High-Efficient Parallel CAVLC Encoders on Heterogeneous Multicore Architectures

Huayou SU, Mei WEN, Ju REN, Nan WU, Jun CHAI, Chunyuan ZHANG

Dept. of Computer, National University of Defense Technology, Changsha, China

huayousu@163.com, { meiwen; renju; nanwu; chaijun200306; cyzhang}@nudt.edn.cn

Abstract. *This article presents two high-efficient parallel realizations of the context-based adaptive variable length coding (CAVLC) based on heterogeneous multicore processors. By optimizing the architecture of the CAVLC encoder, three kinds of dependences are eliminated or weaken, including the context-based data dependence, the memory accessing dependence and the control dependence. The CAVLC pipeline is divided into three stages: two scans, coding, and lag packing, and be implemented on two typical heterogeneous multicore architectures. One is a block-based SIMD parallel CAVLC encoder on multicore stream processor STORM. The other is a component-oriented SIMT parallel encoder on massively parallel architecture GPU. Both of them exploited rich data-level parallelism. Experiments results show that compared with the CPU version, more than 70 times of speedup can be obtained for STORM and over 50 times for GPU. The implementation of encoder on STORM can make a real-time processing for 1080p @30fps and GPU-based version can satisfy the requirements for 720p real-time encoding. The throughput of the presented CAVLC encoders is more than 10 times higher than that of published software encoders on DSP and multicore platforms.*

Keywords

CAVLC, software parallel, heterogeneous multicore, real-time HD.

1. Introduction

In the H.264/AVC [1] baseline profile, CAVLC [2] has been widely used to encode the quantized coefficients, which provides considerable improvement of coding efficiency over the conventional coding of UVLC. However, the high coding gain increase comes mainly from its high computational complexity. In addition, strong data dependence, caused by its characteristic of serial processing, makes it almost impossible to implement a software real-time HDTV encoder when using current general-purpose processors.

In the past few years, several performance-oriented CAVLC encoders have been proposed in terms of hardware acceleration and software optimization. In according with the requirements of applications and designing goals, some CAVLC algorithms are proposed based on specific hardware [3-5]. However, those algorithms are still highly arithmetic. Most researches are concerned with accelerating the CAVLC encoder by dedicated hardware [6-10]. Though high efficiency can be gained, dedicated ASIC designs are inflexible, time-consuming, and expensive. It is very burdensome to realize real-time HD H.264 encoder by utilizing hardware. A few software CAVLC encoders are developed to alleviate the problems described above. In [11], it presents a DSP-based implementation of CAVLC tool. Xiao [12] proposed a parallel CAVLC encoder on fine-grained multicore system. A streaming CAVLC algorithm is described in [13].

Heterogeneous parallel processors have more potential than general multicore architectures in parallel computing [22]. Vendors have commoditized many heterogeneous parallel architectures to accelerate applications, such as the multicore stream processors (for example, SPI STORM, Stanford Merrimac, MIT Tile64) and multithread processors (IBM CELL, NVIDIA GPU, AMD Fusion). Two parallel patterns are usually used to exploit data-level parallelism: single instruction multiple data (SIMD) and single instruction multiple thread (SIMT). Considerable high performance has been achieved in signal processing and scientific computation when using multicore stream processors [14-16]. Currently, GPU has been at the leading edge of many-core parallel computational platforms in many research fields. It is mainly due to the high peak performance, high-speed bandwidth, and efficient programming environments, such as NVIDIA CUDA [18]. Many studies focused on accelerating video processing using GPU, such as GPU-based motion estimation [19], H.264 decoder based on GPU [20].

Heterogeneous multicore architecture can apply rich DLP and ILP. However, it is a challenge to develop efficient parallel programs on heterogeneous processors because of the multilevel memory spaces and the software-managed on-chip memories. In this paper, two efficient parallel CAVLC encoders of H.264 are implemented based on heterogeneous parallel platforms. A block-based SIMD parallel CAVLC encoder is proposed based on multicore stream processor STORM, which can achieve real-time H.264 encoding for 1080p @30fps. For massively parallel architecture GPU, a component-oriented SIMT parallel CAVLC is proposed, which satisfies the requirements of real-time encoding for 720p. In order to eliminate or weaken the dependences (the context-based data dependence, the memory accessing dependence and the control dependence), the whole process pipeline is divided into three stages: two scans, coding, and lag packing. In addition, the fast on-chip memory is used to reduce off-chip memory accessing as much as possible for GPU implementation. The experiments show that the proposed parallel CAVLC encoder gains 70 times of speedup compared with the CPU version when using STORM and 50 times of speedup for using GeForce 260+ GTX. Both of them can support real-time HDTV encoding.

2. Background

2.1 Overview of CAVLC

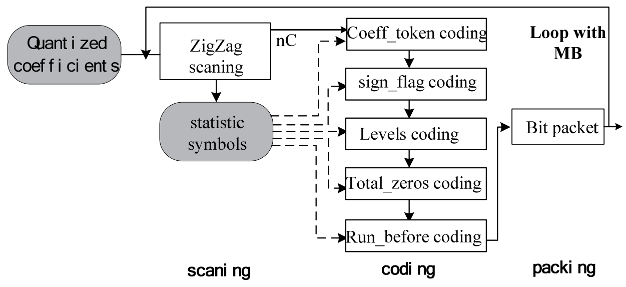


Fig. 1. CAVLC encoder process flow.

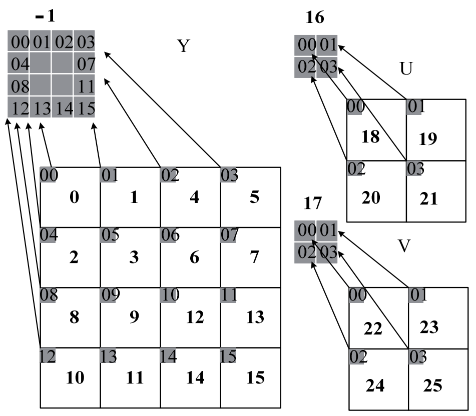


Fig. 2. The order and position of block of a MB.

