Parallel Multi-source Video Processing Based on Software Pipeline

Yang Yang [1, a *], Yao Xiaocheng [1,b], Gao Yong [2,c] and Zeng Weini [1,d]

[1]The 716th Institute of China Shipbuilding Industry Corporation, Lianyungang, 222006, China

[2]Airforce Military Representation Office in Yangzhou Area, Yangzhou 225000, China

[a]yy6551@163.com, [b]yaoxiaocheng@jari.cn, [c]gaoyong@jari.cn, [d]zengweini@jari.cn

Keywords: Software Pipeline, Parallel Processing, Multi-source Video, Multi-task Management System.

Abstract. A technology of parallel multi-source video processing based on software pipeline is presented, which has the features of reducing the design difficulty of the video processing applications, optimizing computing resource utilization rate and improving the real-time performance of the multi-source video processing with a complex algorithm. First, a kind of multi-task management system based on the multi-core CPU is introduced, which is used to schedule tasks and manage the loads of the computing resource, and on this basis, software pipeline is designed out. The software pipeline encapsulates the processing of a frame image as a task which will be submitted to the multi-task management system. Finally, the parallel multi-source video processing experiment was carried out to verify this technology. The results indicate that this technology could effectively improve the real-time performance of the multi-source and complex algorithm video processing, and it also has nice features in optimizing the computing resource utilization rate and load balance.

Introduction

As the expansion of computer vision technology, intelligent video processing system integrated with image processing, pattern recognition, artificial intelligence and some other technologies, has been widely applied not only in the targets recognition field represented by license plate recognition [1] and face recognition [2], but also in the behavior analysis field represented by perimeter prevention, traffic statistics [3] and intelligent tracking [4]. Intelligent video processing system mainly consists of front-end video capture and video processing server. In the targets recognition field, the demand for high definition video prompts the front-end video capture to produce vast quantities of image data. In the behavior analysis field, in order to accommodate the complex and fast-changing scenes and mine the behavior patterns accurately, the complexity of video processing algorithm is increasing day by day, the real-time requirements of the behavior patterns analysis in some key domains are also increasing. At the same time, people are no longer content with the analysis of single source video collected by one path camera, and more inclined to analyze the multi-source video consequently to acquire depth-layer effective information which is contained in big scenes and massive video data. The intelligent video processing system has to face the large volume and multi-source data, high-complexity video processing algorithms and harsh real-time constraint. In addition to the requirement of superior performance server hardware, it is necessary to construct the framework of the intelligent video analysis system software, which is propitious to the design of intelligent video analysis algorithms and the coordination of hardware resource allocation.

The liquidity of video data makes using pipeline technology to analyze video more naturally and efficiently. Pipeline technology disassembles a task into a number of subtasks executed sequentially, the different subtasks are executed by different execution units, and these execution units can work simultaneously, that consequently improves the efficiency and throughput. Currently, most of the intelligent video analysis servers adopt single or multiple multi-core CPU, all the cores will run concurrently, and it forms the basis of the hardware platform to use software implementation mode to simulate the pipeline work flow [5, 6].

This paper first presents a method to structure the software pipeline based on the multi-core CPU hardware platform, and then introduces the application of using software pipelines to process multi-source video. Finally, some results of multi-source video processing are given.

Multi-task management system

For multitasking applications, tasks usually present in the form of threads in multi-core CPU servers, the operating system supports multitasking hardware concurrency, and manages thread scheduling. Despite this, multithread programming is still complex, we must handle thread synchronization, load balancing, execution efficiency and some other high-level issues. In order to reduce the design complexity of multitasking applications, this paper presents a multi-task management system, with multi-level caches, automatic task scheduling, adaptive load balancing, thread pool management, dependency analysis and some other functions. The multi-task management system also provides application programming interface (API), the functions include system initialization or release, task submission and waiting for a task to finish. The task properties are shown in Table 1, and the task has a Run() method to achieve the specific function.

Table 1 Task properties

Property	Commentary
Task_ID	The ID of the task.
Task_State	The state of the task, including not submit, cached, suspend, running and finish.
Task_Priority	The priority of the task.
Num_of_Predecessors	The number of predecessors.
Num_of_Children	The number of children.
Predecessor_list	The list of predecessors.
Successor_list	The list of successors.
Finish_Semaphore	The semaphore used to indicate if a task has been finished.

