of the IEEE 93 (2), 216–231 (2005). Invited paper, Special Issue on Program Generation, Optimization, and Platform Adaptation.

# Doppler Transform: FFT + Corner Turn



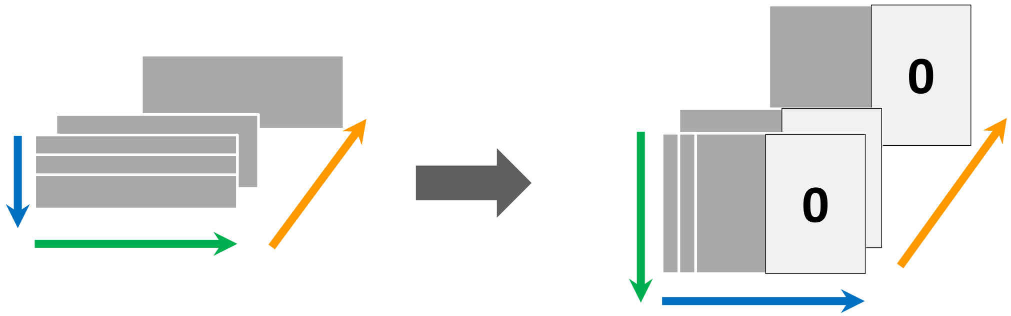
```
for (chan = 0; chan < N_CHAN; ++chan)
{
  for (range = 0; range < N_RANGE; ++range)
  {
    /* Apply a 1D FFT to convert to Doppler space */
    fft(
     (float *)&datacube_pulse_major_padded[chan][range][0],
     N_DOP,
     N_DOP_LOG2,
     FFT_FORWARD);
  }
}
/* Transpose from doppler major to range major*/
corner_turn_to_range_major_order
(doppler_datacube, datacube_pulse_major_padded);
```

```
/* Replace with fftw guru interface*/
plan = fftwf_plan_guru_dft(
            1, dims,
            2, howmany_dims,
            datacube_pulse_major_padded,
            doppler_datacube,
            FFTW_FORWARD,
            FFTW_WISDOM_ONLY);

fftwf_execute(plan);
fftwf_destroy_plan(plan);
```

**Combine loop of FFTs plus subsequent corner turn into one FFTW call**

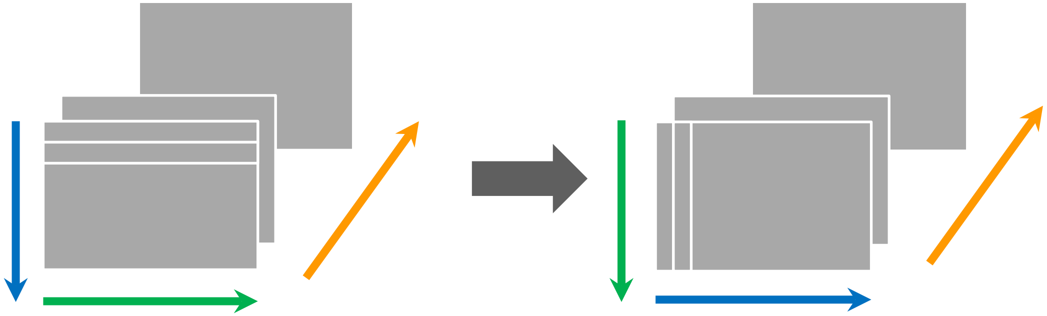
# Use FFTW Guru Interface for Data Copy

## Zero pad plus batch transpose



```
const fftwf_iodim howmany_dims[3] =
{{n:N_RANGE, is:1, os:N_DOP},
 {n:N_PULSES, is:N_RANGE, os:1},
 {n:N_CHAN,
  is:N_RANGE*N_PULSES,
  os:N_RANGE*N_DOP}};

plan =
fftwf_plan_guru_dft(0, NULL,
            3, howmany_dims,
            datacube,
            datacube_pulse_major_padded,
            FFTW_FORWARD,
            FFTW_ESTIMATE);

fftwf_execute(plan);
```

## Batch transpose



```
const fftwf_iodim howmany_dims[3] =
{{n:N_DOP, is:1, os:N_RANGE},
 {n:N_RANGE, is:N_DOP, os:1},
 {n:N_CHAN,
  is:N_RANGE*N_DOP,
  os:N_RANGE*N_DOP}};

plan =
fftwf_plan_guru_dft(0, NULL,
            3, howmany_dims,
            doppler_major,
            range_major,
            FFTW_FORWARD,
            FFTW_ESTIMATE);

fftwf_execute(plan);
```

**FFTW Rank-0 FFT abstracts copy, transpose, gather, scatter**

# Borrowing FFTW’s Guru Interface for BLAS

## FFTW guru interface

strong support for batch FFTs and n-dimensional data cubes

```
typedef struct{
  int n; // size of the dimension
  int is; // stride for input
  int os; // stride for output
} fftw_iodim;
```

```
/* FFTW Guru Interface */
fftw_plan fftw_plan_guru_dft(
    int rank, const fftw_iodim *dims,
    int howmany_rank, const fftw_iodim *howmany_dims,
    fftw_complex *in, fftw_complex *out,
    int sign, unsigned flags);
```

## Standard BLAS interface

Fortran 77 style interface

```
void cblas_cdotc_sub (const int N, const void * x, const int incx,
const void * y, const int incy, void * dotc);
```

## Generalization: BLAS guru interface

borrow FFTW’s data and batch representation for BLAS operations

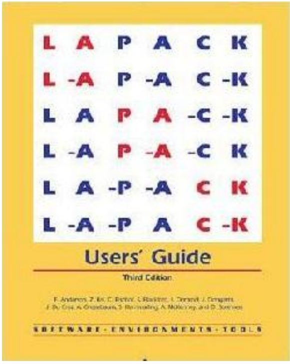
```
typedef struct{
  int n;
  int i0s; // stride for input0
  int i1s; // stride for input1
} blas_indim;
```

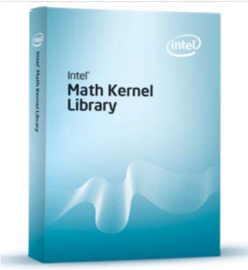
```
/* BLAS Guru Configuration Interface */
blas_conf blas_conf_guru(
// (rank, dims[rank]) describes the basic BLAS operation size
    int rank,
    blas_indim *dims,
// (howmany_rank, howmany_dims[rank]) describes the “vector” size
    int howmany_rank,
    blas_indim *howmany_dims);
```

**Use BLAS guru interface in backend but do not expose to end user**

# Use OpenMP to Avoid Changing BLAS Interface

- **Absolutely cannot change BLAS**
  set-in-stone standard since 1979
- **Supported by everybody**
  MKL, ESSL, ACML, CUBLAS,…
- **Idea: Use OpenMP to express batch**
  mark loops as independent
- **Use compiler to extract batch BLAS descriptor**
  automatically translate BLAS + OpenMP to BLAS Guru Interface





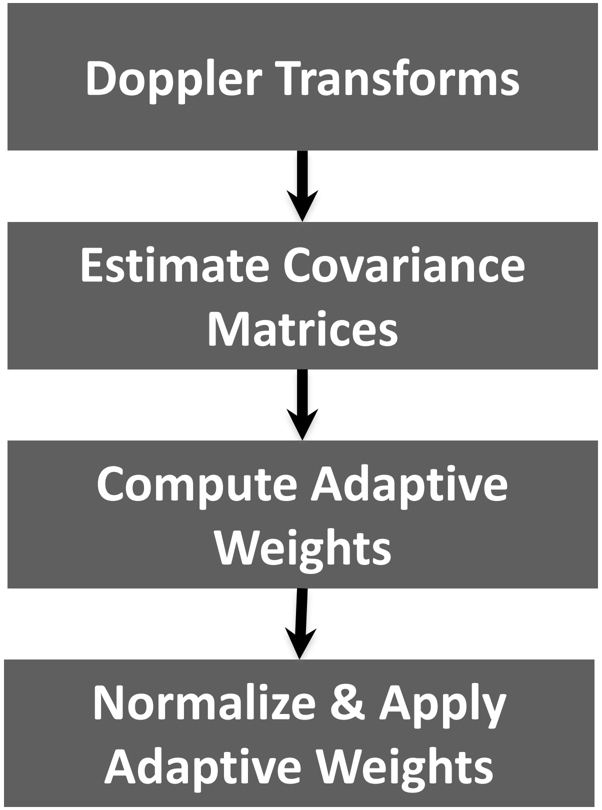
```
#pragma omp for
for (sv = 0; sv < N_STEERING; ++sv) {
#pragma omp for
  for (block = 0; block < N_DOP*N_BLOCKS; ++block) {
    cblas_cdotc_sub(TDOF*N_CHAN,
       (float*)adaptive_weights[block][sv],
       1,
       (float*)steering_vectors[sv],
       1,
       (float*)&accums[block][sv]);
  }
}
```

STAP rewritten with BLAS + OpenMP

compiler

```
blas_guru(
    1,
    {{n:TDOF*N_CHAN, i0s:1, i1s:1}}
    2,
    {{n:N_DOP*N_BLOCKS,
      i0s:N_STEERING*TDOF*N_CHAN,
      i1s:1},
     {n:N_STEERING,
      i0s:TDOF*N_CHAN,
      i1s:TDOF*N_CHAN}});
```

# After Transformation: STAP = 4 Library Calls



Doppler Transforms

Estimate Covariance Matrices

Compute Adaptive Weights

Normalize & Apply Adaptive Weights

FFTW call

BLAS3 call

BLAS3 call

BLAS2 call

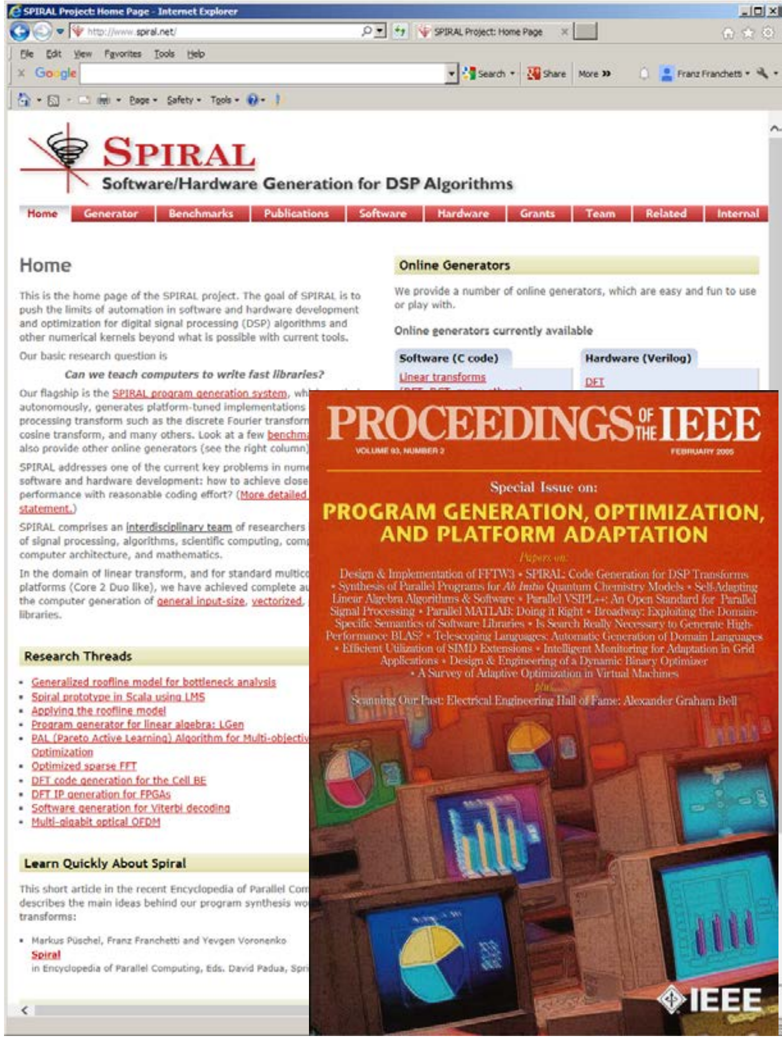
**Result: STAP is expressed using four library calls + OpenMP directives**

# Outline

- Example: STAP
- **Cross-call/cross library optimization with Spiral**
- Library-based hardware acceleration
- Summary

# DSL Compiler for Global Transformations

- **Parse library calls and OpenMP**
  convert to Spiral's operator language (OL)
- **Spiral performs global optimizations**
  kernel merging, data layout transformations,…
- **Output CPU or accelerator code**
  run with Intel MKLK or our own accelerator
- **Utilize BLAS Guru interface**
  our accelerator implements BLAS Guru



SPIRAL

Software/Hardware Generation for DSP Algorithms

Home

This is the home page of the SPIRAL project. The goal of SPIRAL is to push the limits of automation in software and hardware development and optimization for digital signal processing (DSP) algorithms and other numerical kernels beyond what is possible with current tools.

