Parallel Implementation of Xvid Decoder on Multi-Core

Ying Liu
College of Information Science and Engineering, Northeastern University, Shenyang, 110819, China
liuying@ise.neu.edu.cn

Fuxiang Gao
College of Information Science and Engineering, Northeastern University, Shenyang, 110819, China
gaofuxiang@mail.neu.edu.cn

Shiyuan Wang
College of Information Science and Engineering, Northeastern University, Shenyang, 110819, China
wangshiyuan@ise.neu.edu.cn

***Abstract*—With rapid development of multimedia technology, performance of computers is changing constantly. Today even the most ordinary computers have already equipped with multi-core processors. At the same time, high-quality videos have become main requirement of customers. Therefore it is a serious problem how to make video codec process a large number of video data efficiently on multi-core processors. It is a good way to develop the software which is compatible with multi-core. But most video codec has been already designed and developed for single-core processors. So it's a good idea to transform the current sequential program into the parallel one by the parallelization runtime library. In this paper we choose Intel isomorphic quad-core processor as hardware platform, Linux as OS, and use Intel parallel runtime library TBB to transform the decoder. The transformation includes that Loop parallelization, memory parallelization, data parallelization, pipeline parallelization and task-level parallelization. Then, for testing, sequential program and parallel one run on the same environment respectively, and the final results show that after parallelization the performance has improved significantly**

***Index Terms*—Parallelization; TBB; Multi-core;Xvid Decoder**

I. INTRODUCTION

Due to advances in circuit technology and performance limitation, speeding up processor frequency had not worked very well in the earlier part of this decade, so computer architects needed a new approach to improve performance. And the multi-core technology has become the mainstream in CPU designs [1]. The development of a sequential programming hardly involves the processor architecture, while the development of the multi-core application needs to know much about it. You can make full use of the multi-core resource, only if you know the hardware architecture very well. Program speed and CPU processing power is closely related.

Multi-core processor is a chip multiple with multiple processor engines. It's not a new concept, which has been used in embedded systems and specialized applications for some time. But nowadays this technology has become the mainstream because of many commercially available multi-core chips produced by Intel and AMD. In contrast to two and four core machines recently available, some experts believe that "by 2017 embedded processors could sport 4,096 cores, server CPUs might have 512 cores and desktop chips could use 128 cores." [2]

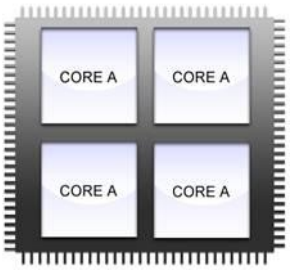

Figure 1. Homogeneous Multi-core Processor Configuration.

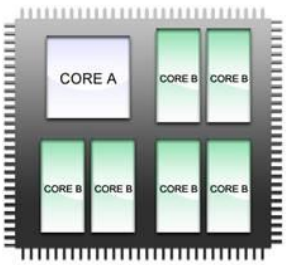

Figure 2. Heterogeneous Multi-core Processor Configuration

Multi-core CPUs can be categorized into two configurations, namely the "homogeneous multi-core processor configuration" and the "heterogeneous multi-core processor configuration". Each core in a

doi:10.4304/jcp.7.7.1639-1646

homogenous system is exactly the same: equivalent frequencies, the same cache sizes, functions, and so on. While, each core in a heterogeneous system may have different functions, different frequencies, memory model, and so on. So there is an apparent tradeoff between processor complexity and customization. As shown in Fig. 1, multiple cores of the same type are implemented in one homogeneous multi-core processor. While in Fig. 2, multiple cores of different types are implemented in one heterogeneous multi-core processor.

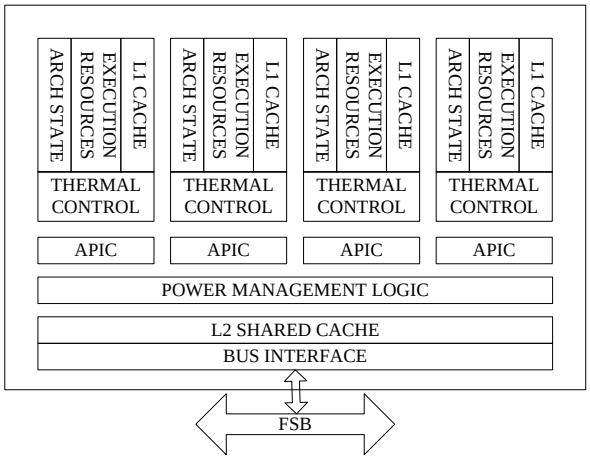

Figure 3. Block Diagram for the Intel Core 2 Quad.

