International Journal of Computational Intelligence and Informatics, Vol. 4: No. 4, March 2015

Performance Analysis of Java NativeThread and NativePthread on Win32 Platform

Bala Dhandayuthapani Veerasamy
Research Scholar
Manonmaniam Sundaranar University
Tirunelveli, Tamilnadu, India
dhanssoft@gmail.com

G M Nasira
Department of Computer Science
Chikkanna Govt Arts College
Tirupur, Tamilnadu, India
nasiragm99@ yahoo.com

Abstract- The most important motivation for using a parallel system is the reduction of the execution time of computation-thorough application programs. To facilitate the development and analysis of parallel programs the performance measures are often used. Performance measures can be based not only on theoretical cost models but also on measured execution times for a specific parallel system. In this article, we selected different computer architectures and iterations to compare the performance results of NativeThread and Hybrid NativeThread with Thread class and Concurrent API as well to compare the performance results of NativePthread and Hybrid NativePthread with Thread class and Concurrent API. The overall performance improvements are five times faster than Thread class and Concurrent API.

Keywords- JVM, NativeThread, NativePthread, Speedup, Performance.

I. INTRODUCTION

The execution time of a parallel program depends on various factors [1] together with the architecture of the execution platform, the compiler and operating system used, the parallel programming environment and the parallel programming model on which the environment is based, as well as properties of the application program such as locality of memory references or dependencies between the computations to be performed. In principle, all these factors [1] have to be concerned when developing a parallel program. However, there may be complex interactions between these factors and it is therefore difficult to consider them all. The factors [1] which decides a program performances are complex, interrelated and oftentimes, hidden from the programmer. Some of factors are listed below.

TABLE I. FACTORS RELATED TO PERFORMANCE ANALYSIS

Application related factors	Hardware related factors	Software related factors
Algorithms	Processor architecture	Operating system
Dataset sizes	Memory hierarchy	Compiler
Memory usage patterns	I/O configuration	Pre-processor
I/O communication patterns	Network	Communication protocols
Task granularity		Libraries
Load balancing		
Amdahl's law		

There are performance analysis tools available to optimizing an application's performance. They can support you in understanding what our program is really doing and suggest how program performance should be enhanced. Multicore programming is warmest part of computing world. There are so many researches going on to improve speedup and performance of the program through parallel programs. Some of our earlier researches concentrated on parallel programming; they are, Setting CPU Affinity in Windows Based SMP Systems Using Java [2], Parallel: One Time Pad using Java [3], JNT - Java Native Thread for Win32 Platform [4], Java Native Pthread for Win32 Platform [5], Java Native Intel Thread Building Blocks for Win32 Platform [6], Overall Aspects of Java Native Threads onWin32 Platform [7]. In this article, the speedup and performance improvements of JNT - Java Native Thread for Win32 Platform [4] and Java Native Pthread for Win32 Platform [5,7] with Java Thread class and Concurrent API.

The performance analysis of parallel program is already available on Linux platform. Performance evaluation and analysis of thread pinning strategies on multi-core platforms on Intel architectures investigated already in Linux operating system [8]. It investigates various cache-aware thread pinning strategies for SPEC OpenMP applications. It demonstrates in fixing an affinity provides statistically significant performance improvements compared the Linux OS strategy.

The Java Virtual Machine (JVM) can carry out many threads of execution at once. These threads independently execute code that works on values and objects exist in a shared main memory. Threads are scheduled on a particular virtual machine. There are two basic variations of thread available [4]; they are native threads and green threads.

The native threads [4] are scheduled by the operating system that is hosting the virtual machine. The operating system threads are logically divided into two pieces: user level threads and system level threads. The operating system itself that is the kernel of the operating system lies at system level threads. The kernel is accountable for managing system calls on behalf of programs that run at user level threads. Any program running under user level needs create and manage threads usually by the operating system kernel.

NativeThread [4] and NativePthread [5,7] are developed for Win32 platform threads through JNI, which enables to execute NativeThread and NativePthread on Java program, as well as Java threads with NativeThread and NativePthread to schedule and execute in hybrid mode. Java Thread class and Concurrent API are green threads. The green threads [4] are scheduled by the JVM itself. This is the original model for JVM mostly follows the idealized priority based scheduling. This kind of thread never uses operating system threads library.

II. ADVANTAGES AND DISADVANTAGES

The advantages and disadvantages of NativeThread and NativePthreads are described here under.

A. *Advantages*

- The native threads are scheduled by the operating system that is hosting the virtual machine.
- If a program performs an illegal function, it can be ended without affecting other programs or kernel.
- Migrating threads among processors: It swaps between threads preempting, switching control from a running thread to a non-running thread at any time.
- It can run on distinct CPUs.
- Hybrid Thread Model: NativeThread with Thread class and NativePthreads with Thread class
- These facilitate more than one execution methods.

