Vector FPGA Acceleration of 1-D DWT Computations using Sparse Matrix Skeletons

Sidharth Maheshwari, Gourav Modi, Siddhartha, Nachiket Kapre
School of Computer Science and Engineering
Nanyang Technological University
Singapore - 639798
Email: m.sidharth@ntu.edu.sg

***Abstract*—We can exploit application-specific sparse structure and distribution of non-zero coefficients in Discrete Wavelet Transform (DWT) matrices to significantly improve the performance of 1-D DWT mapped to FPGA-based soft vector processors. We reformulate DWT computations specifically in terms of sparse matrix operations, where the transformation matrices have a repeating block with a fixed non-zero pattern, which we refer to as a skeleton. We exploit this property to transform the original DWT matrix into a Modified-Matrix-Form to expose abundant soft vector parallelism in the dot products. The resulting form can also be readily compiled into low-level DMA routines for boosting memory throughput. We auto-generate vector routines and memory access sequences tailored for parametric combinations of DWT filter sizes, and decomposition levels as required by the application domain. When compared to embedded ARMv7 32b CPU implementations using optimized OpenBLAS routines, soft vector implementation on the Xilinx Zedboard and Altera DE2/DE4 platforms demonstrate speedups of 12–103x.**

I. Introduction

One-dimensional (1-D) DWT finds application in biomedical signal processing [5], broadband sonar signal processing [1], financial data mining [8], among other domains. Real-time applications involving 1-D signals have witnessed a significant increase in volume of data and rates of their production. This data must be rapidly crunched to operate at high speeds and high accuracy within microseconds [2], particularly in the high frequency finance and biomedical embedded scenarios. The key challenges in the evaluation of DWT are to uncover inherent data parallelism in the algorithm and computation of high volume of data with strict energy and performance constraints.

Among the various implementation formulations, the matrix form realization of DWT provides abundant opportunities for parallelization at the expense of potential storage overheads. The core computations are SIMD-friendly matrix-matrix or matrix-vector arithmetic operations on fixed-point data. In this paper, we propose novel implementation strategies to exploit available inherent parallelism in matrix-form DWT algorithm using soft vector processor. In our implementation, we exploit the sparsity pattern observed in the construction of transformation matrices (TM), and show a representative example in Figure 1. For different DWT parameters, TM can be expressed in the form of short *skeletons* (non-sparse fragments) which have a fixed repetitive pattern within the full TM matrix. We exploit the *skeleton* (SK) (Figure 1) to reduce matrix-vector multiplication involved in 1D DWT evaluation to scalar-vector products, thereby, reducing the complexity from $\mathcal{O}(n^2)$ to $\mathcal{O}(n)$. This, however, comes at the cost of rearrangement penalties on the input data. Apart from the algorithmic modifications, our implementation strategy includes customized direct memory access (DMA) transfers that are tuned to hide data loading costs between the on-chip and off-chip memories. The proposed DWT accelerator is implemented using the soft vector processor, Vectorblox MXP [6], as an FPGA IP core, on different boards such as Altera DE4, DE2 and Xilinx Zedboard. We compare our optimized MXP implementations to similarly optimized OpenBLAS [9] routines on ARM 32b CPUs on the Zedboard and the Beaglebone Black and ARMv6 32b CPU on Raspberry Pi. The contributions in this paper are:

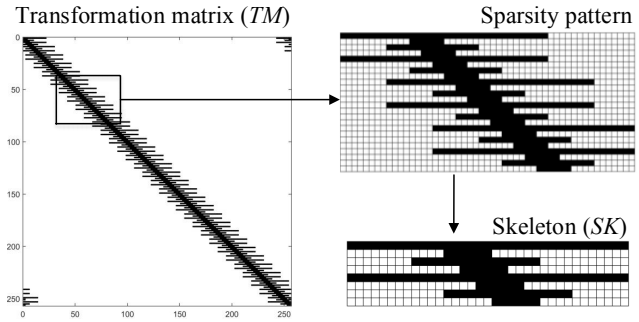


Fig. 1: Transformation matrix (TM) for 6-tap DWT filter ($L = 6$) and 3-level decomposition ($k = 3$) with the noticeable sparsity pattern along with the skeleton (SK) of size 8×36 are shown beside it. The black squares represent the non-zero elements of the skeleton.

