International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:10, No:3, 2016

General Purpose Graphic Processing Units Based Real Time Video Tracking System

Mallikarjuna Rao Gundavarapu, Ch. Mallikarjuna Rao, K. Anuradha Bai

Abstract**—**Real Time Video Tracking is a challenging task for computing professionals. The performance of video tracking techniques is greatly affected by background detection and elimination process. Local regions of the image frame contain vital information of background and foreground. However, pixel-level processing of local regions consumes a good amount of computational time and memory space by traditional approaches. In our approach we have explored the concurrent computational ability of General Purpose Graphic Processing Units (GPGPU) to address this problem. The Gaussian Mixture Model (GMM) with adaptive weighted kernels is used for detecting the background. The weights of the kernel are influenced by local regions and are updated by inter-frame variations of these corresponding regions. The proposed system has been tested with GPU devices such as GeForce GTX 280, GeForce GTX 280 and Quadro K2000. The results are encouraging with maximum speed up 10X compared to sequential approach.

Keywords**—**Connected components, Embrace threads, Local weighted kernel, Structuring element.

I.INTRODUCTION

VISION applications such as, Automated Video Surveillance (VS), autonomous vehicle control, gesture interaction based on dynamic facial features pose a challenge to researchers due to their computational intensity and response times. High speed real-time image processing on video frames is difficult even with the most powerful modern CPUs. As the computational capabilities at software level using traditional serialization approaches have reached their limitations, the new trend is to explore parallelism to meet the growing demands for these applications. Distributed computing with multiple machines was suggested by [1]-[3]. However, the cost of resources and complexity of the required implementation make the researches look to alternatives in multi-core computing [4] and GPGPU computing [5].

Current chip technology favors increasing performance using multiple-core processors on a single chip, which has refocused attention on parallel algorithms balancing concurrency and communication. Recently, multi-core and multi-threaded architectures, along with general purpose Graphics Processing Units (GPU's), have pushed the limits of real-time multimedia and graphics applications that are based on the stream model of computation

Automated Video Surveillance (VS) involves many computer vision algorithms like object detection and tracking [6]-[9], human activity recognition [10] and tracking performance evaluation techniques [11]. There has been extensive research in past two decades in this area which can be reviewed in [12], [13] and computer vision systems have been developed that are used in commercial systems. Rule-based framework [14] has been proposed for behavior and activity detection in traffic videos obtained from stationary video cameras. Moving targets are segmented from the images and tracked in real time. Some successful applications for instance are complete solution for vehicle and traffic surveillance [15], tracking and counting people [16], detecting left luggage [17], face or other biometric recognition etc. VS algorithms represent a class of problems that are both computationally intensive and bandwidth intensive [18].

Mallikarjuna Rao Gundavarapu is with the Department of Computer Engineering, Gokaraju Rangaraju Institute of Engineering and Technology, India (e-mail: raogm333@gmail.com).

Research efforts both from academia and industry have been made to exploit Cell multi-core architecture for applications related to video processing, retrieval and compression [19]. There are quite a number of papers related to parallelizing video encoding algorithms for different standards like JPEG 2000 [20] or H.264 [21]. In [22], details of a Cell BE implementation for different processing stages for compression algorithms are described. The novelty is that such hardware is now available off the shelf: clusters featuring GPGPU FPGA or even Cell processors are affordable to most of the user [23]. Similarly, GPU based motion estimation methods have been proposed using CUDA [24]. However, there has only been limited published work on parallelizing algorithms and operators for video surveillance algorithms optimized for the Cell BE or GPUs.

Stream processors perform extremely well on media/graphics applications with large data sets requiring the execution of similar operation on their data elements such as vector processing applications by incorporating SIMD execution units. The IBM Cell BE processor is a good example of the stream model of computation.

II.GENERAL PURPOSE COMPUTING USING GPU'S

Modern programmable graphics hardware contains powerful co-processors called GPU (Graphics Processing Unit) and is suitable for video analysis in real- time vision systems. While significant acceleration over standard CPU implementations is obtained by exploiting parallelism provided by modern programmable graphics hardware, the CPU is freed up to run other computations parallel. It works on both ATI (Array Technology Incorporated) and NVIDIA graphic cards. Recently, graphic processing units have evolved into an extremely powerful computational resource. For example, The NVIDIA GeForce GTX 280 is built on a 65nm process, with 240 processing cores running at 602 MHz, and

1GB of GDDR3 memory at 1.1GHz running through a 512-bit memory bus. Its processing power is theoretically 933GFLOPS, billions of floating-point operations per second, in other words. As a comparison, the quad-core 3GHz Intel Xeon CPU operates roughly 96 GFLOPS. The annual computation growth rate of GPUs is approximately up to 2.3x. In contrast to this, that of CPUs is 1.4x. At the same time, GPU is becoming cheaper and cheaper. As a result, there is strong desire to use GPUs as alternative computational platforms for acceleration of computational intensive tasks beyond the domain of graphics applications.

CUDA (Compute Unified Device Architecture) is a parallel computing architecture developed by NVIDIA for massively parallel high-performance computing [25]. CUDA supports c-like programming to perform operations on the core of GPU. In the CUDA programming framework, GPU acts as co-processor to the CPU. The GPU has its own DRAM, referred to as device memory, and support the parallel execution of a very high number of threads. More precisely, data-parallel portions of an application are executed on the device as kernels which run in parallel on many threads. Since the computation is local to the machine, communication delays will be minimized. Interaction between hosts and device interaction is given by the following Fig. 1. The device cores segregated into Grids and Blocks and thus enabling parallel computation.

