Video Processing and Retrieval on Cell Processor Architecture

Junqing Yu，Haitao Wei

Computer College of Science & Technology,
Huazhong University of Science & Technology, Wuhan, 430074, China
yjqing@hust.edu.cn, whtaohust@163.com

Abstract. A multi-level parallel partition schema and three mapping model – Service, Streaming and OpenMP model – are proposed to map video processing and retrieval (VPR) workloads to Cell processor. The Cell, with 9 cores in one chip, provides an efficient high performance computation platform to speedup VPR and to boost its performance dramatically. We present a task and data parallel partition plan to partition and distribute intensive computation workloads of VPR to exploit the parallelism of a sequential program through the different processing core on Cell. Service and Streaming mapping model are proposed to map the VPR to Cell, using service allocation-calling mode and stream-data pipeline mode respectively. To facilitate the VPR programming on Cell, OpenMP programming model is loaded to Cell. Some effective mapping strategies are also presented to conduct the thread creating and data handling between the different processors and reduce the overhead of system performance. The experimental results show that such parallel partition schema and mapping model can be effective to speed up VPR processing on Cell multi-core architecture.

1 Introduction

Nowadays, with the rapid development of multimedia and network technologies, the multimedia data have been produced massively. Digital video rapidly becomes an important source for information, education and entertainment. It is much easier to access video content via network than ever. Digital video information has grown exponentially in recent years. How to help people conveniently and fast retrieve the interested video content is becoming more and more important. To this aim, content-based video processing and retrieval (VPR) is extremely needed. VPR has been an international research focus in multimedia domain. However, the intrinsic features of video, such as rich and non-structured data, intensive computing, and complex algorism, have prohibited VPR's further development. VPR programming is challenging and real-time processing and parallelization are expected.

Multi-core chip architecture is becoming the mainstream solution for next generation microprocessor chips. Cell processor, developed by IBM/Sony/Toshiba, provides an efficient high performance computation platform, with 9 cores one PowerPC Processor Element (PPE) and eight Synergistic Processor Elements (SPE).

In order to meet the intensive computation and the real-time processing, we port VPR framework to Cell multi-core processor, with each core performs a part of VPR processing.

A multi-level parallel partition schema and three mapping model (Service, Streaming and OpenMP model) are proposed to map VPR workload to the 9 cores of Cell. In the schema of parallel partition, intensive computation workloads are partitioned and distributed by PPE to the SPE cores, which perform a part of the workloads and send the computation results to PPE. In the Service Model, PPE assigns different services to different SPEs, and the PPE's main process calls upon the appropriate SPEs when a particular service is needed. And Streaming model is implemented to organize the PPE and SPE processors to act as stream-data processors in a serial pipeline to accelerate the data processing. OpenMP is an industrial standard and high level programming model for shared memory multi-processor system. OpenMP programming model for VPR on Cell facilitates the programming compared to Service and Streaming models. The method and performance overhead of mapping OpenMP to Cell are also presented.

The rest of this paper is organized as follows. In the next section, Cell architecture is briefly introduced. In the section 3, we try to use Service and Streaming model to map VPR workloads to Cell. As an experiment in VPR, the Block Matching Algorithm is implemented on Cell simulator to test Cell's performance for VPR by using multi-level parallel partition schema. It is proposed to map OpenMP programming model to Cell in section 4. Section 5 concludes this paper.

2 Cell Architecture Overview

The first-generation Cell processor consists of a 64-bit multithreaded Power PowerPC processor element (PPE) and eight synergistic processor elements (SPE) connected by an internal, high-bandwidths Element Interconnect Bus (EIB) [1]. Fig.1 (a) shows the composing of the Cell elements and the interconnect bus between them and external memory and I/O.

