SINGLE-TRANSPOSE IMPLEMENTATION OF THE OUT-OF-ORDER 3D-FFT

Alexander J. Yee

University of Illinois Urbana-Champaign

The Problem

- FFTs are extremely memory-intensive.
  - Completely bound by memory access.
  - Memory bandwidth is always problem.
    - Single-node shared memory: not enough bandwidth
    - Multi-node: even worse
    - Dominant factor in performance.
  - Naïve implementations also bound by latency.
    - Data-reordering can be many times slower than FFT computation itself!

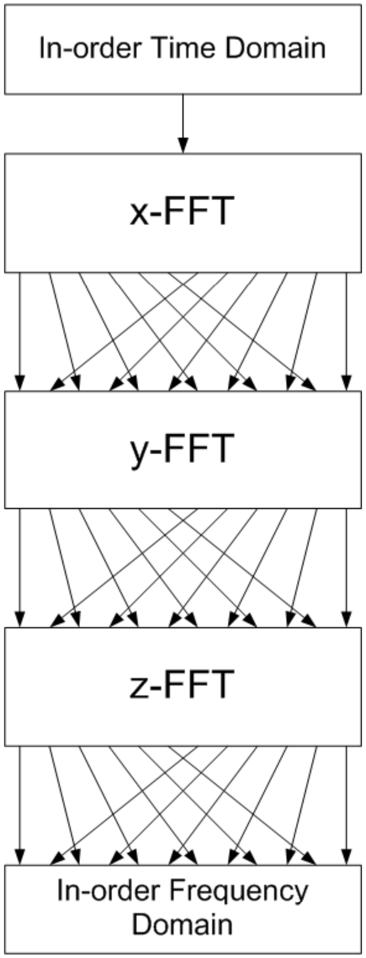
The Classic Approach to 3D-FFT

1. Perform x-dimension FFT.
2. In-memory transpose.
3. All-to-all communication.

4. Perform y-dimension FFT.
5. In-memory transpose.
6. All-to-all communication.

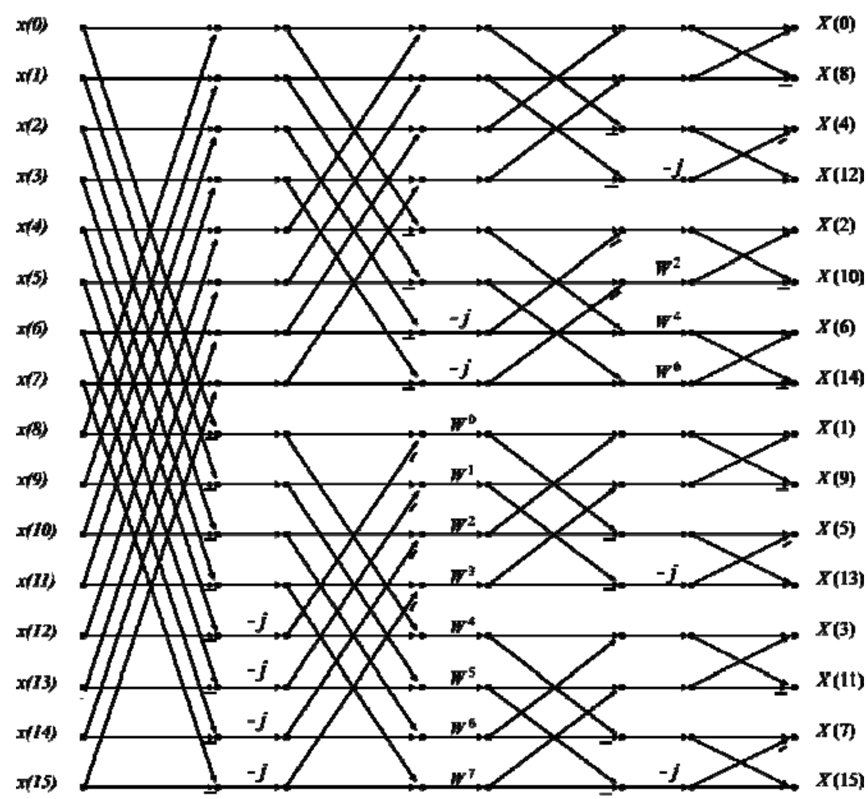
7. Perform z-dimension FFT.
8. In-memory transpose.
9. All-to-all communication.