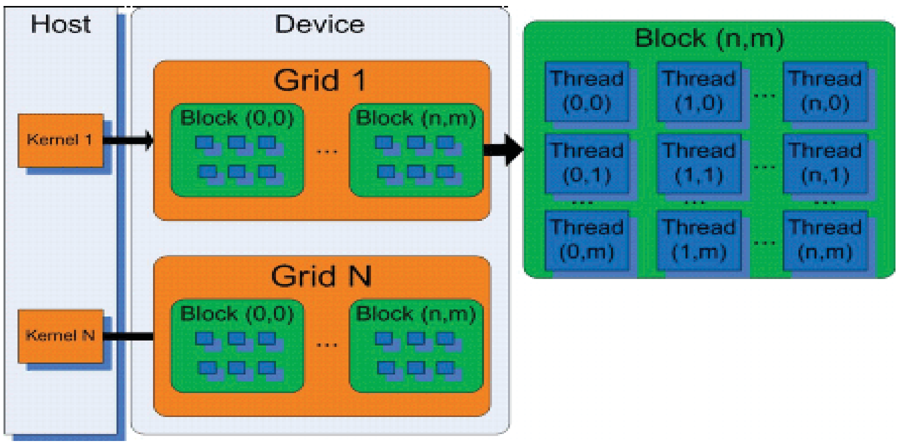


Fig. 1 Segmentation of data intensive program into computational blocks

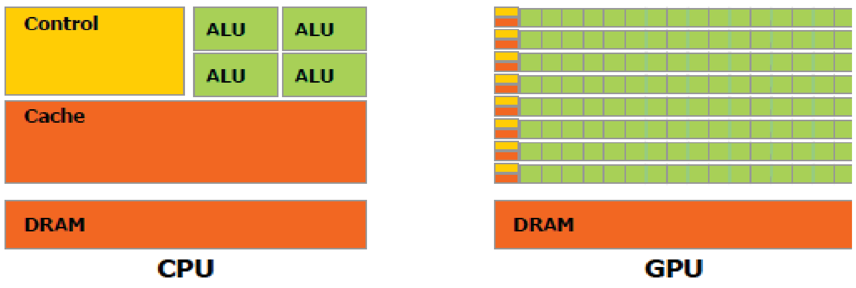


Fig. 2 Chip design of CPU and GPU

A. GPGPU Computation

CPU's are developed and optimized for sequential serial processing while GPU’s support parallel processing through data streaming, as shown in Fig. 2. A serial processor, based on the Von Neumann architecture executes instructions sequentially. Each instruction is fetched and executed by the CPU one at a time. A stream processor on the other hand executes a function (kernel) on a set of input data (stream) simultaneously. The input elements are passed into the kernel and processed independently without dependencies among other elements. This allows the program to be executed in a parallel fashion. The following c-program is performing cubes of 100 element array. The computation is performed in cores. Kernel call cube_array was made by **cube_array <<< Grid , Block >>> (arr , 100)** from main function. Grid contains 10 Blocks and each block contain 10 threads. **_global__ void cube_array** (float *a, int N) instructs the compiler to generate code for cores of the device. Fig. 3 clearly revels the structure of computation.

```
_global__ void cube_array (float *a, int N) {
int idx = blockIdx .x * blockDim .x + threadIdx .x;
if ( idx < N) {
a[idx] = a[idx] * a[idx ] *a[idx];
}
}

int main (void ) {
...
dim3 Block (10 , 10);
11 dim3 Grid (1) ;
12 // allocate memory for array dynamically in Host
14// initialize arry with random numbers
14 // copy the array from Host memory to device memory
15 cube_array <<< Grid , Block >>> (arr , 100) ;
16 // free device memory..
17 // free host memory
18}
```

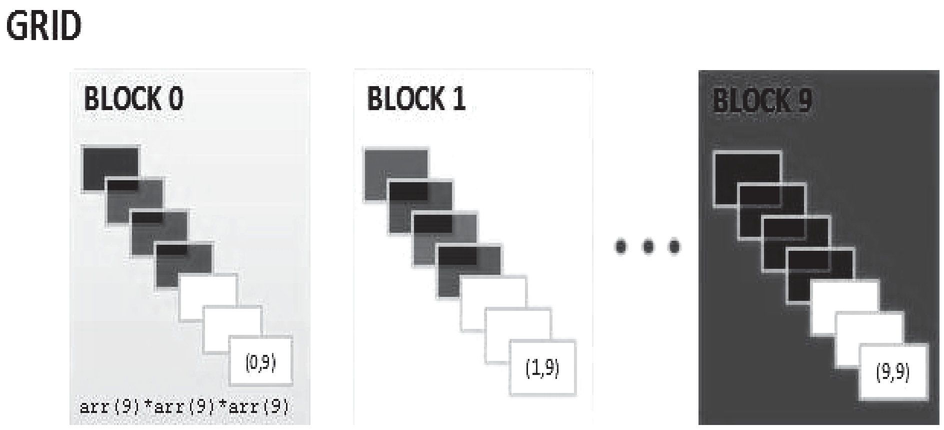


Fig. 3 Device Execution Unit: 1-GRID, 10 Blocks and 10 Threads/block

III. Proposed Video Tracking System

In general, the portion of the image region of the frame corresponding to the real target of the object has to provide highest similarity and therefore unambiguously make itself different from the other object of the frame. However, the cluttered background may result in other objects of the frame generating comparable similarity and hence confuse the tracking system. The local maxima produced by the interference of the background objects make design of tracking algorithms much more complex and increases the computational requirements.

