Towards the Parallelization of Shot Detection - a Typical Video Mining Application Study

Eric Li, Wenlong Li, Tao Wang, Nan Di, Carole Dulong, Yimin Zhang

Microprocessor Technology Lab
Intel Corporation
{eric.q.li, wenlong.li, tao.wang, nan.di, carole.dulong, yimin.zhang}@intel.com

ABSTRACT

As digital video data becomes more pervasive, mining information from multimedia data becomes increasingly important, e.g., extraction of goal events in soccer game automatically from the video content. Though all of these advances in multimedia mining have shown great potential in daily life, the huge computational requirement prohibits its wide use in practice. As computer architecture evolves from uniprocessor to the era of multi-core processors, accelerating the multimedia application by exploiting thread level parallelism would be more promising to boost performance and provide more functionality.

This paper presents three different parallel approaches, i.e., task level, data slicing and hybrid parallel scheme, to parallelize shot detection, a widely used application in the video mining system. The hybrid scheme, with the exploration of data level and task level parallelism, delivers much better performance than the other two schemes. Besides, we also employ several software optimization techniques, e.g. data blocking and thread affinity, to improve the performance by more than 50%. Experimental results indicate that there are no obvious parallel limiting factors in the hybrid parallel scheme. It scales well the increasing number of processors, and exhibits13.6x speedup on 16-way processor system.

1. Introduction

Rapid advances in the technology of media capture and storage have contributed to an amazing growth of digital video content. As content generation and dissemination grows explosively, how to help users efficiently search, browse and manage multimedia contents becomes increasingly important, such as video surveillance, detecting highlight event detection, and mining digital home photos and videos in huge volume [2, 9, 11]. These video mining applications are becoming more popular in daily life. However, though it provides a lot of functionalities, the large input data and the underlying complex algorithm require excessive computation than the commodity PC could afford, e.g., it takes several hours to perform a task on a ten-hour MPEG-2 video raw file.

As a reaction to this complexity, most modern microprocessors are equipped with multithreading capabilities to improve the performance of these applications. In 2002, Intel introduced Hyper-threading technology, which enables a processor to execute multiple threads simultaneously. After that, Intel extends Hyper-threading technology to the Dual-core in these years. Current trends in processor technology indicate that the number of processor cores in one IC chip will continue to grow, which is reasonably to assume that the number of cores would follow Moore's Law in future. These advances in personal computers in addition to higher clock frequency provide the necessary computation power for many multimedia applications. Therefore, exploiting thread level parallelism in multimedia mining applications is critical to utilize the hardware resource and accelerate the mining process [10].

In this paper, we analyze the parallel implementation of a typical video mining application, i.e., shot detection, on multiprocessor system, where several parallel schemes, i.e., task level, data slicing and the hybrid parallel scheme are explored in terms of parallel efficiency. By taking advantage of both data and task level parallelism, the hybrid approach delivers the best performance. It first decomposes the input video data into several independent chunks, and then uses task level parallelization on each chunk of data.

The rest paper is organized as follows. Section 2 gives an overview of the whole video mining system and particularly highlights the shot detection application. Section 3 presents all the candidate parallel schemes. The experimental results and application characterizations are reported in Section 4. Finally, Section 5 summarizes the paper.

