A Parallelization Technique with Integrated Multi-Threading for Video Decoding on Multi-core Systems

Jung-Hyun Hong[1], Won-Jin Kim[2] and Ki-Seok Chung[3]
[1] Department of Electronics and Computer Engineering, Hanyang University, Seoul, Korea
[e-mail : jhhong34@gmail.com]
[2] Design Solution Lab., DMC R&D Center, SAMSUNG Electronics
[e-mail : won-jin.kim@samsung.com]
[3] Department of Electronic Engineering, Hanyang University, Seoul, Korea
[e-mail : kchung@hanyang.ac.kr]
*Corresponding author: Ki-Seok Chung

Received July 4, 2013; revised August 26, 2013; accepted September 21, 2013; published October 29, 2013

Abstract

Increasing demand for Full High-Definition (FHD) video and Ultra High-Definition (UHD) video services has led to active research on high speed video processing. Widespread deployment of multi-core systems has accelerated studies on high resolution video processing based on parallelization of multimedia software. Even if parallelization of a specific decoding step may improve decoding performance partially, such partial parallelization may not result in sufficient performance improvement. Particularly, entropy decoding has often been considered separately from other decoding steps since the entropy decoding step could not be parallelized easily. In this paper, we propose a parallelization technique called Integrated Multi-Threaded Parallelization (IMTP) which takes parallelization of the entropy decoding step, with other decoding steps, into consideration in an integrated fashion. We used the Simultaneous Multi-Threading (SMT) technique with appropriate thread scheduling techniques to achieve the best performance for the entire decoding step. The speedup of the proposed IMTP method is up to 3.35 times faster with respect to the entire decoding time over a conventional decoding technique for H.264/AVC videos.

Keywords: H.264/AVC, video decoder, multi-core systems, parallel processing

"This research was supported by the MSIP (Ministry of Science, ICT & Future Planning), Korea, under the ITRC (Information Technology Research Center) support program supervised by the NIPA (National IT Industry Promotion Agency)" (NIPA-2013-H0301-13-1011)

http://dx.doi.org/10.3837/tiis.2013.10.009

1. Introduction

Demand for high resolution video processing techniques is rapidly increasing as high-definition digital broadcasting services are widely provided. Therefore, standards and techniques for video compression and decoding are being actively developed. One of the most popular video codec standards is H.264/AVC. One of the merits of H.264/AVC is that it is capable of providing good video quality at substantially lower bit rates than previous standards. However, it is very challenging to achieve high performance by a software implementation because decoding is very complex.

The Joint Collaborative Team on Video Coding (JCT-VC) is currently developing the next generation of video coding standards called High Efficiency Video Coding (HEVC), which are targeted to the next-generation high definition televisions [1]. It is expected to deliver 50% better coding efficiency than H.264/AVC, while the decoding complexity will be much higher.

Recently, enhancing performance through intelligent parallel processors with multiple cores integrated on a single chip has been attempted. Since a multi-core system typically has better power efficiency than a single core system with comparable processing power, many high performance embedded mobile systems are adopting multi-core system-on-chip (SoC) platforms. Even though multi-core systems may provide potentially ample computation power, it is not straightforward to achieve a high performance because efficient parallel programming for a multi-core system is difficult. Thus, it is crucial to design software which is more suitable for parallel processing.

Parallelization of video decoding has been studied actively. Data-level parallelism is to divide a block of data into multiple sub-blocks, so they can be processed by multiple cores in parallel. Data-level parallelism should be processed without violating complicated data dependencies in a video decoding algorithm. One of the most popular data-level parallelization methods is 2D-Wave [2]. A video decoder could be parallelized to speed up the decoding with 2D-Wave, except for the entropy decoding stage. In fact, entropy decoding often becomes a performance bottleneck since processing cannot be parallelized in a straightforward manner. Among several entropy coding methods, when a high profile video service is required, Context Adaptive Binary Arithmetic Coding (CABAC) is most preferred because CABAC achieves a better compression efficiency. However, CABAC takes more time since it is complex. Recently, to resolve this concern, Multi-Threaded Syntax Element Partitioning (MT-SEP), which showed good coding efficiency and excellent performance through parallelization, was proposed [3]. It is expected that Ultra High-Definition (UHD) videos, which include 4K (3840 x 2160 pixels) and 8K (7680 x 4320 pixels) resolutions and require more than 12 Gbps of bandwidth, will be commercialized in the near future. Therefore, there is strong motivation for studies on highly parallelized decoding techniques on multi-core systems.

Even though techniques such as 2D-Wave and MT-SEP show good performance through parallelization, these methods are applied only on some specific steps of the H.264/AVC decoding. Therefore, it is not clear whether the overall decoding performance will be sufficiently improved by applying these techniques individually. Therefore, in this paper, we propose a novel method called Integrated Multi-Threaded Parallelization (IMTP), which takes parallelization of the entire decoding process into consideration in an integrated fashion. We implemented IMTP on an Intel i7 multi-core system, and conducted experiments with many benchmark programs to verify performance enhancement. Multi-threading techniques were flexibly and effectively applied to every decoding step of the H.264/AVC decoding to achieve excellent performance.

The remainder of this paper is organized as follows. In Section 2, the steps for decoding H.264/AVC are described. Also, we briefly explain two closely related works: 2D-Wave and MT-SEP. In Section 3, the proposed method called IMTP is addressed. The experimental environment and results are presented in Section 4. Conclusions and future works follow in the last section.

2. Related Work

2.1 H.264/AVC Decoder

H.264/AVC is a video compression standard proposed by ISO/IEC and ITU. Its compression ratio is high and it is adequate for video streaming through networks. Detailed specifications of the H.264/AVC standard are found in the ITU H.264 standard [4] and the ISO MPEG-4/AVC standard [5]. The H.264/AVC decoding consists of entropy decoding, inverse discrete cosine transformation, inverse quantization, intra prediction, motion compensation, and utilization of a deblocking filter. A brief explanation of these steps is given as follows:

2.1.1. Entropy Decoding (ED)

A bit stream in H.264/AVC is received as a unit of discrete packets, called a "Network Abstraction Layer (NAL) unit", and an entropy decoder generates a set of coefficients. There are two popular entropy coding methods: Context Adaptive Binary Arithmetic Coding (CABAC) and Context-Adaptive Variable Length Coding (CAVLC). When the high profile video service is required, CABAC is preferred over CAVLC because CABAC achieves better compression efficiency. A good detailed explanation of CABAC and CAVLC may be found in [6] and [7], respectively.

2.1.2. Inverse Quantization (IQ)/Inverse Transformation (IT)

Inverse Transformation (IT) and Inverse Quantization (IQ) steps process the set of coefficients, which were generated by the entropy decoding, to generate a set of residual data.

2.1.3. Intra Prediction (IP) and Motion Compensation (MC)

Intra Prediction (IP) explores spatial redundancy among neighboring blocks within a frame while Motion Compensation (MC) explores temporal redundancy between successive frames. According to the type of macroblock, either IP or MC is applied. Residual data from IT and IQ are combined with the generated block after IP and MC steps.

2.1.4. Deblocking Filter (DF)

A Deblocking Filter (DF) is used to adaptively control weights to avoid a blocking effect, which reveals the boundary between blocks in the resulting decoded video.