B. *Disadvantage*

- It is platform dependent.
- Memory leaks: This is the classic situation where a component uses memory but fails to release it, gradually reducing the amount of memory available to the system.

III. PERFORMANCE ANALYSIS

Scalability [9] is a measure describing whether a performance improvement can be reached that is proportional to the number of processors employed. Scalability depends on several properties of an algorithm and its parallel execution. Often, for a fixed problem size n a saturation of the speedup can be observed when the number p of processors is increased. But increasing the problem size for a fixed number of processors usually leads to an increase in the attained speedup. In this sense, scalability captures the property of a parallel implementation that the efficiency can be kept constant if both the number p of processors and the problem size n are increased. Thus, scalability is an important property of parallel programs since it expresses that larger problem can be solved in the same time as smaller problems if a sufficiently large number of processors are employed. The increase in the speedup for increasing problem size n cannot be captured by Amdahl's law. Instead, a variant of Amdahl's law can be used which assumes that the sequential program part is not a constant fraction f of the total amount of computations, but that it decreases with the input size. In this case, for an arbitrary number p of processors, the intended speedup $\leq$ p can be obtained by setting the problem size to a large enough value.

The performance of a computer system is one of the most important aspects of its evaluation. Depending on the point of view, different criteria are important to evaluate performance. The user of a computer system is interested in small response times, where the response time of a program is defined as the time between the start and the termination of the program. On the other hand, a large computing center is mainly interested in high throughputs, where the throughput is the average number of work units that can be executed per time unit.

The evaluation of the parallel execution performance is measured with respect to speedup, performance improvement and efficiency with reference to the time taken for both sequential and parallel processing. Speedup measures how much a parallel algorithm is faster than a corresponding sequential algorithm. The speedup calculation [9][10] is based on the following equation.

$$\text{Speedup} = \frac{\text{Sequential execution time}}{\text{Parallel execution time}} \quad (1)$$

The performance improvement [9][10] depicts measurements relative improvement that the parallel system has over the sequential process. This performance is measured based on the following equation.

$$\text{Performace Improvements} = \frac{\text{Sequential extcution time} - \text{Parallel execution time}}{\text{Sequential execution time}} \quad (2)$$

IV. Experimental Results and Analysis

The performance of a computer system becomes larger, if the response times for a given set of application programs become smaller. The response time [9] of a program can be split into

The user CPU time of program, capturing the time that the CPU spends for executing program; the system CPU time of program, capturing the time that the CPU spends for the execution of routines of the operating system issued by program; the waiting time of program, caused by waiting for the completion of I/O operations and by the execution of other programs because of time sharing.

The time needed [9] for the parallel execution of a parallel program depends on the size of the input data n, and possibly further characteristics such as the number of iterations of an algorithm or the loop bounds; the number of processors p; and the communication parameters, which describe the specifics of the communication of a parallel system or a communication library.

We preferred different computer architectures to examine the performance results of to NativeThread [4] and Hybrid NativeThread [4] with Thread class and Concurrent API, correspondingly to compare the performance results of NativePthread[5,7] and Hybrid NativePthread [5,7]with Thread class and Concurrent API. The computer architectures used to examine the performance results are Core 2 Duo 2.10 GHz with RAM 2 GB, Core i5 3.30 GHz with RAM 4 GB and Core i7 3.40 GHz with RAM 4 GB. The operating system for Core 2 Duo 2.10 GHz with RAM 2 GB was Microsoft Windows Vista with 32 bit platform and for Core i5 3.30 GHz with RAM 4 GB and Core i7 3.40 GHz with RAM 4 GB was Microsoft Windows 7 with 32 bit platform. Also, we preferred different iterations to examine the performance results of both cases. The iterations that we preferred are multiplications of 10 such as 10, 100, 1000, 10000 and 100000. We used the following fraction of program code similarly in NativeThread and Hybrid NativeThread with Thread class and Concurrent API, also to compare the performance results of NativePthread and Hybrid NativePthread with Thread class and Concurrent API.

```
longstartTime, stopTime, elapsedTime;
startTime = System.currentTimeMillis();
for(int i=0;i<N;i++){
        stopTime = System.currentTimeMillis();
        elapsedTime = (stopTime - startTime);
        System.out.println("Java Thread\t" +t.getId()+ "\tValue " + i + " used ms " + elapsedTime);
        // getCurrentThreadId() in NativeThread and NativePthread
        //getId() in Thread class }
```

The System.currentTimeMillis() method allows to find the time during execution. This method helped us to fix the starting time of the iterations and as well as stopping time of the iterations. The elapsed time could be obtained from the subtractions of stopping time with starting time. The variable N denotes the number of iterations in the loop. The getId() method will return current thread ID in Thread class likewise the getCurrentThreadId() method will return current thread ID in NativeThread and NativePthread.

