International Journal of Innovative Research in Engineering and Management (IJIREM)
ISSN (Online): 2350-0557, Volume-11, Issue-5, October 2024
https://doi.org/10.55524/ijirem.2024.11.5.9
Article ID IJIR3023, Pages 66-72
www.ijirem.org

# Exploring the Impact of Multithreading on System Resource Utilization and Efficiency

**Preet Bhutani[1], and Amol Ashokrao Shinde[2]**

[1] School of Engineering & Technology, MVN University, Palwal, India
[2] Lead Software Engineer, Mastech Digital Technologies Inc, Pittsburgh PA,United States

Correspondence should be addressed to Preet Bhutani; preetbhutani7@gmail.com

Received: 5 September 2024 Revised: 18 September 2024 Accepted: 3 October 2024



**ABSTRACT-** The goal of this research is to find out the effects of multithreading on the consumption of system resources and efficiency by examining CPU utilization, memory use, Input/Output operation, and power consumption in the multithreaded systems. In order to measure static, adaptive, and dynamic multithreading performance under different workloads, the study compares the three models both through theory and by applying it to various experiments. That is why the results show that, when multithreading is implemented, the general CPU load and I/O performance improve, especially for computational and data-consuming operations. However, some problems include memory contention, context-switching overhead and, higher energy consumption are noted especially when threading is over-provisioned for. Real-time threading control strategies were the most effective as they periodically reconfigured the number of threads in a way that optimizes performance while optimizing resources. In addition, while the asynchronous I/O models provided the best performance improvements, energy use went up when multithreading was incorporated, thus the need for implementing trade-offs between performance and power usage. It offers significant findings related to the tuning of multithreading strategies so as to maximize system performance but with an acceptable level of resource utilization, should be of significance for practitioners, who are working with multicore processors and dealing with high-performing systems.

**KEYWORDS-** Multithreading, system resource utilization, efficiency, CPU utilization, memory management

## I. INTRODUCTION

Multithreading has turned out to be one of the most important and indispensable paradigms of today's computing environments that greatly strive for optimizing the execution of multiple threads. In an environment where every single number crunch, every ounce of data processed matters, multithreading takes on a whole new dimension. As systems become larger and larger and are expected more and more to deliver large solutions and answers, inefficiency in allocations creates problems. Fundamentally, multithreading's primary goals revolve around getting the best of the CPU, reducing usage idle time and making process and applications the most efficient they can be without hitches. Nonetheless, despite bringing a promise of boosted efficiency, multithreading's effect on resource utilization in the system is complex and strongly depends on various parameters related to the system architecture and threading and scheduling policies of OS used [1].

Multithreading is a way of carrying out multiple threads from a single program process concept where the same process is split into several parts. Thus, thanks to the parallelism at the thread level, multithreading enables solution of he same activities in parallel, and therefore in less time. However, such concurrent execution has to be managed carried out in a way to avoid conflicts for resources like memory and I/O. Failure to do so shall result in competition in these resources leading to reduced performance that would negate the utility of multithreading. This tension between using concurrency and coordinating access to shared resources is one of the primary complexities in designing and building multithreaded processes [2].

Scalability is also one of the primary reasons to use multithreaded applications; the claim that such applications yield better CPU usage is often given. The case is that in single-threaded systems, an expensive CPU may be idle, waiting for I/O operations, whereas other resources remain unused, and vice versa. This is done through multi-threading where other threads are also capable to be run genuine the individual thread waits making the CPU never idle. These lead to increased total system throughput of a system that can easily be appreciated especially when the tasks are I/O bounded or there are lots of context switches [3].

But, the advantages of multithreaded applications are not confined to 'concurrent use of herein described CPU resources'. The approach also potentially provides benefits in the way of optimizing the amount of memory required, especially in between-thread shared resource systems. In many-threaded models within a process, threads often work in a shared address space, this can make it difficult to manage processes which requires different address spaces for each process which can increase overhead costs. It is better because memory resources of a multithreaded system can be shared, for instance in cases of processing large amounts of data or when several processes require that data.