2.2. Parallelization of Video Decoding

2.2.1. Data-Level Parallelization for Video Decoding

Parallelization of H.264/AVC decoding depending on the size of the video data has been studied actively. Data parallelism divides a block of data into multiple sub-blocks, and lets them be processed by multiple cores in parallel. Recently, macroblock-based parallelization approaches have been attempted [8]-[12]. The macroblock level parallelization is typically carried out by allocating threads for processing macroblocks. However, data dependencies exist in H.264/AVC as shown in **Fig. 1**. For example, before the intra-prediction step for "Current macroblock (MB)" is conducted, intra-predictions for macroblocks 1, 2, 3, and 4 must be done first.

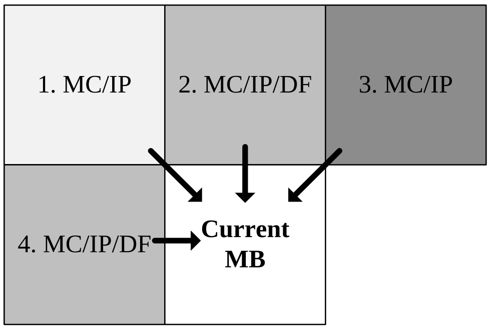

Fig. 1. Spatial data dependencies for a macroblock

One of the most popular macroblock-level parallelization methods is 2D-Wave. **Fig. 2** shows an example of 2D-Wave processing observing data dependencies. Macroblocks MB(4,0), MB(2,1) and MB(0,2) can be processed in parallel. Yet, since the processing time of each macroblock will be different, synchronization needs to occur. In 2D-Wave, a thread is allocated to process a set of macroblocks, and macroblocks are processed in the order that the arrows indicate in **Fig. 2.**

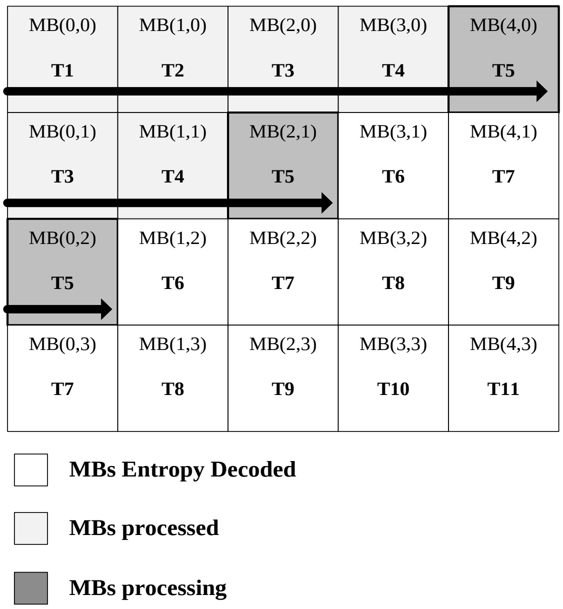

Fig. 2. An example of macroblock level parallelization

2D-Wave does not parallelize every decoding step. The ED step is processed sequentially first. After the ED step is done, MC+IT/IQ or IP+IT/IQ is processed in parallel, then DF is processed in parallel. Entropy decoding should be completed before starting the data-level parallel processing because entropy decoding may not be parallelized easily. When the video resolution is higher, CABAC is commonly used since the compression efficiency is better than CAVLC. However, entropy decoding based on CABAC takes more time. Therefore, various approaches to reduce the decoding time through parallelization have been attempted.

2.2.2. Parallel Entropy Decoding

Entropy coding in H.264/AVC employs either CAVLC or CABAC. CABAC consists of binarization, context modeling, and binary arithmetic coding. In binarization, the symbols of the block are converted into binary strings. In the context modeling stage, the probability of occurrence for each binary string is calculated based on previously coded values or other symbols in the neighborhood. Since the probability in CABAC is not fixed, but continuously updated, the compression efficiency is better than CAVLC. However, the computational complexity is higher. Due to this sequential computation structure, CABAC is known to be very hard to parallelize. To overcome such difficulty, various approaches to speed up CABAC have been proposed [13]-[15]. However, these methods were only used to parallelize specific portions of the CABAC processing, which resulted in limited performance gains and relatively low compression efficiency. To improve the compression efficiency, the CABAC algorithm using syntax element partitioning was proposed [16]. The syntax element is an element of data represented in the bit stream. CABAC decodes every bin of the syntax element using binary arithmetic decoding based on its probability model. In syntax element partitioning, syntax elements are grouped first, and CABAC is processed for each group. For videos, which have been encoded using syntax element partitioning, entropy decoding of MBINFO, PRED, CBP, SIGMAP, and COEFF groups in parallel is possible. The detailed information on the assigned syntax elements in each group is explained in [16]. This syntax element partitioning achieves a better compression efficiency than other parallel CABAC decoding methods.

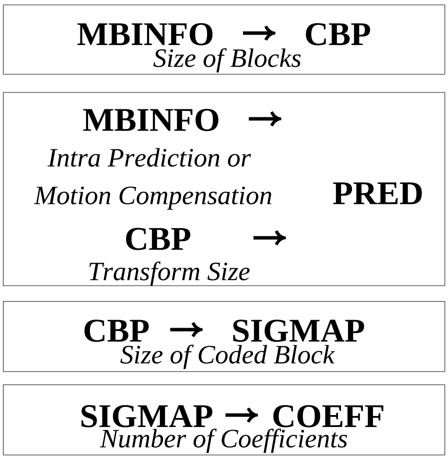

Fig. 3. Dependencies among syntax elements groups

However, dependencies hindering the full parallelization of the decoding process exist among syntax element groups, as shown in **Fig. 3**. The CBP group can be decoded only after the MBINFO group is decoded. The PRED group can be decoded only after the MBINFO group and the CBP group are decoded, and so forth. Particularly, the encoding process of SIGMAP and COEFF are tightly linked. Therefore, it is more efficient to process both groups in one task. MT-SEP was proposed as a 4-stage pipelined parallel CABAC processing method where the first task is allocated for the MBINFO group, the second is for CBP, the third is for PRED, and the fourth task is for SIGMAP and COEFF. To reduce synchronization overhead, multiple groups of syntax elements are processed in one stage. **Fig. 4** shows the 4-stage MT-SEP pipelined processing.

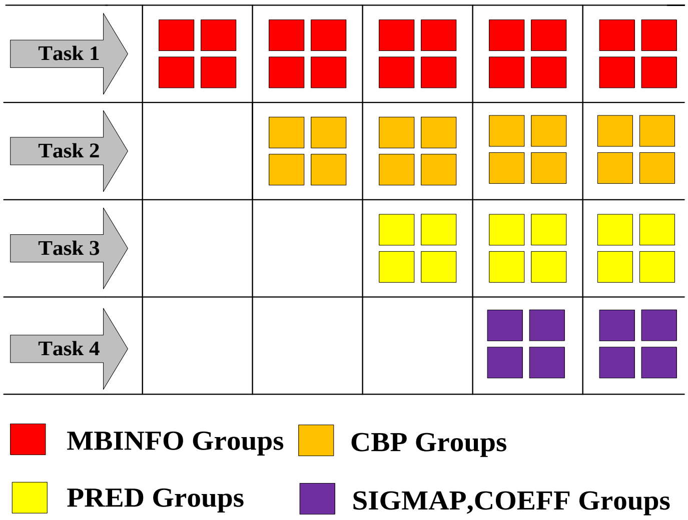

Fig. 4. Thread allocation for multiple groups of MT-SEP

Using existing methods such as 2D-Wave and MT-SEP, we can improve decoding performance for some specific decoding steps. However, a partial parallelization may not result in sufficient performance improvements. Particularly, entropy decoding was often considered separately from other decoding steps since the entropy decoding step might not be parallelized easily. Therefore, we propose a novel method called Integrated Multi-Threaded Parallelization (IMTP) which takes parallelization of every decoding step into consideration in an integrated fashion. In IMTP, multi-threading techniques are flexibly and effectively applied to optimize the overall performance for the H.264/AVC decoding.

