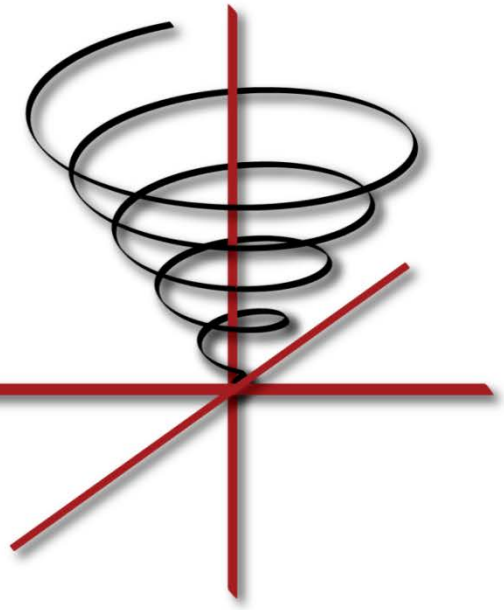
SPIRAL

FFT Library Generation and Autotuning



Franz Franchetti

Carnegie Mellon University

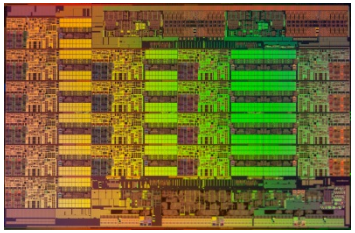
www.spiral.net

In collaboration with the SPIRAL team

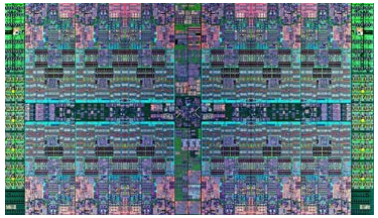
This work was supported by DARPA, ONR, DOE NSF, Intel, Mercury

Today's Computing Landscape

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

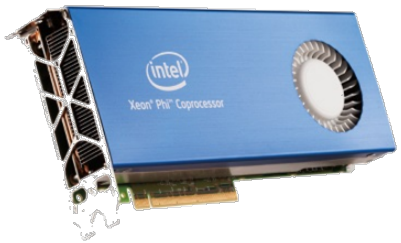


Intel Xeon E5-2699v3
662 Gflop/s, 145 W
18 cores, 2.3 GHz
4-way/8-way AVX2



IBM POWER8
384 Gflop/s, 200 W
12 cores, 4 GHz
2-way/4-way VMX/VSX

Nvidia Tesla M2090
666 Gflop/s, 225 W
512 cores, 1.33 GHz
32-way SIMT



Intel Xeon Phi
1.2 Tflop/s, 300 W
61 cores, 1.24 GHz
8-way/16-way LRBni

Snapdragon 810
10 Gflop/s, 2 W
4 cores, 2.5 GHz
A330 GPU, V50 DSP, NEON

Intel Atom C2750
29 Gflop/s, 20 W
8 cores, 2.4 GHz
2-way/4-way SSSE3

Dell PowerEdge R920
1.34 Tflop/s, 850 W
4x 15 cores, 2.8 GHz
4-way/8-way AVX

BlueGene/Q
10 Pflop/s, 8 MW
48k x 16 cores, 1.6 GHz
4-way QPX

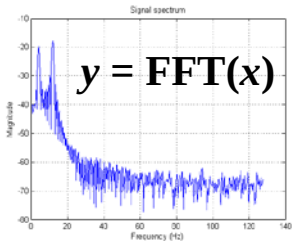
Idea: Go from Mathematics to Software

Given:

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

Wanted:

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



on

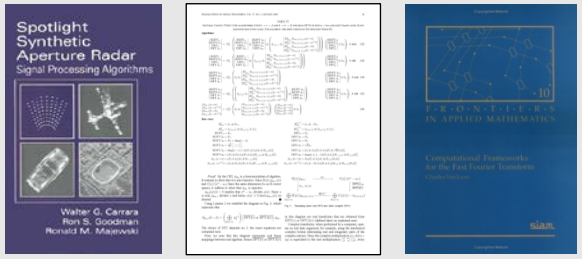
automatic

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

QED.