The PPE contains two levels of globally coherent cache and also supports Vector/SIMD Multimedia Extension (VMX) instruction set to accelerate multimedia and vector application [3]. The major powerful computation ability derives from the eight SPEs. Fig.1 (b) shows the composition of SPE. Each of these identical elements contains non-coherent local store for instructions and data. Almost all instructions provided by SPE operate on 128 bits of data which is representing either 2 64-bit double floats or long integers, 4 32-bit single float or integers, 8 16-bit shorts, or 16 8-bit bytes. SPE's instruction and data is transferred through DMA, supported by DMA commands of Memory Flow Controller (MFC), from PPE's main storage to SPE's local storage. Data is transferred by DMA to the SPE local memory in units of 128 bytes, enabling up to 16 concurrent DMA requests of up to 16KB of data.

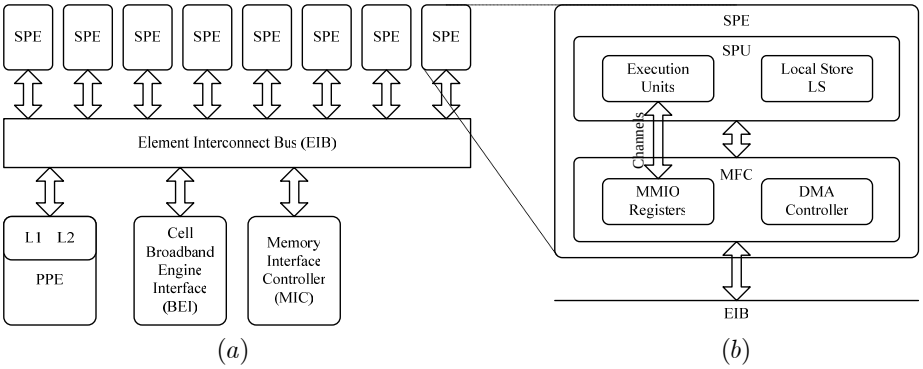

Fig. 1. Cell Architecture Overview (a) and Synergistic Processing Element (SPE) (b) [2].

3 Video Processing and Retrieval on Cell

As digital video data becomes more pervasive, mining information from video data becomes increasingly important, but the huge computation prohibits the wide use in practice. Accelerating the video processing application by exploiting multi-level parallelism on Cell would be a promising way to boost the performance and provide more functionality for VPR.

3.1 Video Processing and Retrieval Framework

The advances in the technology of video capture and storage have contributed to the startling growth of digital video content. As video content generation and dissemination grows explosively, how to help users efficiently search, browse and manage video contents becomes increasingly important, such as video surveillance, highlight event detection, and the affective mining.

Video processing and retrieval aims to help users to search, browse and manage video contents easily. The key components include low-level feature extraction such as visual and audio features, shot boundary detection, scene segmentation, video structural summary, high level semantic concept detection, annotation, indexing and content-based video retrieval [5]. We propose an efficient framework for indexing, browsing, abstracting and retrieval of sports video. As illustrated in Fig. 3, the framework consists of five stages: low-level feature extraction, mid-level feature extraction, high-level feature extraction, representation, browsing and retrieval as well. Mid-level and high-level feature extraction use specific-domain knowledge. The input video is decoded by corresponding video decoder into visual and audio streams. Then the visual low-level features are extracted from visual stream, e.g., color histogram, mean value and standard deviation of each frame's pixel intensities etc. The audio low-level features are extracted from audio stream, e.g., audio power and audio spectrum envelope etc. After the low-level features are ready, more complex high-level and semantic features, e.g., face recognition in video and spoken content in audio, are needed to be extracted from the low-level features. Finally, according to the

match of feature and semantic of input video and library, we can retrieve the video clips we want, e.g., highlight in sports video.