2.2.3. Performance Evaluation for Parallel Video Decoding

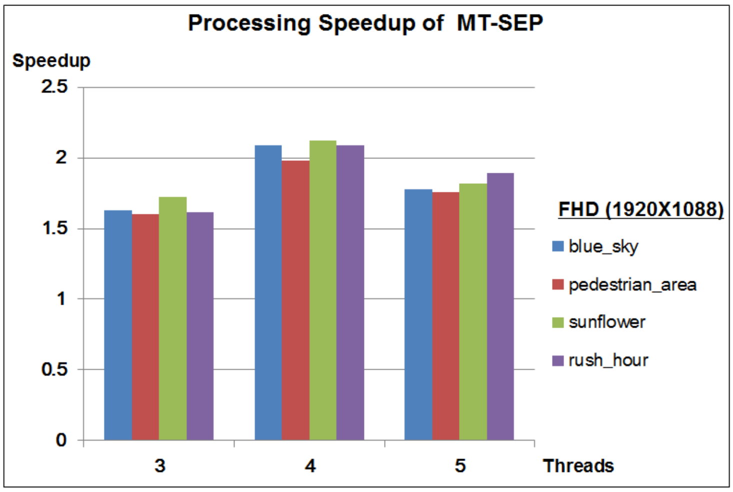

Fig. 5. Performance evaluation for MT-SEP

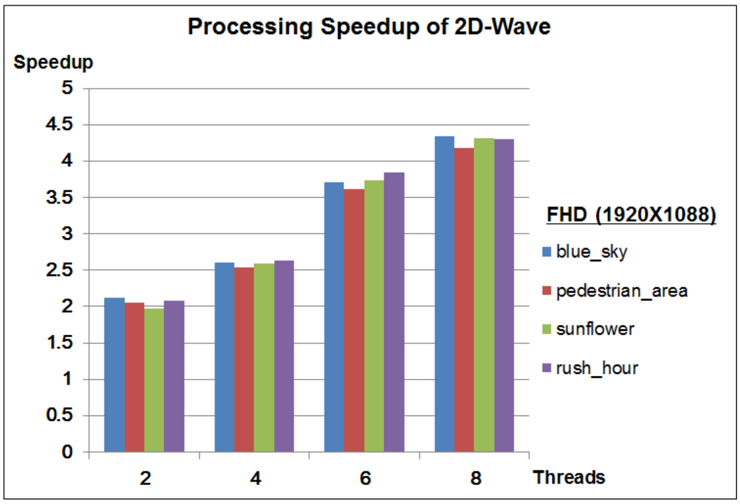

Fig. 6. Performance evaluation for 2D-Wave

Fig. 5 shows the decoding process speedup of the MT-SEP method on an Intel i7 multi-core system. **Fig. 6** shows the performance evaluation when 2D-Wave was applied to the rest of the decoding steps on the same platform. As shown in **Fig. 5**, we observe that allocating 4 threads on an Intel i7 multi-core system leads to the best performance in MT-SEP. On the other hand, in case of 2D-Wave, as we increase the number of threads from 2 to 8, the speedup of the decoding process is enhanced by up to 4.34, as shown in **Fig. 6**. These experimental results imply that there is a clear difference between MT-SEP and 2D-Wave in terms of parallelization potential and scalability. Thus, we need to take this difference into account to maximize the performance enhancement for the overall decoding step. In IMTP, such consideration is the key contribution, and our experimental results verified that it was worthwhile to consider such a difference.

3. Integrated Multi-Threaded Parallelization

3.1. Overview of Integrated Multi-Threaded Parallelization

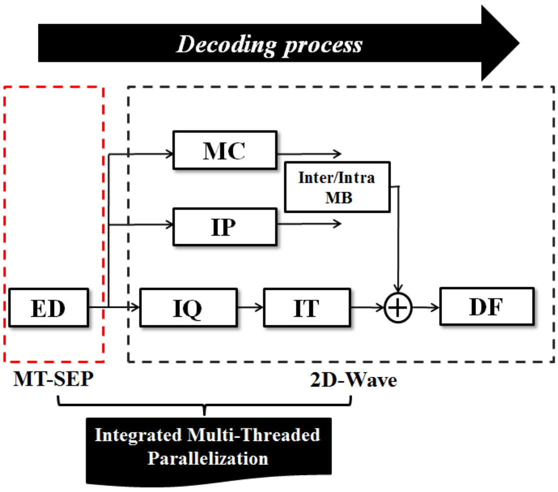

Fig. 7. Overall stucture of Integrated Multi-Threaded Parallelization

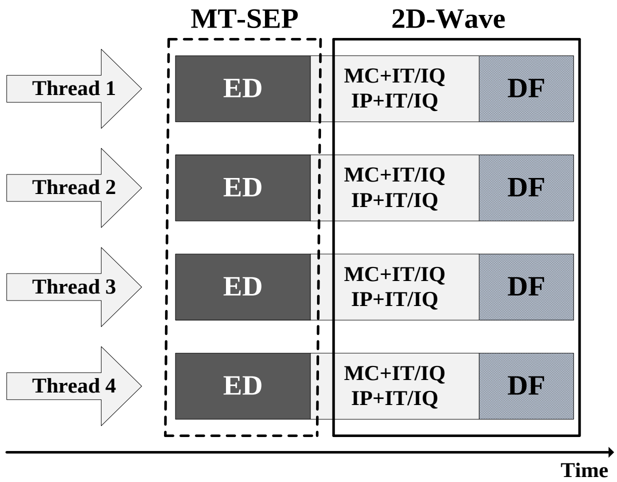

Fig. 8. Thread allocation for Integrated Multi-Threaded Parallelization

The main goal of the proposed IMTP method is maximizing the performance of the entire video decoding process of H.264/AVC through parallelization in an integrated fashion. **Fig. 7** shows the overall parallelization structure of the IMTP method. In IMTP, multi-threading techniques are flexibly and effectively applied to optimize the overall performance for the H.264/AVC decoding. Thread allocation methods in IMTP are shown in **Fig. 8.** ED is parallelized using a modified MT-SEP method and 2D-Wave is applied for MC, IP, IQ/IT and DF.