- Exploitation of inherent parallelism in matrix-form fixed-point 1D DWT evaluations using FPGA-based Vectorblox MXP soft vector processor.
- Identification of sparse memory patterns in multi-level DWT transformation matrices to obtain memory skeletons for efficient hardware resource utilization.
- Customized data rearrangement for optimized DMA transfers and reduction of matrix-vector arithmetic of 1D DWT to low complexity scalar-vector operations.

- Characterization of performance and power of ARMv7, ARMv6 and MXP implementations across various DWT image/signal sizes, and transformation levels.

II. Background

A. MXP Soft Vector Processor

VectorBlox MXP, shown in Figure 2, is a fixed point, multi-platform, customizable FPGA-based soft-vector processor [6]. MXP primarily consists of parallel 32-bit ALUs organized into multiple SIMD lanes, supported by an on-chip scratchpad and DMA engine for data transfers. The host processor that drives the MXP operation can be a soft processor, *e.g.* Altera Nios II/f or Xilinx Microblaze, or a hard processor like the ARM CPU in the Zedboard. The host processor orchestrates DMA transactions and also issues vector instructions to the MXP unit. The DMA engine transfers data between off-chip RAM and on-chip scratchpad. For our work, we use Altera DE4 which has 32 lanes and 128 kB scratchpad running at ≈184 MHz and Altera DE2 and Xilinx Zedboard that have 16 lanes and 64kB scratchpad running at 100 MHz each. The DMA engines on DE4, DE2 and Zedboard support data transfer of 32, 8 and 4 bytes/cycle respectively.

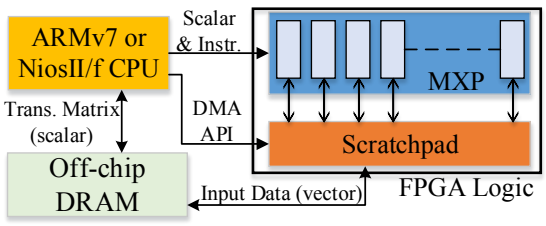


Fig. 2: High-Level picture of the MXP soft vector processor coupled with a host CPU.

B. DWT Using Matrix Form

A DWT implementation is characterized by three parameters – (1) N, length of the input signal, (2) L, length of the filter coefficients, and (3) k, decomposition level of the design. DWT computation involves low-pass $h[l]$ and high-pass $g[l]$ filtering operations followed by down-sampling by a factor of 2 as shown in Equation 1. Number of resulting low-pass and high-pass coefficients, represented as cA and cD in Equation 1, are each equal to half of the number of original input samples (*i.e.* N/2). The convolution operation in Equation 1 can be interpreted as an inner-product between two vectors [3]. The combination of inner products can be represented in a matrix form and DWT coefficients are computed by multiplying the analysis transformation matrix T_{a1} with the input vector X.

$$(cA)[n] = \sum_l h[2n-l]x[l] \qquad (cD)[n] = \sum_l g[2n-l]x[l] \quad \textit{where n = 0, 1, ... , N/2} \tag{1}$$

$$\underbrace{\begin{bmatrix} h[3] & h[2] & h[1] & h[0] & 0 & 0 & 0 & 0 & \dots & 0 & h[5] & h[4] \\ g[3] & g[2] & g[1] & g[0] & 0 & 0 & 0 & 0 & \dots & 0 & g[5] & g[4] \\ h[5] & h[4] & h[3] & h[2] & h[1] & h[0] & 0 & 0 & \dots & 0 & 0 & 0 \\ g[5] & g[4] & g[3] & g[2] & g[1] & g[0] & 0 & 0 & \dots & 0 & 0 & 0 \\ 0 & 0 & h[5] & h[4] & h[3] & h[2] & h[1] & h[0] & 0 & 0 & \dots & 0 \\ 0 & 0 & g[5] & g[4] & g[3] & g[2] & g[1] & g[0] & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 0 & 0 & h[5] & h[4] & h[3] & h[2] & h[1] & h[0] \\ 0 & 0 & 0 & \dots & 0 & 0 & g[5] & g[4] & g[3] & g[2] & g[1] & g[0] \\ h[1] & h[0] & 0 & \dots & 0 & 0 & 0 & 0 & h[5] & h[4] & h[3] & h[2] \\ g[1] & g[0] & 0 & \dots & 0 & 0 & 0 & 0 & g[5] & g[4] & g[3] & g[2] \end{bmatrix}}_{T_{a1}^p} \times \underbrace{\begin{bmatrix} x[0] \\ x[1] \\ x[2] \\ x[3] \\ x[4] \\ x[5] \\ x[6] \\ \vdots \\ x[N-3] \\ x[N-2] \\ x[N-1] \end{bmatrix}}_{X}$$