The basic components of the multi-task management system are shown in Fig. 1, including multi-level cache component, task scheduling module and execution component. Multi-level cache component provides caching support in the phase of task submission and execution, it is composed of submission cache, execution cache and cache management module. Submission cache includes several FIFO queues with different priorities, and it is used to cache tasks submitted by applications. Execution cache is composed of running queue and suspending queue. Suspending queue prioritizes the cached tasks. Running queue is a FIFO queue, in order to ensure a higher priority task can be executed as soon as possible, and take into account the system performance, the length of the running queue is set as two. The number of execution caches is determined by the number of CPU cores, each CPU core corresponds to an execution cache. Cache management module receives tasks submitted by an application, and stores the tasks into submission cache based on their priorities, it also provides execution cache load statistics, task migration and other functions. Task scheduling module selects the highest priority task in the submission cache, if the predecessor list of the task Predecessor_list is empty, according to the execution cache load statistics collected by the cache management module, task scheduling module will choose the execution cache with the lowest load and store the task into its suspending queue. Otherwise, task scheduling module will reselect a highest priority task. Execution component includes thread pool module and computing resource management module. Computing resource management module is responsible for managing hardware resource information. Thread pool module builds and manages thread pool based on the hardware resource information. The number of threads in the thread pool is relative to the number of CPU cores. Implement the binding of thread and CPU core to reduce the consumption of thread operations.

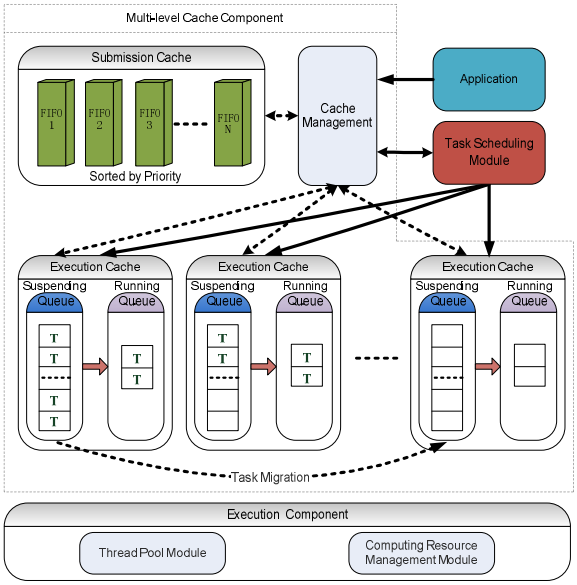

Fig. 1 Multi-task management system diagram

Each thread in the thread pool performs the same procedure as shown in Fig. 2. When the suspending queue and running queue are all empty, the thread will use load statistics and task migration functions to migrate a task to its associated suspending queue from the running cache with the maximum load. Thread waiting function is achieved through semaphore mechanism, and both cases will trigger the thread to continue: 1) scheduling module assigning tasks to execution cache will trigger the semaphore; 2) system exiting will trigger the semaphore. When the system exits, it will set the thread exit flag to true, so that the thread will naturally terminate. Thread executing a task involves two steps: 1) call the Run() function of the task; 2) complete the follow-up work, including: Finish_Semaphore add one, set Task_State as finished, Num_of_Predecessors of subtasks in the Successor_list minus one and delete the task from their Predecessor _list, release the memory space of the task.

