# FFT Compiler:

## From Math to Efficient Hardware

James C. Hoe
Department of ECE
Carnegie Mellon University

joint work with Peter A. Milder, Franz Franchetti, and
Markus Pueschel in the SPIRAL project

with support from NSF ACR-0234293, ITR/NGS-0325687, DARPA NBCH-105000,Intel

# The SPIRAL Project

- High performance implementations of linear DSP transforms (DFT, DCT, DWT, filters, etc) are an important class of design problems
- Hand design and tuning is tricky and expensive
  - needs both math and implementation knowledge
  - time-consuming and tedious
  - needs to repeat effort for every new context
- SPIRAL research goal: A flexible push-button design generation framework that produces SW and HW implementations equal or better than expert hand design

**Today I focus on FFT for HW**

# Why we can do better than hand design

- SPIRAL is only focused on linear DSP transforms
- These transforms are highly structured, highly regular and very well understood mathematically
- Algorithmic implementations of a transform can be enumerated following a known set of rules
- For a given objective function and mapping target, a computer generates a solution at least as good as the best human effort—by trying enough implementations



# Outline

- SPIRAL Formula Framework
- SPIRAL for HW FFT cores
- Conclusions

# Linear Transforms

- Linear transform is matrix-vector multiplication
  - computing by definition takes $O(N^2)$ operations
  - the matrix has structure
- E.g. discrete Fourier transform: $y = DFT_N \cdot x$

$$\begin{bmatrix} y_0 \\ y_1 \\ \cdot \\ y_j \\ \cdot \\ \cdot \\ y_{N-1} \end{bmatrix} = \begin{bmatrix} & k \to 0 \,..\, N\text{-}1 & \\ j \to 0 \,..\, N\text{-}1 & e^{-i2\pi jk/N} & \end{bmatrix} \cdot \begin{bmatrix} x_0 \\ x_1 \\ \cdot \\ x_k \\ \cdot \\ \cdot \\ x_{N-1} \end{bmatrix}$$



# "Fast" Algorithms

- a "fast" algorithm factors the matrix into a sequence of structured, sparse matrices

  *cheaper sparse multiplies* $\Rightarrow O(N \log(N))$ *operations*
- E.g. Cooley-Tukey Factorization of $DFT_4$

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

- Matrix formula representation

$$DFT_{n \cdot m} = \left(DFT_n \otimes I_m\right) D_n^{n \cdot m} \left(I_n \otimes DFT_m\right) L_n^{n \cdot m}$$

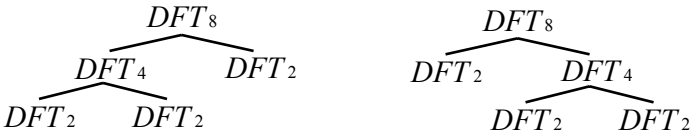
SPIRAL www.spiral.net

# "Fast" Fourier Transform Algorithms

- Recursively factorize by the Cooley-Tukey rule until only leaf cases remain (e.g. $DFT_r$ for radix-r)

$$DFT_8 = (DFT_2 \otimes I_4) D_2^8 (I_2 \otimes DFT_4) L_2^8$$
$$= (DFT_2 \otimes I_4) D_2^8 (I_2 \otimes ((DFT_2 \otimes I_2) D_2^4 (I_2 \otimes DFT_2) L_2^4)) L_2^8$$

- Exponential number of alternatives



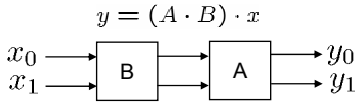

- Each ruletree corresponds a different algorithm
- All cost $O(N \log(N))$

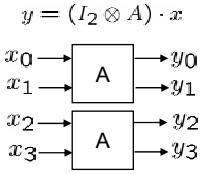


SPIRAL www.spiral.net

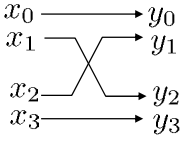
# Formula to HW (Combinational)

- Given $y = M \cdot x$ where $M$ is:
  - $M = A \cdot B$ — apply $B$, then $A$
  - $M$ is a permutation — permute $x$
  - $M = I_n \otimes A$ — apply $A$, $n$ times in parallel
  - $M$ is a diagonal — scale $x$

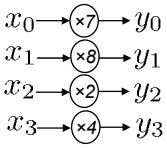



$$y = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot x$$




$$y = \begin{bmatrix} 7 & 0 & 0 & 0 \\ 0 & 8 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 4 \end{bmatrix} \cdot x$$



# How about good HW?

- Matrix formulas have a natural mapping to dataflow and hence combinational datapath
- However, real hardware designs must fit a given resource constraint