Equation 2: Transformation matrix (T_{a1}^p) multiplied with finite signal (X) of length N and 6 filter coefficients (L = 6) for level-1 DWT evaluation.

$$\underbrace{\begin{bmatrix} h[3] & 0 & h[2] & 0 & h[1] & 0 & h[0] & 0 & 0 & 0 & 0 & 0 & \dots & h[5] & 0 & h[4] \\ 0 & \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \dots & 0 & 0 & 0 \\ g[3] & 0 & g[2] & 0 & g[1] & 0 & g[0] & 0 & 0 & 0 & 0 & 0 & \dots & g[5] & 0 & g[4] \\ 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \dots & 0 & 0 & 0 \\ h[5] & 0 & h[4] & 0 & h[3] & 0 & h[2] & 0 & h[1] & 0 & h[0] & 0 & 0 & \dots & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 & \dots & 0 & 0 & 0 \\ g[5] & 0 & g[4] & 0 & g[3] & 0 & g[2] & 0 & g[1] & 0 & g[0] & 0 & 0 & \dots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & h[5] & h[4] & h[3] & h[2] & h[1] & h[0] & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 & 0 & \\ 0 & 0 & 0 & 0 & \dots & g[5] & 0 & g[4] & 0 & g[3] & 0 & g[2] & 0 & g[1] & 0 & g[0] & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & \\ h[1] & 0 & h[0] & 0 & \dots & 0 & 0 & 0 & 0 & h[5] & 0 & h[4] & 0 & h[3] & 0 & h[2] & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & \\ g[1] & 0 & g[0] & 0 & \dots & 0 & 0 & 0 & 0 & g[5] & 0 & g[4] & 0 & g[3] & 0 & g[2] & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 & \end{bmatrix}}_{T_{a2}^p} \times \underbrace{\begin{bmatrix} cA_1[0] \\ cD_1[0] \\ cA_1[1] \\ cD_1[1] \\ cA_1[2] \\ cD_1[2] \\ cA_1[3] \\ \vdots \\ cA_1[N/2] \\ cD_1[N/2] \end{bmatrix}}_{C_1}$$

Equation 3: Transformation matrix (T_{a2}^p) multiplied with level-1 coefficients for 2^{nd} level DWT evaluation.

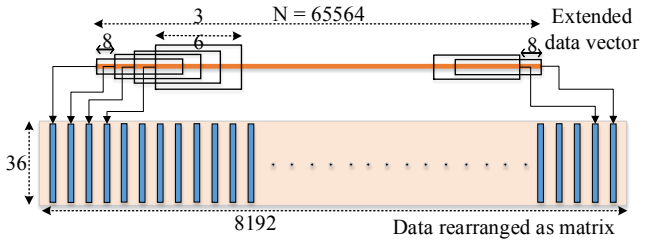
Equation 2 shows the transformation matrix T_{a1}^p for 1^{st} level of decomposition of a finite signal (X) of length N, where subscripts p and a denote periodization and analysis respectively. The process of obtaining T_{a1}^p involves shifts and boundary corrections over Equation 1 and is not shown for the sake of brevity, but discussed in greater detail in [3]. Equation 3 shows the transformation matrix, T_{a2}^p, for the 2^{nd} level of decomposition, which is obtained by performing a DWT decomposition (Equation 1) on the low-pass coefficients (cA) computed from the previous level. The transformation matrix, T_{a2}^p, then needs to be laced with zeros and ones at appropriate places (see Equation 3). Lacing with ones is required to preserve detailed coefficients through various decomposition levels. In general, DWT for the kth-level of decomposition can be obtained by multiplying directly with an appropriate transformation matrix TM (see Equation 4).

