POSTPRINT VERSION. The final version is published here :
Jaber, M. A., Massicotte, D., Jaber, R. A. et Nesmith, K. (2019, 26-29 May 2019). An Efficient Data Parallelization of the Radix-23 (Carbon) FFT on GPU/CPU. Communication présentée 2019 IEEE International Symposium on Circuits and Systems (ISCAS), Sapporo. doi: 10.1109/ISCAS.2019.8702380

An Efficient Data parallelization of The Radix-2^3 (Carbon) FFT on GPU/CPU

Marwan A. JABER, Daniel Massicotte, Radwan A. Jaber and Kevin Nesmith*

Université du Québec à Trois-Rivières, Electrical and Computer Engineering Department
Laboratoire des Signaux et Systèmes Intégrés
{marwan.jaber, daniel.massicotte}@uqtr.ca
*Engineering Design, Development, and Research Software
kevin@eddrs.com

***Abstract* – Solving Complex Problem that is coupled with intensive workloads; necessities the access to a massively parallel computational power. Up to date, Graphic Processing Units (GPUs) are the only architecture that could handle the most complex computationally intensive workloads. In the light of this rapid-growing advancement in computational technologies, this paper will propose a high-performance parallel radix-2^3 FFT suitable for such GPU and CPU systems. The proposed algorithm could reduce the computational complexity by a factor that tends to reach p_r if implemented in parallel (p_r is the number of cores/threads) plus the combination phase to complete the required FFT.**

Keywords: Fast Fourier Transform, Parallel FFT, GPU, CPU

1. Introduction

The Fast Fourier Transform (FFT) is one of the most important topics in digital signal processing that have huge impact in various fields of science and engineering such as wireless communication system based on the orthogonal frequency division multiplexing (OFDM) where the FFT is a major key operator [1]-[2]. Seismic processing applications which is computationally demanding [3]-[4], magnetic induction tomography (MIT) [5] and a lot of none cited domains. One of the recent strategies to reduce the computational and the communication load in an FFT process is by grouping the data with its corresponding coefficients multipliers [6]-[8] that will exclude all trivial multiplications (i.e. ± 1 or $\pm j$) which will reduce the accesses to the coefficient multipliers. The radix-2^3 FFT which is based on the same concept, will further reduces the memory accesses by predicting the occurrence of $\pm\sqrt{2}/2 \pm j\sqrt{2}/2$ where the number of arithmetical operations required for the complex multiplication can be reduced from 6 to 2.

The most significant problem in spectral analysis is mainly concentrated in its data's parallel multiprocessing where the computational of the FFTs is challenged by the inter GPU/CPU/threads communications. The most successful methods in parallelizing the FFT on a GPU/multi-GPU/CPU/multi-CPU are presented in [10]-[13].

This paper is organized as follows: Section 2 briefly describes the radix-2^3 FFT. Section 3 will briefly detail the parallel deployment of the proposed parallel FFT. Section 4 will draw the performance results of the proposed method and Section 5 will provide a conclusive summary of our finding.

2. The proposed radix-2^3 FFT

The definition of the DFT is represented by the following equation:

$$X[k] = \sum_{n=0}^{N-1} x[n] w_N^{nk}, \quad k \in [0, N-1], \tag{1}$$

where $x[n]$ is the input sequence, $X[k]$ is the output sequence, N is the transform length, $w_N^{nk} = e^{-j(2\pi/N)nk}$ is called the twiddle factor in the butterfly structure, and $j^2 = -1$. Both $x[n]$ and $X[k]$ are complex number sequences.

The radix-2 $\mathbf{T}_2$ adder tree matrix in the FFT factorization process is defined as [1] and [14]–[21]:

$$\mathbf{T}_2 = \begin{bmatrix} w_N^0 & w_N^0 \\ w_N^0 & w_N^{N/2} \end{bmatrix}. \tag{2}$$

Therefore, by defining $[\mathbf{T}_2]_{l,m}$ as the element of the l^{th} line and m^{th} column in the matrix $\mathbf{T}_2$, as a result equation (2) could be rewritten as:

$$[\mathbf{T}_2]_{l,m} = w_N^{\left\| (lmN/2) \right\|_N}, \tag{3}$$

with l=0, 1, m=0, 1 and $\|\mathrm{x}\|_N$ represents the operation x modulo N. By defining $\mathbf{W}_{N(m,v,s)}$ the set of the twiddle factor matrix during each stage or iteration as

$$[\mathbf{W}_N]_{l,m(v,s)} = \mathrm{diag}\left(w_{N(0,v,s)}, w_{N(1,v,s)}, \ldots, w_{N(r-1,v,s)}\right), \tag{4}$$

where $v = 0,1,\ldots,V-1$ represents the number of words of size 2 with $V = N/2$ and $s = 0,1,\ldots,S$ is the number of stages (or iterations $S = \log_2 N - 1$).