To maximize the video decoding performance, we minimize the decoding time of the overall H.264/AVC decoder. The goal function of IMTP is described as (1):

$$D_{IMTP}=minimize[\ D_{ED}+max\left[D_{MC+IT|IQ}\ ,\ D_{IP+IT|IQ}\right]+D_{DF}] \quad (1)$$

where D_{ED}, $D_{MC+IT|IQ}$, $D_{IP+IT|IQ}$ and D_{DF} represent the delay of each decoding stage in H.264/AVC, respectively, and $max[D_{MC+IT/IQ}, D_{IP+IT/IQ}]$ denotes the longer delay between the two delay components.

To parallelize the entropy decoding stage, MT-SEP was implemented with a pipelining method. Using OpenMP's parallel section directive, which can define sections for parallelization, an efficient multi-threading mechanism was devised. To implement 2D-Wave, we used OpenMP's parallel section directive to parallelize the decoding steps using multiple threads.

We evaluated the performance of our implementation with respect to various numbers of threads. From our experiments, we learned that MT-SEP showed the best performance when we allocated 4 threads. On the other hand, for 2D-Wave, which parallelized the MC, IT/IQ, IP, DF steps, we could achieve a better performance as we increased the number of threads, since 2D-Wave had ample potential for parallelization. Thus, such a difference should be exploited to achieve the best performance enhancement for the overall decoding step.

3.2. Parallelization Using Simultaneous Multi-Threading

3.2.1. Parallel Entropy Decoding with Two Independent Bins

The main reason why entropy decoding cannot be parallelized easily is that binary symbols, called bins, refer to a previously decoded bins' context model. However, two bins are independent when they do not refer to the same context model. Then, the two bins can be decoded in parallel and the computational overhead due to renormalization can be reduced [13]. Kim *et al*. analyzed the patterns of context models applied to two consecutive bins in a syntax element. They found that context models for the two consecutive bins were mostly independent. They implemented two CABAC decoders in the hardware and when two consecutive bins were independent, two consecutive bins were decoded in parallel to speed up the entropy decoding process.

3.2.2. Improving the Performance of MT-SEP using Simultaneous Multi-Threading (SMT)

In the modified MT-SEP, we make groups of syntax elements, where each group contains 4 syntax elements, and we parallelize the entropy decoding when two bins in one syntax element group are independent. We parallelize the entropy decoding of two bins by simultaneous multi-threading (SMT). SMT allows multiple threads to feed instructions to the instruction pipeline of a superscalar processor, and improves performance by supporting thread-level parallelism. An SMT processor pretends to be multiple logical processors and applications running on an SMT system simultaneously share processor resources. A higher instruction throughput and execution speedup are beneficial for a variety of workloads [17]-[19]. **Fig. 9** shows how we allocate multiple threads for MT-SEP and 2D-Wave differently.

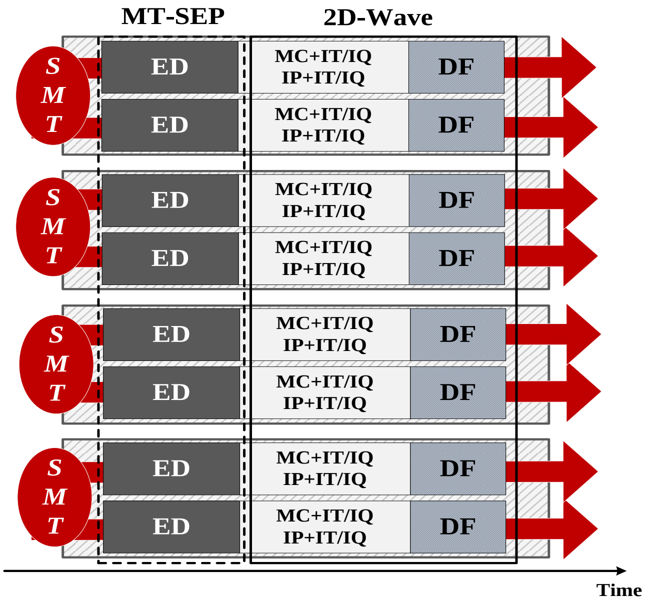

Fig. 9. Simultaneous Multi-Threading allocation for IMTP

By utilizing SMT, we further improved MT-SEP performance, which had been maximized using 4 threads. We can decode two symbols simultaneously if the context models of the decoded bin and the current bin are independent. **Fig.10** shows a flow diagram of the modified MT-SEP. When the context model is loaded in each syntax element group, we check the context models of two consecutive bins. We can tell whether two bins are independent or not by using the context model parameters. If two binary signals do not refer to the same context model, they can be independently processed when the signal is being processed in the pipeline and we switch on SMT. Through this, each of the 4 partitioned syntax element groups (MBINFO, CBP, PRED, SIGMAP and COEFF) can be parallelized using 2 threads independently. Therefore, we could extend the parallelization potential to 8 parallel decoding processes to maximize the parallelization level. In addition, we allocated 8 threads to 2D-Wave parallelization to maximize the performance by increasing the parallelism that can be exploited in each decoding stage.

We considered both MT-SEP and 2D-Wave simultaneously, and by SMT, we parallelized the entire decoding process by using up to 8 threads on 4 physical cores. In applying SMT, the characteristics of each decoding stage were reflected to maximize the parallelization potential.

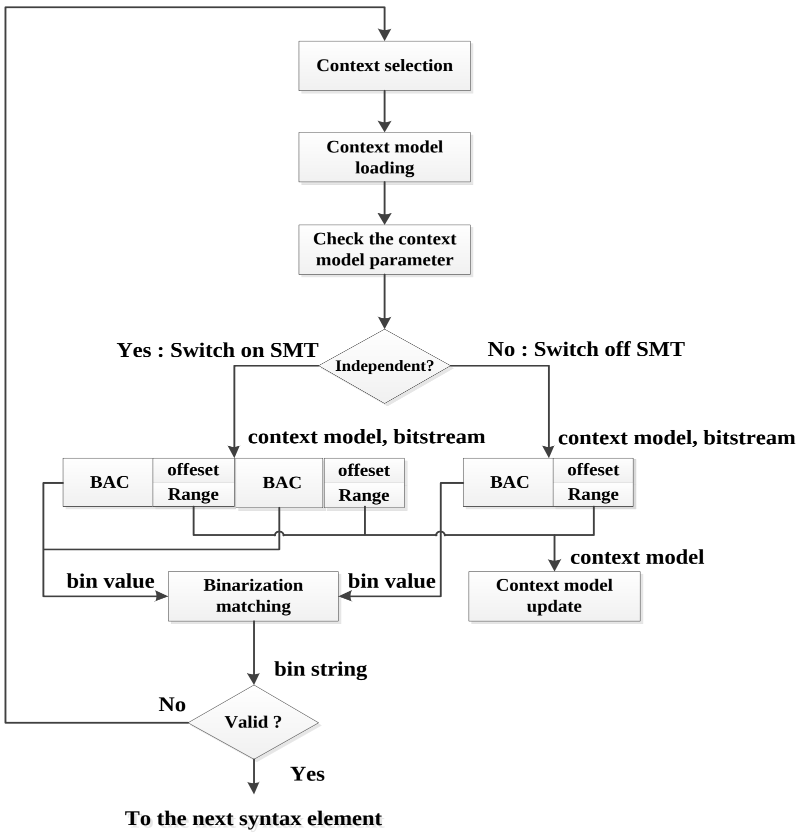

Fig. 10. Flow diagram of the modified MT-SEP

3.3. Maximization of Video Decoding Performance with Synchronization and Scheduling

MT-SEP is a task-level parallelization, and syntax element partitioning is independently processed by a different thread. 2D-Wave is a data-level parallelization, and a sequential multi-threading is applied. Therefore, it is crucial to maximize the decoding performance utilizing load balancing which takes parallelization properties of each decoding step into account. Therefore, we used OpenMP's schedule directive to apply an appropriate thread scheduling method to each decoding stage to maximize the overall decoding performance.