Most multi-core processors used today are both Homogeneous multi-core processors. Intel is one of the mainstream microprocessors manufacturers in the world. It produces many different types of multi-core processors. The Pentium D is used in desktops, the Core 2 Quad is used in both laptop and desktop environments, and the Xeon processor is used in servers. So we choose Intel Core 2 Quad as the experiment processor in this paper. Fig. 3 shows block diagram for the Intel Core 2 Quad. Its architecture is the homogenous quad-core processors. The Core 2 Quad adheres to a shared memory model with private L1 caches and a shared L2 cache. While the eight-core or even more core processors can be the choice for extension.

Intel Core 2 Quad is used as the development platform, four cores of which have the same functions and operations. Each core has it own executive resources, and cache L1. The four cores share 6MB cache L2, and can support truly parallel computing [3].

With the continuous development of codec standards, computation of video processing is more and more, and the computing power of processors is becoming the bottlenecks of video application [4]. Parallelization on multi-core is an ideal solution for video processing, there is a natural parallelism in the process of which. The video codec can be parallelized from two aspects, hardware and software. Parallelization from the perspective of hardware, it's feasible to accelerate through multi-core platforms like DSP, GPU, IBM Cell or Intel. Parallelization from the perspective of software, the whole development cycle is too long time from the design for multi-thread parallelization on multi-core to accomplish according to the traditional parallel model. And the development and debugging of the multithread are also the technical problems themselves, which would cost the programmers a lot of time on learning traditional knowledge of parallel development. Parallelization on multi-core for the current sequential program can just parallelize some parts of it, while its infrastructure and algorithm unchanged. So the development time of a parallelization program design can be accelerated, and the efficiency of program has improved. We can use the parallel runtime library to transform a sequential program to a parallel one. TBB, OpenMP, and MPI are the common used parallel runtime libraries at present. In this paper, we choose Intel TBB as the multi-core runtime library. TBB, Thread Building Block, is a parallel programming tool developed by Intel. It is a C++ template library, which provides the parallel program with a suitable abstraction and the implementation for commonly used algorithm. TBB also has the feature of automatic load balance, flexible scalability, and so on.

It is an open source C++ template library, which can run on the OS like Windows, Linux and UNIX with C++ compiler. The parallelization level of TBB is higher, on the abstraction. It's based on task not thread. It concerns performance, scalability and etc. Therefore TBB is the best choice for parallelization development on multi-core. Compared with the traditional parallelization technology such as OpenMP and MPI, TBB has features and benefits as follows.

1) Task-oriented Parallelization Programming: The Thread functions are specified from the logical task rather than the physical thread. The task is more advanced parallel abstraction than the thread. When we use its C++ template library, we just need to design the task not the thread. That makes us ignore some detail about thread, such as load balancing, scheduling optimization, and so on. The underlying components of TBB have implemented these grunt jobs, which map the tasks to the threads through efficient way, and implement automatic scheduling by work stealing.

2) Easy to Use: TBB is the library implemented by the standard C++ template. It does not contain the special language extensions like in OpenMP, nor is a new programming language. We can choose the efficient parallel algorithm template from the C++ template library provided by TBB, and feel the benefits from multi-core processors.

3) Support crossing Platform: TBB is the library implemented by the standard C++ template. It does not contain the special language extensions like in OpenMP, nor is a new programming language. We can choose the efficient parallel algorithm template from the C++ template library provided by TBB, and feel the benefits from multi-core processors.

4) Supporting Development Tools: There many Intel supporting development tools for TBB, such as Intel Thread Checker, Intel VTune Amplifier and so on. All these can coordinate with TBB to improve the efficiency of developing and debugging.

We have studied the cubic convolution interpolation algorithm for image processing and parallelized it using the parallel programming tools TBB and OpenMP, and compare the performance of parallel and sequential implementations in our Preliminary work.

The paper is organized as follows. Section II briefly introduces the codec-Xvid. What's the Xvid and why we use it. And the Xvid decoder algorithm is introduced in Section III while its implementation is given in Section IV. Section V discusses how to parallelize the decoder program efficiently. The experimental results are shown in Section VI. Finally, Section VII concludes this paper and discusses the future work.

