International Journal of Hybrid Information Technology
Vol.2, No.3, July, 2009

UML Modeling and Performance Evaluation of Multithreaded Programs on Dual Core Processor

Dr. Vipin Saxena[1] and Manish Shrivastava[2]

B.B. Ambedkar University (Central University), Lucknow, India
[1]vsax1@rediffmail.com, [2]mshrivastava@yahoo.com

Abstract

Modern Object oriented programming languages provide the facility of multithreading programming which provides concurrent execution of multiple threads within the same program. Basically, threads provide a way to execute code in parallel within the same program. In high performance computing today, the multi-core CPUs have become more common in nearly all computer systems. These processors have multiple execution cores on a single physical chip. They provide parallelism between instructions and operations. Therefore, the performance measurement of multithreaded programs on these processors is an important aspect.

In the present paper, the detailed architectural modeling of a Dual Core processor is done by the use of well known modeling language i.e. the Unified Modeling Language (UML). The UML design for thread execution is also done. On the basis of UML design, performance of multithreaded programs written in JAVA and C# are evaluated and a comparison between these two is also reported through the table and the graphs.

Keywords: *Dual Core processor, Object-oriented programming, Multithreading, Performance measurement, UML.*

1. Introduction

Object-oriented approach is the most important technique in today's software development. The Object-oriented programming improves code reusability, code maintainability and provides high level of abstraction. All these features are quite suitable for the development of parallel and concurrent applications. In Object-oriented technique, the interactions and message passing among various physically distributed objects is done very efficiently. This feature provides an effective way of simultaneous execution of multiple processes. A thread is a lightweight process. It specifies a lightweight flow that can be executed concurrently with other threads within the same process [1]. All the threads within a process share the same memory space of the enclosing process. Using threads, the performance of an application can be improved by executing multiple threads concurrently.

Modern Computer systems support Dual Core CPUs. Dual Core is an architecture that refers to a Central Processing Unit with two complete execution cores in a single processor. The two cores, their caches and cache controllers are all built together in a single IC. A Dual Core processor executes two threads by running each thread on a different core. Thus the Dual Core processors improve multithreaded throughput, and delivering the advantages of parallel computing to properly threaded mainstream applications [2].

The Unified Modeling Language (UML) released by the Object Management Group (OMG), is a graphical language and has been widely accepted as a standard way for modeling an Object-oriented software system. It can also be applied for modeling of business processes and hardware system architecture and design. The details and good description of the UML notations are given in Booch et al. [1] and Alhir [3]. It also provides extension mechanisms using stereotypes and profiles which can be applied in more domain specific modeling of a system. There are some research papers where UML modeling is described in Computer architecture. Gomma [4] has developed a UML based Concurrent Object Modeling and Architectural Design Method for designing real-time and distributed applications. The UML based modeling of parallel and distributed systems for performance oriented applications, is described by Pllana, S. and Fahringer, T. [5]. Saxena et al. [6] proposed the UML model for the Multiplex system for the process which are executing in distributed environment. In their paper, Fateh Boutekkouk et al. [7] presented a new UML-based methodology for embedded applications design and architectural modeling including the CPU model, the Memory model etc. using stereotypes. An estimation technique of performance is also proposed.

There are many Object-oriented languages for commercial software development. Among these languages, C# and Java are most popular and powerful in today's programming environment. Besides all the features of an Object-oriented language, both Java and C# have built-in support for multithreading. There are very few papers available on quantitative performance comparison of Object-oriented programming languages. However many papers explain the comparison of various programming languages based on their features, technical similarities, differences, and capabilities. Henderson Robert and Zorn Benjamin [8] compared the run-time efficiency and compilation time of the language implementations of four modern programming languages that support object-oriented programming (Oberon-2, Modula-3, Sather and Self), and compared them with C++ also**.**

Glyph Lefkowitz [9] performed a comparison of execution speed between Java and Python by using running some test-cases on Linux platform. Brosgol, Benjamin M. [10] compared the concurrency-related facilities in Ada and Java, focusing on their expressive power, support for sound software engineering, and performance. In their paper, Bulpin and Pratt [11] presented measurements of the performance of a real simultaneous multithreading system. The experiments were conducted on an Intel Pentium 4 Xeon based system running the Linux 2.4.19 kernel. Figueroa, M. [12] presented results of a study to compare Java and C# programming languages features in terms of portability, functional programming and execution time in image processing programming area. Sestoft, P. [13] compared the numeric performance of C, C# and Java on three small cases. The matrix multiplication, a division-intensive loop and a polynomial evaluation were taken as case studies. Recently Saxena and Arora [14] reported a performance evaluation for object oriented software systems using VC++ and C#. The evaluation is done on nodes, equipped with Pentium D and Core 2 Duo processor technologies.