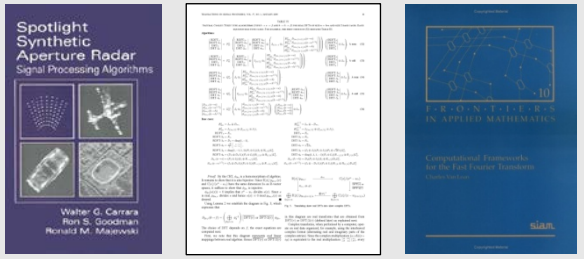
What is Spiral?

Traditionally



High performance library optimized for given platform

Spiral Approach



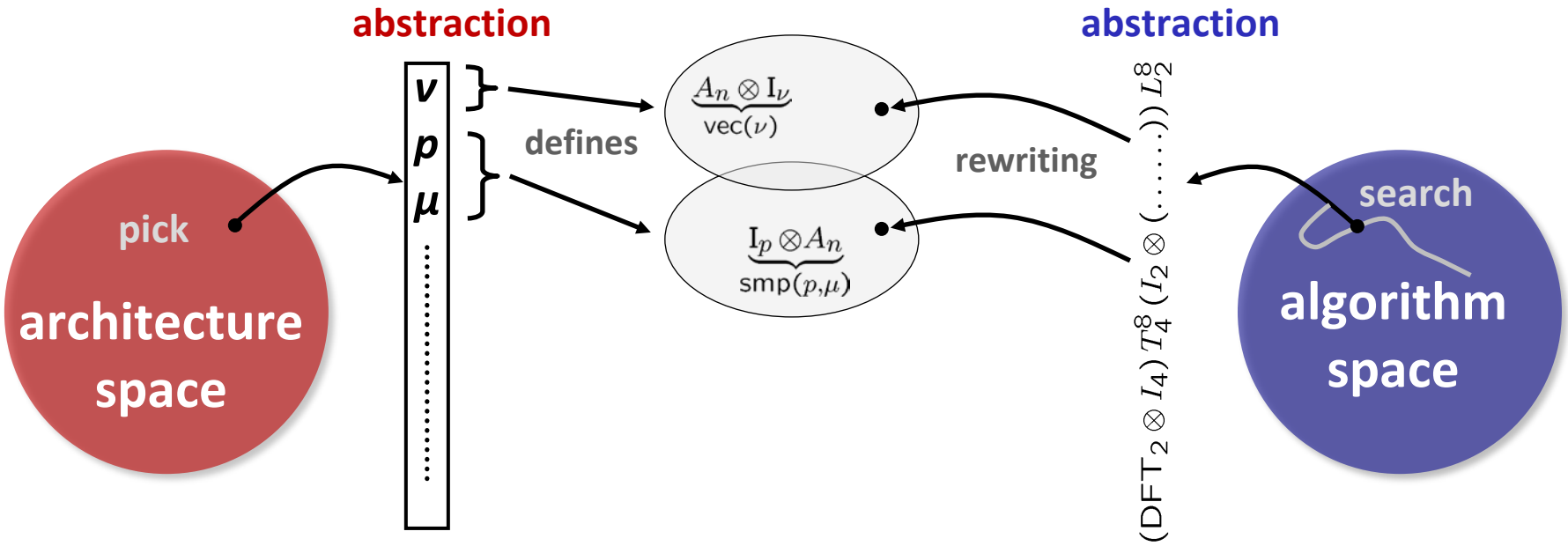
Spiral

High performance library optimized for given platform

Comparable performance

Platform-Aware Formal Program Synthesis

Model: common abstraction
= spaces of matching formulas



Architectural parameter:
Vector length,
#processors, …

optimization

Kernel:
problem size,
algorithm choice

Algorithms: Rules in Domain Specific Language

Linear Transforms

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

Viterbi Decoding

$$
\begin{aligned}
\underbrace{\mathbf{Vit}}_{\mathrm{vec}(v)} &\rightarrow \underbrace{\left(\prod (L \times I) \circ (I \otimes C)\right) \circ Id}_{\mathrm{vec}(v)} \\
&\rightarrow \left(\prod \underbrace{(L \times I) \circ (I \otimes C)}_{\mathrm{vec}(v)}\right) \circ Id \\
\times \quad &\rightarrow \left(\prod (L \otimes I_v \times I) \circ (I \otimes C \otimes I_v) \circ (\vec{L} \times I)\right) \circ Id \\
&\rightarrow \prod (L \otimes I_v \times I) \circ (I \otimes (B \otimes I_v)) \circ (\vec{L} \times I)
\end{aligned}
$$