II. XVID AS PROFILE

There are many codecs in the world. The codec is an abbreviation for [co]der/[dec]oder, hence describes a program to encode and decode digital video. The purpose of encoding video data is to reduce redundancies. After that the video files can be smaller for faster transmission over computer networks or for more efficient storage on computer disks. And the Xvid is one of the most common ones currently. Nearly 90 percent movies on BitTorrent and eMule are suppressed by Xvid. The Xvid can somewhat be seen as a ZIP for video. But unlike ZIP, it is not lossless. That means that a video after compression and decompression with Xvid won't be identical to the original source. Typically however, a difference to the source is visually imperceptible. The Xvid removes information that is not important for human perception, which is somewhat similar to MP3 for audio. This enables very high compression rates that allow to effectively working with digital video on PC. For example: uncompressed digital video is huge and requires about 100 GB per hour at PAL resolution. The same video would require just 500 MB per hour at very high quality when compressed with the Xvid. That is a compression ratio of 200:1.

The Xvid is the Free Software and released under the GNU GPL license. This means that the source code of the software is publically available and programmers are allowed to make modifications to the code which is good news for us. We can obtain the Xvid free of charge, without time or feature limitations. Also the Xvid source code is publically available and can be reviewed by any interested programmer [5]. So we choose it as the experiment object.

It is possible to have Xvid running on many different platforms. And it could still be made available to new platforms or new operating systems by anyone interested without involvement of the original developers.

Since it is the first truly open source code, it's also a typical traditional sequential program. By transforming it into the parallel one, a general solution for parallelization of the typical sequential program is proposed. Most sequential programs can learn how to parallelize from it, including the parallelization analysis, parallelization design, and parallelization implementation.

III. ALGORITHM FOR XVID DECODER

As a video codec standard, Xvid has the open source code online. It has just implemented all functions of Xvid video standard. Although its readability and portability has been implemented, the real-time performance of video code has not been considered. Nor is the practicality for the further extension.

To make the code more practical, we choose decoder of Xvid developed on Linux as the research object. The core of its SMP version can support multi-core processor well. And Linux Kernel 2.6 is chosen as OS, which has a new process scheduler supporting SMP well. It can also keep load balancing and cache effective. Xvid decoder based on MPEG-4 will be parallelized, and it will be more practical.

Xvid decoding algorithm based on MPEG-4 includes three functions, decoder create for creating decoder instance, decoder decode for decoding image frame by decoder, decoder destroy for destroying decoder. There can be many decoders. We can call decoder decode for decoding image frame in cycle call, but only can call decoder create or decoder destroy just once.

In this paper, the parallelization process of decoding algorithm is mentioned. The algorithm process is introduced in detail. In MPEG-4 SP video decoding algorithm, the process of decoder decoding is shown in Algorithm 1.

Algorithm 1. decoder_decode

```
1: initialize stream structure;
2: analyze stream head structure;
3: if BitstreamReadHraders == 1 / 2 then
4:     analyze stream head structure;
5: end if
6: if BitstreamReadHraders == - 3 then
7:     decoder_resize;
8: end if
9: ensure the first decoding is I- frame;
10: decoder_iframe();
 decoder_pframe();
```

The SP level of MPEG-4 video frame has two kinds, Iframe and P-frame. I-frame decoding is decoded by function decoder iframe, while p-frame by function decoder pframe. The two functions both have to decode in cycle for all the macro blocks, which are the focus of parallel transformation. Function decoder pframe is chosen as an example because Pframe is the main type in decoding. The process of P-frame is shown in Algorithm 2.

To decode P-frame, there are three coding modes, Inter, Intra and not coded. The decoding for them is finished in a loop. And we will parallelize it.

```
/*All the Macro blocks in the Loop*/
For(y=0; y<height; y++){
 For(x=0; y<width; x++){
 /* Is it intra */        /* Intra macro block decoding*/
/* Is it inter */         /* Inter macro blockdecoding*/
/* Is it not coded*/   /* no decodereturn directly*/
}
}
```

Algorithm 2. decoder_pframe

```
1: initialize parameters;
```

```
2: while (all the blocks are decoded) do
3:    get quantization step of DC;
4:    if  decoded() == 1 then
5:       get decode information;
6:       if  Intra()==1 then
7:          Intra block decode;
8:       end if
9:       if  Inter()==1 then
10:         Inter block decode;
11:      end if
12:   end if
13:   no decode is needed;
14:   MV clear;
15: end while
```

IV. DECODER IMPLEMENTATION

The purpose of parallelization for traditional programs is to reduce as much workload as possible by rewriting the applications to guarantee performance. Therefore in the parallelization the general algorithm hardly changed. And this is the general principles for our parallelization analysis of sequential program [6]. In the process of specific analysis, the following aspects should be paid attention to mainly.

