

Generating Parallel Transforms Using Spiral

Franz Franchetti
Yevgen Voronenko
Markus Püschel

Electrical and
Computer Engineering

Carnegie Mellon University

Part of the Spiral Team



Sponsors: DARPA DESA program, NSF-NGS/ITR, NSF-ACR, and Intel

Organization

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

Spiral

- Generates programs **from a problem specification** for linear transforms (DFT, DCT, DWT, filters,)
- Optimizes at the **algorithmic level** and at the code level
- Different code types supported:
scalar, fixed point, **vector, parallel, Verilog**
- **Goal 1:** A flexible push-button program generation framework for an entire domain of algorithms
- **Goal 2:** With new architectures, update the tool rather than the individual programs in the library

How Spiral Works

Spiral:

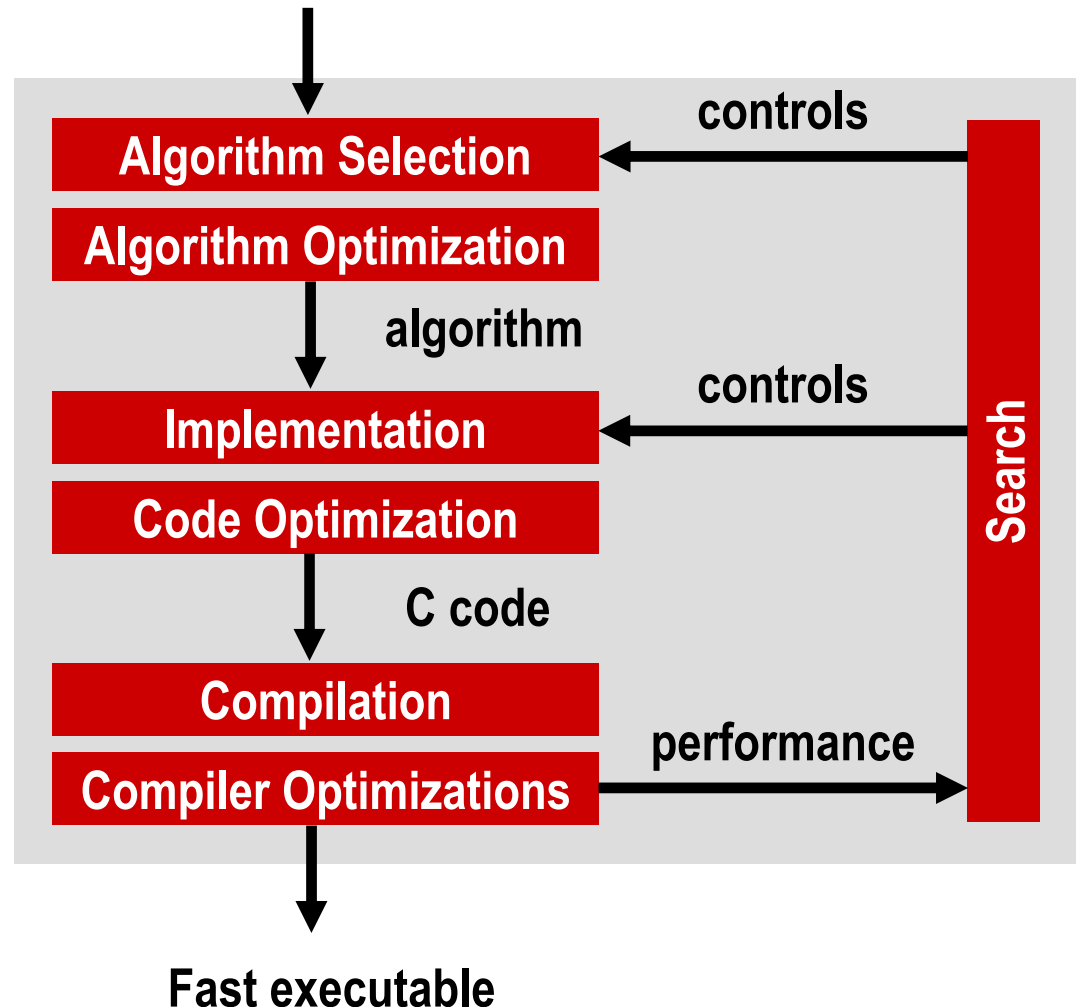
Complete automation of the implementation and optimization task

