Motion Detection in Low Resolution Grayscale Videos Using Fast Normalized Cross Correrelation on GP-GPU

Durgaprasad Gangodkar
Dept. of Electronics and Computer Engineering, Indian Institute of Technology Roorkee, Roorkee, India,
dgangodkar@yahoo.com

Gurbinder Singh Gill
Dept. of Electronics and Computer Engineering, Indian Institute of Technology Roorkee, Roorkee, India,
gurbinder533@gmail.com

Sachin Gupta
Dept. of Electronics and Computer Engineering, Indian Institute of Technology Roorkee, Roorkee, India,
sachingupta006@gmail.com

Padam Kumar
Dept. of Electronics and Computer Engineering, Indian Institute of Technology Roorkee, Roorkee, India,
padamfec@iitr.ernet.in

Ankush Mittal Dr.
Dept. of Computer Science and Engineering, College of Engineering Roorkee, Roorkee, India,
dr.ankush.mittal@gmail.com

Follow this and additional works at: https://www.interscience.in/ijess

---

Motion Detection in Low Resolution Grayscale Videos Using Fast Normalized Cross Correrelation on GP-GPU

Durgaprasad Gangodkar[1], Gurbinder Singh Gill[2], Sachin Gupta[3], Padam Kumar[4], Ankush Mittal[5]

[1234]Dept. of Electronics and Computer Engineering, Indian Institute of Technology Roorkee, Roorkee, India
[5]Dept. of Computer Science and Engineering, College of Engineering Roorkee, Roorkee, India
E-mail : [1]dgangodkar@yahoo.com, [2]gurbinder533@gmail.com, [3]sachingupta006@gmail.com,
[4]padamfec@iitr.ernet.in, [5]dr.ankush.mittal@gmail.com

Abstract— Motion estimation (ME) has been widely used in many computer vision applications, such as object tracking, object detection, pattern recognition and video compression. The most popular block based similarity measures are the sum of absolute differences (SAD), the sum of squared differences (SSD) and the normalized cross correlation (NCC). Similarity measure obtained using NCC is more robust under varying illumination changes as compared to SAD and SSD. However NCC is computationally expensive and application of NCC using full or exhaustive search method further increases required computational time. Relatively efficient way of calculating the NCC is to pre-compute sum-tables to perform the normalization referred to as fast NCC (FCC). In this paper we propose real time implementation of full search FCC algorithm applied to gray scale videos using NVIDIA's Compute Unified Device Architecture (CUDA). We present fine-grained optimization techniques for fully exploiting computational capacity of CUDA. Novel parallelization strategies adopted for extracting data parallelism substantially reduce computational time of exhaustive FCC. We show that by efficient utilization of global, shared and texture memories available on CUDA, we can obtain the speedup of the order of 10x as compared to the sequential implementation of FCC.

Keywords- Motion detection; Fast Normalized Cross Correlation; CUDA; Gray scale videos; Block matching algorithm

I. INTRODUCTION

The explosive growth of digital video content from commodity devices has precipitated a renewed interest in video processing technology, which broadly encompasses the compression, enhancement, analysis and synthesis of digital video. Foundation for many of computer vision and multimedia applications is efficient and robust motion estimation. Block based motion estimation (BMA) has proved to be one of the effective means to determine motion in video sequences. BMA was adopted by many video-coding standards such as MPEG-1/2/4, H.261, H.263 and H.264/AVC etc. [1] [2]. In BMA, motion estimation is performed using a sequence of video frames. Each frame is divided into sub-blocks, called macroblocks (MBs), typically of size 16 x 16 pixels. BMA estimates the amount of motion on a block by block basis, i.e. for each block in the current frame (CF), a block from the previous frame, called Reference Frame (RF) within the search window is found, that is said to match this block based on a certain matching criterion. Best match determines the location of macroblock within the search window (motion vectors). Full search (FS) or exhaustive search algorithms consider every pixel in the block to find out the best match. Although the FS is very efficient in finding out the best match, it is computationally very expensive. In order to reduce the computational complexity of FS

many algorithms like three-step search [3], new three-step search [4], four-step search [5], block-based gradient descent search [6], diamond search [7][8], Hexagon-Based Search [9] etc were proposed. These algorithms make use of heuristic measures to reduce the number of search points and thus obtain sub-optimal results as compared to FS algorithm.

