A NOVEL PARALLEL H.264 DECODER USING DYNAMIC LOAD BALANCE ON DUAL CORE EMBEDDED SYSTEM

Ding-Yun Chen, Chen-Tsai Ho, Chi-Cheng Ju, Chung-Hung Tsai

Multimedia Development Division, Mediatek Inc.
No. 1, Dusing Rd. 1, Hsinchu Science Park, Hsinchu, Taiwan
{dynamic.chen, ct.ho, cc.ju, ch.tsai}@mediatek.com

ABSTRACT

The dual-core environment is more and more popular in embedded system recently. The limited buffer and limited bandwidth are critical for parallel algorithm in embedded system. This paper proposes a novel parallel algorithm using functional partitioning with dynamic load balance for video decoder. The video decoding flow of each macroblock is dynamically separated for different cores according to the buffer queue level. The extra intercommunication buffer size requires only 1.6% buffer size of traditional data partitioning algorithm for 720p decoder. The speed-up ratio is 1.74 times in average compared to original single thread code. The experiment result shows the proposed algorithm can real-time decode H.264 720p high profile on ARM Cortex-A9 400MHz dual-core system.

Index Terms— parallel algorithms, video codec, dynamic load balance, H.264 decoder

1. INTRODUCTION

The multi-core system is more popular in embedded system recently. To improve the performance of existing single thread software, different parallel algorithms need to design according to the characteristic of different applications. Common issues of designing parallel algorithms include load balance, synchronization overhead, memory access contention, etc. In embedded system, the communication buffer size is also an important issue because the embedded systems are characterized by limited memory and limited bandwidth.

Most existing video standard didn't design for parallel computing, so the data dependencies exist in most video format. Several parallel algorithms using different level partitioning are proposed to avoid the data dependencies. The frame level partitioning is not suitable for the embedded system because the inter prediction and buffer size limitation. The open source FFmpeg [10] uses slice level partitioning. However, only MPEG-2 format is useful because MPEG-2 standard defines each MB row is an individual slice. Other video standard, such as MPEG-4, H.264, etc., didn't force multiple slices within a frame. Unfortunately, most commercial bitstream are only one slice within a frame, so the slice level partitioning cannot speed-up in a multi-core system in most cases. The macroblock level partitioning is more suitable for most cases, but data dependencies of left, top-left, top, and top-right macroblock should be considered. In addition, the entropy decoding must be in raster scan order. Those data dependencies of macroblock level cause the challenge of the parallel algorithms for video decoder.

The parallel algorithms using macroblock level can be broadly divided into three different partitioning categories: functional partitioning, data partitioning and mixed partitioning [1]. The functional partitioning takes each thread as a distinct function in a pipelined fashion, and communicates between tasks in an explicit way. The data partitioning use different threads to execute the same function for different parts of input data simultaneously. The mixed partitioning combines the functional partitioning and data partitioning, and processes functional parallelism at coarse granularity and applies data parallelism inside those granularities. In embedded system, the major problem of data partitioning [6], [7] and mixed partitioning [8], [9] is the communication buffer size. The data partitioning needs a VLD buffer in double whole frame size because each VLD coefficient needs two bytes to store. The mixed partitioning needs more communication buffer including a VLD buffer and a prediction buffer in triple whole frame size.

For functional partitioning, Seitner et al. [2], [3] divide the video decoder functionality into a parsing part and a reconstruction part for dual-core environment. A high level simulation is proposed to estimate the performance. Y. Kim et al. [4] propose dynamic load balance in functional partitioning, which off-load the motion compensation to another core if the previous macroblock is in bi-directional inter prediction mode. However, quarter frame buffer size is required for their pipeline structure. M. Kim et al. [5] propose dynamic load balancing method for the functional partitioning to their own dual-core DSP. Their whole system consists of the dual-core DSP, VLD hardware, MC

