Journal of Theoretical and Applied Information Technology
30th September 2011. Vol. 31 No.2
© 2005 - 2011 JATIT & LLS. All rights reserved·
ISSN: **1992-8645** www.jatit.org E-ISSN: **1817-3195**

PERFORMANCE ANALYSIS OF A PARALLEL IMPLEMENTATION OF GLOBAL MOTION ESTIMATION ON MULTIPROCESSORS

[1]HUSSEIN R. AL-ZOUBI AND [2]RAMI A. AL-NAMNEH

[1] Asstt Prof., Computer Engineering Department, Hijjawi Faculty for Engineering Technology, Yarmouk University, Irbid 21163, Jordan.

[2] Asstt Prof., Department of Software Engineering, Faculty of Computer and Information Technology Jordan University of Science & Technology, Irbid 22110, Jordan.

Emails: [1]halzoubi@yu.edu.jo, [2]ramir11@just.edu.jo

ABSTRACT

Video compression is an important field in our daily life. Motion inside videos can be classified as local or global. The two techniques: motion estimation and compensation are used in the modern compression and decompression standards. However, these two techniques require high computational power. This mandates the need for efficient methods to make video coding faster. To this end, we propose a parallel implementation for global motion estimation and present a performance analysis on shared memory multiprocessors.

Keywords: *Video Compression, Global Motion Estimation (GME), Global Motion Compensation (GMC), Parallel Implementation, Levenberg-Marquardt Algorithm (LMA).*

1. INTRODUCTION

Video processing is one of the most important research fields in today's world for its large number of applications including remote vision, video conferencing, and video surveillance [1]. Video processing deals with the manipulation of visual data in order to analyze, compress, or segment them.

The video represents an image sequence (e.g. 30 pictures/second), where each image is a rectangular matrix of picture elements, or pixels. Typically, each pixel requires 24 bits of storage space (8 bits for red, 8 bits green, and 8 bits for blue in the RGB system). This imposes very high storage space requirements for video. Fortunately, video compression can help. However, most video compression techniques are lossy, which means part of the data will be lost by compression.

The motion in a video can in general be classified as local and global motion. Global motions are caused by the motion of the camera like pan, tilt, and zoom. Local motions, on the other side are caused by the movement of individual objects inside the scene. In light of this, any video compression technique should consider these types of motion. Therefore, global motion estimation and compensation are two fundamental methods used in most modern video compression standards [2].

In order to estimate and compensate for global motions, transformation motion models are used like the affine and perspective models [2], as well be illustrated in the next section. Global motion is estimated between the current and the previous images (frames) using a motion model. The output of motion estimation is a set of parameters. For motion compensation, the model parameters and the previous frame are used to generate the current compensated frame. Because the compression is lossy, there will be an image difference between the current and the compensated frames. The overall process is illustrated by the data flow model shown in Figure 1.

Journal of Theoretical and Applied Information Technology
30th September 2011. Vol. 31 No.2
© 2005 - 2011 JATIT & LLS. All rights reserved·
ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Because of the intensive computations needed for motion estimation and compensation, long times are required for compression and decompression. Parallel computing has been used in many applications [5-6]. To reduce the time of global motion estimation and compensation, it is possible to parallelize the operations performed on the images (which are treated as matrices). In this paper, we provide performance analysis of a parallel Implementation of global motion estimation on multiprocessors. To the best of our knowledge, we think that this work is the first to address the parallelization of global motion estimation and compensation.

The rest of the paper is organized as follows: In Section 2, we provide the algorithm specification of the global motion estimation and how to estimate the parameters of the perspective motion model using the Levenberg-Marquardt algorithm (LMA). Next, in Section 3, we illustrate our parallel implementation of the global motion estimation. In Section 4, details the simulation results. Finally, Section 5 concludes the paper.

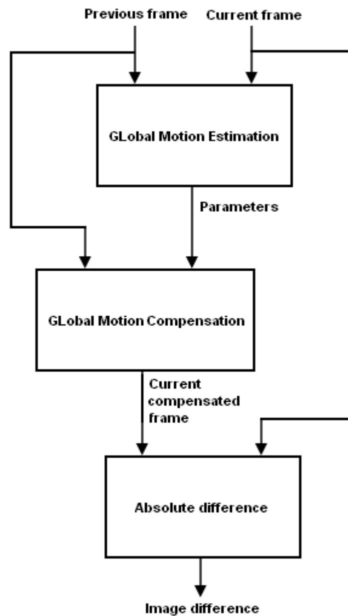

Figure 1: Data Flow model

2. ALGORITHM SPECIFICATION

If one has two consecutive image frames in a video, say I_k (previous frame) and I_{k+1} (current frame) and want to apply global motion estimation on them to produce a predicted image of the current frame $\tilde{I}_{k+1}$, it is desirable to minimize the difference between I_{k+1} and $\tilde{I}_{k+1}$. That is, the goal when performing global motion compensation is to minimize the following sum of squared differences