CAVLC is employed to encode the quantized residual data of the 4x4 or 2x2 blocks. Fig. 1 shows the encoding process of the CAVLC. First, the encoder scans the quantized coefficients in zigzag order block by block and obtains the statistic symbols. Then, five different steps are employed sequentially to encode the symbols. For each macroblock (MB), there are altogether 27 blocks needed to be encoded, including 1 Luma DC block, 16 Luma AC blocks, 2 Chroma DC blocks (size of 2x2) and 8 Chroma AC blocks. As shown in Fig. 2 for a 720p frame, more than 40000 blocks need to be processed, the complexity is very high. The statistic symbols are shown as following:

- **Coeff_token**: the number of nonzero coefficient and number of signed trailing
- **Trailing_Sign_trail**: the sign of trailing ones
- **Levels**: the remaining nonzero coefficients
- **Total_zeros**: the total number of zeros before the last coefficient
- **Run_before**: the number of run zeros preceding each nonzero level in reverse zigzag order

2.2 Heterogeneous Multicore Processors

In this paper, we choose two kinds of heterogeneous multicore processors to implement the CAVLC encoder. One is the typical stream processor. It usually adopts SIMD method to develop parallelism, whereby the execution trace of the instruction can be controlled by programmers. The other one is the massively parallel processor GPU. It executes instructions with SIMT and the routes of instructions are uncontrolled. In the following, two start-of-the-art heterogeneous multicore processors (SPI STORM and NVIDIA GPU) will be described.

STORM-SP16 G220 is a high efficient multicore stream DSP aiming at signal processing and video coding [17]. Fig. 3 shows a basic block diagram of STORM. It contains a system MIPS for scheduling DSP tasks, a DSP MPIS for data handling and the Data Parallel Unit (DPU) for compute-intensive. DPU executes the instructions by SIMD. Each lane executes the same instruction at the same. It can be seen as a static mechanism. Three levels memories are introduced in STORM, including the operation register files (ORF) of each lane, the on-chip local register files (LRF) and the off-chip DRAM. The program consists of two parts: stream program and kernels. The stream program organizes the data stream and kernels. Kernels process data in 16-ways parallel approach.

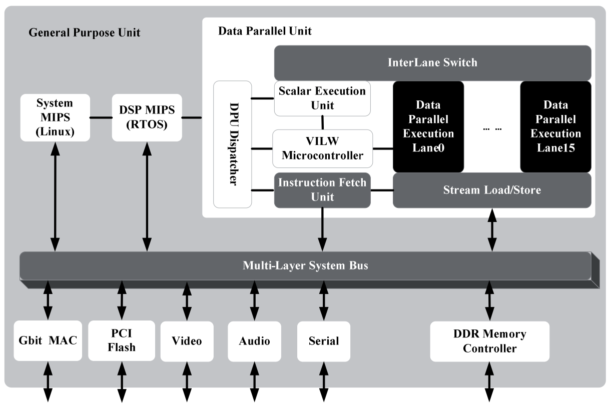


Fig. 3. Architecture of STORM-SP16 G220.

In modern GPU, many parallel processing units called stream multiprocessor (SM) are integrated together. Commonly, each SM contains 8 scalable processors (SP). SM executes the instructions in the way of single instruction multi threads (SIMT) by multiple SPs. In this paper, an abstract architecture of GPU based on CUDA is presented in terms of hardware model, programming model, and memory mode. The CUDA hardware model is a kind of abstract architecture whose core is the scalable SM array, as shown in Fig. 4(a). This architecture consists of SMs and the corresponding memory. In the CUDA framework, computing workloads are encapsulated as kernels. These kernels, executing on GPU, process different data. CUDA program accelerates applications in two levels, including the thread level and the thread-block level. Threads in a block implement the fine-grained parallelism by running on SPs concurrently. They can communicate with each other through shared memory. Relatively, blocks can achieve coarse-grained parallelism, and threads in different blocks can't communicate. Multiple blocks form a grid and complete a computing workload. The programming model is shown in Fig. 4 (b). Fig. 4 (c) shows the CUDA memory model, which consists of a variety of memory devices and corresponding access rules. Each thread has its own local registers and local memory. Each thread block can own a shared on-chip memory. Shared memory provides support for communication between threads in a block.

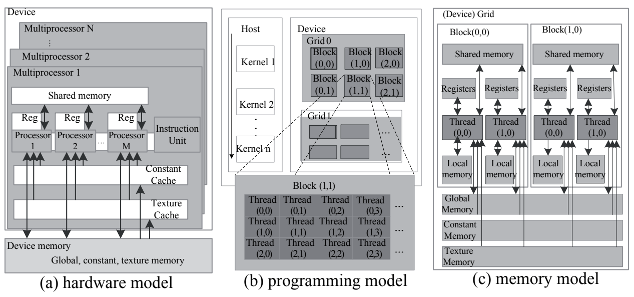
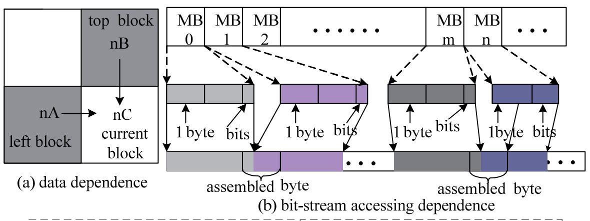


Fig. 4. GPU hardware model, programming model, memory model based on CUDA.



```
if (LumaDC)                                  if( level[i] < 0 )
{                                            {
    nA = BlkA->TotalCoeffLuma;                   i_level_code = -2*level[i] - 1;
    . . .                                    }
}                                            else
else if (LumaAC)                             {
{                                                i_level_code = 2 * level[i] - 2;
    . . .                                    }
}                                            if( i == i_trailing && i_trailing < 3 )
else if (ChromaAC)                           {
{                                                i_level_code -=2;
    nA = BlkA->TotalCoeffChroma;             }
   . . .                                     . . .
}
. . .
```

(c) contral dependence

Fig. 5. Dependences of the CAVLC encoder.

3. Analysis of CAVLC Encoder

In this paper, the x264 [21] is selected as the reference code. Through profiling the instructions of CAVLC, we found three major factors that limit the parallelism of the encoder, including the context-based data dependence, the memory accessing dependence and the control dependence.

Context-based data dependence: Data dependence is caused by the self-adaptive feature of CAVLC. The value of nC is need for look-up tables when coding the symbol coeff_token. The value of nC of the current block is calculated from the total number of non-zero coefficient of the top block and the left block, as shown in Fig. 5(a). The value of nC of current block relies on nA and nB, where nA and nB are the total number of non-zero coefficient of corresponding blocks. This relationship leads to the context-based data dependence. Due to the dependence, the process to current block must wait until its top block and left block are processed.