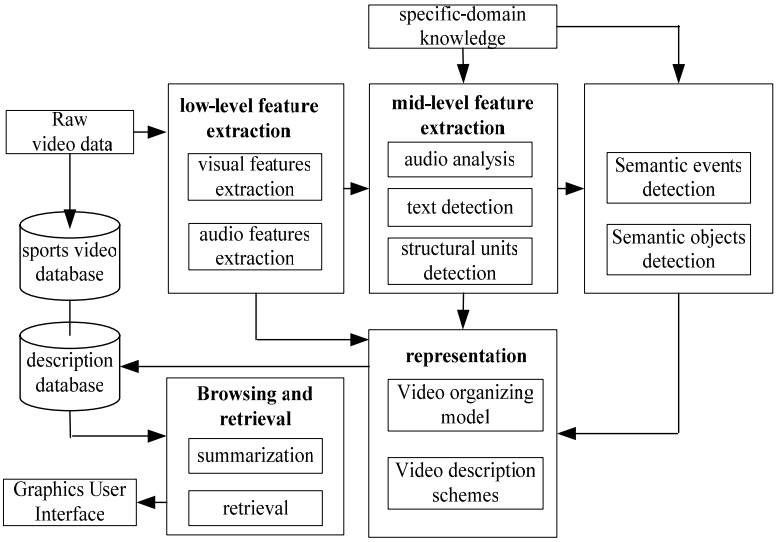

Fig. 3. Overview of video processing and retrieval framework

3.2 Service and Streaming Programming Model for VPR on Cell

The huge data and underlying complex algorithm of VPR require excessive computation, e.g., one second video sequence coded at CIF resolution (352x288) with 25 frames per second and 3 bytes per pixel, yields 7.25 MB data to process. Cell provides an efficient high performance computation platform to speed up the video processing through mining the underlying task and data parallelism.

The key characteristics of VPR application are variety of tasks and intensity of computation. Therefore, the Service Model and the Streaming Model are proposed to be the best candidates.

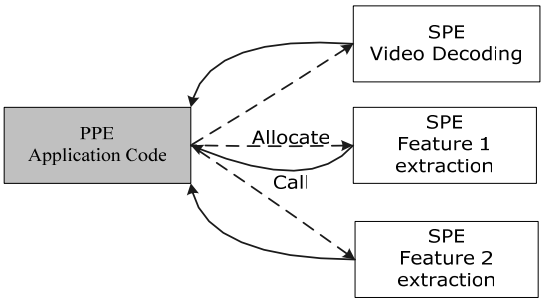

Fig. 4. Service Model of Video Processing

In the Service Model, the processor assigns different services to different coprocessors, and the processor's main process calls upon the appropriate coprocessor when a particular service is needed. Fig. 4 shows the Service Model in video processing. It is well known that feature extraction is a basic and important step in video processing. Here, each feature extraction is regarded as a service, which is assigned to one or more coprocessors and the processor's main process call upon these coprocessors' services needed. Fixed static allocation of coprocessors' services is avoided. These services are virtualized and managed on a demand-initiated basis.

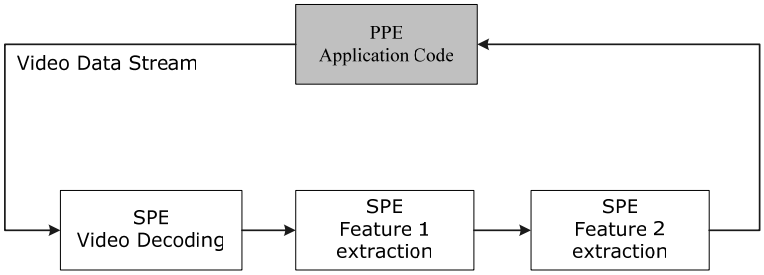

Fig. 5. Stream Model of Video Processing

In the Streaming Model shown in Fig. 5, the main processor acts as a stream controller, and the coprocessors act as stream-data processors in either a serial or parallel pipeline. In video processing, each procedure has inherent computing stage which is mapped to one coprocessor. Here, the first coprocessor decodes the input raw video data stream and outputs decoded video data stream. The second coprocessor takes the decoded video data stream produced from the stage1 as input and extracts the feature 1 in stage 2. Finally, the third coprocessor extracts feature 2 and output the result data to the main processor. Under the control of main processor, all the coprocessors work together in a pipeline to speed up the computation performance.

