SPIRAL-Generated Modular FFTs[*]

Jeremy Johnson Lingchuan Meng

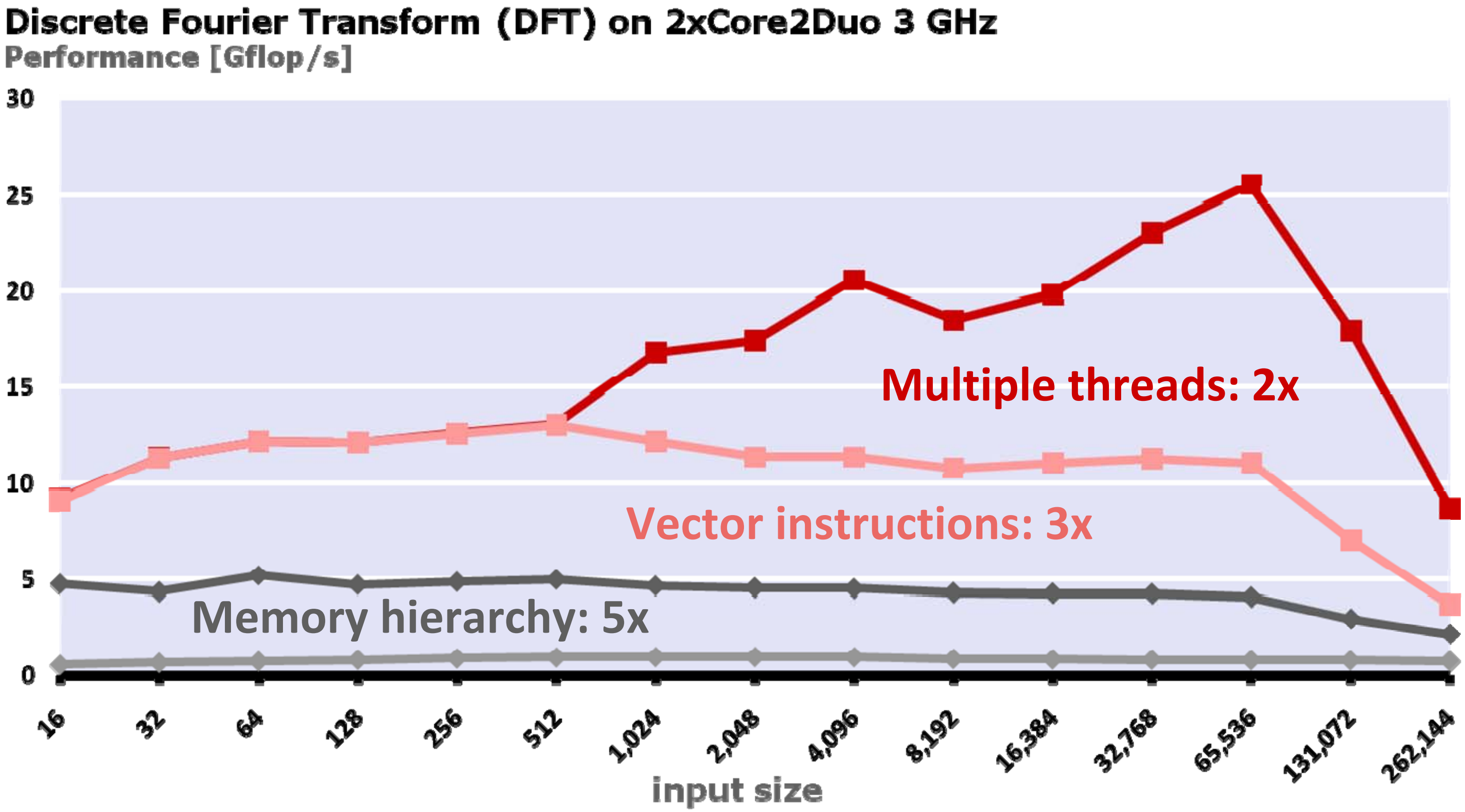
Drexel University

* The work is supported by DARPA DESA, NSF, and Intel.
Material for SPIRAL overview provided by Franz Francheti, Yevgen Voronenko, Markus Püschel , Martin Barbella and the rest of the SPIRAL team.

Outline

- **SPIRAL Overview**
  - Challenge of Obtaining Efficient Code
  - Transforms and Rules
  - SPIRAL Architecture
  - SPIRAL Abstraction Levels
  - Example
  - Performance
- **Modular DFT in SPIRAL**
  - Addition of Modular DFT Transform and Modular FFT Rules
  - Code Generation for Modular FFT
  - Performance
  - Future Work

Challenge of Obtaining Efficient Code



Multiple threads: 2x

Vector instructions: 3x

Memory hierarchy: 5x

High performance library development
has become a nightmare

Spiral Overview

- **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

DSP Algorithms: Transforms & Breakdown Rules

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

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

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

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

$$\mathbf{DCT\text{-}2}_n \rightarrow P_{k/2,2m}^{\top}\left(\mathbf{DCT\text{-}2}_{2m} K_2^{2m} \oplus \left(I_{k/2-1} \otimes N_{2m}\, \mathbf{RDFT\text{-}3}_{2m}^{\top}\right)\right) B_n (L_{k/2}^{n/2} \otimes I_2)(I_m \otimes \mathbf{RDFT}'_k) Q_{m/2,k},$$

