Multithreading Image Processing in Single-core and Multi-core CPU using Java

Alda Kika
Department of Informatics
Faculty of Natural Sciences, University of Tirana
Tirana, Albania

Silvana Greca
Department of Informatics
Faculty of Natural Sciences, University of Tirana
Tirana, Albania

Abstract—**Multithreading has been shown to be a powerful approach for boosting a system performance. One of the good examples of applications that benefits from multithreading is image processing. Image processing requires many resources and processing run time because the calculations are often done on a matrix of pixels. The programming language Java supports the multithreading programming as part of the language itself instead of treating threads through the operating system. In this paper we explore the performance of Java image processing applications designed with multithreading approach. In order to test how the multithreading influences on the performance of the program, we tested several image processing algorithms implemented with Java language using the sequential one thread and multithreading approach on single and multi-core CPU. The experiments were based not only on different platforms and algorithms that differ from each other from the level of complexity, but also on changing the sizes of the images and the number of threads when multithreading approach is applied. Performance is increased on single core and multiple core CPU in different ways in relation with image size, complexity of the algorithm and the platform.**

Keywords—multithreading; image processing; multi-core; Java

I. Introduction

In recent years, Java language has become a popular choice for development of multithreaded applications due to the language multithreading support. Multithreaded programming allows simple identification of the sections of code that can be executed concurrently to exploit parallelism. The programs must be able to exploit this kind of parallelism in order to get performance gains in computing. There are two different ways to implement concurrent applications. The first approach is through the creation of processes, with all the communication made through messages, which are responsible for keeping all the necessary information for the programs, including register content and memory space [1]. The second approach uses thread, which are also known as light processes [2]. A thread is a point of execution within a process and they represent a key concurrency model supported by modern computers, programming languages, and operating systems. The threads exchange information only through shared memory and can be up to 20 times faster in their creation time when compared to processes [3].

A multi-threaded process has multiple points of concurrent execution within the process [4]. The use of multiple threads allows an application to distribute long running tasks so that they can be executed in parallel. This is also possible with significant advances of multi-core systems. Today, multi-core processors are widely deployed in both server and desktop systems. The performance of multi-threaded applications could be improved on multi-core based systems because the workload of threads could be dispatched to cores, which work in parallel [5].

One of the good examples of applications that benefit from multithreading in a multi-core processor is image processing. Image processing "refers to the manipulation and analysis of pictorial information" [6]. The main idea of parallel image processing is to divide the problem into simple tasks and solve them concurrently, in such a way the total time can be divided between the total tasks (in the best case) [7]. The general idea behind image processing involves examining image pixels and manipulating them. Image processing can be a time consuming task based on the matrix structure of the image leading this process towards a multithreading algorithm.

Many authors have addressed the multithreading topic in java on multi-core systems. The java as a suitable programming language for parallel software and the power of multi-core processing is studied by Peter Bertels and Dirk Stroobandtin [8]. In [9], authors studied and implemented parallel multi-threaded implementations of two popular clustering algorithms: k-means and mean-shift. The experimental results show that good parallel implementations of those algorithms are able to achieve nearly linear speedups on multicore processors. In [10] Mahmood has implemented the imaging filter on multi-core processors for win32 platform. He shows that for large images, the parallel implementation approaches Amdahl's ideal curve. In [4] authors observed that multithreading leads to tune the application performance considerably. The performance and scalability issues of multithreaded Java applications on multicore systems are studied in [11]. This interesting topic has solicited many scientific works so far.

In our work we studied how the algorithm performance changed when multithreading approach is applied on different single core and multi-core platforms. We also studied how complexity of the algorithm and the image size of the images influence the performance.

In order to exploit how multithreading can improve the performance of the algorithm, we have done some experiments with several image processing algorithms such as brightness, contrast and steganography executed on either single-core CPU or multi-core CPU using single-thread and multithreading approaches in Java. The algorithms that we have taken in consideration differ from each other from the level of complexity. As the image size increases, the number of calculations increases too. For this reason we have experimented with different sizes of images. The difference in the algorithms and image size would influence on a further change of the level of complexity.

