FFT Program Generation for Shared Memory: SMP and Multicore

F. Franchetti, Y. Voronenko, M. Püschel
Electrical and Computer Engineering
Carnegie Mellon University

Nicola Marcacci Rossi, 31. Oktober 2011

Overview

- First Part: From the algorithm to the code

- Second Part: Spiral extension for Shared Memory

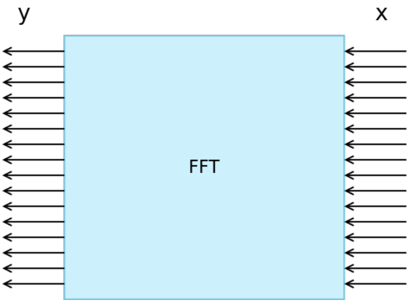
- Discussion

Discrete Fourier Transform

- The problem:

Given: $x \in \mathbb{C}^N$
Compute: $y = \mathrm{DFT}_N x$

$$\mathrm{DFT}_N = [w_N^{kl}]_{0 \leq k,l < N}$$
$$w_n = e^{-2\pi i/N}$$

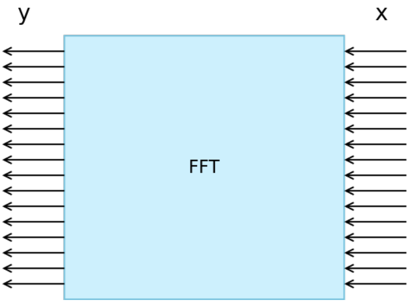


Discrete Fourier Transform

- The problem:

  Given: $x \in \mathbb{C}^N$
  Compute: $y = \mathrm{DFT}_N x$

$$\mathrm{DFT}_N = [w_N^{kl}]_{0 \le k,l < N}$$
$$w_n = e^{-2\pi i/N}$$



- Algorithms:
  - Direct Matrix-Vector Multiplication: $O(N^2)$
  - Fast Fourier Transforms: $O(N \log N)$

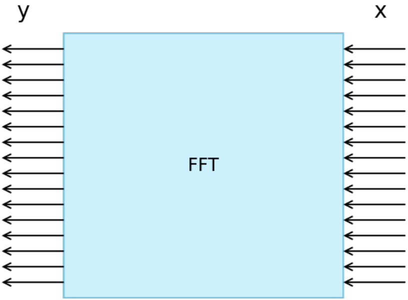
Discrete Fourier Transform

- The problem:

  Given: $x \in \mathbb{C}^N$

  Compute: $y = \mathrm{DFT}_N x$

  $$\mathrm{DFT}_N = [w_N^{kl}]_{0 \leq k,l < N}$$
  $$w_n = e^{-2\pi i/N}$$



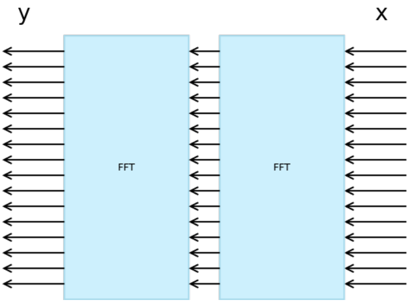
- Algorithms:
  - Direct Matrix-Vector Multiplication: $O(N^2)$
  - Fast Fourier Transforms: $O(N \log N)$
- Cooley-Tukey FFT
  - Divide and Conquer
  - Matrix factorization:
    $\mathrm{DFT}_{mn} x =$
    $(\mathrm{DFT}_m \otimes I_n) D_{m,n} (I_m \otimes \mathrm{DFT}_n) L_m^{mn} x$
  - Assume $N = 2^k$
  - Base case: DFT_2

Discrete Fourier Transform

- The problem:

