Spiral

Automating Library Development

Markus Püschel

... and the Spiral team (only part shown)

With:
Srinivas Chellappa
Frédéric de Mesmay
Franz Franchetti
Daniel McFarlin
Yevgen Voronenko

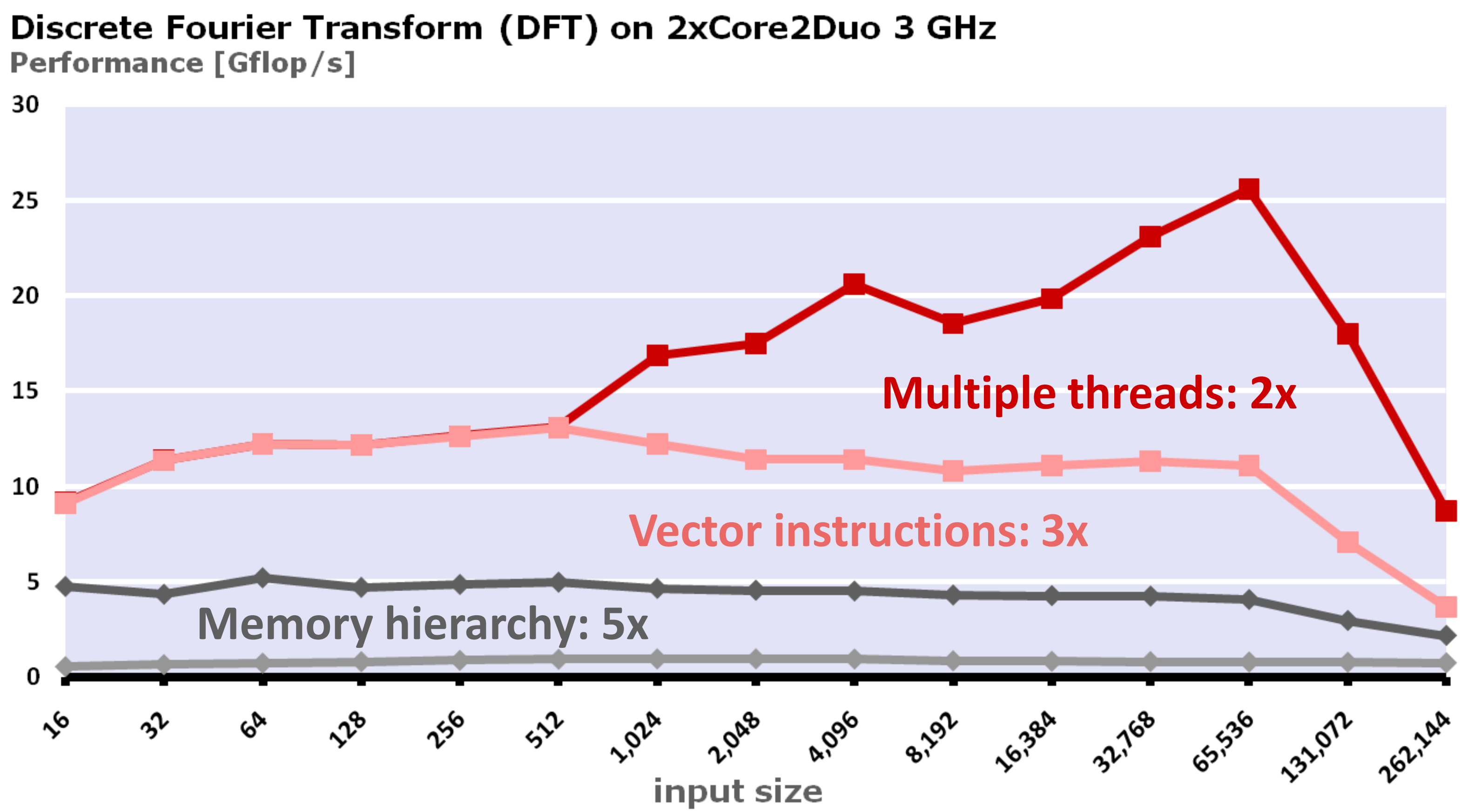
Electrical and
Computer Engineering
Carnegie Mellon University

This work was supported by
DARPA, NSF-NGS/ITR,ACR,CPA, Intel, Mercury, National Instruments

Positions and Thoughts

- **Autotuning definition**
  - Search over space of alternatives and
  - Parameter-based tuning are very important
  - but fails to address some key problems; we need to think about
- **Raising the level of abstraction: Enables**
  - Use of domain knowledge
  - Difficult optimizations: parallelization, vectorization, etc.
  - Faster porting to new platforms and platform paradigms
  - Possibly automatic software development
- **We need coarse platform abstractions**
- **We need more interdisciplinary collaborations**
- **Metrics**
  - Time for code development, porting to new platforms
  - Performance

