A Novel Macro-Block Group Scheme of AVS Coding for Many-Core Processor

Zhenyu Wang[1], Luhong Liang[2], Xianguo Zhang[1], Jun Sun[1], Debin Zhao[3], Wen Gao[1]

[1]Peking University, Beijing, 100871, P. R. China
[2]Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, P.R. China
[3]Harbin Institute of Technology, Harbin 150001, P.R. China

{zywang, lhliang, xgzhang, jsun, dbzhao, wgao}@jdl.ac.cn

Abstract. The slice-level parallelism is popular in parallel video coding. However, the quality loses greatly because the dependency between macro-blocks is broken, especially on many-core platforms. To address this problem, a novel Macro-Block Group (MBG) decomposition scheme is presented for parallel AVS coding. In the proposed scheme, video frames are equally divided into rectangular MBG regions, each consists of more rows and less columns than the slice-level scheme. Since MBG is not supported by AVS, a vertical partitioning scheme is introduced, and the mode confining and MVD adjusting techniques are utilized to keep consistency with the standard. In practice, our parallel encoder is developed on the TILE64 platform, where P/B frames use the MBG-level parallelism and I frames use the macro-block-level parallelism. Experiments show that the proposed scheme can achieve a reduction of 52% (IPPP) and 41% (IBBP) in quality loss while keeping the same speed-up compared with the slice-level parallelism.

Keywords: parallel video encoding, macro-block group, many-core processor

1 Introduction

The new generation of video coding standards such as H.264/AVC [1] and AVS (Audio Video Standard) [2] are becoming increasingly popular. These standards can provide much higher coding performance than earlier ones but cost greatly increased complexities of new tools such as quarter-pixel motion estimation, variable block sizes, and so on [1]. In order to realize the real-time HD (High Definition) video encoder, many technologies have been developed. Among them, parallel algorithms are the most important and efficient solutions for hyper-threading processor, multi-core processor and multi-processor platforms [3]. Recently, with the advance of hardware technology, many-core processors that consist of dozens or hundreds of

CPU cores are emerging. This brings new challenges in designing efficient parallel video coding algorithms.

To date most of the parallel video coding algorithms exploit data parallelism. Typically, there are four kinds of data parallel schemes: GOP-level，frame-level, slice-level and Macro-Block-level (MB-level) parallelisms. The GOP-level parallelism assigns GOPs to different processor cores, which is the most straightforward approach and can be implemented in a very simple manner [4], [5]. However, this coarse-grained parallelism introduces a very long latency. The frame-level parallelism [3] encodes B frames and the next I or P frames at the same time, so that it has much lower latency. However, since only consecutive B frames and the next I or P frames can be encoded in parallel, it is not suitable for the many-core processor. As a popular parallel scheme, the slice-level parallelism [6], [7] considers the independence of slices and encodes all slices in one frame simultaneously. It does not introduce additional latency, but may cause serious performance drop with the increase of slice number, because the dependences between MBs along the slice borders are broken [3]. The macro-block-level parallelism [8] assigns each MB to different processor cores. Since there are dependencies among the MBs, sophisticated scheduling of the encoding tasks is required. It also needs frequent communications among the processor cores, which becomes the bottleneck of coding speed, especially on the many-core processor. Besides the parallel schemes above, there are some hybrid algorithms that combine parallelisms of different levels, e.g. the hierarchical parallelization in [9]. Most recently, a Macro-Block Region Partition (MBRP) is also proposed [10], where video frames are partitioned into several adjoining columns of MBs. Each column can be encoded with the wave-front technique by one processor of a multi-processor system. Although many parallel algorithms for video coding have been introduced, to our best knowledge, there is seldom technology that can be easily extended to many-core platform.

Because of some superior characteristics such as less communication among processor cores and low coding latency, the slice-level parallelism is a promising method for the many-core processor. However, the quality loss may become much serious, because the video frame should be divided into too many slices for the large amount of processor cores. To address this problem, we propose a novel Macro-Block Group (MBG) parallelism for the HD AVS encoder. The MBG is a rectangle region of a frame that consists of more rows and less columns than the slice-level scheme. Each MBG can be encoded in a processor core. Since the number of MBs along the borders in a MBG can be designed much less than that of a slice with the same MB number, the quality loss caused by the broken dependency on the borders is much less. However, the MBG is not a syntax element in AVS [2]. In order to comply with the standard, we induce MBG by vertically partitioning the slices and develop several new techniques such as prediction mode confining and MVD adjusting in parallel coding. Based on the new MBG decomposition, we propose a hybrid parallel scheme with MBG-level parallelism in P/B frames and MB-level parallelism in I frames, and realize an HD AVS encoder on the TILE64 platform [11]. Experimental results show that our method has 52% less quality loss in IPPP GOP and 41% less in IBBP GOP compared with the slice-level parallelism for the same number of processor cores.