3.3 Block Matching Algorithm on Cell Using Parallel Partition

3.3.1 Block Matching Algorithm

Block matching algorithm is an important class of motion compensation algorithms for interframe predictive coding, which is used to eliminate the large amount of temporal and spatial redundancy that exists in video sequences and helps in compressing them. These algorithms estimate the amount of motion on a block by block basis, i.e. for each block in the current frame i, a block from the previous/after frame j is found, that is said to match this block based on a certain criterion.

Here, the block matching criteria is the sum of absolute differences and the algorithm is Full Search or the Exhaustive Search. As Fig.6 shown, each block within a given search window in frame *j* is compared to the current search block *g(x,y)* in frame *i*. At each search step, the sum of absolute differences between the search block and the given block f(x,y) is evaluated using the following criterion.

$$AE = \sum_{i=1}^{K}\sum_{j=1}^{K}\left| f(i,j) - g(i-d_x, j-d_y)\right|$$

Where, K means the size of the block, d_x and d_y refers to the motion vector, which means amount of motion from the current block to the aim block. The best match is obtained if AE achieves the minimum.

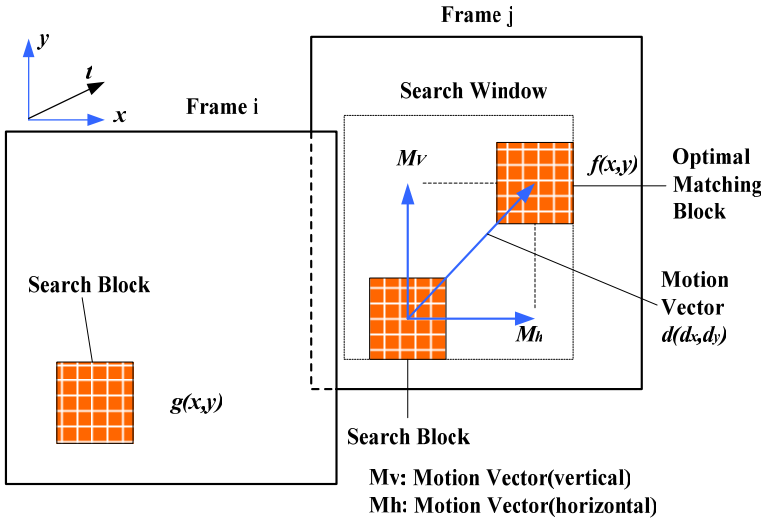

Fig.6. Illustration of Block Matching Algorithm

3.3.2 Parallel Scheme of Block Matching Algorithm

The sequential algorithm uses four nested loops to evaluate the minimum *AE*. As the program following, array *Block[8][8]* and *Frame[M][N]* are allocated for the current search block and search window respectively, and variable *diff* preserves the absolute difference at each searching step. The minimum absolute difference and corresponding motion vector are stored in variable *min*, *x* and *y* respectively.

```
…
For(i=0;i<M-7;i++)
   for(j=0;j<N-7;j++){
         for(l=0;k<8;l++)
            for(m=0;l<8;m++)
           diff+=abs(Block[k][l]- Frame[k+i][l+j]);
   if(diff<min)
   {min=diff;x=i;y=j;}
   }
…
```

In order to contrast the performance improvement and communication overhead generated by different level parallelism, a step by step parallel scheme is adopted on Cell. In the first step, we SIMDize the code for PPE to attain the data level parallelism, and then port the PPE code to multiple SPEs to achieve the task and data

level parallelism. Finally, the overhead of communication is measured by setting the number of thread to one.

