Journal of Theoretical and Applied Information Technology
10th December 2014. Vol.70 No.1

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

MULTI-CORE FRAMEWORKS INVESTIGATION ON A REAL-TIME OBJECT TRACKING APPLICATION

[1]MEI CHOO ANG, [2]AMIRHOSSEIN AGHAMOHAMMADI, [3]KOK WENG NG, [4]ELANKOVAN SUNDARARAJAN,

[5]MARZIEH MOGHARREBI, [6]TECK LOON LIM

[1,2,5]Institute of Visual Informatics, Universiti Kebangsaan Malaysia, Malaysia

[3]Industrial Design Centre, Sirim Berhad, Malaysia

[4]Centre for Software Technology and Management, [1]Center For Artificial Intelligence Technology, Faculty of Information Science and Technology, Universiti Kebangsaan Malaysia, Malaysia

[6]IME Solutions Sdn. Bhd., Malaysia

E-mail: [1]amc@ukm.edu.my, [2]aha.9362@gmail.com, [3]kwng@sirim.my, [4]elan@ukm.edu.my, [5]mogharrebi.marzieh@gmail.com, [6]jeff@imes.com.my

ABSTRACT

The current computer vision-based systems (CVS) are becoming computationally demanding due to the improvement of their functionalities that is difficult to be achieved with single-core frameworks. Such deficiencies of the single-core frameworks have led to the introduction multi-core frameworks to meet the required performance of their functionalities. However, in order to obtain good performance for CVS on multi-core frameworks, it is crucial to utilise parallelism tools efficiently. These parallelism tools need to be utilised on hotspots (most time-consuming functions in algorithm) in order to minimise development time and to reduce application development costs. This is a challenging task and requires an in-depth investigation of multi-core frameworks. This research work investigates the utilisations of multi-core frameworks capability for a real-time object tracking application problem using a parallel software tool known as Intel© Parallel Studio XE tool. In the investigation, two established multi-core frameworks, namely, Threading Building Blocks (TBB) and Open Multi-Processing (OpenMP) were implemented at identified hotspot functions of the tracking algorithm. The performances of these two multi-core frameworks were then evaluated and compared based on computed speedup, efficiency and scalability. The results from this investigation demonstrated that the processing time of real-time object tracking was improved by using hotspots identification. In addition to that, multi-core frameworks could make the tracking algorithm explicitly faster when compared to single-core frameworks and OpenMP outperformed TBB.

Keywords: *Multi-core Frameworks, Parallel Programming, Image Processing, Real-time Object Tracking, OpenMP, Threading Building Blocks (TBB), Intel® Parallel Studio XE*

1. INTRODUCTION

During the last few years, multi-core processors have become available and popular in personal computers as well as in portable devices like cell phones and notebooks [1]. This is because modern CPUs, on both desktop computers and portable devices are employing multi-core to meet the growing demands for computational power. In multi-core-based systems, multi-core frameworks and parallel programming models are employed to fully achieve the potential of parallelism functionalities. Currently, parallel programming models such as Pthreads often permit flexible parallel programming but this model relies on low-level techniques without explicitly considering factors such as processor communication, synchronization, and threads, rendering parallel programming more error-prone and tedious. Hence, it is crucial for the software developer to consider these factors for parallelization methods using suitable instruments and compilers [2]. At the same time, high level multi-core frameworks such as Open Multi-Processing (OpenMP) have emerged to help programmers to avert low-level implementation details of parallel programming [1].

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

OpenMP offers an industry standard for parallel computing API related to shared memory that involves multi-core processors [1]. Intel also enhanced a C++ template library as Threading Building Blocks (TBB) under Intel® Parallel Studio to use multi-core processors to avoid complications of using Pthreads [3].

Real-time object tracking is a vital and challenging task in computer vision community [4]. Object tracking aims to ascertain the location of a target of interest every moment in time [5]. The application of object tracking is found in various fields including automated surveillance, vehicle navigation and human-computer interaction [6]. In order to track objects in a scene, a tracking method is needed with an object detection mechanism that utilise information in a single frame or computed from a sequence of frames [7]. The tracking method is primarily focus on identifying the object of interest and on the detecting that object in frames [8]. Hence, current object tracking methods are exhaustive and time-consuming [9]. Vision sensors are used for object tracking by capturing images from surrounding environments in real-time. Higher frame rate allow better tracking of rapid object movements but requires higher computational power [10]. However, advanced real-time vision systems rarely exceed the standard 10-60Hz range [10].