Basic idea:

Declarative representation of algorithms, rewriting systems to generate and optimize algorithms

automation

Problem specification (transform)



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)

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

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 (currently 55)

$$\text{DCT-2}_n = \left[\cos(k(2l + 1)\pi/2n) \right]_{0 \leq k, l < n},$$

$$\text{DCT-3}_n = \text{DCT-2}_n^T \quad (\text{transpose}),$$

$$\text{DCT-4}_n = \left[\cos((2k + 1)(2l + 1)\pi/4n) \right]_{0 \leq k, l < n},$$

$$\text{IMDCT}_n = \left[\cos((2k + 1)(2l + 1 + n)\pi/4n) \right]_{0 \leq k < 2n, 0 \leq l < n},$$

$$\text{RDFT}_n = [r_{kl}]_{0 \leq k, l < n}, \quad r_{kl} = \begin{cases} \cos \frac{2\pi k l}{n}, & k \leq \lfloor \frac{n}{2} \rfloor \\ -\sin \frac{2\pi k l}{n}, & k > \lfloor \frac{n}{2} \rfloor \end{cases},$$

$$\text{WHT}_n = \begin{bmatrix} \text{WHT}_{n/2} & \text{WHT}_{n/2} \\ \text{WHT}_{n/2} & -\text{WHT}_{n/2} \end{bmatrix}, \quad \text{WHT}_2 = \text{DFT}_2,$$

$$\text{DHT} = \left[\cos(2kl\pi/n) + \sin(2kl\pi/n) \right]_{0 \leq k, l < n}.$$

Examples: Breakdown Rules (currently ≈ 220)

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

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

$$\text{DFT}_p \rightarrow R_p^T (\text{I}_1 \oplus \text{DFT}_{p-1}) D_p (\text{I}_1 \oplus \text{DFT}_{p-1}) R_p, \quad p \text{ prime}$$

$$\text{DCT-3}_n \rightarrow (\text{I}_m \oplus \text{J}_m) \text{L}_m^n (\text{DCT-3}_m(1/4) \oplus \text{DCT-3}_m(3/4))$$

$$\cdot (\text{F}_2 \otimes \text{I}_m) \begin{bmatrix} \text{I}_m & 0 \oplus -\text{J}_{m-1} \\ \frac{1}{\sqrt{2}}(\text{I}_1 \oplus 2\text{I}_m) \end{bmatrix}, \quad n = 2m$$

$$\text{DCT-4}_n \rightarrow S_n \text{DCT-2}_n \text{diag}_{0 \leq k < n} (1 / (2 \cos((2k + 1)\pi / 4n)))$$

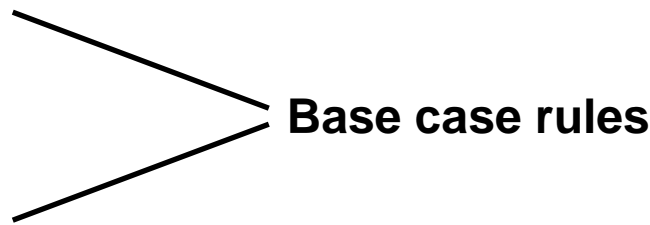
$$\text{IMDCT}_{2m} \rightarrow (\text{J}_m \oplus \text{I}_m \oplus \text{I}_m \oplus \text{J}_m) \left(\left(\begin{bmatrix} 1 \\ -1 \end{bmatrix} \otimes \text{I}_m \right) \oplus \left(\begin{bmatrix} -1 \\ -1 \end{bmatrix} \otimes \text{I}_m \right) \right) \text{J}_{2m} \text{DCT-4}_{2m}$$

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

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

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

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



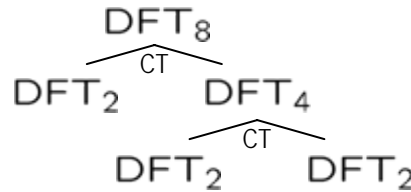
Program Generation (Simplified)