The rest of this paper is organized as follows. Section 2 gives an analysis on quality loss in slice-level parallelism of AVS encoder. Section 3 presents the MBG based parallel scheme. Section 4 introduces the implementation of HD AVS encoder on the TILE64 platform. Section 5 shows the experimental results and Section 6 gives the conclusion.

2 Analysis on quality loss in the slice-level parallelism

As mentioned above, the quality loss is the major disadvantage of slice-level parallelism on the many-core processor. In [3], an analysis on the quality loss of the H.264/AVC coding due to slice partitioning on PC platform is presented. In this section, we analyze the AVS video coding algorithm in order to get more insight into its quality loss in the slice-level parallelism.

We employ 11 HD (1280x720p) sequences (*BigShips*, *City, Crew, Cyclists, Harbour, Night, Optis, Raven, Sheriff, ShuttleStart* and *Spincalendar*) provided by the AVS Working Group [12], each of which has 460 to 900 frames. The AVS1-P2 reference code (*RM52k*) with some reduction of coding tools such as RDO and rate control is used (see details in Section 5).

Our experimental results show that the partition of 45 slices (i.e. only one MB row in each slice) brings 0.5db quality loss on average compared with one-slice-one-frame (i.e. 45 MB rows in the slice). The average quality losses are 0.51db, 0.69db and 0.19db respectively for I/P/B frames. Fig. 1 shows some details of quality loss and bit rate increase against different slice partitions on the *Cyclists* sequence. We can see that, with the decrease of the MB rows number in one slice (i.e. increase of slice number), the quality losses of all kinds of frames are increasing. These results can be explained as below. In the AVS standard, the top, top-left and top-right neighbors of a block, which is on the top border of a slice, are unavailable. For the inter block, it may not get the best predicted motion vector (PMV) only by using the motion vector (MV) of the left neighbor. For the intra block, some potential sub-modes are confined because these neighbors are unavailable. All of these cause the decrease of coding performance. More specifically, the quality loss essentially comes from the MBs on the slice borders. As shown in Fig.2 (a), a slice-level parallel HD (1280x720p) video encoder is realized on a 45-core platform, where each frame should be partitioned into 45 slices, i.e. one MB row in each slice. As a result, 3520 MBs, as large as 97.8% of the total 3600 MBs, are on the slice borders which causes apparent quality drop in the video coding.

Fig.1 also shows that the relation between the quality loss and the MB rows number is nonlinear. When the number of MB rows in a slice increases to 4, most of the dependencies among MBs can be preserved. As a result, about 50% quality loss is avoided. The saving of quality loss is 65% while the number of MB rows in a slice increases to 8. This shows that slice-level parallelism is an acceptable HD (720p) video coding scheme for 6~12 processor cores. However, on the many-core processor that consists of dozens of processor cores, the video frame should be divided into many small slices, even one MB row in one slice. Apparently, the quality loss becomes a very serious problem in this case. Moreover, the number of the processor cores used is also constrained by the maximum number of slices (i.e. 45 slices in AVS for 720p video). Therefore, a new parallelism technique is desired for the many-core processor.