$$E = \sum e^2(i,j), \quad (1)$$

where

$$e(i,j) = I_{k+1}[i,j] - \tilde{I}_{k+1}[i,j]$$
$$= I_{k+1}[i,j] - I_k[x(i,j), y(i,j)]. \quad (2)$$

where i, j are indexes of the current frame and x, y are indexes in the previous frame obtained by transforming pixels in I_k. Many models can be used for transformation. The perspective motion model, which consists of eight motion parameters, m_1 through m_8, is one of the most popular and was adopted by the MPEG-4 video coding standard [2]. The transformation carried by the perspective model can be obtained by the following equations:

$$x(i,j) = \frac{m_1 i + m_2 j + m_3}{m_7 i + m_8 j + 1} \quad (3)$$

$$y(i,j) = \frac{m_4 i + m_5 j + m_6}{m_7 i + m_8 j + 1} \quad (4)$$

The eight motion parameters, m_1 through m_8 of the perspective model can be evaluated using the Levenberg-Marquardt algorithm (LMA) [3], which iteratively minimizes E in (1). At iteration (n+1) of the LMA, the motion vector $\boldsymbol{m} = [m_1, m_2, \dots, m_8]$ is updated by:

$$\boldsymbol{m}^{(n+1)} = \boldsymbol{m}^{(n)} + \boldsymbol{s}^{(n)}, \quad (5)$$

where the update $\boldsymbol{s}^{(n)}$ can be found after solving the liner equation

$$[\boldsymbol{J}^T(\boldsymbol{m}^{(n)})\boldsymbol{J}(\boldsymbol{m}^{(n)}) + \mu^{(n)}\boldsymbol{I}]\boldsymbol{s}^{(n)} = -\boldsymbol{J}^T(\boldsymbol{m}^{(n)})\boldsymbol{r}(m^{(n)}), \quad (6)$$

where $\boldsymbol{I}$ is the identity matrix, and μ is a non-negative scalar parameters. $\boldsymbol{r}(\boldsymbol{m})$ is a column vector given by:

$$\boldsymbol{r}(\boldsymbol{m}) = [e(1,1), e(1,2), \dots, \text{all } i,j] \quad (7)$$

$\boldsymbol{J}(\boldsymbol{m})$, as given in (8), is the jacobian matrix of $\boldsymbol{r}(\boldsymbol{m})$. That is:

Journal of Theoretical and Applied Information Technology
30th September 2011. Vol. 31 No.2
© 2005 - 2011 JATIT & LLS. All rights reserved
ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

$$\boldsymbol{J(m)} = \begin{bmatrix} \frac{\partial e(1,1)}{\partial m_1} & \frac{\partial e(1,1)}{\partial m_2} \cdots & \frac{\partial e(1,1)}{\partial m_8} \\ \frac{\partial e(1,2)}{\partial m_1} & \frac{\partial e(1,2)}{\partial m_2} \cdots & \frac{\partial e(1,2)}{\partial m_8} \\ \cdots & \cdots \quad \cdots & \text{all } i,j \end{bmatrix} \quad (8)$$

The entries of this Jacobian matrix can be evaluated using the following chain rule:

$$\frac{\partial e(i,j)}{\partial m_k} = \frac{\partial e(i,j)}{\partial x}\frac{\partial x}{\partial m_k} \quad (9)$$

$$\frac{\partial e(i,j)}{\partial m_k} = \frac{\partial e(i,j)}{\partial y}\frac{\partial y}{\partial m_k} \quad (10)$$

Table 1 shows what values $\frac{\partial x}{\partial m_k}$ and $\frac{\partial y}{\partial m_k}$ can take, where $D = m_7 i + m_8 j + 1$

And finally,

$$\frac{\partial e(i,j)}{\partial x} = (1 - y_f)(I_k[x_i + 1, y_i] - I_k[x_i, y_i])$$

$$+y_f(I_k[x_i + 1, y_i + 1] - I_k[x_i, y_i + 1]). \quad (11)$$

Similarly

$$\frac{\partial e(i,j)}{\partial y} = (1 - x_f)(I_k[x_i, y_i + 1] - I_k[x_i, y_i])$$

$$+x_f(I_k[x_i + 1, y_i + 1] - I_k[x_i + 1, y_i]). \quad (12)$$

where x_i, y_i are the integer part and x_f, y_f are the fractional part of coordinates x and y, respectively.

