Indian Journal of Science and Technology, *Vol 8(13), 59410, July 2015*
ISSN (Print) : 0974-6846
ISSN (Online) : 0974-5645

Parallel Implementation of Singular Value Decomposition (SVD) in Image Compression using Open Mp and Sparse Matrix Representation

J. SairaBanu, Rajasekhara Babu and Reeta Pandey

School of Computing Science and Engineering, VIT University, Vellore-632014, India

Abstract

The advent of Multi-core processors has offered powerful processing capabilities and provided new avenues for parallel processing. As the traditional methods of computation are inherently based on single core of execution, they are not capable of taking the advantage of high computational power offered by multi core processors, even if available. Singular Value Decomposition (SVD) has been proven well for many image processing techniques such as image compression, quality measure and in watermarking. SVD is a highly compute intensive algorithm that applies numerous matrix operations to an image such as transpose, inverse, multiplication of high orders to form a compressed image. However accelerating the SVD routines to exploit the underlying hardware poses a significant challenge to the programmers. This paper deals with improving the speedup of SVD algorithm used in image compression technique. This is achieved by identifying the areas where data parallelism can be applied and parallel programming model OpenMp is used to run the parallel components on multiple processors. This results in faster execution of the algorithm and hence reduces the execution time when applied to image compression for large images. This paper also addresses the space overhead of storing the matrices of SVD technique, by adapting efficient sparse matrix storage format such as Compressed Row Storage (CSR). Experimental results show that the speedup achieved through OpenMp is around 1.15 and better compression ratio with sparse matrix storage format.

Keywords: CSR, Image Compression, SVD, OpenMp, Sparse Matrix

1. Introduction

A digital image is stored in a computer in the form of a two dimensional array of pixels or picture elements, depending on the height and width of the image. Each pixel in turn, is a combination of the varying intensities of red, green and blue colors which are commonly stored in the form of triplets of 8 bit each in the computer memory. Thus an image size is dependent on the number of pixels it is having which is a function of the dimensions of the image. In general, the space complexity of an image is O (MN) where the M is the height of the image and N is the width of the image. Image compression techniques aim towards reducing the space complexity for the image. Thus less space is required for storing and transmission of an image. In this paper we have used Singular Value Decomposition as an image compression technique and compared the error in the images at different levels of compression. Singular Value Decomposition is a concept of linear algebra which factorizes a real matrix A(MxN) into components U, S and V such that the following relation is satisfied: A = U*S*V'. This relation is the singular value decomposition of A where U is of dimension MxM, S is a diagonal matrix of dimension MxN in which non-diagonal elements are zero, also called the singular values of A and V is of dimension NxN. SVD allows us to write the matrix A as a summation of rank one matrix in the manner shown below: A= U_1*S_1 *V_1' + U_2* S_2*V2'……U_N*S_N*V_N'. The terms of the diagonal matrix S are sorted in value such that S_1> S_2> S_3…>S_N. Therefore we can reconstruct the matrix A completely if we have the complete matrices U, S and V. However, if we remove some of the lower diagonal

* *Author for correspondence*

elements of S, we can still get a good approximation of A. The fact that we can approximate the matrix A even with lesser elements in S is the basis for image compression using SVD. The approximation becomes more accurate with the increase in number of singular values used. As a result of quantization, it is common to have matrices that have a large number of zeroes than the non-zero values. An elegant method of storing such matrices is the use of sparse matrix storage formats. In this approach, we store only the non-zero elements of a matrix thereby reducing the total storage needed for the original matrix. This also helps in reducing the amount of data that needs to be transmitted for communication across computers.

The growth of computing power has also been a fast paced one. Today we have high processing power available in the form of multi-core processors such as the Intel multi-cores processors. In order to fully tap the potential of multiprocessing; calls for a new approach towards programming where we distribute the processing load to different cores thereby reducing execution time and producing faster results. In this paper we have used OpenMp for achieving distribution of workload to different cores. Open Multiprocessing, commonly known as OpenMp is a shared memory multiprocessing paradigm. We have used the OpenMp in C++ for parallel execution of sections of the code which require high performance. OpenMp is based on a fork and join model where a single thread forks into multiple threads for performing the parallel tasks and after the execution of the parallel region, it resumes as a single thread. The performance gain offered by OpenMp comes from the fact that the multiple threads can execute concurrently on different cores in a multi-core processor. OpenMp works best when there is no data dependency among the parallel threads.

In this paper we have used SVD routine implemented in GNU Scientific Library for image compression and parallelized the routines and achieved around 1.15% speedup using dual core Intel processor. We preferred GSL rather than INTEL MKL library because it is an open Source which in turn uses LAPACK and BLAS library. The rest of the paper is organized as follows: Section 2 presents the related work, Section 3 describes the SVD technique and its application in image compression section 4 deals with sparse matrix storage formats, Section 5 explores the OpenMp programming model for SVD, Section 6 presents the results of SVD and Sparse storage format for image compression, Section 7 talks about the conclusion.