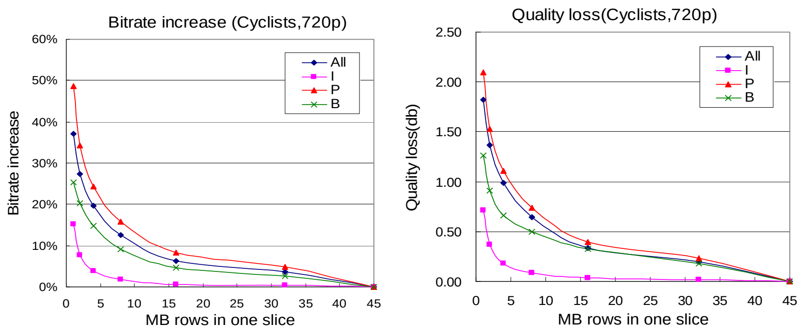


Fig. 1. Quality loss and bitrate increase in different slice partition

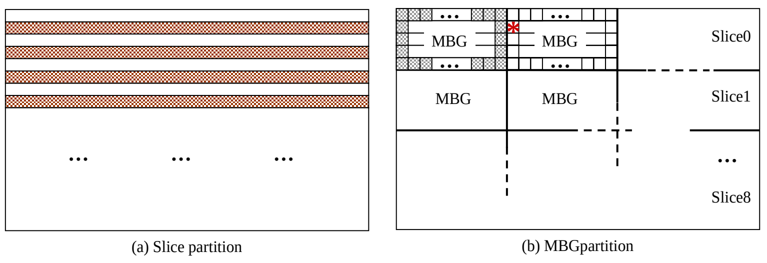


Fig. 2. Slice partition and MBG partition

3 The Macro-Block Group Level Parallelism

3.1 The micro-block group decomposition

As presented in Section 2, the main reason for the quality loss in the slice-level parallelism is the limitation of sub-mode in intra MBs and PMVs in inter MBs along slice borders. In order to solve this problem, we propose a novel MBG decomposition scheme to reduce the number of MBs affected by data partition. As shown in Fig. 2(b), we define the MBG as a rectangle region in a frame that consists of a certain number of MBs, and the MBG is encoded independently on a processor core with minimal communication and overhead. Obviously, the number of MBs on the borders of MBGs is much less than that on the border of slices with the same number of MBs. For example, as for the same HD (1280x720p) video frame, if we partition it into 5$\times$9=45 MBGs, the MBs on the border reduces to 936, i.e. the percentage greatly reduces to 26% compared with 97.8% in the slice partition.

Considering the implementation on the many-core processor, we propose several principals in MBG design as below:

(1) Making the MBG close to a square to minimize the total number of border blocks.

(2) Making all the MBGs consist the same amount of MBs for the load balance in parallel coding.

(3) Considering the architecture of the many-core processor, including number of cores, sizes of L1 and L2 caches, communication mechanism among cores, constrains in data alignment, etc.

Besides the above principals of the MBG design, we should consider the constraints of the AVS standard. The MBG is not a syntax element in AVS. Actually the AVS1-P2 [2] only supports slice with a certain number of complete MB rows[1]. Therefore, we propose to partition the MBG based on the AVS slice. Without loss of generality, supposing 45 MBGs should be partitioned, we firstly set the slice number to 9, i.e. partition the frame to 9 slices with equal MB row number, and then vertically divide each slice into 5 MBGs with equal number of MB columns. Apparently, the borders of the AVS slices can work as the horizontal borders of the MBGs. However, the vertical MBG borders bring inconsistency between encoding and decoding. For example, as shown in Fig.2 (b), the left neighbor of the block marked with the star ("*") is unavailable in the encoding since each MBG is encoded independently, while it becomes available as a block in the same slice in the decoding. We will give detailed analysis on this problem and propose a solution in Section 3.2 and 3.3 respectively.

3.2 Analysis on the inter/intra mode in MBG blocks

In this section, we analyze the inconsistency between encoding and decoding of the blocks along the vertical borders of MBGs.

As for the inter mode of the blocks on vertical MBG borders, the inconsistency comes from the PMV. Without loss of generality, we assume that all the blocks have the same size of 8×8 as shown in Fig.3, where each little square stands for a block. The PMV of the block E along the vertical MBG border relies on the availabilities of block A~D. While encoding, the block A and D are unavailable since this MBG is encoded independently. In this case, the MV of block E is predicted according to block B and C, and then the difference (MVD) between MV and PMV is written into the bit stream. However, when decoding, block A and D are available according to the slice partition since the information of MBG is blind for the decoder. Obviously, the MV calculated according to the availability of the block A~D and the MVD in the bit stream would be inconsistent. As for the blocks like J on the left side of the MBG border, similar failure may happen due to different availability status of block H in encoding and decoding. We will describe that these inconsistency can be solved by confining some prediction sub-modes and adjusting the MVD in the next section.

[1] The AVS1-P2 Jizhun, Zengjiang and Jiaqiang profile only supports slice with complete MB lines, while the Shenzhan profile support rectangular slices, similar to the proposed MBG.

As for the intra mode, the sub-mode prediction causes the similar problem as the PMV. However, it is impossible to solve the inconsistency in some blocks along the MBG border by prediction mode confining or residual adjustments. For example, as for the luminance block B in Fig. 3, in encoding, only the *Intra_8x8_DC* mode can be chosen since both the block D and the block K~M are not available. Moreover, all the predicted pixels must be set to 128 [2]. While in decoding, the block D is available, so that the *Intra_8x8_DC* mode computes the predicted pixel according to pixels in block D, which would not be just 128 in general. As a result, the decoding result would be incorrect. On the other hand, it is impossible to adjust the residual to keep the correctness. In encoding, the residual is transformed and quantized before being written into the bit stream. This is likely to change the adjusted residual, so that the decoder can not get the exact residual values after inverse quantization and inverse transformation. Therefore, we should confine the intra mode in block B to avoid the inconsistency issue.

