OPENCL™ FAST FOURIER TRANSFORM OPTIMIZATIONS FOR INTEL® PROCESSOR GRAPHICS

Dan Petre, Adam Lake, Allen Hux, Michal Mrozek

Presenter: Michal Mrozek

IWOCL '16, April 19-21, 2016, Vienna, Austria
ACM 978-1-4503-4338-1/16/04.
http://dx.doi.org/10.1145/2909437.2909451

Legal Notices and Disclaimers

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

A "Mission Critical Application" is any application in which failure of the Intel Product could result, directly or indirectly, in personal injury or death. SHOULD YOU PURCHASE OR USE INTEL'S PRODUCTS FOR ANY SUCH MISSION CRITICAL APPLICATION, YOU SHALL INDEMNIFY AND HOLD INTEL AND ITS SUBSIDIARIES, SUBCONTRACTORS AND AFFILIATES, AND THE DIRECTORS, OFFICERS, AND EMPLOYEES OF EACH, HARMLESS AGAINST ALL CLAIMS COSTS, DAMAGES, AND EXPENSES AND REASONABLE ATTORNEYS' FEES ARISING OUT OF, DIRECTLY OR INDIRECTLY, ANY CLAIM OF PRODUCT LIABILITY, PERSONAL INJURY, OR DEATH ARISING IN ANY WAY OUT OF SUCH MISSION CRITICAL APPLICATION, WHETHER OR NOT INTEL OR ITS SUBCONTRACTOR WAS NEGLIGENT IN THE DESIGN, MANUFACTURE, OR WARNING OF THE INTEL PRODUCT OR ANY OF ITS PARTS.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined". Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information. The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725, or go to: http://www.intel.com/design/literature.htm

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

All products, computer systems, dates and figures specified are preliminary based on current expectations, and are subject to change without notice.

All products, platforms, dates, and figures specified are preliminary based on current expectations, and are subject to change without notice. All dates specified are target dates, are provided for planning purposes only and are subject to change.

This document contains information on products in the design phase of development. Do not finalize a design with this information. Revised information will be published when the product is available. Verify with your local sales office that you have the latest datasheet before finalizing a design.

Code names featured are used internally within Intel to identify products that are in development and not yet publicly announced for release. Customers, licensees and other third parties are not authorized by Intel to use code names in advertising, promotion or marketing of any product or services and any such use of Intel's internal code names is at the sole risk of the user.

Intel and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries.

Other names and brands may be claimed as the property of others.

Copyright © 2015-2016, Intel Corporation. All rights reserved.

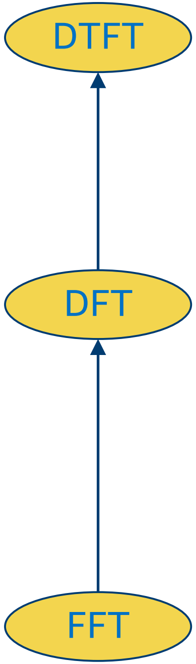
OpenCL and the OpenCL logo are trademarks of Apple Inc. used by permission by Khronos

Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Agenda

- Introduction
- Implementation overview
- Optimizations
- Results
- Conclusion and future work

What is Fast Fourier Transform (FFT)?



DTFT

$$X_{(f)} = \sum_{-\infty}^{+\infty} x(t)\, e^{-2\pi j f t}$$ [1]

DFT

$$X_k = \sum_{n=0}^{N} x_n e^{-j\frac{2\pi}{N}nk}$$

FFT

$$X_k = \sum_{n=0}^{N/2} x_{2n} e^{-j\frac{2\pi}{N/2}2nk} + \sum_{n=0}^{N/2} x_{2n+1} e^{-j\frac{2\pi}{N/2}(2n+1)k}$$

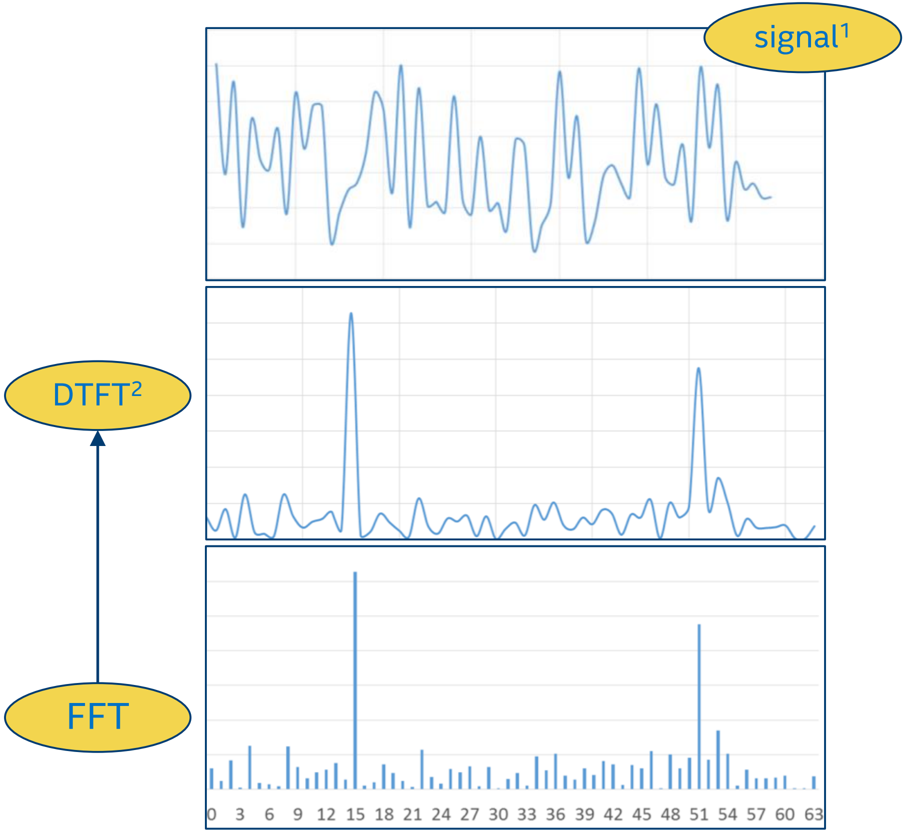
Cooley-Tukey radix 2 decomposition

- Discrete-Time Fourier Transform (DTFT)
  - Infinite series, can’t be computed
- Discrete Fourier Transform (DFT)
  - A sampling of the DTFT
- FFT
  - A fast way to compute DFT

---

[1] $t = n \times T$, T = quanta of time

What is Fast Fourier Transform (FFT)?



