AVS VIDEO DECODER ON MULTICORE SYSTEMS: OPTIMIZATIONS AND TRADEOFFS

Konstantinos Krommydas[1], Christos D. Antonopoulos[2], Nikolaos Bellas[2], Wu-chun Feng[1]

[1]Department of Computer Science, Virginia Tech, USA
{kokrommy, wfeng}@vt.edu
[2]Department of Computer and Communications Engineering, University of Thessaly, Greece
{cda, nbellas}@uth.gr

ABSTRACT

Newer video compression standards provide high video quality and greater compression efficiency, compared to their predecessors. Their increased complexity can be outbalanced by leveraging all the levels of available parallelism, task- and data-level, using available off-the-shelf hardware, such as current generation's chip multiprocessors. As we move to more cores though, scalability issues arise and need to be tackled in order to take advantage of the abundant computational power.

In this paper we evaluate a previously implemented parallel version of the AVS video decoder on the experimental 32-core Intel Manycore Testing Lab. We examine this previous version's performance bottlenecks and scalability issues and introduce a distributed queue implementation as the proposed solution. Finally, we provide insight on separate optimizations regarding inter macroblocks and investigate performance variations and tradeoffs, when combined with a distributed queue scheme.

Index Terms— AVS codec, task queue, video decoding

1. INTRODUCTION

Advances in video compression techniques and display technology have facilitated high definition video (resolutions up to 1920x1080 pixels) and the first generation of three-dimensional television. Meanwhile, Quad Full High Definition is making its first steps, and motion picture and television engineers are paving the way for Ultra High Definition TV, which will offer unprecedented picture clarity of 7680x4320 pixels.

The prevalent video standard nowadays, namely H.264/AVC, is extensively used for high-definition video coding. One video codec less known in the west world is the Chinese Audio Video Standard (AVS), drafted by the AVS Workgroup [1, 3]. AVS workgroup was established by the Chinese Ministry of National Information Industry and AVS is a national standard. AVS can deliver coding efficiency similar to H264/AVC, and more than two times the coding efficiency of MPEG 2.

These standards can efficiently handle nowadays' typical resolutions and their implementations can provide the desired frame rate, dictated by human vision's real-time requirement of about 30 frames per second. The prospective trends, however, for even higher definitions indicate that the already heavy workload will become heavier, as will the technical complexity of future video encoders and decoders.

Programmers have new tools, hardware and even new computing paradigms in their efforts to overcome such problems. Unfortunately, trying to apply solutions tailored to a small number of cores to more introduces a series of issues. These can be related to the scalability of a particular algorithm itself or can pertain to side-effects on the part of the hardware (e.g. cache-related issues).

This paper builds on our previous work [2], where we found that the hyper-threading feature of Intel Core i7 multiprocessor does not cater to further performance gains, mainly because of contention of the cores' shared resources. Of great importance is the lock-free queue used, whose contention limits any performance gains. As a solution, we propose a distributed queue scheme.

In Section 2, we provide background on the AVS standard and briefly present its base and previous parallel implementation. In Section 3, we present related literature, motivate our work and discuss our contribution. Section 4 introduces our evaluation platform. Section 5, describes our distributed queue approach and the inter macroblock (MB) optimization tradeoffs and provides results. Section 6 concludes the paper with some thoughts and future work.

2. AVS DECODER BACKGROUND

2.1. Base implementation

The AVS standard follows the MPEG 2's basic structure and incorporates similar tools. The decoding process (Fig. 1) entails the entropy decoding stage, intra prediction, the motion compensation (MC) procedure for inter prediction, inverse transform, inverse quantization, as well as a smart

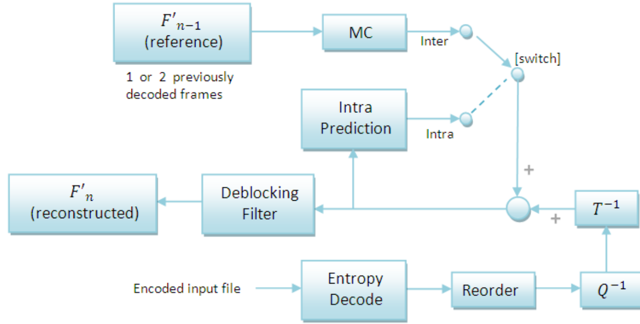

Fig. 1. AVS decoder block diagram

deblocking filter. It sacrifices some of the video quality and coding efficiency in favor of reducing the extra complexity that comes with smaller block sizes [3]. In AVS, intra predictions can be derived from the neighboring pixels in the top left, top, top right and left MBs. A similar dependency set exists for the deblocking filter. For more details on the AVS standard, the reader can refer to [2, 3].