Thread affinity has quickly appeared to be one of the most important factors to accelerate program execution times on multi-core processors. Still now, it is not clear how to decide for the best thread placement that considers all the possible performance factors (data locality, memory bus bandwidth, OS synchronization overhead, NUMA effects, etc.). This article makes an exhaustive empirical study of four and eight thread placement strategies on three distinct machines. Thread class and Concurrent API have no processors core affinities; this can be executed independently on core processors. The JVM will manage thread placement on core processors. On the other side, NativeThread, Hybrid NativeThread, NativePthread and HybirdNativePthread have processor core affinities.

TABLE II. PERFORMANCE ANALYSIS ON NATIVETHREAD AND HYBRID NATIVETHREAD AMONG FOUR THREADS EXECUTIONS IN MS

Processors	Iterations	Thread class	Concurrent API	NativeThread (NT)	Hybrid NativeThread (HNT)	Speedup				Performance Improvements			
						Thread class		*Con. API*		*Thread class*		*Con. API*	
						NT	*HNT*	*NT*	*HNT*	*NT*	*HNT*	*NT*	*HNT*
Core 2 Duo 2.10 GHz With RAM 2 GB	10	2	2	4	4	0.5	0.5	0.5	0.5	-1	-1	-1	-1
	100	125	226	165	65	0.76	1.92	1.37	3.48	-0.32	0.48	0.27	0.71
	1000	1520	4809	1466	1328	1.04	1.14	3.28	3.62	0.04	0.13	0.7	0.72
	10000	21098	47608	14325	13270	1.47	1.59	3.32	3.59	0.32	0.37	0.7	0.72
	100000	192980	431716	263052	299301	0.73	0.64	1.64	1.44	-0.36	-0.55	0.39	0.31
Core i5 3.30 GHz With RAM 4 GB	10	1	5	1	1	1	1	5	5	0	0	0.8	0.8
	100	274	203	43	100	6.37	2.74	4.72	2.03	0.84	0.64	0.79	0.51
	1000	2802	1051	673	678	4.16	4.13	1.56	1.55	0.76	0.76	0.36	0.35
	10000	5842	10719	7097	6620	0.82	0.88	1.51	1.62	-0.21	-0.13	0.34	0.38
	100000	52585	104524	63667	62853	0.83	0.84	1.64	1.66	-0.21	-0.2	0.39	0.4
Core i7 3.40 GHz With RAM 4 GB	10	1	2	1	16	1	0.06	2	0.13	0	-15	0.5	-7
	100	144	79	76	62	1.89	2.32	1.04	1.27	0.47	0.57	0.04	0.22
	1000	456	936	824	717	0.55	0.64	1.14	1.31	-0.81	-0.57	0.12	0.23
	10000	4717	9297	7472	8159	0.63	0.58	1.24	1.14	-0.58	-0.73	0.2	0.12
	100000	47197	92266	51058	47438	0.92	0.99	1.81	1.94	-0.08	-0.01	0.45	0.49
Maximum Values on Speedup and Performance Improvements						6.37	4.13	5	5	0.84	0.76	0.8	0.8
Minimum Values on Speedup and Performance Improvements						0.5	0.06	0.5	0.13	-1	-15	-1	-7

TABLE III. PERFORMANCE ANALYSIS ON NATIVETHREAD AND HYBRID NATIVETHREAD AMONG EIGHT THREADS EXECUTIONS IN MS

Processors	Iterations	Thread class	Concurrent API	NativeThread (NT)	Hybrid NativeThread (HNT)	Speedup				Performance Improvements			
						Thread class		Con. API		Thread class		Con. API	
						NT	HNT	NT	HNT	NT	HNT	NT	HNT
Core 2 Duo 2.10 GHz With RAM 2 GB	10	2	8	6	7	0.33	0.29	1.33	1.14	-2	-2.5	0.25	0.13
	100	222	826	345	167	0.64	1.33	2.39	4.95	-0.55	0.25	0.58	0.8
	1000	3554	9584	1235	3072	2.88	1.16	7.76	3.12	0.65	0.14	0.87	0.68
	10000	43642	98899	25434	29663	1.72	1.47	3.89	3.33	0.42	0.32	0.74	0.7
	100000	612702	1151704	650667	579780	0.94	1.06	1.77	1.99	-0.06	0.05	0.44	0.5
Core i5 3.30 GHz With RAM 4 GB	10	2	3	2	2	1	1	1.5	1.5	0	0	0.33	0.33
	100	597	183	204	100	2.93	5.97	0.9	1.83	0.66	0.83	-0.11	0.45
	1000	5413	2108	1327	1334	4.08	4.06	1.59	1.58	0.75	0.75	0.37	0.37
	10000	11865	22989	13283	13395	0.89	0.89	1.73	1.72	-0.12	-0.13	0.42	0.42
	100000	103403	216567	124898	121406	0.83	0.85	1.73	1.78	-0.21	-0.17	0.42	0.44
Core i7 3.40 GHz With RAM 4 GB	10	3	5	16	16	0.19	0.19	0.31	0.31	-4.33	-4.33	-2.2	-2.2
	100	189	207	141	141	1.34	1.34	1.47	1.47	0.25	0.25	0.32	0.32
	1000	1155	1730	1544	1326	0.75	0.87	1.12	1.3	-0.34	-0.15	0.11	0.23
	10000	9987	17657	16504	15928	0.61	0.63	1.07	1.11	-0.65	-0.59	0.07	0.1
	100000	125980	171886	97410	95782	1.29	1.32	1.76	1.79	0.23	0.24	0.43	0.44
Maximum Values on Speedup and Performance Improvements						4.08	5.97	7.76	4.95	0.75	0.83	0.87	0.8
Minimum Values on Speedup and Performance Improvements						0.19	0.19	0.31	0.31	-4.33	-4.33	-2.2	-2.2

