Science in China Series F: Information Sciences

www.scichina.com
info.scichina.com
www.springerlink.com

Adaptive data-driven parallelization of multi-view video coding on multi-core processor

PANG Yi†, HU WeiDong, SUN LiFeng & YANG ShiQiang

Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China

Multi-view video coding (MVC) comprises rich 3D information and is widely used in new visual media, such as 3DTV and free viewpoint TV (FTV). However, even with mainstream computer manufacturers migrating to multi-core processors, the huge computational requirement of MVC currently prohibits its wide use in consumer markets. In this paper, we demonstrate the design and implementation of the first parallel MVC system on Cell Broadband Engine$^{\mathrm{TM}}$ processor which is a state-of-the-art multi-core processor. We propose a task-dispatching algorithm which is adaptive data-driven on the frame level for MVC, and implement a parallel multi-view video decoder with modified H.264/AVC codec on real machine. This approach provides scalable speedup (up to 16 times on sixteen cores) through proper local store management, utilization of code locality and SIMD improvement. Decoding speed, speedup and utilization rate of cores are expressed in experimental results.

adaptive data-driven, multi-view video coding, Cell Broadband Engine$^{\mathrm{TM}}$ Processor, parallelization

1 Introduction

The deployment of 3D techniques is one of the most promising fields regarding the development of new applications for natural video scenes rendering. With increasing interest in 3D television (3DTV) and free viewpoint video (FVV), the convergence of technologies from computer graphics, computer vision, multimedia and related fields leads to the advancement of these new media. As the core technology of 3D video, the multi-view video technology is in essence capturing a video of an object from different angles with multiple cameras, encoding the different angles, transmitting the data, decoding, and synthesizing a 3D result. Figure 1 shows three main schemes of camera configurations. The multi-view video technology resolves presentation, interaction, codec and transmission of free-view and 3D interactive video formatting. The multi-view video has fueled the rapid expansion of applications into free viewpoint[1], three dimensional displays[2,3], and high performance imaging[4].

Multi-view video coding (MVC) is currently being standardized and used to compress multi-view video for delivery and storage. The Joint Video Team (JVT) comprising Video Coding Ex-

Received May 29, 2008; accepted October 10, 2008
doi: 10.1007/s11432-009-0042-8
†Corresponding author (email: pangy@mails.tsinghua.edu.cn)
Supported partially by the National Natural Science Foundation of China (Grant No. 60503063) and the National High-Tech Research & Development Program of China (Grant No. 2006AA01Z321), and the National Basic Research Program of China (Grant No. 2006CB303103)

perts Group (VCEG) and Moving Picture Experts Group (MPEG) standardization groups issued a Call-for-proposal on MVC[5] and undertook the effort to standardize an MVC standard by extending H.264/AVC and including common tools optimized for MVC. The MVC standard release schedule is 2008.

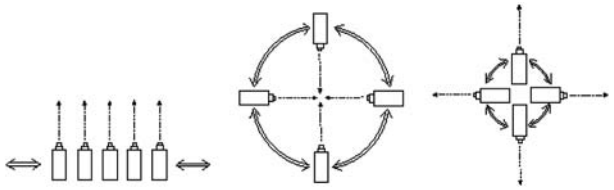

Figure 1 The configurations of multi-view video cameras.

Though all of these advances in MVC have shown great potential in consumer markets, the huge computational requirement prohibits its wide use. MVC inherits the processing algorithm for one frame from H.264/AVC[6−8]. The main difference between H.264/AVC and MVC is the prediction structure. Traditional coding technologies, such as motion compensation and view disparity compensation, are not efficient for MVC. To improve the motion compensation in MVC, we utilize "global motion compensation" which incorporates motion compensation and view disparity compensation. While motion compensation makes use of time redundancy, view disparity compensation makes use of space redundancy. Furthermore, the prediction structure of MVC is static while H.264/AVC is not. Data transmission and computing complexity of MVC increase linearly with the number of views. As the application of MVC increases, real-time coding becomes more and more essential.

As computer architecture evolves from uniprocessor to multi-core processors, exploiting thread level parallelism offers better performance and additional functions from multimedia applications. However, it is not easy to parallelize algorithms for multi-core processor, and is even harder to parallelize them close to linear speedup. As multi-core processor becomes the trend in architecture design, MVC continues taking advantages of the computational resources and good parallelism. Cell Broadband Engine^{TM} (Cell/B.E.) processor is a state-of-the-art multi-core processor co-designed by Sony, Toshiba and IBM (STI), and has parallel program model, broadband data transmission, powerfully media processing, etc. In this study, Cell/B.E. processor and MVC are integrated to increase processing performance by parallel computing. We propose a task-dispatching algorithm which is adaptive data-driven on frame level for MVC and implement a parallel MVC decoder with modified H.264/AVC codec on real machine. This approach provides scalable speedup (up to sixteen times on sixteen cores) through proper local store management, utilization of code locality and SIMD improvement.

This study makes the following contributions:

• design and implement the first parallel MVC system on Cell/B.E. processor with optimization which provides scalable speedup;

• propose an adaptive data-driven parallel algorithm for MVC;

• different from other video decoding parallel algorithms, we make frame parallel granularity.

The paper is organized as follows. Section 2 introduces related works. Section 3 analyzes the parallelism of MVC, proposes our parallel MVC program model, and depicts why we choose data parallel approach and frame as parallel granularity. In section 4, the concept of data-driven is proposed and utilized to MVC. The whole work of MVC is categorized into two types of tasks: tasks dispatching and frame decoding. Section 5 introduces the implementation of MVC on multi-core processor. Cell/B.E. processor as an instantiation of the target multi-core processor template is depicted in detail, and the key technologies of implementation are introduced. In section 6, experimental (speed, speedup, utilization rate of cores, and decoding) results are shown. Section 7 depicts conclusion and future works.

