FFTX Backend: SPIRAL

Presenter: Franz Franchetti

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

SpectralPACK
Convolution
Correlation
Upsampling
Poisson solver
…

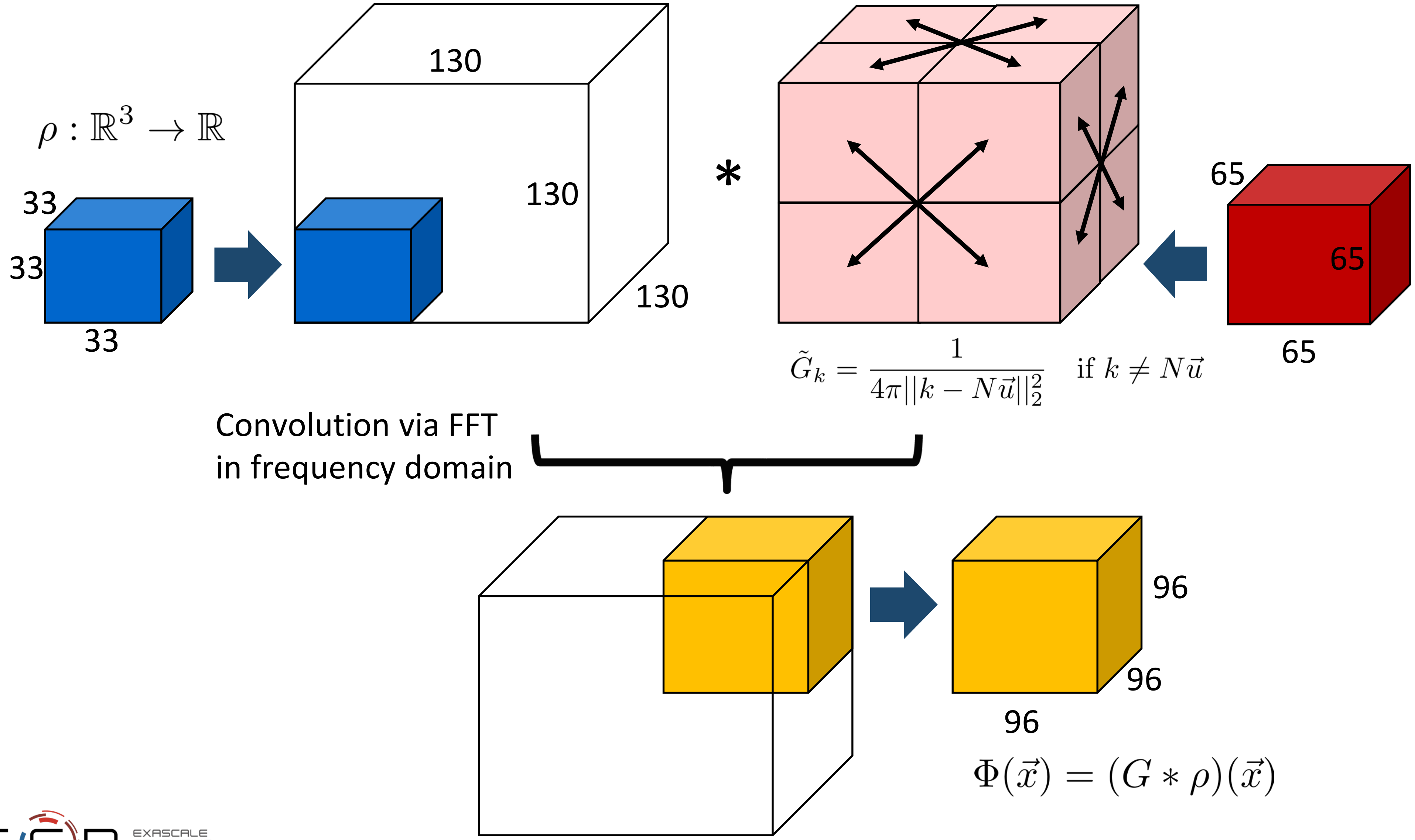
FFTX
DFT, RDFT
1D, 2D, 3D,…
batch

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

Algorithm: Hockney Free Space Convolution

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



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

$$\Phi(\vec{x}) = (G * \rho)(\vec{x})$$

Hockney: Convolution + problem specific zero padding and output subset

Defining / Generating / Using an FFTX Transform

Defining the DAG:

```
const int nx=80; const int ny=80; const int nz=80;
box_t<3> domain(point_t<3>({{1,1,1}}), point_t<3>({{nx,ny,nz}}));
array_t<3,std::complex<double>> inputs(domain);
array_t<3,std::complex<double>> outputs(domain);
std::array<array_t<3,std::complex<double>>,2> intermediates {domain};
setInputs(inputs); setOutputs(outputs); openScalarDAG();
MDDFT(domain.extents(), 1, intermediates[0], inputs);
RCDiag(domain.extents(), 1, intermediates[1], intermediates[0]);
IMDDFT(domain.extents(), 1, outputs, intermediates[1]);
closeScalarDAG(intermediates, "mdconv");
```

Generated code is accessed by an application through a C++ class API.

```
fftx::point_t<3> sz3d = {nx,ny,nz};
fftx::box_t<3> domain(fftx::point_t<3>({{1,1,1}}), sz3d));

fftx::array_t<3,std::complex<double>> input(domain);
fftx::array_t<3,std::complex<double>> output(domain);
fftx::array_t<3,std::complex<double>> filter(domain);

fftx::mdconv<3> mdconv3d(sz3d); // library API.
mdconv3d.transform(input,output,filter); // Call convolution.
```

FFTX / Spiral workflow generates $O(10^3)$ lines per size of C++/HIP/cuda/SYCL

```
            t1629 = (t1609 + t1614);
            t1630 = (s854 + s864);
            t1631 = (s855 - s865);
