VIDEO ANALYSIS AND COMPRESSION ON THE STI CELL BROADBAND ENGINE PROCESSOR

Lurng-Kuo Liu, Sreeni Kesavarapu, Jonathan Connell, Ashish Jagmohan, Lark-hoon Leem, Brent Paulovicks, Vadim Sheinin, Lijung Tang, Hangu Yeo

IBM T.J. Watson Research Center, Yorktown Heights, NY 10598, USA

ABSTRACT

With increased concern for physical security, video surveillance is becoming an important business area. Similar camera-based system can also be used in such diverse applications as retail-store shopper motion analysis and casino behavioral policy monitoring. There are two aspects of video surveillance that require significant computing power: image analysis for detecting objects, and video compression for digital storage. The new STI CELL Broadband Engine (CBE) processor is an appealing platform for such applications because it incorporates 8 separate high-speed processing cores with an aggregate performance of 256Gflops. Moreover, this chip is the heart of the new Sony Playstation 3 and can be expected to be relatively inexpensive due to the high volume of production. In this paper we show how object detection and compression can be implemented on the CBE, discuss the difficulties encountered in porting the code, and provide performance results demonstrating significant speed-up.

1. INTRODUCTION

Video surveillance is a growing business area due to several factors. First, with heightened concerns about terrorism many facilities such as fuel storage depots and commercial airports have been looking for ways to provide better physical security. Second, the associated hardware costs for cameras, interconnectivity, and storage have been steadily dropping, making the installation of video surveillance systems more affordable. There are two aspects of a typical video surveillance system that require significant computing power: image analysis for detecting objects, and video compression for digital storage. At the front end the system detects moving objects in the scene through a process known as "background subtraction" (BGS) and tracks these objects over time. Various details of the objects themselves as well as their trajectories are available for triggering immediate alerts or for use as database keys to later retrieve particular segments of stored video.

Large scale digital video surveillance systems require the use of efficient video compression algorithms prior to storage. The state-of-the-art H.264 video compression standard [1] provides gains of up to 50% in compression efficiency over previous standards such as MPEG2/MPEG4 [2], which are the algorithms of choice in current video surveillance systems. The H.264 standard derives most of its compression gain from the use of efficient context-adaptive binary arithmetic coding, the use of quarter-pixel

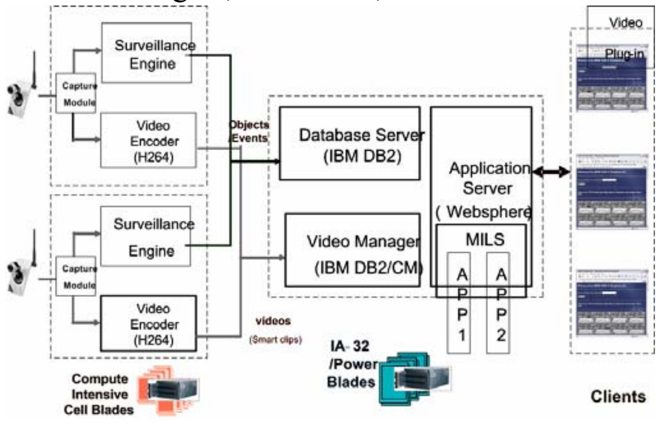

Fig. 1 IBM's S3-MILS video surveillance architecture

accurate motion vectors for motion compensation, and the use of several different macroblock prediction modes and block sizes for encoding macroblocks (i.e. 16 × 16 blocks of luma pixels). The high complexity of the H.264 encoding algorithm and the need for scalability in surveillance systems make the CBE well-suited for use in video encoding.

The rest of the paper describes the details of the algorithms, the system design that leverages the features of the CBE architecture, and some preliminary results on their performance.

2. OVERALL SYSTEM ARCHITECTURE AND CELL BROADBAND ENGINE PLATEFORM

The overall system architecture for our studies is shown in Fig. 1. It is based on IBM's intelligent video surveillance system called S3-MILS [3]. It consists of a front-end subsystem and a back-end subsystem. The front-end subsystem handles video analysis and compression which require significant computing power. The back-end subsystem handles content management and retrieval. Motivated by the need of an efficient and cost-effective way to implement the system and along with the alluring cost-performance ratio of CBE, we have proposed a CBE-based video analysis and compression mechanism aiming to serve as the front-end subsystem of the overall video surveillance system. This involved some innovative software architecture design and algorithm development in order to leverage the features of the CBE.