Different measures have been proposed in the literature as matching criterions like mean of absolute difference (MAD), SAD, SSD and NCC. Choice of the correlation coefficient over alternative matching criteria, such as the sum of absolute differences, has been justified as maximum-likelihood estimation [10]. An empirical study of five template matching algorithms in the presence of various image distortions has found that NCC provides the best performance in all image categories [11]. NCC is also more robust against image variations such as illumination changes than the widely-used SAD and MAD. Thus NCC is a most suitable measure to efficiently determine the motion in low resolution grayscale videos. Due to its effectiveness, NCC has been used in many applications like image registration [12], template matching [13], motion estimation [14] etc.

However, higher computational cost of NCC is a significant drawback in its real-time application. For

correlation-based measures, such as cross-correlation, correlation-coefficient and NCC, there are limited proposals to reduce the search points. This is primarily because the reduction approaches based on distance measures, are not directly applicable to correlation measures. Different approaches have been proposed for reducing the computational complexity of NCC. Approach presented in [15][16] computes standard cross correlation (CC) in the frequency domain using fast Fourier transform and then to compute denominator using sum tables. However, the calculation of the numerator dominates the computational cost even though the FFT is used. Approached proposed in [17] describes the method of approximation of template matching using K-basis functions specifically for applications like template matching that allows the use of sum tables for efficient use of numerator. Though the use of sum-tables reduces the overall computational complexity to a greater extent, the ever pressing need to improve upon robustness of the computer vision systems now demand for processing inputs obtained from multiple cameras rather than a single camera and still expect the results within the real-time deadlines. This poses a challenge for further improving the speedup of these computationally expensive algorithms that will pave the way for processing higher number of frames per second obtained from multiple cameras

Recently, multicore processors have emerged as co-processing units for central processing units (CPUs) to accelerate various compute intensive applications. For example IBM's PowerXCell 8i based on Cell BE architecture has eight SPEs and one PPE delivering an effective peak performance of more than 230 GFLOPs (single precision) and 102 GFLOPs (double precision) [18]. NVIDIA's Compute Unified Device Architecture (CUDA) based GTX280 can provide theoretical peak performance of around 933 GFLOPs (single precision) and 78 GFLOPs (double precision) [19]. These have also proven to be quite economical compared to traditional cluster or grid based solutions. Many researchers have implemented, assessed feasibility and analyzed the performance gain that can be obtained by successfully parallelizing compute intensive image processing algorithms. CUDA enabled device GeForce GTX 260 has been used in [19] to implement the Mixture of Gaussians (MoG) technique. Parallel implementation runs 26 times faster than a sequential algorithm implemented on AMD Opteron 2218 at 2.6 GHz for an image size of 320 x 240 pixels. Other computer vision application like optical flow [20], edge detection, threshold filtering and image resizing [21] have been implemented on multicore processors. The work done till date, demonstrates unprecedented opportunities for researchers to develop strategies to deal with large datasets with acceptable computing performance.

In this paper we present efficient parallelization strategies for full search implementation of FCC applied to grayscale images for motion detection on graphics processing units (GPU's). We demonstrate that by successfully exploiting computational power of multicore processors we can achieve impressive speedup as compared to the sequential implementation. We make use of sum table scheme proposed by [16] which allows the calculations of mean, image variance and cross correlation between images. To the best of our knowledge this is the first proposal for parallel implementation of FCC on multicore processors. The discussion mainly focuses on the parallel implementation of FCC on CUDA architecture. Main contributions of this paper are summarized as follows:

- We present a novel strategy for parallel calculation of sum-tables by making use of prefix-sum algorithm that optimally makes use of texture and shared memories.

- We demonstrate an efficient approach for kernel configuration that can be used to exploit optimum computational capacity of streaming multiprocessors (SMPs) by making use of host of memories.

- We extract data parallelism in the algorithms by dividing computationally intensive tasks for parallel and scalable execution on the multiple cores such that it meets the real-time requirements

- Image processing involves lot of data transfer between GPU (device) and CPU (host) which becomes bottleneck leading to poor parallelization performance. To mitigate this overhead we map host memory to device memory (pinned memory) to exploit full   duplex bandwidth of PCIe bus.