            t1632 = (s854 - s864);
            t1633 = (s855 + s865);
            Y[(a2899 + 256)] = (t1626 + t1630);
            Y[(a2899 + 257)] = (t1627 + t1631);
            Y[(a2899 + 1280)] = (t1626 - t1630);
            Y[(a2899 + 1281)] = (t1627 - t1631);
            Y[(a2899 + 768)] = (t1628 + t1633);
            Y[(a2899 + 769)] = (t1629 - t1632);
            Y[(a2899 + 1792)] = (t1628 - t1633);
            Y[(a2899 + 1793)] = (t1629 + t1632);
        }
    }
}

void mddft3d_1024x1024x1024_mpi(double  *Y, double  *X) {
    dim3 b458(1, 1, 64), b459(1, 1, 1), b460(1, 1, 64), b461(1, 1, 1),
1),
    g4(1, 1, 1), g5(64, 256, 1);
    ker_code0<<<g1, b458>>>(X);
    fftx_mpi_rcperm(Q2, Q1, 1073741824, 2, 3, 1024, 1024, 1024);
    ker_code2<<<g3, b460>>>();
    fftx_mpi_rcperm(Q2, Q1, 1073741824, 1, 3, 1024, 1024, 1024);
    ker_code4<<<g5, b462>>>(Y);
```

This is the kind of interface apps users want!
Hides platform, hides codegen, looks like a sequence of library calls

EXASCALE COMPUTING PROJECT

FFTX C++ Code: Defining a FFTX Transform

```
#include "fftx3.hpp"
...
int main(int argc, char* argv[]
{
  tracing=true;
  const int nx=80; const int ny
  box_t<3> domain(point_t<3>({{

  array_t<3,std::complex<double
  array_t<3,std::complex<double
  std::array<array_t<3,std::com

  setInputs(inputs);
  setOutputs(outputs);
  openScalarDAG();
  MDDFT(domain.extents(), 1, intermediates[0], inputs);
  RCDiag(domain.extents(), 1, intermediates[1], intermediates[0]);
  IMDDFT(domain.extents(), 1, outputs, intermediates[1]);

  closeScalarDAG(intermediates, "mdconv");
```

This is a specification dressed as a program

- Needs to be clean and concise
- No code level optimizations and tricks
- Don't think "performance" but "correctness"
- *For production code and software engineering*