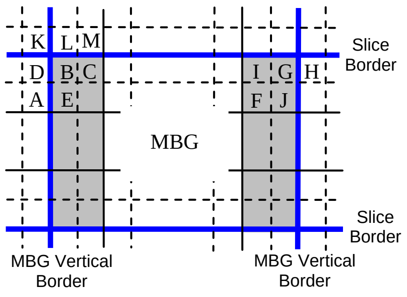


Fig. 3. Inter and intra mode in MBG partition

In summary, we can see that it is impossible to realize the MBG partition in AVS I frames where there is only intra mode blocks. Fortunately, it can be realized in P/B frames by confining intra mode and adjusting the data of some MBs. We will describe the scheme in the next section.

3.3 The MBG-level parallelism for P/B frames

In this section, we introduce the MBG-level parallel encoding scheme for P/B frames. As shown in Fig. 4, the encoding is accomplished in 3 steps. Firstly, all the MBGs are encoded simultaneously on different processor cores, where a new mode confining technique is used for the MBs along the vertical MBG borders. Then, the processor cores that encode the adjacent MBGs exchange the MVs of the blocks on the vertical border. Finally, these MVs are used to adjust the data of MBs along the vertical MBG borders in order to keep encoding and decoding consistency.

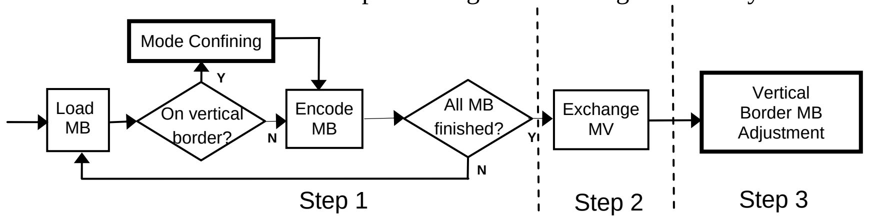


Fig. 4. Encode a MBG

Here, we give the details of Step 1 (mode confining) and Step 3 (vertical border MB adjustment)

- **Prediction mode confining**

  As presented in Section 3.2, it is impossible to solve the inconsistency issue of the intra mode block along the vertical MBG borders, so the intra mode must be confined in the corresponding MBs (See MBs marked by grey background in Fig.3). As for the inter mode, the skip mode in these blocks should also be confined because the skip mode will not write the MVD into the bit stream, and we can not keep the consistency by adjusting MVD. For the same reason, the direct mode and symmetric mode in B frames should also be confined on these blocks. Although the above scheme possibly discards the best mode on these blocks or MBs, it brings little quality loss due to the low percentage of these blocks or MBs. For example, if a HD(1280x720p) video frame is divided into $5\times9=45$ MBGs (5 MBGs in each slice), there are only $45\times2\times4-4=356$ MBs, i.e. only 10% of the MBs are affected.

- **MVD adjustment of the vertical border blocks**

  As for the forward and backward modes, all the block size, including 16x16, 16x8, 8x16 and 8x8 can be used in the blocks on the vertical MBG borders. The MVD of these inter modes should be adjusted to solve the inconsistency issue.

As shown in Fig.5, the block E on the right border of MBG1 is encoded without block C since it is unavailable. More specifically, it calculate the PMV (vector $\overrightarrow{oa}$) using the MVs of the block A, B and D, and search the actual MV (vector $\overrightarrow{ob}$) in a region centered by *a*. Then, it calculates the residual (vector $\overrightarrow{ab}$) of the PMV and the actual MV and writes it into the bit stream as the MVD. In decoding, the PMV is calculated according to the block A, B and C, as represented by vector $\overrightarrow{oc}$, which would be different to $\overrightarrow{oa}$.

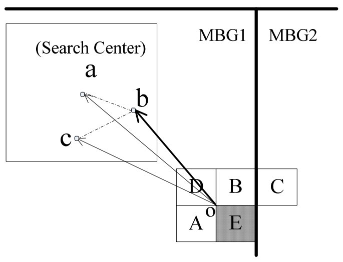


Fig. 5. MVD adjustment in MBs along the vertical MBG borders

In order to compensate this difference, we propose to adjust the MVD data in block E. As mentioned above, the MV of block C is sent to the processor core for MBG1 in the step 2 (see Figure 4). Therefore, the vector $\overrightarrow{oc}$ can be recalculated according to block A~D in the step 3, which is the same as that in the decoder. Then, vector $\overrightarrow{cb}$ can be calculated as the residual of PMV $\overrightarrow{oc}$ and actual MV $\overrightarrow{ob}$ and is written into the bit stream as the MVD instead of vector $\overrightarrow{ab}$. Obviously, in this case the decoder can resume the correct MV $\overrightarrow{ob}$. For other blocks the left and right MBG borders, the solution is the same.

