British Journal of Mathematics & Computer Science
11(2): 1-9, 2015, Article no.BJMCS.19422
ISSN: 2231-0851

SCIENCEDOMAIN *international*
www.sciencedomain.org

A Performance Evaluation of OpenCL and Intel Cilk Plus on a Graphic Rendering Problem

Y. J. Gambo[1*], S. B. Junaidu[2], S. E. Abdullahi[2] and C. I. Saidu[3]

[1]Department of Computer Science, Federal University Wukari, Wukari, Taraba State, Nigeria.
[2]Department of Mathematics, Ahmadu Bello University, Zaria, Kaduna State, Nigeria.
[3]Bingham University, Karu, Nasarawa State, Nigeria.

Article Information

DOI: 10.9734/BJMCS/2015/19422
Editor(s):
(1) Qiang Duan, Information Sciences & Technology Department, The Pennsylvania State University, USA.
Reviewers:
(1) Anonymous, Carnegie Mellon University, USA.
(2) Anonymous, University of Port Harcourt, Nigeria.
(3) Anonymous, Kocaeli University, Turkey.
(4) Samir Kumar Bandyopadhyay, University of Calcutta, India.
Complete Peer review History: http://sciencedomain.org/review-history/11164

Original Research Article

Received: 09 June 2015
Accepted: 01 August 2015
Published: 29 August 2015

Abstract

Parallel programming is fast evolving into a central core of high performance computing. Several computing hardware vendors now embed several processors into computers to increase speed and performance thus the need to maximize and utilize computing resources for maximum performance. Rendering of graphics has also been a challenge in terms of speed that architects and graphic experts have found it difficult to render graphics in a timely manner at minimal cost. In this paper, a comparative study was carried out on two recent and powerful multiprocessor parallel programming languages OpenCL and Intel Cilk Plus which were used to separately program an embarrassingly parallel graphic rendering algorithm, raytracing. The programs were ran and profiled on three different computers with varying specifications on windows using Microsoft Visual Studio 2012 as IDE. The result showed that OpenCL was consistent after first run without restart than Intel Cilk Plus and as raytracing depth increased the performance gap between OpenCL and Intel Cilk Plus also increased with OpenCL showing the better performance.

Keywords: OpenCL; intel cilk plus; parallel programming; raytracing; high performance computing; rendering.

1 Introduction

Speed and performance have become the critical factors in defining computing systems and the Internet in this contemporary time. Computers have moved from being embedded with just a single processor to

**Corresponding author: Email: yaknanjohn@gmail.com;*

multiple processors over the years. This is all in a bid to increase speed and performance. Parallel computing is one of the most exciting technologies to achieve prominence since the invention of electronic computers [1]. Parallel computing can be seen as the use of a parallel computer to reduce the time needed to solve a single computational problem [2]. Many scientific and technological tasks today demand high computing power to solve. One of such areas is the area of graphic rendering. This research exploits the power of parallel computing in improving the performance of ray tracing algorithm in the domain of graphic rendering.

The rise of multicore is bringing shared-memory parallelism to the masses. The community is struggling to identify which parallel models are most productive [3].

However, with this proliferation of parallel computing devices comes the challenge of programmability. Programmers must consider data dependencies, race conditions, communications, etc. As the available parallel computing resources go beyond just the CPU, even more programming complexities arise. To access all available resources, the same routine or algorithm may need to be coded multiple times and in multiple ways. The programmer must consider various type-, vendor-, or platform-specific programming models and/or APIs [4].

A single source code that could be portable across all platforms is a challenge. A better way is needed. In 2008, Apple Computer proposed a draft specification for OpenCL (Open Computing Language) [5] to the Khronos Group. The Khronos group explains that OpenCL lets programmers write a single portable program that uses all resources in the heterogeneous platform [6]. The Khronos Group also asserts that OpenCL is an open royalty-free standard for general purpose parallel programming across CPUs, GPUs and other processors. OpenCL presents programmers with a standard API with which to program; participating vendors write their device drivers to conform to its specification. This abstraction enables programmers to focus less on device-specific programming details. Also, OpenCL works on different hardware, but the software needs to be adapted for each architecture [7].