In this subsection, we will be elaborating the decimation in time (DIT) FFT algorithm, where Eq. (4) would be expressed as:

$$[\mathbf{W}_N]_{l,m(v,s)} = \begin{cases} w_N^{\left\| \lfloor v/2^{(S-s)} \rfloor l\, 2^{(S-s)} \right\|_N} & \text{for } l = m \\ 0 & \text{elsewhere} \end{cases}. \tag{5}$$

Consequently, the l^{th} transform output during each stage could be illustrated as

$$X_{(v,s)}[l] = x_{(v,s)}[0] + x_{(v,s)}[1] w_N^{\left\| lN/2 + \lfloor v/2^{(S-s)} \rfloor r^{(S-s)} \right\|_N}. \tag{6}$$

The reading Address Generator (RAG), writing Address Generator (WAG) and the Coefficients (twiddle factor) Address Generators (CAG) are expressed respectively as [19]:

$$\mathrm{RAG}_{(m,v,s)} = m\left(\frac{N}{2^{(s+1)}}\right) + [\![v]\!]_{2^{(S-s)}} + \left\lfloor \frac{v}{2^{(S-s)}} \right\rfloor 2^{(S+1-s)}, \quad (7)$$

$$CAG_{(m,v,s)} = \left[\!\!\left[m\left(lV + \left\lfloor \frac{v}{2^{(S-s)}} \right\rfloor 2^{(S-s)} \right) \right]\!\!\right]_N \quad (8)$$

$$WAG_{(l,v,s)} = l(N/2) + v \quad (9)$$

Eq. (8), reveals that the data is grouped with its corresponding coefficient multipliers in each stage where the l^{th} butterfly's output will shift if and only if $v = 2^s$ in the DIT process. Since $V = N/2 = 2^S$; the total number of shifting during each stage in the DIT process would be 2^s.

The access to the coefficient multiplier's memory could be reduced in comparison to the conventional radix-2 DIT algorithms by implementing a word counter $2^{(S-s)}$ (*word-counter,* β= 0, 1, …,$2^{(S-s+1)}$-1) and a shifting counter 2^s (*shift-counter,* λ= 0,1,…, 2^s-1). The occurrence of the multiplication by one (i.e w^0) could be easily predicted and avoided when the shifting counter is equal to zero (i.e. $v < 2^s$).

With the same reasoning as above, the complexity of the DIT reading generators could be obtained and will be replaced with simple counters.

As the result, the signal flow graph of the proposed radix 2^3 FFT for an 8-point DIT FFT is illustrated in Fig. 1 [22].

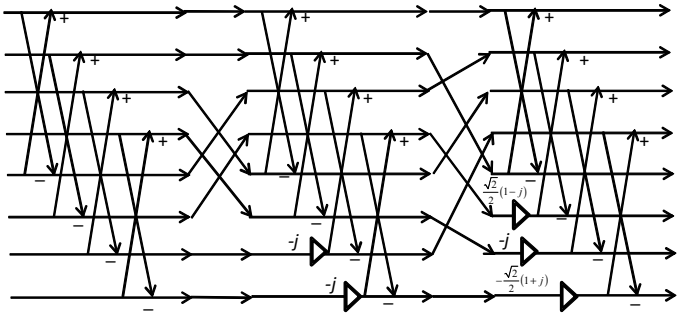

Fig. 1 Signal flow graph (SFG) of 8-point DIT FFT on the proposed structure [22].

The prediction of trivial multiplication in the radix-2^3 FFT can reduce the computational load and will reduce the access to the twiddle factor's memory compared to the most known DIT FFT algorithms [22].