Defines the semantics of a new library call/object

C/C++ FFTX Program Trace: MPI+GPU

```
Load(fftx);
Load(mpi);
Import(fftx, mpi);

pg := [32,32];
d := 3;
N := Replicate(d, 1024
procGrid := MPIProcGri

name := "mdconv1024x10
conf := LocalConfig.mp

Nlocal := N{[1]}::List
localBrick := TArrayND
dataLayout := TPtrGlob
Xglobal := tcast(dataL
Yglobal := tcast(dataL

T1global := var("T1", dataLayout); T2global := var("T2", dataLayout);
symvar := var("symvar", dataLayout);
t := TFCall(TDecl(TDAG([
  TDAGNode(TRC(MDDFT(N, -1)), T1global, Xglobal),
  TDAGNode(RCDiag(FDataOfs(tcast(TPtr(TReal), symvar), 2*Product(N), 0)), T2global, T1global),
  TDAGNode(TRC(MDDFT(N, 1)), Yglobal, T2global)
]), [ T1global, T2global ]), rec(fname := name, params := [ symvar ]));

opts := conf.getOpts(t);
tt := opts.tagIt(t);
c := opts.fftxGen(tt);
opts.prettyPrint(c);
PrintTo(name::".cu", opts.prettyPrint(c));
```

The whole convolution kernel is captured

- DAG with all dependencies
- User-defined call-backs
- Captures prunining, zero-padding and symmetries
- *Lifts sequence of C/C++ library calls to a specification*

Operator DAG abstraction:
Major new concept and capability in SPIRAL due to FFTX

EXASCALE COMPUTING PROJECT

Advanced Operator DAG: WarpX Convolution

```
TFCall(TDecl(TDAG([
  TDAGNode(TResample([n, n, n], [np, np, n], [0.0, 0.0, -0.5]), nth(boxBig0, 0), nth(X, 0)),
  TDAGNode(TResample([n, n, n], [np, n, np], [0.0, -0.5, 0.0]), nth(boxBig0, 1), nth(X, 1)),
  TDAGNode(TResample([n, n, n], [n, np, np], [-0.5, 0.0, 0.0]), nth(boxBig0, 2), nth(X, 2)),
  TDAGNode(TResample([n, n, n], [n, n, np], [-0.5, -0.5, 0.0]), nth(boxBig0, 3), nth(X, 3)),
  TDAGNode(TResample([n, n, n], [n, np, n], [-0.5, 0.0, -0.5]), nth(boxBig0, 4), nth(X, 4)),
  TDAGNode(TResample([n, n, n], [np, n, n], [0.0, -0.5, -0.5]), nth(boxBig0, 5), nth(X, 5)),
  TDAGNode(TResample([n, n, n], [np, np, n], [0.0, 0.0, -0.5]), nth(boxBig0, 6), nth(X, 6)),
  TDAGNode(TResample([n, n, n], [np, n, np], [0.0, -0.5, 0.0]), nth(boxBig0, 7), nth(X, 7)),
  TDAGNode(TResample([n, n, n], [n, np, np], [-0.5, 0.0, 0.0]), nth(boxBig0, 8), nth(X, 8)),
  TDAGNode(TResample([n, n, n], [np, np, np], [0.0, 0.0, 0.0]), nth(boxBig0, 9), nth(X, 9)),
  TDAGNode(TResample([n, n, n], [np, np, np], [0.0, 0.0, 0.0]), nth(boxBig0, 10), nth(X, 10)),
  TDAGNode(TTensorI(MDPRDFT([n, n, n], -1), inFields, APar, APar), boxBig1, boxBig0),
  TDAGNode(TRC(TMap(rmat, [iz, iy, ix], AVec, AVec)), boxBig2, boxBig1),
  TDAGNode(TTensorI(IMDPRDFT([n, n, n], 1), outFields, APar, APar), boxBig3, boxBig2),
  TDAGNode(TResample([np, np, n], [n, n, n], [0.0, 0.0, 0.5]), nth(Y, 0), nth(boxBig3, 0)),
  TDAGNode(TResample([np, n, np], [n, n, n], [0.0, 0.5, 0.0]), nth(Y, 1), nth(boxBig3, 1)),
  TDAGNode(TResample([n, np, np], [n, n, n], [0.5, 0.0, 0.0]), nth(Y, 2), nth(boxBig3, 2)),
  TDAGNode(TResample([n, n, np], [n, n, n], [0.5, 0.5, 0.0]), nth(Y, 3), nth(boxBig3, 3)),
  TDAGNode(TResample([n, np, n], [n, n, n], [0.5, 0.0, 0.5]), nth(Y, 4), nth(boxBig3, 4)),
  TDAGNode(TResample([np, n, n], [n, n, n], [0.0, 0.5, 0.5]), nth(Y, 5), nth(boxBig3, 5)) ]),
  [boxBig0, boxBig1, boxBig2, boxBig3]),
  rec(XType := TPtr(TPtr(TReal)), YType := TPtr(TPtr(TReal)),
      fname := name, params := [symvar ])
```