2. Related Work

SVD routines which are used in many scientific or engineering applications such as Image processing, Image compression, Data clustering, etc. have been implemented in many scientific libraries like LAPACK, GNU Scientific library, INTEL MKL library etc. Chaitanya Gunta et al[6] made an attempt to accelerate the SVD routines in LAPACK scientific Library. Sheetal Lahabar et al[10] proposed parallelizing SVD routines for GPU using CUDA programming. Michal W. Berry et al[2] provided a comprehensive study on SVD for dense and sparse matrices which are suitable for parallel computing platforms. Sivashankaran Rajamanickam[12] has worked in efficient algorithm for sparse singular value decomposition. Mostafa I. Soliman[16] has proposed a new algorithm for computing the singular value decomposition on one side Jacobi based techniques. The above referred papers explore the way to accelerate the SVD technique. Rowayda A. Sadek[15] has conducted an experimental survey on SVD as an efficient image processing applications. Prasantha et al[11] adapted SVD as an image compression technique and performed experimental study using MATLAB. Awwal Mohammed Rufal et al[14] presents a new lossy image compression techniques which combines both SVD and wavelet difference reduction (WDR) to increase the performance of image compression. Vasil Kolev et al[8] has applied SVD for images from scanned photographic plates. Mahendra M. Dixit[4] worked on adaptive SVD algorithm for 2D/3D still image compression application. Z hongxiaojia[7] concerned about accurate computation of SVD using cross product matrices. Taro Konda et al[9] presented the double Divide and Conquer algorithm (dDC) for bidiagonal SVD which performs well in speed, accuracy and orthogonality compared to the standard algorithms such as QR and Divide and Conquer. Literature review reveals that SVD is used as one of the image compression techniques. In this paper, the SVD technique is accelerated and it is employed for image compression application and the space overhead is addressed with the help of efficient sparse matrix storage format.

3. Singular Value Decomposition (SVD)

SVD is an approach of advanced linear algebra. It is based on the packing the maximum energy of a signal into a lesser number of coefficients. It is an effective method to split a matrix into linearly independent constituents where each constituent has its own contribution in terms of energy. The uses of SVD are diverse ranging from areas such an image processing, latent semantic analysis, approximation of the pseudo inverse of a matrix, least square minimization of a matrix, efficient medical imaging, topographical analysis, watermarking schemes and many other areas. In the case of image compression, SVD offers its advantage in the form of its sensitivity to local adaptations in the statistics of an image. The core mathematical foundations of SVD can be summarized as factorizing a matrix A into three components U, known as the matrix of rows, S called the diagonal matrix or the singular values of A and V is called the matrix of columns. These factors of the matrix satisfy the relation $A = U^*S^*V^T$. For a given Matrix A of size MxN the output of SVD has the following components. U: a matrix of dimension MxM. S: the diagonal matrix of dimension MXN, V: a matrix of dimension NxN and V^T represents the transpose of the matrix V. To understand the process of SVD let us take an example of a 2x2 matrix

$$A = \begin{bmatrix} 2 & 2 \\ -1 & 1 \end{bmatrix}$$

The transpose of A, is $A^T = \begin{bmatrix} 2 & -1 \\ 2 & 1 \end{bmatrix}$.

The first step in SVD involves calculating

A^*A^T and $A^T{}^*A$, which as:

$$A^*A^T = \begin{bmatrix} 8 & 0 \\ 0 & 2 \end{bmatrix} \text{ and } A^T{}^*A = \begin{bmatrix} 5 & 3 \\ 3 & 5 \end{bmatrix}$$

Further, we solve for the Eigen vectors of A such that

$|A^*A^T - \lambda I| = 0$.

The values of the Eigen vectors are determined as $\lambda 1 = 8$ and $\lambda 2 = 2$. Upon finding the Eigen vector of A^*A^T, we can easily determine the diagonal matrix S. The singular values of A is defined as the square root of the roots the Eigen vector of A^*A^T.

Thus we have, $S = \begin{bmatrix} \sqrt{8} & 0 \\ 0 & \sqrt{2} \end{bmatrix}$

In the next step we determine the values of the matrix U as follows: to determine the columns of U, we solve for the Eigen vectors of A^*A^T. This should satisfy the relation, $[A^*A^T - \lambda I][x] = 0$. Using the Eigen values, we get the Eigen vectors as $X1 = \begin{bmatrix} -1 \\ 0 \end{bmatrix}$ and $X2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$. To get the columns of U, we determine the unit Eigen vectors of u1 and u2. Thus we determine U as $U = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$. To determine the columns of V we first determine the Eigen vectors of the relation $[A^T{}^*A - \lambda I][x] = 0$. Further we calculate the unit Eigen vectors v1, v2 to determine the columns of V in a similar manner and have $V = \begin{bmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix}$

Thus we have the matrices U, S and V and it can be show that the matrix A has been factorized into the three matrices U, S, V: it can be seen that the relation $A = U^*S^*V^T$ holds good

$$\begin{bmatrix} 2 & 2 \\ -1 & 1 \end{bmatrix} = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} * \begin{bmatrix} \sqrt{8} & 0 \\ 0 & \sqrt{2} \end{bmatrix} * \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix}$$

$$A = U * S * V^T$$

SVD computation involves sequence of vector operations, matrix to matrix and matrix to vector multiplication[13,20]. This feature makes SVD computation a good candidate for parallelization.