When faced to variations of observed execution times, we must use rigorous statistics to study the validity of our empirical conclusions. Empirical conclusions must not rely on sample metrics such as sample means or averages; we must rely on statistical tests. We calculated the actual speedup of the program base on the formula (1) and we calculated the performance improvements base on the formula (2). We consider Thread class program and Concurrent API program according to the need for formula (1) (2) sequential execution time. The speedup and performances examined and arranged in the following tables.

TABLE IV. PERFORMANCE ANALYSIS ON NATIVEPTHREAD AND HYBRID NATIVEPTHREAD AMONG FOUR THREADS EXECUTIONS IN MS

Processors	Iterations	Thread class	Concurrent API	NativeThread (NT)	Hybrid NativeThread(HNT)	Speedup				Performance Improvements			
						Thread class		*Con. API*		*Thread class*		*Con. API*	
						NT	*HNT*	*NT*	*HNT*	*NT*	*HNT*	*NT*	*HNT*
Core 2 Duo 2.10 GHz With RAM 2 GB	10	2	2	2	2	1	1	1	1	0	0	0	0
	100	125	226	27	38	4.63	3.29	8.37	5.95	0.78	0.7	0.88	0.83
	1000	1520	4809	1005	829	1.51	1.83	4.79	5.8	0.34	0.45	0.79	0.83
	10000	21098	47608	8595	10325	2.45	2.04	5.54	4.61	0.59	0.51	0.82	0.78
	100000	192980	431716	263052	299301	0.73	0.64	1.64	1.44	-0.36	-0.55	0.39	0.31
Core i5 3.30 GHz With RAM 4 GB	10	1	5	1	1	1	1	5	5	0	0	0.8	0.8
	100	274	203	127	35	2.16	7.83	1.6	5.8	0.54	0.87	0.37	0.83
	1000	2802	1051	519	563	5.4	4.98	2.03	1.87	0.81	0.8	0.51	0.46
	10000	5842	10719	5760	5773	1.01	1.01	1.86	1.86	0.01	0.01	0.46	0.46
	100000	52585	104524	61949	68293	0.85	0.77	1.69	1.53	-0.18	-0.3	0.41	0.35
Core i7 3.40 GHz With RAM 4 GB	10	1	2	1	1	1	1	2	2	0	0	0.5	0.5
	100	144	79	38	19	3.79	7.58	2.08	4.16	0.74	0.87	0.52	0.76
	1000	456	936	392	506	1.16	0.9	2.39	1.85	0.14	-0.11	0.58	0.46
	10000	4717	9297	4989	5087	0.95	0.93	1.86	1.83	-0.06	-0.08	0.46	0.45
	100000	47197	92266	49862	49641	0.95	0.95	1.85	1.86	-0.06	-0.05	0.46	0.46
Maximum Values on Speedup and Performance Improvements						5.4	7.83	8.37	5.95	0.81	0.87	0.88	0.83
Minimum Values on Speedup and Performance Improvements						0.73	0.64	1	1	-0.36	-0.55	0	0

TABLE V. PERFORMANCE ANALYSIS ON NATIVEPTHREAD AND HYBRID NATIVEPTHREAD AMONG EIGHT THREADS EXECUTIONS IN MS

Processors	Iterations	Thread class	Concurrent API	NativePthread (NT)	Hybrid NativePthread (HNT)	Speedup				Performance Improvements			
						Thread class		*Con. API*		*Thread class*		*Con. API*	
						NT	*HNT*	*NT*	*HNT*	*NT*	*HNT*	*NT*	*HNT*
Core 2 Duo 2.10 GHz With RAM 2 GB	10	2	8	2	2	1	1	4	4	0	0	0.75	0.75
	100	222	826	27	35	8.22	6.34	30.59	23.6	0.88	0.84	0.97	0.96
	1000	3554	9584	953	2335	3.73	1.52	10.06	4.1	0.73	0.34	0.9	0.76
	10000	43642	98899	16009	22336	2.73	1.95	6.18	4.43	0.63	0.49	0.84	0.77
	100000	612702	1151704	650667	579780	0.94	1.06	1.77	1.99	-0.06	0.05	0.44	0.5
Core i5 3.30 GHz With RAM 4 GB	10	2	3	1	1	2	2	3	3	0.5	0.5	0.67	0.67
	100	597	183	39	42	15.31	14.21	4.69	4.36	0.93	0.93	0.79	0.77
	1000	5413	2108	999	1021	5.42	5.3	2.11	2.06	0.82	0.81	0.53	0.52
	10000	11865	22989	10227	11525	1.16	1.03	2.25	1.99	0.14	0.03	0.56	0.5
	100000	103403	216567	130318	125050	0.79	0.83	1.66	1.73	-0.26	-0.21	0.4	0.42

