ACCELERATION AND IMPLEMENTATION OF JPEG2000 ENCODER ON TI DSP PLATFORM

*Chien-Chih Liu and Hsueh-Ming Hang**

Electronics Engineering Department, National Chiao Tung University, Hsinchu, Taiwan, R.O.C.
ccliu.iic93g@nctu.edu.tw, hmhang@mail.nctu.edu.tw

ABSTRACT

JPEG2000 provides excellent compression performance and fine granularity scalability but at the cost of high computational complexity. We propose two speed-up techniques and use the TI DSP optimization tools to accelerate the Tier1 module. We eliminate the unnecessary checking cycles by recording the NBC (Need-to-Be-Coded) samples on a list. Furthermore, the sample index is reordered to facilitate fast execution. In the DSP implementation of the proposed methods, we use code acceleration techniques, cache memory allocation, and TI DSP compiler-level optimization tools. Even when the original program is compiled with the same DSP optimization tools and proper cache assignment, our fast algorithm can still reduce the computation by 45%.

Index Terms— JPEG200, DSP, algorithm acceleration

1. INTRODUCTION

In contrast to the discrete cosine transform (DCT) used in the JPEG standard, the JPEG2000 standard [1] implements an entirely new way of compressing images based on the wavelet transform. It supports lossy and lossless compression of single-component (gray-level) and multi-component (color) images. The major operation blocks of the JPEG2000 encoding system are shown in Figure 1. The pre-processing includes the image tiling, DC-Level shifting, and component transform. The component transform and the discrete wavelet transform have both the irreversible mode and the reversible mode used for lossy and lossless coding, respectively. The entropy coding part of JPEG2000 adopts the EBCOT technique (Embedded Block Coding with Optimized Truncation) [2]. It consists of two major coding steps, Tier-1 and Tier-2. The Tier-1 part is an embedded block coding scheme consists of the context formation (CF) and the arithmetic encoder (AE). The Tier-2 and rate-control part adopts the PCRD (Post-Compression Rate-Distortion) optimization to truncate the embedded bit-stream to minimize the overall distortion.

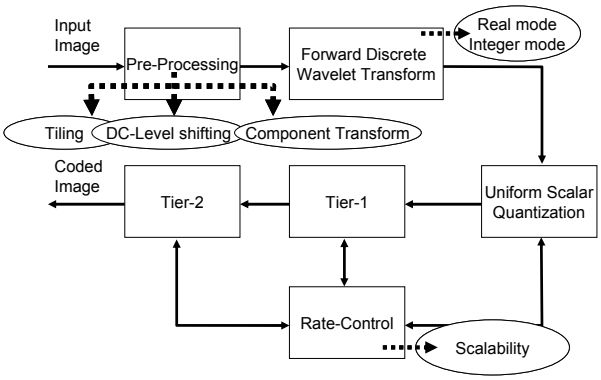

Figure 1 JPEG2000 encoder architecture

2. ENCODER COMPLEXITY ANALYSIS

We implement a JPEG2000 encoder on a DSP platform including two Sundance modules, SMT395 (TI TMS320C6416T DSP) and SMT310. We start with the OpenJPEG (ver.1.0) [3] reference software in C language. Then, the TI CCS (Code Composer Studio ver.3.1) [4] is used to compile the C codes and profile the encoder complexity.

2.1. Profiling results

TI CCS provides many simulation tools. The *C64xx CPU cycles accurate simulator* assumes a flat memory system in simulating the C64xx processor. In contrast, the *C6416 device cycle accurate simulator* can provide an accurate simulation on the C6416 processor, peripherals, and memory system.

Table I Profiling lossless encoding results using two simulators (Goldhill 512x512)

Simulator	C64xx	%	C6416	%	Ratio
DWT	73,327,701	7.8	552,674,115	6.7	13 %
Tier1	846,100,912	90.9	7,509,481,733	91.7	11%
Tier2	1,550,147	<1	15,933,932	<1	9%
Others	9,475,720	1	103,399,183	1.2	9%
Total	930,454,480	100	8,181,488,963	100	11%

*This work was partially supported by National Science Council, Taiwan, R.O.C., under Grant NSC-94-2213-E-009-144.

