Sindh Univ. Res. Jour. (Sci. Ser.) Vol.44 (3) 361-366 (2012)

SINDH UNIVERSITY RESEARCH JOURNAL (SCIENCE SERIES)

Parallel Processing of Multimedia Algorithms

F. A. JOKHIO, S.VIGHIO*, I. A. KOREJO**

Department of Computer Systems , Engineering, Quaid-e-Awam University, Nawabshah, Pakistan
Email: fajokhio@quest.edu.pk Email: saleem.vighio@quest.edu.pk

Received 23rd May 2012 and Revised 14th July 2012

Abstract: This paper discusses an approach for parallelization of multimedia algorithms on multi-core and many-core systems. The parallelism is already available in these algorithms; however, exploiting it efficiently is a challenging task. Existing sequential implementations of multimedia algorithms can be parallelized through several models. We discuss a coarse-grain approach that allows a significant speed-up in terms of execution time. Furthermore, the paper also presents a case study that is based on a JPEG algorithm.

Keywords: JPEG encoding, Multimedia algorithms, Parallel processing

1. INTRODUCTION

In past, single processor systems were very famous and the programming model used was sequential. However, with the passage of time, clock frequency of processing units has increased, and it has become possible to execute programs with more speed. The system designers have increased the CPU clock frequency, the size of instruction pipelines, instruction level parallelism and so on. However, due to the physical limitations of silicon chips, it is not possible to continue the trend of having processing units with higher clock frequency. Furthermore, due to lack of parallelism in applications, instruction level parallelism has also been restricted. If a processor has a very high speed then the memory cannot be accessed with the same speed. Therefore, having slow memory and very powerful processor will not produce any better performance results.

The alternate solution is to use multiple systems and use their processing power with distributed computing. The distributed systems are very expensive in order to be purchased by a common user. The existing sequential programs will also not be appropriate in terms of increasing processing speed without modifications; this brings a new challenge for the programmers.

Nowadays, the trend towards the development and use of multi-core systems is emerging quite rapidly because of their potential of providing higher processing speed. However, these systems may still require programming efforts in order to be used efficiently.

In a single-core processor, multitasking is possible, however, it executes on same physical processor concurrently. The multi-core system can provide dedicated hardware cores to execute different tasks simultaneously. If tasks are independent of each other and no data dependencies are found then the performance of the system will increase. On the other hand, if there are inter-dependencies among different tasks and the data set used is the same, increasing processing speed cannot be achieved even by increasing the number of cores. The multi-core systems can also be classified into two major categories.

Symmetric Multi-core systems. In these systems all cores have exactly the same architecture and capabilities. They are useful for multithreaded software programs. Few examples of such systems include Intel Core 2 Duo and Quad-core processors.

Asymmetric Multi-core systems. These types of systems involve multiple cores and it is possible that different systems have different architecture and processing capabilities. An example of such a system is IBM Cell processor.

Applications designed for one category of multi-core systems might not speed-up and scale on another category. Video and image encoding/decoding algorithms such as MPEG, H.246, are JPEG are very complex, and require very huge computations. These algorithms require several complex operations such as variable length encoding/decoding, motion compensation estimation, quantization, inverse quantization and so on. Most encoder/decoder implementations are sequential and their designs are based on single processor systems where multiple tasks have data and control dependencies. The multi-core

++ Corresponding author: I. A. KOREJO Email: cskorejo@hotmail.com cell.No. +92-3323605603
*Department of Information Technology, Quaid-e-Awam University, Nawabshah, Pakistan
**Institute of Mathematics and Computer Science, University of Sindh, Jamshoro, Pakistan

systems have multiple processing units and their encoder/decoder implementations allow multiple simultaneous processing in order to perform subtasks.

The above discussion indicates that the encoder/decoder implementation has a great impact on the efficiency of any processing unit. The main contribution of this paper is the analysis of a JPEG encoder and its parallelization.

The rest of the paper is organized as follows: Section II presents the background and related work. Section III introduces parallel processing models for multi-core systems. A brief overview of JPEG encoder algorithm and its parallelism is discussed in Section IV. Section V gives system overview, experimental set-up and results. Finally, Section VI concludes the work with discussions.

