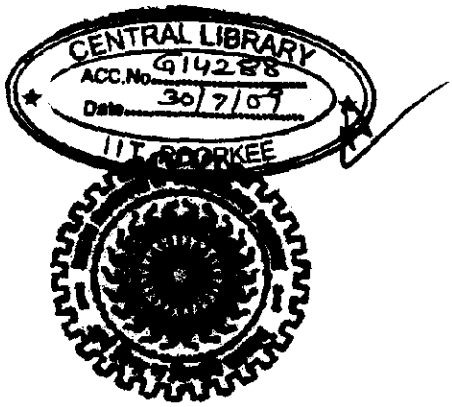
PERFORMANCE IMPROVEMENT OF VIDEO SURVEILLANCE ALGORITHM USING CELL BROADBAND ENGINE

A DISSERTATION

Submitted in partial fulfillment of the requirements for the award of the degree
of
MASTER OF TECHNOLOGY
in
COMPUTER SCIENCE AND ENGINEERING

By

P. VINAY KUMAR



DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY ROORKEE
ROORKEE - 247 667 (INDIA)
JUNE, 2008

Candidate's Declaration

I hereby declare that the work being presented in the dissertation report titled "Performance improvement of Video Surveillance Algorithm using Cell Broadband Engine" in partial fulfillment of the requirement for the award of the degree of Master of Technology in Computer Science and Engineering, submitted in the Department of Electronics and Computer Engineering, Indian Institute of Technology Roorkee, is an authenticate record of my own work carried out under the guidance of Dr. Ankush Mittal, Associate Professor, Department of Electronics and Computer Engineering, Indian Institute of Technology Roorkee.

I have not submitted the matter embodied in this dissertation report for the award of any other degree.

P. Vinay Kumar

Dated: 6th July '08 (P. Vinay Kumar)

Place: IIT Roorkee.

Certificate

is is to certify that above statements made by the candidate are correct to the best of knowledge and belief.

Dated: 6th July '08

Place: IIT Roorkee.

Dr. Ankush Mittal,
Associate Professor,
Department of Electronics and
Computer Engineering, IIT Roorkee,
Roorkee -247667 (India).

ACKNOWLEDGEMENTS

I am thankful to Indian Institute of Technology Roorkee for giving me this opportunity. It is my privilege to express thanks and my profound gratitude to my supervisor Dr. Ankush Mittal, Associate Professor for his invaluable guidance and constant encouragement throughout the dissertation. I was able to complete this dissertation in this time due to constant motivation and support obtained from Dr. Ankush Mittal.

I am also grateful to the staff of Sponsored Research Laboratory and UGPC Laboratory for their kind cooperation extended by them in the execution of this dissertation. I am also thankful to all my friends who helped me directly and indirectly in completing this dissertation.

I am thankful to Mr. Praveen Kumar, Research scholar in my department for his constant encouragement in my work. I am grateful to Mr. Avinash Sharma, Mr. Parikshit Sondhi, Mr. Rajarshi Chowdhury and Mr. C. Sekhar, my colleagues for being excellent peers and creating a congenial environment for work.

Most importantly, I would like to extend my deepest appreciation to my family for their love, encouragement and moral support. Finally I thank God for being kind to me and driving me through this journey.

(P. VINAY KUMAR)

Abstract

With the growing trend of technology in network and computers science arena, many of the high end applications like multimedia related applications are becoming algorithmically complex and data intensive. Video Surveillance is one such growing business application due to its concern in various areas like commercial security, and military applications. With increasing deployment in large scale distributed environment, the challenge is to process huge multimedia stream data with several computation intensive algorithms. Hardware researchers are trying out to find a cost effective solution for the application. A dedicated hardware solution for the application proves cost expensive and decreases scalability. Alternatively, we have HPC solutions such as cluster, grid and multicore processor for the problem. Research community either had addressed solution for Video Surveillance or for a generalized HPC solution for high end applications but none had addressed solution concerning with video surveillance specific requirements.

In this dissertation work, we present a method to parallelize and implement video surveillance algorithm on the STI Cell Broadband Engine. The methodology provides a conceptual architecture for parallelizing applications with requirements similar to video surveillance. We then briefly present our previous work on computer cluster, discuss various issues and challenges related to porting the code, followed by results demonstrating the speed up and comparison of the results of both approaches.

The implementation of programs is done using C, simulation in Cell SDK 2.0, images used were stored on PC, of Windows bitmap format.

List of Figures

List of Tables

Table of Contents

CHAPTER 1
INTRODUCTION

1.1 Introduction

High end applications are classified as those which require high computing power, high bandwidth, and lots of storage. Multimedia stream processing applications are good examples of high end applications which are highly computational and data intensive and have been posing great challenges to the computing arena. Video rendering, gaming, video surveillance, shape analysis and many graphical applications are becoming more and more algorithmically complex and are demanding more computational power. In order to meet the intensive computation and the real-time processing, many research efforts have been done in the past, at various institutions. For instance, Intel has developed various libraries for multimedia related applications. They provide detailed case study of Intel platforms for video surveillance applications [1]. [2] provides description of dedicated computational platform support developed by Sun Microsystems, which includes high-resolution digital video and integrated surveillance data management infrastructure system for video surveillance. [3] provides description of hardware (SpursEngine) platform for video processing applications, developed by Toshiba Corporation, which uses STI Cell Broadband Engine technology, with inbuilt dedicated hardware for decoding and encoding of MPEG-2 and H.264 video, XDR memory interface as well as PCI Express interface.

On the other hand researchers at various academic institutions have been striving hard to provide cost effective and efficient real time solution for the problem. An implementation of distributed architecture for video processing and communication techniques is provided in [4]. The possible HPC solutions for the problem could be 1) cluster 2) deployment of grid 3) multicore processor platforms for the applications. One of the important video processing applications is automatic video surveillance which is very rapidly growing sector in the commercial market due to its wide range of applications, such as a homeland security, a security guard for communities and important buildings, traffic surveillance in cities and detection of military targets, etc [5, 6]. Cluster and Grid solutions suffer from high communicational delays and moreover increase the cost of deployment. Emerging on-chip multiprocessor

architecture provides the capability to the programmer to optimize the execution of instructions and data handling for specific applications. A recent breakthrough with introduction of the STI Cell Processor has provided a new alternative for the researchers to investigate the platform. Hardware giants such as IBM [7], Mercury [8] are bringing up effective solutions to Video Processing applications using this processor. [7] provides a review of implementations of various media and real time applications on the Cell processor, [8] provides implementation of H.264 video compression technique on the Cell processor.

1.2 Motivation

Video surveillance is one of very rapidly growing sector in the commercial market due to its wide range of applications. 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. With the recent advancements in video and network technology, there is a proliferation of inexpensive network based cameras and sensors for widespread deployment at any location, as well as with the development of new video-processing and computer-vision algorithms allowing more complex scenes to be considered. All this progress has made it possible and necessary to consider a new perspective in this field and an opportunity to exploit them. A good review of intelligent and distributed surveillance systems is provided in [9].

As there is deployment of progressively larger systems, often consisting of hundreds or even thousands of cameras distributed over a wide area, there is enormous amount of media stream data that need to be communicated over a network to a central station and processed on high powered dedicated processors. Since computation on the huge data of video streams is large which need to be done in online fashion, one of the major challenges is to achieve real time computation on the incoming video frames [10, 11]. Alongside with these advances surveillance application is also becoming algorithmically complex and data intensive. Therefore there is a serious requirement of a high performance solution to cater the needs of the problem. We provide a parallelization strategy to implement the application on computer cluster and STI Cell

Broadband Engine in our work. Also, we compare performance of the application on both platforms.

1.3 Contributions

In our implementation on cluster we face the major problems of communicational delays and busy waiting. We had tried to optimize the waiting time by distributing the workloads in heterogeneous sizes and buffering them, but the delays form the major challenge where nodes had to communicate results on LAN, in which we had faced problems of network congestion, bandwidth. To provide efficacious solution to the problem, we investigate the performance of application on STI Cell Broadband Engine. Inherently, we faced challenges of limited buffer and synchronization on this platform. We had solved these by selecting a desired programming paradigm that well suits the application specific requirement.

1.4 Problem Statement

In this dissertation work we propose and implement a model to improve a high end application such as video surveillance system using parallel processing techniques and architecture such as Cell broadband engine. The model has an objective to achieve significant performance speedup over the non distributed version, thus enabling a real time implementation.