Core i7 3.40 GHz With RAM 4 GB	10	3	5	1	1	3	3	5	5	0.67	0.67	0.8	0.8
	100	189	207	96	102	1.97	1.85	2.16	2.03	0.49	0.46	0.54	0.51
	1000	1155	1730	969	883	1.19	1.31	1.79	1.96	0.16	0.24	0.44	0.49
	10000	9987	17657	9000	10000	1.11	1	1.96	1.77	0.1	0	0.49	0.43
	100000	125980	171886	98852	98751	1.27	1.28	1.74	1.74	0.22	0.22	0.42	0.43
Maximum Values on Speedup and Performance Improvements						15.31	14.21	30.59	23.6	0.93	0.93	0.97	0.96
Minimum Values on Speedup and Performance Improvements						0.79	0.83	1.66	1.73	-0.26	-0.21	0.4	0.42

V. RESULTS AND DISCUSSIONS

We point out the minimum and maximum execution speedup and improvements in the above tables to recognize where the overall speedup and improvements are occurred.

The table 1 with NativeThread and Hybrid NativeThread among four threads, the maximum speedup 6.37 shows at the Core i5 3.30 GHz processor with 100 iteration execution on Thread class comparison with NativeThread and the minimum speedup 0.06 shows at the Core i7 3.40 GHz with 10 iteration execution on Thread class comparison with NativeThread. Likewise, the maximum performance improvements 0.84 shows at the Core i5 3.30 GHz with 100 iteration execution on Thread class comparison with NativeThread and the minimum performance improvements -15 shows at the Core i7 3.40 GHz with 10 iteration execution on Thread class comparison with NativeThread.

The table 2 with NativeThread and Hybrid NativeThread among eight threads, the maximum speedup 7.76 shows at the Core 2 Duo 2.10 GHz processor with 1000 iteration execution on Concurrent API comparison with NativeThread and the minimum speedup 0.19 shows at the Core i7 3.40 GHz with 10 iteration execution on Thread class comparison with NativeThread. Likewise, the maximum performance improvements 0.87 shows at the Core 2 Duo 2.10 GHz with 1000 iteration execution on Concurrent API comparison with NativeThread and the minimum performance improvements -4.33 shows at the Core i7 3.40 GHz with 10 iteration execution on Thread class comparison with NativeThread.

The table 3 with NativePthread and Hybrid NativePthread among four threads, the maximum speedup 8.37 shows at the Core 2 Duo 2.10 GHz processor with 100 iteration execution on Concurrent API comparison with NativePthread and the minimum speedup 0.64 shows at the core 2 Duo with 100000 iteration execution on Thread class comparison with NativePthread. Likewise, the maximum performance improvements 0.88 shows at the Core 2 Duo 2.10 GHz with 100 iteration execution on Concurrent API comparison with NativePthread and the minimum performance improvements -0.55 shows at the Core 2 Duo 2.10 GHz with 100000 iteration execution on Thread class comparison with NativePthread.

The table 4 with NativePthread and Hybrid NativePthread among eight threads, the maximum speedup 30.59 shows at the Core 2 Duo 2.10 GHz processor with 100 iteration execution on Concurrent API comparison with NativePthread and the minimum speedup 0.79 shows at the core i5 with 100000 iteration execution on Thread class comparison with NativePthread. Likewise, the maximum performance improvements 0.97 shows at the Core 2 Duo 2.10 GHz with 100 iteration execution on Concurrent API comparison with NativePthread and the minimum performance improvements -0.26 shows at the Core i5 3.30 GHz with 100000 iteration execution on Thread class comparison with NativePthread.

According to all four tables the maximum and minimum speedup and performance improvement values are recognized with the overall maximum and minimum speedup and performance improvements. The overall maximum speedup 30.59 obtained at the Core 2 Duo 2.10 GHz processor with 100 iteration and eight thread executions on Concurrent API comparison with NativePthread and the maximum performance improvements 0.97 obtained at the Core 2 Duo 2.10 GHz with 100 iteration and eight thread executions on Concurrent API comparison with NativePthread. Likewise the minimum speedup 0.06 obtained at the Core i7 3.40 GHz with 10 iteration and four thread executions on Thread class comparison with NativeThread and the minimum performance improvements -15 obtained at the Core i7 3.40 GHz with 10 iteration and four executions on Thread class comparison with NativeThread.