Although recalculating MVD introduce some additional computations, it affects the coding speed very little, because the border blocks have rather small percentage, e.g. less than 10% in the 45-MBG-partition mentioned above.

4 Implementation

We develop a parallel AVS encoder on the TILE64 processor [11]. The coding algorithm is based on the AVS1-P2 reference code *RM52k* [12]. The MB-level parallelism in I frames and the MBG parallelism in P/B frames are used. The encoder software is developed and tuned using *Tilera MDE* (multicore development environment) and *Tilera TILExpress-20G™* Card [11].

4.1 The TILE64 platform

TILEPro64 Processor is the latest generation processor features 64 identical processor cores (namely *tiles*) interconnected with Tilera's *iMesh™* on-chip 8x8 network [11]. As shown in Fig.6, each tile is a complete full-featured 700M Hz processor and integrated 8KB L1 cache and 64KB L2 cache. Each tile can access its private memory through the on-chip network directly, and shared memory through another tile's L2 cache, if the shared memory is not allocated by the process on current tile.

The *Tilera MDE* provides three communication mechanisms between the tiles, including channel, message and shared memory. The shared memory includes system management shared memory and user management shared memory. With user management shared memory, all the processes on different tile can access the same shared memory through the L2 cache of the tile which allocates the shared memory. The processes can copy the data in the shared memory to their own L2 cache without considering coherence. We use this shared memory mode by calling APIs provided by *Tilera MDE* to share and exchange data in parallel coding.

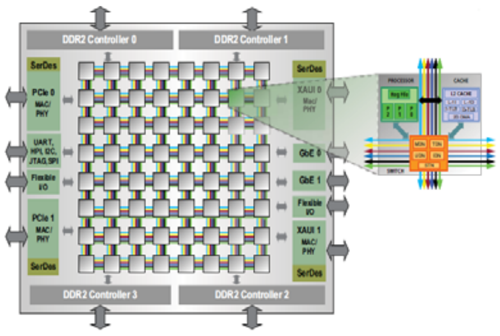
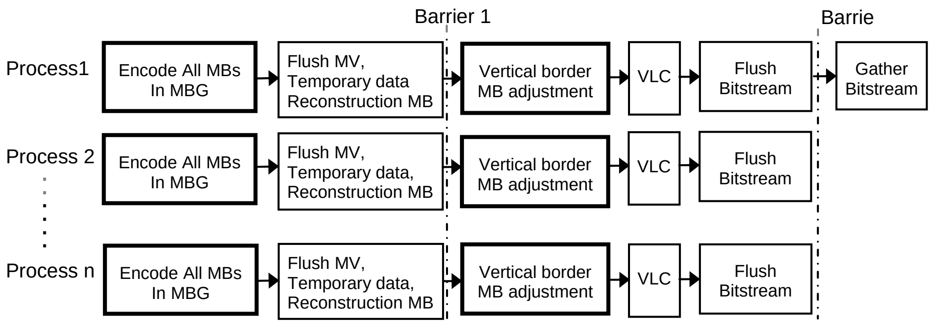


Fig. 6. Architecture of TILEPro 64 processor[11]

4.2 MBG-level parallel scheme for P/B frames

In our parallel encoder, we use the MBG-level parallelism in P/B frames, as shown in Fig. 7. At first, the n MBGs in a frame are assigned to n processes executed on n processor cores. Every process encodes all the MBs in its MBGs independently using the algorithm described in step 1 in section 3.3. The operations including ME, DCT, quantization, inverse quantization, IDCT and MC are done. The reconstructed MBs and temporary data such as the residuals after quantization, the prediction modes and the MVs are stored in the shared memory, but they may be unavailable for other processes at this time. Then, every process flush the data stored in the shared memory and calls a synchronization barrier function to wait until all the flush operations are done. After that, all the data previously stored in the shared memory are visible for all the processes. Next, the MVD adjustment (as described in section 3.3), deblocking, interpolation and VLC are conducted. At last, another synchronization barrier is used to make the bit stream data visible for processes No.1, which gathers the data from every MB row and creates the final bit stream.