- Exact order may differ.
- 3 all-to-all communication steps.
- An extra transpose may be needed at beginning to get data into order.



What is an out-of-order FFT?

- The Out-of-order FFT is mathematically the same as in-order FFT:
  - Frequency domain is not in order.
- Forward Transform:
  - Start from in-order time domain.
  - End with out-of-order frequency domain.
  - Use Decimation-in-Frequency algorithm.
- Inverse Transform:
  - Start from out-of-order frequency domain.
  - End with in-order time domain.
  - Use Decimation-in-Time algorithm.
- Order of Frequency Domain:
  - Bit-reversed is the most common.
  - Other orders exist.
    - Some algorithms are even faster – at the cost of further scrambling up the frequency domain.



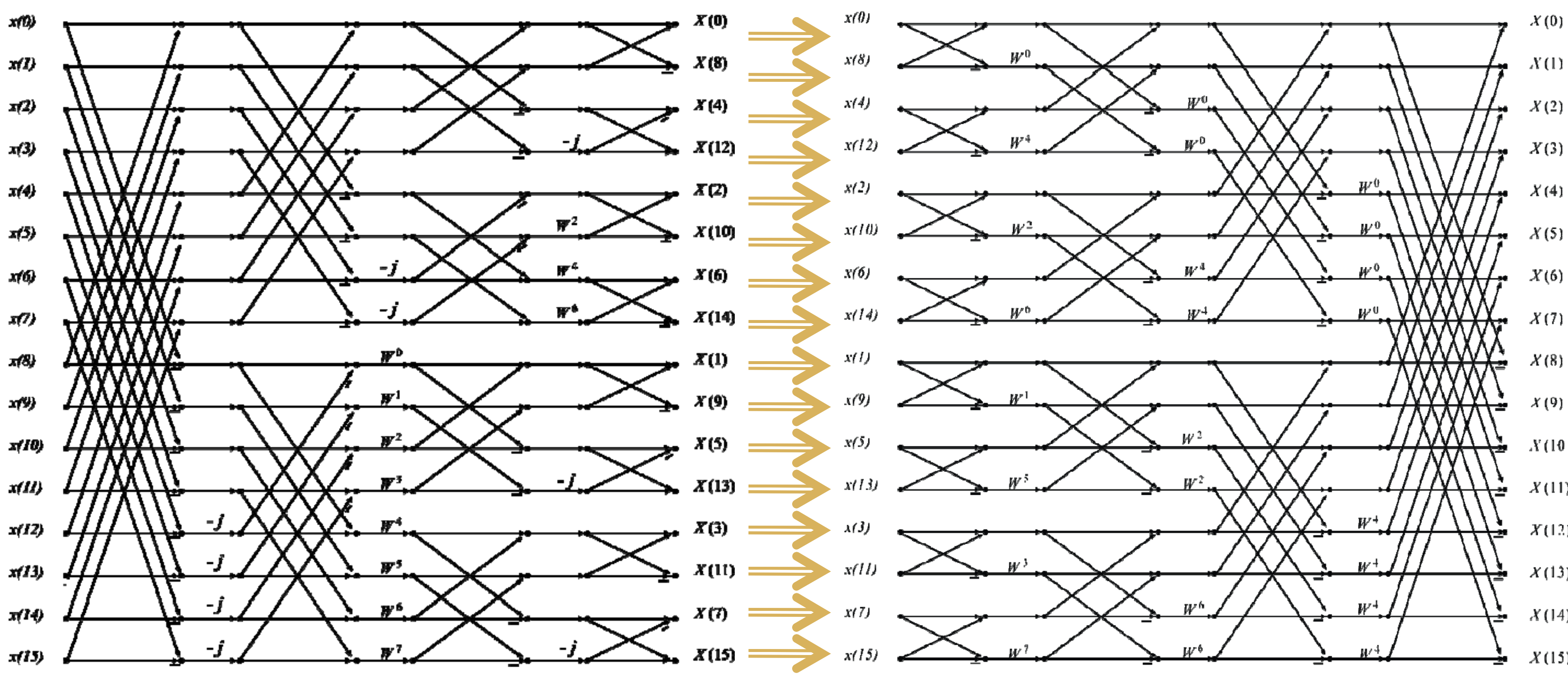
Decimation-in-Frequency FFT

(Image taken from cnx.org)

Why Out-of-Order?

