OpenMP Parallelization of Jacquin Fractal Image Encoding

Hua Cao
School of Software Engineer
Huazhong University of Science and Technology
Wuhan, China
Caohua226@gmail.com

Xi-jin Gu
China Ship Development and Design Center
Wuhan, China
guxiqian@sohu.com

***Abstract*—The high compression ratio of fractal encoding is based on the increasing of the computation complexity. In order to meet the application requirements of real-time sharing and transmission, it is becoming an important research area to advance the encoding speed by developing the parallelization algorithm based on multi-core programming. In this paper, the OpenMP program model is applied to parallelize Jacquin fractal coding algorithm. Experiment results show that the speed-up ratio is more than four times compared with the original sequential program by programming the proposed parallel algorithm to Intel quad-cores CPU.**

Keywords- OpenMP;Fractal Coding;Muli-core Programming

I. INTRODUCTION

Although the storage technology is developing rapidly in recent years, but with the explosive growth of multimedia information, the requirement to reduce the storage space of image information has not been reduced yet. Fractal coding is a new kind of image compression method which different from the traditional transform-based ones. The compression ratio of fractal coding can reach as highest as 10,000:1 and 500:1 in general, much higher than the existing general DCT compression ratio of 20:1 and DWT compression ratio of 200:1[1]. But the drawback that a large search considering many blocks is computationally costly limits its widespread application. Therefore, optimization technique should be developed to reduce the execute time of fractal encoding for real-time application.

Currently, related hardware and software supporting parallelization have achieved greatly development. Among them, PC-based multi-core technology provides a good platform for parallel programming and it is becoming an important research area to advance the encoding speed by parallelization algorithm based on multi-core processors and parallel program environment.

II. RELATED WORK

There are many methods having been proposed to speed up the fractal coding, which can be divided into two categories. One is based on the theoretical framework of fractal coding to pre-classify the blocks using the methods such as DCT and DWT [2][3], or make use of genetic algorithms to improve the search path[4][5]. These methods are available to a considerable acceleration of fractal coding, but they did not get a multiple-level increase. The other is designed based on parallel processing hardware module such as FPGA/DSP [6][7], these methods rest upon specific hardware, the cost is larger and often pre-configured for a specific fractal coding framework, which is lack of flexibility.

With the emergence and extensive use of multi-core processors, multi-core programming has begun to be applied to various fields. PC-based multi-core processors provide a low-cost hardware platform, theoretically speaking, several core there are in multi-core processors can get the algorithm several times speedup, which has greatly significance to application performance improvement. At the same time specific algorithms are still implemented by software in multi-core program environment which in turn has a strong flexibility.

In this paper, with these strategies of partition paralleling, definition and pre-procession of shared variable, parallelization algorithm was presented after analyzing the sequential execute code of fractal coding and implemented under the programming environment of Intel multi-core processors and Visual studio 2008 based on OpenMP programming model. Experiments results show that the parallel algorithm greatly improves the speed of image fractal coding.

III. OPENMP APPLICATION MODEL

OpenMP is a portable, scalable model that gives shared-memory parallel programmers a simple and flexible interface for developing parallel applications for platforms ranging from the desktop to the supercomputer [8]. The OpenMP API uses the *fork-join* model of parallel execution, all OpenMP programs begin as a single process: the master thread. The master thread executes sequentially until the first parallel region construct is encountered. Then the master thread creates a team of parallel threads, which is called *fork*. The statements in the program that are enclosed by the parallel region construct are then executed in parallel among the various team threads. When the team threads complete the statements in the parallel region construct, they synchronize and terminate, leaving only the master thread, this process is called *join*. OpenMP Execution Model is illustrated as figure 1.

Sponsored by National Science Foundation of China (60973085)

978-1-4244-7161-4/10/$26.00 ©2010 IEEE

OpenMP parallelism is specified through the use of compiler directives and the directives format in C/C++ is as follows:

#pragma omp *parallel [clause [clause]...]*

When a thread reaches a *parallel* directive, it creates a team of threads and the code of this parallel region is duplicated and all threads will execute that code.

There are two types of Work-Sharing constructs defined in *clause*: DO /for Directive and SECTIONS Directive.

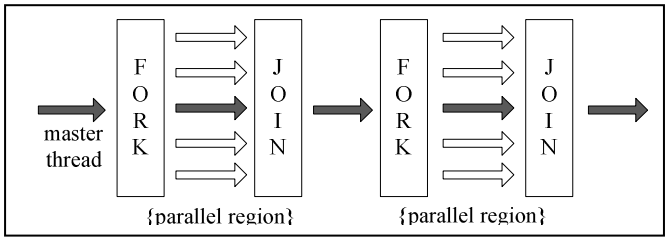


Figure 1. OpenMP Execution Model

The DO/for directive specifies that the iterations in the followed loop must be executed in parallel by the team, which represents a type of "data parallelism". The SECTIONS directive is a non-iterative work-sharing construct. It specifies that the enclosed section(s) of code are to be divided among the threads in the team. Independent SECTION directives are nested within a SECTIONS directive. Each SECTION is executed once by a thread in the team. Different sections may be executed by different threads.