Experiments are conducted on different sizes of video frames with varying characteristics to assess scalability, performance gain etc. The rest of the paper is organized as follows. Section II provides a brief overview of the CUDA. In Sections III we discuss normalized and fast normalized cross correlation algorithms. Section IV details strategy adopted for parallel implementation FCC algorithm. Section V provides experimental details with performance evaluation and section VI summarizes the discussion and indicates some possible future work.

II. NVIDIA's Compute Unified Device Architecture

NVIDIA's CUDA [23], a general purpose computing architecture on a GPU, provides avenues for active research to tackle compute intensive tasks. The CUDA parallel programming model is designed for writing highly scalable parallel code that can run across tens of thousands of concurrent threads and hundreds of processor cores. A CUDA program is organized into a

host program, consisting of one or more sequential threads running on the host CPU, and one or more parallel kernels that are suitable for execution on a parallel processing device like the GPU. The three key abstractions of CUDA are the thread hierarchy, shared memories and barrier synchronization, which render it as only an extension of C. All the GPU threads run the same code, are very light weight and have a low creation overhead. A kernel executes a scalar sequential program on a set of parallel threads. The programmer organizes these threads into a grid of thread blocks.

A kernel can be executed by a 1-D or 2-D grid of multiple, equally-shaped thread blocks. A thread block is a 3, 2 or 1-D group of threads.
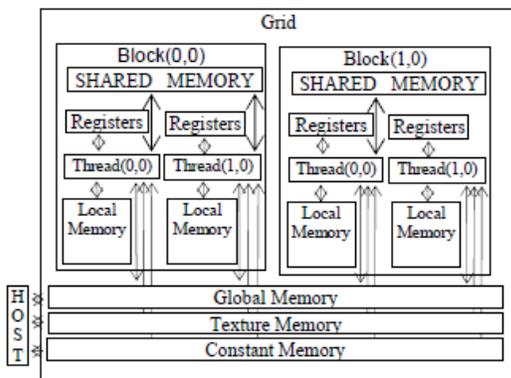

Figure 1. CUDA device memory space

Threads of a single thread block are allowed to synchronize with each other via barriers and have access to a high-speed, per-block shared on-chip memory for inter-thread communication. Threads from different blocks in the same grid can coordinate only via operations in a shared global memory space visible to all threads. CUDA requires that thread blocks be independent, meaning that a kernel must execute correctly no matter the order in which blocks are run, even if all blocks are executed sequentially in arbitrary order without preemption. This restriction on the dependencies between blocks of a kernel provides scalability. It also implies that the need for global communication or synchronization amongst threads is the main consideration in decomposing parallel work into separate kernels. On current GPUs, a thread block may contain up to 512 threads. The multiprocessor executes threads in groups of 32 parallel threads called warps. Threads have access to different types of memories as shown in Fig. 1. The constant memory is useful only when it is required that the entire warp may

read a single memory location. The shared memory is on-chip and the accesses are 100x-150x faster than accesses to local and global memory.

The shared memory, for high bandwidth, is divided into equal sized memory modules called banks, which can be accessed simultaneously. However, if two addresses of a memory request fall in the same memory bank, there is a bank conflict and the access has to be serialized. The banks are organized such that successive 32-bit words are assigned to successive banks and each bank has a bandwidth of 32 bits per two clock cycles. The texture memory space is cached so a texture fetch costs one memory read from device memory only on a cache miss, otherwise it just costs one read from the texture cache. The local and global memories are not cached and their access latencies are high. CUDA 2.2 release provides page-locked host memory and helps in increasing the overall bandwidth when the memory is required to be read or written exactly once. It can be mapped (pinned) to device address space so no explicit memory transfer required. Pinned memory accessed using *cudaHostAlloc* function delivers high bandwidth performance. This can be 2x faster than non-pinned memory accessed by using *cudaMemcpy* because mapped memory is able to exploit the full duplex capability of the PCIe bus by reading and writing at the same time.

