Matching search in fractal video compression and its parallel implementation in distributed computing environments

Meiqing Wang \(^a,^*\), Zhehuang Huang \(^a\), Choi-Hong Lai \(^b\)

\(^a\) College of Mathematics and Computer Science, Fuzhou University, Fuzhou, Fujian 350002, China

\(^b\) School of Computing and Mathematical Sciences, University of Greenwich, Old Royal Naval College, Park Row, Greenwich, London SE10 9LS, UK

Received 14 April 2005; received in revised form 16 May 2005; accepted 20 May 2005
Available online 20 December 2005

Abstract

Fractal video compression is a relatively new video compression method. Its attraction is due to the high compression ratio and the simple decompression algorithm. But its computational complexity is high and as a result parallel algorithms on high performance machines become one way out. In this study we partition the matching search, which occupies the majority of the work in a fractal video compression process, into small tasks and implement them in two distributed computing environments, one using DCOM and the other using .NET Remoting technology, based on a local area network consists of loosely coupled PCs. Experimental results show that the parallel algorithm is able to achieve a high speedup in these distributed environments.

© 2005 Elsevier Inc. All rights reserved.

Keywords: Fractal video compression; Distributed computing; Parallel algorithm; Local area network; DCOM; .NET Remoting

1. Introduction

Amongst all new video compression methods, fractal video compression seems to be a favourable method amongst many researchers \([1–5]\). It is because the method achieves a high compression ratio and needs only a simple decompression algorithm for decompression. In \([6,7]\), we gave different partition methods of fractal video sequences and described respective sequential compression algorithms. However the computational complexity is very high. In our recent studies \([8,9,6]\), approximately 2 hours are needed to compress a VCD sequence consisting of 16 frames of 8-bit grey images each of \(720 \times 576\) pixels. Such algorithms are impractical for the media industry. Recently hybrid image partitioning \([10,6]\) has become an important technique used for fast fractal compression. In addition to optimizing fractal compression algorithms, many researchers tend to use parallel fractal compression algorithms in order to achieve a high speedup on high performance machines. Example parallel algorithms and implementations for fractal image compression include Jackson \([11]\) and

* Corresponding author.

E-mail addresses: mqwang@fzu.edu.cn (M. Wang), c.h.lai@gre.ac.uk (C.-H. Lai).

0307-904X/8 - see front matter © 2005 Elsevier Inc. All rights reserved.
doi:10.1016/j.apm.2005.05.018
Pommer [12]. Jackson and Tinney concentrated on the performance analysis and implementation of fractal image compression algorithms on tightly coupled machines of either shared-memory or distributed-memory. Pommer concentrated on the implementation of the algorithm on shared memory systems. These environments are mostly problem oriented and high costs, and parallel programs have poor portability across high performance platforms. Recently local area networks and the internet become the most popular and latest environments for distributed computing systems in order to simulate low cost high performance computing applications. These computing nodes are usually of low costs and are extendable to grid computing environments. The state-of-the-art technologies used to build distributed computing systems are CORBA, JAVA RMI/EJB, DCOM, .NET Remoting, and Web Service [13,14]. In this paper we use DCOM and .NET Remoting, due to the ease of access, to build two distributed computing systems and examine a parallel fractal video compression algorithm on the two systems. Both of the two distributed computing systems are based on local area network, and there is small time delay due to network communication. The numerical experiments provided in this paper are significantly different from others and consisting of a set of initial tests in an attempt to investigate fractal image compression on systems that are extendable and deployable on the grid. In the current paper the parallel algorithm is based on the sequential fractal compression method proposed in [7]. Attention is paid to the partitioning of the image on the two distributed computing systems. The resulting parallel fractal video compression algorithm achieves good speedup ratios in the local area network used in the tests.

This paper is organized as follows. First the basic theory for fractal video compression and the fixed-partition sequential algorithm is presented. Second a parallel algorithm is described after introducing task partitioning, and the related task communications are also discussed. The technology of DCOM and .NET Remoting for constructing distributed computing systems are briefly reviewed. Finally some experimental tests, results, and conclusions are given of the parallel algorithm on the two distributed computing systems.

2. Fractal video compression

Fractal image compression [15,16] is based on the iterated function system in fractal theory. The main idea [6] of the theory is that for each small part \( R \) of an object, a similar small part of the object can always be found from the same object. According to this similarity, a function can be built to represent \( R \). The function is a contractive mapping. An iterative function system generated by this function has \( R \) as the fixed point.