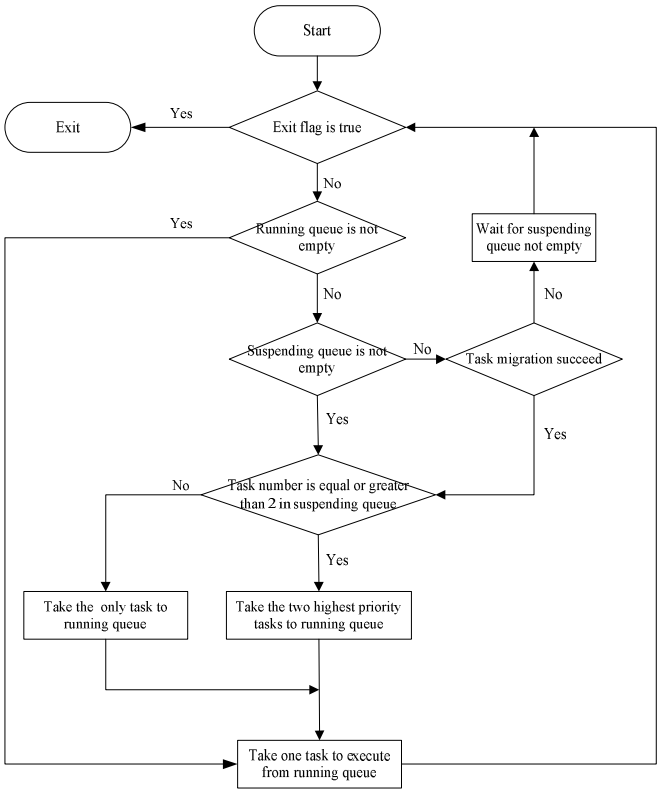

Fig. 2 Thread execution flow chart

Software pipeline in multi-core system

The work pattern of pipeline is to decompose a complex and repetitive process into several sequential and irrelevant sub-processes, and these sub-processes perform in parallel. The sub-process is called pipeline stage. Software pipeline is the mimicry of pipeline by software. It is crucial to ensure the consistency of processing sequence when data blocks pass through each stage of the software pipeline.

A data block is encapsulated in a token, and the token also includes a data block index which is used to indicate the processing sequence of the data block. By passing tokens between the sub-processes of software pipeline, data blocks can be processed stage by stage, and the consistency of processing sequence can be guaranteed. The architecture of software pipeline is shown in Fig. 3. The pipeline consists of several stages. Each stage has its own special process function Operator(token) on data blocks. Each execution of Operator(token) on a data block will be encapsulated as a task, and finally, the task will be submitted to the multi-task management system.

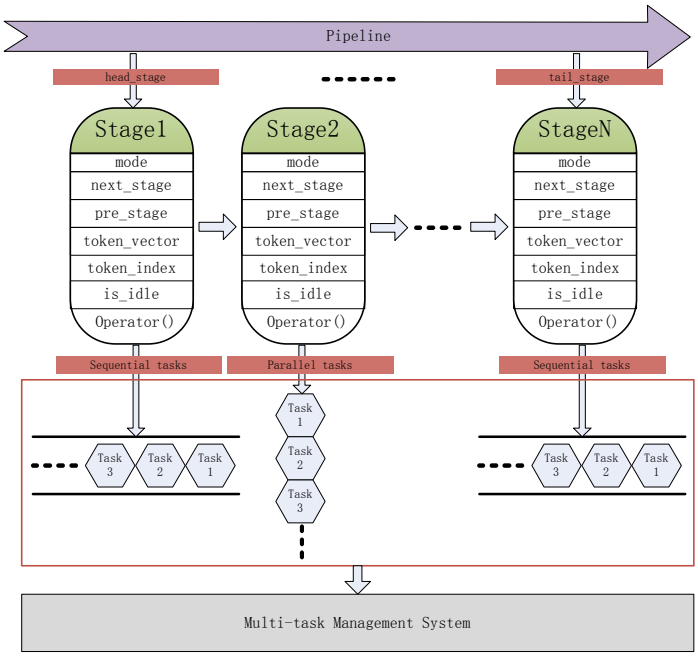

Fig. 3 Software pipeline architecture in multi-core system

The properties of the stage are shown in Table 2. The mode of a stage stage_mode includes serial and parallel, in the parallel mode, the stage can process multiple tokens simultaneously.