3. The proposed parallel deployment of the FFT Algorithm

By chunking the input data into p_r subsets, the FFT process will be broken into p_r subsets, each of which can be executed by a corresponding GPU/CPU/core as shown in [22]. According to [22], the read/write operations with respect to the chunked data are more likely to be accessed sooner than the original data from the main memory which will increase the hit and miss rate of the read/write policies.

According to Eq. (9) the processed butterfly's output data in the radix-2^3 FFT process and the combination phase, is stored in the memory with a heap of N/p_r as shown in Fig. 2. Several tests have proven that storing the data according to Eq. (9) will degrade the overall performance of the parallel FFT.

By writing the data in a contiguous manner as shown in Fig. 3, we observe a significant improvement in performance of an FFT. Experimental results have proven that the writing and reading processes can have a significant impact on the overall performance of the FFT. When the processor sends a write request for a specific address whose data already is in the process of storing, a write hit is obtained as requested, and subsequently, the data can be modified in the appropriate L1 cache.

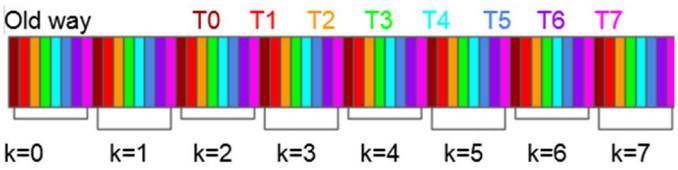

Fig. 2 Conventional writing process for an FFT of size 64 in the radix-2^3 FFT process and the combination phase that are executed in parallel, where the color is associated to the thread T_i with i=0,1,2,…,7.

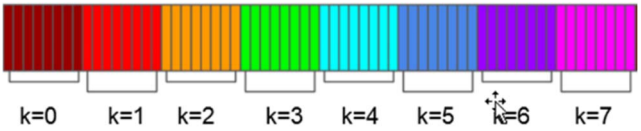

Fig. 3 The proposed writing process for an FFT of size 64 in the radix-2^3 FFT process and the combination phase that are executed in parallel.

4. Performance results

By adopting the same FFT performance as stipulated in [23] which is defined as

$$\mathrm{MFLOPS} = 5N\log_2(N)/t \quad (10)$$

The performance results have been compared to the four well-known FFT methods that have been simulated on CPU and GPU platforms which are listed in Table 1.

Our proposed method has been implemented on the leading available FFT in the market. The results of Fig 4 reflect the improvement of the original proposed multithreaded FFTs (FFTW3, NMKL (Intel's FFT) and NIPP (Intel's FFT)). In this Figure our proposed multithreaded FFTs is referred as NFFTW3, NMKL and NIPP. The obtained results are simulated on Xeon® CPU E5-2686 v4 @ 2.30 GHz with 64 GB RAM (Random Access Memory).

Fig 4 reveals that multithreaded MKL and multithreaded IPP performs better for small FFTs (N<2^7), meanwhile our proposed method NMKL, NIPP and NFFTW3 will maintain the same speed. Compared to the multithreaded FFTW3, multithreaded MKL and multithreaded IPP, our proposed methods manifest a gain in speed by a factor of 3. The most important factor in Fig. 4 is characterized by the increasing

Table I List of references and proposed methods.

Legends	Methods
Methods of reference:	
FFTW3	FFTW method version 3 [23]
MKL	Intel® Math Kernel Library (Intel MKL)
IPP	Intel® IPP FFT functions
FFTW3-16	FFTW3 executed on CPU 16 cores
Proposed Methods:	
NFFTW3	Applied on FFTW3
NMKL	Applied on MKL
NIPP	Applied on IPP
NFFTW3-16	Applied on FFTW3 using 16 cores

speed with the higher number of threads/cores as shown for our proposed NFFTW3-16 (over 16 threads) in which the speed's increment is estimated by a factor of 6.

To be noted that the simulation results of Fig. 4 were performed with writing address generator illustrated in Fig. 2 in which no alignment of the data is performed.

Based on the data alignment as shown in Fig. 3 our proposed radix-2^3 FFT has been parallelized over 256 cores/threads and tested on the V100 NVIDIA® Tesla® accelerated computing platform which has been executed on an Intel® Xeon® CPU E5-2686 v4 @ 2.30 GHz with 64 GB RAM (Random Access Memory). The obtained results of a double precision FFT process with complex input data are compared to cuFFT version 8 and that is executed on TESLA V100 accelerated computing platform with an Intel® Xeon® CPU E5-2686 v3 @ 2.30 GHz with 64 GB RAM [12].