Our basic research question is

*Can we teach computers to write fast libraries?*

Our flagship is the SPIRAL program generation system, wh autonomously, generates platform-tuned implementations processing transform such as the discrete Fourier transform cosine transform, and many others. Look at a few benchma also provide other online generators (see the right column)

SPIRAL addresses one of the current key problems in nume software and hardware development: how to achieve close performance with reasonable coding effort? (More detailed statement.)

SPIRAL comprises an interdisciplinary team of researchers of signal processing, algorithms, scientific computing, comp computer architecture, and mathematics.

In the domain of linear transform, and for standard multico platforms (Core 2 Duo like), we have achieved complete au the computer generation of general input-size, vectorized, libraries.

Research Threads

- Generalized roofline model for bottleneck analysis
- Spiral prototype in Scala using LMS
- Applying the roofline model
- Program generator for linear algebra: LGen
- PAL (Pareto Active Learning) Algorithm for Multi-objectiv Optimization
- Optimized sparse FFT
- DFT code generation for the Cell BE
- DFT IP generation for FPGAs
- Software generation for Viterbi decoding
- Multi-gigabit optical OFDM

Learn Quickly About Spiral

This short article in the recent Encyclopedia of Parallel Com describes the main ideas behind our program synthesis wo transforms:

- Markus Püschel, Franz Franchetti and Yevgen Voronenko
  Spiral
  in Encyclopedia of Parallel Computing, Eds. David Padua, Spri

Online Generators

We provide a number of online generators, which are easy and fun to use or play with.

Online generators currently available

Software (C code)
Linear transforms

Hardware (Verilog)
DFT

PROCEEDINGS OF THE IEEE

VOLUME 93, NUMBER 2 FEBRUARY 2005

Special Issue on:

PROGRAM GENERATION, OPTIMIZATION, AND PLATFORM ADAPTATION

*Papers on:*

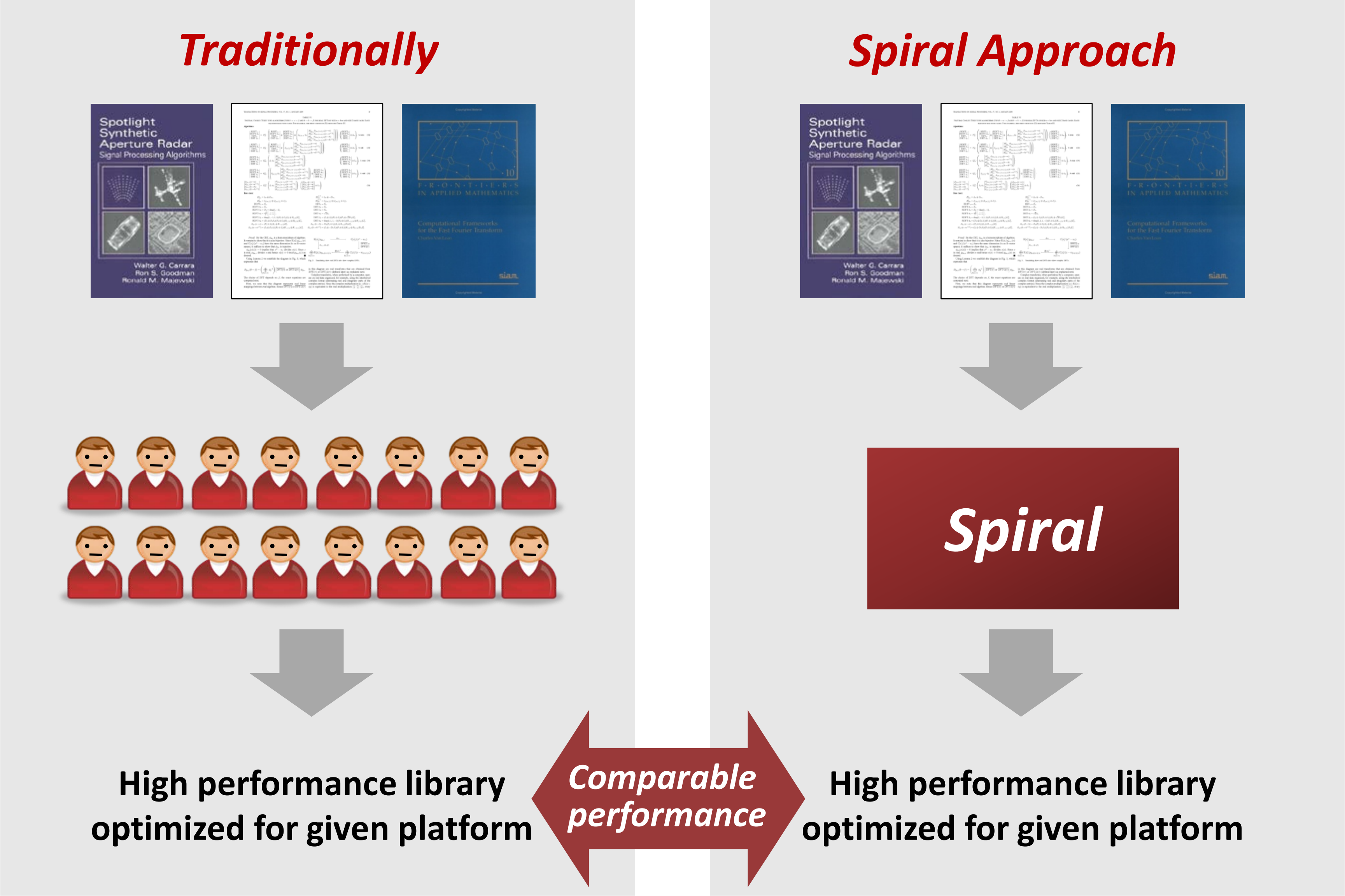
Design & Implementation of FFTW3 • SPIRAL: Code Generation for DSP Transforms • Synthesis of Parallel Programs for *Ab Initio* Quantum Chemistry Models • Self-Adapting Linear Algebra Algorithms & Software • Parallel VSIPL++: An Open Standard for Parallel Signal Processing • Parallel MATLAB: Doing it Right • Broadway: Exploiting the Domain-Specific Semantics of Software Libraries • Is Search Really Necessary to Generate High-Performance BLAS? • Telescoping Languages: Automatic Generation of Domain Languages • Efficient Utilization of SIMD Extensions • Intelligent Monitoring for Adaptation in Grid Applications • Design & Engineering of a Dynamic Binary Optimizer • A Survey of Adaptive Optimization in Virtual Machines

*plus*

Scanning Our Past: Electrical Engineering Hall of Fame: Alexander Graham Bell

M. Püschel, J. Moura, J. Johnson, D. Padua, M. Veloso, B. Singer, J. Xiong, F. Franchetti, A. Gacic, Y. Voronenko, K. Chen, R. W. Johnson, N. Rizzolo, **"SPIRAL: Code Generation for DSP Transforms,"** Proceedings of the IEEE *Special Issue on Program Generation, Optimization, and Adaptation*, 2005

# What is Spiral?



## *Traditionally*

## *Spiral Approach*

Spotlight Synthetic Aperture Radar
Signal Processing Algorithms
Walter G. Carrara
Ron S. Goodman
Ronald M. Majewski
F·R·O·N·T·I·E·R·S IN APPLIED MATHEMATICS
10
Computational Frameworks for the Fast Fourier Transform
Charles Van Loan
siam

*Spiral*

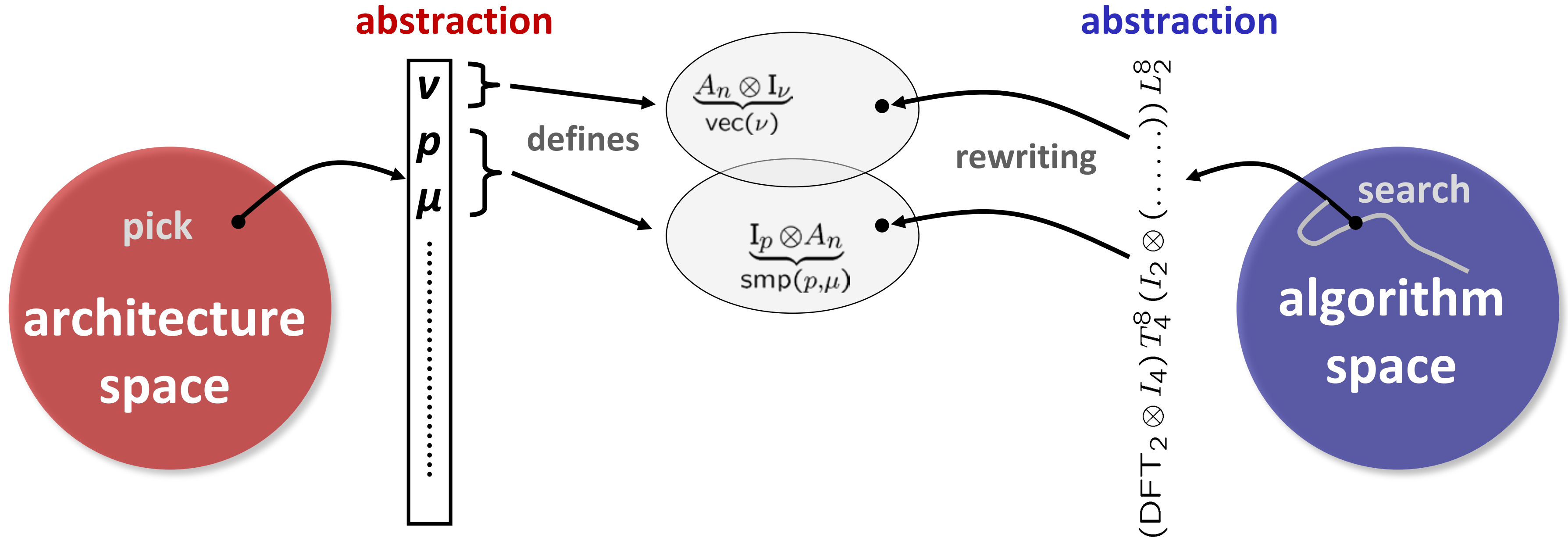
High performance library optimized for given platform

