Science Journal of Circuits, Systems and Signal Processing
2013; 2(2) : 22-28
Published online April 2, 2013 (http://www.sciencepublishinggroup.com/j/cssp)
doi: 10.11648/j.cssp.20130202.11

Novel implementations of recursive discrete wavelet transform for real time computation with multicore systems on chip (SOC)

Mohammad Wadood Majid, Golrokh Mirzaei, Mohsin M. Jamali

Department of Electrical Engineering & Computer Science, University of Toledo, Toledo, USA

Email address:
mohammad.wadoodmajid@rockets.utoledo.edu (M. W. Majid), mohsin.jamali@utoledo.edu (M. M. Jamali)

To cite this article:
Mohammad Wadood Majid, Golrokh Mirzaei, Mohsin M. Jamali. Novel Implementation of Recursive Discrete Wavelet Transform for Real Time Computation with Multicore Systems on Chip (SOC), *Science Journal of Circuits, Systems and Signal Processing*. Vol. 2, No. 2, 2013, pp. 22-28. doi: 10.11648/j.cssp.20130202.11

Abstract: The discrete wavelet Transform (DWT) has been studied and developed in various scientific and engineering fields. Its multi-resolution and locality nature facilitates application required for progressiveness in capturing high-frequency details. However, when dealing with enormous data volume, the performance may drastically reduce. The multi-resolution sub-band encoding provided by DWT enables for higher compression ratios, and progressive transformation of signals. The widespread usage of the DWT has motivated the development of fast DWT algorithms and their tuning on all sorts of computer systems. However, this transformation comes at the expense of additional computational complexity. Achieving real-time or interactive compression/de-compression speed, therefore, requires a fast implementation of DWT that leverages emerging parallel hardware systems. The recent advancement in the consumer level multicore hardware is equipped with Single Instruction and Multiple Data (SIMD) power.In this study, Parallel Discrete Wavelet Transform has been developed with novel Adaptive Load Balancing Algorithm (ALBA). The DWT is parallelized, partitioned, mapped and scheduled on single core and Multicore. The Parallel DWT is developed in C# for single and Intel Quad cores as well as the combination of C and CUDA is implemented on GPU. This brings the significant performance on a consumer level PC without extra cost.

Keywords: Discrete Wavelet Transform, Multicore, GPUs

1. Introduction

Parallel computing is the simultaneous use of multiple computer resources to solve a computational problem by using multiple CPUs. A computationally intensive problem divided into discrete parts for concurrent computation in parallel. Algorithm needs to be partitioned into optimal number of parts and then into tasks. In this study, Novel Adaptive Load Balancing Algorithm (ALBA) is proposed to divide the problem into discrete parts. Intel Nehalem Quad Core processor [1, 2] and NVIDIA GeForce GTX 260 GPU [3-7] are two parallel processing platforms that can be exploited for fast computation of algorithms for real time applications.

Intel Nehalem Quad Core has four physical cores. Each physical core has two virtual processors, running in parallel. It has cache L1 32KB data and 32KB instruction, cache L2 256KB which is shared among virtual cores and cache L3 8MB shared between all physical cores on the chip as shown in Fig 1.

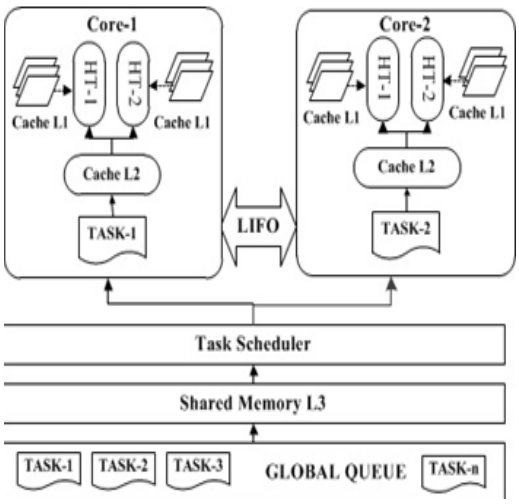

Figure 1. *Multi-core Thread Management System.*

NVidia GeForce GTX 260 [3-7] commonly known as

Graphic Processing Unit (GPU) has a multiprocessor architecture consisting of hundreds of cores as shown in Fig. 2. Each core has eight Streaming Processors (SPs) and each core is designed to operate in a Single Instruction Multiple Data (SIMD) fashion. Each GPU has following on-chip memory structure.