Table 2 Stage properties

Property	Commentary
stage_mode	Stage mode（serial/parallel）
next_stage	Next stage in the pipeline
pre_stage	Previous stage in the pipeline
token_vector	The token_vector is using to cache tokens
WorkOnToken_index	The index of the token which the stage is about to process
is_idle	Using to determine whether the stage is idle
Operator(token)	The operation of the stage for a token, token is a input/output parameter

The running of software pipeline is launched by head stage. Under normal circumstances, the Operator(token) function of the head stage is used to generate data blocks, and package the data blocks as tokens. In serial mode, the execution algorithm of the task generated by the head stage is shown in Algorithm 1. The head stage task passes the generated tokens to its next stage, and creates or

caches next stage tasks. The head stage task continuously generates new tokens by recursively creating sub-tasks, and the new tokens are passed to the subsequent stage to maintain the pipeline running.

Algorithm 1 Execution algorithm of the task generated by the head stage in serial mode

```
        //The first token is created by the pipeline
1:      head_stage -> Operator(token)
2:      ++head_stage->WorkOnToken_index
3:      if (token->data_block==NULL)  then
4:          Delete the token
5:      else
6:          if (head_stage->next_stage==NULL)  then
7:              Delete the token
8:          else
9:              if  (head_stage->next_stage->is_idle && token->index ==
                    head_stage->next_stage->WorkOnToken_index)
10:             then
11:                 head_stage->next_stage->is_idle = false
12:                 Create a task of head_stage->next_stage based on the token
13:                 Submit the task to the multi-task management system
14:             else
15:                 head_stage->next_stage->token_vector.push_back(token)
16:             end if
17:         end if
18:         Create a new token
19:         Create a task of head_stage based on the new token
20:         Submit the task to the multi-task management system
21:     end if
```

The execution flow of the task generated by non head stage is shown in Algorithm 2. First, it performs Operator(token) operation on the incoming tokens from superior stage, and passes the processed tokens to the next stage, then creates or caches the next stage tasks. After that, it looks for if there is a token will be processed in its own token_vector, and if so, it will create a task and submit to the multi-task management system, otherwise is_idle will be set to true. In parallel mode, stage task does not be cached but directly submitted to run.

Algorithm 2 Execution algorithm of the task generated by the non head stage in serial mode

```
        //The token is passed from the predecessor
1:      stage -> Operator(token)
2:      ++stage->WorkOnToken_index
3:      if (stage->next_stage==NULL)  then
4:          Delete the token
5:      else
6:          if  (stage->next_stage->is_idle && token->index ==
                stage->next_stage->WorkOnToken_index)
7:          then
8:              stage->next_stage->is_idle = false
                Create a task of stage->next_stage based on the token
                Submit the task to the multi-task management system
9:          else
10:             stage->next_stage->token_vector.push_back(token)
11:         end if
```

```
12:    end if
13:    for  (token_i in stage->token_vector)
14:        if (token_i->index == stage->WorkOnToken_index)  then
15:            Create a task of the stage based on token_i
16:            Submit the task to the multi-task management system
17:            Delete the token_i from stage->token_vector
18:        return
19:        end if
20:    end for
21:    stage->is_idle=true
22:    return
```

Multi-source video processing

Dynamic vehicle identification is selected as the application background to verify the technology of parallel multi-source video processing based on software pipeline. The open source OpenCV library [7] is used to process the video. When using the software pipeline, it is prerequisite to construct the data block structure included in the pipeline token and the Operator(token) functions of the pipeline stages. Aim at the dynamic vehicle identification, and the data block structure includes properties as shown in table 3.

Table 3 Data block structure properties

Property	Type	Commentary
pFrame	IplImage*	The pointer of a frame image
pFrImg	IplImage*	The pointer of the foreground image
pBkImg	IplImage*	The pointer of the background image
frameIndex	unsigned int	The index of the frame image

The processing flow of each video is shown in Fig. 4. The software pipeline is consist of four stages, including capturing the image frame, calculating the foreground and background image, identifying the moving vehicles and outputting images.