*Comparable performance*

High performance library optimized for given platform

# Platform-Aware Formal 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

# Operators

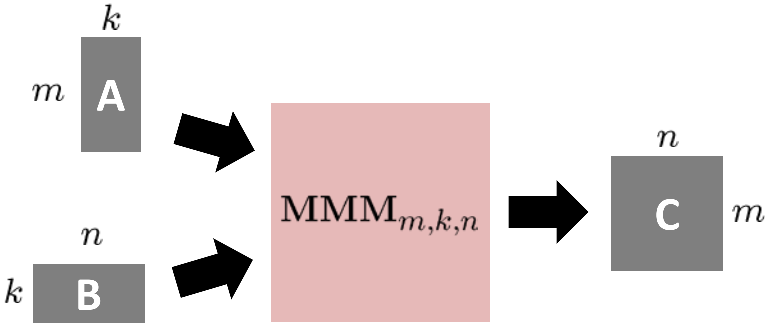
## Definition

- Operator: Multiple complex vectors → multiple complex vectors
- Higher-dimensional data is linearized
- Operators are potentially nonlinear

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

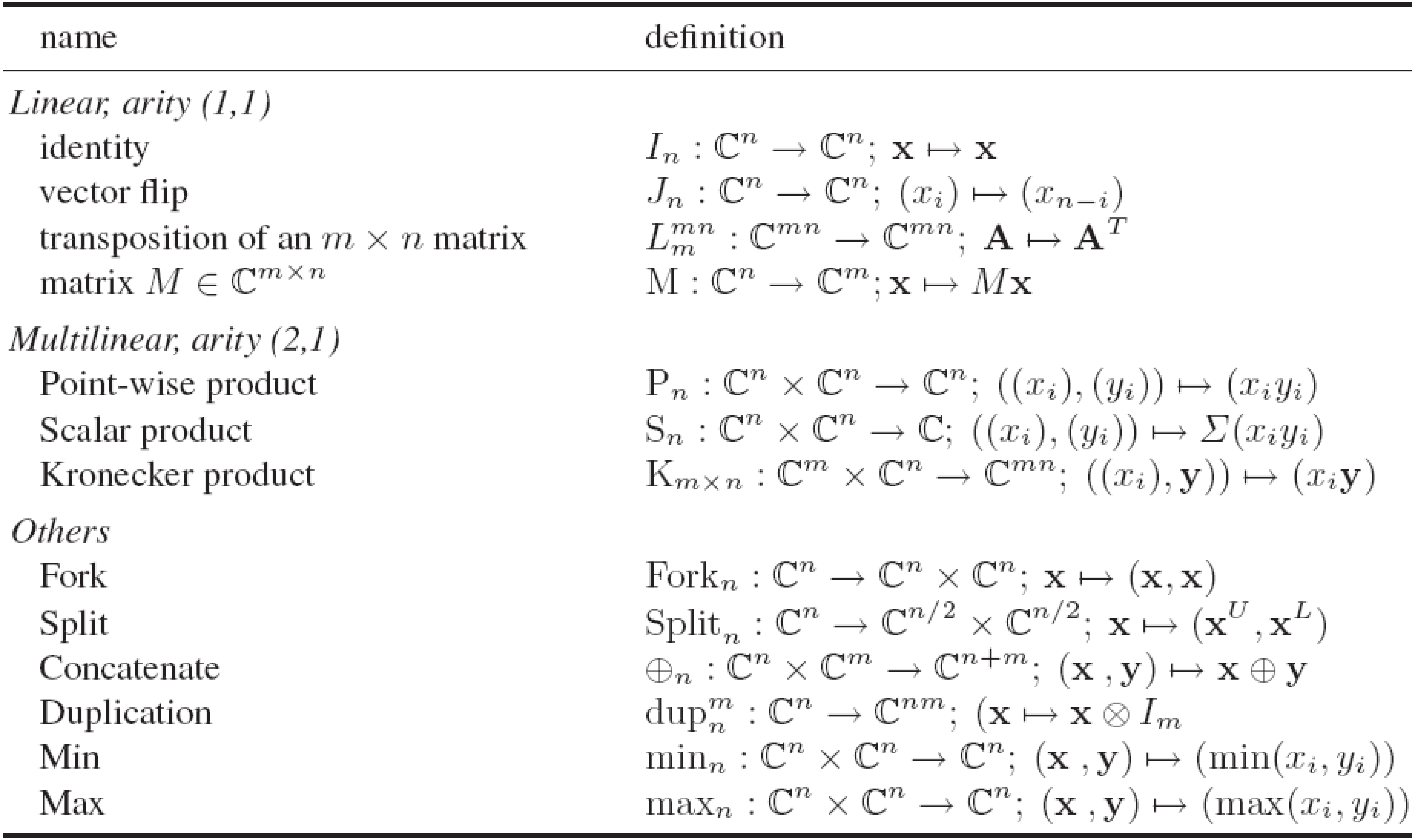
## Example: Matrix-matrix-multiplication (MMM)

$$\begin{aligned} \mathrm{MMM}_{m,k,n} : \mathbb{R}^{mk} \times \mathbb{R}^{kn} &\to \mathbb{R}^{mn} \\ (\mathbf{A}, \mathbf{B}) &\mapsto \mathbf{AB} \end{aligned}$$



**Key to capture FFTs, numerical linear algebra, message passing in one framework**

# Operator Language



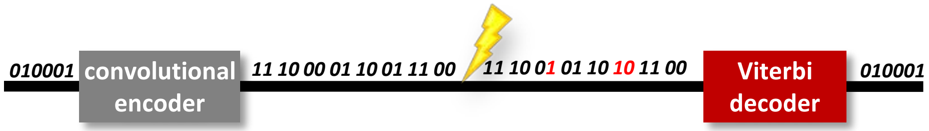
| name | definition |
|---|---|
| *Linear, arity (1,1)* | |
| identity | $I_n : \mathbb{C}^n \rightarrow \mathbb{C}^n;\ \mathbf{x} \mapsto \mathbf{x}$ |
| vector flip | $J_n : \mathbb{C}^n \rightarrow \mathbb{C}^n;\ (x_i) \mapsto (x_{n-i})$ |
| transposition of an $m \times n$ matrix | $L_m^{mn} : \mathbb{C}^{mn} \rightarrow \mathbb{C}^{mn};\ \mathbf{A} \mapsto \mathbf{A}^T$ |
| matrix $M \in \mathbb{C}^{m \times n}$ | $\mathrm{M} : \mathbb{C}^n \rightarrow \mathbb{C}^m; \mathbf{x} \mapsto M\mathbf{x}$ |
| *Multilinear, arity (2,1)* | |
| Point-wise product | $\mathrm{P}_n : \mathbb{C}^n \times \mathbb{C}^n \rightarrow \mathbb{C}^n;\ ((x_i), (y_i)) \mapsto (x_i y_i)$ |
| Scalar product | $\mathrm{S}_n : \mathbb{C}^n \times \mathbb{C}^n \rightarrow \mathbb{C};\ ((x_i), (y_i)) \mapsto \Sigma(x_i y_i)$ |
| Kronecker product | $\mathrm{K}_{m \times n} : \mathbb{C}^m \times \mathbb{C}^n \rightarrow \mathbb{C}^{mn};\ ((x_i), \mathbf{y})) \mapsto (x_i \mathbf{y})$ |
| *Others* | |
| Fork | $\mathrm{Fork}_n : \mathbb{C}^n \rightarrow \mathbb{C}^n \times \mathbb{C}^n;\ \mathbf{x} \mapsto (\mathbf{x}, \mathbf{x})$ |
| Split | $\mathrm{Split}_n : \mathbb{C}^n \rightarrow \mathbb{C}^{n/2} \times \mathbb{C}^{n/2};\ \mathbf{x} \mapsto (\mathbf{x}^U, \mathbf{x}^L)$ |
| Concatenate | $\oplus_n : \mathbb{C}^n \times \mathbb{C}^m \rightarrow \mathbb{C}^{n+m};\ (\mathbf{x}, \mathbf{y}) \mapsto \mathbf{x} \oplus \mathbf{y}$ |
| Duplication | $\mathrm{dup}_n^m : \mathbb{C}^n \rightarrow \mathbb{C}^{nm};\ (\mathbf{x} \mapsto \mathbf{x} \otimes I_m$ |
| Min | $\min_n : \mathbb{C}^n \times \mathbb{C}^n \rightarrow \mathbb{C}^n;\ (\mathbf{x}, \mathbf{y}) \mapsto (\min(x_i, y_i))$ |
| Max | $\max_n : \mathbb{C}^n \times \mathbb{C}^n \rightarrow \mathbb{C}^n;\ (\mathbf{x}, \mathbf{y}) \mapsto (\max(x_i, y_i))$ |

# Some Application Domains in OL

## Linear Transforms

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

## Software Defined Radio

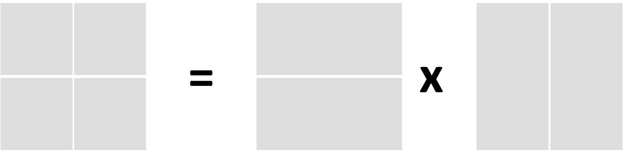


$$
F_{K,F} \rightarrow \prod_{i=1}^{F} \left( (I_{2^{K-2}} \otimes_j B_{F-i,j}) L^{2^{K-1}}_{2^{K-2}} \right)
$$

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

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

## Matrix-Matrix Multiplication



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

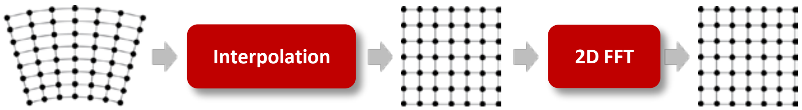
$$\mathrm{MMM}_{m,n,k} \rightarrow (\otimes)_{m/m_b \times 1} \otimes \mathrm{MMM}_{m_b,n,k}$$

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

$$
\begin{array}{l}
\mathrm{MMM}_{m,n,k} \rightarrow ((\Sigma_{k/k_b} \circ (\cdot)_{k/k_b}) \otimes \mathrm{MMM}_{m,n,k_b}) \circ \\
\qquad\qquad\qquad ((L^{mk/k_b}_{k/k_b} \otimes I_{k_b}) \times I_{kn})
\end{array}
$$

$$
\begin{array}{l}
\mathrm{MMM}_{m,n,k} \rightarrow (L^{mn/n_b}_m \otimes I_{n_b}) \circ \\
\qquad ((\otimes)_{1 \times n/n_b} \otimes \mathrm{MMM}_{m,n_b,k}) \circ \\
\qquad\qquad\qquad (I_{km} \times (L^{kn/n_b}_{n/n_b} \otimes I_{n_b}))
\end{array}
$$

## Synthetic Aperture Radar (SAR)



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

# Formal Approach for all Types of Parallelism

- **Multithreading** (Multicore)

  $\mathrm{I}_p \otimes_{\|} A_{\mu n}, \quad \mathbf{L}^{mn}_{m} \bar{\otimes} \mathrm{I}_{\mu}$

- **Vector SIMD** (SSE, VMX/Altivec,…)

  $A \hat{\otimes} \mathrm{I}_{\nu} \quad \underbrace{\mathbf{L}^{2\nu}_{2}}_{\text{isa}}, \quad \underbrace{\mathbf{L}^{2\nu}_{\nu}}_{\text{isa}}, \quad \underbrace{\mathbf{L}^{\nu^2}_{\nu}}_{\text{isa}}$