- One 32-bit register per core.
- A parallel data cache or shared memory
- A read-only constant memory cache that is shared by all cores allowing fast read.
- A read-only texture memory cache that is shared by all cores and allows fast reading from the texture memory.
- The local and global memory spaces are implemented as read-write regions of the device memory and are not cached.

When parallelizing any algorithm, it should be decomposed into parts and each part is further broken down into tasks. Each task is executed on a different core; each task containing threads which are executed simultaneously. To achieve the full benefits of computation power of the multicore, algorithm must have to be divided into optimal numbers of tasks. A task is a simple set of threads which run on a core. Partitioning a task is very challenging and requires optimization. There are opposing forces at work: too many partitions add overhead and too few partitions leave processors idle requiring optimal partitioning.

This study mainly focuses on general purpose multicore systems such as Intel Nehalem Quad Core processor and NVidia GeForce GTX 260 GPU in order to find the efficient strategies for DWT that can partition, map and schedule in these platforms. Different hardware and software techniques are implemented to find out the efficient strategies that can bring about the substantial performance improvement on a consumer level PC without an additional cost.

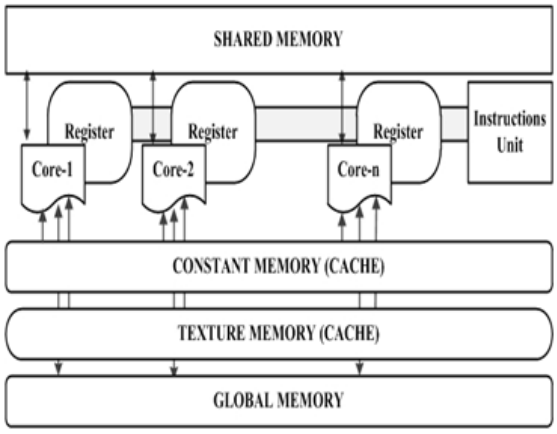

Figure 2. *GPU Architecture.*

2. Related Work

The DWT has appeared in recent years as a key part of various scientific and engineering fields also as a valuable tool for a wide variety of image processing applications. There are two different approaches called multi-machines and multicore systems on chip that are used to parallelize the DWT. There are few works that have been performed on multi-machines to parallelize the wavelet transform in the literature [8-10]. According to these works, the effective way to speed up the wavelet transformation is to utilize parallel technology. However, the parallelism is implemented in a multi-machine network environment which would be inevitably affected by the system maintenance and the bandwidth speed. Therefore, the parallel wavelet based on multi-core technology has great significance.

Significant research work has been done on its efficient implementation on multicore system on chip. In desktop computing, performance problems arise out of the differences between the memory access patterns of the main components of 2-D-DWT, especially when large input data is processed, which causes one of these components to exhibit poor data locality in the implementations of the transform. Consequently, most research is concentrated on cache-aware DWT implementations, which exploit loop transformations and specific data layouts to improve data locality [8-10]. Parallel implementations of the DWT for multiprocessor systems are usually based on the general principles of data domain decomposition; that is, the input data are statically or dynamically partitioned, and the different threads perform the same work on their share of the data [11]. Locality is even more critical in these systems, and performance tuning also focuses on memory hierarchy issues [12, 13]. In embedded systems for parallel implementation of DWT, efficiency refers not only to the execution time, but also to other design objectives such as hardware budget or power consumption. Given these additional constraints, several authors have shown that Lifting Scheme (LS) produces the most efficient designs [13]. Gnavi presented an interesting performance comparison between LS wavelet and Frequency B-Spline (FBS) wavelet on a programmable platform (a Texas Instrument DSP). They assessed the real empirical performance of both schemes in terms of execution speed in the context of the JPEG 2000, concluding that LS was always faster than its FBS counterpart. The actual LS gains heavily depend on the length of the wavelet filters and the number of LS steps, but they were lower than theoretically expected.

In this study, focus is on general-purpose multicore systems such as Intel Nehalem Quad Core processor and NVidia GPU, to find the efficient strategies for DWT to partition, map and schedule on these platforms. It involves developing new implementation strategies following to select the best programming model, in which the available data parallelism is explicitly uncovered so that it can be exploited by the hardware.