3.1 SVD Algorithm in GSL

Generally there are different algorithms for computing SVD such as Golub–Reinsch, High Relative Accuracy Bidiagonal SVD, square root-free algorithm, bisection method, divide and conquer method, one-sided Jacobi method for SVD and biorthogonalization. In GNU scientific Library SVD is implemented

using Golub-Reinsch SVD, modified Golub-Reinsch SVD and one-sided Jacobi orthogonalization. GNU library uses thin version of SVD, a common format with *U* as *M*-by-*N* orthogonal matrix. Full SVD is defined as *U* as an *M*-by-*M* orthogonal matrix and *S* as an *M*-by-*N* diagonal matrix (with additional rows of zeros). Here we conducted experiments on image compression using SVD algorithm both in MATLAB as well as in C using GNU Scientific library. Golub Reinsch algorithm is a most efficient, popular and numerically stable technique for computing SVD of an arbitrary matrix. It is also well suited for multicore and SIMD GPU architecture. GolubReinsch algorithm[21] is also used in LAPACK package. It has two distinct steps such as transforming the given matrix into bidiagonal forms using series of householder transformations and followed by an iterative procedure designed to use orthogonal transformations to produce diagonal matrices that are successively more diagonal. Running time of SVD is $O(mn^2)$.

3.2 GolubReinsch Algorithm

Step 1: Bidiagonalization of A to B

$B \leftarrow Q^T AP$ [A is the original matrix, B is a diagonal matrix and Q and P are unitary householder matrices]

Step2: Diagonalization of B to S

$S \leftarrow X^T BY$ [The matrix B is obtained from the step 1, Σ is a diagonal matrix; X and Y are orthogonal unitary matrices].

Step 3: Compute orthogonal matrix U and V

$U \leftarrow QX$

$V^T \leftarrow (PY)\ T$

Step 4: Compute SVD of A

$A = U\Sigma V^T$

3.3 SVD in Image Compression

The objective of image compression is to represent an image with lesser amount of data than what an image is composed of and the ability to reconstruct the image from its smaller representation. This improves the storage efficiency of an image and also greatly reduces the amount of data that is required to transmit the image across computers[1,5,19]. However, the image formed from its compressed image by an image processing algorithm may or may not be able to recreate the exact copy of original image. A compression technique can be lossy or lossless based on the quality of image it restores. A lossless compression scheme can reconstruct the exact copy of an image whereas a lossy scheme can recreate the image with some data loss, depending on the compression technique used. We have used SVD as lossy image compression scheme. The other methods for image compression are discrete 9/7 biorthogonal wavelet transform, discrete cosine transform, Karhunen-Lohve transform, and combinations of these. The reason why we have used singular value decomposition is it is basic, simple and works almost for all kind of matrix and it is well suited for image compression. As images are stored in the form of matrices in the computer memory, it is imperative to think an image as a matrix[17]. Depending on amount of type of image, colored or grayscale, the space required to store an image depends on the dimension of the image. A grayscale image has the space requirement of mxn where m and n denote the height and the width of the image whereas a colored image has the space requirement of mxnx3, as there are 3 matrices of mxn each representing the colors red, green and blue commonly known as the RGB image. An illustration of both these schemes has been shown in Figure 1.

From the properties of SVD it follows that a matrix A can be represented in the form of its SVD components as a sum of rank 1 matrices of the form:

$$A = U_1 * S_1 * V_1^T + U_2 * S_2 * V_2^T \ldots\ldots\ldots\ldots Un * S\ n * Vn^T$$

In the above relation, it is worth mentioning that the value of $S_1 > S_2 > S_3 > \ldots S_n$. The above relation also implies that, the contribution of the first component of the sum would be highest while the contribution of the last component would be lowest. Thus, it follows

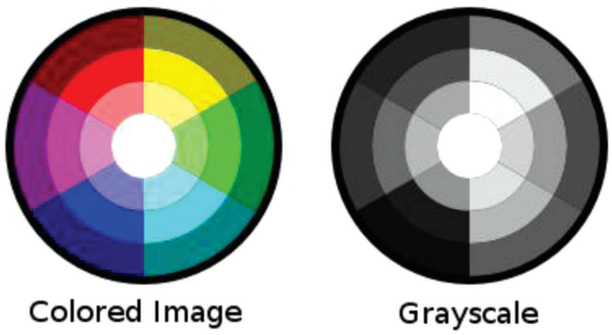

Figure 1. Colored and Grayscale Image.

