Scalable Lossless High Definition Image Coding on Multicore Platforms

Shih-Wei Liao[2], Shih-Hao Hung[2], Chia-Heng Tu[1], Jen-Hao Chen[2]

[1]Graduate Institute of Networking and Multimedia
[2]Department of Computer Science and Information Engineering
National Taiwan University
Taipei, Taiwan 106
{liao, hungsh, d94944008, r94125}@csie.ntu.edu.tw

Abstract. With the advent of multicores in all processor segments including mobile, embedded, desktop and server ones, we are in the new era of multiplying computing power via scaling the number of cores. The multicore approach is more versatile and programmable than the ASIC approach. For instance, the same multicore product can be adapted to the ever-improving potpourri image processing standards. Developing ASIC modules for each standard would pose million-dollar start-up cost and time-to-market disadvantage. However, the multicore approach is a two-edge sword: Unleashing its multiplying power presents significant programming challenges. The harmony between the multiplying power and programming productivity is the holy grail in this field. This paper addresses the challenge in the Digital Cinema domain. This paper presents an oblivious parallelization paradigm in both compressing and decompressing images via JPEG2000 on multicore platforms with maximum productivity. This approach dramatically reduces compression and decompression time in performing JPEG2000 lossless encoding and decoding algorithms on high definition images in almost real time without any extra hardware acceleration. By boosting parallelism coverage, the high resolution images could be compressed and decompressed in near real time: 15 images decoded/encoded per second. To the best of our knowledge, we are the first to propose a software-based coding solution using commodity multicores to achieve near real-time performance result for JPEG2000. This cost-effective approach could be applied to digital cinema on devices with multicores.

Keywords: JPEG2000, Lossless, Multicore SoC, Parallelization, Digital Cinema, Embedded System, Image Compress, Image Decompress.

1 Introduction

Full HD is the standard for the next generation digital TVs. The 2K and 4K digital cinema are also on their way. This kind of huge data flow poses difficult challenges on both storage space and transmission bandwidth. Therefore, compression is a necessity. Furthermore, while consumer product users can tolerate minor image

quality distortion, professional production and military and medical customers cannot. As a result, their compression schemes must be lossless. Several new standards have been proposed including H.264/AVC for motion picture, JPEG2000 for still image, and AAC for audio. These standards have better compression ratio compared to previous standards, and also support lossless coding scheme. Among these standards, JPEG2000 has been adopted by DCI (digital cinema initiative) for future distribution of movies. Therefore, JPEG2000 is taken as case study in this paper. JPEG2000, compared to previous lossless compression standards such as JPEG-LS, has a average of 30% advantage in compression ratio [1]. But it requires about four times computing power in both encoding and decoding [2]. Hardware (ASIC) and software (multicore) based approaches have been proposed to addressing these complexities. Our paper advocates the latter approach.

1.1 Motivation for the Multicore Approach

The adoption of multicores in all product segments including mobile, embedded, desktop and server is an inevitable trend. The market has witnessed many multicore products such as Sun's UltraSPARC T1 and Intel's Xeon 5100/5355/7100 in the high-end market and ARM's MPCore in the embedded market. More computing power can be obtained for a single task if we can utilize more cores to achieve that. Another incentive for utilizing more cores is that energy per core is more efficient and price per core is cheaper when we move to more cores. For instance, Xeon 5300 series has twice the number of cores as Xeon 5100, yet both the price point and the power consumption are about the same.

In contrast to the ASIC approach, multicore solutions present time-to-market advantages. For instance, multicore solutions do not impose additional hardware development time for each fixed function. In addition, the multicore approach is more flexible in adapting to new standards and configurations. The same hardware resources can be shared among different functions. Thus, better resource utilization across different functions can be achieved. On the other hand, ASIC solutions pose million-dollar start-up cost and time-to-market disadvantage on each standard they would support. Each independent IP can only process one kind of application, resulting poor resource utilization and busy bus traffic among IPs. Furthermore, the energy consumption may increase.