2 Related work

Video coding is well known to be computationally demanding. With development of video coding standard and CPU architecture design, the research work of accelerating video coding never stops. Refs. [9–12] introduced parallel computing of MPEG-1, H.261, MPEG-4 and H.264/AVC

respectively. The former work depicts that video coding standard has good parallelism and computationally demanding.

Wang et al.[13] depicted a novel high definition TV (HDTV) video decoder and decentralized control scheme with a data-driven architecture on function level. The data-driven architecture is adopted to make each processing unit operate when the processing data and buffer are available, and therefore exploits high computing efficiency of each unit. In this work, we transplant the principle of Wang's data-driven architecture to MVC.

MVC as an extension of H.264/AVC also has good parallelism, and related works are introduced in refs. [14, 15]. In ref. [14], a parallelization methodology for MVC based on hyper space theory is presented and tested on multi-processor platform with modified H.264/AVC codec. This theory uses a group of frames as parallel granularity. Experimental results show that the proposed method can speed up processing of multi-view video coding and obtain low rate distortion results. As our platform is multi-core processor not multi-processor, we use frame as parallel granularity and achieve higher speed than that in ref. [14]. Ref. [15] analyzed the parallelization of MVC based on selections of optimal theories and methods using frame as the parallel granularity and depicted benefits from execution MVC in parallel on Cell/B.E. blade. In this study, we enhance our dispatching algorithm by implementing on real Cell/B.E. blade and using adaptive rather than static data-driven principle.

Nanda et al.[16] introduced IBM's video surveillance server prototype. This paper depicts an implementation of H.264/AVC on Cell/B.E. processor. The coding process is partitioned into four modules according to the steps of the coding process, and the individual modules are assigned to run on a dedicated Synergistic Processing Element (SPE). In Nanda's implementation, the data file is stored in system memory and the SPE brings in the data needed for processing, then returns the results back to system memory. Because the predictive structure of MVC and H.264/AVC are different, we changed the task-dispatching algorithm in our MVC application, assigned the whole frame decoding process to run on the SPEs and selected adaptive rather than dedicated (static) for the task-assignment.

3 Parallel MVC algorithm design

3.1 The parallelism of multi-view video coding

The multi-view video uses multi-camera to shoot the same scene from different viewpoints, which contains large amounts of inter-view statistical dependencies. Current ongoing MPEG MVC standard combines temporal and inter-view prediction to improve coding efficiencies. Take 3-view video as an example and assume a group of pictures (GOP) in a 3 by 8 frame matrix. There are no inter-dependency between GOPs as shown in Figure 2. Frames are not only predicted from temporally neighboring images but also from corresponding images in adjacent views. The combination of temporal and inter-view prediction introduces computing complexity and needs parallel processing for real-time codec. The frames are categorized into three types: I-, P- and B-frames[3]. Similar to H.264/AVC, when decoding, I-frame only needs its own content, P-frame requires a frame in addition to itself, and B-frame requires 2–4 frames in addition to itself. If frame X refers to frame Y, such that frame X cannot start to decode until the frame Y has been decoded, we say X depends on Y. The dependency of the frames restricts the order of decoding. The arrows in Figure 2 show the dependency among the frames. There is an arrow from Y to X if X depends on Y [7,17]. After analyzing the dependency of the frames in Figure 2, we design the schedule of decoding a 3-view video GOP as Figure 3. Figure 3 shows that some frames can be decoded in parallel. For example, frame 00 and frame 80 can be decoded simultaneously.

In general, the MVC system has eight views[17]. Therefore 8-view video coding is typical, and substantially more complicated than the 3-view video coding. The essential principle of both 3-view and 8-view is the same. Figure 4 shows the prediction structure of 8-view video GOP in MVC[17]. The

meaning of arrows and I-, P- and B-frames are the same as those in Figure 2. We can also get the schedule of coding an 8-view video GOP as shown in Figure 5. For an 8-view video GOP, it is able to have 14 frames decoded in parallel at most. When more GOPs are decoded in pipeline, more frames can be decoded simultaneously.

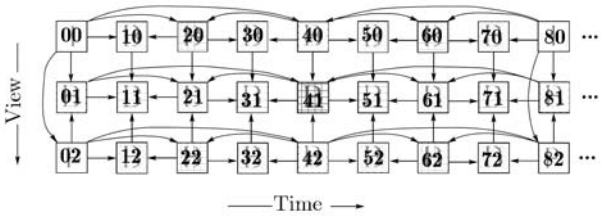

Figure 2 The 3-view video coding prediction structure.

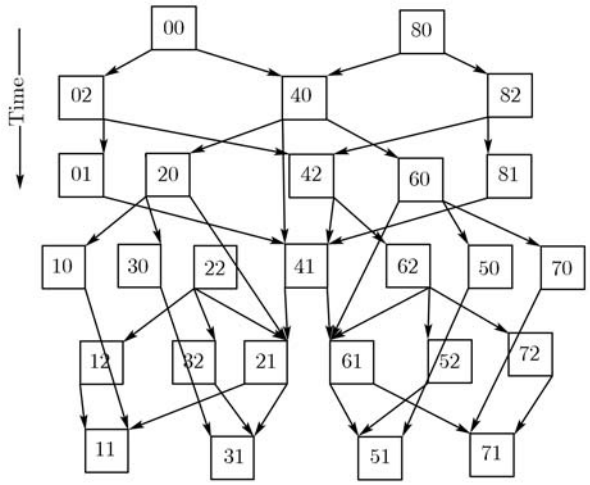

Figure 3 The schedule of coding a 3-view video GOP.