`TResample` = *½ point resampling: defined in time domain but fused with frequency kernel* `rmat`

Advanced DAG: WarpX Pointwise Kernel

```
ix := Ind(xdim), iy := Ind(ydim), iz := Ind(zdim), ii := lin_idx(iz, iy, ix),
    fmkx := nth(nth(symvar, 0), ix), fmky := nth(nth(symvar, 1), iy),
    fmkz := nth(nth(symvar, 2), iz), fcv := nth(nth(symvar, 3), ii),
    fsckv := nth(nth(symvar, 4), ii), fx1v := nth(nth(symvar, 5), ii),
    fx2v := nth(nth(symvar, 6), ii), fx3v := nth(nth(symvar, 7), ii),

rmat := TSparseMat([outFields,inFields], [
   [0, [0, fcv / n^3], [4, cxpack(0, -fmkz * c2 * fsckv / n^3)],
       [5, cxpack(0, fmky * c2 * fsckv / n^3)], [6, -invep0 * fsckv / n^3],
       [9, cxpack(0,   fmkx * fx3v / n^3)], [10, cxpack(0, -fmkx * fx2v / n^3)]],
   [1, [1, fcv / n^3], [3, cxpack(0, fmkz * c2 * fsckv / n^3)],
       [5, cxpack(0, -fmkx * c2 * fsckv / n^3)], [7, -invep0 * fsckv / n^3],
       [9, cxpack(0,   fmky * fx3v / n^3)], [10, cxpack(0, -fmky * fx2v / n^3)]],
   [2, [2, fcv / n^3], [3, cxpack(0, -fmky * c2 * fsckv / n^3)],
       [4,  cxpack(0, fmkx * c2 * fsckv / n^3)], [8, -invep0 * fsckv / n^3],
       [9, cxpack(0,   fmkz * fx3v / n^3)], [10, cxpack(0, -fmkz * fx2v / n^3)]],
   [3, [1, cxpack(0, fmkz * fsckv / n^3)], [2, cxpack(0, -fmky * fsckv / n^3)],
       [3, fcv / n^3], [7, cxpack(0, -fmkz * fx1v / n^3)],
       [8, cxpack(0, fmky * fx1v / n^3)]],
   [4, [0, cxpack(0, -fmkz * fsckv / n^3)], [2, cxpack(0, fmkx * fsckv / n^3)],
       [4, fcv / n^3], [6, cxpack(0, fmkz * fx1v / n^3)],
       [8, cxpack(0, -fmkx * fx1v / n^3)]],
   [5, [0, cxpack(0, fmky * fsckv / n^3)], [1, cxpack(0, -fmkx * fsckv / n^3)],
       [5, fcv / n^3], [6, cxpack(0, -fmky * fx1v / n^3)],
       [7, cxpack(0, fmkx * fx1v / n^3)]]  ])
```

rmat = complicated location-dependent convolution kernel in frequency domain, computed from multiple seed tables

From FFTX DAG to Generated Single Size Code

$$\underbrace{\mathrm{iDFT}_8 \circ D_8 \circ \mathrm{DFT}_8}_{\mathrm{SIMT(CUDA)}}$$