To find out the best thread scheduling for each decoding stage, we conducted experiments to evaluate various multi-thread scheduling methods. For MT-SEP, it turned out that static round-robin thread scheduling was best. In MT-SEP, to minimize the overhead due to frequent synchronization, the decoding is processed with equally sized groups. Hence threads are allocated evenly, and simply repeating the same process is the most efficient. Thus, static round-robin scheduling was chosen. By using static scheduling, iterations of CABAC on equally sized syntax element groups are mapped statically to execution threads in a round-robin fashion. The threads in MT-SEP will execute for the same iteration range in parallel regions. Thus, workloads of the CABAC decoding are well balanced. We used a wait directive, which blocks the next entropy decoding of syntax element group, to synchronize the MT-SEP steps. All the independent threads are synchronized at the end of entropy decoding specified by the barrier directive.

In 2D-Wave, thread allocation needs to be more dynamic, in the sense that threads that finish the allocated job early will be re-scheduled dynamically to take care of other unfinished tasks to improve the overall decoding speed. Because the processing time for each macroblock is not evenly-balanced in 2D-Wave, it is better to allocate threads dynamically to process macroblocks of which dependency constraints have been lifted. The threads produce different iteration spaces by using dynamic scheduling to decode each macroblock. Therefore, we used a wait directive to synchronize the decoding order between upper right and the current macroblocks. **Fig. 11** shows the overview of the synchronization and scheduling in IMTP.

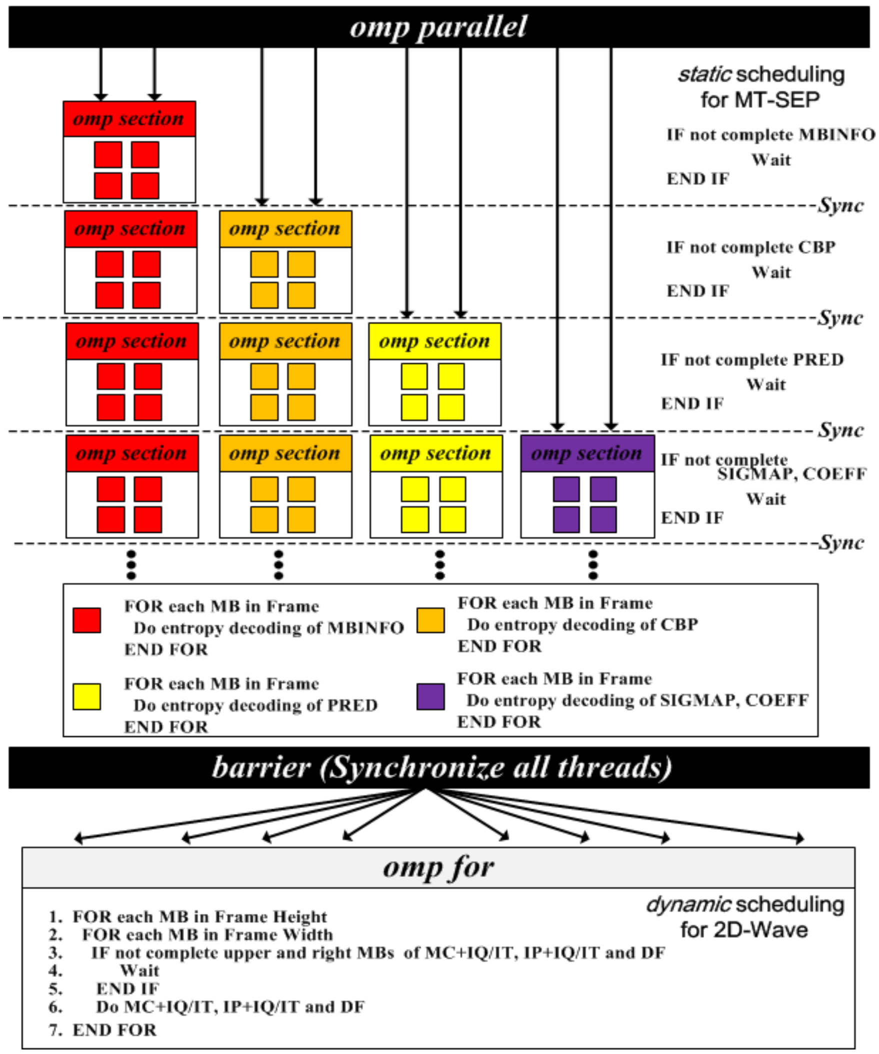

Fig. 11. Synchronization and scheduling overview of IMTP

In summary, by using SMT with properly selected thread scheduling methods for each decoding stage, we took different parallelization potentials inherent in 2D-Wave and MT-SEP, with load balancing and synchronization overhead taken into account, to maximize the overall decoding performance in the proposed IMTP method. As a result, IMTP shows excellent performance compared with other existing methods. Detailed experimental results are addressed in the section 4.

3.4. Storage Requirements of the Proposed Video Decoder

The proposed IMTP decoder and a conventional H.264/AVC Main Profile decoder are compared with respect to the amount of storage requirements. The amount of storage requirements for the Main Profile decoder (SR_{MPD}) can be expressed as (2):

$$SR_{MPD}=((n+1)\times 1.5+2)\times w\times h+\frac{w}{16}\times\frac{h}{16}\times 118+4096 \quad (2)$$

where n is the number of reference frames, w is the width of the frame, and h is the height of the frame. The number of macroblocks in a frame is $(w/16)\times(h/16)$ and one pixel requires 1.5 bytes of storage [20].

The proposed IMTP method parallelizes the decoding process utilizing SMT with 8 threads after MT-SEP and 2D-Wave have been applied. In the first stage of the ED process, 50 syntax element groups for each macroblock are processed in parallel with 8 threads. Thus, the additional storage requirements will be $(w/16)\times(h/16)\times 50\times 8$ [3]. In the remaining decoding steps, the additional storage requirements will be $(w/16)\times(h/16)\times 8$ to store the lines of macroblocks inside a frame for 2D-Wave with 8 threads. Consequently, the overall storage requirements of the proposed IMTP decoder (SR_{IMTP}) will be expressed as (3):

$$SR_{IMTP}=((n+1)\times 1.5+2)\times w\times h+\frac{w}{16}\times\frac{h}{16}\times(118+400+8)+4096 \quad (3)$$

where n is the number of reference frames, w is the width of the picture and h is the height of the frame.

SR_{MPD} is approximately 11.4 Mbytes and SR_{IMTP} is about 14.7 Mbytes in decoding a FHD resolution video frame. Therefore, the proposed IMTP video decoder requires about 30% more storage overhead than a conventional Main Profile decoder.