DFT Plot: Analysis

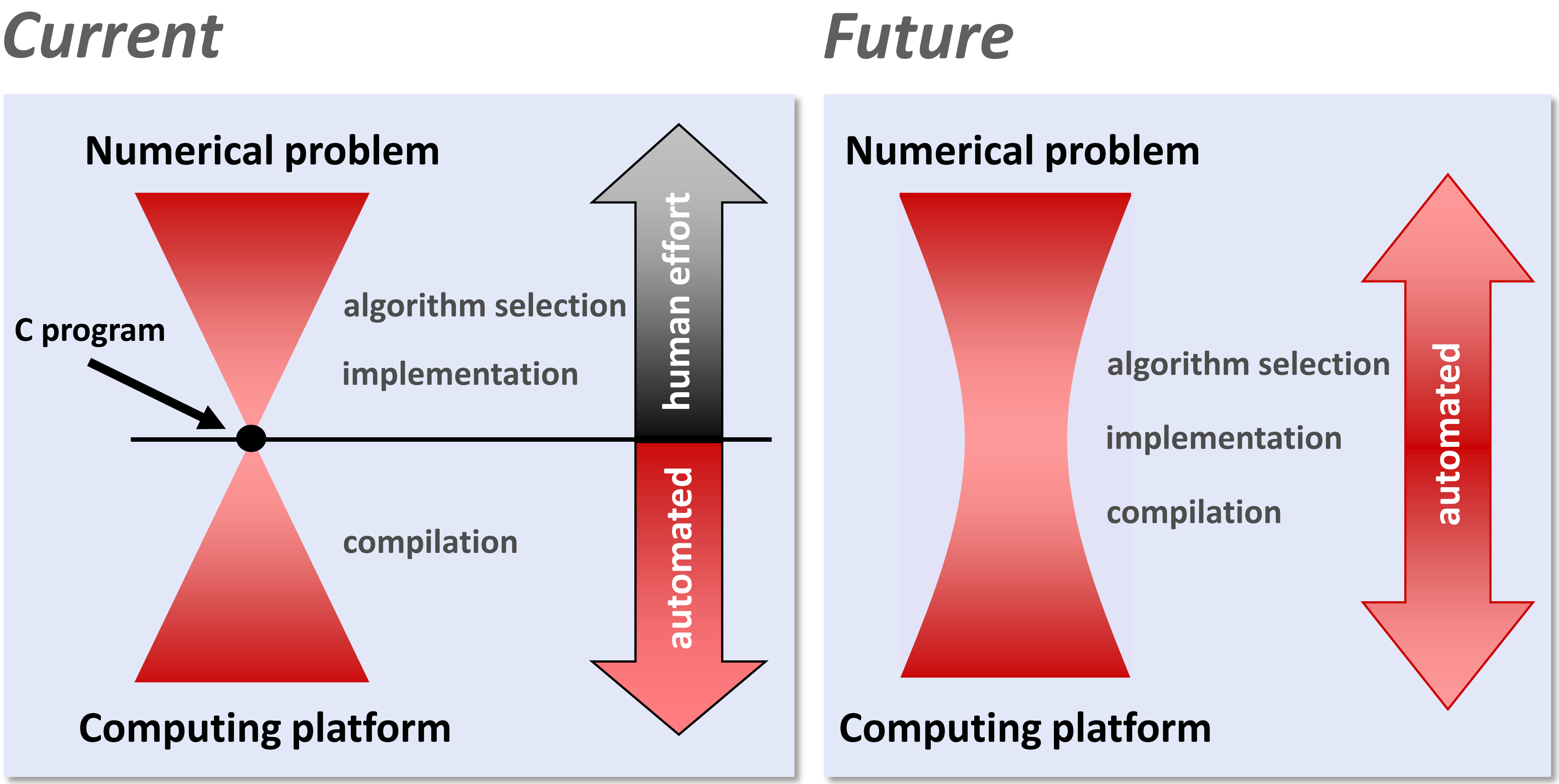


High performance library development has become a nightmare

Spiral

- **Research Goal: "Teach" computers to write fast libraries**
  - Complete automation of implementation and optimization
  - Including vectorization, parallelization
- **Functionality:**
  - Linear transforms (discrete Fourier transform, filters, wavelets)
  - BLAS
  - SAR imaging
  - En/decoding (Viterbi, Ebcot in JPEG2000)
  - … more
- **Platforms:**
  - **Desktop (vector, SMP), FPGAs,** GPUs, distributed, hybrid
- **Collaboration with Intel (Kuck, Tang, Sabanin)**
  - Parts of MKL/IPP generated with Spiral
  - IPP 6.0: ippg domain for Spiral generated code

Vision Behind Spiral



- C code a singularity: Compiler has no access to high level information
- Challenge: conquer the high abstraction level for complete automation

Organization

- **Spiral's framework: Example transforms**
  - Complete automation achieved
- **Beyond transforms**
- **Conclusions and thoughts**

Linear Transforms

- **Mathematically: Matrix-vector multiplication**

$$y = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_{n-1} \end{pmatrix} \qquad \boxed{y = Tx} \qquad x = \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_{n-1} \end{pmatrix}$$

Output vector | **Transform = matrix** | **Input vector**

- **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)

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

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

Breakdown Rules (>200 for >50 Transforms)

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

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

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

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

$$\mathbf{DCT\text{-}2}_n \to P^\top_{k/2,2m}\left(\mathbf{DCT\text{-}2}_{2m}\,K^{2m}_2 \oplus \left(I_{k/2-1} \otimes N_{2m}\,\mathbf{RDFT\text{-}3}^\top_{2m}\right)\right) B_n(L^{n/2}_{k/2} \otimes I_2)(I_m \otimes \mathbf{RDFT}'_k)Q_{m/2,k},$$

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

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

$$\mathbf{DFT}_n \;\to\; (\mathbf{DFT}_k \otimes \mathrm{I}_m)\,\mathrm{T}^n_m(\mathrm{I}_k \otimes \mathbf{DFT}_m)\,\mathrm{L}^n_k, \quad n = km$$

$$\mathbf{DFT}_n \;\to\; P_n(\mathbf{DFT}_k \otimes \mathbf{DFT}_m)Q_n, \quad n = km,\ \gcd(k,m) = 1$$

$$\mathbf{DFT}_p \;\to\; R^T_p(\mathrm{I}_1 \oplus \mathbf{DFT}_{p-1})D_p(\mathrm{I}_1 \oplus \mathbf{DFT}_{p-1})R_p, \quad p \text{ prime}$$

$$\mathbf{DCT\text{-}3}_n \;\to\; (\mathrm{I}_m \oplus \mathrm{J}_m)\,\mathrm{L}^n_m(\mathbf{DCT\text{-}3}_m(1/4) \oplus \mathbf{DCT\text{-}3}_m(3/4))$$
$$\cdot(\mathrm{F}_2 \otimes \mathrm{I}_m)\begin{bmatrix}\mathrm{I}_m & 0 \oplus -\mathrm{J}_{m-1}\\ & \frac{1}{\sqrt{2}}(\mathrm{I}_1 \oplus 2\,\mathrm{I}_m)\end{bmatrix}, \quad n = 2m$$

$$\mathbf{DCT\text{-}4}_n \;\to\; S_n\mathbf{DCT\text{-}2}_n\,\mathrm{diag}_{0\le k<n}(1/(2\cos((2k+1)\pi/4n)))$$

$$\mathbf{IMDCT}_{2m} \;\to\; (\mathrm{J}_m \oplus \mathrm{I}_m \oplus \mathrm{I}_m \oplus \mathrm{J}_m)\left(\left(\begin{bmatrix}1\\ -1\end{bmatrix} \otimes \mathrm{I}_m\right) \oplus \left(\begin{bmatrix}-1\\ -1\end{bmatrix} \otimes \mathrm{I}_m\right)\right)\mathrm{J}_{2m}\,\mathbf{DCT\text{-}4}_{2m}$$

$$\mathbf{WHT}_{2^k} \;\to\; \prod_{i=1}^{t}(\mathrm{I}_{2^{k_1+\cdots+k_{i-1}}} \otimes \mathbf{WHT}_{2^{k_i}} \otimes \mathrm{I}_{2^{k_{i+1}+\cdots+k_t}}), \quad k = k_1 + \cdots + k_t$$

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

$$\mathbf{DCT\text{-}2}_2 \;\to\; \mathrm{diag}(1, 1/\sqrt{2})\,\mathrm{F}_2$$

$$\mathbf{DCT\text{-}4}_2 \;\to\; \mathrm{J}_2\,\mathrm{R}_{13\pi/8}$$

Combining these rules yields many algorithms for every given transform

SPL to Sequential Code

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

$$I_m \otimes A_n = \begin{bmatrix} A_n & & \\ & \ddots & \\ & & A_n \end{bmatrix}$$

Correct code: easy *fast code: very difficult*

Program Generation in Spiral (Sketched)

Transform
user specified

$$\mathrm{DFT}_8$$

Fast algorithm in SPL
many choices

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

∑-SPL:
[PLDI 05]