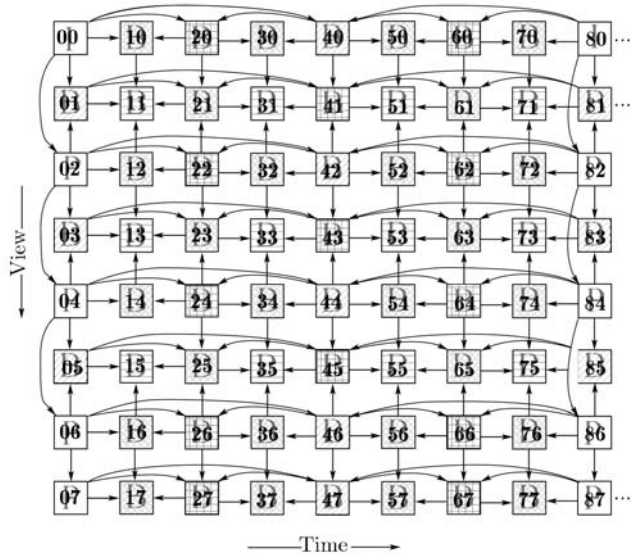

Figure 4 Inter-view-temporal prediction structure of an 8-view video GOP[17].

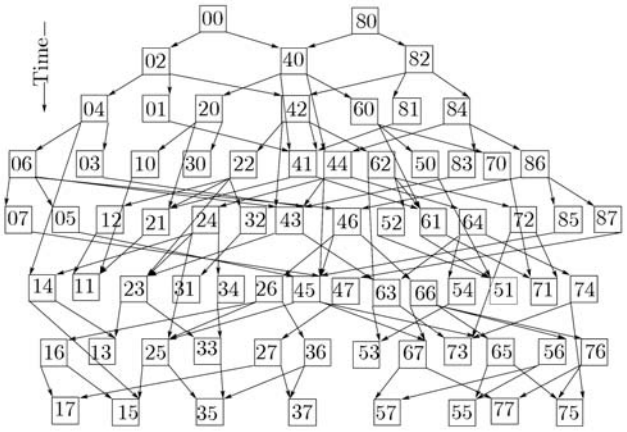

Figure 5 The schedule of decoding an 8-view video GOP.

3.2 Data parallel approach

For parallel program models, there are two approaches: task parallel and data parallel. In task parallel, each core runs a dedicated module and the results are sent from one core to the other. In data parallel, each core runs the whole decoder but on a different sequence of data. Symmetrical multi-core processor favors the latter, while traditional streaming architectures usually selects the task parallel method.

In MVC, as with H.264/AVC, processing can be segmented into the following modules: motion estimation, motion compensation, DCT/IDCT, Q/IQ, entropy coding and loop filter. For task parallel, each core performs one of the modules; however, two types of imbalances can occur due to different workloads of the modules. First, since each module has a different workload, the module with the most computing becomes the bottleneck. Second, with three types of frames (I, P and B) data flow is not static. Although tuning task arrangement can improve workload balance, it would significantly increase design complexity. As for data parallel, workload balance is ensured with tuning size of the data processing unit, and parallel task arrangement of MVC is easily dispatched. Considering the advantages and disadvantages, we select the data parallel approach.

3.3 Design choice of parallel granularity

To determine our choice of design, we verified the issues with concurrency degree and executive cores on multi-core processors and the impact of data

processing units. Some papers suggested to reduce design complexity and minimize time delay using parallel algorithm on multi-core processors. The concurrency degrees should be more than the executive cores in number. In terms of data processing unit in the selection of design, the four types: GOP, frame, slice and macroblock (MB) are important. GOPs decode in parallel, since there is no interdependency. However, GOPs are unfortunately too large to be parallel granularity in real-time application. For example, in 8-view video coding, a GOP consists of 64 (8×8) frames, which is improbable to decode a GOP on a core in real-time. According to MVC standard, the size of frame is static, which is ideal for steady workload, whereas the size of slice is not. Although the size of MB is static but too small, it causes an increase in design complexity. Figure 5 shows the schedule of decoding a GOP using frame as parallel granularity. After data analysis for GOP, frame, slice and MB, we select frame as parallel granularity.

4 Adaptive data-driven parallel algorithm

4.1 Basic concepts of data-driven processing

Compared with other parallel processing methods, the basic concept of data-driven processing is natural. The schedule of decoding a GOP (Figures 3 and 5) is also a graph of data dependencies among nodes, illustrating the basic idea of data-driven processing. Each node represents an elementary frame decoding function and data dependency among the nodes is represented by directed arrows connecting the nodes. In the data-driven architecture, processing unit operates only when the input data is presented and output buffer has enough space to store the results. Basically, each unit is "driven" by data. Operations of each processing unit do not have such a strict time limit as pipeline architecture, which allows appropriate time slots to each operation.

In order to work automatically, a data-driven unit should have two features in common. Firstly, both input and output data buffers are of FIFO (First In First Out buffer) type. Buffers provide signals of state to indicate whether it has enough data to process or enough space to store results. Secondly, each data-driven executive unit controls an Finite State Machine (FSM) which checks the states of both input and output buffer before enabling an operation. When the data in input data buffer is ready and the output buffer has enough space to store the result, the operation is issued[13].

