Formal Loop Merging for Signal Transforms

Franz Franchetti

Yevgen S. Voronenko

Markus Püschel

Department of Electrical & Computer Engineering

Carnegie Mellon University

This work was supported by NSF through awards 0234293, and 0325687.
Franz Franchetti was supported by the Austrian Science Fund FWF,
Erwin Schrödinger Fellowship J2322.

SPIRAL

`http://www.spiral.net`

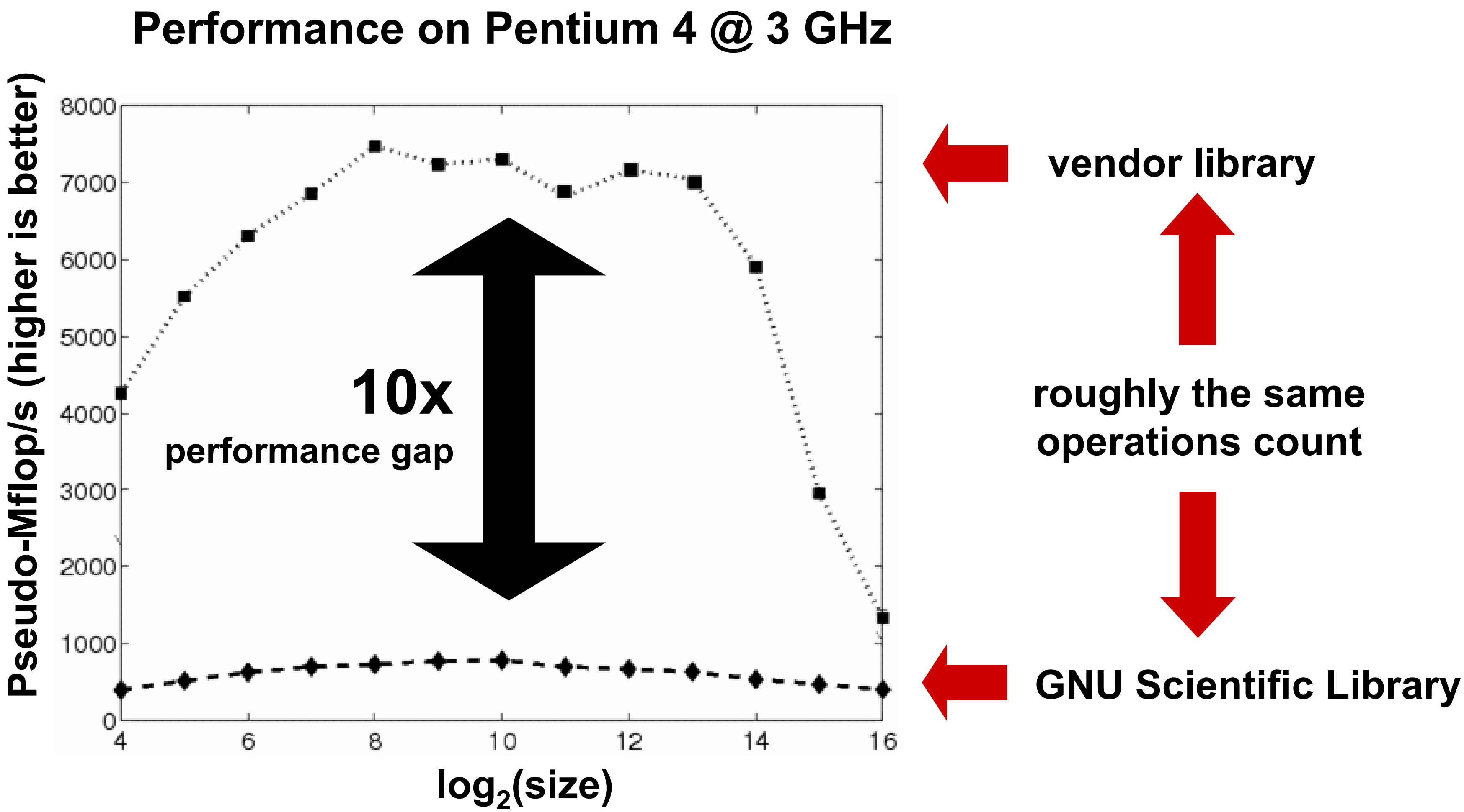
Problem

- **Runtime of (uniprocessor) numerical applications typically dominated by few compute-intensive kernels**
  - Examples: discrete Fourier transform, matrix-matrix multiplication

- **These kernels are hand-written for every architecture (open-source and commercial libraries)**

- **Writing fast numerical code is becoming increasingly difficult, expensive, and platform dependent, due to:**
  - Complicated memory hierarchies
  - Special purpose instructions (short vector extensions, fused multiply-add)
  - Other microarchitectural features (deep pipelines, superscalar execution)

Example: Discrete Fourier Transform (DFT)



Writing fast code is hard. Are there alternatives?

Automatic Code Generation and Adaptation

- **ATLAS:** Code generator for basic linear algebra subroutines (BLAS)
  [Whaley, et. al., 1998] [Yotov, et al., 2005]
- **FFTW:** Adaptive library for computing the discrete Fourier transform (DFT) and its variants
  [Frigo and Johnson, 1998]
- **SPIRAL:** Code generator for linear signal transforms (including DFT)
  [Püschel, et al., 2004]
- **See also:** Proceedings of the IEEE special issue on “Program Generation, Optimization, and Adaptation,” Feb. 2005.

- **Focus of this talk:**
  A new approach to automatic loop merging in SPIRAL

Talk Organization

- **SPIRAL Background**
- **Automatic loop merging in SPIRAL**
- **Experimental Results**
- **Conclusions**

Talk Organization

- **SPIRAL Background**
- Automatic loop merging in SPIRAL
- Experimental Results
- Conclusions

SPIRAL: DSP Transforms

- **SPIRAL generates optimized code for linear signal transforms, such as discrete Fourier transform (DFT), discrete cosine transforms, FIR filters, wavelets, and many others.**

- **Linear transform = matrix-vector product:**

$$y = Mx$$

output **transform matrix** **input**

- **Example: DFT of input vector x**

$$y = \mathrm{DFT}_n\, x$$

$$\mathrm{DFT}_n = \left[\omega_n^{k\ell}\right]_{0\le k,\ell<n}, \qquad \omega_n = e^{-2\pi\sqrt{-1}/n}$$

SPIRAL: Fast Transform Algorithms

- Reduce computation cost from $O(n^2)$ to $O(n \log n)$
- For every transform there are many fast algorithms
- Algorithm = sparse matrix factorization

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \\ 1 & -1 & 1 & -1 \\ 1 & -i & -1 & i \end{pmatrix} x \rightarrow \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & i \end{pmatrix} \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & -1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} x$$