Multicore solutions, as we will show in this paper, are powerful enough to support mainstream applications in the Digital Cinema domain, and the performance scales with the increased number of cores. Thus, the multicore approach brings about a promising platform to support the increasing compute demand and operating modes in embedded environment such as mobile phone and digital TV/Cinema.

1.2 Studying Digital Cinema Solutions

To address the high compute and storage demands of Digital Cinema standard, people have proposed hardware (ASIC) and software (multicore) based solutions.

Both approaches use parallelization to speed up processing. The JPEG2000 standard can be roughly divided into three levels of parallelization paradigms.

The highest level is the oblivious parallelization paradigm. In such scheme the image is divided into several segments and coded independently. The middle level is the tile level parallelization paradigm, in which image is segmented into tiles, and coded altogether into single JPEG2000 code stream. The lowest level is the embedded block coding level parallelization paradigm, which partition each sub-band result from DWT (Discrete Wavelet Transforms) into separate code blocks.

Among hardware solutions, Shirai et al. [3] adopt the oblivious parallelization paradigm to build a system based on the JPEG2000 acceleration boards. There are in total four acceleration boards in the system, connected via PCI-X bus. Each acceleration board is responsible for a quarter of the image; each color component is further delegated to four JPEG2000 ASICs. Using these systems, real-time 4K SHD (4096x2160) quality international conference has been held between Kyoto, Japan and San Diego, USA. Literature [4] shows that embedded code blocking is the most time consuming part of JPEG2000 process. There are several hardware implementations that focus on this portion [5, 6, 7].

Software solutions such as Jasper [8, 9], JJ2000 [10], and Kakadu [11], mostly run in a single process. Threading techniques, used in [11] and [12], are deployed to parallelize embedded code blocking process as hardware solutions do. In this paper, we evaluate Jasper [8] and Kakadu [11] performance with high resolution, real-world pictures. As the result in Figure 1 shows, Kakadu [11] outperforms Jasper in single process with multithreading performance. Furthermore, both coders fail to encode/decode in real-time, 15 frames per second. Based on this result, we hereafter choose Kakadu [11] as our baseline software-based JPEG2000 encoder/decoder. Note that the hardware specification will be given in Section 4.

1.3 Scaling State-of-the-Art Software Solution

Kakadu uses threading to parallelize time-consuming portions. In this paper we investigate Kakadu's scalability in detail. As the figure in Section 2.2 shows, Kakadu fails to scale beyond three times speedup even on an 8-core machine with 32 *CoolThreads*. This poor scalability makes Kakadu unsuitable for meeting real-time, high-resolution performance requirements. This motivates us to propose a software implementation of oblivious parallelization paradigm on multicore architecture.

Our multicore implementation has three distinct advantages: First, it has time-to-market productivity advantage. It takes less than three man-months to deploy. Second, performance scales with increased number of processor cores. Our result shows that the multicore approach can meet real-time Full HD constraints. Even 2K SHD can be processed in real time. Third, the high parallelism coverage of our implementation removes the limitation induced by the 20% sequential portion in the Kakadu encode/decode process.

In summary, the paper demonstrates the feasibility of multicore solutions in the Digital TV/Cinema domains which was traditionally dominated by proprietary ASICs. In addition, we are the first to propose the software-based oblivious

parallelization paradigm on commodity multicores and provide comprehensive evaluation of such scheme for Digital Cinema.

The paper is organized as follows. Section 1 describes the motivation, related works, and overview of the results. Section 2 illustrates the basics of JPEG2000 and the traditional approaches to parallelize JPEG2000. We present our approach in Section 3 and its experimental results in Section 4. Section 5 concludes the paper.