$$C = TM \cdot X, \quad where\ TM = \prod_k T_{ak}^p \tag{4}$$

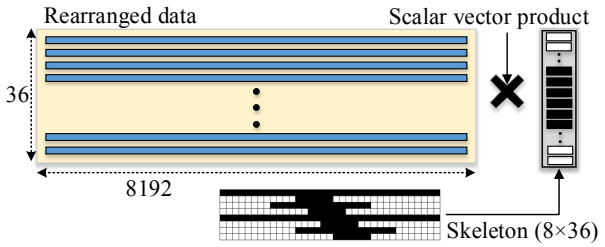
A spy graph generated using MATLAB for TM of size (256×256) for $L = 6$ and $k = 3$ is shown in Figure 1. The noticeable sparsity pattern along with the basic block structure, referred to as a *skeleton* (SK) is shown beside the transformation matrix in Fig. 1. The size of the skeleton depends on two parameters: filter size (L) and levels of decomposition (k). For a given L and k, the skeleton obtained is independent of TM size, *i.e.* N×N.

C. Acceleration Potential

DWT as a transform can be realized in three ways: (1) low and high pass filtering, (2) sequence of lifting steps or (3)



(a) Input signal vector in rearranged matrix-form



(b) Scalar-vector operation

Fig. 3: Example on transformation of original matrix-form DWT to Modified-Matrix-Form DWT (N = 2^{16}, L = 6, k = 3)

multiplication with an appropriate transformation matrix [3]. Existing works in DWT acceleration have typically employed filtering or lifting techniques [1], [4], [10]. However, the filter-based methods are recursive in nature while the lifting schemes have inherent sequential bottlenecks, which results in limited acceleration potential for these methods. Unfortunately, matrix-form DWT also poses two key challenges that limit acceleration potential: (1) the compute complexity of matrix-based DWT is $\mathcal{O}(n^2)$, as opposed to $\mathcal{O}(n)$ [7] for the other two methods, and (2) the matrices are highly sparse, which imposes a severe penalty on runtime performance due to high cache miss rates. We first address these challenges with an algorithmic optimization that transforms the difficult-to-parallellize matrix-form DWT into a SIMD-friendly form, which we refer to as the Modified-Matrix-Form (see Section III-A).

III. 1-D DWT Optimizations

A. Modified-Matrix-Form DWT

The Modified-Matrix-Form (MMF) is an algorithmic transformation that allows us to reduce the compute complexity and tackle sparsity-related challenges found in the naïve matrix-form DWT. In MMF, we first identify the skeleton in the TM matrix, which is the repeating basic block of fixed size and fixed sparsity pattern (see example in Figure 1). Figure 3a shows an example where the skeleton is of the size 8x36. We then rearrange the input signal vector into a matrix-form as shown in Figure 3a, based on the size of the skeleton. Note that the input signal vector has to be extended to account for the corner cases in the original TM matrix (see top-right and bottom-left of TM matrix in Figure 1). The final 1-D DWT result can then be obtained one row at a time by doing a row-wise scalar-vector product and accumulate as shown in Figure 3b. Since we know the sparsity pattern of the skeleton upfront, we can ensure that only the non-zero scalar rows are multiplied and accumulated for the final result. Hence, the MMF method reduces the compute complexity of the 1-D DWT kernel from $\mathcal{O}(n^2)$ to $\mathcal{O}(n)$ and also avoids wasteful multiply-by-zero computations resulting from high sparsity.