Lifting

CUDA SIMT ← $\underbrace{\mathrm{MDConv}_8^{D_8}}_{\mathrm{SIMT(CUDA)}}$ → **MDRConv$_8$**

Base cases

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

$\underbrace{\mathrm{L}^4_2}_{\mathrm{CUDA}}$

$\underbrace{\mathrm{T}^{mn}_n}_{\mathrm{SIMT}}$

Transformation rules

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

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

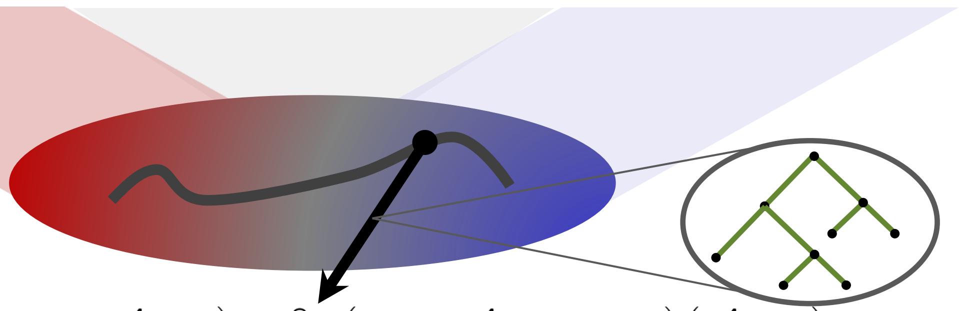
$$A^{m\times m} \otimes \mathrm{I}_n \to (A^{m\times m} \otimes \mathrm{I}_{n/\nu}) \otimes \mathrm{I}_\nu$$

Breakdown rules

$$\mathbf{MDConv}^{D_n}_n \to \mathbf{iDFT}_n \circ D_n \circ \mathbf{DFT}_n$$

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

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



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

DAG specification

↓ Opportunistic pattern matching

Lifted 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/CUDA/HIP/SYCL code

↓ Packaging

Library file

Key FFTX contribution: *Lifiting*, encodes reusable algorithmic patterns (towards SpectralPACK)

Performance Engineering for SYCL, CUDA, and HIP

Transform
user specified

$$\mathrm{DFT}_8$$

Fast algorithm
in SPL
many choices

$$(\mathrm{DFT}_2 \otimes I_4)\, T^8_4\, \Big(I_2 \otimes \big((\mathrm{DFT}_2 \otimes I_2) \cdot T^4_2\, (I_2 \otimes \mathrm{DFT}_2)\, L^4_2\big)\Big)\, L^8_2$$

∑-SPL:

$$\sum \left(S_j\, \mathrm{DFT}_2\, G_j\right) \sum \left(\sum \left(S_{k,l}\, \mathrm{diag}\left(\mathrm{t}_{k,l}\right) \mathrm{DFT}_2\, G_l\right) \sum \left(S_m\, \mathrm{diag}\left(\mathrm{t}_m\right) \mathrm{DFT}_2\, G_{k,m}\right)\right)$$

C Code:

```c
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;
}
```

Optimization at all abstraction levels

parallelization
vectorization
SIMTification

loop and locality optimizations

constant folding
scheduling
ISA specific optimizations
……

MDDFT Pease Algorithm

$$\mathrm{DFT}_{n\times n\times n} \rightarrow \prod_{i=0}^{2} (\mathrm{DFT}_n \otimes \mathrm{I}_{n^2})\, \mathrm{L}^{n^3}_n$$

SVCT for SIMT transformation

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

Loop flattening and regularization

$$\sum_{i=0}^{m-1} \sum_{j=0}^{n-1} A_{i,j} \rightarrow \sum_{k=0}^{mn-1} A_{\lfloor k/n \rfloor,\, k \bmod n}$$

```
simt_loop(i, [0..11], 16, op)
    ->
simt_loop(i, [0..15], 16,
   if(i <= 11, op, skip())
```

ISA specific conf code

```
cudaDeviceSetLimit(cudaLimitMallocHeapSize,
    1073741824);
cudaFuncSetCacheConfig(ker_exaefl_kernel10,
    cudaFuncCachePreferL1);
cudaFuncSetCacheConfig(ker_exaefl_kernel11,
    cudaFuncCachePreferL1);
cudaMalloc(((void  * *) &(hp1)),
    (sizeof(double )*270336));
cudaMemcpyToSymbol(P1, &(hp1),
    sizeof(double  *));
```

How Does SPIRAL JIT Actually Work?