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

  $\mathrm{I}_p \otimes_{\|} A_{n}, \quad \underbrace{\mathbf{L}^{p^2}_{p} \bar{\otimes} \mathrm{I}_{n/p^2}}_{\text{all-to-all}}$

- **Streaming/multibuffering** (Cell)

  $\mathrm{I}_n \otimes_2 A_{\mu n}, \quad \mathbf{L}^{mn}_{m} \bar{\otimes} \mathrm{I}_{\mu}$

- **Graphics Processors** (GPUs)

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

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

  ${\prod_{i=0}^{n-1}}^{\text{ir}} A, \quad \mathrm{I}_s \tilde{\otimes} A, \quad \underbrace{\mathbf{L}^{m}_{n}}_{\text{bram}}$

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

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

# Autotuning in Constraint Solution Space

Intel Core i7 (2nd Gen)

DFT$_{256}$

## Base cases

$\mathbf{I}_4 \otimes_{\|} A_{16n}$

$\mathbf{L}^{mn}_m \bar{\otimes} \mathbf{I}_{16}$

$A \hat{\otimes} \mathbf{I}_4$

$\underbrace{\mathbf{L}^8_2}_{\text{SSE}}, \underbrace{\mathbf{L}^8_4}_{\text{SSE}}, \underbrace{\mathbf{L}^{16}_4}_{\text{SSE}}, \underbrace{\mathbf{L}^4_2 \otimes \mathbf{I}_2}_{\text{SSE}}$

## Transformation rules

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

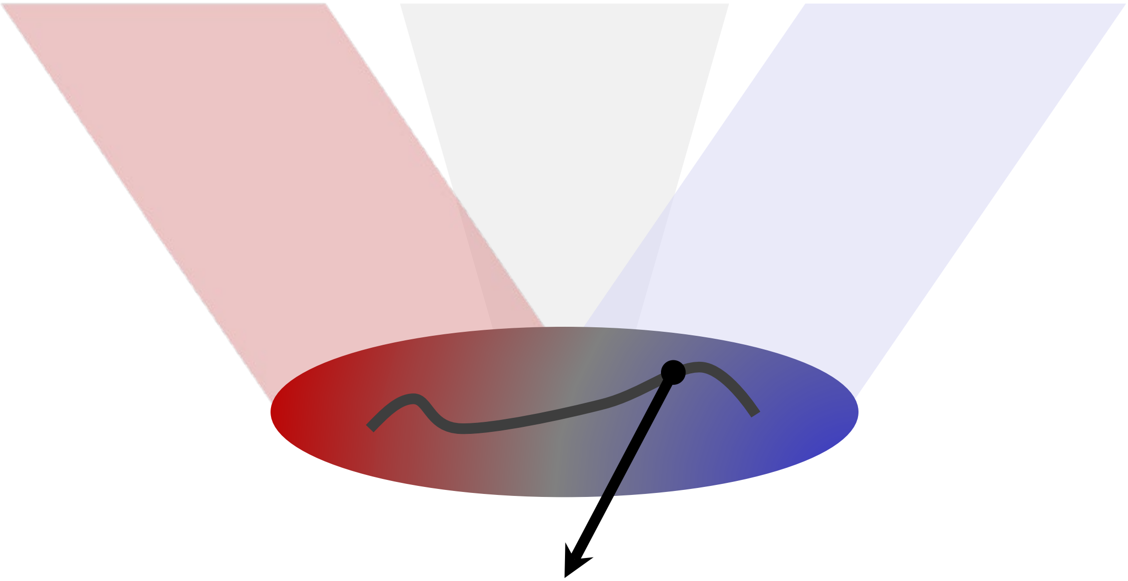
$$\underbrace{\mathsf{L}^{mn}_m}_{\text{smp}(p,\mu)} \to \begin{cases} \underbrace{\left(\mathbf{I}_p \otimes \mathsf{L}^{mn/p}_{m/p}\right)}_{\text{smp}(p,\mu)} \underbrace{\left(\mathsf{L}^{pn}_p \otimes \mathbf{I}_{m/p}\right)}_{\text{smp}(p,\mu)} \\ \underbrace{\left(\mathsf{L}^{pm}_m \otimes \mathbf{I}_{n/p}\right)}_{\text{smp}(p,\mu)}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

$$\underbrace{\mathbf{I}_m \otimes A_n}_{\text{smp}(p,\mu)} \to \mathbf{I}_p \otimes_{\|} \left(\mathbf{I}_{m/p} \otimes A_n\right)$$

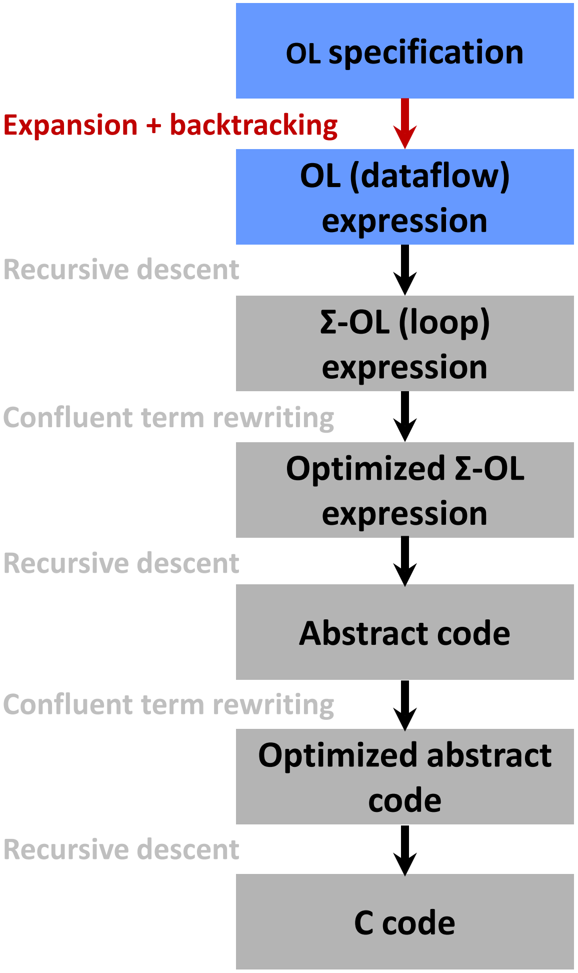
$$\ldots$$

## Breakdown rules

$$\begin{aligned} \mathbf{DFT}_n &\to (\mathbf{DFT}_k \otimes \mathbf{I}_m)\, \mathsf{T}^n_m \\ &\quad \cdot (\mathbf{I}_k \otimes \mathbf{DFT}_m)\, \mathsf{L}^n_k \\ \mathbf{DFT}_n &\to P_n(\mathbf{DFT}_k \otimes \mathbf{DFT}_m) Q_n \\ \mathbf{DFT}_p &\to R^T_p(\mathbf{I}_1 \oplus \mathbf{DFT}_{p-1}) D_p \\ &\quad \cdot (\mathbf{I}_1 \oplus \mathbf{DFT}_{p-1}) R_p \\ \mathbf{DFT}_2 &\to \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \end{aligned}$$



$$\left((\mathsf{L}^{mp}_m \otimes \mathbf{I}_{n/p\mu}) \bar{\otimes} \mathbf{I}_\mu\right)\left(\mathbf{I}_p \otimes_{\|} (\mathbf{DFT}_m \otimes \mathbf{I}_{n/p})\right)\left((\mathsf{L}^{mp}_p \otimes \mathbf{I}_{n/p\mu}) \bar{\otimes} \mathbf{I}_\mu\right)\left(\bigoplus_{i=0}^{p-1}{}_{\|}\, \mathsf{T}^{mn,i}_n\right)\left(\mathbf{I}_p \otimes_{\|} (\mathbf{I}_{m/p} \otimes \mathbf{DFT}_n)\right)\left(\mathbf{I}_p \otimes_{\|} \mathsf{L}^{mn/p}_{m/p}\right)\left((\mathsf{L}^{pn}_p \otimes \mathbf{I}_{m/p\mu}) \bar{\otimes} \mathbf{I}_\mu\right.$$



OL specification

Expansion + backtracking

OL (dataflow) expression

Recursive descent

Σ-OL (loop) expression

Confluent term rewriting

Optimized Σ-OL expression

Recursive descent

Abstract code

Confluent term rewriting

Optimized abstract code

Recursive descent

C code

# Translating an OL Expression Into Code

**Constraint Solver Input:** $\underbrace{\mathrm{DFT}_8}_{\mathrm{double}}$

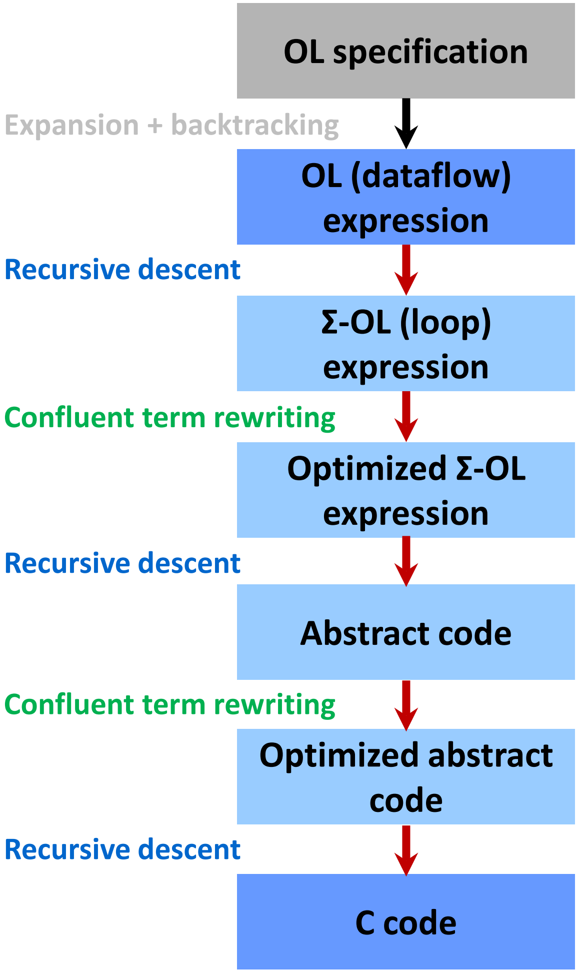
**Output =**

**OL Expression:** $(\mathrm{DFT}_2\otimes I_4)\,T^8_4\left(I_2\otimes\left((\mathrm{DFT}_2\otimes I_2)\,T^4_2\,(I_2\otimes\mathrm{DFT}_2)\,L^4_2\right)\right)L^8_2$

**Σ-OL:** $\sum_{j=0}^{3}\left(S_j\,\mathrm{DFT}_2\,G_j\right)\sum_{k=0}^{1}\left(\sum_{l=0}^{1}\left(S_{k,l}\,\mathrm{diag}\left(\mathrm{t}_{k,l}\right)\mathrm{DFT}_2\,G_l\right)\sum_{m=0}^{1}\left(S_m\,\mathrm{diag}\left(\mathrm{t}_m\right)\mathrm{DFT}_2\,G_{k,m}\right)\right)$

**C Code:**

```
void sub(double *y, double *x) {
  double f0, f1, f2, f3, f4, f7, f8, f10, f11;
  f0 = x[0] - x[3];
  f1 = x[0] + x[3];
  f2 = x[1] - x[2];
  f3 = x[1] + x[2];
  f4 = f1 - f3;
  y[0] = f1 + f3;
  y[2] = 0.7071067811865476 * f4;
  f7 = 0.9238795325112867 * f0;
  f8 = 0.3826834323650898 * f2;
  y[1] = f7 + f8;
  f10 = 0.3826834323650898 * f0;
  f11 = (-0.9238795325112867) * f2;
  y[3] = f10 + f11;
}
```