In the present paper, we have measured the execution time of threads using Java and C# on Intel's Dual Core processor. A performance comparison of multithreaded programs consisting of varying number of threads in Java and C# is reported. As a case study, a common multithreaded program is developed in both these languages. The run time of each program is measured for quantitative comparison of performance of these languages. Modeling of the system architecture and threads processing using the UML is also done and presented. The UML stereotypes for the process, thread and executing cores are defined. UML class diagrams and sequence diagrams are designed for modeling of thread execution.

2. Background

2.1. Concept of Process and Thread

A process is a program in execution. It can be defined as a group of instructions of a program which are assigned to a processor for execution. It specifies a heavyweight flow that can execute concurrently with other processes. A thread is a block of code that runs in concurrent with other threads within the same process. Each thread is a single sequential flow of instructions. It is considered as a lightweight process. When a thread is created, a new thread of control is added to the current process. The threads are easily handled in Object-oriented way. In this, all the threads in the process run simultaneously, and can access the same objects to implement their functionality. They can also communicate to other threads via shared objects.

In the UML, each independent flow of control is modeled as an active object. An Active object (instance of an active class) models the concurrent behavior of real world objects. Active objects own an execution thread and can initiate control activities. Active classes can be implemented by heavyweight processes with their own address space or by lightweight threads sharing the same address space. An active class is represented as a rectangle with thick lines [1].

2.2. UML Representation of Process and Thread

The Figure 1 shows the detailed UML design of a process and a thread and their relationship using the class diagram. In this design, we have used UML stereotypes for defining a process and a thread namely <<process>> and <<thread>>. The stereotypes defined active classes of process and thread namely the classes Process and Thread. The class Process contains multiple instances of the class Thread and described as 1..n relationship in the diagram.