(1) Find out a suitable parallelization level for the original program. In this level, it's not necessary to consider the structure design of the original program. And it will be the key points of performance.

(2)The parallelization level mentioned above may cover the entire program, so the Independence of each data and program segment must be guaranteed.

(3)Data decomposition, task decomposition and pipeline decomposition can be used in the parallelization [7].

By analyzing Xvid decoder based on MPEG-4, the main function of the original program is to decode the Xvid video. The following points can be used in the parallelization parts.

(1) Function parallelization is finished firstly, which means that a frame is divided into several steps and each processor will process one of them.

(2) Data parallelization means that a frame data can be divided into several data blocks, and each processor will process a data block.

(3) Parallel memory allocation is used for allocation of the image frames.

(4) Work stealing strategy is used to balance workload when the image decoding work is distributed unevenly.

(5) There are many loops in the Xvid decoding program, including initialization of parameters in the video decoding and the computing process of decoding. Therefore Loop parallelization will be used through the whole program.

We can parallelize the original program from the following aspects. The first one is data decomposition. When parallelizing the decoder, we should pay more attention to the data dependence. In the parallelization of the data decomposition macro block (16×16 pixel block) group division is used. The second one is loop parallelization. From the source code of the key functions, we know that there are many loops which are not related to the data processing. The increasing number of loops can make computing time increase exponentially, which will be the bottleneck at last. So loop parallelization is the most important point of our work. The last one is task parallelization. The function process of Xvid decoder needs to be considered. If there is no traffic between tasks, or only some dependencies like pipeline, task parallelization can be used. The overall scheme of parallelization for source is shown in Figure. 4.

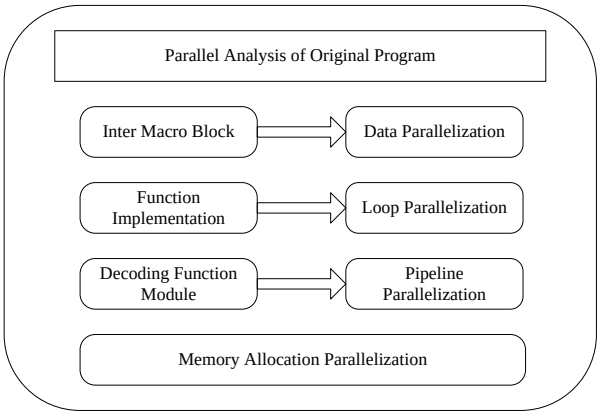

Figure 4. Overall Scheme of Parallelization.

Data Parallelization

Data parallelization of Xvid video decoding is to divide a frame of data into several data blocks, and each processor processes a data block. Macro block decoding function occupies a large amount of computation. It includes Motion compensation, inverse transformation, inverse quantization, and so on each of which needs much computation. Therefore the key to parallelize Xvid decoding is data parallelization. In the process of macro block decoding, the data dependency must be paid attention to. The decoding of the marco block needs to check whether all the related macro blocks are decoded. In the sequential program, the macro blocks are decoded according to the row sequence. When the current macro block is decoded, its reference blocks have been decoded. So macro-block-level parallelization is to find out the macro block whose reference macro blocks have been decoded.

By analyzing the code, we know that each macro block is decoded sequentially in a loop. To parallel a loop correctly, we must find out a macro block group which is independent with each other. If a macro block group can be decoded independently, two or even more groups can run together.

Loop Parallelization

The relevance between data must be analyzed when we study loop parallelization, and the data partition methods are discussed when computing. We should pay special attention to the data relevance. Of course, if there is no sharing data between tasks, these tasks can execute in parallel. If the two tasks which are relevant to each other

are to read the same data in the sharing memory unit, they can also execute in parallel. But when a task write and the other task read or write the same data, the sequence is critical. And they can't execute in parallel. Therefore when the relationship between data is the first two cases, loop parallelization can be finished

A. Function Parallelization

The work flow of Xvid CODEC decoding algorithm includes three steps, initializing CODEC, using CODEC, destroying CODEC. Creating handle and allocating memory is finished in the process of initializing CODEC. MPEG-4 decoding is finished in loop of using CODEC. The inverse process of initialization is finished in destroying CODEC. Xvid CODEC is written in standard C language, whose core module includes transformation copy, DCT transformation, SAD computation, motion compensation, CBP computation and so on. There are complex communications between tasks in the three steps, which are not suitable for parallelization on common multi-core PC. But for video input, frame decoding and frame output, there is the significant dependence, pipeline parallelization can be used. Pipeline parallelization of decoding is shown in Figure. 5.