III. NORMALIZED CROSS CORRELATION

NCC has been commonly used as a metric to evaluate the similarity (or dissimilarity) measure between two compared images. Given a template t it determines the location of t in two-dimensional image f. In this section we provide a brief overview of NCC and FCC [16][24].

Consider a template t of size ⨯ to be matched with an image of size ⨯ where, ≤ and ≤ . Let denote the intensity value at pixel location in where ∈ {0, . . ., }, ∈{0, . . ., } and be the intensity value at pixel location in . The position of the template in image is determined by calculating the NCC value at every step by shifting through steps in direction and steps in direction. Where ∈{0, 1, 2, . . ., − } and ∈ {0, 1, 2, ..., − }. The basic equation for NCC is as given in (1).

$$\gamma_{u,v} = \frac{\sum_{x,y}(f(x,y) - \bar{f}_{u,v})(t(x-u,y-v) - \bar{t})}{\sqrt{\sum_{x,y}(f(x,y) - \bar{f}_{u,v})^2 \sum_{x,y}(t(x-u,y-v) - \bar{t})^2}} \quad (1)$$

where, $\bar{f}_{u,v}$ denotes the mean value of $f_{(x,y)}$ within the area of template t shifted by (u,v) and is calculated as:

$$\bar{f}_{u,v} = \frac{1}{N_x N_y} \sum_{x=u}^{u+N_x-1} \sum_{y=v}^{v+N_y-1} f(x,y) \quad (2)$$

Similarly denotes the mean value of template . Direct computation of (1) involves the order of (–)(– calculations [24], which is very computationally expensive. For example, to match a small 16×16 pixel template with a 250×250 pixel image would require a total of more than 14 million calculations.

A. Fast Normalized Cross Correlation

In order to overcome the computational complexity of NCC, an efficient method proposed by [16] was to calculate the denominator of (1) using the concept of sum-tables as discussed below. This approach makes use of two sum tables and over the image function and image energy . The sum-tables of image function and image energy are computed recursively as given below.

$$s(u,v) = f(u,v) + s(u-1,v) + s(u,v-1) - s(u-1,v-1) \quad (3)$$

$$s^2(u,v) = f^2(u,v) + s^2(u-1,v) - s^2(u,v-1) - s^2(u-1,v-1) \quad (4)$$

$$\sum_{x=u}^{u+Nx-1}\sum_{y=v}^{v+Ny-1} f(x,y) = s(u-Nx-1, v-Ny-1) \\ -s(u-1, v+Ny-1) + s(u-1, v-1) \\ -s(u+Nx-1, v-1) \quad (5)$$

$$\sum_{x=u}^{u+Nx-1}\sum_{y=v}^{v+Ny-1} f^2(x,y) = s^2(u+Nx-1, v+Ny-1) \\ -s^2(u-1, v+Ny-1) + s^2(u-1, v-1) \\ -s^2(u+Nx-1, v-1) \quad (6)$$

When either $u,v=0$, $s(u,v)=s^2(u,v)=0$

IV. Parallel Implementation of Fast Normalized Cross Correlation

Computation time of FCC can be further reduced by efficient utilization of multicore processors. In this section we present the parallelization strategies adopted for implementation of FCC on NVIDIA's CUDA architecture. In order to obtain motion vectors we compute correlation between RF and CF. We adopt two stage approach for parallelizing the algorithm. In the first stage we compute the sum-tables and then in the second stage we compute normalized cross correlation by utilizing the sum-tables as a look up.

A. Computation of sum-tables

The sum tables are generated by first calculating the cumulative sum over the image points and then computing the square values. We make use of parallel prefix-sum algorithm as shown in Figure 2, to optimally compute the sum-tables.

Here n corresponds to image width and j corresponds to the number of threads. Each row of an image has been assigned to a single thread block and there would be as many blocks as the number of rows. On current GPUs, a thread block may contain up to maximum of 512 threads and every streaming multi-processor (SMP) can process maximum 768 threads in parallel from maximum 12 thread blocks, in the warps of 32 threads. Block size of 256 threads has been chosen so that every SMP can schedule three blocks by optimum resource utilization. To overcome global memory access latency, we load RF and CF in texture memory and subsequently every thread block dynamically caches a row of an image in the shared memory. All the threads in a thread block begin by cooperatively computing pre-fix sum of each row using the parallel prefix-sum algorithm. Next step to calculate cumulative sum involves computing pre-fix sum column wise. However, the consecutive indices would be image width apart. In order to keep the locality of references we take a transpose of the partial sum-table calculated up to this point and bring them in row major order. Parallel prefix-sum algorithm is again applied to each row followed by a transpose to obtain the complete sum table.