OL specification

Expansion + backtracking

OL (dataflow) expression

Recursive descent

Σ-OL (loop) expression

Confluent term rewriting

Optimized Σ-OL expression

Recursive descent

Abstract code

Confluent term rewriting

Optimized abstract code

Recursive descent

C code

# STAP in SPIRAL's OL specification

$$(I_D \otimes I_B \otimes (I_V \otimes I_S \otimes (Dot \circ Extract)) \circ (I_V \otimes \frac{1}{\| Dot^{CF} \|_2})) \circ$$

**Inner Products**

$$(I_D \otimes I_B \otimes I_V \otimes (Trsv_{bwd}^{CF} \circ Trsv_{fwd}^{CF})) \circ$$

$$(I_D \otimes I_B \otimes Chol^{CF}) \circ$$

**Linear System Solver**

$$(I_B \otimes I_D \otimes (\frac{1}{S} \times (I_S \otimes (Her^{CF} \circ Extract)))) \circ$$

**Covariance Estimate**

$$(I_C \otimes L_R^{RD}) \circ$$

$$(I_C \otimes I_R \otimes DFT_D)$$

**FFT + Corner Turn**

# STAP: Necessary Optimization

$$(I_D \otimes I_B \otimes (I_V \otimes I_S \otimes (Dot \circ Extract)) \circ (I_V \otimes \frac{1}{\|Dot^{CF}\|_2})) \circ$$

$$(I_D \otimes I_B \otimes I_V \otimes (Trsv^{CF}_{bwd} \circ Trsv^{CF}_{fwd})) \circ$$

$$(I_D \otimes I_B \otimes Chol^{CF}) \circ$$

$$(I_B \otimes I_D \otimes (\frac{1}{S} \times (I_S \otimes (Her^{CF} \circ Extract)))) \circ$$

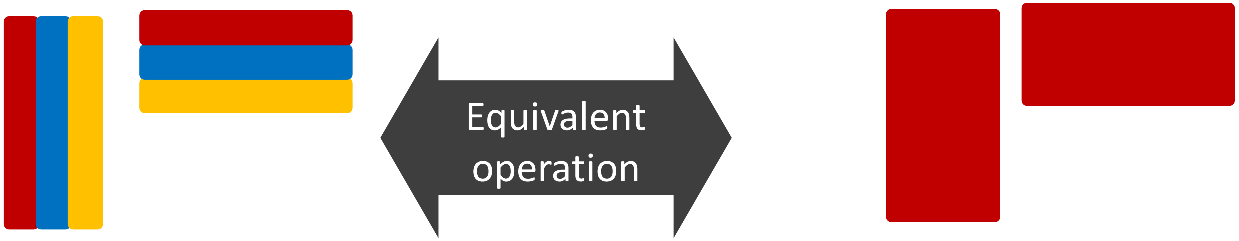
$$(I_C \otimes L^{RD}_R) \circ$$

$$(I_C \otimes I_R \otimes DFT_D)$$

**BLAS2 operations to be optimized into BLAS3 operations**

# Problem: BLAS/LAPACK interface

- Merged Operation

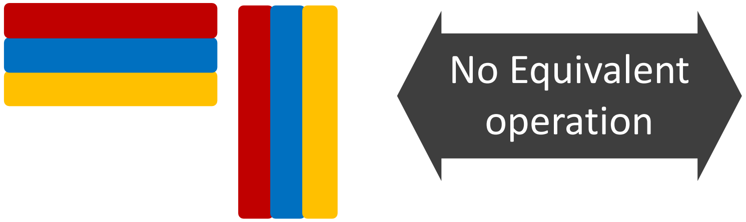


Equivalent operation

```
for i = 0 to 3
    cblas_cher(…)
```

```
cblas_cherk(…)
```

- Batched Operation



No Equivalent operation

?

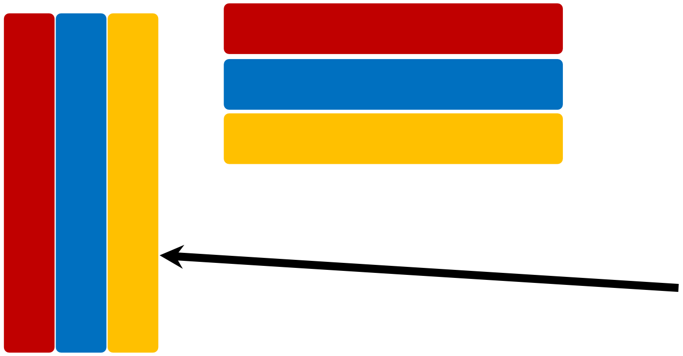
```
for i = 0 to 3
    cblas_cdotu_sub(…)
```

```
blas_guru(…)
```

# STAP: SPIRAL Optimization

**BLAS 2 operation**

$$(I_B \otimes I_D \otimes (\frac{1}{S} \times (I_S \otimes (Her^{CF} \circ Extract))))$$



**Data stored column-major instead of row-major as required**

**After Spiral's transformations**

$$(I_{DB} \otimes (\frac{1}{S} \times (Herk^{CF,S} \circ I_S \otimes (L^{SCF}_{CF} \circ Extract))))$$

**Loop interchange + Loop coalescing**

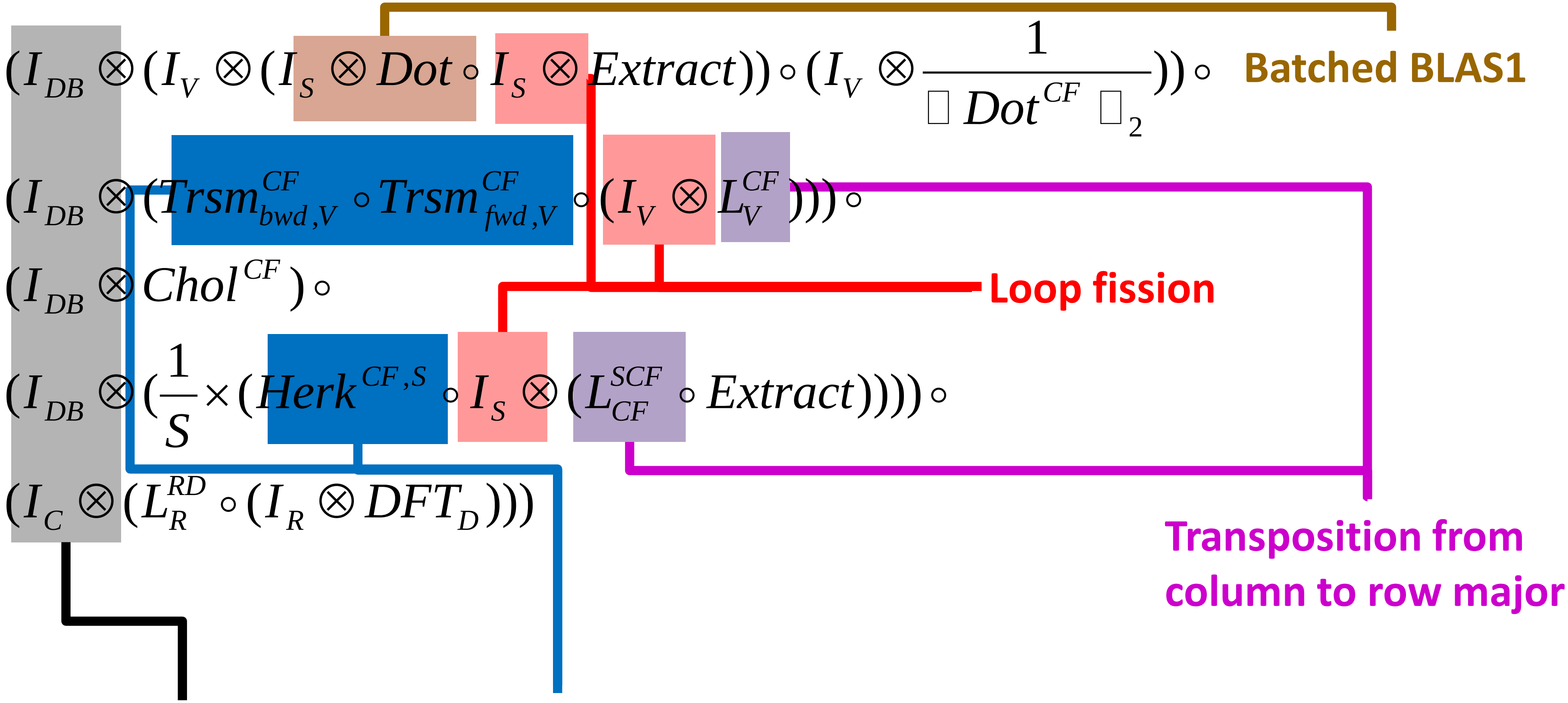
**BLAS2 to BLAS3 call**

**Loop fission**

**Transposition from column to row major**

# Interpreting the OL Specification

After SPIRAL’s transformations



$$
\begin{aligned}
&(I_{DB} \otimes (I_V \otimes (I_S \otimes Dot \circ I_S \otimes Extract)) \circ (I_V \otimes \frac{1}{\| Dot^{CF} \|_2})) \circ \\
&(I_{DB} \otimes (Trsm^{CF}_{bwd,V} \circ Trsm^{CF}_{fwd,V} \circ (I_V \otimes L^{CF}_V))) \circ \\
&(I_{DB} \otimes Chol^{CF}) \circ \\
&(I_{DB} \otimes (\frac{1}{S} \times (Herk^{CF,S} \circ I_S \otimes (L^{SCF}_{CF} \circ Extract)))) \circ \\
&(I_C \otimes (L^{RD}_R \circ (I_R \otimes DFT_D)))
\end{aligned}
$$

Batched BLAS1

Loop fission

Transposition from column to row major

Loop interchange + Loop coalescing

BLAS2 to BLAS3 call

# OL vs. FFTW Guru Inteface

- **FFTW guru-interface**

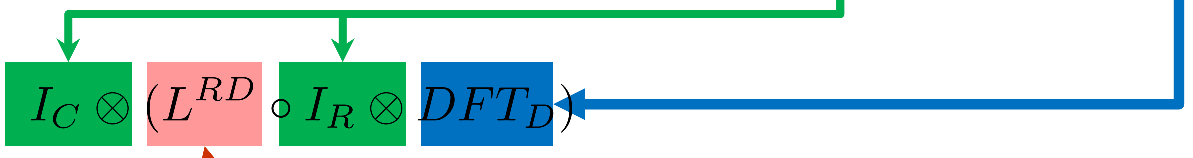
  Batched descriptor for FFT computation

```
fftwf_plan_guru_dft(
    1, dims,
    2, howmany_dims,
    datacube_pulse_major_padded,
    doppler_datacube,
    FFTW_FORWARD,FFTW_WISDOM_ONLY);
```