12 adds
4 mults

4 adds **1 mult** **4 adds**

(when multiplied with input vector x)

$$\mathrm{DFT}_4 \rightarrow (\mathrm{DFT}_2 \otimes \mathrm{I}_2) D (\mathrm{I}_2 \otimes \mathrm{DFT}_2) P$$

- SPIRAL generates the space of algorithms using breakdown rules in the domain-specific Signal Processing Language (SPL)

$$\mathrm{DFT}_{mn} \rightarrow (\mathrm{DFT}_m \otimes \mathrm{I}_n) D (\mathrm{I}_m \otimes \mathrm{DFT}_n) P$$

SPL (Signal Processing Language)

- **SPL expresses transform algorithms as structured sparse matrix factorization**

- **Examples:**

$$\mathrm{F}_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

$$A \otimes B = \left[a_{k,\ell} B \right]_{k,\ell}$$

$$A \oplus B = \begin{bmatrix} A & \\ & B \end{bmatrix}$$

$$\mathrm{I} \otimes B = \begin{bmatrix} B & & \\ & \ddots & \\ & & B \end{bmatrix}$$

- **SPL grammar in Backus-Naur form**

$$\begin{array}{rcl} \langle \text{spl} \rangle & ::= & \langle \text{generic} \rangle \mid \langle \text{symbol} \rangle \mid \langle \text{transform} \rangle \mid \\ & & \langle \text{spl} \rangle \cdot \cdots \cdot \langle \text{spl} \rangle \mid \\ & & \langle \text{spl} \rangle \oplus \ldots \oplus \langle \text{spl} \rangle \mid \\ & & \langle \text{spl} \rangle \otimes \cdots \otimes \langle \text{spl} \rangle \mid \\ & & \ldots \\ \langle \text{generic} \rangle & ::= & \mathrm{diag}(a_0, \ldots, a_{n-1}) \mid \ldots \\ \langle \text{symbol} \rangle & ::= & \mathrm{I}_n \mid \mathrm{J}_n \mid \mathrm{L}^n_k \mid \mathrm{R}_\alpha \mid \mathrm{F}_2 \mid \ldots \\ \langle \text{transform} \rangle & ::= & \mathbf{DFT}_n \mid \mathbf{WHT}_n \mid \mathbf{DCT\text{-}2}_n \mid \mathbf{Filt}_n\,(h[z]) \mid \ldots \end{array}$$

SPIRAL

Compiling SPL to Code Using Templates

$$y = \mathrm{L}_n^{mn}\, x$$

```
for i=0..n-1
   for j=0..m-1
      y[i+n*j]=x[m*i+j]
```

$$y = (A_n \oplus B_m)x$$

```
y[0:1:n-1] = call A(x[0:1:n-1])
y[n:1:n+m-1] = call B(x[n:1:n+m-1])
```

$$y = (\mathrm{I}_n \otimes B_m)x$$

```
for i=0..n-1
   y[im:1:im+m-1] = call B(x[im:1:im+m-1])
```

$$y = (\mathrm{I}_n \otimes B_m)\, \mathrm{L}_n^{mn}\, x$$

```
for i=0..n-1
   y[im:1:im+m-1] = call B(x[i:n:i+m-1])
```

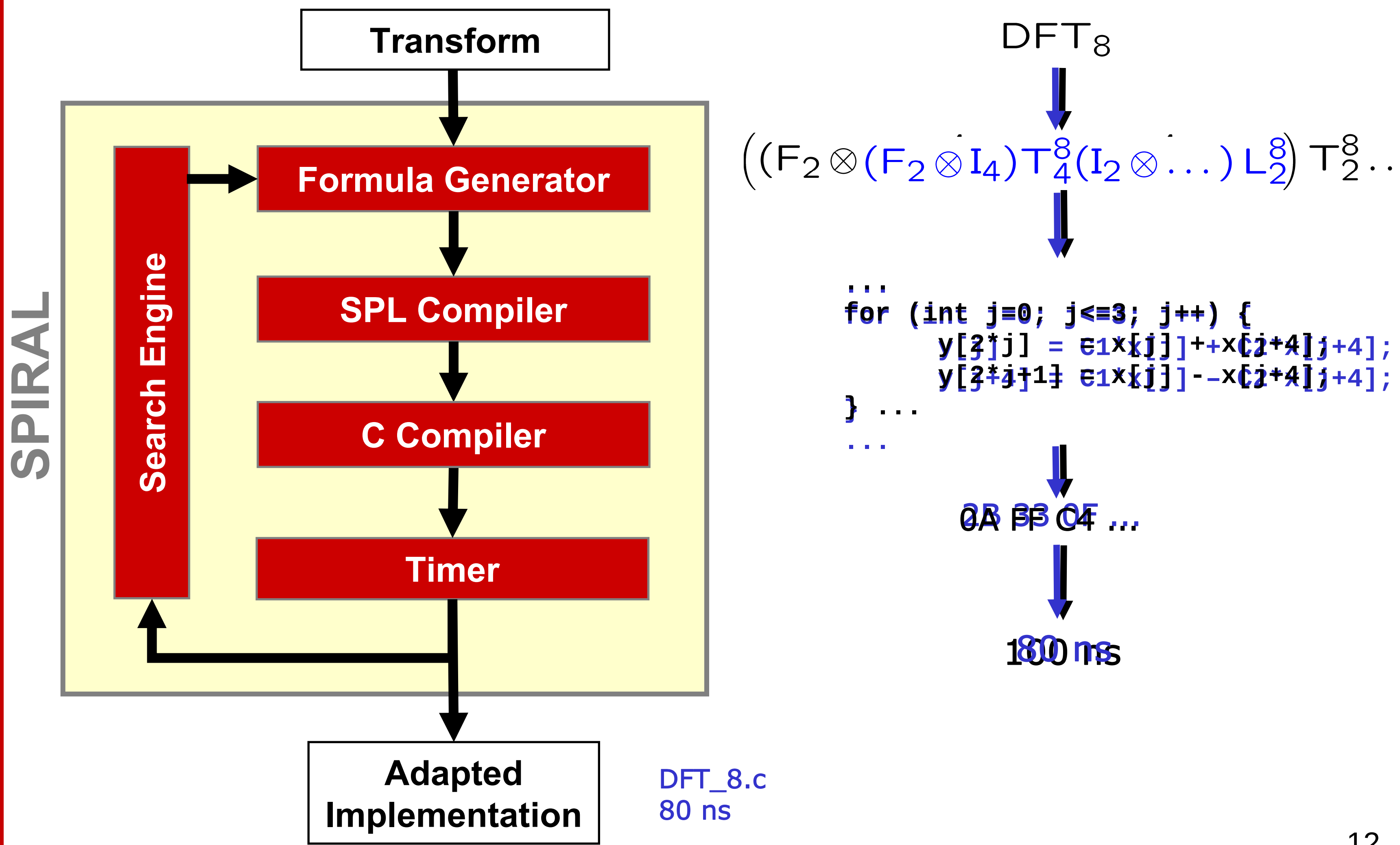
Some Transforms and Breakdown Rules in SPIRAL