1.5 Organization of the Report

The organization of the dissertation is as follows:

Chapter 2 discusses the hardware architecture of the STI Cell Broadband Engine, describing about various modules of the platform.

Chapter 3 gives the background of Video Surveillance model, a typical algorithm for implementation discussing various computational issues and challenges in its implementation.

Chapter 4 describes review of related work on cluster and our previous work in implementing the application on the same. We then discuss various issues like extracting parallelism in the algorithms, communicational delays, initialization and other overheads, synchronization etc.

Chapter 5 describes work done in implementations of Video processing applications on Cell and other platforms, discussing extraction of parallelism, issues and challenges in their approaches. We then explain our proposed model of Video Surveillance Algorithm on the STI Cell Broadband Engine with detailed description of various issues in regard to porting the code and hurdles faced in parallelizing application in regard to implementation on the platform.

Chapter 6 discusses the performance of the application on STI Cell Broadband Engine, statistics describing speed up over non distributed version and on the cluster implementation of the Video Surveillance Application, speed up by varying number of cores used for the application, snapshots of various screens and graphs depicting the performance.

Chapter 7 concludes the dissertation work and gives suggestions for future work.

CHAPTER 2

STI CELL ARCHITECTURE

2.1 Background and Motivation

Cell Broadband Engine (CBE) is an outgrowth of a strategic alliance between Sony Computer Entertainment Inc., Toshiba Corporation and IBM Corporation formed in 2001. The collaboration, known as STI, opened their Austin-based design centre in March, 2001 . In spring, 2004 the first ever operational cell-processor was released. In 2005 Mercury announced Cell Blade servers. In 2006 IBM and Sony Corp. offered Cell Blade server and PlayStation 3 gaming console respectively. Also Cell Processor has been incorporated in IBM's supercomputer named Blue Gene.

The CBE processor is the first implementation of a new family of multiprocessors conforming to the Cell Broadband Engine Architecture which extends 64 bit Power PC Architecture. Although the CBE processor is initially intended for application in media-rich consumer-electronics devices such as game consoles and high-definition televisions, the architecture has been designed to enable fundamental advances in processor performance. These advances are expected to support a broad range of applications in both commercial and scientific fields.

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

The hardware environment of Cell Broadband Engine is described in Figure 2.1 which broadly consists of five parts as described as follows:

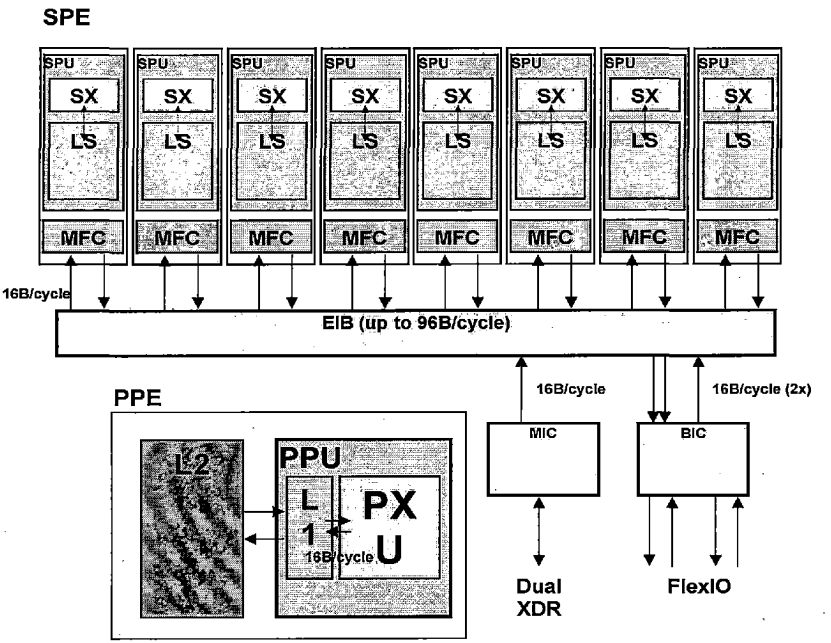


Figure 2.1: The Hardware architecture of Cell Broadband Engine which is based on heterogeneous multiprocessor architecture [12].

(i) PowerPC Processor Element (PPE) :

The PPE contains a 64-bit PowerPC Processor Unit (PPU) with associated caches that conform to 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 also includes a vector multimedia extension unit, called Single Instruction, Multiple Data (SIMD), so that it can do multiple operations simultaneously with a single instruction.

The PPE consists of two main units Power Processor Unit (PPU) and PowerPC Processor Storage Subsystem (PPSS). The PPU performs instruction execution, and it has level 1 (L1) instruction cache, data cache of 32KB each, 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 of 512KB.

The PPU supports two simultaneous threads of execution and can be viewed as a 2-way multiprocessor with shared dataflow. This appears to software as two independent processing units. The state for each thread is duplicated, including all architected and special-purpose registers except those that deal with system-level resources, such as logical partitions, memory, and thread-control. Most non architected resources, such as caches and queues, are shared by both threads, except in cases where the resource is small or offers a critical performance improvement to multithreaded applications.

The PPSS handles memory requests from the PPE and external requests to the PPE from other processors or I/O devices. It includes a unified 512-KB level 2 (L2) instruction and data cache, various queues, and a bus interface unit that handles bus arbitration and pacing on the EIB. Memory is seen as a linear array of bytes indexed from 0 to 264 - 1. Each byte is identified by its index, called an address, and each byte contains a value. One storage access occurs at a time, and all accesses appear to occur in program order. The L2 cache and the address-translation caches use replacement-management tables that allow software to control use of the caches. This software control over cache resources is especially useful for real-time programming.

The PPEs are general-purpose processing units, and can access system management resources. These resources can be for example the memory-protection tables. Hardware resources are mapped explicitly to the real address space as seen by the PPEs. PPE can address any of these resources directly by using an appropriate effective address value.

The primary function of the PPEs is the management and allocation of tasks for the SPEs in a system. In figure 2.2, we can see the placement of the PPE in the actual Cell chip. When data enters the PPE, this element then distributes it among SPEs,

schedules them to be processed on one or more of the SPEs, controls and synchronizes them.

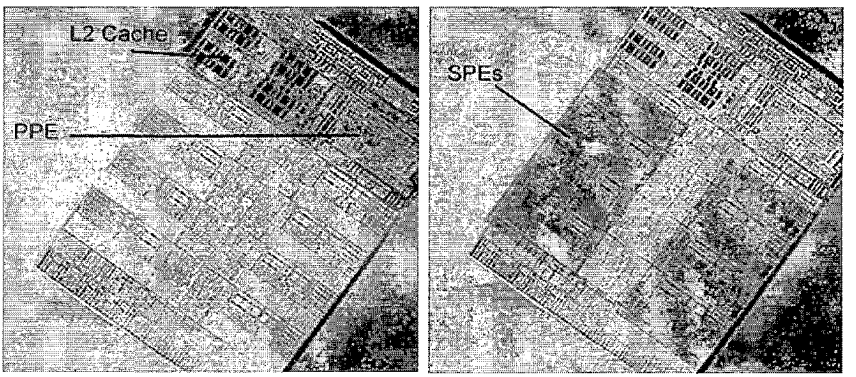


Figure 2.2: PPE and SPE placing on the Cell Processor [12].

(ii) Synergistic Processor Elements (SPEs) :

Each of the eight Synergistic Processor Elements (SPEs) is a 128-bit RISC processor specialized for data-rich, compute-intensive SIMD applications. It consists of two main units, the Synergistic Processor Unit (SPU) and the Memory Flow Controller (MFC). In figure 2.2, we can see the placement of the SPE in the actual Cell chip.

The SPU deals with instruction control and execution. It includes a single register file with 128 registers (each one 128 bits wide), a unified (instructions and data) 256-KB local store (LS), an instruction-control unit, a load and store unit, two fixed-point units, a floating-point unit, and a channel-and-DMA interface. The SPU implements a new SIMD instruction set, the SPU Instruction Set Architecture, which is specific to the Broadband Processor Architecture.

Each SPU is an independent processor with its own program counter and is optimized to run SPE threads spawned by the PPE. The SPU fetches instructions from its own LS, and it loads and stores data from and to its own LS. With respect to accesses by its SPU, the LS is unprotected and un-translated storage.

The MFC contains a DMA controller that supports DMA transfers. Programs running on the SPU, the PPE, or another SPU, use the MFC's DMA transfers to move instructions and data between the SPU's LS and main storage. (Main storage is the effective-address space that includes main memory, other SPEs' LS, and memory-mapped registers such as memory-mapped I/O [MMIO] registers.) The MFC interfaces the SPU to the EIB, implements bus bandwidth-reservation features, and synchronizes operations between the SPU and all other processors in the system.