According to [12] the simulation conditions are summarized as follow:

1. Creating Command Queue is not timed.
2. Creating Buffers is not timed.
3. Enqueuing Write Buffers is not timed.
4. Setting Kernel Arguments is not timed.
5. Enqueuing ND Range Kernel is not timed.
6. Enqueuing Read Buffers is not timed.

Since the execution time of an FFT process on NVIDIA's V100 is 1.78 times faster than the execution time of an FFT process executed on NVIDIA's TESLA V100 [13] therefore, the maximum peak N_p of [12] is estimated as:

$$N_p = 800 \times 1.78 = 1.424 \; TeraFLOPS \tag{11}$$

Despite the disadvantage in the size of RAM between our proposed method and NVIDIA's CUDA FFT (CuFFT version 8), Fig. 5 presents the simulation results of our proposed method that was parallelized over 256 threads/cores (C2N256) under the same simulation conditions in which the data was aligned according to Fig. 3 in the partial FFTs and the combination phase [23].

By taking the maximum peak in both methods' N_p and P_p, the speed-up would be:

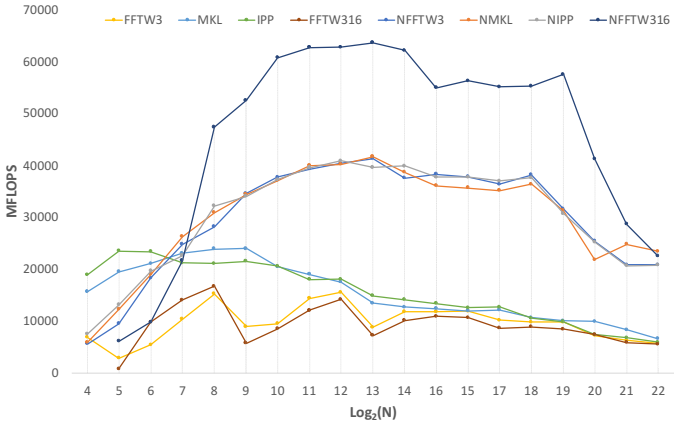

Fig. 4 Simulation results on CPU Intel® Xeon® CPU E5-2686 v4 @ 2.30 GHz of the proposed method with other referenced methods according to Table 1.

$$S = \frac{P_p}{N_p} = \frac{650}{1.424} = 456 \tag{12}$$

If the downloading time of the data to the GPU should be included in the FFT performance metrics as in [10] in which the performance metric is in GigaFLOPS expressed by the following equation:

$$\text{GFLOPS} = 5N\log_2(N) \times batch \times 10^{-9} / t \tag{13}$$

where batch ($batch = 2^{24}/N$ [10]) is the total amount of data sequences being processed and t is the time (in second).

Since the simulation results in the cited reference [10] is performed on K20 and K40 therefore, the results of the cited reference will be limited to K40 whose results are multiplied by 1.8 [12] and then multiplied by 1.8 [13]. To have a fair comparison, the simulations results in the cited reference [10] should be adjusted by a factor of 3.24. As a result, the gap in Fig. 5 should be divided by a factor of 3.24.

The performance results which is based on the metric defined in Eq. (13) is illustrated in Fig. 5. By adopting the same FFT performance, Fig. 6 reveals that our proposed method is running in hundreds of TeraFLOPS which is an advantage and assets that no existing method has. The speed up between the cited methods in [10] and the proposed method is illustrated in Fig. 7. The performance of the proposed method in Fig. 6 was obtained considering $batch$ = 16 384 for all FFT length.

In conjunction with the systems, methods, and devices described above, that can be configured to implement a parallelized FFT in which trivial multiplications and additions can be omitted, and the FFT can be broken into pieces. By doing so, each core/thread can process its portion through multiple stages without inter-core communications that might otherwise introduce delays.