B. MMF Vectorblox MXP Implementation

The static scalar-vector product and accumulate form in MMF is a very SIMD-friendly, as now we can issue just single-cycle vector instructions to compute results in a highly parallel fashion. On the Vectorblox MXP, we can issue up to 32 fixed-point vector instructions (multiply/accumulate) in a single-cycle, depending on the size of the FPGA. We also customize the DMA transfers such that the on-chip scratchpad memory is used efficiently and memory transfer time is hidden behind compute time. The host processor orchestrates the entire DMA transfer and computation flow, ensuring that only non-zero vector computations occur on the MXP to deliver efficient hardware utilization.

IV. Experimental Setup

On the ARM 32b CPUs, we write our MMF 1-D DWT implementation in C (with OpenBLAS) and compare its performance to off-the-shelf signal processing toolbox available in Octave. We develop our FPGA-based MXP implementations on the Altera DE2, DE4, and the Xilinx Zedboard (see Section II-A), and compare its performance to the baseline Octave implementation run on the 32b CPUs in the Zedboard (ARMv7), Beaglebone Black (ARMv7), and Raspberry Pi B+ (ARMv6). All implementations use *OpenBLAS* routines (v0.2.15) and are compiled using *gcc* with *-O3* optimization flags enabled. We measure timing with *PAPI* (v5.4.3). We obtain runtimes for 8b/16b/32b data precision for all MXP implementations. For timing measurements on the MXP, we use the MXP timing API averaged across hundreds of trials while also recording power using the Energenie power meter. We also verify functionality of our vector implementation against the reference Octave solution.

V. Results

A. Software MMF

Simply rewriting the software routines to support our Modified-Matrix-Form (MMF) transformation delivers speedups ranging from 1.8×–5.6× across the three target embedded platforms (Beaglebone Black, Raspberry Pi, and Zedboard). We achieve the best speedup on the Zedboard as there is a small on-chip NEON vector engine capable of providing acceleration support to the CPU. Figure 5 shows the energy vs. throughput profile of the three embedded boards (red line) when running the optimized software MMF implementations.

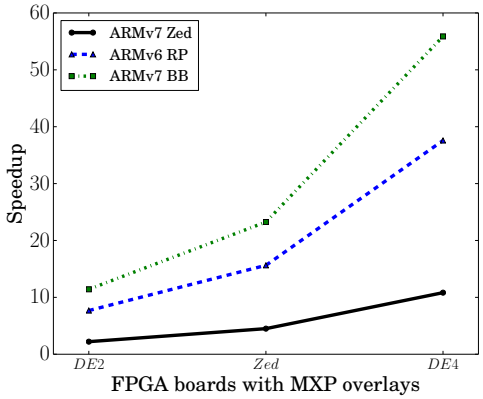


Fig. 4: Speedups from 32b MXP implementations on the DE2, Zedboard (Zed), and DE4 over ARM CPUs on the Zedboard (Zed), Raspberry Pi (RP), and Beaglebone Black (BB). L = 6, k = 3, $N = 2^{16}$.

B. MMF on the Vectorblox MXP

Figure 4 shows the 2–55× speedups obtained from our FPGA-based MMF implementations over all the optimized MMF CPU mappings on the Zedboard, Beaglebone Black, and the Raspberry Pi. We note that the DE4 offers the best acceleration potential among all the FPGA boards, which can be attributed to its larger size (more number of vector lanes), larger onchip scratchpad memory (fewer DMAs), faster clock, and higher DRAM bandwidth. Table I presents execution times on various FPGA MXP overlays for 8/16/32 bitwidths resolutions. Figure 5 shows the energy vs. throughput profile for the 3 FPGA boards (blue line), which reaffirms the dominance of the DE4 board as being the most power efficient board for doing 1-D DWT computations. Finally, our MXP implementations deliver an overall speedup of 12-103× over the baseline filter-based DWT implementations from off-the-shelf libraries available in Octave.

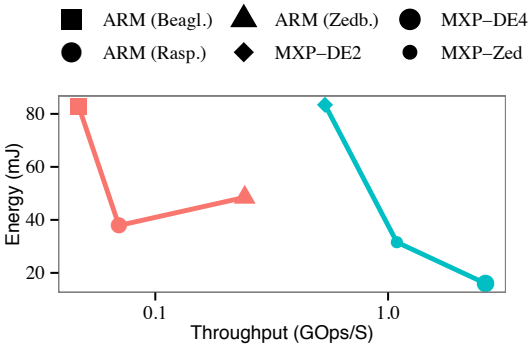


Fig. 5: Energy vs. Throughput for 1-D DWT MMF implementations on various platforms. L = 6, k = 3, $N = 2^{16}$.