that if we consider only the first r members of the above summation, we can still get a considerable approximation of A. This is the property used for SVD based image compression[3,19]. The relation for the compression of an image considering the first r singular values can be show to be:

$$A_r = U_1 * S_1 * V_1^T + U_2 * S_2 * V_2^T \ldots\ldots\ldots\ldots U_n * S_r * V_r^T$$

Here A_r represents the approximation of the image based on the first r singular values of the singular matrix S. Thus, instead of storing the matrix A of size MxN, we can store the matrices U_{mxr},V_{nxr} and the singular vector S_r and reconstruct the image as: $A_r = U_{mx}r * S_r * V^T_{nxr}$. Thus, it leads to a reduction in the amount of space needed to store the image and the space complexity of the compressed image would be given by $A_r = r\ (m + n + 1)$. Depending on the value r of the rank of SVD selected, we can get a compression that would be defined as:

$$\text{Compression ratio } Cr = \frac{m * n}{r(m+n+1)}$$

Mean square error (MSE) = $\Sigma\ (Oij\text{-}R_{ij})^2/mn$ where O represents the original image and R represents the reconstructed image of dimension mxn

Peak signal to noise ratio (PSNR) = 10 log (255^2/Mean Square Error)

4. Sparse Matrix Storage Formats

To reduce the storage requirements of the image we have used sparse matrices. A sparse matrix is one which has lot of redundant information in the form of zero values. Some of the common techniques of storing a spares matrix are diagonal format, ELLPACK, Coordinate format, Compressed Row Storage (CSR), Compressed Column storage CCS, row grouped CSR, Blocked compressed Row storage, Quad tree format, Combination of CSR and quad tree format, Minimal QCSR format and unified SELL-C^{σ}. We have used CSR format in our implementation. In CSR method a matrix is stored in the form of three vectors: A: the vector of Non-Zero Elements, C: the vector of indices Columns of Non-Zero Elements and R: the vector indices of first Non-Zero element in each row. This is the most popularly used format, as it is both a general purpose format, and is very efficient. There have been a lot of other improved versions of this format, like the blocked CSR and the row-grouped CRS (described later), that have been introduced and implemented. The size of the matrices is purely dependent on the number of non-zero elements in the matrix, and hence no matter how many zeroes are padded, the size occupied will remain the same. The row pointer array facilitates fast multiplication, and the code remains unchanged for Sparse Matrix Vector multiplication SpMV[6]. SVD algorithm used for image compression involves matrix matrix multiplication and matrix vector multiplication. The unnecessary zero in these matrices is removed by representing these matrices in efficient sparse matrix storage format such as CSR.

For example:

$$M = \begin{bmatrix} 0 & 4 & 0 & 7 & 0 \\ 2 & 0 & 3 & 0 & 6 \\ 0 & 5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 \\ 1 & 0 & 0 & 6 & 0 \end{bmatrix}$$

A = [4 7 2 3 6 5 2 1 6]
C = [1 3 0 2 4 1 4 0 3]
R = [0 2 5 6 7 9]

5. OpenMp

OpenMp (Open Multiprocessing) software is an Application Programming Interface (API) that supports multi-platform shared memory architecture. It works with fork-join method. The compiler, on receiving the code, generates the multi-threaded version of this code, using the directives. Each thread is given to a separate core, where they are executed simultaneously. In the end, the main thread gives the result. It is preferred here over the other available software because of various reasons; firstly, it has portable multithreaded code, and has unified code for both sequential and parallel implementation (the OpenMp constructs are treated as comments when run sequentially). Secondly, it is very simple to use as it does not deal with message passing, unlike Message Passing Interface (MPI). Finally, the complier directives that are used for achieving parallelism can be easily embedded in C/C++ source

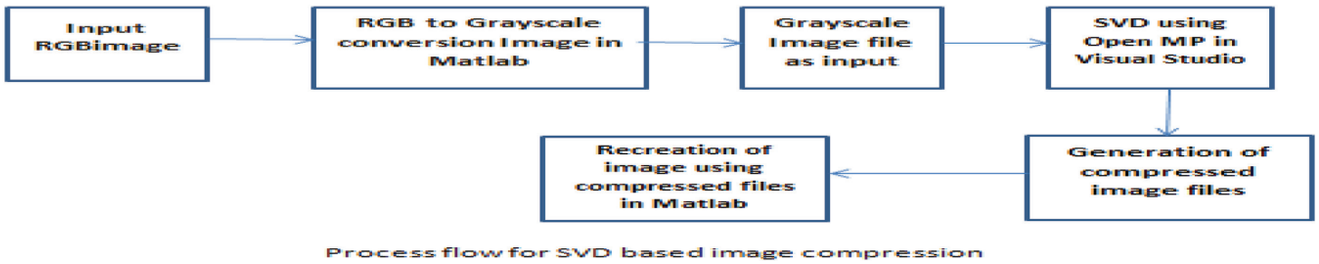

Figure 2. Flow diagram for SVD based image compression.