4.2 Adaptive data-driven parallel MVC

As section 3 highlighted, our parallel MVC algorithm uses data parallel program model and makes frame parallel granularity. We use a core as control center and several cores as executive units. Control center is in charge of dispatching frame decoding tasks to all the executive units, and executive units decode each frame respectively. Figure 6 gives the input and output buffer of executive unit which provides state signals: In_Ref_buffer_Stat, In_Cur_buffer_Stat and Out_buffer_Stat. In_Ref_buffer_Stat, In_Cur_buffer_Stat and Out_buffer_Stat express the state of reference frame input buffer, current frame input buffer and output buffer respectively. The decoding process starts only when driven conditions (Reference frame and current frame data are prepared) are met. Figure 7 and 8 show the FSM of control center and executive units.

In the FSM of control center (Figure 7), after the start, the state machine jumps from the state of Start into the state of Init. To initialize each frame, make rX equal to the number of reference frames of X. When initial process is finished, the state machine jumps into the state of Idle, and keeps checking rX and signals from every executive unit. If there is a frame X whose $rX = 0$, the state machine jumps into the state of Dispatch. Frame X is dispatched to an executive unit, and it goes back to the state of Idle. When control center receives a signal from executive unit which means that executive unit finishes decoding a frame, the state machine jumps from state of Idle into the state of Decrease rX. After frame X decoded, decrease rY by 1 for all frames Y that depend on X.

In the FSM of executive unit (Figure 8), if input buffer is underflow, the state machine jumps into the state of Idle0 and keeps checking In_Ref_buffer_Stat and In_Cur_buffer_Stat. If ref-

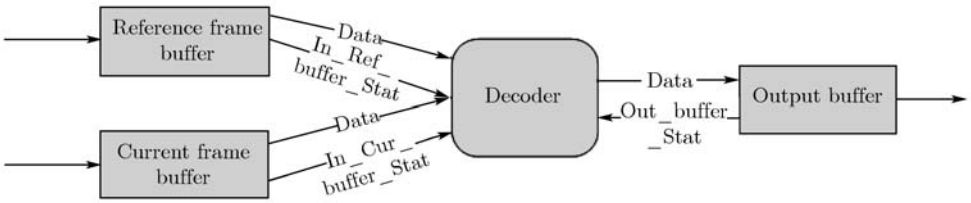

Figure 6 The input and output of decoder on executive unit.

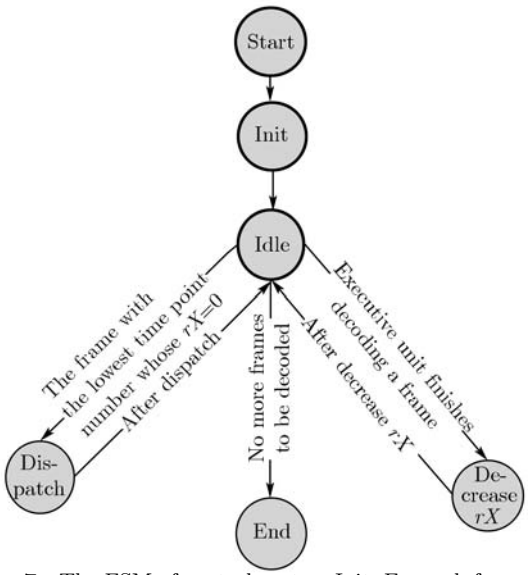

Figure 7 The FSM of control center. Init, For each frame, make rX equal to the number of reference frames of X; Dispatch,dispatch frame X to an executive unit; Decrease rX, after frame X decoded, decreasing rY by 1 for all frames Y that depend on X.

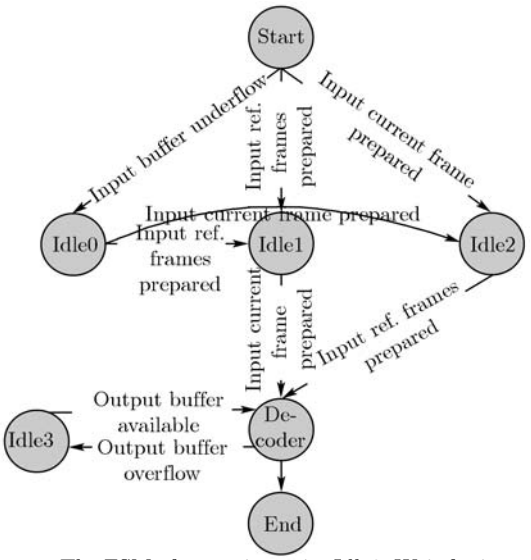

Figure 8 The FSM of executive unit. Idle0, Wait for input data prepared; Idle1, wait for input current frame prepared; Idle2, wait for input reference frame prepared; Idle3, wait for output buffer available; Decoder, decode each frame.

erence frame data is prepared, it jumps into the state of Idle1. If current frame data is prepared, it jumps into the state of Idle2. If both reference and current frames data are prepared, it begins to decode. If output buffer is overflow, it jumps into the state of Idle3, and keeps checking Out_buffer_Stat until the corresponding buffer is available.

5 Implementation of MVC for multi-core processor

5.1 Target multi-core processor architecture

Our adaptive data-driven parallel MVC algorithm explores the feasibility of mapping MVC algorithm onto a multi-core processor based on the architecture template described by Figure 9. This architecture consists of a center processor element (CPE), SPEs, and communication infrastructures. CPE is the control center, controls data flow and dispatches task order. The SPEs are executive units and in charge of basic tasks. All the data and code are transmitted by the BUS which has a wide bandwidth for quick data transfer.