This idea is being used in fractal video compression as follows: a video sequence \( \text{Seq} \) consists of \( S \) frames of sequential images each of \( N \times M \) pixels. \( \text{Seq} \) is partitioned into non-overlapping small cubes known as range cubes. In general, the size of a range cube \( R \) is \( n \times m \times l \), where \( n, m \), and \( l \) can be 16, 8, or 4, and \( l \) can be 8, 4, 2 or 1, according to the rates of image motion. For convenience, let \( N = n \times k_n, M = m \times k_m \) and \( S = l \times k_l \). A vector \( V_R \) is related to \( R \) if the intensities of the pixels in \( R \) are collocated using a row-wise data structure which leads to a vector \( V_R \). The principle of fractal compression is based on the fixed-point theory of the Iterated Function System (IFS), i.e. for a range cube \( R \), there is an IFS whose fixed point is simply the vector \( V_R \) related to \( R \). Therefore, a similar part \( D_R \) of \( R \) needs to be found such that the corresponding related vector \( V_{D_R} \) satisfies \( V_R \approx zV_{D_R} + \beta I \), where \( z \) and \( \beta \) are the scaling factor and the offset factor respectively. The corresponding affine transformation \( W(X) = zX + \beta I \) is a generator of certain IFS, where \( X \) is a vector of \( n \times m \times l \)-dimension space \( \mathbb{R}^{n \times m \times l} \) and \( I \) is the identity of \( \mathbb{R}^{n \times m \times l} \).

A generator of IFS must be a contracted transformation, i.e. \( |z| < 1 \). Note that a simple non-overlapping partition of the video sequence \( \text{Seq} \) leads to \( D_R \) may not necessarily satisfy \( |z| < 1 \). However \( |z| < 1 \) may be achieved using the partition and matching process [15,11] as discussed here. If the video sequence \( \text{Seq} \) is partitioned into overlapping small parts known as domain cubes, where each of these are denoted as \( D \) with size \( 2n \times 2m \times l \), as depicted in Fig. 1. The dimension of \( V_D \) is not the same as that of \( V_R \) and hence they cannot be compared. In order to introduce comparison the domain cube \( D \) is shrunk by averaging the intensities of its four neighbouring pixel of disjoint groups. This leads to an \( n \times m \times l \) array denoted symbolically as \( D \) and is known as a codebook cube.

Fig. 2 shows the relationships between a domain cube, a codebook cube and a range cube. For every range cube \( R \) and codebook cube \( D \), suppose \( V_R \) and \( V_D \) are the related vector respectively. Then least square method can be used to solve the minimization problem.
where \( x \) and \( \beta \) and the location of \( D \) are the compression codes of \( R \).

Suppose \( K = n \times m \times l \), \( \mathbf{V}_R = (r_1, r_2, \ldots, r_K) \), and \( \mathbf{V}_D = (d_1, d_2, \ldots, d_K) \), then the optimal values of \( x \) and \( \beta \) and the minimal r.m.s. error \( E(D, R) \) can be computed as follows [8]:

\[
E(D, R) = \min_{x, \beta} \| \mathbf{V}_R - (x \mathbf{V}_D + \beta \mathbf{I}) \|,
\]

(1)

and

\[
E(D, R) = \sqrt{\frac{1}{K} \left[ \sum_{i} r_i^2 + x \left( \sum_{i} d_i^2 - 2 \sum_{i} d_i r_i + 2 \beta \sum_{i} d_i \right) + \beta \left( K \beta - 2 \sum_{i} r_i \right) \right]}.
\]

(2c)

For a given range cube \( R \) all possible codebook cubes need to be compared to it in order to find an optimal approximation, i.e. find a codebook cube \( D_R \) which satisfies

\[
E(D, R) = \min_{x, \beta} \| \mathbf{V}_R - (x \mathbf{V}_D + \beta \mathbf{I}) \| = \min_{D} E(D, R),
\]

(3)

where \( x \) and \( \beta \) and the index of \( D_R \) are known as the compression codes of \( R \).

The quality of decompressed images compared with original images may be described by the value of the Peak-Signal-Noise-Ratio (PSNR). The PSNR of an 8-bit grey image may be computed by using the formula [17],

\[
\text{PSNR} = 10 \times \log_{10}\left( \frac{255^2}{N \times M \sum_{i,j} (\hat{u}(i,j) - u(i,j))^2} \right),
\]

(4)
where $u(i,j)$ and $\tilde{u}(i,j)$ are the intensities of the original image and decompressed image respectively at the pixel $(i,j)$. Experience shows that the value of PSNR of a decompressed image can be as high as 38 to 40 db [18–20].