- Many applications do not need an in-order frequency domain.
  - Convolution
    - Do not even need to look at Frequency Domain.
- Out-of-order FFT is faster:
  - In-order FFTs require data-reordering -> bit-reversal
    - Very poor memory access.
    - Re-ordering is more expensive than FFT itself!
  - In-order FFTs cannot be easily done in place.
    - Requires double the memory of out-of-order FFT.
    - Aggravates memory bottleneck.
- Out-of-order FFT can be several times faster!
- No need for final transpose for distributed FFTs over many nodes.

Convolution via Out-of-order FFT



Time-domain
(in order)

Pointwise Multiply
(order does not matter)

Time-domain
(in order)

(Images taken from cnx.org)

Implementations of out-of-order FFT

- Prime95/MPrime – By George Woltman
  - Used in GIMPs (Great Internet Mersenne Prime Search)
    - World record holder for the largest prime number found. (August 2008)
    - 9 of 10 largest known prime numbers found by GIMPS.
  - Uses FFT for cyclic convolution.
  - Fastest known out-of-order FFT. (x86-64 assembly for Windows + Linux)

- y-cruncher Multi-threaded Pi Program – By Alexander J. Yee
  - Fastest program to compute Pi and other constants.
  - World Record holder for the most digits of Pi ever computed. (5 trillion digits – August 2010)
  - Uses FFT and NTT for multiplying large numbers.
  - Almost as fast as Prime95. (Standard C with Intel SSE Intrinsics – cross platform)

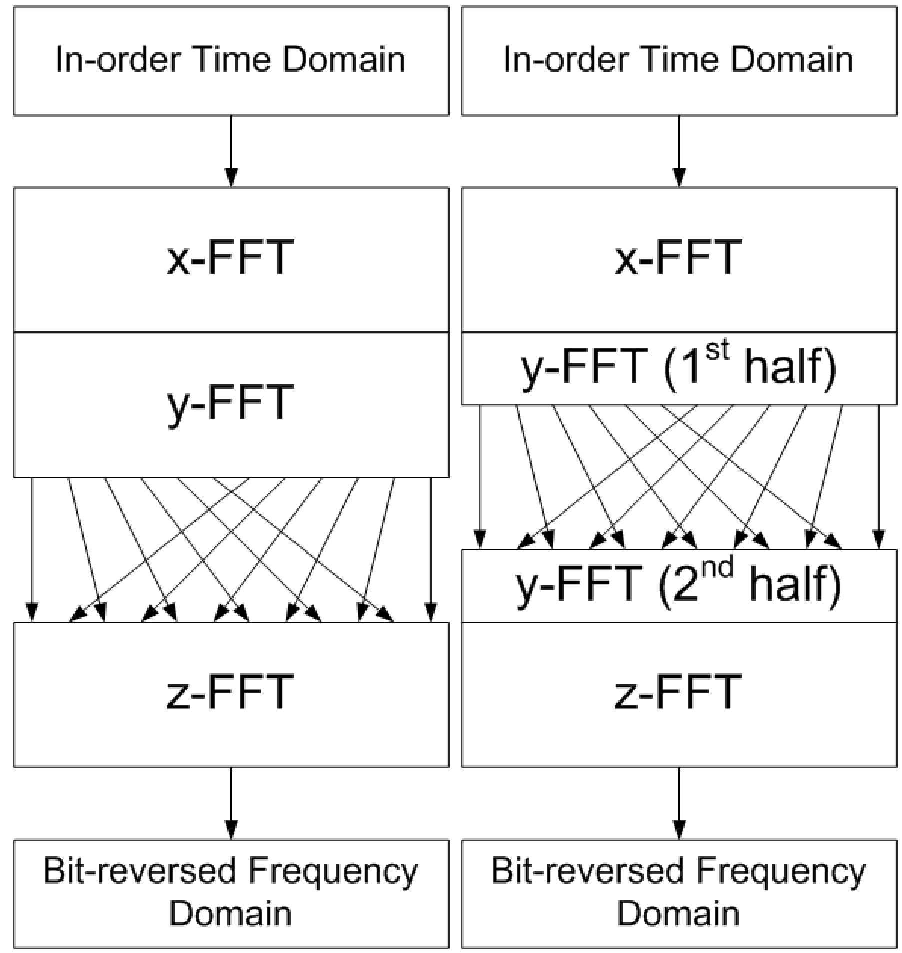
- djbfft – By Daniel J. Bernstein
  - One of the first implementations of out-of-order FFTs.
  - Outperformed FFTW by factors > 3 for convolution.
  - Never widely-used, but motivated other out-of-order FFT projects.