Accessing dependence: Accessing dependence is due to the variable length encoding characteristic of CAVLC. Since the length of bit-stream of each MB is not constant, the output of current MB must be behind the prior ones. The packing of bit-stream is described in Fig. 5(b). The bit-stream of a frame is packed bit by bit in order of MB. Because the bit-stream of each MB is not byte aligned, the first bit of current MB must connect to the last bit of the former MB. As is shown in Fig. 5(b), the first bit of MB1 connects to the last bit of MB0, and the last two bits combined with the first six bits of MB2 to form an integrated byte.

Control dependence: Control dependence is resulted from the inherent feature of CAVLC algorithm. The control dependence lies in two layers: the frame layer and the block layer. In the frame layer, the branch is mainly caused by different frame types and the different components of a frame. For example, the procedures of I frame and P frame are different, but the same situation exists among luma component and chroma component. In addition, the DC component differs from the AC component. The left side of Fig. 5(c) describes the branch caused by computing the value of nC of different component block. In the block layer, the branch comes from the characteristic of data, such as whether sign_trail is 1 or -1, and whether levels are zero or not, etc. The right side of Fig. 5(c) gives the branch processes of computing the symbol of levels.

4. Block-based SIMD Parallel CAVLC Encoder on Stream Processor

4.1 Optimization of CAVLC Architecture

In order to execute the parallel CAVLC encoder, the first step is optimizing the structure of the conventional CAVLC to overcome the limitations described in section 3. Considering that CAVLC encoder is the last step of H.264 encoder and there is no feedback, it can be assumed that the quantized coefficients of the whole frame are obtained

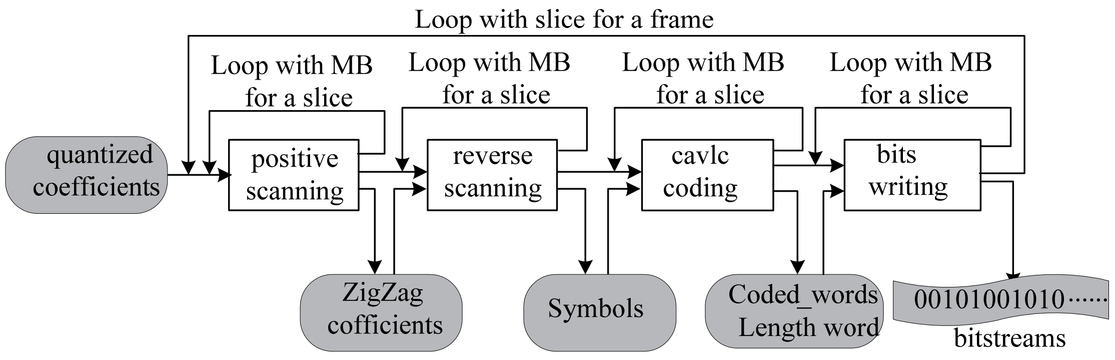


Fig. 6. The proposed CAVLC based on STORM.

before entropy coding. We divided the CAVLC encoder by term of slice into three stages: two scans, coding and lag packing. The proposed CAVLC encoder is shown in Fig. 6.

Two scans: Two scans are employed to gain the statistic symbols: a positive scan and a reverse scan. First, a positive scan is executed to the quantized coefficients which are stored in zigzag order then. The results include the number of non-zero coefficients (total_coeff) of blocks and the zigzagged coefficients. Second, a reverse scan is employed to the zigzagged coefficients and the value of nC is calculated based on the total_coeffs gained in the first scan. The results consist of other symbols and the values of nC. Two advantages are won. The first one is avoiding the redundancy of accessing the quantized coefficients of the adjacent blocks when computing the value of nC, which eliminates the context-based data dependence. The second is reducing the zigzag operations by using clever storage strategy.

Coding: lookup the tables and coding the symbols of an MB in raster order. The results contain two parts: the coded-words and their valid length.

Lag packing: Though the length of bit-stream of each MB is not constant, it is fixed after the symbols are encoded completely. According to the valid length of bit-stream of each MB, the output position can be obtained and a parallel packing can be performed. Thus it can not only eliminate the constraint of accessing dependence, but also improve the performance of bit-stream.

4.2 The Parallel Granularities

The parallel model relies on the parallel granularity. In the field of video coding, sub-block and MB are two common granularities. The parallel patterns on STORM correspond to the two granularities are shown in Fig. 7. For sub-block parallelism, each lane of the STORM processes one block of MB. The 16 lanes can accomplish the coding of an MB. This kind of granularity is fit for the situation of weaken dependence within an MB. For the parallelism of MB-level, an independent MB is assigned to a lane, which is propitious to the case of weaken dependence between MBs. Fortunately, after optimizing the structure of the serial CAVLC encoder, the dependences within MB and between MBs are eliminated or weaken. Therefore, the two granularities mentioned above are suitable. Considering the limitations of the ORF of STORM processor, the block-level parallelism is chosen for implementation in this paper.

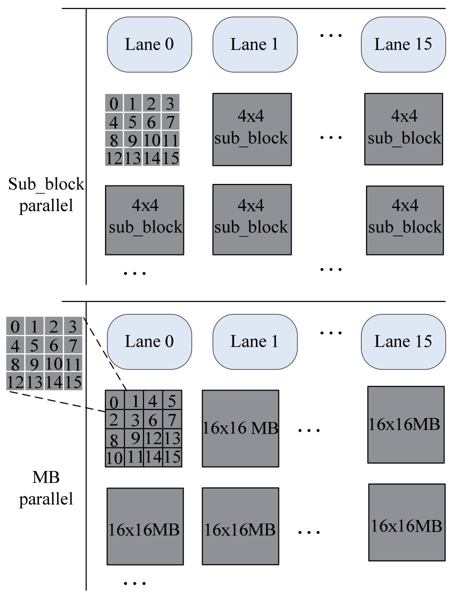


Fig.7. Parallel granularities and the corresponding parallel models on STORM.

