Proceedings of the 9th International Conference on Applied Informatics
Eger, Hungary, January 29–February 1, 2014. Vol. 1. pp. 231–237
doi: 10.14794/ICAI.9.2014.1.231

Parallelization of video compressing with FFmpeg and OpenMP in supercomputing environment*

András Kelemen, József Békési, Gábor Galambos, Miklós Krész

Department of Applied Informatics, University of Szeged
kelemen@jgypk.u-szeged.hu
bekesi@jgypk.u-szeged.hu
galambos@jgypk.u-szeged.hu
kresz@jgypk.u-szeged.hu

Abstract

The application of data compression has an increasing importance in the data transfer on digital networks. In the case of real time applications the limitations of these techniques are the compression/decompression time and the bandwidth of the network. In multicore environment the compression/decompression time is reducible with parallelism. OpenMP (Open Multi-Processing) is a compiler extension that enables to add parallelism into existing source code [1, 2]. It is available in most FORTRAN and C/C++ compilers. H.264 [3] is currently one of the most commonly used video compression format. Freeware version of H.264 is available in FFmpeg library [4]. This paper describes the use of OpenMP and H.264 library in multicore environment.

Keywords: OpenMP, H.264, FFmpeg

1. Introduction

Today's Internet technologies are widely used for viewing and broadcasting TV and video content. However the network throughput rates and the large frame size are strong limits to transmit high resolution uncompressed video in real time.

Video coding is the process of compressing/decompressing video content. Today a lot of video coding standards are available. Among them H.264 is the most widespread one.

*This work was partially supported by the European Union and the European Social Fund through project HPC (grant no.: TAMOP-4.2.2.C-11/1/KONV-2012-0010).

The software which is implementing video coding standard is called video codec. FFmpeg is a free, cross-platform solution for video coding. It contains many video coding standards including X.264, which is an open source implementation of the H.264 standard.

In the case of large frame size (e.g. full HD or higher resolution) the coding/decoding time has an increasing importance. In multicore environment the coding/decoding time reducible with parallelism. OpenMP (Open Multi-Processing) is a compiler extension that enables to add parallelism into existing source code [1, 2].

The aim of this work is to study the applicability of FFmpeg and OpenMP to reduce the coding time in high resolution real time video broadcasting applications. For this reason a test software was developed.

This paper organized as follows. In the Implementation section we describe the architecture of the test software and summarize the usage of FFmpeg and OpenMP in C/C++ environment. The Result and Conclusion sections summarize testing results and draw the conclusions.

2. Implementation

2.1. Test software

We implemented the test software in C++ using MS Visual Studio 2010. It captures frames from a camera by DirectShow. X.264 library is used to encode/decode individual frames. OpenMP is used for parallelization. Like every Internet application the test software consists of two parts: a sender and a receiver. The communication between the sender and the receiver is realized by Windows socket API.

The main loop of the sender part of the test software performs the following operations:

- it captures video signal from HD camera and put it into a memory queue
- it magnifies the fame resolution to destination size
- it encodes the frames using X.264 and OpenMP
- it transmits the encoded frames over TCP packets to the receiver

The receiver part works as follows:

- it receives encoded frames from the sender and put it into a memory queue
- it decodes the frames using X.264 and OpenMP
- it displays decoded frames

In the next we focus only on the encoding and decoding of individual frames.

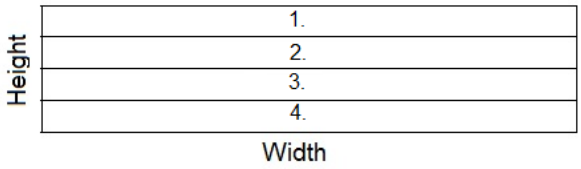

Figure 1: Partition of uncompressed input frame

2.2. Encode and decode

Encode/decode are done in an independent thread by the X.264 library. As we mentioned it in the introduction X.264 is a free open source implementation of H.264 video coding standard. The encoder processes input video stream frame by frame. For each frame it determines differences from previously processed frames and puts the result into the output stream. The decoder restores a video frame from the compressed stream. It uses previously decoded frames to reconstruct current frame. For more details of H.264 see [3-5].

We have to initialize the library with corresponding parameters before use of encoding and decoding procedures. The following code snippet illustrates of the initialization process [Fig. 2].

```
AVCodec *m_enccodec;
AVCodec *m_deccodec;
AVCodecContext *m_cenccontext;
AVCodecContext *m_cdeccontext;

void initCodec()
{
 avcodec_init();
 av_register_all();
 m_enccodec = avcodec_find_encoder(CODEC_ID_H264);
 m_deccodec = avcodec_find_decoder(CODEC_ID_H264);
 m_cenccontext= avcodec_alloc_context();
 m_cdeccontext= avcodec_alloc_context();
 set_ultrafast_preset();
 avcodec_open(m_cenccontext, m_enccodec);
}
```

Figure 2: Code snippet for initialization of X.264 library

The set_ultrafast_preset function, which is not part of the library, sets the context parameters to the fastest encoding [4].
The avcodec_encode_video and avcodec_decode_video library procedures implement the encoding and decoding algorithms. The X.264 library works on YUV color space. DirectShow provides uncompressed input frames in 24 bit RGB format. Thus we had to implement conversion routines between the color spaces.

As we mentioned above, parallelism is added to the encode/decode functions by OpenMP. It contains library routines and pre-processor directives. Like any sequential program an OpenMP application always starts with a single main thread called master thread [6]. To create additional threads the user starts parallel regions. The master thread is a part of the parallel region and it may spawn a team of slave threads as needed. Each thread has its own stack. At the end of the parallel region the slave threads terminate and the master thread continues its run (Fig. 3.).