Let $\Xi_g$ be a group of consecutive frames chosen from a big sequence $\text{Seq}$. In general it is possible to divide the sequence $\text{Seq}$ into groups of frames such that $\text{Seq} = \bigcup_{g=1}^{k_s} \Xi_g$. Each $\Xi_g$ may then be compressed and decompressed as an entity. According to the speed of current available machines, it is appropriate that the frame number $S$ of a particular $\Xi_g$ is less than 16. For a given $\Xi_g$, where $1 \leq g \leq k_s$, the set of its range cubes is $\{R_{i,j,t}: 1 \leq i \leq k^n_g, 1 \leq j \leq k^m_g, 1 \leq t \leq k^l_g\}$ where the size of $R_{i,j,t}$ may be $8 \times 8 \times 4$, $4 \times 4 \times 2$ or $4 \times 4 \times 1$. The symbol $\Omega_g$ is used to denote the set which forms a collection of all codebook cubes of $\Xi_g$, where $1 \leq g \leq k_s$.

The basic fractal video compression algorithm [8] is given by Algorithm 1:

**Algorithm 1.** The cube-based fractal compression for video.

Given the image sequence $\text{Seq}$; Prepare $\Xi_g$, $1 \leq g \leq k_s$;

For $g = 1, \ldots, k_s$;

Prepare $\Omega_g$;

For $t = 1$ to $k^l_g$ do, For $j = 1$ to $k^m_g$ do, For $i = 1$ to $k^n_g$ do

For each $D_k \in \Omega_g$ do

$(x_k, \beta_k) := \text{Solve min } \|V_{R_{i,j,t}} - (xV_{D_k} + \beta I)\|_{F}$;

Compute $E(D_k, R_{i,j,t})$;

End-For

Compute the compression code:

$E(D_{\text{opt}}, R_{i,j,t}) = \|V_{R_{i,j,t}} - (x_{\text{opt}}D_{\text{opt}} + \beta I)\|_{F} := \min_{D_k} \{E(D_k, R_{i,j,t})\}$

Store $x_{\text{opt}}, \beta_{\text{opt}}$ and the index of $D_{\text{opt}}$;

End-For

End-For

From the algorithm, we see that the main computing work in fractal video compression is due to the procedure of match search, i.e. search an optimum codebook cube for every range cube. Since domain cubes can be overlapped, the number of codebook cubes which depends on these domain cubes is very large. Furthermore the search in the set of all codebook cubes is repeated for every range cube. This makes the computational complexity extremely high.

Further studies show that the match search procedure exhibits data parallelism, i.e. it can be effectively parallelized by running it concurrently on each range cube.

3. The parallel algorithm

There are some procedures in fractal video compression which can be implemented in parallel, such as: construction of the set of codebook cubes, matching search for each range cube, and decompression and display. Since the major computational complexity comes from the matching search procedure for each of the range cubes, this paper examines the parallel matching search procedure and implements it in two distributed computing environments.

In order to implement the parallel matching search procedure, we partition the procedure to a series of tasks.

3.1. Tasks partition

Although the match search procedures of any two range cubes do not require any communication between themselves, it is necessary to keep all compression codes generated by match search procedures in the same output file. This will allow easy handling of the compression codes. Hence it is not a good idea to generate small partitions. In our experimental environments the number of computing nodes is 2, 4 or 8. Therefore the problem is partitioned into 4, 8 or 16 tasks. The task partition is shown in Fig. 3.
3.2. Communication

Although it is not necessary to communicate between tasks, every task requires initial data which include the intensities of range cubes and of all codebook cubes. They can be sent to the correspondent computing nodes as arrays after codebook cubes are computed. It is then necessary to send back compression codes from every computing node after computing task done. Each task has to complete the code below.

For \( R_{i,j,t} \) in the task
For each \( D_k \in \Omega_k \) do
    \((a_k, b_k) := \text{Solve min } \| V_{R_{i,j,t}} - (a_k V_{D_k} + b_k I) \|;\)
    Compute \( E(D_k, R_{i,j,t}) \);
End-For
Compute the compression code:
\[
E(D_{\text{opt}}, R_{i,j,t}) = \| V_{R_{i,j,t}} - (a_{\text{opt}} D_{\text{opt}} + b_{\text{opt}} I) \| := \min_{D_k} \{ E(D_k, R_{i,j,t}) \};
\]
Store \( a_{\text{opt}}, b_{\text{opt}} \) and the index of \( D_{\text{opt}} \) as arrays;
End-For

The whole algorithm can be written as follows.


