Parallelism in Spiral

Franz Franchetti

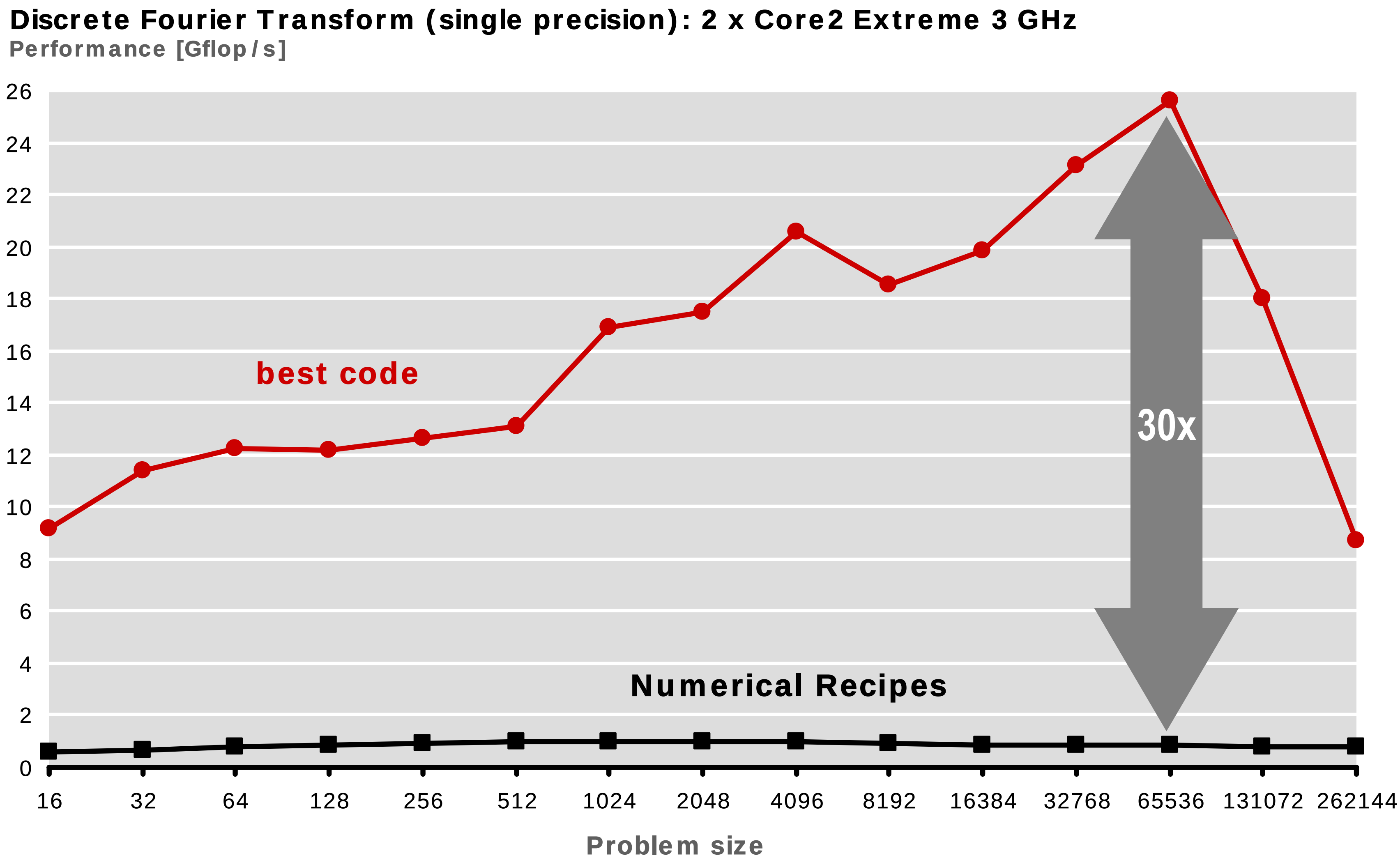
Electrical and
Computer Engineering
Carnegie Mellon University

… and the Spiral team (only part shown)

Joint work with
Yevgen Voronenko
Markus Püschel

This work was supported by
DARPA DESA program, NSF-NGS/ITR, NSF-ACR, and Intel

The Problem



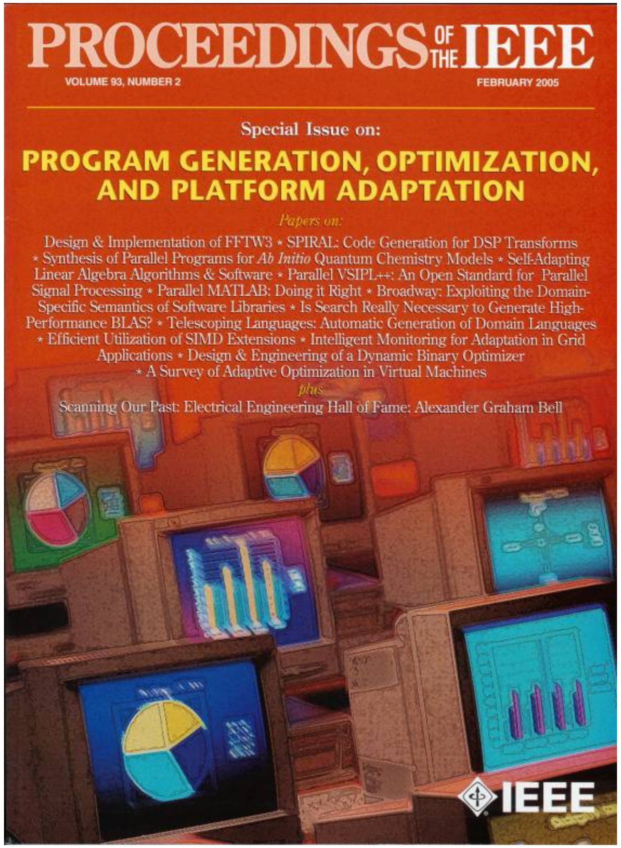
What's going on?

Automatic Performance Tuning

- **Current vicious circle:** **Whenever a new platform comes out, the same functionality needs to be rewritten and reoptimized**

- **Automatic Performance Tuning**
  - BLAS: ATLAS
  - Linear algebra: Bebop, Spike, Flame
  - Sorting
  - Fourier transform: FFTW
  - Linear transforms: Spiral
  - …others
  - New compiler techniques

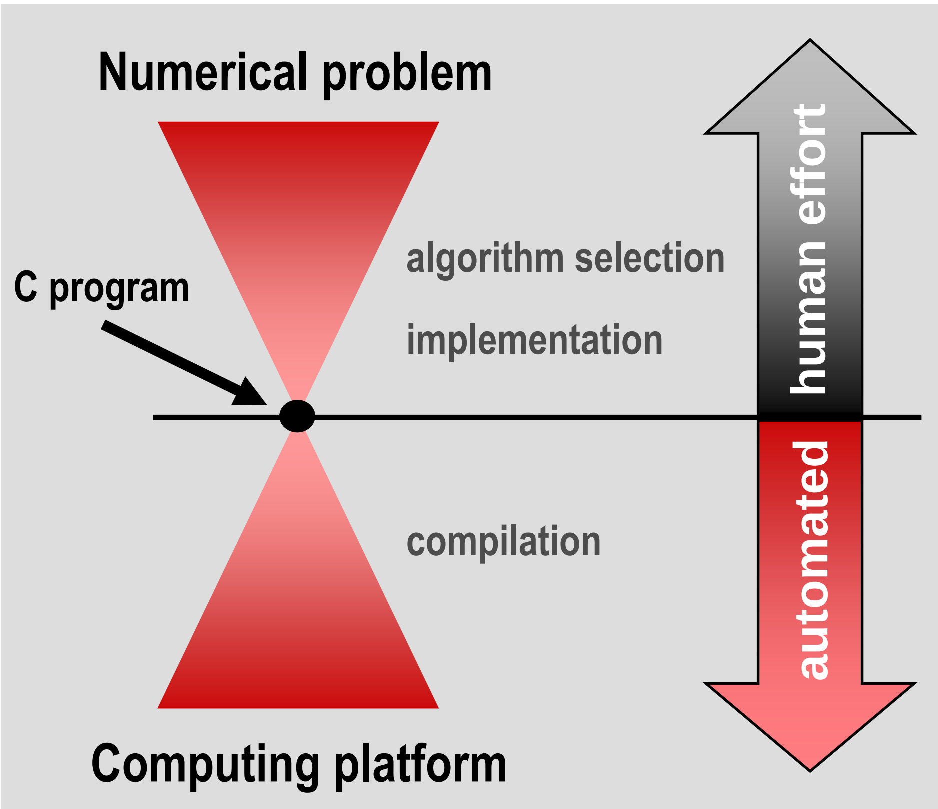
But what about parallelism … ?



