Hindawi Publishing Corporation
Mathematical Problems in Engineering
Volume 2014, Article ID 692921, 6 pages
http://dx.doi.org/10.1155/2014/692921

Research Article

A TBB-CUDA Implementation for Background Removal in a Video-Based Fire Detection System

Fan Wang, Xiao Jiang, and Xiao Peng Hu

School of Computer Science and Technology, Dalian University of Technology, Dalian 116024, China

Correspondence should be addressed to Fan Wang; wangfan@dlut.edu.cn

Received 7 December 2013; Revised 30 January 2014; Accepted 30 January 2014; Published 3 March 2014

Academic Editor: Massimo Scalia

Copyright © 2014 Fan Wang et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

This paper presents a parallel TBB-CUDA implementation for the acceleration of single-Gaussian distribution model, which is effective for background removal in the video-based fire detection system. In this framework, TBB mainly deals with initializing work of the estimated Gaussian model running on CPU, and CUDA performs background removal and adaption of the model running on GPU. This implementation can exploit the combined computation power of TBB-CUDA, which can be applied to the real-time environment. Over 220 video sequences are utilized in the experiments. The experimental results illustrate that TBB+CUDA can achieve a higher speedup than both TBB and CUDA. The proposed framework can effectively overcome the disadvantages of limited memory bandwidth and few execution units of CPU, and it reduces data transfer latency and memory latency between CPU and GPU.

1. Introduction

Video-based fire detection systems play an important role in the existing surveillance systems. Compared with conventional fire detection methods based on particle sensors [1], visual fire detection is more suitable in an open or large space, and it can provide abundant and intuitive information. For video-based fire detection, motion and color are the ordinary characteristics. There are several specific methods to find moving and flame color pixels by integrating background removal algorithms with Gaussian distribution models [2–4]. In addition to ordinary motion and color clues, flame and fire flickers can be detected by analyzing the video in wavelet domain [5–7]. These methods have been successfully applied in surveillance systems and proven effective. However, the demands of real-time processing require the acceleration of fire detection. Parallel processing is a suitable way to provide satisfactory performance for realistic applications.

GPU (graphics processing unit) has recently become a popular parallel platform for large-scale data computing. CUDA (Compute Unified Device Architecture), created by NVIDIA [8], provides a data-parallel programming framework and enables parallel execution of C function kernels [9]. For this reason, many developers have taken advantage of the high performance of CUDA to accelerate computation across various problem domains, such as signal processing, computer vision, computational geometry, and scientific computing [10–13]. However, the focus of CUDA is on complicated calculations. The memory latency and data transfer latency between CPU and GPU in data processing still need further consideration. TBB (Intel Threading Building Blocks) is a running-based parallel library that offers a rich methodology to express parallelism in C++ programs [14, 15]. As a typical fine-grain parallel model, TBB supports parallel tasks which run on threads. In addition, TBB implements task stealing to balance parallel workload across available processing cores, leading to the reduction of load imbalance, increase of core utilization, and adaptability to dynamic environments. Some researches take advantage of TBB to improve algorithms such as Floyd-Warshall algorithm [16], a Three-tier Parallel Genetic Algorithm (TPGA) [17], and Large Dense Linear Equations [18].

In our work, a parallel programming framework of CUDA+TBB is provided. We apply TBB to initialize work running on CPU, and CUDA to perform background removal and adaption of model running on GPU. The hybrid

parallel mode overcomes the disadvantages of limited memory bandwidth and less execution units of CPU in TBB. CUDA+TBB can effectively overcome the major drawback of CUDA by reducing the unnecessary data transfer latency and memory latency between GPU and CPU, resulting in computation acceleration. The rest of this paper is organized as follows. Section 2 presents the background modeling techniques based on single-Gaussian model and adaption of the model parameters. Parallelization to accelerate background removal is discussed in Section 3. Section 4 presents experiments where the parallel implementations are applied to video-based fire detection. We end this paper in Section 5 with conclusions and future work.