Transform:

DFT₈



Ruletree:



$$\text{DFT}_8 \rightarrow (\text{DFT}_2 \otimes I_4) \cdot D \cdot (I_2 \otimes \text{DFT}_4) \cdot P$$

$$\text{DFT}_4 \rightarrow (\text{DFT}_2 \otimes I_2) \cdot D \cdot (I_2 \otimes \text{DFT}_2) \cdot P$$



SPL Formula: $(\text{DFT}_2 \otimes I_4) T_4^8 \left(I_2 \otimes \left((\text{DFT}_2 \otimes I_2) T_2^4 (I_2 \otimes \text{DFT}_2) L_2^4 \right) \right) L_2^8$



Σ-SPL: $\sum_{j=0}^3 (S_j \text{DFT}_2 G_j) \sum_{k=0}^1 \left(\sum_{l=0}^1 (S_{k,l} \text{diag}(t_{k,l}) \text{DFT}_2 G_l) \sum_{m=0}^1 (S_m \text{diag}(t_m) \text{DFT}_2 G_{k,m}) \right)$



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;
}
  
```

SPL to Sequential Code

SPL construct	code
$y = (A_n B_n)x$	<pre>t[0:1:n-1] = B(x[0:1:n-1]); y[0:1:n-1] = A(t[0:1:n-1]);</pre>
$y = (I_m \otimes A_n)x$	<pre>for (i=0;i<m;i++) y[i*n:1:i*n+n-1] = A(x[i*n:1:i*n+n-1])</pre>
$y = (A_m \otimes I_n)x$	<pre>for (i=0;i<m;i++) y[i:n:i+m-1] = A(x[i:n:i+m-1]);</pre>
$y = \left(\bigoplus_{i=0}^{m-1} A_n^i\right)x$	<pre>for (i=0;i<m;i++) y[i*n:1:i*n+n-1] = A(i, x[i*n:1:i*n+n-1]);</pre>
$y = D_{m,n}x$	<pre>for (i=0;i<m*n;i++) y[i] = Dmn[i]*x[i];</pre>
$y = L_m^{mn}x$	<pre>for (i=0;i<m;i++) for (j=0;j<n;j++) y[i+m*j]=x[n*i+j];</pre>

Example: tensor product

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

Correct code: easy; fast code: very difficult

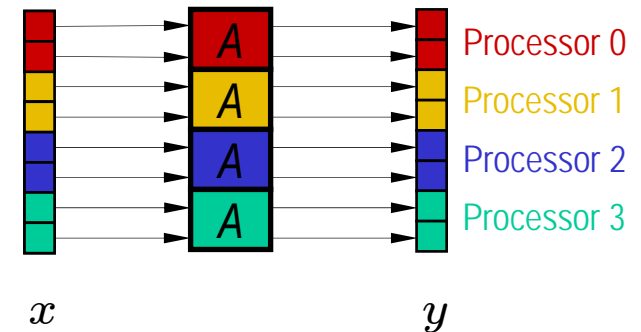
Organization

- Spiral overview
- **Parallelization in Spiral (focus on shared memory)**
- Results
- Concluding remarks

SPL to Shared Memory Code: Basic Idea

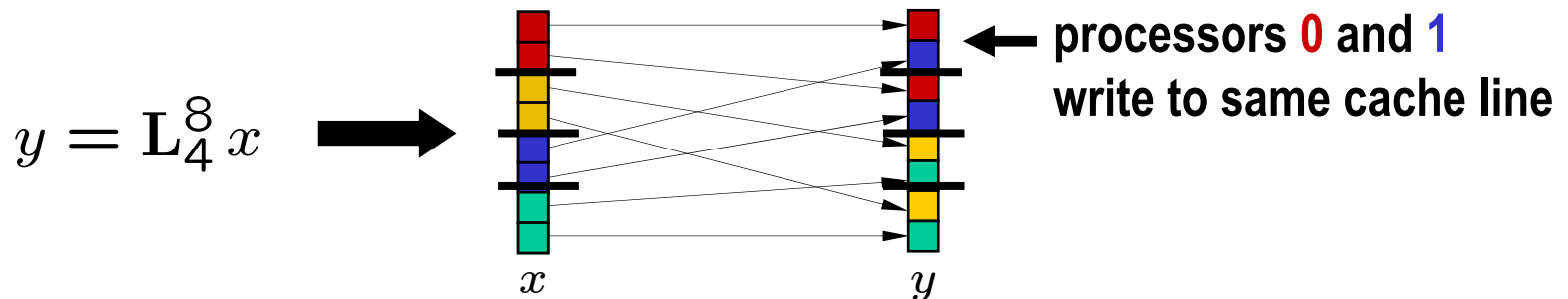