4.3 Implementation

As mentioned in section 2, maximum 27 blocks (4x4 block or 2x2 block) need to be coded for an MB. In this paper, the 27 blocks are allocated into 16 Lanes of the STORM processor shown in Fig. 8. For simplifying the programming, two blocks are allocated to a Lane. Owing to only 27 blocks within an MB while the target processor contains 16 Lanes, some Lanes process useless data. As is shown, one block is valid in Lane0, which is the Luma DC block. From Lane1 to Lane8, two luma AC blocks are processed. Lane9 deals with the two Chroma DC blocks. The Chroma AC blocks are assigned to Lanes from 10 to 13. Lane14 and Lane15 are invalid. For STORM, the parallel degree is always 16, five kernels are designed to perform the CAVLC coding process. The kernels are organized as Fig. 9, which is one kind of producer-consumer relation. Limited by the capacity of the LRF, the kernels process one row of MBs in each time. The output stream of the last kernel is used as the input stream for the next kernel, which can reduce the accessing to off-chip memory.

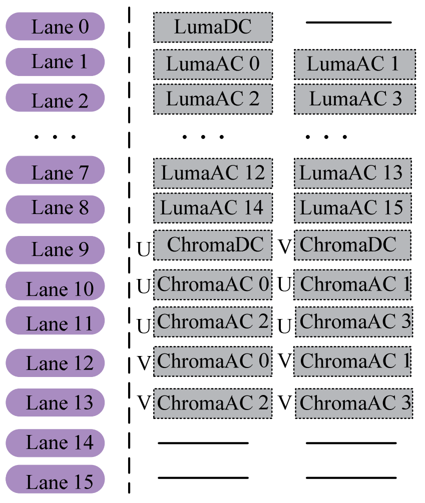


Fig.8. The allocation of data of CAVLC encoder.

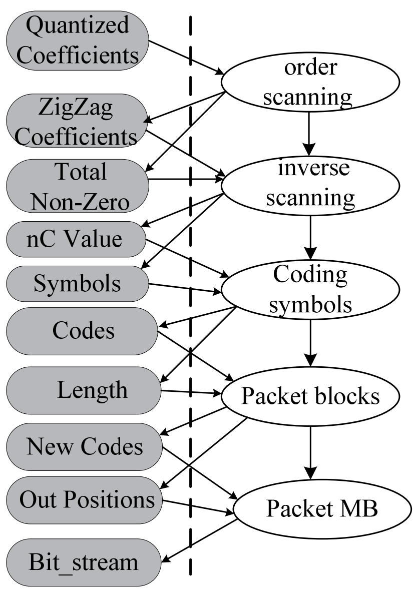


Fig 9. The organization of the kernels and streams.

5. Component-oriented SIMT Parallel CAVLC Encoder on GPU

GPU can offer more powerful computational capacity and bigger memory spaces. Large amounts of parallelism and efficient hiding delay strategy are critical for high efficient performance on such architecture. In order to execute the parallel CAVLC encoder on GPU, an innovative CAVLC is proposed based in Fig. 10, which is called component-oriented CAVLC. As shown in Fig. 10, each stage of CAVLC pipeline is divided in term of frame. For the sake of minimizing the performance loss of the target parallel CAVLC encoder owing to branch operations, a component-oriented coding is used instead of the MB-oriented approach. It processes the coefficients frame by frame in order of Luma DC, Luma AC, Chroma DC, Chroma AC, instead of processing the four component coefficients MB by MB. For example, until all the coefficients of Luma DC of a frame are executed, the component of Luam AC can be encoded, and so on. The unnecessary branches can be effectively reduced through this way. After optimizing the architecture, the algorithm is designed based on block for each component of CAVLC and can develop high parallelism. The performance of CUDA program relies on the parallel level, the organization of data (memory model) and the characteristic of the data to be encoded. Therefore, we choose the optimal parallel configuration according to the characteristic of data and use shared memory to reduce the accessing to global memory as much as possible. In the discussion of this section, 1080p (1920x1080) video frame is chosen as the input.

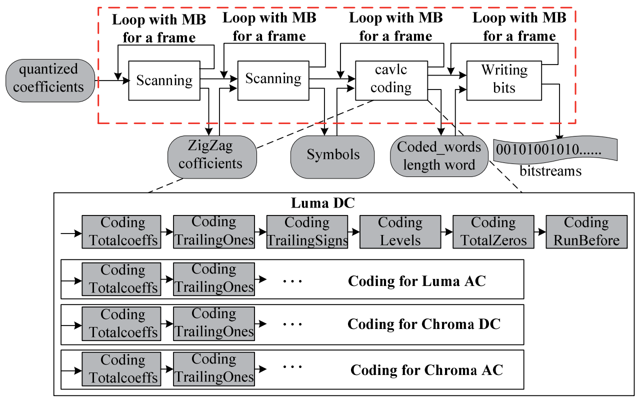


Fig.10. The component-oriented CAVLC encoder.

5.1 Scanning the Quantized Coefficients

A. The first scanning

The first scanning aims at the quantized coefficients and calculates the number of non-zero coefficient of each block (TotalCoeffs). It is a forward scan. In this stage, each thread was assigned to deal with a 4x4 block. Considering that a 4x4 block contains 16 coefficients, we configure the thread block with 128 threads. 16 sequential threads take charge of an MB together and 8 MBs are encoded by a thread block. In order to increase the number of thread blocks within a grid, components of Luma and Chroma are performed in the same kernel. For the sake of avoiding branch within a warp, threads in a warp deal with one kind of component only. The implementation process is shown in Fig. 11. The interval between the start accessing position of adjacent threads is 32B (16 coefficients) when visiting their corresponding residual data of blocks. So if each thread reads its data from global memory to registers directly, it can't meet the requirement of combined-access. Rather, 128 times of accessing are needed to read the 256 coefficients of different blocks for the threads of a half-warp. Each accessing, in turn, will transform 64B data, but the effective data are 4B. In order to optimize this issue, the shared memory is used as a buffer. First, the data needed by a half-warp of threads is loaded to the shared memory from the global memory by utilizing the mechanism of combined-access supported by global memory. Then, each thread visits the corresponding data through different banks supported by shared memory. Through this way, the throughput can be improved significantly and the pressure of the register can be relaxed. All data transformed from global memory are valid and 512B data can be obtained by 16 times of combined-access. In addition, after scanning the quantized coefficients, zigzag storage strategy is introduced to write back these coefficients.