To support DMA transfers, the MFC maintains and processes queues of DMA commands. After a DMA command has been queued to the MFC, the SPU can continue to execute instructions while the MFC processes the DMA command autonomously and asynchronously. The MFC also can autonomously execute a sequence of DMA transfers, such as scatter-gather lists, in response to a DMA-list command. This autonomous execution of MFC DMA commands and SPU instructions allows DMA transfers to be conveniently scheduled to hide memory latency.

Each DMA transfer can be up to 16 KB in size. However, only the MFC's associated SPU can issue DMA-list commands. These can represent up to 2,048 DMA transfers, each one up to 16 KB in size. DMA transfers are coherent with respect to main storage. Virtual-memory address translation information is provided to each MFC by the operating system running on the PPE. Attributes of system storage (address translation and protection) are governed by the page and segment tables of the PowerPC Architecture. Although privileged software on the PPE can map LS addresses and certain MFC resources to the main-storage address space, enabling the PPE or other SPUs in the system to access these resources, this aliased memory is not coherent in the system.

The SPEs provide a deterministic operating environment. They do not have caches, so cache misses are not a factor in their performance. Pipeline-scheduling rules are simple, so it is easy to statically determine the performance of code. Although the LS is shared between DMA read and write operations, load and store operations, and instruction prefetch, DMA operations are accumulated and can only access the LS for at most one of every eight cycles. Instruction pre fetch delivers at least 17 instructions

sequentially from the branch target. Thus, the impact of DMA operations on loads and stores and program-execution times is, by design, limited.

Memory-mapped mailboxes or atomic MFC synchronization commands can be used for synchronization and mutual exclusion. A detailed explanation of architectural details regarding SPE is given in [13].

(iii) Memory Flow Control

Cell processor contains a dual channel next-generation Rambus XIO macro which interfaces to Rambus XDR memory. The memory interface controller (MIC) is separate from the XIO macro and is designed by IBM. The XIO-XDR link runs at 3.2 Gbit/s per pin. Two 32 bit channels can provide a theoretical maximum of 25.6 GB/s.

The system interface used in Cell, also a Rambus design, is known as FlexIO. The FlexIO interface is organized into 12 lanes, each lane being a unidirectional 8-bit wide point-to-point path. Five 8-bit wide point-to-point paths are inbound lanes to Cell, while the remaining seven are outbound. This provides a theoretical peak bandwidth of 62.4 GB/s (36.4 GB/s outbound, 26 GB/s inbound) at 2.6 GHz. The FlexIO interface can be clocked independently, typ. at 3.2 GHz. 4 inbound + 4 outbound lanes are supporting memory coherency. Figure 6 explains the memory flow control system in the CBE. Memory Flow Control System consists of the following:

- •DMA Unit
  - •LS <-> LS, LS<-> Sys Memory, LS<-> I/O Transfers
  - •8 PPE-side Command Queue entries
  - •16 SPU-side Command Queue entries
- •MMU similar to PowerPC MMU
  - •8 SLBs, 256 TLBs
  - •4K, 64K, 1M, 16M page sizes
  - •Software/HW page table walk
  - •PT/SLB misses interrupt PPE
- •Atomic Cache Facility
  - •4 cache lines for atomic updates

•2 cache lines for cast out/MMU reload

•Up to 16 outstanding DMA requests in BIU

•Resource / Bandwidth Management Tables

•Token Based Bus Access Management and TLB Locking.

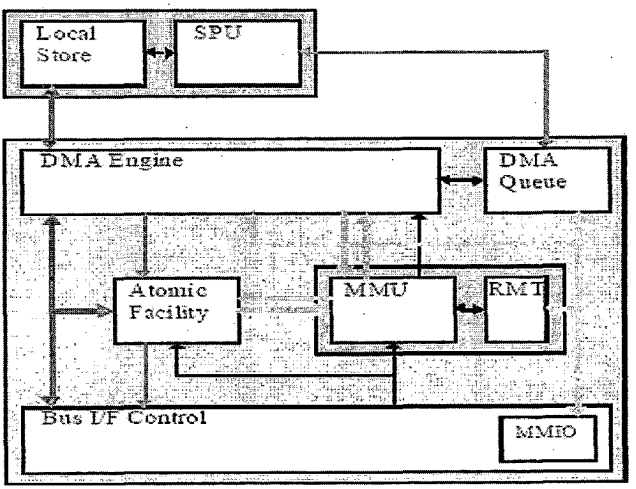


Figure 2.3: Memory Flow Control System [13].

(iv) Element Interconnect Bus (EIB) :

The EIB is a communication bus internal to the Cell processor which connects the various on-chip system elements: the PPE processor, the memory controller (MIC), the eight SPE coprocessors, and two off-chip I/O interfaces, for a total of 12 participants. The EIB also includes an arbitration unit which functions as a set of traffic lights.

The EIB is presently implemented as a circular ring comprised of four 16B-wide unidirectional channels which counter-rotate in pairs. When traffic patterns permit, each channel can convey up to three transactions concurrently. As the EIB runs at half the system clock rate the effective channel rate is 16 bytes every two system clocks. At maximum concurrency, with three active transactions on each of the four rings, the peak instantaneous EIB bandwidth is 96B per clock (12 concurrent transactions * 16 bytes wide / 2 system clocks per transfer). While this figure is often quoted in IBM

literature it is unrealistic to simply scale this number by processor clock speed. The arbitration unit imposes additional constraints.

Each participant on the EIB has one 16B read port and one 16B write port. The limit for a single participant is to read and write at a rate of 16B per EIB clock (for simplicity often regarded 8B per system clock). Note that each SPU processor contains a dedicated DMA management queue capable of scheduling long sequences of transactions to various endpoints without interfering with the SPU's ongoing computations; these DMA queues can be managed locally or remotely as well, providing additional flexibility in the control model.

Data flows on an EIB channel stepwise around the ring. Since there are twelve participants, the total number of steps around the channel back to the point of origin is twelve. Six steps is the longest distance between any pair of participants. An EIB channel is not permitted to convey data requiring more than six steps; such data must take the shorter route around the circle in the other direction. The number of steps involved in sending the packet has very little impact on transfer latency: the clock speed driving the steps is very fast relative to other considerations. However, longer communication distances are detrimental to the overall performance of the EIB as they reduce available concurrency.

Despite IBM's original desire to implement the EIB as a more powerful cross-bar, the circular configuration they adopted to spare resources rarely represents a limiting factor on the performance of the Cell chip as a whole. In the worst case, the programmer must take extra care to schedule communication patterns where the EIB is able to function at high concurrency levels.

Element Interconnect Bus or EIB uses data ring for internal communication. The features of EIB are:

- Four 16 byte data rings, supporting multiple transfers
- 96B/cycle peak bandwidth
- Over 100 outstanding requests
- Each EIB Bus data port supports 25.6GBytes/sec* in each direction

- The EIB Command Bus streams commands fast enough to support 102.4 GB/sec for coherent commands, and 204.8 GB/sec for non-coherent commands.
- The EIB data rings can sustain 204.8GB/sec for certain workloads, with transient rates as high as 307.2GB/sec between bus units.
- Physically overlaps all processor elements
- Central arbiter supports up to three concurrent transfers per data ring
- Two stage, dual round robin arbiter
- Each element port simultaneously supports 16B in and 16B out data path
- Ring topology is transparent to element data interface

(v) Memory Interface Controller (MIC)

The MIC provides the interface between the EIB and main storage. It supports two Rambus Extreme Data Rate (XDR) I/O (XIO) memory channels and memory accesses on each channel of 1-8, 16, 32, 64, or 128 bytes.

(vi) Cell Broadband Engine Interface (BIC)

The BEI manages data transfers between the EIB and I/O devices. It provides address translation, command processing, an internal interrupt controller, and bus interfacing. It supports two Rambus FlexIO external I/O channels. One channel supports only non coherent I/O devices. The other channel can be configured to support either non coherent transfers or coherent transfers that extend the logical EIB to another compatible external device, such as another Cell Broadband Engine.

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 [14, 15].

3.3 Software Development Kit:

An SDK is available for the Cell Broadband Engine. The SDK contains the essential tools required for developing programs for the Cell Broadband Engine. The SDK consists of numerous components including the following [16]:

- The IBM Full System Simulator for the Cell Broadband Engine, systemsim.