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

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;
}
```

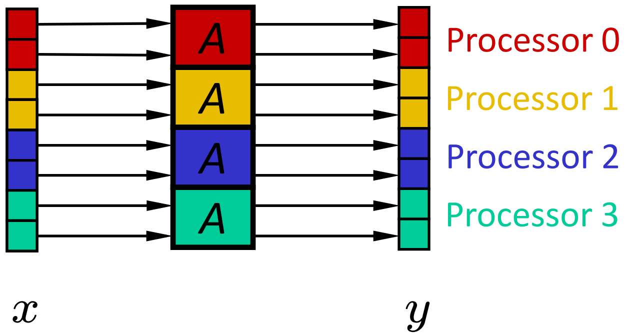
Optimization at all abstraction levels

- parallelization
  vectorization
- loop optimizations
- constant folding
  scheduling
  ……

SPL to Shared Memory Code: Basic Idea [SC 06]

- **Governing construct: tensor product**

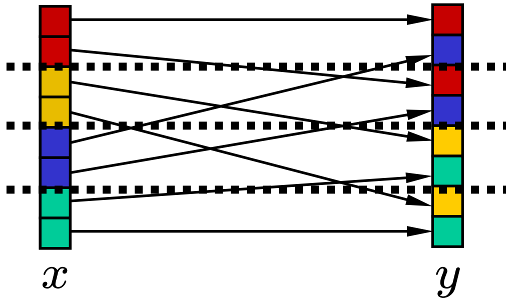
$$y = \left(\mathrm{I}_p \otimes A\right) x$$



p-way embarrassingly parallel, load-balanced

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

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



Task: Rewrite formulas to
extract tensor product + keep contiguous blocks

Parallelization by Rewriting

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

$$\ldots$$

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

coarse platform model

$$\ldots$$

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

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

Load-balanced

No false sharing

Same Approach for Other Parallel Paradigms

Message Passing

$$
\begin{aligned}
\underbrace{\mathrm{DFT}_{mn}}_{\mathrm{par}(p)} &\rightarrow \underbrace{(\mathrm{DFT}_m \otimes \mathrm{I}_n)}_{\mathrm{par}(p\leftarrow q)} \underbrace{\mathrm{T}_n^{mn}}_{\mathrm{par}(q)} \underbrace{(\mathrm{I}_m \otimes \mathrm{DFT}_n)}_{\mathrm{par}(q)} \underbrace{\mathrm{L}_m^{mn}}_{\mathrm{par}(q\leftarrow p)} \\
&\dots \\
&\dots \\
&\dots \\
&\rightarrow \underbrace{\left(\mathrm{I}_p \otimes \mathrm{L}_{m/p}^{mn/p}\right)}_{\mathrm{par}(p)} \underbrace{\left(\mathrm{L}_p^{p^2} \otimes \mathrm{I}_{mn/p^2}\right)}_{\mathrm{par}(p\leftarrow q)} \underbrace{\left(\mathrm{I}_q \otimes (\mathrm{I}_{p/q} \otimes \mathrm{L}_p^n \otimes \mathrm{I}_{m/p})\right)}_{\mathrm{par}(q)} \underbrace{\left(\mathrm{I}_q \otimes (\mathrm{I}_{n/q} \otimes \mathrm{DFT}_m)\right)}_{\mathrm{par}(q)} \\
&\quad \underbrace{\left(\mathrm{I}_q \otimes \mathrm{L}_{m/q}^{mn/q}\right)}_{\mathrm{par}(q)} \underbrace{\left(\mathrm{L}_q^{q^2} \otimes \mathrm{I}_{mn/q^2}\right)}_{\mathrm{par}(q)} \underbrace{\left(\mathrm{I}_q \otimes (\mathrm{L}_q^n \otimes \mathrm{I}_{m/q})\right)}_{\mathrm{par}(q)} \underbrace{\mathrm{T}_n^{mn}}_{\mathrm{par}(q)} \underbrace{\left(\mathrm{I}_q \otimes (\mathrm{I}_{m/q} \otimes \mathrm{DFT}_n)\right)}_{\mathrm{par}(q)} \\