We profile the OpenJPEG encoder by using these two simulators as shown in Table I. The profiling results show that Tier1 module is the most complex part in the encoder. Also, the total cycles of the C64xx simulator are only 11% of that of the C6416 simulator. We calculate the execution time based on the cycles generated by TI C6416 simulator. It takes about 8.18 sec and the actual running time of the emulator (DSP hardware platform) also takes about 8.74 sec.

2.2. Block coding analysis

The Tier1 module takes the most of total cycles in the JPEG2000 algorithm and the block coding procedure is the main part in the Tier1 module. Details of the block coding process are described in [2]. Each bit-plane of the wavelet outputs is coded by 3 passes. We collect the statistics of these Pass processes on an image in Figure 2. For each bit-plane, we count the numbers of samples coded by each Pass. The most-significant-bit plane of every sample can be different. At the higher bit-planes, Pass3 is most active and its activity reduces to nearly zero rapidly. The Pass1 and Pass2 processes encode about a quarter of all samples as shown in Table II. According to the standards, each Pass needs to check the state of every sample and encodes the NBC (need-to-be-coded) samples with designated context labels. Since only a small percentage of samples are truly coded in each pass, most checking cycles are wasted.

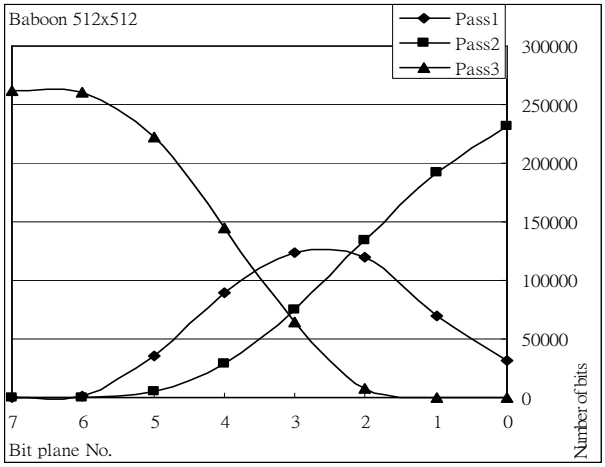

Figure 2 Analysis of Pass contributions
(Gray-level Baboon, 512x512)

Table II Coded samples in each Pass processes

Image	Pass1 process	Pass2 process	Pass3 process
Goldhill	24 %	22 %	54 %
Barb	22 %	24 %	54 %
Lena	22 %	17 %	61 %
Baboon	26 %	36 %	38 %

2.3. Major encumbrances and known speed-up methods

According to the profiling results, firstly we have to decrease the memory accessing time. Accessing the external memory causes many stall cycles in the total executing cycles. Our DSP platform has 1 Mbytes internal memory. We can turn on the Level-2 cache and modify the data structure to improve the utilization rate of the internal memory. Secondly we like to reduce the execution cycles of the Tier1 module. There are several known speed-improving methods such as the CUPS (Clean Up Pass Skipping) and the PP (Pass Predicting) [5] [6], the SS (Sample Skipping) and the GOCS (Group Of Column Skipping) methods [7], and the PPP (Pipelined Processing of Pass) method [8] [9].

3. PROPOSED SPEED-UP METHODS

3.1. VGOSS (Variable-Group-Of-Sample-Skip)

Extended from known methods, we propose a method to improve the block coding process and call it VGOSS (Variable-Group-Of-Sample-Skip) [10]. We first rearrange the code-block samples and set up a flag-block as shown in Figure 3. The Code-block records each sample in a block. The Flag-Block records the context orientation and the already-visited information for each bit-plane. It is also expanded and padded with shaded samples shown in Figure 4 to include the neighboring samples. This rearrangement does not change the coding performance; it costs few cycles in conversion but saves a lot of cycles later.