We measured the algorithms processing time on different platforms and image sizes that we took in consideration.

In this paper, we have provided insight into Java multithreading approach performance on single and multi-core CPU platforms and hardware suggesting the combination complexity and image size that gives the best performance.

Section I gives a brief introduction and related work. The rest of the paper is organized as follows. Section II presents an overview of multithreading programming and the algorithms implemented with this approach. Section III outlines the experimental setup including test programs and specifications of the experimental platforms. Section IV presents the results and section V gives a brief analysis of the results. The paper is concluded in Section VI and VII by presenting the conclusions and further work.

II. Multithreading Programming

First, in creating a multithreaded algorithm, there are three basic considerations. The first step is to identify the parallelism. This may mean simply decomposing the problem domain of a conventional algorithm into several sections. The second step is to control the access to shared data items. The third step is to optimize the algorithm [4].

The application running in the operating system is either single-threaded or multi-threaded. Single threaded applications require one thread to run on the CPU. Whereas a multithreaded program contains two or more parts that can run concurrently.

We have implemented three image processing algorithms using Java. Java has become a leading programming language soon after its release and it is an emerging option for High Performance Computing [12]. Java supports multithreading and we can construct single-thread as well as multi-thread application with it. A multi-threaded program in java has many entry and exit points, which are run concurrently with the main() method. The three constructed algorithms have been implemented using single thread approach and multithreading approach. In the multithreading approach the shared memory in which the threads operate is the matrix of the image pixels. We have used the Java packages to grab the pixel matrix of the image that has to be processed. Then different threads manipulate different parts of the matrix depending on the algorithm. The work task and the part of the matrix that each thread has to manipulate are determined by the main thread. The time that is necessary to manipulate all the matrix either by a single thread or by all the threads is registered.

III. Experimental Setup

A. Testing programs

Initially, three image processing algorithms have been chosen to test the multithreading approach on different single-core and multiple-core platforms. The first and the second algorithms change brightness and contrast of the chosen image and display the changing image. Brightness and contrast are very important features of an image. Brightness refers to the overall lightness or darkness, whereas contrast is the difference in brightness between objects or regions. For the algorithm of the brightness we have used the formula :

Arithmetic mean model = (r + g + b) / 3   (1)

After finding the mean from the formula 1 we use a value input from the user to change the brightness of the image following these steps:

- Subtract the overall mean from every color value of every image pixel.
- Add the mean value multiplied by the brightness value to every color value.

In the Figure 1 a view from the brightness application is presented.

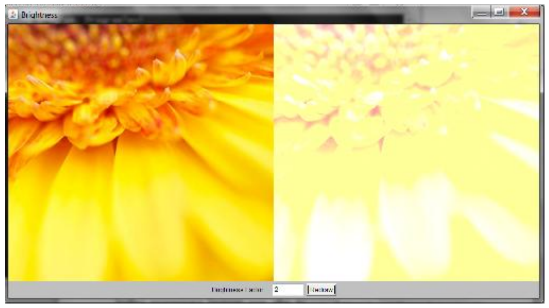

Fig. 1. A screenshot from the brightness application

The contrast algorithm will multiply every color value by a scale factor determined by the user. It has a little less calculation than the brightness algorithm.

The third algorithm is from the field of steganography. Steganography is the art of hiding the fact that communication is taking place by hiding information in other information [13]. In our implementation of steganography algorithm we hide a message in an image in such a way that the modification of the image is imperceptible. Since people perceive red and green color brighter than the blue color, the image would be darker in the red and green pixels and brighter in the blue pixels. In order to hide the original message in the image we have constructed an algorithm that change only the last bit of the blue color of the image pixels using the message and a private key as well [14]. The complexity of this algorithm is greater than the other two. A view from the application that implement the third algorithm designed with multithreading approach is given in the Fig. 2.