Given the image sequence \( \text{Seq} \); Prepare \( \Xi_g, 1 \leq g \leq k_s; \)
For \( g = 1, \ldots, k_s \)
    Prepare \( \Omega_g; \)
Send initial data to computing nodes;
Parallel execute tasks in computing nodes;
Receive compression codes from computing nodes;
Store compression codes in an output file according some given order; 
End-For

4. Distributed computing environments

The distributed computing technology being used in this paper relies on the Client/Server mode. User requests of tasks through client machines are transmitted by the distributed system to a server machine which in turn transmits the tasks to computing nodes. Distributed computing environments can be built on technologies such as CORBA, Java RMI/EJB, DCOM, .NET Remoting, or Web Service. In this paper DCOM and .NET Remoting technologies are chosen to build the distributed computing environment because of their easy implementation.

4.1. Using the DCOM technology

Distributed Component Object Model (DCOM) [13] supports remote objects by running a protocol known as the object Remote Procedure Call (ORPC), is built on top of the Remote Procedure Call, and interacts with the run time services of the COM. A DCOM server is a piece of code capable of serving particular objects at run time. Each DCOM server object supports multiple interfaces, each of which represents a different behaviour of the object. A DCOM client calls into the exposed methods of a DCOM server by acquiring a pointer to one of the interfaces of the server object. The client object then starts calling the exposed methods of the server object through the acquired interface pointer as if the server object resided in the client’s address space.

The distributed computing system based on DCOM includes two parts: a server pool and a client machine. The server pool consists of computing nodes \( \{N_1, N_2, \ldots, N_m\} \). Each computing node, \( N_i \), needs to complete the following jobs:

(i) Develop a DCOM component to compute the task submitted by the client.
(ii) Register this component with regsvr32.
(iii) Deploy this component with Dcomcnfg.

The client side requires to complete the following jobs:

(i) Partition a computing problem into sequential tasks and parallel tasks.
(ii) Run the sequential tasks in the main thread.
(iii) Create \( m \) threads, each of which connect to a computing node and call the remote method provided by the DCOM component in the corresponding computing node.
(iv) Write corresponding results from each thread into an output file according to certain order.

The structure of the distributed computing system based on the DCOM technology is shown as Fig. 4. DCOM works well and the performance is adequate when applications run on the network consisting of computers of the same type. However, one drawback of DCOM in the Internet connected world is that it relies on a proprietary binary protocol which is not always supported by all object models. This hinders interoperability across platforms. On the other hand DCOM communicates over a range of ports that are typically blocked by firewalls.

4.2. Using the .NET Remoting technology

The .NET Remoting [14] technology eliminates the drawbacks of DCOM by supporting different transport and communication protocols. It is an enabler for application communication and a generic system for different applications to communicate with one another. .NET objects are exposed to remote processes, thus allowing communication between processes. The applications can be located on the same computer, different computers of the same network, or even computers across different networks. This allows .NET Remoting
to be adaptable to the network environment in which it is being used. It is easy to implement load balance by maintaining a task pool in a system based on .Net Remoting.

The distributed computing system based on the .NET Remoting technology has also the server side and the client side. The server-side can be partitioned as a monitor and a server pool. The relationships between the monitor, the server pool and clients are shown as Fig. 5.

In this distributed computing system, clients submit computing tasks to the monitor in which each of the tasks is partitioned into many smaller tasks which are then put into a task pool. The main thread of the monitor is responsible for connecting to computing nodes in the server pool, sending tasks located in the task pool to the computing nodes, and gathering results from the computing nodes. The main thread also deploys two separate threads, known as the node-monitor thread and the task-monitor thread, to manage the server pool and the task pool. The node-monitor thread manages computing nodes and the task-monitor thread manages small tasks located in the task pool. The management of computing nodes and the task pool is achieved by using a list.

5. Experiments and the results

Two sequences of motion images, one from a videoconference and the other from a movie sequence, were used in the numerical experiments. The original sequence of motion images from the videoconference consists of frames of 8-bit grey image each of $256 \times 256$ pixels. The original sequence of motion images from an extract
of a movie consists of frames of 8-bit grey image each of $720 \times 576$ pixels. Some of original images are shown in Fig. 6. Using the sequential algorithm the values of PSNR are 35.10, 35.38, 38.28, and 38.44 for frame numbers 1, 2, 14, and 15 respectively of the videoconference image sequence. On the other hand the values of PSNR are 32.33, 32.36, and 32.16 for frame numbers 1, 2, and 3 respectively of the video image sequence [9]. The PSNR obtained by the distributed algorithms are to be validated against these benchmarking numbers.