B. The second scanning

The calculation of the value of nC needs the TotalCoeff coefficients of its adjacent left block (nA) and that of the top block (nB). In order to make better use of the local data, we divide a frame into several regions of 4MBx2MB. One thread block executes the values of nC of blocks in the same region, as is shown in Fig. 12. The program first loads all data needed to the shared memory, then each thread visits nA and nB, where one TotalCoeff coefficient can be used as either nA or nB, as is shown in the small black grid of Fig. 12. During this scanning, other symbols (Trailing_Sign_trail, Levels, TotalZeros, RunBefores) are counted. It is a reverse scan to the zigzaged coefficients generated in the first scanning.

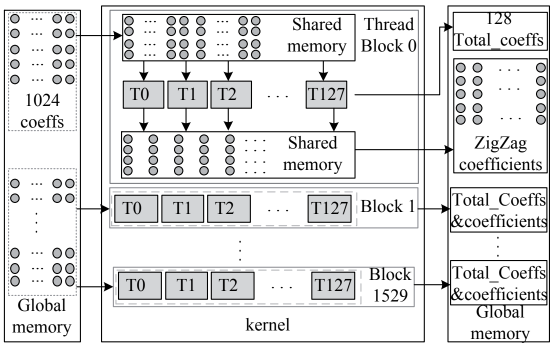


Fig. 11. The parallel execution of calculation of TotalCoeffs.

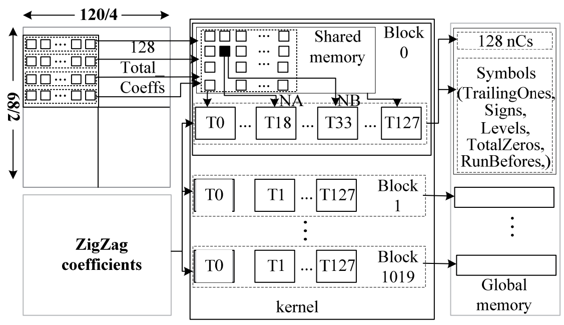


Fig. 12. Calculation of nC and other symbols.

5.2 Coding the Symbols

The process of coding symbols is almost the same for different components of a frame except for different look-up tables. Below we just explain the implementation of parallel coding for component Luma AC by CUDA. Since the process of coding is block-based, what's needed is to encode the symbols according the value of nC. The configuration is similar to that of calculating the value of nC. In addition, the look-up tables are firstly loaded to the shared memory to speed up the lookup operation. Because the bit-streams are kept until all symbols are encoded, temporary memories are required for each block to store the corresponding bit-streams. In our implementation, maximal of 26 short-words is used for keeping the symbols of a block. Therefore, 26 words are necessary for each block to store the bit-stream of each symbol and its corresponding valid length. Among those memory units, some of them are not used. Fig. 13 shows the organization of a thread block for encoding the symbols. In the grids of Coded-words, the gray area represents the valid bit-stream, while the white region is the redundant space for each block.

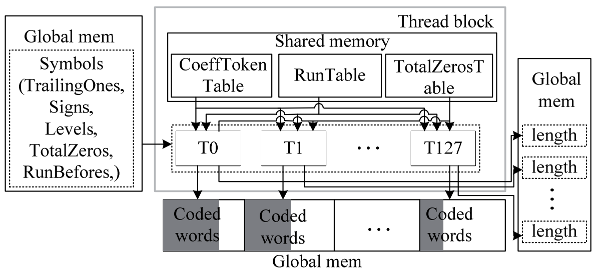


Fig.13. Organization of a thread block when coding symbols.

5.3 Parallel Packing

We first analyze the necessity of parallel packing. Tab. 1 shows some major performance parameters of CAVLC encoder based on GPU for an I frame in the situations of serial packing and parallel packing when using 1080p and 720p as test sequence. As can be seen from the table, the execution time of parallel method is significantly less than that of the serial method. But more crucially, the data transferred between CPU and GPU when adopting serial output is far larger than the amount of parallel one. The reason is that only the valid data of Coded-words is transferred with parallel packing, while the white region of Coded-words and the memory of length are copied back to CPU with serial output. Though other tools of H.264 encoder can achieve a very significant improvement, it is impossible to satisfy the requirement of real-time HDTV if parallel packing is not adopted. In this article, two steps are employed to complete the parallel packing. The first step executes the combination of bit-stream of a MB and the computation of the out position, the shift bits and shift mode of the bit-stream for each MB. The second step performs parallel packing based on the parameters obtained in the first step.

A. Calculation the out position for each MB

In order to implement parallel output, some parameters are needed as follows.

Parameters	Blue_sky (1080p)			In_to_tree (720p)		
	Serial	Parallel	Speedup	Serial	Parallel	Speedup
Execution time (ms)	29.8	2.53	11.78	15.6	1.35	11.56
Transform time (ms)	23.4	0.39	60	10.1	0.28	36.1
Total (ms)	52.2	2.92	17.87	25.7	1.63	15.77
Transform data size (KB)	23300.7	94.7	246	10279.8	51.4	200

Tab. 1. Performance parameters of CAVLC encoder for I frame.

1) The number of integral byte of bit-stream for each MB (n)
2) The number of the remaining bits less than one byte of each MB (m, $m < 8$)
3) The shift mode and shift bits for each MB

The first step is packing the bit-stream of different blocks of an MB to form an integrated one. A thread processes an MB and computes the length (n*8+m) of the bit-stream. According to the length, the start position of output for each MB can be obtained. The iteration method is adopted to speed up the calculation as shown in Fig. 14. In each iteration, the number of valid threads is half of the total and the interval between valid threads becomes closer, which can keep the warps from diverging gradually. Furthermore, the results from the last iteration are reused in the next iteration.

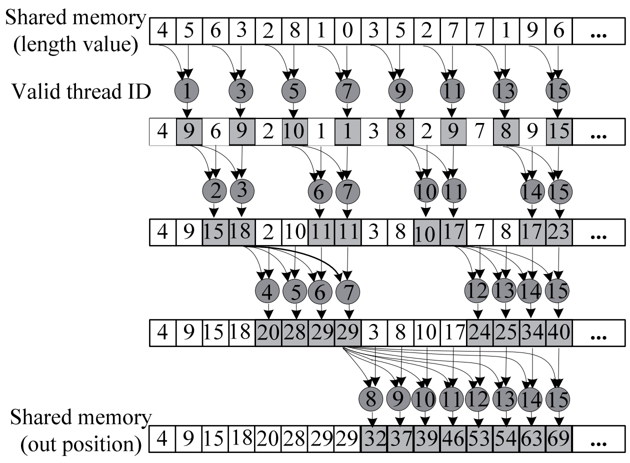


Fig. 14. Calculation of start position for each MB.