Scheme 1: SIMDize the code for execution on PPE

The main idea in this scheme is to utilize the VMX instruction set to accelerate the data computation. As the following code shows, VMX instructions-*vec_perm*, *vec_abs*, *vec_add*, *and vec_sums*-are used to execute the vector absolute evaluation by allocating the array *block[8][8]* and *frame[M][N]* into one dimension array of vector. By this way, computation complexity is reduced by about 8 times and the number of loops is decreased to 3.

```
…
for(i=0;i<M-7;i++)//row of the frame
for(j=0;j<N-7;j++) { //colum of the frame
   rowsum_v=(vector signed short)(0);
   for(k=0;k<8;k++){
         addr=&frame[i+k][j];
       row_v=vec_perm(vec_ld(0,addr),vec_ld(16,addr),ve
       c_lvsl(0,addr));
        rowabs_v=vec_abs(vec_sub(block_v[k],row_v));
        rowsum_v=vec_add(rowsum_v,rowabs_v);
      }

sumall.sumall_v=vec_sums(vec_sum4s(rowsum_v,zero_v),zero_
v);
     if(sumall.index[3]<min)
     {min=sumall.index[3];placeX=i;placeY=j;}
}
…
```

Scheme 2: Parallelize Code for Execution across Multiple SPEs

In this scheme, PPE initializes to partition and distribute the task to each SPE. It divided the search window into some sub-windows, and then maps the matching procedure in each sub-window to each SPE to evaluate the local minimum of AE and correlative local motion vector. At last, PPE reduces the global AE and correlative global motion vector comparing all the local AE. SPEs receive the sub-window and current block from PPE, search the best local match to get the local minimum of AE and correlative motion vector, and sends the results to PPE to evaluate the global AE and motion vector.

The different instruction set between PPE and SPE makes the program running on Cell partitioned into two sections – PPE code for PPE and SPE code for SPE. The following *PPE Code* shows that the subroutine *spe_create_thread* creates a group threads running on SPE, the address of the data can be transmitted through the parameter `ctx`. After the threads are created, subroutine *spe_wait* is called to wait all the SPE threads to finish. At last, PPE gets the global result from local result returned from the SPE. In the *SPE Code*, each SPE gains the data from PPE through DMA transform, and then evaluates local minimum *AE* and the coordinate motion vector with SIMD instruction in SPE. After matching, each SPE sends the result to PPE by DMA.

PPE Code:

```
…
//correspond to ‘fork’ in OpenMP
For(i=0,offset=0;i<SPE_THREADS;i++,offset+=P){
// Create SPE thread of execution passing the context
as a parameter.
spe_ids[i]  =  spe_create_thread(0,  &match_multi_spu,
&ctx[i], NULL, -1, 0);   }
…
// correspond to ‘join’ in OpenMP
 for(i=0;i<SPE_THREADS;i++){
(void)spe_wait(spe_ids[i],&status,0);  }
//find the globe minimum, coordinate x,y
for (i=0; i<SPE_THREADS;i++){
 if(diff[i]<globemin)
   {globemin=diff[i];x=posx[i];y=posy[i];}
}
```

SPE Code:

```
…
//gain the  data from ppe
spu_mfcdma32(…, MFC_GET_CMD);
…
//compute the absolute and the local minimum and
//the coordinate x,y with SIMD instruction in spu.
…
uschRowAbs_v[k]=spu_absd(model_v[k],uschRow_v[k]);
if(diff[k]<localmin)
   {localmin=diff[i];x=posx[k];y=posy[k];}
…
spu_mfcdma32(…, MFC_PET_CMD); //return the resulte
```

Scheme 3: Set 1 thread to Measure the Communication Overhead

Comparing to the program only running on PPE without communication with SPE, the program is ported to one SPE to measure the communication overhead setting the MARO *SPE_THREADS* of the SPE thread to one.