Benefiting from the shared memory mechanism, only two synchronization barriers are required in the MBG-level parallelism, which are much less than those in the MB-level parallelism. Compared with the slice-level parallelism, the proposed scheme uses one more synchronization barrier, but our experiments show it affects the coding speed very little.


Barrier 1
Barrie
Process1
Encode All MBs In MBG
Flush MV, Temporary data Reconstruction MB
Vertical border MB adjustment
VLC
Flush Bitstream
Gather Bitstream
Process 2
Encode All MBs In MBG
Flush MV, Temporary data, Reconstruction MB
Vertical border MB adjustment
VLC
Flush Bitstream
Process n
Encode All MBs In MBG
Flush MV, Temporary data, Reconstruction MB
Vertical border MB adjustment
VLC
Flush Bitstream
Fig. 7. flow chart of MBG-level parallelism scheme for P/B frames

4.3 MB-level parallel scheme for I frames

As for I frames, the wave-front parallelization algorithm [8] is used. As shown in Fig.8, six processes P1~P6 are employed as an example, where each process encodes one MB row. The number in the MB in Fig.8 stands for the steps when the MB is coded. In step 1, the process P1 encodes the first MB in the top row and sends the intra sub-mode and reconstructed MB data to the process P2. In step 2, the process P1 does the same work on the second MB. Then, in step 3 the process P2 can start encoding the first MB in the second row since the all the necessary information on its neighboring MBs are available, meanwhile the process P1 continues coding the MBs in the top row. In the same way, one new process can be activated every two steps until all MB rows are assigned to different processes and encoded simultaneously. In practice, message mechanism is used for data exchange between processes.

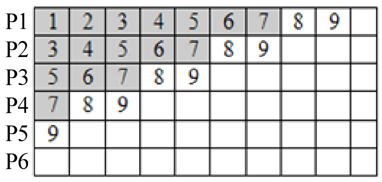


Fig. 8. MB-level parallel scheme for I frames

5 Experiments

We employ 11 HD video sequences presented in Section 2 to evaluate the proposed MBG-level parallelism. The encoder software is developed based on the

AVS1_P2 reference code (*RM52k*) [12]. Some coding algorithms are modified (e.g. using the more efficient hexagon-based block search algorithm [13] instead of the MLS_FME in the AVS reference code [12]) and SIMD optimization is applied to the SAD and MAD functions. Moreover, rate distortion optimization (RDO) and rate control are closed, and the GOP length is set to be 30.

In our system, 45 processor cores on the *Tilera TILExpress-20G*™ card are used for video coding and the other 19 cores are reserved for audio, system and other applications. I frames are encoded using the MB-level parallelism as described in Section 4.3 and P/B frames are encoded using the MBG-level parallelism as described in Sections 3.3 and 4.2. In the MBG-level parallelism, 9$\times$5 MBGs are defined, i.e. 9 slices in each frame and 5 MBGs in each slice. In the following paper, we call this encoder "*MBG Parallelism Encoder*"(*MBGcoder*).

In order to compare with the related parallel schemes, we also develop three anchor systems based on the same modified AVS reference code:

(1) *One Slice Single Core Encoder* (*OScoder*): Single process implementation (for one single core) with one slice in the I/P/B frames;

(2) *Slice Parallelism Encoder* (*SLICEcoder*): The slice-level parallelism for I/P/B frames, where each frame is divided into 45 slices (for 45 cores);

(3) *MB-Slice hybrid Parallelism Encoder* (*MB-SLICEcoder*): The MB-level parallelism in I frames and the slice-level parallelism in P/B frames for a fair comparison to the *MPE* encoder.