2.2. Parallel implementation

The AVS reference decoder performs decoding in a raster scan order, where MBs are processed from top to bottom and from left to the right. Any parallelization effort has to take into account the dependencies related to intra prediction and deblocking. Inter encoded MBs, namely these that do not depend on MBs of the same video frame, can start decoding as soon as the reference MBs (in previously decoded frames) have been decoded. The latter is ensured by the way frames are decoded (out of order decoding- in the IPBB form). Experiments show that even in high-bitrate encoded videos, inter MBs are abundant and provide an excellent source of available parallelism [2].

Available work, i.e. MBs that can start the decoding (MC) process, is dynamically put into a single shared queue. The worker thread pool takes work from that queue, and when possible bypasses the queue using a tail submit scheme [4] (dependence-driven self-scheduling scheme - Fig. 2). The latter optimization is beneficial for performance in AVS, as it is in H.264. When a thread finds an available task in the queue, it takes and executes it, and updates the dependency numbers of the neighboring MBs. The first one whose dependencies it zeroes, it takes for decoding without putting it in the queue. The others (if any such exist) it puts in the queue for another thread to take.

In [2] we chose not to assign a single thread exclusively for bitstream parsing (BP) and variable length decoding (VLD). In an effort to present a pragmatic, practical approach we did not just decouple BP and VLD from the rest of the decoding and measure pure decoding time. Instead, the first thread to enter the BP/VLD critical section, proceeds with BP/VLD, and starts putting eligible MBs in the queue. This way, the other threads, which would otherwise be blocked, do actual work, which is abundant, especially for inter-decoded frames. Readers interested in the full set of optimizations (sequential code optimizations, vectorization) can refer to our original paper [2].

3. RELATED WORK

Most research has focused on H.264. Since AVS and H.264 are based on the same basic principles, much of the work and conclusions for H.264 are applicable to the AVS (and vice versa). First, we list some of the literature regarding H.264 which relates/applies to our work and then we examine research on the AVS standard, in particular.

Earlier works of Van der Tol et al. [5], and Chen et al. [6] have investigated different levels of parallelization for H.264, albeit with limiting assumptions (i.e. static MB-level scheduling, limited frame-level parallelism). In [7], Mesa et al. extend the above works about parallel scalability of the H.264 decoder. They find that task-level parallelism does not scale well, in contrast to data-level parallelism methods. Distribution of the computation at the MB level proves to be the best solution in terms of scalability and load balancing. In [4], they present findings on a cache-coherent NUMA multiprocessor and comment on the limitations of the single shared queue. They conclude that a work stealing technique or a tail submit method could shift these limitations.

Concerning AVS, not much has been done in terms of optimization for multi-core systems. Instead, there has been enough research on VLSI design of specific kernels, such as Motion Compensation (MC) [8], and Inverse Quantization kernel [9]. Optimization efforts have also been made for the heterogeneous Tensilica SIMD processor, as well as embedded System on Chip designs [10]. However, with multi-core processors and very high definition videos becoming the norm, a detailed study of scalable techniques of the AVS standard to such multiprocessors is necessary.

To the best of our knowledge, the only all-around optimization strategy for the AVS decoder on a commodity multiprocessor system is [2]. Our paper tries to contribute to the limited literature for AVS, extending the aforementioned work by investigating the applicability of already known techniques (from H.264 literature) and by trying to apply new ones. We extend the work of [4] by implementing a distributed task-queue and measuring its performance and compare its performance to the second proposed method in [7] (tail submit). Finally, we investigate some new optimizations regarding inter macroblocks and their applicability to multi-cores with different number of cores.

4. EVALUATION PLATFORM

For our experiments, we used the Intel® Manycore Testing Lab (MTL) [13]. MTL consists of four socket Intel Xeon X7560 processors, totaling 32 cores, each running at 2.26GHz. Each of the four multiprocessors features a large

24MB last-level cache. Total system memory amounts to 64GB. Intel MTL has the Intel® Compiler, Vtune profiler and other useful tools for code inspection and optimization.

The executables were all compiled with the 11.1 version of the Intel® C/C++ Compiler, with the same set of optimization options (for fairness). The Linux kernel version running on our test machine was 2.6.18.

Throughout the paper, results refer to the "Rush hour" [14] encoded video file, which is indicative of the average case for AVS (according to [2]), at FullHD (1920x1080 pixels), at 20Mbps. It contains a typical amount of inter encoded MBs (in P/B frames) in order to showcase some of the optimization techniques. The benchmark video follows the encoding pattern of $\{I(BBP)_5\}_n$ and YUV 4:2:0 format.