Recent real-time object tracking algorithms are increasingly considering multi-core frameworks to improve their performance [11-13]. However, it is a challenging task to obtain good object tracking performance on multi-core frameworks by using parallelism tools. This is because these parallelism tools need to be exploited to reduce the time spent to find hotspots (most time-consuming functions in algorithm), minimise development time and address performance bottlenecks to achieve good performance. Thus, the performance of an image processing algorithm need to be boosted using multi-core frameworks to recognize and track the marked objects in high frame rate video processing better [13].

One of the parallelism tools that is considered comprehensive and can offer developers to enhance their productivity is the Intel® Parallel Studio XE [14]. This tool enables programs to use multi-core processors from Intel along with different parallelism frameworks. This tool also helps to generate, debug, develop, and track threaded as well as non-threaded applications using C++/C and Fortran programming language in Windows and Linux operating systems. Intel® Parallel Studio XE is composed of parallel software development instruments such as, parallel programming models (Intel® Threading Building Blocks (Intel® TBB) and Intel® Cilk™ Plus), advanced threading and performance profiler (Intel® VTune™ Amplifier XE), memory and threading debugger (Intel® Inspector XE), and threading prototyping Tool (Intel® Advisor XE).

Intel® Parallel Studio is used to improve the performance of a real-time object tracking algorithm by providing the following contributions:

(1) Analysing image processing algorithms to determine hotspot sections.

(2) Apply multi-threading method to improve processing time of moving object detection. This will reduce processing time as well as to meet the growing demands for computational power of object tracking algorithms.

This research work presents a novel approach to analyse a real-time object tracking algorithm using Intel® Parallel Studio XE package. The real time object tracking algorithm was implemented on two frameworks, namely OpenMP, and Threading Building Blocks (TBB) for parallelization on multi-core architectures. These two frameworks are selected for this research because they are frequently utilised in the literature [11-13].

The next section of this paper, Section 2 presents a literature review on the research work related to applications on multi-core architectures such as object tracking and image processing. Section 3 introduces parallelization frameworks, namely, OpenMP and TBB. Section 4 elaborates on the experiments and findings from the experiments. Finally, Section 6 concludes this paper.

2. RELATED WORKS

Most of the image processing applications have intensive computation operations and need large memory to achieve good performance and thus, parallelised implementation give an attractive solution [11]. There are a number of works related to the parallelisation frameworks on image processing and object tracking methods in the literature.

Bera et al. [12] reported a real time algorithm based on mean-shift and particle trackers to track pedestrians in crowded scenes at real time rates on a multi-core desktop. Their algorithm was tested with a multi-core processor (4 Cores) and compared

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

with six different algorithms and their findings were promising.

Membarth et al. [15] evaluated five different multi-core frameworks, namely OpenMP, Cilk++, Threading Building Blocks, RapidMind, and OpenCL for 2D/3D image registration. The five frameworks were carried out on a real medical imaging application based on two parallelization strategies and all these frameworks were evaluated on various aspects, namely performance, usability, and overhead. They found that there was no a single best framework that was able to obtain the best results and at the same time fulfil usability criteria and has minimal overheads.

Zhang [16] proposed a particle filter parallel to improve the traditional particle filter algorithm based on OpenMP where stages in the algorithm were run simultaneously in parallel. The simulation results indicated that the proposed algorithm had improved performance, was able to make full use of the computing power and improve the filtering accuracy.

Saha [11] implemented a 3D facial pose tracking system based on OpenMP platform. They showed how dataflow modeling techniques can be used to exploit parallelism effectively in a simple way. The parallelised implementation improved the system performance and met the required target frame rate.

Kwak [13] implemented a real-time object recognition and tracking algorithm based on integration of ORB and optical flow. The results of their implementation showed that parallelization using OpenMP improved the processing speed.

