MULTICORE IMPLENENTATION OF WAVELET TRANSFORM USING SCILAB

Aneeta S Antony[1], Sneha Sara Jacob[2], Anusha Danday[3],Kavitha S S[4]

[1] Department of E &C, NIE, Mysuru, India
[2] Department of E &C, NIE, Mysuru, India
[3] Department of E &C, NIE, Mysuru, India
[4] Department of E &C, NIE, Mysuru, India

Abstract

In this paper, we discuss parallel implementation of Wavelet Transform on multicore platforms. We compare performance of the wavelet transform implementation on different number of cores. We report the work carried out using Haar Wavelet. We have selected SciLab as our platform for experimentation. SciLab on Linux supports multiprocessing as function. Linux also supports enabling and disabling of cores. We use this feature to ensure the same environment for different number of cores. We compare the performance of the SciLab's inbuilt DWT function and our own function.

Keywords: *Multicore, Parallel Algorithm, Wavelet Transform, Haar Wavelet*

--***--

1. INTRODUCTION

Faster computations are always in demand for many applications. The simplest method is to increase the clock frequency of the system. Even-though the clock frequency of the CPU is directly related to the performance, it increases power consumption thereby increasing the temperature of the device. Moreover, clock frequency has reached the plateau and it is impracticable to increase clock speed exorbitantly. Alternate methods are the need of the hour to increase the computing efficiency, at the same time keeping heat of the devices under control.

A good beginning point to increase the performance is to exploit the multicore feature available in the present-day computers. Multicore CPUs involve more than one CPU core fabricated into a single chip. This allows parallel processing by allowing us to execute multiple tasks simultaneously. Obviously, as more tasks are executed simultaneously, computational efficiency would increase. Since there are multiple cores, the instantaneous power consumed may be more. But, since the execution takes less time, the watt-hour (power × time) consumed would be lesser.

Image transforms, in general,are used in many applications – including analysis and compression. Wavelets, in particular, have been used in applications like compression, segmentation and multiresolution analysis. Wavelet, like other transforms is compute intensive. Even with the availability of fast and efficient algorithms, computation time involved might not be suitable for real-time applications. But the good thing is the implementation of wavelet can be parallelized. With the advent of multicore platforms, the wavelet implementation can shared across different cores and computation time can be reduced. In this work, we explore the possibility of parallelizing wavelet algorithm on multicore platforms. We consider matrices of different sizes and run it on different number of cores. We study the computation time in each case. We compare our code with the standard code available with SciLab[1].

Scilab is a popular open source equivalent of Matlab [2]. Scilab is used extensively in signal processing, communication, control among other applications. Scilab has many toolboxes. It has a toolbox of the multicore applications too. As of now, the multicore support is available only on Linux and not on Windows[3]. Thus, we have used Linux (Ubuntu 14.0) for our experimentation reported in this work.

2. WAVELETS

Wavelets, like Fourier transform, are a class of orthogonal transforms. Fourier transform has a limitation that we get either time information (time domain) or frequency information (in transform domain). We can get both. That is, in time domain frequency information is lost and in transform (frequency) domain time information is lost. There are some applications, like seismology, where we need both. Even-though we can use Windowed Fourier transform (WFT)for such applications, fixed window size becomes bottleneck for WFT. We need a transform that has a variable window size. Wavelets satisfies these requirements.

Wavelet transform pair are defined by

$$W(a,b)=\int_{-\infty}^{\infty} f(t)\frac{1}{\sqrt{|a|}}\psi\left(\frac{t}{b-a}\right) \tag{1}$$

and

$$f(t)=\frac{1}{C_{\psi}}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} W_f(a,b)\,\psi_{ab}(t)\frac{1}{a^2}\,da\,db \tag{2}$$

Where,

$$C_\Psi = \int_{-\infty}^{\infty} \frac{|\Psi(\Omega)^2|}{\Omega} d\Omega \qquad (3)$$

Wavelets need to satisfy following properties

1. $$\int_{-\infty}^{\infty} \psi(t)\, dt = 0 \qquad (4)$$

2. $$\int_{-\infty}^{\infty} |\psi(t)^2|\, dt < \infty \qquad (5)$$

The $\psi(t)$ is called *prototype* or *mother wavelet,* and can be represented as

$$\Psi_{ab} = \frac{1}{\sqrt{(a)}} \psi\left(\frac{t-b}{a}\right) \Rightarrow \Psi_{ab}(\Omega) \qquad (6)$$

As per the Eqn.(4), system should have wavelike property and as per Eqn.(5), the wave should die out. As per the Eqn.(6), the parameter *b* will *translate* the wavelet along the time axis and parameter *a* will *scale* the wavelet

We observer that *W* in Eqn. (1) is a function of two variables, which corresponds to time and frequency. Thus we can say that wavelets resolve the signal into its time as well as frequency components. Moreover, unlike Fourier, where we have complex exponential as the basis, we do not have a unique basis for the wavelets. Instead the basis is represented by $\psi(t)$, and as long as it satisfies equations (4)-(6). This implies that there are many different basis for wavelets[4]-[6].