2. Overview of video mining system

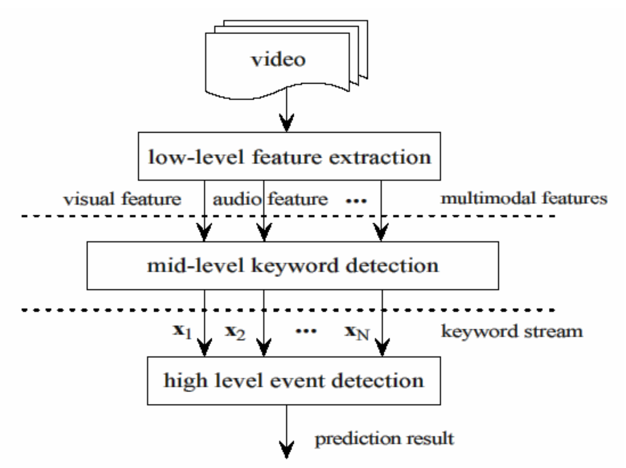

Figure 1. Overview of the video mining system

Personal desktop multimedia mining aims to help users to search, browse and manage their media contents on desktop PC easily. The key components include low-level feature extraction (visual, auditory, and textual features), short boundary detection, scene/story segmentation, video structural summary, high level semantic concept detection, intelligent annotation, indexing and content based image/video retrieval. In order to analyze video content semantically, it is essential to fuse multi-modality information to bridge the gap between human semantic concepts and computer low-level features from both the video sequence and audio streams. Fig.1 shows a typical video mining system framework, which often consists of three layers. First, MPEG decoder decodes the input video into visual, audio and motion streams. Then the shot boundary detection determines the shot boundary according to general features extracted from the visual streams, e.g., color histogram, mean value and standard deviation of each frame's pixel intensities etc. A few key frames are selected from each shot to represent the shot's content. After the key frames are ready, more complex features are extracted from these key frames to detect the mid-level keywords by multi-modality fusion, i.e. visual and audio keywords of face, building, road, excited speech, music and explosion etc. Finally, according to the mid-level keywords representation in time series, we can derive the high-level events, e.g. highlights in sports video.

Similar to text mining based on parsing of word, sentence, paragraph and whole document, parsing a video often consists of four levels, i.e., frame, shot, scene and the whole video sequence. To analyze the video content semantically, shot boundary detection is a prerequisite step [9]. In general, a shot is a set of video frames captured by a single camera in one consecutive shoot action. According to whether the transition between shots is abrupt or not, the shot boundaries are categorized into cut and gradual transition (GT). According to the characteristics of different editing effects, the GTs can be further classified into dissolve, wipe, and fade out/in etc. types [13]. In this paper, we use shot detection algorithm from Tsinghua University who achieved the best result in the world competition of TRECVID 2004 and 2005 [13]. The cut detector uses 2nd order derivatives of color histogram, flash light detector, and GT filter. The GT detector uses the same features as the cut detector plus motion vectors [7, 13].

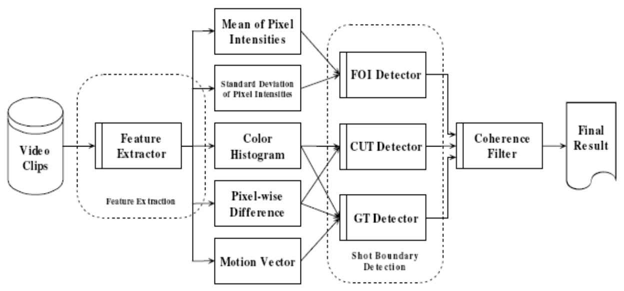

Figure 2. Overview of the shot boundary detection system

Fig.2 depicts the overview of the shot detection in the video mining system. It has 48 bins color histogram in RGB space and 16 bins for each channel. Pixel-wise difference feature, as a supplement to color histogram, is used to represent the spatial information. To detect flashlight effect and monochrome frame, the mean value and standard deviation of each frame's pixel intensities are calculated, at the same time, abrupt change of illumination is detected by tracking the variation of mean gray value. Besides features from uncompressed domain, motion vectors from compressed domain are also extracted to reflect the global motion of a frame [13].

3. Parallel schemes

There are many approaches in parallelizing the shot detection application by exploiting thread level parallelism. In this section, we present our considerations in different parallel schemes study and compare their parallel efficiencies in detail.

3.1. Parallelism study

The shot detection application can be partitioned into three modules: video decoding, feature extraction, and shot boundary detection. We use a MPEG-2 video decoder to decode the input video stream into a number of consecutive frames, followed by the feature extraction module to extract a set of visual features from the decoded frames. The process proceeds until the video decoder reaches the tail of video stream. After that, all the visual features are fed into the shot boundary detection module to compute the final shots. The execution time breakdown indicates that the video decoding and feature extraction module are most time-consuming, constituting around 20% and 80% of total time respectively. The ratio between these two modules also varies with different data input, ranging from 1:5 to 1:3.5 on average. The shot boundary detection module is extremely fast, therefore, is not considered in our parallel framework.

