Fractal Image Compression in Parallel Computing Platform

E4750_2017Fall_FICP_report
Patyus Pati pp2636, Keerthana Madhu Moorthy km3280
Columbia University

Abstract

Fractal image compression is a form of compression which depends on the self-similarity of the image. It is not used widely because of its extensive computation complexity in the encoding stage. We identified three parallelisation aspects in the method. In this paper, we have implemented Fractal Image Compression using OpenCL to achieve better speed-up in the encoding and decoding process.We compared OpenCL performance with that of CPU implementation and also computed the PSNR values to evaluate the decoding procedure. Significant speed-ups were observed in all the three aspects.

1. Overview

1.1 Problem in a Nutshell

Image Compression is a type of data compression used for optimising the cost of storage and transmission. Fractal Image Compression is a lossy compression method which is most suited for images which have a lot of similar textures and patterns which can used for self-referential encoding.

Fractal Compression is based on Iterated System Functions(IFS). IFS are a method of constructing fractals. Such systems consist of a set of transformations which include rotation, scaling and shifting. These transformations can generate complicated fractal images. Fractal image compression does the inverse of fractal image generation. It searches the image for a set of fractals which can be used to represent the entire image. The set of fractals obtained represent the compact image data.

JPEG is the most widely used method of compression in the present day. Fractal Image Compression is better than JPEG. It does not depend on the resolution of the image and also uses a virtual codebook whereas JPEG is resolution dependent and uses actual image pixel information for transmission. The only drawback of Fractal Compression is its high computationally expensive encoding step.

The goal of our project is to identify the parallelization aspects in this method of compression and reduce its computational complexity.

1.2 Prior Work

The paper [2] explains Fractal Image Compression and applies parallelization. It shows that implementation of the method on FPGA using OpenCL is faster than a high end GPU, which in turn is faster than a multi-core CPU. It uses predefined scales and offsets for encoding process. The paper [3] explains a different version of encoding procedure which calculates scales and offsets from the codeblocks and the domain regions prior to codebook matching. It implements both parallelized and sequential versions of the algorithm on medical images and compares the performance.

2. Description

In this section we would provide the basic background behind fractal compression and describe the aspects of parallelization we identified and the implementation of the problem using OpenCL. Section 2.1 explains the main objective of our paper and also the challenges we faced during implementation. Section 2.2 explains fractal compression theory , 2.3 and 2.4 provide the technical aspects of our implementation in a detailed manner.

2.1. Objectives and Technical Challenges

The main objective of the problem is to achieve feasible speed-up of fractal image compression by parallelizing the task in GPU.

The potential challenges of this method of parallel implementation is scalability to different resolutions and handling of different data types in the kernel. To allow extension of the problem to any resolution, we assigned a single thread for codebook matching in encoding stage and for the region update in the decoding stage. The data types of scales, offsets and image pixel values should be handled carefully inside the kernel for consistent results.

2.2. Fractal Compression

In our project, for each 8x8 region we generate a 4x4 codeblock This is done by averaging every 2x2 region into one pixel. This is shown in the figure 1.

This codebook library is used to represent the entire image by appropriately selecting a scaled and shifted codeblock for every 4x4 region of the image. The codebook length determines the computational intensity of encoding procedure.

To match a region of the image to a scaled and shifted codeblock , the following metric should be minimised.

$$R = \sum_{i=1}^{n} (s.a_i + o - b_i)^2 \qquad (1)$$

Here a_i is the pixel of codeblock and b_i is the pixel of image range to which we are mapping the codeblock. s and o are scale and offset respectively.

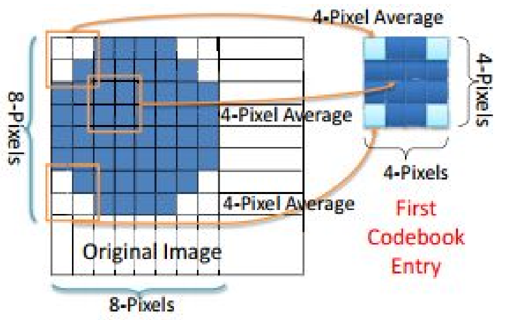

Fig.1 Generation of codebook[1]

The scale and offset values are calculated by the following formulae:

$$s = \frac{n(\sum_{i=1}^{n} a_i b_i) - \sum_{i=1}^{n} a_i \sum_{i=1}^{n} b_i}{n \sum_{i=1}^{n} a_i^2 - \sum_{i=1}^{n} a_i \sum_{i=1}^{n} a_i} \qquad (2)$$

$$o = \frac{\sum_{i=1}^{n} b_i - s \sum_{i=1}^{n} a_i}{n} \qquad (3)$$

Each range block is matched with one codeblock which gives least LMSE value as in equation (1). Here compression occurs because we require only three values to represent a 4x4 region. The triple of scale, offset and index of the codeblock is used to encode each 4x4 range of the input image.