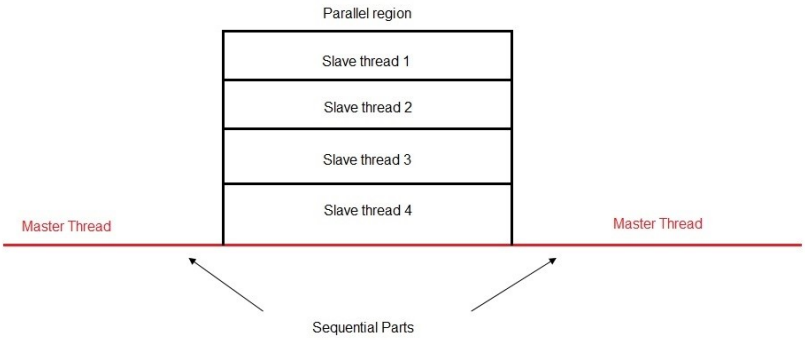

Figure 3: Flow diagram of OpenMP

OpenMP supports parallelism by pragmas. The syntax in C++ is the following:

```
#pragma omp <directive> [clauses]
```

There are numerous directives, but in this paper we focus only on the section directive. The threads are implemented as OpenMP sections as shown in Fig 4.

```
#pragma omp parallel
  {
        #pragma omp sections
        {
              #pragma omp section
              {
                // salve thread 1
              }
              #pragma omp section
              {
                // slave thread 2
              }
              #pragma omp section
              {
                // slave thread 3  ;
```

```
            }
            #pragma omp section
            {
              // slave thread 4
            }
        }
 }
```

Figure 4: Code snippet to implement OpenMP sections

Each parallel region starts with #pragma omp parallel directive. The scope resolution of pragmas determines by curly brackets. For more details of OpenMP see [1, 2, 6, 7, 8.].
In our test software we implemented a Codec class to wrap X.264 library functions. ProcessVideoFrame is a main function of the encoder. It works as shown in Fig 5.

```
Codec m_codec[4];

void ProcessVideoFrame( void )
{
 CaptureFrame();
 for(int i=0; i<4; i++)
 {
   m_codec[i].CopyFrameIntoCodec();
 }

 #pragma omp parallel num_threads(4)
 {
  #pragma omp sections
  {
   #pragma omp section
   {
        m_codec[0].convertToDestSize();
        m_codec[0].encode();
   }
   #pragma omp section
   {
        m_codec[1].convertToDestSize();
        m_codec[1].encode();
   }
   #pragma omp section
   {
        m_codec[2].convertToDestSize();
        m_codec[2].encode();
   }
   #pragma omp section
   {
        m_codec[3].convertToDestSize();
```

```
      m_codec[3].encode();
   }
  }
 }

 for(int i=0; i<4; i++)
 {
   SendToNetwork(m_codec[3].CommpressedFrame);
 }
}
```

Figure 5: Pseudo code to capture encode and transmit video frame

The video capture and the network transmission are in the sequential part, and the encoding part is in the parallel region. We implemented the decoder part of the test software similarly to the encoder.

3. Results

To evaluate our application we used Intel Core i7 920 2.6 GHz hardware platform with 6 GB RAM. The installed operation system was Windows 7 SP1. The resolution of test videos was 1920 x 1080 and 3840 x 2160 pixels. To test real time transfer we used the following test cases:

1. Compressed parallelized video
2. Compressed but not parallelized video
3. Not compressed parallelized video
4. Not compressed and not parallelized video

We tested the transfer rate on a crossover connection between two computers with CAT-6 cable standard and on a university network (max bandwidth 100 MB/s, CAT-5 cable standard). Table 1 and table 2 show the result of transfer times.

Test	Crossover	University Network
#1	1- 6	4-19
#2	5-26	18-82
#3	90	120
#4	328	474

Table 1: Averaged transfer times of 1920 x 1080 resolution in ms on different networks

The results show that real time transfer of high resolution video is possible with parallelism.

Test	Crossover	University Network
#1	5-30	20-90
#2	25-110	80-328
#3	364	510
#4	1320	2044

Table 2: Averaged transfer times of 3840 x.2160 resolution in ms on different networks

4. Conclusion

We developed a test software for transferring high resolution video in real time on computer network. We used compression and parallelization to increase the effectiveness of the transfer. We found that real time transfer of this kind of video is possible with good quality.

References

[1] Chandra, R., Menon, R., Dagum, L., Kohr, D., Maydan, D., McDonald, J., Parallel Programming in OpenMP. Morgan Kaufmann, (2000). ISBN 1-55860-671-8.

[2] `bisqwit.iki.fi/story/howto/openmp/`

[3] Richardson, I.,E.,G., H.264 and MPEG-4 Video Compression. Wiley, (2003) ISBN 0-470-84837-5

[4] `www.ffmpeg.org`

[5] Michail Alvanos, George Tzenakis, Dimitrios S. Nikolopoulos ,Angelos Bilas, Task based Parallel H.264 Video Encoding for Explicit Communication Architectures IEEE Proc. Int. Conf. on Embedded Comp. Systems: Architectures, Modeling, Simulation – IC-SAMOS, Greece, July 2011; pp. 217-224

[6] `www.soi.city.ac.uk/~sbbh653/publications/OpenMP_SPM.pdf`

[7] B. Chapman, G. Jost, R. van der Pas, D.J. Kuck (foreword), Using OpenMP: Portable Shared Memory Parallel Programming. The MIT Press (October 31, 2007). ISBN 0-262-53302-2

[8] Quinn Michael J, Parallel Programming in C with MPI and OpenMP McGraw-Hill Inc. 2004. ISBN 0-07-058201-7