For the implementation of the parallel MVC algorithm, IBM Cell/B.E. processor assumes the instantiation of the architecture template. Cell/B.E. was released by STI in August 2005[18]. The architecture of Cell/B.E. processor shown in Figure 10 has nine cores. One core, the power processor element (PPE), consists of a 64-bit power processing unit (PPU) with L1 and L2 caches. The other eight cores are SPEs, each of which consists of a synergistic processing unit (SPU) designed for high performance computation on 128-bit vectors with a 256 KB local store (LS). The PPE is control center, the SPEs are executive units and an SPU is capable of dual issue. The SPE is smaller than a standard CPU core and in its intended usage does the "real" work in a data parallel or streaming manner, while the PPE core executes synchroniza-

tion tasks and non-parallizable code. The LS is for both code and data used by its SPU; the SPU loads and stores only according to the contents of the LS. Four 16-byte data rings support multiple transfers per ring simultaneously, 96 bytes/cycle peak bandwidth, and over 100 outstanding requests for BUS. Direct memory access (DMA) is used to data transfer between memory and LS, two SPEs, or the PPE and an SPE. Besides DMA, there exist interfaces to off-chip (e.g. MIC, BIC, etc.), as Figure 10 shows. Independently exchanging data or information between two SPUs is not allowed and PPE has to control it[19].

5.2 Adaptive data-driven parallel MVC on Cell/B.E. blade

As section 4.2 represented, MVC is categorized into two types of tasks: one is task dispatching which runs on control center; and the other is frame decoding which runs on each executive unit. For Cell/B.E. processor, control center corresponds to PPE and executive unit corresponds to SPE. The schedule of MVC on Cell/B.E. processor is shown in Figure 11.

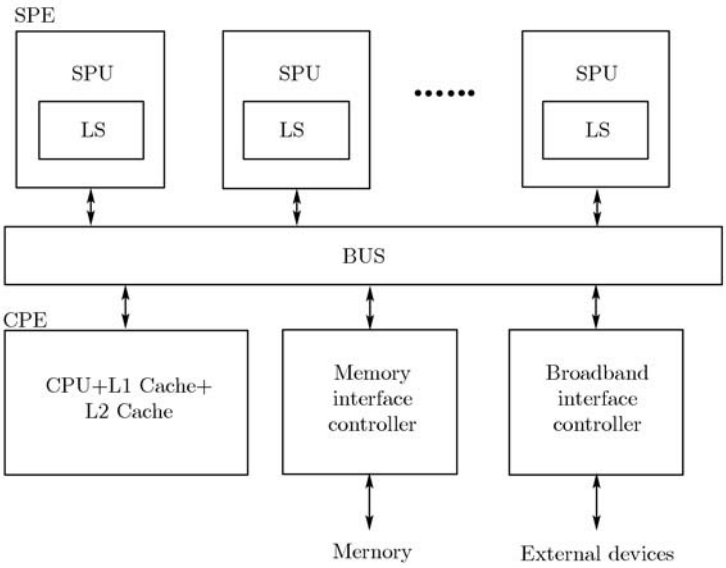

Figure 9 Target multi-core processor architecture.

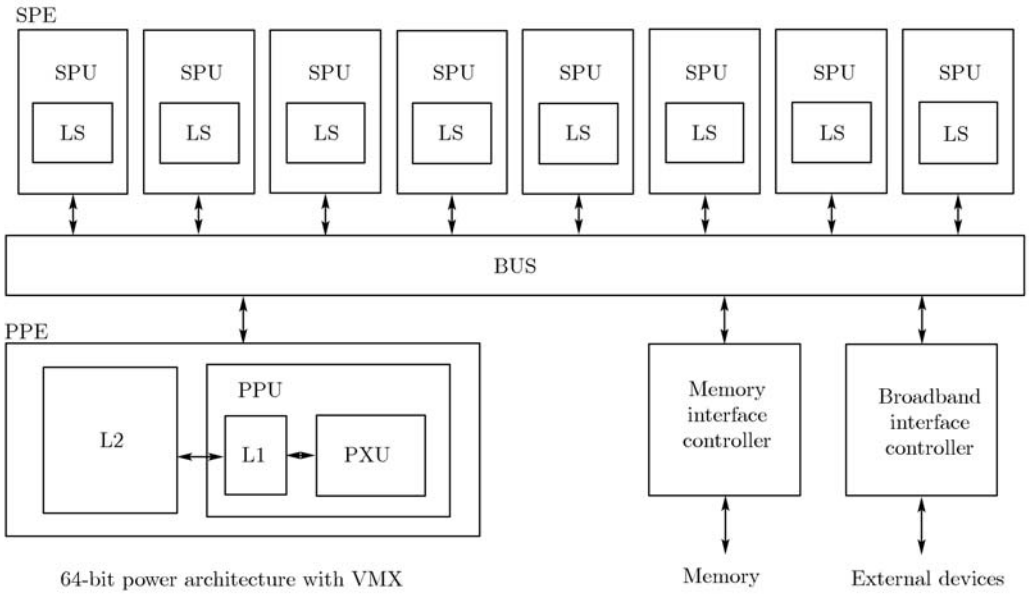

Figure 10 The architecture of Cell/B.E. processor.