$$\mathbf{DCT\text{-}3}_n \rightarrow \mathbf{DCT\text{-}2}_n^{\top},$$

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

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

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

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

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

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

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

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

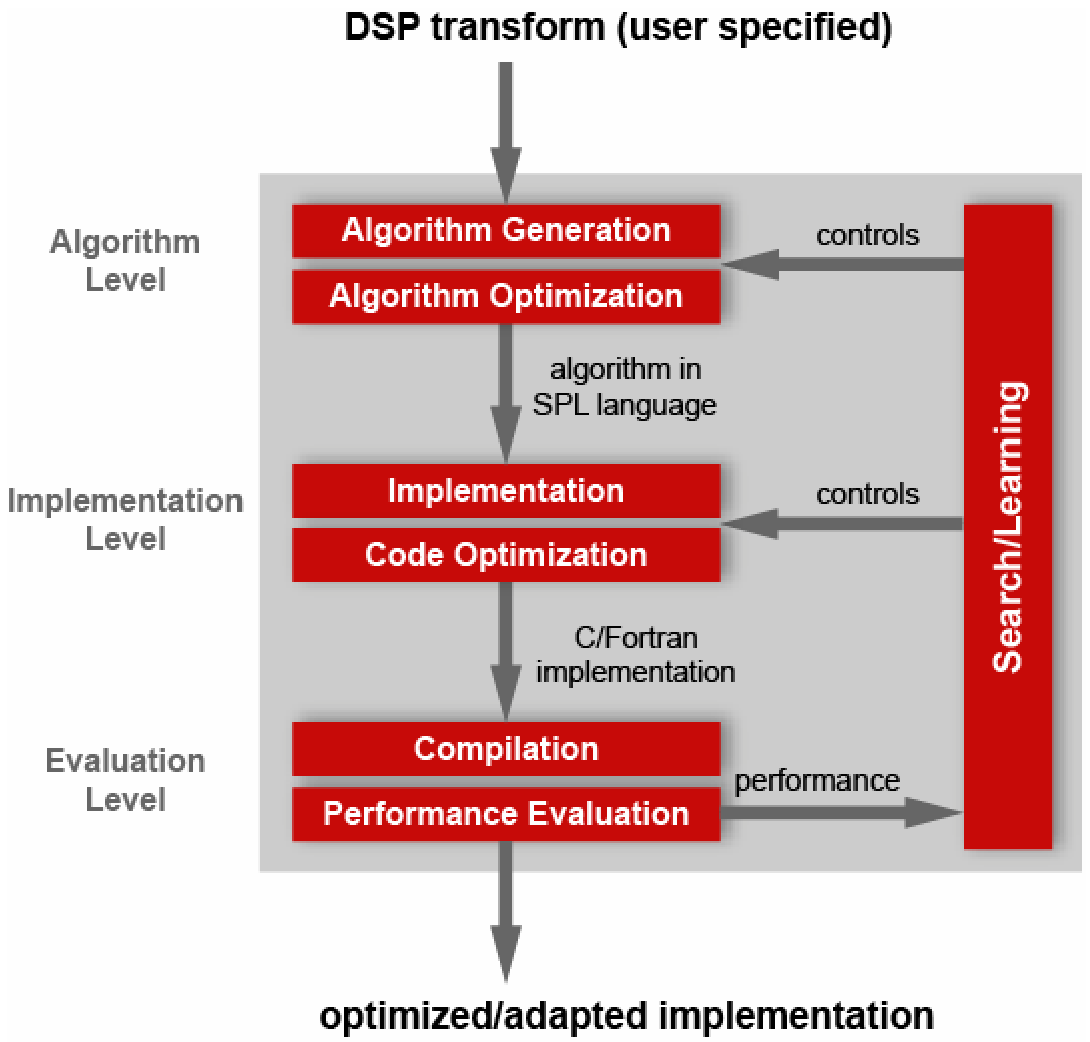
$$\mathbf{DFT}_2 \rightarrow \mathsf{F}_2$$

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

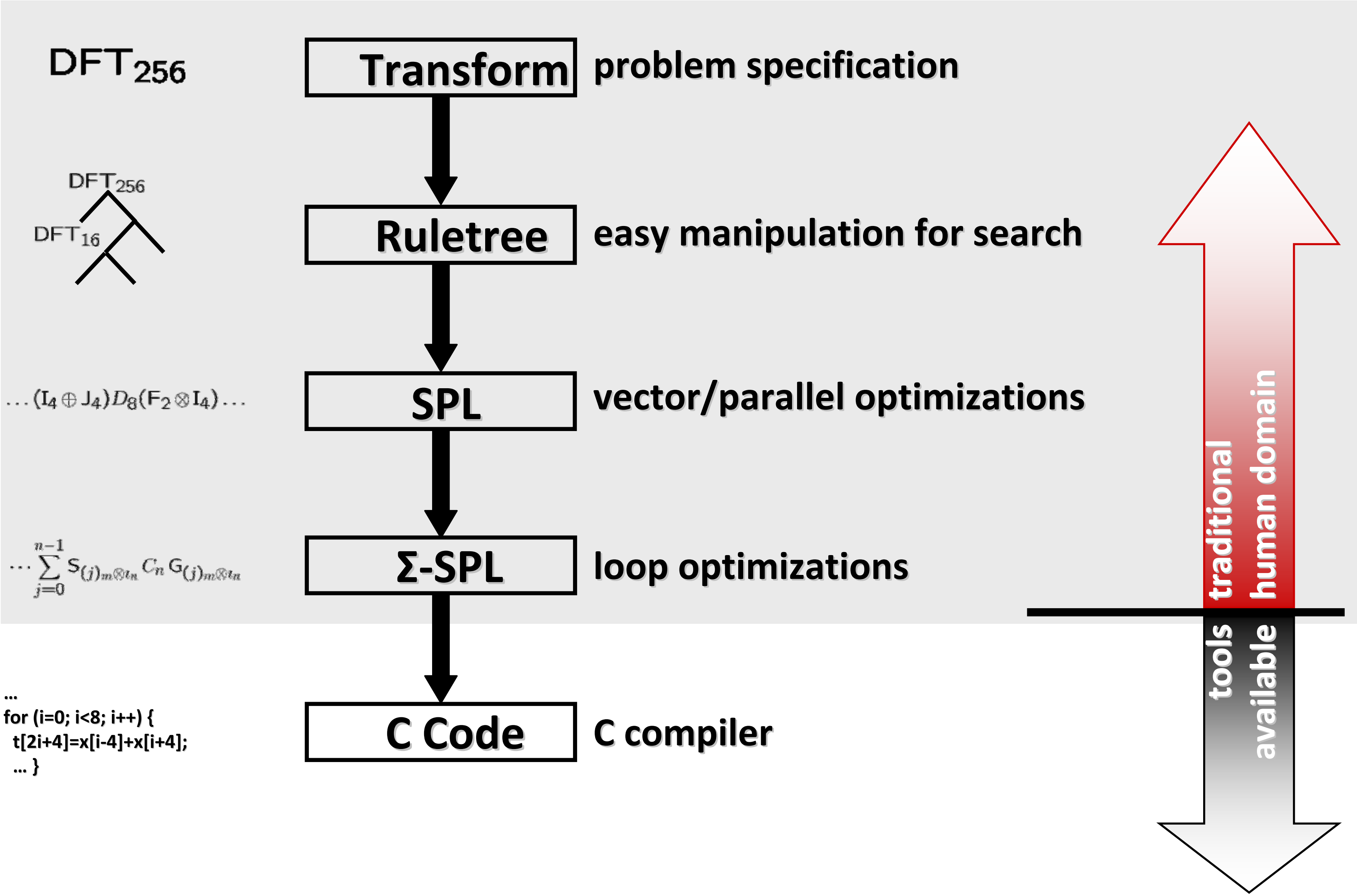
$$\mathbf{DCT\text{-}4}_2 \rightarrow \mathsf{J}_2\, \mathsf{R}_{13\pi/8}$$

