Recursive Cooley-Tukey FFT

\[
\text{DFT}_{km} = (\text{DFT}_k \otimes I_m) T_{m}^{km} (I_k \otimes \text{DFT}_m) L_{km}^{k}
\]
\[
\text{DFT}_{km} = L_{m}^{km} (I_k \otimes \text{DFT}_m) T_{m}^{km} (\text{DFT}_k \otimes I_m)
\]

- For powers of two \( n = 2^t \) sufficient together with base case

\[
\text{DFT}_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}
\]
Example FFT, \( n = 16 \) (*Recursive, Radix 4*)

\[
\text{DFT}_{16} = \text{DFT}_4 \otimes I_4 \quad T_4^{16} \quad I_4 \otimes \text{DFT}_4 \quad L_4^{16}
\]

Fast Implementation (≈ FFTW 2.x)

- Choice of algorithm
- Locality optimization
- Constants
- Fast basic blocks
- Adaptivity
1: Choice of Algorithm

- Choose recursive, not iterative

\[
\text{DFT}_{km} = (\text{DFT}_k \otimes \text{I}_m) \tau_k^{km} (\text{I}_k \otimes \text{DFT}_m) L_k^{km}
\]

Radix 2, recursive

Radix 2, iterative

2: Locality Improvement

\[
\text{DFT}_{16} = \text{DFT}_4 \otimes \text{I}_4 \quad T_4^{16} \quad I_4 \otimes \text{DFT}_4 \quad L_4^{16}
\]
\[
\text{DFT}_{km} = (\text{DFT}_{k} \otimes I_{m}) T_{m}^{km}(I_{k} \otimes \text{DFT}_{m}) L_{k}^{km}
\]

one loop  

one loop

// code sketch
void DFT(int n, cpx *x, cpx *y) {
  int k = choose_dft_radix(n); // ensure k <= 32
  
  for (int i = 0; i < k; ++i)
    DFTrec(m, x + i, y + m*i, k, 1); // implemented as DFT(…)
  for (int j = 0; j < m; ++j)
    DFTscaled(k, y + j, t[j], m); // always a base case
}

3: Constants

- FFT incurs multiplications by roots of unity
- In real arithmetic: Multiplications by sines and cosines, e.g.,
  \[
y[i] = \sin(i \cdot \pi/128) \cdot x[i];
  \]
  Very expensive!

- **Observation**: Constants depend only on input size, not on input
- **Solution**: Precompute once and use many times
  
d = DFT_init(1024); // init function computes constant table
d(x, y); // use many times
4: Optimized Basic Blocks

Just like loops can be unrolled, recursions can also be unrolled.

Empirical study: Base cases for sizes $n \leq 32$ useful (scalar code).

Needs 62 base cases or “codelets” (why?)
- $DFT_{rec}$, sizes 2–32
- $DFT_{scaled}$, sizes 2–32

Solution: Codelet generator (codelet = optimized basic block)

```c
void DFT(int n, cpx *x, cpx *y) {
    int k = choose_dft_radix(n); // ensure $k \leq 32$
    if (use_base_case(n))
        DFTbc(n, x, y); // use base case
    else {
        for (int i = 0; i < k; ++i)
            DFTrec(m, x + i, y + m*i, k, 1); // implemented as DFT(\ldots) is
        for (int j = 0; j < m; ++j)
            DFTscaled(k, y + j, t[j], m); // always a base case
    }
}
```

FFT W Codelet Generator

- $n$ 
  - $n$ Codelet Generator
    - FFT Codelet
    - Twiddle Codelet for $DFT_n$
  - DAG Generator
    - DAG
  - Scheduler
Small Example DAG

DAG:

One possible unparsing:

\[ f_{0} = x[0] - x[3] ; \]
\[ f_{1} = x[0] + x[3] ; \]
\[ f_{2} = x[1] - x[2] ; \]
\[ f_{3} = x[1] + x[2] ; \]
\[ f_{4} = f_{1} - f_{3} ; \]
\[ y[0] = f_{1} + f_{3} ; \]
\[ y[2] = 0.7071067811865476 \times f_{4} ; \]
\[ f_{7} = 0.9238795325112867 \times f_{0} ; \]
\[ f_{8} = 0.3826834323650898 \times f_{2} ; \]
\[ y[1] = f_{7} + f_{8} ; \]
\[ f_{10} = 0.3826834323650898 \times f_{0} ; \]
\[ f_{11} = (-0.9238795325112867) \times f_{2} ; \]
\[ y[3] = f_{10} + f_{11} ; \]

DAG Generator

- Knows FFTs: Cooley-Tukey, split-radix, Good-Thomas, Rader, represented in sum notation

\[ y_{n_{2}j_{1}+j_{2}} = \sum_{k_{1}=0}^{n_{1}-1} (\omega_{n_{1}}^{j_{2}k_{1}}) \left( \sum_{k_{2}=0}^{n_{2}-1} x_{n_{1}k_{2}+k_{1}} \omega_{n_{2}}^{j_{2}k_{2}} \right) \omega_{n_{1}}^{j_{1}k_{1}} \]

- For given \( n \), suitable FFTs are recursively applied to yield \( n \) (real) expression trees for outputs \( y_{0} \ldots, y_{n-1} \)
- Trees are fused to an (unoptimized) DAG
**Simplifier**