Base case rules

Spiral contains 30+ transforms and 100+ rules

SPIRAL Architecture

Approach: Empirical search over alternative recursive algorithms



Problem: Fusing Permutations and Loops

$y = (I_4 \otimes F_2) L^8_4 \, x$

Two passes over the working set
Complex index computation

direct mapping

```
void sub(double *y, double *x) {
   double t[8];
   for (int i=0; i<=7; i++)
      t[(i/4)+2*(i%4)] = x[i];
   for (int i=0; i<4; i++){
      y[2*i] = t[2*i] + t[2*i+1];
      y[2*i+1] = t[2*i] - t[2*i+1];
   }
}
```

C compiler cannot do this

hardcoded special case

One pass over the working set
Simple index computation

```
void sub(double *y, double *x) {
   for (int j=0; j<=3; j++){
      y[2*j] = x[j] + x[j+4];
      y[2*j+1] = x[j] - x[j+4];
   }
}
```

State-of-the-art
SPIRAL: Hardcoded with templates
FFTW: Hardcoded in the infrastructure

How does hardcoding scale?

General Loop Merging Problem

= permutations

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

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

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

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

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

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

- **Combinatorial explosion: Implementing templates for all rules and all recursive combinations is unfeasible**
- **In many cases even theoretically not understood**

Our Solution in SPIRAL

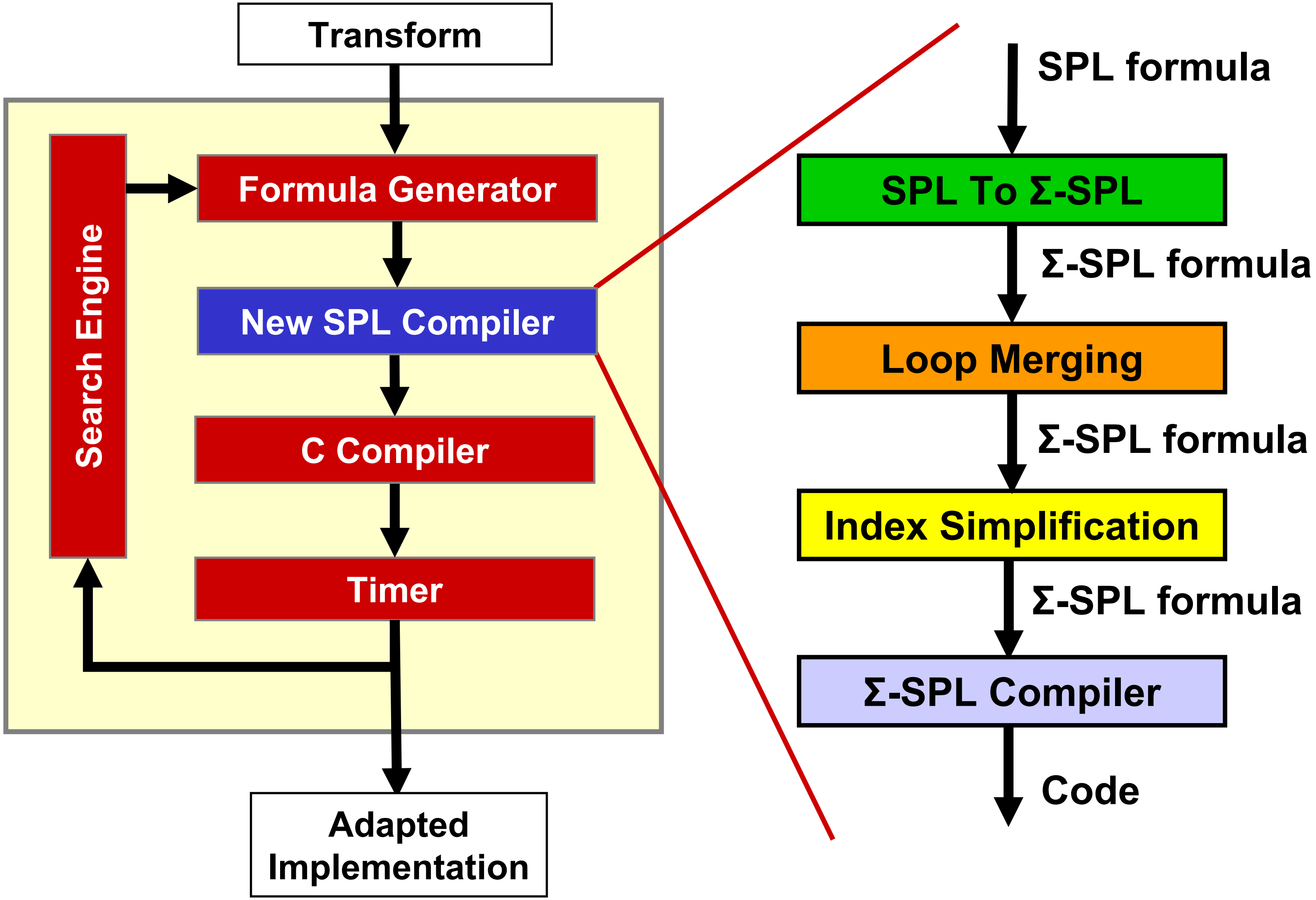
- **Loop merging at C code level: impractical**
- **Loop merging at SPL level: not possible**
- **Solution:**
  - **New language Σ-SPL – an abstraction level between SPL and code**
  - **Loop merging through Σ-SPL formula manipulation**

SPIRAL

Talk Organization

- SPIRAL Background
- **Automatic loop merging in SPIRAL**
- Experimental Results
- Conclusions

New Approach for Loop Merging



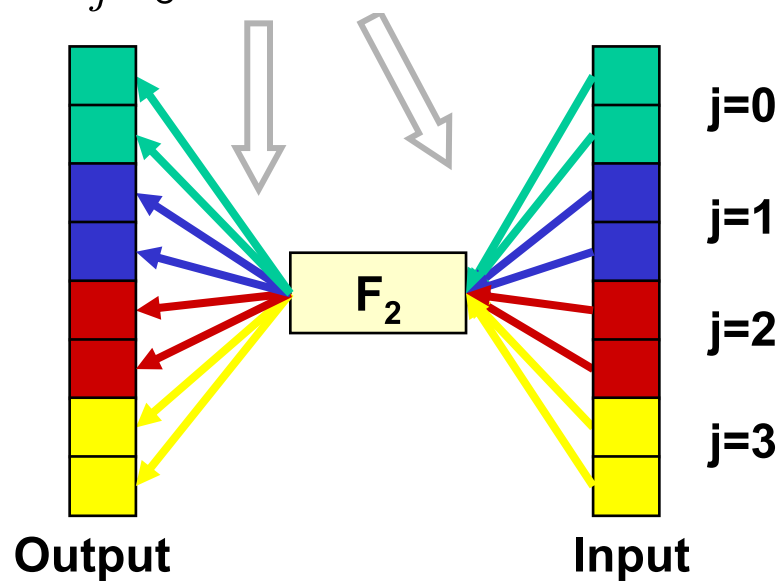
Σ–SPL

- Four central constructs: **Σ, G, S, Perm**
  - Σ (sum) – makes loops explicit
  - G_f (gather) – reads data using the index mapping f
  - S_f (scatter) – writes data using the index mapping f
  - Perm_f – permutes data using the index mapping f