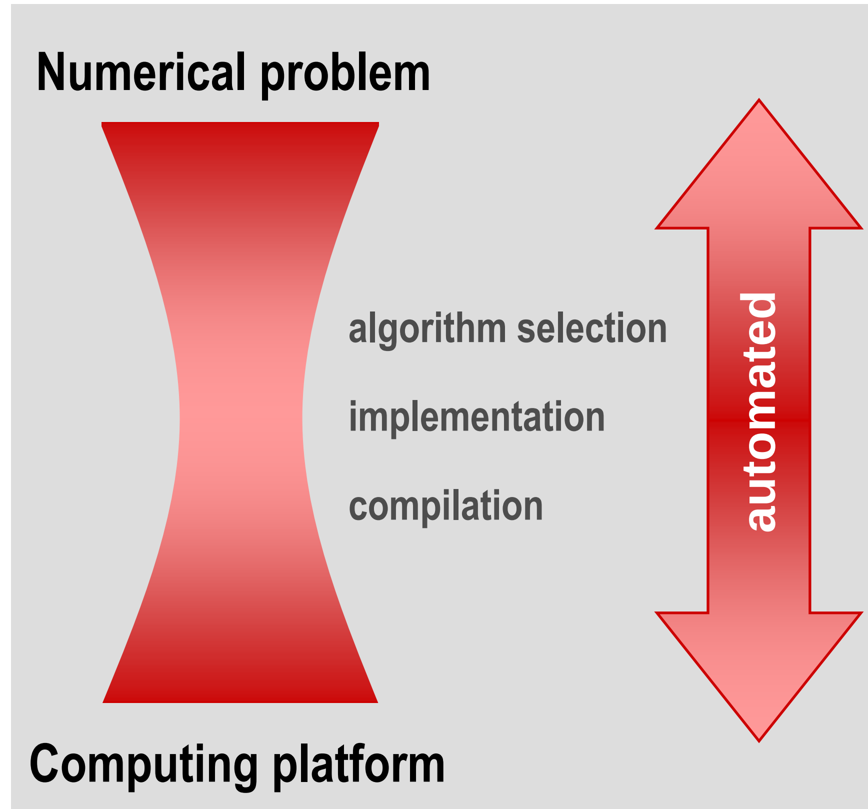
Proceedings of the IEEE special issue, Feb. 2005

Vision Behind Spiral

Current



Future



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

Organization

- **Spiral overview**
- Parallelization in Spiral
- Results
- Concluding remarks

Spiral

- Library generator for linear transforms
  (DFT, DCT, DWT, filters, ….) *and recently more …*
- Wide range of platforms supported:
  scalar, fixed point, vector, parallel, Verilog, GPU
- Research Goal: "Teach" computers to write fast libraries
  - Complete automation of implementation and optimization
  - Conquer the "high" algorithm level for automation
- When a new platform comes out:
  Regenerate a retuned library
- When a new platform paradigm comes out (e.g., vector or CMPs):
  Update the tool rather than rewriting the library

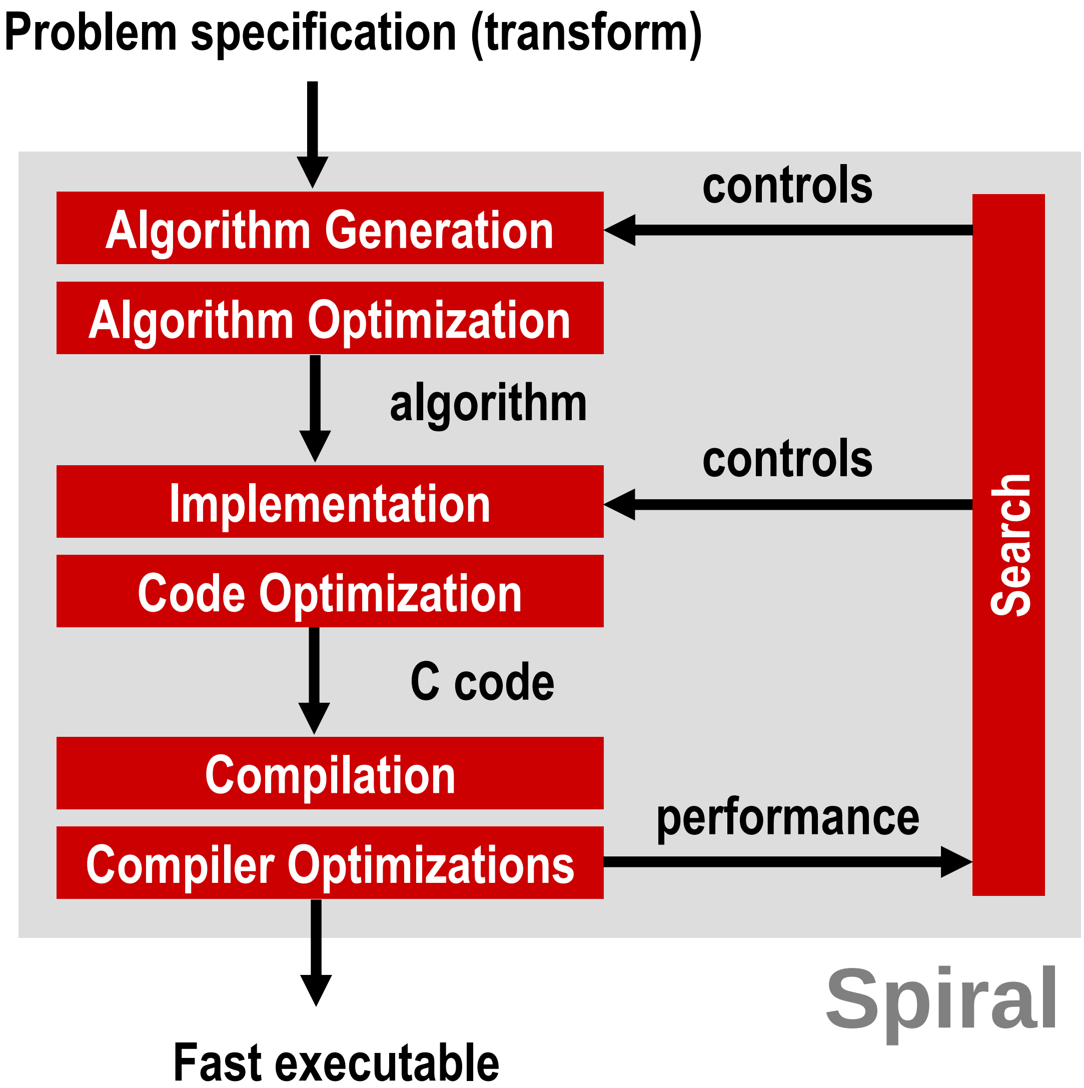
Intel has started to use Spiral to generate parts of their MKL library

How Spiral Works

Spiral:
Complete automation of the implementation and optimization task

Basic idea:
Declarative representation of algorithms

Rewriting systems to generate and optimize algorithms



What is a (Linear) Transform?

- **Mathematically: Matrix-vector multiplication**

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

input vector (signal)

output vector (signal)

transform = matrix

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

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

Transform Algorithms: Example 4-point FFT

Cooley/Tukey fast Fourier transform (FFT):

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

Fourier transform

Diagonal matrix (twiddles)

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

Kronecker product

Identity

Permutation

- Algorithms reduce arithmetic cost $O(n^2) \rightarrow O(n\log(n))$
- Product of structured sparse matrices
- Mathematical notation exhibits structure: SPL (signal processing language)

Examples: Transforms

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

Spiral currently contains 55 transforms

Examples: Breakdown Rules (currently ≈220)

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

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

DFT

DCT-3

- “Teaches” Spiral about existing algorithm knowledge (~200 journal papers)
- Includes many new ones (algebraic theory, Pueschel, Moura, Voronenko)

DCT-4

IMDCT ... DCT-4$_{2m}$

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

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

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

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

Base case rules

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];)`

$\boxed{y = (I_m \otimes A_n)x}$	`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

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

Program Generation in Spiral (Sketched)

Transform
user specified

$$\mathrm{DFT}_8$$

Fast algorithm in SPL
many choices

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

∑-SPL:

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

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
……

Organization

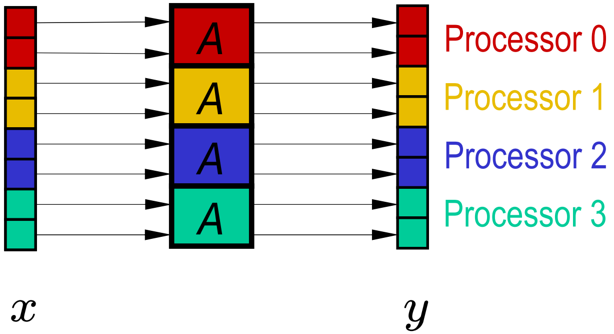
- Spiral overview
- **Parallelization in Spiral**
- Results
- Concluding remarks

SPL to Shared Memory Code: Basic Idea

[SC 06]

- Governing construct: tensor product

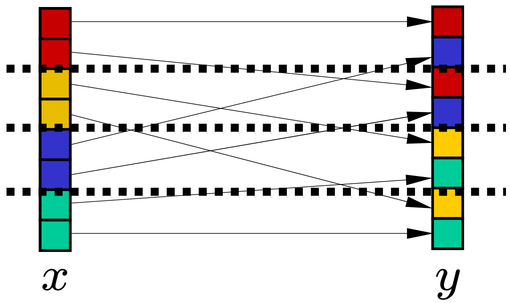
$$y = \left(\mathrm{I}_p \otimes A\right) x$$



Independent operation, load-balanced

- Problematic construct: permutations produce false sharing

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



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

Step 1: Shared Memory Tags

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

- **Introduce them as tags in SPL**

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

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

Step 2: Identify "Good" Formulas

- **Load balanced, avoiding false sharing**

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

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

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

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

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

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

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

where A and B are fully optimized.