- Discrete-Time Fourier Transform (DTFT)
  - Infinite series, can’t be computed
- Discrete Fourier Transform (DFT)
  - A sampling of the DTFT
- Fast Fourier Transform
  - A fast way to compute DFT

[1] Here the signal is actually complex, the chart shows magnitude.
[2] Not really a DTFT, just an approximation… DTFT can’t be computed.

Typical Ways to Compute Fast Fourier Transform (FFT)

Cooley-Tukey

- Recursively split the FFT into smaller equal sizes

Split-Radix

- Radix 4:2:2

Stockham

- Eliminates the need for rearranging the inputs/outputs that is specific to Cooley-Tukey

Prime-factor

- Decompose into **relatively** prime numbers

Bluestein (Chirp-Z)

- For arbitrary sizes, uses Cooley-Tukey and convolution theorem

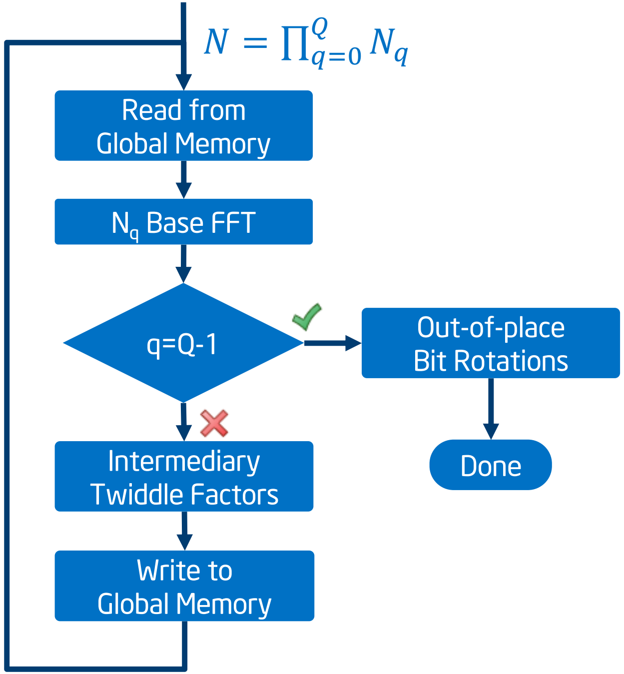
many others…

Main Differences versus Other Implementations

- No shared local memory, no barriers
  - Maximize device occupancy
  - Reduced code complexity
- Column major in/out
  - Intended for progression to 2D Fast Fourier Transform (FFT)
  - Maximizes memory bandwidth (GB/s)
  - Cache-friendly
- FFT decomposed into smaller FFTs
  - Called here “base FFTs”
  - Maximize register use
  - Reduced code complexity
- Multi-kernel
  - Each base FFT as a separate kernel
- Code generation for
  - Any local/global size configuration
  - Any register size/SIMD size

genFFT Code Generator and Execution Flow

- Decompose the Fast Fourier Transform (FFT) into factors power of two
  - Base FFTs must maximize register use
  - Minimize the difference of the last two factors – best performance
- Generate each base FFT
  - Local twiddle factor lookup table (LUT) in registers
- Generate intermediary twiddle factors
  - LUT in global memory
- Generate bit rotations
  - In order to unscramble the data at the end



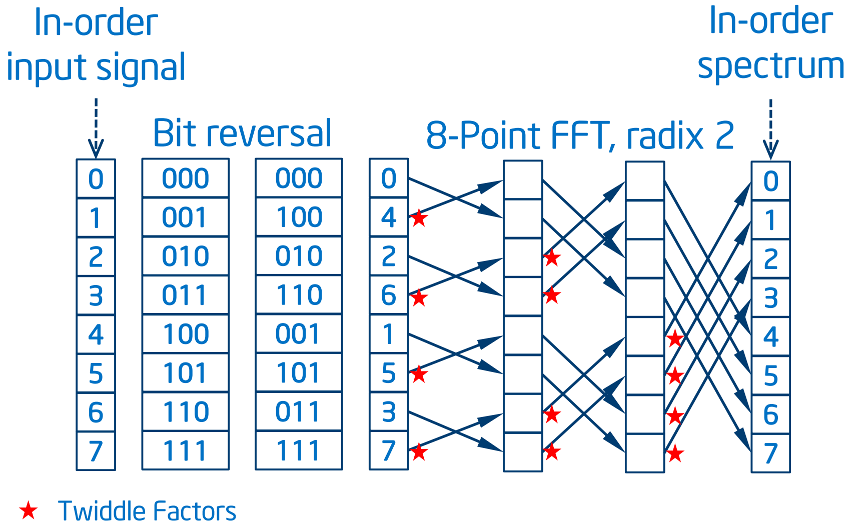
$N = \prod_{q=0}^{Q} N_q$

Base Fast Fourier Transform (FFT) Butterflies

- Cooley-Tukey radix 2 butterflies
  - 8-Point DIT FFT on the right
    - Decimation-In-Time
- Pseudo-code

1. Read signal
2. Bit reversal
3. Perform the butterflies
   - ★ Apply twiddle factors
4. Write spectrum

$$X_k = \sum_{n=0}^{N/2} x_{2n} \underbrace{e^{-j\frac{2\pi}{N/2}2nk}}_{\star} + \sum_{n=0}^{N/2} x_{2n+1} \underbrace{e^{-j\frac{2\pi}{N/2}(2n+1)k}}_{\star}$$



In-order input signal

In-order spectrum

Bit reversal

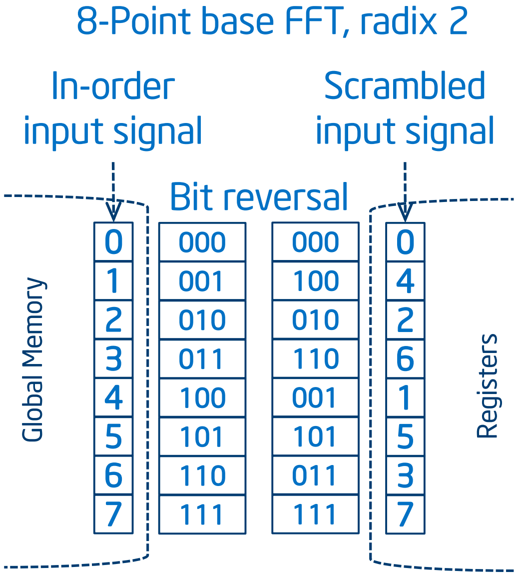
8-Point FFT, radix 2

0	000	000	0	0
1	001	100	4	1
2	010	010	2	2
3	011	110	6	3
4	100	001	1	4
5	101	101	5	5
6	110	011	3	6
7	111	111	7	7