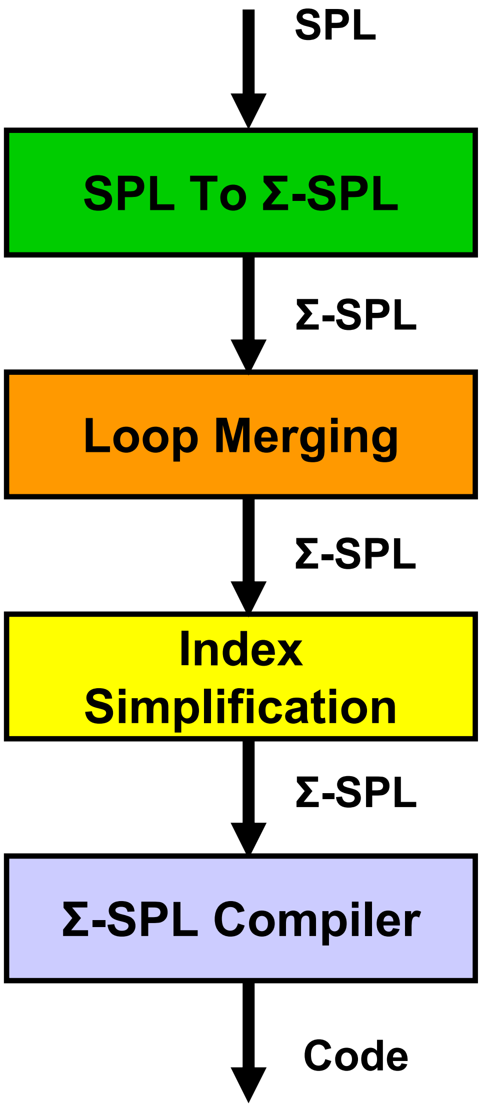
- Every Σ-SPL formula still represents a matrix factorization

Example: $(\mathrm{I}_4 \otimes \mathrm{F}_2) \rightarrow \sum_{j=0}^{3} \mathrm{S}_{f_j}\, \mathrm{F}_2\, \mathrm{G}_{f_j}$

$$\begin{bmatrix} \mathrm{F}_2 & & & \\ & \mathrm{F}_2 & & \\ & & \mathrm{F}_2 & \\ & & & \mathrm{F}_2 \end{bmatrix}$$



Loop Merging With Rewriting Rules



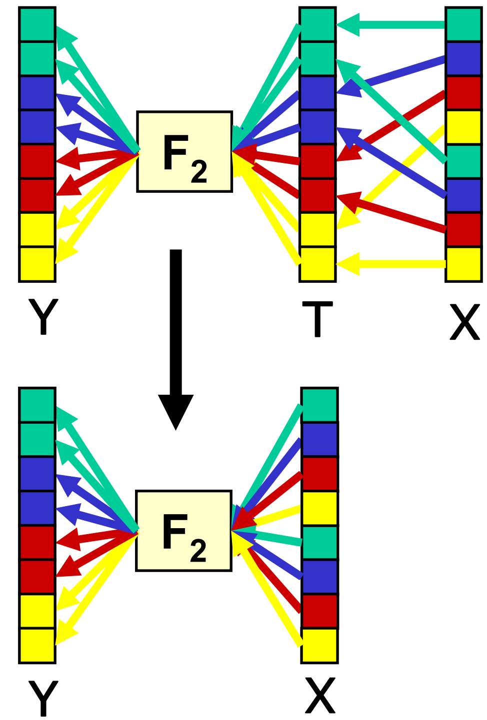
$$y = (I_4 \otimes F_2)\, L^8_4\, x$$

$$\left(\sum_{j=0}^{3} S_{(j)_4\otimes\imath_2}\, F_2\, G_{(j)_4\otimes\imath_2}\right) \mathrm{Perm}_{\ell^8_4}$$

$$\left(\sum_{j=0}^{3} S_{(j)_4\otimes\imath_2}\, F_2\, G_{\ell^8_4\circ((j)_4\otimes\imath_2)}\right)$$

$$\left(\sum_{j=0}^{3} S_{(j)_4\otimes\imath_2}\, F_2\, G_{\imath_2\otimes(j)_4}\right)$$

```
for (int j=0; j<=3; j++) {
        y[2*j]   = x[j] + x[j+4];
        y[2*j+1] = x[j] - x[j+4];
}
```



Rules:

$$G_r\, \mathrm{Perm}_p \;\rightarrow\; G_{p\circ r}$$

$$\ell^{mn}_m \circ \left((j)_m \otimes \imath_n\right) \;\rightarrow\; \imath_n \otimes (j)_m$$

Application: Loop Merging For FFTs

DFT breakdown rules:

Cooley-Tukey FFT $\mathbf{DFT}_{km} \rightarrow (\mathbf{DFT}_k \otimes \mathrm{I}_m)\, \mathsf{T}^{km}_m (\mathrm{I}_k \otimes \mathbf{DFT}_m)\, \mathsf{L}^{km}_k$

Prime factor FFT $\mathbf{DFT}_{km} \rightarrow \mathsf{V}^T_{k,m} (\mathbf{DFT}_k \otimes \mathrm{I}_m)(\mathrm{I}_k \otimes \mathbf{DFT}_m)\, \mathsf{V}_{k,m}$

$\gcd(k, m) = 1$

Rader FFT $\mathbf{DFT}_p \rightarrow \mathsf{W}^T_p (\mathrm{I}_1 \oplus \mathbf{DFT}_{p-1}) D_p (\mathrm{I}_1 \oplus \mathbf{DFT}_{p-1})\, \mathsf{W}_p$

p - prime

Index mapping functions are non-trivial:

$\mathsf{L}^{km}_k \rightarrow \mathsf{Perm}_{\ell^{km}_k}$ $\qquad \ell^{km}_k(i) = \left\lfloor \frac{i}{m} \right\rfloor + k(i \bmod m)$

$\mathsf{V}_{k,m} \rightarrow \mathsf{Perm}_{v_{k,m}}$ $\qquad v_{k,m}(i) = \left(m \left\lfloor \frac{i}{m} \right\rfloor + k(i \bmod m) \right) \bmod km$

