A FAST MIXED-BAND LIFTING WAVELET TRANSFORM ON THE GPU

Tran Minh Quan and Won-Ki Jeong

School of Electrical and Computer Engineering,
Ulsan National Institute of Science and Technology (UNIST), Ulsan, South Korea
E-mail: {quantm, wkjeong}@unist.ac.kr

ABSTRACT

Discrete wavelet transform (DWT) has been widely used in many image compression applications, such as JPEG2000 and compressive sensing MRI. Even though a lifting scheme [1] has been widely adopted to accelerate DWT, only a handful of research has been done on its efficient implementation on many-core accelerators, such as graphics processing units (GPUs). Moreover, we observe that rearranging the spatial locations of wavelet coefficients at every level of DWT significantly impairs the performance of memory transaction on the GPU. To address these problems, we propose a *mixed-band* lifting wavelet transform that reduces uncoalesced global memory access on the GPU and maximizes on-chip memory bandwidth by implementing in-place operations using registers. We assess the performance of the proposed method by comparing with the state-of-the-art DWT libraries, and show its usability in a compressive sensing (CS) MRI application.

***Index Terms*—** Mixed-band, Wavelet, Denoising, CUDA, GPU, Parallel Computing, Compressive Sensing, MRI.

1. INTRODUCTION

Wavelet transform is a widely used sparsifying transformation having a broad range of signal processing applications. In a conventional DWT, the input signal is recursively convolved with high- and low-pass filters to separate different frequency bands where each low-pass filtered signal serves as an input to the next-level decomposition. One required step in this approach is to rearrange the low- and high-frequency coefficients so that the coefficients from the same frequency band are grouped and stored together (Fig. 1 upper row). This reallocation has to be performed at the end of each DWT decomposition process, which results in write memory transactions with a large stride and can be a performance bottleneck on the GPU. There have been research efforts to make DWT faster on multi- and many-core architectures [2, 3, 4, 5], but none of them has fully addressed this write memory transaction problem yet.

The authors would like to thank Sohyun Han, professor Hyungjoon Cho and professor Arvid Lundervold for providing the MRI datasets. This work has been partially supported by the National Research Foundation of Korea (NRF-2012R1A1A1039929, NRF-2013K2A1A2055315).

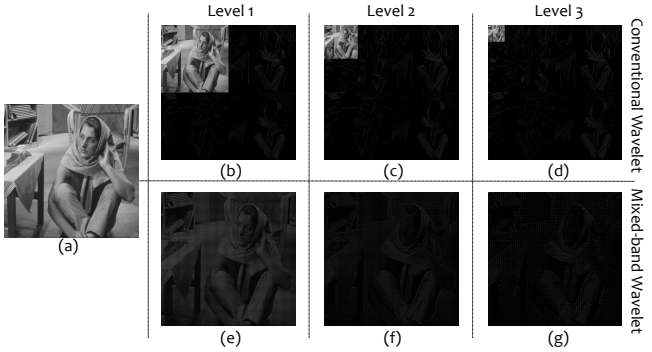

Fig. 1: Barbara image (a), the first 3-level conventional (b), (c), (d) and mixed-band wavelet decompositions (e), (f), (g).

A lifting scheme by Swelden [1] is a computationally-efficient, second-generation wavelet transform that allows in-place filtering. However, its efficient implementation on the GPU has been studied only recently. Van Der Laan et al. [4, 5] proposed a sliding window technique for a lifting wavelet transform on the GPU. The proposed method successively applies a 1D wavelet transform and writes intermediate results back to global memory in two passes (horizontal and vertical). In order to reuse data, their procedure actively caches the necessary data in shared memory with multi-column processing in the vertical pass and performs DWT without a transpose of the data matrix. Therefore, it can be classified as a *line-based* approach where the transform along each dimension is done separately and expensive global memory communication must be involved in-between. Unlike this previous work, we observed that employing a *block-based* approach can potentially increase the performance and scalability for large data by processing multiple levels of wavelet decomposition on fast shared memory without communicating to global memory, which also fits better to the optimization strategies such as tiling and privatization for massively parallel processing introduced in [6].

The main contribution of this paper is two-fold. First, we propose a *mixed-band* wavelet transform that effectively re-

duce memory transaction overhead on the GPU. Since we realize that completely removing uncoalesced memory access is unfeasible in wavelet transform, our approach is to minimize it by modifying the memory access pattern. Unlike the conventional wavelet transform that splits high- and low-frequency subbands, our method does not separates them, which enables read and write location from global memory is always same (Fig. 2 right). This simple modification results in reducing the total uncoalesced memory transactions and significantly improves the performance of DWT on the GPU. Second, we propose a *block-based* multi-level update scheme using registers so that on-chip memory bandwidth is maximized and expensive (compared to registers) shared and global memory transactions are reduced. We introduce a full-3D compressive sensing MRI reconstruction as an application of the proposed mixed-band lifting DWT on the GPU.