The tests include the parallel matching search procedure in the fixed-partition cube-based fractal video compression algorithm with sizes of range cubes being $4 \times 4 \times 2$ and $4 \times 4 \times 1$ for the videoconference sequence and the movie sequence respectively.

The experimental network environment is a local network consisting of five PCs each of 1.7 GHz CPU with a 256 MByte RAM. These PCs are connected via a 10 M/100 M Switch and the operating system is Windows 2000 professional. A summary of the numerical tests for the motion images are listed below.

<table>
<thead>
<tr>
<th>Performance</th>
<th>Number of frames</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Four frames</td>
</tr>
<tr>
<td></td>
<td>Run time (s)</td>
</tr>
<tr>
<td>Sequential algorithm (videoconference)</td>
<td>69</td>
</tr>
<tr>
<td>Two computing nodes (videoconference)</td>
<td>35</td>
</tr>
<tr>
<td>Four computing nodes (videoconference)</td>
<td>21</td>
</tr>
<tr>
<td>Sequential algorithm (video sequence)</td>
<td>464</td>
</tr>
<tr>
<td>Two computing nodes (video sequence)</td>
<td>245</td>
</tr>
<tr>
<td>Four computing nodes (video sequence)</td>
<td>141</td>
</tr>
</tbody>
</table>
Test 1: Using the DCOM technology to build the distributed computing systems. The programming language is Visual C++. The run times of the parallel algorithms and their speedup ratios obtained are shown in Table 1.

Test 2: Using the .NET Remoting technology to build the distributed computing systems. The programming language is Visual C++ and C#. The run times of the parallel algorithms and their speedup ratios obtained are shown in Table 2.

Figs. 7 and 8 contain decompressed images with the corresponding PSNR values showing the same qualities as that acquired by the sequential algorithm.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Four frames</th>
<th>Eight frames</th>
<th>Sixteen frames</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequential algorithm (videoconference)</td>
<td>Runtime (s)</td>
<td>Speedup</td>
<td>Runtime (s)</td>
</tr>
<tr>
<td>Four computing nodes (videoconference)</td>
<td>23</td>
<td>3.000</td>
<td>89</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Four frames</th>
<th>Eight frames</th>
<th>Sixteen frames</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequential algorithm (videoconference)</td>
<td>69</td>
<td>–</td>
<td>270</td>
</tr>
<tr>
<td>Two computing nodes (videoconference)</td>
<td>38</td>
<td>1.816</td>
<td>145</td>
</tr>
<tr>
<td>Four computing nodes (videoconference)</td>
<td>23</td>
<td>3.000</td>
<td>89</td>
</tr>
</tbody>
</table>

Fig. 7. Decompressed images using the parallel algorithm (the compression ratio is 6.72).

Fig. 8. Decompressed images using the parallel algorithm (the compression ratio is 3.65).
6. Conclusion

In this paper a parallel algorithm of partitioning the matching search for range cubes in fractal video compression into small tasks is presented. The algorithm is implemented in two distributed environments, one based on DCOM technology and the other based on .NET Remoting technology. Experimental results show that run-times reduced with good speedup ratios. The values of PSNR for different cases of the decompressed images obtained are the same as those of the sequential algorithm. This property reflects the fact that the compression files generated by the parallel algorithms and by the sequential algorithm are the same. The distributed computing system based on .NET Remoting technology, which consists of homogeneous computing nodes, is easier to program and use in a grid computing environment than the one based on DCOM technology, which may consist of heterogeneous computing nodes. The timings observed for .NET system are generally smaller compared to the DCOM system. Tests for the videoconference image sequence have been performed by means of two computing nodes on the .NET Remoting technology as shown in Table 2. Early results demonstrated that load balance is not an issue and can be achieved successfully. Judging from the parallel algorithms and without lost of generality the load balancing property can be extended easily. Hence the .NET Remoting technology should be easier in reaching load balance dynamically.

Acknowledgement

The motion images used in the second set of the numerical experiments were provided by Meiah Entertainment Ltd, Hong Kong, China. This work was partially supported by Fuzhou University, China, under Grant No. XJB-0304 and partially supported by Fujian Development and Reform Commission, China.

References