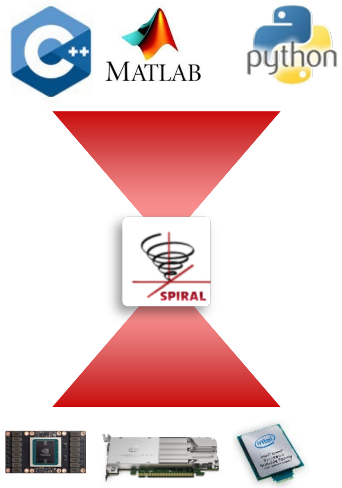
```
// CUDA kernel
__global__ void ker_mdprdft3d_4x4x42(double  *Y) {
    double s100, s101, s102, s97, s98, s99;
    int a191, a192;
    a191 = ((24*blockIdx.y) + (6*blockIdx.x));
    s97 = P2[a191];
    s98 = P2[(a191 + 4)];
    s99 = (s97 + s98);
    s100 = (s97 - s98);
    s101 = (2.0*P2[(a191 + 2)]);
    s102 = (2.0*P2[(a191 + 3)]);
    a192 = ((16*blockIdx.y) + (4*blockIdx.x));
    Y[a192] = (s99 + s101);
    Y[(a192 + 2)] = (s99 - s101);
    Y[(a192 + 1)] = (s100 + s102);
    Y[(a192 + 3)] = (s100 - s102);
}

// main transform function, calls kernels
void mdprdft3d_4x4x4(double  *Y, double  *X) {
    dim3 b37(1, 1, 1), b38(1, 1, 1), b39(1, 1, 1),
        g1(4, 3, 1), g2(4, 3, 1), g3(4, 4, 1);
    ker_mdprdft3d_4x4x40<<<g1, b37>>>(X);
    ker_mdprdft3d_4x4x41<<<g2, b38>>>();
    ker_mdprdft3d_4x4x42<<<g3, b39>>>(Y);
}
```

```
// kernels as strings for the CUDA JIT
kernels[2] = { "__global__ void ker_mdprdft3d_4x4x42(double  *Y) {\n"
"    double s100, s101, s102, s97, s98, s99; \n"
"    int a191, a192; \n"
"    a191 = ((24*blockIdx.y) + (6*blockIdx.x)); \n"
"    s97 = P2[a191]; \n"
"    s98 = P2[(a191 + 4)]; \n"
"    s99 = (s97 + s98); \n"
"    s100 = (s97 - s98); \n"
"    s101 = (2.0*P2[(a191 + 2)]); \n"
"    s102 = (2.0*P2[(a191 + 3)]); \n"
"    a192 = ((16*blockIdx.y) + (4*blockIdx.x)); \n"
"    Y[a192] = (s99 + s101); \n"
"    Y[(a192 + 2)] = (s99 - s101); \n"
"    Y[(a192 + 1)] = (s100 + s102); \n"
"    Y[(a192 + 3)] = (s100 - s102); \n"
"} \n", 1, DOUBLE_PY};

// main transform function encoded as integer table
transform = {MDPRDFT, 3, {4, 4, 4}, 3,
    { kernels[0], dim3(4, 3, 1), dim3(1, 1, 1), 1, DOUBLE_PX },
    { kernels[1], dim3(4, 3, 1), dim3(1, 1, 1), 0 },
    { kernels[2], dim3(4, 4, 1), dim3(1, 1, 1), 1, DOUBLE_PY }};

void mdprdft3d_4x4x4(double  *Y, double  *X) {
    // one-time JITs kernel, interprets and executes integer table
    execute(transform, MODE_JIT_CACHE);
}
```

FFTX API is general enough: accommodates collection of fixed sizes and general size FFTs.