All the same, multithreading is not without some troubles as it will be shown in the following sections. One of the

most important one is synchronization problem. Where there are common variables that can be used by many threads at a time then there likely to be inconsistency of data or even the data may get corrupted where issues like lock or semaphore are not put in place. These synchronization methods though useful add overhead that has the effect of reducing the benefits of multithreading. Moreover, weak coupling of the interdependent part can bring negative consequences including deadlock where several threads are waiting for resources to be freed by other threads [4].

A related issue connected with multithreading is the context switching time which is also rather high. The multithreading is useful for parallel execution, but it is necessary to keep per-thread state, including stack and register. And when the operating system switches between the threads, it has to store the information of the current thread and reload the information of the next thread. While this procedure of switching through the various threads is not very slow, there is always some overhead that can be demanding if the switch through the threads occurs most often. This overhead can offset performance advantage that parallel execution provides in multithreaded systems especially where threads are often preempted or where the work load among threads is not uniformly distributed [5].

Hence, according to the facts stated earlier, it is basically possible to realise the necessity to use the multithreaded control not only for the CPU and memory but other components as well. I/O operations – or input/output operations – also affect the efficiency cumulatively as a result of the chosen multithreaded system. I/O operations are often one of the biggest issues with the given system, particularly in applications that require substantial file input, like database or large-scale web serving. By another sequence of execution of a compute-bound code while a thread is waiting for an I/O operation, multi-threaded environment can overcome this bottleneck, thus, increase the overall performance of the system. Though this introduces some issues of thread synchronization especially when the system has several I/O bound threads to insulate these threads from becoming stranded.

As seen previously, the effectiveness of multithreading in enhancing system performance relies greatly with the existing hardware platform. Modern day CPUs are designed for multithreaded where several cores of the same CPU are designed to handle several threads at an instance. These multicore architectures are further suitable for multithreaded application for true parallelism of threads in opposition to the pseudo parallelism for threads offered in single-core systems through time slicing. Nevertheless, multiple-threading on a multicore system is not always advantageous in all the same ways. Sometimes even application developed by using multiple threads may not have a linear relation between the number of available cores and its performance due to some reasons such as memory bandwidth constraints or cache conflicts or thread scheduling.

The third fundamental involves how the operating system handles threads and when to execute them is also a great determinant of the efficiency of multithreaded systems. Several scheduling algorithms are used by operating systems to decided which threads are to be run at a particular time to fully utilize the CPU without frequently switching between threads and to limit competition for shareable resources. Of the operating systems some are more capable of handling multithreaded workloads especially those operating systems that are capable of exploiting multi core processors. On the other hand, the style of operation also has an influence on the performance of multithreading by relation to the exact load that is being run. CPU-intensive tasks, where the computation spends much time, may receive more benefit from multithreading than from I/O intensive tasks, where the tasks spend more time waiting for reading or writing data.

## II. LITERATURE REVIEW

The study of multi-threading and its potential effects on resources consumed by the system as well as the performance has attracted much interest in recent years, especially consequent to the continuing fast development of computers and the complication of software. Last three years, from 2022 till 2024, also show strong interest of authors in enhancing the multithreading strategies for matching the demands of modern applications that need to be computed fast but do not want to pay for the overhead of the parallel computation. Guaranteeing OS functionality and resource management has become important as processors have developed including multicore processors and the use of multithreading [6].

Research conducted in this year 2022 is confirming that simple relationship between theories of multithreading and CPU usage is complicated. Several authors have pointed out the correlation of multicore systems with better thread control and enhanced performance throughput. A Multithreaded Systems study done by Li et al. (2022) sought to explain how these systems can benefit from multicore processors, by ensuring load balancing so as to efficiently utilize the processor. The study showed that with right thread scheduling algorithms in multicore systems that they could greatly minimize idle time especially with computations. This research also noted that care has to be taken in how many threads are used in relation to the number of cores to avoid being beaten by high thread contention and context switching overheads that offset the gains that have been realized [7].