3. Discrete Wavelet Transform

3.1. One Dimension DWT

The Discrete Wavelet Transform is a relatively new mathematical technique and can be used in signal processing. In time domain signal, the independent variable is time and the dependent variable is the amplitude. Most of the information is hidden in the frequency content. By using wavelet transform, the frequency information can be obtained which is not possible by working in time-domain. The analysis of a non-stationary signal using the Fourier Transform and Short Time Fourier Transform does not give satisfactory results. DWT provides the time-frequency representation of the signal which gives multi-resolution outlook of the signal. This resolution makes the DWT technique superior to the Fast Fourier Transform (FFT) in many cases.

DWT uses filter banks and special wavelet filters for the transform and reconstruction of signals. It analyze the signal at different frequency bands with different resolutions, decomposes the signal into an approximation and detail information which contains different steps as shown in Fig. 3. The 1-D discrete wavelet transform is a fast, linear operation that operates on a data vector whose length is usually an integer power of two. The signal is decomposed into its constituent parts, each of these constituent parts are high pass and low pass filtering. A high pass and low pass filters are defined as:

$$H_k = \sum_n x[n].g[2k-n] \quad (1)$$

$$L_k = \sum_n x[n].h[2k-n] \quad (2)$$

For Example, to obtain coefficients for a n sample long signal sampled at x MHz frequency where the spanning frequency band is zero to $\pi \frac{rand}{s}$. At the first level, the signal is passed through the low pass and high pass filters.

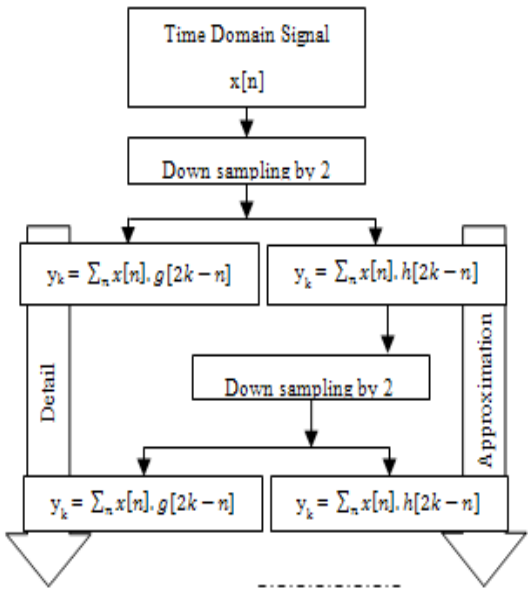

Figure 3. *Filtering with Wavelet Transform.*

The high pass filter output or the details coefficients are $\frac{n}{2}$samples long in the frequency range of $\left[\frac{x}{4}\frac{x}{2}\right]$ MHz.The low pass filter output or approximation is also $\frac{n}{2}$ samples long but spanning the frequency band of $[0 \quad \frac{x}{4}]$ MHz. At next level, only low pass filter elements are used for further filtering. The procedure continues, for low pass filter elements at each level until only 1 wavelet coefficient is computed [14-17] .

3.2. Two Dimension DWT

Wavelets allow a time series signal to be viewed in multiple resolutions. The Haar wavelet technique is one of the wavelet computation techniques that are appropriate for parallel processing on a multi-core system. It takes an average and difference of a signal with length of 2n. The haar wavelet technique takes an average and difference of a signal. All the wavelet algorithms work on time series with a power of two values 2^n. Each step of the wavelet transform produces two sets of values: a set of averages and a set of differences which are referred to as wavelet coefficients. Each step produces a set of averages and coefficients that is half the size of the input data. This continues until one average and one coefficient is calculated.

The method of averaging and differencing can also be expressed as filtering the data. Averaging corresponds to a low pass filtering. It removes high frequencies of the data; the averaging procedure tends to smooth the data. When the data is averaged, move this filter along our input data.