2. Background Modeling

Gaussian distribution is a common probability model that is widely used in pattern recognition and image processing to depict some random variables such as pixels and noise. For digital image processing, single-Gaussian model is used in the foreground extraction algorithm of the image whose background is single and stable. In the fire detection system, natural fire flames are seen as dynamic objects in the video images, which a fixed camera observed. The color of the fire regions is also a very important feature for distinguishing fire from others, so color information helps us to obtain fire regions more precisely. In this paper, we use a single-Gaussian model with mean and covariance matrix extracted from the video frames, which is based on RGB color spaces [19].

2.1. Background Removal Algorithm. The first step is to initialize the background model. In image processing, all operations are based on pixels: $\eta(x, \mu_t, \sigma_t)$, where x is the value of pixel, μ_t is the mean value of pixel, and σ_t is the standard deviation value of the pixel. Subscript t denotes time t. In this project, we use N ($N = 20$) frames to initialize the background. The formula of the mean and the standard deviation is as follows:

$$u_i(x,y) = \frac{1}{N}\sum_{t=1}^{N} I_i^t(x,y), \quad i \in \{R,G,B\},$$
$$\sigma_i = \sqrt{\frac{1}{N}\sum_{t=1}^{N}\left(I_i^t(x,y) - u_i(x,y)\right)^2}, \quad i \in \{R,G,B\}, \tag{1}$$

where $\mu_i(x, y)$ is the mean value of $I_i(x, y)$, $\sigma_i(x, y)$ is the standard deviation of $I_i(x, y)$, $i \in \{R, G, B\}$.

The second step is to classify the pixel. In order to reduce the computation, every channel of color space is assumed to be independent, so for each pixel, the color probability is calculated by [19]

$$p(I(x,y)) = p_R(I_R(x,y)) + p_G(I_G(x,y)) + p_B(I_B(x,y)), \tag{2}$$

where p_R, p_G, p_B are, respectively, distribution models for red, green, and blue channels, $I(x, y)$ is pixel value at coordinate (x, y), and $p(I(x, y))$ is provability density of $I(x, y)$. For certain color channel i, each distribution is given as follows:

$$p_i(I_i(x,y)) = \frac{1}{\sqrt{2\pi}\sigma_i(x,y)} \exp\left(\frac{\left(I_i(x,y) - \mu_i(x,y)\right)^2}{2\sigma_i^2(x,y)}\right), \quad i \in \{R,G,B\}. \tag{3}$$

Using the model parameters in (1), the following formula can show if the pixels are foreground or background:

$$D_i(x,y) = \begin{cases} 1, & \text{if } \left(\left|u_i(x,y) - I_i(x,y)\right|\right) \geq \alpha\sigma_i(x,y), \\ 0, & \text{otherwise,} \end{cases}$$
$$MD(x,y) = \begin{cases} 1, & \text{if } \left(\left(D_R + D_G + D_B\right) \geq 2\right), \\ 0, & \text{otherwise,} \end{cases} \tag{4}$$

where $D_i(x, y)$ denotes the result of i channel's detection. $D_i(x, y)$ is set to 1 if spatial location of (x, y) changed, 0 otherwise. α is the constant which can affect the final change detection. In this experiment, $\alpha = 2.5$. $MD(x, y)$ denotes the result of the pixel's detection. If there are changes at least in two-color channels, $MD(x, y)$shows that the pixel is marked as a foreground pixel, which is regarded as fire suspected area, otherwise it is considered to be a background.