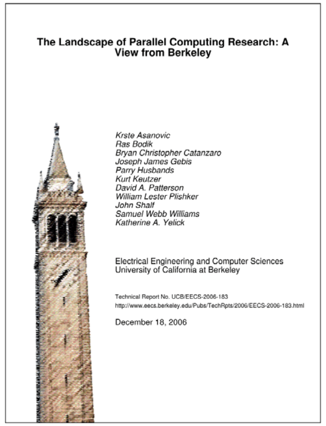
Bigger Ecosystem: ECP, X-Stack, and DARPA

Multi-language, Multi target



Powered by SPIRAL
code generation/synthesis

Cross-motif optimization



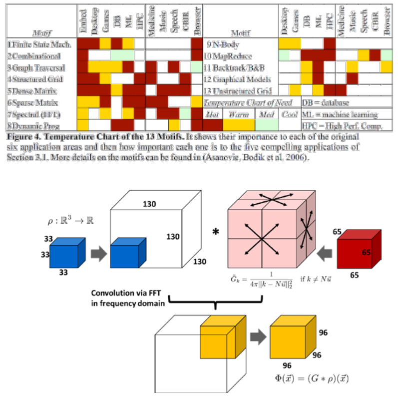
The Landscape of Parallel Computing Research: A View from Berkeley

Krste Asanovic
Ras Bodik
Bryan Christopher Catanzaro
Joseph James Gebis
Parry Husbands
Kurt Keutzer
David A. Patterson
William Lester Plishker
John Shalf
Samuel Webb Williams
Katherine A. Yelick

Electrical Engineering and Computer Sciences
University of California at Berkeley

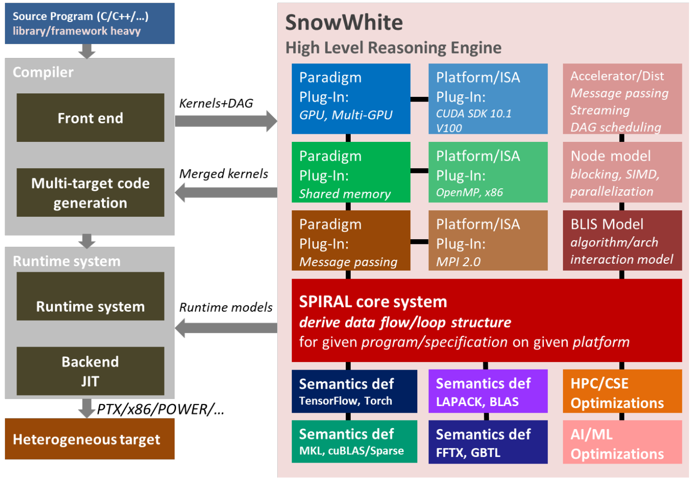
Technical Report No. UCB/EECS-2006-183
http://www.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-183.html

December 18, 2006



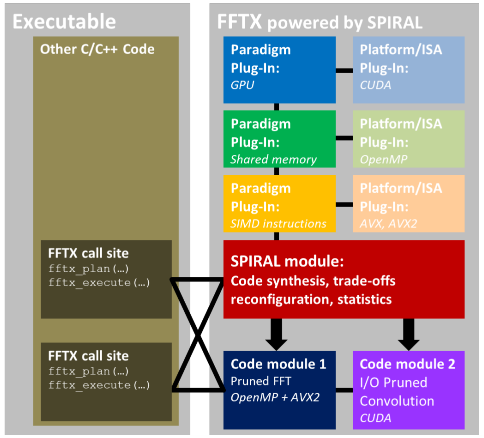
Cross-call, cross-library, cross-motif

SnowWhite: SPIRAL inside compilers



X-Stack Bluestone

LibraryX, powered by SPIRAL



GBTLX NTTX

One instance of a family of active libraries

FFTX (ECP)

Multiple active libraries, one infrastructure

EXASCALE COMPUTING PROJECT