RESEARCH ARTICLE OPEN ACCESS

FAST JPEG Encoding Using OpenMP

M. M. Raghuwanshi [1], Malhar Ujawane, Sameer Bhute,
Sankalp Saoji, Madhushree Warkhade, Shivani Kadpati
Professor [1]
Department of Computer Technology
Yeshwantrao Chavan College of Engineering
Nagpur – India

ABSTRACT
Multi-core programming has given rise to advanced implementations in computing. Increasing number of compression algorithms are being remodelled to utilize the multi-core computational power available at hand. In JPEG (Joint Photographic Experts Group) compression, DCT is a computationally intensive function which bottlenecks the performance of the algorithm. This paper presents a fast solution for DCT with the help of parallel programming, using OpenMP for the implementation on a multi-core CPU. The results indicate that in the parallel implementation exhibits considerable speedup as compared to the serial implementation.

Keywords :- JPEG, DCT, image compression, OpenMP, parallel programming, multi-core processor**.**

I. INTRODUCTION

One of the recent innovations in computer engineering has been the development of multicore processors, which are composed of two or more independent cores in a single physical package. Today, many processors, including digital signal processor (DSP), mobile, graphics, and general purpose central processing units (CPUs) [2] have a multicore design, driven by the demand of higher performance. Major CPU vendors have changed strategy away from increasing the raw clock rate to adding on-chip support for multi-threading by increases in the number of cores; dual and quad-core processors are now commonplace. Image processing progammers can benefit dramatically from these advances in hardware, by modifying single-threaded code to exploit parallelism to run on multiple cores. This article describes the use of OpenMP (Open Multi-Processing) to multi-thread image processing applications to take advantage of multicore general purpose CPUs. OpenMP is an extensive and powerful application programming interface (API), supporting many functionalities required for parallel programming. The purpose of this article is to provide a high level overview of OpenMP, and present simple image processing operations to demonstrate the ease of implementation and effectiveness of OpenMP. More sophisticated applications could be built on similar principles [1].

In recent years many new standards have emerged for image compression. JPEG (Joint Photographic Experts Group) is one of the most widely used and adapted standard for still images. JPEG is a common standard for lossy image compression, especially images produced digitally. Modifications to the JPEG baseline algorithm allow us to balance trade-offs between storage size and quality.

In this paper, we describe an approach to implement the JPEG encoder in parallel on a multiple cores of the CPU. The computationally intensive element of the encoding scheme namely DCT, will be run in parallel on the multi-core CPU. The program for parallel JPEG encoder with modified DCT function is implemented in OpenMP. The evaluation of the execution is done by comparing the serial and parallel execution times.

II. JPEG ENCODING

JPEG, which stands for Joint Photographic Experts Group (the name of the committee that created the JPEG standard), is a lossy compression algorithm for images. A lossy compression scheme is a way to inexactly represent the data in the image, such that less memory is used yet the data appears to be very similar. This is why JPEG images will look almost the same as the original images they were derived from most of the time, unless the quality is reduced significantly, in which case there will be visible differences. The JPEG algorithm takes advantage of the fact that humans can't see colors at high frequencies. These high frequencies are the data points in the image that are eliminated during the compression. JPEG compression also works best on images with smooth color transitions.[9]

For each mode, one or more distinct codecs are specified. As seen in Fig. 1, they are: downsampling, forward DCT (Discrete Cosine Transform), quantization, entropy encoding. Codecs within a mode differ according to the precision of source image samples they can handle or the entropy coding method they use. Although the word codec (encoder/decoder) is interoperable, there is no requirement that implementations must include both an encoder and a decoder. Many applications will have systems or devices which require only one or the other [9].

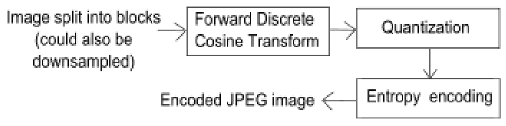
Image split into blocks (could also be downsampled)
Forward Discrete Cosine Transform
Quantization
Entropy encoding
Encoded JPEG image

Fig. 1 Baseline JPEG encoding process

The four modes of operation and their various codecs have resulted from JPEG's goal of being generic. The multiple pieces can give the impression of undesirable complexity, but they should actually be regarded as a comprehensive "toolkit" which can span a wide range of continuous-tone image applications. It is unlikely that many implementations will utilize every tool -- indeed, most of the early implementations now on the market (even before final ISO approval) have implemented only the Baseline sequential codec [9].