Step 3: Identify Rewriting Rules

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

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

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

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

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

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

Simple Rewriting Example

$$A_m \otimes \mathrm{I}_n$$

Loop splitting + loop exchange

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

fully optimized

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

Parallelization by Rewriting

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

$$\ldots$$

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

$$\ldots$$

$$\rightarrow\; \Big((\mathsf{L}^{mp}_m \otimes \mathrm{I}_{n/p\mu}) \otimes_\mu \mathrm{I}_\mu\Big)\Big(\mathrm{I}_p \otimes_\| (\mathbf{DFT}_m \otimes \mathrm{I}_{n/p})\Big)\Big((\mathsf{L}^{mp}_p \otimes \mathrm{I}_{n/p\mu}) \otimes_\mu \mathrm{I}_\mu\Big)$$
$$\Big(\bigoplus_{i=0}^{p-1}\Vert\; \mathsf{T}^{mn,i}_n\Big)\Big(\mathrm{I}_p \otimes_\| (\mathrm{I}_{m/p} \otimes \mathbf{DFT}_n)\Big)\Big(\mathrm{I}_p \otimes_\| \mathsf{L}^{mn/p}_{m/p}\Big)\Big((\mathsf{L}^{pn}_p \otimes \mathrm{I}_{m/p\mu}) \otimes_\mu \mathrm{I}_\mu\Big)$$

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

Same Approach for Other Parallel Paradigms

Message Passing: [ISPA 06]

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

$$\ldots$$

$$\ldots$$

$$\ldots$$

$$\rightarrow \underbrace{\left(\mathrm{I}_p \otimes \mathrm{L}_{m/p}^{mn/p}\right)}_{\mathrm{par}(p)} \underbrace{\left(\mathrm{L}_p^{p^2} \otimes \mathrm{I}_{mn/p^2}\right)}_{\mathrm{par}(p\leftarrow q)} \underbrace{\left(\mathrm{I}_q \otimes (\mathrm{I}_{p/q} \otimes \mathrm{L}_p^n \otimes \mathrm{I}_{m/p})\right)}_{\mathrm{par}(q)} \underbrace{\left(\mathrm{I}_q \otimes (\mathrm{I}_{n/q} \otimes \mathrm{DFT}_m)\right)}_{\mathrm{par}(q)}$$

$$\underbrace{\left(\mathrm{I}_q \otimes \mathrm{L}_{m/q}^{mn/q}\right)}_{\mathrm{par}(q)} \underbrace{\left(\mathrm{L}_q^{q^2} \otimes \mathrm{I}_{mn/q^2}\right)}_{\mathrm{par}(q)} \underbrace{\left(\mathrm{I}_q \otimes (\mathrm{L}_q^n \otimes \mathrm{I}_{m/q})\right)}_{\mathrm{par}(q)} \underbrace{\mathrm{T}_n^{mn}}_{\mathrm{par}(q)} \underbrace{\left(\mathrm{I}_q \otimes (\mathrm{I}_{m/q} \otimes \mathrm{DFT}_n)\right)}_{\mathrm{par}(q)}$$

$$\underbrace{\left(\mathrm{I}_q \otimes (\mathrm{I}_{p/q} \otimes \mathrm{L}_{m/p}^{mn/p})\right)}_{\mathrm{par}(q)} \underbrace{\left(\mathrm{L}_p^{p^2} \otimes \mathrm{I}_{mn/p^2}\right)}_{\mathrm{par}(q\leftarrow p)} \underbrace{\left(\mathrm{I}_p \otimes (\mathrm{L}_p^n \otimes \mathrm{I}_{m/p})\right)}_{\mathrm{par}(p)}$$

With Bonelli, Lorenz, Ueberhuber, TU Vienna

Vectorization: [IPDPS 02, VecPar 06]

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

$$\ldots$$

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

$$\ldots$$

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

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

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

Cg/OpenGL for GPUs:

$$\underbrace{\left(\mathbf{DFT}_{r^k}\right)}_{\mathrm{gpu}(t,c)} \rightarrow \underbrace{\left(\prod_{i=0}^{k-1} \mathrm{L}_r^{r^k} \left(\mathrm{I}_{r^{k-1}} \otimes \mathbf{DFT}_r\right)\left(\mathrm{L}_{r^{k-i-1}}^{r^k}(\mathrm{I}_{r^i} \otimes \mathrm{T}_{r^{k-i-1}}^{r^{k-i}})\,\underbrace{\mathrm{L}_{r^{i+1}}^{r^k}}_{\mathrm{vec}(c)}\right)\right)\mathrm{R}_r^{r^k}}_{\mathrm{gpu}(t,c)}$$

$$\ldots$$

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

$$(\mathrm{L}_r^{r^n/2} \mathbin{\vec{\otimes}} \mathrm{I}_2)(\mathrm{I}_{r^{n-1}/2} \otimes_\times \underbrace{\mathrm{L}_r^{2r}}_{\mathrm{shd}(t,c)})(\mathrm{R}_r^{r^{n-1}} \mathbin{\vec{\otimes}} \mathrm{I}_r)$$

With Shen, TU Denmark

Verilog for FPGAs: [DAC 05]

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

$$\ldots$$

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

$$\ldots$$

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

With Milder, Hoe, CMU

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)

Cooley-Tukey FFT in OL: $\mathrm{DFT} \rightarrow (\mathrm{DFT} \otimes \mathrm{I}) \circ D \circ (\mathrm{I} \otimes \mathrm{DFT}) \circ \mathrm{L}\,.$

Viterbi in OL: $\mathrm{Vit} \rightarrow \pi \circ \Big(\prod \big(\mathrm{I} \otimes \mathrm{V}\big) \circ \big(\mathrm{L} \times \mathrm{I}\big)\Big) \circ \big(\mathrm{C} \times \mathrm{C} \times \mathrm{I}\big)$

Mat-Mat-Mult:

$\mathrm{MMM} \rightarrow \mathrm{I} \otimes \mathrm{MMM}$

$\mathrm{MMM} \rightarrow \big(\mathrm{I} \otimes \mathrm{L}\big) \circ \big(\mathrm{MMM} \otimes \mathrm{I}\big) \circ \big(\mathrm{I} \times (\mathrm{I} \otimes \mathrm{L})\big)$

OL Rewriting Rules

- SPL rules reused
- Only few OL-specific rules required

$$\underbrace{\left(\mathrm{I}_k \otimes \mathrm{L}^{mn}_n\right)}_{\mathrm{smp}(p,\mu)} \circ \underbrace{\mathrm{L}^{kmn}_{km}}_{\mathrm{smp}(p,\mu)} \rightarrow \left(\mathrm{L}^{kn}_k \otimes \mathrm{I}_{m/\mu}\right) \bar{\otimes} \mathrm{I}_\mu$$

$$\underbrace{\mathrm{L}^{kmn}_n}_{\mathrm{smp}(p,\mu)} \circ \underbrace{\left(\mathrm{I}_k \otimes \mathrm{L}^{mn}_m\right)}_{\mathrm{smp}(p,\mu)} \rightarrow \left(\mathrm{L}^{kn}_n \otimes \mathrm{I}_{m/\mu}\right) \bar{\otimes} \mathrm{I}_\mu$$

$$\underbrace{\mathrm{A} \circ \mathrm{B}}_{\mathrm{smp}(p,\mu)} \rightarrow \underbrace{\mathrm{A}}_{\mathrm{smp}(p,\mu)} \circ \underbrace{\mathrm{B}}_{\mathrm{smp}(p,\mu)}$$

$$\underbrace{\mathrm{A}^{k \times m \rightarrow n} \otimes \mathrm{I}^{1 \times p \rightarrow p}}_{\mathrm{smp}(p,\mu)} \rightarrow \underbrace{\mathrm{L}^{pn}_n}_{\mathrm{smp}(p,\mu)} \circ \left(\mathrm{I}_{1 \times p \rightarrow p} \otimes_{\|} \mathrm{A}^{k \times m \rightarrow n}\right) \circ \underbrace{\left(\mathrm{I}_k \times \mathrm{L}^{pm}_p\right)}_{\mathrm{smp}(p,\mu)}$$

$$\underbrace{\left(\mathrm{A} \times \mathrm{B}\right)}_{\mathrm{smp}(p,\mu)} \circ \underbrace{\left(\mathrm{C} \times \mathrm{D}\right)}_{\mathrm{smp}(p,\mu)} \rightarrow \underbrace{\left(\mathrm{A} \circ \mathrm{C}\right)}_{\mathrm{smp}(p,\mu)} \times \underbrace{\left(\mathrm{B} \circ \mathrm{D}\right)}_{\mathrm{smp}(p,\mu)}$$

New OL rules

Example: Viterbi Decoder in OL

- **Viterbi decoder (forward part) as operator**

$$\mathsf{Vit}^{e,f,\mathbf{x}}_{m,n,N} : \mathbb{R}^{nN} \rightarrow \mathbb{R}^{2^m} \times \mathbb{N}^{2^m n}$$

- **Viterbi kernel (butterfly)**

$$\mathsf{V}^{e,f}_{i,j} : \mathbb{R}^2 \times \mathbb{R}^{2n} \times \mathbb{R}^{nN} \rightarrow \mathbb{R}^2 \times \mathbb{R}^{2n} \times \mathbb{R}^{nN}; \left(\mathbf{x}, \mathbf{d}, \mathbf{c}\right) \mapsto \left(\mathbf{y}, \mathbf{d}', \mathbf{c}\right) \quad , \quad 0 \leq i < n,\ 0 \leq j < 2^{m-1}$$

- **Viterbi algorithm as breakdown rule**

$$\mathsf{Vit}^{e,f,x}_{m,n,N} \rightarrow \pi_{(\mathbf{x},\mathbf{d})} \circ \left(\prod_{i=0}^{n-1} \left(\mathrm{I}_{2^{m-1} \times 2^{m-1} \times 1} \otimes_j \mathsf{V}^{e,f}_{i,j} \right) \circ \left(\mathsf{L}^{2^m}_{2^{m-1}} \times \mathrm{I}_{2^m n \times nN} \right) \right) \circ \left(\mathsf{C}_{\mathbf{x}} \times \mathsf{C}_{\mathbf{0}} \times \mathrm{I}_{nN} \right)$$

