NOVEL APPROACHES TO PARALLEL H.264 DECODER ON SYMMETRIC MULTICORE SYSTEMS

Kue-Hwan Sihn[*], Hyunki Baik[*], Jong-Tae Kim[*], Sehyun Bae[**], Hyo Jung Song[*]

*Software Lab., SAIT, Samsung Electronics **Visual Display Division, Samsung Electronics

ABSTRACT

Novel approaches to parallel H.264 decoder for symmetric multicore processors are presented. The basic partitioning of the decoder is coarse-grained and hybrid method of the data partitioning and functional partitioning. We investigate the performance bottleneck of the parallelized decoder, and propose two new approaches, software memory throttling and fair load balancing. The software memory throttling limits the number of cores involved in the parallel motion compensation to achieve better speedup and power-saving. The fair load balancing for deblocking filter reduces load imbalance caused by the conventional static partitioning method. From the evaluation on two different symmetric multicore platforms, proposed approaches show up to 24% of speedup when there is much bandwidth contention.

Index Terms— multicore, H.264, parallel processing, memory bandwidth

1. INTRODUCTION

Recently, multicore processors became the main stream and will continue to expand their application area, which includes multimedia processing. H.264/AVC is now widely accepted codec but notorious for its complexity and internal dependency when considering parallelization on multicore processors. Several reasons such as internal dependency and frequent synchronization in intra predictions (IP), variable length decoding (VLD), and deblocking filter, make it more difficult to parallelize H.264 decoder. Although multicore processors claim the high performance at low power consumption, they can suffer from shared hardware resources like on-chip shared cache and off-chip memory bandwidth. H.264 decoder tends to require much memory bandwidth and it limits the speedup on the multicore processors.

In the multicore era, it is required a new approach to the parallelization of multimedia application like H.264 decoder. In this paper, we design and implement a parallel H.264 decoder that can work on two different platforms, and analyze the major bottlenecks of the parallelization, and propose new approaches to overcome them.

First, we build a state-of-the-art parallel H.264 decoder for the symmetric multicore processors like Intel Core2Quad and ARM MPCore processor. Coarse grained partitioning and the most recent parallelization techniques are applied to find the further bottleneck of the performance.

Second, we propose a bandwidth-saving technique when parallel motion compensation (MC) is being processed. Our approach monitors dynamic status of the overlapping of VLD and MC and limits the allocated bandwidth of MC if VLD slowdown is observed.

Third, we propose the fair partitioning algorithm for deblocking filter to mitigate load imbalance, which is found in coarse-grained parallel deblocking filter. In the recent coarse-grained parallel deblocking filter, workload is spatially partitioned and statically assigned to the processors [1], but if the computation time of the deblocking filter is not equally distributed, load imbalance can arise.

2. RELATED WORK

Parallelization of H.264 decoder on multicore processors has been popular because it requires a large amount of computation power. The approaches can be coarsely categorized by job partitioning granularity.

Macroblock (MB) level parallelization methods are fine-grained ones and provide better load balancing. Tol et al. proposed MB-level parallelization based on wavefront method after completing the VLD function [2]. This fine-grain parallelization technique divides MBs and assigns whole MBs to each core, after the dependency is resolved. This provides good load balancing, but the frequent synchronization can create much overhead.

Another MB level parallelization [3] is proposed for efficient partitioning of MBs for frames that have sparse dependencies. It builds the weighted dependency graph with expected complexity and dependency for each MB, and assigns MBs to each core by using dynamic level scheduling (DLS) algorithm. This algorithm can put extra partitioning overhead, because every time the scheduling graph and partitioning have to be calculated during the run-time.

When considering synchronization cost and relatively small number of cores, coarse-grained partitioning method [4, 5] is preferable if we can reduce the load imbalance in the parallel decoder system.

Our basic parallel decoder is based on coarse-grained partitioning and has some optimization features over [5]. This partitioning method employs data partitioning for MC, IP, and deblocking filter (Deblock), and also adopts functional partitioning between VLD and other stages (MC, IP, Delock).