  $\Rightarrow$ *sequential datapath that reuse available HW*
  - identify repeated kernels
  - instantiate kernels under resource constraints
  - schedule computation to reuse instantiated kernels

*We want to do the analysis, mapping and scheduling at formula level, with high-level algorithm knowledge*



# Regular Structure for HW

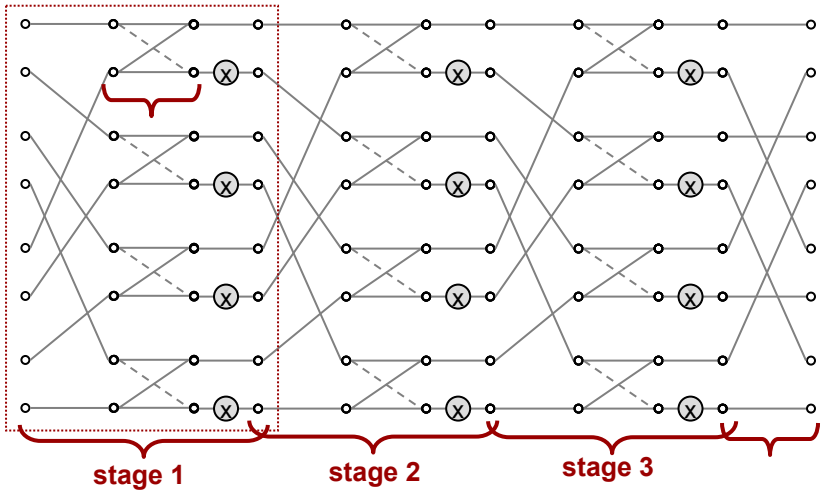
- Simple regular structure embodied in Pease FFT

$$\mathrm{DFT}_{2^k} = R_{2^k} \left( \underbrace{\prod_{i=0}^{k-1}} T_i \underbrace{(I_{2^{k-1}} \otimes F_2)} L^{2^k}_{2^{k-1}} \right)$$

- Example:

$$\mathrm{DFT}_8 = R_8 \left(T_0(I_4 \otimes F_2)L^8_4\right) \left(T_1(I_4 \otimes F_2)L^8_4\right) \left(T_2(I_4 \otimes F_2)L^8_4\right)$$

# Pease DFT Example: $DFT_8$

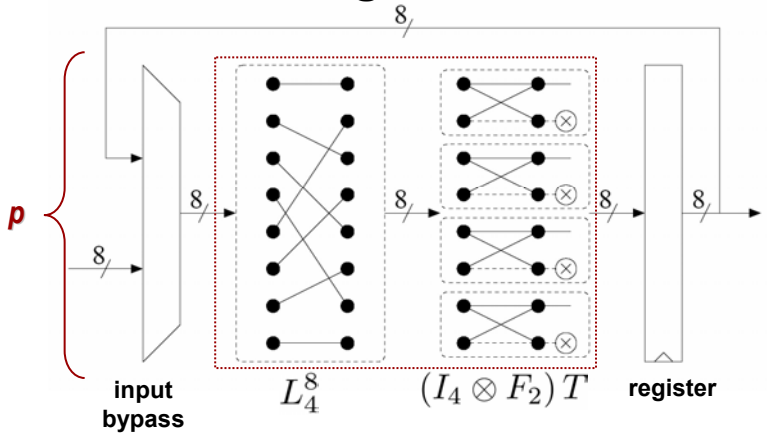



$$\mathrm{DFT}_8 = R_8 \cdot T_2(I_4 \otimes F_2)L_4^8 \cdot T_1(I_4 \otimes F_2)L_4^8 \cdot T_0(I_4 \otimes F_2)L_4^8$$

(formula is applied from right to left)

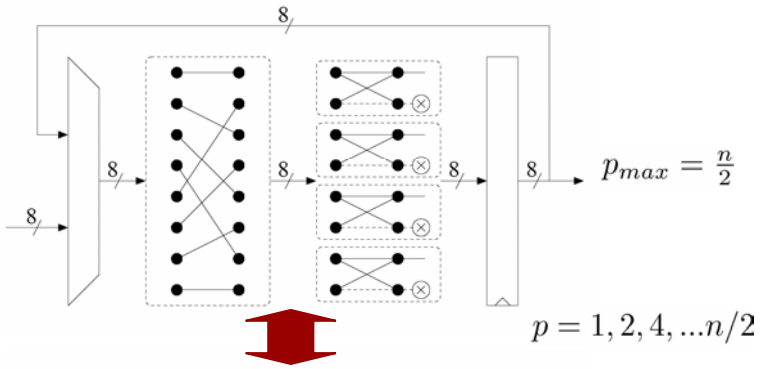


# Horizontal folding




- a baseline datapath for DFT *(sans bit-reverse)*
- degree of freedom: vertical parallelism
  - parameter *p*