A. Our Proposed Real Time Video Tracking System

We have gone through the following sequential image processing algorithms and developed its parallel version with different approaches in core i5 & i7 processors and found good results with variable numbers of threads passed as input parameters. We focused on the most promising and well-supported techniques with an emphasis on image processing application. Fig. 4 reveals the details of our proposed video tracking system.

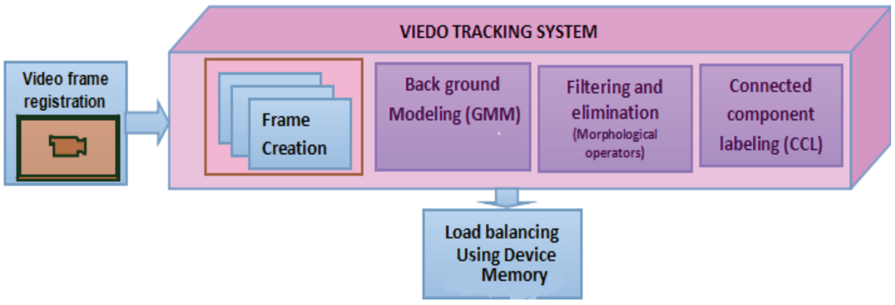

VIEDO TRACKING SYSTEM
Video frame registration
Frame Creation
Back ground Modeling (GMM)
Filtering and elimination (Morphological operators)
Connected component labeling (CCL)
Load balancing Using Device Memory

Fig. 4 Real-Time Video Tracking System

The following steps are used for developing a real-time video tracking system.

1. Video Frame Registration

This step is required whenever the camera is mounted on a moving platform for example in airborne video surveillance. First among the three steps in the registration process involves motion estimation with a block matching algorithm. Next, a least-square fit is used to reduce the field of motion vectors to an affine transformation, and this transformation is accumulated with the previous transformation to produce the cumulative transformation over the video stream. Finally, the input video frame is transformed according to this cumulative transformation to produce the registered video frame. The most computational part of this process involves the first step of motion estimation using block matching. Thus, parallel implementation of block matching algorithm is highly desirable.

2. Background Modeling and Detection of Foreground Regions

This step forms the bulk of computation which varies depending on the complexity and robustness of the algorithm used. We plan to implement a popular technique called

Mixture of Gaussian (MOG) background modeling [26] and a state-of-the-art Flux tensor motion estimation algorithm for this purpose. Compute intensive characteristic and available parallelism in these algorithms make them a suitable candidate for parallelizing on multi-core processors. If enough numbers of threads are created, then SPMD (single program multiple data) is quite convenient for exploring the parallelism.

3. Consolidation, Filtering and Elimination using Binary Morphology

With a suitable threshold operation, a binary image or mask corresponding to moving regions is created. Morphological operations "opening" and "closing" are applied to the objects [27]. There is high degree of parallelism in this step, and it is computationally expensive step, as these operators have to be applied in several passes on the whole image. Therefore, faster parallel implementation is not only intuitive but indispensable for real-time processing.

4. Connected Component Labeling

Connected components labeling is one of the most time consuming operations. Several authors proposed variant approaches to reduce the time requirement. In this process each region or blob must be uniquely labeled, in order to uniquely characterize the object pixels underlying each blob. Since there is spatial dependency at every pixel, it is not straightforward to parallelize it.

5. Object Statistics and Tracking

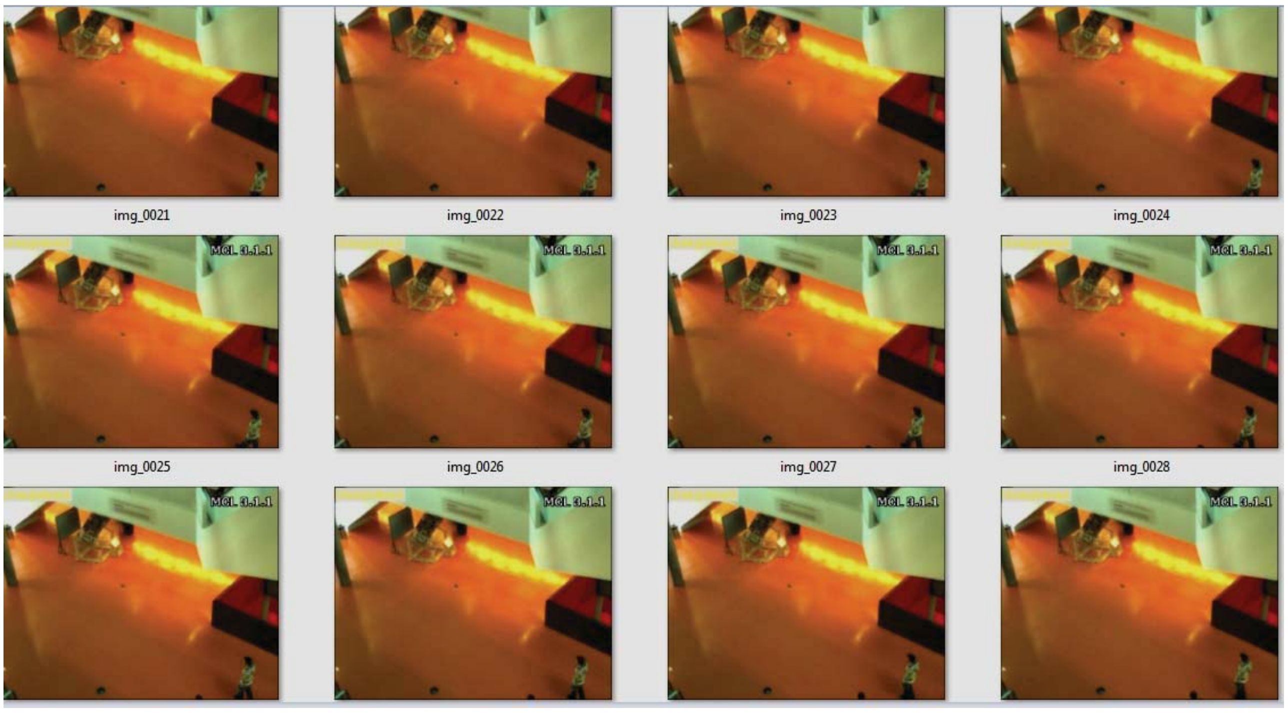
Object statistics (bounding box, centroid, area, perimeter etc.) for each blob/object is calculated and used for tracking them over a sequence of frames for trajectory analysis. There are various approaches for parallelizing an algorithm on a heterogeneous multi-core processor, depending on the data partitioning strategy. Bounding box approach is used in our paper. Consequently, different levels of optimization can be achieved for extracting parallelism.