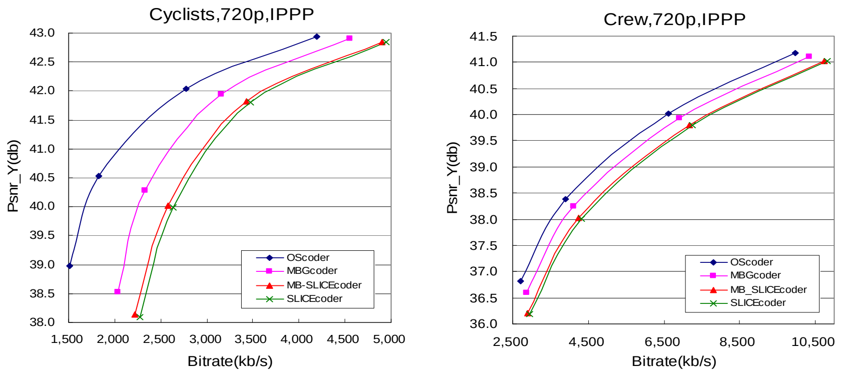


Fig. 9. Performance in different scheme

Fig. 9 shows the performance comparison of the 4 encoders above on two representative sequences (*Cyclists* and *Crew*). It can be seen that quality loss in the proposed *MBGcoder* is obviously less than the *SLICEcoder*. Compared with the *MB-SLICEcoder*, the *MBGcoder* also has apparent performance gains. Since the *MBGcoder* and the *MB-SLICEcoder* employ the same MB-level parallelism, the above results show the effectiveness of the proposed MBG-level parallelism for P/B frames. On the other hand, the *MB-SLICEcoder* does not have much gain compared with the *SLICEoder*, which means the MB-level parallelism contributes very little to the coding quality. This proves that the MBG-level parallelism makes the major contribution to the quality loss saving.

Table 1 shows the gains of the *MBGcoder* and the *SLICEcoder* compared with the *OScoder* using IPPP and IBBP GOPs. The proposed *MBGcoder* saves 52% quality loss in IPPP GOPs and 41% in IBBP GOPs. Table 2 compares the performances of the *MBGcoder* and the *MB-SLICEcoder*. The proposed *MBGcoder* saves about 44% quality loss in IPPP GOPs and 30% in IBBP GOPs by using the MBG-level parallelism in P/B frames.

Table 1. Performance comparison of the proposed *MBGcoder* and the *SLICEcoder*

GOP Structure	scheme	Cyclists	Big Ships	Crew	Raven	Night	Optis	Harbour	City	Spin calendar	Shuttle Start	Sheriff	(average)
IPPP	SLICEcoder (dB)	-1.31	-0.08	-0.64	-0.70	-0.29	-0.2	-0.08	-0.19	-0.18	-0.29	-0.13	-0.37
	MPEcoder (dB)	-0.77	-0.05	-0.28	-0.36	-0.09	-0.12	-0.01	-0.09	-0.08	-0.06	-0.06	-0.18
	Gain Save (%)	41%	42%	56%	49%	68%	39%	87%	55%	53%	81%	56%	**52%**
IBBP	SLICEcoder (dB)	-1.82	-0.19	-0.77	-1.15	-0.4	-0.07	-0.22	-0.19	-0.27	-0.23	-0.15	-0.5
	MBGcoder (dB)	-1.25	-0.16	-0.34	-0.78	-0.17	-0.1	-0.07	-0.10	-0.15	-0.02	-0.08	-0.29
	Gain Save (%)	31%	14%	56%	32%	57%	-33%	69%	46%	45%	91%	48%	**41%**

Table 2. Performance comparison of the proposed *MBGcoder* and the *MB-SLICEcoder*

GOP Structure	scheme	Cyclists	Big Ships	Crew	Raven	Night	Optis	Harbour	City	Spin calendar	Shuttle Start	Sheriff	(average)
IPPP	MB-SLICEcoder (dB)	-1.25	-0.06	-0.59	-0.62	-0.23	-0.20	-0.02	-0.15	-0.14	-0.15	-0.09	-0.32
	MBGcoder (dB)	-0.77	-0.05	-0.28	-0.36	-0.09	-0.12	-0.01	-0.09	-0.08	-0.06	-0.06	-0.18
	Gain Save (%)	38%	24%	52%	42%	59%	38%	55%	43%	38%	64%	41%	**44%**
IBBP	MB-SLICEcoder (dB)	-1.70	-0.16	-0.69	-1.04	-0.33	-0.07	-0.15	-0.14	-0.21	-0.02	-0.10	-0.42
	MBGcoder (dB)	-1.25	-0.16	-0.34	-0.78	-0.17	-0.10	-0.07	-0.10	-0.15	-0.02	-0.08	-0.29
	Gain Save (%)	26%	1%	51%	25%	47%	-42%	54%	28%	29%	-4%	19%	**30%**

Our experiments also show that the *MBGcoder* can achieve real-time coding on SD (Standard Definition) videos using 36 cores, and 14fps on average on 720p HD videos using 45 processor cores. Table 3 shows the average coding times of the *SLICEcoder* and the *MBGcoder* for IPPP GOP structure. The *MBGcoder* only costs 1.7ms (2.5%) more coding time per frame on average, but saves 52% quality loss as shown in table1.

The above experimental results show that the proposed MBG-level parallelism can effectively reduce the quality loss and is more feasible for the many-core processor.