First non-transform supported by Spiral

Viterbi: Vectorization Through Rewriting

$$\underbrace{\mathsf{Vit}^{e,f,x}_{m,n,N}}_{\mathsf{vec}(\nu)} \rightarrow \underbrace{\pi_{(\mathbf{x},\mathbf{d})} \circ \left(\prod_{i=0}^{n-1} \left(\mathrm{I}_{2^{m-1}\times 2^{m-1}\times 1} \otimes_j \mathsf{V}^{e,f}_{i,j}\right) \circ \left(\mathsf{L}^{2^m}_{2^{m-1}} \times \mathrm{I}_{2^m n \times nN}\right)\right) \circ \left(\mathsf{C}_{\mathbf{x}} \times \mathsf{C}_{\mathbf{0}} \times \mathrm{I}_{nN}\right)}_{\mathsf{vec}(\nu)}$$

$$\rightarrow \pi_{(\mathbf{x},\mathbf{d})} \circ \left(\prod_{i=0}^{n-1} \underbrace{\left(\mathrm{I}_{2^{m-1}\times 2^{m-1}\times 1} \otimes_j \mathsf{V}^{e,f}_{i,j}\right) \circ \left(\mathsf{L}^{2^m}_{2^{m-1}} \times \mathrm{I}_{2^m n \times nN}\right)}_{\mathsf{vec}(\nu)}\right) \circ \left(\mathsf{C}_{\mathbf{x}} \times \mathsf{C}_{\mathbf{0}} \times \mathrm{I}_{nN}\right)$$

$$\ldots$$

$$\rightarrow \pi_{(\mathbf{x},\mathbf{d})} \circ \left(\prod_{i=0}^{n-1} \left(\mathrm{I}_{2^{m-1}/\nu\times 2^{m-1}\times 1} \otimes_j \left(\left(\underbrace{\mathsf{L}^{2\nu}_{\nu}}_{\mathsf{reg}(\nu)} \times \mathrm{I}_{2m\times nN}\right) \circ \left(\underline{\mathsf{V}^{e,f}_{i,4j+k} \vec{\otimes}_k \mathrm{I}_{\nu\times 1\times 1}}\right)\right)\right) \circ \left(\left(\underline{\mathsf{L}^{2^m}_{2^{m-1}/\nu} \vec{\otimes} \mathrm{I}_\nu}\right) \times \mathrm{I}_{2^m n\times nN}\right)\right) \circ \left(\mathsf{C}_{\mathbf{x}} \times \mathsf{C}_{\mathbf{0}} \times \mathrm{I}_{nN}\right)$$