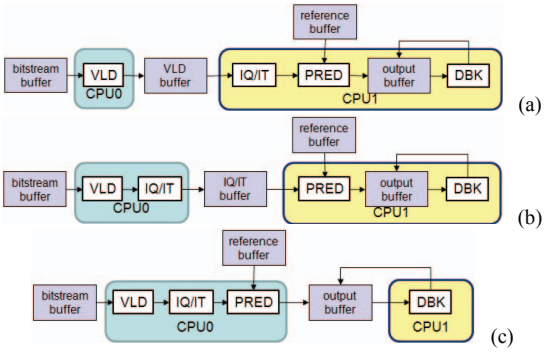

Fig. 1. (a)~(c) Three kinds of different fixed functional partitioning for dual-core system

hardware, deblocking hardware and a SDRAM. So, the dual-core DSP handle only few parts of video decoder flow. The first DSP core can offload inverse transformation/quantization for intra macroblock or boundary strength calculation for inter macroblock from second DSP core if the communication buffer level is larger than threshold. In this paper, we propose a novel functional partitioning algorithm using dynamic load balance for pure software video decoder. The intercommunication buffer is to communicate between dual cores, and requires only 32 macroblock for 720p decoder, which is less than 2% buffer size of the data partitioning and the mixed partitioning.

2. FUNCATIONAL PARTITIONING WITH DYNAMIC LOAD BALANCE

The main idea of the proposed algorithm is that the video decoding flow of each macroblock is dynamically separated for different cores according to the communication buffer level. The separation should consider the data dependencies of different sub-module in different macroblock. Section 2.2 describes the algorithm to avoid the data dependencies when separating the decoding flow.

2.1. The proposed decoding flow

The video decoding flow can be divided into four sub-module: entropy decoding (VLD), inverse quantization and transformation part (IQ/IT), prediction (PRED) and deblocking (DBK). In dual-core system, it's intuitive to have three different ways for fixed functional partitioning, as shown in Fig. 1 (a)~(c). However, the executed time of each sub-module is variable according to the input bitstream. The fixed functional partitioning will face the load imbalance of different cores.

Dynamic load balance is one of the best approaches to solve the load imbalance. For dual core, Fig. 2 shows the combination of three different partitioning ways from Fig. 1(a)~(c). The CPU 0 can dynamically select one of the three partitioning ways for each macroblock according to the buffer queue level. When the queue level is low or empty,

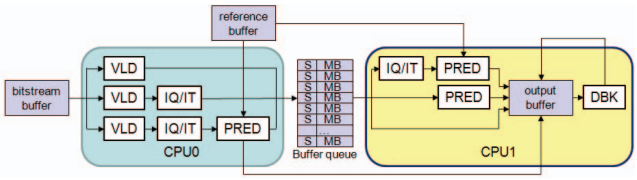

Fig. 2. Functional partitioning with dynamic load balance

CPU 0 needs to finish the job quickly and send it to the queue to avoid CPU 1 waiting for queue. On the other hand, when the queue level is high, it means the CPU 1 is too busy, so the CPU 0 can do more to offload the job from CPU 1.

The buffer queue is for sending the data and information from CPU 0 to CPU 1. Each buffer queue slot includes a flag, macroblock header information and the macroblock data. The CPU 1 read the flag to know what is the partitioning of this macroblock by CPU 0. The macroblock data can be the result of VLD or IT or PRED according to the flag. The PRED result from CPU 0 can also put to output buffer directly. In our implementation, the buffer size of each queue slot is 1420 bytes.

The size of buffer queue should be less than a macroblock row so that the data dependencies need not to check to reduce synchronization overhead. That is, the small queue size can ensure top-left, top and top-right macroblock are already done when queue is available to push or pop. Another advantage of the small queue size is to reduce the external memory bandwidth and latency, which is important especially in embedded system. That is, the whole buffer queue can be hold inside L1/L2 cache so the data communication between two CPUs is via hardware cache coherence. In our implementation, the queue size is 32 macroblock for 720p decoder, and the total communication buffer size requires only 44.4 Kbytes. The extra required buffer size is quite small when comparing with other parallel algorithms. For 720p decoding, the data partitioning [6] and mixed partitioning [8], [9] require 2.64 Mbytes and 3.96 Mbytes respectively. So, the buffer size of the proposed algorithm for 720p decoder is 1.6% and 1.0% buffer size of data partitioning and mixed partitioning respectively.