Our Approach to 3D-FFT

- Recognize that n-D FFT is same as 1D FFT.
  - Different Twiddle Factors.
  - Same Memory Access. Same Data-Flow.
- Implement a 1D-FFT with modified twiddle factors instead!
  - All tricks for optimizing 1D-FFT are now available.
- Use Bailey's 4-step algorithm for 1D FFT.
  1. First computation pass.
  2. Matrix Transpose
  3. Second computation pass.
  4. Matrix Transpose data back to initial order.
- Out-of-order FFT -> No need for final transpose!
- Total: 1 transpose -> Only 1 all-to-all communication.

Our Approach to 3D-FFT (cont.)

- To apply Bailey's 4-step method: Break the FFT into 2 passes.
- Easy way (slab decomposition):
  - x and y into one pass.
  - z by itself in second pass.
  - (y can go with x or z)
- Hard way (split dimension):
  - Split the y dimension across the two passes.
  - Overcomes scalability issue with 1st method. (see next section)
- Both are being implemented.
- Frequency Domain will be Bit-reversed.



Drawbacks

Slab Decomposition	Split Dimension

Slab Decomposition

- Can use standard FFT libraries.
  - Optimal performance will still require custom sub-routines.
- Input data can be contiguous.
  - Most common representation.
- # of nodes must divide evenly into either x or z dimension.
  - Scalability is limited to N nodes for N^3 3D-FFT
  - Blue Waters will have more than 10,000 nodes…

Split Dimension

- Cannot use standard libraries.
  - Everything must be written from scratch.
- Input data must be strided.
  - Could imply extra transpose.
- # of nodes must divide evenly into x*y or x*z.
  - Scalability is limited to $N^{3/2}$ nodes for N^3 3D-FFT.
  - Not a problem on Blue Waters.

Some Implementation Details

- All code written from scratch.
  - No libraries.
  - Everything is customized.
- SIMD
  - SSE for x86-64
  - AltiVec for PowerPC
  - “Struct of Arrays” layout
  - Will extend to AVX and FMA in the future. (Next-gen Intel/AMD x64.)
- Radix 4 FFT
  - Good performance.
  - Fits into 16 registers.
  - Not too bad for cache associativity.
- Pre-compute Twiddle Factors
  - Duplicate tables to ensure sequential access.

Different Representations:

{r0 + i0*i, r1 + i1*i, r2 + i2*i, r3 + i3*i}

Array of Structs (classic approach)