- system root image containing Linux execution environment for use within systemsim.
- GNU tools including C and C++ compilers, linkers, assemblers and binary utilities for both PPU and SPU.
- IBM XLC (C and C++) compiler for both PPU and SPU.
- newlib for the SPU. newlib is a C standard library designed for use on embedded systems.
- gdb debuggers for both PPU and SPU with support for remote gdbserver debugging. The PPU debugger also provides combined, PPU and SPU, debugging.
- PPC64 Linux with CBE enhancements.
- SPE Runtime management library supporting SPE thread services - libspe. A next generation prototype SPE Runtime management, libspe2, is also provided.
- Static timing analysis timing tool, spu_timing, that instruments assembly source (either compiler or programmer generated) with expected instruction timing details.
- System wide profiler for Linux call oprofile.
- An Eclipse based Integrated Development Environment (IDE) to improve programmer productivity and integration of development tools.
- Standardized SIMD math libraries for the PPU's Vector/SIMD Multimedia Extension and the SPU.
- Example source code containing samples, libraries, workloads, and prototype tools. See the following section for more details.

CHAPTER 3
VIDEO SURVEILLANCE SYSTEM

3.1 Evolution of Surveillance Systems

"First generation" video-based surveillance systems started with analog CCTV systems, which consisted of a number of cameras connected to a set of monitors through automated switches. But the human supervision being expensive and ineffective due to widespread deployment of such systems, they are more or less used as a forensic tool to do investigation after the event has taken place. By combining computer vision technology with CCTV systems for automatic processing of images and signals, it becomes possible to proactively detect alarming events rather than passive recording. This led to the development of semi-automatic systems called "second generation" surveillance systems, which require a robust detection and tracking algorithm for behavioral analysis. Second-generation surveillance systems constitute the current state of the art from a commercial viewpoint.

The main technical innovation in second-generation surveillance systems is the introduction of digital video representation. Second-generation surveillance systems had explored the advantages of digital approaches to acquisition, transmission, processing, storage, and visualization. Third generation surveillance system is aimed towards the design of large distributed and heterogeneous surveillance systems for wide area surveillance like monitoring movement of military vehicles on borders, surveillance of public transport etc. Many projects have been undertaken for development of third generation surveillance systems with network of cameras and distribution of processing capacity. For example the Defense Advanced Research Projection Agency (DARPA) supported the Visual Surveillance and Monitoring (VSAM) project [17] in 1997, whose purpose was to develop automatic video understanding technologies that enable a single human operator to monitor behaviors over complex areas such as battlefields and civilian scenes. The usual design approach of these vision systems is to build a wide network of cooperative multiple cameras and sensors to enlarge the field of view. From an image processing point of view, they are based on the distribution of processing over the network and the use of

embedded signal processing devices to give the advantages of scalability and robustness potential of distributed systems.

3.2 Basic Model

The general framework of an automatic video surveillance system is shown in Figure 3.1. 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 throughout a network to a control and visualization center that manages, for example, alerts. The main video processing stages include background modeling, motion segmentation, object identification and object tracking.

The model 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 [18], where we compute the mean and variance over the last N frames which serve the model for the next N frames.

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.

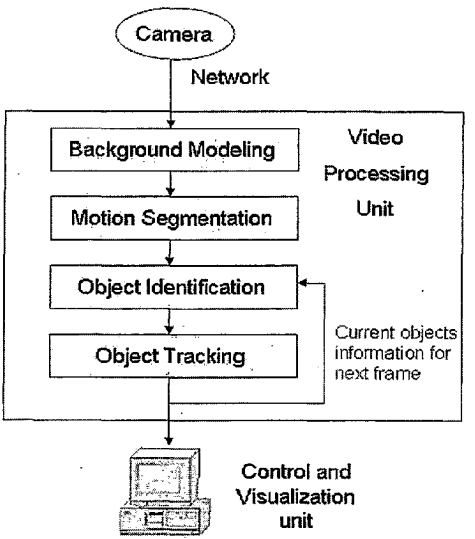


Figure 3.1: General framework of automated visual surveillance system [19].

3.3 Module Description

(i) Background Modeling:

In Background Modeling, we get a background for frames to be processed (although the term 'background' is not defined scientifically defined and their meaning may vary across various applications). For example, a moving car should usually be considered as a foreground object but when it parks and remains still for a long period of time, it is expected to become background.

(ii) Motion Segmentation:

In computer vision, segmentation refers to the process of partitioning a digital image into multiple regions or set of contours. Each of the pixels in a region is similar with respect to some characteristic or computed property, such as color, intensity, or texture. The goal of segmentation is to simplify and/or change the representation of an

image into something that is more meaningful and easier to analyze. Image segmentation is typically used to locate objects and boundaries in images. After the background calculations we identify a local region of interest by segmenting subsequent images, each containing the moving objects. The foreground and background gradient information within each region are then combined into a contour saliency map (highlighting the object's boundary). The gradients are thinned into contour fragments which correspond to the region of interests of image which are processed further to binary large objects (i.e., blobs).

(iii) Object Identification:

Object identification aims at segmenting regions corresponding to moving objects such as vehicles and humans from the rest of an image. Identifying moving regions provides a focus of attention for later processes. The regions processed in previous stage are further processed, their gradients are further thinned using morphological operations to segment out as blobs. An object is made up of one or more blobs.

(iv) Object Tracking:

Tracking concerns a process starting with determining the current and past locations of the objects position and trajectory. Object tracking aims at finding the locations of objects in the scene, labeling and tracking the objects in the scene. The positions of the regions or blobs processed by previous routines are calculated, labeled, and various other parameters like frame number, area occupied by the object in the scene are stored if object was seen at first instance else its parameters are updated accordingly as found in the new scene. Our implementation of the Video Surveillance model is based on [19].

3.4 Algorithm

The algorithm consists of various components like Background updation, Segmentation routine (call to routines roi, blobs), Matching roùtines and Track routine (call to routine match). Figure 2 shows the program structure of algorithm. Update Background is used to update background for every N frames (where N=10 in our implementation). ROI routine is used to select region of interest in the image which is currently under processing. Blobs routine is used to make a boundary or

bounding box over the region of interests like humans, moving objects in the image. Track routines like match is used to match the current observed object with previous objects collected so far, if it is new one it is collected here. The algorithm is shown below. Figure 2 shows the program structure for the implementation of the Algorithm, and Figure 3 shows the objects detected in the scene by Video Surveillance Algorithm.

ALGORITHM 1

```
Algorithm Video Surveillance
{
Step1: for I=1 to M
        {
Step2: Read the current video frame/image (I)
Step3: If (I % N==0)   Background =Update Background (past N frames)
Step4: ROI=Segmentation routine (current frame, Background);
Step 5: Objects=Object Identification (ROI, Objects Info)
Step 6: Objects Info=Track routine (Objects)
        }
}
```

Video Surveillance Algorithm.

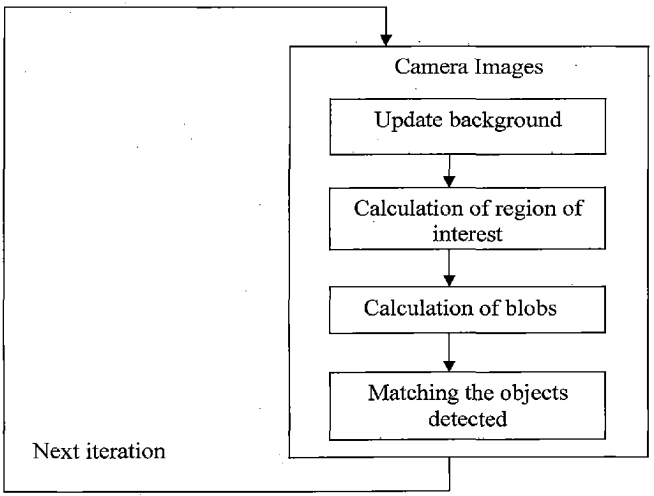


Figure 3.2: Program structure for the Video surveillance Model.

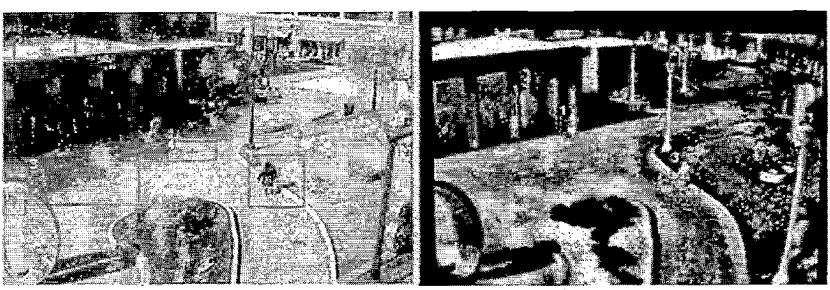


Figure 3.3(a): Visible Image showing various objects detected in the scene.
3.3(b): Infra Red Image showing various objects detected in the scene.

