Implementation of Video Surveillance analytics on the STI Cell Processor

P. Vinay Kumar

Department of Electronics and
Computers Engineering,
Indian Institute of Technology
Roorkee, India 247667

Praveen Kumar

Department of Computer
Science and Engineering,
Sagar Institute of Science and
Technology, Bhopal, India,

Ankush Mittal

Department of Computer
Science and Engineering,
Graphics Era University,
Dehradun, India

ABSTRACT

The Cell Broadband Engine (CBE) processor provides the potential to achieve an impressive level of speed up for multimedia applications. Video Surveillance is a growing multimedia application due to its concern in various areas like commercial security, military applications. In this paper, we present CBE as a cost effective computational solution for the application and demonstrate the real time performance of its parallel execution on the platform. We present a method to implement the algorithm on the CBE, along with introduction to our previous work in implementing on computer cluster discussing various issues related to porting the code on CBE, followed by simulated results demonstrating a 43x speed up over non distributed version of the algorithm and comparison of the results with implementation of same on computer cluster.

General Terms

Video surveillance, high performance computing, cluster computing.

Keywords

Cell processor; Video Surveillance; Parallel Computing; Computer Cluster; Image Processing.

1. INTRODUCTION

Recently, Sony, Toshiba, and IBM (known as STI) have jointly developed the Cell processor that integrates nine processor cores in a single chip. The chip consists of one core of Power PC processor, and eight cores of Synergistic Processor Units (termed accelerator cores in IBM literature), in all employing RISC architectures. The accelerator cores on the heterogeneous multicore on chip processor offer a new degree of parallelism by supporting independent compute and transfer threads within each accelerator core. To get maximum benefit of such a powerful chip, it is essential to exploit the parallelism for the application optimally. This architecture gives a cutting edge solution in terms of optimization of execution of instructions and data handling for the programmers. A good review of implementations of Video processing applications on CBE, done by research teams at IBM and Mercury respectively is provided in [1, 2].

With the recent advancements in video and network technology, Video surveillance is very rapidly growing in the commercial market due to its wide range of applications, such as a homeland security, security guard for important buildings and shopping malls, traffic surveillance in cities and detection of military targets, etc [3, 4]. The aim of automatic video surveillance is to automatically detect the interesting objects in the monitored area, track their motion and automatically take appropriate action like alerting a human supervisor. Video Surveillance application requires deployment of

cameras and sensors in a widespread location, their interconnectivity with a computational system to process the media stream delivered from the site [5]. Along with these advances the surveillance application is becoming algorithmically complex and data intensive. One of the major challenges posed is to make the processing real time, in which the processing rate of media must be at a higher rate compared to the rate at which media is delivered.

There have been some efforts laid by researchers of various institutions to provide an efficient platform for surveillance applications. For instance, Intel has developed various libraries for multimedia related applications, [6] provides detailed case study of Intel platforms for video surveillance applications, [7] provides description of computational platform support developed by Sun Microsystems, which includes high-resolution digital video and integrated surveillance data management infrastructure system for video surveillance. The architectural description of Video Servers designed Moxa Inc. is given in [8], which has many inbuilt dedicated units for video processing that help accelerating the application. Despite of the developments in the platforms, they have been expensive and thereby limiting to commercial use. The current challenge posed is to develop a cost effective hardware solution to enable general purpose use of both application and platform.

In this paper, we focus on cell processor enabling the basic model of video processing algorithm, addressing various challenges like extracting parallelism in the algorithm, meeting the application with various architectural characteristics of the platform like memory constraints, communicational delays, availing the heterogeneity of the platform for the application and discussing issues regarding the implementation of algorithm with the overall aim of achieving an improvement in the performance of the algorithm to enable a real time implementation. Also, a review of our previous implementation on computer cluster is presented, and comparison of speed ups on both solutions demonstrating the potential of CBE.

The outline of the paper is as follows: Section 2 briefly explains the hardware architecture of Cell processor. Section 3 explains the basic video surveillance algorithm. Section 4 discusses our previous work on implementing the application on computer cluster. Section 5 presents a review of various approaches of parallelizing video processing applications on Cell processor. Section 6 discusses our approaches in parallelizing Video Surveillance application on STI Cell Broadband engine processor. Section 7 presents our experimental results and finally we conclude and give direction for future work in Section 8.