Compute descriptor

Batch descriptor

- **OL representation of FFTW guru interface**

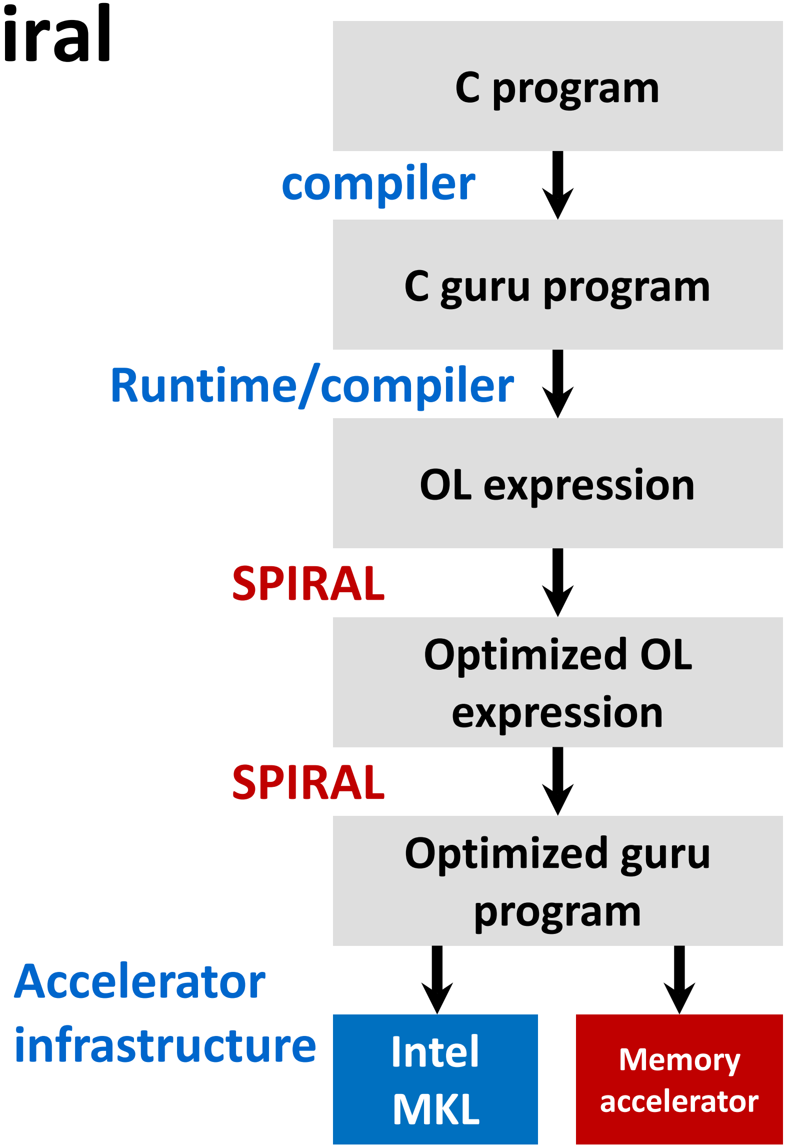


$$I_C \otimes (L^{RD} \circ I_R \otimes DFT_D)$$

Implicit Reshape in Compute Descriptor

# Optimizing STAP with Spiral

- Input: C program with BLAS and FFTW calls
- **Spiral Input:** STAP C program only using FFTW and BLAS guru calls
- This is equivalent to a OL formula in Spiral
- Enables full-program data layout optimization and kernel merging
- **Final program:** efficient on CPU, can target novel accelerators



This paves the way for our 3DIC memory side accelerator (discussed next)

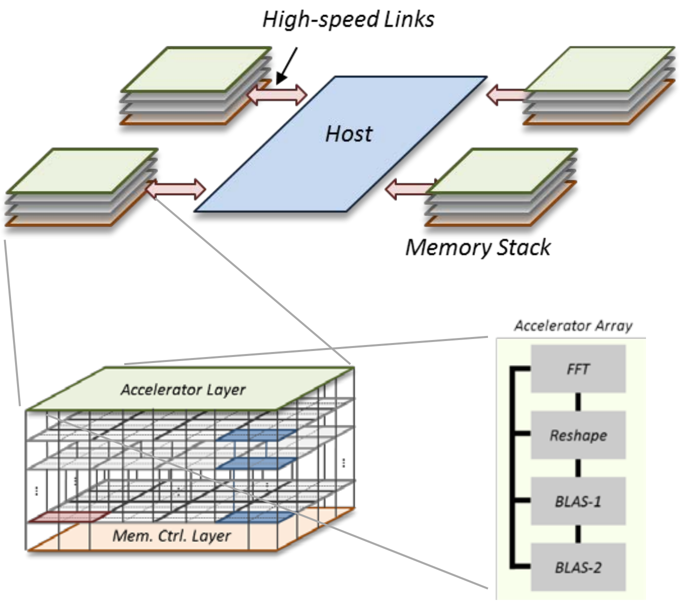
# Outline

- Example: STAP
- Cross-call/cross library optimization with Spiral
- **Library-based hardware acceleration**
- Summary

Q. Guo, T.-M. Low, N. Alachiotis, B. Akin, L. Pileggi, J. C. Hoe, and F. Franchetti: **Enabling Portable Energy Efficiency with Memory Accelerated Library.** IEEE/ACM MICRO-48, 2015, *to appear*.

# Overview: Memory-Side Accelerators

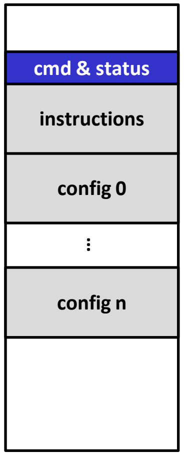
- **Accelerator behind DRAM interface**
  - No off-DIMM data traffic and SERDES
  - Huge problem sizes possible
  - 3D stacking is enabling technology
- **Configurable array of accelerators**
  - Domain specific, highly configurable
  - Cover DoD-relevant kernels
- **System CPU**
  - Standard: Multicore CPU+GPU
- **Software stack**
  - User level: standard numeric libraries BLAS1/2, FFTW, SpMV/SpGEMM,…
  - OS, driver, compiler, runtime system support necessary for drop-in replacement

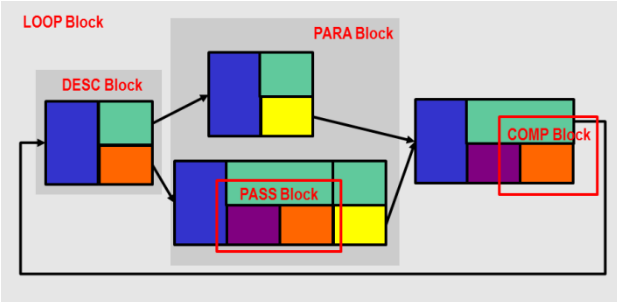


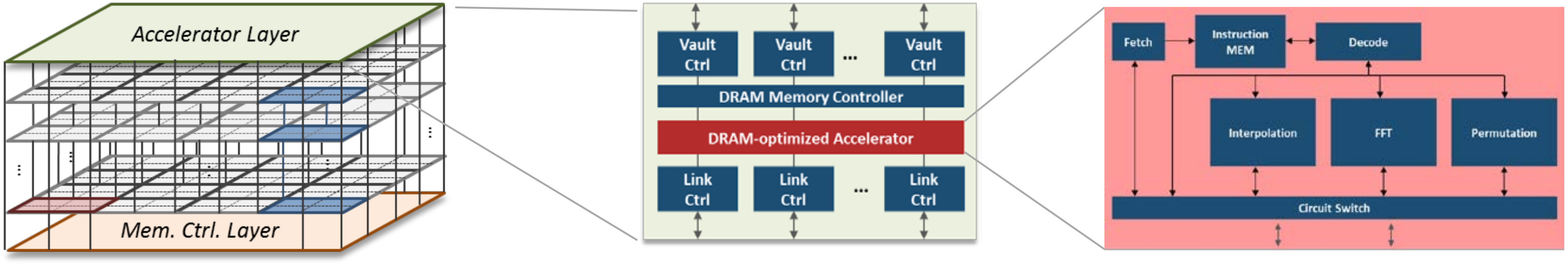
# Accelerator Hardware Architecture

- **Accelerator tiles**
  - One accelerator tile per 3D-DRAM vault
  - Can be linked to one big accelerator
  - Configurable/programmable: units, interconnect, iterators
- **Programming model**
  - Memory mapped device
  - Physical memory addressing
  - Data and command segment in DRAM
  - Controlled via read/writes to command addresses

```
LOOP
 numLoop = 10;
 DESC
  descID = 0;
  numPass = 1;
  PASS
   passID = 0;
   numComp = 1;
   COMP
    compID = 0;
    Inst = FFT;
    Conf = “fft.conf”
   ENDCOMP
  ENDPASS
 ENDDESC
 ...
ENDLOOP
```







# Accelerator Software Architecture

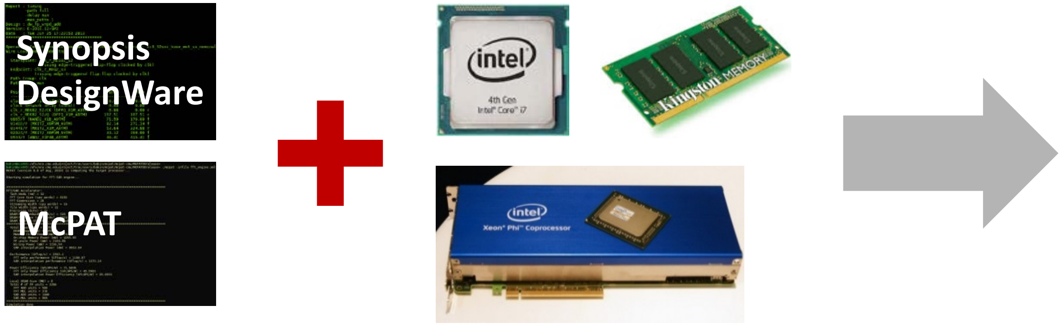
- **Accelerator software abstraction**
  - memory mapped device
  - command and data address space
  - kernel/virtual memory configuration
- **Low level user API**
  - device driver interface
  - low level C configuration library
- **Application level API**
  - transparent device-aware malloc/free
  - standard low intensity math libraries: BLAS-1, BLAS-2, FFTW, sparse mat/vec, copy/reshape, corner turn
  - “guru” math library interfaces
  - OpenACC directives + proto compiler

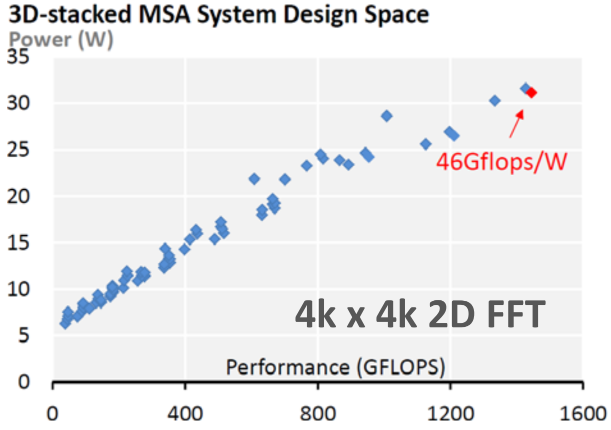
```
// TAV STAP example
//
// kernel reimplemented with BLAS and OpenACC
//

float *adaptive_weights;
float *steering_vectors;

#pragma declare \
    device_resident(adaptive_weights);
#pragma declare \
    device_resident(steering_vectors);

adaptive_weights = XMALLOC(sizeof(complex)*
    num_adaptive_weight_elements);
steering_vectors = XMALLOC(sizeof(complex)*
    num_steering_vector_elements);

...
#pragma acc data copyin(adaptive_weights, \
    steering_vectors), copyout(accums)
#pragma acc kernels loop
for (sv = 0; sv < N_STEERING; ++sv)
{
  for (block = 0; block < N_DOP*N_BLOCKS; ++block)
  {
    accum.re = accum.im = 0.0f;
    cblas_cdotc_sub(TDOF*N_CHAN,
       (float*)adaptive_weights[block][sv],
       1,
       (float*)steering_vectors[sv],
       1,
       (float*)&accums[block][sv]);
  }
}
...
```