Intel Cilk Plus is the easiest, quickest way to harness the power of both multicore and vector processing. It is an extension of C and C++ programming languages, designed for multithreaded parallel computing [8]. On July 31, 2009, Cilk Arts, producers of the Cilk++ programming language, announced that its products and engineering team were now part of Intel Corporation. Intel and Cilk Arts integrated and advanced the technology further resulting in a September 2010 release of Intel Cilk Plus. To further make parallel programming easier, Phoronix [9], said that Intel is planning on implementing the Intel Cilk Plus C and C++ extensions to the GNU Compiler Collection (GCC). This is already a reality as GCC 4.9 has Intel Cilk Plus implemented.

Different strategies have evolved over the years on how to develop a parallel application. According to Luis [10], there are three (3) basic strategies:

a. **Strategy 1: Automatic Parallelization**, whose ultimate goal is to relief programmers from parallelizing task. It takes rough codes and produces efficient parallel object code with little or no additional work by the programmer. The strategy is described in Fig. 1.1:

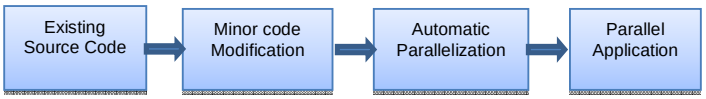

Fig. 1.1. Automatic Parallelization Strategy (Luis, 1999)

b. **Strategy 2: Parallel Libraries**: This approach has been more successful than the previous one. The basic idea is to encapsulate some of the parallel code that is common to several applications into a parallel library that can be implemented in a very efficient way. Such a library can then be reused by several codes. This is described in Fig. 1.2.

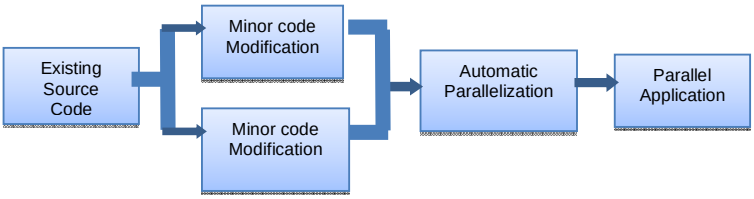

Fig. 1.2. Parallel Libraries Strategy (Luis, 1999)

c. **Strategy 3: Major Recoding,** the third strategy, which involves writing a parallel application from the very beginning as illustrated in Fig. 1.3 below, gives more freedom to the programmer who can choose the language and the programming model. However, it may make the task very difficult since little of the code can be reused.

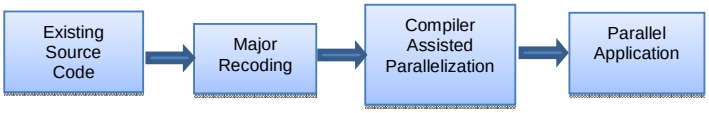

Fig. 1.3. Major Recoding Strategy (Luis, 1999)

Programmers have often opted for parallel library strategy for creating parallel applications. For these programmers, the extensions to existing languages or run-time libraries are a viable alternative. In this research work we are going to consider OpenCL and Intel Cilk Plus, Parallel Libraries that have evolved as powerful extensions of C++.

OpenCL and Intel Cilk Plus are fast taking center stage in parallel programming and this research carries out a performance evaluation of the two in the domain of parallel graphic rendering.

1.1 Research Objective

The objectives of this research are as follows:

i. To design a parallel raytracing engine using OpenCL and another Using Intel Cilk Plus.
ii. To compare the performance of OpenCL implementation against that of Intel Cilk Plus.