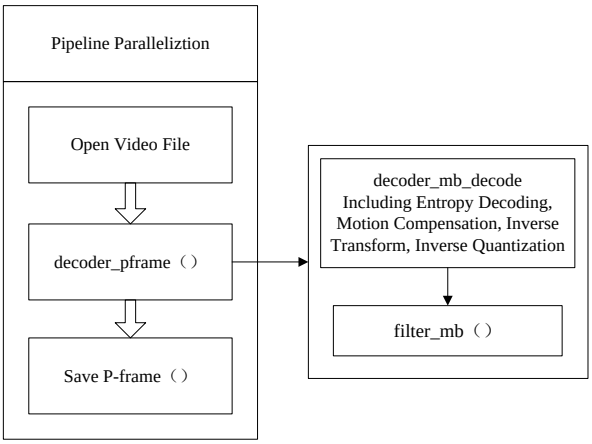

Figure 5. Pipeline Parallelization of Decoding

Overall, we can improve the parallel efficiency greatly by using pipeline parallelization structure on the whole, data decomposition parallelization in the frame decoding process, and loop parallelization in anywhere that it can be used. The purpose of parallelization solution is that scalability, high performance and correctness. And scalability means that as the core number of a processor and data amount processed increasing, performance and efficiency can be both improved. High performance means higher performance computing on multi-core. Correctness means that parallel program must have the same result as the sequential one.

IV. DECODER IMPLEMENTATION

According to the scheme in the Section IV and TBB specific algorithm, the overall structure of the sequential program parallelization is proposed, which is shown in Figure 6..

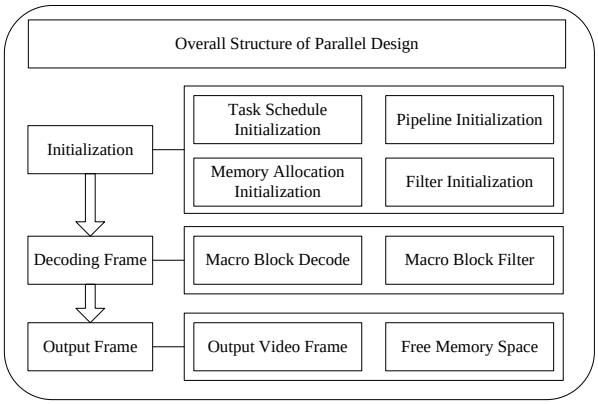

Figure 6. Overall Structure of Parallel Design .

Loop Parallel Implementation

Data parallelization of Xvid video decoding is to divide a frame of data into several data blocks, and each processor processes a data block. Macro block decoding function occupies a large amount of computation. It includes Motion compensation, inverse transformation, inverse quantization, and so on each of which needs much computation. Therefore the key to parallelize Xvid decoding is data parallelization. In the process of macro block decoding, the data dependency must be paid attention to. The decoding of the marco block needs to check whether all the related macro blocks are decoded. In the sequential program, the macro blocks are decoded according to the row sequence. When the current macro block is decoded, its reference blocks have been decoded. So macro-block-level parallelization is to find out the macro block whose reference macro blocks have been decoded.

Grain size is the measure of computing size and traffic size. In our case, that includes grain size of the application and grain size of the machine. Generally speaking, the grain size refers to the application grain size.

Application grain size is the ratio of computation and traffic. Traffic refers to overhead of data sharing or message passing among threads or processes. Machine gain size is the ratio of computing power and communications power.

Grain size makes a great impact on analyzing, coding and running the parallel program. When application running, a suitable grain size should be chosen. For a fine-grained application, if it runs in the coarse-grained, the efficiency is very poor. Because traffic becomes its bottleneck, and the coarse-grained makes it even worse.

TBB Grain size partition tool is used for grain size partition of Xvid parallelization directly. Meanwhile, tbb blocked range and tbb parallel for are used together for loop parallelization in this paper.

The parallel for is used for loop in the TBB library. Iteration is the simplest scalable way. By parallel processing of the circulation element in a loop, the speed of circulation process can be improved. The parallel for is

used in the parallelization of independent iterative process. In the Xvid decoding source, there are many loop operations in the process of frame decoding, which can be parallelized by parallel for.