&\quad \underbrace{\left(\mathrm{I}_q \otimes (\mathrm{I}_{p/q} \otimes \mathrm{L}_{m/p}^{mn/p})\right)}_{\mathrm{par}(q)} \underbrace{\left(\mathrm{L}_p^{p^2} \otimes \mathrm{I}_{mn/p^2}\right)}_{\mathrm{par}(q\leftarrow p)} \underbrace{\left(\mathrm{I}_p \otimes (\mathrm{L}_p^n \otimes \mathrm{I}_{m/p})\right)}_{\mathrm{par}(p)}
\end{aligned}
$$

Vectorization

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

Cg/OpenGL for GPUs:

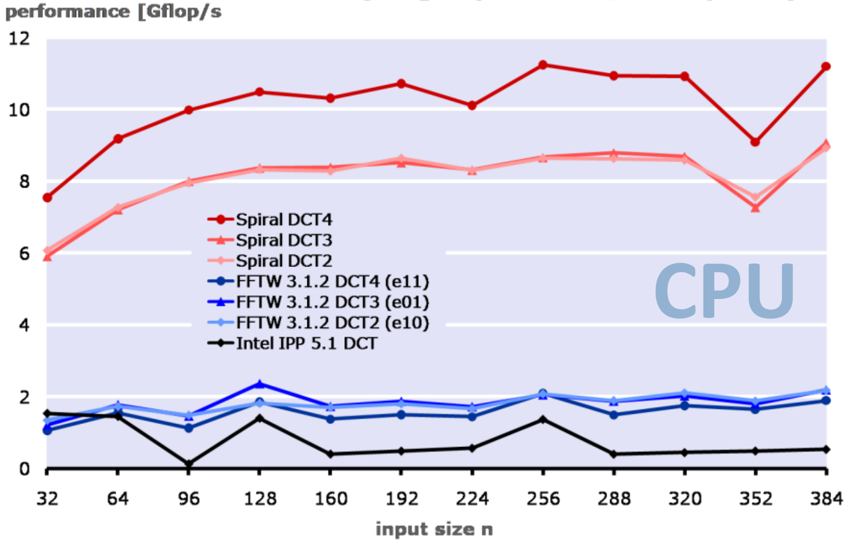
$$
\begin{aligned}
\underbrace{\left(\mathbf{DFT}_{r^k}\right)}_{\mathrm{gpu}(t,c)} &\rightarrow \underbrace{\left(\prod_{i=0}^{k-1} \mathrm{L}_r^{r^k}\left(\mathrm{I}_{r^{k-1}} \otimes \mathbf{DFT}_r\right)\left(\mathrm{L}_{r^{k-i-1}}^{r^k}(\mathrm{I}_{r^i} \otimes \mathrm{T}_{r^{k-i-1}}^{r^{k-i}})\,\underbrace{\mathrm{L}_{r^{i+1}}^{r^k}}_{\mathrm{vec}(c)}\righ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

Verilog for FPGAs:

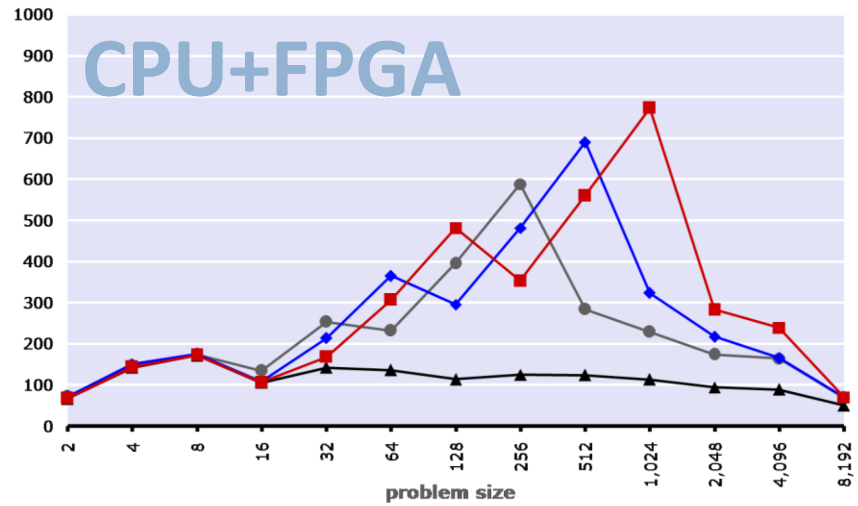
$$
\begin{aligned}
\underbrace{\left(\mathbf{DFT}_{r^k}\right)}_{\mathrm{stream}(r^s)} &\rightarrow \underbrace{\left[\prod_{i=0}^{k-1} \mathrm{L}_r^{r^k}\left(\mathrm{I}_{r^{k-1}} \otimes \mathbf{DFT}_r\right)\left(\mathrm{L}_{r^{k-i-1}}^{r^k}(\mathrm{I}_{r^i} \otimes \mathrm{T}_{r^{k-i-1}}^{r^{k-i}})\,\mathrm{L}_{r^{i+1}}^{r^k}\right)\right]\mathrm{R}_r^{r^k}}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

Example Results

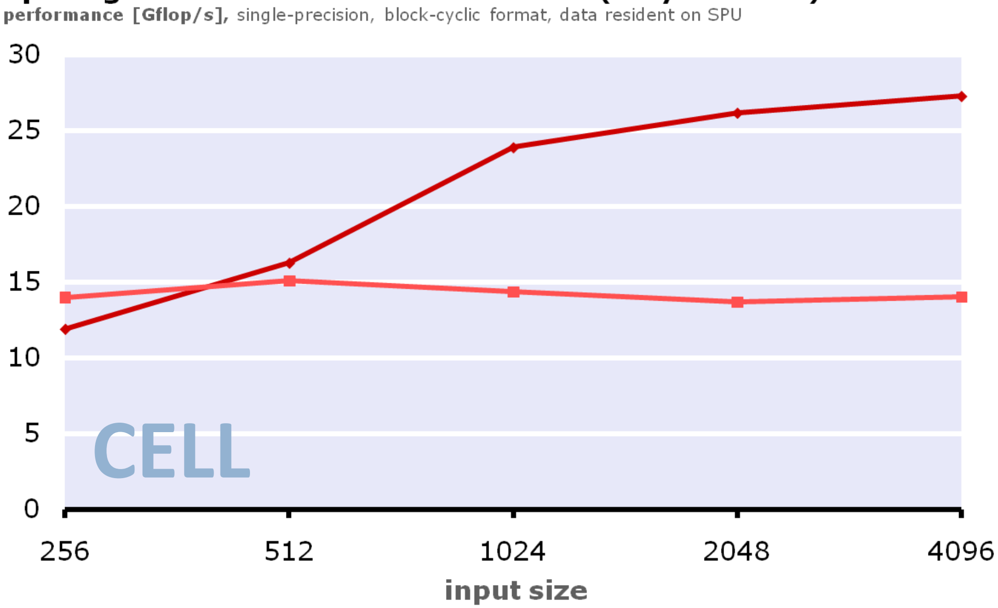
DCT on 2.66 GHz Core2 (single-precision, 4-way SSE)



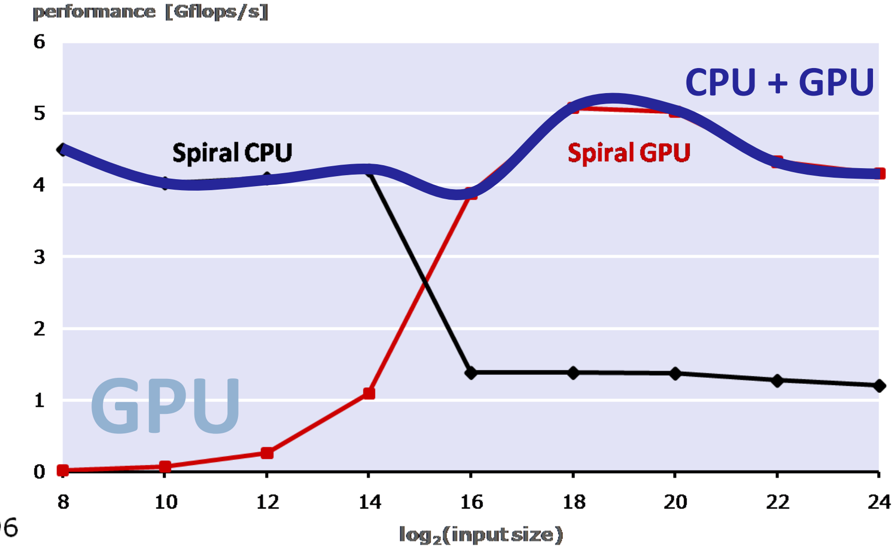
DFT (16 bit fixed point): Hardware Accelerated Software on Xilinx Virtex-II Pro FPGA and Embedded PowerPC 405 Processor



Spiral-generated FFT on 3.2 GHz Cell BE (PlayStation 3)



WHT (single precision) on 3.6 GHz Pentium 4 with Nvidia 7900 GTX



Summary: Complete Automation for Transforms

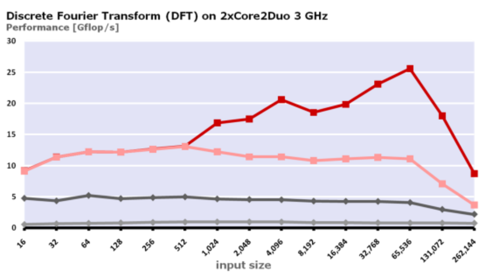
- Platform: Off-the-shelf desktop
- Often: generated code faster than competition (if exists)

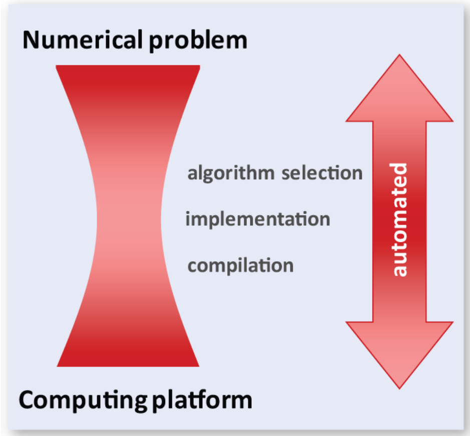
- **Memory hierarchy optimization**