In this work, we have used simplest of the wavelet, namely the Haar Wavelet, defined by I Daubechies[5] as.

$$\psi(x) = \left\{ \begin{matrix} 1, & 0 \le x \le 1/2 \\ -1, & 1/2 \le x \le 1 \\ 0 & otherwise \end{matrix} \right\}$$

Then the discrete wavelet of a sequence *f* is obtained as inner product of *f* with $\psi(x)$ as $\langle f, \psi \rangle$

3. IMPLEMENTATION

As noted in the previous section, wavelet can be implemented as an inner product. The summation of inner product can be parallelised[9] - each part of summation can be done on different cores[7]. We too, exploit the same technique, and use ScliLab's parallel processing toolbox.

SciLab allows parallel execution of the code as a subroutine. It can be called as parallel_run(). The parameters to be passed include the array to be used in parallel, size of the array and the name of the function that is to be executed. The function parallel_run(); will execute the commands within the function, concurrently. User does not have the flexibility to balance the load or to schedule threads for different cores. All the scheduling is done by the SciLab tool box. We use the function directly and the wavelet calculation code is written within the subroutine. We implement wavelet as the inner-product or 1-D wavelet as the function. This is called from parallel_run();. We have not used any scheduling algorithms or load balancing techniques, but have relied on the inbuilt functions.

SciLab has the function etime() and getdate() to estimate the time of execution. We use it at the beginning and end of our function call (parallel_run();), the difference in time is noted as the time of execution. This will ensure time of execution is evaluated for the core of the algorithm (wavelet transform calculation), and does not consider other bookkeeping operations like reading the file. We run the program 10 times and the average of the time is taken as the time of execution for our program. To ensure sufficient computation time, we take the wavelet transform of a 2-D signal (image). We send each row data(i,:); at a time for the DWT function, followed by each column as data(:,i). This will ensure enough computation time for the study of performance.

We use the same code, without *any* modification to run on different number of cores. Linux provides command to enable or disable a core. As a superuser (root), executing the command

```
echo 0 >/sys/devices/system/cpu/cpu1/online
```

will disable core-1. Changing cpu1 to cpu2, we can disable core-2 and so on. Similarly echoing '1' (instead of 0) will enable the specific core. Enabling and disabling the cores thus, and not modifying the code for different cores will ensure that the environment to execute the program remains the same and hence, the measured time is accurate.

To test the performance, we also use the built-in dwt() function, with the same Haar Wavelet. Performance of both these are studied.

4. RESULTS and CONCLUSIONS

We studied the performance of the parallel algorithm for three different sizes of images, namely 128x128, 256x256 and 512x512. For each image, we run the algorithm on single, dual, three and four cores, by disabling the appropriate number of cores. In each case, time of execution is calculated as an average of 10 runs. This will ensure a more realistic timings. The Results of this experimentation are depicted in the graphs Fig.1-3 The experiment is repeated for the built-in DWT function also, and the results are plotted in Fig. 4-6. In all the graphs, X-axis is the number