4. Experimental Results and Analysis

4.1. Experimental Environments

To evaluate the performance of IMTP, we parallelized KTA (Key Technical Area) 2.7 which has been developed for the next generation standard. KTA 2.7 is a software package based on JM 11.0 and contains algorithms for the next generation video processing. JCT-VC is currently developing the next generation video coding standard called HEVC. We first encoded all the sample videos using KTA 2.7. The encoding environment was based on the H.264/AVC Main Profile at 30 frames/second provided by KTA 2.7, and the quantization parameters (QPs) were set to 26, 32 and 38. The number of previous frames used for inter-motion search was 5, and a weighted prediction was used. The resolutions of sample videos were HD and FHD.

The operating system was Linux Ubuntu 11.04 with kernel version 2.6.31. The parallel implementation was executed on an Intel i7 processor. The Intel Quad-Core i7 processor has 4 physical cores but it works as 8 logical cores to allocate 8 threads by SMT.

GCC v4.5.2 was used as the compiler and OpenMP [21], [22] was used for parallelization. OpenMP is an application program interface standard for a shared memory multi-processor. Since OpenMP is a pragma-based parallelization mechanism, the application code itself doesn't have to be modified. Therefore, it's a simple way to parallelize a code originally written for sequential processing. We inserted OpenMP pragmas to parallelize the decoding process. The inserted OpenMP pragmas were pre-processed by an OpenMP-compliant compiler, and as a result, multiple threads for parallel processing were generated.

4.2. Experimental Results

Existing parallelization techniques such as 2D-Wave and MT-SEP are focused only a certain portion of the decoding process without considering the entire decoding process. Therefore, we proposed an IMTP method to optimize the entire decoding process. We compared the performance of IMTP with 2D-Wave without parallelizing the entropy decoding and MT-SEP, which only parallelized the entropy decoding on a multi-core system. All the experimental results are the minimum decoding times of benchmarks for one video frame, and speedup is used as a performance parameter, which refers to how much a proposed decoder is faster than existing decoder. The decoding times are reported in a microsecond resolution and they were measured by a clock_gettime( ) system call, which obtains the system's notion of the current time using a high resolution timer (HRT) [23].

Table 1 shows the decoding times for the entire decoding process when no parallelization technique was applied. **Table 2** shows the decoding times of MT-SEP method applied to the entropy decoding step. **Table 3** summarizes the decoding times of 2D-Wave. **Table 4** compares the decoding time for one frame of the IMTP method with existing parallelization methods. **Table 5** summarizes the decoding time for one frame after we applied the proposed scheduling techniques in IMTP. Last, **Table 6** summarizes the decoding time for one frame with different quantization parameters.

Ideally, performance may be improved by four times on a quad-core processor over a single-core processor when everything is perfectly parallelized. However, it is almost impossible to achieve such improvement in practice since execution may not be fully parallelized due to data and control dependencies and synchronization overhead. As we observe from **Table 2**, with application of MT-SEP, we achieved the process speedup of up to 2.12 in entropy decoding. However, the process speedup was only 1.19 with respect to the entire decoding performance. In **Table 3**, when 2D-Wave was applied to MC+IQ/IT, IP+IQ/IT and DF operations, excluding entropy decoding, on a multi-core system with 4 physical cores, the process speedup was up to 2.64. Moreover, when we additionally used SMT to allocate up to 8 threads, we improved the performance, and the process speedup was up to 4.34 times faster. However, since parallelization of entropy decoding was not included, the process speedup was only 1.8 times faster with respect to the entire decoding time.

Table 4 summarizes the performance improvement of the proposed IMTP with respect to the entire decoding time. The process speedup of the proposed IMTP was up to 2.7 times faster. **Table 5** shows performance results when we applied different thread scheduling methods to each decoding stage to fully utilize the characteristics of each decoding stage. The speedup of proposed IMTP method shows up to 3.35 times faster with respect to the entire decoding time.

In [24], the authors claimed that MC step took the longest average decoding time among the H.264/AVC decoding steps because interpolation of reference samples to generate a motion-compensated prediction is pretty complex. Therefore, the more inter-coded macroblocks exist, the longer time it takes to decode the video. To understand the correlation, we conducted experiments of measuring decoding times of the proposed method when we varied the number of inter-coded macroblocks for a FHD resolution video sample with a QP of 26. **Fig. 12** shows the result. The decoding time is almost linearly correlated with the number of inter-coded macroblocks.

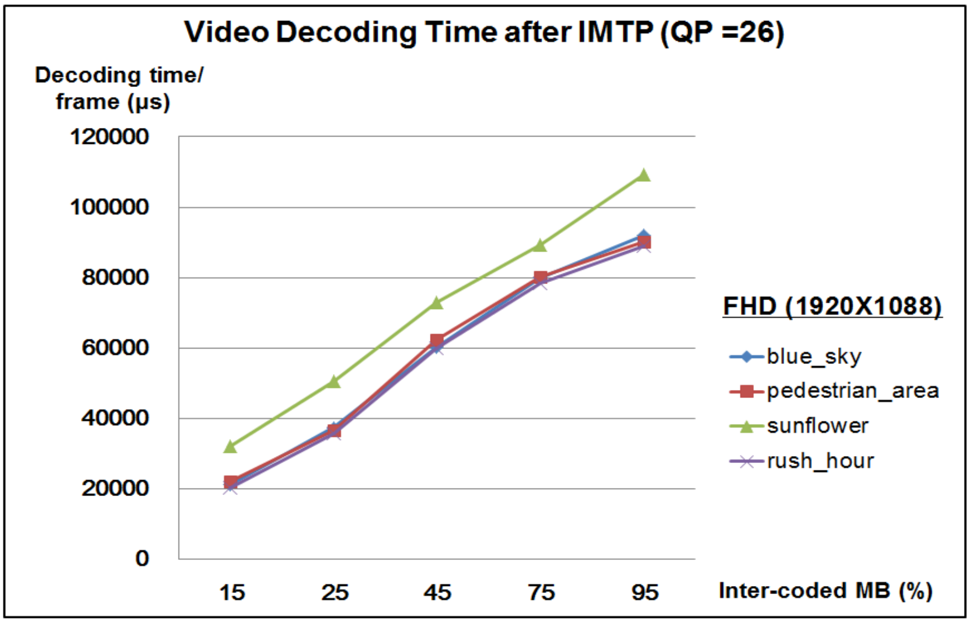

Fig. 12. Video decoding time after IMTP with different number of inter-coded macroblocks

Finally, we compared the performance of IMTP with different QPs : 26, 32 and 38. **Table 6** shows the maximum, minimum, and average decoding time per frame of the testing sequences and standard deviation values. The maximum deviation value of the decoding time is 919.45 microsecond when decoding the blue_sky test sequence with a QP of 26. When the QP is increased by 6, the quantization step size is increased by a factor of 2, and the video decoding time per frame is decreased. The result verifies that our proposed IMTP method is truly effective since it clearly shows that the total decoding time is actually improved by the application of the method.

5. Conclusion

Demand for high resolution video processing techniques is rapidly increasing as high-definition digital broadcasting services become more widely provided. Therefore, highly efficient video coding and decoding techniques should be studied actively. To parallelize the CABAC entropy decoding, the MT-SEP method was proposed. For the rest of the decoding stages, 2D-Wave is one of the most popular parallelization techniques. However, since these methods focused only a certain portion of the decoding process without considering the entire decoding process, they are not sufficiently effective in optimizing the entire decoding process.