2. THE CELL BROADBAND ENGINE (CBE)

2.1 Background

The Cell Broadband Engine (CBE) was developed in a joint program between Sony, Toshiba, and IBM. The CBE processor was initially intended for application in media-rich consumer-electronics devices such as game consoles and high-definition televisions, but the design is also supportive to enable fundamental advances in processor performance. These advances are expected to support a broad range of applications in both commercial and scientific fields [9].

The most distinguishing feature of CBE processor is that, although all processor elements share memory, their function is specialized into two types: the Power Processor Element (PPE) and the Synergistic Processor Element (SPE). The CBE processor has one PPE and eight SPE's.

2.2 Hardware Architecture

A review of hardware design and description of CBE is provided in [10-13]. We present here a brief overview of CBE that help an application accelerate, and exploiting the artifacts built in the platform. Figure 2 describes the hardware environment of Cell Broadband Engine which broadly consists of five parts as described as follows

(i) PowerPC Processor Element (PPE) :

The PPE is the main processor. It contains a 64-bit PowerPC Architecture reduced instruction set computer (RISC) core with a traditional virtual memory subsystem. It runs an operating system, manages system resources, and is intended primarily for control processing, including the allocation and management of SPE threads. It can run legacy PowerPC Architecture software and performs well executing system-control code. It supports both the PowerPC instruction set and the Vector/SIMD Multimedia Extension instruction set. The PPE consists of two main units Power Processor Unit (PPU) and PowerPC Processor Storage Subsystem (PPSS). The PPU performs instruction execution, it has level 1 (L1) instruction cache, data cache, and six execution units. The PPSS handles memory requests from PPU and external requests to the PPE from SPEs or I/O devices. It has a unified level 2 (L2) instruction and data cache.

(ii) Synergistic Processor Elements (SPEs) :

The SPEs are SIMD processors optimized for data-rich operations allocated to them by the PPE. Each of these eight identical elements contains a RISC core, 256-KB, software-controlled local store for instructions and data, and a large (128-bit, 128-entry) unified register file. The SPEs support a special SIMD instruction set, and they rely on asynchronous DMA transfers to move data and instructions between main storage (the effective-address space that includes main memory) and their local stores. SPE DMA transfers access main storage using PowerPC effective addresses. As on the PPE, address translation is governed by PowerPC Architecture segment and page tables. The SPEs are not intended to run an operating system. Memory-mapped mailboxes or atomic MFC synchronization commands can be used for synchronization and mutual exclusion.

Fig. 1: The Hardware architecture of Cell Broadband Engine which is based on heterogeneous multiprocessor architecture.

(iii) Element Interconnect Bus (EIB) :

The PPE and SPEs communicate coherently with each other and with main storage and I/O through the EIB. The EIB is a 4-ring structure (two clockwise and two counterclockwise) for data, and a tree structure for commands. The EIB's internal bandwidth is 96 bytes per cycle, and it can support more than 100 outstanding DMA memory requests between main storage and the SPEs.

A good review explaining the potential of the STI Cell Broadband Engine, and its programming models helping the applications to accelerate faster could be viewed in [13, 14].

3. TYPICAL VIDEO SURVEILLANCE ALGORITHM

Place Tables/Figures/Images in text as close to the reference as possible (see Figure 1). It may extend across both columns to a maximum width of 17.78 cm (7").

The general framework of an automatic video surveillance system is shown in Figure1. Video cameras are connected to a video processing unit to extract high-level information identified with alert situation from the incoming video frames. This processing unit could be connected throughput a network to a control and visualization center that manages, for example, alerts. The main video processing stages include background modeling, object segmentation, object identification and object tracking. The algorithm aims to segment out regions corresponding to moving objects such as vehicles and humans from the rest of an image and track their motions over time for behavior analysis. Background modeling assumes that the video scene is composed of a relatively static model of the background, which becomes partially occluded by objects that enter the scene. These objects are assumed to differ significantly from those of the background model. Since the background is dynamic due to lighting changes and movement of static objects, continuous updating of the model is required. Here we implement a mean and variance background model [15], where we compute the mean and variance over the last N frames which serve the model for the next N frames.