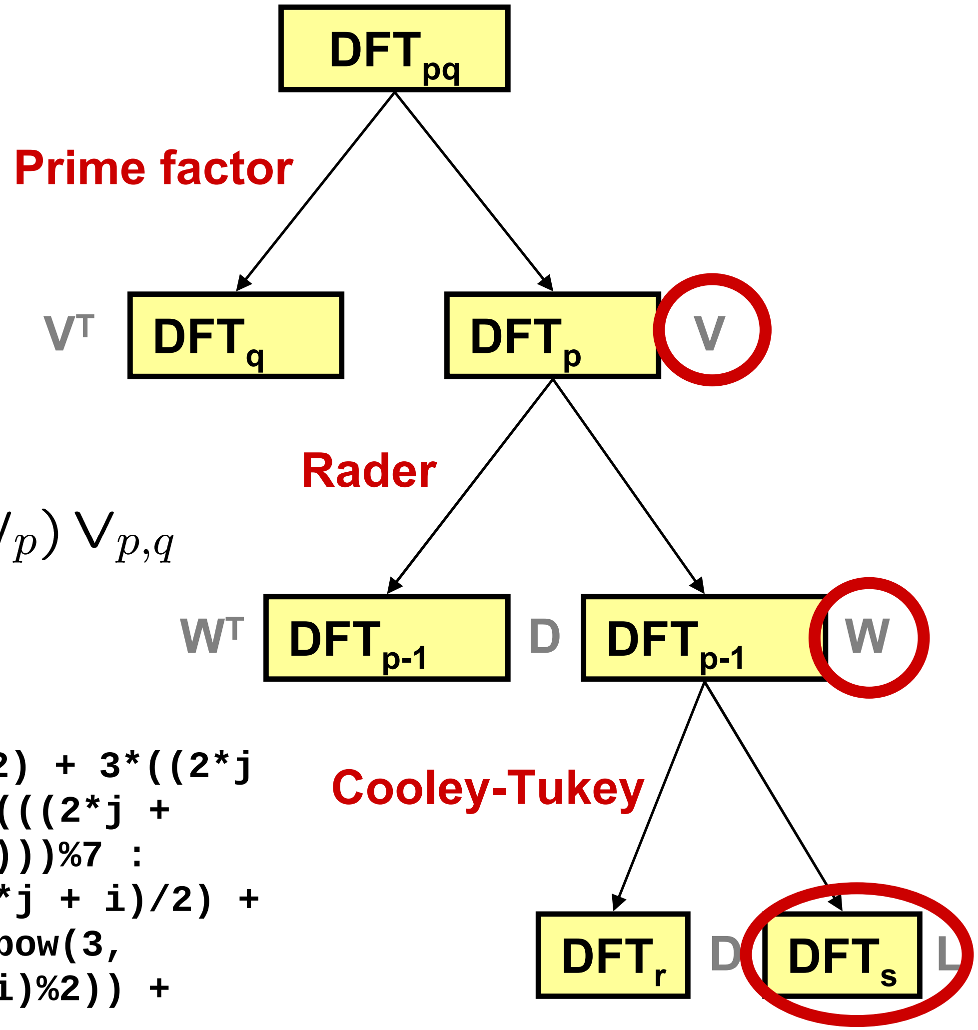
$\mathsf{W}_p \rightarrow \mathsf{Perm}_{w^p_{1,g}}$ $\qquad w^p_{\phi,g}(i) = \begin{cases} 0, & i = 0, \\ \phi g^i \bmod p, & \text{else.} \end{cases}$

Example

Given DFT_{pq}

p – prime

p-1 = rs



Formula fragment

$$(\mathrm{I}_p \otimes (\mathrm{I}_1 \oplus (\mathrm{I}_r \otimes \mathrm{DFT}_s)\, \mathrm{L}_r^{rs})\, \mathrm{W}_p)\, \mathrm{V}_{p,q}$$

Code for one memory access

```
p=7; q=4; r=3; s=2;
t=x[((21*((7*k + ((((((2*j + i)/2) + 3*((2*j
     + i)%2)) + 1)) ? (5*pow(3, ((((2*j +
     i)/2) + 3*((2*j + i)%2)) + 1)))%7 :
     (0)))/7) + 8*((7*k + ((((((2*j + i)/2) +
     3*((2*j + i)%2)) + 1)) ? (5*pow(3,
     ((((2*j + i)/2) + 3*((2*j + i)%2)) +
     1)))%7 : (0)))%7))%28)];
```

Task: Index simplification

Index Simplification: Basic Idea

- **Example: Identity necessary for fusing successive Rader and prime-factor step**

$$\left(\varphi g^{(b+si) \bmod N'}\right) \bmod N = \left((\varphi g^b)(g^s)^i\right) \bmod N$$

$$s|N',\ N'|N,\ 0 \le i < n$$

- **Performed at the Σ-SPL level through rewrite rules on function objects:**

$$\overline{w}_{\phi,g}^{N'\to N} \circ \overline{h}_{b,s}^{n\to N'} \quad \to \quad \overline{w}_{\phi g^b, g^s}^{n\to N}$$

- **Advantages:**
  - **no analysis necessary**
  - **efficient (or doable at all)**