Also in 2023, Kumar and his colleagues studied on ways of using adaptive threading techniques in resource management in ever changing environments. The analysis made in this work allowed to understand how in case of different workloads the systems relying on the static multithreading can have various inefficiencies. There is a model of Kumar et al. suggesting a mechanism to regulate an amount of active threads in accordance with current resources and load. In this way, the system avoided conflicts between resources and optimized intensive use of CPU resources. On intensive data applications their experiments showed substantial performance increase over the classical model of static multithreading approach. This work showed that several of the historical problems with multithreading could be mitigated with adaptive work-load distributive schemes, eliminating complaints like over-provisioning and underutilization of resources [8].

Another extension in 2023 was devoted to the synchronization techniques in the context of the multithreaded environment. A key idea was found by Zhang and Wong, in their work on the impact of fine-grained versus more coarse-grained locking in multithreaded

programs. Their work said that although fine-grained locks enhance concurrency and system throughput, the situation comes with high overhead because of so many lock operations. Fine-grained locks provide less blocking overhead but cause serious contention problems especially when the number of active threads is high while on the other end we have course-grained locks that provide less block overhead but may result in a lot of contention for resources. Cochin and Min submitted that their study found that a hybrid of the application of fine-grained and coarse-grained locks was the most optimal for the concurrent as well as efficient use of the applications. This paper aimed at exploring and emphasizing on how synchronizations strategy it is possible to hold levels of performance of multithreaded systems of high level without compromising the reliability of data as well as resulting in resource contention [9].

The subject of thread scheduling algorithms as related to multithreaded scheduling environments was remained in focus for the researchers in the year 2024. Patel and Kaur discussed the effects of improvement scheduling algorithms for priority-based and timers and real-time schedulers on system performance. Their work discovered that though the round-robin and FCFS schedulers might offer fair schedules, they may not be the best for complex runtime environments. Priority-based scheduling was found to be most effective in enhancing the efficiency of multithreaded systems by dedicating more CPU time to high priority based threads to avoid wait times. But the researchers also observed that such algorithms have to be fine-tuned to minimize the priority inversion, the phenomenon where lower priority tasks remove or obscure higher priority ones due to control over resources [10].

Another rather explored topic in the literature is the marker between multithreading and energy. But steady power consumption is a growing problem for future generations of computers and other multifunctional platforms, particularly mobile and embedded systems, therefore interest in investigating the effects of multithreading on power consumption has emerged among researchers. Chen et al. (2023) investigated in his study how thread management strategies influence energy consumption in multicore processors. From their results, they deduced that while multithreading can improve through put, it may degrade efficiency because there are more ‘live cores’ in circulation and the cost of context switching. Instead, they collectively presented an energy-aware multithreading model with intelligent thread scheduling to control and optimize the flow of threads in line with the available power budget of the interconnect. This research stresses the need for architects to consider energy factors in the design and Use of multithreaded systems especially in energy-sensitive environments such as mobile devices [11].

In 2023, computer cloud environments where multithreading is extensively used to serve multiple users and concurrently procession multiple requests have equally experienced significant improvement. In a recent published research endeavour, Gupta et al., the authors investigated the effects of multithreading on resources in the cloud system. Their work was targeted at investigation of interactions of multithreaded applications that interact with operates in virtualized environments in an effort to enhance their throughput and lessen latency. Based on the interaction matrix of hypervisors and multithreaded workloads, it was determined that the incorrect manner of thread scheduling at the hypervisor could cause resource concurrency and, in consequence, system performance decrease. To this, the authors developed a new scheduling algorithm that could enhance the multithreaded applications and resource allocation of virtual CPUs in cloud environment. This research is particularly significant in the present world where most of the data processing and Web-based applications are implemented using cloud infrastructures [12].

Other works of testing and debugging have also recently focused on being@Module and testMultithreaded systems. As we progress through 2022, Smith and Johnson are among the researchers who focused on novel approaches to detecting and preventing bugs in multithreaded software.” They also stressed that the matters like race conditions, deadlocks, and thread starvation are even harder to notice when in highly parallel systems. The work of the authors brought new idea of automated testing that involves the usage of both static and dynamic analysis in order to detect threads-related defects at early stages of the development phase. This approach cut the time taken to diagnose and resolve concurrency anomalies considerably ensuring more dependable multithreaded applications. This work is important because multithreaded systems become only more complex, and simpler methods of debugging are no longer as effective.