The Baseline sequential codec is inherently a rich and sophisticated compression method which will be sufficient for many applications. Getting this minimum JPEG capability implemented properly and interoperably will provide the industry with an important initial capability for exchange of images across vendors and applications [9].

III. JPEG ENCODING PROCESS

The algorithm behind JPEG is relatively straightforward and can be explained through the following steps:

1. Take an image and divide it up into 8x8-pixel blocks. If the image cannot be divided into 8x8 blocks, then add-in empty pixels around the edges, essentially zero-padding the image.

2. For each 8-by-8 block, get image data such that you have values to represent the color at each pixel.

3. Take the Discrete Cosine Transform (DCT) of each 8-by-8 block.

4. After taking the DCT of a block, matrix multiply the block by a mask that will zero out certain values from the DCT matrix.

5. Finally, to get the data for the compressed image, take the inverse DCT of each block. All these blocks are combined back into an image of the same size as the original [4].

A. Downsampling

Due to the densities of color- and brightness-sensitive receptors in the human eye, humans can see considerably more fine detail in the brightness of an image (the Y component) than in the color of an image (the Cb and Cr components). Using this knowledge, encoders can be designed to compress images more efficiently [10].

The transformation into the YCbCr color model enables the next step, which is to reduce the spatial resolution of the Cb and Cr components (called "downsampling" or "chroma subsampling"). The ratios at which the downsampling can be done on JPEG are 4:4:4 (no downsampling), 4:2:2 (reduce by factor of 2 in horizontal direction), and most commonly 4:2:0 (reduce by factor of 2 in horizontal and vertical directions). For the rest of the compression process, Y, Cb and Cr are processed separately and in a very similar manner [10].

B. DCT

The Discrete Cosine Transform (DCT) is a Fourier-like transform, which was first proposed by Ahmed *et al.* (1974). While the Fourier Transform represents a signal as the mixture of sines and cosines, the Cosine Transform performs only the cosine-series expansion. The purpose of DCT is to perform decorrelation of the input signal and to present the output in the frequency domain. The DCT is known for its high "energy compaction" property, meaning that the transformed signal can be easily analyzed using few low-frequency components. It turns out to be that the DCT is a reasonable balance of optimality of the input decorrelation (approaching the Karhunen-Loève transform) and the computational complexity. This fact made it widely used in digital signal processing and image processing [5].

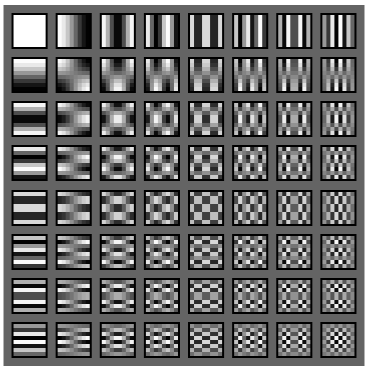

Fig. 2 Sample DCT 8x8 blocks

There are several types of DCT [6]. The most popular is two-dimensional symmetric variation of the transform that operates on 8x8 blocks (DCT8x8) and its inverse (Fig. 3). The DCT8x8 is utilized in JPEG compression routines and has become a de-facto standard in image and video coding algorithms and other DSP-related areas [5]. This can be seen in Fig. 2.

$$\mathrm{DCT}(i,j) = \frac{1}{\sqrt{2N}} C(i)\,C(j) \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} \mathrm{pixel}(x,y)\, \mathrm{COS}\left[\frac{(2x+1)i\pi}{2N}\right] \mathrm{COS}\left[\frac{(2y+1)j\pi}{2N}\right]$$

$$C(x) = \frac{1}{\sqrt{2}} \text{ if x is 0, else 1 if } x > 0$$

Fig. 3 Formula for 2-D DCT function

DCT is a computationally intensive task and hence parallelising this function would greatly improve the execution time.

C. Quantization

Quantization, in mathematics and digital signal processing, is the process of mapping a large set of input values to a (countable) smaller set. Rounding and truncation are typical examples of quantization processes. Quantization is involved to some degree in nearly all digital signal processing, as the process of representing a signal in digital form ordinarily involves rounding. Quantization also forms the core of essentially all lossy compression algorithms [4].

D. Entropy Encoding

Entropy coding is a special form of lossless data compression. It involves arranging the image components in a "zigzag" order employing run-length encoding (RLE) algorithm that groups similar frequencies together, inserting length coding zeros, and then using Huffman coding on what is left [10].