2.2. Partitioning for prediction data dependencies

The proposed dynamic partitioning algorithm has large scalability for each core. That is, the CPU 0 can do from VLD only to VLD/IQ/IT/PRED, and the CPU 1 can do from IQ/IT/PRED/DBK to DBK only. Most previous works of dynamic load balance in video decoder only offload a small function. For example, parts of motion compensation are divided to other core for dynamic load balance in [4]. IQ/IT for intra macroblock or boundary strength calculation for inter macroblock are offloaded to other core for dynamic load balance in [5]. The advantage of our large scalability of

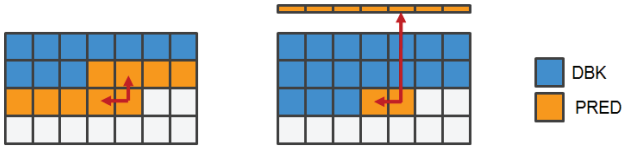

Fig. 3. (a)(b) Two ways of intra prediction data

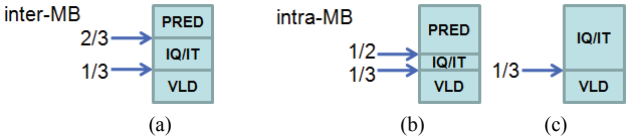

Fig. 4. The queue level threshold for dynamic load balance for (a) inter macroblock and (b)(c) intra macroblock

dynamic partitioning is the load balance for different kinds of input bitstream. However, the large scalable partitioning has several data dependencies especial in PRED sub-module.

First, to decouple VLD with other sub-module, the motion vector prediction and intra mode prediction should be calculated together with VLD. This is because the results needs by VLD sub-module for next macroblock. As a result, VLD sub-module in CPU 0 can go ahead and doesn't care if the prediction of left macroblock is done or not.

Next, the intra prediction needs to reference the PRED result of top macroblock row before DBK. Fig. 3 (a) shows one method that the DBK always delay one macroblock row, and Fig. 3 (b) shows another method that the PRED backup one pixel row of top macroblock row before doing DBK. In this case, the DBK has to always delay only one macroblock because the intra prediction also needs to reference the PRED result of left macroblock. In our implementation, we use the later one because our experiment shows the later one perform slight better.

Finally, because the intra prediction needs to reference the PRED result of left macroblock, the partitioning criterion should make sure the PRED of left macroblock and current intra macroblock run in the same CPU. On the other word, if CPU 1 does PRED of left macroblock and CPU 0 does PRED of current macroblock in intra prediction, the CPU 0 needs to check if left macroblock is done from CPU1. To reduce the synchronization overhead, once CPU 0 partition PRED to CPU 1, the PRED of residual right macroblock in the same macroblock row needs also to partition to CPU 1 if the macroblock is in intra. The inter macroblock has no this limitation, and can be partitioned into different CPUs according to queue level.

Fig. 4 shows the queue level threshold of the dynamic functional partitioning. For inter macroblock, CPU 0 will do VLD only if the queue level is less than 1/3 queue size, and do VLD/IQ/IT if the queue level is within 1/3~2/3 queue size, and do VLD/IQ/IT/PRED if the queue level is larger than 2/3 queue size. For intra macroblock, the threshold moves from 2/3 to 1/2 for PRED to keep more PRED in CPU 0 if the queue level is higher than 1/2. Fig. 4 (c) shows once PRED is assigned by CPU1, CPU 0 never does PRED in the same macroblock row for intra macroblock.

Table 1. ARM Versatile Express Platform Specification

Core Tile Name	CoreTile Express A9x4
CPU Type	Cortex™-A9MPCore
Num CPUs, Speed	Quad, 400MHz
Coprocessors	NEON
L1 cache I/D	32KB/32KB
L2 cache	512KB
SDRAM	1GB DDR2, 266MHz, 32-bit
Internal AMBA bus speed	200MHz
External AMBA bus speed	50MHz (M), 30MHz (S)