  - Rewriting and search for algorithm selection
  - Rewriting for loop optimizations
- **Vectorization**
  - Rewriting
- **Parallelization**
  - Rewriting

fixed input size code

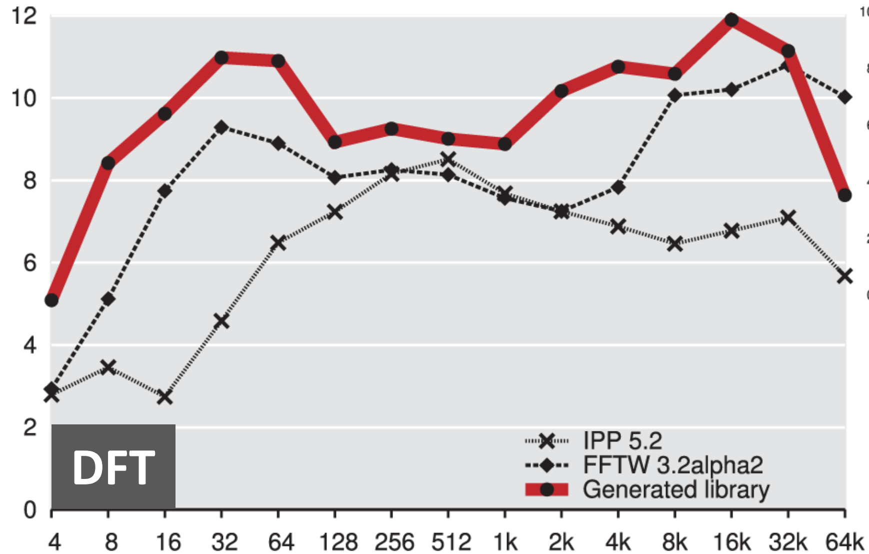
- **Derivation of library structure**
  - Rewriting
  - Other methods

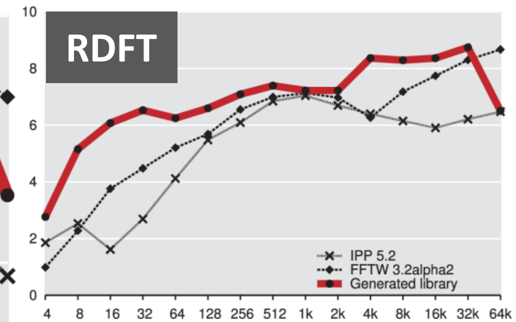
general input size library

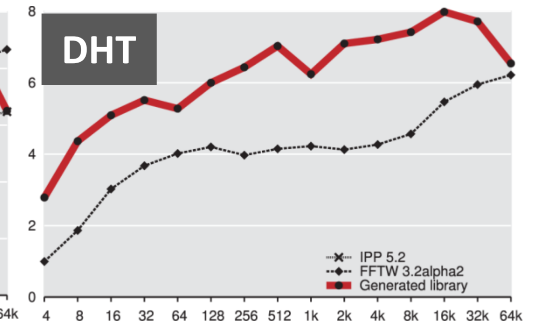


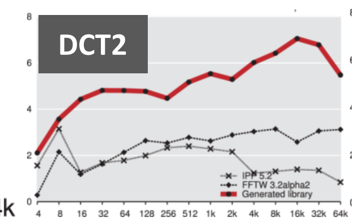


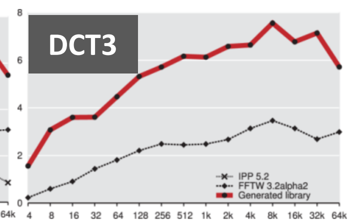
Generated Libraries

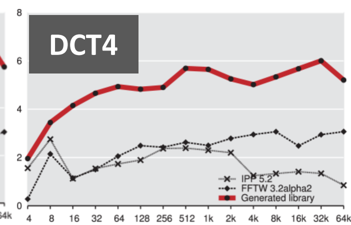


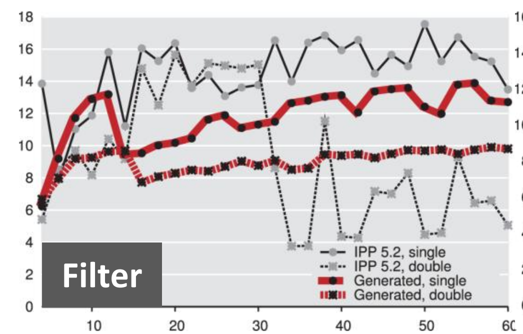


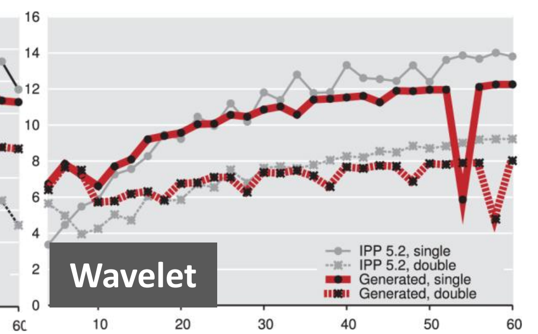












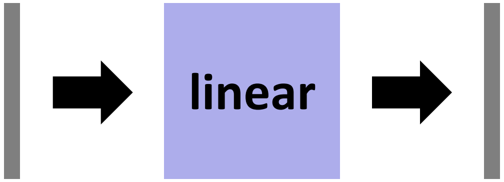
- 2-way vectorized, 2-threaded
- Most are faster than hand-written libs
- Recursion steps: 4–17
- Code size: 8–120 kloc or 0.5–5 MB
- Generation time: 1–3 hours

Organization

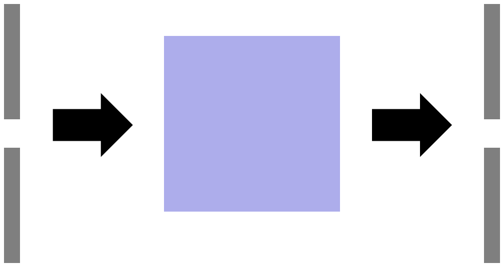
- **Spiral's framework: Example transforms**
  - Complete automation achieved
- **Beyond transforms**
  - Operator language
  - BLAS, Viterbi decoding, SAR imaging, Ebcot encoding
- **Conclusions and thoughts**

Going Beyond Transforms

- **Transform =**
  linear operator with one vector input and one vector output



- **Key ideas:**
  - Generalize to (possibly nonlinear) operators with several inputs and several outputs
  - Generalize SPL (including tensor product) to OL (operator language)
  - Generalize rewriting systems for parallelizations



Operator Language

name	definition
basic operators	
projection	$\pi_{\mathbf{x}} : \mathbb{C}^m \times \mathbb{C}^n \to \mathbb{C}^m;\ (\mathbf{x}, \mathbf{y}) \mapsto \mathbf{x}$
linear transform	$\mathsf{M} : \mathbb{C}^n \to \mathbb{C}^m; \mathbf{x} \mapsto M\mathbf{x}$
stride	$\mathsf{L}^{mn}_m : \mathbb{C}^{mn} \to \mathbb{C}^{mn};\ \mathbf{x} \mapsto \mathsf{L}^{mn}_m \mathbf{x}$
vector sum	$\Sigma_n : \mathbb{C}^n \to \mathbb{C};\ \mathbf{x} \mapsto \sum_{i=0}^{n-1} x_i$
vector minimum	$\mathsf{min}_n : \mathbb{C}^n \to \mathbb{C};\ \mathbf{x} \mapsto \mathsf{min}(x_0, \ldots, x_{n-1})$
constant vector	$\mathsf{C}_{\mathbf{c}} : \varnothing \to \mathbb{C}^n;\ () \mapsto \mathbf{c}$
operations	