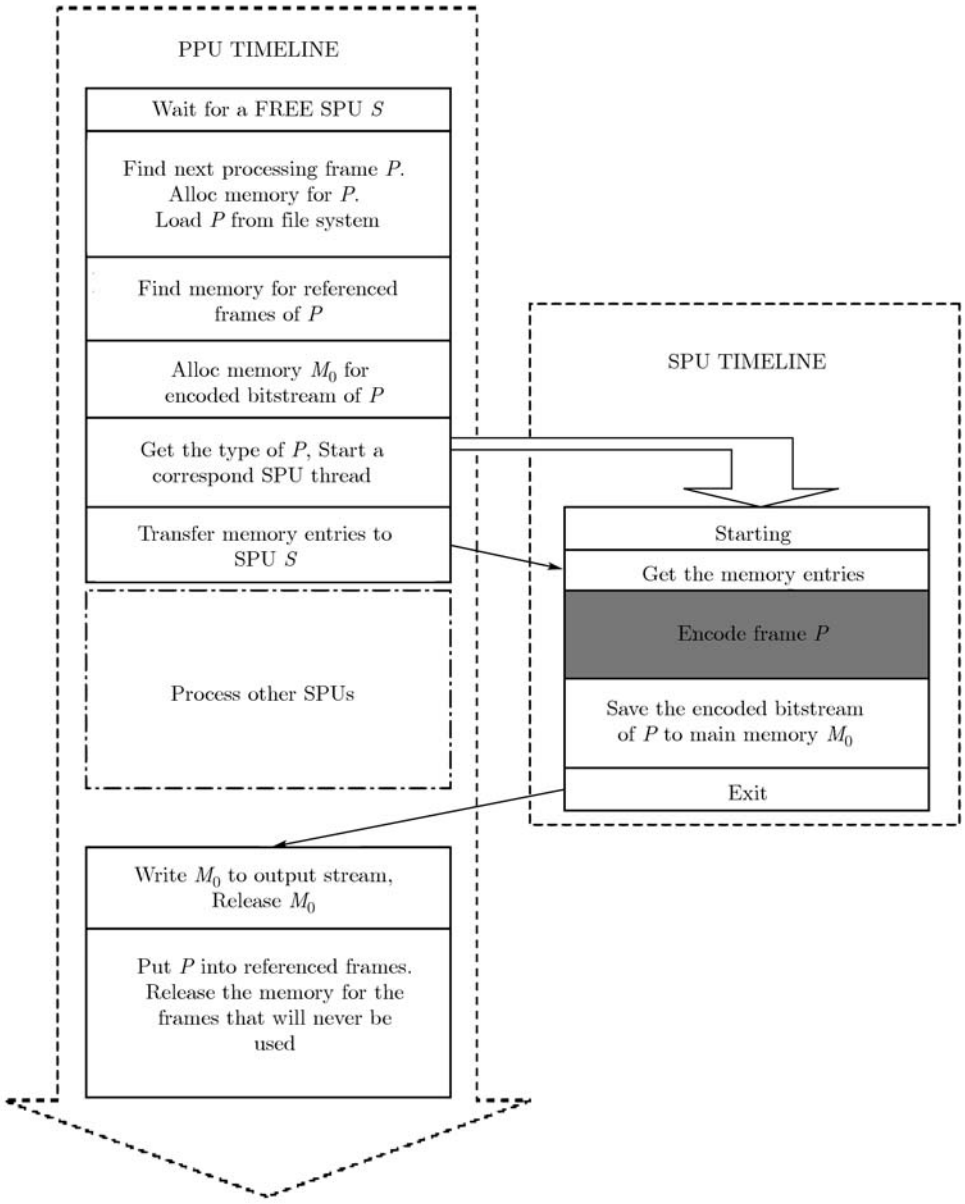

Figure 11 Schedule of MVC on Cell/B.E. processor.

The Init function is called when the program starts to run. When an SPU $SPUx$ is free, we call OnSPUFree($SPUx$) to allocate frame X to $SPUx$. When an SPU finishes its work of decoding a frame X, we call OnFinishEncoding($SPUx$, X) to change all the number of reference frames of frame Y that depend on frame X. The three functions determine how we manage the SPUs. The real program that considers more situations is much more complicated. The required frames are calculated by examining the mathematic properties of the frame number and stored in a fixed table because each 8×8 frames (a GOP) form the same structure.

5.3 Key technologies of implementation

5.3.1 Local store management based frame type. The size of LS for each SPU is only 256 K. It is impossible to put all the code of I-, P- and B-frames into an LS. However, LS allows us to store the code and data for decoding one type of frames, hence we design a code for each of the types. A frame

consists of MBs which have 16×16 pixels. Make MB a process unit to avoid influence of different resolution. A frame is decoded one MB by one MB in loop. When an MB is decoded, other MBs are used for reference. According to the type of frame, we create a proper size buffer for the current MB and its reference MBs. The data transfer process and the decoding process are separated.

The process is as follows:

• Calculate the entries for the decoding MB and its reference MBs.

• Use DMA to transfer the data of the MBs from main memory to LS of the SPU.

• Decode the MB.

• Use DMA to transfer the decoded MB back to the main memory for reference in the future.

5.3.2 Utilizing code locality. As section 5.3.1 indicated, because of the size limitation of LS, we respectively design I-, P- and B-frames decoding process. We import each kind of decoding process into SPE LS once. When an SPE X is idle and a frame A is ready to decode, the program detects what type of code is in the SPE X LS already. If it is the same as the type of ready frame A, it does not need to change the code in LS. This kind of detection utilizes code locality.