B. Computation of normalized cross correlation

We divide computational tasks as follows. Frames to be compared are divided into sub blocks and motion vectors are computed by applying NCC as a block matching criteria for each sub block. We divide CF into a motion window of 16x16 pixels and RF into a search window of 32x32 pixels. The cross correlation value is calculated by utilizing the sum-tables as lookup by moving the motion block over the referenced search window pixel by pixel, such that it covers the entire search window and the highest correlation representing the motion vector i.e. the displacement relative to current block is stored. Subsequent motion block (shifted by 16 pixels) is used to

obtain the next highest correlation as shown in Figure 3. This procedure is repeated for the entire frame. We carry out the computation of best match for each motion block in parallel using data parallelism. We assign the task of computing best match for each motion block to a single thread. Thus every thread computes the best match by searching the search window pixel by pixel and stores the result. Grouping of threads into thread blocks and thread blocks into grids has been carried out quite efficiently to exploit host of memories.

for each i in 0 to $[log_2(n)] - 1$ *do*

for each j in 0 to n − 1 do in parallel

if $j \geq 2^i$ *then*

$x_j \leftarrow x_j + x_{j-2^i}$

end if

end parallel for

end for

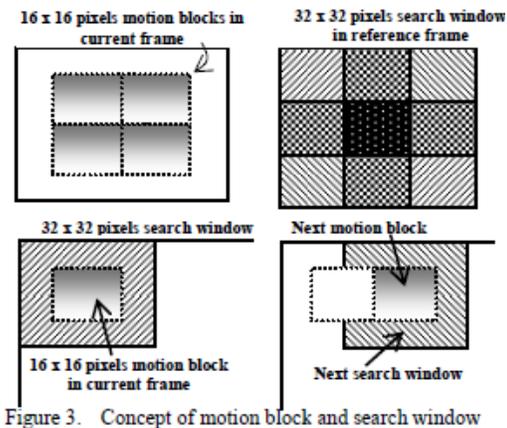Fig. 2 : Parallel pre-fix sum algorithm

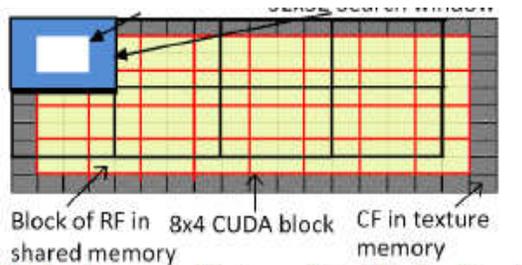
Figure 3. Concept of motion block and search window

Figure 14. Strategy for memory allocation and thread block configuration

Intricacies of our strategy are discussed as below. The crucial aspect of thread configuration is utilization of faster shared and texture memories to mitigate the higher memory latency involved in accessing global memory. The memory layout of our proposed strategy is as shown in Figure 4. The shared memory available per multiprocessor is limited to 16 KB. The number of threads per block is therefore restricted by the limited memory resources of a processor core. Thus it is a challenge to develop the thread configuration for addressing the limitations of CUDA. In order to efficiently utilize the available shared memory, we are caching 64 macroblocks of RF amounting to 16 KB in shared memory and arranging threads in 2-D of size 8 x 4 totaling 32 threads per thread block. Thus every SMP can optimally schedule two thread blocks and make use of full shared memory.

Further the CF is loaded into texture memory. The texture memory space is cached, so a texture fetch costs one memory read from device memory only on a cache miss, otherwise it just costs one read from the texture cache. The texture cache is optimized for 2D spatial locality, so threads of the same warp that read texture addresses that are close together will achieve best performance.