Algorithm Video Surveillance

1. For I=1 to M
2. Read the current video frame/image (I)
3. If (I % N==0)
 - Background=UpdateBackground(past N frames)
4. ROI=Segmentation (current frame, Background);
5. Objects=Matching routine (ROI, Objects Info);
6. Objects Info=Track routine (Objects)
7. End

We call N the refresh rate. Then in motion segmentation, we subtract the current frame from the background frame and threshold to get the regions of interest (ROI). Subsequently, these regions are further processed to remove noise and matched with previously tracked regions to identify the objects (old and new ones). Finally the objects are tracked and the current information is passed on for identifying the objects in the next frame. A review on the implementation of Video Surveillance could be viewed in [16]. The algorithm can be summarized in following steps:

Fig.2: General framework of an automated video surveillance system

For computation purpose, each frame is a matrix of size $p \times q$ (say 240×320). The first sub procedure Update Background is concerned with finding mean and variance over N frames (we have used $N=5$ in our implementation). For this we need to read n frames, sort each pixel of frame with respect to other frame and apply an exponential series procedure to find mean and variance. Thus, the time complexity goes up to $O(2pqn)$ for every call. Also the second sub procedure involves many matrix operations like convolution, multiplication etc. Apart from these there are many image processing procedures involved to reduce noise, fill gaps etc. Now the video has data rate ranging from 20-30 frames/second. Update Background forms the most data intensive operation than other components of the algorithm. Its implementation consists of operating on a number of images of order nearly equal to (240×320) , updating some three dimensional data structures and using a number of loops. Thus the computational requirements of a real time video surveillance system can be satisfied by suitably parallelizing the algorithm.

4. IMPLEMENTATION OF VIDEO SURVEILLANCE ON COMPUTER CLUSTER

In our previous work, we had proposed and implemented a model of video surveillance on computer cluster [17]. Cluster, a coordinated resource sharing concept, would aptly suit for such implementation where we could exploit idle desktops present in the campus.

We assert that this architecture comfortably adapts to Video Surveillance and applications similar, where jobs arriving at a cluster are sets of tasks which have some dependency between them. The main advantage we achieve in this architecture is less waiting time for a node, when it is waiting for a result of a task executing on other node and which is needed for execution of current task on the former node. In this architecture, we propose a method in which we arrange set of different clusters in a hierarchy which form giant cluster architecture. We elect a node as a leader of each cluster and in turn a leader elected from those elected leaders of cluster. Each leader is responsible to schedule tasks onto its local nodes, collect the results from them and then return the result to leader above its level. The root node is responsible to schedule jobs to nodes (leader) under it. The advantage of having root node and leader nodes is to have some hierarchical control over the clusters. Apart from this, we also achieve giant cluster architecture by connecting just the leader nodes of clusters through the root node.

Fig. 3: Proposed Cluster Set up.

After the tasks are distributed to leaf nodes, they start executing. The nodes at higher level also execute some portion of the job scheduled to them by the nodes above it. The advantage we achieve is every node executes the portion of task it can without waiting for result from any node and buffering the results achieved. It continues to execute independent portions of tasks and buffering the results, thus by this strategy each machine is utilized and the waiting time for each node is minimized. When a node finishes execution of a task, it does two things before executing new set of tasks scheduled on to it. First, it passes the result to the next node waiting for it, second, it deallocates the memory it used for buffering. This process of passing results goes on until the last node waiting, receives results and then passes the final result to the leader node of the cluster. When the results are passed to leader node, it completes all tasks it had buffered and passes them on to the higher level. The root node in turn completes all tasks which are buffered and again schedules some jobs to each leader under it. This root node schedules jobs to nodes under it either after processing tasks buffered or before processing them.

4.1 Algorithm

In the algorithm implemented on cluster, every iteration consists of executing Segmentation routine (call to routine roi), Identification routines (call to routine blobs), and Track routine (call to routine match). In the algorithm we observe that result of each particular routine is used as parameter in the next routine. The algorithm looks serial where in output of each particular routine is input for next routine, but some things could be exploited for parallelizing the algorithm. Background routine runs for every N iterations and computes a background which is used for next N-1 iterations, and Segmentation (roi), Identification (blobs) routine could be run independently for each iteration as whose return value is used for that current iteration and not for next iterations, where as the data structures for Object and Object info are updated every iteration. When all the no. of iterations are completed in the Video Surveillance algorithm it returns Object, Object info as results. Therefore background, segmentation, identification form the portion of iteration which could be run independently at each node and buffer ROI values, wait for (objects) value from previous node.