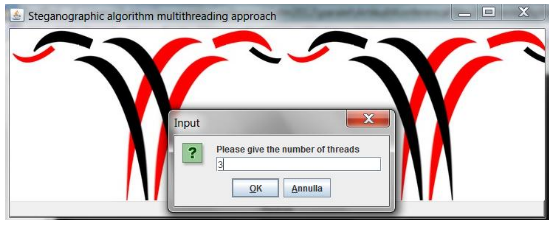
Steganographic algorithm multithreading approach
Input
Please give the number of threads
3
OK
Annulla

Fig. 2. A screenshot from the steganography algorithm application

B. *Experimental platforms*

Since the first multicore processor was released over a decade ago, today's processors implement up to a dozen cores and this number is expected to increase because of Moore's law [15]. Multi-core CPU-s support advanced capabilities, such as multithreading and parallel processing. The advantage of a multi-core processor is increased speed. They are widely used across many application domains including embedded, network, graphics etc. The experiments were conducted on the Windows Operating Systems (Windows XP and Windows 7), single-core and multiple-core CPU-s platforms.

In Table 1 is given a brief description of the four used experimental platforms.

TABLE I. THE DESCRIPTION OF EXPERIMENTAL PLATFORMS

Platform	Processor	Operating System	Number of Cores
Platform1	AMD Athlon™, 1.11GHz	Windows XP	1
Platform2	*Intel Pentium 4*, 3.2 Ghz	Windows XP	1
Platform3	Intel Core^{TM}2 Duo Processors, 2.2 GHz	Windows 7	2
Platform4	Intel Core™ i5-4570R Processor, 2.7 GHZ	Windows 7	4

We used the first and second platform to test the algorithms according to the single thread and multithreading approach in single core, and we used the third and fourth platform to test the algorithms in the multi-core CPU.

C. *Input Data*

Not only the platforms but also the size of the input images influence on the execution time of the algorithms. We perform the experiments using three image sizes: 371*281, 500*500 and 1024*768. We refer to the dimension of the images in the experiments as S(Small), M(Medium) and L(Large). The results of these experiments are presented in the next section.

IV. EXPERIMENTAL RESULTS

Every algorithm was repeated 10 times and the average of all the results was recorded.

From the implementation of the tests of the single thread approach algorithms on the single-core and multiple-core platforms we recorded the results shown in Table 2:

TABLE II. SINGLE-THREAD APPROACH RESULTS

Platform Image Size	Algorithms performance(ms)		
	Contrast	*Brightness*	*Steganography*
1(S)	30	40	190
1(M)	50	40	351
1(L)	110	100	872
2(S)	29	29	62
2(M)	29	29	172
2(L)	47	62	296
3(S)	15	20	50
3(M)	16	21	70
3(L)	16	31	172
4(S)	10	14	32
4(M)	11	16	47
4(L)	18	26	109

We redesigned the same algorithms with multithreading approach. Then we varied the number of threads from 1 to 10 and then the multiples of 5 thus 15, 20, 25, 30 …100.

The results from the experiments showed that on the platform with single core (platform 1 and platform2) the processing time of the algorithms decreases in a considerably rates when multithreading approach is applied. However when the number of threads increases over 10 threads the processing time does not decrease, but remains almost constant with a tendency to increase more than in the case when the single thread approach is applied. Generally when the number of threads increased from 1 to 10 the average processing time has the tendency to remain constant and when the number of threads increases more than 10 the processing time increases and it becomes bigger than the executing time in single thread approach.

Table 3 shows the result for the first platform. The result refers to the average executing processing time for the 10 first threads for the small, medium and large images.

TABLE III. AVERAGE RESULTS OF THE FIRST 10 THREADS FOR THE FIRST PLATFORM

Image Size	Algorithms		
	Contrast	*Brightness*	*Steganography*
Small	18	20	151.3
Medium	41	27.1	295.5
Large	109	69	824.3

The tables 4,5,6 display the average executing processing time for the 10 first threads in case of small, medium and large images for the second, third and fourth platform.