Sufficient to vectorize one input

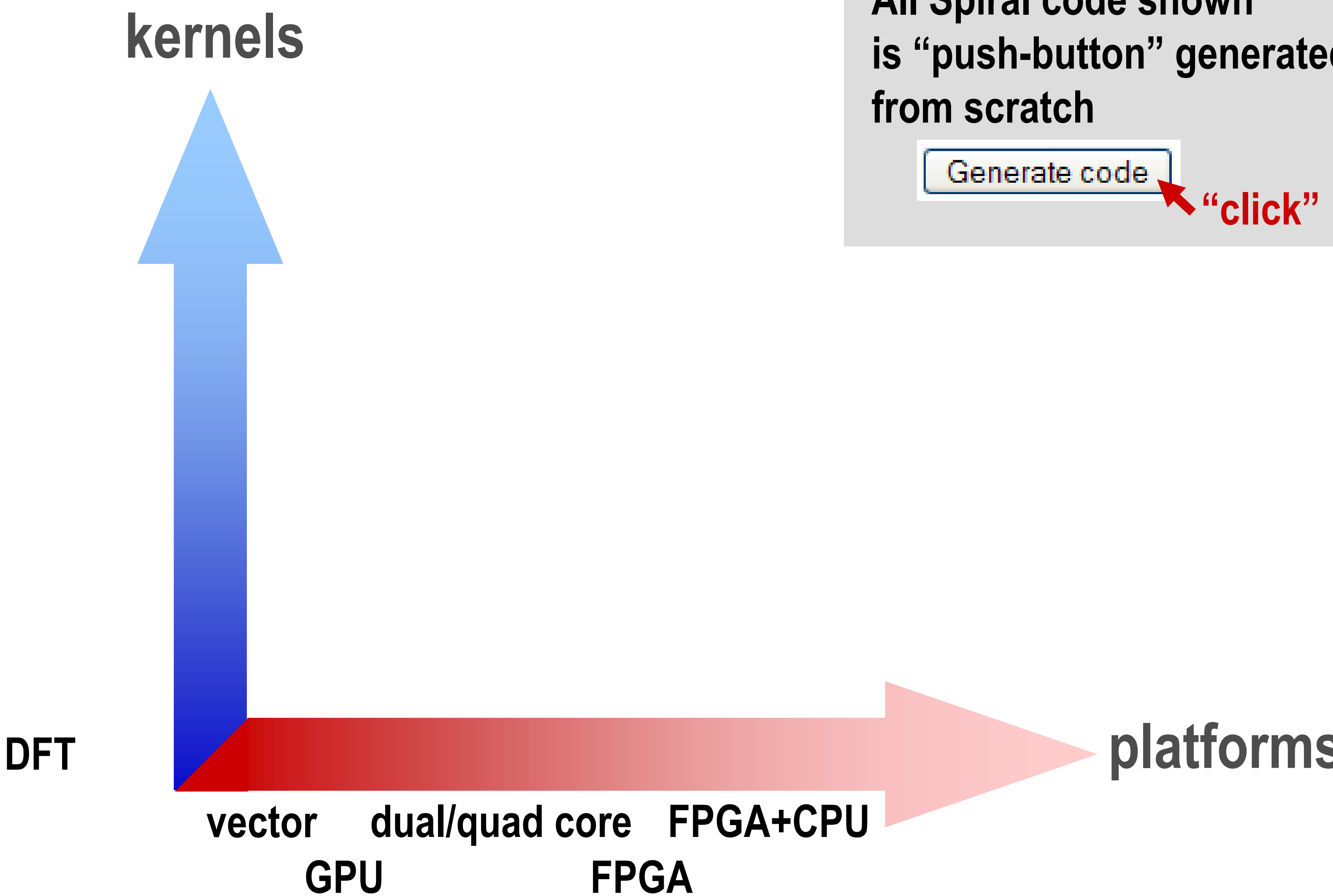
Vectorized kernel

In-register shuffle operation

Organization

- Spiral overview
- Parallelization in Spiral
- **Results**
- Concluding remarks

Benchmarks

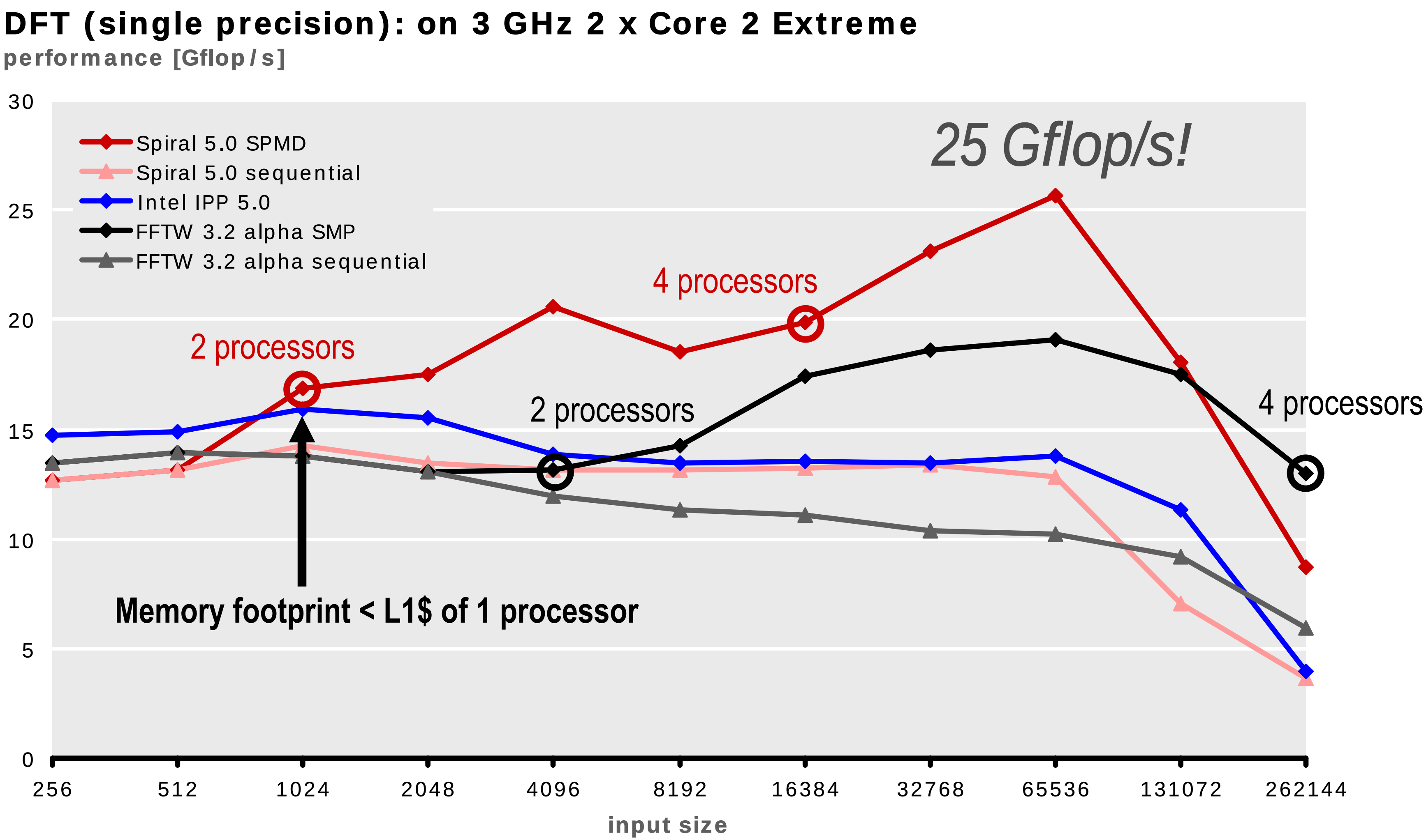


All Spiral code shown
is "push-button" generated
from scratch

Generate code

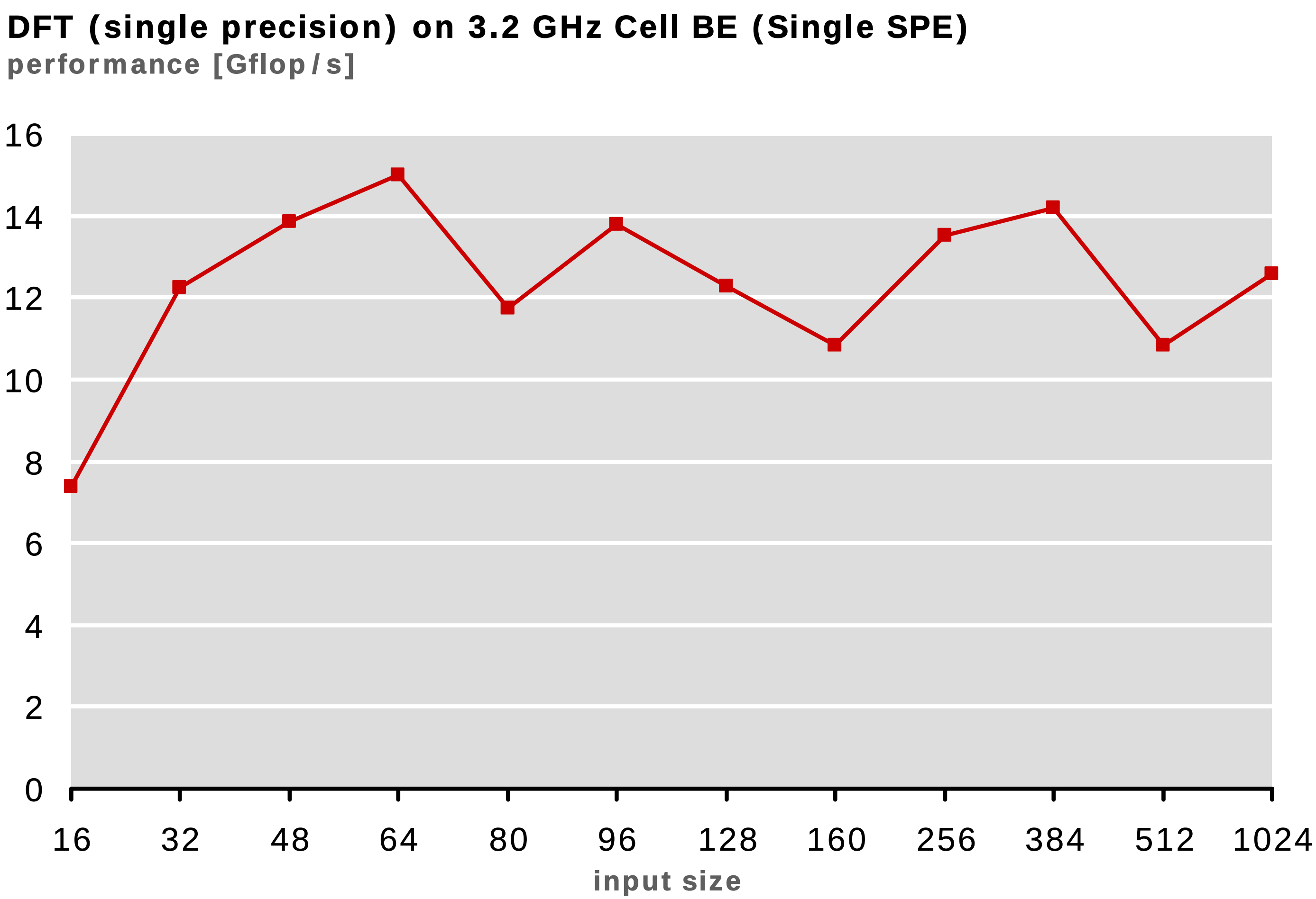
"click"

Benchmark: Vector and SMP



4-way vectorized + up to 4-threaded + adapted to the memory hierarchy

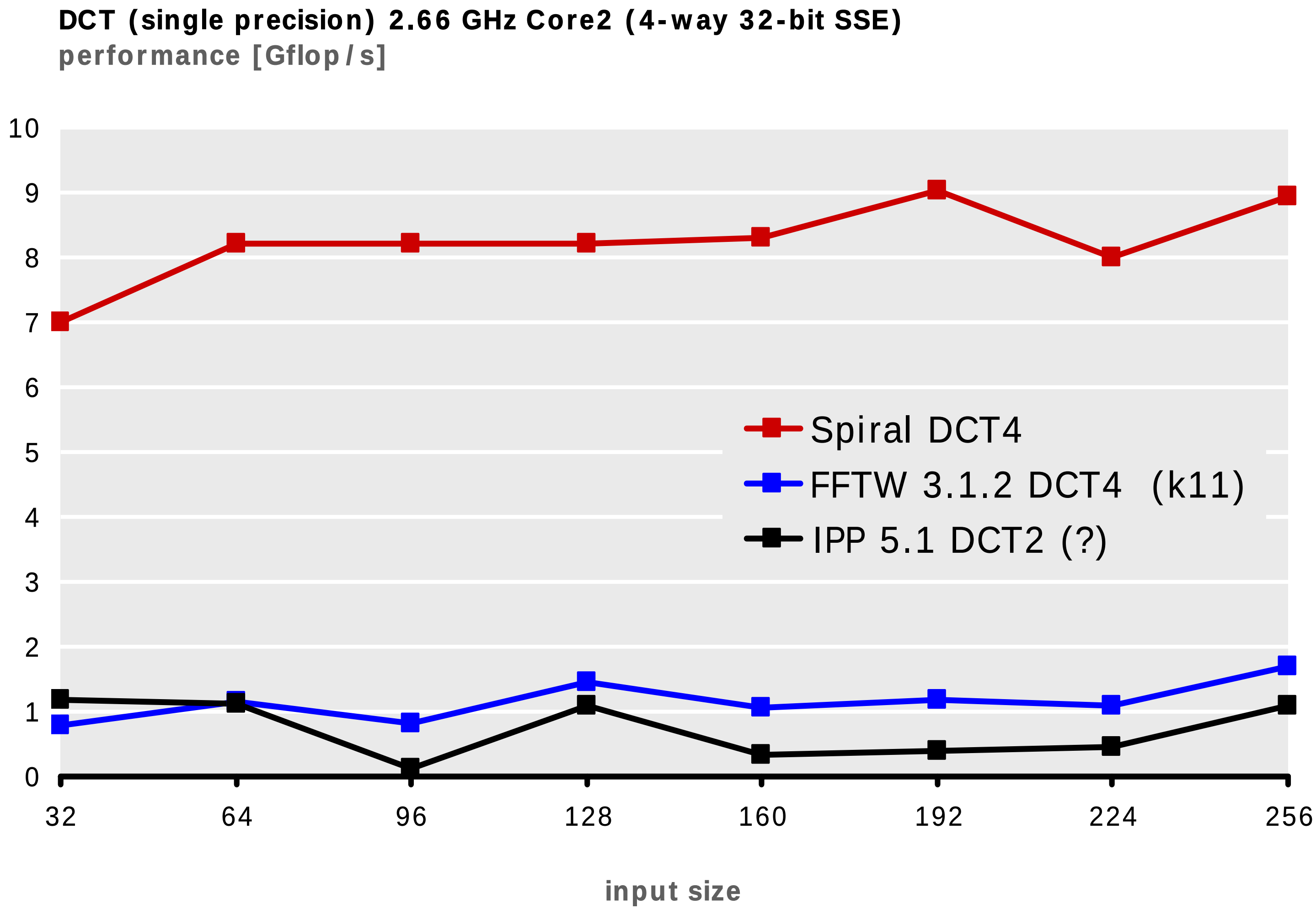
Benchmark: Cell (1 processor = SPE)



Generated using the simulator; run at Mercury (thanks to Robert Cooper)

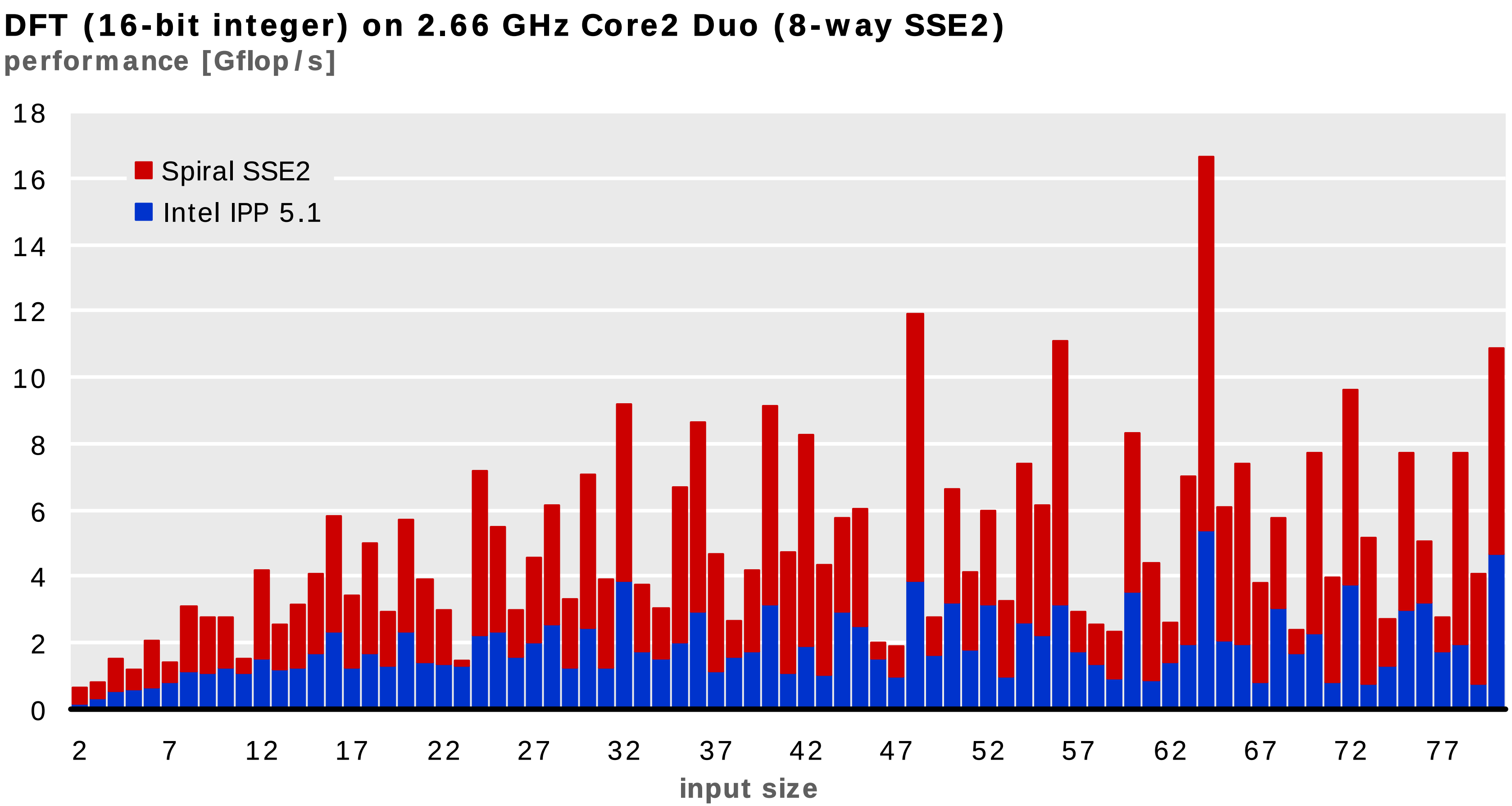
Joint work with Th. Peter (ETH Zurich), S. Chellappa, M. Telgarsky, J. Moura (CMU)