The CELL Broadband Engine [4] was developed in a joint program between Sony, Toshiba, and IBM. The CBE consists of 8 synergistic processing elements (SPEs) coordinated by a PowerPC-based general processing element (PPE). The main computing engines on CBE for high performance computing are the SPEs.

The SPE [5] delivers performance by executing up to two

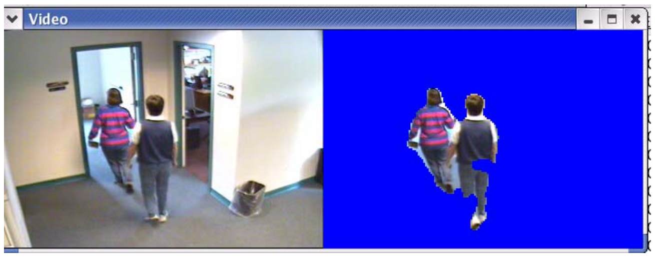

Figure 2 Output of the Background Subtraction (BGS) system.

instructions per cycle. Each of these instructions operates in a single instruction, multiple-data (SIMD) fashion on four 32-bit words in parallel. Instruction throughput is maximized with instructions that eliminate or help predict branches and 128 registers. The SPE offers a high bandwidth interface to a direct memory access (DMA) engine that can transfer 32 GB/sec to and from the 256 KB local memory. Each SPE has its own memory flow controller (MFC), and can initiate up to 16 independent DMA transfers to and from its local store. To achieve maximum throughput, a program can use the DMA engine to schedule data movement in parallel with computation.

The SPE local storage is a limited resource. Only 256 Kbytes is available for program, stack, local data structures and DMA buffers. Many of the optimization techniques require additional pressure on this limited resource. As such, all optimization may not be possible for a given application. Therefore, programmers need to evaluate the feasibility of their choose optimization approach based on the limited local store constraint.

In general, the process of CBE programming consists of three major phases. First, uniprocessor code needs to be partitioned into code to be run on the PPE and SPEs. Second, the SPE code should be vectorized to exploit the strength of the vector engines in the SPEs. Finally tasks should be scheduled optimally to bring the best speedup with the least idle time in the SPEs. Programming models for Cell architecture differ as to how code is partitioned and how SPEs are used. With still limited support from compiler and other tools, programmers may require some creative thinking in their application development.

3. ALGORITHM AND ARCHITECTURE OF BGS ON CELL BROADBAND ENGINE

The background subtraction system finds objects by looking for moving regions against a stationary background. It does this by comparing the current video frame to a stored reference frame representing the “empty” scene. The result of the computation is a binary mask indicating where the moving objects are, as shown in Fig. 2. While conceptually simple, for a robust implementation there are many sophisticated details to consider. First, it is desirable to normalize the input video frames as much as possible. This keeps the system from detecting spurious objects resulting from differences induced by things like camera color variations, swaying of the camera in the wind, or noise introduced by video compression.

Also, straight pixel subtraction is not a very good motion detection scheme. Provisions must be made for finding and eliminating shadow and highlight regions without ignoring subtly shaded regions. For this reason, the BGS subsystem combines

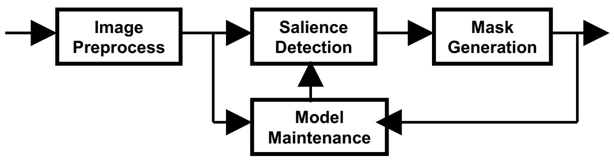

Fig. 3 Background Subtraction (BGS) processing stages.

evidence from color, intensity, texture, and motion variations to generate an overall saliency map. This map is then thresholded to give a raw version of the foreground. A number of morphological operators and connected component analyses are then performed on the raw mask to generate a cleaner segmentation.

A further complication arises in building and maintaining the reference background image. In the case where there are always moving objects in the field of view, this reference image needs to be created incrementally in the relatively stable regions. The reference image also needs to be slowly updated in the case of weather changes or for sun angle progression. Finally, moving objects sometimes stop and persist for a long time at the same place (e.g. a car parks in a lot or a door closes). In this case, to allow proper detection of new moving object in front of these former foreground objects, a mechanism must be provided to absorb such objects into the background model.