The differencing corresponds to high pass filtering. It removes low frequencies and responds to details. It also responds to noise, since noise usually is located in the high frequencies. When there is a difference in data, the filter is moved along our input data. The low pass and high pass filters form a filter bank referred in signal processing language.

$$LP = \frac{x_i+x_{i+1}}{\sqrt{2}}, \frac{x_{i+2}+x_{i+3}}{\sqrt{2}} \ldots \ldots \frac{x_{i+(n-1)}+x_{i+n}}{\sqrt{2}} \quad (3)$$

$$HP = \frac{x_i-x_{i+1}}{\sqrt{2}}, \frac{x_{i+2}-x_{i+3}}{\sqrt{2}} \ldots \ldots \frac{x_{i+(n-1)}-x_{i+n}}{\sqrt{2}} \quad (4)$$

Where $i = 1$ and n is the number of samples of signal

To transform the 2-D signal, first apply the 1-D Haar transform on each row. Take the resultant signal, and then apply the 1-D Haar transform on each column. This gives us the final transformed data [12].

4. Parallelization Strategies and Distribution of Data

4.1. Hardware Techniques

4.1.1. Adaptive Load Balance Algorithm

4.1.1.1. Data level Parallelism

In order to get the best performance of the DWT, the data is divided into optimal number of sub-data to create tasks. To create a task is a common divide-and-conquer technique for

using the memory hierarchy effectively. Since the cache may only be large enough to hold a small piece of data, the data has already been removed out of the cache before it is reused. The processor will thus continually be forced to access data from the main memory, decreasing the algorithm's performance. The signal is divided into tasks of smaller sub signal, and sending these tasks to the cache L1 of each core before moving on to the next task (sub-signal). Creating tasks in that fashion that each task fit into cache. This better exploits cache locality so that data in the cache can be reused before being replaced. The level of performance benefit from tasking depends on the size of the task and the cache sizes. The idea is to partition the signal into uniform tasks so that tasks are carried out task by task. Choosing the optimal task size is very important, which is being sent into L1 cache so that the cache misses are minimized. DWT algorithm is adjusted for execution in multi-processor, especially cache-memory where the communication of data between processors does not need to be planned in advance, so distinct sub-signal can be executed on different processors. With data level parallelism, the goal is to maximize the utilization of cache memory for each core. In addition to the use of parallelism, it is of great importance to use memory resources efficiently, as it is apparent that the maximum performance obtainable from current microprocessors is mostly limited by the memory access. Since the cache cannot hold a large part of data, it can be removed out of the cache before it is reused. The processor will thus be continuously forced to access data from the main memory, decreasing the algorithm's performance. Memory hierarchy optimization performance is obtained from current microprocessors with row - major layout.

Analysis of the profiler indicates that if installed RAM in a computer system is between 2GB to 4GB, then 40% of L1 cache is consumed by the Operating System (OS) of each core.

The float data type takes 4 bytes for each data point. To find the maximum number of samples that the cache L1 can hold can be calculated as:

$$Max\ (C)\ = \frac{(C_{L1} - C_{L1} * 0.4)}{D} \qquad (5)$$

Where

- C_{L1} is total L1 cache (32KB)
- $C_{L1} * 0.4$ is cache used by operating system
- D is the data type
- Max (C) is maximum data points to be stored into the available L1 cache.

Therefore the maximum data points which can be stored in L1 cache would be 4864 samples. In this way, each sub problem should not be more than 4864 samples to get the maximum performance of L1 cache of each core. Eqn. 4 and 5 are used to calculate the total bytes for all data points.

Therefore the maximum data points which can be stored in L1 cache would be 4864 samples. In this way, each sub problem should not be more than 4864 samples to get the maximum performance of L1 cache of each core. Eqn. 4 and 5 are used to calculate the total bytes for all data points.

$$\mathrm{Mem(\ Bytes)} = \ \mathrm{D} * (\mathrm{L} + 1) \qquad (6)$$

Where
D is the data type
L is the total number of data points
Mem is the total bytes for all data points
1 is considered as an extra element as OS required four bytes for creation of an instance of an object.

$$\mathrm{L_T} = \frac{\mathrm{x\ (n)}}{\mathrm{Max\ (C)}} \qquad (7)$$

Where L_T is total number of tasks.

If the number of samples is less than or equal to L_T, then the entire signal is considered as a single task. Otherwise the signal is divided into n sub-signals where each sub-signal is considered as a task as shown in the following pseudo code. In this work, it has been found that most of the time, the data is greater than L_T samples.