Therefore, we proposed the Integrated Multi-Threaded Parallelization (IMTP) method to optimize the entire decoding process.

In IMTP, we used simultaneous multi-threading (SMT) to allocate up to 8 threads on an Intel i7 multi-core system with 4 physical cores. Also, we applied different thread scheduling methods to different decoding stages to effectively utilize the parallelization potentials inherent in each decoding stage. The speedup of the proposed IMTP method improves up to 3.35 times with respect to the entire decoding time. Future works to apply IMTP methods to other multi-core system such as mobile multi-core systems are ongoing. Finally, we are working on a technique to minimize power consumption while maintaining high performance in parallel video decoding.

Table 1. Processing speed of frame per video decoding before parallelization

Before Parallelization (QP=26)		ED	MC+IT/IQ IP+IT/IQ and DF	Total
		(μs)	(μs)	(μs)
mobical	HD, 1280X720	26215	62916	89131
stockholm	HD, 1280X720	34810	83544	118354
shileds	HD, 1280X720	22215	53316	75531
blue_sky	FHD,1920X1088	36906	85931	122837
pedestrian area	FHD,1920X1088	35722	78088	113810
sunflower	FHD,1920X1088	38116	97719	135835
rush_hour	FHD,1920X1088	33691	84374	118065

Table 2. Processing speed of frame per video decoding after MT-SEP

ED (QP=26)		Before MT-SEP	MT-SEP		Total	
		(μs)	(μs)	Speedup	(μs)	Speedup
mobical	HD, 1280X720	26215	17373	1.51	80289	1.11
stockholm	HD, 1280X720	34810	26657	1.31	110201	1.07
shileds	HD, 1280X720	22215	13300	1.67	66616	1.13
blue_sky	FHD,1920X1088	36906	17644	2.09	103575	1.19
pedestrian area	FHD,1920X1088	35722	18045	1.98	96133	1.18
sunflower	FHD,1920X1088	38116	17941	2.12	115660	1.17
rush_hour	FHD,1920X1088	33691	16128	2.09	100502	1.17

Table 3. Processing speed of frame per video decoding after 2D-Wave

MC+IT/IQ, IP+IT/IQ,DF (QP=26)		Before 2D-Wave	2D-Wave (4 threads)		2D-Wave with SMT (8 threads)		Total	
		(μs)	(μs)	Speedup	(μs)	Speedup	(μs)	Speedup
Mobical	HD, 1280X720	62916	25166	2.50	15099	4.17	41314	2.16
Stockholm	HD, 1280X720	83544	32582	2.56	26392	3.17	61202	1.93
Shileds	HD, 1280X720	53316	22925	2.33	13756	3.88	35971	2.10
blue_sky	FHD,1920X1088	85931	32950	2.61	19795	4.34	69856	1.76
pedestrian area	FHD,1920X1088	78088	30796	2.54	18685	4.18	66518	1.71
Sunflower	FHD,1920X1088	97719	37600	2.60	22623	4.32	75716	1.79
rush_hour	FHD,1920X1088	84374	31917	2.64	19604	4.30	65608	1.80

Table 4. Processing speed of frame per video decoding after IMTP

IMTP (QP=26)		Before Parallelization	Only MT-SEP		Only 2D-Wave		IMTP	
		(μs)	(μs)	Speedup	(μs)	Speedup	(μs)	Speedup
Mobical	HD, 1280X720	89131	80289	1.11	41314	2.16	33051	2.70
Stockholm	HD, 1280X720	118354	110201	1.07	61202	1.93	50286	2.35
Shileds	HD, 1280X720	75531	66616	1.13	35971	2.10	28776	2.62
blue_sky	FHD,1920X1088	122837	103575	1.19	69856	1.76	48622	2.53
pedestrian area	FHD,1920X1088	113810	96133	1.18	66518	1.71	48329	2.35
Sunflower	FHD,1920X1088	135835	115660	1.17	75716	1.79	52681	2.58
rush_hour	FHD,1920X1088	118065	100502	1.17	65608	1.80	51786	2.70

Table 5. Processing speed of frame per video decoding after IMTP with scheduling techniques

IMTP (QP=26)		Before Parallelization	IMTP (Before Scheduling)		IMTP (After Scheduling)	
		(μs)	(μs)	Speedup	(μs)	Speedup
Mobical	HD, 1280X720	89131	33051	2.70	28012	3.18
Stockholm	HD, 1280X720	118354	50286	2.35	38217	3.10
Shileds	HD, 1280X720	75531	28776	2.62	22869	3.30
blue_sky	FHD,1920X1088	122837	48622	2.53	37439	3.28
pedestrian area	FHD,1920X1088	113810	48329	2.35	36730	3.10
sunflower	FHD,1920X1088	135835	52681	2.58	40564	3.35
rush_hour	FHD,1920X1088	118065	51786	2.70	35732	3.18

Table 6. Processing speed of frame per video decoding with different quantization parameter

IMTP (Decoding time /frame)		Maximum decoding time	Minimum decoding time	Average decoding time	Standard deviation
		(μs)	(μs)	(μs)	(μs)
QP=26					
Mobical	HD, 1280X720	30013	28012	28930.83	883.55
Stockholm	HD, 1280X720	40306	38217	39133.64	877.27
Shileds	HD, 1280X720	24914	22869	23790.75	879.34
blue_sky	FHD,1920X1088	39573	37439	38392.12	919.45
pedestrian area	FHD,1920X1088	38912	36730	37690.23	903.21
Sunflower	FHD,1920X1088	42716	40564	41514.72	887.21
rush_hour	FHD,1920X1088	37932	35732	36694.28	897.83
QP=32					
Mobical	HD, 1280X720	28212	26331	27194.98	830.54
Stockholm	HD, 1280X720	37888	35924	36785.62	824.63
Shileds	HD, 1280X720	23419	21497	22363.31	826.58
blue_sky	FHD,1920X1088	37199	35193	36088.59	864.28
pedestrian area	FHD,1920X1088	36577	34526	35428.82	849.02
Sunflower	FHD,1920X1088	40153	38130	39023.84	833.98
rush_hour	FHD,1920X1088	35656	33588	34492.62	843.96
QP=38					
Mobical	HD, 1280X720	26832	25043	25864.16	789.89
Stockholm	HD, 1280X720	36034	34166	34985.47	784.28
Shileds	HD, 1280X720	22273	20445	21268.93	786.13
blue_sky	FHD,1920X1088	35378	33470	34322.56	821.99
pedestrian area	FHD,1920X1088	34787	32837	33695.07	807.47
sunflower	FHD,1920X1088	38188	36264	37114.16	793.17
rush_hour	FHD,1920X1088	33911	31944	32804.69	802.66

References

[1] G. J. Sullivan, J. R. Ohm, W. J. Han and T. Wiegand, "Overview of the high efficiency video coding (HEVC) standard," *IEEE Transactions on Circuits and Systems for Video Technology*, Vol.22, Dec. 2012. Article (CrossRef Link).

[2] E. Tol, E. Jaspers, and R.Gelderblom, "Mapping of H.264 decoding on a multiprocessor architecture," *Image and Video Communications and Processing*, pp.707-718, May 2003. Article (CrossRef Link).