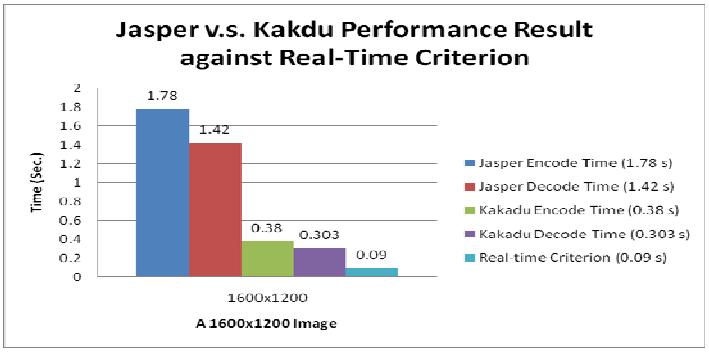

Fig. 1. Performance result of Jasper and Kakadu against Real-Time Criterion, 15 frames per second, on a 1600x1200 image.

2 Traditional Parallelization Approach in JPEG2000

We present the JPEG2000 flow in Section 2.1 and the traditional approaches to parallelize JPEG2000 in Section 2.2.

2.1 JPEG2000 Flow

JPEG2000, as a state-of-the-art image coding system, is a wavelet-based image compression and decompression standard created by Joint Photographic Experts Group committee. Since detailed descriptions of JPEG2000 features, internal algorithms, and functionality set are duly presented in the literature [13], here we only outline the core system and possible applications of JPEG2000 for the completeness.

The fundamental building blocks of JPEG2000 are as follows: pre-processing (component transform), discrete wavelet transform, quantization, tier-1 coding (entropy coding or arithmetic coding), and tier-2 coding (output code-stream creation). These building blocks are mainly processing logical structures of components, tiles, sub-bands, and code-blocks. Now, let us describe the encoding flow of JPEG2000 as an example to understand the relationships among those keywords above. First, the image data is decomposed into components (for example, Red, Green and Blue) and then, the components are further partitioned into (rectangular or non-overlapping) equal-sized tiles. Note that each tile could be coded independently later. Second, the tiles are broken into coefficients for sub-bands by

wavelet transform. Each intra-tile's frequency characteristics are kept in the coefficients. Third, wavelet transform is followed by quantization, which is responsible for quantizing and partitioning the coefficients into code-blocks. Quantization is also the stage making decision of rate control (distortion). Fourth, code-blocks, the fundamental entities of tier-1 coding, are independent coded and the result, compressed data, is fed into next stage. Finally, those compressed data generated by previous stage is organized as packets (code-stream) by tier-2 coding.

Due to the diverse functionalities that JPEG2000 have, especially in that JPEG2000 provides both lossless and lossy coding scheme, there are many applications that could benefit from its nature capabilities, such as image distribution through internet, security systems over network, digital photography and medical imaging. From the applications above, we could know that JPEG2000 is an image coding technology for the higher quality and smaller data size. Thus, as long as the demands in seeking for perfect image quality exist, the need for JPEG2000 will not stop.

2.2 Traditional Approach in Parallelizing JPEG2000 Coding

Much work has been done in the literature in either parallelizing or optimizing the performance of JPEG2000 coding [12]. They take advantage of the multi-level parallelization opportunities in JPEG2000 standard, such as tiles, sub-bands, and code-blocks. In this section we would discuss about how far as they go and where the obstacles are. One typical obstacle is the poor scalability with respect to the number of cores.

Based on the literature and our experiments below [12, 14], the maximum speedup obtained from previous work is 3. Also, previous work typically does not focus on high resolution images or real-world images. Thus, we first run tests on our machines and compare the performance of two JPEG200 coders, Jasper and Kakadu. Next, we determine the maximum speedup we could get from these software implementations and choose the better one as our baseline. Note that the detailed information about the machine environment and tested images will be given in Section 4.

Performance comparison between Jasper and Kakadu is presented in Figure 2. Apparently, Kakadu outperforms Jasper in the entire spectrum of images ranging from 300 mega to 100 mega pixels. Single process and single thread are used in both runs. We will only experiment with Kakadu in the remaining paper, since Jasper is slower.