TABLE IV. AVERAGE RESULTS OF THE FIRST 10 THREADS FOR THE SECOND PLATFORM

Image Size	Algorithms		
	Contrast	*Brightness*	*Steganography*
Small	15	15	48.4
Medium	20.6	15.5	100
Large	46.5	29.9	271.8

TABLE V. AVERAGE RESULTS OF THE FIRST 10 THREADS FOR THE THIRD PLATFORM

Image Size	Algorithms		
	Contrast	*Brightness*	*Steganography*
Small	10	11	31.3
Medium	18.8	15.4	59.6
Large	51.3	46.9	162.3

TABLE VI. AVERAGE RESULTS OF THE FIRST 10 THREADS FOR THE FOURTH PLATFORM

Image Size	Algorithms		
	Contrast	*Brightness*	*Steganography*
Small	13.5	12.3	19.7
Medium	17.8	15.7	43.7
Large	31.2	23.2	101

V. PERFORMANCE ANALYSIS

As we expected when single thread approach of the algorithms is applied, the processing time decrements if we increase the speed and the number of the cores from one platform to the other (table 2). When multithreading is applied, the CPU without cores improves the processing time with a greater rate than when the number and cores of the processor increase in the case when the size of the image is small or medium and the complexity of algorithm is not big.

In the first and second platform where the number of cores is 1, the increase rate of processing time is more than 48.27 % when the size is small and the algorithms are brightness and contrast. In the other cases this rate varies from 15.9 to 44.82. The results do not fall in this interval when the size of the image is large and the algorithm is contrast (first platform 0.9 and the second 2) and when the algorithm is steganography (first platform 5.5 and the second 8.1).

The third and the fourth platform give different results. The result of performance increase rates can be even negative which means that the processing time of the single thread approaches are better than multithreading approach. The result for the third and fourth platform is displayed in the tables 7 and 8.

TABLE VII. PERFORMANCE INCREASE RATE FOR THE THIRD PLATFORM

Image Size	**Performance increase rates(%)**		
	Contrast	***Brightness***	***Steganography***
Small	33,3	45	44,7
Medium	-18,7	28,5	14,3
Large	-218,7	-51,6	5,8

TABLE VIII. PERFORMANCE INCREASE RATE FOR THE FOURTH PLATFORM

Image Size	**Performance increase rates(%)**		
	Contrast	***Brightness***	***Steganography***
Small	-40	14,2	37,5
Medium	-63,6	0	6,3
Large	-72,2	11,5	7,3

The third platform gives the best result when the size of the image is small for all the algorithms as shown in the table 7. When the size of the image is large it gives the worst result. When the complexity of the algorithm is bigger the processing time tends to be less.

In the fourth platform the image size seems to have no influence as much as in third platform in the processing time. But the complexity of the algorithm influence the processing time by reducing it.

From both platforms with multi core CPU the combination small image size and complex algorithm gives the best results, improving the processing time when multithreading occurs.

VI. CONCLUSIONS

Java language is very suitable to develop image processing applications due to its features and the free packages that it offers for this purpose. New open source java image processing packages is often added [16].

It is very important to understand how to improve the performance of this kind of applications using managed languages like Java. Through some experiments with the image processing algorithms the impact that multithreading approach has on performance is analyzed in single-core and multi-core platforms.

The results showed that the multithreading approach improves the performance processing time of algorithms either in single-core or multi-core CPU platforms but this improvement is different.

In single core the best results is given by the combination of small image size and less complex algorithm whereas in multi-core CPU the combination of small image size and more complex algorithm improves the performance. Multithreading programming can improve the performance on multi-core CPU when complex image processing algorithms is applied.