code. OpenMp accomplishes parallelism exclusively by means of threads. It is an explicit programming model and allows the user full control over parallelization. Here, we use OpenMp in Windows environment, which requires Visual Studio. To parallelize a program using OpenMp, first, the code is divided into two parts- the part that can be parallelized, and the part that cannot. The part that can be parallelized is then examined and dependent variables are identified. Based on these observations, various directives are used to parallelize various parts of the code. In the implementation of SVD, the parallelization is done with OpenMp using the "pragma" directive and special care is taken for variable sharing.

5.1 Implementation Details

Figure 2 shows the steps followed in SVD based Image compression. A colored image is represented in the RGB scheme as a 3 matrices of order MxN each containing the intensities of different shades of Red Green and Blue. The input RGB image that is taken has a JPG file format. The file also contains the metadata required for JPEG files which has to be removed. For processing of the image; we remove the JPEG header from the file and convert it to its corresponding Gray scale version. This conversion is done using Mat lab and the generated image is an MxN gray scale image. This gray scale image is written to a binary file on which further processing to be done. RGB to Gray scale conversion: the JPG file is read in Matlab using the imread function. This function reads the jpg image and converts it into a matrix of the order MxNx3, where each matrix is the representation of the colors corresponding to Red, Green and Blue. To convert this image to its gray scale version, we use the function rgb2gray. This function converts the RGB to gray scale

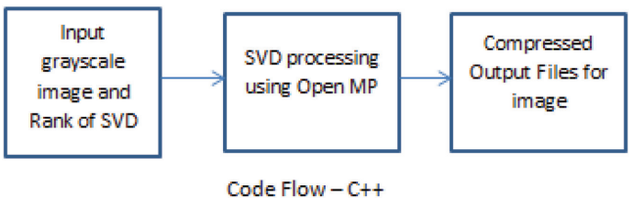

Figure 3. Processing of image.

using the relation: Gray scale intensity = 0.2989*red+ 0.5870*green + 0.1140*blue. This function converts the MxNx3 image to an MxN gray scale matrix. This matrix is further written to a binary file which can be used for further processing.

5.2 Singular Value Decomposition using OpenMp

OpenMp is a parallel, scalable and highly portable programming model based on a shared memory concept. The rise in use of OpenMp has been primarily due to the availability of multi-core processors. Open MP gives the power to distribute work to different cores in a processor and share the load of one processor to many processors. We have used OpenMp for the faster execution of the sections of code which can be scheduled concurrently. Figure 3 depicts the processing of Image. The application for image compression has been developed in visual C++ using OpenMp constructs for parallelizing the code. The motivation for using OpenMp for SVD comes from the fact that with the increase in the size of the image a high computation power is needed. Using OpenMp on a multi-core processor can help reduce the execution time. Also, the code is highly portable and most of the compilers today have the support for OpenMp. The following flow diagram explains the flow of the code:

The application takes the input in the form the grayscale binary file, the dimensions of the image (row and columns) and also the rank for the SVD computation. With these inputs, the SVD processing is done by sharing the work among the cores and the final output in the form of compressed files is generated.

5.3 Generation of Compressed Image Files

By using sparse matrices to represent the final output and using the 'thin' version of SVD, we are able to achieve compression and the resulting images has a reduced space complexity as compared to the input image file from O(mn) to O (r(m+n+1)).

5.4 Reconstruction

Reconstruction is carried out in Matlab by reading the output files generated by the C++ application and displaying the gray scale image using imshow function of Matlab.

6. Results

We have taken three different colored images of different sizes. The images are first converted to the gray scale images and then SVD is performed on them. The performance of SVD in these images vary based on the type of image as shown in Figure 4 three images that have been considered here are Fruits (320x240) Human Face (550x500) and Room (610x423). Figure 4a, 4b and 4c shows the reconstruction of images with different rank values. The accuracy of the image compression and reconstruction is measured by the PSNR and the Mean Square Error and it is shown in Figure 5 and 6. Performance gain is measured by the Speedup that is achieved. Speedup for a parallel architecture is defined as Speedup = Taken by Serial Code/Time taken by Parallel Code. Table 1, 2 and 3 show execution time of sequential and parallel version of SVD for various images and Figure 7, 8, and 9 shows the graph of the execution time.