★ Twiddle Factors

Base Fast Fourier Transform (FFT) Bit Reversal

- Each base FFT performs its own bit reversal
  - No global bit reversal at the beginning
  - A distributed bit reversal strategy
- Advantages
  - The preprocessor eliminates bit reversal math at compile time
  - The input is read directly into the correct registers
- Disadvantages
  - This still requires bit rotations at the end of a multi-kernel FFT pipeline[1]



8-Point base FFT, radix 2

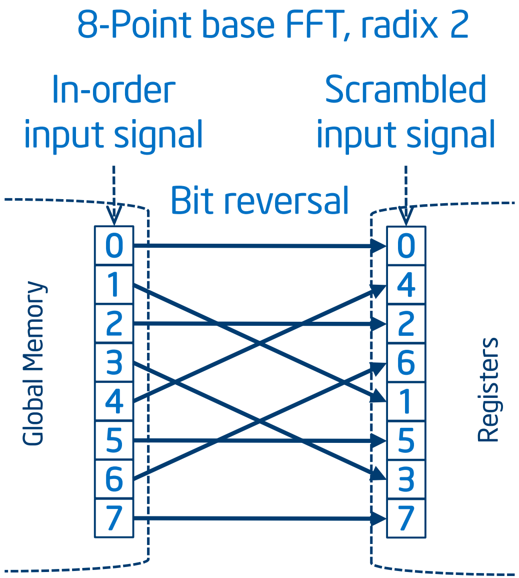
0	000	000	0
1	001	100	4
2	010	010	2
3	011	110	6
4	100	001	1
5	101	101	5
6	110	011	3
7	111	111	7

---

[1] It can be argued that the bit rotations exhibit a more regular memory access pattern than a global bit reversal.

Base Fast Fourier Transform (FFT) Bit Reversal

- Each base FFT performs its own bit reversal
  - No global bit reversal at the beginning
  - A distributed bit reversal strategy
- Advantages
  - The preprocessor eliminates bit reversal math at compile time
  - The input is read directly into the correct registers
- Disadvantages
  - This still requires bit rotations at the end of a multi-kernel FFT pipeline[1]



---

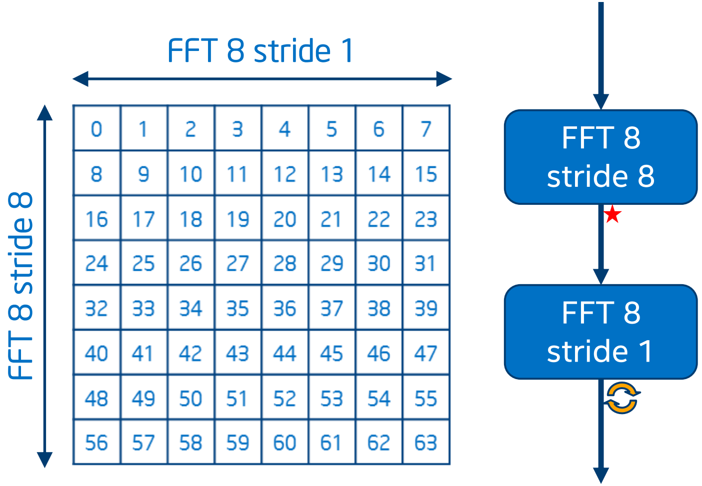
[1] It can be argued that the bit rotations exhibit a more regular memory access pattern than a global bit reversal.

Multi-Kernel Fast Fourier Transform (FFT) Implementation

$$FFT_N = FFT_{N_1}\left(W_N^{n_1 k_2} FFT_{N_2}\right)$$

- Cooley-Tukey decomposition into factors power of 2
  - Each factor treated as a radix-2 Cooley-Tukey FFT with a different stride
  - Each FFT as an independent kernel—base FFT
  - Base FFTs make efficient use of registers
  - Each base FFT performs its own bit reversal on inputs
    - Intermediary twiddle factors
    - Bit rotations required at the end

$$N = 64 = 8 \times 8$$



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
32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55
56	57	58	59	60	61	62	63

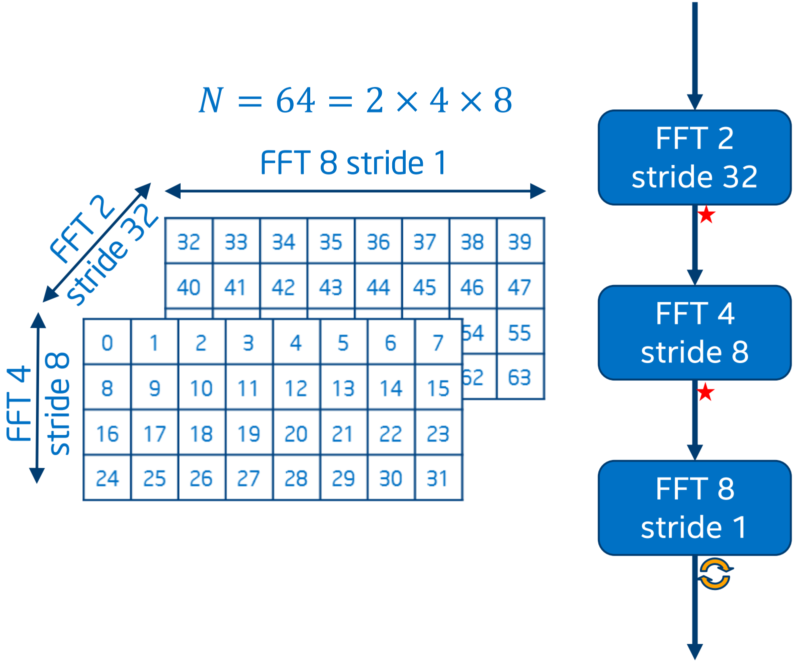
★ Intermediary Twiddle Factors

⟳ Bit Rotations

Multi-Kernel Fast Fourier Transform (FFT) implementation

- Cooley-Tukey decomposition into factors power of 2
  - Each factor treated as a radix-2 Cooley-Tukey FFT with a different stride
  - Each FFT as an independent kernel—base FFT
  - Base FFTs make efficient use of registers
  - Each base FFT performs its own bit reversal on inputs
    - Intermediary twiddle factors
    - Bit rotations required at the end

