Available online at www.sciencedirect.com

ScienceDirect

AASRI Procedia 5 (2013) 140 – 146

AASRI Procedia

www.elsevier.com/locate/procedia

2013 AASRI Conferenceon Parallel and Distributed Computing Systems

Analyzing Performance of the Parallel-based Fractal Image Compression Problem on Multicore Systems

Roberto de Quadros Gomes[a], Vladimir Guerreiro[a], Rodrigo da Rosa Righi[a*], Luiz Gonzaga da Silveira Jr.[a], JinyoungYang[b]

[a]*Applied Computing Graduate Program –PIPCA - UNISINOS–Unisinos Avenue 950, São Leopoldo, RS, Brazil*
[b]*Korean Advanced Institute of Science and Technology –KAIST – 291 Daehak-ro, Yuseong-gu, Daejeon 305-701,South Korea*

Abstract

Both the size and the resolution of images always were key topics in the graphical computing area. Especially, they become more and more relevant in the big data era. We can observe that often a huge amount of data is exchanged by medium/low bandwidth networks or yet, they need to be stored on devices with limited space of memory. In this context, the present paper shows the use of the Fractal method for image compression. It is a lossy method known by providing higher indexes of file reduction through a highly time consuming phase. In this way, we developed a model of parallel application for exploiting the power of multiprocessor architectures in order to get the Fractal method advantages in a feasible time. The evaluation was done with different-sized images as well as by using two types of machines, one with two and another with four cores. The results demonstrated that both the speedup and efficiency are highly dependent of the number of cores. They emphasized that a large number of threads does not always represent a better performance.

Keywords: Parallel Programing, Fractal Coding, Multicore architectures, Parallel modeling, Threads, Performance analysis

[*]Corresponding author: Tel: +55 51 35911122; fax: +55 51 35911133
Email address: rrrighi@unisinos.br

1. Introduction

Image compression is a special topic inside the computer graphics areas which becomes more and more important on Big-Data era [1–3]. Its main objective consists in reducing the irrelevance and redundancy of the image data in order to be able to store or transmit data in an efficient way. In this context, a technique called Fractal Image Compression (FIC) appears as one of most efficient solutions for reducing the size of files [4], [5]. FIC method is characterized by higher rates of compression, but itsencoding phase is time-consuming [9]. On the other hand, the decoding one occurs quickly enabling users to download compressed images from Web servers and visualize them in their hosts in a feasible time, for instance.

Aiming to reduce of the time on the encoding phase of FIC method, it is possible to design this phase as a parallel program to take advantage of multiple cores present on common machine architectures [6, 7]. Multi-core processors are widely used across many application domains.It is important to emphasize the improvement in performance gained by the use of a multicore processor depends on the employed software algorithms, as well as their implementation.This paper describes the FIC technique and presents a new threads-based approach for modeling the problem as a parallel application. Besides the model itself, we developed a prototype that was evaluated by using different-sized images and different multiprocessing machines. Especially, we obtained the results by using both a dual and quad-core machines.

2. Characterizing the FIC Problem and Parallel Program Modeling

FIC applies transformations, which approximate smaller parts of the image by larger ones. The smaller parts are called ranges and the larger ones domains. All ranges together form the image. A complete domain-poll of an image of size *txt*with square domains of size $d \times d$ consists of $(t - d + 1)^2$ domains. This paper presents a parallel modeling for multicore systems by using threads in order to explore such an architecture.

As already presented by Stapleton's work [2, 11], FIC has a natural parallelism. Each comparison between ranges and domains is independent. The main idea is to start more than one FIC process at the same time. Each one is responsible for a subset of the original image. Figure 1 illustrates the proposed model of parallelism when 4 threads are employed. The first step of the model consists in splitting the original image in *n* equal subsets of ranges. The value of *n* indicates the number of threads that will be employed on compression coding. Each thread works with whole set of domains and must test all element of this set against each range element received previously.The final step concerns the appending of all blocks for generating the final compresses image. This task is done after a synchronization point that waits for the ending of all threads.