- Good construct: tensor product

$$y = \left(I_p \otimes A \right) x$$



p-way embarrassingly parallel, load-balanced

- Problematic construct: permutations produce false sharing



**Task: Rewrite formulas to
extract tensor product + make cache lines atomic**

Step 1: Shared Memory Tags

- **Identify crucial hardware parameters = coarse hardware abstraction**
 - Number of processors: p
 - Cache line size: μ
- **Introduce them as tags in SPL:**

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

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

- **Tags express hardware constraints within the rewriting system**

Step 2: Identify “Good” Formulas

- Load balanced, avoiding false sharing

$$y = (I_p \otimes A)x \quad \text{with} \quad A \in \mathbb{C}^{m\mu \times m\mu}$$

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

$$y = (P \otimes I_\mu)x \quad \text{with} \quad P \text{ a permutation matrix}$$

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

$$I_m \otimes A \quad \text{or} \quad AB$$

where A and B are fully optimized.

Step 3: Identify Rewriting Rules

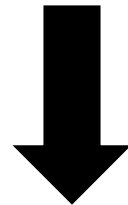
■ Goal: Transform formulas into fully optimized formulas

- Formulas rewritten, tags propagated
- There may be choices

$$\begin{aligned}
 \underbrace{AB}_{\text{smp}(p,\mu)} &\rightarrow \underbrace{A}_{\text{smp}(p,\mu)} \underbrace{B}_{\text{smp}(p,\mu)} \\
 \underbrace{A_m \otimes I_n}_{\text{smp}(p,\mu)} &\rightarrow \underbrace{\left(L_m^{mp} \otimes I_{n/p} \right) \left(I_p \otimes (A_m \otimes I_{n/p}) \right) \left(L_p^{mp} \otimes I_{n/p} \right)}_{\text{smp}(p,\mu)} \\