3.3.3 Experimental Results

The experiment of block matching algorithm was realized on IBM Cell simulator v2.0, using the language of PPE SPE C/C++ language extension (Intrinsics), with the SDK v2.0. [10]

As Fig. 7 shows, parallelization boosts the performance dramatically with the resolution accreting. Here, approximate 5x speedup is achieved referring to the resolution of 2000×1500 contrasting to 2x speedup referring to resolution of 800 ×600. The performance achieves a peak with SPE increasing, but over parallelization deteriorates the performance. Referring to the resolution of 800×600, 1600×1200, and 2000×1500, the number of SPE to achieve the best performance is 4, 6, 7, respectively. It is illustrated that with the SPE increasing, the computation ability is

strengthened, but the communication overhead is expanding as well. Only when the balance point reaches, the best performance is achieved.

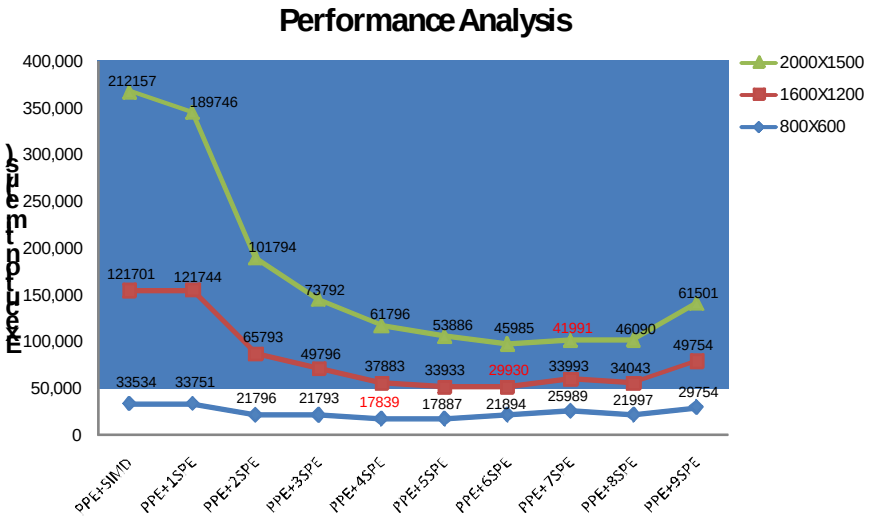

Fig. 7. Performance Analysis on Block Matching Algorithm

4 Loading OpenMP Programming to Cell

Cell provides an efficient computation resource for application of intensive computation with the data and the task level parallelism. But, programming on Cell for VPR is not an easy job, e.g., the block matching algorithm in section 3, programmer has to be acquainted with the Cell architecture, PPE and SPE instruction sets, DMA transfer, and register files, etc. Although suit for VPR, Service and Streaming models are low level and inconvenient. All of these constrain the improvement of facility and performance for programming. OpenMP is an industrial standard for shared memory parallel programming agreed on by a consortium of software and hardware vendors [7]. It consists of a collection of compiler directives, library routines, and environment variables that can be easily used for VPR programming on Cell.

4.1 Loading Strategy

The parallel computation needs to be split among PPE and SPE processors. The parallelism is represented in OpenMP using “parallel” directive. PPE and SPE processors can be viewed as a group of “threads”. Through data allocation PPE tells each SPE processor to execute specific regions in parallel. PPE executes the region as

the master thread of the team. At the end of a parallel region, PPE waits for all other SPE to finish and collect the required data from each SPE [11].

4.2 Exploitation Locality

The shared data is allocated in the main storage of PPE, and can be accessed exclusively through the atomic operation on Cell. Operation on shared data, each SPE needs to translate the local address to the effective address.

The solution is to keep a copy of the shared data in the local storage of each SPE instead of the direct accessing to main storage when reading/writing the shared data. Then, if the local copy of the shared data is updated, the changed data will be written through DMA transfers back to the main storage after the end of parallel section.