[3] W. Kim, K. Cho, K. Chung, "Multi-Threaded Syntax Element Partitioning for Parallel Entropy Decoding," *IEEE Transactions on Consumer Electronics*, vol. 57, pp.897-905, May 2011. Article (CrossRef Link).

[4] ITU-T Recommendation H.264, SERIES H: Infrastructure of audiovisual services- Coding of moving video, May 2003. Article (CrossRef Link).

[5] ISO, Information Technology-Coding of Audio-Visual Objects, Part10-Advanced Video Coding, ISO/IEC 14496-10. Article (CrossRef Link).

[6] T. Wiegand, G. J. Sullivan, G. Bjontegaard, and A. Luthra, "Overview of the H.264/AVC Video Coding Standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 7, pp. 560-576, July 2003. Article (CrossRef Link).

[7] M. Horowitz, A. Joch, F. Kossentini, and A. Hallapuro, "H.264/AVC Baseline Profile Decoder Complexity Analysis," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 7, pp. 704-716 July 2003. Article (CrossRef Link).

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

[9] J. Chong, N. R. Satish, B. Catanzaro, K. Ravindran, and K.Keutzer, "Efficient parallelization of h.264 decoding with macro block level scheduling," in *Proc. of 2007 IEEE International Conference on Multimedia and Expo*, July 2007. Article (CrossRef Link).

[10] K.Nishihara, A. Hatabu, T. Moriyoshi, "Parallelization of H.264 video decoder for embedded multicore processor," *in Proc. of ICME'2008*. pp. 329-332. Article (CrossRef Link).

[11] A. Azevedo, C. Meenderinck, B. Juurlink, A. Terechko, J. Hoogerbrugge, M. Alvarez, and A. Rammirez, "Parallel H.264 Decoding on an Embedded Multicore Processor," *in Proc. of the 4th International Conference on High Performance and Embedded Architectures and Compilers -HIPEAC*, Jan. 2009. Article (CrossRef Link).

[12] Won-Jin Kim, Keol Cho, Ki-Seok Chung, "Stage-based frame-partitioned parallelization of H.264/AVC decoding," *IEEE Trans. on Consumer Electronics*, vol. 56, pp. 1088-1096, May 2010. Article (CrossRef Link).

[13] Chung-Hyo Kim, In-Cheol Park, "Parallel Decoding of Context-Based Adaptive Binary Arithmetic Codes Based on Most Probably Symbol Prediction," *IEICE Trans. on Information and Systems*, vol. E90-D, no. 2, pp. 609-612, February 2007. Article (CrossRef Link).

[14] Jian-Hung Lin, Keshab K. Parhi, "Parallelization of Context-Based Adaptive Binary Arithmetic Coders," *IEEE Trans. on Signal Processing*, vol. 54, no. 10, pp. 3702-3711, October 2006. Article (CrossRef Link).

[15] Andrew Segall, Jie Zhao, "Entropy slices for parallel entropy decoding," ITU-T SGI 6/Q.6 Doc. COM16-C405. Geneva, Switzerland, April, 2008. Article (CrossRef Link).

[16] Vivienne Sze, Anantha P. Chandrakasan, "A High Throughput CABAC Algorithm Using Syntax Element Partitioning," in *Proc. of IEEE International Conference on Image Processing (ICIP)*, pp. 773-776, November, 2009. Article (CrossRef Link).

[17] Jingren Zhou, John Cieslewicz, Kenneth A. Ross, Mihir Shah, "Improving database performance on simultaneous multithreading processors," in *Proc. of VLDB '05, 31st international conference on Very large data bases*, pp.49-60. Article (CrossRef Link).

[18] D. M. Tullsen, S. J. Eggers, and H. M. Levy, "Simultaneous multithreading: Maximizing on-chip parallelism," *in Proc. of ISCA Conference*, 1995. Article (CrossRef Link).

[19] Xinmin Tian, Yen-Kuang Chen, Girkar, M., Ge, S., Lienhart, R., Shah, S., "Exploring the Use of Hyper-Threading Technology for Multimedia Applications with Intel OpenMP Compiler," *Parallel and Distributed Processing Symposium*, pp.6, April 2003. Article (CrossRef Link).
[20] M. Horowitz, A. Joch, F. Kossentini, and A. Hallapuro, "H. 264/AVC baseline profile decoder complexity analysis," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol.13,no.7, pp.704-716, July, 2003. Article (CrossRef Link).
[21] C. Liao, Z. Liu, L. Huang and B. Chapman, "Evaluating OpenMP on Chip MultiThreading Platforms," *in Proc. of First international workshop on OpenMP*, Eugene, Oregon USA, June 2005. April, 2004. Article (CrossRef Link).
[22] M. Curtis-Maury, X. Ding, C. Antonopoulos, D. S. Nikolopoulos, "An evaluation of OpenMP on current and emerging multithreaded/multicore processors," *in Proc. of the First International Workshop on OpenMP (IWOMP)*, Eugene, Oregon USA, June, 2005. Article (CrossRef Link).
[23] Eugene Antsilevich, "Capturing Timestamp Precision for Digital Forensics," *JMU-INFOSEC-TR-2009-002*, James Madison University Infosec Techreport Department of Computer Science, Jan. 2009. Article (CrossRef Link).
[24] Xiaosong Zhou, Eric Q. Li and Yen-Kuang Chen, "Implementation of H.264 Decoder on General-Purpose Processors with Media Instructions," in *Proc. of SPIE Conf. on Image and Video Communication and Processing*, Jan. 2003. Article (CrossRef Link).

Jung-Hyun Hong received his B.S. degree in Media Communication Engineering from Hanyang University, Seoul, Korea in 2011. Since 2011, he has been taking a Unified M.S. and Ph.D. course at Hanyang University, Seoul, Korea. His research interests include low power embedded system design, parallelization, image processing and embedded multi-core architecture.

Won-Jin Kim received the B.S. degree in Mechanical Engineering with a minor in Electronic Engineering from Hanyang University, Ansan, Korea in 2002. He was an Engineer at SENA Corp. in Seoul from 2002 to 2004, and an Engineer at MGAME Corp. in Seoul from 2004 to 2006. He received the M.S. and Ph.D. degrees in Electronic Engineering from Hanyang University, Seoul, Korea in 2008 and 2012. He is currently working as senior engineer in Design Solution Laboratory of DMC R&D Center, SAMSUNG Electronics. His research interests include low power embedded system design, image processing, parallelization and multi-core architecture.

Ki-Seok Chung received his B.E. degree in Computer Engineering from Seoul National University, Seoul, Korea in 1989 and Ph.D. degree in Computer Science from University of Illinois at Urbana-Champaign in 1998. He was a Senior R&D Engineer at Synopsys, Inc. in Mountain View, CA from 1998 to 2000, and was a Staff Engineer at Intel Corp. in Santa Clara, CA from 2000 to 2001. He also worked as an Assistant Professor at Hongik University, Seoul, Korea from 2001 to 2004. He is now an Associate Professor at Hanyang University, Seoul, Korea. His research interests include low power embedded system design, multi-core architecture, image processing, reconfigurable processor and DSP design, SoC-platform based verification, and system software for MPSoC.