2. MATERIAL AND MATHODS

Background And Related Work

The JPEG standard provides highly compressed and good quality images. It uses a set of algorithms for image data compression. The JPEG encoding algorithm is implemented in many different hardware and software platforms. It is not possible to cover many details about the related work here; however, we provide some selected contributions. Hardware based implementations of JPEG algorithm provides the maximum performance. Nethra imaging NI2065/66 (Nethra Inc. 2007) chip has achieved performance results of more than 70 Mega pixels/second. There are several hardware based implementations, some of them are discussed in (Kovac *et al*. 1995). In addition to hardware-based implementations, software-based implementation of JPEG algorithm also exists. In (Shee *et al.,* 2006) Shee has used Xtensa SL, and Osorio has used Ambric architectures in (Osorio *et. al.,* 2009). Both Xtensa SL and Ambric are message passing architectures. There are some shared memories architectures where parallelization of JPEG encoding is also addressed see for example, (Oh *et al.,* 2003), (Kodaka *et al*., 2002). The studies show that, the maximum performance using software-based JPEG encoders is 31 Mega Pixels/second (Castells-Rufas *et al.,* 2010).

Parallel Processing Models

The task level parallelism must be identified first in an application and then a suitable model must be selected for mapping tasks to a multi-core processor. There are several models available; however, we discuss only a few of them which are most suitable for our application. Those models are a Master-Slave model where the work is distributed by Master among slaves, i.e., one core will have a Master process and other cores will have slave processes. Other model is the Data Flow Model (DFM). In this model there is no central control and the work flows in the system in different stages. In the OpenMP execution model a program always starts with a single thread that executes sequentially. After some time, parallel constructs are encountered and a team of threads is created. Before selecting an appropriate model, the workload must be identified. The workload includes instructions required for computation and data access. The total execution time of an application is the number of processor cycles spent on performing the work. Furthermore, the hardware on which the application will be mapped must also be kept in mind. It is possible that a parallel processing model might be suitable for one hardware architecture but not for another hardware architecture. Since Master-Slave and Data Flow models are mostly used for parallel programming, therefore, we have an open choice for implementing them with any tool. The two models are discussed below:

A. Master-slave model

In Master-Slave model the control is centralized while execution is distributed. A single master core is responsible for the scheduling of tasks whereas the executable threads are allocated to available cores. If an application has many small and independent tasks then it is suitable for such a model. Real-time load balancing is a challenge for such applications because thread activation can happen randomly. There might be different throughput requirements for different threads. The master is responsible to maintain a list of available resources (cores) and distributes the work to them. The distribution of work should be balanced. **(Fig. 1)** shows the allocation of tasks using Master-Slave model.

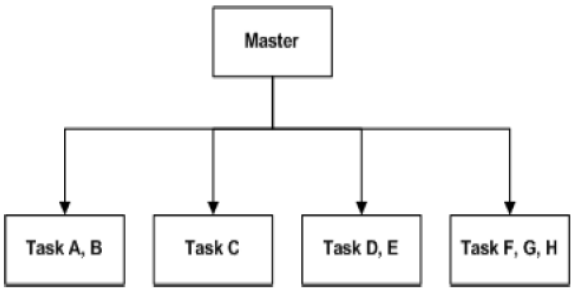


Fig.1. Master-slave model

B. Dataflow model

In data flow model control and execution are both distributed. A task executes on a core and data is passed to other cores for further processing. There is no central control and the scheduling depends on the availability of data. Applications which are suitable for this model involve several inter-dependent tasks. Video and image encoder/decoder are suitable applications for

data flow model where data patterns are regular and are processed uniformly. The main problem with the applications using data flow model is task mapping and data flow control through the system. Tasks scheduling using data flow model is shown in **(Fig. 2)**.

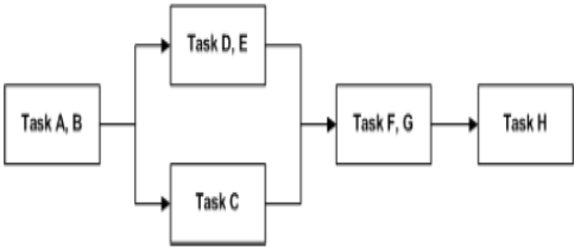


Fig.2. Dataflow model

Jpeg Compression

JPEG (Joint Photographic Experts Group) is a standard for storing digital image data (Bradley *et al*., 2002). JPEG compression can be lossy, non-lossy, progressive, and hierarchical (Rabbani *et al*., 1995). Among these compressions, lossy compression gives better compression results and is used mostly in practical applications. Usually, one single still-image (as human eyes can observe) contains large amount of pixels in it. Each pixel is formed by red, green, and blue colours where each colour element is represented by one single byte of data. When an image is being transferred before compression e.g., an image size of 512 x 420, the actual data memory needed for it is almost 645 Kbytes (512 x 420 x 3 = 445120 bytes). It occupies relatively large memory and the data transmission time using the internet is also relatively long. This is when compression is required in order to solve these problems (limited memory, time). The lossy compression method of JPEG file is also popular because of its Discrete Cosine Transform (DCT) technique. Even with a slight loss of data, it can still maintain high quality of image data after 30 times of compression rate of its original size. Because of its advantages, lossy compression method is still used in most of applications.