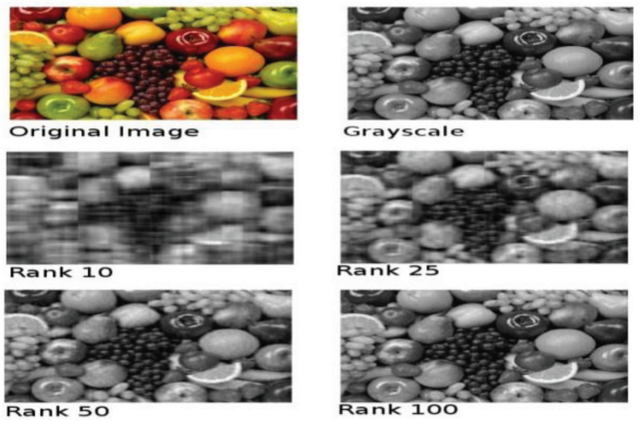

Figure 4a. Reconstruction of the image Fruits.

Figure 4b. Reconstruction of Human Face.

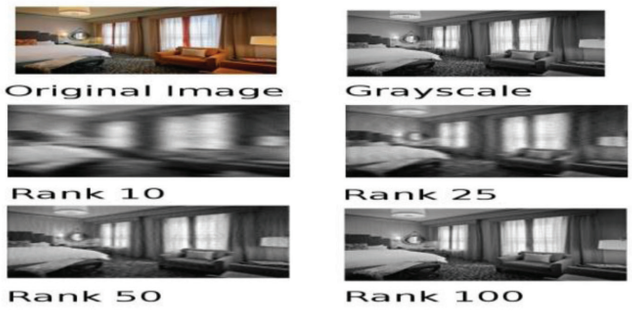

Figure 4c. Reconstruction of Room at different Ranks.

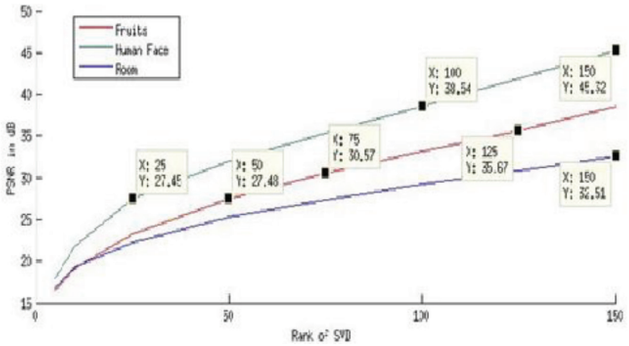

Figure 5. Variation of PSNR with Rank.

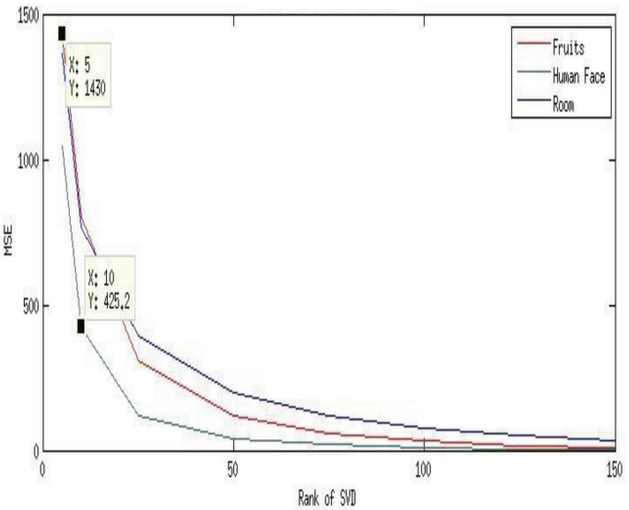

Figure 6. Variation of MSE with Rank.

Table 1. Performance details of SVD on image Fruits

Rank	5	10	25	50	100	125
Compression Ratio	27.4	13.76	5.47	2.74	1.37	1.09
Speed Up	1.11	1.12	1.12	1.10	1.13	1.12
Ex Time OpenMP(sec)	0.81	0.80	0.81	0.82	0.83	0.83
Ex Time Serial	0.90	0.90	0.91	0.91	0.94	0.93
Reconstruction Time ms	100	100	200	300	300	400

PSNR has been computed considering the maximum value of a pixel to be 255 in gray scale. A higher value for PSNR indicates a better reconstruction of the image. Table 4, 5 and 6 tabulates the image compression achieved through SVD using sparse matrix with CSR format for different rank values and its corresponding graph is shown in the Figure 9, 10 and 11.