Entire codebook transmission is not required for decoding. The image can be reconstructed by starting from a random image and iteratively updating each region by the formula

$$R_i = s_i \,.\, D_j + o_i \qquad (4)$$

According to fractal theory, by original value of the pixel can be obtained by iteratively updating by equation (4). This is illustrated in figure 2.

Peak Signal-to-Noise_Ratio is calculated to evaluate the accuracy of decoding procedure.

$$PSNR = 10 * log \frac{N^2 \,.\, Maxval^2}{\sum_i (Decoded(i) - Original(i))^2} \qquad (5)$$

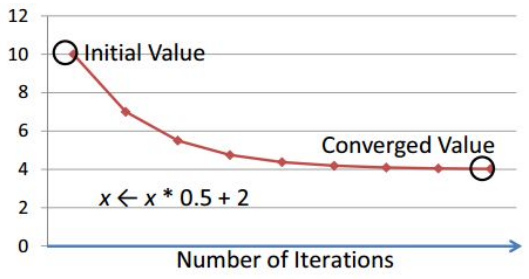

Fig.2 Convergence in Fractal Theory[1]

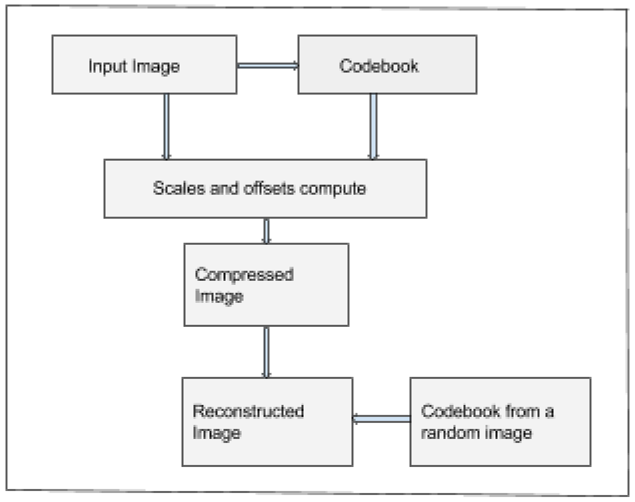

Fig. 3 Flowchart of Fractal Image Compression

2.3. Problem Formulation and Design

As there are many block-wise operations in the method , these operations can be separated as individual tasks and parallelized.

Parallelizable aspects of the algorithm:

1. Codebook Generation: Each codeblock in the codebook is a de-sampled region of the image. As downsampling of a region is independent of other regions, this step can be parallelized.

2. Encoding: The image is mapped to a set of scales, offsets and codebook indices. Each region encoding is independent of other regions.
3. Decoding: The iterative updation of the image is done for each 4x4 region individually. This stage also allows parallelization scope.

The aim of the project is to compare sequential implementation of fractal compression and the parallel implementation taking into account the above three aspects.

2.4 Software Design

OpenCL is used for parallel implementation of fractal compression. Input Image is taken and codebook is generated by calling the kernel function for codebook generation. This obtained codebook is used as input parameter for encoding while calling encode kernel function. Encoding function gives scales , offsets and codebook indices. These values are used and decoding is done iteratively on a random image. A random image is chosen , its codebook is generated and the random image is updated using the scaled and shifted versions of codeblocks. The decoding procedure is repeated until the PSNR value converges.

Codebook Generation:

The codebook is generated by averaging every 2x2 region in the input image. While parallelising this task , each averaging operation is assigned to a single thread.

Algorithm 1 Codebook Generation

```
1: procedure MAKECODEBOOK(Input, Output, Size)    ▷ Input - Image used
   for Codebook Generation, Output - Codeblock, Size - Size of Input
2:     $i, j$ ← Global Thread Indices
3:     Output[4][4] ← 0                        ▷ Each entry initialized to 0
4:     Block[8][8] ← Input[$8i : 8i + 8, 8j : 8j + 8$]         ▷ Get an 8x8 block
5:     for each 2x2 sub-block in the 8x8 block do
6:         Output[$n$][$m$] ← AVERAGE(sub-block[$n$][$m$])
7:     end for
8: end procedure
```

Encoding:

The input parameters for this step are input image and codebook and output parameters are scales , offsets and indices. The codebook is represented as a downsampled image of the input image. Due to scalability feature allowance, we assigned a single thread to matching a 4x4 region with a codeblock. This means that a single thread does the computation of scales, offsets for all the codeblocks corresponding to a single 4x4 range of input image and identifies the index of codeblock for which LMSE value is minimized.