# Vertical (V-)folding according to *p*




**cost** $\propto p$ **latency** $\propto \frac{1}{p}$

**Fine-grained control over cost/latency tradeoff**



# DFTgen [Milder, et al., DAC'05, FPGA'06]

Input form

| | parameter | value | range | usage |
|---|---|---|---|---|
| functionality | n | 2048 | 4–16384 | DFT size (?) |
| | data width | 16 | 6–32 | datapath width, fixed point precision (?) |
| | twiddle width | 16 | 2–16 | twiddle factor bitwidth (?) |
| | data ordering | bit-reversed out | | bit-reversed input or output (?) |
| | scaling mode | unscaled | | unscaled or scaled arithmetic (?) |
| control area/latency tradeoff | p | 1 | 1–1024 | degree of parallelism (?) |
| | twiddle storage | block RAM | | store twiddles in dist. RAM or BRAM (?) |
| | FIFO threshold | Select | 2–1024 | FIFO of >= depth will be built as BRAM. (?) |

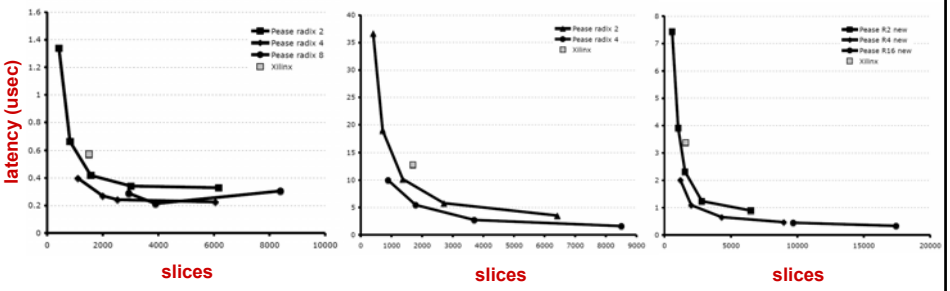
Generate Verilog Reset

Core characteristics and resource usage (dynamically updated)

resource model

| char./res. | value | how obtained |
|---|---|---|
| cycles | 11297 | exact formula |
| slices | 876 | estimated with linear fit formula |
| BRAM | 44 | exact formula |
| multipliers | 4 | exact formula |
| num. error | | to be implemented |

# Cost Performance Tradeoff




$DFT_{64}$ $DFT_{256}$ $DFT_{1024}$

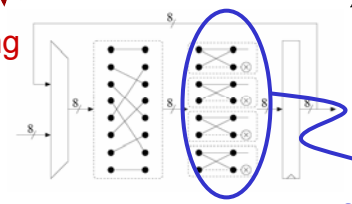


# Structures of Interest for Parameterized Cores

- Simple regular structure embodied in formula

$$\mathrm{DFT}_{2^k} = R_{2^k} \left( \prod_{i=0}^{k-1} T_i^{2^k} (I_{2^{k-1}} \otimes F_2) L_{2^{k-1}}^{2^k} \right)$$

- Product: Horizontal folding



- Tensor Product: parallelism or streaming vertical folding


- Stride permutation: rewiring or buffer/delay

# Applicability to other transforms?

- DFT radix 2 $$R_{2^k} \prod_{i=0}^{k-1} \left[ T_i \left( I_{2^{k-1}} \otimes DFT_2 \right) L^{2^k}_{2^{k-1}} \right]$$
- DFT radix $2^r$ $$R_{2^k} \prod_{i=0}^{k/r-1} \left[ T_i \left( I_{2^{k-r}} \otimes DFT_{2^r} \right) L^{2^k}_{2^{k-r}} \right]$$
- 2-D $DFT_{nxn}$ $$\prod_{i=0}^{1} \left[ L^{n^2}_n \left( I_n \otimes DFT_n \right) \right]$$
- WHT $$\prod_{i=0}^{k/r-1} \left[ \left( I_{2^{k-r}} \otimes WHT_{2^r} \right) L^{2^k}_{2^{k-r}} \right]$$
- DCT (type II) $$DP \prod_{i=0}^{k-1} \left[ A_{k-i} L^{2^k}_2 \right] L^{2^k}_{2^{k-1}} L^{2^k}_{2^{k-1}} P^H_{2^k}$$



# Conclusions

- Mathematical approach to DSP transform implementation
- Encapsulating domain knowledge in a domain specific tool for **100% automatic** design generation and optimization
- Generalizable to other linear DSP transforms
- Thank you (Arvind)