 \underbrace{L_m^{mn}}_{\text{smp}(p,\mu)} &\rightarrow \begin{cases} \underbrace{\left(I_p \otimes L_{m/p}^{mn/p} \right)}_{\text{smp}(p,\mu)} \underbrace{\left(L_p^{pn} \otimes I_{m/p} \right)}_{\text{smp}(p,\mu)} \\ \underbrace{\left(L_m^{pm} \otimes I_{n/p} \right)}_{\text{smp}(p,\mu)} \underbrace{\left(I_p \otimes L_m^{mn/p} \right)}_{\text{smp}(p,\mu)} \end{cases} \\
 \underbrace{I_m \otimes A_n}_{\text{smp}(p,\mu)} &\rightarrow I_p \otimes_{\parallel} \left(I_{m/p} \otimes A_n \right) \\
 \underbrace{(P \otimes I_n)}_{\text{smp}(p,\mu)} &\rightarrow \left(P \otimes I_{n/\mu} \right) \bar{\otimes} I_\mu
 \end{aligned}$$

Simple Rewriting Example

$$\underbrace{A_m \otimes I_n}_{\text{smp}(p, \mu)}$$



Loop tiling and scheduling
hardware-conscious (knows p and μ)

$$\left(L_m^{mp} \otimes I_{n/p} \right) \left(I_p \otimes_{||} (A_m \otimes I_{n/p}) \right) \left(L_p^{mp} \otimes I_{n/p} \right)$$

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 for Shared Memory by Rewriting [SC 06]

Hardware parameters: p processors, cache line length μ

$$\begin{aligned}
 \underbrace{\text{DFT}_{mn}}_{\text{smp}(p,\mu)} &\rightarrow \underbrace{\left((\text{DFT}_m \otimes \text{I}_n) \text{T}_n^{mn} (\text{I}_m \otimes \text{DFT}_n) \text{L}_m^{mn} \right)}_{\text{smp}(p,\mu)} \\
 \dots & \\
 &\rightarrow \underbrace{\left(\text{DFT}_m \otimes \text{I}_n \right)}_{\text{smp}(p,\mu)} \underbrace{\text{T}_n^{mn}}_{\text{smp}(p,\mu)} \underbrace{\left(\text{I}_m \otimes \text{DFT}_n \right)}_{\text{smp}(p,\mu)} \underbrace{\text{L}_m^{nm}}_{\text{smp}(p,\mu)} \\