5.3.3 SIMD instruction optimization. The SPEs support an SIMD-RISC instruction set[20]. When supported by conventional RISC processors, separate register files are for integer, floating point, and SIMD data types. The SPEs support only a single 128-entry, 128-bit unified register file to store all types of data[21]. We can input four integers, eight shorts or sixteen chars in a vector. It is possible to do the same operation on different data simultaneously. We revise DCT/IDCT model of MVC by SIMD instruction, and the speed of the model upgrades to 3 times more than the original.

6 Performance evaluation

Table 1 shows our experimental environment. To study the scalability of our parallel algorithm, we decode and record each SPE work states and work time as the system runs on from one SPE to sixteen SPEs. Figure 12 shows the utilization rate of the sixteen SPEs. The utilization rate is the ratio of working time to total time for each SPE. The average SPE utilization rate is the average of utilization rate for all the SPEs. Speed, the average SPE utilization rate and speedup are shown in Figures 13–15.

The average utilization rate of SPEs is almost

Table 1 Experimental environment

Item	Description
Processor	Cell/B.E., 3.2 GHz ×2
Blade	QS20
PPE	32(D)/32(I) KB L1, 512 KB L2 cache.
SPE	256K local store. ×8
EIB	16B data rings, 96B/cycle peak
OS	Linux 2.6.20-CBE #1
SDK	Version 2.1
Memory	1GB
Test sequence	KDDI videos[22]
Resolution	320×240
Ref. software	T264

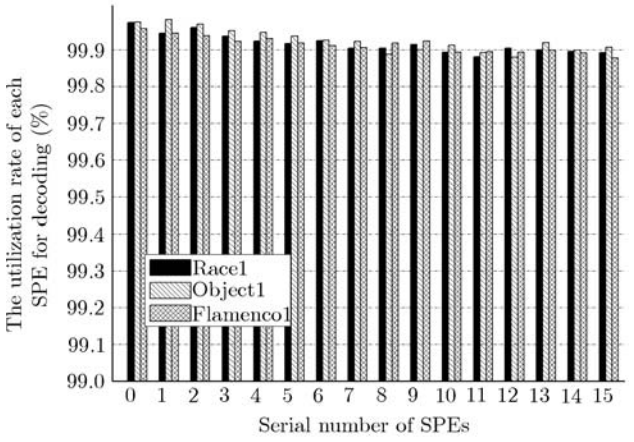

Figure 12 The utilization rate of each SPE (on sixteen SPEs, decoding).

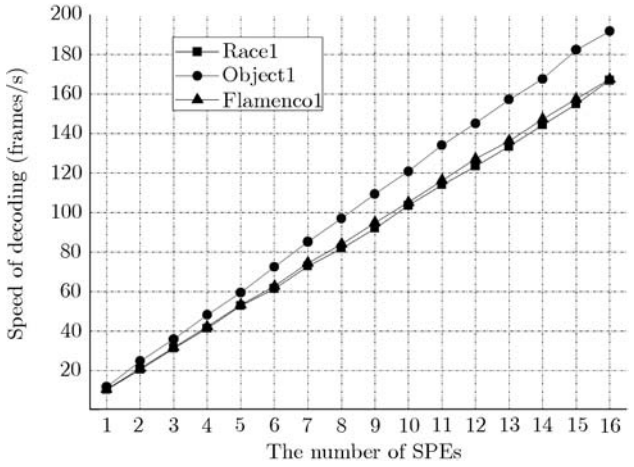

Figure 13 The speed of decoding using one SPE to sixteen SPEs.

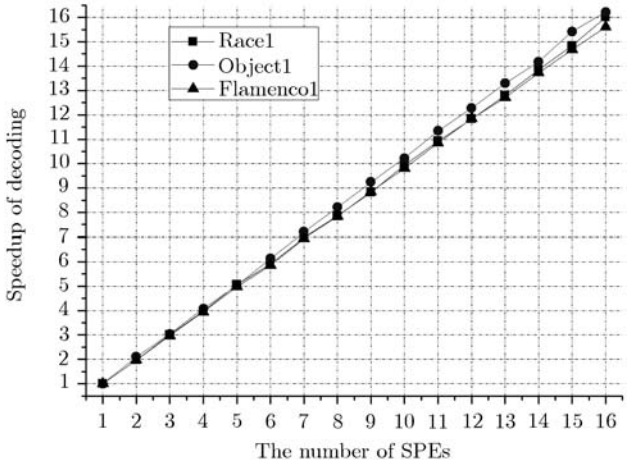

Figure 14 The speedup of using one SPE to sixteen SPEs (decoding).

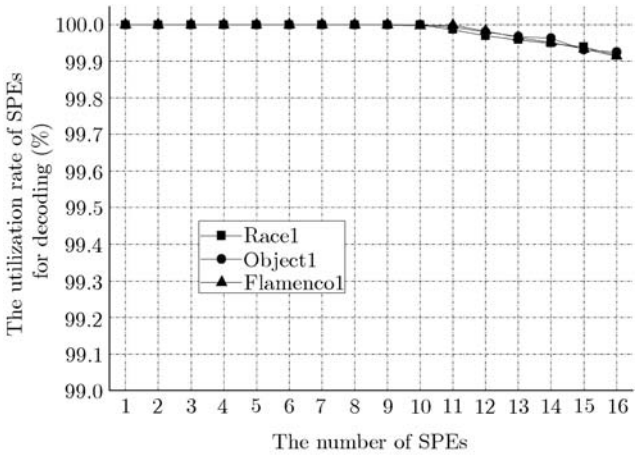

Figure 15 The average utilization rate of used SPEs (decoding).

Figure 16 Decoding result pictures of 8-view video coding.