2.2. Adaptation of Model Parameters. This step is to update the background model by adapting parameters. In general, the scene observed can change by lighting, or with other natural effects. In order to respond to environmental changes, the updating method of adapting pixel's parameters is given by

$$\mu_i^t(x,y) = \beta_i\mu_i^{t-1}(x,y) + (1-\beta_i)I_i^t(x,y),$$
$$\sigma_i^t(x,y) = \beta_i\sigma_i^{t-1}(x,y) + (1-\beta_i)\left|I_i^t(x,y) - \mu_i^t(x,y)\right|, \tag{5}$$

where $I_i^t(x, y)$ denotes value of pixel at coordinate (x, y) in ith color channel at time t, $\mu_i^t(x, y)$ and $\mu_i^{t-1}(x, y)$ denote mean values at times t and $t - 1$, respectively, and $\sigma_i^t(x, y)$ and $\sigma_i^{t-1}(x, y)$ are standard deviations at times t and $t - 1$, respectively. β_i is a constant for updating model parameters in ith color channel, which ranges from 0 to 1.

3. Parallel of Background Removal Model

In this section, we apply application of TBB+CUDA for background removal model. The hybrid architecture of CPU and GPU is shown in Figure 1. Firstly, video was decoded from the AVI formats on CPU. From the decoded video frames, we initialized mean and standard variance on CPU and transmitted initialized result to global memory on GPU. Secondly, kernel function reads the parameters from global memory to perform the multithreaded computing tasks. After all the calculations are finished, the result should be transmitted from GPU to CPU.

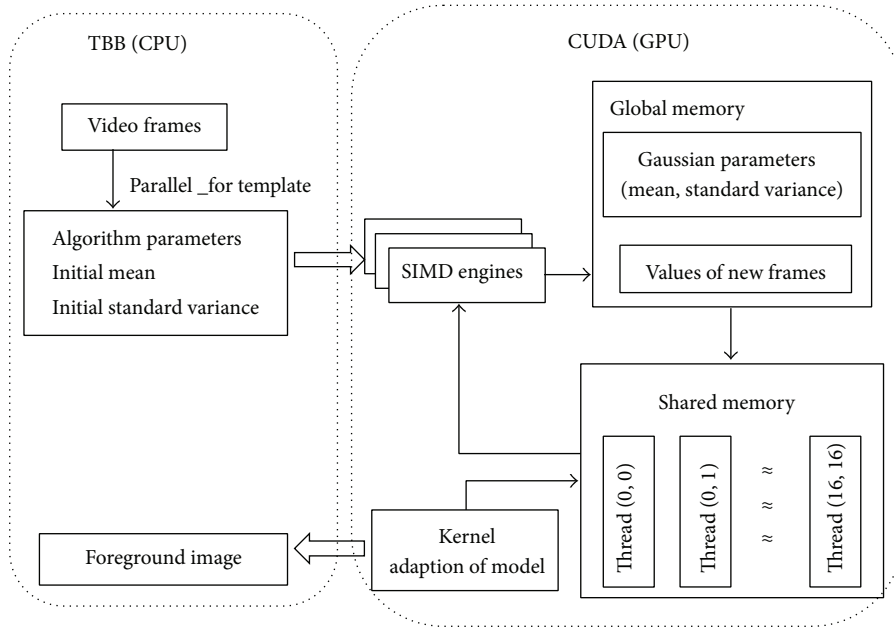


Figure 1: Framework of hybrid TBB-CUDA architecture.

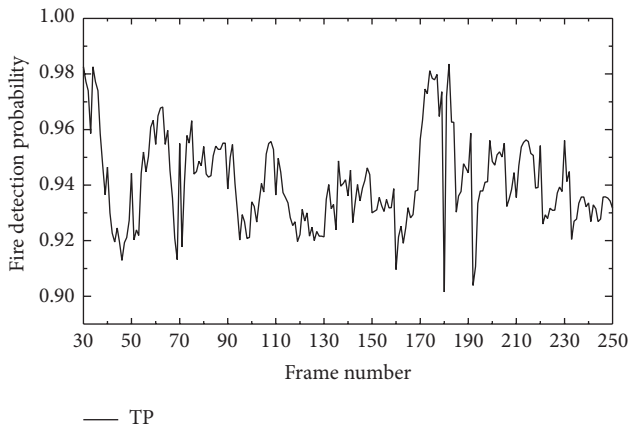


Figure 2: Fire detection probability.