6. Load Distribution between Processors

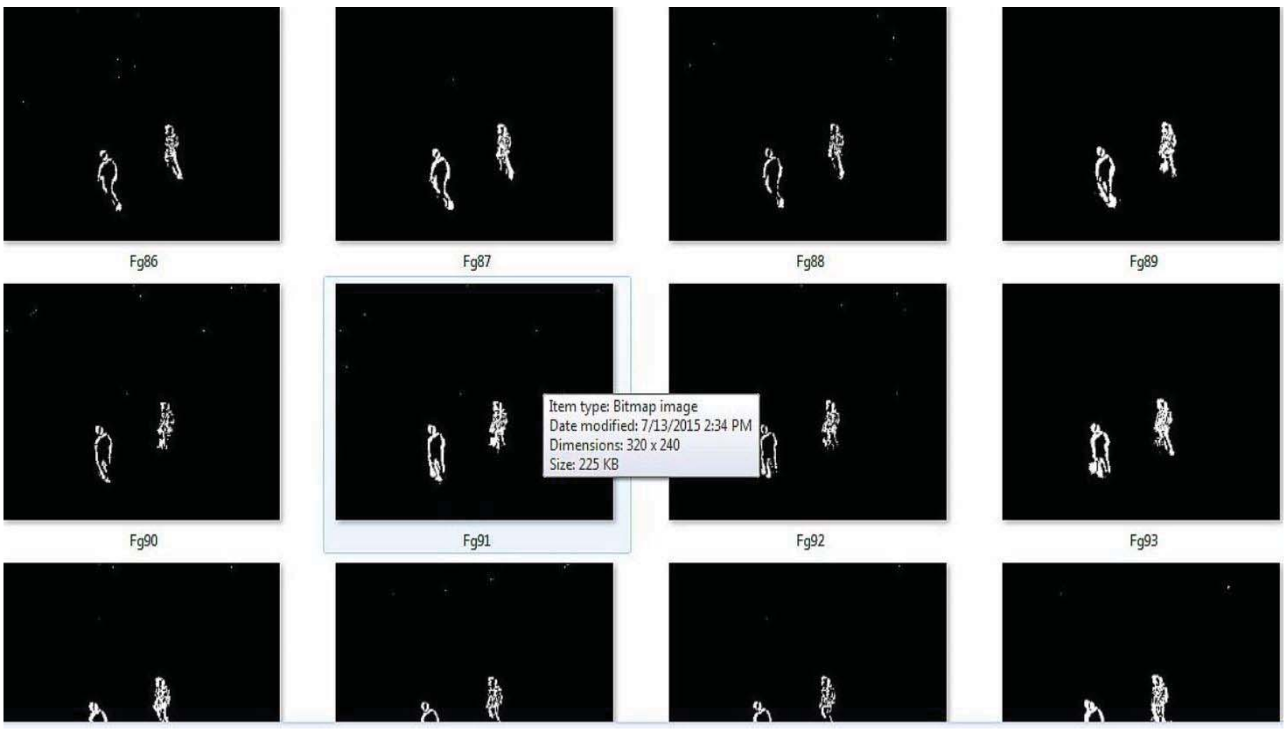
The main step of these algorithms is to determine the number of tiles to be generated. The number of tiles corresponds to the amount of threads. If only a thread exists, the computation is just sequential computation. With two or more threads then the image is divided into distinct areas. Each thread is responsible for processing the pixels included in its tile and to execute different tasks but considering maintaining synchronization between all the processor, otherwise there will be the situation of deadlock between processors.

B. Performance Analysis

In order to do performance analysis, comparisons will be made by taking images of differing sizes. Comparisons tables and diagrams will be made based upon the outcomes of the experimental results. Different image quality metrics and other parallel computing parameters such as speedup, efficiency, serial time, parallel time, response time and resource utilization will be considered to evaluate the performance of proposed algorithm. Fig. 5 reveals the results of various steps employed, starting from RGB frames through to bounding box-based tracking of selected objects.