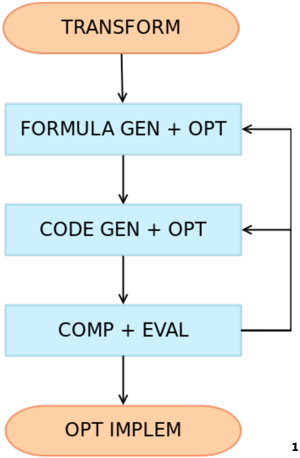
Given: $x \in \mathbb{C}^N$
Compute: $y = \mathrm{DFT}_N x$

$$\mathrm{DFT}_N = [w_N^{kl}]_{0 \le k,l < N}$$
$$w_n = e^{-2\pi i/N}$$



- Algorithms:
  - Direct Matrix-Vector Multiplication: $O(N^2)$
  - Fast Fourier Transforms: $O(N \log N)$
- Cooley-Tukey FFT
  - Divide and Conquer
  - Matrix factorization: $\mathrm{DFT}_{mn} x = (\mathrm{DFT}_m \otimes I_n) D_{m,n} (I_m \otimes \mathrm{DFT}_n) L_m^{mn} x$
  - Assume $N = 2^k$
  - Base case: DFT_2

Spiral: generating optimized implementations



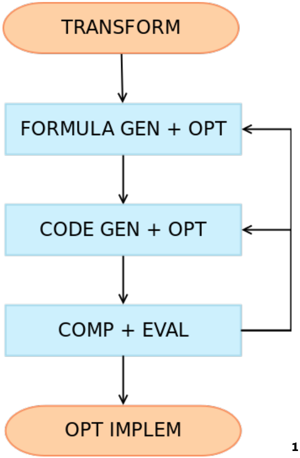
[1]

- Start formula: DFT_N
- Cooley-Tukey Rule: $\mathrm{DFT}_{mn} \rightarrow (\mathrm{DFT}_m \otimes In)D_{m,n}(I_m \otimes \mathrm{DFT}_n)L^{mn}_m$

---

[1] source: FFT Program Generation for Shared Memory: SMP and Multicore

Spiral: generating optimized implementations

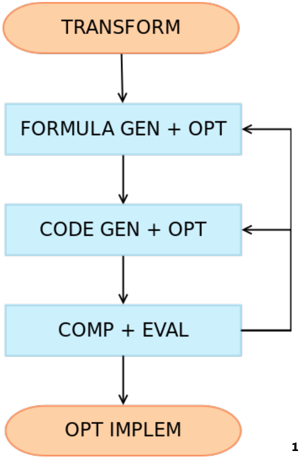


[1]

- Start formula: DFT_N
- Cooley-Tukey Rule:
  $\mathrm{DFT}_{mn} \rightarrow$
  $(\mathrm{DFT}_m \otimes In)D_{m,n}$
  $(I_m \otimes \mathrm{DFT}_n)L_m^{mn}$
- Example:
  $\mathrm{DFT}_8 =$
  $(\mathrm{DFT}_2 \otimes I_4)D_{8,4}$
  $(I_2 \otimes (\mathrm{DFT}_2 \otimes I_2)D_{4,2}$
  $(I_2 \otimes \mathrm{DFT}_2)L_2^4)L_2^9$

[1] source: FFT Program Generation for Shared Memory: SMP and Multicore

Spiral: generating optimized implementations



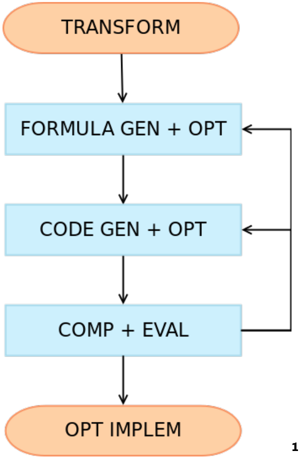
[1]

- Start formula: DFT_N
- Cooley-Tukey Rule:
  $\mathrm{DFT}_{mn} \rightarrow$
  $(\mathrm{DFT}_m \otimes In)D_{m,n}$
  $(I_m \otimes \mathrm{DFT}_n)L_m^{mn}$
- SPL to code translation table:

$y = (A_n B_n)x$	`t[0:1:n-1] = B(x[0:1:n-1]); y[0:1:n-1] = A(t([0:1:n-1]);`
$y = (I_m \otimes A_n)x$	`for (i=0;i<m;i++) y[i*n:1:i*n+n-1] = A(x[i*n:1:i*n+n-1]);`
...	...