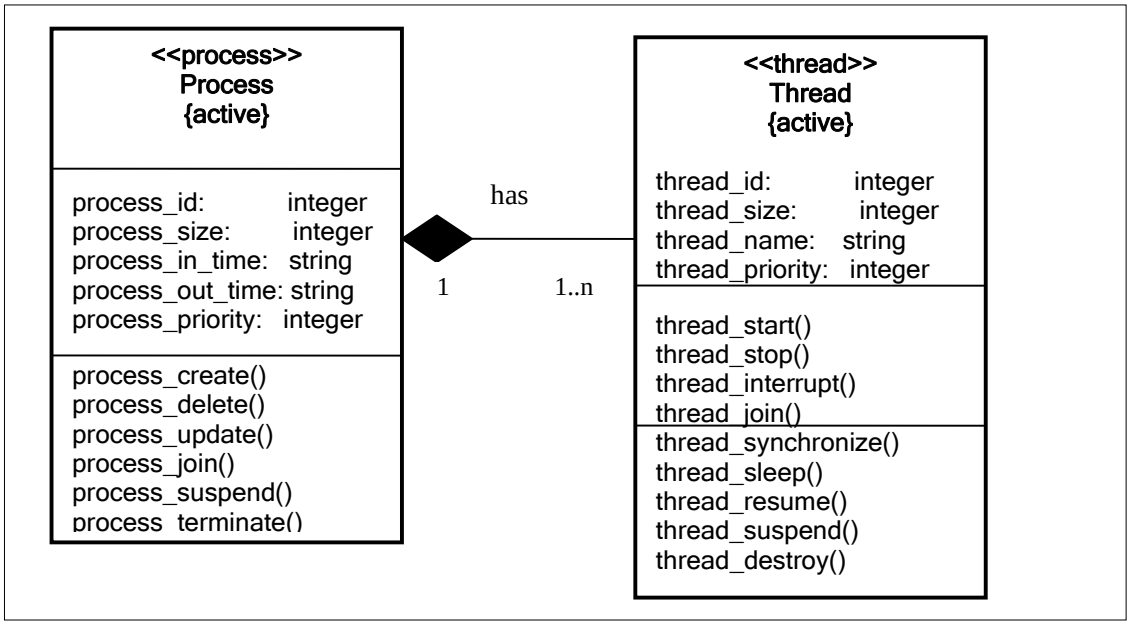

Figure 1. UML Class Definition of Process and Thread

2.3. UML Representation of Execution Cores

The Intel's Dual Core architecture is based on the Intel Core micro-architecture that uses CMP (i.e. core multi processors) technology, where two or more CPUs (known as Cores) share a single chip. In this architecture, processors move blocks of hundreds or thousands of instructions into cache before executing them in blocks of four or more at a time, trying to execute many complex instructions in one clock tick.

The UML modeling for representing the execution cores performed using UML. The UML stereotypes for the execution cores are defined. The Figure 2a shows the UML stereotype <<Execution_core>>, which is derived from the Base class. The Figure 2b shows the class diagram for representing a core and the Figure 2c shows the single and multiple instances of cores.

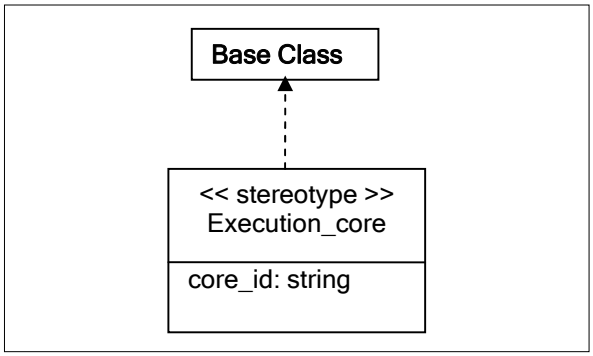

Figure 2a. Stereotype of Execution Core

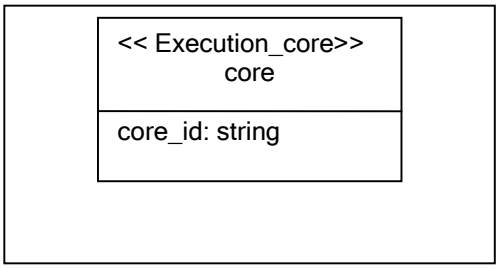

Figure 2b. Class diagram of Core

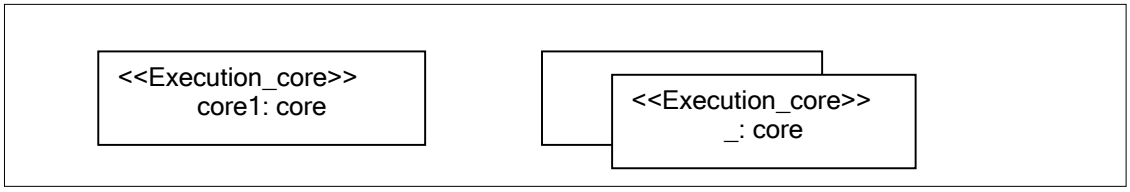

Figure 2c. Single and Multiple instances of Core

3. UML Modeling for Multithreaded Programs

3.1. UML Modeling for Processor Architecture

The Intel's Core micro-architecture technology provides more efficient decoding stages, execution units, caches, and buses for increasing the processing capacity, reducing latency and thus achieving high performance. The architectural details of Dual Core are described in [15] and [16]. The Figure 3 shows the complete architectural model of Dual Core processor architecture. The class Process is interacting through the class Thread to the class Process_Execution_Controller (PEC), which is responsible for the execution of the assigned task. The PEC is controlling the processes by message exchanging between the classes Processor and Memory. The PEC is also responsible for the threads controlling. The Processor class contains two cores, i.e. Core1 and Core2 and each core contains many components responsible for process execution as shown in the figure. The class diagram of the entire memory unit is also shown in the figure. Here class L2_Cache is shared between two cores and caches instructions through the class I_ Cache whereas the class D_Cache is responsible for caching the data, which is a sub class of L1_Cache.