$$N = 64 = 2 \times 4 \times 8$$

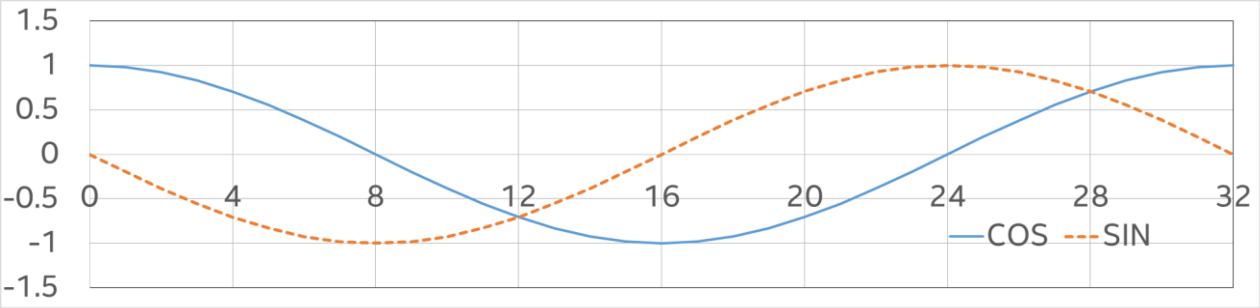


★ Intermediary Twiddle Factors

⟳ Bit Rotations

Base FFT Twiddle Factors Lookup Table

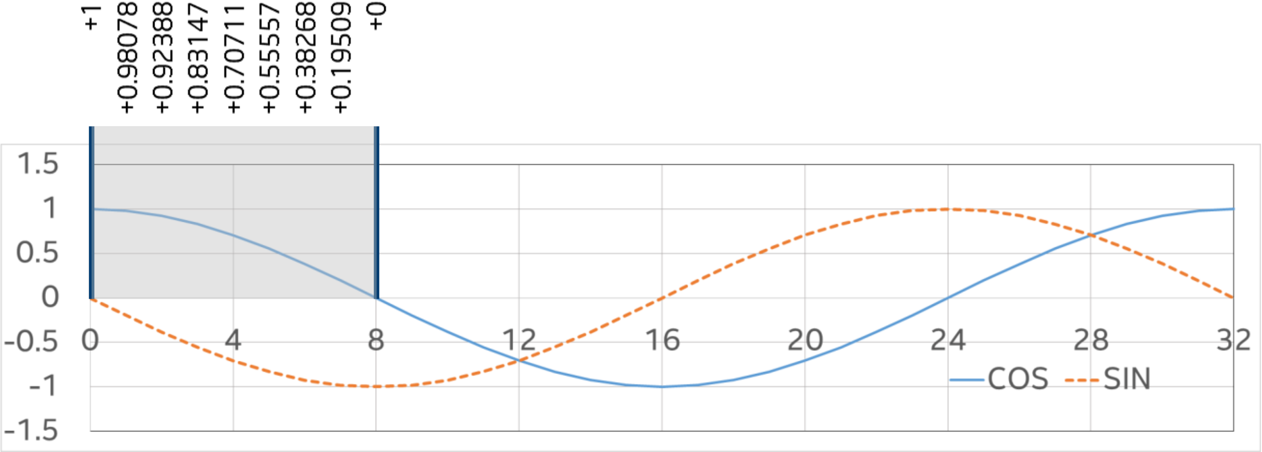
- Storage requirements reduced from N/2 cl_float2 values to N/4+1 cl_float values
  - Private array lookup table small enough to fit into registers



$$W_N^k = e^{-j\frac{2\pi}{N}k} = \cos(-2\pi k/N) + j\sin(-2\pi k/N)$$

Base FFT Twiddle Factors Lookup Table

- Storage requirements reduced from N/2 cl_float2 values to N/4+1 cl_float values
  - Private array lookup table small enough to fit into registers