VI. Related Work

Zhang [10] propose a spatial FPGA-based implementation of 1-D DWT that requires (N+J) cycles at a clock period of 8.7 ns, where N is number of input samples and J is level of decomposition. However, the computation time reported* does not include data transfer time. Using our proposed implementation, we obtain a total runtime of 0.089–0.327 ms on various FPGA boards, inclusive of data transfer time. This translates to 1.7–6.4× better performance over [10].

TABLE I: Execution time (ms) using MXP overlays on FPGA.

Resolution	8-bit	16-bit	32-bit
DE4	0.12	0.23	0.80
Zedboard	0.48	0.96	1.92
DE2	0.99	1.97	3.91

VII. Conclusion

In this paper, we improve the matrix-based multi-level Discrete Wavelet Transform (DWT) kernel with our Modified-Matrix-Form (MMF) technique to unlock inherent parallelism, which we then exploit using the Vectorblox MXP soft-vector processor. Our MMF approach takes advantage of the repeating sparse matrix skeletons in order to reduce the compute complexity from matrix-vector operations ($\mathcal{O}(n^2)$) to scalar-vector operations ($\mathcal{O}(n)$). Our final 1-D DWT MXP implementations deliver performance speedups of 12-103× over state-of-the-art signal processing libraries available in Octave, while improving energy efficiency simultaneously.

References

[1] S. Cheng, C. H. Tseng, and M. Cole. Efficient and effective vlsi architecture for a wavelet-based broadband sonar signal detection system. In *Electronics, Circuits and Systems, 2007. ICECS 2007. 14th IEEE International Conference on*, pages 593–596, Dec 2007.

[2] A. Haldane. Patience and finance. Speech, Oxford China Business Forum, Sept 2010.

[3] A. Jensen and A. la Cour-Harbo. *Ripples in Mathematics: The Discrete Wavelet Transform*. Springer-Verlag Berlin Heidelberg, Berlin, Germany, first edition, 2001.

[4] S. K. Madishetty, A. Madanayake, R. J. Cintra, and V. S. Dimitrov. Precise vlsi architecture for ai based 1-d/ 2-d daub-6 wavelet filter banks with low adder-count. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 61(7):1984–1993, July 2014.

[5] S. Maheshwari, A. Acharyya, P. E. Puddu, E. B. Mazomenos, G. Leekha, K. Maharatna, and M. Schiariti. An automated algorithm for online detection of fragmented qrs and identification of its various morphologies. *J. R. Soc. Interface*, 10(89), 2013.

[6] A. Severance and G. G. F. Lemieux. Embedded supercomputing in fpgas with the vectorblox mxp matrix processor. In *Hardware/Software Codesign and System Synthesis (CODES+ISSS), 2013 International Conference on*, pages 1–10, Sept 2013.

[7] K. K. Shukla and A. K. Tiwari. *Efficient Algorithms for Discrete Wavelet Transform: With Applications to Denoising and Fuzzy Inference Systems*, chapter Filter Banks and DWT, pages 21–36. Springer London, London, 2013.

[8] E. W. Sun and T. Meinl. A new wavelet-based denoising algorithm for high-frequency financial data mining. *Eur. J. Oper. Res.*, 217(3):589–599, 2012.

[9] Q. Wang, X. Zhang, Y. Zhang, and Q. Yi. Augem: Automatically generate high performance dense linear algebra kernels on x86 cpus. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '13, pages 25:1–25:12, New York, NY, USA, 2013. ACM.

[10] C. Zhang, C. Wang, and M. O. Ahmad. A pipeline vlsi architecture for high-speed computation of the 1-d discrete wavelet transform. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 57(10):2729–2740, Oct 2010.

*for 2^{16} samples, $\frac{8.7\times65536}{10^6} = 0.57$ ms