Implementation of Multicore Algorithm for Image Compression using PCA

I.P.S.C. Project Report

Sai Charan
20172114

Sai Harsh
20172116

Abstract

In this project we are going to demonstrate the compression and decompression techniques of an image using a well known approach called PCA. Here, we read an image as a matrix format and then we calculate PCA of the given matrix, as we know that matrix computations in parallel setting is faster when compared to the sequential setting. At the end we conclude that applying parallelism in this kind of problem will result in significant speedup without compromising the quality given by sequential setting.

Keywords— PCA, openmp, compression, eigenvectors, eigenvalues, largest eigenvalues

1 Introduction

Principal Component Analysis (PCA) It is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components.

Parallel Algorithms A parallel algorithm is an algorithm that can execute several instructions simultaneously on different processing devices and then combine all the individual outputs to produce the final result.

- **Multi-core Parallalism** We use OpenMP for multi-core parallelism, it's a method of parallelizing, a master thread (a series of instructions executed consecutively) forks a specified number of worker threads and the system divides a task among them. The threads then run concurrently, with the run-time environment allocating threads to different processors.

2 Principal Component Analysis (PCA)

2.1 Compression using PCA

We know that a image can be represented as a square matrix[NXN], it's clear to see that a matrix contains some redundant information which is not as important as other values, our intention is to remove such redundant information to reduce the image size, algorithm 1 explain the steps to be followed to compress an image.
Algorithm 1 takes the input as an matrix of size NXN and output a compress matrix whose size is KXN, here K $\leq$ N.
We briefly explain the Algorithm 1 in below lines,

- Step-1: Find the Mean(NX1) vector, mean[i] = $\frac{1}{N}\sum_{j\leq N}^{j=1} A[i][j]$.
- Step-2: Compute D = A - Mean i.e substrate each column of A with mean.
- Step-3: Find co-variance matrix(NXN) as $\Sigma = \frac{1}{c-1} \times D \times D^T$.
- Step-4: Find the eigen vectors of co-variance matrix using algorithm 3.
- Step-5: Transpose the output of algorithm 3 i.e $(V^T)^T = V$ and sort the columns of V matrix based on square of singular values.
- Step-6: Take top-k eigen vectors and construct a matrix W(NXK).
- Step-7: Final step, Compressed Matrix(C(KXN)) = W^T(KXN) * D(NXN).

2.2 De-Compression using PCA

3 Parallel Algorithm for PCA

Algorithm 1: Image Compression using PCA

Data: Image in form of a matrix $I(r \times c)$
// r represents rows & c represents columns
Result: Compressed Matrix C
Function `Compression(`I`):`
  $I = [I_1, I_2, ...I_c]$ // where I_k represents k^{th} column of matrix I
  for *each column* I_k *in* I **do in parallel**
    /* Calculate & Store the values of sum for each column. */
    sumArray[k] = sum(I[k:])
  $\mu = sum(\text{sumArray})/c$ // μ represents Mean
  $D = I$ - μ
  $\Sigma = \frac{1}{c-1} \times D \times D^T$ // Σ represents Covariance matrix
  Calculate eigenvalues(λ) and eigenvectors(v) of Σ in parallel // Refer Algorithm 3
  Sort eigenvectors based on their eigen values.
  /* v_1, v_2, v_3... v_k are the top k eigenvectors */
  Construct $W = [v_1, v_2, v_3... v_k]$
  $C = W^T \times D$
  return C

Algorithm 2: Image De-Compression using PCA

Data: Compressed matrix C, top-k eigenvectors W, mean μ
Result: Reconstructed Image I_R
Function `DeCompression(`C, W, μ`):`
  $I_R = W \times C + \mu$
  return I_R

Algorithm 3: Singular Value Decomposition

Data: Matrix A[NXN]
Result: Decompose A as A=UDV'
1 Decompose A into the product of three matrices U1, B, V1', A = U1 B V1' where B is a bi-diagonal matrix, and U1, and V1 are a product of Householder transformations.
2 Use Given' transformations to reduce the bi-diagonal matrix B into the product of the three matrices U2, D, V2'. The singular value decomposition is then UDV' where U = U2 U1 and V' = V1' V2'.
3 Sort the matrix D in decreasing order of the singular values and interchange the columns of both U and V to reflect any change in the order of the singular values.

4 Jacobi Vs Golub

In 1955 Jacobi is proposed which uses rotations to find SVD of a matrix, later on in 1965 Golub proposed a method which uses reflectors to find SVD. Algorithm 3 explains it.
As we know that every matrix doesn't have eigen values and vectors because Finding eigen values is equivalent to solve a polynomial equation($A - \lambda I = 0$), we know that every polynomial equation doesn't have real roots.
Algorithm 3 is an iterative algorithm, for step-2, decomposing a bi-diagonal matrix into product of three matrices so

there is a possibility that it will go to infinite loop, to overcome this problem we kept a threshold. If the iteration count crosses the threshold we return as failure.

5 Results

(a) 512X512 (b) 100X100 (c) 50X50

(d) 512X512-Retrieved (e) 100X100-Retrieved (f) 50X50-Retrieved

Figure 1: Original and Retrieved images with different file sizes

References

[1] "Handbook for Automatic Computation vol II - Linear Algebra" edited by Wilkinson and Reinsch and published by Springer-Verlag, 1971.

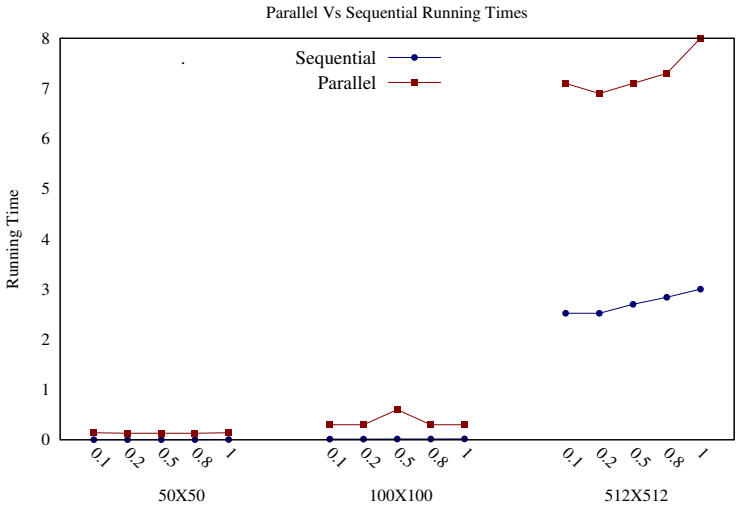

Figure 2: Parallel Vs Sequential Running Times