In the above table 1-4 all positive values are improved performances and negative values are degraded performances. We have 2 columns of performance improvements with each processor cores have 5 rows. Therefore each table 1-4 of comparisons has 60 comparisons and totally on table 1-4 have 240 comparisons. Among these 240 comparisons, only 48 comparisons proceeded negative points and 192 comparisons positive. Along with negative 48 points of comparisons, 36 comparisons are Thread class based and remaining 12 comparisons belongs to Concurrent API based. As on average performance improvements obtained one times faster than Concurrent API and four times faster than Thread class and overall five times faster than Thread class and Concurrent API.

We demonstrate that fixing an affinity provides statistically significant performance improvements compared. The following figure 1-4 show the speedup and performance improvements based on the above tables. In the figures X axis shows statistics values for different core processors, the X axis values from 1 to 5 values Core 2 Duo 2.10 GHz, the X axis values from 6 to 10 shows Core i5 3.30 GHz and the X axis values from 11 to 15 shows Core i7 3.40 GHz. In the figures Y axis values shows the actual speedup and the performance improvements on different core processor with different iterations of executions.

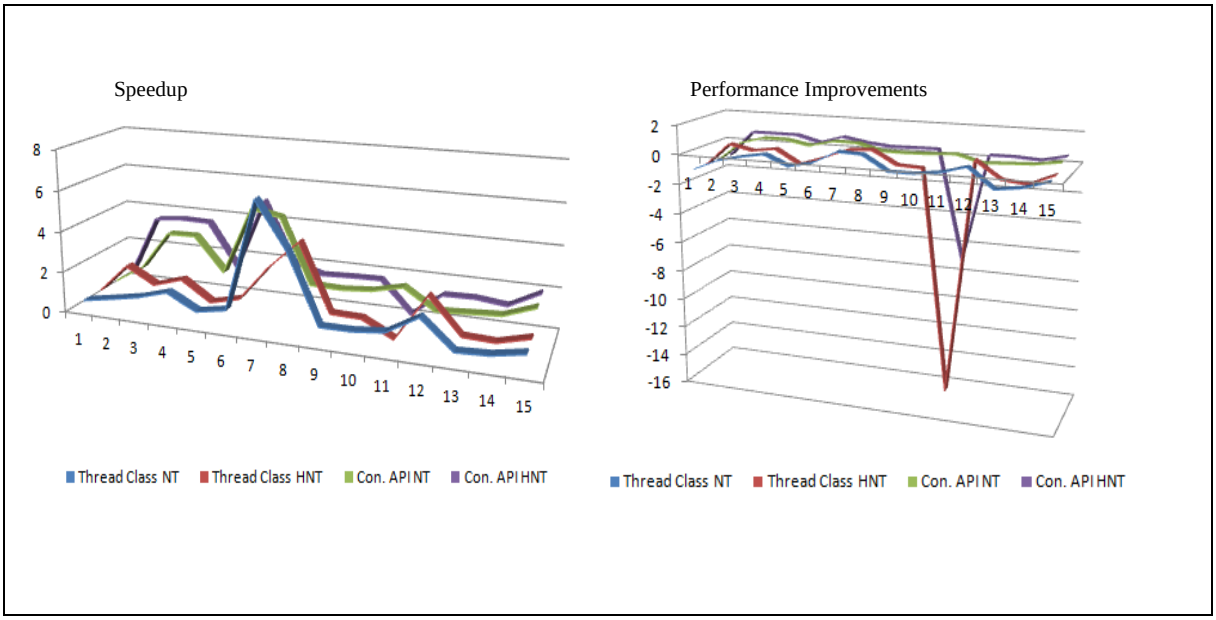

Figure 1. Graph on NativeThread and Hybrid NativeThread among four threads

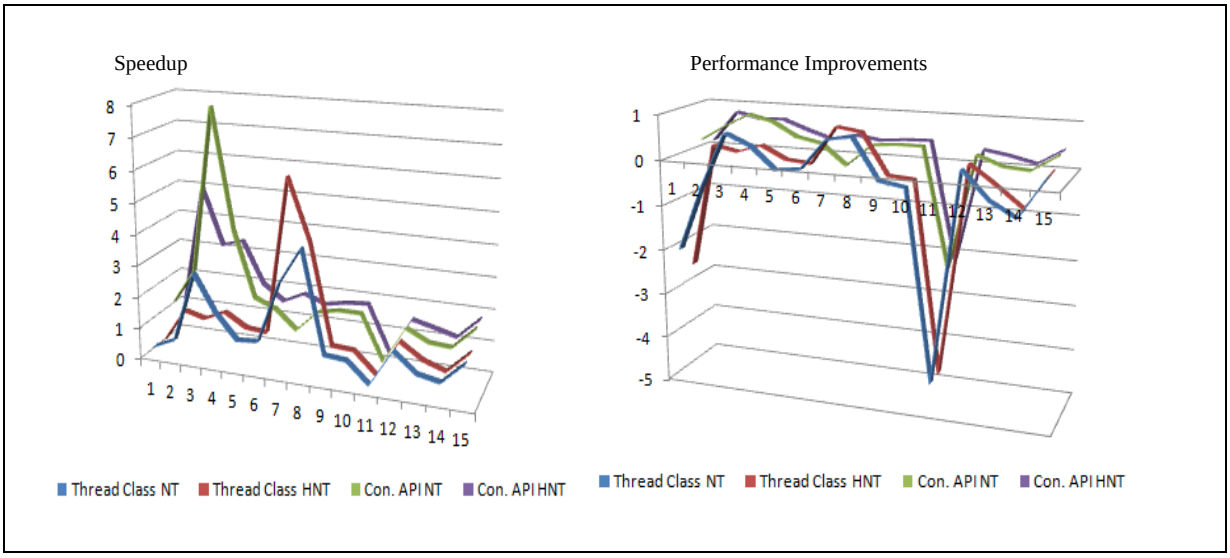

Figure 2. Graph on NativeThread and Hybrid NativeThread among eight threads

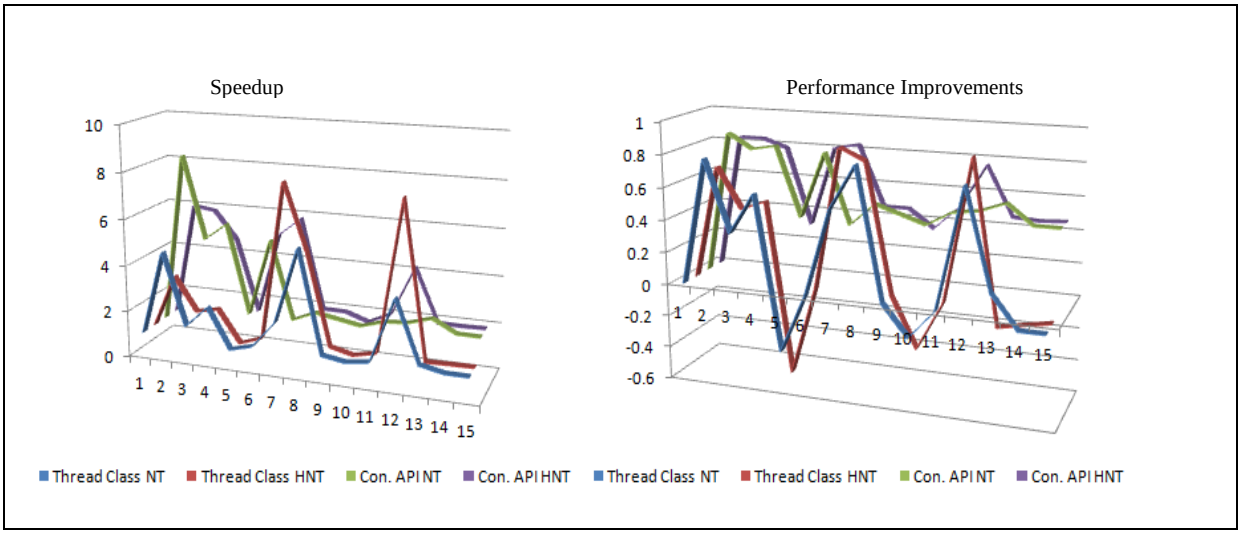

Figure 3. Graph on NativePthread and Hybrid NativePthread among four threads

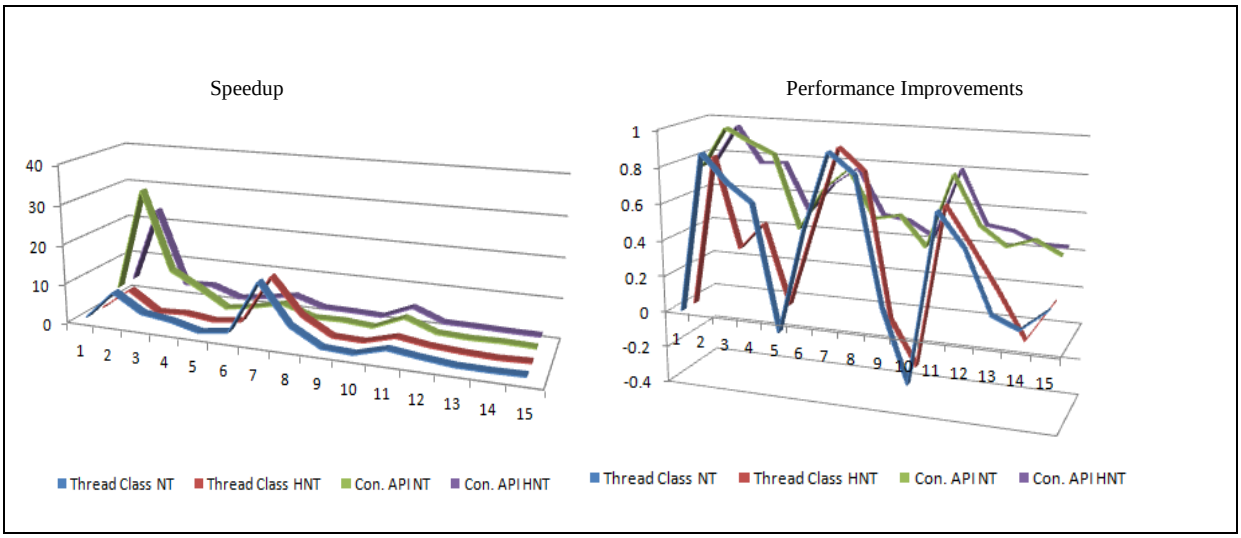

Figure 4. Graph on NativePthread and Hybrid NativePthread among eight threads

VI. CONCLUSION

