Optimizing the Fast Fourier Transform over Memory Hierarchies for Embedded Digital Systems: A Fully In-Cache Algorithm

James E. Raynolds, College of Nanoscale Science and Engineering

Lenore Mullin, Computer Science

University at Albany, State University of New York, Albany, NY 12309

New FFT algorithm for embedded systems

- **Maximize in-cache operations through use of repeated transpose-reshape operations**
- **Similar to partitioning for parallel implementation**
- **Do as many operations in cache as possible**
- **Re-materialize the array to achieve locality**
- **Continue processing in cache and repeat process**

Example

- **Assume cache size c = 4; input vector length n = 32; number of rows r = n/c = 8**

- **Generate vector of indices:**

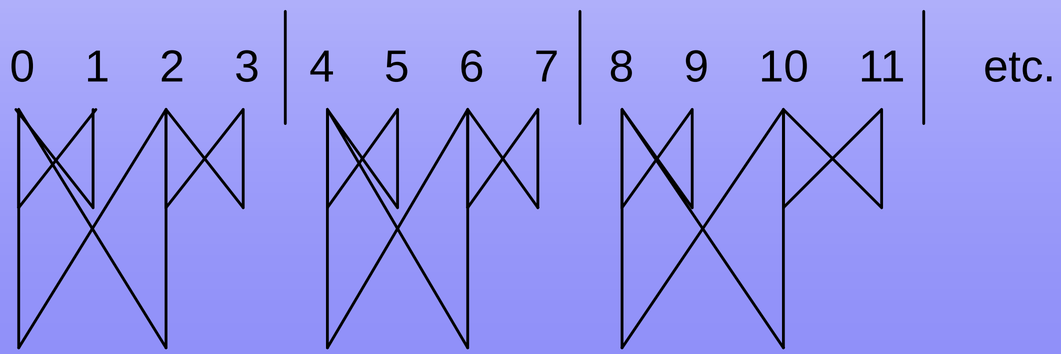
$$v = i(n) = < 0\ 1\ 2 \ldots 31 >$$

- **Use re-shape operator r to generate a matrix**

Starting Matrix

- **Each row is of length equal to the cache size**
- **Standard butterfly applied to each row as...**

$$A \equiv \langle rc \rangle \hat{\rho} v = \begin{bmatrix} 0 & 1 & 2 & 3 \\ 4 & 5 & 6 & 7 \\ 8 & 9 & 10 & 11 \\ 12 & 13 & 14 & 15 \\ 16 & 17 & 18 & 19 \\ 20 & 21 & 22 & 23 \\ 24 & 25 & 26 & 27 \\ 28 & 29 & 30 & 31 \end{bmatrix}$$



Next transpose

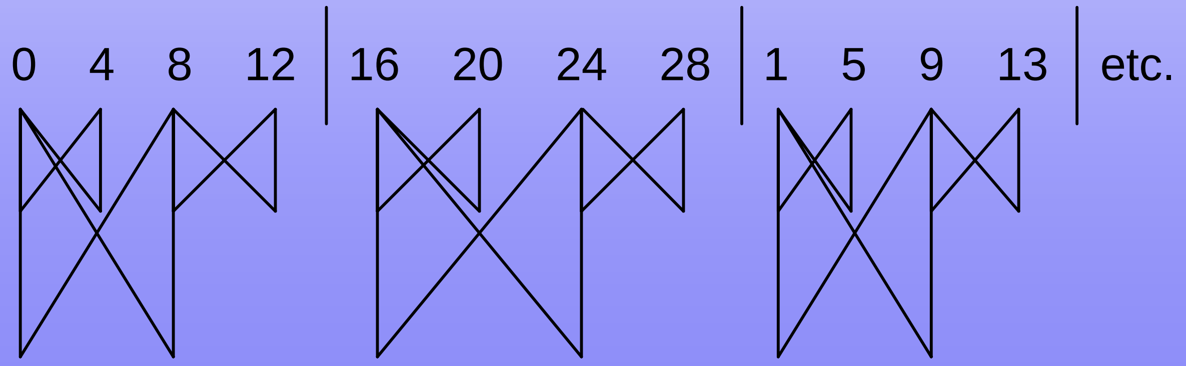
- **To continue further would induce cache misses so transpose and reshape.**
- **Transpose-reshape operation composed over indices (only result is materialized.**
- **The transpose is:**

$$A^T = \begin{bmatrix} 0 & 4 & 8 & 12 & 16 & 20 & 24 & 28 \\ 1 & 5 & 9 & 13 & 17 & 21 & 25 & 29 \\ 2 & 6 & 10 & 14 & 18 & 22 & 26 & 30 \\ 3 & 7 & 11 & 15 & 19 & 23 & 27 & 31 \end{bmatrix}$$

Resulting Transpose-Reshape

- **Materialize the transpose-reshaped array B**
- **Carry out butterfly operation on each row**
- **Weights are re-ordered**
- **Access patterns are standard...**

$$B \equiv \langle rc \rangle \hat{\rho}(A^T) = \begin{bmatrix} 0 & 4 & 8 & 12 \\ 16 & 20 & 24 & 28 \\ 1 & 5 & 9 & 13 \\ 17 & 21 & 25 & 29 \\ 2 & 6 & 10 & 14 \\ 18 & 22 & 26 & 30 \\ 3 & 7 & 11 & 15 \\ 19 & 23 & 27 & 31 \end{bmatrix}$$



Transpose-Reshape again

- **As before: to proceed further would induce cache misses so:**
- **Do the transpose-reshape again (composing indices)**
- **The transpose is:**

$$B^T = \begin{bmatrix} 0 & 16 & 1 & 17 & 2 & 18 & 3 & 19 \\ 4 & 20 & 5 & 21 & 6 & 22 & 7 & 23 \\ 8 & 24 & 9 & 25 & 10 & 26 & 11 & 27 \\ 12 & 28 & 13 & 29 & 14 & 30 & 15 & 31 \end{bmatrix}$$

Last step (in this example)

- **Materialize the composed transpose-reshaped array C**
- **Carry out the last step of the FFT**
- **This last step corresponds to cycles of length 2 involving elements 0 and 16, 1 and 17, etc.**

$$C \equiv \langle rc \rangle \hat{\rho}(B^T) = \begin{bmatrix} 0 & 16 & 1 & 17 \\ 2 & 18 & 3 & 19 \\ 4 & 20 & 5 & 21 \\ 6 & 22 & 7 & 23 \\ 8 & 24 & 9 & 25 \\ 10 & 26 & 11 & 27 \\ 12 & 28 & 13 & 29 \\ 14 & 30 & 15 & 31 \end{bmatrix}$$

Summary

- **All operations have been carried out in cache at the price of re-arranging the data**
- **Data blocks can be of any size (powers of the radix): need not equal the cache size**
- **Optimum performance: tradeoff between reduction of cache misses and cost of transpose-reshape operations**
- **Number of transpose-reshape operations determined by the data block size (cache size)**