3. BASIC PARALLELIZATION

The basic parallelization of H.264 decoder is focused on running efficiently on various symmetric multiprocessor (SMP) platforms. The base single-core decoder is from ffmpeg [6] and supports baseline profile. The resulting parallel H.264 decoder can run on Intel Core2Quad and ARM 11 MPCore system, and can decode

variable size input streams including QCIF, CIF, SD and HD resolution.

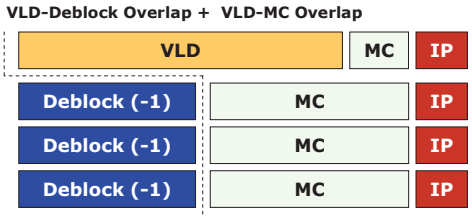
3.1. Overall Pipeline Structure

We apply the hybrid approach that combines both data partitioning and overlapping decoding functions.

The H.264 decoding process includes the following functions: VLD, MC, IP, Inverse Quantization/Inverse Transformation (IQ/IT) and deblocking filter. IQ/IT is always performed together with either MC or IP. From now on, MC means MC/IQ/IT and IP means IP/IQ/IT for simplicity.

Data partitioning provides a good load balancing and easily supports various number of cores, however, applying it for all decoding functions is difficult except MC function due to the data dependencies. In particular, VLD requires sequential execution which is dependent on previous context, it is difficult to parallelize. MC, IP, Deblock functions employ data partitioning technique for multiple cores, details are shown in next section.

Even if VLD function is not finished when Deblock function is completed, MC function, which is the next function of VLD, is executed to reduce the idle time. It is possible because MC function does not have any data dependency between MBs in a frame. We call it as the VLD-MC overlapping. Fig. 1 shows the timeline when both VLD-MC and VLD-Deblock overlapping applied.



Deblock (-1) means the processing of the previous frame

Fig. 1 Overall Pipeline Structure

3.2. Data Parallelism

When VLD function finishes execution for a MB, it fills the decoded MB context structure where stores the information used by MC, IP and Deblock functions, and then these three functions access required parameters in this context and run in data-parallel manners, which are different from each other.

First, MC function can be executed with no dependency synchronization because it does not depend on the neighboring MBs. To support the VLD-MC overlapping, VLD and MC share the one MB context queue (shortly, MB queue) and behave like producer-consumer model. When there are sufficient entries in the MB queue, cores for MC fetch the job as much as specified quantum size. This usually happen after VLD processing is over. However, if there are not sufficient MB queue entries, a core for MC fetches all the remaining MB queue entries to get rid of idle time of MC cores. This aggressive fetching helps the performance for many cases, but sometimes it can slow down VLD processing, which will be discussed in a later section.

Second, data partitioning for IP is not trivial because a decoded MB has dependencies on top, top-left and left neighboring MBs, the decoder has to keep the sequence of processing according to the spatial dependency of IP blocks. Usually, inter frames have less dependencies than intra frames and it is possible to partition the frame in order to have less dependencies [3]. However, this partitioning requires a lot of computation time and is not suitable for the symmetric multicore processor from our experiment, thus, wavefront data partitioning [5] is used both for intra frame and inter frame in our implementation. To reduce the synchronization cost, our decoder switches to the single-core processing if the synchronization cost in the inter frame which has very sparsely located intra prediction blocks excesses the benefit of parallel IP processing. This usually happens because the time needed by IP processing is very small in a inter frame.

Third, we applied coarse-grained data partitioning for deblocking filter [1], because fine-grained partitioning of deblocking filter incurs the inherent synchronization overhead problem of wavefront data partitioning. In this method, post-processing has to be done after parallel filtering for the contaminated boundary area. However, statically partitioned workload can cause load imbalance between cores because the characteristics of the frame differ from frame to frame. A new approach to cope with the load imbalance in the deblocking filer will be discussed in a later section.