There are many parallelisation frameworks being investigated to improve the performance of the image processing and object tracking applications; however, very few studies compared the performance of different parallelisation frameworks and to recommend the best parallelisation frameworks to be used in object tracking problem. Thus, researchers face difficulties to select the best parallelization framework for their work. Additionally, there are many usability issues being identified in [2] and their findings indicate that there is a need of software tools to support programmers better in the implementation of the parallelization frameworks. In this work, we have implemented and compared two different parallelisation frameworks, namely OpenMP and BB for a real time object tracking algorithm. The parallelization frameworks were implemented with the support of Intel® Parallel Studio XE tool to minimise usability issues such as the development time and application development cost.

3. PARALLELIZATION FRAMEWORKS

This section introduces a brief background of the parallelization frameworks, namely, Open Multi-Processing (OpenMP) and Threading Building Blocks (TBB) that were implemented in this paper.

3.1 Open Multi-Processing (OpenMP)

Open Multi-Processing (OpenMP) was released by the OpenMP Architecture Review Board (ARB) in 1997. It is an Application Programming Interface that supports shared memory parallelism in C, C++, and FORTRAN programs. OpenMP has a set of compiler directives to extend C/C++ and FORTRAN compilers capabilities. Such directives, indicated with "#pragma omp", help users explicitly build parallelism using constructs such as: single program multiple data (SPMD), tasking, work-sharing and synchronization constructs. OpenMP requires specific compiler support and these directives need to be recognised and interpreted by compilers. OpenMP has several features including library functions to control and query the runtime environment [17]. OpenMP was initially designed to parallelise loop-based sequential programs based on a fork-join model. The model allows one master thread to perform tasks throughout the whole program and forks off threads to process parts of the program that needed to run in parallel [1]. Newer version of OpenMP has considered irregular constructs such as while loops and recursive structures.

3.2 Threading Building Blocks (TBB)

Threading Building Blocks (TBB) was first released in 2006 by Intel. It is a C++ template library for scalable data-parallel programming that can be used with any operating system and any C++ compiler. Data-parallel programming scales to the number of cores being used and parallel program performance scales up (increases) when more core added. TBB performs load balancing on processor resources using "task stealing" scheduler strategy and tasks are moved to less-loaded processors from busy processors [18]. In programming, tasks are much lighter weight than raw threads and thus operating on tasks are faster than operating on thread. TBB makes use of this idea and a program in TBB is described in terms of fine-grained tasks. TBB library maps the user-specified tasks onto threads and users are able to avoid tedious low level threading work [19].

Journal of Theoretical and Applied Information Technology
10th December 2014. Vol.70 No.1

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

4. EXPERIMENT AND RESULTS

The real time object tracking algorithm is implemented with the support of the Intel® Parallel Studio XE software tool. Specifically, this tool helped us to generate, debug and develop the implementation in C++. The real time object tracking algorithm was implemented on two frameworks, namely OpenMP, and Threading Building Blocks (TBB) for parallelization on multi-core architectures with the aim to reduce processing time and improve computing power for object tracking algorithm. This section is further divided into two smaller sections to describe Intel® Parallel Studio Implementation and the Performance Evaluation.

4.1 Intel® Parallel Studio Support Tools

The Intel® Parallel Studio includes Microsoft Visual Studio 2013 (VC++), OpenCV 2.4.8 and Parallel Studio XE 2013. OpenCV is a computer vision library and it is employed to access the image processing algorithms. Parallel Studio XE 2013 is a package developed by Intel which includes Intel® Threading Building Blocks, Intel® VTune™ Amplifier XE, Intel® Inspector XE and Intel® Advisor XE. The workflow of Intel® Parallel Studio XE is shown in Figure 1 [20]. A Dell XPS 17 L702X-6237 notebook with the Intel Core i7 2670QM (4 Cores) CPU and 8GB RAM was used in the implementations. The Intel Core i7 is a high-end quad-core processor and it allows hyper threading. In hyper threading or simultaneous multi-core, these four cores can handle up to eight threads.

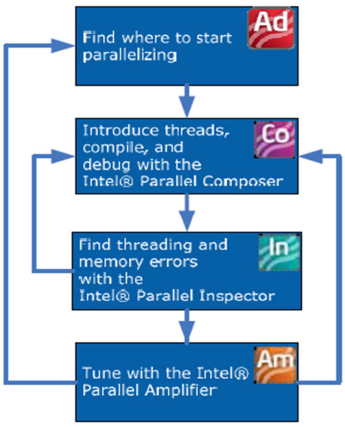

Figure 1: The Intel® Parallel Studio Workflow

The analysis tools and compiler in Intel Parallel Studio is supporting both TBB and OpenMP platforms. As shown in Figure 1, there are four basic steps to make parallel programs in Parallel Studio.