[1] source: FFT Program Generation for Shared Memory: SMP and Multicore

Spiral: generating optimized implementations



[1]

- Start formula: DFT_N
- Cooley-Tukey Rule:
  $\mathrm{DFT}_{mn} \rightarrow$
  $(\mathrm{DFT}_m \otimes In)D_{m,n}$
  $(I_m \otimes \mathrm{DFT}_n)L_m^{mn}$
- SPL to code translation table:

$y = (A_n B_n)x$	`t[0:1:n-1] = B(x[0:1:n-1]); y[0:1:n-1] = A(t([0:1:n-1]);`
$y = (I_m \otimes A_n)x$	`for (i=0;i<m;i++) y[i*n:1:i*n+n-1] = A(x[i*n:1:i*n+n-1]);`
...	...

- Search space:
  factorizations and base cases

[1] source: FFT Program Generation for Shared Memory: SMP and Multicore

Extending Spiral for Shared Memory

- New tag:

$$\underbrace{\mathrm{DFT}_N}_{\mathrm{smp}(p,\mu)}$$

  - Number of processors: p
  - Cache line size: μ

- Find parallel Cooley-Tukey:

$$\underbrace{\mathrm{DFT}_{mn}}_{\mathrm{smp}(p,\mu)} \rightarrow \cdots$$

Extending Spiral for Shared Memory

- New tag:

$$\underbrace{\mathrm{DFT}_N}_{\mathrm{smp}(p,\mu)}$$

  - Number of processors: p
  - Cache line size: μ

- Find parallel Cooley-Tukey:

$$\underbrace{\mathrm{DFT}_{mn}}_{\mathrm{smp}(p,\mu)} \rightarrow \cdots$$

- Steps:
  - Identify parallel constructs (and their implementation)
  - Define rewriting rules
  - Derive a parallel Cooley-Tukey

Extending Spiral for Shared Memory

- New tag:

$$\underbrace{\mathrm{DFT}_N}_{\mathrm{smp}(p,\mu)}$$

  - Number of processors: p
  - Cache line size: μ

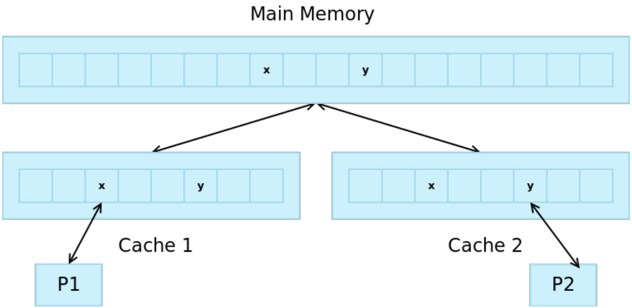
- Find parallel Cooley-Tukey:

$$\underbrace{\mathrm{DFT}_{mn}}_{\mathrm{smp}(p,\mu)} \rightarrow \cdots$$

- Steps:
  - Identify parallel constructs (and their implementation)
  - Define rewriting rules
  - Derive a parallel Cooley-Tukey

- Issues:
  - Load Balancing
  - Synchronization overhead
  - False Sharing

False sharing

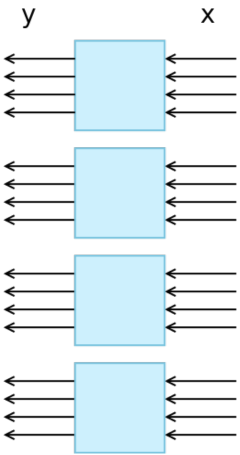


- Circumstances:
  - Different data
  - Same cache line
  - Consecutive accesses
- Leads to cache line thrashing
- Solution: one processor per cache line