In the loop tbb::block range is used for gain size partition. Grain size is default in the example above for the loop parallelization. The grain size will make a great impact on parallelization performance, so it's necessary to study it. For partitioning, the way of recursive division is better than static. blocked range is the iteration space from 0 to n-1. Its common form is [8]:

blocked range<T> (begin, end, gransize)

The iteration space is partitioned according to parameter gransize by parallel for, so the size of gransize makes a direct impact on the parallelization result. In the worse case, because of the wrong division of the subinterval, overhead of creating and destroying subintervals is much larger than performance improved by parallelization. So it's necessary to find out the best grain size partition. TBB has provided a default automatic partitioner. Set the value of gransize as auto partitioner, then automatic partitioner will give the proper grain size like 500, according to the system case. But there is a defect that the best gain size can not be given. So we have to propose another method to find a better grain size. By setting gransize as higher value and testing the result in the experiments, modify the value gradually. By analyzing the relationship between execution time and value size, the best gransize value will be found. Certainly, it's too tedious. If the loop parallelization is not used frequently, default value auto partitioner is recommended. Finally, task scheduler scheduler init will distribute tasks in parallel.

Memory Parallel Implementation

Xvid video decoding needs to allocate and free memory frequently. And it may occupy a lot of execution time. Therefore, we use TBB memory allocator to read video images and free memory.

The template of memory allocator provided by TBB is scalable_allocator<T>. The memory blocks can be allocated in advance and reused, which would reduce the overhead of system calls. TBB memory allocator uses thread private heap, not global lock. So it will not lead to lock contention.

TBB scalable memory allocator includes just TBB scalable memory allocator library, not TBB universal library. So it can be independent with other algorithm or container template. It replaces malloc, free, realloc, and calloc in C++. Accordingly, the memory allocated by scalable allocator should be freed by itself.

Data Parallel Implementation

To implement parallelization based on macro block group, it needs to be decoded according to the dependence. So thedependence of the macro block groups must be expressed correctly. It's implemented by a two-dimensional matrix. Each element in the matrix represents a macro block group in the corresponding coordinate set of the image. The value of element is the number of dependent macro block groups that haven't been decoded. When the value reduces to 0, the dependent groups have been all decoded. So a task can be created for decoding this macro block group. After that, the value of this group in the matrix is modified at the same time and check if there is any new group can be decoded. This process is shown as Algorithm 3.

Algorithm 3. parallel design

```
1: build two-dimensional data;
2: i=x/2;
3: j=y/2+x%2;
4: while value(i; j)!= 0 do
5:    decode group in coordinates(i,j);
6:    values of coordinates((i+1),j)-1;
7:    values of coordinates((i+1),(j+1))-1;
8:    values of coordinates(i,(j+1))-1;
9: end while
10: decode macro block group(i,j);
```

Pipeline Parallel Implementation

Pipeline parallel implementation needs to use TBB pipeline and filter class. Data processing node should inherit from filter class, which is a filter in the pipeline. Filter constructor has just a Boolean parameter, which is used to decide whether it is a sequential operation or parallel one. "True" represents the sequential operation while "false" represents the parallel one.

For CPU-intensive and IO-intensive applications in the program, pipeline model is chosen. The CPU-intensive part is put in a filter, and IO-intensive is put in another one. The two filters can be distributed different threads, and the speed difference between them is matched by the queue linking them. Therefore we can get the best efficiency of concurrent.

In the implementation, there is a pure virtual function void*operator()(void *item), and it must be implemented. The input parameters are the data from the previous node, and the processed data will be returned. Local Xvid video files input, loading video image, initialization, decoding video frame, and video frame output are finished at the nodes. So four filters need to be established, including input filter, load filter, frame decode filter, and frame output filter.

Task Parallel Implementation

Task scheduling mechanism is the core of all the algorithms. TBB uses the tasks with the threads, and the task scheduling mechanism is introduced. When using this mechanism, both TBB_scheduler_init and TBB_task should be used. TBB initializes a global thread pool implicitly [9], which can manage the thread pool automatically. All these functions can make programmers write high performance code easily.

The scheduler_init is the initialization of TBB task scheduler. TBB::task_scheduler_init must initialize TBB before using task scheduler or any TBB parallel algorithm template.

Task scheduler is used to create a thread for each core, and map it to the logic thread to make each thread run in full load. When a thread is free, it will "steal" tasks from other thread. And it will achieve dynamic threads load balance.

Every TBB task inherits from ”task”, and it must finish the following steps from its creating to running.

(1) Allocate memory space for tasks. The overloading operator “new” and task::allocateroot is used to allocate memory. The root suffix in the names represents that the task created have no father task, and it’s the root of the task tree.

(2) Create the task according to corresponding constructor, and it will be called in the overloading operator “new”.