## III. RESEARCH METHODOLOGY

Based on the research objective set for this paper the research methodology is aimed at evaluating the effect of multithreading on the resources utilized by the system, and efficiency. The idea is to use theoretical analysis of known algorithms and their modifications, including the experimental implementation in special conditions and performance evaluations. To achieve the research objective, a mixed-methods approach will cover all the system objects: CPU, memory, I/O, and other performance metrics as part of the system. This approach gives a possibility to analyze how multithreading correlates with the usage of the resources while including the quantitative results along with qualitative observations.

The first process engages literature review to establish the factors affecting multithreaded system performance with regards to the recent literature (2022–2024). This review supports the situating of the research in the existing multithreading literature and gives an understanding of the trends and difficulties in the management of system resources. Through the synthesis of literature regarding synchronization mechanisms, scheduling algorithms and memory management strategies, the literature review lays out a theoretical framework for experimental design of the research. The review also reveals the knowledge gaps to be filled in the subsequent study, particularly with reference to dynamic thread management and energy consumption.

The realisation of the research work thus includes creation of a multithreaded system prototype for the evaluation of the performance impact of the various multithreading approaches and strategies under different workload conditions. The design and implementation of the prototype are performed in a programming environment, which supports multithreaded programming languages and

tools: C++ and Java include powerful libraries and frameworks that provide control over thread creation and synchronization. The system was built to perform a set of computational and I/O-intensive operations and to create a proper environment to test various approaches to multithreading. The tasks are chosen to embody typical problems of contemporary computation, for example, massive computation, many requests from the web server at the same time, or real-time functioning of a system to make the results as realistic as possible.

Next, in order to prove that CPU usage is not increased by multithreading, the prototype system is run under different threading models: no threading, static and adaptive. In each model, all the particular tasks can be run in parallel, and the corresponding CPU load, idle time, and context-switching overhead may be compared. Instead, the CPU load is calculated with the help of system instrumentation tools including top, htop and perf that routinely present data on CPU activity level. Furthermore, the total number of tasks processed by the system is measured to compare throughputs with and without multithreading and enhanced application parallelism. Through comparing outcomes across the threading different models, the experiment aims at establishing the maximum number of threads that can be processed to consume the processor power ought to be utilized to the maximum with little or no effect of context switch overhead or concourse on shared resources.

The other common area of concentration in the experimental phase is management of memory. It also contains provisions to develop both shared and independent-memory modes so that the locking mechanism of multithreading can bewildered on memory usage. The kinds of memory profiling tools include 'valgrind' and 'heaptrack' which are employed on the memory for tracking the performances of the resources for allocation and also for searching for memory leaks or high fragmentation, which arises from the characteristic of multithreaded run. The experiment also focuses on investigating on how memory management strategies like garbage collection in java and memory management by the programmer on C++ affects the performance of the system when there is use of multiple threads. The results of these experiments are useful in understanding the ways in which memory is used in multithreaded systems and how their usage can be done in a way that does not favor one particular manner over another.

The effects of multithreading on I/O activity is tested by emulating different types of I/O intensive loads, including file access, network connections and database access. These tasks are chosen in order to represent a broad range of loads seen with today's systems where the involvement of I/O transactions is often a key rate-limiting factor. Asynchronous I/O, synchronous I/O and event driven models of I/O scheduling are introduced and the efficiency of multithreading in enhancing I/O throughput and minimizing latency is examined via experimentation. The utilities available include `iostat' that log statistics like I/O throughput to give a clear picture of how multi-threading influences the utilization of available resources in I/O intensive processes and `blktrace'.

The last aspect of the experimental methodology deals with analysis of the energy efficiency in multithreaded systems, which has become a focus in recent work. The prototype system is run on a multicore processor platform with facilities for power measurement built into the chip, for example Intel's RAPL technology, where power usage during system operation is monitored. The experiment measures the energy consumption of executing the test program in single-threaded and multithreaded models to determine how the workloads are traded off for energy efficiency improvements. Through these results, this research intends to make a modest addition to the current literature on energy-conscious multithreading to help elucidate how systems themselves can achieve the high performance while also limiting energy consumption in low-voltage scenarios.