r0	i0	r1	i1	r2	i2	r3	i3

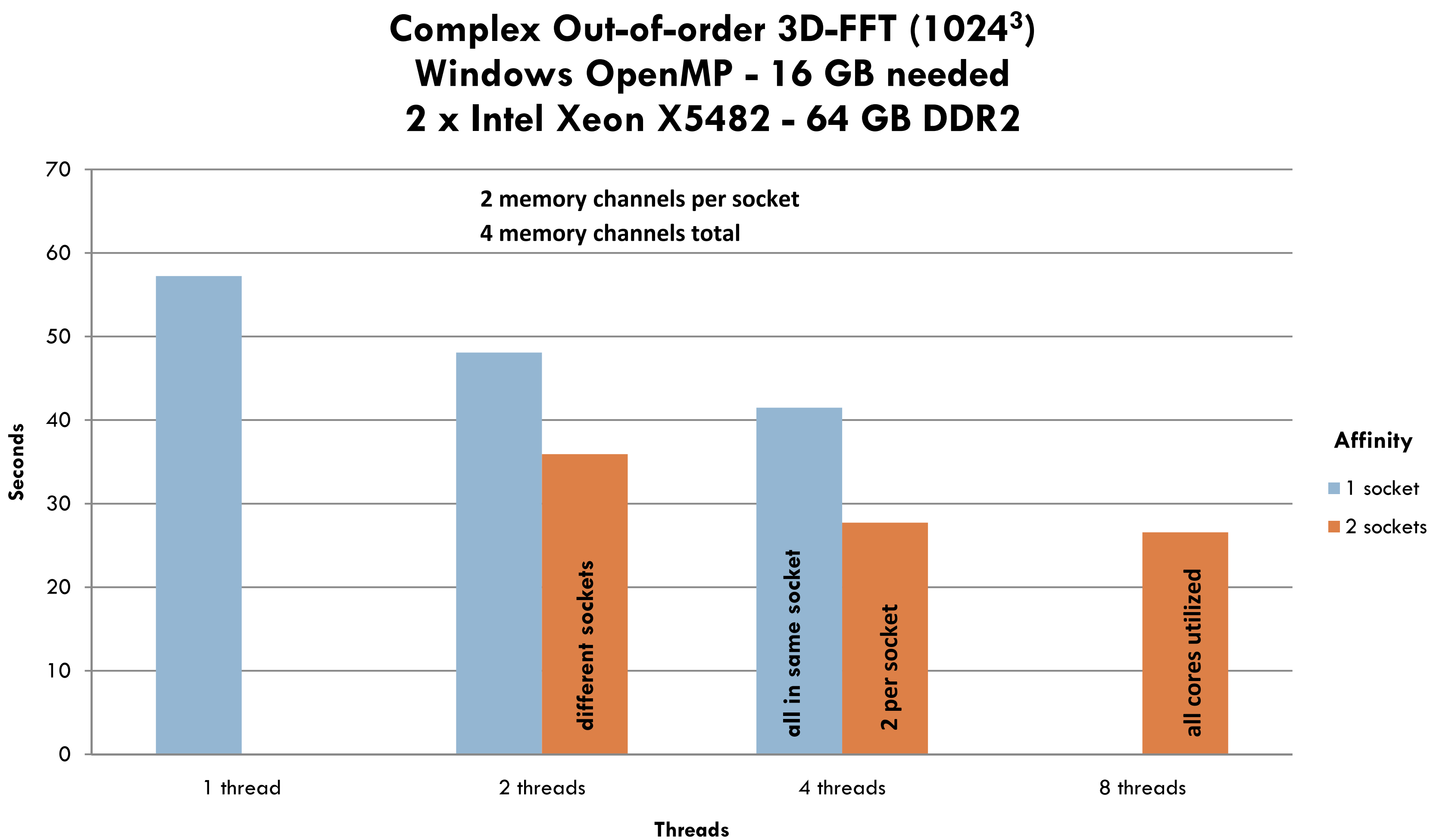
Complex Multiplication requires unpacking.
SSE3 addsubpd helps a little bit. But still slow.