4. BOTTLENECKS AND APPROACHES

4.1 Identifying Bottlenecks

4.1.1. Limited Off-Chip Memory Bandwidth

Sometimes, a parallelized application performs worse when working on a multicore processor than on a single-core processor because of the insufficient shared resources such as last-level shared caches and off-chip memory bandwidth. Moreover, despite the slowdown, the cores involved in multicore version consume more power than in single-core version.

Fig. 2 shows a typical example of performance degradation along with core numbers on Intel quadcore processor. Our basic parallel decoder shows gradual speedup along with core numbers for the CIF-size stream, but for the SD-size streams, the slowdown is observed on an Intel quadcore processor system. The cache interference appeared to be relatively low from the cache analysis with the memory traces. We found that the multiple MC threads which have dependency on VLD occupy much off-chip memory bandwidth and can cause VLD's slowdown for some input streams. The slowdown of VLD affects the overall performance.

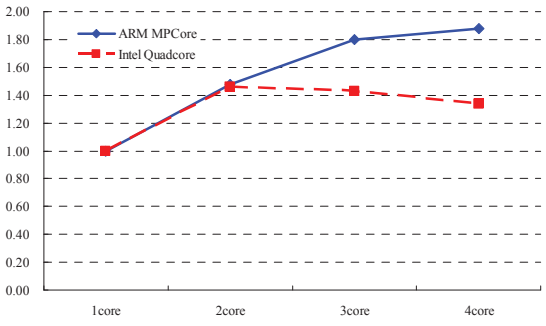


Fig. 2 Performance Degradation on a SD-size Stream

4.1.2. Load Imbalance

Since MC and IP functions which employ dynamic load balancing with small data partitions in a frame, the load is relatively well balanced. In Deblock function, however, load imbalance can arise more frequently because each core is assigned one large, statically partitioned workload from a frame.

In spite of the same partition size, which is assigned to each core, load imbalance can arise due to the characteristic of input data in Deblock function.

4.2 Software Memory Throttling for Parallel MCs

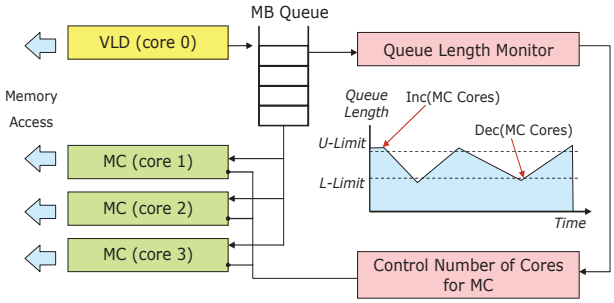


Fig. 3 Software Memory Throttling Mechanism

Since the term *memory throttling* in hardware domain is used to cool down memory system and has different purpose than our technique, we call our method s*oftware memory throttling* to differentiate from hardware method. Software memory throttling method controls the activity of bandwidth-consuming threads, so that the higher priority thread can get more chances to access memory. For example, it is desirable to assign more bandwidth to VLD when VLD and multiple MCs are running concurrently in our parallel H.264 decoder, especially if VLD suffers from insufficient memory bandwidth.

The software memory throttling technique can be applied to any producer-consumer style application running on multicore processor. This technique inspects the shared queue between a producer and consumers and limits the active producers in order to achieve better performance and power saving. The overall mechanism of the software memory throttling is illustrated in Fig. 3.

When consumer (MC) side tries to process few items with many cores, the producer may lack the bandwidth, and the length of the shared queue decrease. In this situation, to avoid performance degradation, our method tries to reduce the number of cores assigned to consumer (MC) side when the queue length reaches the lower limit. This lessens the bandwidth occupied by consumers and effectively assigns more bandwidth to the producer. Fewer working cores also mean power-saving, for the cores which are not assigned can be turned off.