Constant data being collected and analyzed in groups, statistical methods are applied in order to achieve validity and reliability of experiments in the course of the experiment phase Efficiently. CPU time, CPU load percentage, memory activity, I/O rates, and power targeting numbers are collected for every scenario and compared with appropriate statistical tests to evaluate any differences between threading models. Specifically, analysis of variance (ANOVA) is employed to compare the performance of diverse multithreading strategies across a range of applications and to guarantee that the results will be statistically significant. Moreover, regression has been used also to establish the relationship between the number of threads and the resulting efficiency of the system in order to give more generalized results for thread count in various circumstances.

Thus, the research methodology is based on theoretical analysis as well as on practical experiments conducted in order to investigate the effect of multithreading on the usage of system-resources and their performance. With a view to build a multithreaded system and evaluate this under different scenarios the objective of this research would be to present empirical results evidencing the potential of various multithreading techniques in determining the uppercase efficiency of CPU usage, memory utilization, I/O and energy consumption.

## IV. RESULTS AND DISCUSSION

From the findings of this research, the following lessons can be deduced in relation to utilization of system resources by multithreading. The information collected during the experimental phase for CPU usage, memory consumption, I/O operations, and energy consumption give a clear picture of efficiencies and compromises that are linked to different multithreading approaches.

These preliminary performance results suggest that multithreading is indeed an efficient way to optimize CPU usage rather than executing the instrumentation code in a single thread. In systems that are likely to run computationally intensive tasks, static multithreading showed that CPU usage was higher, so greatly eliminated idle time, as the processor was engaged throughout the duration of the task. Nevertheless when the thread number rose over the numeric value of cores, the process of context switching jumped in as a limiting factor. In such cases, while improvements in overall CPU performance tended to level off, and in some instances even decrease slightly where excessive threading resulted in increased levels of thread swapping, the improvements facilitated efficient system and task management. The best approach turned out to be adaptive multithreading, whereby the number of

threads can be changed depending on the load, yet this way the cores' usage remains high and the drawbacks of overthreading are avoided.

Memory consumption discussion provided quite contrasting outcomes. In all the shared-memory configurations, multithreading was more effective in terms of memory consumption where large data sets were processed. This changed the fact that in the past each thread could only access its own memory blocks, which led to increased overhead in having to allocate blocks for threads. Still, synchronization problems, particularly in fine-grained multithreading, created conflicts for using shared memory and sometimes stimulated performance degradation. Independent-memory scenarios where each threat had its memory eliminated the mentioned contention problems but consumed more memory space and therefore used more memory. In the context of data structures, where memory available is usually a major consideration for certain realizations, such a trade-off has to be made to optimize the balance of performance and memory usage (see figure 2).

On I/O performance, multithreading was tightly responsible for performance hikes especially for tasks that have asynchronous I/O operations. The findings indicated that use of multithreading created several threads to process several requests at the same time thus improving on latency and utilizing more system throughput. In cases where threads were being waited on by the I/O, the feature of other threads running improved the throughput of the system by reducing the time that threads were idle. However, the type of I/O scheduling that was under investigation had a significant impact on these results. Synchronous I/O and event driven were found to be most efficient because they didn't keep the threads waiting for I/O finish most of the time. Synchronous I/O was found to have some sort of 'bottlenecks' were observed ((Parallel and Synchronous I/O)) particularly in multi-threaded applications whereby some threads had to wait for I/O operations to be completed before continuing. Therefore, the choice of I/O model that is utilized in the application greatly influences the effectiveness with which multithreading can enhance the interaction between applications and their environment with a view of enhancing resource utilization among data intensive applications.