To study the parallel scheme in this application, we use the top-down analysis methodology to analyze the application. In general, multimedia applications tend to use data rather than functional parallelism to take advantage of its embarrassing data parallelism. For video decoder, the straightforward way is to exploit the parallelism at the Group of Picture (GOP) or slice level [1, 6], while for feature extraction, the decoded frames are independent with each other, and hereby, can be processed simultaneously. Though the functional level parallel scheme is also interesting, e.g., different functions, like IDCT, MC, VLD etc. in video decoding, and different features serving as the basic data processing unit, it suffers a lot from the load imbalance among different threads and cannot provide enough parallelism with a large processor number.

Though each module has abundant parallelism separately, when they are put in the whole framework, the overall parallel scheme will be reconsidered since the scalability performance is often limited by the slowest scaling module. In general, exploring parallelism within each module separately cannot offer the maximal performance benefit. In the following, we propose several candidate parallel schemes to discover which one can provide the best parallel performance.

3.2. Task level parallel scheme

The workflow of the shot detection application is very similar to the producer-consumer model, where the video decoder serves as a task producer to generate a sequence of video frames, here we consider these video frames as independent tasks, and they will be put into a shared buffer. Then the feature extraction module reads in the decoded frame from the buffer subsequently and extracts the visual features for the final shot boundary detection.

This scheme perfectly matches the task queue model provided by Intel OpenMP extension [5], which supports a workqueue execution model with taskq and task constructs to allow users to efficiently exploit parallelism among irregular patterns and complicated control structures. Fig.3 depicts a basic OpenMP taskQ working model, when all the threads encounter the taskq pragma, one is chosen to initialize task queue, and then the code inside the taskq block is executed single-threaded. When a task pragma is encountered within a taskq block, the code inside the task block is conceptually enqueued as a task and put in the queue. All the other worker threads will wait to fetch a task from the queue until task is available.

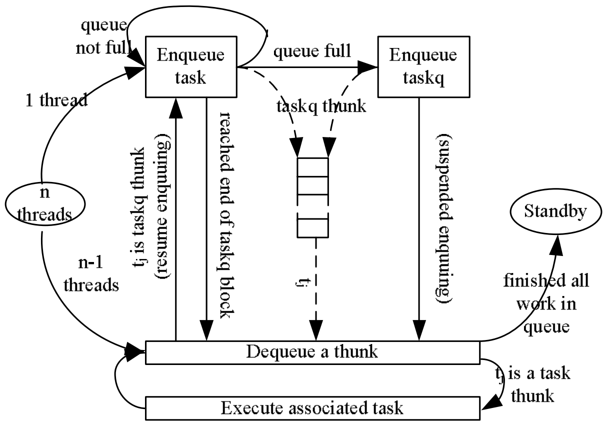

Figure 3. Execution model of task parallel scheme

Fig.4 shows the parallel sample code with OpenMP taskQ primitives. In order to keep the dependency amongst the adjacent decoded frames, we use an "ordered" pragma to guarantee that the video decoder runs in the same order as it executes sequentially.

```
#pragma intel omp parallel taskq ordered
{
  // First phase: video decoding
  #pragma intel omp ordered shared(frame)
  frame = decoding_video();

  // Second phase: feature extraction
  #pragma intel omp task captureprivate(frame)
  extract_feature(frame);
}
```

Figure 4. Sample code for the task level parallel scheme

Besides the convenience of exploiting task level parallelism with TaskQ, the associated dynamic scheduling capability also enables achieving good load balance in the parallel implementation. When the task queue is full, the producer thread will turn into the worker thread, take out one frame from the task queue and perform feature extraction. With the dynamic scheduling mechanism, the task level parallel implementation dramatically overcomes the load imbalance and increases the processor utilization.

Though the idea is conceptually straightforward, it has several scaling limitations. Since the video decoder essentially runs in serial, therefore, the maximal speedup of this parallel scheme is determined by the ratio of feature extraction and video decoding module. As previously mentioned, this ratio ranges from 5:1 to 3.5:1, indicating that the maximal speedup is 5x according to Amdahl's law. Though we can further improve the scaling performance by multithreading the video decoding module, e.g., using the slice level parallelism in one picture to balance the granularity and the parallel efficiency, previous works in MPEG2 decoding parallelization with slice level parallelism [3] indicates that only 1.6x speedup is obtained on a dual-processor system, where the non-parallelizable serial portion significantly degrades the scaling performance. Another alternative solution is to exploit GOP level parallelism, however, it has excessive memory requirement and assumes the GOPs have no interdependencies [1], but this assumption does not always hold for most MPEG-2 bitstreams.