3.5 Computational Issues and Challenges

For computation purpose, each frame is a matrix of size p x q (say 240 x 320). The first sub procedure Update Background is concerned with finding mean and variance over N frames. 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. 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 expensive operation than other components of the algorithm. In its implementation we operate on many matrices of order nearly (240 x 320), and update some three dimensional data structures and using many loops. Loop unrolling [20] benefits in increasing performance of this routine, but the operation on matrices and three dimensional data structures make it computationally intensive. The other computational components in descending order are track routine, identification routine, segmentation routine. So, to process the incoming video frames in real time makes the application very computation intensive. Seeing the above perspective of video surveillance algorithm it suggests that it is a computationally intensive application and needs to be parallelized to increase its performance.

CHAPTER 4
IMPLEMENTATION OF VIDEO SURVEILLANCE ON CLUSTER

4.1 Introduction

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

4.2 Review of related work

Czarnul et al. [21] explains implementation of parallel image processing scheme for GIMP plug in which enables to invoke a series of filter operations in a pipeline in parallel on a set of images loaded by the plug in. DAMPVM environment was used for scheduling workloads on to cluster, where the cluster was based on Linux based Intel platforms. A methodology for scheduling tasks is presented, where the strategy is based on selecting idle nodes, queuing tasks and monitoring the load. A speed up of 14x is achieved, in which the speed up raises linearly on increasing the number of processors used.

Buyya et al. [22] presents taxonomy of various scheduling approaches for building and executing workloads on grids, providing details of their characterization. Also, a review of Grid workflow systems developed is provided. Taxonomies where workflow design was based on workflow structure, specification, composition and quality constraints are presented.

4.3 Our implementation

4.3.1 Cluster Setup

In our previous work we had proposed a parallel architecture for mapping the application on a cluster as shown in figure 4.1. 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 a 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.

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.

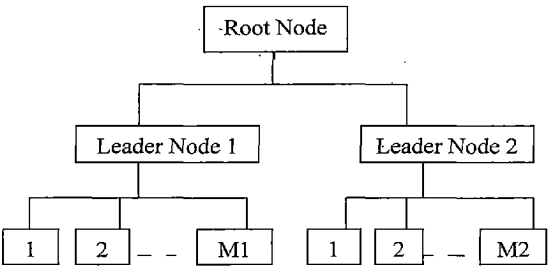


Figure 4.1: Proposed Cluster Set up.

4.3.2 Implementation of Video Surveillance Algorithm for parallelization

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. In this model the observation could be made that the ratio of parallelizable portion of iteration versus non-parallelizable portion is considerably good (as Background, blobs, roi forms parallelizable portion where Background forms most computational intensive task and track routine (match) forms non-parallelizable portion).

The program structure (shown in figure 4.2) of algorithm implemented consists of execution of program at leader node (who is responsible to schedule jobs onto other nodes), and at nodes. Leader node is any one selected amongst the cluster of nodes. Algorithms for implementation of application on cluster are shown below.

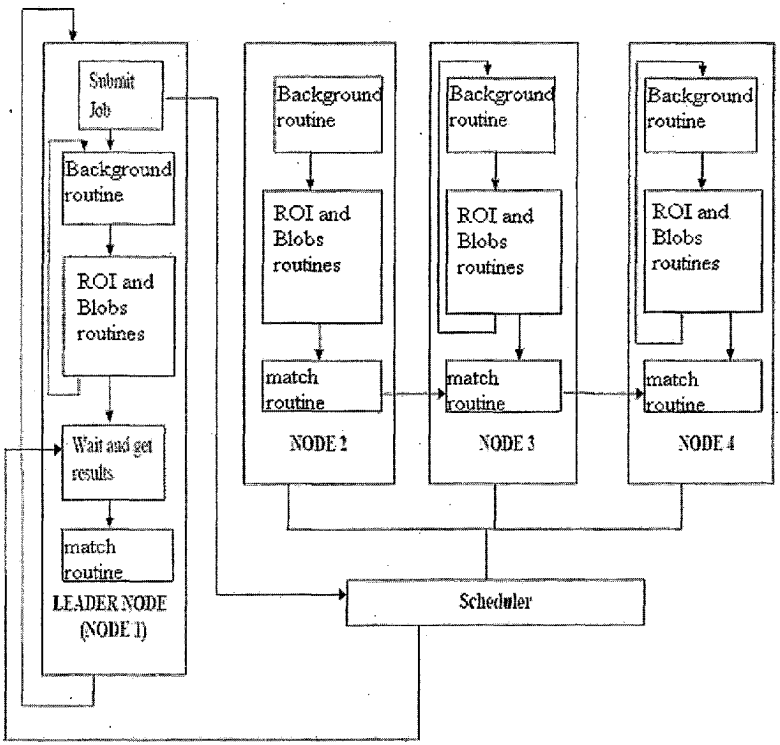


Figure 4.2: Implementation of Video Surveillance on cluster.

ALGORITHM 2

```
Algorithm Parallel Code
{
  Step 1: contact the scheduler and check its status;
  Step 2: while(i<=M)
          {
  Step 3:  If( i % N==0)  B=Update Background(past N frames)
  Step 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;
                                  }
  Step 5:  R=roi(B,current frame);
  Step 6:  Bl=Blobs(R);
  Step 7:  If( i>temp) Buffer the results of routine Blobs in array Bl2;
  Step 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 for Implementation of Single Camera Model for Video Surveillance on Leader Node.

ALGORITHM 3