2 Materials and Methods

In this paper a detailed study into graphic rendering and the impact of high performance computing was carried out. The performance of two (2) key parallel programming tools, OpenCL and Intel Cilk Plus were tested on raytracing algorithm for graphic rendering

Table. 1.1. Software Implementations

Implementation	Compute device	Compute cores	Execution style	Alias
C++	CPU	Single Core	Recursive	GCC CPU Recursive
C++	CPU	Single Core	Iterative	GCC CPU Iterative
OpenCL C	CPU	Multi core	Iterative	OpenCL CPU
Intel Cilk plus	CPU	Multi core	Iterative	Intel Cilk CPU

Four (4) software implementations were designed as shown in Table 1.1. The Recursive and Iterative C++ implementations are for benchmarking purposes only. The implementations were ran on three (3) different multi core/processors system with the following specifications:

a. **Hardware Specification:** The computers used in this work were selected randomly because Intel and Khronos group the makers of Intel Cilk plus and OpenCL respectively expect these software to work on any multicore or multiprocessor computers that are now very common. The hardware specification are as shown in Table 1.2 :

Table 1.2. Hardware Specification

	Computers		
	PC One	**PC Two**	**PC three**
Make	HP	Dell	Samsung
OS	Windows 8	Windows 7	Windows 7
Processor	Dual AMD E-350 processor 1.6GHz	Intel Core i3-2310M CPU @ 2.10GHz	Intel Core i3 CPU M 370 @ 2.40GHz 2.40GHz
Installed Memory	3.00GB (2.60GB Usable)	2.00GB	4.00GB (3.80 GB Usable)
System Type	32 bits OS	64-bit OS	64-bit OS

b. **Software Specification:** The software specification of the systems used in this research are shown in Table 1.3:

Table 1.3. Software Specification

	Computers		
Software	**PC One**	**PC Two**	**PC Three**
OS	Window 8	Windows 7	Windows 7
IDE	Visual Studio 2008	Visual Studio 2008	Visual Studio 2008
OpenCL Version	AMD APP SDK v 2.7	Intel_sdk_for_ocl_application _2012	Intel_sdk_for_ocl_application _2012
Compiler	Intel C++ Studio XE 2013 update4	Intel C++ Studio XE 2013 update4	Intel C++ Studio XE 2013 update4

All the implementations were ran several times without restarting and the timing for each implementation was stored in separate files. These results were then put in tables and then used to plot graphs in order to analyze and draw conclusions. It is important to reiterate here that the focus of this research work is on the OpenCL and Intel Cilk Plus implementations. The Implementations were run 10 times without restarting at different rendering depths of 0, 1, 2, 3, 4 and 16.

3 Results

Here we examine the implementations at the different ray-tracing depths on all the three (3) test computers.

a. Ray-tracing Depth of 0: The results obtained at this depth are as shown in Fig. 1.4.

Fig. 1.4 clearly shows that the different implementations have slightly different behaviors on the different test computers. The implementations on PC One and PC Three show a more uniform behavior compared to that of PC One, at rendering depth of 0, where there is theoretically no ray fired into the rendering scene. It is the point of minimal computational demands. It was noticed that at first run across all the PCs OpenCL showed a rather higher run time, this can be attributed to the initial time OpenCL takes to setup its computing devices, context and command queues. At this depth Intel Cilk Plus seems to be performing better than OpenCL. Overall, these differences can be attributed to the memory management of the Operating Systems and hardware specifications.

PC One which has an AMD processor showed a rather haphazard behavior especially on the recursive implementation. This can be explained by the fact that recursive algorithms utilize more memory than iterative ones. Also, the processor in PC One is much slower than those in PC Two and PC Three.