Combining these rules yields many algorithms for every given transform

SPIRAL: Architecture



SPIRAL: Abstraction Levels



DFT_{256}

Transform problem specification

Ruletree easy manipulation for search

$\ldots(\mathrm{I}_4 \oplus \mathrm{J}_4) D_8 (\mathrm{F}_2 \otimes \mathrm{I}_4)\ldots$

SPL vector/parallel optimizations

$\ldots \sum_{j=0}^{n-1} \mathrm{S}_{(j)_m \otimes \iota_n} C_n \mathrm{G}_{(j)_m \otimes \iota_n}$

Σ-SPL loop optimizations

```
...
for (i=0; i<8; i++) {
  t[2i+4]=x[i-4]+x[i+4];
  ... }
```

C Code C compiler

DSP Algorithms: E.G. 4-point DFT

$$x \mapsto y = M \cdot x$$

input vector (signal)

output vector (signal)

transform = matrix

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \\ 1 & -1 & 1 & -1 \\ 1 & -i & -1 & i \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & i \end{bmatrix} \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & -1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Fourier transform **Diagonal matrix (twiddles)** **Input vector**

$$y = (\mathrm{DFT}_2 \otimes \mathrm{I}_2) \cdot \mathrm{T}^4_2 \cdot (\mathrm{I}_2 \otimes \mathrm{DFT}_2) \cdot \mathrm{L}^4_2 \; x$$

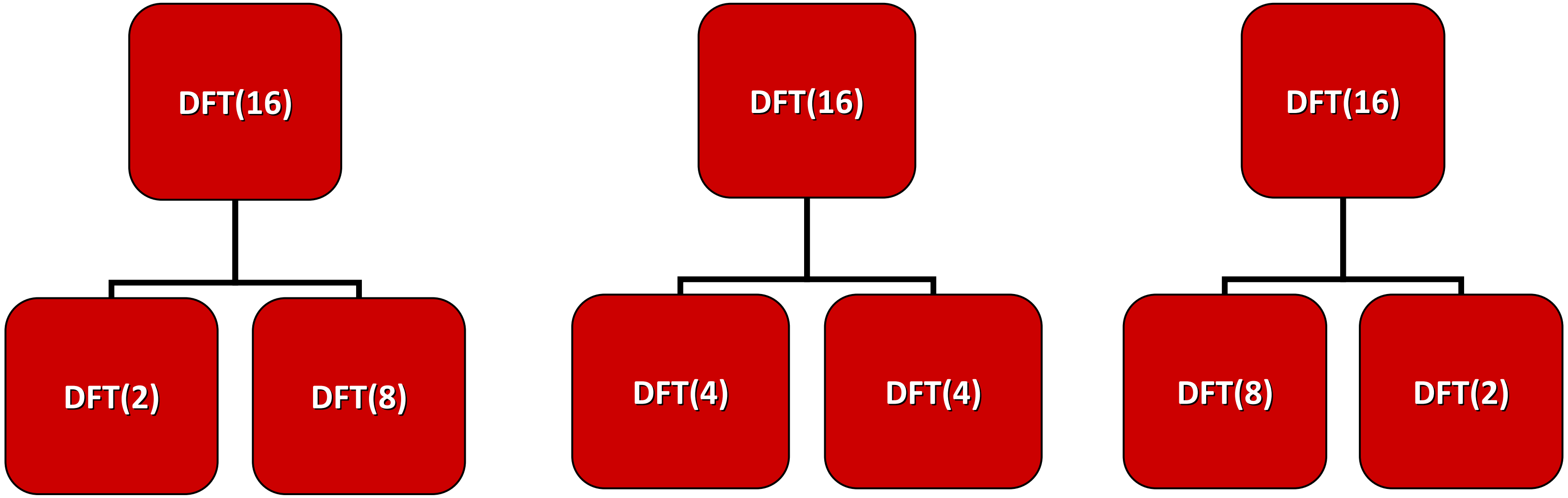
Output vector **Kronecker product** **Identity** **Permutation**

Cooley-Tukey Rule and DP

Diagonal matrix (twiddles)

$$DFT_{mn} = (\mathrm{DFT}_m \otimes \mathrm{I}_n) \cdot \mathsf{T}^{mn}_{n} \cdot (\mathrm{I}_m \otimes \mathrm{DFT}_n) \cdot \mathsf{L}^{mn}_{m}$$

Kronecker product **Identity** **Permutation**



DFT(16)

DFT(2) DFT(8)

DFT(16)

DFT(4) DFT(4)

DFT(16)

DFT(8) DFT(2)

Dynamic Programming (DP) searches over different applications of CT Rule using best transform found for recursive calls

Program Generation in Spiral (Sketched)

Transform
user specified

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

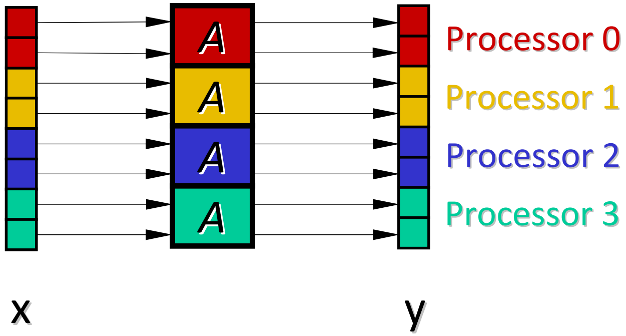
loop
optimizations

constant folding
scheduling
……

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

- **Governing construct: tensor product**

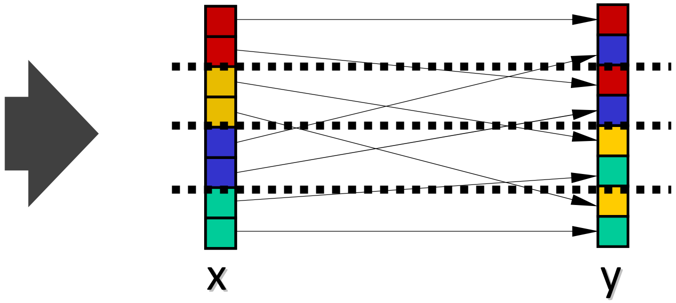
$$y = \left(\mathrm{I}_p \otimes A \right) x$$