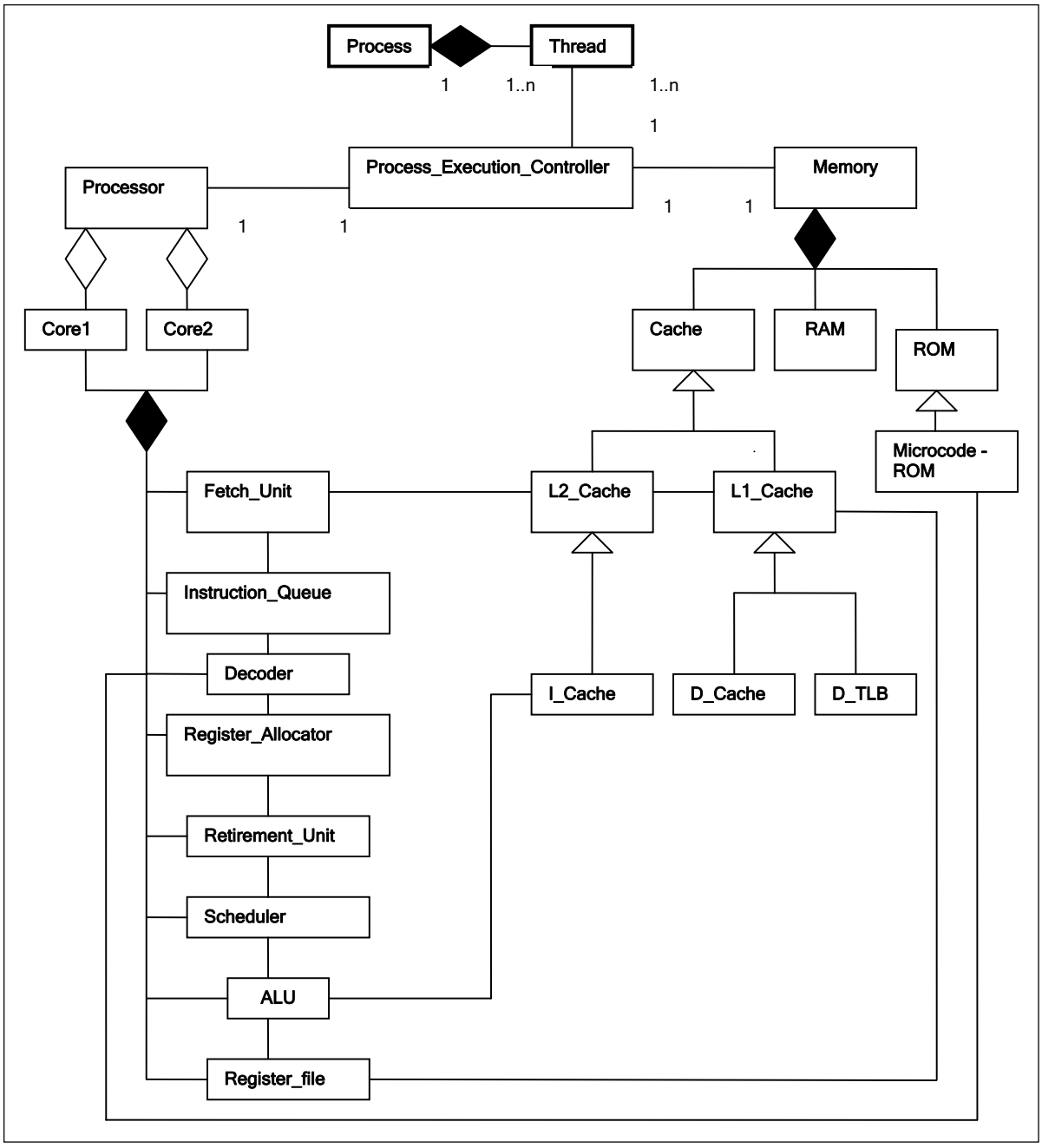

Figure 3. UML Class Diagram for Processor Architecture

3.2. UML Modeling of a Thread Life Cycle

Each thread has several states and passes through these states during its life cycle. In Java and C#, the states are the same but defined using different names. We summarize and draw a general UML model for understanding the thread life cycle. Immediately after creating a new

thread does not start its execution. Threads in executing state on a CPU are called active. A thread is Active if it is running, and actually occupies a processor at the current time. This normally starts when it is given a start operation instruction. A thread is suspended if the thread has been suspended and not doing any processing. Similarly it can go into a sleep state for a prescribed time and then after resume its running. A thread is terminated if the thread has finished execution. Finally it can be destroyed. The UML provides a graphical representation of states, transitions, events and actions using its State chart diagram. Figure 4 shows a simplified state chart diagram for thread states transitions.