3.1. Implementation on Cell Broadband Engine

The BGS system is divided into four separate stages as shown in Fig 3. They are Image Pre-processing, Salience Detection, Mask Generation, and Model Maintenance. The goal of the partitioning was for each module to contain functions with logic coherency; and for them to have similar running times. A straight forward mapping of multiple BGS onto a Cell processor might statically assign ’n’ SPEs to ’m’ BGS, constrained by the 8 available SPEs. However, statically assigning SPEs will prevent the system from dynamically adapting to workload changes.

In order to make the most efficient use of the CBE’s resources and be able to handle multiple video streams with any give number of available SPEs, each SPE is assigned to complete a module and then ready to be reassigned. The application in PPE will inquire for an available SPE resource first. The PPE then update its command control block and pass the command to the SPE. The SPE updates its code module as needed

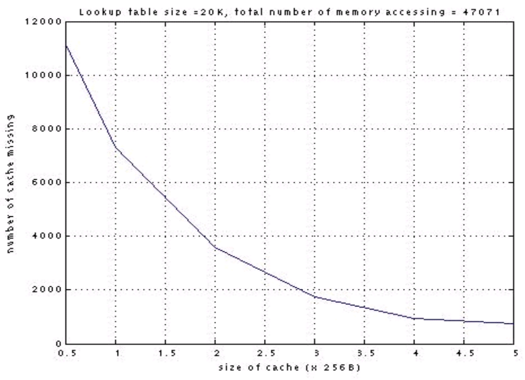

Figure 4 Cache miss v.s. SPE soft cache size.

and then starts the process of the module. The data requires by the module will then be DMAed between main memory and its local store.

As in most of the image processing library, the video analysis functions in BGS need at least one or two video frames as input and generate another one as output, which is impossible to keep in SPE's local store all at once. We thus use a DMA load operation to bring in a small block of data to SPE local store at a time, let the SPE process the data in local store, write the processed data back to PU memory with a DMA store operation (if necessary). The overhead of the DMA operations can generally be hid by using double-buffering scheme. There are still functions need to process a large scale data but in a random manner, where the next element to be accessed is unpredictable or depends on the processing result of the current element. The same double-buffering DMA scheme is then not applicable to this case. To handle this problem, we use a buffer in local store as a soft cache for the data in PU system memory. We then do a DMA request only if there is a local cache miss. Fig 4 shows the performance at different soft cache size. In our case, we empirically determined to choose the cache size at 1KB, which entails a cache miss rate of around 3%. That is, we can eliminate 97% of the DMA requests as compared to doing DMA without this soft cache.

4. LOW-COMPLEXITY H.264 MODE-SELECTION

The present H.264 encoder implementation is aimed at providing efficient compression for bit-rates from 1-3 Mbps (SD video at 30 fps) and variable frame rate and resolution, which is well-suited for typical surveillance applications. Accordingly, the encoder employs a strict subset of the prediction modes provided by the H.264 standard. In particular, the encoder is restricted to the use of 16×16 macroblocks for non intra-coded frames—this significantly lowers computational complexity while having only a small effect on compression performance for the requirements at hand. Further, the encoder contains all main-profile tools with the exception of interlaced coding and weighted prediction.

In our implementation for prediction mode selection, we make use of a learning-theoretic approach [6] to select from among the main prediction classes, and to further select the best prediction mode within the selected class. Both the inter-intra and Intra4-Intra16 decisions are made using supervised binary classification using classifier trees [7]. As detailed in [6], classification is performed as follows: Prior to classification the 16 × 16 current macroblock m is down-sampled to a 4×4 array, denoted m4. Denoting the 4×4 Hadamard transform of m_4 as $T_H m_4$, the following four features, as shown in Equation (1) are used for I4-I16 discrimination: (1) The macroblock high frequency content (2) The macroblock horizontal frequency content, (3) The macroblock vertical frequency content. In addition, for intra-inter classification, the following features were used: (4) The absolute sum of the transform coefficients of the prediction error between the down-sampled current macroblock m_4, and the down-sampled motion compensated predictor p_4.

$$\begin{aligned} f_1 &= \sum_{i=2}^{4} \sum_{j=2}^{4} |T_H m_4(i,j)|, \quad f_2 = \sum_{j=2}^{4} |T_H m_4(0,j)| \\ f_3 &= \sum_{i=2}^{4} |T_H m_4(i,0)|, \quad f_4 = \sum_{i=1}^{4} \sum_{j=1}^{4} |T_H (m_4 - p_4)(i,j) \end{aligned} \qquad (1)$$

The learning-theoretic algorithm is trained offline using a set of standard training sequences, and only a small amount of computation is required, to compute the learned discrimination functions, during encoding. Thus the time complexity of mode selection, which is a significant constituent of the overall time complexity of the reference H.264 encoder, is significantly reduced.