5. OPTIMIZATIONS

5.1. Distributed queues

Mesa et al. [7] conclude that a single task queue scheme is one contention point that prevents video decoders (H.264 and AVS respectively) from scaling well at large numbers of cores. The more threads probing the queue, the more the work distribution gets serialized. In this paper, we propose a distributed scheme of multiple task queues, along with a work stealing technique.

In particular we extend the single lock-free queue of [2], by assigning a separate queue to each worker thread. The thread that performs VLD is responsible for assigning the inter MBs (i.e. zero intra dependencies) to each of these queues in a round-robin fashion. In inter frames, where most of the MBs are inter-decoded, this leads to good load balancing. Although decoding time per MB may vary, the work stealing technique takes care of maintaining a good balance. Worker threads that have work available in their queues continue with actual decoding. When they have no work, they resort to work stealing, by referring to the other queues in a linear fashion. The update_dependencies function employs the tail submission technique (queue bypassing) for the first dependency-free MB, and puts the rest of the MBs it finds with zero dependencies in the respective queue. We chose this simple scheme for the update_dependencies function, since the load imbalance it may introduce is negligible, compared to data locality gains.

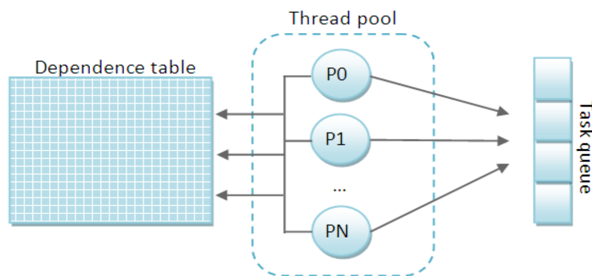

Fig. 2. Parallelization technique schematic.

Since we still use a single dependency table (Fig.2) and other shared data structures (such as queue head pointers), we have to be extra careful of the false sharing effect [11]. Neighboring data in the same cache line may get invalidated without reason, leading to high off-chip memory transfers. This effect is even worse in architectures with larger cache lines. Appropriate techniques, such as padding and proper alignment along cache lines were applied to minimize such negative effects.

5.2. Inter MB decoding optimizations

In [2] we made use of a feature of inter MBs. That is, identify the type of the MB during the VLD phase, and enqueue it if it is of inter type (P or B). This (we call it *P/B optimization*) would intuitively allow for the worker threads to immediately start decoding such enqueued MBs. However, for typical encoding bitrates and videos without special characteristics (e.g. explosions, irregular patterns, sudden movement), inter frames (P/B) consist mainly of inter MBs. This effectively leads to most MBs being enqueued during the VLD phase, and thus limiting the utilization of the tail submit feature. This, in turn, leads to higher contention if a single queue is used and, accordingly, performance deterioration.

On the other hand, when we want to make use of the distributed queues scheme, this optimization makes sense, in that it fills all the queues 'on-the-fly' (during VLD), and the limited use of tail-submit is counterbalanced by the large number of threads working concurrently on their private queues. While for a small number of threads, the tail submit technique (without the P/B optimization) is more efficient, the new scheme (*distributed queues*) overtakes it as more threads are added, and is suspected to scale well for more than the 32 cores available in our experimental platform.

A different approach, in respect to the single queue approach, would be not to enqueue inter MBs on the fly, but zero the number of their dependencies in the dependence table (we name this technique *P/B zero*). Unfortunately, this is practically equal to using the first P/B optimization. The only difference is that tail-submit is used for one in four MBs (on average), whereas in the original P/B optimization, tail submit was utilized even less (talking of inter frames). A combination of the above technique with the distributed queues scheme might be a good compromise for a medium number of cores, but we would need a more efficient technique for the update_dependencies procedure.

5.3. Results

We present results for the above optimization combinations in Table 1, in frames per second (VLD is subtracted from the measurement- we focus on 'pure' MB decoding). Due to space restrictions, we showcase only the 8, 16, 32 core runs.

We observe that for 8 and 16 cores, the distributed queues scheme performs worse than the two single queue ones. This is mainly due to the extra overhead related to the logistics of the queues and worse data locality. A single queue still scales well up to that number of cores. Moreover, P/B zero outperforms P/B opt., as it takes better advantage of the tail submission technique in inter frames. In P/B opt. inter MBs enter the single queue immediately as they are VLDed. This limits the number of MBs bypassing the queue. Yet, P/B zero zeroes inter MBs' dependencies during VLD, but leaves the enqueuing process to the update_dependencies procedure. This way, one of the zeroed MBs bypasses the queue, and the rest are enqueued. This is confirmed by our results (frame-rate and the measured number of MBs that bypass the queue).