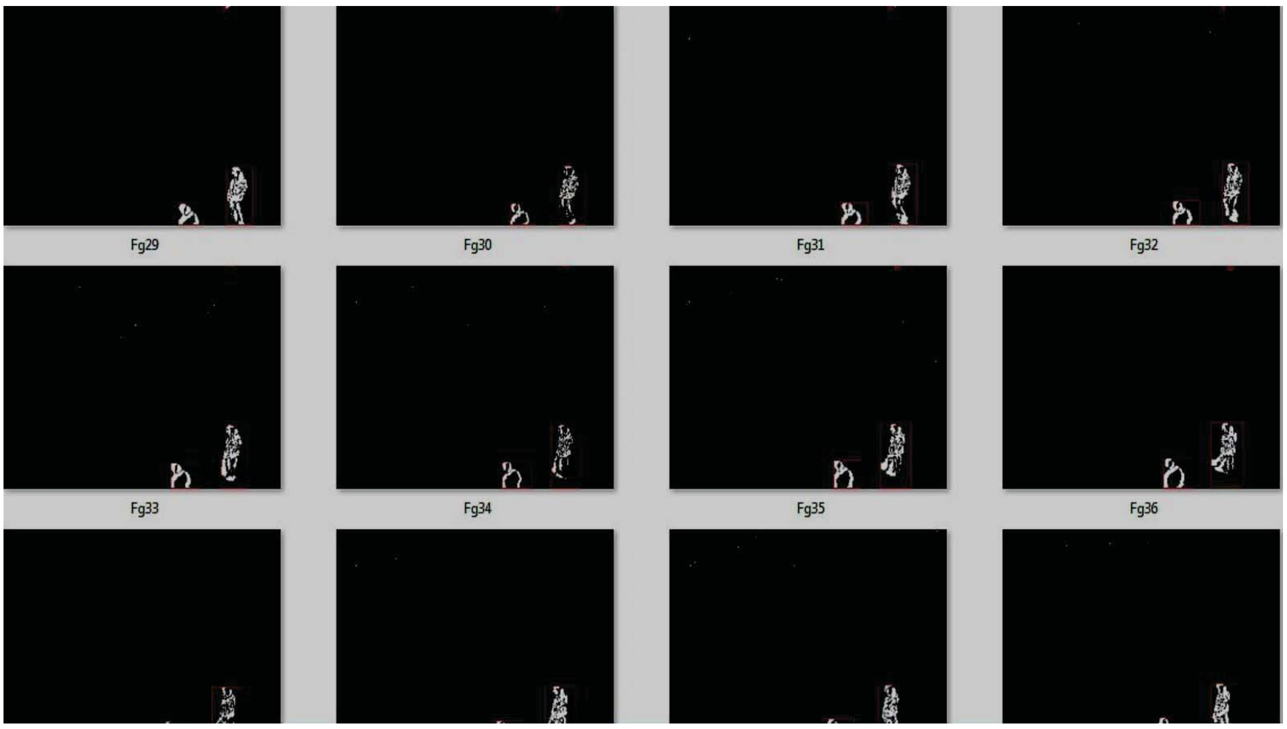


(a)