Matrix-Matrix Multiplication

$$
\begin{aligned}
&\mathrm{MMM}_{1,1,1} \rightarrow (\cdot)_1 \\
&\mathrm{MMM}_{m,n,k} \rightarrow (\otimes)_{m/m_b \times 1} \otimes \mathrm{MMM}_{m_b,n,k} \\
&\mathrm{MMM}_{m,n,k} \rightarrow \mathrm{MMM}_{m,nb,k} \otimes (\otimes)_{1 \times n/nb} \\
&\mathrm{MMM}_{m,n,k} \rightarrow ((\Sigma_{k/k_b} \circ (\cdot)_{k/k_b}) \otimes \mathrm{MMM}_{m,n,k_b}) \circ \\
&\qquad\qquad ((L^{mk/k_b}_{k/k_b} \otimes I_{k_b}) \times I_{kn}) \\
&\mathrm{MMM}_{m,n,k} \rightarrow (L^{mn/n_b}_m \otimes I_{n_b}) \circ \\
&\qquad ((\otimes)_{1 \times n/n_b} \otimes \mathrm{MMM}_{m,n_b,k}) \circ \\
&\qquad\qquad (I_{km} \times (L^{kn/n_b}_{n/n_b} \otimes I_{n_b}))
\end{aligned}
$$

Synthetic Aperture Radar (SAR)

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

Formal Approach for all Types of Parallelism

- **Multithreading** (Multicore)
- **Vector SIMD** (SSE, VMX/Altivec,…)
- **Message Passing** (Clusters, MPP)
- **Streaming/multibuffering** (Cell)
- **Graphics Processors** (GPUs)
- **Gate-level parallelism** (FPGA)
- **HW/SW partitioning** (CPU + FPGA)

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

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

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

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

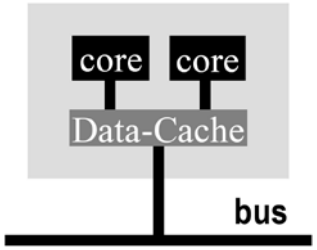
$$\prod_{i=0}^{n-1} A_i, \quad A_n \hat{\otimes} \mathrm{I}_w, \quad P_n \otimes Q_w$$

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

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

Modeling Hardware: Base Cases

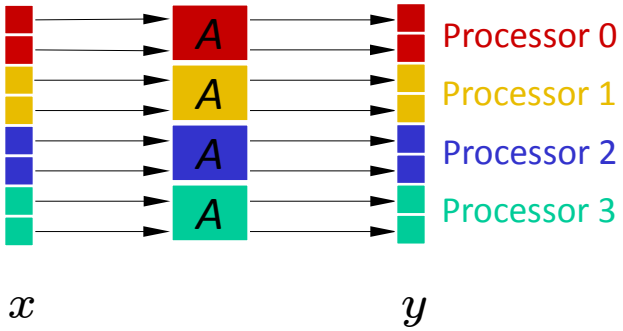
- **Hardware abstraction: shared cache with cache lines**



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

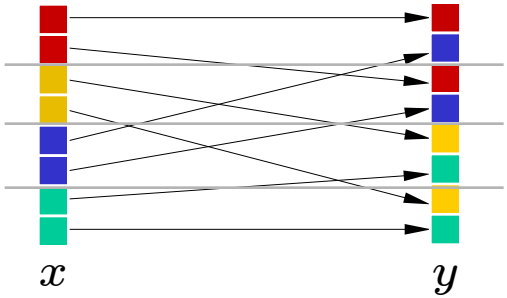
- **Tensor product: embarrassingly parallel operator**