```
Algorithm Node n(i, B, M)
{
   Step1:xn=no of iterations node n has to execute.
   Step2:while(a<=xn)
         {
   Step3: If( i % N==0)  B=Update Background(past N frames)
   Step4: R=roi(current frame,B);
   Step5: Bl=Blobs(R);
   Step6: If(node==2)  M=match(Bl);
          else  Buffer the results of routine Blobs in array Bl2;
    Step7:  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.
                         }
   }
    Step8: If(node<=4)  Pass results (M) to next node (node+1)
}
```

Algorithm for Implementation of Single Camera Model for Video Surveillance on Other Node.

4.3.3 Problems associated with Cluster 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.

Observing the factors there was possibly a need for an investigation of Video Surveillance algorithm on an on chip multicore processor like CBE, who had bus and nine cores itself on a chip offering a EIB bus speeds at higher speeds.

CHAPTER 5
VIDEO SURVEILLANCE APPLICATION ON CBE

5.1 Introduction

CBE offers a platform with heterogeneous processing capabilities, which forms a suitable architecture for various multimedia and scientific applications. It provides eight cores of SPEs which form accelerators, and provide a platform for applying various parallelization schemes for performance improvement. We provide some review of research done in video processing on CBE in the next section.

5.2 Review of Video Processing Applications on CBE

In [23], 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, Salience 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. [24] presents a scheme for parallelizing video processing and retrieving model on CBE. A multilevel parallel partition schema of video processing is suggested in his work. In his approach workloads were partitioned namely Service, Streaming models which form the Video Processing and Retrieval (VPR) framework, and are mapped on nine cores of CBE is presented. In this schema of parallel

partition, intensive computation workloads are partitioned and distributed by PPE to the SPE cores, which perform their part of the workloads and send the computation results to PPE. In the implementation of 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. 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. These computations were done in parallel among PPE and SPE processors viewing them 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.

In [25], Azevodo 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 [26], Park et al. proposed an approach for parallelizing X264 encoding algorithm of H.264 encoding scheme. They have proposed and implemented a pipelining model for parallelizing application. The algorithm was partitioned into three sections two for frame data processing and one for macro-block processing. In their implementation, a

frame is broken into blocks in which the encoding for each block was done in pipelined fashion along with maintaining data dependency between processing of blocks. PPE was responsible all data transfers and synchronization in processing of frame among SPEs.

In [27], Marcenaro et al. proposes a distributed architecture for multimedia surveillance. In their work, they have decomposed surveillance functionalities like segmentation and tracking into a set of modules among set of physical processing units structured into a distributed (using Java threads), heterogeneous, intelligent hierarchical surveillance network.

5.3 Study of Algorithm for parallelization on CBE

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 moreover to exploit the accelerators of CBE (SPE), we schedule Background modeling routine on PPE and others on the SPE's. In this scheduling we optimize by pre calculating background before its use for further iterations which run on SPE, and hence hiding the I/O overhead with execution of video frames. For simplicity purpose we had carried out our operations on images by converting them

onto grayscale and black and white for processing. We can convert an RGB image into grayscale by analyzing its intensities, or modifying the color map according to intensity.

We can use the standard NTSC conversion formula that is used for calculating the effective luminance of a pixel:

$$\text{Intensity} = 0.2989*\text{red} + 0.5870*\text{green} + 0.1140*\text{blue}.$$

The crucial aspect for the implementation was limited storage capacity of SPE (256 KB). So, 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 (amount maximum 16 KB) repeatedly 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 and hence all those DMA operations are done in parallel by SPEs following with computational operations which are performed later. Thus, by this approach we could bring out data parallel programming amongst the SPE's. In this approach we load a portion of image into local store of SPU by performing DMA. Figure 5.1 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 [28] ratio of algorithm. To address this we use double buffering scheme by which DMA latency can be hidden up to some extent.

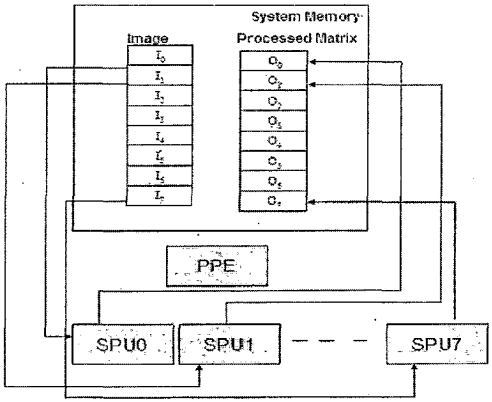


Figure 5.1: A sample image which is processed, each SPU processes its portion of image and achieving data parallelism.

Apart from above issues, we face other challenges 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 5.2 illustrates our approach for parallelization.

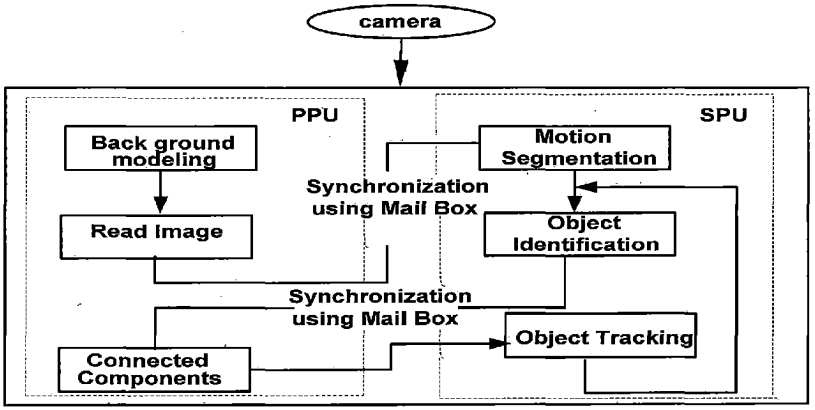


Figure 5.2: Parallelization approach used to implement Video Surveillance on CBE.

5.4 Algorithm

In this section we present the parallel algorithm used for implementation on CBE. The algorithm described below shows various routines mapped on to different cores of CBE. In the PPU side, we execute Background routine for every N frames, wait until SPUs finishes processing a frame and calculate connected components from an image and iterate upon until it finishes processing required number of frames. On the SPU side, we execute Segmentation routine (roi) and notify PPU through mail box for reading next frame so as SPU could later perform DMA operation to get next frame into its local store, we then run object identification routines (blobs), notify PPU through mailbox for calculation of connected components and finally track the objects found and iterate upon until it finishes processing required number of frames.

In overall implementation of our algorithm on the CBE, we had tried 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.

In the overall approach used, on comparison with the cluster implementation, the communicational delays between different cores was less as CBE offers a high speed

EIB (as discussed in the above chapter). Moreover as each SPE unit works on portion of an image some challenges which we face during cluster implementation were met here, as in case of CBE we could perform as many DMA operations to get image from DRAM and handle this issue.The algorithm can be summarized in following steps:

Algorithm 4

1. Read image into a matrix and evaluate Background.