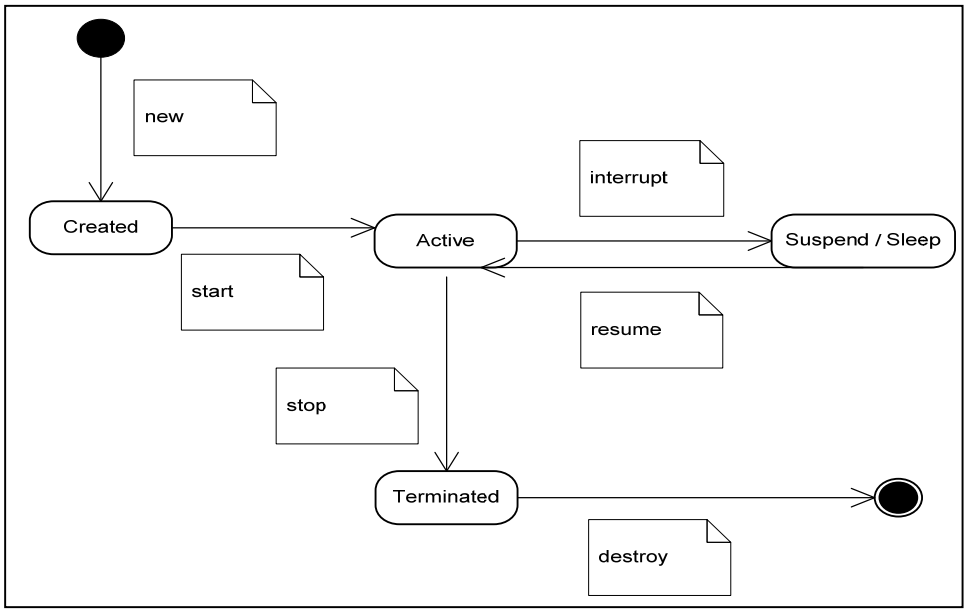

Figure 4. UML State Chart Diagram for Thread Life Cycle

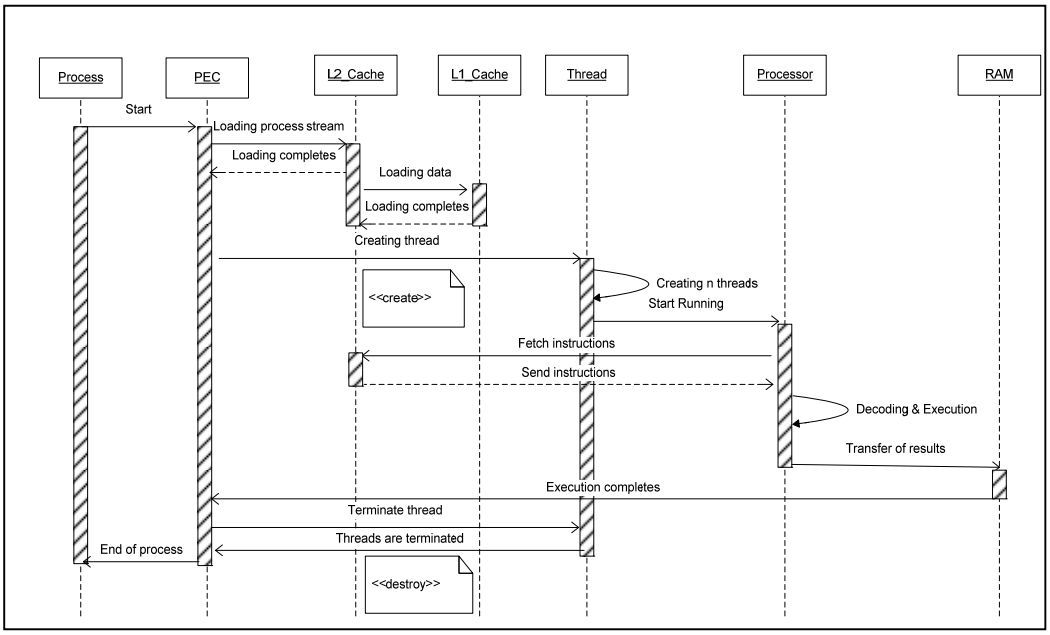

Figure 5. UML Sequence Diagram for Thread Execution

3.3. UML Sequence Diagram for Thread Execution

The UML sequence diagram for thread execution inside a core is shown in figure 5. Here the messages are exchanged among various class objects like PEC, L2_Cache and L1_Cache are shown. Instructions are fetched from L2_Cache, decoded into the executable micro operations. The data are loaded from L1_Cache. Here PEC creates multiple threads and the Processor executes the threads. After execution, the final results of these operations are passed to the RAM. The two standard stereotypes namely <<create>> and <<destroy>> are used to represent the creation and final destroying the threads.

4. Experimental Results and Discussions

The experimental results are obtained by executing a common code written in Java and C# as a console application. A sample code for creating threads repetitively inside a loop is taken to evaluate the performance. Both Java and C# have built-in class libraries for supporting the threads management. In Java, there is a set of classes in java.lang package that allows programmers to create and manage threads. Similarly, in C# there is a System.Threading namespace which contains required classes that allow thread creation and manipulation. In this experimental work, the C# program was developed and executed using Microsoft.Net framework v2.0.50727. The Java program was developed and executed using JDK1.5.0_11. We measured the execution time spent in a critical loop where multiple threads are created. The experiment was conducted on a Dual Core processor. The architectural specifications of