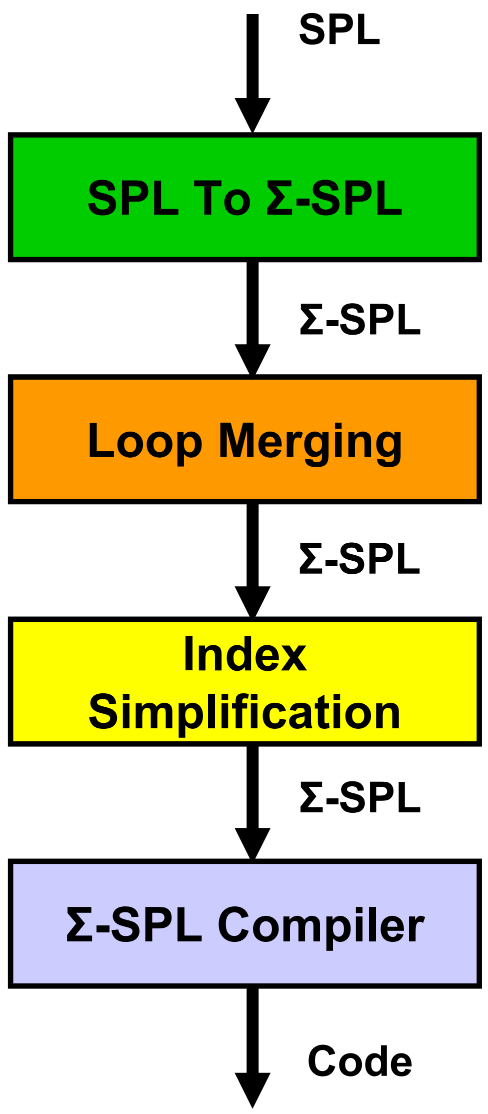
Index Simplification Rules for FFTs

Cooley-Tukey	$\ell^{mn}_{m} \circ \left((\jmath)_m \otimes f^{k\rightarrow n} \right)$	$\rightarrow$	$f^{k\rightarrow n} \otimes (\jmath)_m$
	$\left(f^{1\rightarrow m} \otimes h \right) \circ g$	$\rightarrow$	$f \otimes (h \circ g)$
	$\left(h \otimes g^{1\rightarrow n} \right) \circ f$	$\rightarrow$	$(h \circ f) \otimes g$
	$(f_0 \otimes f_1) \circ (g_0 \otimes g_1)$	$\rightarrow$	$(f_0 \circ g_0) \otimes (f_1 \circ g_1)$
Transitional	$\imath_n$	$\rightarrow$	$h^{n\rightarrow n}_{0,1}$
	$f^{m\rightarrow M} \otimes g^{1\rightarrow N}$	$\rightarrow$	$h^{M\rightarrow MN}_{g(0),N} \circ f$
	$g^{1\rightarrow N} \otimes f^{m\rightarrow M}$	$\rightarrow$	$h^{M\rightarrow MN}_{Mg(0),1} \circ f$
Cooley-Tukey + Prime factor	$v_{r,s} \circ h^{s\rightarrow rs}_{b,1}$	$\rightarrow$	$\overline{h}^{s\rightarrow rs}_{b,r}$
	$h^{nk\rightarrow mnk}_{b_1,s_1} \circ h^{n\rightarrow nk}_{b_2,s_2}$	$\rightarrow$	$h^{n\rightarrow mnk}_{b_1+s_1 b_2, s_1 s_2}$
	$\overline{h}^{nk\rightarrow mnk}_{b_1,s_1} \circ h^{n\rightarrow nk}_{b_2,s_2}$	$\rightarrow$	$\overline{h}^{n\rightarrow mnk}_{b_1+s_1 b_2, s_1 s_2}$
	$\overline{h}^{nk\rightarrow mnk}_{b_1,s_1} \circ \overline{h}^{n\rightarrow nk}_{b_2,s_2}$	$\rightarrow$	$\overline{h}^{n\rightarrow mnk}_{b_1+s_1 b_2, s_1 s_2}$
Cooley-Tukey + Prime factor + Rader	$w^{N}_{\phi,g} \circ (0)^{1\rightarrow N}_{+}$	$\rightarrow$	$(0)^{1\rightarrow N}_{+}$
	$w^{N}_{\phi,g} \circ (N-1)^{N-1\rightarrow N}_{+}$	$\rightarrow$	$\overline{w}^{N-1\rightarrow N}_{\phi,g}$
	$\overline{w}^{N'\rightarrow N}_{\phi,g} \circ h^{n\rightarrow N'}_{b,s}$	$\rightarrow$	$\overline{w}^{n\rightarrow N}_{\phi g^b, g^s}$
	$\overline{w}^{N'\rightarrow N}_{\phi,g} \circ \overline{h}^{n\rightarrow N'}_{b,s}$	$\rightarrow$	$\overline{w}^{n\rightarrow N}_{\phi g^b, g^s}$

These 15 rules cover all combinations.
Some encode novel optimizations.

SPIRAL

Loop Merging For the FFTs : Example (cont'd)



$$(\mathrm{I}_p \otimes (\mathrm{I}_1 \oplus (\mathrm{I}_r \otimes \mathsf{DFT}_s)\, \mathsf{L}_r^{rs})\, \mathsf{W}_p)\, \mathsf{V}_{p,q}$$

$$\sum_{j_1=0}^{p-1} \Big(\mathsf{S}_{\left((j_1)_p \otimes \imath_q\right) \circ (0)_+^{1\to q}}\, \mathsf{G}_{v^{p,q} \circ \left((j_1)_p \otimes \imath_q\right) \circ w_{1,g}^q \circ (0)_+^{1\to q}} + \sum_{j_0=0}^{r-1} \mathsf{S}_{\left((j_1)_p \otimes \imath_q\right) \circ (1)_+^{q-1\to q} \circ ((j_0)_r \otimes \imath_s)}\, \mathsf{DFT}_s\, \mathsf{G}_{v^{p,q} \circ \left((j_1)_p \otimes \imath_q\right) \circ w_{1,g}^q \circ (1)_+^{q-1\to q} \circ \ell_r^{rs} \circ ((j_0)_r \otimes \imath_s)} \Big)$$

$$\sum_{\substack{j_1=0 \\ b_1 = qj_1}}^{p-1} \Big(\mathsf{S}_{h_{0,q}^{p\to pq}(j_1)}\, \mathsf{G}_{\overline{h}_{0,q}^{p\to pq}(j_1)} + \sum_{\substack{j_0=0 \\ \phi_1 = g^{j_0}}}^{r-1} \mathsf{S}_{h_{qj_1+sj_0+1,1}^{s\to pq}}\, \mathsf{DFT}_s\, \mathsf{G}_{\overline{h}_{b_1,p}^{q\to pq} \circ \overline{w}_{\phi_1,g^s}^{s\to q}} \Big)$$

```
// I
in
fo
    y[7*j1] = x[(7*j1%28)];
    p1 = 1; b1 = 7*j1;
    for(int j0 = 0; j0 <= 2; j0++) {
        y[b1 + 2*j0 + 1] = x[(b1 + 4*p1)%28] + x[(b1 + 24*p1)%28];
        y[b1 + 2*j0 + 2] = x[(b1 + 4*p1)%28] - x[(b1 + 24*p1)%28];
        p1 = (p1*3%7);
    }
}
```

```c
// Input: _Complex double x[28], output: y[28]
double t1[28];
for(int i5 = 0; i5 <= 27; i5++)
    t1[i5] = x[(7*3*(i5/7) + 4*2*(i5%7))%28];
for(int i1 = 0; i1 <= 3; i1++) {
    double t3[7], t4[7], t5[7];
    for(int i6 = 0; i6 <= 6; i6++)
        t5[i6] = t1[7*i1 + i6];
    for(int i8 = 0; i8 <= 6; i8++)
        t4[i8] = t5[i8 ? (5*pow(3, i8))%7 : 0];
    {
        double t7[1], t8[1];
        t8[0] = t4[0];
        t7[0] = t8[0];
        t3[0] = t7[0];
    }
    {
        double t10[6], t11[6], t12[6];
        for(int i13 = 0; i13 <= 5; i13++)
            t12[i13] = t4[i13 + 1];
        for(int i14 = 0; i14 <= 5; i14++)
            t11[i14] = t12[(i14/2) + 3*(i14%2)];
        for(int i3 = 0; i3 <= 2; i3++) {
            double t14[2], t15[2];
            for(int i15 = 0; i15 <= 1; i15++)
                t15[i15] = t11[2*i3 + i15];
            t14[0] = (t15[0] + t15[1]);
            t14[1] = (t15[0] - t15[1]);
            for(int i17 = 0; i17 <= 1; i17++)
                 t10[2*i3 + i17] = t14[i17];
        }
        for(int i19 = 0; i19 <= 5; i19++)
            t3[i19 + 1] = t10[i19];
    }
    for(int i20 = 0; i20 <= 6; i20++)
        y[7*i1 + i20] = t3[i20];
}
```

Before, 11 Loops.

```c
// Input: _Complex double x[28], output: y[28]
  int p1, b1;
  for(int j1 = 0; j1 <= 3; j1++) {
     y[7*j1] = x[(7*j1%28)];
     p1 = 1; b1 = 7*j1;
     for(int j0 = 0; j0 <= 2; j0++) {
        y[b1 + 2*j0 + 1] = x[(b1 + 4*p1)%28] +
                           x[(b1 + 24*p1)%28];
        y[b1 + 2*j0 + 2] = x[(b1 + 4*p1)%28] –
                           x[(b1 + 24*p1)%28];
        p1 = (p1*3%7);
     }
  }
```

After, 2 Loops.