The performance of a computer system is one of the most important aspects of its evaluation. Always, computer users interested in small computer response times or execution time. On the other hand, a large computing center is mainly interested in high throughputs and scalability. The execution time of a parallel program depends on many factors including the architecture, compiler, operating system, parallel programming model as well as properties of the application program such as locality of memory references or dependencies between the computations to be performed. The evaluation of the parallel execution performance is measured with respect to speedup, performance improvement and efficiency with reference to the time taken for both sequential and parallel processing. Speedup measures how much a parallel algorithm is faster than a corresponding sequential algorithm. In this article, we preferred different computer architectures, program iterations and different numbers of threads compared the speedup and performance improvements results of NativeThread and Hybrid NativeThread with Thread class and Concurrent API as well as compared the speedup performance improvements results of NativePthread and Hybrid NativePthread with Thread class and Concurrent API. As an average performance improvement, we obtained one times faster than Concurrent API and four times faster than Thread class and the overall performance improvements are five times faster than Thread class and Concurrent API.

REFERENCES

[1] Bala Dhandayuthapani Veerasamy, An Introduction to Parallel and Distributed Computing through java, Number of Pages 824, First Edition, Penram International Publishing (India) Pvt, Mumbai, India, ISBN-10: 81-87972-84-X, ISBN-13: 978-81-87972-84-6.

[2] Bala Dhandayuthapani Veerasamy and Dr. G.M. Nasira,Setting CPU Affinity in Windows Based SMP Systems Using Java, International Journal of Scientific & Engineering Research, USA, vol-3, no-4, pp-893-900, 2012, ISSN 2229-5518 11