Figure 3 and 4 show the speedup of encoding an image by Kakadu on both HP DL380 with 4 logical processors and SUN Fire T2000 with 32 *CoolThreads*, respectively. The resolution of the image is 3648x2736. The speedup trend shown in Figure 3 and 4 demonstrate the parallelism coverage of Kakadu is not high enough. That is why speedup shown in Figure 3 remains below 3 while the number of tiles increases. As we introduce more threads, run on SUN Fire T2000 with 32 *CoolThreads* enabled, the speedup shown in Figure 4 remains below 1.15. It is worthy to note that the speedup trend of decoding the image on both machines shares the same characteristics with Figure 3 and 4, respectively.

From the experiments, we conclude that due to low parallelism coverage the performance will not scale even with more hardware resources. In the next section, we propose an oblivious parallelization paradigm to boost parallelism coverage and scale with the increasing number of cores.

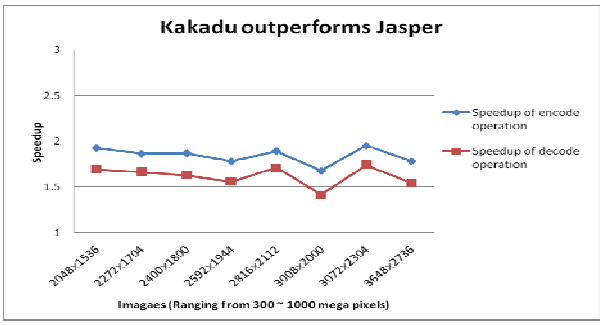

Fig. 2. Performance comparison between Jasper and Kakadu on real world images ranging from 300 to 1000 Mega Pixels.

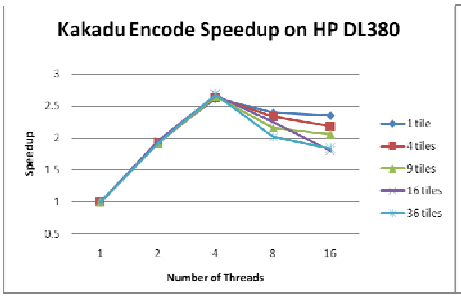

Fig. 3. Kakadu Encode Speedup on HP DL380. (A 3648x2736 image).

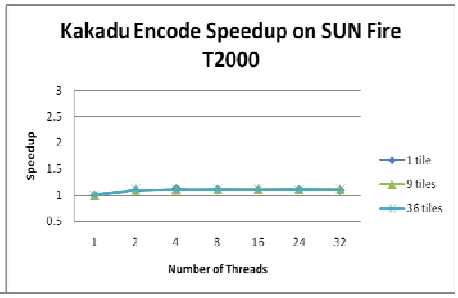

Fig. 4. Kakadu Encode Speedup on SUN Fire T2000. (A 3648x2736 image).

3 Oblivious Parallelization Paradigm

Traditional approach employs hierarchical parallelization from different stages, tiles, sub-bands, and code-blocks. Amdahl's Law states that the speedup from running a program in parallel is limited by the percentage of the program executed in parallel. We call such percentage the parallelism coverage. Section 2 indicates that while the traditional approach may be fine on a uni-processor with rich ILP (Instruction-Level Parallelism) and vector units, it does not scale with the increasing number of cores. Our profile data shows that about 80% of time is spent on decoding an input image into internal code stream. That means the ideal speedup is 5 when only this portion of the code is being parallelized in the traditional approach. This limitation on the traditional JPEG2000 parallelization approaches motivates the oblivious parallelization paradigm.

3.1 Illustration and Rationale

Our proposed approach aims at boosting the parallelism coverage yet helping programmer's productivity at the same time. Intuitively, the speedup would be *N* if we could process *N* pieces of the original image in parallel rather than the whole image in serial. Figure 5 illustrates the concept of oblivious parallelization paradigm. Note that the nature of JPEG2000 supports such style of parallel coding.

In comparison, although it is a common and effective way to process image coding in parallel, tiling may lead to the block effect in the resulting image. Besides, the parallelism coverage is lower than our oblivious parallelism scheme. To alleviate the block effect, researchers also propose other parallelized regions such as wavelet transform and tier-1 coding. However, the coverage is still lower.