2. METHOD

2.1. Mixed-band wavelet transform

The core idea of mixed-band transform is to reduce uncoalesced memory transactions by modifying the memory access pattern. To do this, the mixed-band method does not rearrange the locations of output wavelet coefficients based on the frequency bands – it writes each resulting coefficient back to the original location. This is an unconventional way to arrange wavelet coefficients, but there is no problem as long as the inverse wavelet transform can reconstruct the original image. Fig. 2 is a pictorial description of 1D Haar wavelet transform in conventional (left) and mixed-band (right) approaches. If we consider that the wavelet transform is running on the GPU, then the input and output pixel arrays can be viewed as global memory and the dotted box for wavelet transform is regarded in shared memory. In the conventional wavelet transform (Fig. 2 left), read memory transactions are always coalesced (blue arrow) and write memory transactions are always uncoalesced (red arrow). This is because each wavelet decomposition process separates subbands so that wavelet coefficients from the same frequency band will be stored into a contiguous block of memory. Therefore, write memory transactions will be uncoalesced with a stride of the subband size, while read memory transactions are always coalesced. If we do not segregate subbands, then read and write transactions in the first level of wavelet decomposition are coalesced, but the rest of memory transactions are all uncoalesced (Fig. 2 right).

One may wonder why the mixed-band transform performs better because in Fig. 2 the conventional approach seems to have more coalesced transactions than the mixed-band approach (four versus two). This is because all the memory transactions in the first level of the mixed-band approach are coalesced and it is the dominant factor affecting the entire performance. In fact, the wavelet transform reduces the domain size by $\frac{1}{4}$ for 2D and $\frac{1}{8}$ for 3D per every decomposition step.

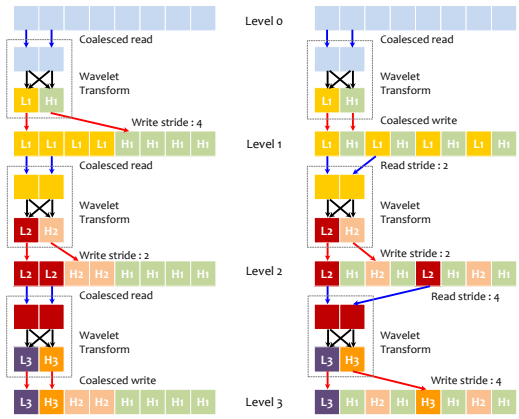

Fig. 2: Memory transactions from global memory to shared memory (dotted boxes) on the GPU of conventional (left) and mixed-band (right) Haar wavelet transform.

Therefore, if we consider the first three levels in 2D wavelet decomposition, the total uncoalesced memory transactions in the conventional approach is $n + \frac{1}{4}n = 1.25n$ and that of the mixed-band approach is $2\frac{1}{4}n + 2\frac{1}{8}n = 0.75n$ (n is the data size). We can easily generalize this and show that the mixed-band approach always requires less number of uncoalesced memory transactions than the conventional approach. Since uncoalesced GPU global memory transaction is slow, the proposed mixed-band approach can significantly reduce memory access time on the GPU.

2.2. Block-based GPU lifting scheme

As we have seen in the previous section, global memory access can be optimized by employing a mixed-band method. Further improvement can be made in the actual wavelet transform process – dotted boxes in Fig. 2. One approach is using shared memory to reuse data multiple times without going back and forth from the GPU core to global memory. For example, if we implement a separable wavelet transform, then whenever an 1D wavelet decomposition is applied to one direction, the intermediate result has to be written back to global memory in order to continue 1D decomposition on the next direction. Our approach uses shared memory to read in a *block* of data and process multi-level wavelet decomposition of the entire block. For instance, we can load an $8\times8\times8$ block, perform three levels of wavelet decomposition (each level's decomposition consists of three 1D wavelet transforms, one per each axis in 3D), and write back the result (single low-frequency coefficient and 511 high-frequency coefficients) to global memory. It is still a separable wavelet transform, but intermediate results are stored in shared memory during the process.