DCT4, Multiples of 32: 4-way Vectorized



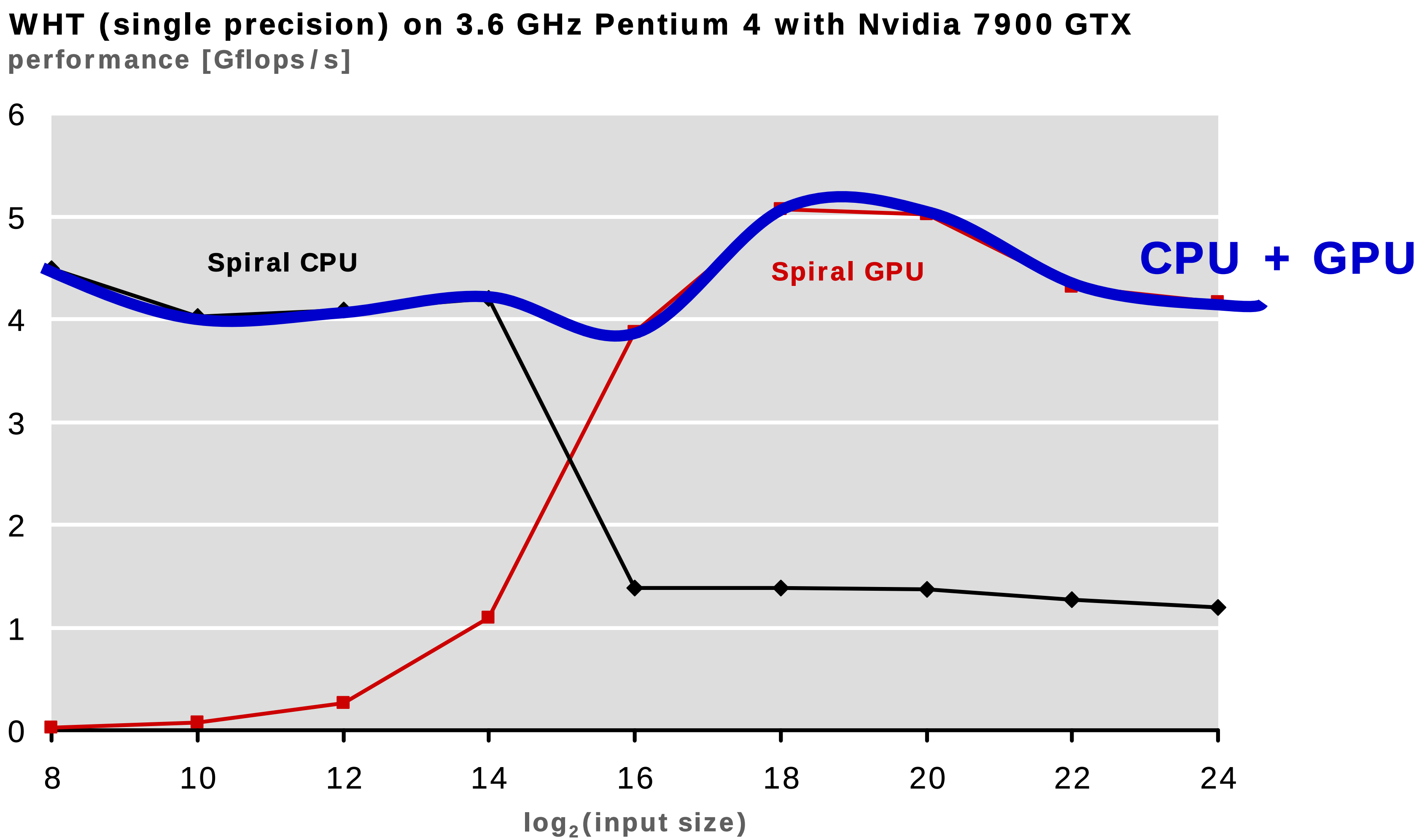
- *novel algorithm (algebraic algorithm theory)*

DFT, 8-way Vectorized: All Sizes Up To 80



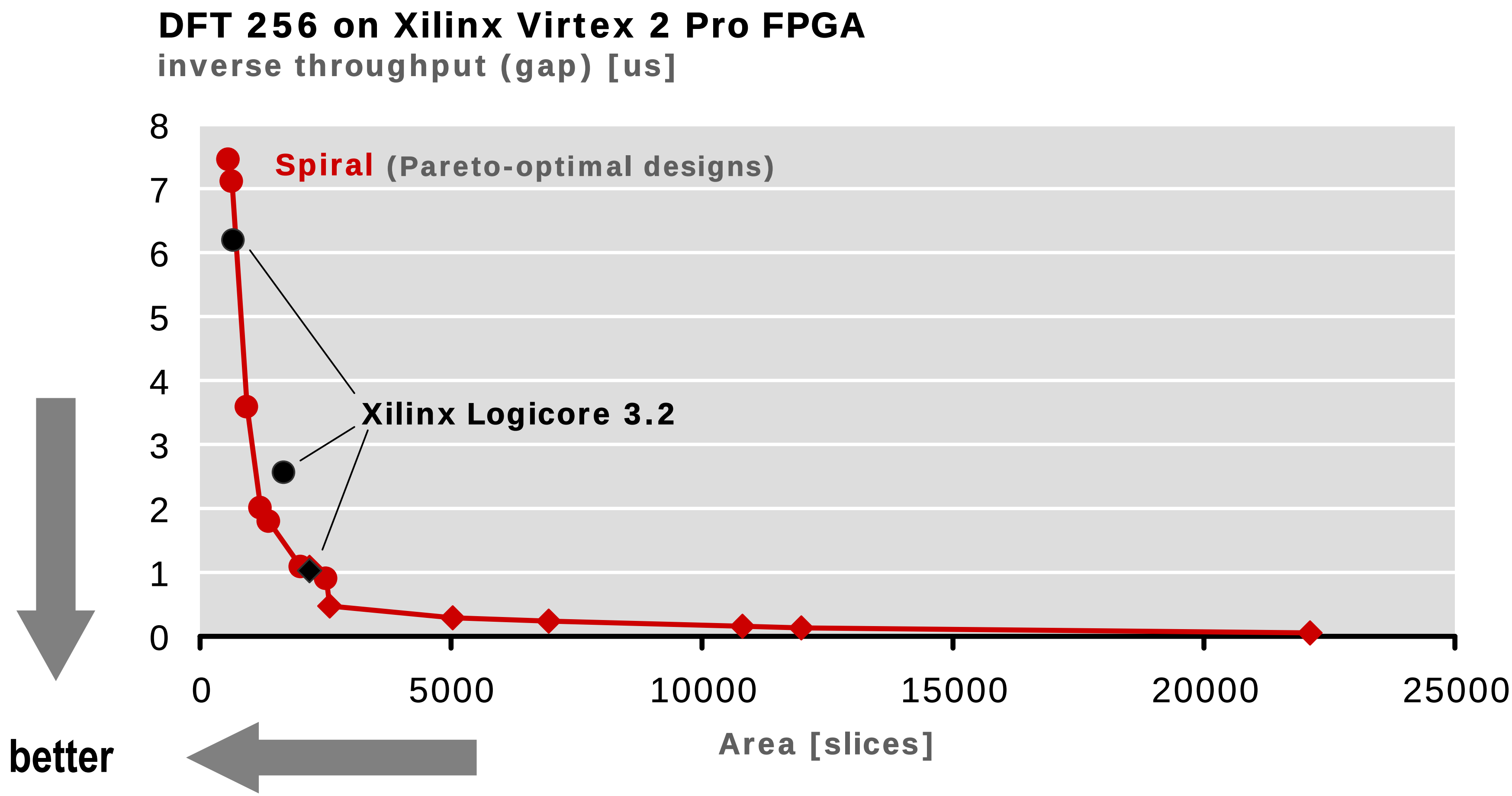
- *first 8-way DFTs for all sizes*
- *arbitrary vector length /arbitrary DFT sizes in principle solved*