JPEG compression requires several steps; the diagram given in **(Fig. 3)** shows these steps. Various JPEG compression algorithms are available that use different compression approaches for example, High Dynamic Range (HDR) algorithm uses image encoding algorithm with wavelet-based approach as discussed in (Xu *et al*., 2005). The techniques for decreasing encoding time for JPEG200 is discussed in (Hashimoto *et al.*, 2005) where Discrete Cosine Transform (DCT) (Khayam 2003) based approach is used instead of wavelet transform. The JPEG compression steps are briefly discussed below:

Sections division. Sections division is used in order to perform DCT. Under this step an image is divided into 8 x 8 sections.

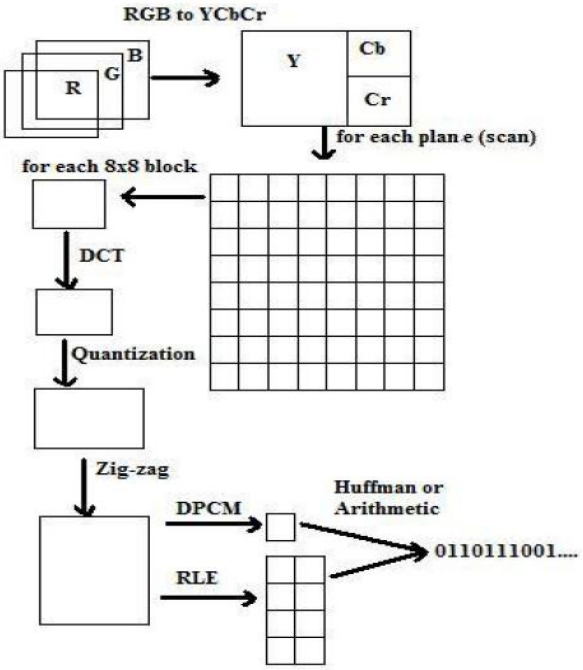


Fig.3. JPEG compression

Colour space transformation. This step converts RGB to YUV (YCbCr).

Down sampling. According to ratio of Cb and Cr, 16 x 16 sections with each containing sections of 8 x 8 frames is sampled.

Discrete Cosine Transform (DCT). Initially, an image data is in spatial domain, Discrete Cosine Transform (DCT) is used to transform this image data into frequency domain. In result, it produces 1 DC coefficient and 63 AC coefficients in 8 x 8 sections. DCT transformation is shown in **(Fig. 4)**. The basic operation of the DCT is as follows:

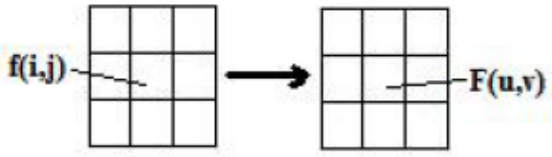


Fig.4. DCT transform

1) The size of input image is N x M.
2) f(i,j) represents the intensity of the pixel in row i. and column j.
3) F(u,v) represents the DCT coefficient of DCT matrix in row u and column v.
4) In an image, the lower frequencies have more signal energy; this lies in DCT at top left corner.

5) The compression can be achieved by neglecting the higher frequency, because they have less energy and there will be very small distortion.

Quantization. It brings better compression effects by minimizing coefficients after DCT. Quantization is used to reduce the number of bits.

Entropy coding. Image components will be arranged in a zigzag order and then Huffman coding will be used. Zigzag scan is used to group low frequency coefficients to the top of vector. Zigzag maps 8x8 to a 1x64 vector.

A. Parallelization of JPEG encoding algorithm

Encoding algorithm of JPEG consists of several other algorithms. These algorithms need to be executed on block level. For each block, we need to perform them in exactly the same order in order to preserve inter-dependencies among them. Among these algorithms DCT (Discrete Cosine Transform) requires more computation. **(Fig. 5)** shows the order in which these algorithms need to be executed.



Fig.5. Algorithms in JPEG encoding

It is possible to map each of the above algorithms on a separate processing unit; however, due to inter-core communication overhead, it is not be possible to get significant gain in terms of speed-up of execution time. The core, on which DCT task is mapped, will be busy all the time while other cores will have very less work to do. Even if we map all other tasks on single core and DCT on another core, the work load will not be balanced **(Fig. 6)**.