4.2 Problems associated with implementation

In our implementation, we had faced problems of communicational delays which were fatal for performance. In the parallel version of the algorithm, there were several factors increasing communication delays. For instance, we could observe that the leader node had to check at every instance after a frame it had processed, whether job submitted to its respective leaf nodes has been completed or not, along with running its part of workload, thus increasing communication between nodes. Moreover, the other factors hindering the performance were that the images had to be transmitted to the nodes for processing, adjacent nodes had communicate the results of processed frame and thus summing these factors had increased the communicational delays to a major extent. Other factors like memory needed to store images, process them were also effective but due to increased improvements in hardware technology this factor could be hid considerably.

Algorithm Parallel Code

```

1: contact the scheduler and check its status;
2: while(i<=M) {
3: If( i % N==0) B=Update Background(past N frames)
4: If( job is complete) {
    temp=i;
    create a job with tasks of xi iterations for ith node,pass
    parameters (i,B,M) to nodes, submit job to scheduler.
    i=i+x+i%10;
  }
5: R=roi(B,current frame);
6: Bl=Blobs(R);
7: If( i>temp) Buffer the results of routine Blobs in array
Bl2;
8: If(job is complete) {
    Inspect job for errors and if display them;
    Execute match routine M=match (blv2[]) for iterations
    whose blobs output was buffered at this node.
  }
}
```

Algorithm Node n(i, B, M)

```

{
1:xn=no of iterations node n has to execute.
2:while(a<=xn)
{
3: If( i % N==0) B=Update Background(past N
frames)
4: R=roi(current frame,B);
5: Bl=Blobs(R);
6: If(node==2) M=match(Bl);
else { Buffer the results of routine Blobs in array
Bl2; }
7: If(node~=2)
{
Wait until node receives results of previous
iterations (M)
Execute match routine M=match (blv2[]) for
iterations whose blobs output was buffered at
this node.
}
}
8: If(node<=4) Pass results (M) to next node
}
```

5. REVIEW OF SOME VIDEO PROCESSING APPLICATIONS ON CBE

In [18], Liu et al. explains an implementation of Background subtraction system (BGS) system on STI Cell Broadband Engine (CBE). BGS finds objects by looking for moving regions against a stationary background. The BGS system is divided into four separate stages Image Pre-processing, Saliency Detection, Mask Generation and Model Maintenance. In order to make most efficient use of CBE's resources and be able to handle multiple video streams with any given number of SPEs, each SPE is assigned to complete a unit of work and then ready to be reassigned. 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 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 processed data back to PU memory with a DMA store operation. The overhead of the DMA operations can generally be hid using double buffering scheme. He could achieve nearly 6-9x improvement of speed up over the non parallel version of the application.

Yu et al. [19] presents a scheme for parallelizing video processing and retrieving (VPR) model on CBE. A parallel partition schema of video processing is suggested in his work. In his approach workloads were mapped on to SPEs using namely using Service, Streaming models. In the Service Model, the PPE assigns different services to different SPEs, and calls upon the appropriate SPEs when a particular service is needed. In the Streaming Model, the PPE acts as a stream controller and the SPEs 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 SPE.

In [20], Azevedo et al explains an implementation of Video filtering approach on CBE. In their work, they have implemented Deblocking Filter (DF) using scalar and vector (SIMD) approaches on the platform. PPE was used only for reading the parameters from the input files and to store them in main memory. After storing the parameters, the SPE threads were spawned. Thereafter, the PPE thread sends a signal to all SPEs to start the computation. Each SPE thread processes one frame and the processing starts by reading the input pointers for the samples and parameters from the main memory. Each frame was divided, to use the SPEs ability of performing computation and data communication in parallel. This partition is based on several factors such as the latency, maximum DMA transmission package size, number of DMA transfers, and organization of the data in the memory. The processing of the frame at each SPE was performed as a software pipeline and used a double buffering strategy. First, a part of data was requested, followed by the request of the data for the second portion. After the data of the first portion was available in the LS it is filtered. This way the processing of first portion is performed in parallel with the data transmission of second portion. In this way they have exploited the double buffering scheme of CBE.

In [21], Park et al. proposed an approach for parallelizing X.264 encoding algorithm of H.264 encoding scheme. They have suggested a macro block level parallelizing scheme for implementation, explaining scheme is preferable due to limited storage size of SPE. The algorithm was partitioned into three sections two for frame data processing and one for macro-block processing. In their implementation, a frame was broken into macro blocks in which the encoding for each block was done in pipelined fashion along with maintaining data dependency between processing of blocks. The paper had also presented a detailed analysis of the partitioning scheme explaining various performance related factors.