Struct of Arrays

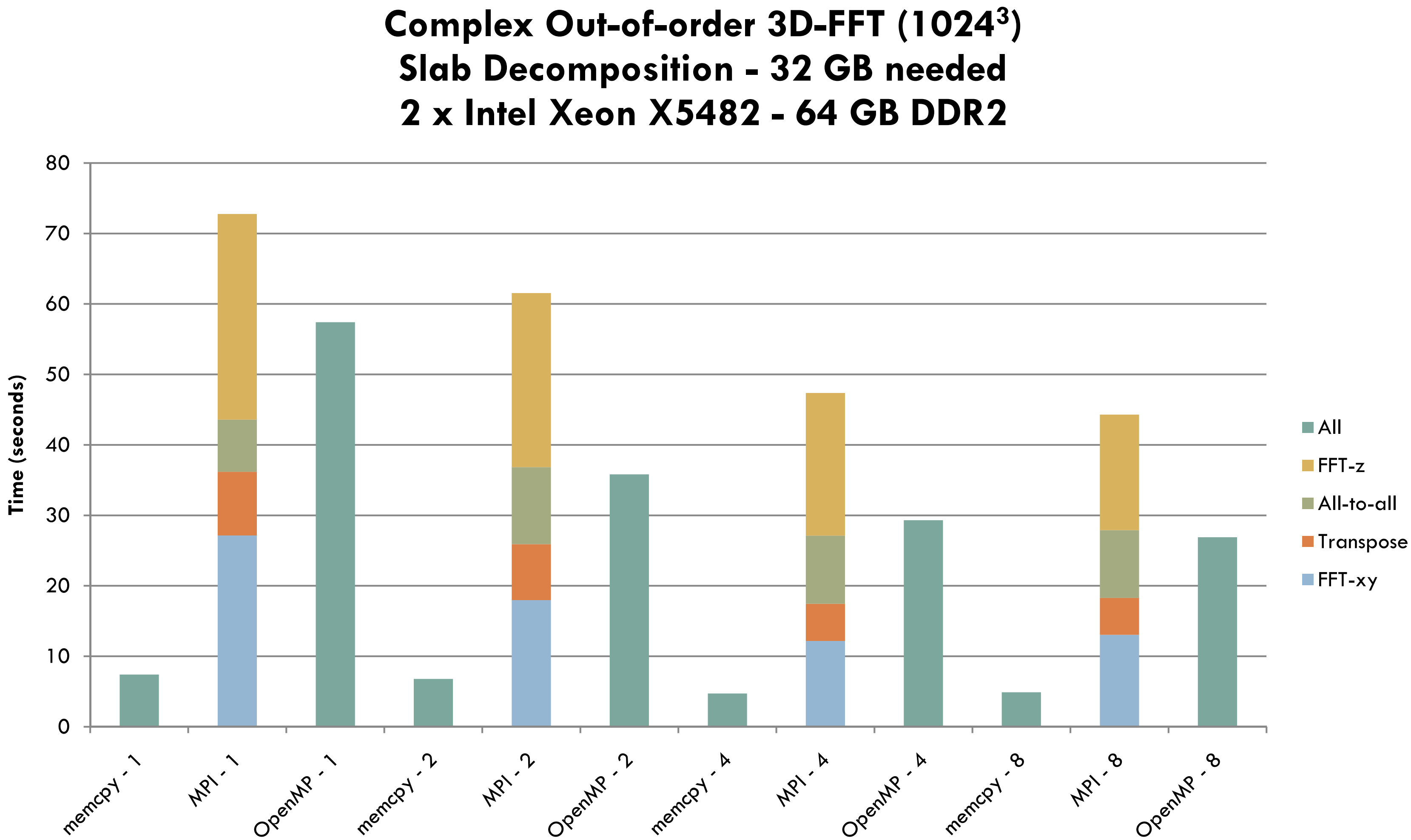
r0	r1	i0	i1	r2	r3	i2	i3

No unpacking needed.
When adjacent points need to operate:
SSE3 Horizontal Instructions! (30% faster)