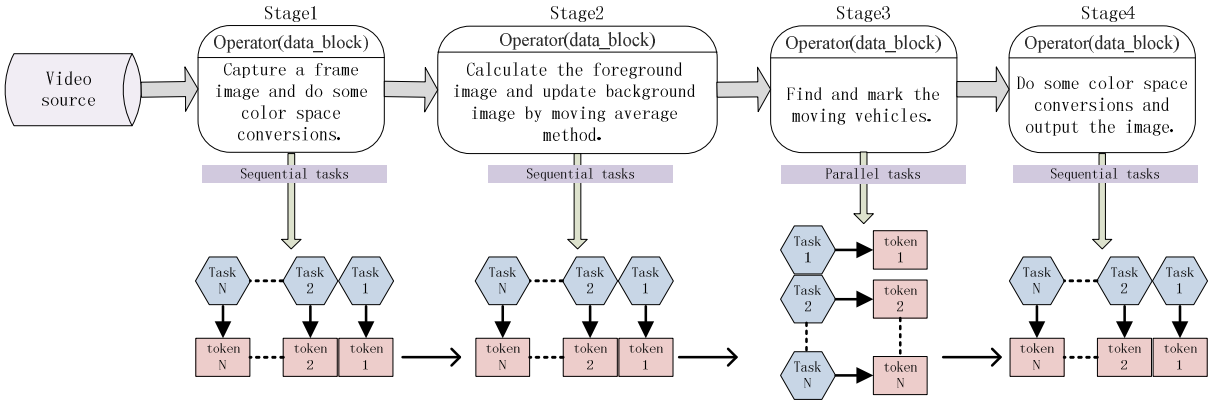

Fig. 4 Processing flow of each video

The hardware platform of the experiment adopts two Intel Xeon E5-2650 CPU, and it has 32 hardware threads of the hardware platform. The memory is 8GB. We process 6 videos simultaneously. In order to better illustrate the effects of the parallel multi-source video processing, we choose the videos with the same content, it contains 750 frames, and the frame size is 640*480. A frame image is intercepted in the course of processing as shown in Fig. 5.

Fig. 5 Result of parallel multi-source video processing

In order to illustrate the effects of the real-time performance improvement when using a complex algorithm to process the multi-source video, add 200 ms delay to the Operator(token) function of stage 3 to simulate the execution of the complex algorithm, set the mode of stage 3 as serial and parallel respectively, and the results are shown in Fig. 6. It can be seen that the real-time performance has been greatly enhanced under the parallel mode.

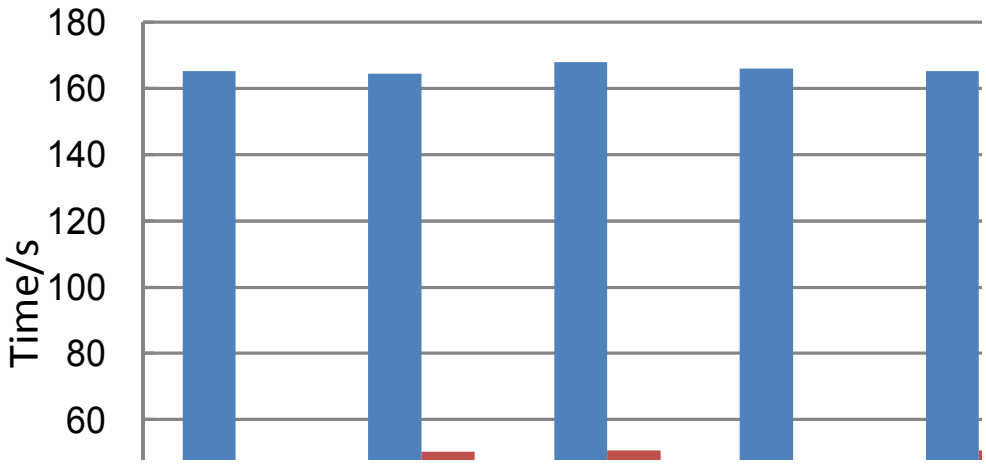

Fig. 6 Execution time in serial and parallel mode

Fig. 7 shows the task number executed by each thread in the thread pool. It shows a better effect on load balance.