Benchmark: GPU



Joint work with H. Shen, TU Denmark

Benchmark: FPGA

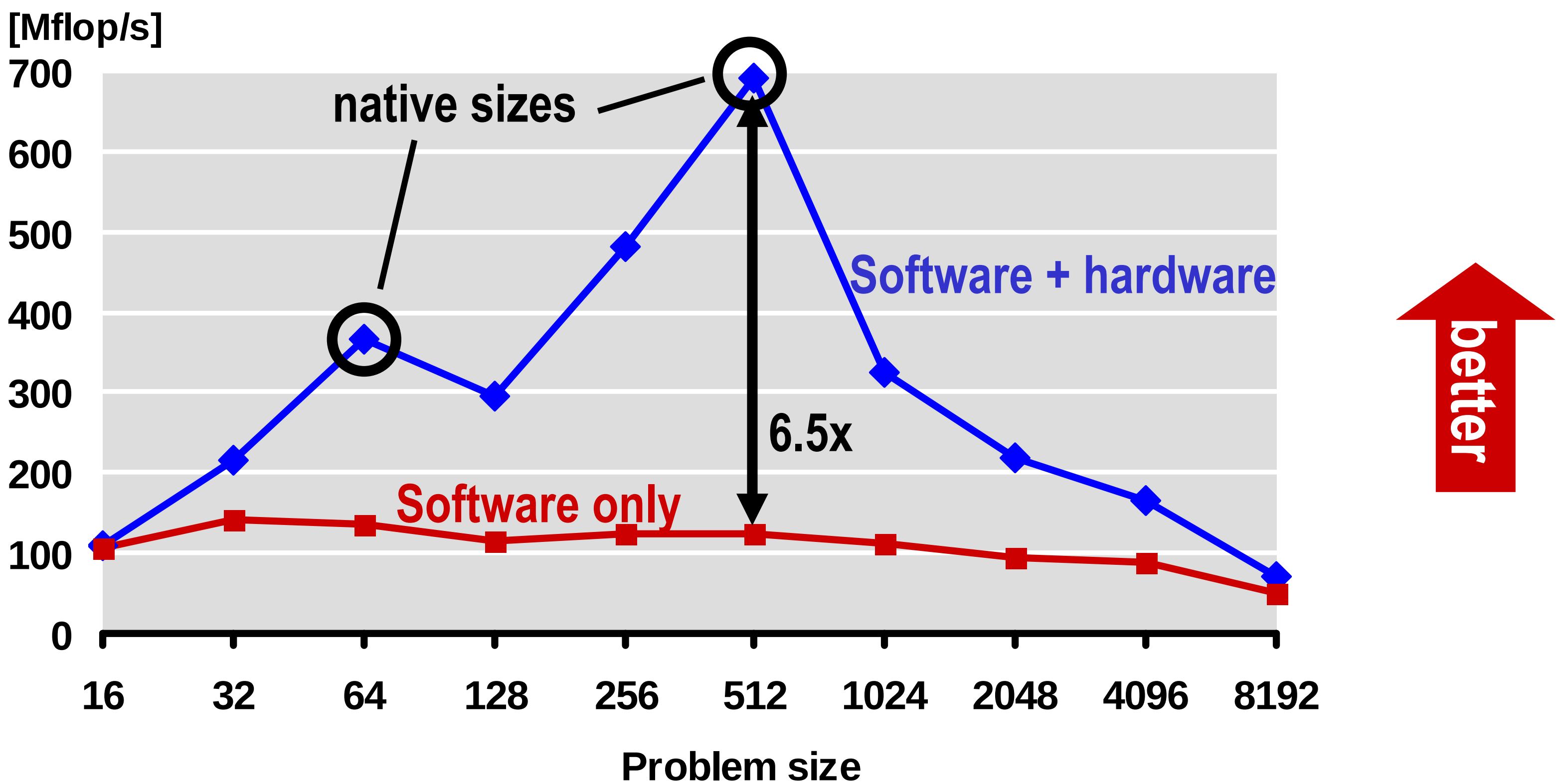


- competitive with professional designs
- much larger set of performance/area trade-offs

Joint work with P. Milder, J. Hoe (CMU)

Benchmark: Hardware Accelerator (FPGA)

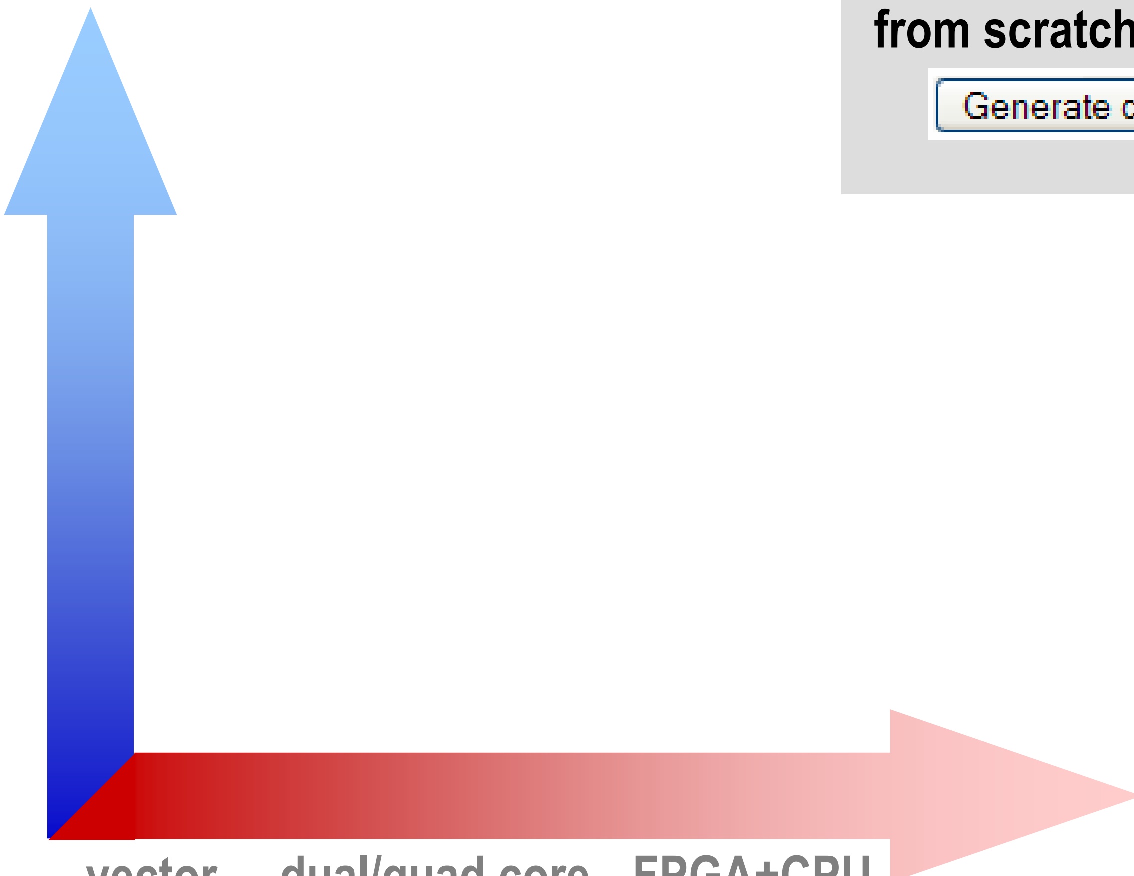
Xilinx Virtex 2 Pro FPGA: 1M gates @ 100 MHz + 2 PowerPC 405 @ 300 MHz



Fixed set of accelerators speed up a whole library

Joint work with P. D'Alberto (Yahoo), A. Sandryhaila, P. Milder, J. Hoe, J . M. F. Moura (CMU), J. Johnson (Drexel)