The creating task algorithm is as follows;

- The input signal x(n) is store as data points in to array
- Calculate maximum data points to be stored into available L1 cache as shown in Eqn. 5.
- Calculate total bytes for all data points as shown in Eqn. 6
- Calculate total number of tasks shown in Eqn. 7
- If Max(C)>= Memhen consider as a task

Otherwise

- $\mathrm{x} = \frac{\text{Eqn. 6}}{\text{Eqn. 5}}$
- $\mathrm{y} = \mathrm{x} - \mathrm{round(x)}$
- if $\mathrm{y} \leq 0.5$

$$\mathrm{n} = \mathrm{round(x)}$$

else

$$\mathrm{n} = \mathrm{x}$$

for 1 to n

$\mathrm{Task_n} = \mathrm{x}$ Where i=1,2…..n,$\mathrm{Task_{n+1}} = \left|(\mathrm{n} - \mathrm{x}) * \mathrm{round(x)}\right|$

4.1.1.2. Task Level Parallelism

Thread level parallelism is a form of parallelization on the thread level across multiple cores in the parallel computing environment. A thread is the smallest unit of processing scheduled within a task. In thread level parallelism, every virtual core is assigned a thread and a task is said to be completed when all the threads within the task are executed by the virtual cores. The processor used in this study is Intel Nehalem Quad Core processor which has four physical cores where each physical core has two virtual processors and GForce GTX 260 has hundreds of cores. Each GPU core has eight Streaming Processors (SPs) running in parallel and sharing the workload between them. After sending the

tasks to different physical cores, parallelism can be improved by using the thread level parallelism. The tasks are assigned to the physical cores and threads are assigned to the virtual processors in case of multicore and streaming processor for GPU. This reduces the task execution time considerably. In this study, since there are two virtual processors, the task execution time reduces to 40%. The low pass and high pass filter apply to the incoming signal. The received signal is divided into different number of tasks depending upon the L1 cache. The tasks are sent to task scheduler and then to different cores. Furthermore, the threads which are low pass and high pass filter of each task on each core are scheduled again in parallel to execute on virtual cores. The contribution in this part is to efficiently use the load balancing by using the thread level parallelism.

4.2. Software Techniques

4.2.1. Cache Optimization

A jagged array is an array whose elements are in the form arrays. Jagged arrays have certain advantages over traditional arrays. They can be more space efficient, because not all rows must have the same number of columns. Jagged arrays are faster to access elements due to optimization in the run time. Jagged arrays can be effectively used for enhancing the performance with high hierarchy cache memory architecture on multi-core. They can be more space efficient as array elements can of different size. In this way jagged arrays can be effectively used for enhancing the performance with high hierarchy cache memory architecture on multi-core. Traditional and jaggad arrays are used to store the signal and then the result is compared

4.2.2. Loop Fusion

Loop fusion is a transformation which takes two adjacent loops (Low pass Filter and High pass Filter) having the same iteration space traversal and combines their bodies into a single loop. Loop fusion also called loop jamming is the inverse transformation of loop distribution or loop fission which breaks a single loop into multiple loops with the same iteration space. Loop fusion is allowed as long as there are no output dependencies exits. Fusing two loops results in a single loop, which contains more instructions in its body and therefore offers increased instruction level parallelism. Furthermore, only one loop is executed, thus reducing the total loop overhead by approximately a factor of two. Loop fusion also improves data locality. Assume that two consecutive loops perform global sweeps through an array as in the code shown in pseudo code bellow.

4.2.3. Row Major

The cache hit performance is improved by the concept of row-major organization. The element of the matrix is accessed column wise, and therefore is not in sequential order in memory. This is due to the fact that matrices are stored in memory as row-major order. As a result, the DWT algorithm is bandwidth limited and displays poor performance and low efficiency because of time spent in loading data rather than computing when performing the 1-D Haar Transform on each column. In order to optimize the performance, transpose the data matrix and applied the 1-D Haar transform on each row as shown in Fig. 4.