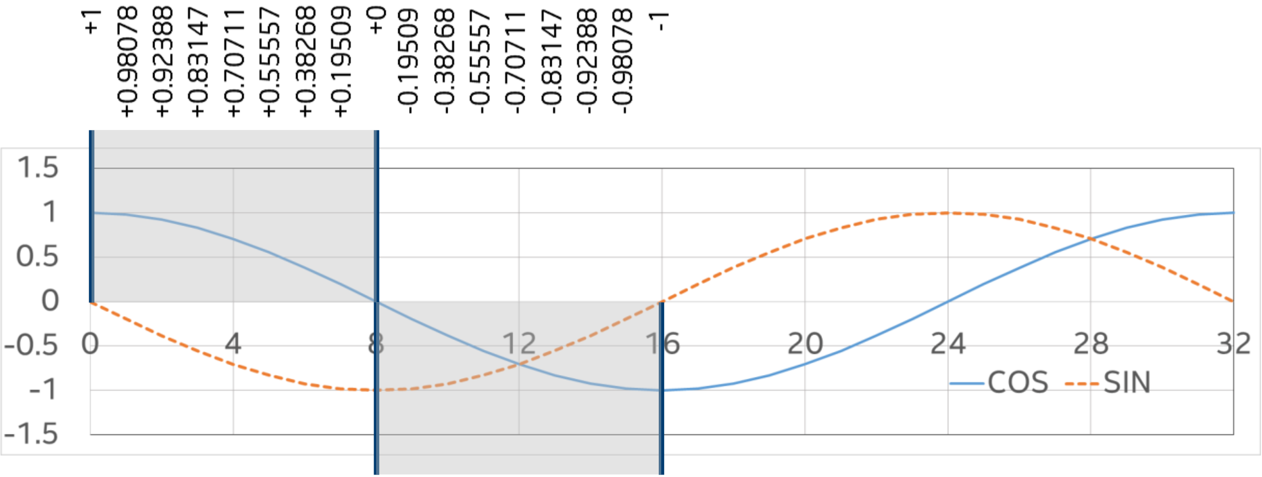


$$W_N^k = e^{-j\frac{2\pi}{N}k} = \cos(-2\pi k/N) + j\sin(-2\pi k/N)$$

Optimization Notice

Base FFT Twiddle Factors Lookup Table

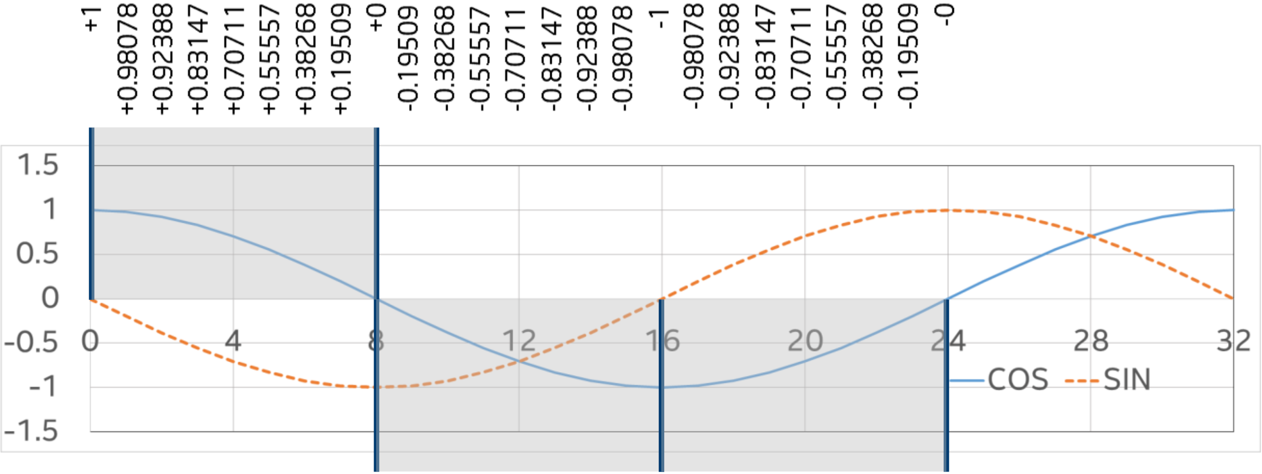
- Storage requirements reduced from N/2 cl_float2 values to N/4+1 cl_float values
  - Private array lookup table small enough to fit into registers



$$W_N^k = e^{-j\frac{2\pi}{N}k} = \cos(-2\pi k/N) + j\sin(-2\pi k/N)$$

Base FFT Twiddle Factors Lookup Table

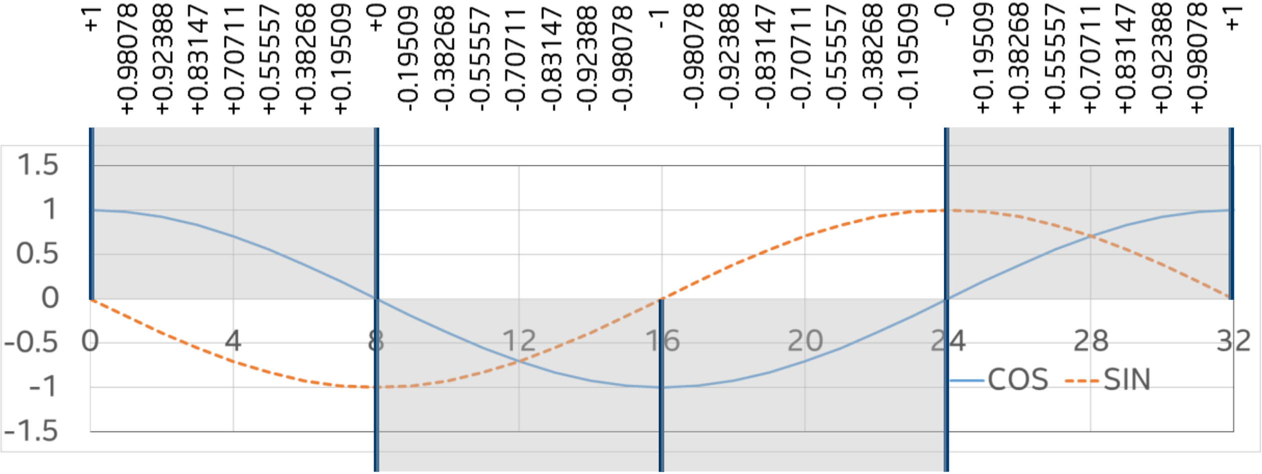
- Storage requirements reduced from N/2 cl_float2 values to N/4+1 cl_float values
  - Private array lookup table small enough to fit into registers



$$W_N^k = e^{-j\frac{2\pi}{N}k} = \cos(-2\pi k/N) + j\sin(-2\pi k/N)$$

Base FFT Twiddle Factors Lookup Table

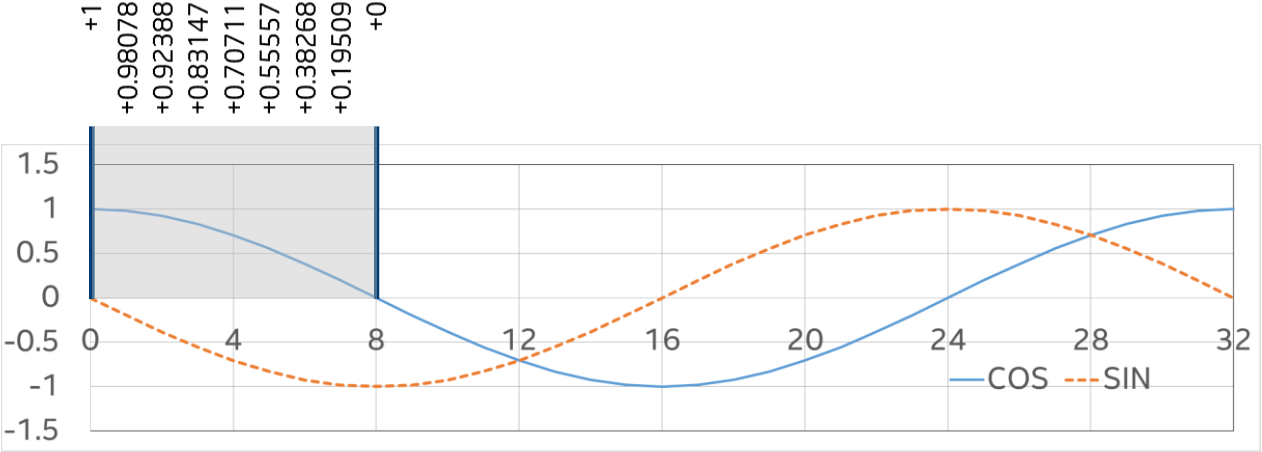
- Storage requirements reduced from N/2 cl_float2 values to N/4+1 cl_float values
  - Private array lookup table small enough to fit into registers



$$W_N^k = e^{-j\frac{2\pi}{N}k} = \cos(-2\pi k/N) + j\sin(-2\pi k/N)$$

Base FFT Twiddle Factors Lookup Table

- Storage requirements reduced from N/2 cl_float2 values to N/4+1 cl_float values
  - Private array lookup table small enough to fit into registers



$$W_N^k = e^{-j\frac{2\pi}{N}k} = \cos(-2\pi k/N) + j\sin(-2\pi k/N)$$

Base FFT Twiddle Factors Lookup Table

```
constant float W[9] =
{
           +0.0f,
    +0.19509f,
    +0.38268f,
    +0.55557f,
    +0.70711f,
    +0.83147f,
    +0.92388f,
    +0.98078f,
           +1.0f,
};
```