 \dots & \\
 &\rightarrow \underbrace{\left((\text{L}_m^{mp} \otimes \text{I}_{n/p\mu}) \bar{\otimes} \text{I}_\mu \right)}_{\text{red}} \underbrace{\left(\text{I}_p \otimes_{\parallel} (\text{DFT}_m \otimes \text{I}_{n/p}) \right)}_{\text{blue}} \underbrace{\left((\text{L}_p^{mp} \otimes \text{I}_{n/p\mu}) \bar{\otimes} \text{I}_\mu \right)}_{\text{red}} \\
 &\quad \underbrace{\left(\bigoplus_{i=0}^{p-1} \text{T}_n^{mn,i} \right)}_{\text{blue}} \underbrace{\left(\text{I}_p \otimes_{\parallel} (\text{I}_{m/p} \otimes \text{DFT}_n) \right)}_{\text{blue}} \underbrace{\left(\text{I}_p \otimes_{\parallel} \text{L}_{m/p}^{mn/p} \right)}_{\text{blue}} \underbrace{\left((\text{L}_p^{pn} \otimes \text{I}_{m/p\mu}) \bar{\otimes} \text{I}_\mu \right)}_{\text{red}}
 \end{aligned}$$

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

Vectorization by Rewriting [IPDPS 02, VecPar 06]

Hardware parameter: vector length ν

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

Fully vectorized

Parallelization for Distributed Memory [ISPA 06]

Hardware parameter: p processors

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

Fully parallelized

Parallelization for GPU – Ongoing Work

Hardware parameter: c-way vectors (pixel format), shader computes t pixels

$$\underbrace{\left(\text{DFT}_{r^k} \right)}_{\text{gpu}(t,c)} \rightarrow \underbrace{\left(\prod_{i=0}^{k-1} L_r^{r^k} \left(I_{r^{k-1}} \otimes \text{DFT}_r \right) \left(L_r^{r^k} \left(I_{r^{k-i-1}} \otimes T_{r^{k-i-1}} \right) \underbrace{L_r^{r^k}}_{\text{vec}(c)} \right) \right)}_{\text{gpu}(t,c)} R_r^{r^k}$$