The scales and offsets vector outputs from the kernel include scale and offset values for every 4x4 region of the input image. The indices vector output includes x and y index of each codeblock. Each thread computes the scales and offsets for the codeblocks ,gets the best match and gives the corresponding scale, offset and index value for the corresponding 4x4 input image region.

Algorithm 2 Encoding

```
1: procedure ENCODE(image, codebook, offsets, scales, indices, input-size,
   codebook-size)
2:     $i, j$ ← Global Thread Indices
3:     min ← INFINITY
4:     R[4][4] ← image[$4i : 4i + 4, 4j : 4j + 4$]
5:     for each block (x,y) in codebook do
6:         block-sum, block-sq-sum, image-sum, cross-sum ← 0
7:         for each pixel position (m, n) in the block and the image do
8:             block-sum ← block-sum + block[m][n]
9:             block-sq-sum ← block-sq-sum + $(\text{block}[m][n])^2$
10:            image-sum ← image-sum + R[m][n]
11:            cross-sum ← cross-sum + (block[m][n])(R[m][n])
12:        end for
13:        scale ← SCALE(block-sum, block-sq-sum, image-sum, cross-sum)
14:        offset ← OFFSET(image-sum, block-sum, scale)
15:        dist ← DISTANCE(R, block, scale, offset)
16:        if dist ¡ min then
17:            min ← dist
18:            final-scale ← scale
19:            final-offset ← offset
20:            final-indices ← (x,y)
21:        end if
22:    end for
23:    scale[i, j] = final-scale
24:    offsets[i, j] = final-offset
25:    indices[i, j] = final-indices
26: end procedure
```

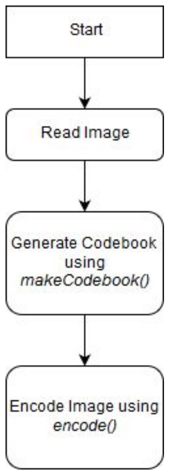
Start
Read Image
Generate Codebook using *makeCodebook()*
Encode Image using *encode()*

Fig. 4 Flowchart showing encoding

Decoding:

Decoding of an image involves iterative updating of the regions using codeblocks. A single thread is used for updating a 4x4 region using its corresponding scale, offset and codeblock.

Algorithm 3 Decoding-Update

```
1: procedure DECODE-UPDATE(image, codebook, offsets, scales, indices,
   input-size, codebook-size )
2:     $i, j$ ← Global Thread Indices
3:     R[4][4] ← image[$4i : 4i + 4, 4j : 4j + 4$]
4:     D[4][4] ← codebook(indices[i, j])
5:     scale ← scales[i][j]
6:     scale ← scales[i][j]
7:     R ← scale(D) + offset
8: end procedure
```

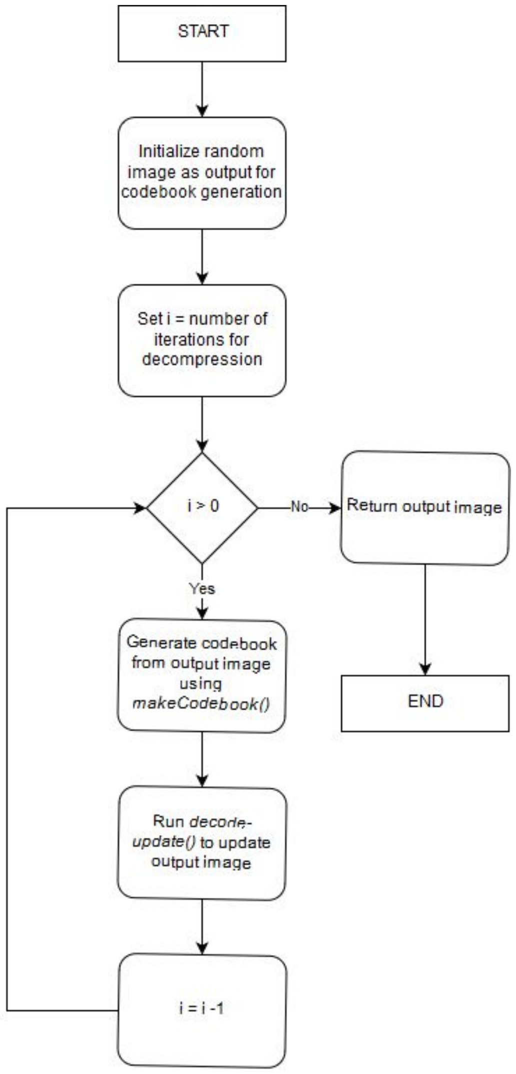
START
Initialize random image as output for codebook generation
Set i = number of iterations for decompression
i > 0
No
Return output image
END
Yes
Generate codebook from output image using *makeCodebook()*
Run *decode-update()* to update output image
i = i -1