Concerning the model explained in Figure 1, we developed an application written in C Language that uses the routines of the Pthreadslibrary for threads facilities. The main program takes a time for calculating the FIC problem (*t2 –t1*) in the following manner: time *t1* before launching the threads and time *t2* after the synchronization directive. The environment is composed byboth dual-core (2.93 GHz) and quad-core (2.53 GHz) multiprocessor machines. Since the parallel application will be evaluated with 2, 4, 8, 16 and 32 threads, our idea consists in observing the speedup and the efficiency on different configuration machines.

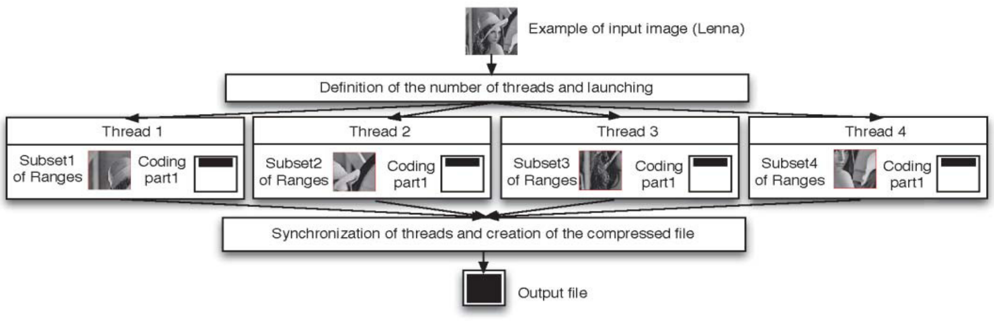


Fig. 1. Parallel Model for the FIC problem with 4 threads

$$PSNR[dB] = 10x\log\left(\frac{255^2}{MSE(f, f`)}\right) \tag{1}$$

According with [13] the error or distortion between original image *f* and a decoded image *f˝* is measured by the peak signal-to-noise ratio (PSNR). Equation 1 defines this value. MSE indicates the mean square error. PSNR metric is used in several works to qualify the reconstruction of a lossy compression method.

3. Results and Discussions

We used two input images for performing our evaluation. The first has 256x256px (Lenna) and second with 512x512px (Coliseum). We intend to analyze both the speedup and the parallel efficiency when varying the number of threads and the dimension of the ranges. Each experiment ran 30 times and we got both the mean value and the standard deviation. Table 1 presents the obtained PSNR when varying the number of ranges. The 2x2-sized range obtained the best results due to its better entropy when compared to larger ranges.

Table 1.Analysing the Obtained PSNR (measured in decibels) for both evaluated images when varying the dimension of the ranges

Input Image \ Range	2x2	4x4	8x8	16x16	32x32
Lenna	35	31	26	22	19
Coliseum	38	27	22	19	18

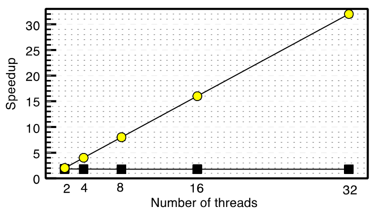
Tables 2 and 3 present the evaluation of both input images when using a dualcore machine. As expected, the best results appear when testing 2 or 4 threads. For example, 6.57 seconds were obtained when testing only one thread with a range dimension equal to 4. On the other hand, the execution with 32 threads presented the highest execution time when comparingexecutions of multiple threads. This behavior is explained by the overhead of mutex, synchronization and thread management primitives. Figure 2 illustrates the speedup (sequential_time/parallel_time) and the parallel efficiency (Speedup/processors) for the tests with 2x2 range.Our application presents a poor speedup because the number of threads is greater than the number of execution cores. This statement becomes clear in the efficiency graph. Considering that we have only 2 physical cores, the execution with two threads presented the highest efficiency (92%). The execution with 4 or more threads expresses the problem of concurrence, since they compete for the use of the cores.