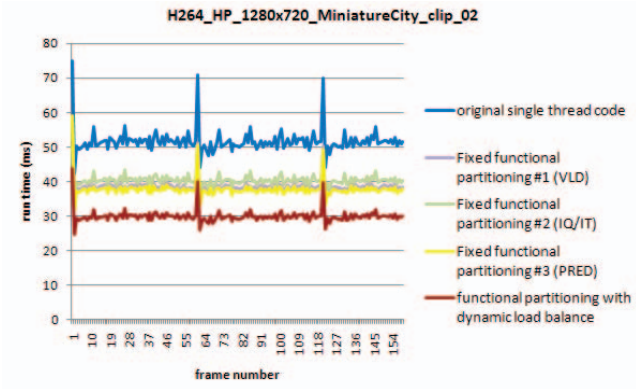

Fig. 5. Execution time of different functional partitioning

3. EXPERIMENTAL RESULTS

The experiments run on the ARM Versatile Express platform [11], which including Cortex-A9 quad-core with NEON. Table 1 shows the detail specification. We use only dual Cortex A9 cores in our experiments. The operation system runs Linux and supports symmetric multiprocessing (SMP). In our implementation, all the threading and synchronization use Linux pthread API, including pthread_xxx(), sem_xxx(), pthread_mutex_xxx().

The test cases for this experiment are 20 H.264 high profile bitstream with CABAC in 1280x720 resolution and 166 frames in average. The original single thread code comes from a commercial H.264 decoder, which has been optimized by NEON instruction level parallelism.

Fig. 5 shows the experimental results of a typical case. The original single thread code takes 51.6ms to decode a frame in average, that is, decode 19.4 frames per second (fps). Three fixed functional partitioning from Fig. 1 (a)~(c) take 38.5ms, 40.3ms, 37.7ms to decode a frame, and the performance are 26.0 fps, 24.8 fps, 26.5 fps, which improve only 1.34, 1.28 and 1.37 times respectively. The proposed functional partitioning takes 29.9ms to decode a frame, and the performance is 33.5 fps, which can improve 1.73 times. That is, no matter which fixed functional partitioning is selected according to the frame or slice header information, the dynamic load balance algorithm will perform better.

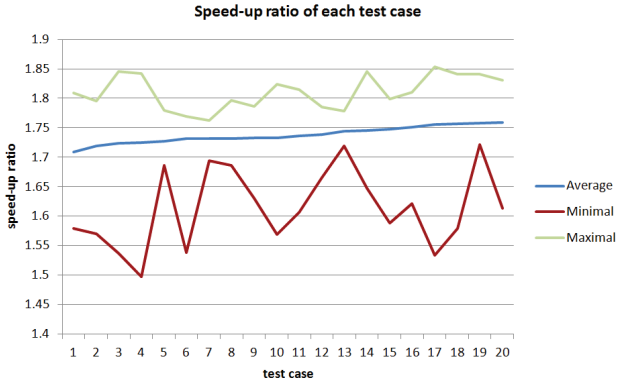

Fig. 6. Average, minimal and maximal speed-up ratio of each test case

Fig. 6 shows the average, minimal and maximal speed-up ratio of all 20 test cases. The average speed-up ratio is 1.74 times from all test cases. The minimal speed-up ratio is 1.49 times from all test frames. This is because, however, the frames are too simple, as shown in Fig. 7. Fig. 8 shows that the first several frames have low speed-up ratio. The run time is fast enough even the speed-up ratio is not high. As we can see, the original single thread code takes 51.1ms to decode a frame in average, and the performance is 19.6 fps. The proposed algorithm takes 29.5ms to decode a frame in average, and the performance is 33.9 fps. So the experiments show the algorithm can real-time decode H.264 720p high profile with CABAC on dual Cortex-A9 cores in 400MHz.

4. CONCLUSION

This paper proposes a novel parallel algorithm for H.264 decoder using functional partitioning with dynamic load balance. The proposed algorithm can also be applied to other video format decoders. The data partitioning and mixed partitioning algorithm are not suitable to embedded system because the cost of extra huge buffer size. The buffer size of our proposed algorithm requires only 32 macroblock for 720p decoder, and is only 1.6% and 1.0% buffer size of data partitioning and mixed partitioning respectively. The speed-up ratio of our algorithm is 1.74 times in average compared to original single thread code. The experiments show the algorithm can decode H.264 720p on ARM 400MHz dual Cortex-A9 cores in real time. The future work is to extend the algorithm to more multiple cores.