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



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

$$y = \mathrm{L}^{8}_{4}(x)$$



Example Program Transformation Rule Set

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

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

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

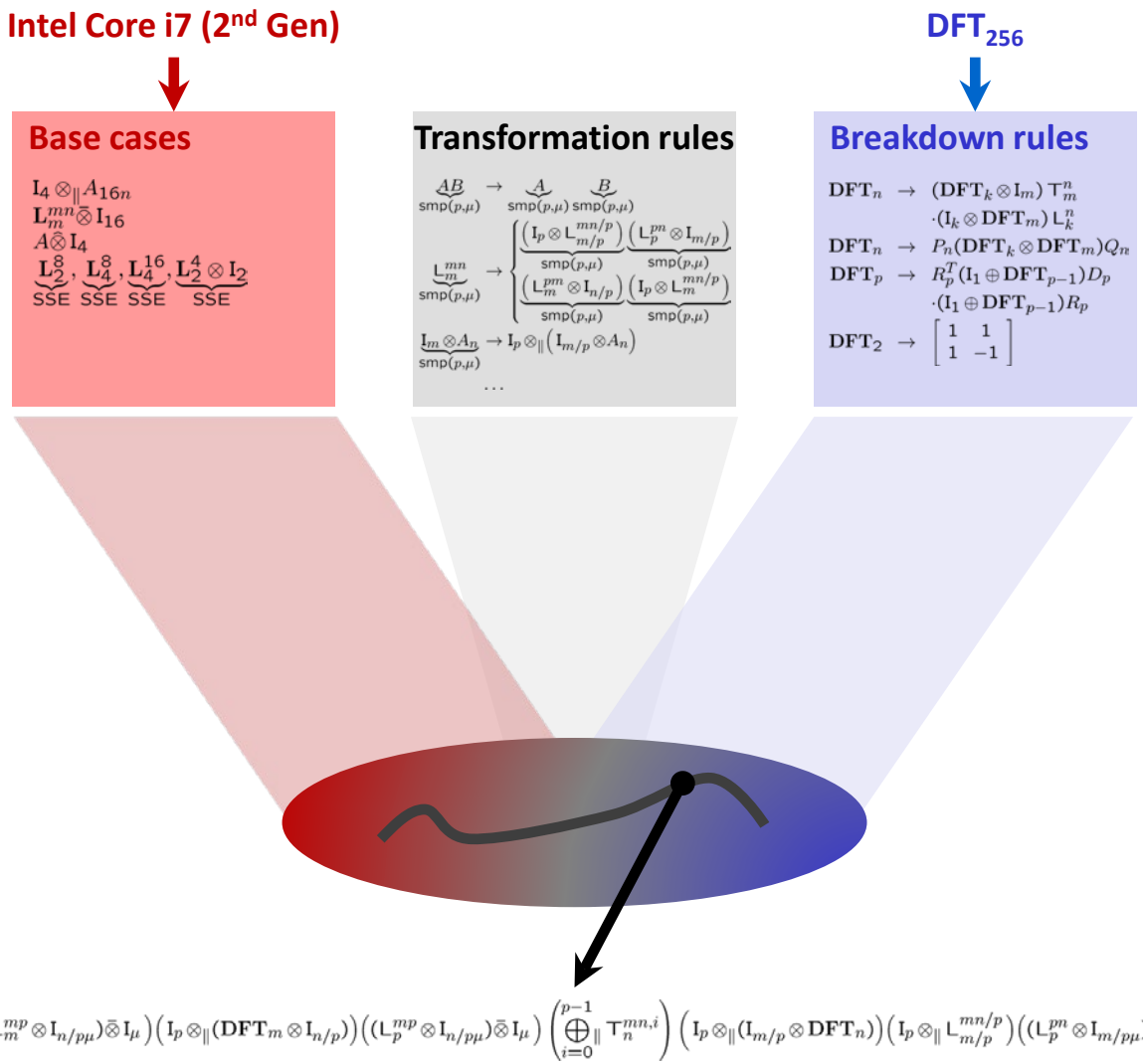
Recursive rules

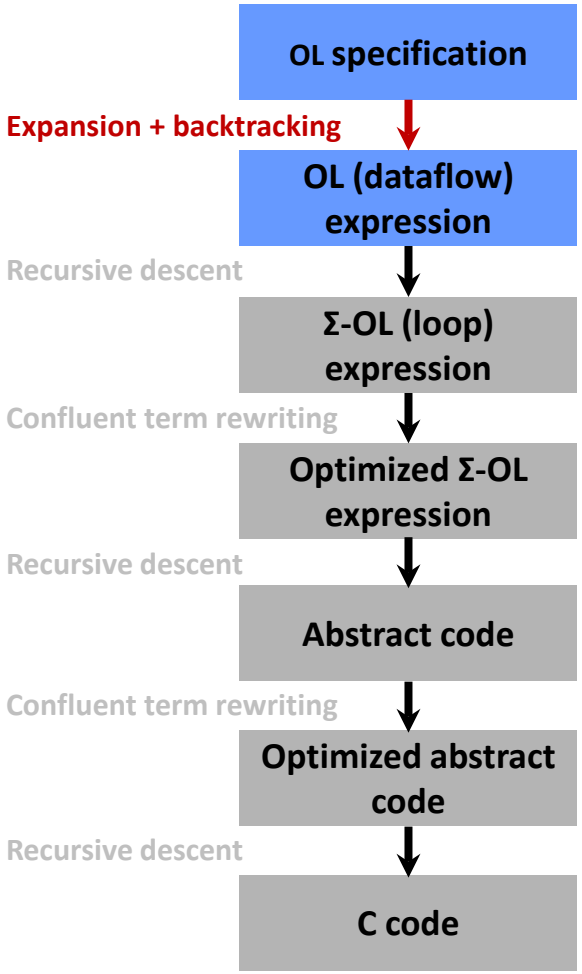
$$\underbrace{\mathrm{I}_m \otimes A_n}_{\mathsf{smp}(p,\mu)} \rightarrow \mathrm{I}_p \otimes_{\|} \left(\mathrm{I}_{m/p} \otimes A_n\right)$$