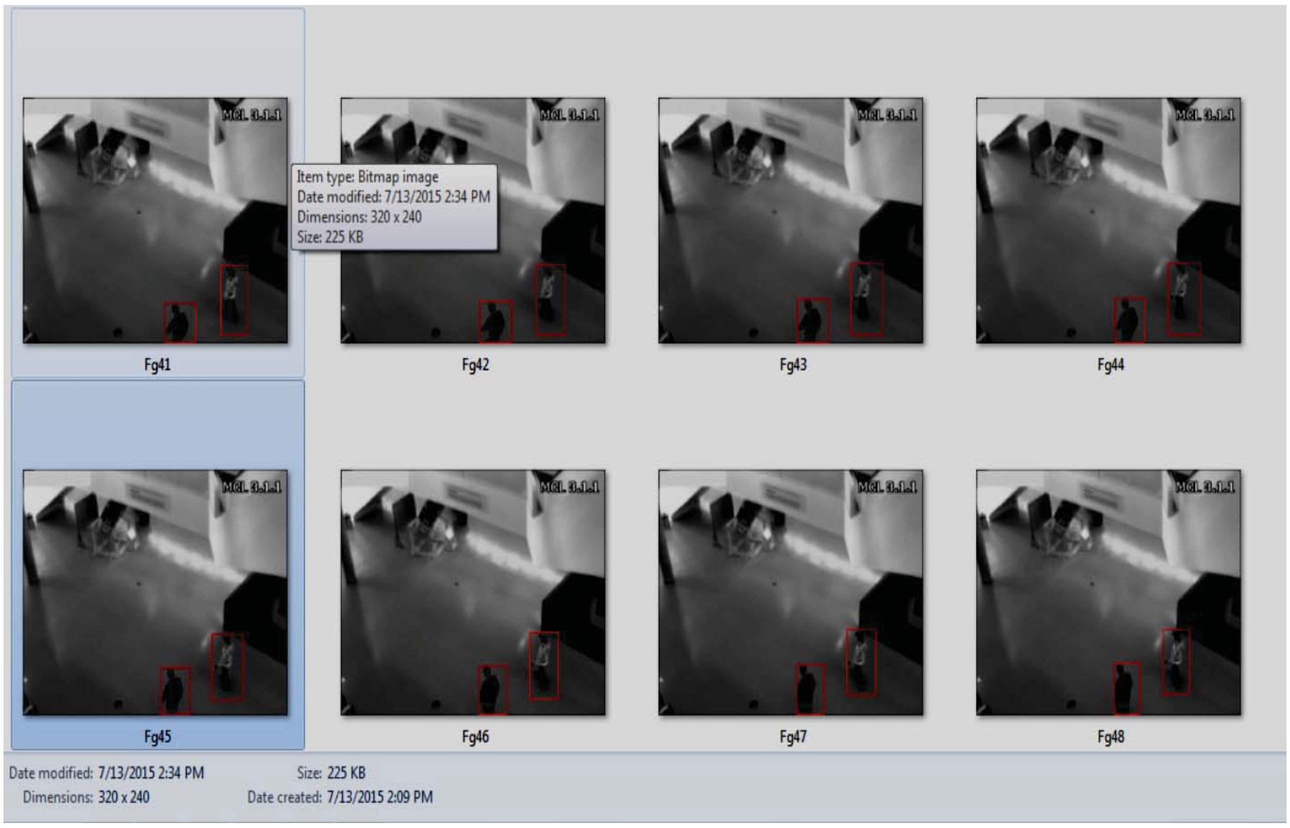


Item type: Bitmap image
Date modified: 7/13/2015 2:34 PM
Dimensions: 320 x 240
Size: 225 KB

(b)



(c)



Item type: Bitmap image
Date modified: 7/13/2015 2:34 PM
Dimensions: 320 x 240
Size: 225 KB

Date modified: 7/13/2015 2:34 PM Size: 225 KB
Dimensions: 320 x 240 Date created: 7/13/2015 2:09 PM

(d)

Fig. 5 Video Tracking System Responses (a) Original RGB Frames, (b) Morphological operation: Erosion, (c) Morphological operation: Dilation, (d) Bounding boxes using Connected Components

IV. Experimental Approach

The parallel implementation of the Real Time Video Tracking system workload was executed on three NVIDIA GPUs, the first GPU used is the GeForce GTX 280 on board a 3.2 GHz Intel Xeon machine with 1GB of RAM. The second one was the GeForce 8400 GS on board a 2 GHz Intel Centrino Duo machine. The third was Quadra K2000 with 384 cores on board Xeon workstation Z620.

The GTX 280 has a single precision floating point capability of 933 GFlops and a memory bandwidth of 141.7 GB/sec, it also has 1 GB of dedicated DDR3 video memory and consists of 30 multiprocessors with eight cores each, hence a total of 240 stream processors [28]. The 8400 GS has a memory bandwidth of 6.4 GB/sec and has two multiprocessors with eight cores each, i.e. 16 stream processors, single precision floating point capability of 28.8 GFlops and 128 MB of dedicated memory. It belongs to the compute capability 1.2. The NVIDIA Quadro K2000 offers the perfect blend of performance and the latest productivity enhancing technical innovations at a very reasonable cost for a wide range of leading professional applications. As well as, 2GB of GDDR5 GPU memory, 384 SMX CUDA parallel processing cores, the ability to drive up to four displays simultaneously, and full Shader Model 5 compatibility; all in a single slot form factor which requires no auxiliary power to deliver full performance.

The development environment used was Visual Studio 2010 and the CUDA profiler version 2.2 was used for profiling the CUDA implementation. The image sizes that have been used are 1600×1200, 1024×768, 640×480, 320×240 and 160×120.

TABLE I
Execution Time for GMM

Image Size	K2000 (ms)	GTX 280 (ms)	8400 GS (ms)
320×240	0.023	0.155	3.551
640×480	0.0494	0.393	17.585
1024×768	0.0817	0.632	31.866
1280×720	0.1905	0.728	42.856
1600×1200	0.392	1.127	94.772

TABLE II
Time for Morphological Operations

Image Size	K2000 (ms)	GTX 280 (ms)	8400 GS (ms)
320×240	0.023	0.0586	0.465
640×480	0.112	0.1254	1.75
1024×768	0.24	0.2429	3.61
1280×720	0.62	0.65	12
1600×1200	0.582	0.5625	11.7

The GMM, used for background modeling, has the kind of parallelism that is required for implementation on a GPU. As evident by Table I, the time of execution rises with the increase in image size and the amount of speedup achieved also increases, almost proportionately; this is due to the execution of a large number of threads that keeps the GPU busy.

Morphological image operations contribute a major portion of the computational expense of the AVS workload. In our approach we are able to drastically reduce their execution time compared to the sequential approach.

As evident by Table II, the time of execution for morphological operations increases with image size. Further it is evident from the table that K2000 GPU has taken less times compared to other (GTX 280, 8400 GS).