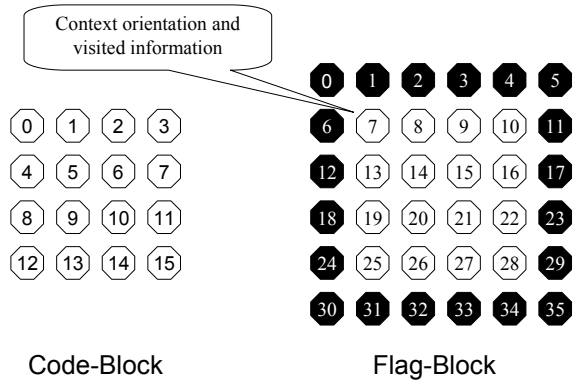

Figure 3 Code-Block and Flag-Block

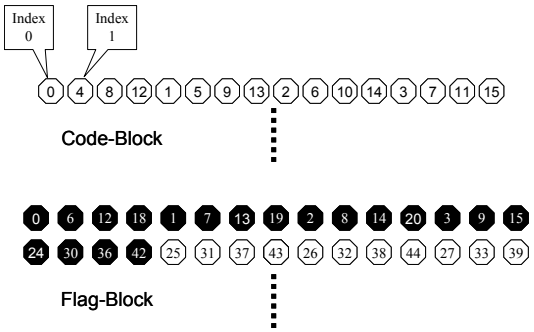

Figure 4 Rearranged Code-Block and Flag-Block

Then, the Flag-Block updating procedure is modified for the rearranged Flag-Block as shown in Figure 5. There are four samples in a stripe and four cases are executed. The modified procedure takes about 62% in calculation comparing to the original one.

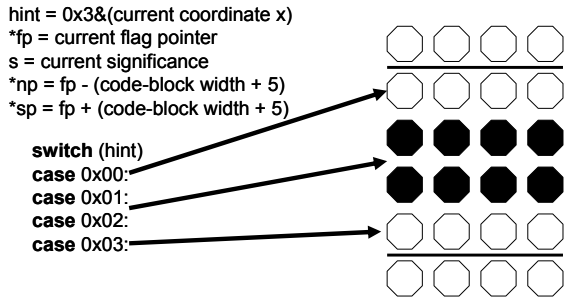

Figure 5 Modified updating procedure

The VGOSS flowchart is given in Figure 6. Each coding block is rearranged. Three pass processes are executed repetitively until all bit-planes are coded. The Pass1 process checks the states of all samples and records the offsets of NBC in a VGOSS table as results of the other two processes.

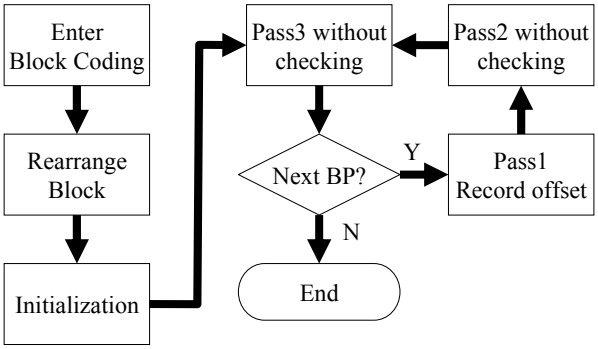

Figure 6 VGOSS flowchart

Thus, the following Pass2 and Pass3 processes can encode the NBC samples without checking the flag-block as shown in Figure 7. If the associated list is empty, the process can be skipped.

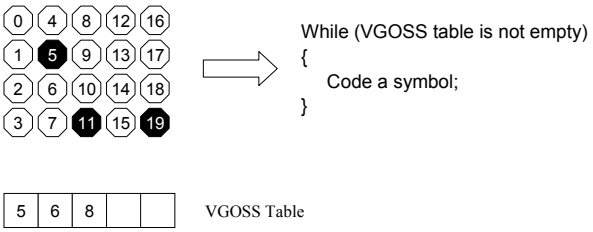

Figure 7 Coding without checking states

Our method is different from the PP method. We do not compare three tables in a pass process and there is no missing sample in our algorithm. Our algorithm can skip Pass3 without extra effort. Also, the fixed group size in GOCS is not efficient on our implementation [10] and the experimental results in [7] also give the similar result. We test the GOCS and SS method on our DSP platform. The experimental results are discussed in Section 4.