Nevertheless, the consumption tests showed that energy consumption was not a simple metric, leveraging multithreading in some cases and not in others. The findings revealed that, although multithreading realised performance gains, it acquired energy demands, especially in large threadcount configurations, which necessitated more context switches or CPU core invocation. Among the models of adaptive multithreading that were tested and included the dynamic increase in the number of threads based on the received work load, the efficiency of the process was highest for a certain number of threads. Here this model avoids the over-provisioning by maintaining only and only required number of active threads, which minimises the power consumption, along with the increase in the resultant performance. However, in the domain where energy issue is a priority, for example in mobile/embedded systems, the additional energy consumption involved in the context of multithreading should be considered with regard to benefits derived from it.

The study of these descriptors in the previous section is complemented with statistical analysis, where ANOVA tests further validates that the differences in CPU utilization (see figure 1), I/O, and power consumption between threading models are statistically significant. Regression analysis also confirms the conclusion that number of threads also has an optimum value with respect to the work load since beyond this value the utilization of the system resources decreases because of the overhead of time utilized in resource contention and synchronization.

Therefore, the outcomes of this research reveal that it is possible to enhance the performance of a system by introducing multithreading; however, the orthogonality of threading models and the type of utilised resources can be detrimental to the efficiency of multithreading. While static multithreading is ideal for the workload that is easy to predict in terms of the number of threads or processes, adaptive threading is extolled as more suitable when the workload may vary. In this case, issues of memory management and synchronization are still very crucial and the selection of the right I/O model will significantly determine the efficiency of the system. Last of all, it has to be noted that utilization of multithreading results in improved performance at the same time consuming more energy and this factor plays an essential role in energy critical applications. These results can be used to inform future work that is designed to fine tune multithreading policies for today's tremendous computing environments seeking both conductivity and economy (see figure 1-6)

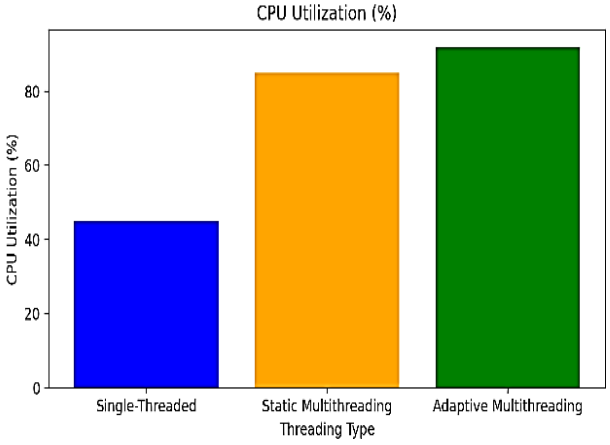


Figure 1: CPU Utilization (%)

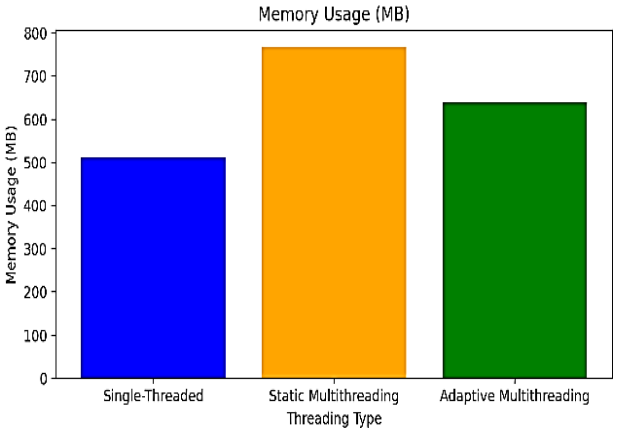


Figure 2: Memory Usage (MB)

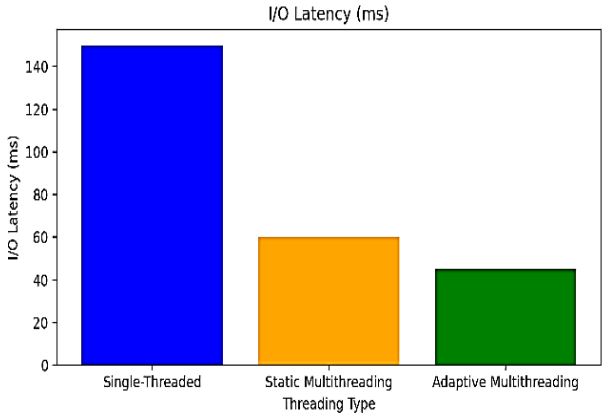


Figure 3: I/O Latency (ms)