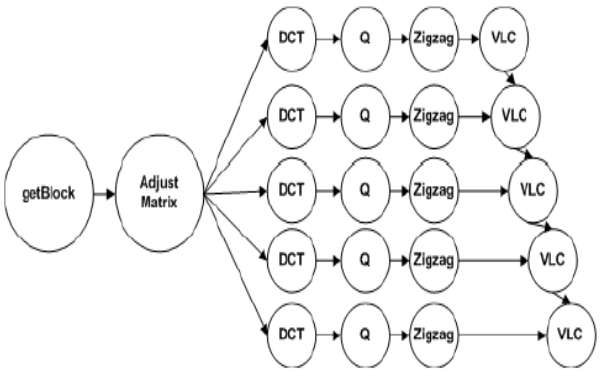


Fig.6. Parallelization of JPEG encoding

If there is no intra-prediction, then blocks within a JPEG image are independent of each other. It is possible that each block of data be sent to a separate core and performs DCT, quantization, zigzag scan and variable length encoding for that block. A proposed parallelization model of JPEG algorithm is shown in Fig. 6. In order to perform block level encoding, we will require creating a separate thread per block-level and performing DCT, quantization, zigzag scan and variable length encoding with that thread of the same block. In this way, it will not be required to transfer data among different cores while performing different operations on a single block. The result is that, there will be less communication overhead. It must be noted that there are interdependencies among the blocks for performing the VLC algorithm.

4. RESULTS AND DISCUSSION

Profiling results of JPEG encoding algorithm are shown in **(Fig. 7)**. In the Figure, we show only those functions which require some computational power. It can be seen that DCT requires 66.67% of total encoding time. This profiling result is taken before actual parallelization of the algorithm is performed.

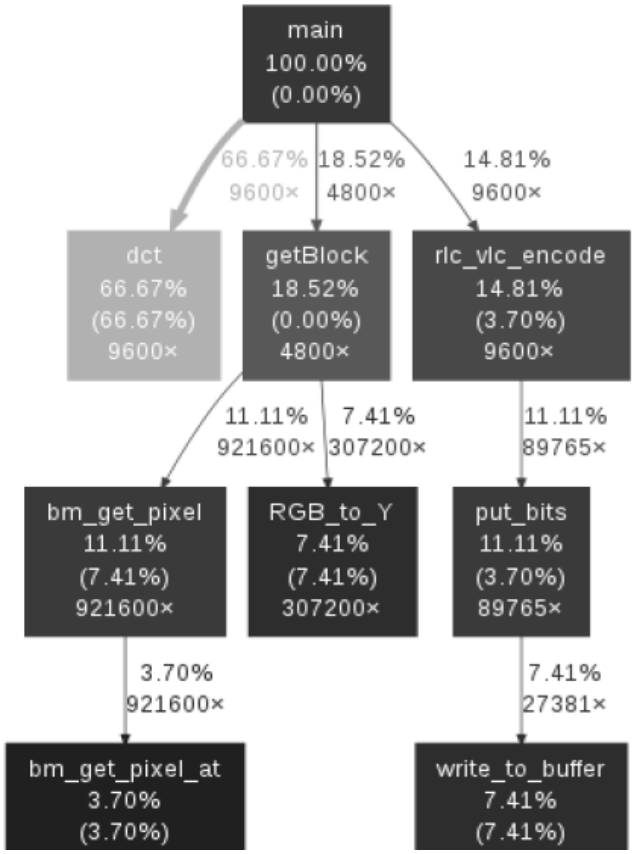


Fig.7. Profiling JPEG

In order to paralleling JPEG algorithm we use the POSIX threads. This will introduce some overhead, however, with an increasing number of cores, gain be improved as compared to the overheads. Speed-up

obtained with increasing number of cores is shown in **(Fig. 8)**. The experiments are performed on a machine having quad-core processor with hyper threading support. The machine has Intel(R) Core(TM) i7-2630QM CPU @ 2.00 GHz processor with 6.00 GB of installed memory (RAM). The results are obtained for a set of images and then an average of speed-up of all images is presented. The amount of speed-up is almost similar in different images due to the fact that all images have large number of blocks and it is possible to equally distribute those blocks on a small number of processing cores.