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

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

$$\ldots$$

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

$$\ldots$$

$$\rightarrow \underline{\Big((\mathrm{L}^{mp}_m \otimes \mathrm{I}_{n/p\mu}) \otimes_\mu \mathrm{I}_\mu\Big)}\,\underline{\Big(\mathrm{I}_p \otimes_\parallel (\mathrm{DFT}_m \otimes \mathrm{I}_{n/p})\Big)}\,\underline{\Big((\mathrm{L}^{mp}_p \otimes \mathrm{I}_{n/p\mu}) \otimes_\mu \mathrm{I}_\mu\Big)}$$

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

↑

coarse
platform
model

Load-balanced

No false sharing

Same Approach for Other Parallel Paradigms

Message Passing *[ISPA 06]*

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

$\ldots$

$\ldots$

$\ldots$

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

Vectorization *[VECPAR 06]*

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

$\ldots$

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

$\ldots$

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

Cg/OpenGL for GPUs

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

$\ldots$

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

Verilog for FPGAs *[DAC 08]*

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

$\ldots$

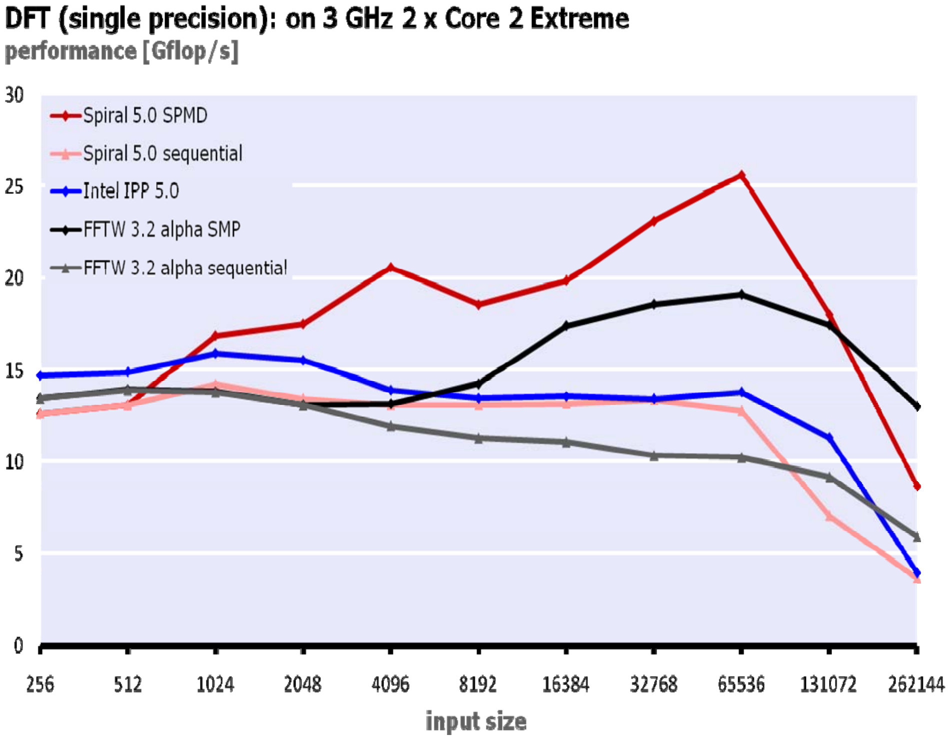
$$\rightarrow \left[\prod_{i=0}^{k-1} \underbrace{\mathrm{L}^{r^k}_r}_{\mathrm{stream}(r^s)} \underbrace{\left(\mathrm{I}_{r^{k-1}} \otimes \mathbf{DFT}_r\right)}_{\mathrm{stream}(r^s)} \underbrace{\left(\mathrm{L}^{r^k}_{r^{k-i-1}}(\mathrm{I}_{r^i} \otimes \mathrm{T}^{r^{k-i}}_{r^{k-i-1}})\,\mathrm{L}^{r^k}_{r^{i+1}}\right)}_{\mathrm{stream}(r^s)}\right]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

$\ldots$

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

Example Results

Multicore *[SC 06]*



Code written by the computer is faster (for many sizes) than any human-written code

Modular DFT in SPIRAL

- **New Transform: ModDFT(n, p, w, i)**
  - ***n*,** the size of the transform
  - ***p*** , the prime modulus
  - ***w*** , primitive n^{th} root of unity
  - ***i*** , forward/inverse flag
  - ***Same as DFT except must store p and w***

- **Breakdown Rule for ModDFT: Cooley-Tukey in Z_p**
  - Same as complex DFT except different roots of unity are computed
- **CUnparser for ModDFT: Generate Z_p C code**
  - C code generated for modular arithmetic operations
  - Alternative approaches explored

Cooley-Tukey in Z_p : Example

$$x \longmapsto y = M \cdot x$$

Input Vector **Output Vector** **Transform matrix in** Z_p

$$ModDFT(4,17,3) = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 13 & 16 & 4 \\ 1 & 16 & 1 & 16 \\ 1 & 4 & 16 & 13 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 16 & 0 \\ 0 & 1 & 0 & 16 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 13 \end{bmatrix} \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 16 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 16 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Modular Fourier transform **Diagonal matrix (twiddles)** **Input vector**

$$y = (ModDFT_2 \otimes I_2) \cdot T^4_2 \cdot (I_2 \otimes ModDFT_2) \cdot L^4_2 \cdot x$$

Output vector **Kronecker product** **Identity** **Permutation**

Modular FFTs: Cooley-Tukey in Z_p

- **Cooley-Tukey algorithm**
  A transform of size mn breaks down into smaller transforms of size m and n

$$DFT(m \cdot n) \longrightarrow DFT(m), DFT(n)$$
$$\longrightarrow Tensor(DFT(m), I(n)) \cdot T_m^{mn}$$
$$Tensor(I(n), DFT(m)) \cdot L_n^{mn}$$

- **Cooley-Tukey algorithm in Z_p**
  Primitive root of P used in twiddle factors instead of complex root of unity

$$ModDFT(m \cdot n, p, \omega) \longrightarrow ModDFT(m, p, \omega^m), ModDFT(n, p, \omega^n)$$
$$\longrightarrow Tensor(ModDFT(m, p, \omega^m), I(n)) \cdot T_m^{mn}(\omega^m) \cdot$$
$$Tensor(I(m), ModDFT(n, p, \omega^n)) \cdot L_n^{mn}$$