1. To trace where to start parallelizing by determining the time-consuming sections in the algorithm, such as tracing the hotspot, or most time-consuming function.
2. To introduce parallelism into the application by introducing a threading approach to the application such as OpenMP or TBB.
3. To debug the parallel program for correctness by finding and getting rid of common threading and memory errors.
4. To tune the program to ensure good thread and CPU utilization by tuning the threaded application for multi-core performance scalability, and finding the poor concurrency.

Following workflow in Figure 1, the experiments are conducted in the similar steps. First, Intel® Advisor was employed to explore top hotspot which is the most time consuming process in our algorithm. Then, Intel® VTune Amplifier was used to investigate the detail of the hotspots. As an example, when we run the object tracking algorithm with 2000 frames in serial codes (no multi-threading), we will reach to hotspots as shown in Figure 2. These functions are the more active functions in our application. Thus, tuning these hot spots functions will improve the overall application performance. Figure 2 shows the CPU time of each hotspot functions.

Top Hotspots

Function	CPU Time
RtlUserThreadStart	79.599s
func@0x180053804	16.578s
func@0x18003bda0	14.572s
func@0x18005368b	10.903s
func@0x1801f1670	7.922s
[Others]	64.411s

Figure 2: Top Hotspots

Intel® Advisor also help us to choose possible code regions. In our application, the "loop" to capture the image frame and object segmentation sections is the most time-consuming procedure as shown in the analysis in Figure 3. Thus, the loops in Figure 3 are recommended to be paralleled. Double click on the "main.cpp:83" as displayed in Figure 3, more detail of this time-consuming loop is appeared as shown in Figure 4.

Journal of Theoretical and Applied Information Technology
10th December 2014. Vol.70 No.1

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

In the second step, we implement the multi-core frameworks, namely OpenMP and TBB frameworks to parallelise the hotspot functions. The pseudo codes for the TBB and OpenMP multi-core frameworks are as below:

- TBB

```
parallel_for (blocked_range<int>(0, k),
        [=](const blocked_range<int>& r) {
      for (int i=r.begin(); i<=r.end() ; i++){
            …
            …
            }
      });
```

- OpenMP

For implementing the OpenMP in Microsoft Visual Studio, a file name as, #include <omp.h> should be included to access OpenMP functions and capabilities. An OpenMP multi-core directive is shown below:

```
#pragma omp parallel for
      for (int i=0; i<= k ; i++){
            …
            …
            }
```

In TBB and OpenMP, *k*, and *r* variables are numbers of frames and *i* is a loop counter.

In the third step, Intel Inspector was used to verify the application reliability and detect the challenging threads and memory errors. Figure 5 shows the thread errors generated by the Intel Inspector.

In the last step, Intel Amplifier was used to analyse thread performance in our application. **Error! Reference source not found.**6 shows the application tuning through Intel Amplifier. Figure 6 displays the list of functions sorted via time spent in CPU. It represents the time consumed for each function and thread.

The results for each multi-core frameworks were collected and their performance in terms of execution time was computed via *tick_count* function. The *tick_count* is used as a timestamp and it returns wall-clock timestamp [19].

Top time-consuming loops

Consider adding parallel site and task annotations around these time-consuming loops found during Survey analysis.

Loop	Source Location	CPU Total Time
main	main.cpp:83	13.6441s
main	main.cpp:132	0.0100s
std::allocator<class cv::Vec<int,4> >::destroy<class cv::Vec<int,4> >	xmemory0:606	0.0013s