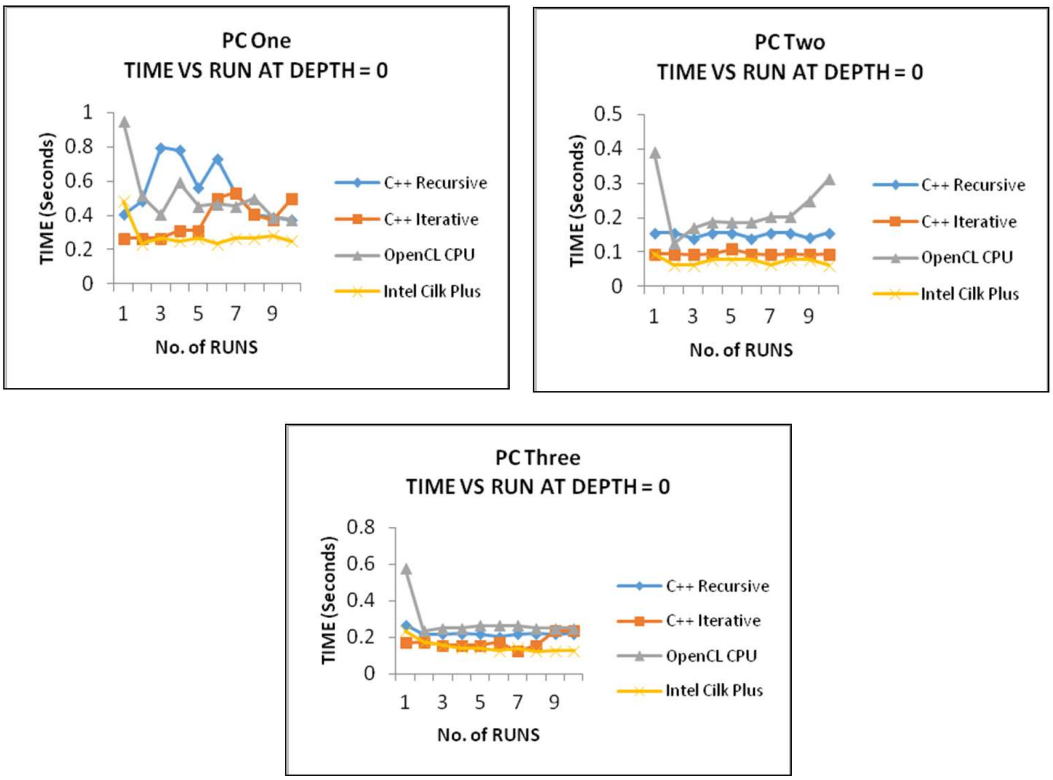

Fig. 1.4. Ray-tracing at Depth = 0

b. Ray-tracing Depth of 1: The results obtained for ray-tracing at depth of 1 is as shown in Fig.1.5.

At this depth there are no secondary rays due to reflection in the scene. Here Intel Cilk Plus implementation showed varied results on the three (3) test computers but the OpenCL CPU implementation showed a more consistent behavior across the test computers. It was noticed that at first run (RUN = 1) OpenCL takes a lot of time but stabilizes from the second run (RUN = 2). Again, this can be attributed to the time it takes OpenCL to setup the computing devices, context, and command queues. Clearly, OpenCL CPU performed far better than Intel Cilk Plus implementation at this ray-tracing depth.

c. Ray-tracing Depth of 2: Fig. 1.6 shows the performance of the implementations on the three (3) test PCs.

Here again, OpenCL CPU performed better than Intel Cilk Plus in all test computers. The C++ Iterative and OpenCL CPU implementations show more consistency in all the test computers but the C++ Recursive and Intel Cilk Plus showed different behaviors. Perhaps this is due to the impact of the Operating System and the hardware specifications especially demands on memory.