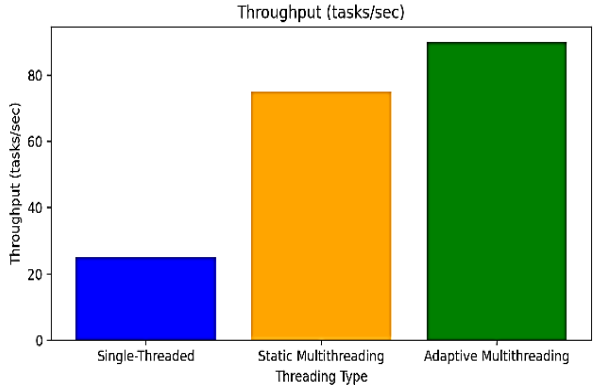


Figure 4: Throughput (tasks/sec)

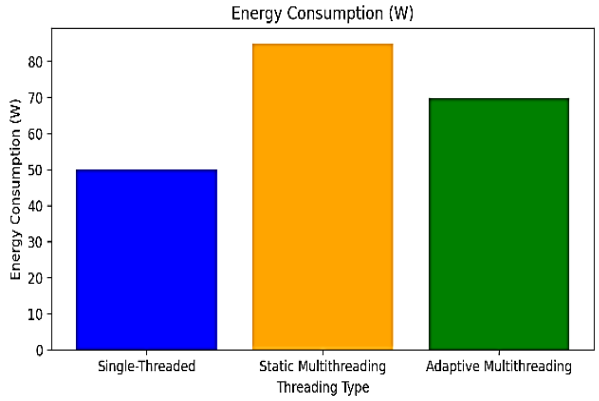


Figure 5: Energy Consumption (W)

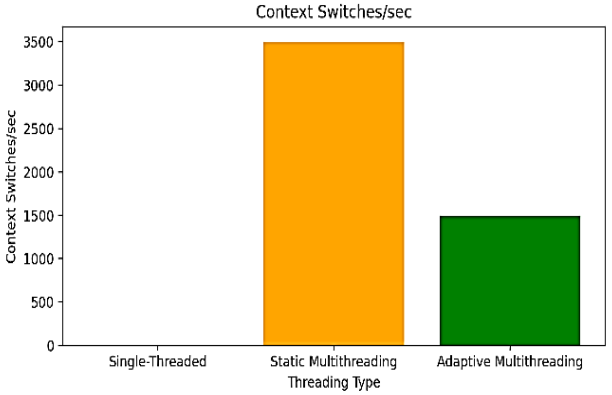


Figure 6: Context Switches/sec

## V. CONCLUSION

As it will be evidenced by the following findings of this research, multithreading offers a strong impetus to enhance the utilisation of resources and overall efficiency of a system. Multithreading can provide real advantages to CPU and I/O usage where high levels of parallelism are needed, for example where there are many calculations to perform or where there are many simultaneous I/O requests. But the efficiency in question has certain constraints. To be more concise, as the number of threads increases the system starts to experience the phenomenon of degradation, caused by context switching time and synchronization issues. This is most prevalent in shared memory because too much reliance on threads for synchronization leads to an overhead in resource contention. Another key component is memory management as using independent memory for every thread generates overhead of significant memory consumption while shared memory approach generates delays in synchronization.

Of these/methodologies, Adaptive multithreading was seen as the most effective way across the two to achieve both high performance as well as efficiency in utilization of the resources. Since adaptive threading varies the thread count in accordance with the real-time workload, it does not include unneeded extra threads that lead to drawbacks concerning context switching and data competition. This approach does not only provide an optimum condition for CPU usage but also provides better management of memory and it also provides better management of energy resources than static models of multithreading. The studies also focused on choosing the right I/O models; it is illustrated that the asynchronous types of I/O models are much more effective in the context of the multithreading paradigm than the synchronous types of I/O models which may cause performance issues.