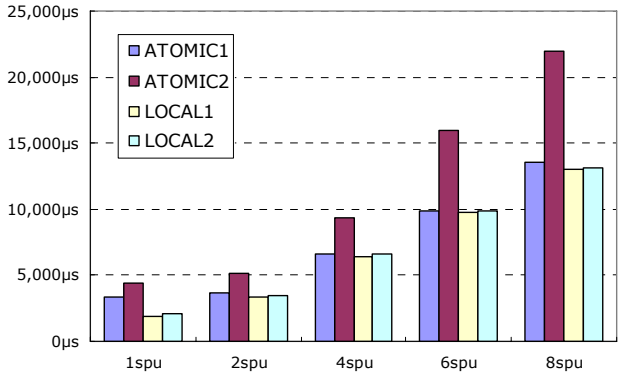

Fig. 8. Overhead of the Shared Data Accessing.

Fig. 8 illustrates the overhead due to the exclusive accessing to the shared data adopting two different mechanisms: atomic accessing and local accessing with mutex variable and DMA transfer. Loop iteration, in spite of a simple operation, may be used in most programs. In this case, a shared integer array with 128 entries is allocated in the main storage, and each SPE executes the iterations of addition operation on each entry of the shared array. By setting 10 iterations, the overhead of the atomic (ATOMIC1) and local (LOCAL1) mechanism for shared data accessing are measured almost the same. But with the number of iteration increasing to 100, the overhead of the atomic (ATOMIC2) mechanism increases sharply, especially the num of the SPU reaches 6 and 8. Contrast to the atomic mechanism, the overhead of local (LOCAL2) mechanism is almost the same as the 10 iteration one (LOCAL1), except the additional accessing time for more iteration. The local mechanism for shared data accessing experiment reduces 40% overhead for 8 SPEs. It is shown that with the accessing frequency increasing, a larger overhead reduction can be achieved.

4.3 Parallel Synchronization

The parallel synchronization is implemented with the mechanisms of memory tag polling and event trigger on Cell. The first method is to allocate a tag memory for each SPE with values initialized to “0” and each SPE polls the tag to wait the task assigned by PPE. PPE wakes up each SPE by setting the tag to “1”. If detecting the tag set to “1”, then each SPE runs the computation immediately, and after the computation is completed, the tag is set to “2” to notice PPE. At the same time, the PPE polls the tag until all of them are set to “2” by the SPEs, and then the PPE cleans the tag to “1” to resume the SPE execution. The second method is to utilize the mailbox and signal mechanism for parallel synchronization. The PPE deploys the task to each SPE by sending signal, once receiving the “start” signal each SPE does the work from PPE. And at the time of barrier, each SPE will send “completion” event to PPE by mailbox and then sleep. Once receiving all the events from SPEs, PPE will resume the SPEs to continue to work.

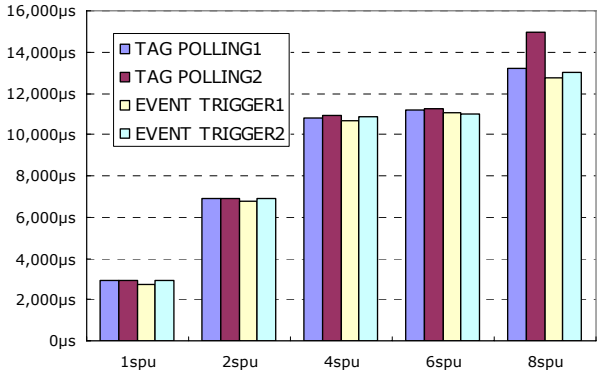

Fig. 9. Overhead of the Parallel Synchronization

As Fig 9 illustrates, the overhead of the parallel synchronization is measured with the mechanism of the memory tag polling and event trigger. The PPE assigns the parallel work of array summing (like section 4.2) to each SPE, and the synchronization operation occurs at the begging and the end of parallel work. With the calculation increase from 10 to 100 iterations, the tag polling mechanism (from TAG POLLING1 to TAG POLLING2) and event trigger mechanism shows the similar execution overhead, except the abrupt overhead increase with 8 SPE. But, it is believed that the event trigger mechanism will be a better candidate for parallel synchronization, besides the less little reduction than tag polling, it can also save the storage and execution time because of its un-blocking execution.