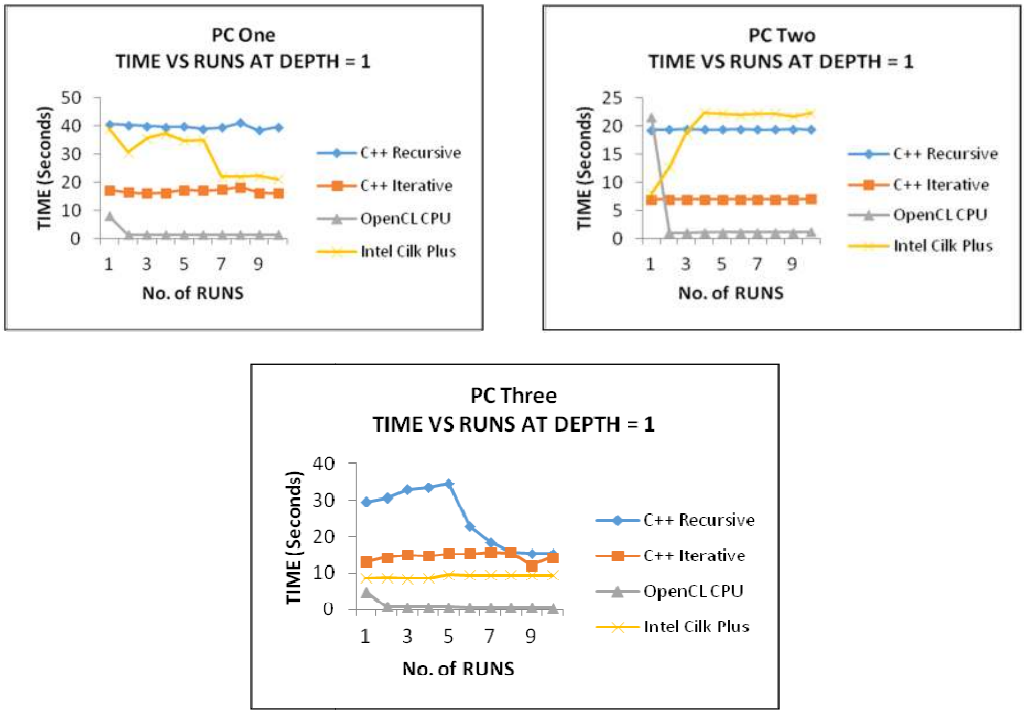

Fig. 1.5. Ray-tracing at Depth = 1

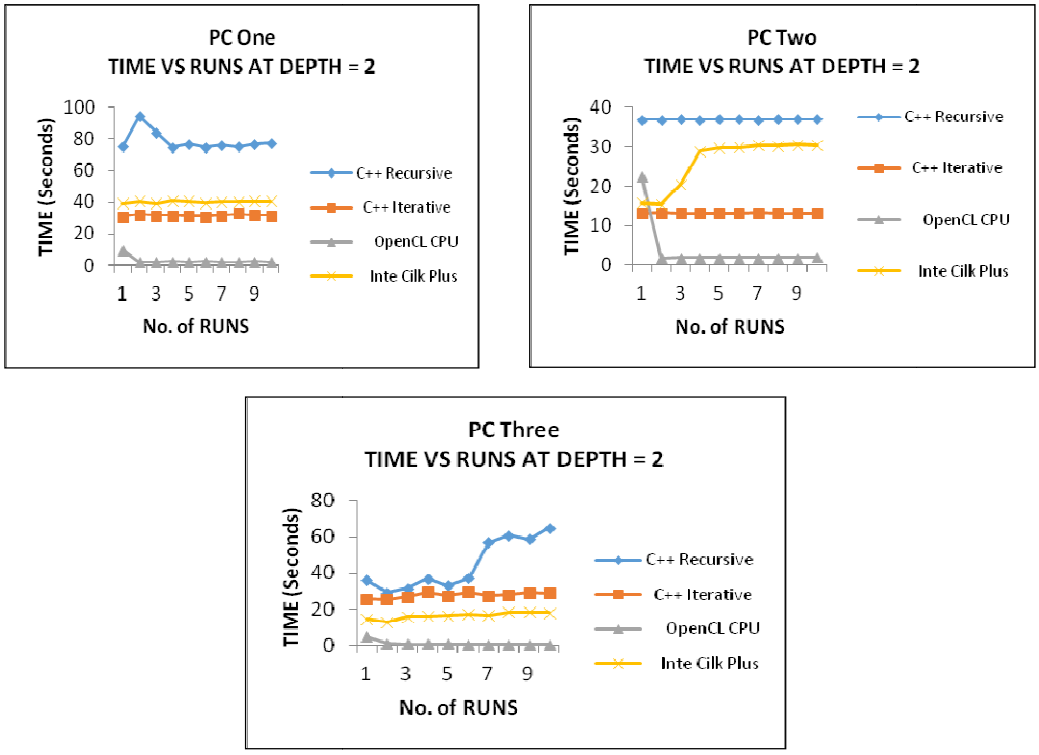

Fig. 1.6. Ray-tracing at Depth = 2

d. Ray-tracing Depth of 3: Fig. 1.7 shows the performance of the implementations on the test PCs.