Figure 3: Top time-consuming loop in our application based on Intel® Advisor

View Source

Function Call Sites and Loops	Total Time %	Total Time	Self Time	Hot Loops	Source Location
Total Unpaused	100.0%	15.3004s	0s		
RtlUserThreadStart	100.0%	15.3004s	0s		
BaseThreadInitThunk	100.0%	15.3004s	0s		
mainCRTStartup	90.5%	13.8473s	0s		crtexe.c:456
_tmainCRTStartup	90.5%	13.8473s	0s		crtexe.c:473
main	90.5%	13.8473s	0s		main.cpp:38
[loop at main.cpp:83 in main]	89.2%	13.6441s	0.4650s		main.cpp:83
cv::SparseMatConstIterator::operator++	34.8%	5.3268s	0s		
cv::SparseMatConstIterator::operator++	34.8%	5.3268s	0.0402s		
cv::SparseMatConstIterator::operator++	34.6%	5.2867s	1.5714s		
cv::SparseMatConstIterator::operator++	24.3%	3.7152s	3.3556s		
cv::Mat::MSize::operator[]	32.8%	5.0127s	0s		
cv::Mat::step1	18.4%	2.8184s	0.0531s		
[loop at main.cpp:132 in main]	0.1%	0.0100s	0s		main.cpp:132
cv::Mat::MSize::operator[]	0.1%	0.0100s	0s		

Annotations Example

Figure 4: Detail of time-consuming loop in our application based on Intel® Advisor

Journal of Theoretical and Applied Information Technology
10th December 2014. Vol.70 No.1