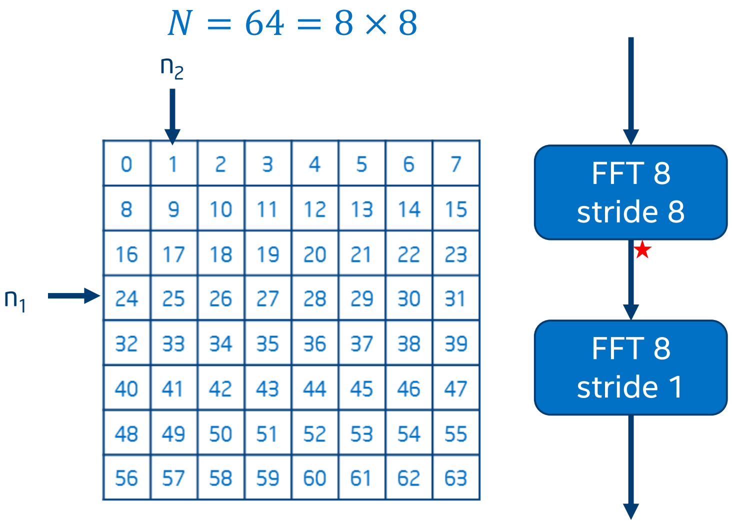
	cos(i)	sin(i)
0	1.0	0.0
1	0.98079	-0.19509
2	0.92388	-0.38268
3	0.83147	-0.55557
4	0.70711	-0.70711
5	0.55557	-0.83147
6	0.38268	-0.92388
7	0.19509	-0.98079
8	0.0	-1.0
9	-0.19509	-0.98079
10	-0.38268	-0.92388
11	-0.55557	-0.83147
12	-0.70711	-0.70711
13	-0.83147	-0.55557
14	-0.92388	-0.38268
15	-0.98079	-0.19509
16	-1.0	0.0

```
cos(k) = +W(FFT_SIZE/4-FFT_SIZE/crtFFT * k);
sin(k) = -W(FFT_SIZE/crtFFT * k);
```

```
cos(k) = -W(FFT_SIZE/crtFFT * k);
sin(k) = -W(FFT_SIZE/4-FFT_SIZE/crtFFT * k);
```

Intermediary Twiddle Factors Lookup Table

- Intermediary twiddle factors: $W_N^{n_1 n_2} = e^{-j\frac{2\pi}{N}n_1 n_2}$
  - Different than the base FFT twiddle factors
  - Total of N cl_float2, quite large
  - We reduced it to N/4+1 cl_float values (8x)
    - At the cost of some pretty complicated indexing math
  - Can be further reduced to half of that
    - At the cost of extra math (sqrt) and accuracy
  - >10 percent performance loss
  - In the end: global array of N cl_float2 kernel argument



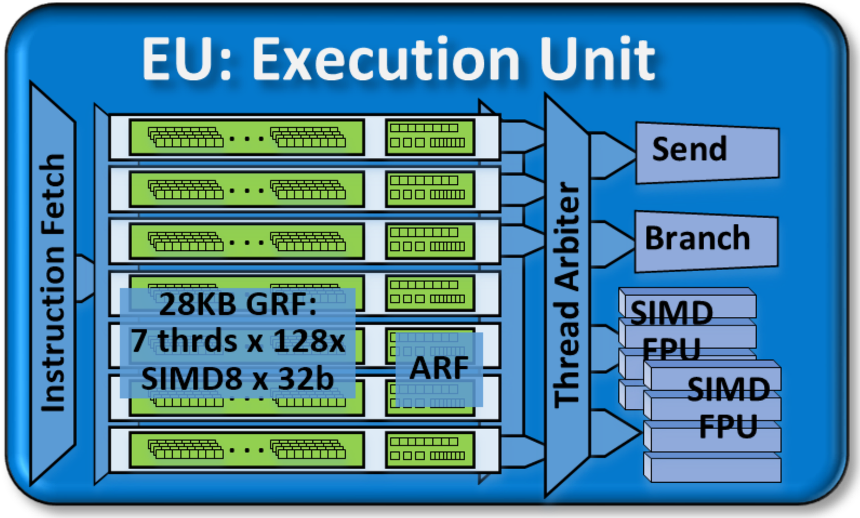
$N = 64 = 8 \times 8$

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
32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55
56	57	58	59	60	61	62	63

★ Intermediary Twiddle Factors

Efficient Use of the Execution Unit Registers

- 7 hardware threads
- 128 registers per hardware thread
  - 32 bytes per register
- 512 bytes per work item
  - In SIMD 8 mode
- $N_r = 32$, max FFT size that fits in registers
  - Up to 64 cl_float2 per work item
  - leave room for other program variables