Table 2.Evaluation with a dualcore computer and a 256x256-sized image (Lenna) - Time in seconds and SD (standard deviation)

Range Parameter	Sequencial		ParallelEvaluation									
			2 threads		4 threads		5 threads		16 threads		32 threads	
	Time	SD	Time	SD	Time	SD	Time	SD	Time	SD	Time	SD
2x2	45.379	0.657	24.899	0.530	25352	0.417	25583	0.385	25490	0.366	25548	0.416
4x4	6.576	0.119	3.523	0.045	3459	0.037	3520	0.037	3540	0.042	3562	0.041
8x8	1.249	0.013	0.834	0.028	0.779	0.014	0.790	0.016	0.796	0.014	0.825	0.022
16x16	0.268	0.005	0.219	0.009	0.213	0.006	0.219	0.008	0.226	0.005	0.245	0.006
32x32	0.058	0.004	0.059	0.004	0.062	0.003	0.069	0.006	0.081	0.008	0.104	0.006

Table 3.Evaluation with a dualcore computer and a 512x512-sized image (Coliseum) - Time in seconds and SD (standard deviation)

Range Parameter	Sequencial		ParallelEvaluation									
			2 threads		4 threads		5 threads		16 threads		32 threads	
	Time	SD	Time	SD	Time	SD	Time	SD	Time	SD	Time	SD
2x2	695.097	0.825	349.028	1.012	336.326	0.621	338.427	0.652	344.849	0.710	350.961	0.766
4x4	118.911	1.035	59.910	2.150	58.306	0.760	58.250	0.703	59.042	0.829	59.792	0.958
8x8	21.523	0.214	10.848	0.138	11.028	0.078	11.076	0.094	11.234	0.377	11.309	0.141
16x16	4.563	0.035	2.476	0.027	2.442	0.015	2.492	0.010	2.525	0.011	2.593	0.019
32x32	1.023	0.006	0.661	0.010	0.623	0.009	0.637	0.006	0.665	0.009	0.725	0.012



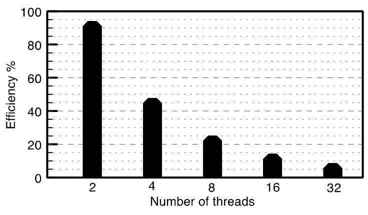


Fig. 2 Speedup (left) and parallel efficiency (right) when using a 256x256 sized image, 2x2 range and dualcore machine.

Figure 3 depicts the speedup evaluation results of the Coliseum image over a dualcore machine. In this way, the execution with 2 threads reaches indexes up to 1.97 of speedup, which is considered a good measure since the ideal speedup for this configuration is 2. Besides this analysis, it is possible to observe other two behaviors in the graph of Figure 3.Firstly, the larger the dimension of the ranges, the lower the captured speedup. For example, the execution with a range of 32x32 presents a lowest computation grain per each thread. Secondly, we can observe an execution pattern among the threads.

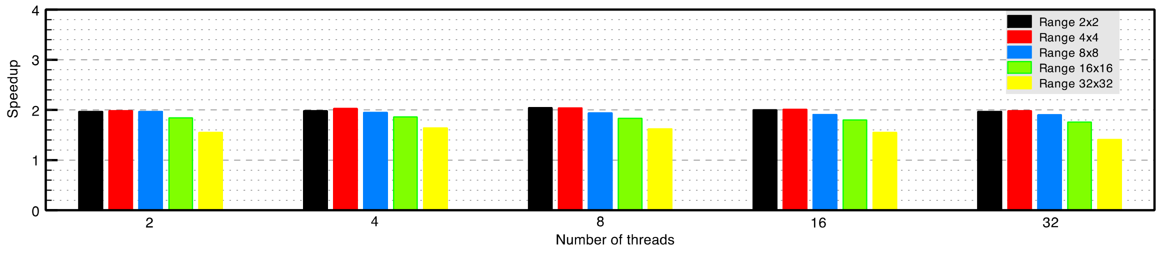


Fig. 3 Speedup with a 512x512-sized image and a dual-core machine

Both Tables 4 and 5 present the results when changing the infrastructure to a quadcore machine. Figure 4 shows the efficiency results obtained with the compression of the Coliseum image by using 2 and 4 threads. As expected, both dualcore and quadcore machines present higher indexes of efficiency when treating 2 threads. The main results of this figure appear on the right side.While the dualcore computer achieves an average efficiency of 46.3% for 4 threads, a quadcore configuration presents an index of 68.2% for the same metric. The quadcore performance must be carefully analyzed since the larger the number of cores the larger memory access contention.Future tests concern the employment of processor-level profiles in order to verify possible shared components that act as bottleneck in the system performance.