The JPEG standard also allows, but does not require, the use of arithmetic coding, which is mathematically superior to Huffman coding. However, this feature is rarely used as it is covered by patents and because it is much slower to encode and decode compared to Huffman coding. Arithmetic coding typically makes files about 5% smaller [10].

While using Huffman coding, the zigzag ordering of the JPEG components is done as:

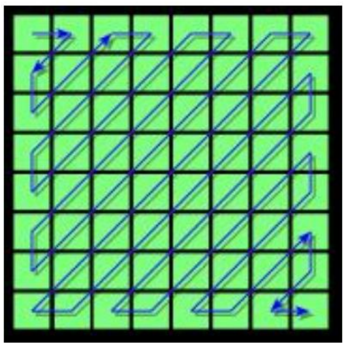

Fig. 4 Zig-Zag ordering in JPEG [10]

E. Compression Ratio and Artifacts

The resulting compression ratio can be varied according to need by being more or less aggressive in the divisors used in the quantization phase. Ten to one compression usually results in an image that cannot be distinguished by eye from the original. 100 to one compression is usually possible, but will look distinctly artifacted compared to the original. The appropriate level of compression depends on the use to which the image will be put [10].

At many times JPEG images appear with certain irregularities which are due to compression artifacts. These are due to the quantization step of the JPEG algorithm. They are especially noticeable around sharp corners between contrasting colours (text is a good example as it contains many such corners). They can be reduced by choosing a lower level of compression; they may be eliminated by saving an image using a lossless file format, though for photographic images this will usually result in a larger file size. Compression artifacts make low-quality JPEGs unacceptable for storing height maps. The images created with ray-tracing programs have noticeable blocky shapes on the terrain. Compression artifacts are acceptable when the images are used for visualization purpose. Unfortunately subsequent processing of these images usually result in unacceptable artefacts [10].

IV. OPENMP

Historically, a key challenge in parallel computing has been the lack of a broadly supported, simple to implement parallel programming model. As a result, numerous vendors provided different models, with often mixed degrees of complexity and portability. Software programmers subsequently found it difficult to adapt applications to take advantage of multicore hardware advances. OpenMP was designed to bridge this gap, providing an industry standard, parallel programming API for shared memory multi-processors, including multicore processors. A vendor-independent OpenMP Architecture Review Board (ARB), which includes most of the major computer manufacturers, oversees the OpenMP standard and approves new versions of the specification. Support for OpenMP is currently available in most modern Fortran and C/C++ compilers as well as numerous operating systems, including Microsoft Windows, Linux, and Apple Macintosh OS X. Version 1.0 of OpenMP was released in 1997. The latest version, 3.0, was released in 2008 [1,3].

A. Using OpenMP

OpenMP works as a set of pre-processor directives, runtime-library routines, and environment variables provided to the programmer, who instructs the compiler how a section of code can be multithreaded. In Fortran, the directives appear as comments, while in C/C++ they are implemented as pragmas. In this way, compilers that do not support OpenMP will automatically ignore OpenMP directives, while compilers that do support the standard will process, and potentially optimize the code based on the directives. Since the OpenMP API is independent of the machine/operating system, properly written OpenMP code for one platform can easily be recompiled and run on another platform. An OpenMP application always begins with a single thread of control, called the master thread, which exists for the duration of the program. The set of variables available to any particular thread is called the thread's execution context. During execution, the master thread may encounter parallel regions, at which the master thread will fork new threads, each with its own stack and execution context. At the end of the parallel region, the forked threads will terminate, and the master thread continues execution. Nested parallelism, for which forked threads fork further threads, is supported [1].

V. PARALLEL DCT IN JPEG ENCODER

The two-dimensional input signal is divided into the set of nonoverlapping 8x8 blocks and each block is processed independently. This makes it possible to perform the block-wise transform in parallel, which is the key feature of the DCT8x8 [5].

As the pixel data for each 8-by-8 block is independent of the other's, the data can be passed in parallel – SIMD. It is a case of data parallelism where different blocks are sent in parallel to the *dct* function. When multiple blocks are passed to the *dct* function, it takes as input *DCTMatrix* which is a 8x8 block of the image and performs the DCT on the block. Our parallel JPEG encoding scheme relies on OpenMP for implementing the 2-D DCT. Every time the *dct* function is called, the OpenMP loop is executed. OpenMP *parallel for* clause along with the *schedule()* clause is used to run the loop in parallel. This *schedule(runtime)* scheduling routine leaves it on the processor to decide onto which thread the iteration of the loop is to be processed. This ultimately permits for an optimal use of the available CPU threads.