3.1. Realization of Parallel Algorithm Based on TBB. TBB supports scalable parallel programming using standard ISO C++ code, and it puts focus on computation in parallel without having to explicitly deal with threads. In TBB, we specify tasks instead of threads. Tasks are mapped and scheduled to physical threads by the TBB scheduler [20]. Moreover, TBB can abstract platform details and simplify parallel programming. In addition, TBB provides a template-based runtime library which contains a series of data structures and algorithms, and it enables developers to devote themselves to address identifying concurrency rather than worrying about its management [21]. Through TBB relevant template class, mean value and standard deviation are assigned to different threads, resulting in making full use of multicore resources. In particular, the algorithm based on TBB includes the following steps.

Step 1. Installation of TBB parallel computing platform and setting environment for the preprocessing.

Step 2. Initialization of a TBB task scheduler. A task scheduler object is task_scheduler_init, which is responsible for supporting the allocation of multiple threads.

Step 3. Development of parallel computing template class. The template of parallel_for is selected to obtain the mean and the standard deviation of N frames in the body object.

Step 4. Invocation of the parallel template class "parallel_for." Once we have the loop body written as a body object, the general form of the constructor is parallel_for (block_range<T>(begin, end, grainsize)). Parallel_for breaks this iteration space into trunks and runs each trunk on a separate thread. Each operator implements a grainsize. In this project, the value of grainsize is set 1000 [11].

Step 5. End the TBB task scheduler and get results.

3.2. Realization of Parallel Algorithm Based on CUDA. In this section, we present the CUDA-based background removal model. Details can be described as follows.

Step 1. Declare shared memory. Each block has 16 K shared memories to store the mean value and standard deviation of 256 pixels. Shared memory is divided into 16 banks. The size of each bank is 32 bits and adjacent 32 bits are organized. The instruction of SM is executed in a half-warp as a unit. So threads in a half-warp read the data linearly from the banks in shared memory.

Step 2. Assign the number of threads and the size of block. We set 256 as the number of threads of a block

(a) (b)

Figure 3: Image 320 ∗ 240 (a) and background removal (b).

Table 1: Runtime on different image size (ms).

Image size	320 ∗ 240	432 ∗ 324	600 ∗ 480	1024 ∗ 768
TBB	0.2×10^6	0.315×10^6	0.625×10^6	0.138×10^7
CUDA	0.26×10^5	0.39×10^5	0.69×10^5	0.186×10^6
CUDA + TBB	0.25×10^5	0.31×10^5	0.59×10^5	0.156×10^6
Serial	0.24×10^6	0.41×10^6	0.875×10^6	0.242×10^7

Table 2: Speedup of three modes on different image size.

Image size	320 ∗ 240	432 ∗ 324	600 ∗ 480	1024 ∗ 768
TBB	1.2	1.3	1.4	1.77
CUDA	9	10.5	12.6	13
CUDA + TBB	9.7	13	14.8	15.6

(*THREAD_NUM*). The size of the blocks in a grid is width ∗ height/*THREAD_NUM*.

Step 3. Compute the position of the first pixel. In this project, each thread handles a pixel. For each thread, CUDA sets the thread number as threadIdx.x and block number as blockIdx.x. Each thread can determine the location of the corresponding data source according to threadIdx.x and blockIdx.x. The formula is as follows:

$$POSITION = \text{blockIdx.x} * THREAD\_NUM + threadIdx.x. \quad (6)$$

The value of *POSITION* is the start of the data source. Each thread executes the kernel program which is satisfied with the parallel operations of CUDA.

Step 4. Copy data from global memory to shared memory. Because the threads of a block enjoy the same shared memory, and the access latency of share memory is less than global memory, we transfer the data from global memory to the shared memory which included $u_i(x, y), \sigma_i(x, y), I_i^t(x, y),\ i \in (R, G, B)$.

Step 5. Transfer the updated data from GPU to CPU and release the memory of GPU.