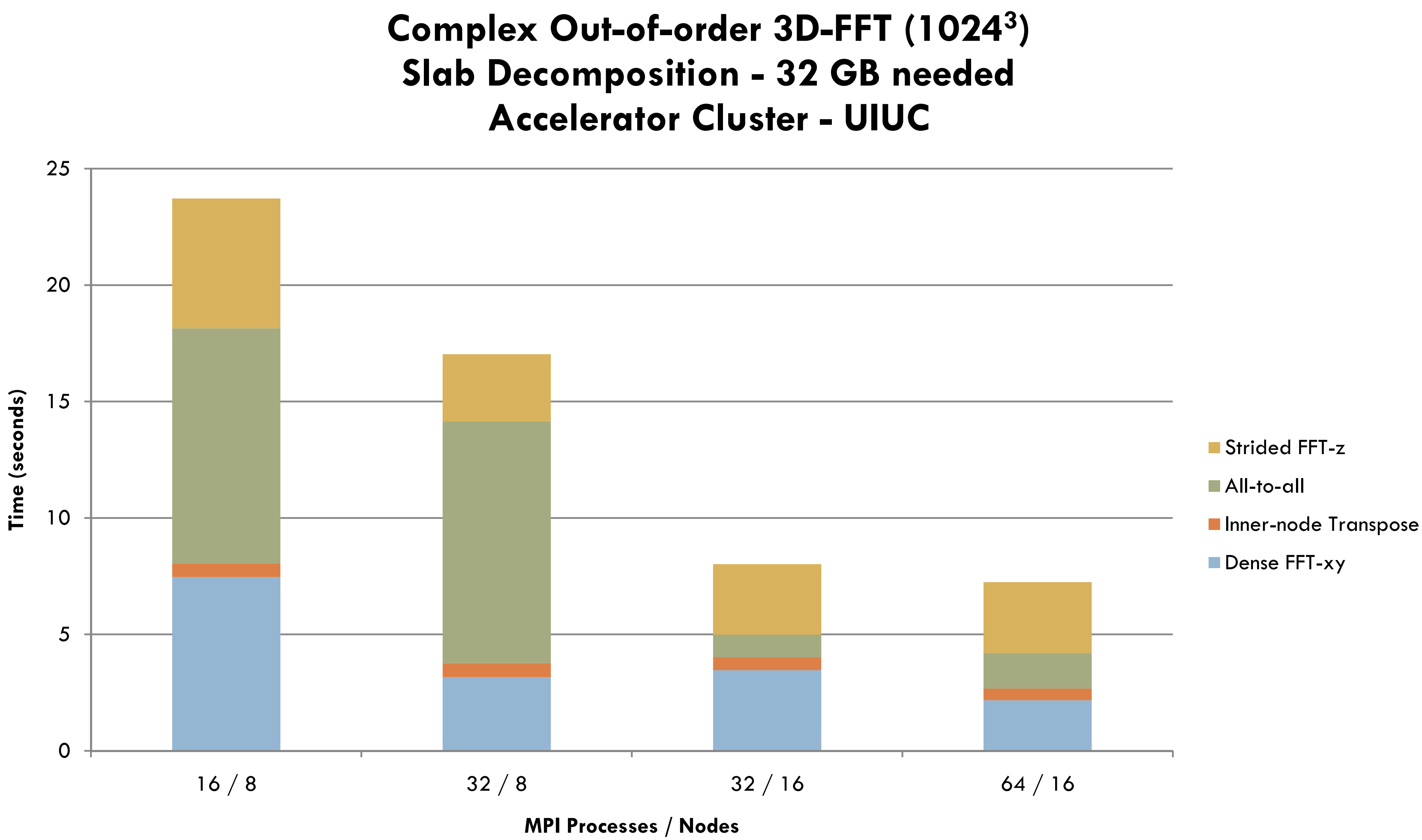
Benchmarks – Memory Bottleneck



Benchmarks - Shared Memory



Early Benchmarks - Distributed



Analysis

- All-to-all communication steps reduced:
  - Reduced from 3 to 1 for out-of-order FFT.
  - In-order FFT doable by adding one transpose at end.
- Possibly communication optimal:
  - No data is transferred more than once.
    - Some data is never transferred at all.
    - Hard to further reduce the # of bytes transferred. (Is our current approach optimal?)
  - Maybe possible to improve communication pattern instead?
- Lots of room for improvement within the node.
  - FFT computation can be better optimized.
- Difficult to imagine more nodes than x or z dimension.
  - Blue Waters: > 10,000 nodes
  - 10,000 may be greater than one of the dimensions.
    - May not be possible (or efficient) to use slab-decomposition.

Next Steps

- Test current code on larger systems.
  - Make sure the current implementation scales
- Port the code to PowerPC AltiVec.
  - Currently implemented using x86-64 SSE3.
- Implement blocking and padding.
  - Breaks cache associativity -> allows higher radix transforms.
- Overlapped communication and computation.
- Support for prime factors other than 2
  - $3*2^k$, $5*2^k$, and maybe $7*2^k$
- Real-input transforms.
- In-order FFT.

Thanks for Listening

- Questions?