addition	$(\mathsf{M} + \mathsf{N})(\mathbf{x}, \mathbf{y}) = \mathsf{M}(\mathbf{x}, \mathbf{y}) + \mathsf{N}(\mathbf{x}, \mathbf{y})$
multiplication	$(\mathsf{M} \cdot \mathsf{N})(\mathbf{x}, \mathbf{y}) = \mathsf{M}(\mathbf{x}, \mathbf{y}) \cdot \mathsf{N}(\mathbf{x}, \mathbf{y})$
direct sum	$(\mathsf{M} \oplus \mathsf{N})(\mathbf{x} \oplus \mathbf{u}, \mathbf{y} \oplus \mathbf{v}) = \mathsf{M}(\mathbf{x}, \mathbf{y}) \oplus \mathsf{N}(\mathbf{u}, \mathbf{v})$
cartesian product	$(\mathsf{M} \times \mathsf{N})(\mathbf{x}, \mathbf{y}, \mathbf{u}, \mathbf{v}) = \mathsf{M}(\mathbf{x}, \mathbf{y}) \times \mathsf{N}(\mathbf{u}, \mathbf{v})$
composition	$(\mathsf{M} \circ \mathsf{N})(\mathbf{x}, \mathbf{y}) = \mathsf{M}(\mathsf{N}(\mathbf{x}, \mathbf{y}))$
iterative composition	$\left(\prod_{i=0}^{n-1} \mathsf{M}_i\right)(\mathbf{x}, \mathbf{y}) = (\mathsf{M}_0 \circ \cdots \circ \mathsf{M}_{n-1})(\mathbf{x}, \mathbf{y})$
tensor product	$\mathrm{I} \otimes \mathsf{M},\ \ \mathsf{M} \otimes \mathrm{I}$

Example: Matrix Multiplication (MMM)

Breakdown rules = algorithm knowledge:

capture various forms of blocking

breakdown rule	description
$\mathrm{MMM}_{1,1,1} \rightarrow (\cdot)_1$	base case
$\mathrm{MMM}_{m,n,k} \rightarrow (\otimes)_{m/m_b \times 1} \otimes \mathrm{MMM}_{m_b,n,k}$	horizontal blocking
$\mathrm{MMM}_{m,n,k} \rightarrow \mathrm{MMM}_{m,nb,k} \otimes (\otimes)_{1 \times n/nb}$	interleaved blocking
$\mathrm{MMM}_{m,n,k} \rightarrow ((\Sigma_{k/k_b} \circ (\cdot)_{k/k_b}) \otimes \mathrm{MMM}_{m,n,k_b}) \circ ((L^{mk/k_b}_{k/k_b} \otimes I_{k_b}) \times I_{kn})$	accumulative blocking
$\mathrm{MMM}_{m,n,k} \rightarrow (L^{mn/n_b}_{m} \otimes I_{n_b}) \circ ((\otimes)_{1 \times n/n_b} \otimes \mathrm{MMM}_{m,n_b,k}) \circ (I_{km} \times (L^{kn/n_b}_{n/n_b} \otimes I_{n_b}))$	vertical blocking

Parallelization through rewriting

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

$$\rightarrow \underbrace{\left(\mathrm{I}_m \otimes \mathsf{L}_p^n\right) \circ \left(\mathsf{MMM}_{m,n/p,k} \otimes (\otimes)_{1\times p\rightarrow p}\right) \circ \left(\mathrm{I}_{km} \times (\mathrm{I}_k \otimes \mathsf{L}_{n/p}^n)\right)}_{\mathsf{smp}(p,\mu)}$$