A. Memory Sharing between Device and Host

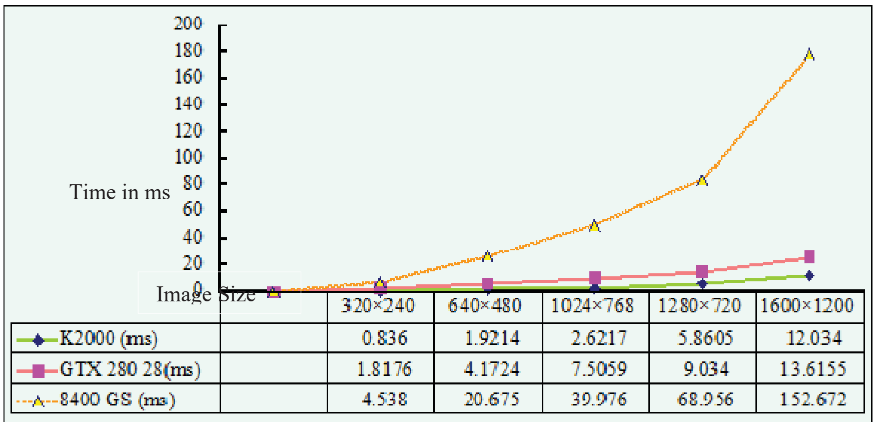
Shared memory was used to reduce the global memory accesses keeping in view the shared memory size (16 KB). As can be seen from the a total of 4 blocks (192×4 threads out of a maximum of 1024 threads) could be executed in parallel on a multiprocessor, giving an occupancy of 0.75. As a result of using K=4, all the global memory loads were coalesced due to lesser bank conflicts. The use of streaming reduced the memory copy overhead greatly in 8400 GS to the extent anticipated in other cases. This is due to the efficient memory copying in K2000 and GTX 280 - compute capability 1.3. Moreover, the use of texture memory and address clamp modes have reduced the percentage of divergent threads to <1%. On the 8400 GS also a significant speedup has been achieved. In each of the above kernels page-locked host memory (a feature of CUDA 2.2) has been used whenever only one memory read and write were involved which increased the memory throughput. Connected Component Labeling is extremely time consuming one compared to GMM and Morphological Operations. For each frame and for each pixel 4-neighbors have to be processed, and based on their value, a label will be made. Equivalencies have to be resolved among the labels. The Floyd-Warshall (F-W) algorithm has been used for this purpose. Execution timings across above GPUs are provided in Table III.

TABLE III
TIME FOR CONNECTED COMPONENT LABELING

Image Size	K2000 (ms)	GTX 280 (ms)	8400 GS (ms)
320×240	0.79	1.604	0.522
640×480	1.76	3.654	1.34
1024×768	2.3	6.631	4.5
1280×720	5.05	7.656	14.1
1600×1200	11.06	11.926	46.2

V. CONCLUSIONS

Architectures dedicated to video surveillance cost as much as millions of dollars at high range, thousands of dollars at medium range and hundreds of dollars at low range. Processing of high resolution frames with higher frame rate require high-end surveillance systems while low resolution frames with low frame rates are supported by medium and low range systems. We focus on K2000, GeForce GTX 280 and the 8400 GS (medium range) and low range (GPUs) for this purpose. Image size of 1024 x 768, 15 frames per second could be processed with 8400 GS and an image size of 640×480 close to 30 frames per second could be easily processed on all the above as shown in Fig. 6.



		320×240	640×480	1024×768	1280×720	1600×1200
◆ K2000 (ms)		0.836	1.9214	2.6217	5.8605	12.034
■ GTX 280 28(ms)		1.8176	4.1724	7.5059	9.034	13.6155
▲ 8400 GS (ms)		4.538	20.675	39.976	68.956	152.672

Fig. 6 Comparison of total time for image of different sizes

ACKNOWLEDGMENT

We are grateful to the management of Gokaraju Rangaraju Institute of Engineering and Technology for extending the facilities. We would like to convey our thanks Director Sri P.S Raju and our principal Sri Jandhyala N. Murthy. The author thanks TEQIP-II for the financial support.

REFERENCES

[1] Amazon Elastic Cloud Compute. http://aws.amazon.com/documentation/ June 2009.
[2] Android open source project: Designing for performance. http://developer.android.com/guide/practices/design/performance. Html, April 2009.
[3] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool. Speeded-up robust features (surf). Computer Vision and Image Understanding, 110(3):346–359, June 2008.
[4] D. Jacquet, F. Hasbani, P. Flatresse, R. Wilson, F. Arnaud, G. Cesana, T. Di Gilio, C. Lecocq, et al, "A 3 GHz Dual Core Processor ARM