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

Base case rules

Autotuning in Constraint Solution Space





Translating an OL Expression Into Code

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

Output =

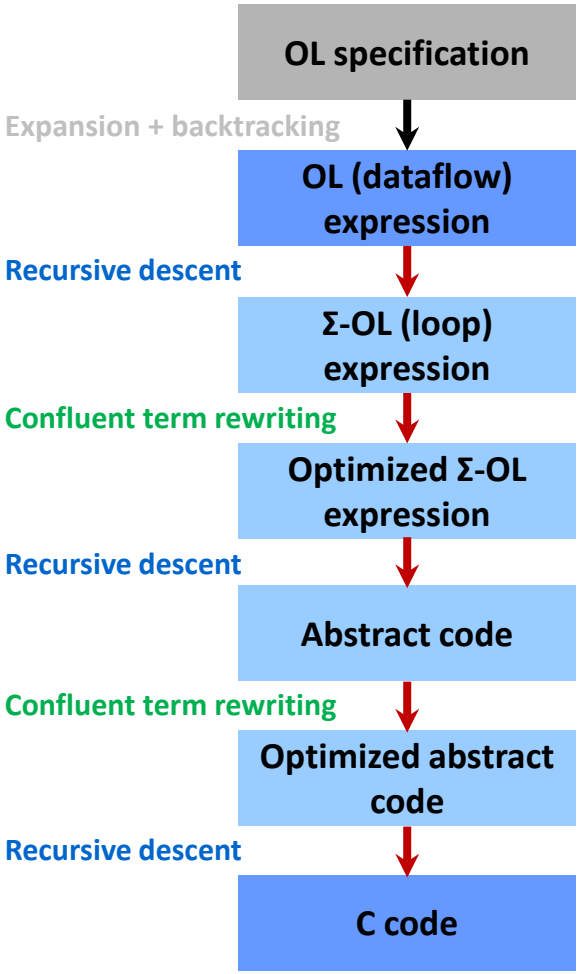
OL Expression: $(\mathrm{DFT}_2 \otimes I_4)\, T^8_4 \left(I_2 \otimes \left((\mathrm{DFT}_2 \otimes I_2)\, T^4_2\, (I_2 \otimes \mathrm{DFT}_2)\, L^4_2\right)\right) L^8_2$

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:

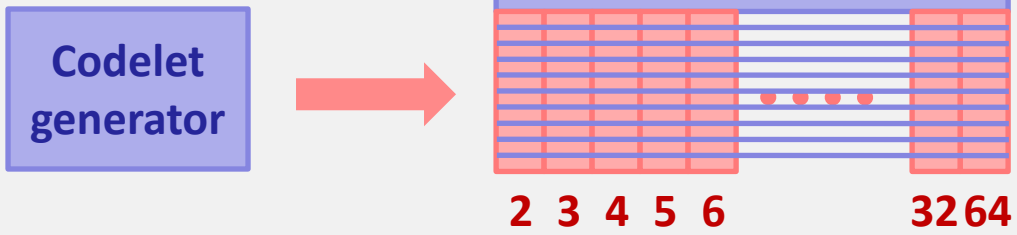
```
void sub(double *y, double *x) {
  double f0, f1, f2, f3, f4, f7, f8, f10, f11;
  f0 = x[0] - x[3];
  f1 = x[0] + x[3];
  f2 = x[1] - x[2];
  f3 = x[1] + x[2];
  f4 = f1 - f3;
  y[0] = f1 + f3;
  y[2] = 0.7071067811865476 * f4;
  f7 = 0.9238795325112867 * f0;
  f8 = 0.3826834323650898 * f2;
  y[1]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



SPIRAL: Meta-Tool to FFTW

blue = hand-written, red = generated

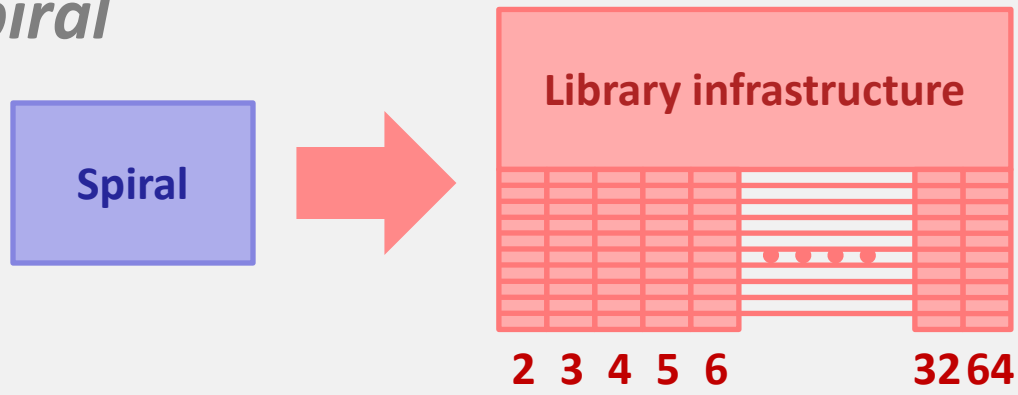
FFTW



Recursive functions needed
Vectorization (partially)
Threading
Adaptation mechanism (plan)

Codelets (10–20 types)

Spiral



Recursive functions needed
Vectorization
Threading
Adaptation mechanism (plan)

Codelets (10–20 types)

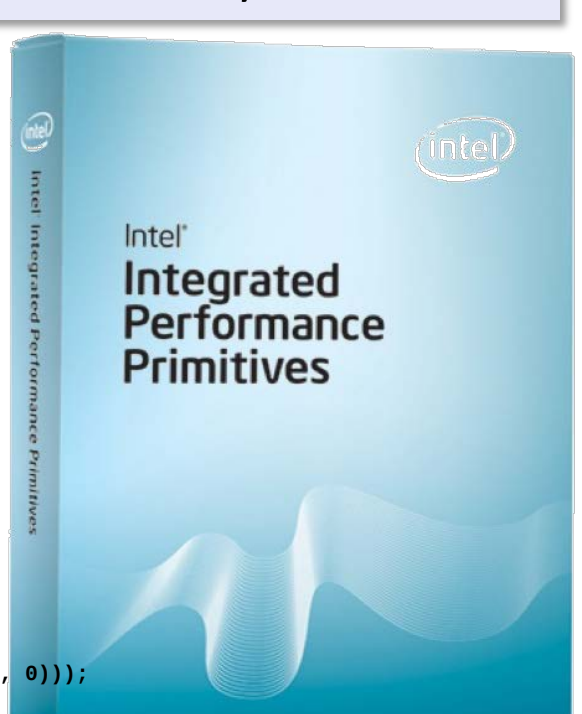
Production-Quality Code