IV. PARALLELIZATION OF JACQUAIN FRACTAL ENCODING

A Jacquin Fractal Encoding

Fractal coding is a lossy image compression method using fractals. Barnsley led development of fractal coding in 1987 which based on human intervention. Arnaud Jacquin implemented the first automatic algorithm in software in 1992. After that, most of development in fractal coding are based on the Jacquin's algorithm. All methods are based on the fractal transform using iterated function systems which relying on the fact that parts of an image often resemble other parts of the same image. Fractal algorithms convert these parts into mathematical data called "fractal codes" which are used to recreate the encoded image. The approach for Jacquin's fractal encoding is the following:

Step1: Partition the image domain into Range blocks R_i of size $s \times s$.

Step2: For each R_i, search the image to find a Domain block D_i of size $2s \times 2s$ that is very similar to R_i.

Step3: For each R_i, record the best matched mapping parameters of D_i, including: coordinate of D_i, type of affine transformation to D_i, factors of contrast and translation of gray scale, et al.

Fractal encoding is extremely computationally expensive because of the search used to find the self-similarities. Decoding however is quite fast. While this asymmetry has made it impractical for real time applications. In order to solve this problem, OpenMP programming model is introduced which can make full use of parallel processing based on multi-core to reduce the time of execute code.

B Parallelization of Jacquain Algorithm

By the following steps, parallelization of Jacquin Algorithm is achieved: Search range in sequential program of Jacquin' fractal encoding is partitioned into a team of sub search range, which be allocated to a group of threads implemented in parallel based on OpenMP program model. The specific procedure is described in C++ like language as follows:

1) Sequential Execution

```
EncImage (BYTE *pbits, CEncodedData *pd)
// pbits is the original image , pd is the fractal codes
{
 for(i=0;i<iBlk1;i++)
    for(j=0;j<jBlk1;j++) // Ri loop
      GetImageBlock(pbits,imgMin); //Get the Ri from original image
       for(i2=0;i2<iBlk0;i2++)
         for(j2=0;j2<jBlk0;j2++) // Di loop
          { GetImageBlock(pbits,imgMax); / / Get the Di from original image
              DeflateImage(imgMax,imgMin2);
              // retracting transformation
              double ss,oo,dd;
             for(int iType=0;iType<8;iType++)
            // Eight types of affine transformation
             {
               Transform(imgMin2,imgTrans,iMin,iType);
               CalcParams(imgMin,imgTrans,iMin,&ss,&oo,&dd);
               // Calculate the variance between Range and Domain blocks
               if(dd<d) pd->SetAt(i,j,ed);// Record the fractal codes
          }
}
```

The first for-loop (called R_i loop) complete the extraction of non-overlapping range blocks. The second for-loop (called D_i loop) complete contraction transform and eight types of affine transformation for each extracted Domain block. Variances between each range block and all of transformed Domain blocks are calculated to find the most similar domain block which possess the minimal variance, whose parameters are recorded in the vector *pd* as fractal codes.

2) Parallel Execution

In the sequential execution procedure of Jacquin's fractal coding, the fractal codes are generated by calculating variance between each range block in R_i loop with all of transformed domain blocks. It is compute independent for each R_i loop searching for the best matched domain block in D_i loop and there is also no data correlation between different R_i loop, which meet the requirements of parallel execution. Therefore, R_i loop computation can be assigned into the multi-core parallel threads by OpenMP directives. Issues involved in parallelization of Jacquin's fractal coding are described as follows:

a) Data Partition Parallel

In this paper, the means of parallel blocks is applied to partition the data involved in R_i loop along the horizontal direction. Therefore, several slice area are formed and a multi-core thread is responsible for searching the fractal codes of

every range block involved in R_i loop at every slice region. The final fractal codes are combined with the ones of each slice region. Partition parallel processes are shown in figure 2.