5. REFERENCES

[1] J. Eijndhoven, J. Hoogerbrugge, M.N. Jayram, P. Stravers and A. Terechko, "Cache-Coherent Heterogeneous Multiprocessing as Basis for Streaming Applications", Dynamic and Robust Streaming in and between Connected Consumer-Electronic Devices, vol. 3, pp. 61-80, SpringerLink, 2005.

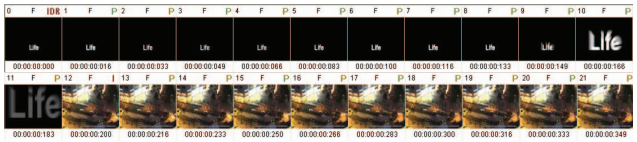

Fig. 7. First 22 frames of test case with low speed-up ratio

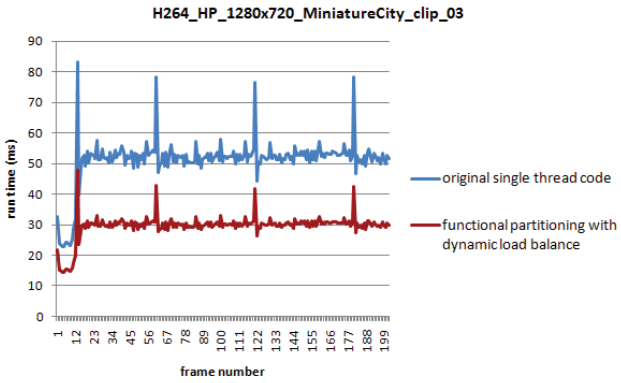

Fig. 8. Test case from Fig. 7 has low speed-up ratio.

[2] F. H. Seitner, R. M. Schreier, M. Bleyer and M. Gelautz, "A macroblock-level analysis on the dynamic behaviour of an H.264 decoder", Proc. of IEEE Intl. Symp. on Consumer Electronics (ISCE), pp. 1-5, June 2007.
[3] F. H. Seitner, M. Bleyer, M. Gelautz, R. M. Beuschel, "Development of a High-Level Simulation Approach and Its Application to Multicore Video Decoding", IEEE Trans. on Circuits and Systems for Video Technology, vol. 19, no. 11, pp.1667-1679, Nov. 2009.
[4] Y. Kim, J.-T. Kim, S. Bae, H. Baik and H. J. Song, "H.264 decoder on embedded dual core with dynamically load-balanced functional partitioning", Proc. of IEEE Intl. Conf. on Multimedia and Expo (ICME), pp. 1001-1004, Apr 2008.
[5] M. Kim, J. Song, D. Kim and S. Lee, "H.264 decoder on embedded dual core with dynamically load-balanced functional paritioning", Proc. of IEEE Intl. Conf. on Image Processing (ICIP), pp. 3749-3752, Sept 2010.
[6] E. B. Van Der Tol, E. G.T. Jaspers, and R. H. Gelderblom, "Mapping of H.264 decoding on a multiprocessor architecture", Proc. of the SPIE, vol. 5022, pp. 707–718, May 2003.
[7] C. Meenderinck, A. Azevedo, M. Alvarez, B. Juurlink and A. Ramirez, "Parallel Scalability of H.264", Proc. of workshop on Programmability issues for multicore computers (MULTIPROG), jan 2008.
[8] K. Nishihara, A. Hatabu and T. Moriyoshi, "Parallelization of H.264 video decoder for embedded multicore processor", Proc. of IEEE Intl. Conf. on Multimedia and Expo (ICME), pp. 329-332, Apr 2008.
[9] K.-H. Sihn, H. Baik, J.-T. Kim, S. Bae and H. J. Song, "Novel approaches to parallel H.264 decoder on symmetric multicore systems", Proc. of IEEE Intl. Conf. on Acoustics, Speech and Signal Processing (ICASSP), pp, 2017-2020, Apr 2009.
[10] The FFmpeg website. [Online]. Available: http://ffmpeg.org/
[11] The ARM Versatile Express platforms. [Online]. Available: http://www.arm.com/products/tools/development-boards/versatile-express/index.php