Table 2. Performance details of SVD on Room

Rank	5	10	25	50	100	125
Compression Ratio	49.9	24.97	9.99	4.9	2.49	1.9
Speed Up	1.15	1.16	1.15	1.16	1.17	1.16
Ex Time OpenMP (sec)	5.2	5.22	5.25	5.3	5.30	5.37
ExTime Serial (sec)	6.0	6.10	6.13	6.20	6.22	6.26
Reconstruction Time ms	400	400	500	600	600	700

Table 3. Performance details of SVD on the Human Face 1

Rank	5	10	25	50	100	125
Compression Ratio	52.3	26.1	10.47	4.72	2.6	2.09
Speed Up	1.154	1.154	1.15	1.16	1.14	1.15
Ex Time OpenMP (sec)	7.1	7.14	7.19	7.23	7.42	7.45
ExTime Serial (sec)	8.20	8.24	8.32	8.39	8.53	8.58
Reconstruction Time milli sec	300	400	400	500	500	600

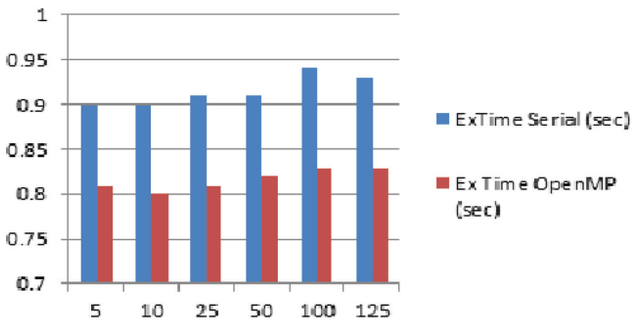

Figure 7. Graph for execution times of parallel versus serial for Fruit Image.

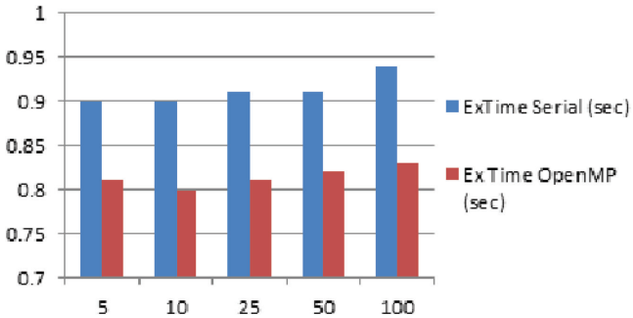

Figure 8. Graph for execution times of parallel versus serial for Room Image.

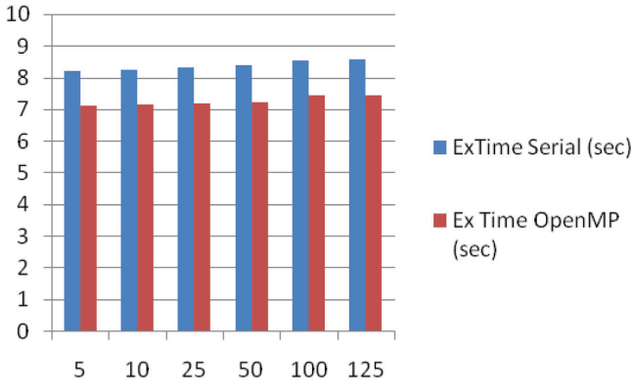

Figure 9. Graph for execution times of parallel versus serial for Human Face Image.

Table 4. Fruits file size for different rank values

Rank SVD	Size of original image in Kb	Size of compressed image (sparse matrix) in Kb
5	300	12
25	300	56
50	300	110
100	300	219
150	300	329

Table 5. Human face file size for different rank values

Rank SVD	Size of original image in Kb	Size of compressed image (sparse matrix) in Kb
5	1075	21
25	1075	103
50	1075	206
100	1075	411
150	**1075**	**616**

Table 6. Room file size for different rank values

Rank SVD	Size of original image in Kb	Size of compressed image (sparse matrix) in Kb
5	1008	21
25	1008	102
50	1008	205
100	1008	405
150	1008	606

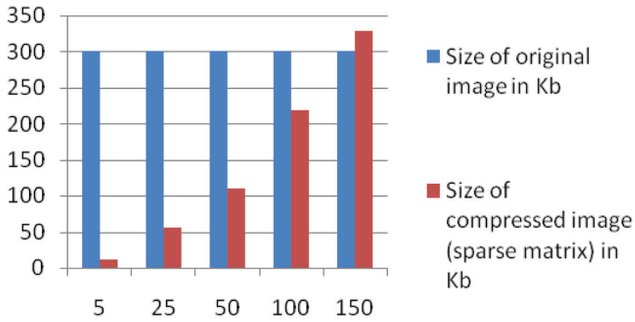

Figure 10. Fruit Original image size vs compressed image size.

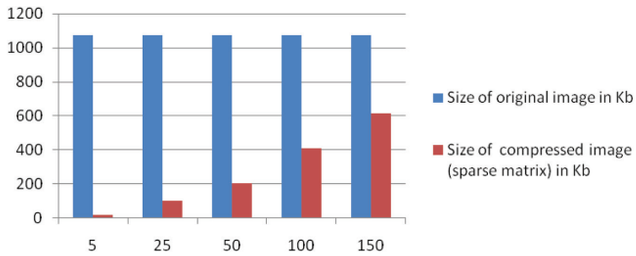

Figure 11. Human face Original image size vs compressed image size.