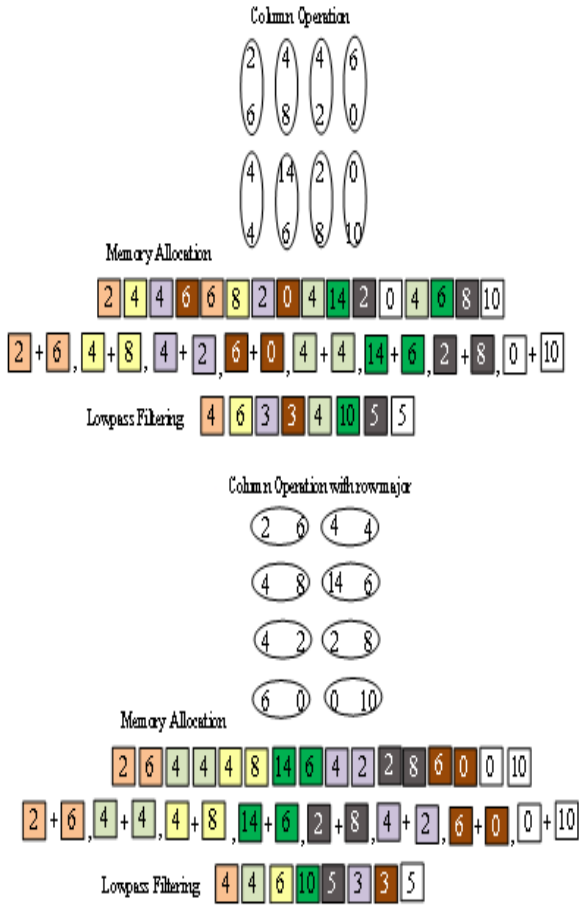

Figure 4. *Load and compute from cache with Row-Major technique.*

5. Experimental Setup and Conclusion

DWT algorithmhas been parallelized and implemented on Intel Nehalem Quad Cores and NVidia GeForce GTX 260. Moreover, the algorithm is also implemented on the single core for comparison purposes. The algorithm is developed with visual studio.net 2010 and uses C#for Intel Nehalem Quad Core. It uses combination of C and CUDA for its implementation on NVidia GeForce GTX 260 GPU.

DWT has been simulated using traditional parallelism and proposed adaptive load balancing algorithm on single and multicore systems. Results are shown in Table 1. It can be seen that adaptive load balancing with equal number of tasks provides minimum computation time. The parallel computation is further improved with different software techniques. Table 2 shows the improvement using different software techniques. The overall comparison of single and multicore (Intel Nehalem Quad Core and NVidia GeForce GTX 260 GPU) with different software techniques are shown in Fig. 5.

Table 1. *Computation Time of Load Adaptive and Non-Load Adaptive Algorithm in milliseconds.*

HARDWARE TECHNIQUES												
	Traditional Parallelization						Adaptive Load Balancing					
	Un-Equal Tasks			Equal Tasks			Un-Equal Tasks			Equal Tasks		
	Data Level	Thread Level	Total	Data Level	Thread Level	Total	Data Level	Thread Level	Total	Data Level	Thread Level	Total
GTX 260	3.215	2.291	5.506	2.873	2.421	5.294	2.053	1.772	3.825	1.964	1.154	3.118
Nehalem	3.363	2.493	5.857	3.040	2.306	5.346	2.128	1.812	3.941	2.085	1.667	3.752
Single core			18.127			18.127			14.864			14.864
Improvement	GTX 260 (3.292) Nehalem (3.090)			GTX 260 (3.424) Nehalem (3.390)			GTX 260 (3.886) Nehalem (3.771)			GTX 260 (4.767) Nehalem (3.961)		

Table 2. *Improvement with Software Techniques.*

SOFTWARE TECHNIQUES					
Nehalem			GTX 260		
Cache Optimization	Loop Fusion	Row Major	Cache Optimization	Loop Fusion	Row Major
2.744	2.250	1.350	2.650	2.094	1.193