Parallel Constructs: Block-Diagonal Products

- Given $A \in \mathbb{C}^{n\mu \times n\mu}$:

$$(I \otimes A)x = \begin{bmatrix} A & & & \\ & A & & \\ & & \ddots & \\ & & & A \end{bmatrix} x$$



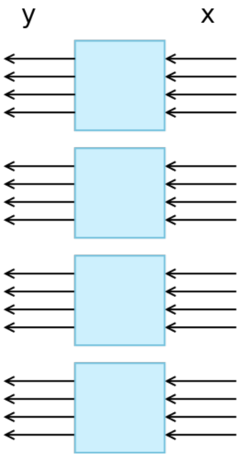
Parallel Constructs: Block-Diagonal Products

- Given $A \in \mathbb{C}^{n\mu \times n\mu}$:

$$(I \otimes A)x = \begin{bmatrix} A & & & \\ & A & & \\ & & \ddots & \\ & & & A \end{bmatrix} x$$

- Multiplication with a Block-Diagonal Matrix

- Embarassingly parallel



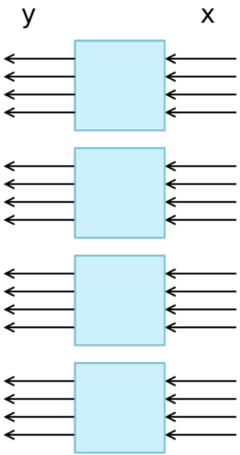
Parallel Constructs: Block-Diagonal Products

- Given $A \in \mathbb{C}^{n\mu \times n\mu}$:

$$(I \otimes A)x = \begin{bmatrix} A & & & \\ & A & & \\ & & \ddots & \\ & & & A \end{bmatrix} x$$

- Multiplication with a Block-Diagonal Matrix
- Embarassingly parallel
- Implementation: OMP for

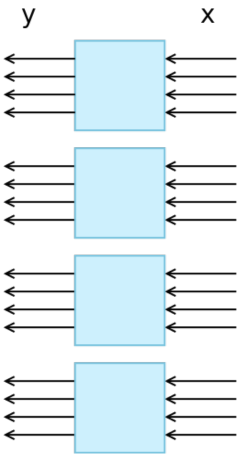
```
#pragma omp parallel for schedule(static)
shared(x, y)
for (i=0; i<p; i++)
    y[i*n:1:i*n-1] = A(x[i*n:1:i*n+n-1]);
```



Parallel Constructs: Block-Diagonal Products

- Given $A_i \in \mathbb{C}^{n\mu \times n\mu}$:

$$\left(\bigoplus_{i=0}^{p-1} A_i\right) x = \begin{bmatrix} A_1 & & & \\ & A_2 & & \\ & & \ddots & \\ & & & A_n \end{bmatrix} x$$

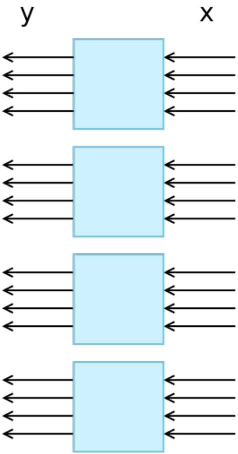


Parallel Constructs: Block-Diagonal Products

- Given $A_i \in \mathbb{C}^{n\mu \times n\mu}$:

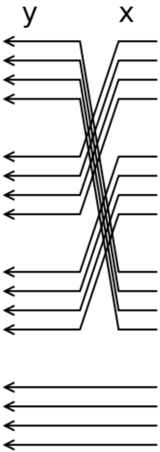
$$\left(\bigoplus_{i=0}^{p-1} A_i\right) x = \begin{bmatrix} A_1 & & & \\ & A_2 & & \\ & & \ddots & \\ & & & A_n \end{bmatrix} x$$

- Multiplication with a Block-Diagonal Matrix
- Embarassingly parallel
- Implementation: OMP for



