Very Large Fast DFT (VL FFT) Implementation on KeyStone

Multicore Applications

Outlines

- Basic Algorithm for Parallelizing DFT
- Multi-core Implementation of DFT
- Review Benchmark Performance

Goals and Requirements

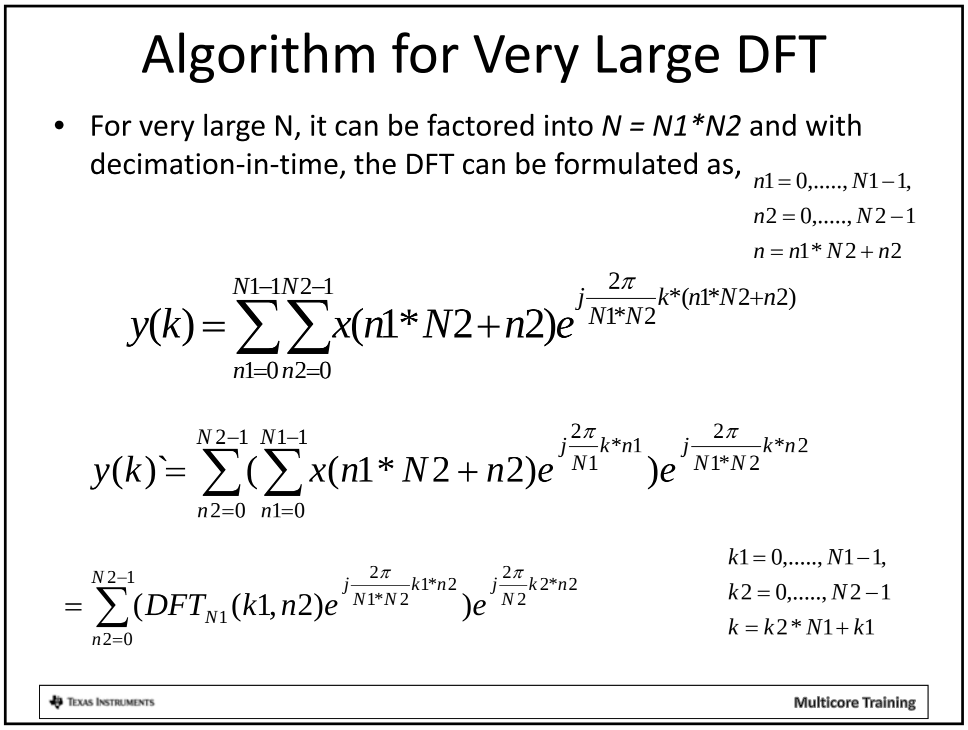
- **Goal:**
  - To implement very large floating point fast DFT on TI C66x multicore devices
- **Requirements:**
  - FFT sizes: 4K – 1M samples
  - Configurable to run on different number of cores: 1, 2, 4, 8
  - High performance

Algorithm for Very Large DFT

- A generic discrete Fourier transform (DFT) is shown below,

$$y(n) = \sum_{n=0}^{N-1} x(n) e^{j\frac{2\pi}{N}k*n} \qquad k = 0, \ldots, N-1$$

- Here *N* is the total size of DFT ,



Algorithm for Very Large DFT

- For very large N, it can be factored into *N = N1*N2* and with decimation-in-time, the DFT can be formulated as,