3.3. Data slicing parallel scheme

Since both slice and GOP parallel schemes cannot provide enough parallelism, we turn to explore higher level data parallelism, e.g., partitioning the raw data into several video bitstream chunks. Each thread performs the similar routine as the serial application, decoding the chunk, and extracting features from the decoded frames. Because the raw video stream is split naturally, each thread has to find the new sequence synchronization position. Furthermore, each frame is marked with a time stamp to allow the synchronization between two adjacent threads. The cost of synchronization is light-weighted and has little impact on the final scaling performance.

In the data slicing scheme, each thread executes the whole work on the assigned segment, e.g. if there are totally four threads and the video stream length is 200 megabytes, each thread will operate on a 50M bytes data respectively. This is a natural embarrassingly parallel scheme with data statically assigned to each thread. Since the execution time of video decoding and feature extraction keeps almost the same for the equal segmented chunk, there will be very little load imbalance with moderate number of video decoding threads. Apparently, this scheme is only applicable for the offline mode, where the raw video data is already available in disk, and can be accessed at any position of the file.

In contrast to the task level parallelism, this parallel scheme provides more concurrency without suffering from the parallel limitation of one video decoder. However, it also has some deficiencies. First, it works pretty well for large data set, but cannot support enough threads with very small video bitstream, since it may not even find the sequence synchronization code on such a small piece of data, consequently, it has to traverse and decode the video stream in serial. Second, with more video decoder and sliced data, the parallel overhead increases dramatically. The fine grained data are more likely to cause synchronization overhead and load imbalance among different threads, and becomes more sensitive with the increase of processor number. Finally, all the simultaneous video decoders may excessively use the shared resources, such as bus bandwidth, which results in the performance loss of the whole memory system. It particularly holds true for the SMP system where several processors share only one bus.

3.4. Hybrid parallel scheme

To take advantage of both task level and data slicing parallel scheme, we propose a hybrid parallelization approach to combine these two schemes together. At first, we decompose the video stream into several chunks, then we use the same task level scheme on each data chunk as illustrated in Section 3.2. Fig.5 shows the OpenMP sample code for the hybrid scheme.

```
// Specify the number of threads
int NumThrds;
// Specify the number of threads per group
int NumThrdsPerGrp;
// Compute the number of group
int GrpNum = NumThrds / NumThrdsPerGrp;
#pragma omp parallel for num_threads(GrpNum)
{
  for (int gid = 0; gid < GrpNum; gid++)
  {
     #pragma intel omp parallel taskq  ordered
num_threads(NumThrdsPerGrp)  shared(frame)
     {
       #pragma intel omp ordered
       frame = decoding_video();
       #pragma intel omp task captureprivate(frame)
       extract_feature(frame);
     }
  }
}
```

Figure 5. Sample code for the hybrid scheme

Obviously the hybrid scheme employs a nested parallel model with two hierarchies, and the outer level classifies the threads into different groups. Since nowadays a majority of shared memory multiprocessor systems are using a tree-like architecture hierarchy, e.g., a NUMA system consists of several clusters, and each cluster may have a group of processors sharing the last level cache (LLC) or local memory. Meanwhile, the architecture research also indicates that the future multi-core processor may adopt this tree-like cache hierarchy. Our proposed hybrid parallel scheme can well suit this particular tree-like memory hierarchy system. Generally, we set the thread number in each group to 2 to 4, equal to the number of processors in a cluster when the system has cluster-wise topology. On the other hand, the number does not exceed the maximal ratio as mentioned before, to keep all the tasks busy within each threads group.

It can be easily observed that the task level and data slicing parallel schemes are the two extreme examples of the hybrid parallel scheme. On one hand, it becomes the task level parallel scheme when the number of group is set to 1, on the other hand, it turns out to be the data slicing parallel scheme when each group has only one thread. As a result, the hybrid parallel approach is a good balance between the other two schemes, and flexibly adapt to the particular parallel system in use.