VII. FUTURE WORK

Others studies are needed to be done to understand the causes of these behaviors. Future work includes developing a C# environment with image processing algorithms in order to compare it with Java applications in different platforms. The influence that parallel implementation of these multithreading approach algorithms will be another interesting task for the future. We assume that we can improve the processing time as it was shown for Wu-Lee Steganography Algorithm [17].

REFERENCES

[1] J. F. CORSINI, L. G. FREITAS, and F. D. ROSSI, “Comparando Processos entre Unix e Windows". Revista INFOCAMP. March, 2006.

[2] SCHEFFER, R. “Uma visao Geralsobre Threads". Revista Campo Digit@l. Volume 2.N_umero 1.Paginas: 7-12, 2007.

[3] B. BARNEY, “POSIX Threads Programing". Lawrence Livermore National Laboratory. 2011. <https://computing.llnl.gov/tutorials/pthreads>.

[4] M.Shanthi, A.Anthony Irudhayaraj. Multithreading, “An Efficient Technique for Enhancing Application Performance”, International Journal of Recent Trends in Engineering, Vol 2, No. 4, November 2009, pp. 165-167.

[5] Kuo-Yi Chen, Fuh-Gwo Chen, “The Smart Energy Management of Multithreaded Java Applications on Multi-Core Processors”, International Journal of Networked and Distributed Computing, Vol. 1, No. 1, January 2013, pp.53-60.

[6] Gregory A. Baxes, Digital Image Processing: Principles and Applications. John Wiley and Sons, Inc, New York, NY, 1994.

[7] S. Saxena, N. Sharma, Sh. Sharma, “Image Processing Tasks using Parallel Computing in Multi core Architecture and its Applications in Medical Imaging”, International Journal of Advanced Research in Computer and Communication Engineering Vol. 2, Issue 4, April 2013

[8] Peter Bertels, Dirk Stroobandt, “Java and the Power of Multi-Core Processing”. The Second International Conference on Complex, Intelligent and Software Intensive Systems, CISIS 2008, pp. 627 – 631

[9] Honggang Wang, Jide Zhao, Hongguang Li, Jianguo Wang. “Parallel Clustering Algorithms for Image Processing on MultiCore CPUs”. International Conference on Computer Science and Software Engineering, 2008.

[10] Faran Mahmood, “Parallel Implementation of Imaging Filters on Multi-Core Processors for Win32 Platform”. Proceedings of the 4th International Conference on Open-Source Systems and Technologies, ICOSST, 2011.

[11] Kuo-Yi Chen,, J. Morris Chang, Ting-Wei Hou. :Multithreading in Java: Performance and Scalability on Multicore Systems:, IEEE Transactions on Computers 2011 (vol. 60 no. 11), pp. 1521-1534

[12] G.L. Taboada, J. Touriño, R. Doallo, “Java for high performance computing: assessment of current research and practice:, in: Proc. 7th Intl. Conference on the Principles and Practice of Programming in Java, PPPJ’09, Calgary, Alberta, Canada, 2009, pp. 30–39.

[13] Obaida Mohammad Awad Al-Hazaimeh, “Hiding Data in Images Using New Random Technique”, IJCSI International Journal of Computer Science Issues, Vol. 9, Issue 4, No 2, July 2012, pp 49-53.

[14] S. Greca, A. Kika, O. Qirici, ”A new steganography method for hiding the message in the image”, Buletin of the Natural Sciences Faculty, No. 14, 2012, pp. 211-220.

[15] G. E. Moore. “Readings in computer architecture. Chapter Cramming more components onto integrated circuits”, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2000, pp 56–59..

[16] Tobias Pietzsch, Stephan Preibisch, Pavel Tomanc and Stephan Saalfeld, “ImgLib2 generic image processing in Java”, BioInformatics, Vol. 28 no. 22 2012, pp. 3009–3011.

[17] S. Greca, E.Martiri, “Wu-Lee Steganographic Algorithm on Binary Images Processed in Parallel”, International Journal of Video & Image Processing and Network Security IJVIPNS-IJENS Vol: 12 No: 03,2012, pp 1-4.