(3) Run the task, until task::spaw_root_and_wait is finished.

VI. EVALUATION

Test Environment

Operating System: Red Hat Enterprise Linux 5
Processor: HP IntelRCoreTM2 Quad CPU Q8400 @2.66GHz
Memory Size: 4G
Decoding Program Compiler: GCC 4.12

After parallelization of sequential program on multi-core, Xvid video decoding is tested on quad-core. The testing sequence [10] is shown in Table I. Among them, CIF is the common standard image format, which is equal to 352288 pixels in the H.323 protocol cluster. While QCIF is also the common standard image format, which is equal to 176144 pixels in the H.323 protocol cluster.

TABLE I.
TEST SEQUENCE

Sequential	Resolution	Coded frames	Video file size
Akiyo(qcif)	QCIF	300	4.8MB
Coastguard	CIF	300	13.1MB
Container	CIF	300	15.9MB
Foreman	CIF	300	20.2MB

Sequential Program Data

In sequential experiments, the corresponding functions of Xvid decoder based on MPEG-4 run on Linux, and the decoding time is recorded. From Table II, we can find that decoding performance of small size video file is excellent, when it is not parallelized. But with the size of video file increasing, the speed has not changed a lot. In the following tables fps is short for frames per second

TABLE II.
DECODING TIME OF XVID DECODER

Sequential	Size	Frame Number	Decoding Frame Rate
Akiyo_qcif	QCIF	300	231.3547fps
Coastguard_cif	CIF	300	87.5061fps
Container_cif	CIF	300	66.0995fps
Foreman_cif	CIF	300	54.5729fps

A. Parallel Program Test

Evaluation capacity: Decoding Frame Rate in Sequential, DFRS for short; Decoding Frame Rate in Parallel, DFRP for short, Speedup which is equal to the ratio of DFRP and DFRS.

(1) Test of Gain Size Partition and Loop Parallelization The results are shown in Table III.

TABLE III.
RESULTS OF GAIN SIZE PARTITION AND LOOP PARALLELIZATION

Sequential	DFRS	DFRP	File Size	Speedup
Akiyo	231.3547fps	305.3882fps	4.8MB	1.320
Coastguard	87.5061fps	101.7696fps	13.1MB	1.163
Container	66.0995fps	77.9974fps	15.9MB	1.180
Foreman	54.5729fps	67.9432fps	20.2MB	1.245

(2) Test of Data Parallelization The results are shown in Table IV.

TABLE IV.
RESULTS OF DATA PARALLELIZATION

Sequential	DFRS	DFRP	File Size	Speedup
Akiyo	231.3547fps	282.2527fps	4.8MB	1.220
Coastguard	87.5061fps	100.6320fps	13.1MB	1.150
Container	66.0995fps	78.6584fps	15.9MB	1.190
Foreman	54.5729fps	67.1247fps	20.2MB	1.230

(3) Test of Pipeline Parallelization, The results are shown in Table V..

TABLE V.
RESULTS OF PIPELINE PARALLELIZATION

Sequential	DFRS	DFRP	File Size	Speedup
Akiyo	231.3547fps	224.4141fps	4.8MB	0.970
Coastguard	87.5061fps	83.1308fps	13.1MB	0.950
Container	66.0995fps	62.1335fps	15.9MB	0.940
Foreman	54.5729fps	50.7528fps	20.2MB	0.930

From the testing results, we can find that the program decoding speed hasn’t been improved but a little lower in the pipeline parallelization. Because that the introduction of Intel pipeline parallelization model leads the possible of lock conflict which is much larger than before with the hardware threads increasing. To the quad-core processor used for testing, the number of its hardware threads became the bottleneck of the whole experiments, so its efficiency is lower than the sequential one. But it doesn’t mean that pipeline parallelization is not feasible. When the number of the hardware threads is more than 32, pipeline parallelization is the most efficient.

(4) Test of Memory Allocation Parallelization. The results are shown in Table VI.

TABLE VI.
RESULTS OF MEMORY ALLOCATION PARALLELIZATION

Sequential	DFRS	DFRP	File Size	Speedup
Akiyo	231.3547fps	246.3928fps	4.8MB	1.065
Coastguard	87.5061fps	91.7939fps	13.1MB	1.049
Container	66.0995fps	69.6028fps	15.9MB	1.053
Foreman	54.5729fps	57.7927fps	20.2MB	1.059

(5) Test of General Parallelization

Compared sequential decoding sequence with parallel decoding sequence, after computing we can get the data shown in Table VII.