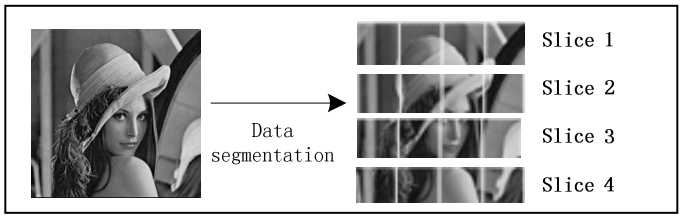


Figure 2. Partition-level parallel strategy

Data segmentation is implemented through controlling the range of the for-loop, the procedure is described as follows:

```
#pragma omp parallel for
for (num=0; num<nthreads; num++)
{
   for ( i= (iBlk1/nthread )*num; i<(iBlk1/nthread)*(num+1);  i++ )
    for (j=0; j<jBlk1; j++) // Ri loop
    {
          Get Domain Blocks from image;
          Deflate and Affine Transform to each Domain Blocks;
          Calculate variance between Range Blocks and Domain Blocks;
          Generate the fractal codes;
    }
}
```

In the procedure, nthread is the number of parallel threads and the iBlk1 is the total number of horizontal range blocks in the image.

b) *Pretreatment of Shared Varial*

In the sequential program of Jacquin algorithm, the operations of deflate and affine transform to each domain block are repeated in every one of R_i loop, which consumes a large amount of computing resources. In parallelization of Jacquin algorithm, this operation is moved out of the R_i loop and pretreated before parallel region to generate a sequence of deflate and affine transform of all domain block. The matrix sequence is set to be shared variable in compiler directives of OpenMP, so that all of threads in the parallel region can read the data in the matrix sequence to search best matched blocks. The procedure is described as follows:

```
Pretreat (BYTE *pbits, BYTE ***pDomainMatrix[row][line][type])
{
   Fetch all Domain Blocks from image;
   Affine Transformation for each Domain Blocks;
}
#pragma parallel for firstprivate(pd),lastprivate(pd), share(pDomainMatrix)
for (num=0; num<nthreads; num++)
{
    for(i= (iBlk1/nthread)*num;i<(iBlk1/nthread)*(num+1); i++)
      for(j=0; j<jBlk1; j++) // Ri loop
          {Calculate variance between Range Blocks and Domain Blocks;}
}
```

pDomainMatrix is the matrix sequence of the deflate and affine transformation of all domain blocks, *row* and *line* is the coordinate of each domain block, *type* is the type of affine transformation.

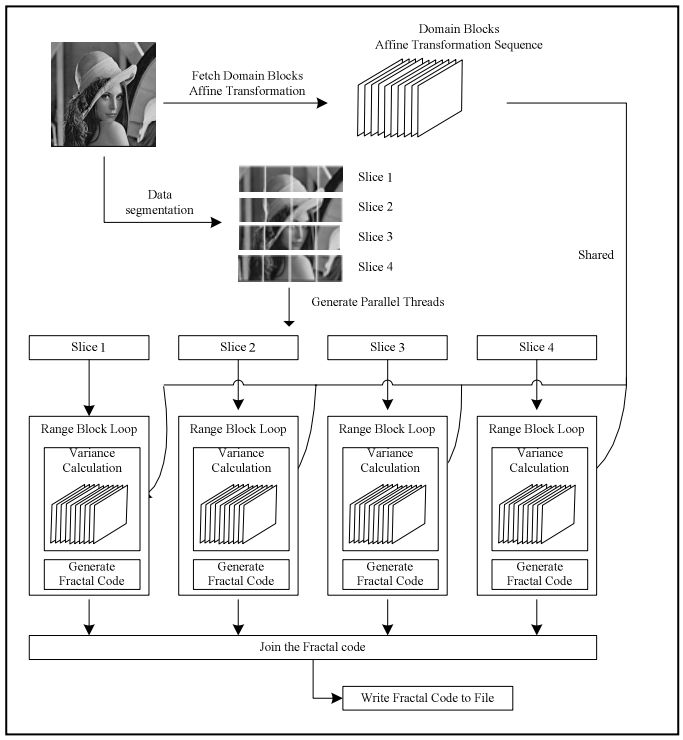


Figure 3. Parallelization Scheme of Jacquin Encoding

c) *Fork and Join of Fractal code*

pd is the complete fractal codes of the image, which is the combination of sub fractal codes generated from each threads in parallel partitions. The fractal codes vector of *pd* is forked by the clause of firstprivate(*pd*) into variables with the same name in parallel partitions being initialized according to the value of their original objects prior to entry into the parallel construct. Similarly, lastprivate (*pd*) clause combines the value with copies from the loop iteration or section to the original variable object so that the fractal codes vector is joined again. The clause is described as follows:

```
CEncodedData *pd=0;
#pragma parallel for firstprivate (pd), lastprivate (pd), share (pDomainMatrix)
```

Through deal with the above issues, the whole parallelization procedure of Jacquin algorithm is shown in Figure 3.

V. EXPERIENT RESULTS

Experimental platform is the Intel (R) Core (TM) 2 8200 processor (2.33GHz) with four cores, RAM is 2GB. The operating system is Windows XP SP3 and the compiler is the Visual Studio 2008. In the project's properties dialog box, change "Language" page under the "C / C + +"on the left box "Configuration Properties" to "Yes / (OpenMP)" A patch file,

"vcomp90.dll" also need to be installed into the directory of windows/system32, otherwise there would be a compile error when the OpenMP program be running.