2. Perform Background modeling for next
N frames until SPU finishes reading image matrix.
3. Perform Background Modeling until SPUs call PPU for either reading image or finding connected components.
4. Find connected components in an image.
5. Goto Step2 until all images are processed.
6. End.

Algorithm for Implementation of for Video Surveillance on PPE.

Algorithm 5

1. Read image matrix.
2. Perform Motion Segmentation.
3. Send signal to PPU through mailboxes, to start reading next image.
3. Perform Object Identification.
4. Send signal to PPU through mailboxes, for evaluating connected components in an image at PPU side.
5. Perform Object Tracking
6. Goto Step2 until all images are processed.
7. End.

Algorithm for Implementation of for Video Surveillance on SPE.

CHAPTER 6
RESULTS AND DISCUSSIONS

6.1 Experimental results on CBE

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 iterations (frames) processed. The speed up is 43.1 times faster compared to implementation of Video Surveillance on a Windows based Pentium workstation. Table 6 shows comparison between both approaches. Figure 6.1 demonstrates the execution time of Video Surveillance Algorithm based on number of iterations run.

	Processing time taken /16 frames (refresh rate of N=10)
Implementation on Windows based Pentium platform	5203.1 milliseconds
Simulated time using Cell SDK 2.0	120.7 milliseconds
SPEED UP	**43.1077**

Table 6.1: Speed up on Cell in comparison with Pentium platforms.

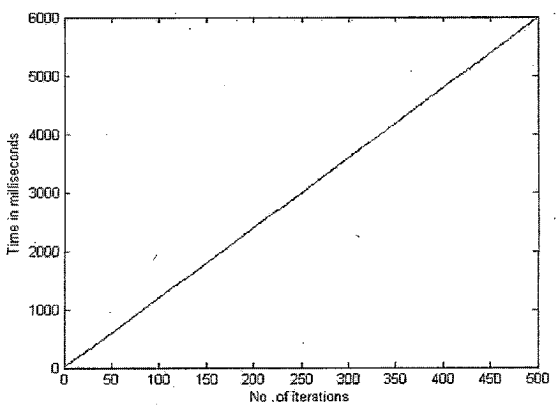


Figure 6.1: Execution time of Video Surveillance on CBE Simulator (in terms of number of iterations).

The above graph demonstrates execution times of application run for 500 iterations (processing nearly 800 images) in milliseconds. The execution times for all iterations are nearly same as the calculation of background (which is the most I/O intensive routine is hid with the execution of computational routines on SPEs. Thus with these optimizations, we could observe that with the exception of first iteration, execution times of all other iterations are nearly a same value. Hence, from the above results we could observe a linear increase in execution time and constant speed up with increase in amount of workloads.

Comparison of Results varying number of SPEs

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

Table 6.2: Execution times and speed up of Video Surveillance Algorithm on CBE on varying number of SPEs used.

The above table 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.

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 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.

6.2 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. Table 8 shows the speed up comparison between CBE and computer cluster approaches. The speed up 19.386 is times faster compared to implementation of Video Surveillance on a cluster.

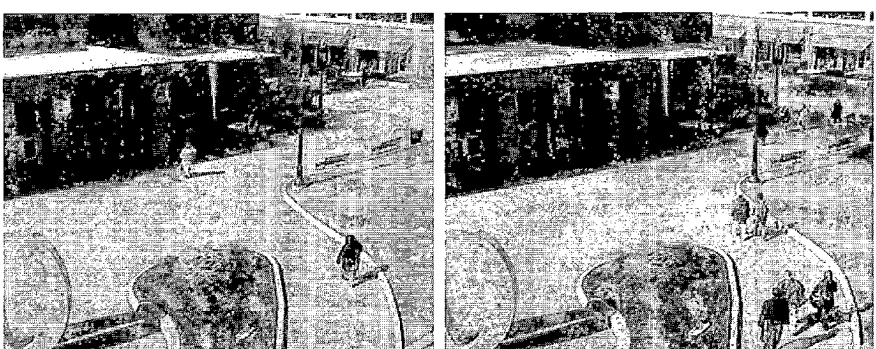
	Processing time taken /16 frames (refresh rate of N=10)
Implementation on computer cluster	2340 milliseconds
Simulated time using Cell SDK 2.0	120.7 milliseconds
SPEED UP	19.386

Table 6.3: Speed up on Cell in comparison with cluster implementation of the algorithm.

The factors such as high speed EIB bus, on chip multicore processing ability of CBE have helped the application achieve marked performance over the cluster implementation. Moreover, the transmission of images which were a substantial part of delay in our previous work, here they have been simply altered as all cores reside on same chip and address a same DRAM.

6.3 Test Data Used

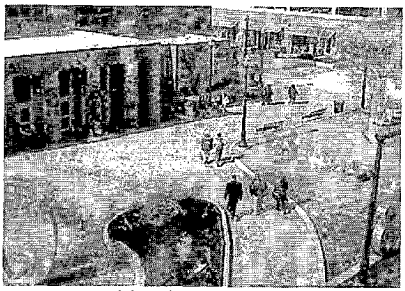
The test data used were sample video frames, where Video Surveillance found interesting objects, and tracks them throughout the video frames. Figure 6.2 shows some of the sample video frames used.



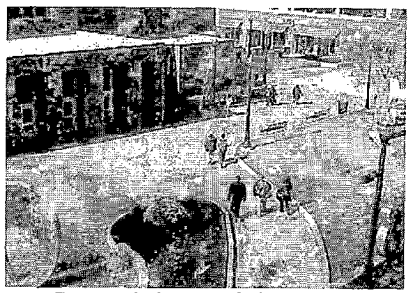
Video frame number 0 Video frame number 834

Figure 6.2: Sample images used for implementation.

The above data set was used and converted to grayscale and then processing was done on black and white images. Figure 6.3 shows image used, background image, region of interest calculated and objects detected by the implementation of Video Surveillance. From the below figure, we could also infer the accuracy of implementation.

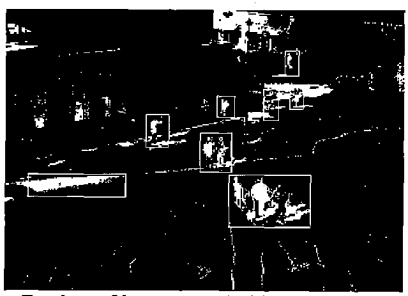


Video frame no 1198



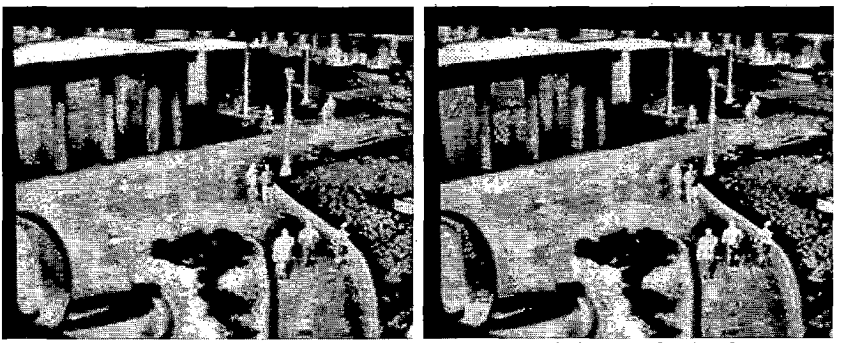
Gray scale image of video frame

Background image

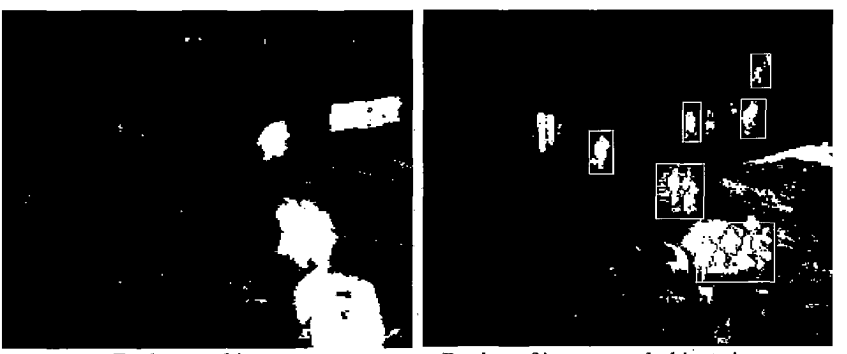


Region of interest and objects in scene

Figure 6.3 (a): Objects detected in video frames of data set 1.
(data set 1: Visible camera bitmap images)



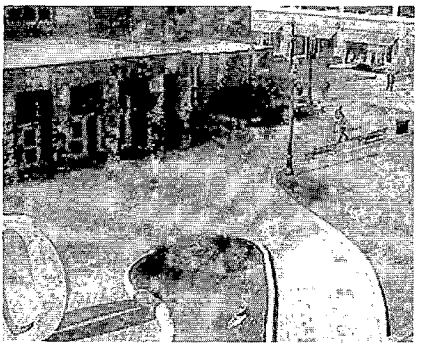
Video frame no 1301 | Gray scale image of video frame



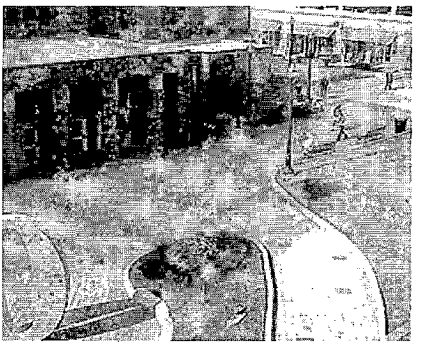
Background image | Region of interest and objects in scene

Figure 6.3(b): Objects detected in video frames of data set 2.

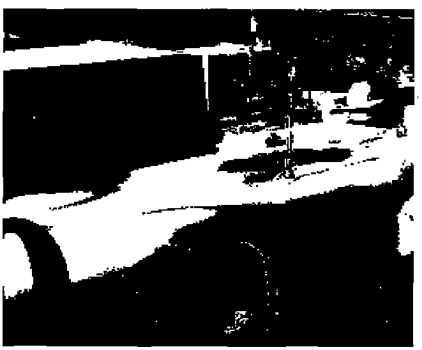
(data set 2: Infrared camera bitmap images)



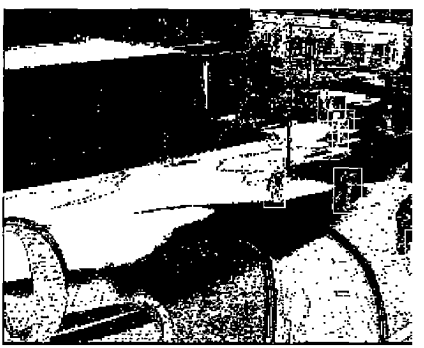
Video frame no 2100



Gray scale image of video frame



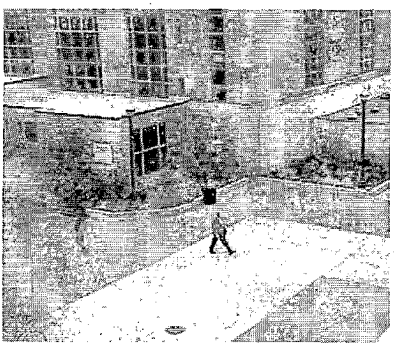
Background image



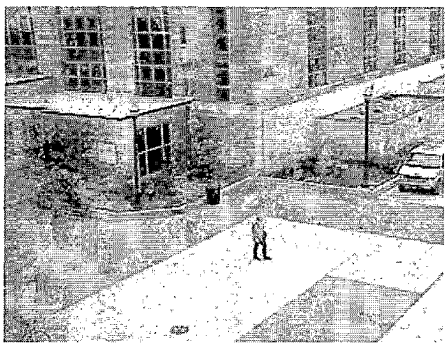
Region of interest and objects in scene

Figure 6.3(c): Objects detected in video frames of data set 3.

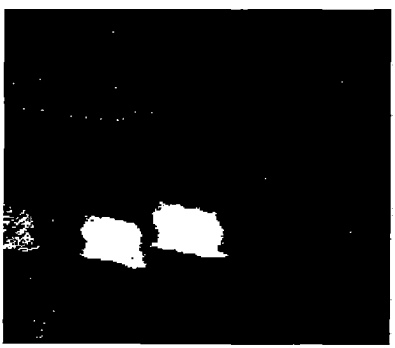
(data set 3: Visible camera bitmap images)



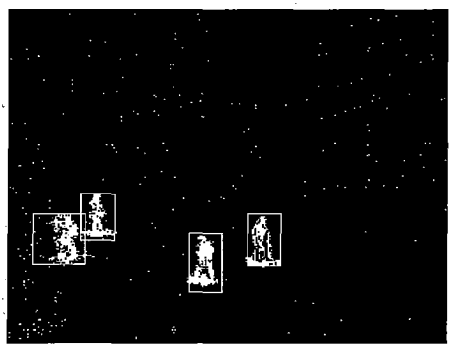
Video frame no 264



Gray scale image of video frame



Background image



Region of interest and objects in scene

Figure 6.3(d): Objects detected in video frames of data set 4.

(data set 4: Visible camera bitmap images)

CHAPTER 7
CONCLUSIONS AND SCOPE FOR FUTURE WORK

In this work, we have proposed a parallelization model for implementation of Video Surveillance application on the STI Cell platform. This model could be efficaciously used for applications similar to Video Surveillance whose tasks entail high intensive data and computational requirements with application specific modifications.

The implementation of a video surveillance algorithm on Cell 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. We had achieved a marked improvement of 43x over the non distributed version and 19x over the cluster implementation. Also, the result images illustrate the accuracy of the implementation on the STI Cell platform.

Issues such as limited storage of SPE and optimizing the DMA operations were solved. A data parallel programming paradigm was used to exploit the accelerator cores of CBE, so as the storage problem of SPE could be solved inherently using this approach. The SPU intrinsic libraries available on the platform could have been exploited, so as the vectorization of the code could be done both at application and hardware level. Moreover, the scalability issues such as exploiting the dual thread support of PPE for better management of I/O and SPE threads and image processing part to work on various formats of images could have been implemented in our work.