However, in the same measure we also find that energy consumption is another aspect of multithreading and it is rather unfortunate that multithreading takes more energy than other forms of threading because the large number of threads always remain active at any given time. Overhead of energy consumption aside from preemergent cores can be a big factor affected in power sensitive environments. This suggests that although using multithreading could enhance performance, energy-conscious approaches are urgent in some forms of devices including mobile and the other embedded systems.

In conclusion, multithreading is a powerful way to optimize system performance but to obtain maximum result the specifics of work load, resources and constraints must be studied. The conclusion of this study highlights the necessity of the approach to the fine-tuning of multithreaded systems to achieve optimization in performance, power, and energy consumption to help developers and system architects enhance system performance.

## CONFLICTS OF INTEREST

The authors declare that they have no conflicts of interest.

## REFERENCES

[1] S. Saini, H. Jin, R. Hood, D. Barker, P. Mehrotra, and R. Biswas, "The impact of hyper-threading on processor

resource utilization in production applications," in *2011 18th International Conference on High Performance Computing*, Dec. 2011, pp. 1-10. Available from: https://doi.org/10.1109/HiPC.2011.6152743

[2] M. Curtis-Maury, "Improving the efficiency of parallel applications on multithreaded and multicore systems," Ph.D. dissertation, 2008. Available from: https://shorturl.at/pJtX8

[3] A. Fedorova, M. I. Seltzer, C. A. Small, and D. Nussbaum, "Performance of multithreaded chip multiprocessors and implications for operating system design," 2005. Available from: https://dash.harvard.edu/handle/1/24829606

[4] H. Wang et al., "Speculative precomputation: Exploring the use of multithreading for latency," *Intel Technology Journal*, vol. 6, no. 1, 2002. Available from: https://shorturl.at/VCylu

[5] T. Moseley, J. L. Kihm, D. A. Connors, and D. Grunwald, "Methods for modeling resource contention on simultaneous multithreading processors," in *2005 International Conference on Computer Design*, Oct. 2005, pp. 373-380. Available from: https://doi.org/10.1109/ICCD.2005.74

[6] K. Datta, "An efficient design space exploration framework to optimize power-efficient heterogeneous many-core multi-threading embedded processor architectures," Ph.D. dissertation, Univ. of North Carolina at Charlotte, 2011. Available from: https://shorturl.at/1wUQX

[7] S. Schildermans, J. Shan, K. Aerts, J. Jackrel, and X. Ding, "Virtualization overhead of multithreading in X86 state-of-the-art & remaining challenges," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 10, pp. 2557-2570, Oct. 2021. Available from: https://doi.org/10.1109/TPDS.2021.3064709

[8] F. Salgado et al., "Exploring metrics tradeoffs in a multithreading extensible processor," in *2012 IEEE International Symposium on Industrial Electronics*, May 2012, pp. 1375-1380. Available from: https://doi.org/10.1109/ISIE.2012.6237291

[9] J. Wadden, A. Lyashevsky, S. Gurumurthi, V. Sridharan, and K. Skadron, "Real-world design and evaluation of compiler-managed GPU redundant multithreading," *ACM SIGARCH Comput. Archit. News*, vol. 42, no. 3, pp. 73-84, 2014. Available from: https://doi.org/10.1145/2678373.2665686

[10] S. E. Raasch and S. K. Reinhardt, "The impact of resource partitioning on SMT processors," in *2003 12th International Conference on Parallel Architectures and Compilation Techniques*, Sep. 2003, pp. 15-25. Available from: https://doi.org/10.1109/PACT.2003.1237998

[11] W. Magro, P. Petersen, and S. Shah, "Hyper-Threading Technology: Impact on compute-intensive workloads," *Intel Technology Journal*, vol. 6, no. 1, 2002. Available from: https://shorturl.at/fXTYw

[12] A. Fedorova, M. Seltzer, C. Small, and D. Nussbaum, "Throughput-oriented scheduling on chip multithreading systems," Tech. Rep. TR-17, 2004. Available from: https://shorturl.at/I9qXI