$$n1 = 0,\ldots\ldots, N1-1,\quad n2 = 0,\ldots\ldots, N2-1,\quad n = n1*N2+n2$$

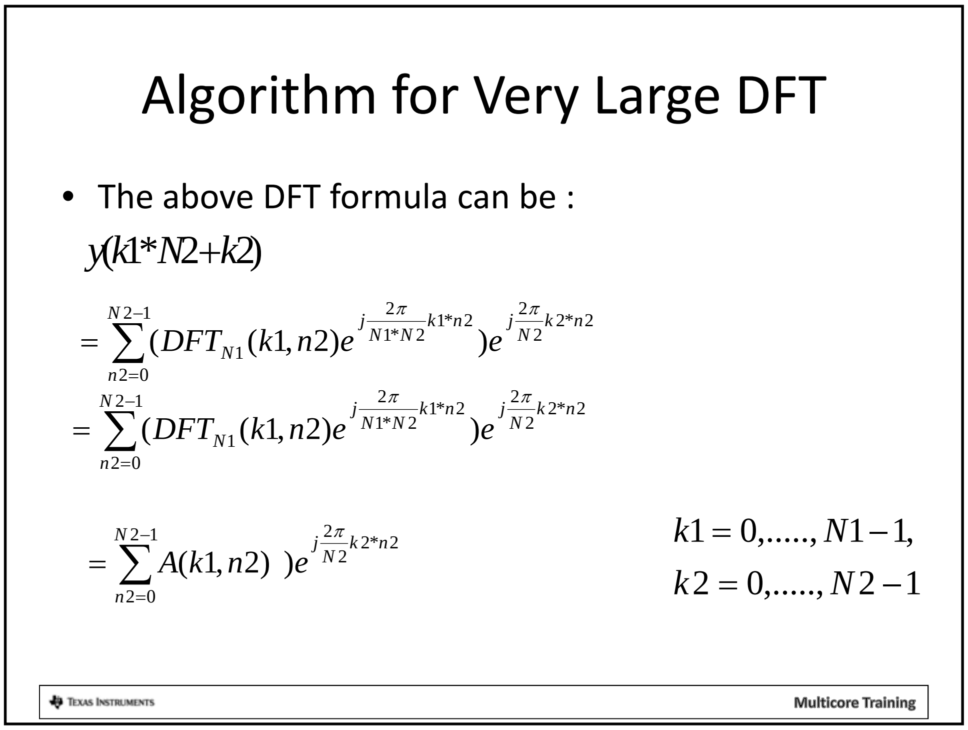
$$y(k) = \sum_{n1=0}^{N1-1}\sum_{n2=0}^{N2-1} x(n1*N2+n2)e^{j\frac{2\pi}{N1*N2}k*(n1*N2+n2)}$$

$$y(k)` = \sum_{n2=0}^{N2-1}\left(\sum_{n1=0}^{N1-1} x(n1*N2+n2)e^{j\frac{2\pi}{N1}k*n1}\right)e^{j\frac{2\pi}{N1*N2}k*n2}$$

$$= \sum_{n2=0}^{N2-1}\left(DFT_{N1}(k1,n2)e^{j\frac{2\pi}{N1*N2}k1*n2}\right)e^{j\frac{2\pi}{N2}k2*n2}$$

$$k1 = 0,\ldots\ldots, N1-1,\quad k2 = 0,\ldots\ldots, N2-1,\quad k = k2*N1+k1$$

Texas Instruments Multicore Training



Algorithm for Very Large DFT

- The above DFT formula can be :

$$y(k1*N2+k2)$$

$$= \sum_{n2=0}^{N2-1}\left(DFT_{N1}(k1,n2)e^{j\frac{2\pi}{N1*N2}k1*n2}\right)e^{j\frac{2\pi}{N2}k2*n2}$$

$$= \sum_{n2=0}^{N2-1}\left(DFT_{N1}(k1,n2)e^{j\frac{2\pi}{N1*N2}k1*n2}\right)e^{j\frac{2\pi}{N2}k2*n2}$$

$$= \sum_{n2=0}^{N2-1}A(k1,n2)\ )e^{j\frac{2\pi}{N2}k2*n2}$$

$$k1 = 0,\ldots\ldots, N1-1,\quad k2 = 0,\ldots\ldots, N2-1$$

Texas Instruments Multicore Training

Algorithm for Very Large DFT

- **A very large DFT of size *N=N1*N2* can be computed in the following steps:**

1) Formulate input into N1xN2 matrix
2) Matrix transpose: N1xN2 -> N2xN1
3) Compute N2 FFTs and multiply twiddle factors. Each FFT is N1 size.
4) Matrix transpose: N2xN1 -> N1xN2
5) Compute N1 FFTs. Each is N2 size.
6) Matrix transpose: N1xN2 -> N2xN1

Implementing VLFFT on Multiple Cores

- Two iterations of computations
- 1st iteration
  - N2 FFTs are distributed across all the cores.
  - Each core implements matrix transpose and computes **N2/numCores FFTs and multiplying twiddle factor**.
- 2nd iteration
  - N1 FFTs of N2 size are distributed across all the cores
  - Each core computes **N1/numCores FFTs and** implements **matrix transpose before and after FFT computation**.