In an ideal case, the speed-up should be linear. If the number of processing units grows, the speed-up also grows with the same factor. Usually an algorithm has both serial and parallel parts. The well-known Amdh's model states that "If P is the part of a program which execute in parallel, and (1-P) is a part that does not execute in parallel, then the maximum speed-up that can be obtained with N processors is"

$$S(N) = \frac{1}{(1-P) + \frac{P}{N}}$$

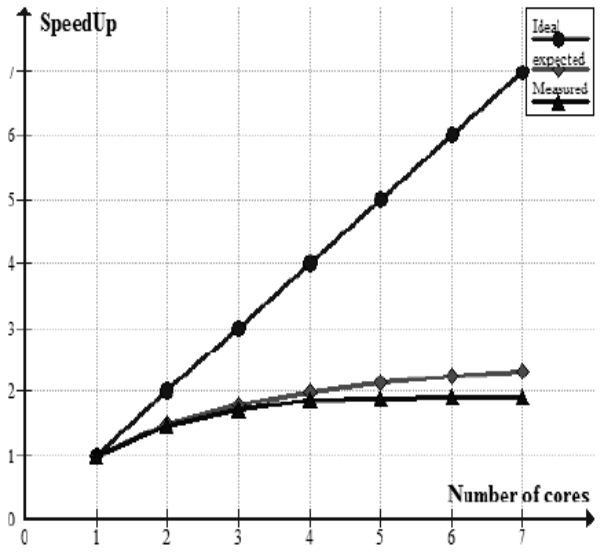


Fig.8. Speedup

Fig. 8 shows an ideal (linear line), expected, and obtained speed-ups. Here the part of the program which can be parallelized is between 65% to 70%. The bottleneck for this algorithm is the variable length coding. An efficient implementation of the VLC will contribute in more speedup. As it can be seen in Fig. 8, the obtained speed-up is very close to the expected speed-up of four processing cores and there is no similar pattern. This is due to the fact that we have only four physical cores and other four cores are available due to the hyper threading support. There is one more important factor which is the communication overhead between different cores of the processor. Since all tasks are not completely independent of each other, inter-core communication is required which slows down the performance. As for as the quality of image is concerned, we just executed the same JPEG algorithm in parallel and did not modified its functionality, hence it was having same quality as if it was encoded on a uniprocessor system.

5. CONCLUSION

A sequential algorithm does not perform better on multicore system until a programmer makes some efforts to parallelize it. In this paper we obtained a sequential implementation of JPEG encoder algorithm and parallelized it with POSIX threads. The speed-up obtained with different number of processor cores is very close to the expected speed-up. Experimental results showed that, a better implementation of variable length coding will improve the speedup.

REFERENCES:

Bradley A. P. and F. W. M. Stentiford (2002) JPEG 2000 and region of interest coding. In *Digital Image Computing Techniques and Applications (DICTA2002)*, Melbourne, Australia.

Castells-Rufas D., J. Joven J. and J. Carrabina (2010) Scalability of a parallel jpeg encoder on shared memory architectures, In *proceedings of the 2010 39th International Conference on Parallel Processing, ser. ICPP '10*, 502–507.

Hashimoto M., K. Matsuo and A. Koike (2005) "JPEG 2000 encoding method for reducing tiling artifacts," IEICE Transactions, vol. **(88-D):** No. 12, 2839–2848.

Khayam S. A., (2003) The discrete cosine transform (DCT): Theory and application (Department of electrical and computing engineering.

Kodaka T., K. Kimura and H. Kasahara (2002) Multigrain parallel processing for jpeg encoding on a single chip multiprocessor, In *proceedings of the International Workshop on Innovative Architecture for Future Generation High-Performance Processors and Systems (IWIA'02), ser. IWIA '02*, Washington, DC, USA.

Kovac M., N. Ranganathan and M. Zagar (1995) A prototype VLSI chip architecture for jpeg image compression," In *proceedings of the 1995 European conference on Design and Test, ser. EDTC '95.*

Nethra Imaging Inc (2007) NI-2065/66. 3.2 megapixel smart camera module image processor with jpeg encoder.

Oh J., S. W. Kim and C. Kim (2003) *OpenMP* and compilation issue in embedded applications, In *proceedings of the OpenMP applications and tools 2003 international conference on OpenMP shared memory parallel programming, ser. WOMPAT'03*, 109–121.

Osorio R. R., C. Daz-Resco and J. D. Bruguera (2009) Highly parallel image processing on a massively parallel processor Array. XX Jornadas de Paralelismo, A Coruna, Spain. 283-288.

Rabbani M. and P. W. Jones (1999) Digital image compression techniques, volume = tt7, Publisher: SPIE press, Bellingham, WA., USA.

Shee S. L., A. Erdos and S. Parameswaran (2006) Heterogeneous multiprocessor implementations for jpeg: a case study. In *proceedings of the 4th international conference on Hardware/software co-design and system synthesis, ser. CODES+ISSS '06*, 217–222.

Xu R., S. N. Pattanaik and C. E. Hughes (2005) High-dynamic range still-image encoding in jpeg 2000, *IEEE Computer Graphics and Applications*, vol. **(25):** No. 6, 57–64.

.