TABLE VII.
RESULTS OF GENERAL PARALLELIZATION

Sequential	DFRS	DFRP	File Size	Speedup
Akiyo	231.3547fps	393.9970fps	4.8MB	1.703
Coastguard	87.5061fps	139.1347fps	13.1MB	1.590
Container	66.0995fps	101.1322fps	15.9MB	1.530
Foreman	54.5729fps	91.1367fps	20.2MB	1.670

In order to show the results of that experiment vividly, we put the data in Table III, Table IV, Table V, Table VI and Table VII together in one diagram. In Fig. 7 we can see that the general parallelization has the best speedup while the pipeline parallelization has the worst speedup. And we know that for different environments, the result maybe totally different.

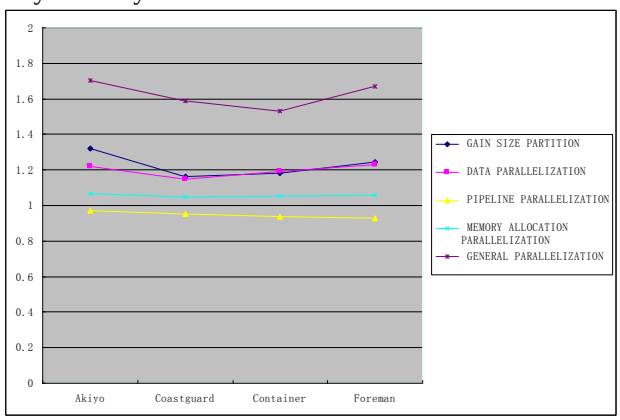

Figure 7. Note how the caption is centered in the column.

As shown in Table VII that there are two aspects: firstly, the video decoding frame rate is related to video file formats and video file sizes. Taking the QCIF format as the example, its decoding frame rate is significantly faster than the rate of the CIF format, while its speedup is also larger relatively; secondly, in terms of the same format, as the video file increases, the parallel decoding obtains more obvious advantages. Therefore, the parallel decoding based on the Xvid improves the utilization of the processor, especially for large video file, and its speedup even reaches 1.67.

VII. CONCLUSION

The parallel implementation of applications on Linux is studied in the paper. We have discussed the feasibility and solution of parallelization, and implemented it in the Xvid decoding application. The code needs to be optimized, and that's our work in the paper. By analysis of Xvid decoding open source code, a parallelization solution for it by TBB library is proposed. The parallelization includes loop parallelization, data parallelization, memory allocation parallelization, and so on. The performances of parallel decoder processing different sizes video files are tested in the experiments. And from the results, we can know that the performance of its functions has been improved by parallelization. We have just an initial exploration about parallelization. In future, further exploration will be taken about decoding on embedded systems, or the other multi-core CPU environments

REFERENCES

[1] L. Peng et al, "Memory Performance and Scalability of Intel‟s and AMD‟s Dual-Core Processors: A Case Study", IEEE, 2007
[2] R. Merritt, "CPU Designers Debate Multi-core Future", EETimes Online, February 2008, http://www.eetimes.com/showArticle.jhtml?articleID=206105179
[3] Lu Jinzheng, Zhou Dongmei. Visual C++ Audio/Video Processing Technology and Engineering Practice. Electronics Industry Press, 2009, 519-563.
[4] Zhou Shuxian. OpenMP-based Programming on Multi-core PC. SCIENCE & TECHNOLOGY INFORMATION, 2010, 9: 78-79.
[5] Xvid. http://www.xvid.org/, 2011
[6] Xie Xianghui, Hu Sutai, Li Hongliang. Multi-core/many-core processor and its influences on computer architecture design. Journal of Frontiers of Computer Science and Technology, 2008, 2(6), 641-650.
[7] TBB opensource. http://www.threadbuildingblocks.org, 2008.
[8] Wang Hai tao, Liu Shu fen. Parallel Computing Based on Linux Cluster. Computer Engineering, 2010, 1(36), 64-66.
[9] Xi Jie; Chen Jie. Parallel Intra Prediction on Multi-core Platform. Science Technology and Engineering, 2010, 10(6), 1379-1383.
[10] Video Sequence Download. http://bbs.chinavideo.org/viewthread.php?tid=1006, 2007.

Ying Liu, doctoral student, studies in College of Information Science and Engineering, Northeastern University, China

Fuxiang Gao, professor, works in College of Information Science and Engineering, Northeastern University, China

Shiyuan Wang, master student, studies in College of Information Science and Engineering, Northeastern University, China