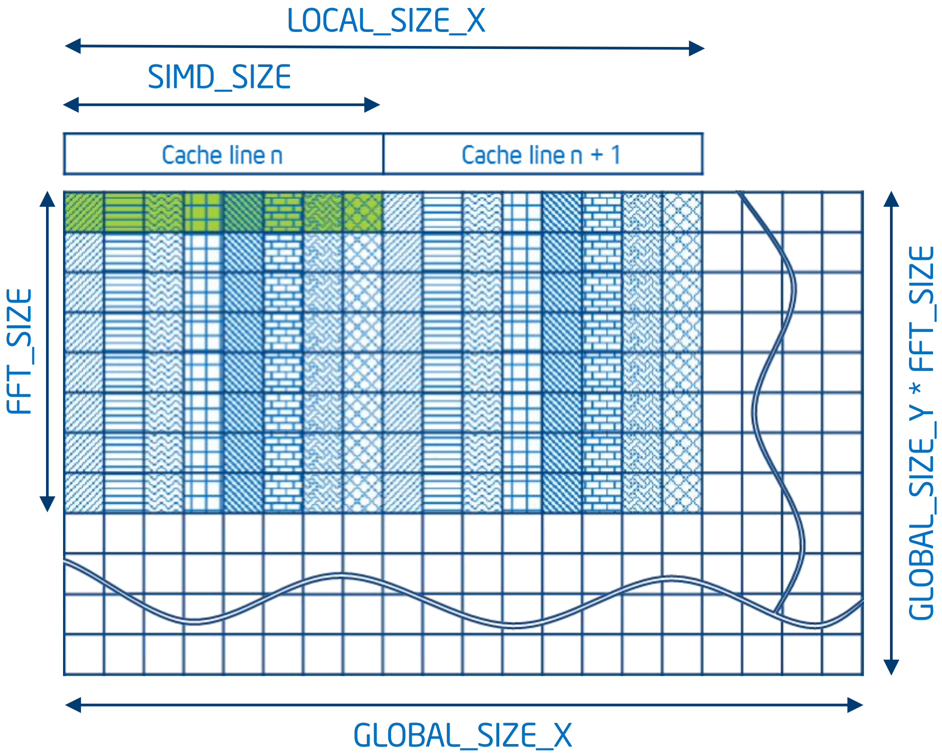


[Junkins 2014-2015]

Efficient Use of Cache and Memory Bandwidth

- Goals
  - Reduce number of cache lines touched
  - Reduce the number of memory address requests that an execution unit makes to the data port
- 90 percent of peak last level cache bandwidth (GB/s)
  - For base FFT kernels
- On Intel® Processor Graphics we recommend reading up to 4x32-bit data per work item
  - 2x32-bit was a good balance of performance and code complexity
  - Input signal in column major order

8-Point FFT, LS 16x1x1, SIMD size 8



Improving Kernel Performance

- Eliminate the use of shared local memory and barriers
  - Maximize device thread occupancy
  - Reduced code complexity
  - Code can be tuned for various architectures
- Perform bit reversal in registers
  - Reduced penalty
  - Preprocessor eliminates index math

- Math transcendentals versus lookup table (LUT)
  - For the **intermediary** twiddle factors
  - Math transcendentals require FRM for performance → poor accuracy
  - LUT in global memory
    - FRM-agnostic
    - <10 percent slower than sin/cos
    - Very good accuracy

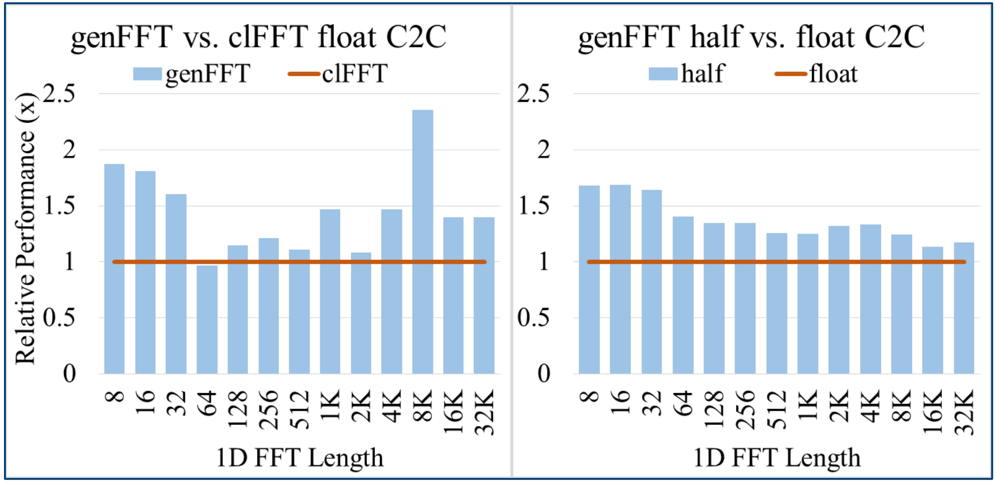
Improving Kernel Performance

- Fuse two or more kernels
  - Straightforward for two consecutive base FFTs of the same length:
    - 8x8, 16x16, 32x32
  - Kernels become function calls with a barrier in between
  - Workgroup size increased to account for data dependencies
  - Can do 64-point FFT and 256-point FFT but can't generalize to any FFT length

- Increase work per work item
  - Straightforward change due to the reduced complexity of our code
  - By itself it doesn't lead to performance gains
  - 2x work per work item coupled with kernel fusing leads to 15–30 percent performance gains for 64-point FFT and 256-point FFT

Results

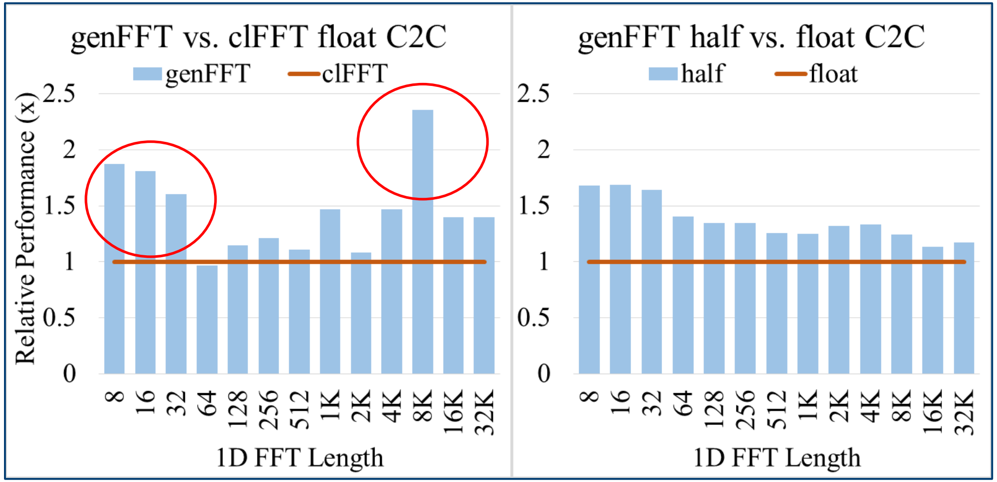
- clFFT is an excellent implementation
- genFFT beats clFFT for:
  - 8, 16, and 32-FFT by more than 1.5x
  - 8k-FFT by more than 2x
  - But penalty for each additional kernel
    - 64 (8x8), 2k (32x8x8), and so on
- clFFT much better at fusing FFTs
  - Big penalty at 8k
- genFFT supports cl_half
  - Additional performance gains
  - There's still room for improvement