Data Buffers

- **DDR3: Three float complex arrays of size N**
  - Input buffer, output buffer, working buffer
- **L2 SRAM:**
  - Two ping-pong buffers, each buffer is the size of 16 FFT input/output
  - Some working buffer
  - Buffers for twiddle factors
    - Twiddle factors for N1 and N2 FFT
    - N2 global twiddle factors

Global Twiddle Factors

- Global Twiddle Factors:

$$e^{j\frac{2\pi}{N1*N2}k1*n2} \qquad n2 \in [0,\cdots,N2-1] \qquad k1 \in [0,\cdots,N1-1]$$

- Total of N1*N2 global twiddle factors are required.
- N1 are actually pre-computed and saved.

$$e^{j\frac{2\pi}{N1*N2}n2} \qquad n2 \in [0,\cdots,N2-1]$$

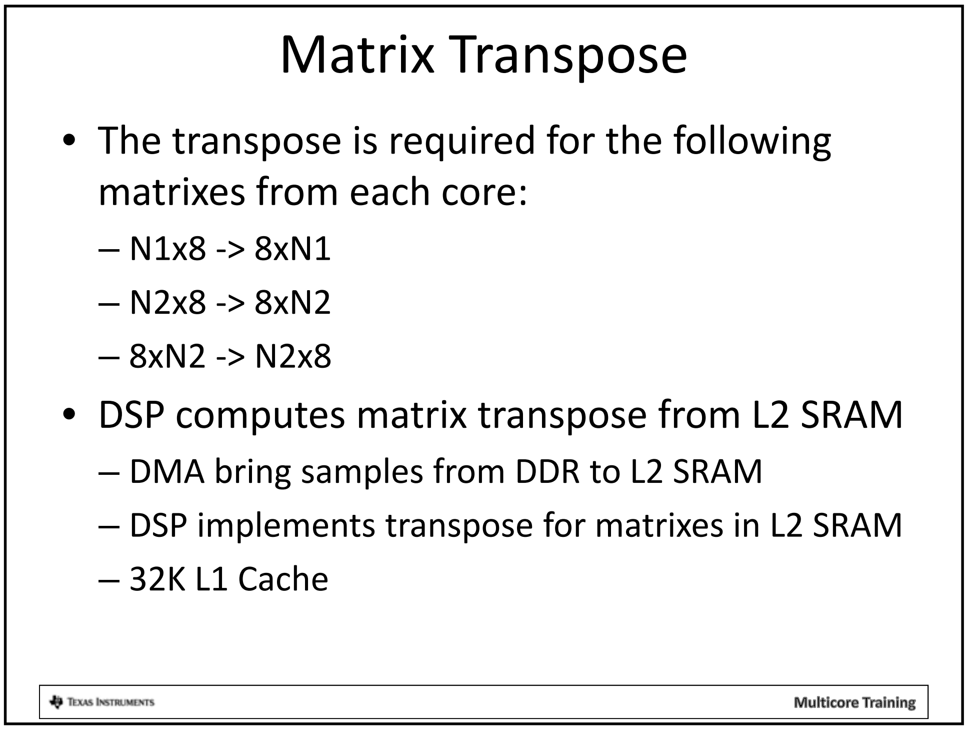
- The rest are computed during run time.

DMA Scheme

- Each core has dedicated in/out DMA channels
- Each core configures and triggers its own DMA channels for input/output
- On each core, the processing is divided into blocks of 8 FFT each.
- For each block on every core
  - DMA transfer 8 lines of FFT input
  - DSP computes FFT/transpose
  - DMA transfers 8 lines of FFT output