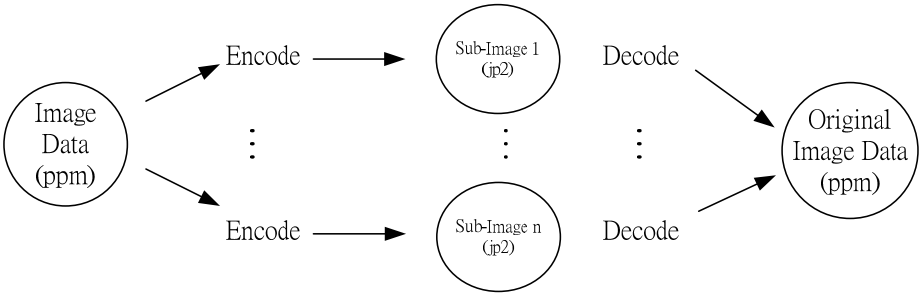

Fig. 5. Oblivious Parallelization Paradigm

The overhead of dividing and merging data in Figure 5 would be negligible, since the encoding operation (*ppm* format to *jp2* format) occurs mostly when the image capturing device reads out the raw data and stores (transforms) them into memory card or embedded memory in a compressed format, say *jp2*, and the decode operation (*jp2* format to *ppm* format) takes place mostly when the compressed file are going to be displayed on the machine where the *jp2* file located. On either circumstance, the proposed approach will mainly stress the memory system and file system. Oblivious parallelism paradigm will overlap the CPU and memory/file accesses better than the CPU-focused tiling approach.

Intermediate file size in *jp2* format is another possible issue. However, as our experiment in Section 4 shows, only less than 10% space overhead results, even if we divide an image of 2048x1036 into 140 pieces.

Finally, the division and blocking effect would not be a concern if we only focus on lossless JPEG2000 coding. The essence of reversible transformation is that either encoded or decoded data would be the same, regardless of the number of pieces we process concurrently.

3.2 Modeling the Computation Power of Platform

To bound the JPEG2000 coding time for different images, we propose a Random-Generated Image Algorithm. The algorithm generates an image with the same

resolution specified at input. Due to the random nature, each pixel is independent of its surrounding pixels. This requires more computation during encoding and decoding. Thus, these images serve to predict how much time it takes to encode/decode images with certain resolutions in the worst case.

It is worth noting that this algorithm provides compute power estimation with respect to JPEG2000 processing power before any input image is used. Finally, we predict speedup using this model while oblivious parallelization paradigm is applied. The model also implies how many sub-images are needed to start with. More results in the next section demonstrate the use of this model.

Algorithm 1 Random-Generated Image Algorithm.

Input: Resolution, *X* and *Y*, of the image to be generated, where *X* represents width and *Y* represents height of the image.

Output: The image, *I*, with resolution, *X* and *Y*, where each pixel is generated based on Uniform Distribution.

for each unit *px* in *X*
  for each unit *py* in *Y*
    Red component of pixel in coordinate (*px*, *py*) is set to a random number with uniform distribution.
    Green component of pixel in coordinate (*px*, *py*) is set to a random number with uniform distribution.
    Blue component of pixel in coordinate (*px*, *py*) is set to a random number with uniform distribution.
  end for
end for

4 Experimental Results

Table 1. Experiment environments.

Machine Model	HP DL380	SUN Fire T2000
CPU	Xeon 3.2GHz x 2	UltraSPARC T1 @ 1.0GHz x 8
Main Memory	2048MB	16376 MB
Operating System	Linux 2.6.19	Solaris 11
Threading	4 *logical processors*	32 *CoolThreads*

This section presents the performance results, including the upper-bound execution time from random-generated images, execution time from real-world images, performance speedup on both environments, the performance trend with increasing number of processors, and the size overhead of proposed paradigm. Also, two main contributions, real time capability and scalability, are provided in this section.