[3] Bala Dhandayuthapani Veerasamy and Dr. G.M. Nasira,Parallel: One Time Pad using Java, International Journal of Scientific & Engineering Research, USA, vol-3, no-11, pp.1109-1117, 2012, ISSN 2229-5518 11

[4] Bala Dhandayuthapani Veerasamy and Dr. G.M. Nasira,JNT-Java Native Thread for Win32 Platform, International Journal of Computer Applications, USA, vol-71, no- 1, 2013, ISSN 0975-8887. DOI: 10.5120/12212-8249

[5] Bala Dhandayuthapani Veerasamy and Dr. G.M. Nasira, Java Native Pthreads for Win32 Platform, "World Congress on Computing and Communication Technologies (WCCCT'14), vol., no., pp.195-199, 2014, IEEE Xplore, ISBN: 978-1-4799-2876-7, DOI: 10.1109/WCCCT.2014.13

[6] Bala Dhandayuthapani Veerasamy and Dr. G.M. Nasira, Java Native Intel Thread Building Blocks for Win32 Platform, Asian Journal of Information Technology, Accepted on for publication on March 03, 2014. Medwell Publishing, ISSN: 1682-3915.

[7] Bala Dhandayuthapani Veerasamy and Dr. G.M. Nasira, Overall Aspects of Java Native Threads on Win32 Platform, Second International Conference on Emerging Research in Computing, Information, Communication and Applications (ERCICA-2014), vol. II, pp.667-675, 2014, Bangalore, published in ELSEVIER in India. ISBN: 9789351072621.

[8] Abdelhafid Mazouz, Sid-Ahmed-Ali Touati, Denis Barthou, Performance evaluation and analysis of thread pinning strategies on multi-core platforms: Case study of SPEC OMP appli cations on intel architectures, High Performance Computing and Simulation (HPCS), pp.273 -279, 2011.

[9] Thomas Rauber, Gudula Runger, Parallel Programming For Multicore and Cluster Systems,Springer-Verlag Berlin Heidelberg, ISBN 978-3-642-04817-3, 2010, DOI 10.1007/978-3-642-04818-0

[10] Shameem Akhter,Jason Roberts,Multi-Core Programming Increasing Performance through Software Multi-threading,Intel Corporation, 2006