In summary, the hybrid approach has several advantages over the other two schemes. First, it dramatically improves the scalability performance by manipulating multiple task queues in parallel. Second, it shows lower parallel overhead, load imbalance compared to the data slicing scheme, and can be easily transformed to the task level scheme for some extreme small video stream case. Finally, the hybrid scheme offers more flexibility, highly utilizes the system resource, and achieves the optimal scaling performance. It allows the video stream to be partitioned smartly to align with the target system, e.g. on a 4-way hyper-threading enabled system, we can use 4 group of threads and each group contains 2 threads to reuse the data on a HT enabled processor, or alternatively, use 2 groups of threads and each group consists of 4 threads to reduce the video decoding memory contention when it is found to be memory bandwidth limited.

4. Performance characterization and analysis

This section examines the overall performance of parallel shot detection application. The measurements are conducted on a 16-way Intel Xeon shared-memory multiprocessor system. It has 16 x86 processors running at 3.0GHz. Each processor is equipped with 8K L1 data cache, 512KB L2 unified cache, 4MB L3 unified cache, and each 4 processors share a 32MB L4 cache. As for the interconnect, the system uses two 4x4 crossbars. The input data are chosen from the TRECVID data suite [7] with MPEG-2 video format.

We use OpenMP programming model to implement different parallel schemes as illustrated in Section 3, where the taskQ workqueue model, serving as the extension of OpenMP standard by Intel is used. To generate highly optimized executable codes, Intel 8.1 OpenMP compiler tool chain and highly optimized IPP library [4] are used. Furthermore, we also use Intel VTune[5] performance analyzer to identify the hot spots in functional profiling and guide optimization, e.g., data blocking, data structure reorganization, and loop fusion. To characterize the parallel performance, Intel VTune Thread Profiler[5] is employed to quantify the low level metrics, e.g., synchronization, locks, load imbalance, etc.

4.1. Performance optimization

Before studying the parallel performance, we first describe several optimization techniques to improve the application's performance. These optimizations are applied to all the candidate parallel schemes we discussed before.

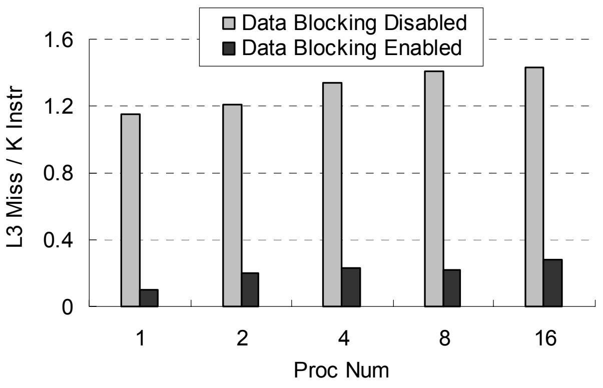

Figure 6. L3 cache misses per 1,000 instructions for the hybrid scheme

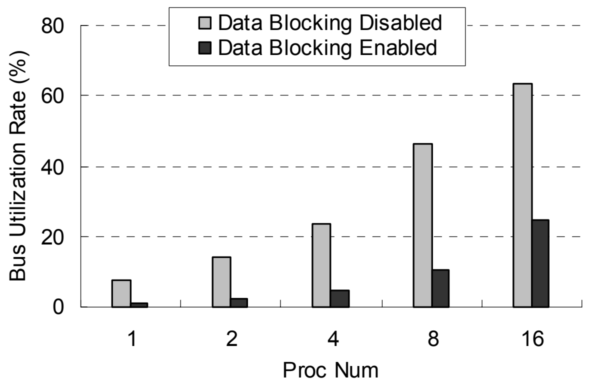

Figure 7. Bus utilization ratio for the hybrid scheme

With in-depth understanding of the application, we can easily find that shot detection operates on a very large data set, e.g., the frequently used memory buffer for MPEG2 data is around 8.0MB, which exceeds the size of L2 or even L3 cache. Fig.6 and Fig.7 depict the L3 cache miss per 1,000 instructions and bus utilization rate for the hybrid scheme. As expected, it experiences very high cache capacity misses, and the L3 cache misses keep at a very high level. On the other hand, the memory bandwidth consumption goes up steadily for 8 processors, but starts saturation on 16 processors. Without data blocking, we can only obtain at most 8x speedup on 16 processors due to the poor memory system performance.