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

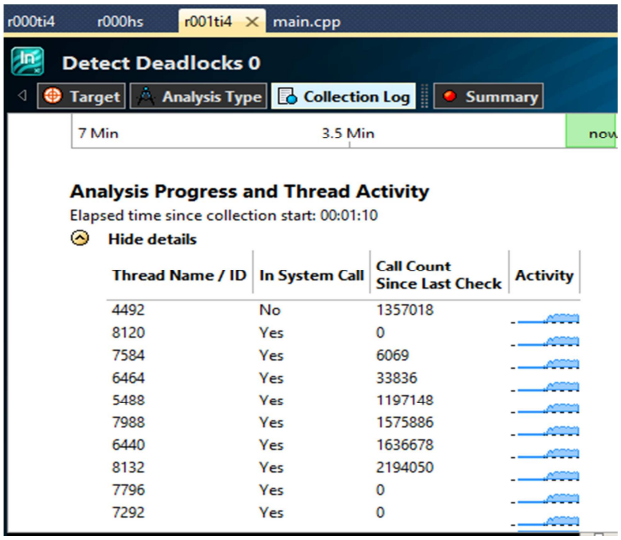

Figure 5: Intel Inspector threading validation

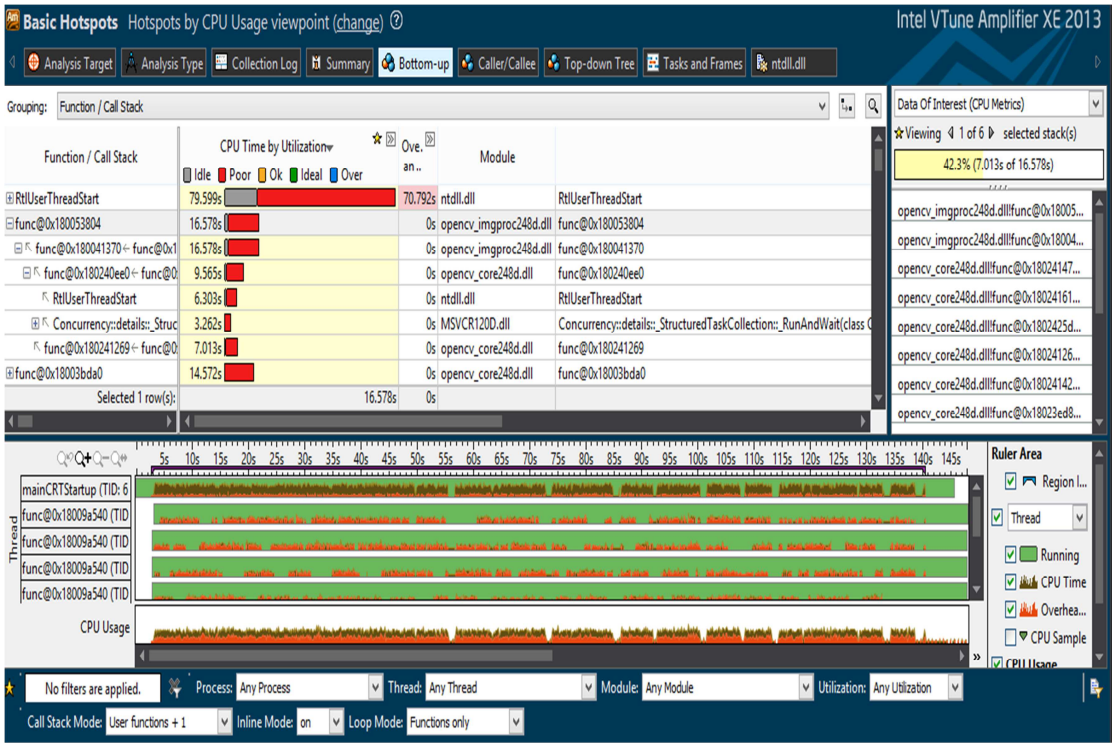

Figure 6: Application Tuning using Intel® Amplifier

Journal of Theoretical and Applied Information Technology
10th December 2014. Vol.70 No.1

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

4.2 Performance Evaluation

This section presents the performance assessment of the implementations using two multi-core frameworks, namely TBB and OpenMP and a sequential program. In this paper, speedup, efficiency and scalability are applied to evaluate the two multi-core frameworks.

4.2.1 Speedup and efficiency

In parallel computing, it is ideal to divide equal work among the available cores. This means if a parallel program with n threads/processes is to be executed with n cores, then each core is assigned with one thread or process, the parallel program can be executed n times faster than a serial program. This ideal relation is named as linear speedup. If T_{serial} is the serial program runtime, and $T_{parallel}$ is the parallel program runtime, then $T_{parallel} = T_{serial}/n$. In this context, the ratio $T_{serial}/T_{parallel}$ is the speedup parameter [18] and it is define as

$$S = \frac{T_{serial}}{T_{parallel}} \quad (1)$$

However, in practice, it is rarely possible to obtain linear speedup in parallel program. In fact, overheads increases when the number of processes or threads increases. In parallel implementation, more threads usually imply more threads need to access critical parts in a parallel program and more data need to be communicated between cores [18]. Parallel efficiency, E, which is used to describe the efficiency of the processors is defined in equation (2) where p is the number of threads.

$$E = \frac{S}{P} = \frac{\left(\frac{T_{serial}}{T_{parallel}}\right)}{p} = \frac{T_{serial}}{p.T_{parallel}} \quad (2)$$

4.2.2 Scalability

The term "scalable" is widely used for performance evaluation in multi-core systems. A parallel program is scalable when it is able to manage the increasing problem sizes. For scalability evaluation, the multi-core frameworks should be tested in different problem sizes by testing the frameworks using different frames.

In this paper, the same object tracking algorithm was considered for 100, 200, 400 and 800 frames. The processing times were recorded for similar frames based on serial computation as well as multi-core frameworks.

4.2.1 Results and Discussions

Table 1 shows the processing times for various number of frames based on OpenMP framework. The processing times for single threads are the results for the serial computation. The speedup as well as efficiency was determined based on Equation 1 and 2. **Error! Reference source not found.** and**Error! Reference source not found.** show the speed up and efficiency of the object tracking algorithm for various number of frames according OpenMP and TBB with respect to the serial computation. The speedups and efficiency of the object tracking framework based on OpenMP are plotted and illustrated in Figure 7 and 8.

Table 1: Processing time based on OpenMP framework (in seconds)

Number of frames	*Number of threads to be parallelised*			
	1	2	4	8
100	13.48	8.62	4.96	2.99
200	19.22	15.15	7.76	5.83
400	36.1	23.33	13.29	9.81
800	61.13	32.04	18.94	13.23

Table 2: Speedups (S) and efficiency (E) based on OpenMP framework

Number of frames	Speedups (S) and Efficiency (E)	*Number of threads*			
		1	2	4	8
100	*S*	1.00	1.56	2.72	4.51
	E	1.00	0.78	0.68	0.56
200	*S*	1.00	1.27	2.48	3.30
	E	1.00	0.63	0.62	0.41
400	*S*	1.00	1.55	2.72	3.68
	E	1.00	0.77	0.68	0.46
800	*S*	1.00	1.91	3.23	4.62
	E	1.00	0.95	0.81	0.58

As shown in **Error! Reference source not found.**7 and 8, S and E are dependent on the number of threads and the number of frames (the problem size). It was observed that when the number of frames increased, the speedups were also increased but the efficiencies were decreased. This showed that the processing times of both multi-core frameworks were improved when compared to the serial computation. However, as the number of threads increased, overheads of the algorithm were also increased and the efficiency for both of the frameworks were dropped. In all experiments, TBB

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

and OpenMP were able to handle large number of frames and thus both TBB and OpenMP frameworks had good scalability.