For example, a frame size of 1024×1024 has 64×64 macroblocks of 16×16 pixels each. These are allocated to 64×64 thread blocks having 32 threads each, arranged in to a 2-D grid of size 8×16. Since data must be transferred to and from the GPU, the memory transfer time affects performance. To mitigate overhead of data transfer, we make use of "mapped" pinned buffers that map host memory into the CUDA address space. Pinned memory is able to exploit the full duplex capability of the PCIe bus by reading and writing at the same time [25].

V. EXPERIMENTAL DETAILS AND PERFORMANCE EVALUATION

We evaluated and measured the execution time and speedup of our proposed parallel implementation FCC algorithm for images of different sizes. The sequential code was implemented on Intel Xeon 3.2 GHz processor with 1 GB of DRAM and 32 bit Windows XP OS. Parallel code was implemented on NVIDIA GTX 280 having 1 GB of DDR3 onboard Intel Xeon 3.2 GHz processor with 1 GB of DRAM and 32 bit Windows XP OS. GTX 280 has 30 multiprocessors with 8 cores each, totaling 240 cores and having single precision floating

point capability of 933 GFLOFS with compute capability of 1.3. Visual Studio 2005 was used as the development environment and the CUDA profiler version 2.2 was used for profiling the CUDA implementation. Image sizes of 256 x191, 320 x 240, 512x 512, 640 x 480, 1024 x 1024, were used for performance evaluation [26].

Table I tabulate the execution time required for sequential and parallel implementation of FCC algorithm for different frame sizes. It can be observed that for the frame size of 1024x1024 we could achieve the considerable reduction in execution time from 852 ms to 90 ms yielding a speedup of around 10x. We have efficiently utilized shared and texture memories to overcome the memory access latencies which have contributed to the overall performance gain. The computation load is varied by changing the size of the image and it is observed that the implementation scales well with higher image sizes. We have developed total five kernel functions for computation of motion vectors. We retain the device memory pointers on host that map the calculations carried out by individual kernels. Different kernel functions are launched from host by passing only memory references between the kernel calls. This overcomes need for transferring the results from device to host and vice-versa. We have made use of pinned buffers for mapping host memory with device memory which resulted into avoiding explicit copying of resultant motion vectors from device to host. Referring to the profiler output tabulated in Table II, we find the appreciable GPU occupancy of 0.75 for prefix-sum computation and maximum i.e. 1 for transpose operation. Further during sum-table calculations, the ratio of divergent branches to total branches is also quite low. For the CC kernel, as every thread computes the best match for entire search window, GPU occupancy remains low. However it is evident that every thread block uses maximum shared memory and drastically reduces the divergent branches.

From the above discussion it can be observed that every thread has been assigned an independent task of computing the motion vector which eliminates inter-thread communication, inter-thread dependencies and synchronization. The arrangement of threads into blocks and grids has been done to exploit the optimal computational capacity of CUDA architecture. Further, we have also devised efficient strategies to make use of the faster shared and texture memories to overcome memory access latency. Major overhead associated with data transfer has been eliminated by using pinned memory.

TABLE I. SPEEEDUP OBTAINED BY PARALLEL IMPLEMENTATIONS OF FCC ALGORITHM

Image size in pixels	Seq. Code	Execution time in ms			Speedup
		Parallel Code			
		Sum-table	Cross Correl.	Total	
256x191	33	1.60	10.42	12.02	3
320x240	54	2.30	10.40	12.70	5
512x512	188	8.20	20.30	28.50	7
640x480	225	10.00	21.45	31.45	8
1024x1024	852	37.40	52.60	90.00	10

TABLE II. CUDA PROFILER OUTPUT FOR FRAME SIZE OF 1024x1024

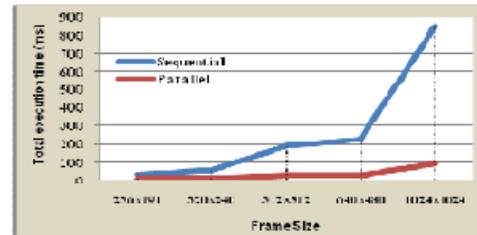| Kernel	Occupancy	Grid Size		Block Size		Shared Memory per block	Total Branches	Divergent Branches
		x	y	x	y			
Pre-fix sum	0.75	1	1024	256	1	4128	144568	816
Transp.	1	64	64	16	16	1120	9928	136
CC	0.031	8	16	8	4	8236	22336	4