3. THE PARALLEL IMPLEMENTATION

The serial implementation to perform global motion estimation on two consecutive images consists of the following steps:

1. Input two consecutive frames.
2. Vector and matrix calculations:
   a. Calculation of $\boldsymbol{r(m)}$ vector.
   b. Calculation of $\boldsymbol{J(m)}$ matrix.
   c. Calculation of $\boldsymbol{J^T(m)J(m)}$ matrix.
   d. Calculation of $\boldsymbol{J^T(m)r(m)}$ vector.
3. Gaussian algorithm:
   a. Solve for $\boldsymbol{s}$ using Equation (6).
4. Update the value of vector m using Equation (5).

In light of the above steps and using the shared memory approach to parallelize the global motion estimation, our focus was on parallelizing steps 2 and 3. For matrix multiplication, shared address space was used in the parallel algorithm, where matrices were divided into blocks of rows. Blocks were assigned to threads and each thread was responsible for multiplying its share, an example can be shown in Figure 2 below. Data decomposition was used in this project because there is no data dependency.

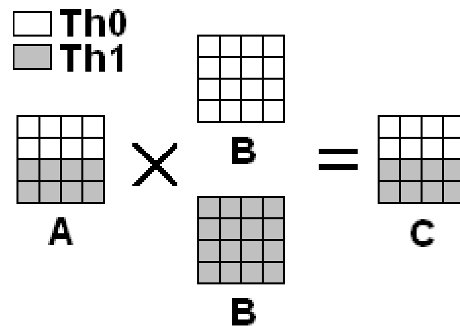

Fig 2: Parallel matrix multiplication

In Gaussian elimination, threads should process rows sequentially because there is dependency in data, which in turn requires that all threads should finish their tasks before going to the next row, which is called "barrier" as shown in figure 3.

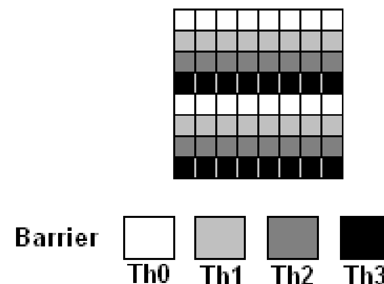

Fig 3: Parallel Gaussian elimination

4. SIMULATION RESULTS

The simulator and cross-compiler binaries are compiled for an IA32 native machine running Linux. Therefore, Linux was installed on a

© 2005 - 2011 JATIT & LLS. All rights reserved.
ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

machine with a processor based on an IA32 ISA. The cross-compiler is based on gcc-3.2 and is used to compile C/C++ source code into MIPS binaries for use with the simulator. The simulator that we have used in the experiments is based on SESC (sesc.sourceforge.net), which is an advanced execution-driven simulation environment that supports a dynamic superscalar processor model in which every component of the architecture is modeled cycle by cycle. It supports both uniprocessor and multiprocessor architectures.

The execution time in terms of the number of clock cycles is shown in Table 1, where we have used 5 image sizes: 32×32, 64×64, 128×128, 256×256, and 512×512 pixels. Table 1 shows the number of clock cycles required to complete GME when 1, 2, 3, 4, 5, 6, 7, and 8 threads were used. it is obvioues that the number of cycles decrease as the number of threads increase.

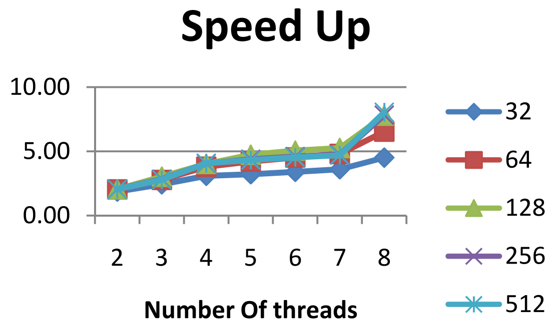

Fig 4: Speed up

The obtained speedup is shown in Figure 4. The relation should be theoretically linear and should express the following relation:

$$Optimal\ speedup = number\ of\ threads \quad (13)$$

Practically, the relation was not linear in all cases. The reason is that an 8×8 matrix when solved by the Gaussian elimination method, unbalanced distribution will show up:

$$8\ mod\ (number\ of\ threads) \neq 0 \quad (14)$$

To illustrate, consider the case when seven threads are utilized, so that one of the threads (say thread 0) processes two rows while the other threads each will process one row which means that these threads will wait until (thread 0) finishes its task . The efficiency is defined as follows:

$$Efficiency = \frac{Speedup}{Number\ of\ threads} \quad (15)$$

Because of the non-linear speed up, the efficiency is affected as shown by Figure 5. We can notice from Figure 5 that there are some values of efficiency higher than one, these values represent cases called "super speed up". Super speed up case occurs when:

$$Speedup > Number\ of\ thread \quad (16)$$

This is due to cashing and cash prefetching. The only limitation in our algorithm is that our algorithm generates an 8×8 matrix which cannot be parallelized by more than 8 threads. In other words, we cannot use more than 8 threads (processes) to process this matrix of size 8×8.