4. Experiment Results

The experiments were conducted on a Pentium (R) Dual-Core E6500 2.94 GHz personal computer equipped with NVIDIA's GeForce 210 GTX graphics card. The software is Intel TBB and CUDA 4.0. We applied three different parallel methods including TBB, CUDA, and TBB+CUDA,

Table 3: Proportion of CPU-GPU data transfer latency.

Image size	320 ∗ 240	432 ∗ 324	600 ∗ 480	1024 ∗ 768
Latency of CUDA	8.7	11.3	19.02	30.54
Total time cost of CUDA	15.3	21.62	48.53	63
Latency of CUDA + TBB	0.78	1.1	1.78	2.64
Total time cost of CUDA + TBB	2	2.3	4.65	7.3
Proportion of (CUDA) %	56.8	52.3	39.2	48.4
Proportion of (TBB + CUDA) %	39	47	38	36.16

respectively. The video used in the experiments is real-world image sequences, which were taken from a random selection of commercial video clips. The type of fire is wild fire from mountains north of Athens. Matlab was used to convert the video sequence to 250 frames whose image sizes range from 320 ∗ 240 to 1024 ∗ 768.

The experimental results of runtime on different image sizes are described in Table 1. Runtime of TBB is obviously less than serial algorithm. TBB decides the number of threads that will be used by task scheduler automatically. Then it sends the threads to different cores according to the working stealing algorithm. The impact of TBB is emphasized when comparing the results obtained for serial approach.

Comparisons of the three methods' speedups on different image sizes are listed in Table 2. Speedup is the ratio of sequential runtime to parallel runtime for the same task. Because the required number of threads is much larger than the dual cores of CPU, what is more, GPU has many execution cores and a larger number of registers for data processing. As it can be seen, TBB achieves a lower performance when compared to CUDA. The speedup of CUDA is nearly at least 9 times higher than TBB, where TBB+CUDA achieve a higher speedup than CUDA.

In Contrast with TBB and CUDA method, TBB+CUDA can significantly accelerate the single-Gaussian distribution model. It can reduce the communication overhead of data transfer latency and memory latency between CPU and GPU. Total runtime includes data transfer time from CPU to GPU, runtime of kernel, and data transfer time from GPU to CPU, so it is better to cut down the unnecessary latency and reduce the proportion of CPU-GPU data transfer time on GPU. As shown in Table 3, the latency of CUDA is nearly 10–19 times larger than CUDA+TBB. TBB is arranged to get the parameters and the proportion of latency with CUDA+TBB has shown 2%–17% improvement over CUDA.

In this experiment, images were captured at nighttime. Experimental results are presented in Figure 2 and Figure 3, where the true positive (TP) of fire detection is demonstrated. An average TP of 94% over 220 test video sequences in the experiment can be obtained. Figure 3 shows the degree of the flame and the effects of background removal in video frames of 320 ∗ 240.

5. Conclusion

In order to accelerate the process of the background removal, this paper proposes a hybrid parallel mode. This parallel mode consists of two phases: the initializing phase by TBB running on CPU and the parallel computing phase for background removal and adaption of the model running on GPU with CUDA. The experimental results indicated that our solution makes full use of computation resources of GPU and CPU, leading to a higher speedup than TBB or CUDA. The hybrid parallel mode is generic to certain extent and can also be applied to other areas such as traffic routing and logistics location.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

This research was supported by "National Natural Science Foundation of China" (no. 61272523) and "the National Key Project of Science and Technology of China" (no. 2011ZX05039-003-4).

References

[1] W. W. Jones, "An algorithm for fast and reliable fire detection," in *Proceedings of the 8th Fire Suppression and Detection Research Application Symposium*, 2004.

[2] T. X. Truong and J. M. Kim, "Fire flame detection in video sequences using multi-stage pattern recognition techniques," *Engineering Applications of Artificial Intelligence*, vol. 25, no. 7, pp. 1365–1372, 2012.

[3] D. C. Wang, X. Cui, E. Park et al., "Adaptive flame detection using randomness testing and robust features," *Fire Safety Journal*, vol. 55, pp. 116–125, 2013.