Relative performance of genFFT vs clFFT on Intel® HD Graphics 530 capped @ 750 MHz

Results

- clFFT is an excellent implementation
- genFFT beats clFFT for:
  - 8, 16, and 32-FFT by more than 1.5x
  - 8k-FFT by more than 2x
  - But penalty for each additional kernel
    - 64 (8x8), 2k (32x8x8), and so on
- clFFT much better at fusing FFTs
  - Big penalty at 8k
- genFFT supports cl_half
  - Additional performance gains
  - There’s still room for improvement

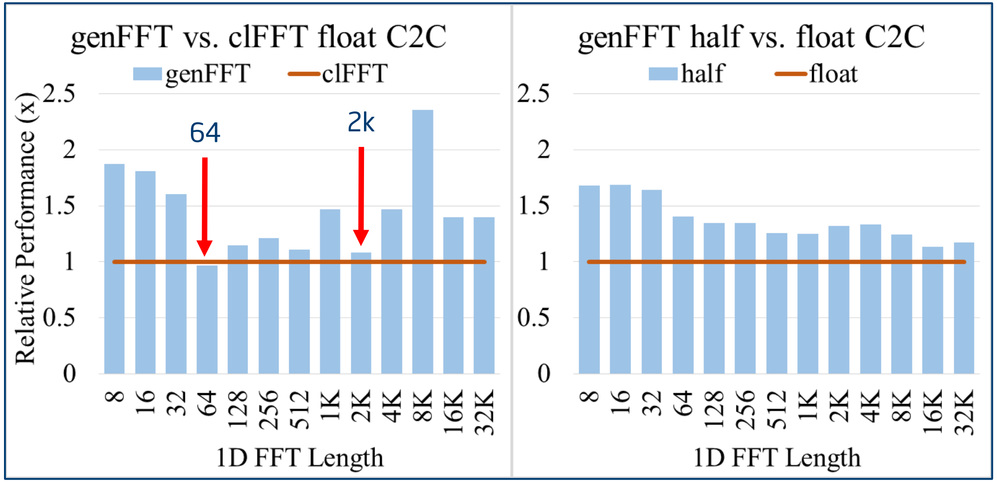


Relative performance of genFFT vs clFFT on Intel® HD Graphics 530 capped @ 750 MHz

Optimization Notice

Results

- clFFT is an excellent implementation
- genFFT beats clFFT for:
  - 8, 16, and 32-FFT by more than 1.5x
  - 8k-FFT by more than 2x
  - But penalty for each additional kernel
    - 64 (8x8), 2k (32x8x8), and so on
- clFFT much better at fusing FFTs
  - Big penalty at 8k
- genFFT supports cl_half
  - Additional performance gains
  - There's still room for improvement



Relative performance of genFFT vs clFFT on Intel® HD Graphics 530 capped @ 750 MHz

Conclusion

- Can be tuned for
  - Variable SIMD size
  - Available register space
  - Compiler ability to promote private memory to registers
  - Any combination of global-local sizes
- Avoids use of shared local memory and barriers
  - Reduced code complexity
  - Potentially better performance portability
- Cache-friendly implementation
- Penalty from enqueueing many kernels

Future Work

- cl_half performance improvements
  - Currently reading 1x32-bit quantities per work item while optimum is 4x32-bit
- Investigate other kernel fusing methods
  - Device-side enqueue
- Intel® Processor Graphics improvement opportunity
  - Optimize execution of pipelines of kernels that reuse buffers
- Expand work to any size 1D FFT and 2D FFT

References

1. Cochran, W.T., and Cooley, J.W. 1967. What is the Fast Fourier Transform. *IEEE Trans. Audio and Electroacoustics* AU-15 (June 1967), 44–55.
2. Cooley, J.W., and Tukey, J.W. 1965. An algorithm for the machine computation of complex Fourier series. *Mathematics of Computation* 19 (90). 297–301.
3. Duhamel, P., and Hollmann, H. 1984. Split-radix FFT algorithm, *Electron. Lett*. 20 (Jan 1984), 14–16.
4. Gaster, B., Kaeli, D. R., Howes, L., Mistry, P., and Schaa, D. 2011. *Heterogeneous Computing With OpenCL*. Elsevier Science & Technology.
5. Intel® Processor Graphics Gen8-Gen9 Developer Guides. 2015. Retrieved from: https://software.intel.com/en-us/articles/intel-graphics-developers-guides.
6. Johnson, S. G., and Frigo, M. 2007. A modified split-radix FFT with fewer arithmetic operations. *IEEE Trans. Signal Process*. 55(1), 111–119.
7. Junkins, Stephen. 2014. The Compute Architecture of Intel® Processor Graphics Gen8. Retrieved from: https://software.intel.com/en-us/file/compute-architecture-of-intel-processor-graphics-gen8pdf.
8. Junkins, Stephen. 2015. The Compute Architecture of Intel® Processor Graphics Gen9. Retrieved from: https://software.intel.com/en-us/file/the-compute-architecture-of-intel-processor-graphics-gen9-v1d0pdf.
9. Khronos OpenCL Working Group. The OpenCL specification version 1.2, 2.0. 2015. Retrieved from: http://www.khronos.org/registry/cl/.
10. Lloyd, D. B., Boyd, C., and Govindaraju, N. 2008. Fast computation of general Fourier transforms on GPUs. Microsoft. IEEE International Conference on Multimedia and Expo. (ICME 2008), 5–8.
11. Lyons, R. G. 2004. *Understanding Digital Signal Processing*, 3rd Ed., Prentice Hall Publishing, Upper Saddle River, NJ.
12. Yavne, R. 1968. An economical method for calculating the discrete Fourier transform. Proc. AFIPS Fall Joint Comput. Conf., Washington DC, 1968, 33. 115–125.

Acknowledgements

Thanks to **Murali Sundaresan**, **Mike MacPherson**, **John Wiegert**, **Tim Bauer**, **Adam Herr**, **Robert Ioffe**, **Jonathan Pearce**, and **Simon Finn** for input and feedback on the FFT work and this presentation!

Most importantly we acknowledge our presenter, **Michal Mrozek**, who kindly volunteered for the task.

Q&A

Thank you!

Legal Disclaimer & Optimization Notice

INFORMATION IN THIS DOCUMENT IS PROVIDED "AS IS". NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

Copyright © 2016, Intel Corporation. All rights reserved. Intel and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries.

Optimization Notice

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804

Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.