B. Parallel Writing bit-stream of MBs

In this step, each thread disposes the writing back of bit-stream for an MB. If the remaining bits are less than a byte then the missing bits is fetched from the next MB. In our implementation, a composed byte is generated by shifting the previous bit-stream towards left and the next bit-stream towards right. The bit-shifted is 8-m for left-shift and m for right-shift. Fig. 15 shows the progressing of parallel output. In the first writing, thread T0 writes the first byte of the bit-stream of MB0. Thread T1 writes the composed byte of the last two bits of the first byte and the first six bits of the second byte of the bit-stream of MB1. The data which thread T0 writes in the last time is a composite byte of the last two bits of MB0 and the first six bits of MB1. Though the lengths of the bit-streams of MB are varied, it will result in branch within a warp. The high parallelism and less data transformed can improve the speed of packing.

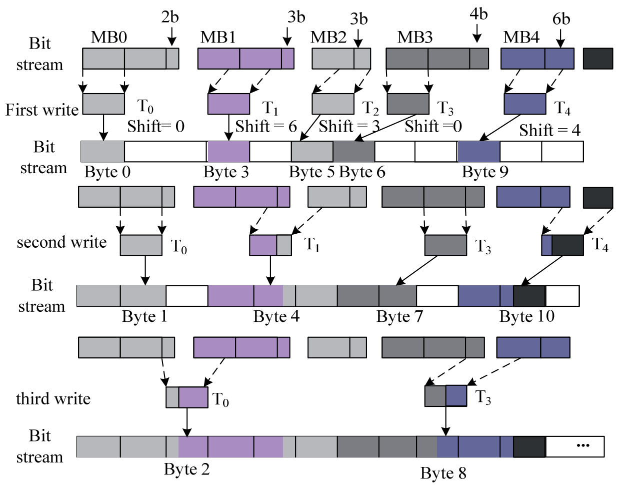


Fig. 15. Parallel packing.

6. Experimental Results

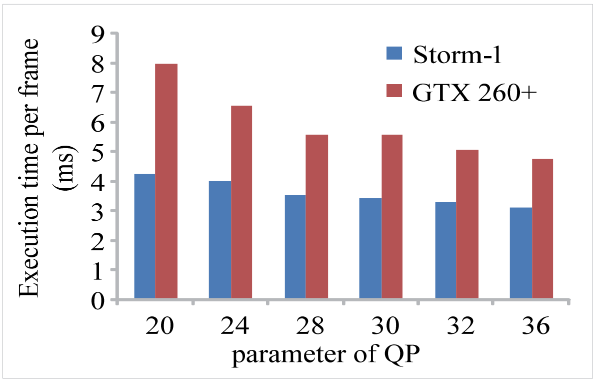
To evaluate the performance of the proposed parallel CAVLC encoders, the following development environments are used: AMD Athlon 5200+ X2 Dual Core 2.7 GHz with 2GB memory, stream processor STORM G220 (700 MHz), NVIDIA GeForce 260+GTX(1.29 GHz) with 889MB DRAM. Since our target is real-time HDTV, 1080p (Blue_sky) and 720p (Into_tree) are selected as test sequences.

The performance differs from different configuration of parameters. Varied encoding patterns and values of parameter QP will impact the performance of CAVLC encoder significantly. That's why we first test the performance of the proposed parallel CAVLC encoders under different values of QP. The results are shown as Fig. 16. The bigger the value of QP is, the faster the speed of CAVLC encoder is. Analysis to the reference program tells us that the execution time of CAVLC occupies about 15% of the total time. According the percentage of CAVLC encoder occupied in the whole H.264 encoder, real-time coding of HDTV 1080p can be satisfied when using STORM and real-time coding requirements of 720p can be met on GPU. In fact, the actual situation is even better. After mapping other tools (motion estimate, intra coding, de-block filter) of H.264 encoder onto the target architectures, the coding speed of the H.264 encoder based STORM can achieve real-time processing requirements of 30fps for 1080p video format and the performance of the

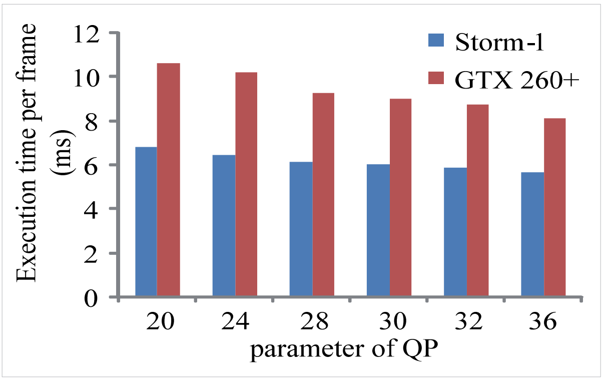
H.264 encoder on GPU can accommodate the real-time encoding of 720p @30fps. More detailed information can be gained from Tab. 2. The high performance mainly comes from the following three aspects. First, when all computation-intensive components of H.264/AVC encoder are performed with parallel methods, the number of data transferred between systems (for STORM, they are system MIPS and DSP MIPS; for GPU, they refer CPU and GPU) is the smallest. Second, TLP can be employed to hide the delay caused by data transfers. Three, motion estimation is the most time-consuming tools in H.264/AVC, which is proportionally around 70% but it achieves the best parallelism. From the graph, it can be seen that the throughput of the encoder based on STORM is much higher than that of the proposed encoder based on GPU, which comes from the different of the two architectures. Accessing of data is almost on-chip memory access in STORM. The access cycle is about 5 to 10 cycles. While in GPU, almost all of the data are firstly stored in global memory which is a kind of off-chip memory. Its visit's time reaches 400 to 600 cycles, about 100 times slower than accessing in STORM. Except that the time of data transfers between CPU and GPU is an important factor that limits the performance of the GPU-based applications.