Note that throughout this paper, real-world images represent the images captured by different digital cameras with different resolution. Here, we use real-world images as an example to exhibit performance results of oblivious parallelization paradigm because of wide spread use of digital camera in recent years. Our work might not only

contribute to digital devices that perform image coding, but also extend to other field, such as medical image coding.

We first describe the experimental setups and the input images. Table 1 outlines our environments, which are commodity main stream machines on the market. Note that the experiments done on HP DL380 machine are all run with 4 logical processors, while the experiments run on SUN Fire T2000 are with either 32 *CoolThreads*, or varied *CoolThreads*, to observe the relationship between the number of cores and speedup. The mapping between tasks and execution units in multicore is handled either by user level library or operating system, where the tasks refer to sub-images to be encoded or decoded. Table 2 listed the resolution of real-world images used in this paper. All the images, ranging from 300 mega to 1000 mega pixel (MP), are transformed into ppm file format before the experiment.

Table 2. Testing images information (captured by different digital cameras).

Resolution	Image size (.ppm)	Pixels
3648x2736	29,241 KB	9,980,928
3488x2616	29,388 KB	9,124,608
3264x2448	25,735 KB	7,990,272
3072x2304	20,736 KB	7,077,888
3008x2000	17,211 KB	6,016,000
2816x2112	17,051 KB	5,947,392
2592x1944	14,416 KB	5,038,848
2400x1800	12,359 KB	4,320,000
2272x1704	11,342 KB	3,871,488
2048x1536	9,000 KB	3,145,728

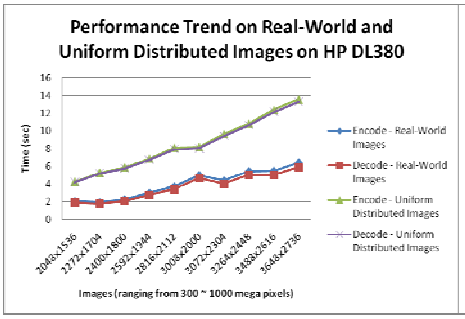

Fig. 6. Performance trend on both Real-World and Uniform Distributed images.

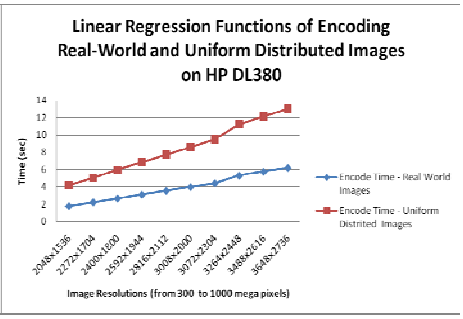

Fig. 7. Linear Regression Functions of encoding Real-World and Uniform Distributed Images on HP DL380.

Figure 6 depicts the trend while the image resolution increases. It is obvious that the higher the image resolution is the longer execution time it needs to process (decode/encode) the image. As expected, random-generated images (uniform-distributed images) cost more time than the real-world images do. Unrelated pixels result in extra processing time. Also, this leads to larger size of compressed file, jp2 file, than that of the original ppm file. This property could help us analyze the

computing power of certain hardware and software combination. In our example, the combination is HP DL380 and Linux operating system. In order to quickly achieve what we have done above rather than testing random-generated images one resolution by one resolution, we model the patterns from the results we have. Interestingly, we obtain a linear curve in Figure 7 and the curve passes most of the points in Figure 6. This suggests that we could gain the upper-bound execution time by simply running few random-generated images instead of those in whole spectrum.

In the remaining paper, we use the smallest image, 2048x1536. Larger images would give better results. Average speedup of each image being coded on HP DL 380 is shown in Figure 8. Furthermore, the speedup of real-world images and random-generated images are shown. Note that this diagram shows significant speedup on a random-generate images (about 50 to 60 times) faster than the sequential mode run by Kakadu. In contrast to a flat line at 100 sub-images of encoding/decoding real-world images, the speedup of random-generated images continues to increase. Note that the results suggest that the best sub-image size to perform oblivious parallelization paradigm for real-world images is about 90KB (9,000/100 KB). Furthermore, the result shows that real time encoding/decoding, 15 frames per second, is possible if the hardware resources are available. While it takes about 2.05 second to encode the 2048x1536 image in serial, the oblivious parallelization paradigm yields a speedup of 29 in the case of 44 sub-images. In addition, it takes about 0.04 second to encode the image when the speedup of 100 sub-images is 51.25.