# Hardware in the Loop Simulation

- ## Accelerator Simulation
  - **Timing:** Synopsis DesignWare
  - **Power:** DRAMSim2, Cacti, Cacti-3DD, McPAT
- ## Full System Evaluation
  - Run code on real system (Haswell, Xeon Phi) or in simulator (SimpleScalar,…)
  - Normal DRAM access for CPU, but trap accelerator command memory space, invoke simulator

```
#include “fftw.h”
#include “papi_util.h”
...
{
  fftw_complex *in, *out;
  fftw_plan p;
  ...
  in = (fftw_complex*)
    fftw_malloc(sizeof(fftw_complex) * N * N);

  out = (fftw_complex*)
    fftw_malloc(sizeof(fftw_complex) * N * N);

  p = fftw_plan_dft_2d(N, N, in, out,
    FFTW_FORWARD, FFTW_MEASURE);

  ///// Measurement Code /////
  papi_begin();
  ////////////////////////////

  fftw_execute(p);

  ///// Measurement Code /////
  papi_end();
  ////////////////////////////

  fftw_destroy_plan(p);
  fftw_free(in); fftw_free(out);
}
```

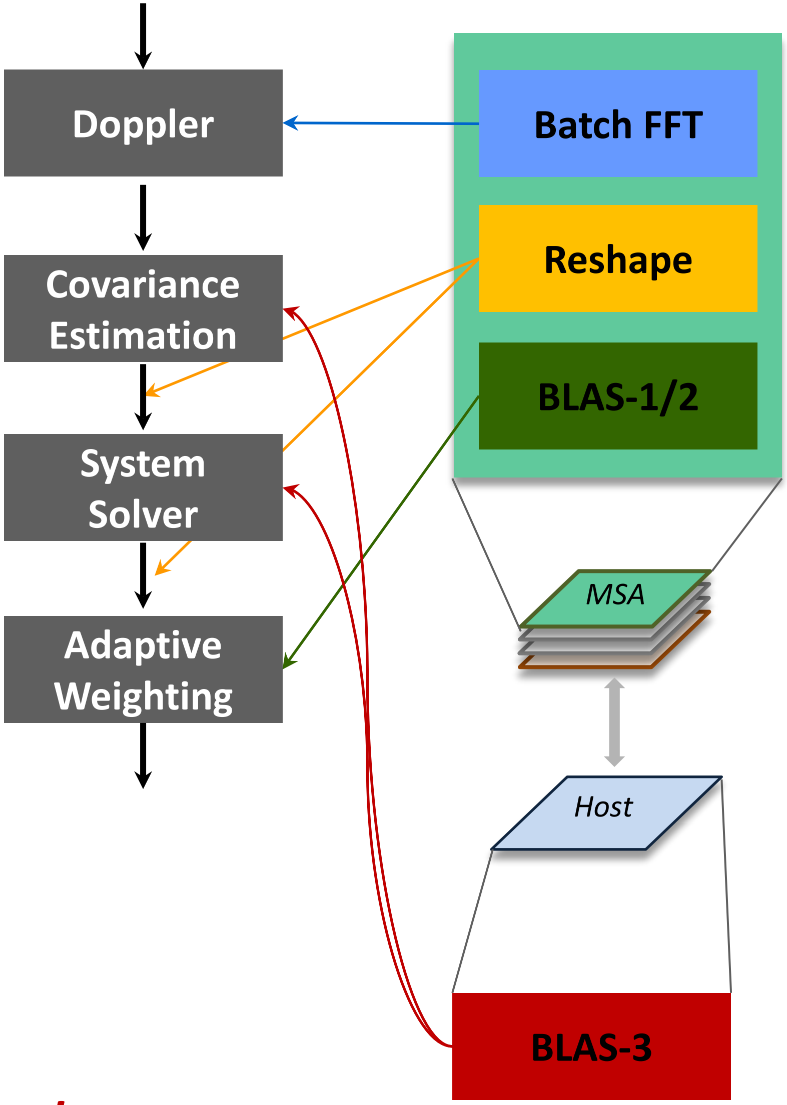



3D-stacked MSA System Design Space

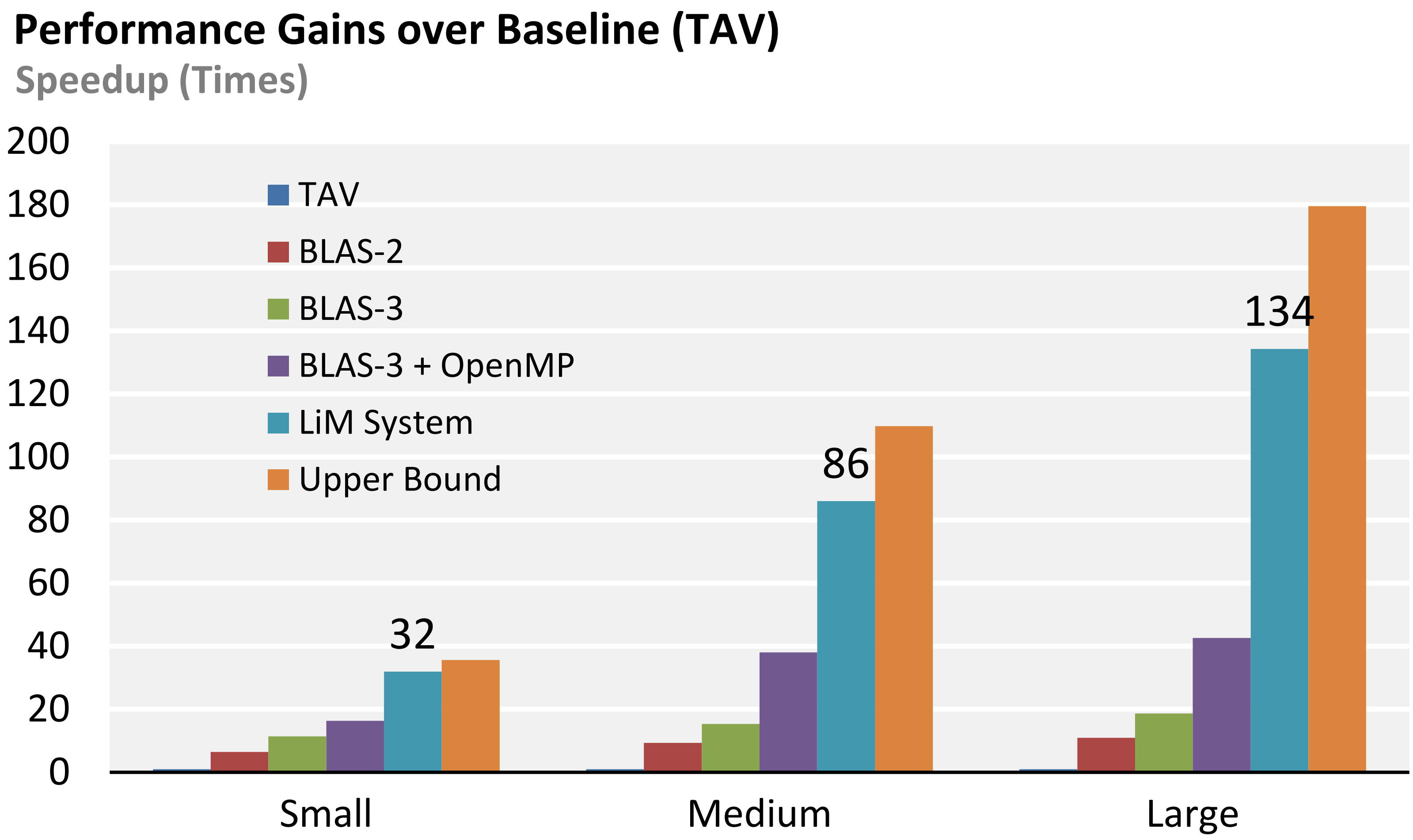
# Accelerating the TAV STAP Benchmark

- **STAP structure: 4 phases**
  - **Doppler:** batch FFT
  - **Covariance estimation:** outer product
  - **System solver:** Cholesky or QR
  - **Adaptive weighting:** inner product
- **STAP memory-side acceleration**
  - **On CPU:** BLAS-3/Cholesky
  - **On accelerator:** FFT, BLAS-1, BLAS-2
- **Software interface**
  - Drop-in replacement for Intel MKL: FFTW, BLAS-1/2/3
  - OpenACC pragmas for batch BLAS-1/2

***No DMA/copy-in/copy-out necessary for accelerator***



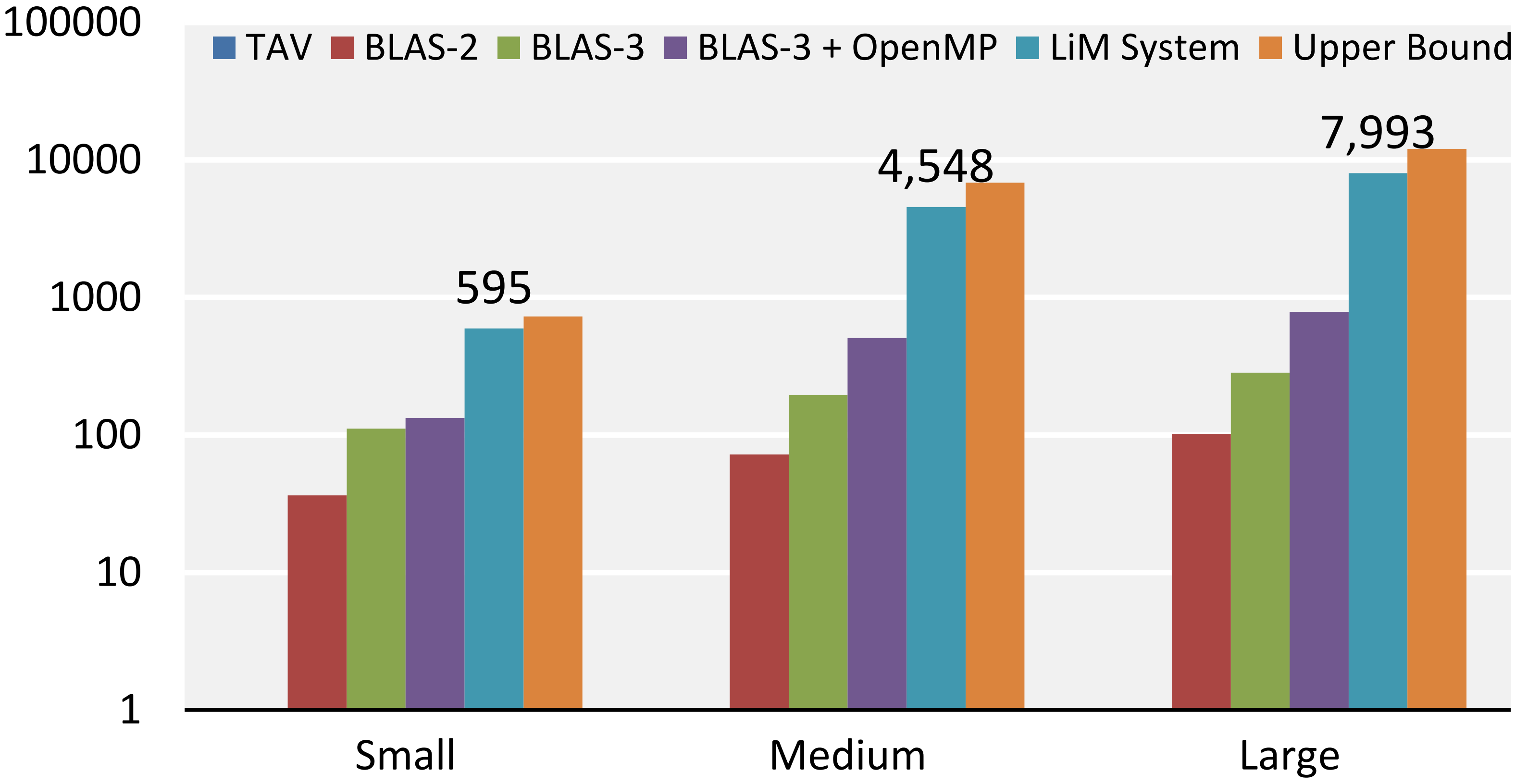
# STAP Performance Results



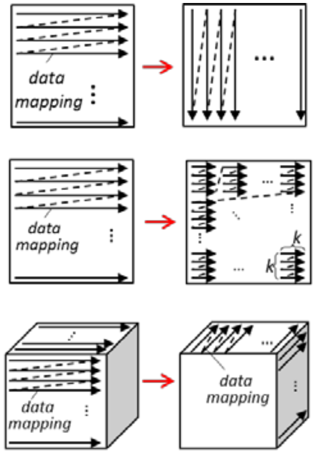
# STAP Power Efficiency Results

**GFLOPS/W Gain over Baseline (TAV)**

EDP Reduction (Times)