Table 3. Coding speed comparison of the *MBGcoder* and the *MB-SLICEcoder*

scheme	Cyclists	Big Ships	Crew	Raven	Night	Optis	Harbour	City	Spin calendar	Shuttle Start	Sheriff	(average)
SLICEcoder(ms)	64.5	73.2	74.6	69.8	68.2	72.0	72.0	69.3	73.7	65.0	69.7	**70.2**
MBGcoder(ms)	68.5	73.4	74.8	69.6	73.7	72.5	71.0	74.2	74.8	67.3	71.5	**71.9**

6 Conclusions and Future Work

In this paper, we proposed a novel Macro-Block Group (MBG) decomposition scheme for parallel AVS coding on the many-core processor. The MBG is defined as a rectangle region in a frame that consists of a certain number of MBs, which can be encoded independently. In order to keep the consistency with the AVS standard, we developed several techniques such as slice-based MBG partition, mode confining and MVD adjustment. In practice, the MBG-level parallelism and the MB-level parallelism are realized in P/B frames and I frames respectively. On the *TILE64* platform, the proposed scheme achieves 52% (IPPP） and 41% (IBBP) quality saving while keeping the same encoding speed-up compared with the slice-level parallelism.

Future work includes cache optimization, nimbler MBG partitions for better load balance, and finally to achieve real-time on HD videos.

Acknowledgements

This paper is partially supported by the National Basic Research Program of China (973 Program) under contract No. 2009CB320902, the National Natural Science Foundation of China under contract No. 60832004. Especially, we would like to express deep gratitude to our colleagues *Kaijin Wei* and *Qian Huang*. This paper would be incomplete without their tremendous contributions.

References

1. Wiegand, T., Sullivan, G.J., Bjontegaard, G., Luthra, A.: Overview of the H.264/AVC video coding standard. In: IEEE Transactions on Circuits and Systems for Video Technology, Volume 13, Issue 7, July 2003 Page(s):560 – 576 (2003)
2. “Information technology - Advanced coding of audio and video - Part 2:Video” GB/T20090.2 (2006)
3. Chen, Y.K., Li, E.Q., Zhou, X.S., Ge, S.: Implementation of H.264 encoder and decoder on personal computers. In: J. Vis. Commun. Image R. 17 (2006) 509－532 (2005)
4. Barbosa, D.M., Kitajima, J.P., Jr, W.M.: Real-time MPEG encoding in shared-memory multiprocessors, Int. Conf. Parallel Comput. Syst. (1999).
5. Shen, K., Delp, E.J.: A parallel implementation of an mpeg1 encoder: Faster than real-time. In: Proceedings of the SPIE, VOL. 2419, Digital Video Compression: Algorithms and Techniques. (1995)
6. Li, P., Veeravalli, B., Kassim, A.A.: Design and implementation of parallel video encoding strategies using divisible load analysis. In: IEEE Transactions on Circuits and Systems for Video Technology, Volume 15, Issue 9, Sept. 2005 Page(s):1098-1112 (2005)
7. Jung, B., Jeon, B.: Adaptive slice-level parallelism for h.264/avc encoding using pre macroblock mode selection. In: Journal of Visual Communication and Image Representation, Volume 19, no. 8, pp. 558-572, December 2008 (2008)
8. Zhao, Z., Liang, P.: Data partition for wavefront parallelization of H.264 video encoder. In: IEEE International Symposium on Circuits and Systems, 2006. ISCAS 2006. Proceedings. 2006 0-0 0 Page(s):4 pp. – 2672 (2006)
9. Rodriguez, A., Gonzalez, A., Malumbres, M.P.: Hierarchical Parallelization of an H.264/AVC Video Encoder. In: International Symposium on Parallel Computing in Electrical Engineering, 2006. PAR ELEC 2006. 13-17 Sept. 2006 Page(s):363 – 368 (2006)
10. Sun, S.W., Wang, D., Chen, S.M.: A Highly Efficient Parallel Algorithm for H.264 Encoder Based on Macro-Block Region Partition. In: High Performance Computing and Communications, Volume 4782/2007, Page(s):577-585. (2007)
11. Tilera Corporation: ProductBrief_TILEPro64_Web_v2, http://www.tilera.com/
12. Audio Video coding Standard Workgroup of China, http://www.avs.org.cn
13. Zhu, C., Lin, X., Chau, L.P.: Hexagon-based search pattern for fast block motion estimation. In: IEEE Transactions on Circuits and Systems for Video Technology, Volume 12, Issue 5, May 2002 Page(s):349 – 355 (2002)