- **Blackboard**
- **Applies:**
  - Algebraic transformations
  - Common subexpression elimination (CSE)
  - DFT-specific optimizations
- **Algebraic transformations**
  - Simplify mults by 0, 1, -1
  - Distributivity law: \(kx + ky = k(x + y), \ kx + lx = (k + l)x\)
    - Canonicalization: \((x-y), (y-x)\) to \((x-y), -(x-y)\)
- **CSE: standard**
  - E.g., two occurrences of \(2x+y\): assign new temporary variable
- **DFT specific optimizations**
  - All numeric constants are made positive (reduces register pressure)
  - CSE also on transposed DAG

---

**Scheduler**

- **Determines in which sequence the DAG is unparsed to C**
  - (topological sort of the DAG)
  - **Goal: minimizer register spills**
- A 2-power FFT has an operational intensity of \(I(n) = O(\log(C))\), where \(C\) is the cache size [1]
- Implies: For \(R\) registers \(\Omega(n \log(n)/\log(R))\) register spills
- FFTW’s scheduler achieves this (asymptotic) bound \textit{independent} of \(R\)
- **Blackboard**

typedef struct {
    double* input;
    double* output;
} spiral_t;

cast double x708[] = { 1.0, 0.9238795325112867, 0.7071067811865476, 0.3826834323650898,
    const double x709[] = { -0.0, 0.3826834323650898, 0.7071067811865476, 0.9238795325112867, 1.0, 0.9238795325112867, 0.7071067811865476 }

cast staged(spiral_t *, * x0) {
    double* x1 = x0->input;
    double* x2 = x0->output;
    double x6 = x1[0];
    double x22 = x1[16];
    double x38 = x6 + x22;
    double x14 = x1[8];
    double x30 = x1[24];
    double x46 = x14 + x30;
    double x343 = x38 + x46;
    double x10 = x1[4];
    double x26 = x1[20];
    double x42 = x10 + x26;
    double x18 = x1[12];
    double x34 = x1[28];
    double x50 = x18 + x34;
    double x344 = x42 + x50;
    double x345 = x343 + x344;
    double x8 = x1[2];
    double x24 = x1[18];
    double x115 = x8 + x24;
    double x16 = x1[26];
    double x119 = x16 + x115;
    double x112 = x115 + x119;
    double x127 = x80 + x112;
    double x12 = x1[6];
    double x28 = x1[22];
    double x119 = x12 + x28;
    double x20 = x1[30];
    double x127 = x12 + x20;
    double x28 = x1[32];
    double x124 = x34 + x127;
    double x12 = x1[0];
    double x28 = x1[1];
    double x32 = x1[5];
    double x47 = x12 + x32;
    double x76 = x39 + x47;
    double x11 = x1[10];
    double x31 = x1[25];
    double x43 = x11 + x31;
    double x80 = x43 + x51;
    double x88 = x76 + x80;
    double x9 = x1[3];
    double x24 = x1[19];
    double x39 = x9 + x24;
    double x15 = x1[27];
    double x32 = x9 + x15;
    double x11 = x1[19];
    double x51 = x32 + x11;
    double x27 = x1[29];
    double x43 = x11 + x27;
    double x16 = x1[15];
    double x30 = x1[26];
    double x50 = x16 + x30;
    double x88 = x50 + x15;
    double x80 = x43 + x16;
    double x80 = x39 + x16;
    double x34 = x1[28];
    double x12 = x1[12];
    double x28 = x1[24];
    double x119 = x12 + x28;
    double x36 = x19 + x119;
    double x42 = x36 + x119;
    double x344 = x42 + x50;
    double x345 = x343 + x344;
    double x346 = x8 + x345;
    double x347 = x346 + x345;
    double x348 = x346 + x345;
    double x349 = x346 + x348;
    x2[0] = x349;
}

FFT, n = 16

First cut

4 independent components
4 independent components
Codelet Examples

- Notwiddle 2
- Notwiddle 3
- Twiddle 3
- Notwiddle 32

- Code style:
  - Single static assignment (SSA)
  - Scoping (limited scope where variables are defined)

5: Adaptivity

// code sketch
void DFT(int n, cpx *x, cpx *y) {
  int k = choose_dft_radix(n); // ensure k <= 32
  if (use_base_case(n))
    DFTbc(n, x, y); // use base case
  else {
    for (int i = 0; i < k; ++i)
      DFTrec(m, x + i, y + m*i, k, 1); // implemented as DFT
    for (int j = 0; j < m; ++j)
      DFTscaled(k, y + j, t[j], m); // always a base case
  }
}

Choices used for platform adaptation

d = DFT_init(1024); // compute constant table; search for best recursion
d(x, y); // use many times

- Search strategy: Dynamic programming
- Blackboard
<table>
<thead>
<tr>
<th></th>
<th>MMM Atlas</th>
<th>Sparse MVM Sparsity/Bebop</th>
<th>DFT FFTW</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cache optimization</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Register optimization</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Optimized basic blocks</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Other optimizations</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Adaptivity</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>MMM Atlas</th>
<th>Sparse MVM Sparsity/Bebop</th>
<th>DFT FFTW</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cache optimization</td>
<td>Blocking</td>
<td>Blocking (rarely useful)</td>
<td>Recursive FFT, fusion of steps</td>
</tr>
<tr>
<td>Register optimization</td>
<td>Blocking</td>
<td>Blocking (changes sparse format)</td>
<td>Scheduling of small FFTs</td>
</tr>
<tr>
<td>Optimized basic blocks</td>
<td>Unrolling, scalar replacement and SSA, scheduling, simplifications (for FFT)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Other optimizations</td>
<td></td>
<td></td>
<td>Precomputation of constants</td>
</tr>
<tr>
<td>Adaptivity</td>
<td>Search: blocking parameters</td>
<td>Search: register blocking size</td>
<td>Search: recursion strategy</td>
</tr>
</tbody>
</table>