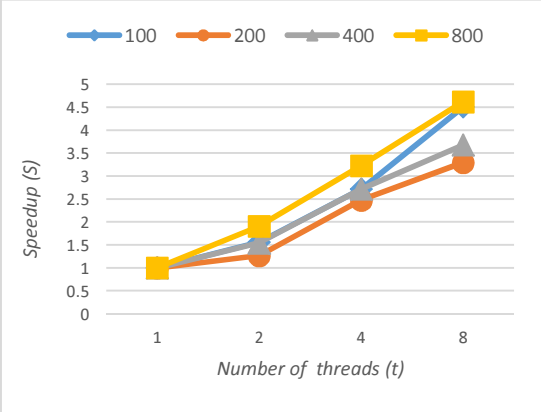

Figure 7: Speedups of parallel program using OpenMP

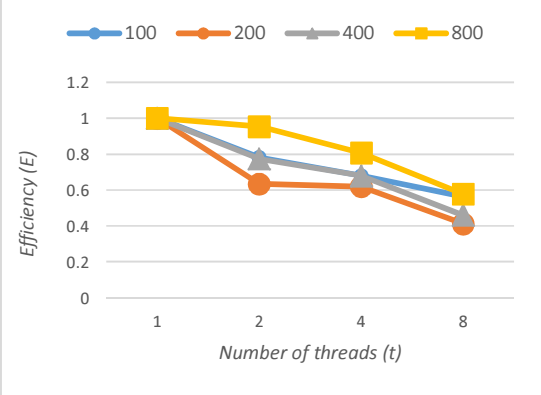

Figure 8: Efficiency of parallel program using OpenMP

Table 3: Speed ups and efficiency metrics based on TBB framework

Number of frames	Speedups (S) and Efficiency (E)	*Number of threads*			
		1	2	4	8
100	*S*	1.00	1.54	2.48	4.24
	E	1.00	0.77	0.62	0.53
200	*S*	1.00	1.26	2.53	3.34
	E	1.00	0.63	0.63	0.42
400	*S*	1.00	1.51	2.49	3.51
	E	1.00	0.76	0.62	0.44
800	*S*	1.00	1.87	3.01	4.32
	E	1.00	0.93	0.75	0.54

5. CONCLUSIONS

In this paper, we implemented a real time object tracking algorithm using two multi-core frameworks namely, Threading Building Blocks (TBB) and Open Multi-Processing (OpenMP) to compare their performance. We have analysed and improved our implementations by using Intel® Parallel Studio XE tool. This tool was able to identify the hotspots and performance bottleneck in the object tracking algorithm. The identified hotspot functions were then parallelised using TBB and OpenMP. In order to evaluate the performance of multicore frameworks, the speedup and efficiency were computed and compared in different problems sizes. Detailed experiment results showed that multi-core frameworks were explicitly faster than serial computation (one core) and OpenMP is faster than TBB in the this work.

ACKNOWLEDGMENT:

The authors would like to thank Universiti Kebangsaan Malaysia (UKM) for sponsoring this research by using the funding of the research project Industri-2013-021 and FRGS/2/2013/TK01/UKM/ 02/4.

REFRENCES:

[1] G. Slabaugh, R. Boyes, and X. Yang, "Multicore Image Processing with Openmp", *Signal Processing Magazine, IEEE*, 134-138, 2010, pp. 134-138.

[2] R. Membarth, F. Hannig, J. Teich, M. Korner, and W. Eckert, "Frameworks for GPU Accelerators: A Comprehensive Evaluation Using 2D/3D Image Registration", *IEEE 9th Symposium on Application Specific Processors (SASP)*, San Diego, CA, USA, June 5-6, 2011, pp. 78-81.