3.2. Modified VGOSS

According to the PP method, the absolute coordinates are recorded in the prediction table. It may become more efficient if the missing sample and the sorting problems can be solved. We thus modify the VGOSS method to record the absolute address in a code-block. Basically, all the pass procedures are similar to the original VGOSS procedure. Only the Pass1 procedure is modified as shown in Figure 8, and all pass processes use the absolute index in the VGOSS table.

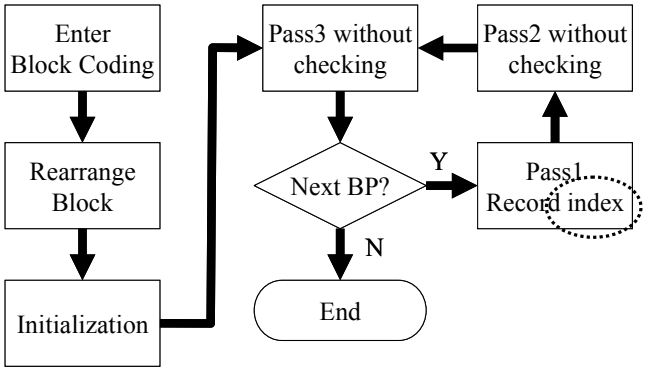

Figure 8 Modified VGOSS flowchart

3.3. Software speed-up techniques

We use a few intrinsic SIMD (Single Instruction Multiple Data) functions of TI DSP to improve the speed [4]. Thus, two short type string data can be executed at the same time. Furthermore, the JPEG 2000 arithmetic coding algorithm contains sequential processing steps, nested conditional operations, and inner while loops. The LMBD intrinsic function can reduce the complexity of RENORME function as well [11]. Appropriate data allocation improves the memory accessing by the DATA_SECTION and the CODE_SECTION. The direct internal memory access is still faster than using the L2 cache. Therefore, proper memory allocation in different memory banks is critical for the overall speed. In addition, we unroll the loops, modify C code style, and use UNROLL pragma [4].

4. EXPERIMENTAL RESULTS

The proposed algorithm is added on the OpenJPEG ver.1.0 encoder and tested on the TI DSP platform. Four test images are tested for lossless encoding. The TI compiler-level is set on the highest level (file-level) optimization. The L2 cache is adopted as well. We also implement the GOCS and SS methods using the same environment as marked "M1". The

VGOSS method is marked as "M2" and the modified VGOSS method is marked as "M3". When the software speed-up techniques are included, they are marked as "M2+" and "M3+". The simulation results on the C64xx and the C6416 simulators are shown in Figure 9 and Figure 10. The experimental results on the hardware platform are shown in Figure 11. Comparing all the experimental results, they are consistent with our prediction.