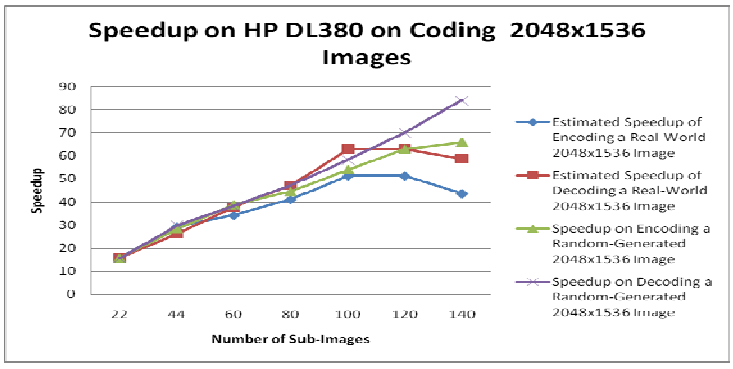

Fig. 8. Average execution time speedup on Encoding/Decoding 2048x1536 images, Real-World and Random-Generated images, on HP DL 380.

To show the performance speedup with different number of *CoolThreads*（Virtual Processors）, in SUN Fire T2000, we use minimum and maximum number of sub-images. As shown in Figure 9, the speedup remains at 6 while 140 sub-images are used. Alternatively, the speedup rises while the numbers of Virtual Processors increase. This shows that our paradigm is ready for multicore environment and with the maximum productivity.

The size overhead is shown in Figure 10. Apparently, the diagram shows that little overhead is introduced by our paradigm. Only less than 10% extra size is required while the numbers of sub-image are 140.

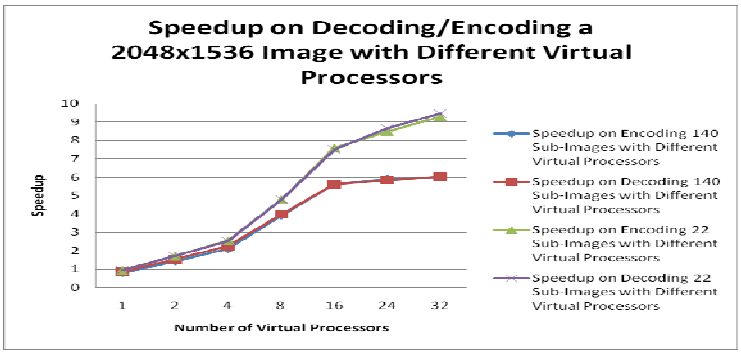

Fig. 9. Speedup on Encoding/Decoding a 2048x1536 image with Different Virtual Processors on HP DL 380 and SUN Fire T2000.

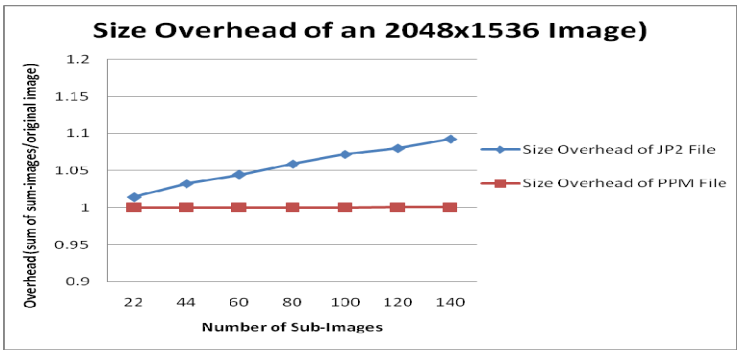

Fig. 10. Size overhead introduced by Oblivious Parallelization Paradigm.