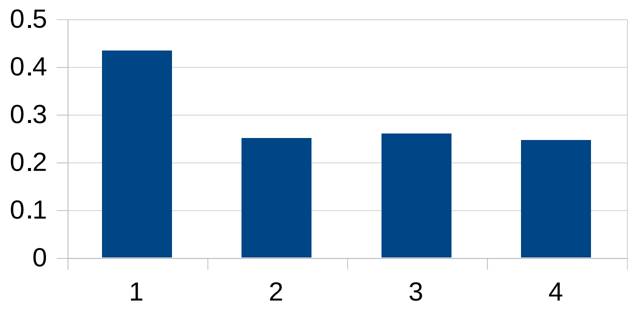

Figure-1 Time of execution for 128x128 data

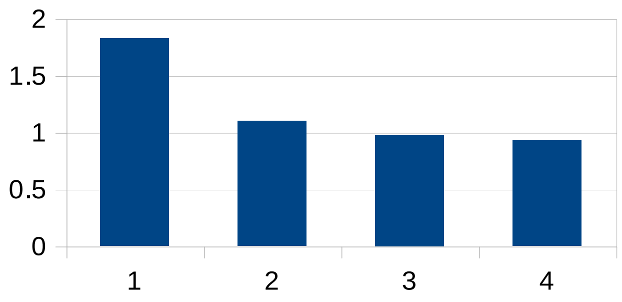

Figure-2 Time of execution for 256x256 data

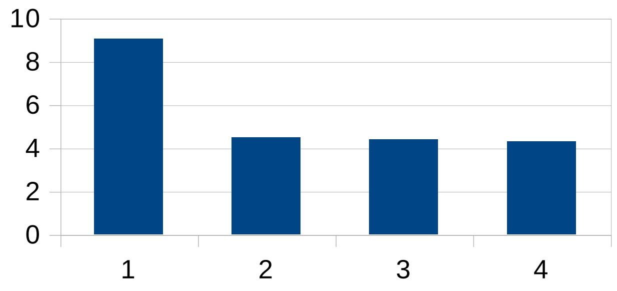

Figure-3 Time of execution for 512x512 data

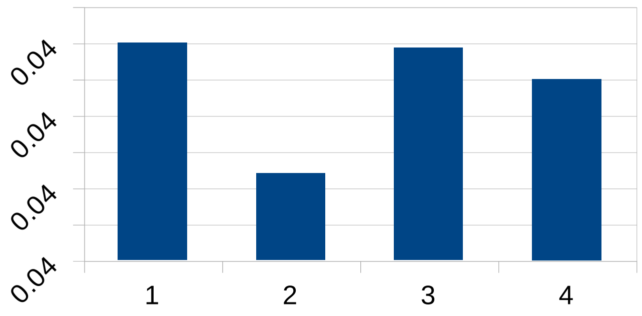

Figure-4 Time of execution for 128x128 data using built-in function

We see that speedup is not significant between 2-4 cores, but it becomes significant when compared with single core for different sizes of the inputs. However, there is no significant improvement for different cores when we use built-in function. Time taken by built-in function is significantly less compared to our code. This may be due to efficient usage of arrays and memory in the built-in functions. Results of parallization is not as good as the one reported by Swathi *et.al.*,[7]. The reason is the use of tools Swathi uses OpenMP[8], which gives a flexibility for thread creation, load balancing and sharing. However, our results show that the DWT can be implement on multicore, using SciLab also. Also built-in functions does not provide significant improvement with multicore

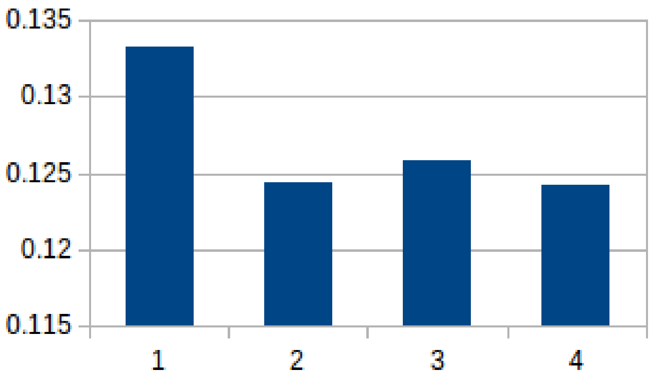

Figure-5 Time of execution for 256x256 data using built-in function

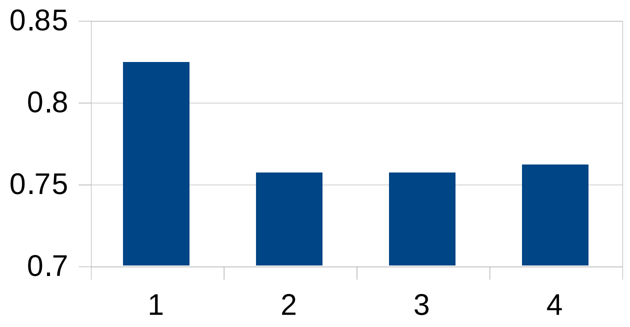

Figure-6 Time of execution for 512x512 data using built-in function

ACKNOWLEDGEMENT

We acknowledge our sincere thanks to Dr Narasimha Kaulgud for his relentless support and encouragement in this endeavour.

REFERENCES

[1] www.scilab.org
[2] www.matlab.com
[3] http://bugzilla.scilab.org/show_bug.cgi?id=8361
[4] Oliver Rioul and Martin Vetterli, "Wavelets and Signal processing", *IEEE Signal Processing Magazine, 10 (14-38), 1991*
[5] I Daubechies *"Ten lecture on Wavelets",* SIAM, Philadelphia, 1992
[6] Ali N Akansu, Wouter Serdijn and Iwan Selesnik, "Emerging applications of wavelets: A review", *Elseiver Physical communication, 3, (1-18), 2010*
[7] Swathi N, Spriha Deshpande and Narasimha Kaulgud, Performance measure of multicore systems for orthogonal transforms, *Journal of ISoI, V45,N3,* pp:60-63, March 2015
[8] www.openmp.org
[9] Michael J Quinn, "*Parallel Programming",* McGraw-Hill, New Delhi, 2003

BIOGRAPHIES

Aneeta S Antony has completed her M.tech in Signal Processing from VTU Belegavi and currently works as Assistant Professor in NIE, Mysuru.

Anusha Danday has completed Mtech in Communication Systems from Anna University,Chennai. and currently works as Assistant Professor in NIE, Mysuru.

Sneha Sara Jacob has completed her Mtech in Electronics Design & technology from NIT Calicut and currently works as Assistant Professor in NIE, Mysuru.

Kavitha S S has completed Mtech in VLSI Design and Embedded system from VTU Belagavi and currently works as Assistant Professor in NIE,Mysuru.