In order to improve the overall scaling performance, we first apply data blocking technique to segment the whole decoded frame into several strips to ensure each piece of strip can fit in the L3 cache. Then the feature extraction module is performed on each strip with a temporary buffer, after the completion for all strips in one frame, the final results will be merged for later use. As plotted in

Fig.6 and Fig.7, the cache misses and memory bandwidth reduce significantly with data blocking optimization. Fig.8 shows the accumulated performance improvement with different optimization techniques. The data blocking optimization improves the performance by more than 40% on 16 processors. Furthermore, we can notice the parallel application gets more benefit than the serial one. The reason can be attributed to the alleviation of the memory bandwidth contention by minimizing the L3 cache misses.

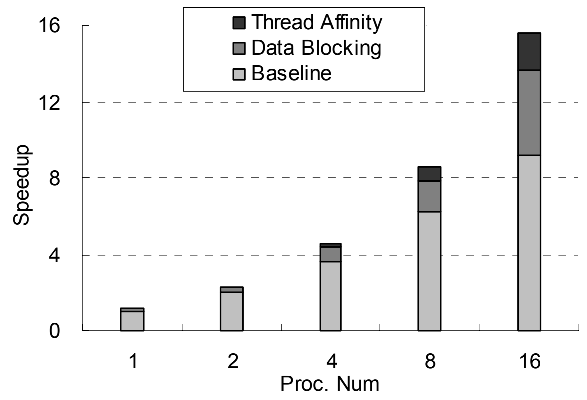

Figure 8. Optimizations for the hybrid scheme (speedups are normalized to the baseline on single processor)

Thread scheduler, an essential component in OS, plays an important role in driving the application to a high performance. There are already a lot of studies on thread scheduling, for instance, [8] gives a comprehensive study on different scheduling techniques and concludes that OS decision has a significant impact on performance with a relatively large processor count. Fig.8 reports the performance benefit when enabling the thread affinity mechanism by binding threads to specific processors, to minimize the threads migration and context switches among processors. In addition, it improves the data locality performance and mitigates the impact of maintaining the cache coherency among all the processors. With thread affinity, the L3 cache misses per 1,000 instructions and context switches per second are reduced by 3% and 10% respectively. Moreover, we also observe that with thread scheduling optimization, the performance improvement for the hybrid scheme is slightly better than the data slicing scheme, which is reasonable since the threads within each group of the hybrid scheme are more tightly coupled compared with the data slicing scheme.

4.2. Scalability performance study

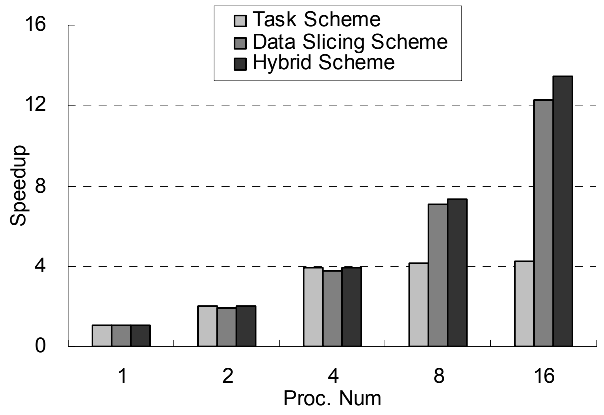

Figure 9. Speedup for three different schemes

Fig.9 reports the speedup of video mining system for three different parallel schemes with 30 minutes MPEG-2 video dataset on up to 16 processors. As expected, the hybrid scheme delivers the highest scalability performance, while the task parallel scheme scales poorly beyond 4 processors. The data slicing scheme lies in the middle, much better than the task level scheme by providing more parallelism, and a little worse than the hybrid scheme due to more load imbalance and excessive use of the memory sub-system. The results perfectly match our analysis in Section 3. Since the hybrid scheme outperforms the other two in terms of scalability performance, we will only study this scheme in the rest of the paper.