When it comes to more threads (note that we use 1 thread/physical core), we can see that the distributed queue scheme starts performing better than the single queue techniques. While the first two techniques show a decline in decoding frame rate from 16 to 32 cores, the distributed queues scheme demonstrates a constant frame-rate increase.

6. CONCLUSIONS/ FUTURE WORK

Video decoders, as many other applications, need to be tackled from a different perspective as the number of cores of future multiprocessors grows. New problems arise and new programming paradigms may have to be eventually employed to continue accruing performance gains.

Effective and more complex queue schemes with architecture-aware work stealing, have to be used in order to avoid contention and make best use of available resources. In our case, we conclude that a distributed queue scheme is useful only after a (big) number of cores. Small multi-core systems will still perform reasonably well with single task queues, combined with techniques that exploit inter MBs' same frame independence property, as those presented.

Additionally, sequential video decoder parts (mainly Variable Length Decoding) constitute a serious bottleneck, and need to be optimized to the fullest to reduce Amdahl's Law implications on parallelism gains.

At the same time, GPU architectures (e.g. Nvidia CUDA) become prevalent in the area of scientific computation and more computationally powerful many-core GPUs become commercially available. Future work revolves around how video decoding algorithms (both as independent kernels, and as a whole) could harness the power of current GPUs, in an efficient CPU-GPU co-scheduling scheme.

Table 1. Results

# cores	P/B opt.	P/B zero	Distr. Queues
8	69 fps	80 fps	59 fps
16	106 fps	115 fps	88 fps
32	104 fps	106.5 fps	112 fps

ACKNOWLEDGMENTS

This work was supported primarily by the Institute for Critical Technology and Applied Science (ICTAS).

We would like to thank the management, staff, and facilities of the Intel® Manycore Testing Lab [12].

7. REFERENCES

[1] AVS Workgroup, *http://www.avs.org.cn/en/*.

[2] K. Krommydas, *et al.*, "Mapping and optimization of the AVS video decoder on a high performance chip multiprocessor," in *Multimedia and Expo (ICME), 2010 IEEE International Conference on*, 2010, pp. 896-901.

[3] G. Wen, "AVS standard - Audio Video Coding Standard Workgroup of China," *Wireless and Optical Communications, 2005. 14th Annual WOCC 2005. International Conference on*, 2005

[4] Mauricio Alvarez, *et al.*, "Performance Evaluation of Macroblock-level Parallelization of H.264 Decoding on a CC-NUMA Multiprocessor Architecture," 4CCC: 4th Colombian Computing

[5] E. van der Tol, Jaspers, E., Gelderblom, R, "Mapping of H.264 Decoding on a Multiprocessor Architecture.," in *Proc. SPIE Conf. on Image and Video Communications and Processing*, 2003.

[6] Y. Chen, Li, E., Zhou, X., Ge, S, "Implementation of H. 264 Encoder and Decoder on Personal Computers," *Journal of Visual Communications and Image Representation,* vol. 17, 2006.

[7] M. A. Mesa, *et al.*, "Scalability of Macroblock-level Parallelism for H.264 Decoding," in *Parallel and Distributed Systems (ICPADS), 2009 15th International Conference on*, 2009, pp. 236-243.

[8] Z. Dajiang and L. Peilin, "A Hardware-Efficient Dual-Standard VLSI Architecture for MC Interpolation in AVS and H.264," in *Circuits and Systems, 2007. ISCAS 2007. IEEE International Symposium on*, 2007, pp. 2910-2913.

[9] S. Bin, *et al.*, "An implemented VLSI architecture of inverse quantizer for AVS HDTV video decoder," in *ASIC, 2005. ASICON 2005. 6th International Conference On*, 2005, pp. 244-247.

[10] J. Xin, *et al.*, "AVS video standard implementation for SoC design," in *Neural Networks and Signal Processing, 2008 International Conference on*, 2008, pp. 660-665.

[11] J. Torrellas, *et al.*, "False sharing and spatial locality in multiprocessor caches," *Computers, IEEE Transactions on,* vol. 43, pp. 651-663, 1994.

[12] Home: www.intel.com/software/manycoretestinglab
Intel® Software Network: www.intel.com/software

[13] Raw benchmark videos: ftp://ftp.ldv.e-technik.tu-muenchen.de/pub/test_sequences/1080p/