On the other hand, if fewer cores are running for consumers than required, the average queue length will increase. This can also introduce a different type of performance degradation, thus our method tries to increase the number of cores assigned to consumer side when the queue length reaches the upper limit. Consequently, the decoder can maintain near-optimal performance even when the system suffers from the lack of the off-chip bandwidth.

For example, when decoding an SD-size stream, the performance drops at 3 and 4 cores on an Intel quadcore processor (Q6600) thus not scales well as shown in Fig. 2. We found that the queue length is extremely short when the performance degradation is noticed. Because it is difficult to calculate optimal values of upper limit and lower limit mathematically, we choose the values empirically. From our experiment, 20 for upper limit and 16 for lower limit shows good performance for both CIF- and SD-size streams.

4.3 Fair Load Balancing for Deblocking Filters

The load imbalance caused by equal partitioning in deblocking filter is closely related to the characteristic of input data. The execution time of deblocking filter can be varied by the boundary strength (BS) values, which is a number assigned to each MB and subblock. The bigger BS values need the more time for deblocking filter calculation.

However, using BS values for estimating the complexity of Deblock function is inefficient because it requires sequential calculation of BS values before the partitioning and parallel execution. Instead of directly using BS values themselves, MB type can be used as a good approximation of BS values. An intra MB has the larger BS value sum inside the MB and requires around 3 times longer Deblock execution time than inter MB.

From this observation, we propose the improved load balancing method for deblocking filter, in which workloads are partitioned more fairly by estimating the deblocking filter complexity from MB types. When calculating the complexity of one macroblock line, intra blocks are weighted by 3 while inter blocks are not weighted. Our method reduces up to 7% of execution time in Deblock function by fairer load balancing.

5. PERFORMANCE EVALUATION

5.1. Input Streams and Evaluation Environment

We used total two SD-size and three CIF-size H.264 streams for test streams. All the streams are encoded with JM 13.2 encoder, at baseline profile. We used two different symmetric multicore systems, Intel Core2Quad Q6600 and ARM 11 MPCore Platform Baseboard. All experiments were performed on the Linux 2.6.x.

First, we measured the speedup along with the number of cores of decoder to which only basic parallelization is applied. Second, we performed evaluation with the version in which memory throttling and fair load balancing is applied for SD-size streams on Intel quadcore processor.

5.2. Evaluation Result

The performance summary is presented in Fig. 4. The speedup with CIF-size streams (football, foreman, mobile) was relatively good on both systems while the speedup with SD-size streams (3D-animation, action movie) on ARM MPCore was less significant. Moreover, on Intel quadcore processor, the performance at 3, 4 cores are not better than 2 cores, as described in previous sections. The SD-size streams used for evaluation tend to require more memory bandwidth for MC than CIF-size streams. In Fig. 5, the evaluation result of the software memory throttling method and the fair load balancing method is shown.

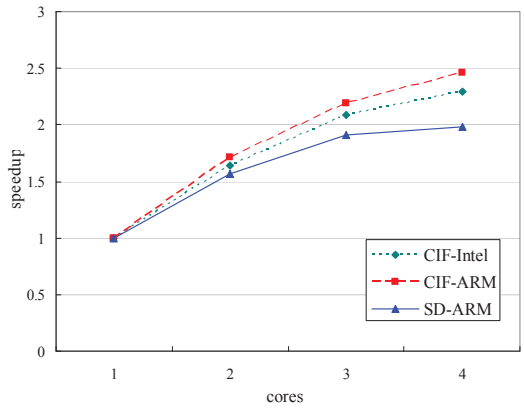


Fig. 4 Performance Summary of the Basic Parallelization

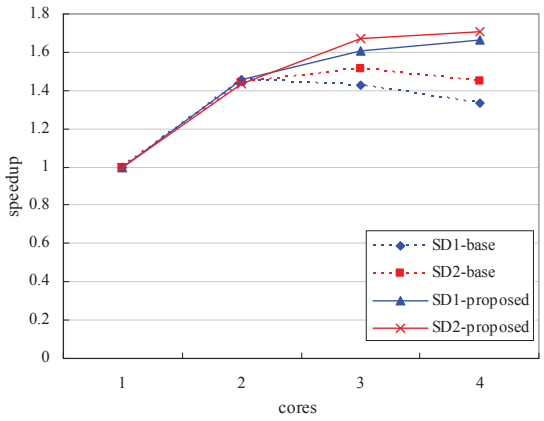


Fig. 5 The Effect of Proposed Methods (on Intel Q6600)