Talk Organization

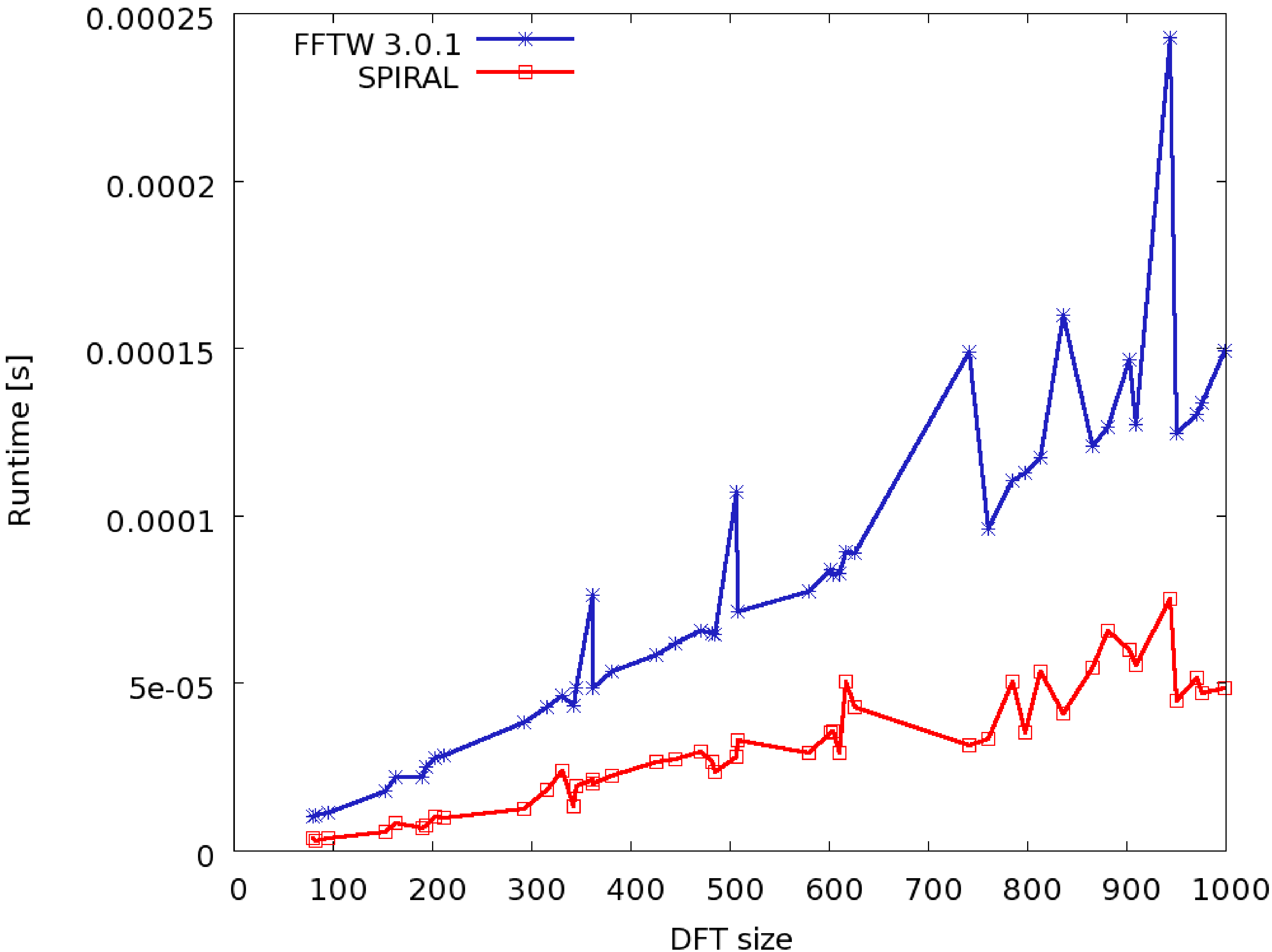
- SPIRAL Background
- Automatic loop merging in SPIRAL
- **Experimental Results**
- Conclusions

Benchmarks Setup

- **Comparison against FFTW 3.0.1**
- **Pentium 4 3.6 GHz**
- **We consider sizes requiring at least one Rader step (sizes with large prime factor)**
- **We divide sizes into levels depending on number of Rader steps needed (Rader FFT has most expensive index mapping)**