VLFFT Pseudo Code

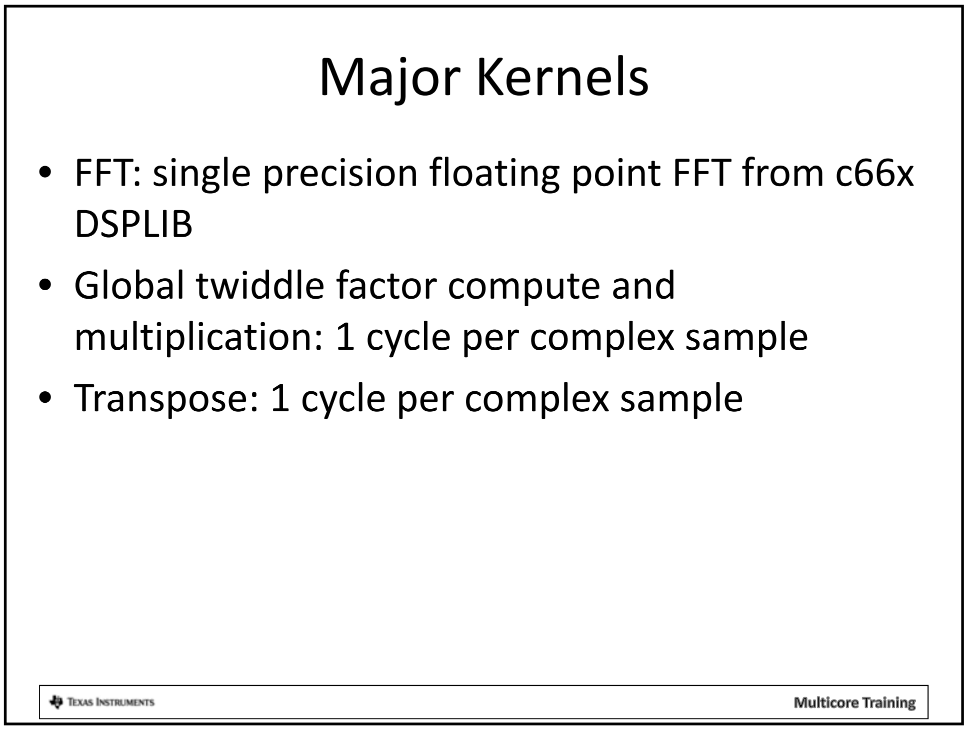
```
VLFFT_start:
1) Core0 sends message to each core to start 1st iteration processing.
2) Each core does the following,
      Wait message from core 0 to start,
      numBlk = 0;
      While( numBlk < totalBlk )
         {
            1) Trigger DMA to transfer (n+1)th blk from Input Buffer to L2 and to transfer (n-1)th blk output from
                L2 to Temp Buffer
            2) Implement transpose, compute FFT, and multiply twiddle factors for nth blk
            3) wait for DMA completion
            4) numBlk++
         }
      Send a message to core 0
3) Core0 waits for message from each core for completion of its own processing
4) After receiving all the messages from all the other cores, core0 sends message to each core to start 2nd
   iteration processing
5) Each core does the following,
      Wait message from core 0 to start,
      numBlk = 0;
      While( numBlk < totalBlk )
         {
            1) Trigger DMA to transfer (n+1)th blk from Temp Buffer to L2 and to transfer (n-1)th blk output from
                L2 to Output Buffer
            2) Compute FFT and transpose for nth blk
            3) wait for DMA completion
            4) numBlk++
         }
      Send a message to core 0
6) Core0 waits for message back from each core for completion its own processing
VLFFT_end:
```



Matrix Transpose

- The transpose is required for the following matrixes from each core:
  - N1x8 -> 8xN1
  - N2x8 -> 8xN2
  - 8xN2 -> N2x8
- DSP computes matrix transpose from L2 SRAM
  - DMA bring samples from DDR to L2 SRAM
  - DSP implements transpose for matrixes in L2 SRAM
  - 32K L1 Cache