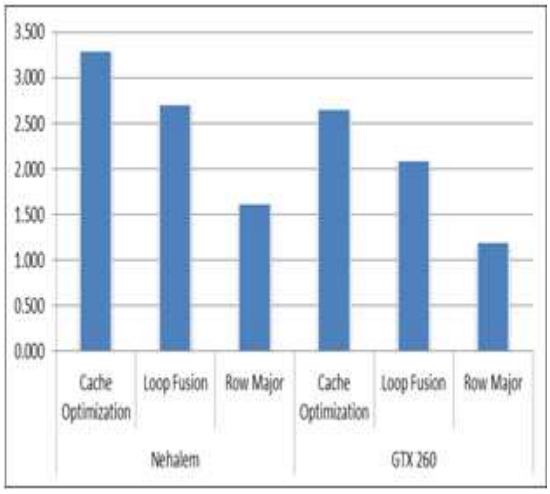

Figure 5. *Improvement with Software Techniques.*

Simulation results show that the proposed algorithm (ALBA) outperformed Traditional Parallelization algorithm. Thus the DWT algorithm implemented using hardware technique on Nehalem Quad Cores and GeForce GTX 260 consumed very less time compared to that performed by single core. Improvement was also observed in case of equal and unequal data points in tasks using traditional parallelization and proposed ALBA. The GTX 260 showed an improvement of 4.7 times and Nehalem showed an improvement of 3.9 times compared to the single core for equal data point in the tasks with proposed ALBA. It is concluded that the tasks with equal data points gave better performance as compared to the tasks with un-equal data points.

The computation time was further improved with different software techniques. However, the improvement was observed on both Intel Nehalem Quad Core and NVidia GeForce GTX 260 GPU. It is perceived that Cached Optimization technique provided further 12% improvement for Nehalem and 14.87% for GTX 270. Furthermore, Loop Fusion and Row Major improved the computation time 17.97% for Nehalem, 21% for GTX 270 and 40.13% for Nehalem and 43.25% for GTX respectively. Parallelism has been exploited for efficient use of available memory hierarchy. It is also concluded that the NVidia GeForce GTX 270 GPU provides better computational performance than the Intel Nehalem Quad. This result is in line with our own intuition that the GPU performs better than the general purpose Intel Nehalem Quad Core.

Reference

[1] Intel Corporation, "*Intel Nehalem* 2010" Available from: http://www.intel.com/technology/architecture-silicon/next-gen.

[2] Singhal, R., "*Inside Intel ® Nest Generation Nehalem Microarchitecture*." 2009.

[3] NVidiaCorporation, *"NVIDIA CUDA Compute Unified Device Architecture, Programming Guide, Version 4.0*" 2012.

[4] NVidiaCorporation, "*NVIDIA GeForce GPU Architecture Overview, Technical Brief"*, November 2010.

[5] NVidiaCorporation,"*GeForce GTX 260* 2011", Available from:http://www.nvidia.com/object/product_geforce_gtx_260_us.html.

[6] NVidiaCorporation,"*CUDA Software Development Kit 5.0*", Available from:https://developer.nvidia.com/cuda-downloads.2012;

[7] NVidiaCorporation, "*Get Started - Parallel Computing*",

2012.

[8] JeiH. I., "*Parallel Image Compression and Decompression based on Wavelet Principle on Multi-core Cluster*", Inner Mongolian University, 2008.

[9] University of Tsinghus, "*The writing group of the multi-core series of the textbooks: Multi-core programming*", The Press of the Tsinghua University, Beijing2007.

[10] Ling Y., "Parallel wavelet analysis based on multi-core cluster.",2005.

[11] Chatterjee, S.B. Cache-Efficient Wavelet Lifting in JPEG 2000. in IEEE Conference on Multimedia and Expo. 2002.

[12] A. Shahbahrami, B.J., and S. Vassiliadis, "Improving the Memory Behavior of Vertical Filtering in the Discrete Wavelet Transform", Third Conf. Computing Frontiers (CF '06)2006. p. 253-260.

[13] P. Meerwald, R.N., and A. Uh.,"*Cache Issues with JPEG2000 Wavelet Lifting*", SPIE Electronic Imaging, Visual Comm. and Image Processing. 2002.

[14] Daubechies, I., "*The wavelet transforms time-frequency localization and signal analysis*", IEEE Transitions on Information Theory, 2002.

[15] ImanElyasi, S.Z., "*Elimination Noise by Adaptive Wavelet Threshold",* World Academy of Science, Engineering and Technology 2000.

[16] Kaiser, G., *Friendly Guide To Wavelets*1994: Birkhauser.

[17] Polikar, R., "The Engineer's Ultimate Guide to Wavelet Analysis", Iowa State University 2000.