Parallel Constructs: "Reordering" Cache Lines

- $(P \otimes I_\mu)$
  P a permutation matrix



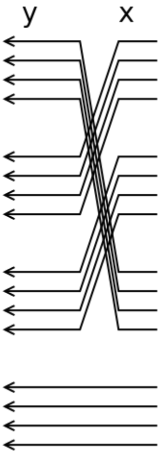
Parallel Constructs: "Reordering" Cache Lines

- $(P \otimes I_\mu)$
  P a permutation matrix

- Example:

$$P = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$(P \otimes I_\mu) = \begin{bmatrix} & I_\mu & & \\ & & I_\mu & \\ I_\mu & & & \\ & & & I_\mu \end{bmatrix}$$



Parallel Constructs: "Reordering" Cache Lines

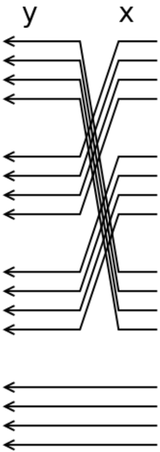
- $(P \otimes I_\mu)$
  P a permutation matrix

- Example:

$$P = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$(P \otimes I_\mu) = \begin{bmatrix} & I_\mu & & \\ & & I_\mu & \\ I_\mu & & & \\ & & & I_\mu \end{bmatrix}$$

- Implementation (no data permutation!):



Parallel Constructs: "Reordering" Cache Lines

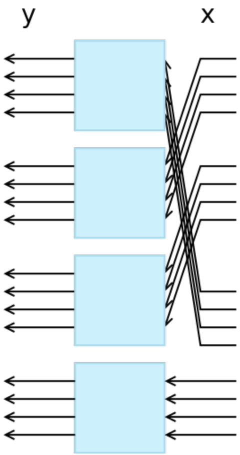
- $(P \otimes I_\mu)$
  P a permutation matrix

- Example:

$$P = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$(P \otimes I_\mu) = \begin{bmatrix} & I_\mu & & \\ & & I_\mu & \\ I_\mu & & & \\ & & & I_\mu \end{bmatrix}$$

- Implementation (no data permutation!):
  - Modify access pattern of next construct



Rewriting Rules

- Identify appropriate Rewriting Rules
  - Transform Cooley-Tukey FFT to a Multicore version
  - Don't exist for all Transforms
  - They exist for FFT: a major contribution of the paper

Rewriting Rules

- Identify appropriate Rewriting Rules
  - Transform Cooley-Tukey FFT to a Multicore version
  - Don't exist for all Transforms
  - They exist for FFT: a major contribution of the paper

$$\underbrace{AB}_{\text{smp}(p,\mu)} \to \underbrace{A}_{\text{smp}(p,\mu)} \underbrace{B}_{\text{smp}(p,\mu)} \tag{1}$$

$$\underbrace{A_m \otimes I_n}_{\text{smp}(p,\mu)} \to \underbrace{(L_m^{mp} \otimes I_{n/p})(I_p \otimes (A_m \otimes I_{n/p}))(L_p^{mp} \otimes I_{n/p})}_{\text{smp}(p,\mu)} \tag{2}$$

$$\underbrace{L_m^{mn}}_{\text{smp}(p,\mu)} \to \begin{cases} \underbrace{(I_p \otimes L_{m/p}^{mn/p})}_{\text{smp}(p,\mu)} \underbrace{(L_p^{pn} \otimes I_{m/p})}_{\text{smp}(p,\mu)} \\ \underbrace{(L_m^{pm} \otimes I_{n/p})}_{\text{smp}(p,\mu)} \underbrace{(I_p \otimes L_m^{mn/p})}_{\text{smp}(p,\mu)} \end{cases} \tag{3}$$

$$\underbrace{I_m \otimes A_n}_{\text{smp}(p,\mu)} \to I_p \otimes (I_{m/p} \otimes A_n) \tag{4}$$

$$\underbrace{(P \otimes I_n)}_{\text{smp}(p,\mu)} \to (P \otimes I_{n/\mu}) \otimes I_\mu \tag{5}$$

$$\underbrace{D}_{\text{smp}(p,\mu)} \to \bigoplus_{i=0}^{p-1} D_i \tag{6}$$

The Result: A Multicore Cooley-Tukey FFT

- Recall the Cooley-Tukey FFT rule:

$$\mathrm{DFT}_{mn} \rightarrow (\mathrm{DFT}_m \otimes I_n) D_{m,n} (I_m \otimes \mathrm{DFT}_n) L_m^{mn}$$