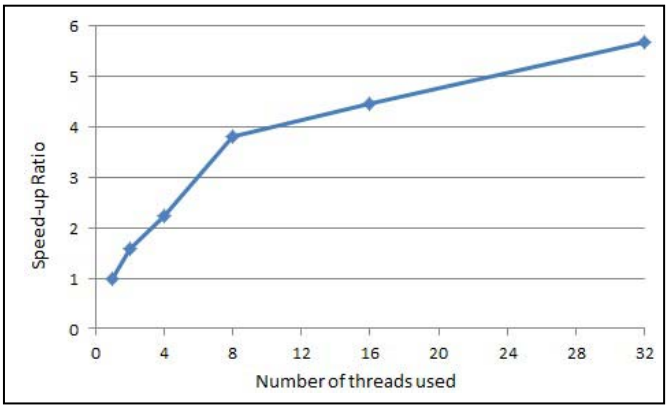
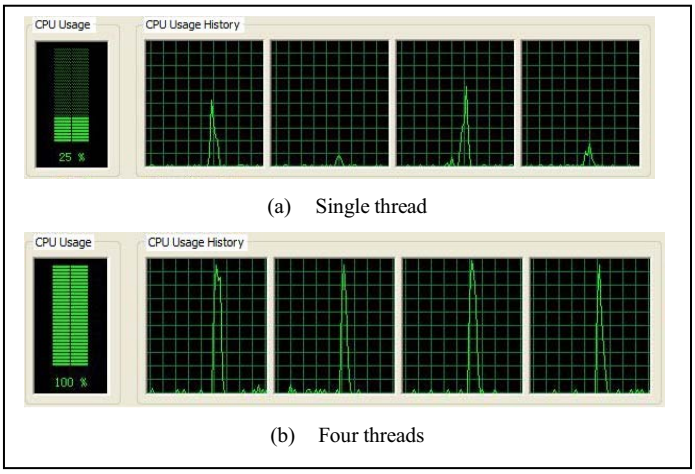


Figure 4. Speed-up ratio with different threads

Test image is the 512×512 grayscale Lena image, the speed-up ratio that runtime in single-thread and multi-thread is the index of performance evaluation. The performance with 2,4,8,16 and 32 threads was tested and the results shown in Figure 4.



(a) Single thread

(b) Four threads

Figure 5. CPU Usage of single thread and multi-threads

The experimental results in Figure 4 show that, with the increase in the number of parallel threads, the performance improvement is very obvious. When the number of threads is under eight, the performance increase almost linearly with the number of multi-threads and four cores are made full used of. Figure 5 present that the usage of CPU research 100% in four threads, in sharp contrast with the one of 25% in single thread. This is mainly attributed to the low data relationship in the search process of Jacquin algorithm, which is very suitable for parallelization. However, performance improvement under more than eight threads become not obvious, that is due to the method of fork-join be used in OpenMP program model written in run-time using fork, merge methods, a substantial increase in the number of threads, thread creation will increase accordingly, destroy overhead. A recommended OpenMP thread is 2 times the number of multi-core, experimental verification of this result.

VI. CONCLUSION

By analyzing the sequential version of Jacquin fractal coding algorithm, parallelization of Jacquin algorithm is researched and implemented based on OpenMP parallel programming model. Experimental results show that the proposed parallel algorithm for fractal coding is feasible and improvement in the encoding speed is extremely obvious.

REFERENCES

[1] Jacquin A E. Fractal image coding: a review[C]. Proceeding of the IEEE,1993,81(10):1451-145.

[2] Tomas Zumbakis, Jonas Valantinas. A new approach to improving fractal image compression times[C]. Proceedings of the 4th International Symposium on Image and Signal Processing and Analysis, 2005, 468-473.

[3] Jeng J H, Truong T K, Sheu J R. Fast fractal image compression using the hadamard transform [J]. IEEE Transactions on Image Signal Process, 2000, 147(6):529-534..

[4] Trieu-Kien Truong, Jyh-Horng Jeng, Irving S Reed, et al. A fast encoding algorithm for fractal image compression using the DCT inner product[J]. IEEE T ransactions on Image Process2ing,2000,9(4):529-534.

[5] Zhu wei yong, Yu Hai, Song Chun lin, et al. Novel fast fractal image compression approach based on error threshold and hierarchical search[J]. Mini2M icro System s, 2005, 26 (2) : 277-280.

[6] S McBader, P Lee, NC SpA. An FPGA implementation of a flexible, parallel image processing architecture suitable for embedded vision systems. Parallel and Distributed Processing, 2003.

[7] J Batlle, J Marti, P Ridao, J Amat. A New FPGA/DSP-Based Parallel Architecture for Real-Time Image Processing. Real-Time Imaging,2002,8(10):345-356.

[8] http://www.openmp.org