$$\rightarrow \underbrace{\left(\mathrm{I}_m \otimes \mathsf{L}_p^n\right)}_{\mathsf{smp}(p,\mu)} \circ \underbrace{\left(\mathsf{MMM}_{m,n/p,k} \otimes (\otimes)_{1\times p\rightarrow p}\right)}_{\mathsf{smp}(p,\mu)} \circ \underbrace{\left(\mathrm{I}_{km} \times (\mathrm{I}_k \otimes \mathsf{L}_{n/p}^n)\right)}_{\mathsf{smp}(p,\mu)}$$

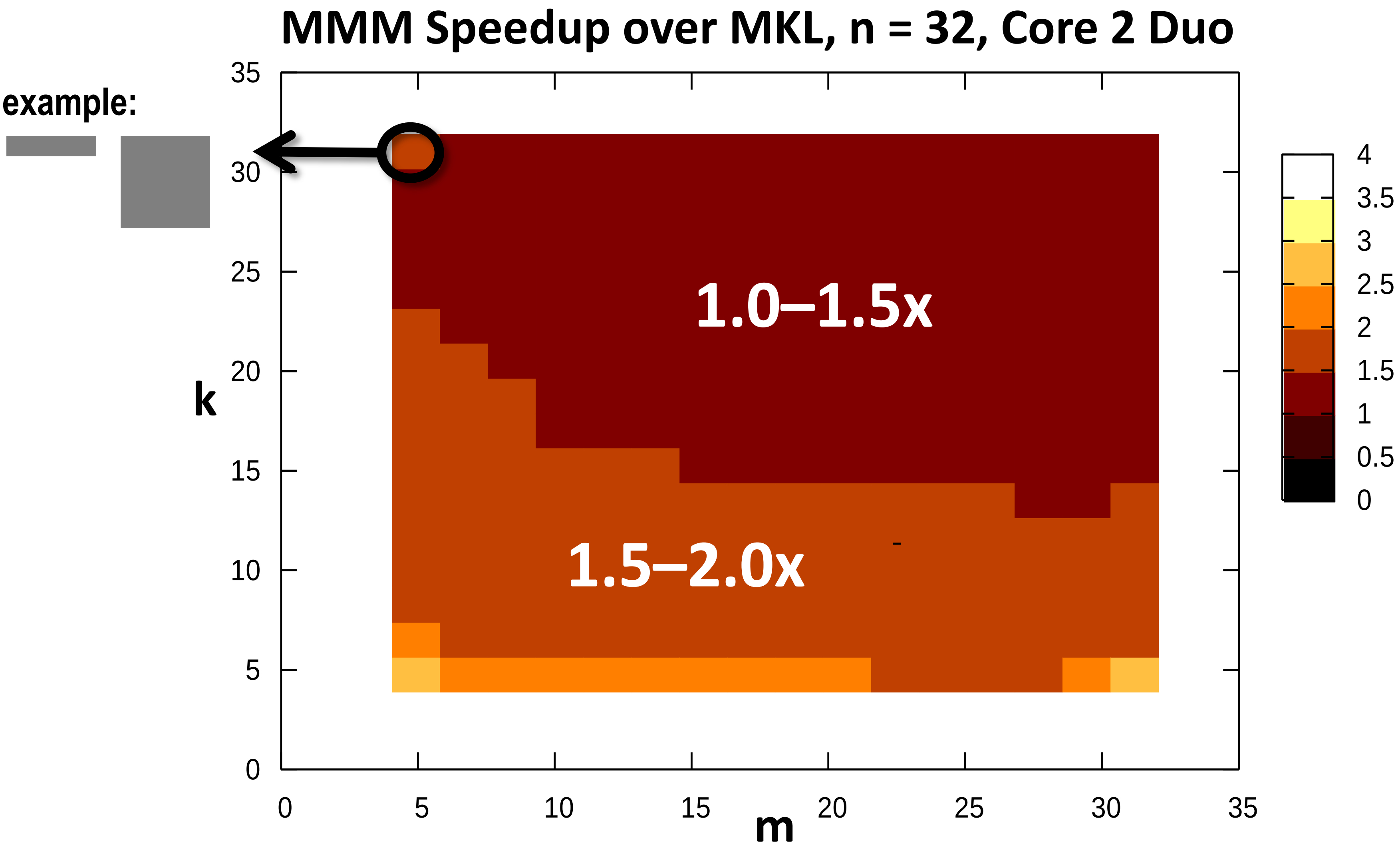
$$\rightarrow \underbrace{\left(\mathrm{I}_m \otimes \mathsf{L}_p^n\right)}_{\mathsf{smp}(p,\mu)} \circ \underbrace{\mathsf{L}_{m/pn}^{mn}}_{\mathsf{smp}(p,\mu)} \circ \left((\otimes)_{1\times p\rightarrow p} \otimes_{\|} \mathsf{MMM}_{m/p,n,k}\right) \circ \underbrace{\left(\mathrm{I}_{km} \times \mathsf{L}_p^{kn}\right)}_{\mathsf{smp}(p,\mu)} \circ \underbrace{\left(\mathrm{I}_{km} \times (\mathrm{I}_k \otimes \mathsf{L}_{n/p}^n)\right)}_{\mathsf{smp}(p,\mu)}$$

$$\rightarrow \left((\mathsf{L}_m^{mp} \otimes \mathrm{I}_{n/(p\mu)}) \bar{\otimes} \mathrm{I}_\mu\right) \circ \left((\otimes)_{1\times p\rightarrow p} \otimes_{\|} \mathsf{MMM}_{m,n/p,k}\right) \circ \left((\mathrm{I}_{km/\mu} \bar{\otimes} \mathrm{I}_\mu) \times ((\mathsf{L}_p^{kp} \otimes \mathrm{I}_{n/(p\mu)}) \bar{\otimes} \mathrm{I}_\mu)\right)$$

Load-balanced

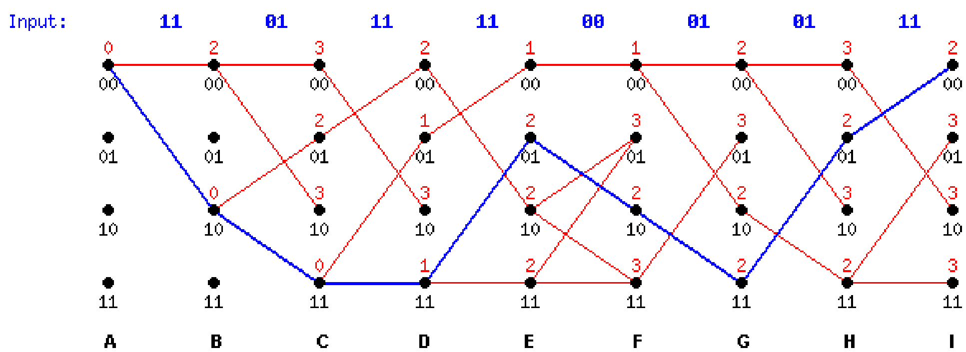
No false sharing

Speed-up (m x k) *times* (k x n)



- Comparison to GotoBLAS similar

Viterbi Decoding in OL



http://www.ece.unb.ca/tervo/ee4253/convolution3.htm

- **Operator for Viterbi decoder** $\mathrm{Vit}_{r,K,N,p} : \mathbb{N}^{rN} \rightarrow \mathbb{N}^{2^{K-1}} \times \mathbb{N}^{N2^{K-1}}$
- **Breakdown rules**

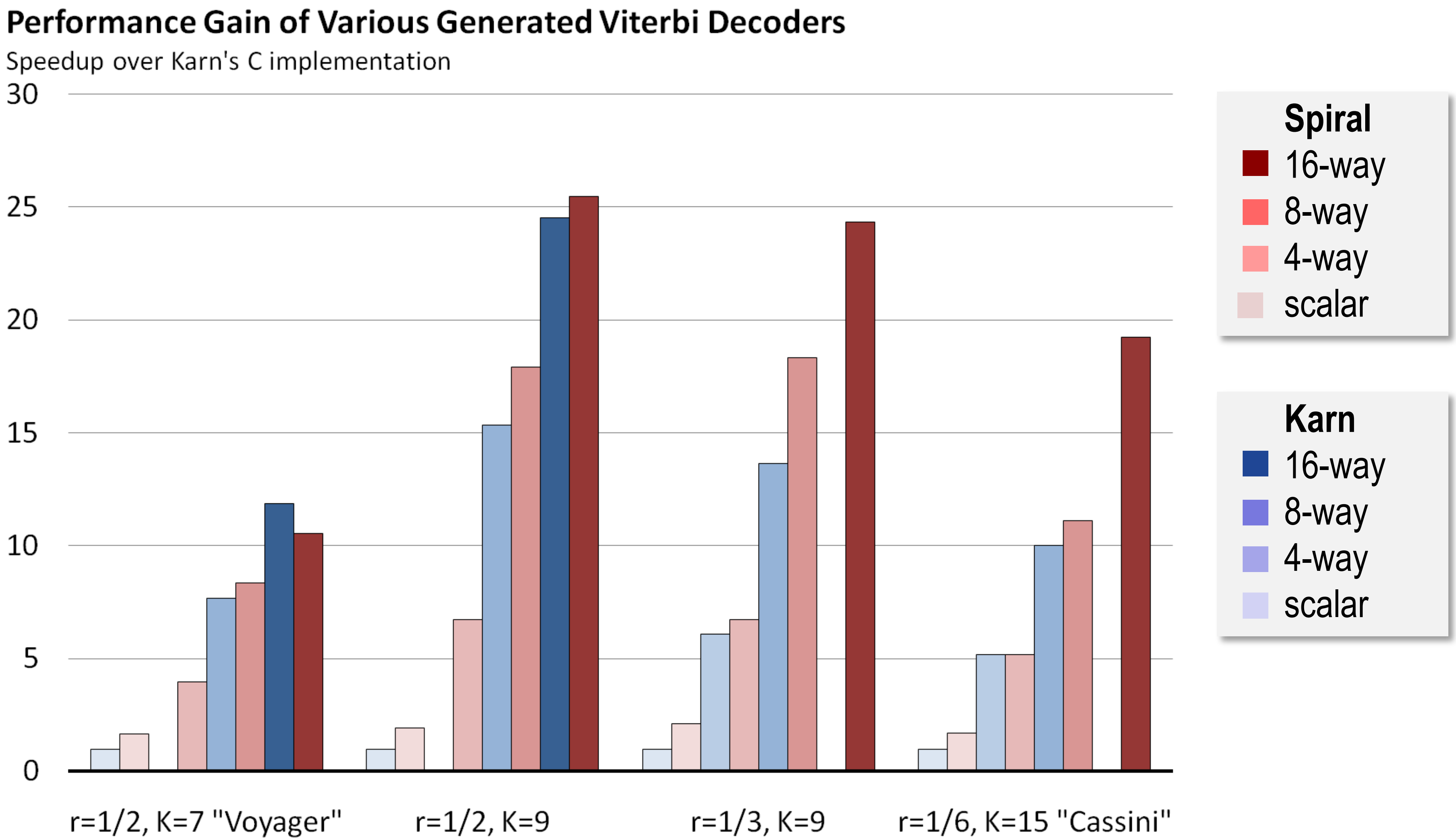
$$\mathrm{Vit}_{r,K,N,p} \rightarrow$$

$$\pi_{2,3} \circ \left(\prod_{0 \leq x < N} (L^{2^{K-1}}_{2^{K-2}} \times I_{r2^{K-1} \times N2^{K-1}}) \circ (I_{2^{K-2} \times 2^{K-2} \times 1} \otimes C^{x}_{r,K,p}) \right) \circ Id_{(1)_2 \otimes i_{2^{K-2}}}$$

$$C^{x}_{r,K,p} \circ Id_y \rightarrow B^{x,y}_{r,K,p}$$

Results

Karn's implementation: **hand-written assembly for 4 Viterbi codes**



EBCOT Coding in OL

$$\mathrm{SC}(\chi_{m,n}, \sigma_{m,n}) : (\mathbb{Z}_2^9 \times \mathbb{Z}_2^9) \to (\mathbb{N}, \mathbb{Z}_2)$$