[4] J. Chen, Y. He, and J. Wang, "Multi-feature fusion based fast video flame detection," *Building and Environment*, vol. 45, no. 5, pp. 1113–1122, 2010.

[5] B. U. Töreyin, Y. Dedeoğlu, U. Güdükbay, and A. E. Çetin, "Computer vision based method for real-time fire and flame detection," *Pattern Recognition Letters*, vol. 27, no. 1, pp. 49–58, 2006.

[6] B. U. Toreyin, Y. Dedeoglu, and A. E. Cetin, "Wavelet based real-time smoke detection in video," in *Proceedings of the European Signal Processing Conference*, 2005.

[7] Y. Dedeoglu, B. U. Töreyin, U. Güdükbay, and A. E. Çetin, "Real-time fire and flame detection in video," in *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP '05)*, pp. II669–II672, March 2005.

[8] NVIDIA, CUDA C Programming Guide, v. 3. 2. Nvidia Corp., 2010.

[9] S. Zhang, Y. Zhu, and K. Zhao, *GPU High Performance Computation-CUDA*, Water Power Press, Bei Jing, China, 2009.

[10] A. Hanani, M. J. Carey, and M. J. Russell, "Language identification using multi-core processors," *Computer Speech and Language*, vol. 26, no. 5, pp. 371–383, 2012.

[11] M. Czapiński, "An effective parallel multistart Tabu search for quadratic assignment problem on CUDA platform," *Journal of Parallel and Distributed Computing*, vol. 73, no. 11, pp. 1461–1468, 2013.

[12] K. A. Hawick, A. Leist, and D. P. Playne, "Parallel graph component labelling with GPUs and CUDA," *Parallel Computing*, vol. 36, no. 12, pp. 655–678, 2010.

[13] H. Hamzaçebi, *Cuda Based Implementation of Flame Detection Algorithms in Day and Infrared Camera Videos [Ph.D. dissertation]*, Bilkent University, 2011.

[14] J. Reinders, *Intel Threading Building Blocks: Outfitting C++ for Multi-Core Processor Parallelism*, O'Reilly Media, 2010.

[15] Electronic Publication: Digital Object Identifiers (DOIs), "Intel threading building blocks tutorial," 2007, http://www.threadingbuildingblocks.org/documentation.

[16] J. Ma, K.-P. Li, and L.-Y. Zhang, "A parallel Floyd-Warshall algorithm based on TBB," in *Proceedings of the 2nd IEEE International Conference on Information Management and Engineering (ICIME '10)*, pp. 429–433, April 2010.

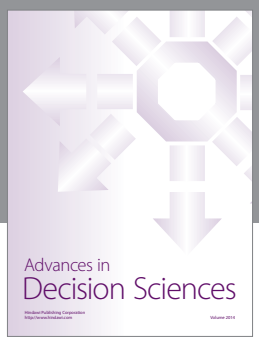
[17] L. Zhang, Y. Sun, J. Ma, and J. Sun, "A Parallel Genetic Algorithm based on TBB for resolving the bin-packing problem," in *Proceedings of the International Conference on Management and Service Science (MASS '11)*, August 2011.

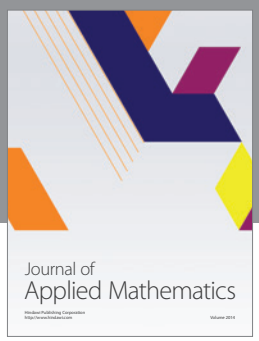
[18] S. Zhang, Z. Wei, and W. Xuben, "Implementation of multi-core parallel computation for solving large dense linear equations based on TBB," in *Proceedings of the IEEE International Conference on Control Engineering and Communication Technology (ICCECT '12)*, 2012.