In addition, we can further reduce shared memory access by maximizing the use of registers. This is due to the fact that the memory bandwidth of register is much higher

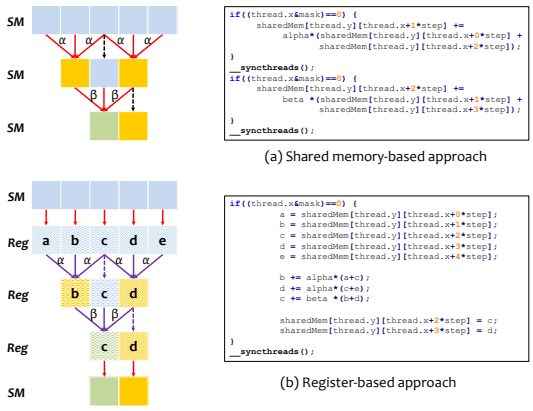

Fig. 3: Two lifting scheme implementations of biorthogonal CDF5/3 wavelet and their NVIDIA CUDA code snippets.

than that of shared memory even though both are on-chip memory [7]. Fig. 3 shows two different implementations of Cohen-Daubechies-Feauveau (CDF) 5/3 wavelet transform using a lifting scheme. In the original lifting scheme, the data is first split into odd and even arrays. Next, the odd array values are predicted from three values using a weight $\alpha = -1/2$. We then update the even array from the predictions with the coefficient $\beta = 1/4$, which completes one level of wavelet decomposition. Fig. 3 (a) is a naive implementation where in-place computation of the lifting scheme is done via writing back and forth from shared memory. In contrast, Fig. 3 (b) shows the implementation using registers as temporary memory, where only red arrows represent data read from or write to shared memory. As shown in this illustration, the total number of shared memory transactions is reduced in register-based implementation. For example, if we consider a 1D block of eight pixels, then in the first step of (a) 5 intermediate results have to be computed, which results in 15 reads and 5 writes of shared memory. After the first step, all threads must be synchronized in order to share the intermediate results. In the second step, only 4 output values will be computed by 3 reads and 1 write of shared memory per each value. Assuming that we are using four concurrent threads to produce eight values, 6.75 reads, 2.25 writes, and one `__syncthreads()` function call are required per thread. In contrast, the register-based implementation shown in (b) only requires 5 reads and 2 writes of shared memory per thread without a `__syncthreads()` call in between. In practice, we observed up to 20% of performance gain when registers are used as a replacement of shared memory.

3. RESULTS

3.1. Comparison of mixed-band wavelet performance

In this section, we evaluate the performance of our mixed-band lifting wavelet transform. We implemented our optimized single core CPU and many core GPU version of a conventional and mixed-band lifting 2D Haar wavelet transform, and compared them with wavelet implementations used in kt-SPARSE [8] (MATLAB) as well as GNU Scientific Library (GSL) [9] (C). Table 1 lists the running time of each method in double-precision input data with various sizes for full transform (up to the maximum level). For a small data size, MATLAB version appears to be slower than GSL version, but it gets faster as the data size grows. Nevertheless, our optimized single core CPU version is faster than both of them under any test case. Our mixed-band lifting wavelet on a single core CPU is slower than our optimized conventional wavelet for small and medium image sizes, but it outperforms the conventional wavelet for large data size. This can be explained that uncoalesced memory access effect on small data is hidden by processor's cache but it is more visible when the data size outgrows the cache size (such as the image of size 4096×4096 pixels). The last row reports our mixed-band wavelet on an NVIDIA Fermi GPU (M2090). As shown in this table, the GPU mixed-band lifting wavelet gains a nearly 700× speed up compared to a single core optimized wavelet transform. Other wavelet families, such as CDF5/3, CDF9/7 [10], require larger supports from the neighbor pixels and are obviously difficult to achieve multi-level decomposition using shared memory in a single GPU kernel call. However, if we consider specific applications, such as JPEG compression (Discrete Cosine Transform is applied for each block of 8×8), block-based mixed-band lifting CDF5/3 wavelet can be useful.

We also tested our mixed-band lifting wavelet on 3D datasets. Three-level Haar wavelet decomposition on a 512^3 3D volume took around ∼60 milliseconds on the GPU. We briefly compared this with the result of Van Der Laan et al. [5]. In our experiment, processing three levels of a double precision 512^3 volume results in about 8.9GB/s throughput. Van Der Laan et al. reported that the volume of same size with 16 bit integer data took about 150ms, which is roughly 1.7GB/s throughput. Even we consider the effect of different hardware used for testing (GF110 we used could be about 3∼4 times faster than G80), our mixed-band lifting scheme is faster than (or comparable to) existing GPU lifting wavelet methods.