[3] T. Willhalm and N. Popovici, "Putting Intel® Threading Building Blocks to Work", *Proceedings of the 1st International Workshop on Multicore Software Engineering*, May 11, 2008, pp. 3-4.

[4] R.L.d. Carvalho, D.S.C. Carvalho, Félix Mora-Camino, P.V.M. Lima, and F.M.G. França, "Online Tracking of Multiple Objects Using WiSARD", *22nd European Symposium on Artificial Neural Networks, Computational Intelligence And Machine Learning (ESANN 2014)*, Bruges, Belgium, 23-25 April, 2014.

[5] T.S. Ling, L.K. Meng, L.M. Kuan, Z. Kadim, and A.A.B.a. Al-Deen, "Colour-based Object Tracking in Surveillance Application", *Proceedings of the International*

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

MultiConference of Engineers and Computer Scientists, March 18-20, 2009.

[6] H. Yang, L. Shao, F. Zheng, L. Wang, and Z. Song, "Recent Advances and Trends in Visual Tracking: A Review", *Neurocomputing,* Vol. 74, No. 18, 2011, pp. 3823-3831.

[7] A. Yilmaz, O. Javed, and M. Shah, "Object Tracking: A Survey", *ACM Computing Surveys,* Vol. 38, No. 4, 2006.

[8] H.S. Parekh, D.G. Thakore, and U.K. Jaliya, "A Survey on Object Detection and Tracking Methods", *International Journal of Innovative Research in Computer and Communication Engineering,* Vol. 2, No. 2, 2014, pp. 2970-2978.

[9] K.-Y. Eom, T.-K. Ahn, G.-J. Kim, G.-J. Jang, and M.-h. Kim, "Fast Object Tracking in Intelligent Surveillance System", in *Computational Science and Its Applications–ICCSA 2009*, Springer LNCS, Vol. 5593, 2009, pp. 749-763.

[10] A. Handa, R. Newcombe, A. Angeli, and A. Davison, "Real-Time Camera Tracking: When is High Frame-Rate Best?", in *Computer Vision – ECCV 2012*, Springer LNCS, Vol. 7578, 2012, pp. 222-235.

[11] S. Saha, C.-C. Shen, C.-J. Hsu, G. Aggarwal, A. Veeraraghavan, A. Sussman, and S.S. Bhattacharyya, "Model-based OpenMP Implementation of a 3D Facial Pose Tracking System", *2006 International Conference on Parallel Processing Workshops, ICPP 2006*, August 14-18, 2006, pp. 66-73.

[12] A. Bera, N. Galoppo, D. Sharlet, A. Lake, and D. Manocha, "AdaPT: Real-time Adaptive Pedestrian Tracking for crowded scenes", Technical Report, UNC Chapel Hill, 2013.

[13] J.C. Kwak, T.R. Park, Y.S. Koo, and K.Y. Lee, "Implementation of Object Recognition and Tracking Algorithm on Real-time Basis", *IEEE EUROCON*, 2013, pp. 2000-2004.

[14] Intel, Intel® Parallel Studio XE 2013 SP1, Leading Development Tools for Top Performance, 2013. Available: http://goo.gl/4HDWfD

[15] R. Membarth, F. Hannig, J. Teich, M. Körner, and W. Eckert, "Frameworks for multi-core architectures: a comprehensive evaluation using 2D/3D image registration", in *Architecture of Computing Systems-ARCS 2011*, Springer, 2011, pp. 62-73.

[16] Y. Zhang, "Particle Filter Parallel of Improved Algorithm Based on OpenMp", in *Advanced in Computer Science and its Applications*, Springer, 2014, pp. 1279-1285.

[17] P. Kegel, M. Schellmann, and S. Gorlatch, "Using OpenMP vs. Threading Building Blocks for Medical Imaging on Multi-cores", in *Euro-Par 2009 Parallel Processing*, Springer, 2009, pp. 654-665.

[18] P. Pacheco, *An Introduction to Parallel Programming*, Burlington, USA: Elsevier, 2011.

[19] J. Reinders. *Intel Threading Building Blocks: Outfitting C++ for Multi-Core Processor Parallelism*, O'Reilly Media, Inc., 2007.

[20] Intel, Getting Started with the Intel® Parallel Studio 2009. Available: https://www.microway.com.au/catalog/intel/getting_started_parallel_studio.pdf