...

$$\rightarrow \left(\prod_{i=0}^{k-1} \left(L_r^{r^{n/2}} \vec{\otimes} I_2 \right) \left(I_{r^{n-1}/2} \otimes \times \underbrace{\left(\text{DFT}_r \vec{\otimes} I_2 \right) L_r^{2r}}_{\text{shd}(t,c)} \right) T_i \right) \left(L_r^{r^{n/2}} \vec{\otimes} I_2 \right) \left(I_{r^{n-1}/2} \otimes \times \underbrace{L_r^{2r}}_{\text{shd}(t,c)} \right) \left(R_r^{r^{n-1}} \vec{\otimes} I_r \right)$$

Fully optimized

Parallelization/Streaming for FPGAs [DAC05, FPGA06]

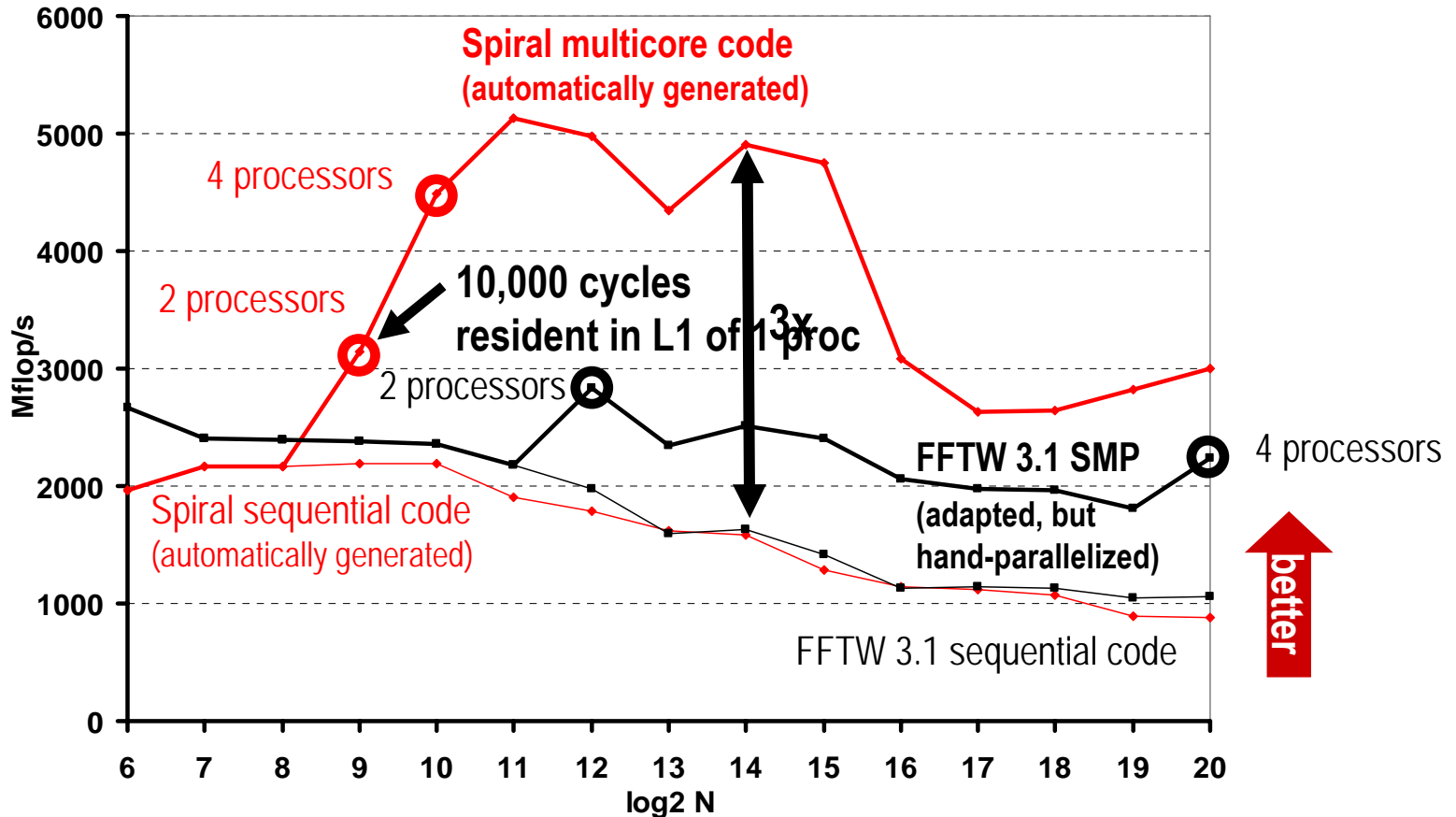
Hardware parameter: streaming width r^s

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

Organization

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

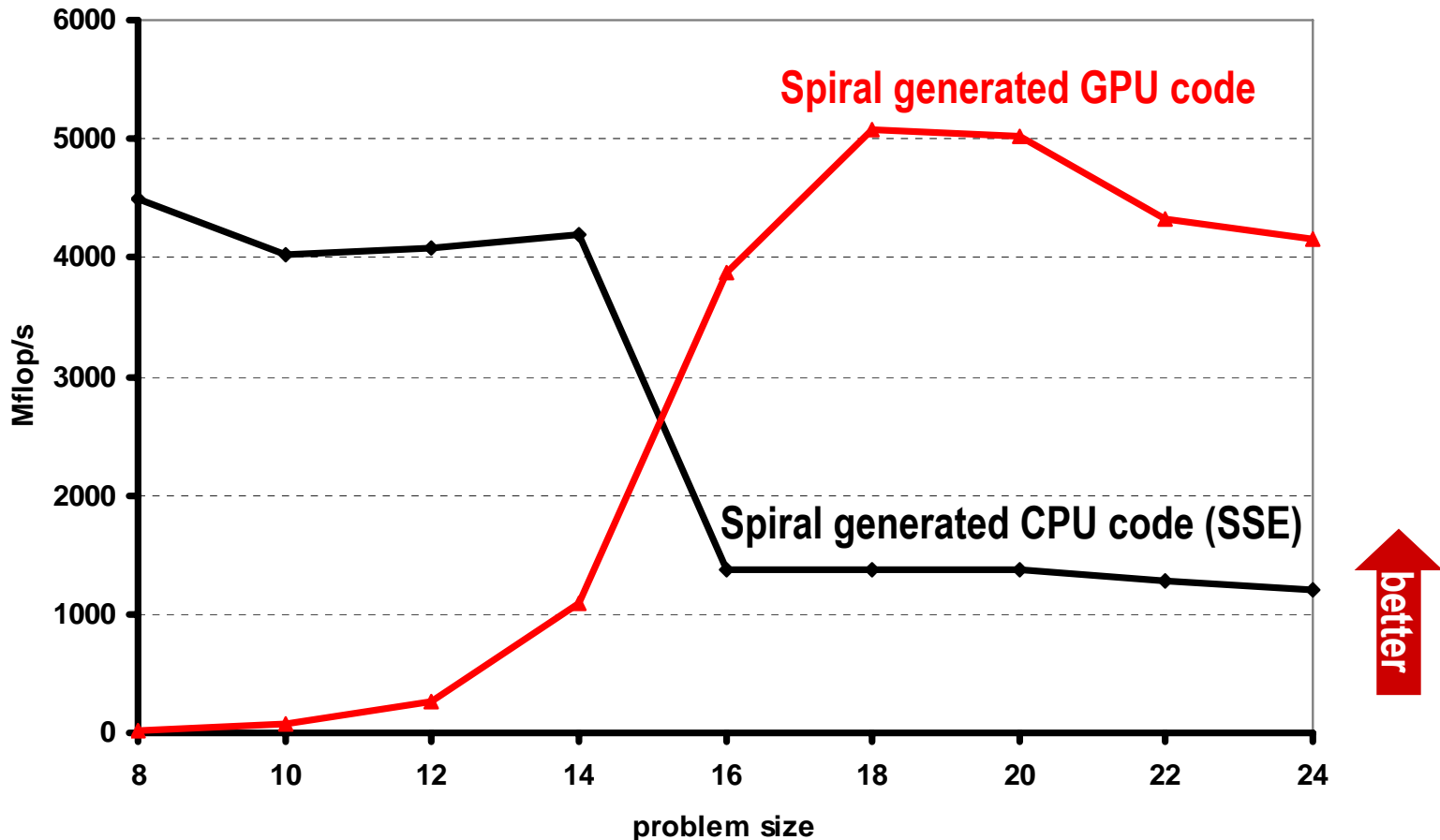
Benchmark: Generated Multicore Code



Complex double 1D DFT on 2.2 GHz Opteron Dual-core (4 processors)

Spiral parallelizes much earlier as FFTW on multicore CPUs

Benchmark: GPU vs. CPU



Walsh-Hadamard transform (float) on 3.6 GHz Pentium 4 + Nvidia 7900 GTX

Use GPU to overcome memory bottleneck?

Organization

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

Conclusions

- **Spiral: The computer implements and optimizes transforms**
 - From problem specification to very fast code---automatically
- **High level, mathematical, declarative language and approach**
 - Rules represent the algorithm knowledge
 - Rewriting systems to generate and optimize algorithms at a high level of abstraction
 - Same approach for different type of parallelism, software and hardware
 - Very extensible
- **We have ideas how to generalize the approach beyond transforms**