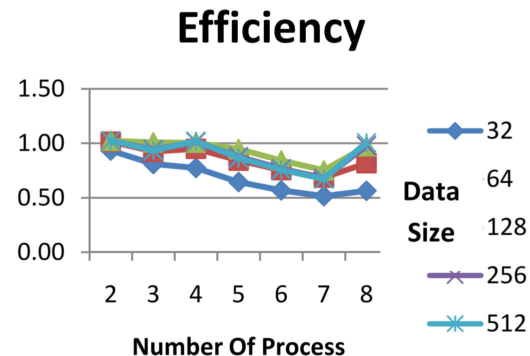

Fig 5: Efficiency

5. CONCLUSIONS

In this paper, we have implemented global motion estimation in shared memory multiprocessors and obtained high speed up using different image sizes. The results show that, for the same data sizes, as we increase the number of threads, the clock cycles decrease. Moreover, the results (figures and tables) show that we can get linear speed-up when we use large data sizes (larger than 512). However, practically, the relation was not linear in all

© 2005 - 2011 JATIT & LLS. All rights reserved
ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

cases; this is due to unbalanced distribution of threads when we use power of two data-sizes and odd threads (data size mod (number of threads) ≠ 0). For example, if the data size is 8 and number of thread are 3, two threads process three rows and the third thread processes two rows (unbalanced load). Another thing that worth to be mentioned is that the efficiency is more than 100% in some cases, since cashing and pre-fetching using shared memory multiprocessors reduce the cash misses and decrees the total time required to parallelize our code .

REFERENCES

[1] H. Yin, C. Du, C. Ren, Z. Chen, H. Min, and C. Lin, "A secure and scalable video conference system based on peer-assisted content delivery networks," ," *International Journal of Computer Systems Science and Engineering*, vol. 24, no. 5, September 2009.

[2] F. Dufaux and J. Konrad, "Efficient, robust, and fast global motion estimation for video coding," *IEEE Trans. Image Processing*, vol. 9, no. 3, March 2000, pp. 497-501.

[3] MPEG-4 video verification model version 18.0, in: ISO/IEC JTC1/SC29/WG11 N3908, Pisa, Italy, 2001.

[4] W. Press, S. Teukolsky, W. Vetterling, and B. Flannery, *Numerical Recipes in C: The Art of Scientific Computing*, Second Edition, Cambridge University Press, 2002.

[5] D. M. Quan, J. Altmann, and L. T. Yang, "Improving the capability of the SLA workflow broker with parallel processing technology," *International Journal of Computer Systems Science and Engineering*, vol. 24, no. 5, September 2009.

[6] Y. Tang, Y. Zhang, and H.Chen, "A parallel shortest path algorithm based on graph-partitioning and iterative correcting," *International Journal of Computer Systems Science and Engineering*, vol. 24, no. 5, September 2009.

AUTHOR PROFILES:

Dr. **Hussein R. Al-Zoubi** received his MSE and Ph.D. in Computer Engineering from the University of Alabama in Huntsville, USA in 2004 and 2007, respectively. Since 2007, he has been working as an assistant professor with the Department of Computer Engineering, Hijjawi Faculty for Engineering Technology, Yarmouk University, Jordan. His research interests include Image Processing, Video Coding, Machine Learning, Computer Architecture, Wireless Networks, and Parallel Processing.

Dr. **Rami A. Al-Namneh** received his MSE and Ph.D. in Computer Engineering from the University of Alabama in Huntsville, USA in 2003 and 2006, respectively. Since 2006, he has been working as an assistant professor with the Department of software Engineering, Faculty of Computer Information Technology, Jordan University of Science and Tecnology, Jordan. His research interests include Parallel algorithms and parallel programming.

© 2005 - 2011 JATIT & LLS. All rights reserved·

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Table 1: Partial derivatives of x and y with respect to motion parameters.

k	1	2	3	4	5	6	7	8
$\frac{\partial x}{\partial m_k}$	$\frac{i}{D}$	$\frac{j}{D}$	$\frac{1}{D}$	0	0	0	$-\frac{xi}{D}$	$-\frac{xj}{D}$
$\frac{\partial y}{\partial m_k}$	0	0	0	$\frac{i}{D}$	$\frac{j}{D}$	$\frac{1}{D}$	$-\frac{yi}{D}$	$-\frac{yj}{D}$

Table 2: Execution time in cycles

	1	2	3	4	5	6	7	8
32	1520194	813483	625295	490984	471705	445942	421328	336632
64	5619336	2766428	2026344	1478712	1334595	1241813	1170898	858718
128	42715747	20864745	14110953	10623504	9021732	8452800	8124291	5490458
256	170645659	83497328	60379490	42094571	38908211	37094441	35905198	21817401
512	681934334	332834337	243708431	167034859	157573411	150194605	144945089	84762585