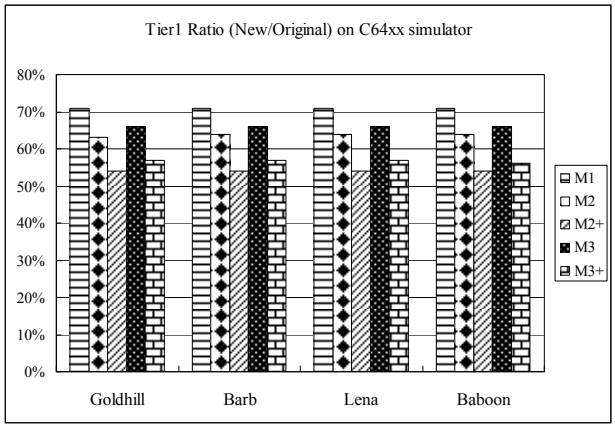

Figure 9 Comparison on C64xx simulator

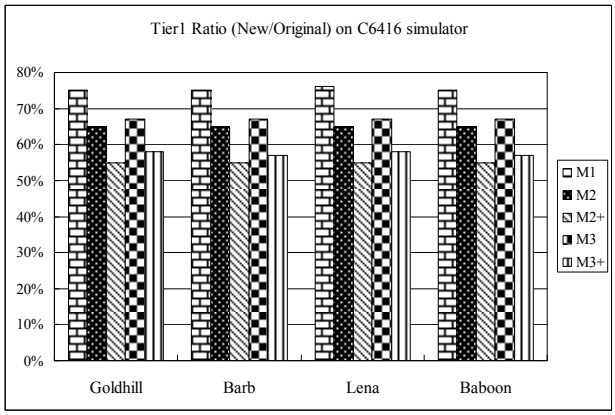

Figure 10 Comparison on C6416 simulator

5. CONCLUSIONS

The proposed VGOSS method is constructed on the re-ordered code-block samples and is a combined extension of the GOCS and PP methods. It encodes only the NBC (need-to-be-coded) samples and thus reduces the checking cycles in the original pass processes. It is easy and simple to implement and has a good performance on the TI DSP platform. If the DSP compiler-level optimization technique is applied to the original codes, our proposed method can still reduce about 45% computation. The improvement is mainly due to two factors, (1) our proposed VGOSS method and (2) the software speed-up techniques. In summary, the memory bottleneck is still a challenge to the embedded system. The speed gap between the idea cycles and real cycles indicates that there is still room for improvement to accelerate the JPEG2000 algorithm on different types of the embedded systems.

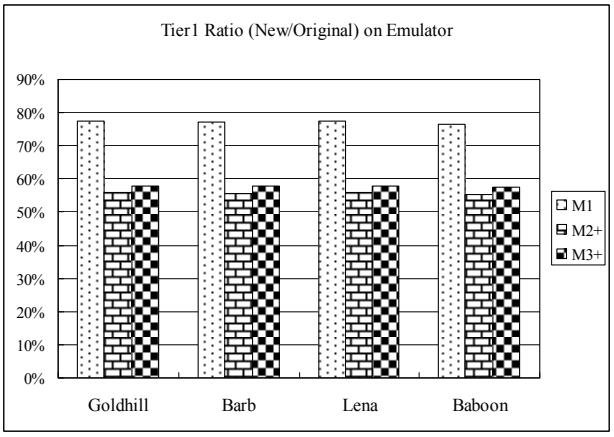

Figure 11 Comparison on hardware platform

7. REFERENCES

[1] JPEG2000 Part I Final Draft International Standard (ISO/IEC FDIS 15444-1). ISO/IEC JTC1/SC29/WG11 N1855, Aug.2000.

[2] D. Taubman and et al, "Embedded Block Coding in JPEG2000", in *Proceedings of IEEE International Conference on Image Processing*, vol. 2, Vancouver, Canada, Sept. 2000, pp.33-36

[3] http://www.openjpeg.org/index.php?menu=main

[4] Texas Instruments, T*MS320C6000 Programmer's Guide*, Literature number SPRU198I, Mar. 2006.

[5] K.L. Lin, *Analysis and Architecture Design for JPEG2000 Still Image Encoding System*, M.S. thesis, Department of Electrical Engineering, National Central University, Chung-Li, Taiwan, ROC, 2002.

[6] T.H. Tsai and L.T. Tsai, "JPEG2000 Encoder Architecture Design with Fast EBCOT Algorithm", I*EEE VLSI-TSA International Symposium*, page 279-282, April 2005.

[7] C.J. Lian and et al, "Analysis and Architecture Design of Block-coding Engine for EBCOT in JPEG2000", *IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 13, No. 3, March 2003.

[8] B.D. Choi, and et al, "DSP Implementation of Real-time JPEG2000 Encoder Using Overlapped Block Transferring and Pipelined Processing", *International Conference on High Performance Computing* (HiPC) 2004, Vol. 3296, page 333-341.

[9] J.K. Cho and et al, "Fast DSP implementation of JPEG2000", TENCON 2004, Vol. A, page 231-234 Vol. 1, Nov. 2004.

[10] C.C.Liu, *Acceleration and Implementation of JPEG2000 Encoder on TI DSP Platform,* M.S. thesis, College of Electrical and Computer Engineering, National Chiao Tung University, Hsinchu, Taiwan, ROC, 2006

[11] B. Valentine and O. Sohm, "Optimizing the JPEG2000 Binary Arithmetic Encoder for VLIW Architectures", IEEE *International Conference on International Conference on Acoustics, Speech, and Signal Processing* (ICASSP), Vol. 5, P.5, May 2004