the system are given in table 1 below. All the experimental results are averaged from 5 different runs. Table 2 shows the execution time computed in milliseconds and the comparison between average execution times of multithreaded programs in Java and C#. Based on the experimental results, it is clearly shown that C# gives better results (performs better) and is more efficient Object-oriented programming language in comparison to Java. It is clear from the table 2 that the execution time is lesser in case of C# in comparison to Java.

Table 1. Architectural details of Pentium Dual Core Processor

Specifications	Intel® Pentium® Dual Core CPU
Number of cores	02
Model number	E2180
Clock speed	2.00 GHZ
FS Bus speed	800 MHZ
Level 1 cache size	2 x 32 KB instruction caches, 2 x 32 KB data caches
Level 2 cache size	shared 1 MB
Instruction sets	MMX instruction set, SSE, SSE2, SSE3, EM64T
Memory size	1 GB
Operating System	Windows XP Professional, Ver. 2002, Service pack2
Make	HP Compaq

Figures 5a clearly displays above results in the form of graph as a performance comparison in terms of execution time of programming codes having 10, and 10^2 threads and the figure 5b indicates the same for 10^3 and 10^4 threads.

Table 2. Execution Time of Threads

	C#				JAVA			
No. of Threads	10	10^2	10^3	10^4	10	10^2	10^3	10^4
Execution Time in Milli Seconds	15.625	78.125	406.250	3703.125	18.213	97.267	686.387	6658.001
	15.625	78.125	406.250	3671.875	18.290	97.373	687.690	6521.507
	15.625	78.125	406.250	3687.500	18.266	97.450	680.114	6676.244
	15.625	78.125	390.625	3718.750	18.187	98.484	684.449	6698.573
	15.625	78.125	406.250	3703.125	18.254	97.883	685.326	6494.581
Average execution time (in milli seconds)	15.625	78.125	403.125	3696.875	18.242	97.691	684.793	6609.781