5 Conclusion and Future Work

This paper presents an oblivious parallelization paradigm to boost parallelism coverage in high definition image compression and decompression for Digital Cinema domain. To the best of our knowledge, we are the first to propose a pure software-based solution on JPEG2000 image coding. Near real-time performance could be obtained from proposed method.

The contributions of this paper are three-fold. First, we show that images for 2K Digital Cinema could be compressed or decompressed in real-time, 24 frames per second, if the hardware resources are adequate. Second, the potential scalability and overhead of oblivious parallel paradigm are well presented. Finally, we have also proposed the Random-Generated Image Algorithm to predict image coding performance for specific hardware and software combination. Furthermore, according to our results, this modeling could be done faster by running few numbers of random-generated images. This could help to shorten the time for finding upper-bound execution time of image coding on specific platform.

In the future, we plan to set up an environment to do real-time video streaming system based on proposed algorithm. Also, we would like to extend our work to medical image coding, since lossless image coding is vital in medical image area. Finally, we would like to add hand-held devices into our system to play real-time video streaming films.

Acknowledgments. We would like to thank Dr. Chung-Jr Lian for his valuable comments and suggestions, Yu-Hong Liao and Yi-Di Lin for technical support on this project and to the unknown referees for their valuable suggestions to improve the quality of this work.

References

1. Novosel, D., Kovac, M.: Still image compression analysis. International Symposium Electronics in Marine (2004) 567-572
2. Santa-Cruz, D., Grosbois, R., Ebrahimi, T.: JPEG 2000 performance evaluation and assessment. Signal Processing: Image Communication **17** (2002) 113-130
3. Shirai, D., Yamaguchi, T., Shimizu, T., Murooka, T., Fujii, T.: 4K SHD Real-Time Video Streaming System With JPEG 2000 Parallel Codec. IEEE Asia Pacific Conference on Circuits and Systems (2006) 1855-1858
4. Lian, C., Jr., Chen, K.-F., Chen, H.-H., Chen, L.-G.: Analysis and architecture design of block-coding engine for EBCOT in JPEG 2000. IEEE Transactions on Circuits and Systems for Video Technology **13** (2003) 219-230
5. Chang, Y.W., Cheng, C.C., Chen, C.C., Fang, H.C., Chen, L.G.: 124 MSamples/s Pixel-Pipelined Motion-JPEG 2000 Codec Without Tile Memory. IEEE Transactions on Circuits and Systems for Video Technology **17** (2007) 398-406
6. Fang, H.-C., Chang, Y.-W., Wang, T.-C., Lian, C., Jr., Chen, L.-G.: Parallel embedded block coding architecture for JPEG2000. IEEE Transactions on Circuits and Systems for Video Technology **15** (2005) 1086-1097
7. Andra, K., Chakrabarti, C., Acharya, T.: A high-performance JPEG2000 architecture. IEEE Transactions on Circuits and Systems for Video Technology **13** (2003) 209-218
8. Adams, M.D., Kossentini, F.: JasPer: a software-based JPEG-2000 codec implementation. IEEE International Conference on Image Processing (2000) 53-56
9. JasPer JPEG2000 codec. http://www.ece.uvic.ca/~mdadams/jasper.
10. JJ2000, A JAVATM implementation of JPEG 2000. http://jj2000.epfl.ch.
11. Kakadu JPEG2000 codec. http://www.kakadusoftware.com.
12. Meerwald, P., Norcen, R., Uhl, A.: Parallel JPEG2000 Image Coding on Multiprocessors. IEEE International Parallel and Distributed Processing Symposium (2002) 2-7
13. Skodras, A., Christopoulos, C., Ebrahimi, T.: The JPEG 2000 still image compression standard. IEEE Signal Processing Magazine **18** (2001) 36-58
14. Hong Man, A.D., Kossentini, F.: Performance Analysis of the JPEG 2000 Image Coding Standard. Multimedia Tools and Applications **26** (2005) 27-57