Fig. 1.7 shows the rendering result at ray-tracing depth = 3 on the three (3) test computers. Here OpenCL CPU implementation still performs better than Intel Cilk Plus and it seems like as computational demands increases OpenCL CPU still remains consistent and stable. This is slightly not so with the Intel Cilk Plus implementation.

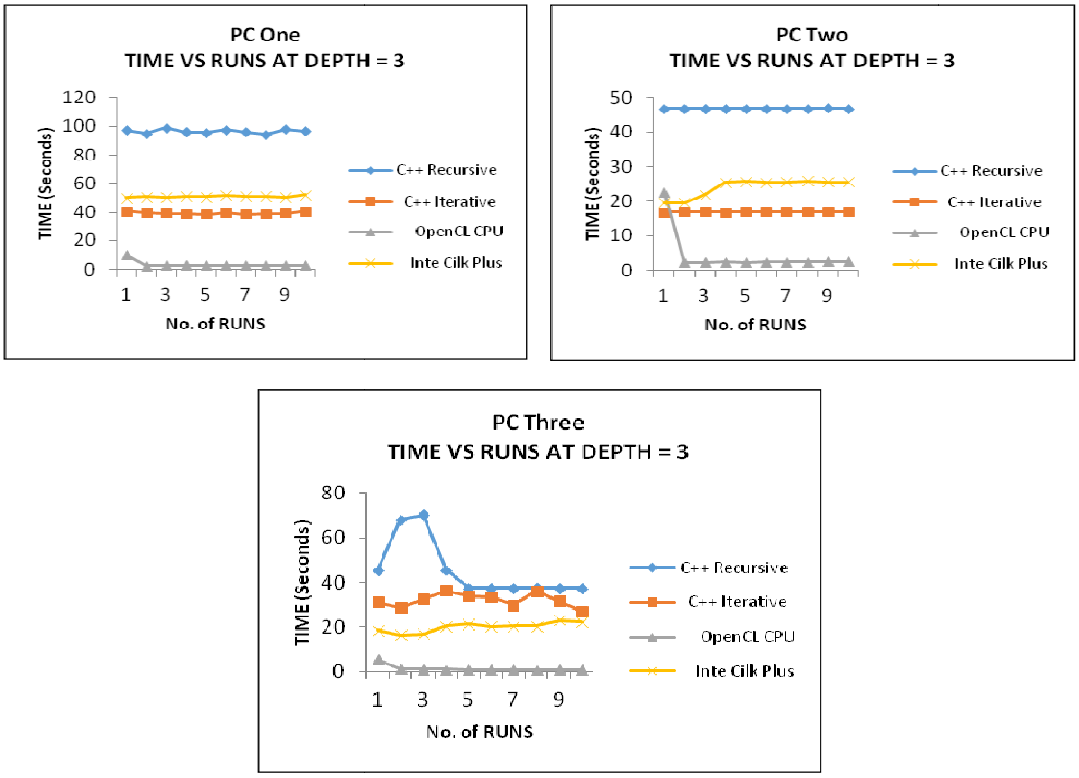

Fig. 1.7. Ray-tracing at Depth = 3

PC Three seems to give a more accurate result compared to PC One and PC Two. This is because a parallel program is expected to be faster than its serial version. Perhaps there are still some race conditions in the Intel Cilk Plus implementation or perhaps as the number of processors increase, the Intel Cilk Plus implementation will perform better.

e. Ray-tracing Depth of 4: Fig. 1.8 shows the performance at this ray-tracing depth.

Here the consistency of OpenCL cannot be equaled across the test computers. Again PC Three gave a more acceptable result because it has more computing device clocked at 2.4GHz and the more computing device the more parallel implementations perform better to a certain limit.

f. Ray-tracing Depth of 16: Fig. 1.9 shows the performance at ray-tracing depth of 16 the maximum depth for our implementation.