To deeply understand the scaling limiting factors, we characterize the parallel performance from the high level general parallel overheads, e.g., synchronizations penalties, load imbalance, and sequential sections, to the detailed memory hierarchy behavior, e.g., cache miss rates and front side bus bandwidth.

Table.1 Thread profiling information for the hybrid scheme on 30m MPEG-2 data

#Threads	Parallel (%)	Seq. (%)	Imbalance (%)	Locks. & Sync(%)	Para o/h (%)
1	99.99	0.01.	0.0	0.0	0.0
2	99.87	0.03	0.09	0.0	0.0
4	99.66	0.08	0.26	0.0	0.0
8	98.59	0.27	1.14	0.0	0.0
16	95.84	1.06	3.06	0.0	0.04

Table.1 gives the general parallel profiling metrics for the hybrid scheme, where "Parallel" means running time inside the parallel region, and "Imbalance" represents time spent waiting for other threads to reach the end of a parallel region. The profiling information reveals that the hybrid parallel scheme has very low synchronization and parallel overhead. The sequential area and load imbalance goes up steadily with the increase of processor number, but remains at a relatively low percentage. Overall speaking, the general parallel limiting factors are insignificant and will not hurt the scalability performance for the hybrid parallel scheme on 16-way system. Furthermore, when comparing the load imbalance performance, we find the hybrid scheme is slightly better than the data slicing scheme, which confirms that slicing the input video stream into smaller pieces will introduce more execution instability, and result in more load imbalance and parallel overhead.

Table.2 Memory characterization for the hybrid scheme on 30m MPEG-2 data

#Threads	L3 cache misses /KInstr.	Cache coherency ratio (%)	Bus utilization ratio (%)
1	0.08	0.0	0.5
2	0.14	3.8	1.7
4	0.24	14.0	4.6
8	0.24	16.2	10.9
16	0.29	17.2	24.2

Besides the general scalability performance factors, memory subsystem also plays an important role in identifying the scaling performance bottlenecks. Table 2 shows the L3 level cache miss rate, cache coherency ratio and the memory bus utilization rate, where the cache coherency ratio is defined as the ratio of bus requests satisfied by another cache over the total number of read requests sent on the bus.

From table.2, it is interesting to see that the L3 cache misses shows an upward trend with increasing number of processors, but stalls at 8 processors. At the same time, the cache coherency ratio also displays the similar trend, which indicates that the data exchange between threads take a large fraction of the total memory traffic within each group of threads. On the other hand, there is little data transfer among different groups of threads, as a matter of fact, the data slicing scheme exhibits very low cache coherency traffics. Along with the increase of L3 cache misses, the bus utilization ratio also ascends linearly with the number of processors, but far from the saturation (e.g., 60% for Xeon MP system) even up to 16 processors. As a result, with low L3 cache misses and moderate memory bandwidth requirement, the parallel application with hybrid scheme delivers very good scalability performance.

4.3 Data set scaling performance study

To investigate the performance with different input data sets, we use a set of MPEG-2 video streams to scale the data in length. As depicted in Fig.10, the parallel performance of the application appears to be closely correlated with the length of input data, e.g., for 3 minutes video sequence, the speedup is linear for four processors, but starts deteriorating when eight or sixteen processors are used. The slowdown for more processors is caused by the decreasing granularity of the work assigned to each processor. The trend is also similar for 30 minutes video data, where the speedup ascends almost linearly for up to eight processors but gets a little slower for 16 processors. Furthermore, when the input data stream is so small in length that we cannot even find a video sequence synchronization code with the sliced data, there will be no parallelism among the input data stream at all and only one video decoder can be used, hence, the scaling performance will be significantly degraded.