Figure 5. Comparison of execution time required for sequential and parallel implementations of FCC algorithm.
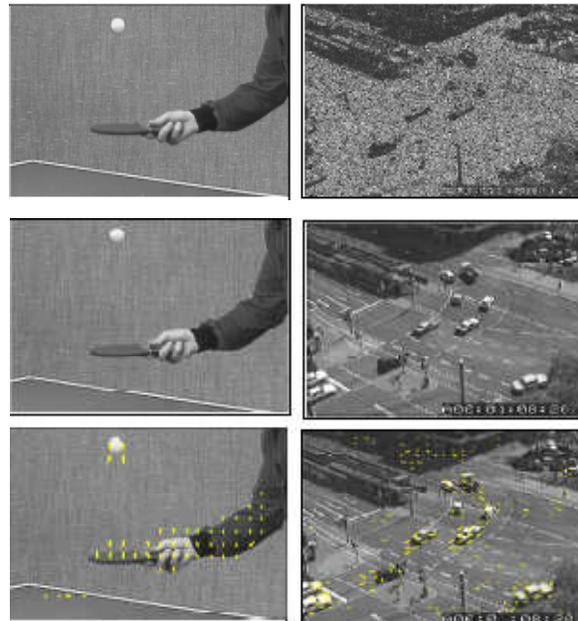

Figure 6 : First row RF, second row CF and third row motion vectors

Required number of thread blocks can be easily arranged by varying the grid size to match low resolution or high resolution frames. Figure 6 shows the motion vectors plotted for two sets of frames. A noise component due to distortion is also introduced. Morphological operations like dilation and erosion can be used to reduce the noise and the results can be used in further processing.

VI. CONCLUSION

The results show that significant reduction in computation time can be obtained by exploiting computational power of multicore processors. Fast normalized cross correlation has been used as an efficient measure for detecting motion in low resolution grayscale videos. Our strategy was to pre-compute sum-tables that act as a look up table for computation of correlation measure. For computing the cross correlation, our approach was to divide the task such that every thread works on independent data that avoided sharing of computational results and inter thread synchronization. While calculating the cross correlation, computational load has been equally distributed amongst the threads eliminating idling of any thread. Motion vector space in CPU memory is mapped to GPU memory that reduces the execution time by avoiding explicit data transfer. Efficient usage of shared and texture memories coupled with optimal thread block and grid formation has further improved the performance gain. Many-core processors are proving to be a cost effective alternative to cluster based solutions for real-time computer vision applications, especially in video surveillance systems. Our future work will include implementation of image registration and template matching using above algorithm on GPUs.

REFERENCES

1. K. R. Rao and J. J Hwang, "Techniques and standards for image, video and audio coding", Englewood Cliffs, NJ, Prentice Hall, 1996

2. T. Wiegand and G. Sullivan, "Joint video team (JVT) of ISO/IEC MPEG and ITU-T VCEG. Draft ITU-T recommendation and final draft international standard of joint video specification (ITU-T Rec. H.264|ISO/IEC 14496-10 AVC)", document JVTG050d35.doc, 7th Meeting: Pattaya, Thailand, March 2003

3. T. Koga, K. Iinuma, A. Hirano, Y. Iijima, and T., Ishiguro, "Motion compensated interframe coding for video conferencing" in Proc. Nat. Telecommun. Conf., New Orleans, LA, pp. G5.3.1–G5.3.5., November 29–December 3, 1981

4. R. Li, B. Zeng and M. L. Liou "A new three-step search algorithm for block motion estimation", IEEE Trans. Circuits Syst. Video Technol., Vol. 4, No. 4, pp. 438–442, Aug. 1994.

5. L. M. Po and W. C. Ma, "A novel four-step search algorithm for fast block motion estimation", IEEE Trans. Circuits Syst. Video Technol., Vol. 6, No. 3, pp. 313–317, June 1996.

6. L. K. Liu and E. Feig, "A block-based gradient descent search algorithm for block motion estimation in video coding", IEEE Trans. Circuits Syst. Video Technol., Vol. 6, No. 4, pp. 419–423, Aug. 1996.