6. IMPLEMENTATION OF VIDEO SURVEILLANCE ON CELL BROADBAND ENGINE

The CBE consists of eight cores of SPE and one core of PPE, which forms altogether a heterogeneous platform and applications ported on it, must be parallelized in accordingly keeping in view of this aspect. In general, implementation of a system on CBE consists of three phases. First the 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 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. SPE form the accelerator cores of CBE which could be exploited for computational intensive operations. Our goal is to select the programming paradigm that offers the

simplest possible expression of an algorithm while being capable of fully utilizing the hardware resources of the Cell processor.

The Video surveillance system consists of Background Modeling, Motion Segmentation, Object Identification, and Object Tracking of which most of I/O operations are performed in Background modeling, and other routines perform the computational operations on the image read. As PPE has more access to I/O over SPEs and also to exploit the accelerators of CBE (SPE), we schedule Background modeling routine on PPE and others on the SPE's.

Fig. 4: A sample image which is processed, each SPU processes its portion of image and achieving data parallelism.

The crucial aspect for the implementation was, as the SPU has a limited storage capacity of 256 KB, it cannot accommodate an image totally to operate (a matrix read from an image size of 240×320 is nearly about 307 KB), and it needs to perform DMA operation of maximum repeatedly 16 KB to fetch image into its local memory. To cater this issue we need to distribute an image carefully on all SPE's so that they could operate synchronously. Thus, by this approach we could bring out data parallel programming amongst the SPE's [22]. In this approach we load a portion of image into local store of SPU by performing DMA. Figure 3 highlights the scheme which shows the break up of image into eight parts, and each SPU processing its portion of image. Even though, in this approach we have reduced the amount of DMA operations, they bring high communicational delays in the implementation of the algorithm, which brings down the computation to communication ratio or CCR [23] ratio of algorithm. To address this we use double buffering scheme by which overhead of DMA could be hidden upon an extent.

Apart from above issues, we face other challenge, of synchronization. Since all SPE's process in parallel an image they need to get synchronized while identifying and tracking an object. For, instance, in implementation of Object identification we find connected components in an image to get a region of interest, in which all the SPU's need to get synchronized so that objects are identified correctly. To handle this issue we perform DMA operation by which we store the processed matrices of SPU's at contiguous locations in DRAM and process them sequentially at PPE. The synchronization of SPU's could be done by using mailboxes where each SPU signals PPU whether it has finished its DMA, so as PPU could start processing to get connected

components in the image. Once we get connected components of an image, SPU executes its residual processing on the image. Figure 4 explains the parallelization technique overall used.

In overall implementation of our algorithm on the CBE, we try to minimize the idle time of the PPE by buffering the image, calculating background in advance of, completion of refresh rate of Video surveillance algorithm. The utilization of SPE was maximized by reducing DMA operations, unless waiting for synchronization with other SPE's in this approach. The algorithm summarized in table 3 below.

7. SUMMARY OF RESULTS

7.1 Experimental Results on CBE platform

The above algorithm has been simulated using CellSDK 2.0 simulator running on VMware Player (running on Windows based platform). The parameter that was measured was the total execution time of the algorithm with respect to the total number of frames processed. The speed up is 20.8 times faster compared to implementation of Video Surveillance on a Windows based Pentium workstation. Table 1 shows comparison between both approaches.

Table 1: Speed up on Cell in comparison with Pentium platforms.

	Processing time taken /frame
Implementation on Windows based Pentium platform	520 milliseconds
Simulated time using Cell SDK 2.0	12.07 milliseconds
SPEED UP	43.1077

Fig 5: Execution time of Video Surveillance on CBEA Simulator.

The table 2 describes the execution time of Video Surveillance Algorithm on Cell SDK by varying the number of SPEs used. The table shows algorithm run on (2, 4,5,6,8) number of SPEs using an image (of dimension 240*320). The experiment could not be done using 3,7 SPEs using the above parallelization strategy as image could not be fragmented by 3, 7 respectively with maintaining homogenous tasks for all

SPEs and moreover maintaining data alignment for DMA operations. Also, in case of using single SPE the total of image, matrices, data structures for objects used, and program at SPE exceeds local store capacity of 256KB, and needs excessive DMA operations where I/O factor dominates over computational exploitation, and hence degrading performance.

Table 2: Execution times and speed up of Video Surveillance Algorithm on CBE on varying number of SPEs used.