Then, we evaluate the performance of the proposed CAVLC encoder and compare the results with those of using CPU version. The detailed information is shown in Tab. 3. The time of the column others includes transforming time and startup costs of kernels. As can be seen from the table, compared with execution time when using CPU only, the parallel CAVLC can achieve 72.27x speedup when using STORM and 48.4x speedup with the assistance of GPU. Fig. 17 depicts the percentage of execution time for each major model of the proposed encoder when using 1080p video format. It can be seen from the figure, in the STORM implementation version, the time speeding on packing bit-stream occupies about 45%. That is because almost all the operations of packing bit-stream are bits operations, while the STORM are designed aiming at integer operation. In the GPU-based CAVLC encoder, the execution time of various parts of our implementation is very even, ranging from 20% to 30%, as shown in Fig. 17(b). That is to say our system shows good balance. In order to avoid the problem of bottleneck in zigzag scan in [12], we use clever storage strategy and the total time of the order scan occupies about 15.5%, as can be seen in Fig. 17(b). The percentage of packing bit-stream (packing_blocks and packing_MB) is about 30%, which is far less than 66% published in [13]. A proportion of time of the proposed CAVLC for different component is shown in Fig. 17(c). From the figure, the time speeding on Luma AC is over 50%.

The compare between the proposed CAVLC encoders and other published software ones can be seen in Tab. 4. It can be seen from the table, compared with the CAVLC encoder on DSP, 17.18 and 11.17 times of speedup can be gained for the CAVLC based on STORM and the one based on GPU. Compared to AsAP [12], the speedups are 9.68 and 6.29 times. The performance of the proposed block-based CAVLC encoder on STORM is close to that of the MB-based parallel CAVLC encoder described in [13].



(a) performance for 720p



(b) performance for 1080p

Fig. 16. Performance of CAVLC encoder under varied QP.

Test sequences	Encoder	Execution time per frame (ms)					Speed (fps)
		ME	Intra coding	CAVLC	Filter	Others	
In_to_tree (720p)	STORM	9.17	1.20	3.44	0.98	2.07	59.3
	GTX 260+	15.4	3.21	5.29	3.54	4.42	31.4
Blue_sky (1080p)	STORM	17.49	3.36	6.06	1.65	4.44	30.3
	GTX 260+	25.52	6.01	9.14	6.61	8.39	18.0

Tab. 2. Performance of the H.264 encoder based on heterogeneous platforms.

Test sequences	Platforms	Execution time per frame (ms)					Speedup Ratio
		Scan	Coding	Packing	Others	Total	
In_to_tree (720p)	CPU only	NA	NA	NA	NA	201	1
	STORM	1.03	0.66	1.48	0.29	3.44	58
	GTX 260+	1.47	1.10	1.35	1.37	5.29	38
Blue_sky (1080p)	CPU only	NA	NA	NA	NA	438	1
	STORM	1.81	1.15	2.72	0.38	6.06	72
	GTX 260+	2.82	1.86	2.61	1.76	9.05	48

Tab. 3. Percentage and speedup of various parts of the proposed CAVLC encoder.

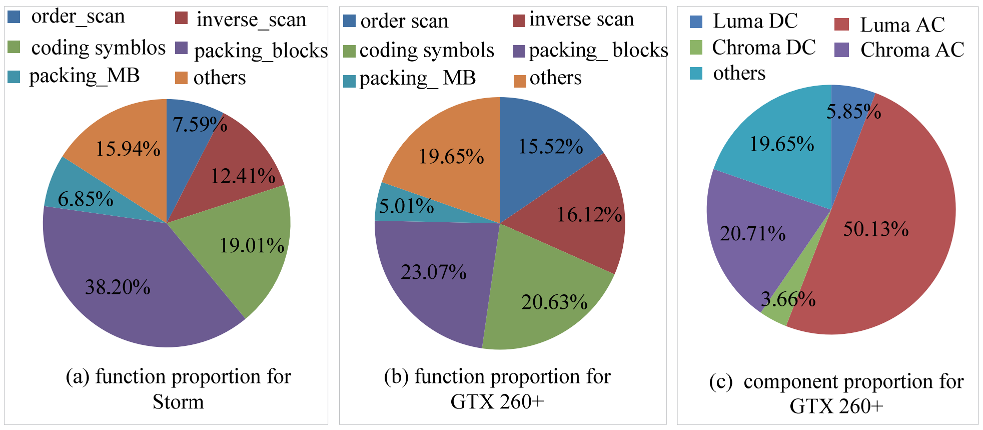


Fig. 17. The proportion of different parts of the proposed CAVLC encoders.

Platforms	Processor type	Frequency	Test sequence	Execution time per frame
DSP TI C642 [11]	8-way VLIW	600MHz	720p QP = 24	59.1ms
Multi-core AsAP [12]	15 cores MIMD	1.07GHz	720p QP = 20	33.3ms
Stream processor STORM [13]	16 lane SIMD	700MHz	1080p QP = 30	5.1ms
Stream processor our work	16 lane SIMD	700MHz	720p QP = 30	3.44ms
GPU our work	216 cores SIMT	1.29GHz	720p QP = 30	5.29ms

Tab. 4. Performance of CAVLC encoder on different platforms.

7. Conclusion

In this article, a high-performance SIMD parallel CAVLC encoder based on multicore stream processor STORM and an efficient SIMT parallel one based on GPU are presented. In order to make full use of the power computational resources of processors, we first optimize the architecture of the conventional CAVLC encoder. For STORM processor, a segmentation of functional model is introduced in term of slice, which eliminates or weakens the dependences of the serial CAVLC encoder. Aiming at the GPU architecture, a component-oriented CAVLC is proposed. In summary, three strategies are introduced as following:

- **Two scans:** to eliminate the context-based data dependence.
- **Component-oriented coding:** to weaken the control dependence
- **Lag packing:** to solve the problem of parallel packing.

Experiments results show that the proposed parallel CAVLC encoders can achieve significant performance. Compared with the CPU version, more than 70 times of speedup can be obtained for STORM and over 50 times for GPU. The implementation of STORM can make a real-time processing for 1080p @30fps and GPU-based version can satisfy the requirements for 720p real-time encoding. The throughput of the presented CAVLC encoder is more than 10 times higher than that of published software encoders on DSP and multicore platforms. From the results, we also found that the differences between CAVLC encoder corresponding to the two heterogeneous multicore platforms are mostly due to the organization of the different memory spaces.