Further, the FFT process provides an address generator that allows the FFT data to be written to and read from contiguous memory addresses, reducing memory access overhead and thereby improving overall performance

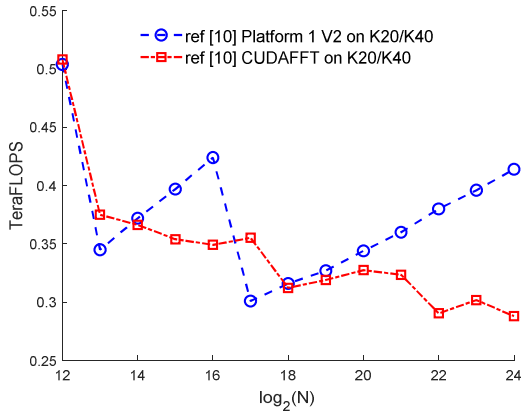

Fig. 5 Performance results in terms of TeraFLOPS proposed method and the cited in [10].

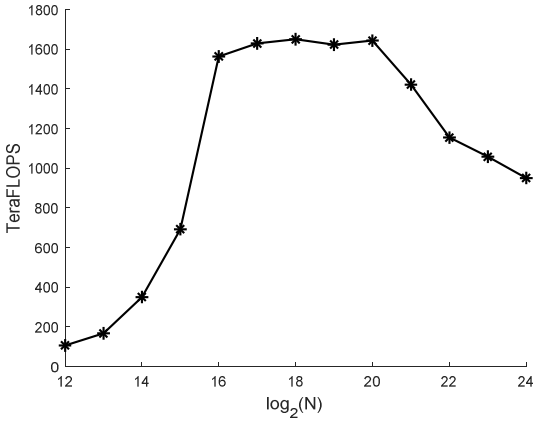

Fig. 6 Simulation results of the proposed method under the same conditions of [12] on Volta V100 SMX2 16 GB. Host system Intel® Xeon® 16 cores CPU E5-2686 v4 @ 2.30GHz with 64 GB system memory.

It should be appreciated that the FFT of size N may be broken into multiple parts according to the number of GPU/cores p_r, such that each core processes a portion of the FFT of size N/p_r. Each of the FFT portions may then be provided to one of the GPU/cores, which may be configured to compute the FFT value. Rather than exchanging data between the cores, the parallelized FFT is configured to combine and reorder the data in a single combination stage to provide the FFT in a natural order.

5. Conclusion

This paper has presented a more efficient way for data parallelization to compute Carbon's FFT on a GPU and CPU where the optimization process is heavily concentrated in the writing policies of the radix-2^3 FFT and the combination phase. According to cited literature and up to our knowledge, we have presented an FFT that runs in hundreds of Teraflops on a GPU. Furthermore, for a large FFTs, our proposed method could be partitioned and scaled on multiple GPU which is an advantage and assets that no existing method has. We strongly believe that the obtained results could be greatly improved by applying the proposed concept that should be parallelized over multiple GPUs in each of which the partial FFTs are parallelized over multiple cores/threads where the inter-core communications will be reduced to its minimum value.

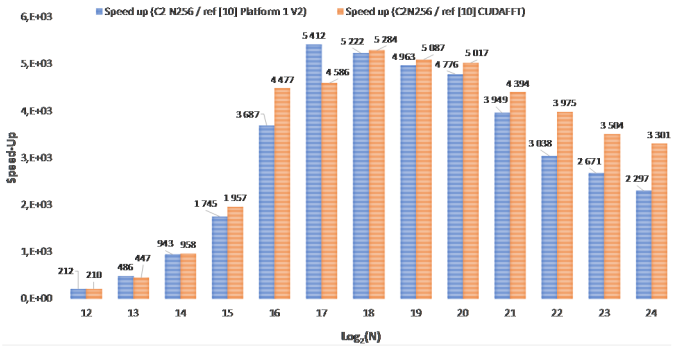

Fig. 7 Speed-up of the proposed method on V100 SMX2 16GB compared to methods cited in [10], MS-ID-FFT.V1/V2 CUDAFFT on K20/K40.

Acknowledgements

This work has been funded the Natural Sciences and Engineering Research Council of Canada grants with the collaboration of Erik Drewry Carbon-technologies' CEO (carbon-technologies.com).

References

[1] T. Widhe, "Efficient Implementation of FFT Processing Elements" Linkoping studies in Science and Technology, Thesis No. 619, Linkoping University, Sweden, June 1997.