5 Conclusion

This paper presents a parallel partition schema and three mapping model to load video processing and retrieval (VPR) workloads to Cell multi-core processor. By means of

partition and distributing the intensive computation workload to PPE and SPEs, the VPR processing is accelerated notably and a remarkable speedup is achieved. The proposed Service and Streaming models which use the mode of allocating-calling and data-streaming pipeline, are suitable and efficient, but somewhat of inconvenient. In addition, OpenMP programming model is presented on Cell to facilitate mapping VPR to Cell. Some effective strategies for data distribution and processor synchronization are also proposed by comparing the different approaches of data distribution and processor synchronization. And the overhead of the strategies for data sharing and processor synchronization are tested.

The proposed partition schema and mapping model are reasonable for VPR on Cell, but there is a lot of work left to do, for example, the effectiveness of VPR mapping and the improvement of data distribution and synchronization mechanism between the PPE and the SPEs.

References

1. D. Pham, S. Asano, M. Bolliger, M. N. Day, H. P. Hofstee, C. Johns, J. Kahle, A. Kameyama, J. Keaty, Y. Masubuchi, M. Riley, D. Shippy, D. Stasiak, M. Suzuoki, M. Wang, J. Warnock, S. Weitzel, D. Wendel, T. Yamazaki, and K. Yazawa: The Design and Implementation of a First-Generation CELL Processor. In: Proc. IEEE International Solid-State Circuits Conference (2005) 184-185
2. IBM Systems and Technology Group: Cell Broadband Engine Programming Tutorial v2.0. IBM Corporation (2006) 17-19
3. IBM Microelectronics Division: PowerPC Microprocessor Family: AltiVec Technology Programming Environments Manual. IBM Corporation (2004) 282-308
4. Alexandre E. Eichenberger, Kathryn O'Brien, Kevin O'Brien, Peng Wu, Tong Chen, Peter H. Oden, Daniel A. Prener, Janice C. Shepherd, Byoungro So, Zehra Sura, Amy Wang, Tao Zhang, Peng Zhao, and Michael Gschwind: Optimizing Compiler for a CELL Processor. In: Proc. 14th International Conference on Parallel Architectures and Compilation Techniques (2005)
5. Eric Li, Wenlong Li, Tao Wang, Nan Di, Carole Dulong, Yimin Zhang: Towards the Parallelization of Shot Detection – a Typical Video Mining Application Study. In: Proc. 35th International Conference on Parallel Processing (2006)
6. Feng Liu and Vipin Chaudhary: Extending OpenMP for Heterogeneous Chip Multiprocessors. In: Proc. 32nd International Conference on Parallel Processing (2003).
7. OpenMP Architecture Review Board: OpenMP C and C++ Application Program Interface Version 2.0. http://www.openmp.org (2002)
8. Christian Brunschen, Mats Brorsson: OdinMP/CCp – A portable implementation of OpenMP for C. MSc thesis, Lund Universtiy, Sweden (1999)
9. Vassilios V. Dimakopoulos, Elias Leontiadis, George Tzoumas: A portable C compiler for OpenMP V.2.0. In: Proc. European Workshop on OpenMP. Aachen, Germany (2003)
10. IBM Corporation: IBM Full-System Simulator User's Guide version 2.0. IBM Corporation (2006)
11. Haitao Wei, Junqing Yu: Mapping OpenMP to Cell: A Effective Compiler Framework for Heterogeneous Multi-Core Chip. In: Proc. International Workshop on OpenMP 2007 (to appear)