```
...
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

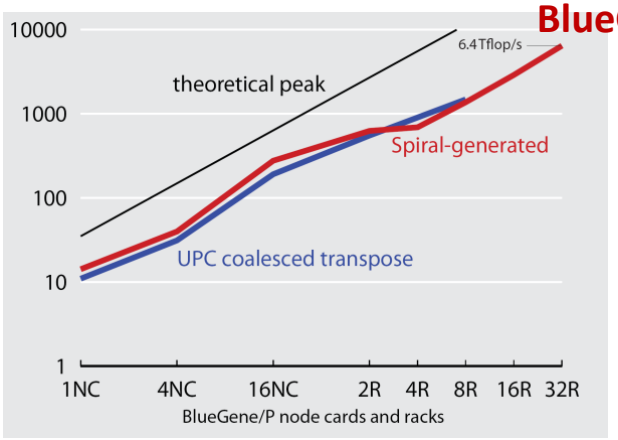
```
s3017 = _mm_loadl_pi(a772, ((float  *) (X + 14)));
s3018 = _mm_loadh_pi(_mm_loadl_pi(a772, ((float  *) (X + 24))), ((float  *) (X + 20)));
s3019 = _mm_loadl_pi(a772, ((float  *) (X + 8)));
s3020 = _mm_loadh_pi(_mm_loadl_pi(a772, ((float  *) (X + 10))), ((float  *) (X + 4)));
s3021 = _mm_loadl_pi(a772, ((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), (-0.0833333333333333
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), _m
s3047 = _mm_add_ps(_mm_mul_ps(_mm_set_ps(0.15689139105158462, (-0.15355568557954136), (-0.011599105605768193), 0.29071724147084099), t3799), _m
```



SPIRAL: Success 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)



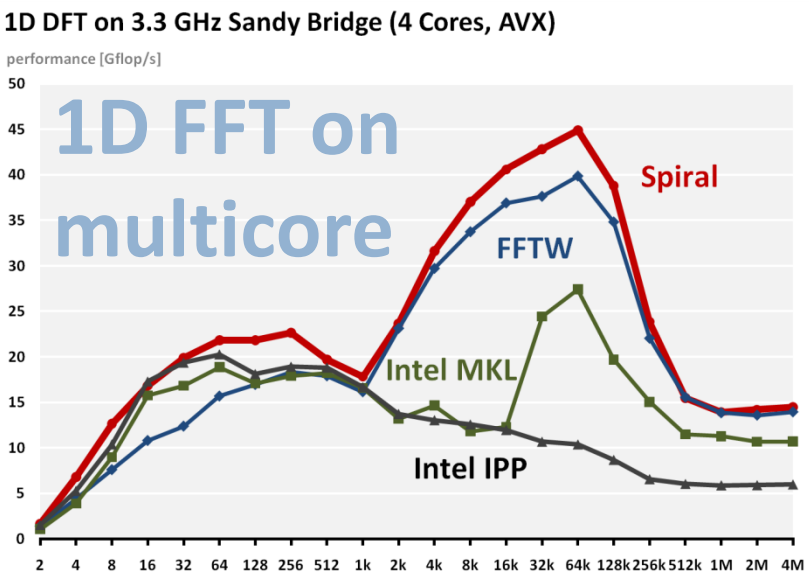
6.4 Tflop/s on BlueGene/P

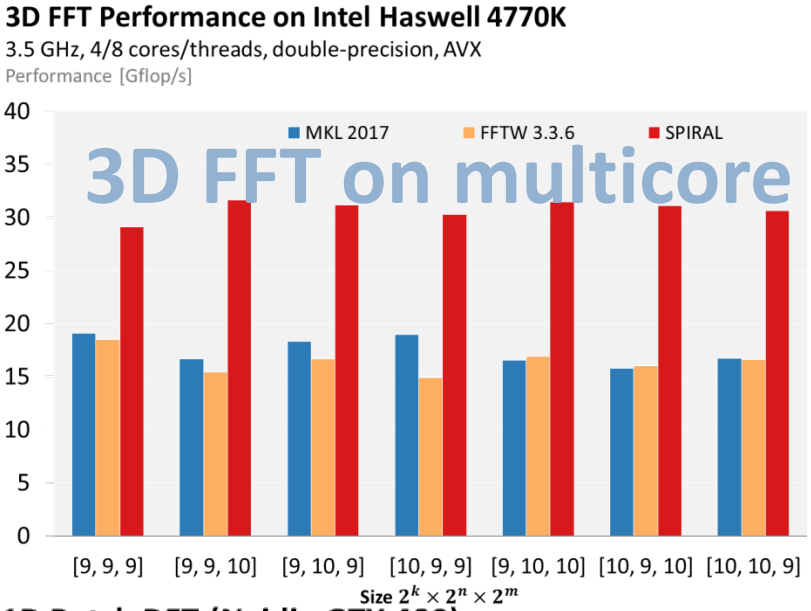
BlueGene/P at Argonne National Laboratory
128k cores (quad-core CPUs) at 850 MHz