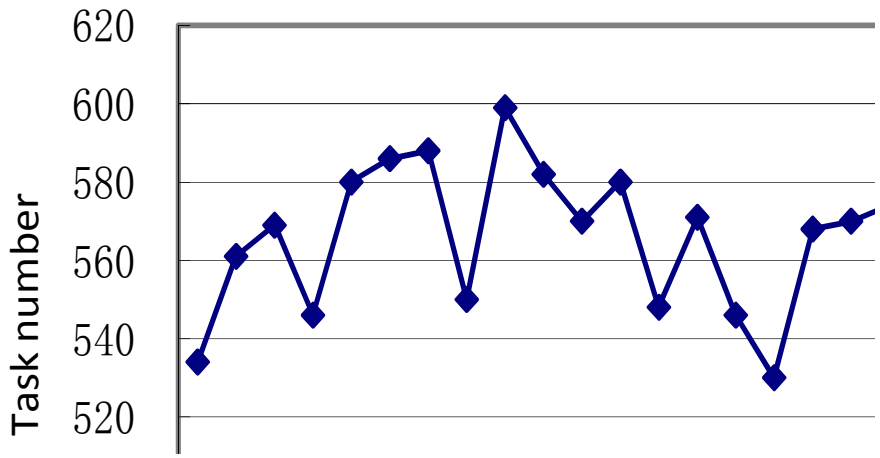

Fig. 7 Task number executed by each thread

Fig. 8 shows the utilization rate of each CPU core at some point, it can be seen that every core was used in the procedure and the utilization rate was very similar.

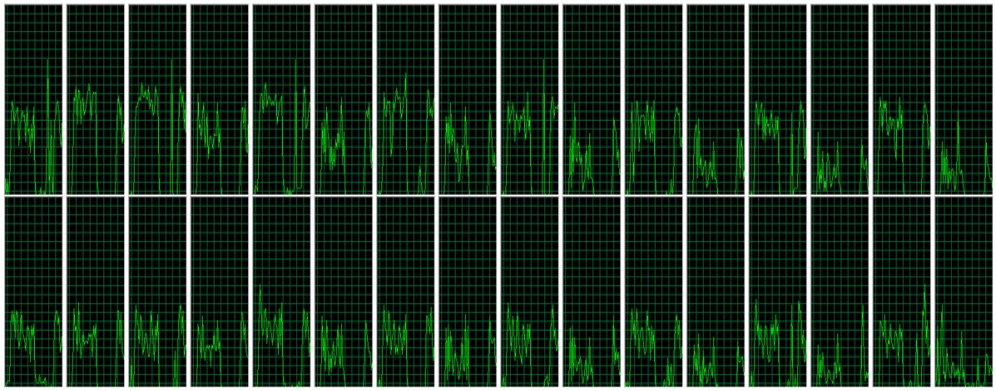

Fig. 8 Utilization rate of each CPU core at some point

Conclusions

The intelligent video processing system has to face the large volume and multi-source data, high-complexity video processing algorithms and harsh real-time constraint. It is necessary to construct the framework of the intelligent video analysis system software, which is propitious to the design of intelligent video analysis algorithms and the coordination of hardware resource allocation. This paper presents a technology of parallel multi-source video processing based on software pipeline. First, a kind of multi-task management system is constructed based on the multi-core CPU, and on this basis, software pipeline is designed out. Finally, it obtains a better anticipative effect by the experiment. But there still exist a number of problems in the processing speed and load balance. Next step, the heterogeneous computing platform will be considered to improve the processing speed. In terms of load balance, this paper adopts the sequence query method to find the lowest load computing resource, when the time of task execution is less than that of the task scheduling, tasks will focus on the front several CPU cores. A better strategy of load balance will be researched later.

References

[1] S. A. El-said: Soft Comput. Vol. 19 (2015), p. 225

[2] S. Nikan, M. Ahmadi : Iet Image Process. Vol. 9 (2015), p.12

[3] S. Li, H. Yu and J. Zhang: Iet Intell. Transp. Sy. Vol. 8 (2014), p.164

[4] A. M. S. Rahma, A. S. Jamil: Int. J. Comput. Appl. Vol. 99 (2014), p. 32

[5] C. Adamson, R. Beare and M. Walterfang: Neuroinformatics Vol. 12 (2014), p.595

[6] J. Wang, C. Vachet, A. Rumple: Front. Neuroinformatics Vol. 8 (2014), p.7

[7] A. Kaehler, G. R. Bradski: Learning OpenCV (Oreilly Media, Sebastopol 2008)