4.1. H.264 Encoding on Cell Broadband Engine

The H.264 encoding process was divided into three basic modules: motion estimation (ME), transform and quantization (PI), and CABAC (AC). Again, the approach we used was to assign an SPE to an encoder to complete a module and then to be reassigned. Each of these functions operates on an entire frame at a time and constitutes the unit of work for an SPE. The SPE is passed a command block which contains the operation to be performed and pointers to the frame buffers on which it is to perform the function. As the SPE completes the function on a frame, it notifies the PPE which reassigns it to the next pending work unit. In this way, any mix of frame rates, frame sizes, and IPB stream structures can be managed automatically by the CBE.

Motion estimation (ME) was, perhaps, the most difficult to port to an SPE due to the large amount of buffering this function requires. In our implementation, the range for motion estimation was limited to a window of up to eight macroblocks in the vertical direction but can search the full width of the frame in the horizontal direction. That is, a window of at most eight rows of macroblocks from the reference frame is DMAed and maintained inside the SPE and one row of macroblocks from the current frame is processed against it. As the ME function proceeds thru the frame, one row of macroblocks is fetched from both the reference and the current frames. Though several search heuristics are used to limit the number of SAD computations, search results are to quarter-pixel resolution. Besides motion estimation, this function also computes the heuristic used to select between intra and inter coding for a macroblock. This, along with the motion vectors and motion compensated prediction is returned to system memory after each row is processed.

As with the ME function, the second function, transform and quantization (PI), processes a frame by row of macroblocks. Each row of macroblocks from the current frame is read into the SPE along with the motion-compensated prediction for that row and the intra/inter decision made by the ME function. Though the intra/inter decision was made by ME, intra prediction needs further refinement to select either 16 × 16 or 4 × 4 mode which is, again, decided by a heuristic. After this, an exhaustive search is made of either the four 16 × 16 possible predictions or the nine 4 × 4 predictions. From these predictions are generated the macroblock coefficients (transform and quantization) and the reconstructed frame (inverse quantization, inverse transform and loop-filter). Once PI has completed the reconstructed frame is available to be used as a reference for the next frame.

The last function is the CABAC encoding (AC) of the motion vectors and coefficients. Despite the serial nature of the CABAC code, single SPE is able to perform this function in approximately the same amount of time as the PPE.

5. SUMMARY OF PERFORMANCE RESULTS

We use the standard mobile_and_calendar test sequence for the compression performance analysis on our presented low-complexity H.264 encoder. Fig. 5 compares the convex hull of the PSNR-Rate performance curve for the three codecs for the given test sequence. As can be seen, the IBM H.264 codec is about 1 dB inferior to the reference H.264 codec, and is about 1 dB superior to the reference MPEG4 codec. This loss in compression efficiency

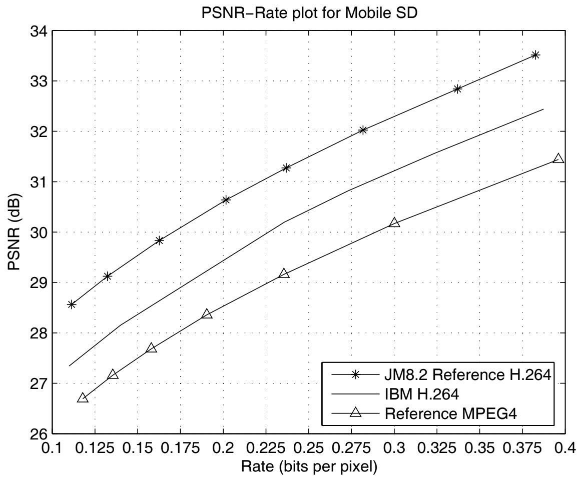

Figure 5 PSNR-Rate plots for standard definition mobile_and_calendar sequence.

vs. the reference H.264 encoder is accompanied by a large gain in computational efficiency—a software implementation of the presented encoder, running on a Windows based Pentium workstation, is about 40 times faster than a corresponding software implementation of the public H.264 reference coder, and is about 10 times faster than the MPEG4 reference coder.

Table 1 Processing times for one SPE on 2.4 GHZ Cell Broadband Engine for Cell encoding functions.