Fig. 5 Flowchart depicting decoding procedure

3. Results

The project was implemented using PyOpenCL which is a python library for accessing the OpenCL parallel computing API. It provides various methods for transferring the data from the host to the GPU memory, for building the OpenCL kernel and for handling the size of the threads and blocks allocated to the task. The program code or kernel code which runs on each thread is written in OpenCL which is an extension of C, with additional variables and functions for memory management and control flow. The OpenCL kernel was built from within Python and transferred to the GPU. Python's numpy library was extensively used for reading and transforming the images and preparing it before transferring it to the GPU memory.

The code was tested for a 512x512 grayscale image.

Fig. 4 Input Image for encoding

The objective of the project was twofold - to compare the running time of the three parallelizable tasks between python and OpenCL and to evaluate the decoding accuracy through PSNR computation.

	OpenCL	Python	Speedup
Codebook Generation	0.0005s	0.3s	~600
Encoding	0.85s	14,090s*	~16,000*
Decoding	0.00046s	0.82s	~1800

*Projected values based on the time taken to encode one block.

Table .1 Fractal Image Compression Results

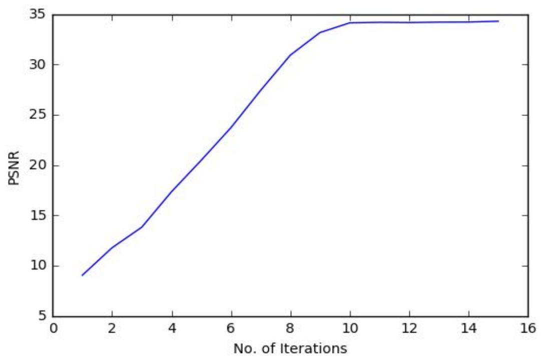

Fig. 6 PSNR of decoded image

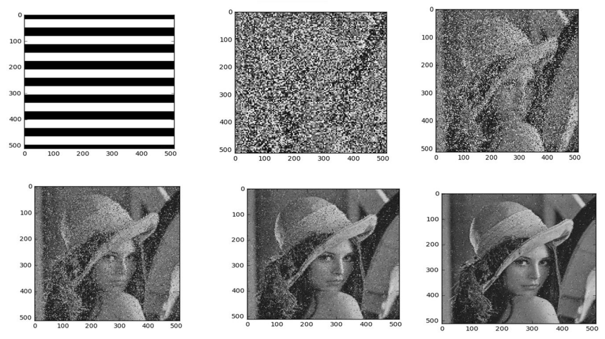
Fig. 7 Decoded Image after each iteration

4. Discussion and Further Work

Highly significant speed-ups were achieved in all the three steps of the procedure. Due to the implementation in python, which is a slower language than other languages like C/C++, enormous speed-up was observed at the Encoding step. The speed-up values for codebook generation and decoding step were observed to be reasonable. Due to python drawbacks and complete sequential implementation of encoding step,high speed-ups were observed. This could be converted to a more realistic situation by implementing an optimised version of encoding step in C/C++.

In this paper we focused on OpenCL implementation for a fixed block size and resolution. This work can be extended by using different block sizes and using a higher resolution random image at the receiver end. Future work can also include codebook generation optimization and exploring kernel time optimizations by employing shared memory access and coalesced memory access.

5. Conclusion

In this project, we have implemented fractal image compression sequentially and in parallel and compared the performance in each case. The PSNR value of the decompressed image observed to be converged after 10 iterations. A compression ratio of approximately 1.3 was observed. Significant speedup was achieved due to parallelization.

6. Acknowledgements

We would like to thank the course instructors for providing us with the knowledge to be able to implement this project.

7. References

[1] https://bitbucket.org/pp92/fractal_compression/overview

[2] D. Chen and D. Singh, “Fractal Video Compression in OpenCL: An Evaluation of CPUs, GPUs, and FPGAs as Acceleration Platform,” Proc. 18th Asia and South Pacific Design Automation Conf. (ASP-DAC 13), 2013, pp. 297-304

[3] Md. Enamul Haque, Abdullah Al Kaisan, Mahmudur R Saniat, and Aminur Rahman, “GPU Accelerated Fractal Image Compression for Medical Imaging in Parallel Computing Platform”
Link: https://arxiv.org/abs/1404.0774

[4] Priyadarshini K S, Dr. G S Sharvani, “A Survey on Parallel Computing of Image Compression Algorithms”, Proceedings of the International Conference , “Computational Systems for Health & Sustainability” 17-18, April, 2015.