Modular FFTs: Unparser

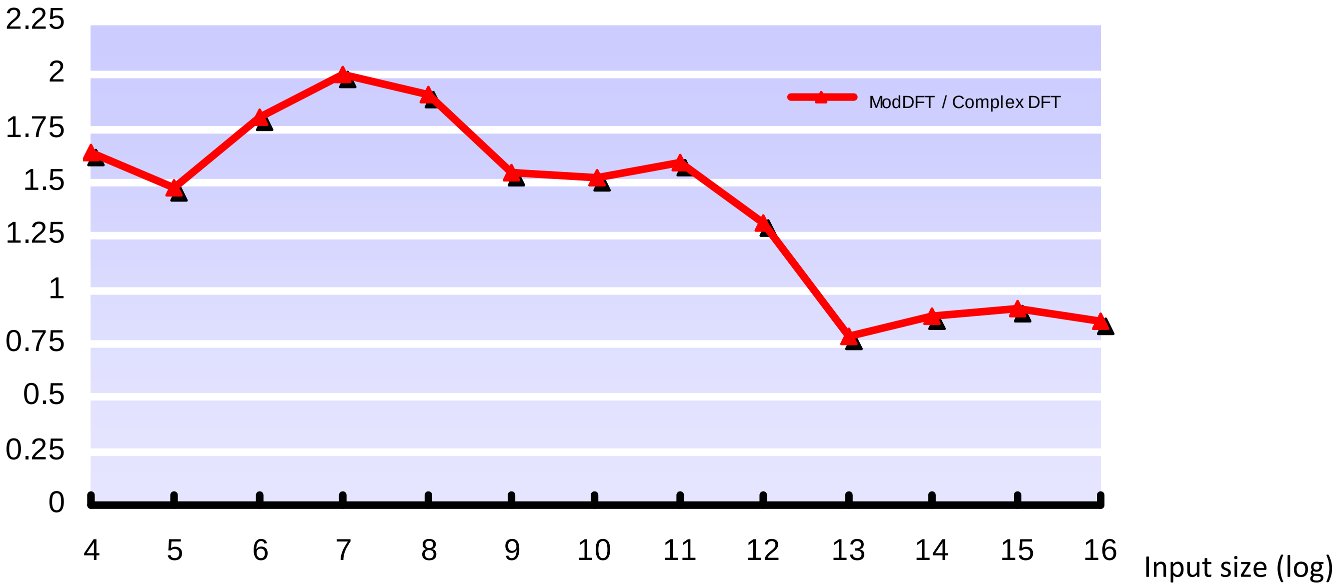
- **Unparser: translate symbolic arithmetic operations to corresponding C code**
- **Unparser for Modular FFTs**
  - add (a, b) → (a + b) % p
  - sub (a, b) → (a – b) % p
    - All elements are positive and smaller than p
    - 0 < a + b < 2p → (a + b) % p = (a + b < p)? (a + b) : (a + b – p)
    - -p < a - b < p → (a - b) % p = (a - b < 0)? (a - b + p) : (a - b)
      - Faster than division though may introduce branches
      - Can use Conditional Move instead of branch (avoid pipeline stalls)
  - mul (a, b) → a * b % p

Experiments

- **Platform**
  - AMD Phenom II X4 940 quad-core processor
    - Operating Frequency: 3.0 GHz
    - L1 Cache: 4 X 128 KB (64KB Instruction + 64KB Data)
    - L2 Cache: 4 X 512 KB
    - L3 Cache: 6 MB
- **Experiments**
  - Compare SPIRAL performance of modular FFT and complex floating point FFT
    - Optimal algorithms found may be different
  - Compare SPIRAL modular FFT to numeric recipes (NR) code modified to perform modular DFT [NR uses iterative FFT]
  - Compare optimal SPIRAL modular FFT found by DP to radix 2 recursive FFT

SPRIAL Code Performance

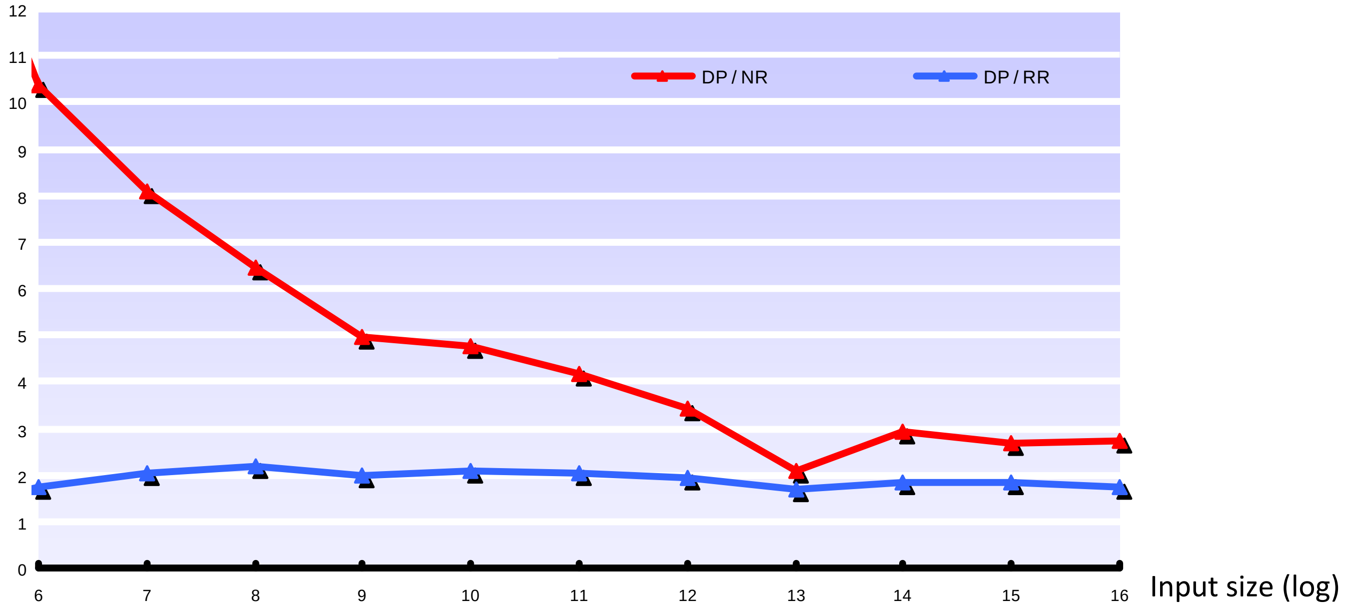
Best Code found by DP
Modular DFT vs. Complex DFT