The Result: A Multicore Cooley-Tukey FFT

- Recall the Cooley-Tukey FFT rule:

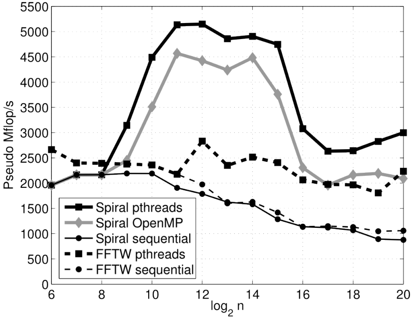
$$\mathrm{DFT}_{mn} \rightarrow (\mathrm{DFT}_m \otimes I_n) D_{m,n} (I_m \otimes \mathrm{DFT}_n) L_m^{mn}$$

- Cooley-Tukey FFT adapted for Shared Memory:

$$\underbrace{\mathrm{DFT}_{mn}}_{\mathrm{smp}(p,\mu)} \rightarrow$$

$$((L_m^{mp} \otimes I_{n/p\mu}) \otimes I_\mu)(I_p \otimes (\mathrm{DFT}_m \otimes I_{n/p}))((L_p^{mp} \otimes I_{n/p\mu}) \otimes I_\mu)$$
$$\left(\bigoplus_{i=0}^{p-1} D_{m,n}^i\right)$$
$$(I_p \otimes (I_{m/p} \otimes \mathrm{DFT}_n))(I_p \otimes L_{m/p}^{mn/p})$$
$$((L_p^{pn} \otimes I_{m/p\mu}) \otimes I_\mu)$$

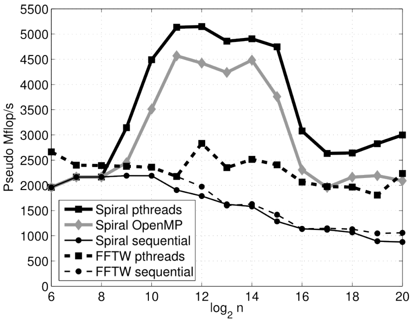
Discussion



2.2 GHz Opteron Dual-core (4 processors) [2]

[2] source: FFT Program Generation for Shared Memory: SMP and Multicore

Discussion

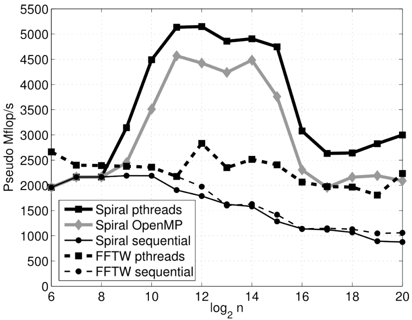


2.2 GHz Opteron Dual-core (4 processors) [2]

- Peculiarities of FFTW
  - State-of-the-art multithreading DFT implementation
  - Optimized for big problem sizes and many processors (overhead)
  - Does not use μ, p explicitly

[2] source: FFT Program Generation for Shared Memory: SMP and Multicore

Discussion



2.2 GHz Opteron Dual-core (4 processors) [2]

- Peculiarities of FFTW
  - State-of-the-art multithreading DFT implementation
  - Optimized for big problem sizes and many processors (overhead)
  - Does not use μ, p explicitly
- Advantages of Spiral
  - Enables high-level optimizations and reasoning
  - No need for loop analysis
  - Automatizes implementation

[2] source: FFT Program Generation for Shared Memory: SMP and Multicore