The software memory throttling method effectively controls the number of cores assigned to MC, so that it worked well both with CIF-size streams and SD-size streams. Especially, the memory throttling achieved 17% (action movie) and 24% (3D-animation) of performance improvement in SD-size streams over the basic parallelization, on the Intel quadcore processor. However, the memory throttling failed to achieve meaningful performance improvement on the ARM MPCore processor. We attribute this to the relatively slow speed of VLD on ARM 11 processor; therefore it is not affected much by MC processors. In other words, if a producer shows high computation/communication ratio, the memory throttling approach is not very effective for the overall performance even when the memory bus is saturated. Nevertheless, as long as the performance meets the real-time requirement, the software memory throttling approach gives chance of power-saving.

As for the fair load balancing, even though the execution time of deblocking filter is reduced (Table 1), the overall decoding time is not affected much (below 0.5%). It is because deblocking filter occupies relatively small portion of the overall execution time, and VLD-Deblock overlapping is considered one of the reasons.

Stream	Size	Reduced Execution Time in Deblocking Filter (%), @4 cores	
		Intel Q6600	ARM MPCore
Mobile	CIF	0.53	1.00
Football	CIF	0.79	0.24
Foreman	CIF	3.19	2.26
3D Animation	SD	1.12	1.74
Action movie	SD	7.34	6.68

Table 1 Performance Result of Fair Load Balancing

6. CONCLUSION

In this paper, we designed and implemented a parallel H.264 decoder for multicore platforms. We found that the memory bus bandwidth limits the speedup when several cores for MC processing contend for the memory bandwidth. We proposed the software memory throttling technique to overcome performance degradation, and showed the proposed method is effective for the performance drop when 3 or 4 cores are running. We also proposed a more fair partitioning method for the deblocking filter workload by checking the macroblock types. Various multicore platforms can have different bottleneck points for the scalable performance in multimedia applications, and need more investigation into them.

REFERENCES

[1] S. Yang, S. Wang, and J. Wu, "A Parallel Algorithm for H.264/AVC Deblocking Filter Based on Limited Error Propagation Effect," *Proc. of the Intl. Conf. on Multimedia & Expo (ICME)*, July 2007.

[2] E. B. van der Tol, E. G. T. Jaspers, and R. H. Gelderblom, "Mapping of H.264 decoding on a multiprocessor architecture," *Proc. of SPIE Conference on Image and Video Communications*, Vol. 5022, pp.707-718, January 2003.

[3] J. Chong, N. Satish, B. Catanzaro, K. Ravindran, and K. Keutzer, "Efficient Parallelization Of H.264 Decoding with Macroblock Level Scheduling," *Proc. of the Intl. Conf. on Multimedia & Expo (ICME)*, July 2007.

[4] Y. Kim, J. Kim, S. Bae, H. Baik, and H. Song, "H.264/AVC Decoder Parallelization And Optimization on Asymetric Multicore Platform Using Dynamic Load Balancing," *Proc. of the Intl. Conf. on Multimedia & Expo (ICME)*, June 2008.

[5] K. Nishihara, A. Hatabu, and T. Moriyoshi, "Parallelization of H.264 Video Decoder For Embedded Multicore Processor," *Proc. of the Intl. Conf. on Multimedia & Expo (ICME)*, June 2008.

[6] ffmpeg, http://ffmpeg.mplayerhq.hu/