3.2. Application: 3D CSMRI reconstruction

To further assess the usability of the proposed GPU mixed-band lifting wavelet transform, we applied our wavelet transform as a regularizer in a 3D CSMRI reconstruction problem, which can be generally formulated as follows:

$$\min_{U} \{J(U)\} \quad s.t. \quad \sum \|Ku - f\|_2^2 < \sigma^2 \qquad (1)$$

where $J(U)$ stands for the energy function we want to minimize, f is the measurement of the MRI data (i.e., *k-space*), and $K = RF$ is the matrix multiplication that combines a

Table 1: Running time of conventional and mixed-band lifting 2D full Haar DWT on the CPU and GPU. Unit: msec

2D Haar	Image size	64×64		512×512		4096×4096	
	Direction	Forward	Inverse	Forward	Inverse	Forward	Inverse
Conventional	MATLAB CPU	50.129	51.213	317.39	315.13	9319.7	9432.5
	GSL CPU	0.692	0.680	322.90	318.56	282986.4	282546.5
	Optimized CPU	0.234	0.219	21.25	20.23	3473.6	3417.2
Mixed-band	Optimized CPU	0.261	0.281	26.13	27.66	3257.9	3229.7
	Optimized GPU	0.008	0.009	0.081	0.081	4.599	4.597

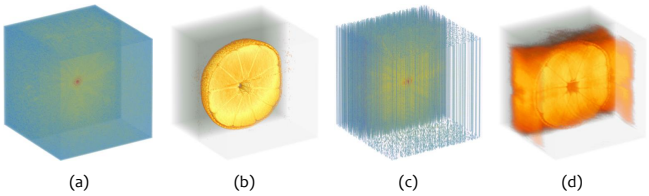

Fig. 4: Full 3D k-space (a), ×4 undersampling k-space (c) and their reconstructions (b), (d), respectively.

sampling pattern R and a Fourier matrix F for 3D data. Examples of k-spaces and their reconstructions are shown in Fig. 4. $J(U)$ can comprise many terms, for example Total Variation (TV) [11], Wavelet constraint [11, 8], Fourier constraint [8, 12], and the fidelity of the data. More recent GPU implementations on MRI reconstructions with different solvers can be found in [13] and references therein.

For instance, a 3D CSMRI energy function is described as follows:

$$J(U) = \|\nabla_{xyz} U\|_1 + \|W_{xyz} U\|_1 \quad (2)$$

In this equation, ∇ is the gradient operator that enforces the smooth variation of pixel intensity in spatial dimensions, and W is a 3D DWT. In order to solve the minimization problem of Eq. 1 using the energy function given above, we use the Split Bregman iterative algorithm [11].

We include the wavelet term in $J(U)$ and compare its reconstruction to the zero filling version and without wavelet, on 3D fruit MRI in term of Peak-Signal-To-Noise ratio (PSNR) to the full reconstruction. The experiments are conducted with $\times 4$ and $\times 8$ mask patterns on the kiwi and tangerine data set (size 128×128×128). It took ∼7 seconds to run the solver on the GPU with the number of inner and outer loops are 32 and 16, respectively. Kim et al. [14] reported a similar study result (3D Split Bregman on the GPU) in [14]. Although it is difficult to make a direct comparison between two methods since the testing environment and energy function are different, we concluded that our method is roughly ∼2.5× faster than their method after proper scaling of parameters. Note that Kim et al. did not include wavelet transform so our CSMRI solver is heavier in arithmetic intensity, but still outperforms Kim's method for the same number of iterations.

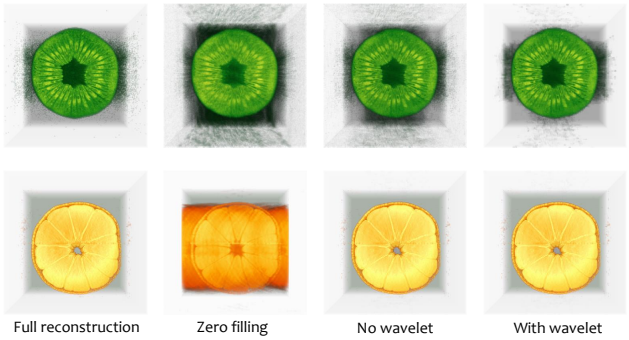

Fig. 5: CSMRI reconstruction on 3D data with ×4 mask on kiwi (first row) and tangerine dataset (second row)

As shown in Table 2, adding a 3D wavelet term can effectively increase the PSNR of the reconstructed images. Fig. 5 visualizes the CSMRI 3D volumes of kiwi and tangerine data which have been cut at the middle to show the inside. It can be seen that the wavelet term helps to decrease the noise-like effects and enhance the quality of 3D MRI reconstruction.