Acknowledgements

The authors gratefully acknowledge supports from National Nature Science Foundation of China under NSFC No. 61033008, 60703073 and 61103080, Research Fund for the Doctoral Program of Higher Education of China under SRFDP No. 20104307110002.

References

[1] Joint Video Team (JVT) of ISO/IEC MPEG and ITU-T VCEG: *Draft ITU-T recommendation and final draft international*

standard of joint video specification (ITU-T Rec. H.264/ISO/IEC 14 496-10 AVC). JVT-G050 (2003).

[2] BJØNTEGAARD, G., LILLEVOLD, K. Context-adaptive VLC (CAVLC) coding of coefficients. *Doc.JVT-028, JVT of ISO MPEG&ITU VCEG*. 3rd Meeting, Fairfax (Virginia, USA), May. 2002.

[3] HEO, J., KIM, S. H., HO, Y. S. New CAVLC encoding algorithm for lossless intra coding in H.264/AVC. In *Proceedings of Picture Coding Symposium 2009*. Chicago (USA), May 2009, p. 77-80.

[4] DONG, Z. D., HAI, D. Q. Improvement of CAVLC code LUT algorithm in H.264 encoder. *Television Technique*, 2004, vol. 1, p. 26-27.

[5] ZHANG, D., ZHANG, M., ZHANG, J., ZHENG W. A new kind of Adaptive Variable Length Coding algorithm. *Zhe Jiang University Transaction*, 2006, vol. 40, no. 5, p. 783-786.

[6] XU, M. H., LI, K., XUAN, X. G., FAN, Y. L. Optimization of CAVLC algorithm and its FPGA implementation. In *International Conference on Electronic Packaging Technology & High Density Packaging2008*. Shanghai (China), 2008, p. 1-4.

[7] CHIEN, C., LU, K., SHIH, Y., GUO, J. A high performance CAVLC encoder design for MPEG-4 AVC/H.264 video coding applications. In *Proceedings of ISCAS 2006*. Island of Kos (Greece), 2006, p. 3838-3841.

[8] HAN, C. S., LEE, J. H. Area efficient and high throughput CAVLC encoder for 1920×1080@30p H.264/AVC. In *Proceedings of International Conference on Consumer Electronics 2009*, p. 1-2.

[9] YI, Y., SONG, B. C. High-speed CAVLC encoder for 1080p 60-Hz H.264 codec. *Signal Processing Letters*, 2008, vol. 15, p. 891-894.

[10] TSAI, T. H., CHANG, S. P., FANG, T. L. Highly efficient CAVLC encoder for MPEG-4 AVC/H.264. *Circuits, Devices & Systems,* 2009, vol. 3, no. 3, p. 116-124.

[11] DAMAK, T. H., WERDA, I., SAMET, A. DSP CAVLC implementation and optimization for H.264-AVC baseline encoder. In *Proceedings of International Conference on Electronics, Circuits and Systems*, 2008, p. 45-48.

[12] XIAO, Z., BAAS, B. A high-performance parallel CAVLC encoder on a fine-grained many-core system. *In Proceedings of International Conference on Computer Design*, 2008, p. 248-254.

[13] REN, J. HE, Y., WU, W., WEN, M., WU, N., ZHANG, C. Y. Software parallel CAVLC encoder based on stream processing. In *IEEE/ACM/IFIP 7th Workshop on Embedded Systems for Real-Time Multimedia*, 2009, p. 126-133.

[14] KHAILANY, B., DALLY, W. J., RIXNER, S. Imagine: signal and imagine processing with streams. *Hotchips 2000*, Stanford, CA.

[15] THIES, W. StreamIt: A language for streaming applications. In *Proceedings of International Conference on Compiler Construction*, 2002. Grenoble (France), 2002.

[16] DALLY, W. J., HANRAHAN, P., EREZ, M., KNIGHT, T. J. Merrimac: supercomputing with streams. In *SC2003*. Phoenix (USA), 2003, 8 p.

[17] Stream Processors Inc. *SPI software Documentation.* Available at: http://www.streamprocessors.com, 2008.

[18] NVIDIA, *NVIDIA CUDA Compute Unified Device Architecure-Programming Guide Version 1.1*, 2007.

[19] HO, C.-W. Motion estimation for H.264/AVC using programmable graphics hardware. In *Proceedings of International Conference on Multimedia and Expo ICME2006.* Toronto (Canada), 2006, p. 2049-2052.

[20] SHEN, G., GAO, G. P., LI, S., SHUM, H. Y., ZHANG, Y. Q. Accelerate video decoding with generic GPU. *IEEE Transactions on Circuits and Systems for Video Technology*, 2005, vol. 15, p. 685-693.

[21] *Reference software X264-060805*, Available at: http://www.videolan.org/developers/x264.html.

[22] HILL, M. D., MARTY, M. R. Amdahl's law in the multicore era. *Computer*, 2008, vol. 41, no. 7, p. 33-38.

About Authors ...

Huayou SU was born in 1985 in Guilin, P. R. China. He received his M.Sc. from the National University of Defense Technology (NUDT) in 2008. His research interests include multimedia computing, parallel programming and computer architecture. Now he is a Ph.D. student at the same faculty. He focuses on parallel programming models aiming at multimedia applications with his classmates.

Mei WEN is an associate professor in the National Laboratory for Parallel and Distributed Processing of NUDT, China. Her research interests include computer architecture and parallel processing. Wen has a BS, a MS, and a PhD in computer science and technology from the National University of Defense Technology.

Ju REN received the M.Sc. and Ph.D. degree in the Computer School of NUDT, in 2006 and 2010, respectively. His research interests include multimedia processing and parallel computing.

Nan WU received the M.Sc. and Ph.D. degree in the Computer School of NUDT, in 2005 and 2008, respectively. His research interests include computer architecture and parallel processing.

Jun CHAI was born in 1985 in Chongqing, P. R. China. He received his M.Sc. from NUDT in 2007. Now he is a Ph.D. student at the same faculty. He focuses on parallel programming especially for scientific computing.

Chunyuan ZHANG graduated in 1985 from NUDT and became a teacher in the Dept. of Computer where he received his further degrees (M.Sc. 1990, Ph.D. 1996). Now he is the head of education department of CS School of NUDT. A scholar who holds particular subvention of State Department of China. His current research is focused on computer architecture, operating system supports for heterogeneous platform and multimedia processing. He is an author or co-author of about 60 research articles published in international journals or conference proceedings.