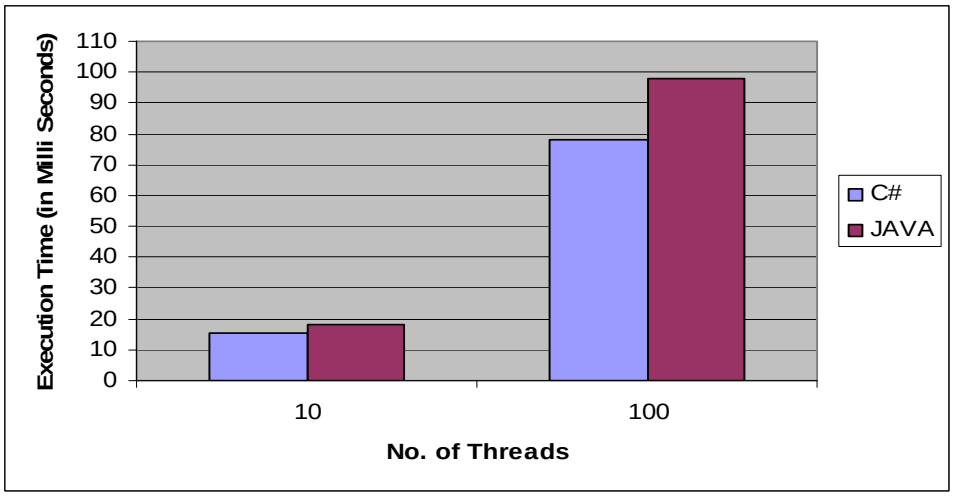

Figure 5a. Performance comparison for 10 and 100 Threads Execution

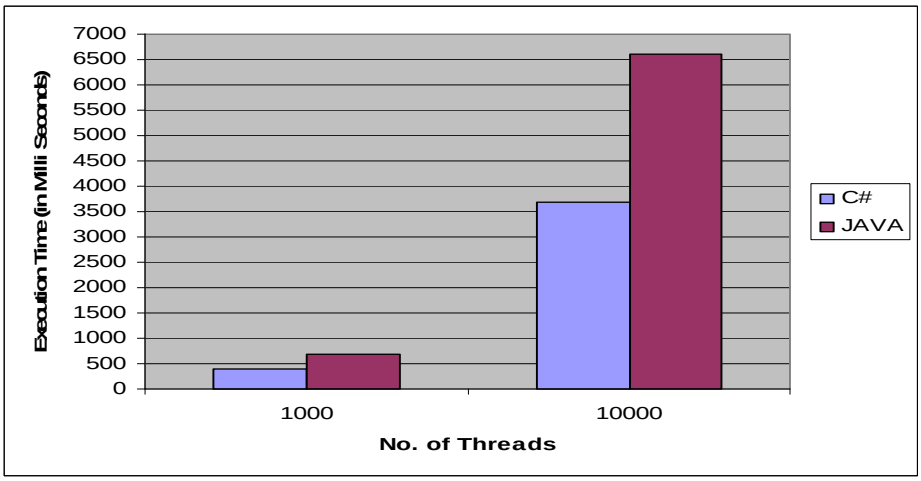

Figure 5b. Performance comparison for 1000 and 10000 Threads Execution

5. Concluding Remarks

Java and C# are two popular Object-oriented programming languages. Both provide threading facilities built into the language for thread creation and manipulation. Intel's Dual Core processors improve the performance of applications by executing multiple programs at a time. The objective of the present paper is to evaluate the performance of multithreaded programs developed in C# and Java on Intel's Dual Core processors. The performance comparison in term of execution time is reported. It is concluded that C# takes less execution time as compared to Java over similar processor architectures. It is therefore concluded that the performance of C# better than JAVA for multithreaded programs and therefore, recommended for large number of threads computations.

Acknowledgements

The authors are very thankful to Prof. B. Hanumaiah, Vice-Chancellor, Babasaheb Bhimrao Ambedkar University (A Central University), Vidya Vihar, Rae Bareilly Road, Lucknow, India, for providing excellent computation facilities in the University campus. Thanks are also due to the University Grant Commission, India, for providing financial assistance to the Central University for research work.