Number of SPEs used for simulation	Processing time taken 1 iteration processing 1 frame	SPEED UP
2 SPEs	129 milliseconds	4.03
4 SPEs	76 milliseconds	6.84
5 SPEs	45.467 milliseconds	11.443
6 SPEs	17.067 milliseconds	30.486
8 SPEs	12.07 milliseconds	43.1077

We could observe that total execution time increases with decreasing number of SPEs, not only due to increase in computational workload on each SPE but also due to the increment in number of DMA operations. Each DMA operation could fetch at maximum of 16 KB of data into or out of a local store, and the amount of DMA operations double from fetching 32 rows to 122 rows of image when SPEs used get halved from 8 (using all cores of CBE) to 2. Moreover, while using less number of SPEs for implementation, the number of computations increases as each SPE gets more data to process.

Other factor which could be noticed is that speed up drastically increases when we increase the usage of SPEs from 4 to 5 and 5 to 6, as in first case the amount of DMA calls reduce which amounts to the speed up, and in second case the amount of data transferred and decrease in computational workload per SPE accounts to speed up. Thus, from above results we could observe a steep increase in speed up, nevertheless utilizing the all the accelerator cores of CBE the desired speed up could be achieved.

7.2 Comparison of results with implementation on computer cluster

The Video Surveillance algorithm described in Section 2 was implemented on MATLAB R2006a version using the distributed computing toolbox as a part of our previous work. A local cluster was setup using the processors that formed part of the campus LAN in IIT Roorkee and were connected through coaxial cables MATLAB Distributed Computing Environment (MDCE) was used for configuring the cluster environment. Figure5 describes the comparison of experimental results between the cluster implementation and simulation using CellSDK based on number of iterations processed, and Table 3 shows the speed up comparison between CBE and computer cluster approaches.

Table 3: Speed up on Cell in comparison with cluster implementation of the algorithm.

	Processing time taken /frame
Implementation on computer cluster.	234 milliseconds
Simulated time using Cell SDK 2.0	12.07 milliseconds
SPEED UP	19.386

8. CONCLUSIONS AND FUTURE WORK

The implementation of a video surveillance algorithm in a Cell environment was carried out and its performance was shown to display a considerable improvement. The various issues related to implementation of the algorithm are general, yet specific algorithms will have to be developed for different surveillance algorithms.

The next step in this course would be to experiment with an actual Cell implementation and explore the performance of the model in a multi-camera surveillance scenario. The surveillance cameras could be connected via the internet to distant processors. A model for implementation for multi-camera fusion based surveillance system is provided in [24]. Also, performance of the algorithm could be increased by introducing some fast techniques of morphological operations so as to optimize the at image processing operations. Methods for Fast Morphological Image Transforms are provided in [25]. Another important issue to address in this case would be scalability and security aspects