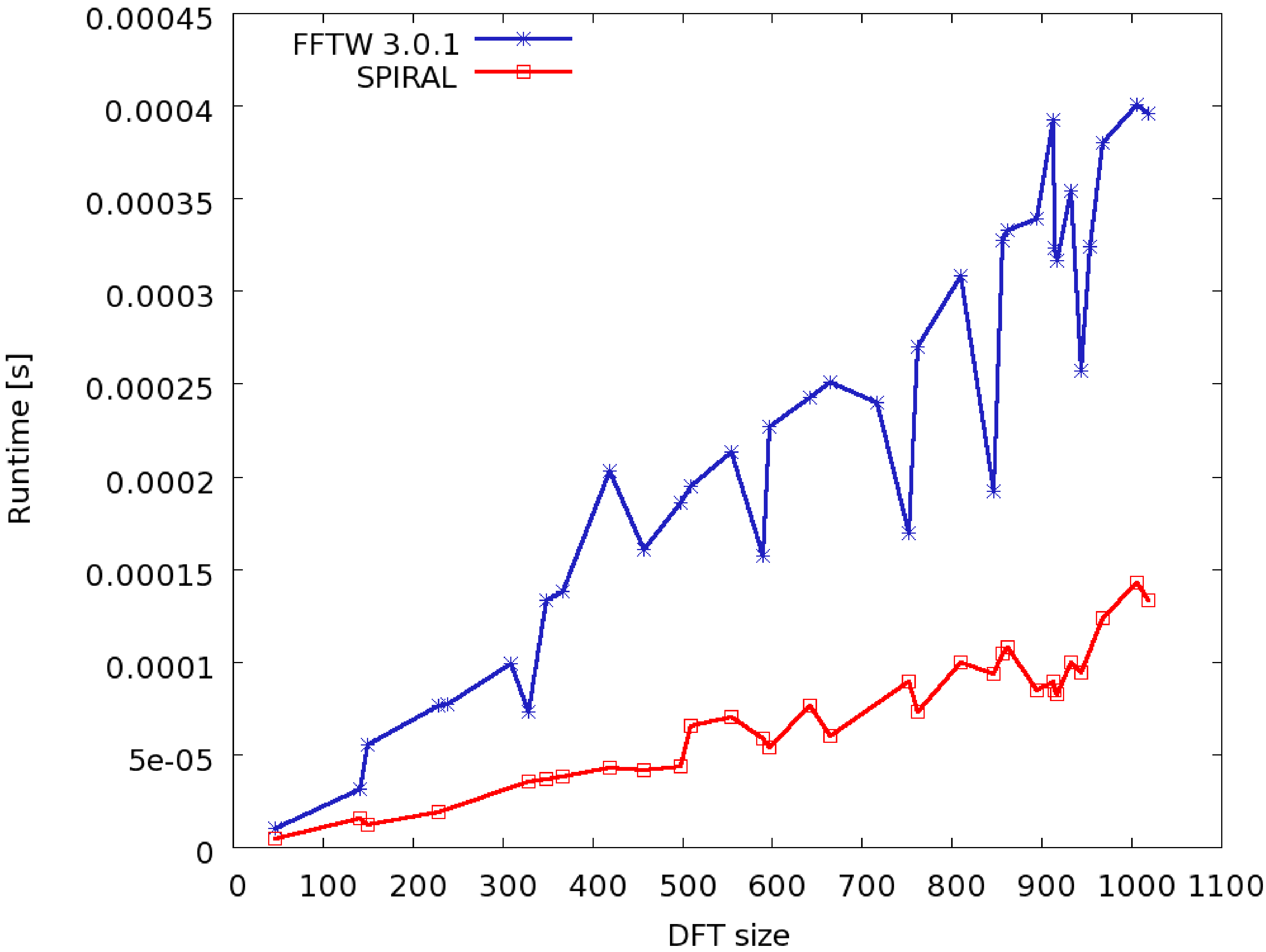
<u>One</u> Rader Step

Average SPIRAL speedup: <u>factor of 2.7</u>



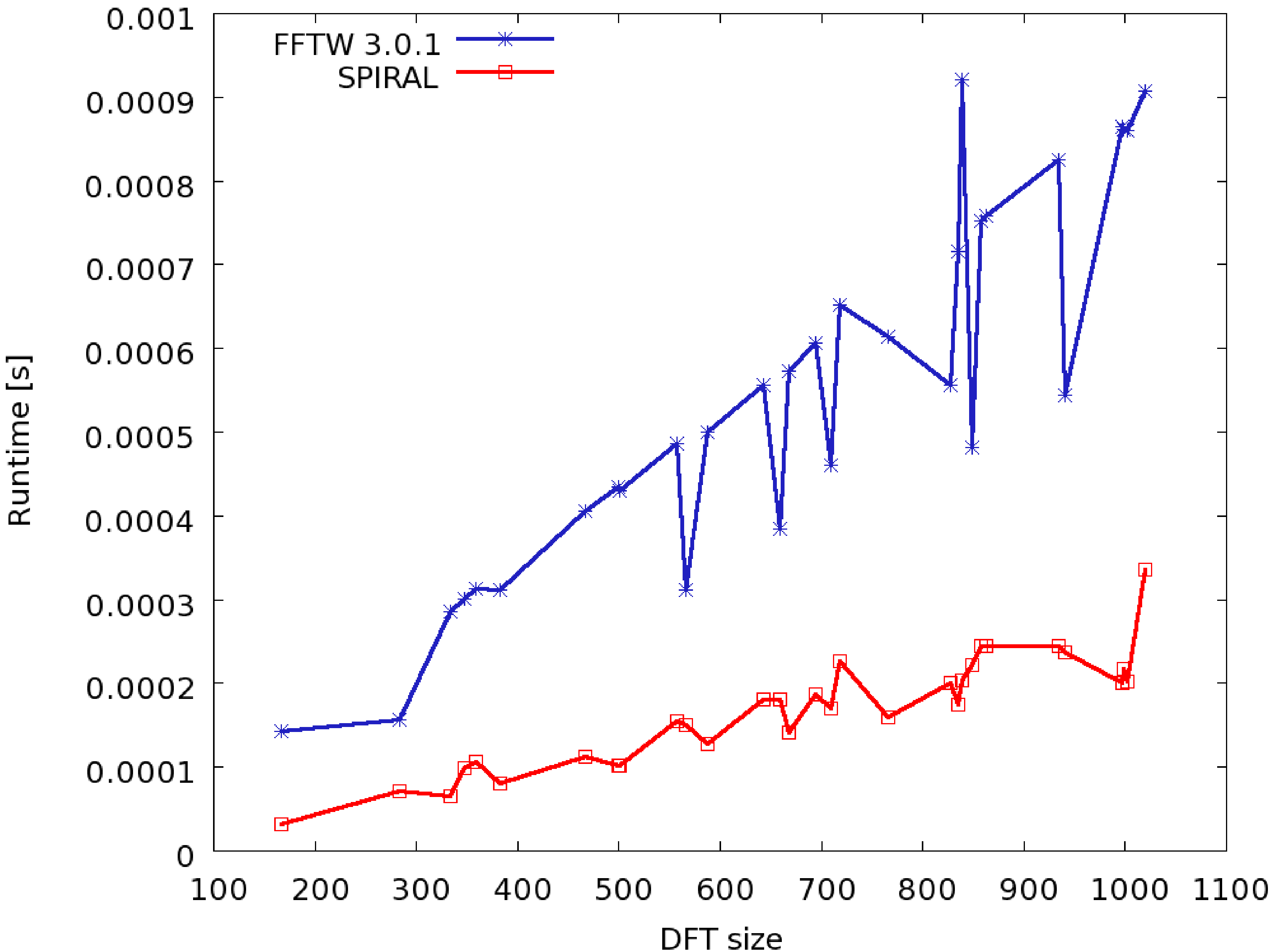
Two Rader Steps

Average SPIRAL speedup: factor of 3.3



Three Rader Steps

Average SPIRAL speedup: factor of 3.4



Talk Organization

- SPIRAL Background
- Automatic loop merging in SPIRAL
- Experimental Results
- **Conclusions**

Conclusion

- General loop optimization framework for linear DSP transforms in SPIRAL
- Loop optimization at the “right” abstraction level: Σ-SPL
- Application to FFT: Speedups of a factor of 2-5 over FFTW
- Future work: Other Σ-SPL optimizations
  - Loop merging for other transforms
  - Loop elimination, interchange, peeling

`http://www.spiral.net`