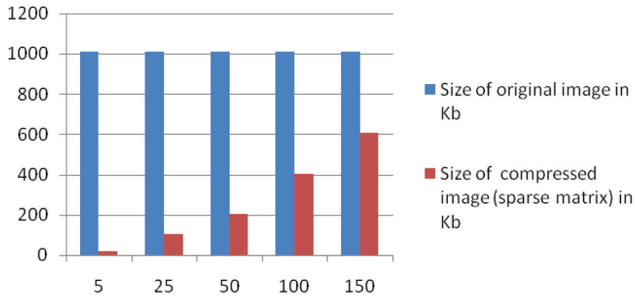

Figure 12. Room Original image size vs compressed image size.

7. Conclusion

The performance of SVD varies depending on the type of image being compressed. The PSNR for human face is better than the other two images. Thus SVD can perform better for compressing human faces. The quality of the image increases with high rank values but it results in lesser compression. The approach applied to a gray scale image can also be extended to the RGB matrices and thus compress colored images. The use of sparse matrix plays a significant role in reducing the total size required for storing the

compressed image. The use of OpenMp has increased the execution speed of our program and has resulted in faster compression time of images. While OpenMp is a widely used extension for parallel programming, its performance depends greatly on the parallelism of the code being executed. A data dependency in the loops can result in serial execution of the code and thus offer minimal benefits. However, in scenarios where the code is completely loop independent, it can give very high performance benefits. When OpenMp constructs are applied to loops with lesser iteration, it gives a lesser performance benefit due to the overheads involved in managing the threads.

8. References

1. Andrews HC, Patterson C. Singular value decomposition (SVD) image coding. Commun IEEE Trans. 1976; 24(4):425–32.
2. Berry MW, Mezher , Philippe B, Sameh A. Parallel algorithms for the singular value decomposition. Stat Textb Monogr. 2006; 184(117).
3. Ding CHQ, Ye J. 2-Dimensional Singular Value Decomposition for 2D Maps and Images. SDM. 2005; p. 32–43.
4. Dixit, MM, Vijaya C, et al. Computational analysis of adaptive Singular Value Decomposition algorithm to 2D and 3D still image compression application. Communication Control and Computing Technologies (ICCCCT). IEEE; 2010. p. 482–7.
5. Golub GH, Van Loan CF. Matrix computations. JHU Press; 2012.
6. Gunta C, Khan SN, Saha K, Pau DP. Acceleration of SVD routines in LAPACK. EUROCON. IEEE; 2013. p. 1733–7.
7. Jia Z. Using cross-product matrices to compute the SVD. Numer Algorithm. 2006; 42(1):31–61.
8. Kolev Vasil, Tsvetkova K, Tsvetkov M. Singular Value Decomposition of Images from Scanned Photographic Plates. arXiv Prepr, arXiv13101869. 2013.
9. Konda Taro, Nakamura Y. A new algorithm for singular value decomposition and its parallelization. Parallel Comput. 2009; 35(6):331–44.
10. Lahabar, S, Narayanan P. Singular value decomposition on GPU using CUDA. Parallel and Distributed Processing. IEEE; 2009. p. 1–10.
11. Prasantha HS, Shashidhara HL, Balasubramanya Murthy K. Image compression using SVD. Conference on Computational Intelligence and Multimedia Applications. IEEE; 2007. p. 143–5.
12. Rajamanickam S. Efficient algorithms for sparse singular value decomposition. 2009.
13. Rangarajan A. Learning matrix space image representations. Energy Minimization Methods in Computer Vision and Pattern Recognition. Springer; 2001. p. 153–68.
14. Rufai AM, Anbarjafari G, Demirel H. Lossy image compression using singular value decomposition and wavelet difference reduction. Digit Signal Process. 2014; 24:117–23.
15. Sadek RA. SVD based image processing applications: State of the ART, Contributions and Research Challenges. arXiv Prepr, arXiv12117102. 2012.
16. Soliman MI, Rajasekaran S, Ammar R. A block JRS algorithm for highly parallel computation of SVDs. High Performance Computing and Communications. 2007; 4782:346–57.
17. Van der Schaaf van A, van Hateren J van. Modelling the power spectra of natural images: statistics and information. Vision Res. 1996; 36(17):2759–70.
18. Yang J-F, Lu C-L. Combined techniques of singular value decomposition and vector quantization for image coding. Image Process IEEE Trans. 1995; 4(8):1141–6.
19. Yang J, Zhang D, Frangi AF, Yang J. Two-dimensional PCA: a new approach to appearance-based face representation and recognition. Pattern Anal Mach Intel IEEE Trans. 2004; 26(1):131–7.
20. Ye J. Generalized low rank approximations of matrices. Mach Learn. 61AD; 1-3:167–91.