- Expensive modular operations slow down ModDFT within the cache boundary
- Memory accesses dominate performance when input size crosses cache boundary ($\geqslant 2^{13}$)
  - Cache boundary reached one size earlier for complex DFT

SPRIAL Code Performance

Complex DFT Code
Best SPIRAL code (DP) vs. Radix 2 Recursive Code & NR

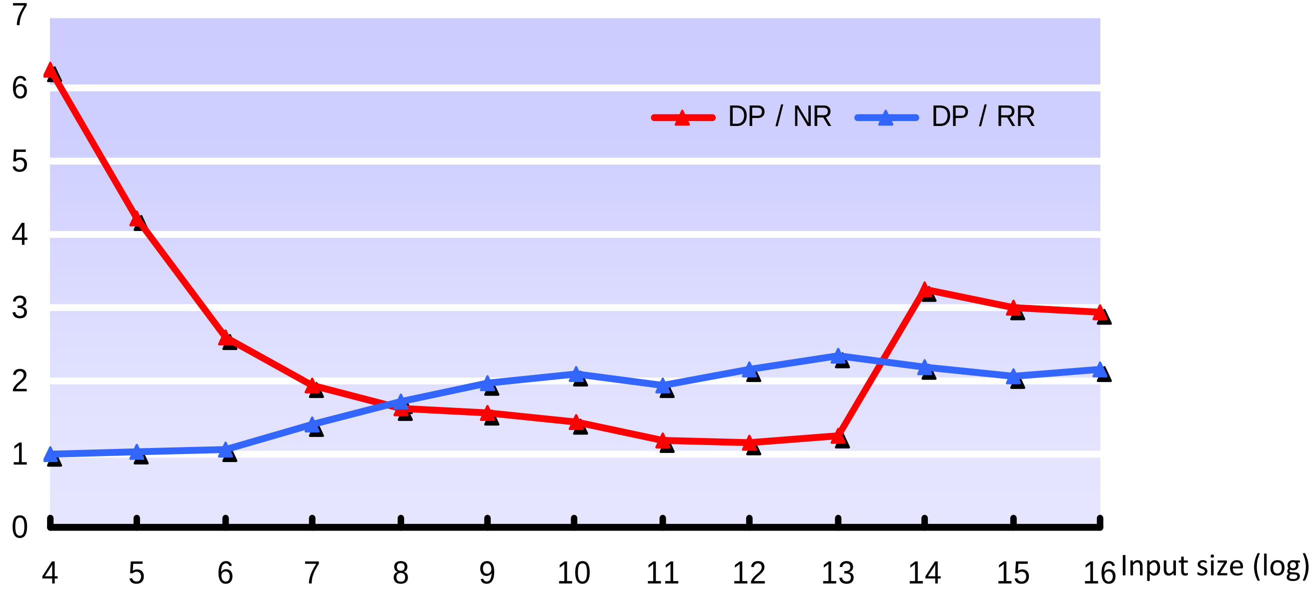


- SPIRAL best code about twice faster than its right recursive code
  - Better algorithm found by the search engine

- SPIRAL best code averagely 4.8 times faster the numerical recipes code.
  - Better cache/memory utilization by SPIRAL
- Performance grouped into 3 regions (unrolled, in-cache, outside L1 cache)

SPRIAL Code Performance

Modular FFT
Best SPIRAL code (DP) vs. Radix 2 Recursive Code & NR



- Best SPIRAL code twice as fast as right recursive code
  - Better algorithm found by the search engine
- Best SPIRAL code on average 3 times faster than NR code.
  - Better cache/memory utilization by SPIRAL
- Similar performance trends for modular and complex DFTs
  - Larger jumps across unrolling and cache boundaries

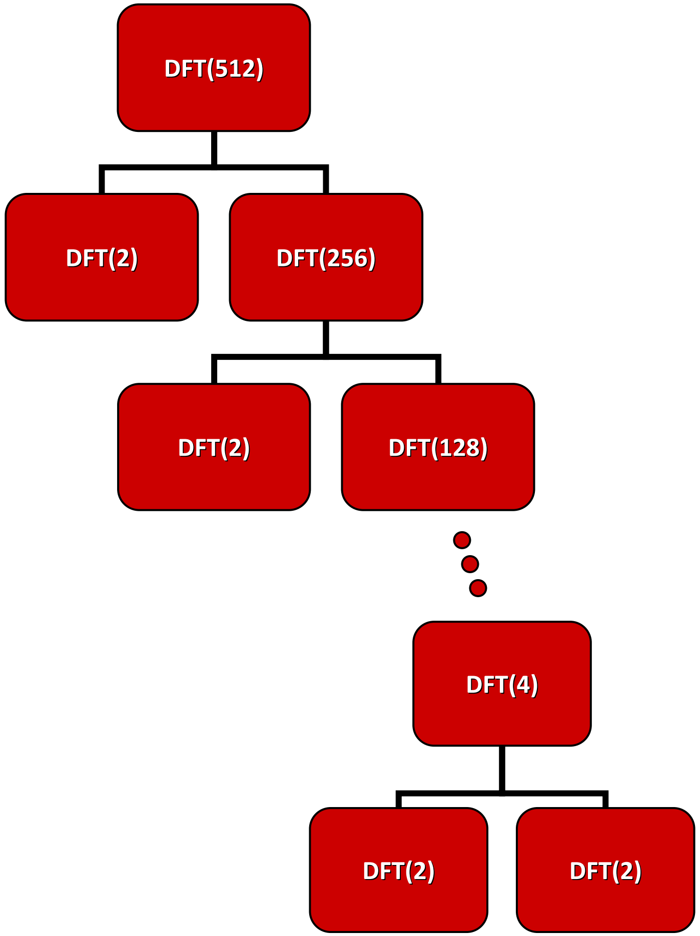
Experimental Data

Input Size	Radix-2 Right Recursive		Dynamic Programming		Numerical Recipes	
	ModDFT	DFT	ModDFT	DFT	ModDFT	DFT
2^3	*107	73	106	68	1852	5202
2^4	271	183	271	165	3002	6073
2^5	701	559	684	465	5402	7636
2^6	2146	1950	2033	1130	10130	11713
2^7	7183	5113	5054	2528	19194	20382
2^8	19491	12769	11290	5909	39687	38049
2^9	46169	30097	23490	15214	81820	75729
2^{10}	109924	71651	52297	34368	164575	163198
2^{11}	256097	170499	132688	83046	363737	345289
2^{12}	611067	424499	284737	217446	800674	748075
2^{13}	1440588	1316808	613370	778572	5555759	1628003
2^{14}	3814025	3648241	1737673	1979379	13952347	5782618
2^{15}	9987370	9698445	4866960	5301481	34076665	14269816
2^{16}	24608317	22981890	11386662	13381989	86195220	36172945