Again, PC Three gives a more realistic result at ray-tracing depth of 16. Also, OpenCL CPU still out performs Intel Cilk Plus at this depth. It was observed that the average time of run increased as the rendering depths increased.

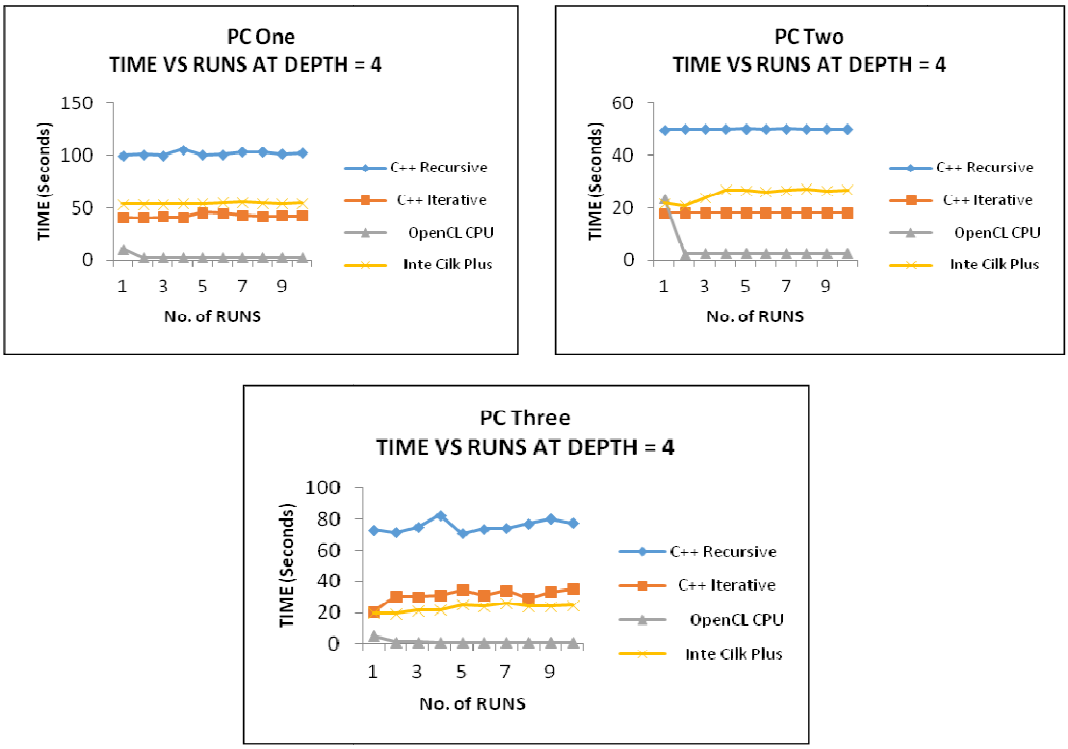

Fig. 1.8. Ray-tracing at Depth = 4

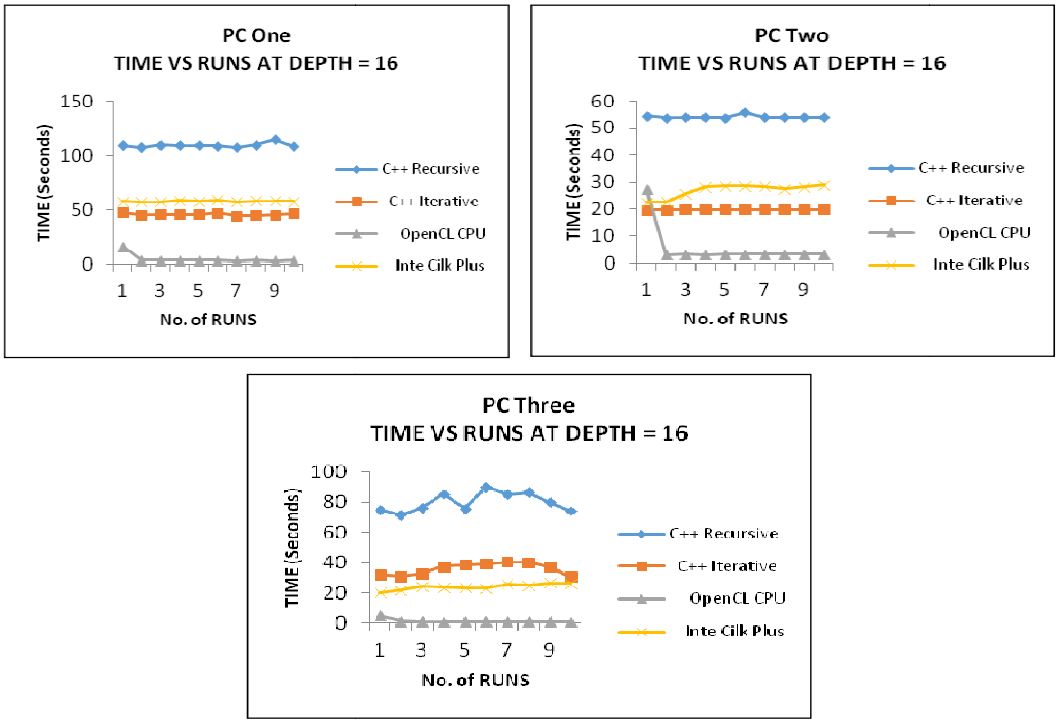

Fig. 1.9. Ray-tracing at Depth = 16

4 Conclusion

The research shows that at all the varied depths, OpenCL showed more consistency after the first run without restarting than Intel Cilk Plus. Also, across all rendering depths OpenCL seems to take longer time at first run before evening out. Overall a more acceptable result is that of PC Three because parallel implementations are expected to outperform serial ones especially in an embarrassingly parallel problem like we have in this research. It would be interesting to see the performance of OpenCL and Intel Cilk Plus implementation of ray-tracing in a massively parallel system.

Finally, it was observed that as the ray-tracing depth increases the performance difference between the parallel and the serial implementations widens as seen in PC Three but between the two parallel implementations, OpenCL performed better than Intel Cilk Plus from these results.

Competing Interests

Authors have declared that no competing interests exist.

References

[1] Lester BP. The Art of Parallel Programming. Prentice Hall; 1993.