Future works in this course would be to experiment with an actual Cell implementation and explore the performance of the platform in audio and 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 [29].

Another suggestion for future works could be to experiment the multi-camera surveillance scenario on real grid system, where a grid and the sensors deployed at various sites would communicate via the internet. A model for implementation of a media tracking system of vehicle plates using grid as computational platform could be viewed in [30].

The other way we could extend our work is by introducing some fast techniques of morphological operations in Video Surveillance algorithm to increase its performance at image processing level. Methods for Fast Morphological Image Transforms are provided in [31]. Another important issue to address in this case would be scalability and security aspects.

REFERENCES

[1]T. P. Chen, H. Houssecker, A. Bovyrin, 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, pages: 109-118, May 2005.

[2]"Sun Reference Architecture for Video Surveillance ", White Paper, Sun Microsystems, Inc. 2007. http://www.sun.com/videosurveillance (last accessed on 2 June 2008).

[3]"Toshiba to demonstrate prototype of new "SpursEngine ™" processor at CEATEC JAPAN 2007", http://www.toshiba.co.jp/about/press/2007_09/pr2001.htm (last accessed on 22 May 2008).

[4]G Kola, T Kosar, M Livny. "A fully automated fault-tolerant system for distributed video processing and off-site replication", Proceedings of the 14th international workshop on Network and operating systems support for digital audio and video Cork, pages: 122 – 126, June 2004.

[5] R. Collins, A. Lipton, T. Kanade, H. Fujiyoshi, D. Duggins, Y. Tsin, 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.

[6] I. Pavlidis, V.Morellas, P. Tsiamyrtzis, and S. Harp. "Urban surveillance systems: from the laboratory to the commercial world". Proceedings of IEEE, 89, (10), pages:.1478-1489, 2001.

[7] 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.

[8] 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 18 May 2008).

[9] M. Valera, S.A. Velastin, "Intelligent distributed surveillance systems: a review". In Image and Signal Processing, IEEE Proceedings, vol. 152, pages: 192 – 204, April 2005.

[10] A. Mahalanobis, J. Cannon, R. Stanfill, R. Muise, and M.Shah, "Network Video Image Processing for Security, Surveillance, and Situational Awareness". Keynote at SPIE conference of Digital Wireless Communication VI, Orlando, 2004, April 12-13.

[11] [redacted], S.P. Kumar, B.A. Hamilton, "Sensor networks: evolution, opportunities, and challenges".Proceedings of the IEEE, vol. 91, pages: 1247- 1256, 2003

[12] CBEA Programming Tutorial, IBM Systems & Technology Group, Version 2, December 2006

[13]Synergistic Processing In Cell's Multicore Architecture - Michael Gschwind, H. Peter Hofstee, Brian Flachs, Martin Hopkins, IBM, Yukio Watanabe, Toshiba, Takeshi Yamazaki, Sony Computer Entertainment IEEE Computer Society, pages:10-24, 2006

[14] 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.

[15] M. Gschwind, "The Cell Broadband Engine: Exploiting Multiple Levels of Parallelism in a Chip Multiprocessor", International Journal of Parallel Programming, vol. 35, no. 3, pages 233-262, June 2007.

[16] M. Gschwind, D.Erb, S. Manning, M. Nutter,"An Open-source Environment for Cell Broadband Engine System Software", IEEE Computer Society, vol.40, no. 6, pages 37-47, June 2007.

[17] T. Kanade, R. Collins, A. Lipton, P. Anandan, and P. Burt. Cooperative multisensor video surveillance. In Proceedings of the 1997 DARPA Image Understanding Workshop, vol. 1, pages 3–10, May 1997.

[18] J.W. Davis ,_V.Sharma. " Fusion-Based Background- Subtraction using Contour Saliency", Computer Vision and Pattern recognition, pages: 20-26 June, 2005.

[19]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.

[20] S. Hiroyuki, Y. Teruhilo. "Characteristics of Loop Unrolling Effect : Software Pipelining and Memory Latency Hiding", Inn pages:ovative Architecture for Future Generation High-Performance Processors and Systems, IEEE, pages:63-72, January 2001.

[21]P. Czarnul A. Ciereszko, M. Fraczak "Towards Efficient Parallel Image Processing on Cluster Grids Using GIMP", LNCS, Springer Verlag Publication, vol. 3037, pages: 451-458, 2004.

[22]R. Buyya, J .Yu. "A Taxonomy of Workflow Management Systems for Grid Computing", Journal of Grid Computing, vol.3, nos 3-4, pages:171-200, 2005.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] L. Marcenaro, F. Oberti, G. L. Foresti, C.S. Regazzoni ," Distributed Architectures and Logical-Task Decomposition in Multimedia Surveillance Systems",IEEE Proceedings, vol. 89,no. 10, pages: 1419 – 1440, October 2001.

[28]Y.Yu,B.Krishnamachari,V.K.Prasanna. "Issues in Designing Middleware for Wireless Sensor Networks".IEEE Network, pages: 15-21, January/February 2004.

[29] P. Kumar, A. Mittal and P. Kumar. "A Multimodal Audio, Visible and Infrared Surveillance System (MAVISS)". In Proceedings of the 3rd IEEE International Conference on Intelligent Sensing and Information Processing (ICISIP), pages: 151-157, 2005.

[30] Z. B. Musa, J. Watada, " A Grid–Computing based Multi–Camera Tracking System for Vehicle Plate Recognition", Kybernetika--The Journal of the Czech Society for Cybernetics and Information Sciences, vol. 42,no. 4 , pages: 495–514, August 2006.

[31] 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.

PUBLICATIONS

P.V.Kumar, P.Kumar, A.Mittal, R.Dubey. "A Scheduling Architecture for Distributed Video Surveillance System", International Conference on Systemics, Cybernetics, Informatics, Hyderabad, January 2008, pages 104-112.

P.V.Kumar, P.Kumar, A.Mittal. "A Basic Video Surveillance Architecture on the STI Cell Broadband Engine Processor", National Conference on Research and Development in Hardware and Systems (CSI-RDHS 2008), Kolkata, June 2008.

P.V.Kumar, P.Kumar, A.Mittal. "An Implementation of a Basic Video Surveillance Algorithm on the STI Cell Broadband Engine", International Conference on Information Processing, Bangalore, August 2008.

CENTRAL LIBRARY
ACC.No.
Date
I.I.T, ROORKEE

GLOSSARY

ster is a group of coupled computers that work together closely so that
ots they can be viewed as though they are a single computer. The
of a cluster are commonly, but not always, connected to each other
fast local area networks. Clusters are usually deployed to improve
nance and/or availability over that provided by a single computer, while
ually being much more cost-effective than single computers of comparable speed
vailability.

rid

A computational grid is a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities. Grid computing is distinguished from typical cluster computing systems that grids tend to be more loosely coupled, heterogeneous, and geographically dispersed. Also, while a computing grid may be dedicated to a specialized application, it is often constructed with the aid of general purpose grid software libraries and middleware.