Table 4.Evaluation with a quad-core computer and a 256x256-sized image (Lenna) - Time in seconds and SD (standard deviation)

Range Parameter	Sequencial		ParallelEvaluation									
			2 threads		4 threads		5 threads		16 threads		32 threads	
	Time	SD	Time	SD	Time	SD	Time	SD	Time	SD	Time	SD
2x2	39.782	0.825	23.804	1.012	20.735	0.621	20.493	0.652	20.518	0.710	20.717	0.766
4x4	6.241	1.035	3.490	2.150	2.700	0.760	2.704	0.703	2.726	0.829	2.763	0.958
8x8	1.476	0.214	0.768	0.138	0.612	0.078	0.557	0.094	0.562	0.377	0.597	0.141
16x16	0.345	0.001	0.191	00.002	0.159	0.005	0.157	0.003	0.163	0.003	0.213	0.006
32x32	0.077	0.001	0.044	0.002	0.040	0.001	0.048	0.004	0.078	0.005	0.112	0.003

Table 5.Evaluation with a quad-core computer and a 512x512-sized image (Coliseum) - Time in seconds and SD (standard deviation)

Range Parameter	Sequencial		Parallel									
			2 threads		4 threads		5 threads		16 threads		32 threads	
	Time	SD	Time	SD	Time	SD	Time	SD	Time	SD	Time	SD
2x2	627.877	0.473	329.060	1.821	281.961	2.622	273.816	1.525	271.517	2.456	272.759	0.186
4x4	126.557	0.408	64.227	0.106	43.187	0.083	43.158	0.468	43.340	0.044	43.657	0.150
8x8	24.872	0.087	13.404	0.034	8.513	0.051	8.527	0.012	8.604	0.034	8.751	0.022
16x16	5.869	0.006	2.977	0.002	1.946	0.018	1.967	0.015	1.993	0.010	2.103	0.024
32x32	1.381	0.004	0.711	0.005	0.523	0.036	0.512	0.006	0.536	0.001	0.613	0.013

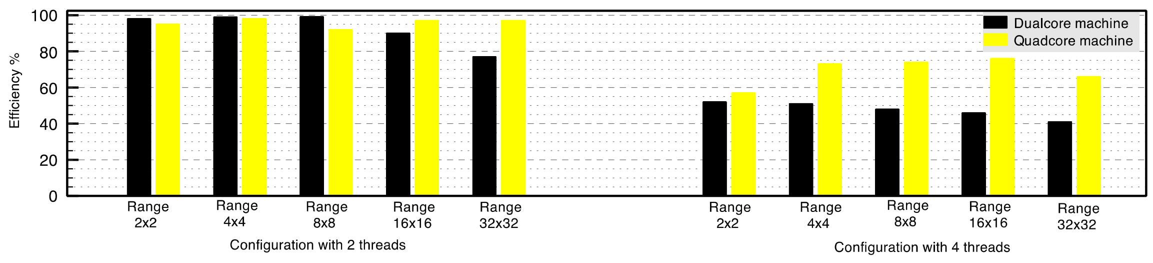


Fig. 4 Comparison of parallel efficiency when using 2 and 4 threads on both dualcore and quadcoretestbeds

4. Related Work

A variety of works explore the thematic of the FIC compression problem [8, 10, 11, 12, 13]. Lin [13] presented a fast search strategy using a classifier of ranges based on particle swarm optimization and Dihedral transformation. Iano, Silva and Cruz [10] utilize a hybrid solution with a lowpass filter (wavelet transform) in order to obtain a better quality of the compressed image. In addition, Wang's [14] work uses inter range correlation to avoid unnecessary searches. Stapleton et al [11] showed similar solution with nCUBE2 family. Hunfnagel[12] uses fractal algorithms on SIMD (*Single Instruction Multiple Data*) environments. Therefore, We found a lack on measuring the power of multicore machines for calculating the FIC problem. This opportunity of work was explored in this paper.