9. REFERENCES

- [1] A. K. Nanda, J. R. Moulic, R. E. Hanson, G. Goldrian, M. N. Day, B. D. 'Amora, S. Kesavarapu., "Cell/B.E. blades: Building blocks for scalable, real-time, interactive, and digital media servers", IBM Journal of Research and Development, vol. 51, no. 5, pages:573-582, September 2007.
- [2] T. King, I. Shearer, "The Cell BE Processor: A Broadband Engine for Broadcast Applications", White paper, Mercury Systems Inc. 2007. <http://www.mc.com/uploadedFiles/Mercury-IPV-whitepaper.pdf> (last accessed on 10 June 2008).
- [3] R. Collins, A. Lipton, T. Kanade, H. Fujiyoshi, D. Duggins, Y. Tsing, D. Tolliver, N Enomoto, O. Hasegawa, P. Burt, and L. Wixson, "A system for video surveillance and monitoring", Technical report, CMU-RI-TR-00-12, Robotics Institute, Carnegie Mellon University, 2000.
- [4] I. Pavlidis, V. Morellas, P. Tsiamyrtzis, and S. Harp. "Urban surveillance systems: from the laboratory to the commercial world". Proceedings of IEEE, 2001, 89, (10), pp.1478149.
- [5] M. Valera and S.A. Velastin, "Intelligent distributed surveillance systems: a review". In Image and Signal Processing, IEEE Proceedings, April 2005, volume 152, pages 192 – 204.
- [6] T. P. Chen, H. Houssecker, A. Bovyryn, R. Belenov, K. Rodyushkin, A. Kuranov, V. Eruhimov, "Computer Vision Workload Analysis: Case Study of Video Surveillance Systems", Intel Technology Journal, Vol 9, Issue 2, May 2005.
- [7] "Sun Reference Architecture for Video Surveillance", White Paper, Sun Microsystems, Inc. 2007. <http://www.sun.com/videosurveillance> (last accessed on 2 June 2008).
- [8] "Video Over IP Servers from MOXA", http://www.moxa.com/land/Video_over_IP_Servers.htm (last accessed on 8 June 2008).
- [9] "Broadband Engine," Book IV for DD1.0, Version 1.0, SCEI/Toshiba/IBM, May 10, 2004.
- [10] M. Gschwind, H. P. Hofstee, B. Flachs, M. Hopkins, Y. Watanabe, T. Yamazaki, "Synergistic Processing in Cell's Multicore Architecture", IEEE, pages: 10-24, 2006.
- [11] H. P. Hofstee, "Introduction to the Cell Broadband Engine", White Paper, IBM Corporation 2008, <http://www-01.ibm.com/chips/techlib/techlib.nsf/techdocs/D21E662845B95D4F872570AB0055404D> (last accessed on 7 June 2008).
- [12] M. Gschwind, "Chip Multiprocessing and the Cell Broadband Engine", Proceedings of the 3rd conference on Computing Frontiers, pages: 1 - 8, 2006.
- [13] S. Williams, J. Shalf, L. Oliker, P. Husbands, S. Kamil, K. Yelick, "The Potential of the Cell Processor for Scientific Computing", Technical report, LBNL-59071, Lawrence Berkeley National Laboratory, University of California, 2005.
- [14] M. Gschwind, "The Cell Broadband Engine: Exploiting Multiple Levels of Parallelism in a Chip Multiprocessor", International Journal of Parallel Programming, June 2007, vol. 35, no. 3, pages 233-262.
- [15] J. W. Davis and V. Sharma. "Fusion-Based Background-Subtraction using Contour Saliency". Computer Vision and Pattern Recognition, 20-26 June, 2005.
- [16] P. Kumar, A. Mittal, P. Kumar. "Study of Robust and Intelligent Surveillance in Visible and Multimodal Framework", Informatica, vol. 31, part 4, pages 447-462, 2007.
- [17] P.V.Kumar, P. Kumar, A. Mittal, R. Dubey. "A Scheduling Architecture for Distributed Video Surveillance System", Proceedings of ICSCI 2008, Hyderabad, pages: 104-112, January 2008.
- [18] L. Liu S. Kesavarapu, J. Connell, A. Jagmohan, A. Leem, L. Paulovicks, B. Sheinin, V. L. Tang H. Yeo, "Video Analysis and Compression on the STI Cell Broadband Engine Processor", IEEE International Conference on Multimedia and Expo, pages: 29-32, July 2006.
- [19] J. Yu, H. Wei. "Video Processing and Retrieval on Cell Processor Architecture", Entertainment Computing – ICEC 2007, LNCS, Springer Verlag Publication, volume 4740, pages :1-12, 2007.
- [20] A. Azevedo, C. Meenderinck, B. Juurlink. "Analysis of Video Filtering on the Cell Processor", Proceedings of

International Symposium on Circuits and Systems (ISCAS, pages: 488-491, May 2008.

- [21] J. Park, S. Ha. "Performance Analysis of Parallel Execution of H.264 Encoder on the Cell Processor", In the Proceedings of IEEE/ACM/IFIP Workshop on Embedded Systems for Real-Time Multimedia, pages: 27-32, 2007.
- [22] CBEA Programming Tutorial, IBM Systems & Technology Group, Version 2, December 2006.
- [23] Y.Yu,B.Krishnamachari,V.K.Prasanna. "Issues in Designing Middleware for Wireless Sensor Networks".IEEE Network ,January/February 2004,Pages 15-21.
- [24] P.Kumar, A.Mittal, and P.Kumar. "Fusion of Thermal Infrared and Visible Spectrum Video for Robust Surveillance." , LNCS, Springer Verlag Publication 2006, Volume 4338, Page(s):528-539.
- [25] R. D. Boomgaard, R. Balen. "Methods for Fast Morphological Image Transforms Using Bitmapped Binary Images", CVGIP: Graphical Models and Image Processing vol. 54, no. 3, pages: 252-258, May 1992