Cortex TM -A9 in 28 nm UTBB FD-SOI CMOS with Ultra-Wide Voltage Range and Energy Efficiency Optimization", IEEE Journal of Solid-State Circuits, vol. 49, no. 4, pp. 812-826, April 2014.
[5] D. B. Kirk and W. mei W. Hwu, Programming Massively Parallel Processors: A Hands-on Approach (Applications of GPU Computing Series). Morgan Kaufmann, 2010.
[6] Rymut B, Kwolek B. Real-time multiview human pose tracking using GPU-accelerated particle swarm optimization, Concurrency and Computation: Practice and Experience, 2014. DOI: 10.1002/cpe.3329.
[7] Sinha, S.N., Frahm J.-M., Pollefeys M., and Genc Y. Feature tracking and matching in video using programmable graphics hardware. Machine Vision and Applications (MVA), 2007
[8] W. Niu, L. Jiao, D. Han, and Y. Wang. Real-time multi-person tracking in video surveillance. Proceedings of the Pacific Rim Multimedia Conference, 2:1144-1148, 2003.
[9] I. Haritaoglu, D. Harwood, and L. S. Davis. W4: Real-time surveillance of people and their activities. IEEE Transaction on Pattern Analysis and Machine Intelligence, 22:809-830, 2000.
[10] S. Hongeng, R. Nevatia, and F. Bremond. Video-based event recognition: activity representation and probabilistic recognition methods. Computer Vision I mage Understanding, 96(2):129-162, 2004.
[11] Nghiem A., Bremond F., Thonnat M., Ma R. New evaluation approach for video processing algorithms. Proceedings of the IEEE Workshop on Motion and Video Computing (WMVC07); Austin, Texas, USA. February 23–24 2007.
[12] W. Hu, T. Tan, L. Wang, and S. Maybank. A survey on visual surveillance of object motion and behaviors. IEEE Transactions on Systems. Man and Cybernetics, 34(3):334-352, August 2004.
[13] Regazzoni C. S., Visvanathan R., Foresti G. L. Scanning the issue / technology - Special Issue on Video Communications, processing and understanding for third generation surveillance systems. Proceedings of the IEEE. 2001 Oct; 89: 1355–1367
[14] P. Kumar, S. Ranganath, Huang Weimin, and K. Sengupta. Framework for real-time behavior interpretation from traffic video. IEEE Transactions on Intelligent Transportation Systems, 6(1):43-53, 2005.
[15] Tracking and counting moving people. Proceedings of The Second IEEE International Conference on Image Processing, pages 212-216, 1994.
[16] S. An, W. Liu, and S. Venkatesh. Face recognition using kernel ridge regression. pages 1-7, 2007.
[17] F. Lv, X. Song, B. Wu, V. K. Singh, and R. Nevatia. Left-luggage detection using Bayesian inference. 9th International Workshop on Performance Evaluation in Tracking and Surveillance (PETS-CVPR'06), pages 83-90, 2006.
[18] M. Gschwind. The Cell broadband engine: Exploiting multiple levels of parallelism in a chip multiprocessor. International Journal of Parallel Programming, 35(3):233-262, 2007.
[19] S. Williams, J. Shalf, L. Oliker, S. Kamil. P. Husbands, and K. A. Yelick. Scientific computing kernels on the Cell processor. Int. J. Parallel Programming. 35:263-298. 2007.
[20] M. Hidemasa, D. Munehiro, N. Hiroki, and M. Yumi. Multilevel parallelization on the Cell/B.E. for a motion JPEG 2000 encoding server. Proc. I 5th International Conference Multimedia, pages 942-951, 2007
[21] L. Liu, S. Kesavarapu, J. Connell, A. Jagmohan, A. Leem, L. Paulovicks, B. Sheinin, V. L. Tang, and H. Yeo. Video analysis and compression on the STI Cell broadband engine processor. IEEE International Conference on Multimedia and Expo, 2006.
[22] K. Fatahalian, T. J. Knight, M. Houston, M. Erez, D. R. Horn, L. Leem, J-Y. Park, M. Ren, A. Aiken, W. J. Daily, and P. Hanrahan. Sequoia: Programming the memory hierarchy, p. online. Proc. ACM/IEEE Conference Supercomputing, 2006.
[23] B. Bouzas, R. Cooper, J. Greene, M. Pepe, and M-J. Prelle. Multicore framework: An API for programming heterogeneous multicore processors. Technical report, Mercury Computer Systems, Inc., 2006.
[24] Fan, Z., Qiu, F., Kaufman, A., and Yoakum-Stover, S.GPU Cluster for High Performance Computing. in Proc. Of the 2004 ACM/IEEE Conf. on Supercomputing, 2004.
[25] Ohmer, J. F., Maire, F., and Brown, R. 2006. Real-Time Tracking with Non-Rigid Geometric Templates Using the GPU. In Proc. of the Int. Conf. on Computer Graphics, Imaging and Visualisation (July 26 - 28, 2006). CGIV. IEEE Computer Society, Washington, DC, 200-206.
[26] NVIDIA, CUDA Programming Guide Version 2.0. 2008, NVIDIA Corporation: Santa Clara, California.
[27] Li Yao and Miaogen Ling, An Improved Mixture-of-Gaussians Background Model with Frame Difference and Blob Tracking in Video Stream, The Scientific World Journal Volume 2014 (2014), Article ID 424050.
[28] Intel, Quad-Core Intel® Xeon® Processor 5400 Series.2008, Intel Corporation: Santa Clara, California.

Mallikarjuna Rao Gundavarapu. obtained B.Tech degree from Nagarjuna University in the specialization Electronics and Communication in 1988. He is a double postgraduate (ME, M. Tech). He is pursuing PhD from JNTUH. His research interests are Pattern Recognition, Parallel Computing and Artificial Neural Networks. Mr Rao is presently working as a professor in Gokaraju Rangaraju Institute of Engineering and Technology. Professor Rao has 12 publications in various international journals and conferences. He is instrumental in established ***Parallel Computing and Operating Systems Lab*** at GRIET under MODROB scheme. He proposed salable and portable feature extraction technique Local Active Pixel Pattern, LAPP.

Mallikarjuna Rao Ch. received the B. Tech degree in computer science from Dr. Baba Sahib Ambedkar Marathwada University, Aurangabad, Maharastra in 1998, and the M. Tech Degree in Computer Science and Engineering from J.N.T.U Anantapur, Andhrapradesh in 2007. He is currently pursuing his PhD degree from JNTU Ananthapur University.

Anuradha Bai K., She completed her master's degree, M. Tech, from Gokaraju Rangaraju Institute of Engineering and Technology in Computer Science and Engineering.