VI. EXPERIMENTAL SETUP

System Parameters	Value
CPU	Intel Core i7 6700k @ 4.00GHz
RAM	8.00GB
Operating System	Ubuntu 16.04 (64-bit)
Compiler	gcc 5
OpenMP	OpenMP 4.0
OpenCV	OpenCV 3.0
Image Type	BMP
Image Size	512x512

Table 1. System Configuration

In our experimental setup, gcc was used to compile the program, with OpenCV being used for reading and writing of image data and OpenMP to parallelise the program. We considered different sets of 512x512 images.The program was run for "n" images and the serial v/s parallel results for varying n values were obtained. The time taken was calculated using "get time of the day".

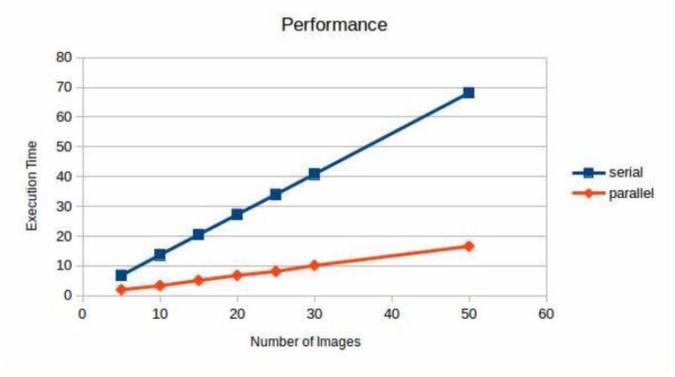

Fig. 5 Performance comparison serial v/s parallel

Speed Up = (Time for Serial)/ (Time for Parallel)

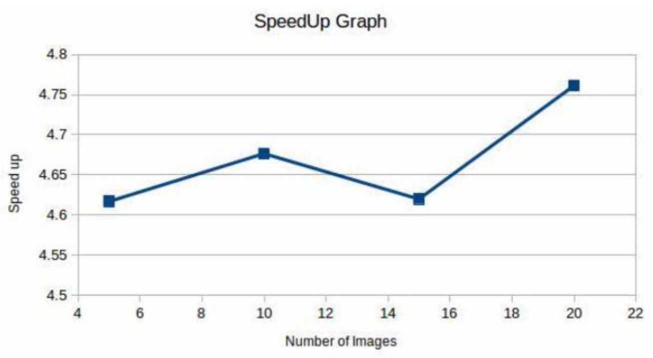

Fig. 6 Speed up Graph for parallel implementation

Average speed up = $T_{serial} / T_{parallel}$ = 4.7

VII. CONCLUSIONS

Modern computers with multi-core CPUs possess great computational power and an opportunity to increase processing performance with the use of parallel programming. This paper presented an optimization in the JPEG encoder by parallelizing computationally intensive DCT function. OpenMP was used for this purpose. Our experimental results show that the re-modeled implementation of the JPEG encoder with parallel DCT function performed considerably faster as compared to the serial implementation. It was observed that using runtime scheduling gave the best performance. Also, the speed up increased as we increased the number of images.

The parallel implementation results exhibit a 4.7 times speed up in the processing speed as compared to the serial implementation.

REFERENCES

[1] "Multicore Image Processing with OpenMP", Greg Slabaugh, Richard Boyes, Xiaoyun Yang.

[2] G. Blake, R. G. Deslinski, and T. Mudge, "A survey of multicore processors: A review of their common attributes," IEEE Signal Processing Magazine, vol. 26, no. 6, pp. 26–37, 2009.

[3] The OpenMP website. [Online]. Available: http://www.openmp.org/

[4] Matt Marcus, "JPEG Image Compression", Dartmouth College

[5] Nvidia Developers website. "Discrete Cosine Transform for 8x8 Blocks with CUDA" [Online]. Available: http://developer.download.nvidia.com/assets/cuda/files/dct8x8.pdf

[6] P. Soille, Morphological Image Analysis. Springer-Verlag, second ed., 2003.

[7] A. Jain, Fundamentals of Digital Image Processing. Prentice Hall, first ed., 1989.

[8] Stanford Image Compression Techniques [Online]. Available: http://cs.stanford.edu/people/eroberts/courses/soco/projects/data-compression/lossy/jpeg/dct.htm

[9] Gregory K. Wallace "The JPEG Still Picture Compression Standard", North Carolina State University, December 1991

[10] Harsh Vardhan Dwivedi, "Design of JPEG Compressor", National Institute of Technology, Rourkela (2009)