References

[1] Booch, G., Rumbaugh, J., Jacobson, I. (2004), The Unified Modeling Language User Guide, Twelfth Indian Reprint, Pearson Education.
[2] R.M. Ramanathan, "White Paper Intel® Multi-Core Processors: Making the Move to Quad-Core and Beyond", White paper from Intel Corporation, Retrieved from www.intel.com/technology/architecture/downloads/quad-core-06.pdf
[3] Alhir, S.S. (1998), UML in a Nutshell: A Desktop Quick Reference, O'Reilly & Associates, First Indian Reprint.
[4] Gomaa, H. (2001) "Designing Concurrent, Distributed, and Real-Time Applications with UML", Proceedings of the 23rd International Conference on Software Engineering (ICSE'01), IEEE Computer Society.
[5] Pllana, S. and Fahringer, T. (2002), "UML based modeling of Performance Oriented parallel and Distributed Applications", Winter Simulation Conference, 2002.
[6] Saxena, V., Arora D. and Ahmad S. (2007), "Object Oriented Distributed Architecture System through UML", IEEE International Conference on Advanced in Computer Vision and Information Technology, Nov. 28-30, 2007.
[7] Fateh Boutekkouk, Mohammed Benmohammed (2009), "UML for Modelling and Performance Estimation of Embedded Systems", Journal of Object Technology, vol. 8, no. 2, March-February 2009, pp. 95-118, http://www.jot.fm/issues/issue_2009_03/article1/
[8] Henderson, Robert and Zorn Benjamin (1994), "A Comparison of Object-oriented Programming in Four Modern Languages", Software—Practice and Experience, vol. 24, no. 11, pp. 1077–1095, John Wiley & Sons, Ltd.
[9] Glyph Lefkowitz (2000), "A subjective analysis of two high-level, object-oriented languages Comparing Python to Java", Retrieved from http://twistedmatrix.com/~glyph/rant/python-vs-java.html
[10] Brosgol, Benjamin M. (1998), "A Comparison of the Concurrency Features of Ada 95 and Java", Proceedings of the 1998 annual ACM SIGAda international conference on Ada**,** Washington, D.C., November 08 - 12, pp.175 – 192, Association for Computing Machinery.
[11] Bulpin, James R. and Pratt, Ian A. (2004), "Multiprogramming Performance of the Pentium 4 with Hyper-Threading", In the Third Annual Workshop on Duplicating, Deconstructing and Debunking (WDDD2004) held at ISCA '04. pp 53-62.
[12] Figueroa, María Isabel Díaz (2004), "Image Processing using Java and C#: A Comparison Approach", Retrieved from www.ece.uprm.edu/crc/crc2004/papers/Mar%92aDiaz.pdf
[13] Sestoft, Peter (2005), "Numeric performance in C, C# and Java", Retrieved from www.itu.dk/~sestoft/papers/numericperformance.pdf

[14] Saxena, Vipin and Arora, Deepak (2009), "Performance Evaluation for Object Oriented Software Systems" SIGSOFT Software Engineering Notes, March 2009, vol. 34, no. 2.
[15] Simcha Gochman, Avi Mendelson, Alon Navh and Efraim Rotem (2006), " Introducttion to Intel Core TM DUO Processor Architecture" , Intel technology Journal, vol. 10, issue 2, May, 15, 2006
[16] Ofri Wechsler (2006), "Inside Intel® Core™ Microarchitecture: Setting New Standards for Energy-Efficient Performance", Technology@Intel Magazine, March 2006

Authors

Dr. Vipin Saxena: He is a Reader, Founder and Ex-Head, Dept. of Computer Science, Babasaheb Bhimrao Ambedkar University, Lucknow, India. He got his M.Phil. Degree in Computer Application in 1992 & Ph.D. Degree work on Scientific Computing from University of Roorkee (renamed as Indian Institute of Technology, India) in 1997. He has about 14 years of teaching experience and 17 years research experience in the field of Scientific Computing & Software Engineering. He has published more than 75 International and National publications. Phone: +91-9452372550, Fax: +91-522-2440821, E-mail: vsax1@rediffmail.com

Manish Shrivastava: He is a Research Scholar, Dept. of Computer Science, Babasaheb Bhimrao Ambedkar University, Lucknow, India. He got his M.Phil. Degree in Computer Applications in 1992. He has more than 12 years of teaching experience. Currently he is actively engaged in the research work on the Unified Modeling Language. He has produced several outstanding research publications. Phone: +91-9453847114 E-mail: mshrivastava@yahoo.com