	ME (ms)	PI (ms)	AC (ms)
table tennis (1 Mbps)	11.9	14.9	13.6
table tennis (2 Mbps)	11.4	15.6	17.4
table tennis (3 Mbps)	11.1	17.8	21.5
mobile (1 Mbps)	13.6	15.6	13.0
mobile (2 Mbps)	13.0	16.7	17.9
mobile (3 Mbps)	12.9	16.7	20.9

Table 1 shows the performance of the presented H.264 encoder running on CBE., Virtually all of the encoding is performed by the SPEs with only a small amount of NAL unit processing being done by the PPE. The average time each function (ME,PI,AC) takes for three different video test sequences, each encoded at three different bitrates. These times were measured on a single SPE of a 2.4 GHz CBE. This shows that two SPEs can do SD video encoding at 30 fps. Thus a CBE can encode four SD video sequences at 30 fps. As a rough point of reference, the (non-optimized) software implementation of our same H.264 encoder on a 3.0 GHz Pentium IV processor can be used to encode at most one SD sequence at about 5-6 fps. This is about 20x performance gain from a single CBE.

Table 2 Execution time of BGS partitioned modules

Stage	Prep	Salience	Model	Mask
time (ms)	4.49	1.34	4.86	4.55

Table 2 shows the average execution time of each stage in BGS, which is equivalent to about 65 fps. However, without tools support for overlay at this early stage, dynamic SPE allocation using continuous thread creation/killing introduces additional overheads. In this case, one SPE can handle BGS at 42fps. We expect the performance to be further improved when an overlay support is available. As a rough point of reference, the BGS subsystem runs at about 58 fps on a 2.4GHz Xeon with 2GB memory using MMX/SSE/SSE2 parallelization. This shows that the BGS subsystem running on a 2.4GHz CBE with 512MB RAM achieve a 6-9x overall performance speed-up. It is worth to note from our observation that, comparing the scalar version codes running on SPE, the SIMD features of SPE do provide significant performance improvement. Around 50% functions we tested have between 4x~16x improvement, 25% functions gain a 16x~40x improvement, and the other 25% of the functions are in the 1.5x~4x improvement category.

6. DISCUSSIONS

We present an implementation of an H.264 video encoding algorithm and video analysis for background subtraction on the CBE, wherein each CBE can encode and analyze two channels of video at 30 fps, and a full capacity blade center containing seven blades each carrying two CBEs can handle 28 streams of video. Thus, the proposed architecture can be used to serve as the core of a large-scale surveillance system which would be scalable, and have far smaller cost of ownership than traditional solutions.

Yet the applications porting were not trivial due to limited tools support at this early stage of product development. The difficulties faced were primarily memory issues. First, the SPU did not have as much program memory as we would have liked and there are limited tools support for different programming models, thus requiring manual code partitioning or user management of overlays. Second, many of the optimization techniques require additional pressure on this limited resource. As such, all optimization may not be possible for a given application. For example, algorithms based on tables lookup might need to be modified to use computation instead. Therefore, applications developers require some creative thinking in order to develop an appropriate optimization scheme. In the long term there may be more tools support and even architectural updates that can help ameliorate these difficulties.

7. REFERENCES

[1] T. Wiegand, G.J. Sullivan, G. Bjontegaard, and A. Luthra, "Overview of the h.264/avc video coding standard," IEEE Transactions on Circuits and Systems for Video Technology, vol. 13, no. 7, pp. 560–576, July 2003.

[2] ISO/IEC JTC 1/SC 29/WG 11, "Mpeg-4 video coding standard," 2000.

[3] A. Hampapur et al, "Smart Video Surveillance," IEEE Transactions on Signal Processing, Vol. 22, No. 2, March 2005.

[4] "Broadband Engine," Book IV for DD1.0, Version 1.0, SCEI/Toshiba/IBM, May 25, 2004.

[5] "Synergistic Processing Unit Core," Book IV for DD1.0, SCEI/Toshiba/IBM, May 10, 2004.

[6] A. Jagmohan and K. Ratakonda, "Time-efficient learning theoretic algorithms for h.264 mode selection," in Proc. IEEE Int. Conf. Image Processing, 2004, pp. 749–752.

[7] P. Chou, "Optimal partitioning for classification and regression trees," IEEE Trans. Pat. Anal. Mach. Intel., vol. 13, no. 4, pp. 340–354, 1991.