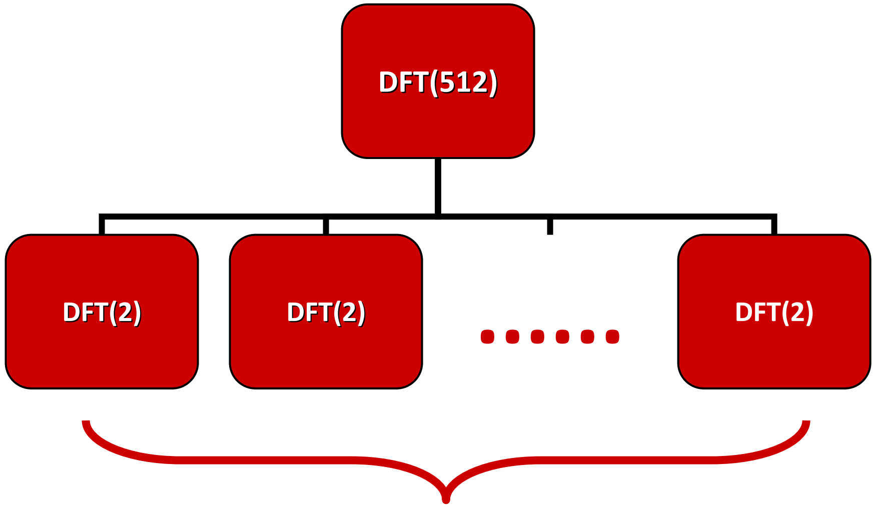
* Number of cycles on AMD Phenom II 940 3.0 GHz Quad-core processor