Table 2: Comparison of PSNRs of 3D CSMRI reconstruction on 128×128×128 datasets. Unit: dB

Dataset	Mask	Zero filling	No Wavelet	With Wavelet
Kiwi	×4	18.7245	27.8483	28.3488
	×8	17.2263	25.6509	26.1510
Tangerine	×4	22.0418	35.6646	43.6321
	×8	20.6875	33.3996	36.2106

4. CONCLUSION

In this paper, we introduce a fast 3D mixed-band lifting DWT on the GPU and its application in the CSMRI reconstruction problem. Our solution overcomes the main bottlenecks of conventional 3D DWT by minimizing uncoalesced memory access and employing a block-based multi-level processing method based on fast GPU register memory. We also showed that the proposed wavelet transform can effectively improve the quality of the CSMRI reconstruction. In the future, we plan to apply our mixed-band method to various wavelet families and real-time clinical MRI reconstruction.

5. REFERENCES

[1] W. Sweldens, "The lifting scheme: A construction of second generation wavelets," *SIAM J. Math. Anal.*, vol. 29, pp. 511–546, Mar. 1998.

[2] J. Franco, G. Bernabe, J. Fernandez, and M. Acacio, "A parallel implementation of the 2D wavelet transform using CUDA," in *2009 17th Euromicro International Conference on Parallel, Distributed and Network-based Processing*, pp. 111–118, 2009.

[3] J. Franco, G. Bernabe, J. Fernandez, and M. Ujaldon, "Parallel 3D fast wavelet transform on manycore GPUs and multicore CPUs," *Procedia Computer Science*, vol. 1, pp. 1101–1110, May 2010.

[4] W. Van der Laan, J. B. T. M. Roerdink, and A. Jalba, "Accelerating wavelet-based video coding on graphics hardware using CUDA," in *Proceedings of 6th International Symposium on Image and Signal Processing and Analysis, 2009. ISPA 2009*, pp. 608–613, 2009.

[5] W. Van der Laan, A. Jalba, and J. B. T. M. Roerdink, "Accelerating wavelet lifting on graphics hardware using CUDA," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 1, pp. 132–146, 2011.

[6] J. Stratton, C. Rodrigues, I.-J. Sung, L.-W. Chang, N. Anssari, G. Liu, W.-M. Hwu, and N. Obeid, "Algorithm and data optimization techniques for scaling to massively threaded systems," *Computer*, vol. 45, no. 8, pp. 26–32, 2012.

[7] V. Volkov and J. W. Demmel, "Benchmarking GPUs to tune dense linear algebra," in *Proceedings of the 2008 ACM/IEEE Conference on Supercomputing*, SC '08, (Piscataway, NJ, USA), pp. 31:1–31:11, IEEE Press, 2008.

[8] M. Lustig, D. Donoho, and J. M. Pauly, "Sparse MRI: the application of compressed sensing for rapid MR imaging," *Magnetic Resonance in Medicine*, vol. 58, no. 6, pp. 1182–1195, 2007.

[9] M. Galassi, J. Davies, J. Theiler, B. Gough, G. Jungman, M. Booth, and F. Rossi, *GNU Scientific Library: Reference Manual*. Network Theory Ltd., Feb. 2003.

[10] I. Daubechies, *Ten Lectures on Wavelets*. CBMS-NSF Regional Conference Series in Applied Mathematics, Society for Industrial and Applied Mathematics, 1992.

[11] T. Goldstein and S. Osher, "The split bregman method for $\ell 1$–regularized problems," *SIAM Journal on Imaging Sciences*, vol. 2, pp. 323–343, Jan. 2009.

[12] H. Jung, J. C. Ye, and E. Y. Kim, "Improved k–t BLAST and k–t SENSE using FOCUSS," *Physics in Medicine and Biology*, vol. 52, p. 3201, June 2007.

[13] M. Murphy, M. Alley, J. Demmel, K. Keutzer, S. Vasanawala, and M. Lustig, "Fast $\ell 1$-SPIRiT compressed sensing parallel imaging MRI: scalable parallel implementation and clinically feasible runtime," *IEEE Transactions on Medical Imaging*, vol. 31, no. 6, pp. 1250–1262, 2012.

[14] D. Kim, J. Trzasko, M. Smelyanskiy, C. Haider, P. Dubey, and A. Manduca, "High–performance 3D compressive sensing MRI reconstruction using many–core architectures," *Journal of Biomedical Imaging*, vol. 2011, Jan. 2011.