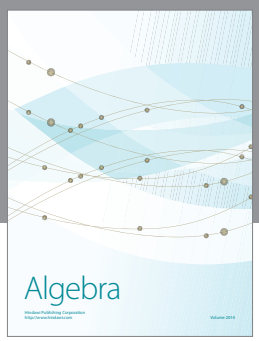
[19] T. Celik, H. Demirel, H. Ozkaramanli, and M. Uyguroglu, "Fire detection using statistical color model in video sequences," *Journal of Visual Communication and Image Representation*, vol. 18, no. 2, pp. 176–185, 2007.

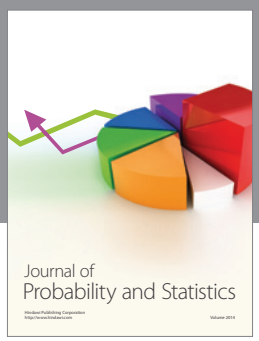
[20] D. Serfass and T. Peiyi, "Comparing parallel performance of Go and C++ TBB on a direct acyclic task graph using a dynamic programming problem," in *Proceedings of the 50th Annual Southeast Regional Conference*, ACM, 2012.

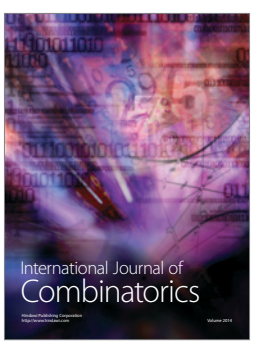
[21] X. Chen, W. Chen, J. Li, Z. Zheng, L. Shen, and Z. Wang, "Characterizing fine-grain parallelism on modern multicore platform," in *Proceedings of the 17th IEEE International Conference on Parallel and Distributed Systems (ICPADS '11)*, pp. 941–946, December 2011.

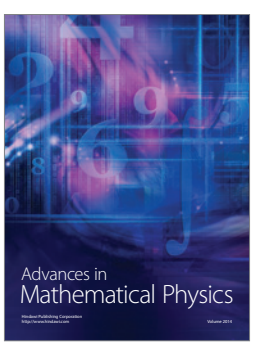


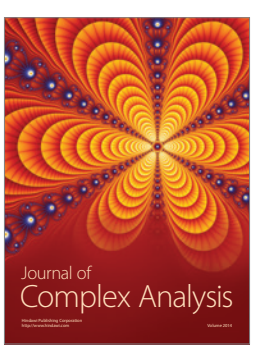


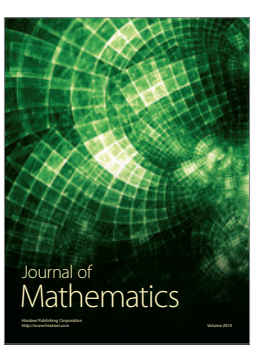


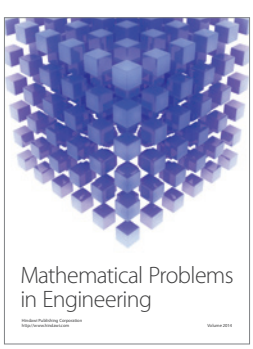


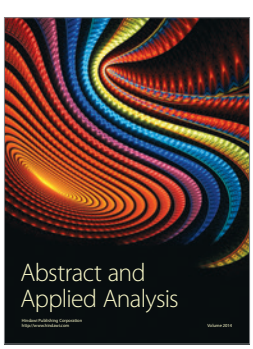


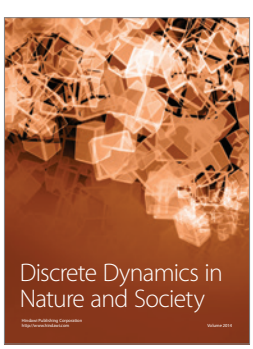


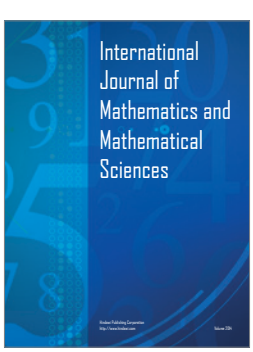












Hindawi

Submit your manuscripts at
http://www.hindawi.com