5. Conclusion

This paper presented a parallel modeling of the so-called Fractal Image Compression (FIC) problem. Our model is based on a copy of the entire *D* (Domain) set for each thread. Furthermore, each one receives from the main program its own subset of ranges, which represents a subpart of the input image. The results showed gains up to 48% when comparing both multiple and single-thread scenarios. The tests revealed a different treatment when using either a dualcore or quadcore machine. The parallel efficiency is abruptly reduced when passing from 2 to 4 threads with a dualcore machine (from 97.1% to 46.3%). This is not verified on a quadcore configuration, where the average efficiency remains in 68.2%.

Acknowledgements

This work is partially supported by the following Brazilian Agencies: CAPES, CNPq (Science Without Borders project) and FAPERGS (CloudMig Project).

References

[1] M. Sundaresanand E. Devika, "Image compressionusing h.264 anddeflatealgorithm," in PatternRecognition, InformaticsandMedical Engineering (PRIME), 2012 Int. Conference on, pp. 242 –245, 2012.
[2] M. Stammand K. Liu, "Anti-forensics of digital image compression," Information Forensicsand Security,

IEEE Transactions on, vol. 6, no. 3, pp. 1050 –1065, 2011.
[3] S. Chen, X. Cheng, and J. Xu, “Research on image compressionalgorithmbased on rectanglesegmen-tationand storage withsparse matrix,” in Fuzzy Systems and Knowledge Discovery (FSKD), 2012 9th International Conference on, pp. 1904 –1908, 2012.
[4] J.-H. Jeng, C.-C. Tseng, and J.-G. Hsieh, “Study on huber fractal image compression,” Image Processing, IEEE Transactions on, vol. 18, no. 5, pp. 995 –1003, 2009.
[5] L. George and E. Al-Hilo, “Fractal color image compressionbyadaptive zero-meanmethod,” in Computer Technology and Development, 2009. ICCTD ’09, vol. 1, pp. 525 –529, 2009.
[6] B. Hopson, K. Benkrid, D. Keymeulen, and N. Aranki, “Real-time ccsdslosslessadaptivehyperspectral image compression on parallel gpgpuamp; multicore processor systems,” in Adaptive Hardware and Systems (AHS), 2012 NASA/ESA Conference on, pp. 107 –114, 2012.
[7]A.Wakatani, “Implementation of fractal image codingforgpgpu systems andits power-awareevaluation,” in Systems Conference (SysCon), 2012 IEEE International, pp. 1 –5, 2012.
[8] H. Cao and X. qianGu, “Implement research of fractal image encodingbased on openmpparallelization model,” in Electric Information and Control Engineering, 2011 Int. Conference on, pp. 462 –465, 2011.
[9] Y. Fisher, Fractal Image Compression: Theoryand Application. Springer London, Limited, 2012.
[10] Y. Iano, F. da Silva, and A. Cruz, “A fastandefficienthybridfractalwavelet image coder,” Image Processing, IEEE Transactions on, vol. 15, no. 1, pp. 98–105, 2006.
[11] W. Stapleton, W. Mahmoud, and D. Jackson, “A parallel implementation of a fractal image compressionalgorithm,”, 28th SoutheasternSymp. on System Theory, pp. 332–336, 1996.
[12]C. Hufnagland A. Uhl, “Algorithmsfor fractal image compression on massively parallel simd arrays.” Real-Time Imaging, vol. 6, no. 4, pp. 267–281, 2000.
[13] C. W.-L. LIN, Yih-Lon, “Fast Search Strategiesfor Fractal Image Compression” n. 28, pp. 17–30, 2012.
[14] C.-C. Wan and C.-H. Hsieh, “An efficient fractal image-codingmethodusinginterblockcorrelation search,” Circuits and Systems for Video Technology, IEEE, vol. 11, no. 2, pp. 257 –261, 2001.