100%. The speedup is around sixteen times on sixteen SPEs, which demonstrates that our parallel algorithm is scalable. The statistical results highlight that our MVC system takes full advantages of all 16 SPEs. There is no dependency between PPE and SPEs, and therefore the task-dispatching algorithm on PPE is not the bottleneck.

7 Conclusion and future work

The results show that MVV decoder has good parallelism and is integrated ideally with multi-core processors. Our solution largely meets the large computing requirements of MVV decoder and is scalable, as speedup of sixteen SPUs nearly enlarges sixteen times. Although it is much faster than reference software JMVM, the speed is not enough to decode a 8-view video in real-time ($30 \times 8 = 240$ frames/s)(see Figure 16). Our maximum performance for decoding is 176 frames/s at a resolution of 320×240. The frame size is small, which is ideal for web viewing but inadequate in the world of HD. Our parallel algorithm is scalable with increasing resolution. The performance of encoding is much more complicated than decoding.

Our future work will focus on the increase of codec speed. In order to guarantee the scalability of our parallel algorithm and to provide more concurrency degrees, hybrid method of using both data parallel and task parallel approach is necessary. We are also to review Cell Broadband Engine^{TM} Architecture (CBEA), including DMA optimization. DMA optimization adopts double buffer or tri-buffer to manage LS. We can improve dispatching algorithm according to the prediction structure in order to reduce data transmission. SIMD optimization is also one of the future works. We have detailed the core and instruction level parallelism for MVC in this paper. In the future, we will begin research work in high level parallelism. Parallel computing for multi-processor level parallelism inside a blade works naturally, however, it causes new problems for blade level parallelism as with communication and data transmission between two blades. We take advantages of the 16-core SPE resource, but the blade has eighteen cores and twenty threads. We will exploit this in the future. How to porting MVC onto more kinds of multi-core processor to find the difference between different multi-core system, and to compare MVC on Cell/B.E. processor to MVC on general PC are also our interests.

The authors thank Peter Hofstee from IBM in Austin, Texas; CDL, IBM China; and Peng Li and Songliu Guo from Tsinghua University for assistances. Shifei Jin, Yanjie Li and Xiaojun Feng from Tsinghua University contributed much to this work.

1 Smolic A, Kauff P. Interactive 3-D video representation and coding technologies. Proceedings of the IEEE, 2005, 93: 98–110
2 Dodgson N A, Wiseman N E, Lang S R, et al. Autostereoscopic 3D Displays. IEEE Comput, 2005, 38(8): 31–36
3 Tanimoto M. FTV (free viewpoint television) creating ray-based image engineering. In: ICIP, 2005
4 Wilburn B, Joshi N, Vaish V, et al. High performance imaging using large camera arrays. ACM Trans Graph, 2005, 24(3): 765–776
5 ISO/IEC JTC1/SC29/WG11, Updated Call for Proposals on Multi-view Video Coding, MPEG document N7567, Nice, France, 2005
6 Vetro A, Wang Y K, Pandit P, et al. Text of ISO/IEC 14496-10:2008/FDAM 1 Multiview Video Coding, MPEG document N9978, Hannover, Germany, 2008
7 Vetro A, Pandit P, Kimata H, et al. Study Text of ISO/IEC 14496-10:200X/PDAM 1 Multiview Video Coding, MPEG document N9445, Shenzhen, China. 2007
8 Martinian E, Behrens A, Xin J, et al. Extensions of H.264/AVC for multi-view video compression. In: ICIP, 2006
9 Shen K, Delp E J. A parallel implementation of an MPEG1 encoder: Faster than real-time! In: SPIE CDVCAT, 1995
10 Yung H C, Leung K K. Spatial and temporal data parallelization of the H.261 video coding algorithm. IEEE Trans Circuits Sys Video Tech, 2001, 11(1): 91–104
11 Erik B T, Egbert G J. Mapping of MPEG-4 decoding on a flexible architecture platform. In: SPIE Media Processor, 2002, 4674: 1–13
12 Erik B T, Egbert G J, Gelderblom R H. Mapping of H.264 decoding on a multiprocessor architecture. In: IVCP, 2003
13 Wang H, Mao X, Yu L. A novel HDTV decoder and decentralized control scheme. IEEE Trans Consumer Electr, 2001, 47(4): 723–728
14 Yang Y, Jiang G, Yu M, et al. Parallel process of hyper-space-based multiview video compression. In: ICIP, 2006
15 Pang Y, Sun L F, Guo S L, et al. Spatial and temporal data parallelization of multi-view video encoding algorithm. In: MMSP, 2007
16 Nanda A K, Moulic J R, Hanson R E, et al. Cell/B.E. blades: Building blocks for scalable, real-time, interactive, and digital media servers. IBM J Res & Dev, 2007, 51(5): 573–582
17 Vetro A, Pandit P, Kimata H, et al. JVT-V209, Joint Draft 2.0 on Multi-view Video Coding. In: MPEG 22nd Meeting, 2007
18 Kahle J A, Day M N, Hofstee H P, et al. Introduction to the cell multiprocessor. IBM Sys J, 2005, 49(4): 589–604
19 Flachs B, Asano S, Dhong S H, et al. A streaming processing unit for a CELL processor. In: ISSCC, 2005
20 MGschwind M, Hofstee H P, Flachs B, et al. A novel SIMD architecture for the cell heterogeneous chip multiprocessor. In: Hot Chips 17 Conference, 2005
21 Hofstee H P. Introduction to the cell broadband engine. 2005. www-01.ibm.com
22 Kawada R. KDDI multiview video sequences for MPEG 3DAV use, MPEG document M10533, Munich, Germany, 2004