7. S. Zhu and K. K. Ma, "A new diamond search algorithm for fast block matching motion estimation", IEEE Trans. Image Processing, Vol. 9, No. 2, pp. 287–290, Feb. 2000.

8. J. Y. Tham, S. Ranganath, M. Ranganath, and A. A. Kassim, "A novel unrestricted center-biased diamond search algorithm for block motion estimation," IEEE Trans. Circuits Syst. Video Technol., Vol. 8, No. 4, pp. 369–377, Aug. 1998

9. Ce Zhu, Xiao Lin and Lap-Pui Chau, "Hexagon-based search pattern for fast block motion estimation", IEEE Trans. Circuits Syst. Video Technol., Vol. 12, No. 5, pp. 349-355 MAY 2002,

10. T. W. Ryan, "The prediction of cross-correlation accuracy in digital stereo-pair images", PhD thesis, University of Arizona, 1981.

11. P. J. Burt, C. Yen, and X. Xu. "Local correlation measures for motion analysis: A comparative study", Procd. of IEEE Conf. Pattern Recognition Image Processing, Las Vegas, pp. 269-274, June 14-17, 1982.

12. L. Essannouni, E. Ibn-Elhaj and D. Aboutajdine, "Fast cross-spectral image registration using new robust correlation", Journal of Real-Time Image Processing, Springer, Vol. 1, No. 2, pp. 123-129, Dec. 2006

13. M. Minoru and K. Kunio, "Fast template matching based on normalized cross correlation using adaptive block partitioning and initial threshold estimation", Procd. of IEEE International Symposium on Multimedia, Taichung, Taiwan, pp. 196 – 203, 13-15 Dec. 2010

14. J. Luo, E.E Konofagou, "A fast normalized cross-correlation calculation method for motion estimation", IEEE Trans. on Ultrasonics, Ferroelectrics and Frequency Control, Vol. 57, No. 6, pp. 1347 - 1357, Jun. 2010

15. J. P. Lewis, "Fast template matching", Vision Interface 95, Procd. Canadian Image Processing and Pattern Recognition Society, Quebec City, Canada, pp. 120–123, May 15– 19, 1995

16. J. P. Lewis, "Fast normalized cross-correlation", Industrial Lights and Magic

17. K. Briechl and U. D. Hanebeck, "Template matching using fast normalized cross correlation", Procd. of SPIE, Vol. 4387, No. 95, AeroSense Symposium, Orlando, Florida, 19 April, 2001.

18. IBM BladeCenter QS22(2008), Available at: http://www-07.ibm.com/systems includes/content/bladecenter/hardware/servers/qs2 2/BLD03019USEN.PDF,

19. Nvidia GTX280 documents, www.nvidia.com/content/GTC/documents/ SC09_Dongarra.pdf

20. T. Fábian and J. Gaura, "Parallel implementation of recursive background modeling technique in CUDA for tracking moving objects in video traffic surveillance", Proceedings of 4th Doctoral Workshop on Mathematical and Engineering Methods in Computer Science, Znojmo, Czech Republic, November 14 -16, 2008

21. H. K. Kidwai, T. Rabie and F. N. Sibai, "Parallel video processing performance evaluation on the IBM Cell Broadband Engine processor", International Journal of Computer Science and Applications, Technomathematics Research Foundation, Vol. 6, No. 1, pp. 13 – 25, Jan. 2009

22. H. K. Kidwai, F. N. Sibai and T. Rabie ,"Highly parallel image processing on the STI Cell", Procd. of IEEE/ACS International Conference on Computer Systems and Applications, AICCSA, Rabat, Morocco, pp. 849 – 852, May 10 -13, 2009

23. NVIDIA CUDA Programming Guide, Version 2.2, page 10, 27-35, 75-97, 2009

24. A. J. H. Hii, C. E. Hann, J. G. Chase and E. E. W. Van Houten, "Fast normalized cross correlation for motion tracking using basis functions", Journal of Computer Methods and Programs in Biomedicine, Elsevier, Vol. 82, No. 2, pp. 144– 156, 2006

25. Rob Farber, "CUDA, Supercomputing for the Masses"- Part 12: http://www.drdobbs.com/high-performance-computing/217500110

26. Image Sequence Server: http://i21www.ira.uka.de/ image_sequence