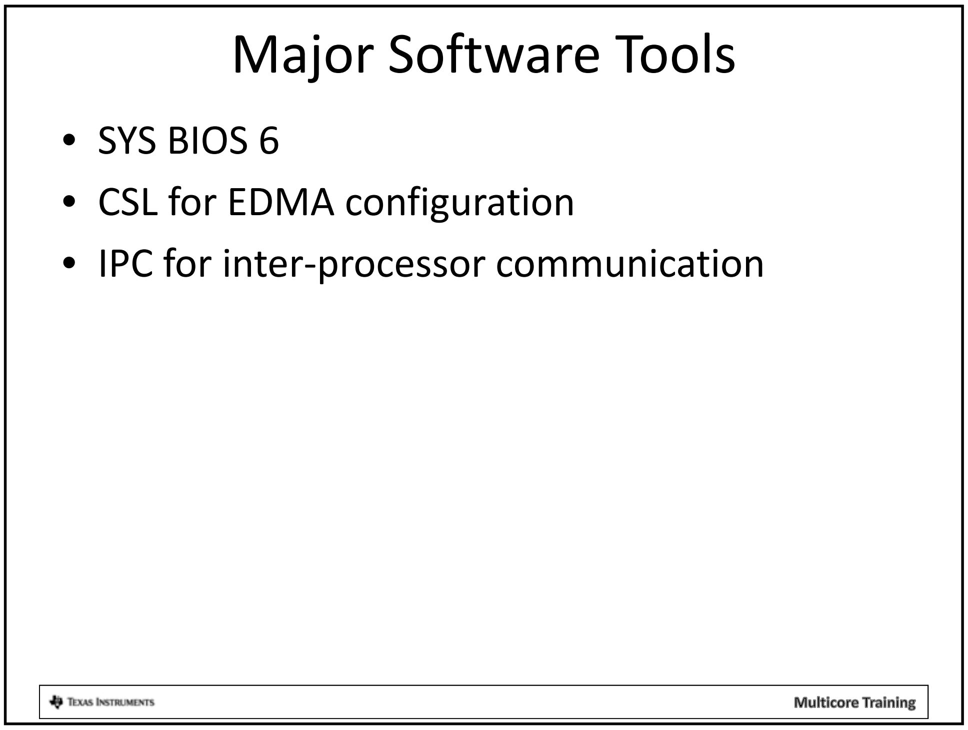
Texas Instruments Multicore Training



Major Kernels

- FFT: single precision floating point FFT from c66x DSPLIB
- Global twiddle factor compute and multiplication: 1 cycle per complex sample
- Transpose: 1 cycle per complex sample

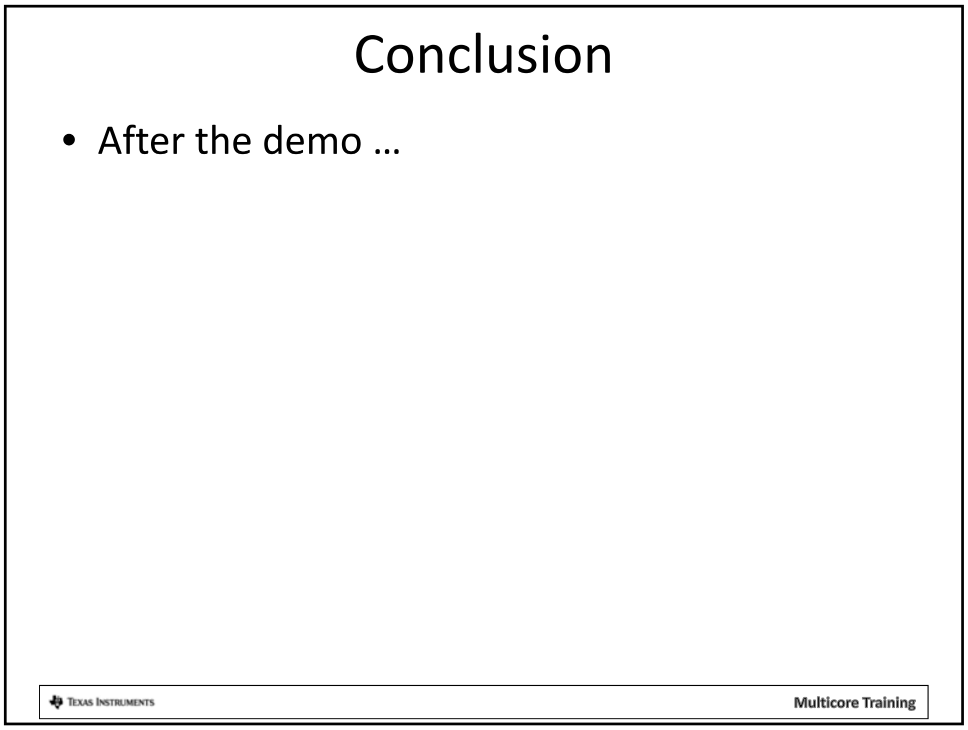
Texas Instruments Multicore Training



Major Software Tools

- SYS BIOS 6
- CSL for EDMA configuration
- IPC for inter-processor communication

Texas Instruments Multicore Training



Conclusion

- After the demo …

Texas Instruments Multicore Training