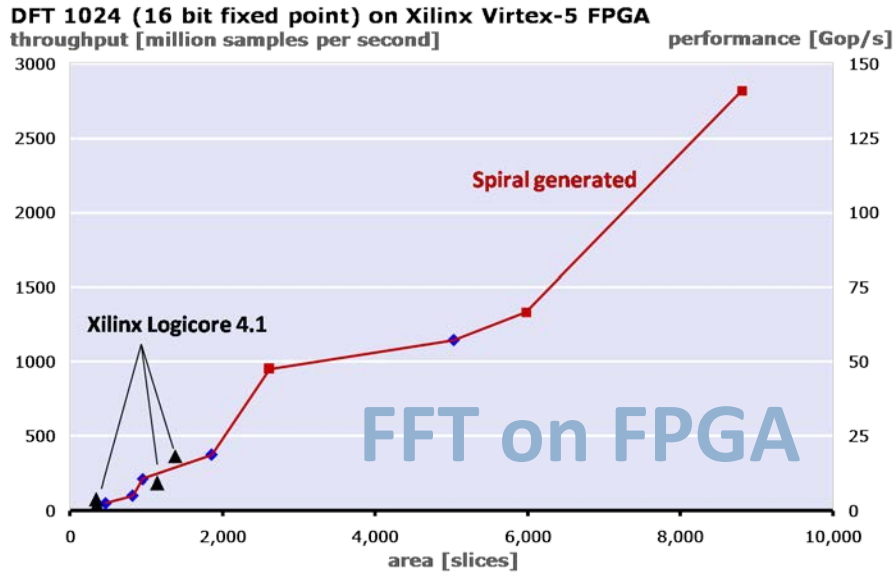
2006 Gordon Bell Prize (Peak Performance Award) with LLNL and IBM

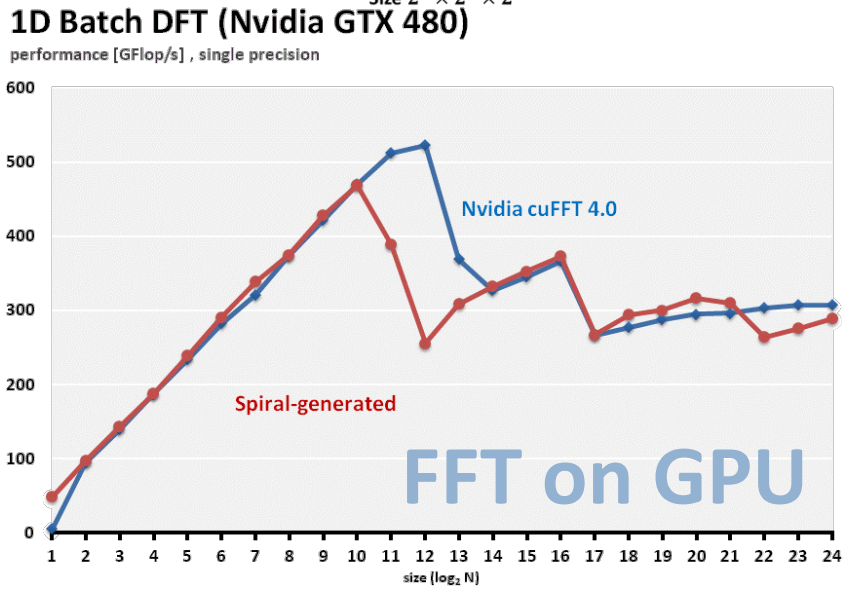
2010 HPC Challenge Class II Award (Most Productive System) with ANL and IBM

SPIRAL: Performance Across Platforms









Thrust 1: Repeat The Success of LAPACK for FFTs

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

Thrust 2: Updated FFTW Interface for Node FFTs

Front End: application facing

- **Backwards compatible to FFTW 2.X and 3.X**
  old code runs unmodified and gains substantially but not fully
- **Delayed execution/futures**
  asynchronous execution, offloading, triggering of execution
- **Data location and layout control**
  specify location of data (host/device, NUMA, NVRAM,…)
- **Performance metrics**
  user control of resource/performance/energy requirements and adaptation

Backend: developer and hardware facing

- **Extensible to novel and yet-unknown hardware and system architectures**
  future-proof platform extension API: transparently enable accelerators, etc.
- **Plug-in mechanism for special case acceleration packs**
  transparent access to application/architecture specialized FFT kernels

More Information:

www.spiral.net

www.spiralgen.com

We are about to make SPIRAL available as open source under a BSD style license