[2] Quinn MJ. Parallel Programming in C with MPI and OpenMP. McGraw-Hill; 2004.

[3] Lorin H, Victor RB. A preliminary empirical study to compare MPI and Open MP. 2011;ISI-TR-676.

[4] CUDA NVIDIA Corporation; 2011 [Online].
Available:http://www.nvidia.com/object/cuda_home_new.html

[5] Khronos, Khronos Group, 2010, June. Open CL Introduction and Overview; 2010. [PDF,Online].
Available:http://www.khronos.org/assets/uploads/developers/library/overview/OpenCL-Overview-Jun10.pdf

[6] Khronos Open CL Working Group. (2010, Sep.). Open CL Specification. Version: 1.1. Revision: 36. [PDF, Online].

[7] Stream Computing Performance Engineers (Online Accessed 13 Nov. 2012)
Available: http://www.streamcomputing.eu/blog/2011-06-22/opencl-vs-cuda-misconceptions/

[8] Intel (2012) Intel Cilk Plus Open Source Retrieved 5th December, 2012.
Available: fromhttp://www.cilkplus.org/

[9] Phoronix, (2012), Easy Parallel Programming: Cilk plus Ported to GCC Retrieved 5-12-2012
Available: http://www.phoronix.com/scan.php?page=news_item&px=OTc5NQ

[10] Luís M. E., Rajkumar B. Parallel Programming Models and Paradigms in R. Buyya (ed.), High Performance Cluster Computing: Architectures and Systems. 1999;2. Prentice Hall PTR, NJ, USA, 1999; Chapter 1 Pg. 9.

Peer-review history:
The peer review history for this paper can be accessed here (Please copy paste the total link in your browser address bar)
http://sciencedomain.org/review-history/11164