Benchmarks

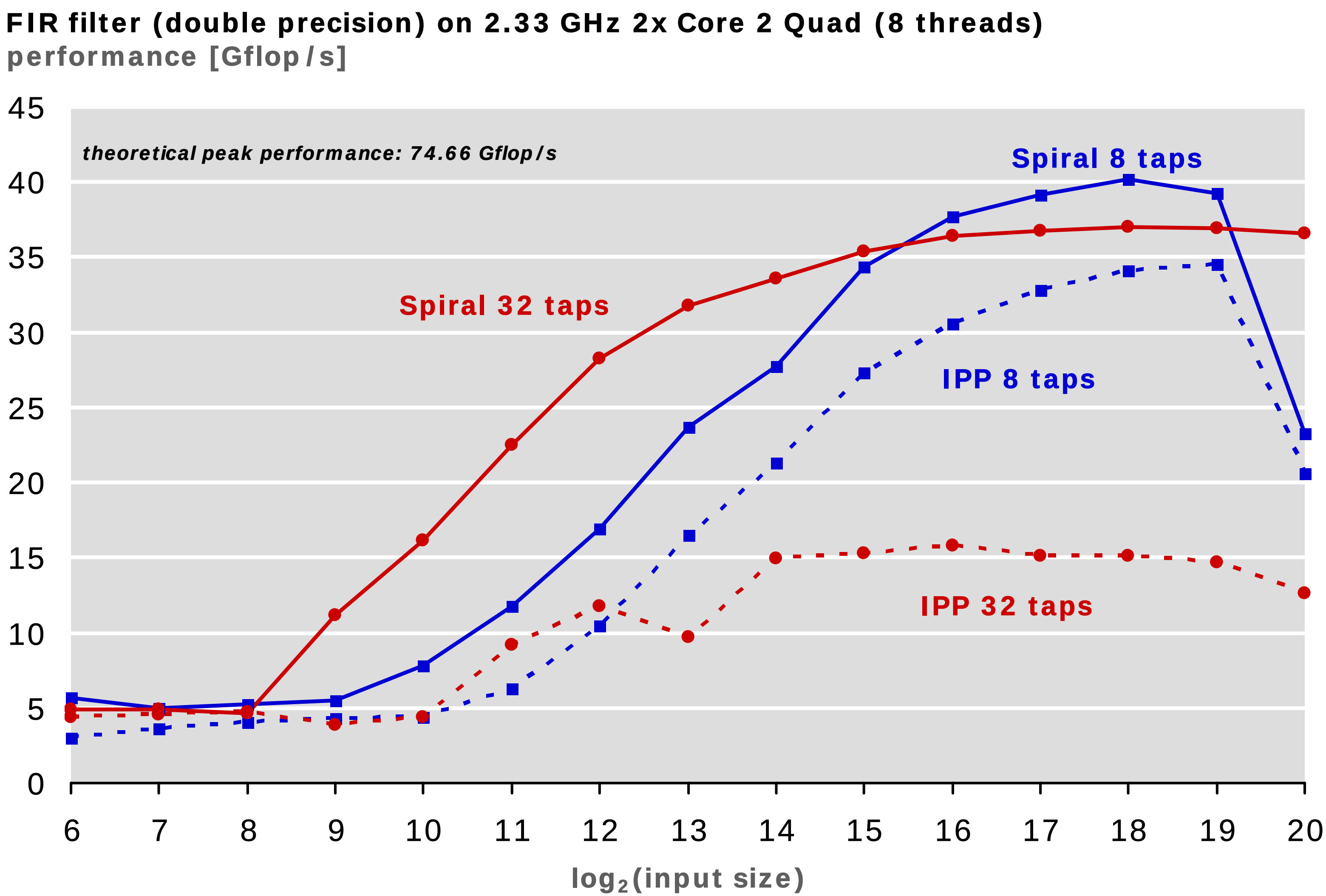


All Spiral code shown is "push-button" generated from scratch

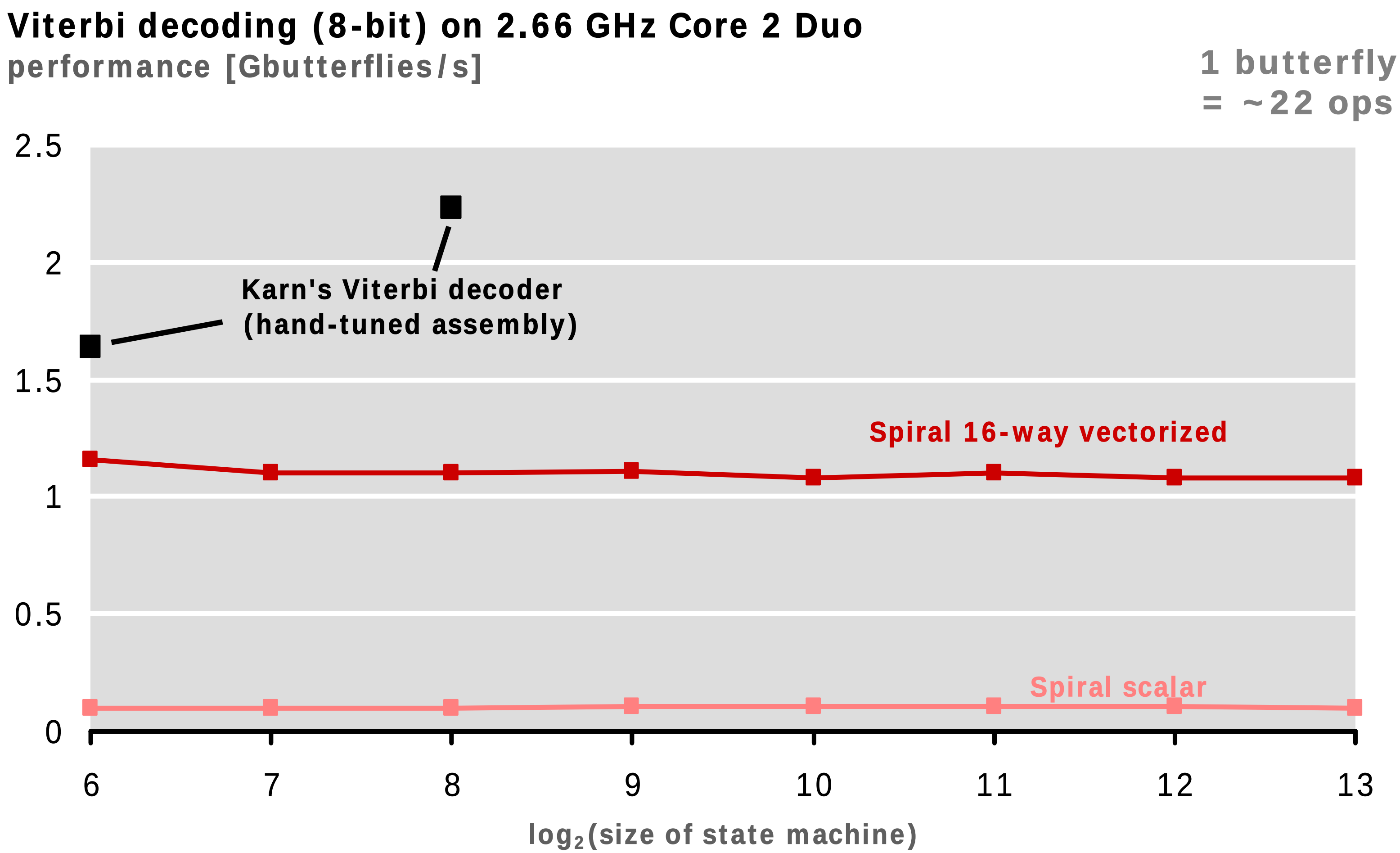
Generate code

"click"

Benchmark: Finite Impulse Response Filter



Beyond Transforms : Viterbi Decoding

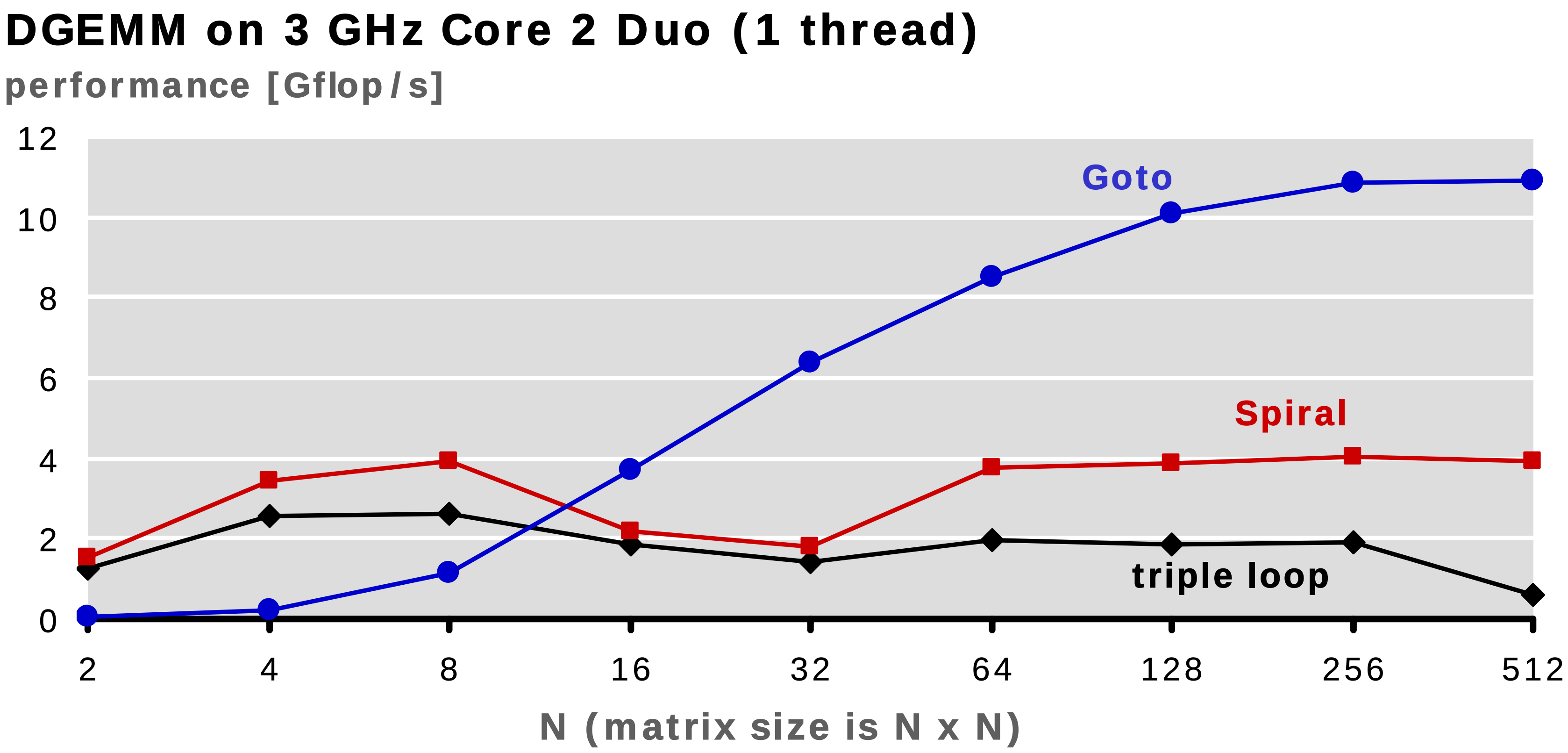


Vectorized using practically the same rules as for DFT

Joint work with S. Chellappa, CMU

Karn: http://www.ka9q.net/code/fec/

First Results: Matrix-Matrix-Multiply



work with F. de Mesmay, CMU

Organization

- Spiral overview
- Parallelization in Spiral
- Results
- Concluding remarks

Conclusions

- **Automatic generation of very fast and fastest numerical kernels is possible and desirable**

- **High level language and approach**
  - Algorithm generation
  - Algorithm optimization

- **Same approach for loop optimization, different forms of parallelism, SW and HW implementations**

Spiral Web Interface @spiral.net (beta version)

1. Select platform

2. Select functionality

3. Generate code

"click"

Program Generation Interface collapse

Target platform for optimization: 2x Intel Xeon 3.6 GHz with 2048K L2 cache

parameter	value	explanation
Transform	DCT2 (Discrete Cosine Transform, type 2)	The transform for which you want to request C code
Data type	double	The data type of input and output vector
Transform size	6	The size of the transform = the length of the input vector
Optimize for	runtime	What you want to optimize the code for
Search method	Dynamic Programming	The search method SPIRAL uses (Dynamic Programming is a good choice)
Compiler profile	gcc -O3	Compiler and compiler options used for compilation

Generate code

Browse Archive expand

Filter by Platform: All Platforms Selected

Filter by Transform: All Transforms Selected

Filter by Size: All Sizes Selected

Query Database