$$(I \times \mathrm{xor}_2) \circ (T_{SC} \times I) \circ (H \times V \times I) \circ (\underline{L_4^2} \times G_4) \circ (\underline{\begin{pmatrix} 1 \\ 1 \end{pmatrix}} \times \underline{\begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}})$$

$$\mathrm{H, V} : (\mathbb{Z}_2^9 \times \mathbb{Z}_2^9) \to \mathbb{N}$$

$$\mathrm{H}: \quad h \circ (f \times f) \circ (G_1 \times \mathrm{C}_{-2} \times G_1 \times G_7 \times \mathrm{C}_{-2} \times G_7) \circ \underline{L_4^2} \circ (\underline{\binom{1}{1}} \times \underline{\binom{1}{1}})$$

$$\mathrm{V}: \quad h \circ (f \times f) \circ (G_3 \times \mathrm{C}_{-2} \times G_3 \times G_5 \times \mathrm{C}_{-2} \times G_5) \circ \underline{L_4^2} \circ (\underline{\binom{1}{1}} \times \underline{\binom{1}{1}})$$

$$f : \mathrm{mul}_2 \circ (I \times \mathrm{sub}_2) \circ (I \times \mathrm{C}_1 \times \mathrm{mul}_2)$$

$$h : \mathrm{min}_2 \circ (\mathrm{C}_1 \times \mathrm{max}_2) \circ (\mathrm{C}_{-1} \times \mathrm{sum}_2)$$

$$\mathrm{sppCode}\ (\sigma_{m,n}) : \mathbb{Z}_2^9 \to \mathbb{Z}_2$$

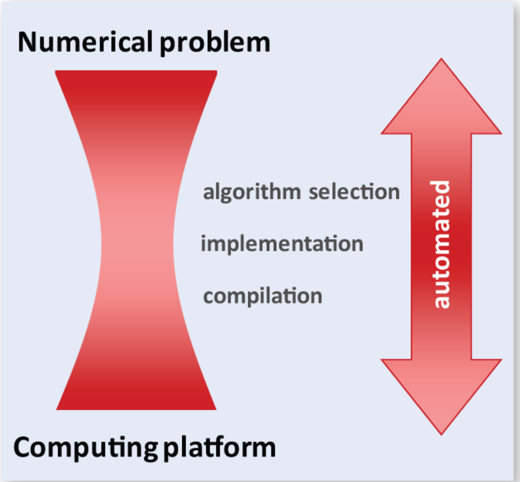
$$\mathrm{and}_2 \circ (\mathrm{eqz} \times \mathrm{nez}) \circ (G_{(4)_9} \times (G_0 + G_1 + G_2 + G_3 + G_5 + G_6 + G_7 + G_8)) \circ \underline{\begin{pmatrix} 1 \\ 1 \end{pmatrix}}$$

Organization

- Spiral's framework: Example transforms
  - Complete automation achieved
- Beyond transforms
- **Conclusions and thoughts**

Raising the Abstraction Level

- **Formally describe and structure algorithms/applications**
  eternally valid
- **In Spiral**
  - Domain-specific, declarative, mathematical language OL
  - Difficult optimizations/transformations by rewriting
  - What it enables
    - Vectorization, parallelization using domain knowledge
    - Efficient retargeting to new platforms and new platform paradigms
    - Complete automation in some cases



- **Other examples**
  - Libraries
  - Identification and definition of BLAS
- **Parameter tuning**
  - Indispensable tool but cannot achieve the above

Interdisciplinary Research Needed

Programming languages
Program generation

Symbolic Computation
Rewriting

Automating
High-Performance
Parallel Library
Development

Software
Scientific Computing

Algorithms
Mathematics

Compilers

We Need to Work Together