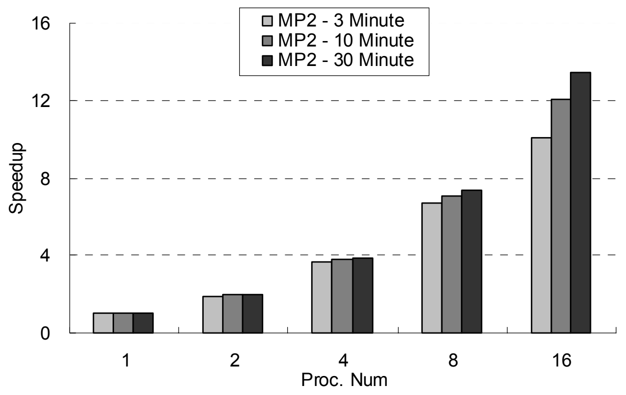

Figure 10. Speedup for different input data for hybrid scheme

To summarize, the hybrid parallel scheme consistently outperforms the task level and data slicing parallel scheme, exhibiting the best scaling performance and highly utilizing the hardware architecture. It is very promising to scale well on more than one hundred processors, especially when we have multiple CPU-cores on the same die in the near future, offering much higher interconnect bandwidth among the CPUs.

5. Conclusion

In this paper, we presented a novel hybrid parallel scheme to accelerate the shot detection application in video mining system. The basic idea is to expose two-level nested parallelism, which first slices the video stream into several chunks, and then exploits the task level parallelism within each data chunk. Experiments show that it achieves much better performance than the other two parallel schemes, i.e, task level and data slicing, and can tailor to different hierarchies of multiprocessor system. In addition, the exploration of different parallel scheme study in shot detection, in another aspect, reveals the future parallelization strategy of more video mining applications with multiple processors.

Besides the comparison among different parallel schemes, we also perform several optimizations techniques to improve its scalability performance, e.g., data blocking and thread affinity, to speed up the execution by more than 50%. The application analysis results indicate that the parallel shot detection application is a computation intensive application, experiencing very low cache miss and memory bandwidth requirement. In addition, there are also no obvious scaling limiting factors, e.g., very low synchronization and load imbalance issues even with up to 16 processors. As a result, parallel shot detection application exhibits fairly good scaling performance, which can be expected to scale well on even more processors.

Our future work will include studying more advanced modules and parallel schemes in video mining system, e.g., integrating the mid-level features in this video mining framework, using different task queuing (centralized, distributed, or hybrid) methods to cope with load imbalanced tasks, and extending the parallel shot detection application to more than 64 processors to characterize its performance on many-core machines.

6. References

[1] A. Bilas, J. Fritts, J. P. Singh. Real-time parallel MPEG-2 decoding in software. In Proceedings of the 11th international symposium on parallel processing, 1997

[2] L.Y. Duan, M. Xu, et al. A mid-level representation framework for semantic sports video analysis, in ACM MM03, pp.33-44, 2003.

[3] M. Holliman, E. Li, Y.K. Chen MPEG Decoding Workload Characterization, in Proc. of Workshop on Computer Architecture Evaluation using Commercial Workloads 2003

[4] Intel Corp. Intel® Integrated Performance Primitives (Intel® IPP). Available at http://www.intel.com/software/products/ipp

[5] Intel Corp. VTune performance analyzer. Available at http://www.intel.com/software/products/vtune

[6] E. Iwata and K. Olukotun. Exploiting coarse-grain parallelism in the MPEG-2 Algorithm, Stanford University Computer Systems Laboratory, Technical Report CSL-TR-98-771, September 1998.

[7] W. Kraajj, A.F. Smeaton, P. Over, TRECVID 2004-An introduction, in TRECVID 2004 Proceedings, http://www-nlpir.nist.gov/projects/trecvid/

[8] Robert C. Kunz. PhD dissertation, Performance bottlenecks on large-scale shared-memory multiprocessors. 2004

[9] C.W. Ngo, T.C Pong, and HJ Zhang. Recent advances in content-based video analysis. In International Journal of Image Graphics, 1(3):445–468, 2001.

[10] F.J. Seinstra, C.G.M. Snoek, D. Koelma, etc. User Transparent Parallel Processing of the 2004 NIST TRECVID Data Set, in IPDPS 2005

[11] S.W. Smoliar, HJ Zhang, Content-based video indexing and retrieval, in IEEE Multimedia, vol.1(2), pp.62-2,1994.

[12] Ernesto Su, Xinmin Tian, Milind Girkar, et. Compiler support of the workqueuing execution model for Intel SMP architectures. In the fourth European workshop on OpenMP (EWOMP), 2002

[13] Jinhui Yuan, Wujie Zheng, Le Chen, etc. Tsinghua University a TRECVID 2004: shot boundary detection and high-level feature extraction, 2004.