[2] G. Klang, "A Study of OFDM for Cellular Radio Systems, M.Sc. Thesis, Linkoping University, Sweden 1994.

[3] D. Komatitsch et al, "High-order finite-element seismic wave propagation modeling with MPI on a large GPU cluster," Journal of Computational Physics, Elsevier, vol. 229, no 20, 1 Oct. 2010, pp. 7692-7714.

[4] Y. Cui et al, "Physics-based seismic hazard analysis on peta-scale heterogeneous supercomputers," Proceedings the International Conference for High Performance Computing, Networking, Storage and Analysis Denver, Colorado - 17-22 Nov. 2013, pp. 1-12.

[5] Y. Maimaitijiang et al, "Evaluation of Parallel FFT Implementations on GPU and Multi-Core PCs for Magnetic Induction Tomography", World Congress on Medical Physics and Biomedical Engineering, Sept. 7 - 12, 2009, Munich, Germany, pp 1889-1892.

[7] Y. Wang et al, "Novel Memory Reference Reduction Methods for FFT Implementations on DSP Processors," IEEE Transactions on signal processing, Vol. 55, No. 5, May 2007, pp. 2338-2349.

[8] T. Sun, Y. Yu, "Memory Usage Reduction Method for FFT Implementations on DSP Based Embedded System", IEEE International Symposium on Consumer Electronics, 25-28 May, 2009, pp. 812-815.

[10] A. Perez Dieguez et al, "Solving Large Problem Sizes of Index-Digit Algorithms on GPU: FFT and Tridiagonal System Solvers," IEEE Transactions on Computers, vol. 67, no. 1, January 2018 pp. 86-101.

[11] X. Zhang et al, "Design and Implementation of Parallel FFT on CUDA," IEEE International Conference on Dependable, Autonomic and Secure Computing, 21-22 December 2013, Chengdu, Sichuan, China, 26 June 2014, pp. 583-589.

[12] "CUDA 8 Performance Overview," Accelerated computing performance report, Nvidia, Nov. 2016, 43 pages.
[13] P. D'Souza and A. Nitsure, "NVidia V100 Performance Characteristics on IBM AC922 (POWER9) System and HPC Application Performance, ISDL, IBM India Pvt Limited.
[14] M. Jaber, D. Massicotte, "A New FFT Concept for Efficient VLSI Implementation: Part I – Butterfly Processing Element", 16th International Conference on Digital Signal Processing (DSP'09), Santorini, Greece, 5-7 July 2009, pp. 1-6.
[15] M Jaber and R. A Jaber, "Apparatus and Methods of Providing Efficient Data Parallelization for Multi-Dimensional FFTs" US Patent Application No.: 15/981,331 and International PCT application Number PCT/US2018/032957.
[16] M. Jaber, D. Massicotte, R. Jaber, "Radix-2^3 Fast Fourier Transform for an Embedded Digital Signal Processor," US patent Application No.: 62/677,610.
[17] M. Jaber and R. A. Jaber "Apparatus and Methods of Providing Efficient Data Parallelization for Multi-Dimensional FFTs" US Patent Application No.: 15/981,331 and International PCT application Number PCT/US2018/032957.
[18] M. Jaber, D. Massicotte and Y. Achouri, "A Higher Radix FFT FPGA Implementation Suitable for OFDM Systems," IEEE International Conference on Electronics, Circuits, and Systems (ICECS), Beirut Lebanon, Dec. 2011, pp.1-4.
[19] M. Jaber, D. Massicotte, "The Self-Sorting JMFFT Algorithm Eliminating Trivial Multiplication and Suitable for Embedded DSP Processor," IEEE International NEWCAS Conference June 17-20, 2012, Montreal, Quebec, pp. 1-4.
[20] M. Jaber, "Address Generator for the Fast Fourier Transform Processor" US-6,993, 547 82 and European patent application Serial no: PCT/USOI /07602.
[21] M. Jaber, D. Massicotte, "A New FFT Concept for Efficient VLSI Implementation: Part II – Parallel Pipelined Processing ", 16th International Conference on Digital Signal Processing (DSP'09), Santorini, Greece, 5-7 July 2009.
[22] M. Jaber, "Low complexity and high performance of the Fourier transform," Ph.D. Thesis, Université du Québec à Trois-Rivières, 2012.
[23] http://www.fftw.org