Standard FFT Ruletrees

- Radix-2 Right Recursive Rule Tree



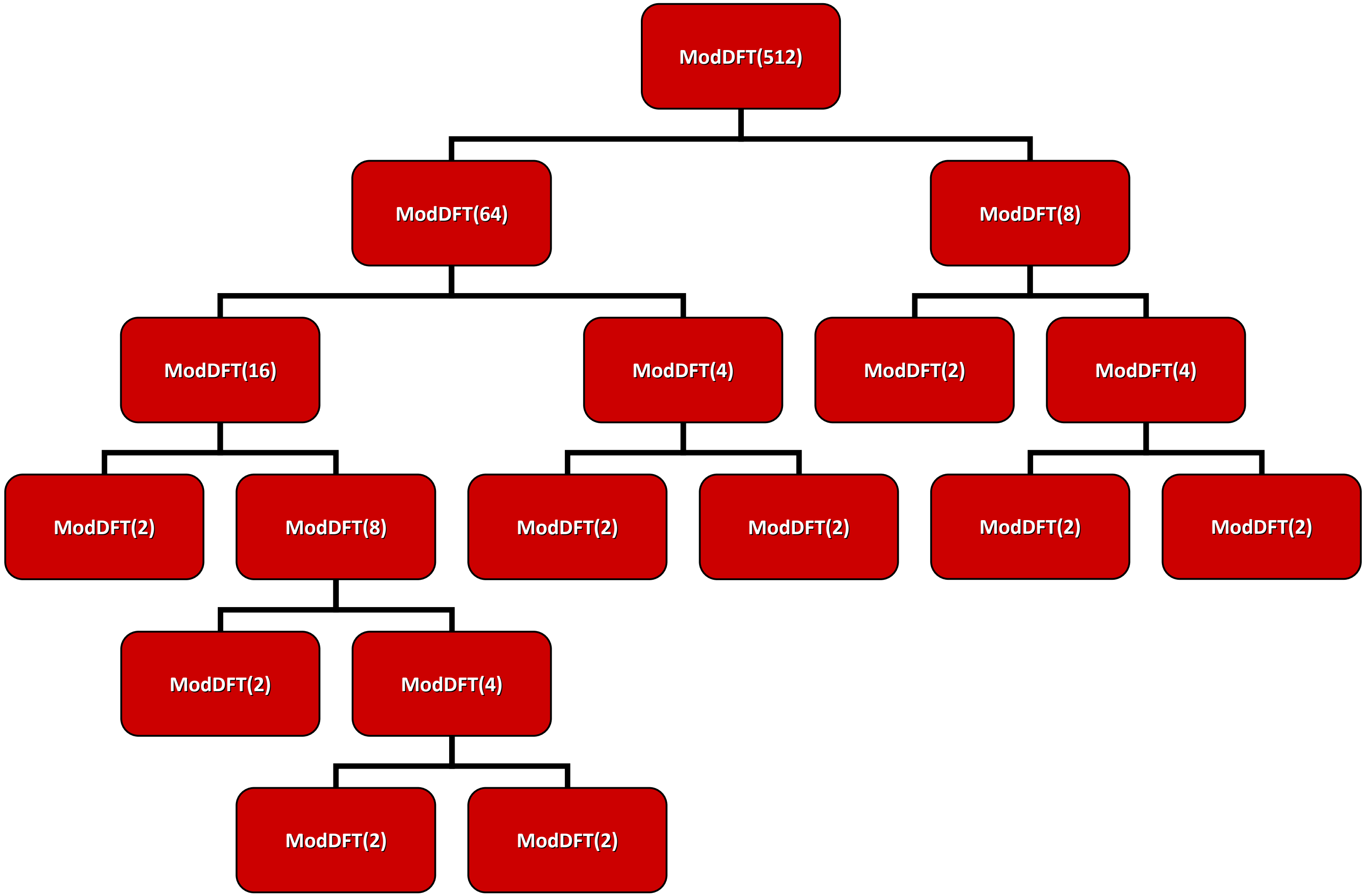
- Iterative Tree in Numerical Recipes



9 DFT(2)

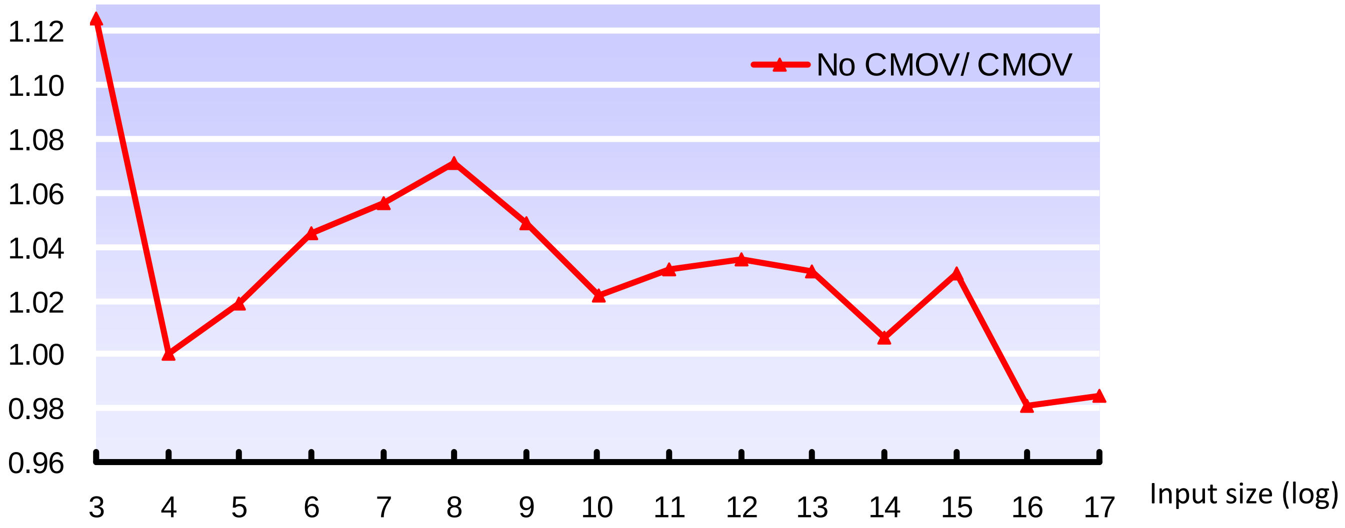
Optimal Modular FFT Ruletree

- **Best Rule Tree found by Dynamic Programming**



Conditional Move

- **Conditional Moves**



- CMOVcc theoretically reduces pipeline penalty by converting branches to branchless codes in ternary operations of add/sub
- Experiments show improvement only for large sizes
  - Short branches of ternary operations optimized by CPU’s multi-instruction fetch and conditional execution
  - Branch prediction table entries overwritten in large-sized programs
  - Instructions fetches insufficient for conditional execution of larger programs

Future Work

- **Montgomery Algorithm [MATHCOMP 85]**
  - Modular multiplication without division
  - Enable vectorization when vector division not available
- **Vectorization**
  - Extend SPIRAL vectorization to support modular DFT
- **Multicore Parallelization**
  - Extend SPIRAL parallelization to support modular DFT

Conclusion

- **Minor modifications to SPIRAL to support generation of efficient modular FFT code**
- **SPIRAL generated code 2-5 times faster than standard implementations**
  - Better memory utilization
  - Reduced loop/recursion overhead
- **Modular code 1.38 times slower than floating point code**
  - Beyond cache boundary modular code becomes faster
- **SPIRAL machinery can be used to obtain vectorized/parallel code**

References

- **[CSSP 90]** J. R. Johnson, R. W Johnson, D. Rodriguez, and R. Tolimieri, **A methodology for designing, modifying, and implementing Fourier Transform algorithms on various architectures**, Circuits Systems Signal Process., (9)4: 449-500, 1990.
- **[IEEE 05]** Markus Püschel, José M. F. Moura, Jeremy Johnson, David Padua, Manuela Veloso, Bryan Singer, Jianxin Xiong, Franz Franchetti, Aca Gacic, Yevgen Voronenko, Kang Chen, Robert W. Johnson and Nicholas Rizzolo, **SPIRAL: Code Generation for DSP Transforms**, Proc. of the IEEE, special issue on "**Program Generation, Optimization, and Adaptation**", Vol. 93, No. 2, pp. 232- 275, 2005.
- **[IPDPS 02]** Kang Chen and Jeremy Johnson, **A Prototypical Self-Optimizing Package for Parallel Implementation of Fast Signal Transforms**, Proc. International Parallel and Distributed Processing Symposium (IPDPS), pp. 58-63, 2002.
- **[SC 06]** Franz Franchetti, Yevgen Voronenko and Markus Püschel, **FFT Program Generation for Shared Memory: SMP and Multicore**, Proc. Supercomputing (SC), 2006.
- **[ISPA 06]** Andreas Bonelli, Franz Franchetti, Juergen Lorenz, Markus Püschel and Christoph W. Ueberhuber, **Automatic Performance Optimization of the Discrete Fourier Transform on Distributed Memory Computers**, Proc. International Symposium on Parallel and Distributed Processing and Application (ISPA), Lecture Notes In Computer Science, Springer, Vol. 4330, pp. 818-832, 2006.

References

- **[DAC 08]** Peter A. Milder, Franz Franchetti, James C. Hoe and Markus Püschel, **Formal Datapath Representation and Manipulation for Implementing DSP Transforms**, Proc. Design Automation Conference (DAC), 2008.
- **[IPDPS 04]** Jeremy Johnson and Kang Chen, **A Self-Adapting Distributed Memory Package for Fast Signal Transforms**, Proc. International Parallel and Distributed Processing Symposium (IPDPS), pp. 44-, 2004.
- **[PLDI 01]** Jianxin Xiong, Jeremy Johnson, Robert W. Johnson and David Padua, **SPL: A Language and Compiler for DSP Algorithms**, Proc. Programming Languages Design and Implementation (PLDI), pp. 298-308, 2001.
- **[PLDI 05]** Franz Franchetti, Yevgen Voronenko and Markus Püschel, **Formal Loop Merging for Signal Transforms**, Proc. Programming Languages Design and Implementation (PLDI), pp. 315-326 , 2005.
- **[MATHCOMP 85]** Peter Montgomery, **Modular Multiplication Without Trial Division**, Mathematics of Computation (MATHCOMP), Vol. 44, pp. 519-521, 1985.
- **[VECPAR 06]** Franz Franchetti, Yevgen Voronenko and Markus Püschel, **A Rewriting System for the Vectorization of Signal Transforms**, Proc. High Performance Computing for Computational Science (VECPAR), Lecture Notes in Computer Science, Springer, Vol. 4395, pp. 363-377, 2006