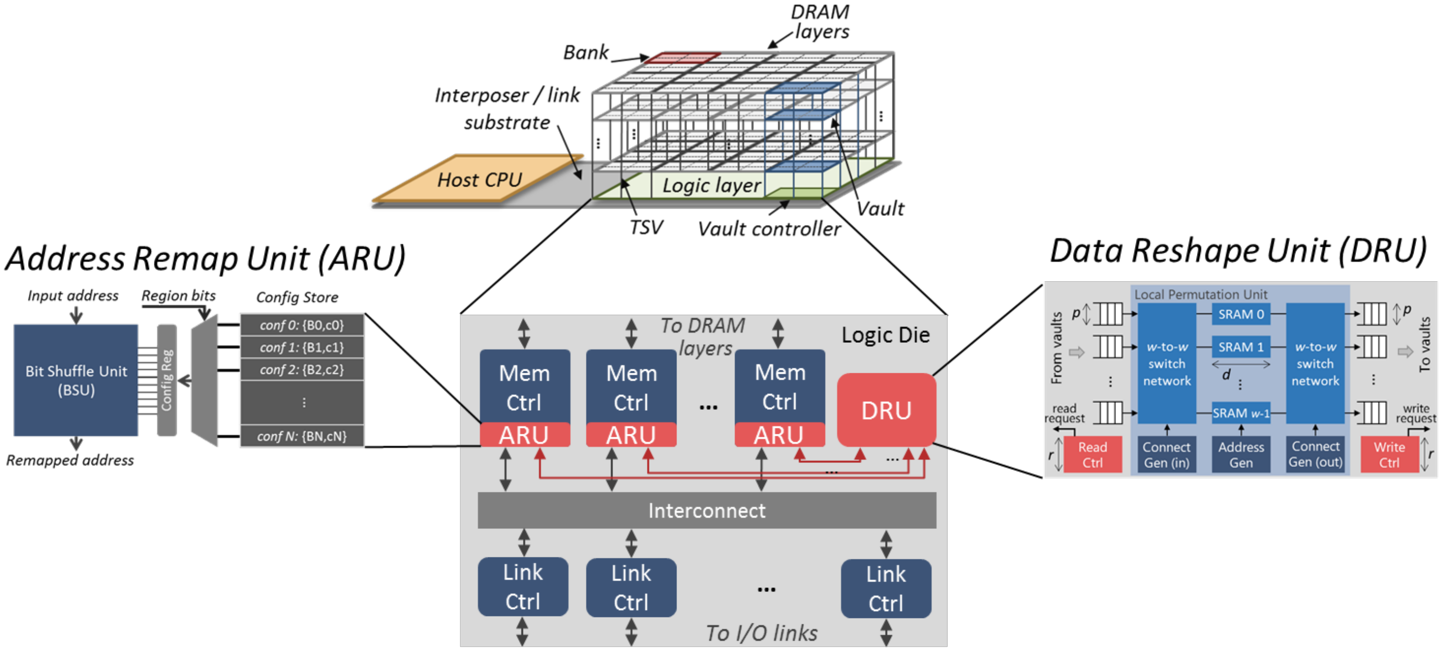


# Memory Layout Accelerator



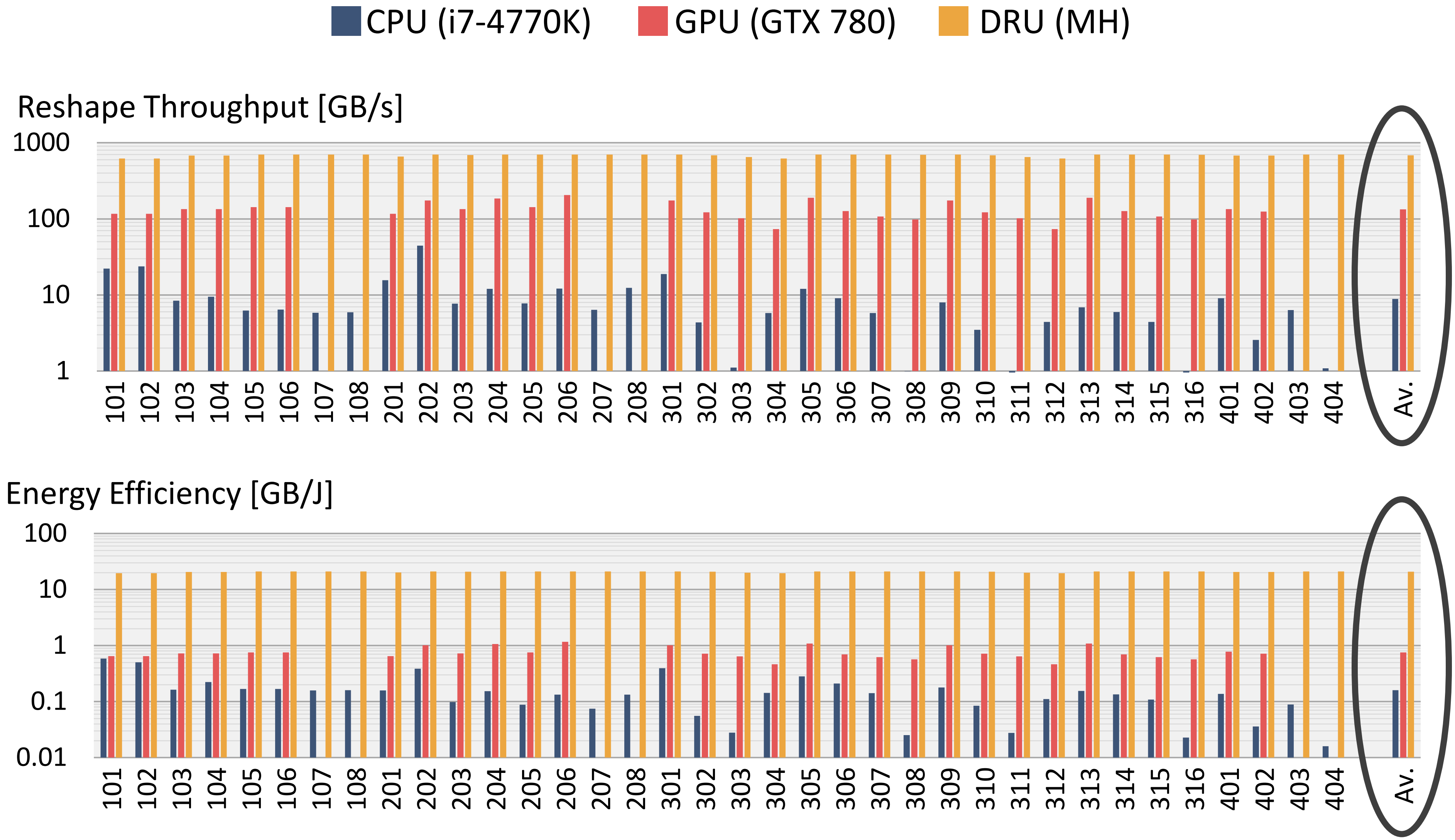
## Intel MKL (Math Kernel Library) functions:

```
void  mkl_simatcopy (const char ordering, char trans, size_t rows,size_t cols,
                     const float alpha, float * AB, size_t lda, size_t ldb);
void  mkl_somatcopy (char ordering, char trans, size_t rows, size_t cols,
                const float alpha, float * A, size_t lda, float * B, size_t ldb);
void  pstrmr2d (char *uplo , char *diag , MKL_INT *m , MKL_INT *n , float *a,
                , …, MKL_INT *jb , MKL_INT *descb , MKL_INT*ictxt ); // ScaLAPACK
vsPackI ( n, a, inca, y );
vsUnpackI ( n, a, y, incy );
void  cblas_scopy (const MKL_INT n, const float *x, const MKL_INT incx, …);
void  cblas_sswap (const MKL_INT n, float *x, const MKL_INT incx, float *y,…);
pslared2d (n, ia, ja, desc, byrow, byall, work, lwork)
pslaswp (direc, rowcol, n, a, ia, ja, desca, k1, k2, ipiv)
...
```



B. Akin, F. Franchetti, J. C. Hoe: **Data Reorganization in Memory Using 3D-stacked DRAM,** 42nd International Symposium on Computer Architecture (ISCA), 2015.

# Accelerating MKL Reorganization Routines

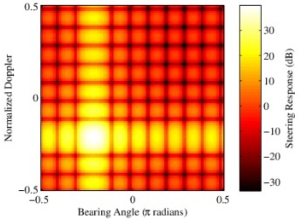


# Outline

- Example: STAP
- Cross-call/cross library optimization with Spiral
- Library-based hardware acceleration
- **Summary**

# Summary

- **Target: dense scientific and HPEC kernels**
  math heavy, regular, good library coverage

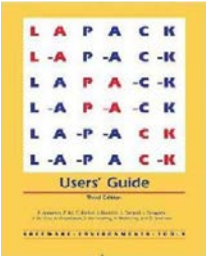


- **HPC Library + OpenMP + C as DSL**
  View program as specification

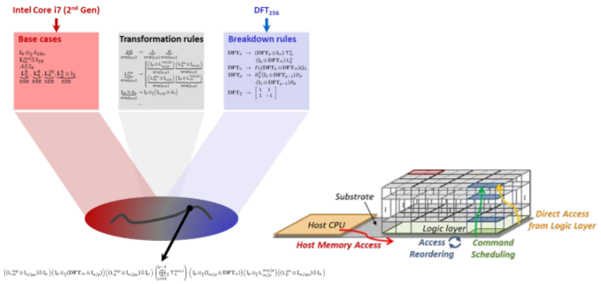
OpenMP

FFTW

OpenACC



- **Enormous efficiency gains are possible**
  Novel accelerators + Spiral optimization


Intel Core i7 (2nd Gen)
DFT256
Base cases
Transformation rules
Breakdown rules
Substrate
Host CPU
Logic layer
Host Memory Access
Access Reordering
Command Scheduling
Direct Access from Logic Layer