International Journal of Modern Research
Vol. 2, No. 1, 17-27, 2022

Self-aware Execution Environment Model (SAE2) for the Performance Improvement of Multicore Systems

Surendra Kumar Shukla
Computer Science and Engineering
Graphic Era Deemed to be University, Dehradun, India
Email: surendrakshukla21@gmail.com

Vishan Kumar Gupta*
Computer Science and Engineering
Graphic Era Deemed to be University, Dehradun, India
Email: vishangupta@gmail.com

Kireet Joshi
Computer Science and Engineering
Graphic Era Deemed to be University, Dehradun, India
Email: joshikireet@gmail.com

Ankit Gupta
Computer Science and Engineering
Graphic Era Deemed to be University, Dehradun, India
Email: ankitguptamiet1500@gmail.com

Mukesh Kumar Singh
Computer Science and Engineering
IMS Engineering College, Ghaziabad, India
Email: mukesh.singh@imsec.ac.in

**Corresponding author:* Vishan Kumar Gupta

(Received on March 24, 2022; Accepted on April 5, 2022)

Abstract
Multicore systems are known for operating in a dynamic execution environment. Various conventional approaches/efforts carried out so far towards achieving the dynamism have gradually become obsolete. And there is a dire need to find out and integrate novel paradigms, research theories (such as self-awareness) in Multicore systems to find possible performance optimization alternatives. Self-awareness is one of the important principles of autonomic computing and has been shown a remarkable hope in building dynamic systems. Taking the inspiration from such systems, in this paper a "Self-aware Application Execution Environment" (SAE2) has been proposed. The aim of the SAE2 model is to explore and find out the impact of "self-awareness" in the performance of Multicore systems. The "SAE2 model" is driven through the "autonomic computing principles" and exploits parameter tuning and tradeoff attributes to leverage the Multicore system's potential. The proposed model has a feedback-based mechanism, where an application could interact with the system and signal for various performance issues (at run time) and get the inputs to get adopted as per the system resources availability. A novel "application cooperative behavior" has been introduced to address various performance issues of Multicore systems.
Keywords- Multi-core; performance; self-awareness; adaptive; feedback.

1. Introduction

Multicore processors being the backbone of numerous computing fields have revolutionized the Information Technology (IT) industry (Abbasi et al., 2021). The growth is witnessed in the field of AI, Cloud computing, data analytics and computer vision. Nevertheless, critical factors at the hardware level such as defects in manufacturing; performance degradation over time, Power-performance tradeoff hinders the true potential of Multicore systems (Alferaidi et al., 2022). Other challenges (on the software side) like task mapping/scheduling, fairness; interference is becoming the performance wall for the Multicore systems (Asbeutah et al., 2020). To address such challenges conventional approaches are not sufficient (Dhiman et al., 2022). And therefore, we need to move towards more intrinsic and abstract attributes like self-awareness with learning (Dinakarrao, 2021). Self-aware systems are those systems that are aware of their state and could adopt themselves considering the state of the overall system (Götzinger et al., 2016). Various efforts of self-awareness have been available in the literature, which attempted to find the scope/impact of autonomy in terms of models to improve performance.

In the same context, a framework inspired by self-awareness has been proposed, which takes application goals as input and fulfills them according to a fixed set of actions (Gupta and Rana, 2019a). The proposed system reduces the involvement of programmers in performance optimization.

The contribution of this article is enlisted as (Gupta and Rana, 2019b):

- Design of self-aware based SAE2 model to find out the scope for performance improvement of multi-core systems.
- Proposed a set of policies to realize the Self-awareness in the proposed model.
- Developed and integrated the application and system interface to facilitate the application programmer and system engineer.
- Finally, a concise discussion of the implementation issues.

Section 2 of this paper describe the model imperatives and their causes-effects, Section 3 is devoted to exploring the application execution flow considering the proposed policies adopted in the model. Section 4 details the proposed model and its subcomponents. Section 5 covers a thorough discussion to understand the critical factors related to model implementation and finally, the paper is concluded.

2. Model Imperatives

The model imperatives details are following (Gupta et al., 2022):

- It is assumed that the threads are going to be executed in the proposed model.
- The size of the thread is assumed to be the size of the program.
- Threads include thread control code (run time).
- Input for the model would be the application threads, related parameters. Contrary, the output will be measured through the performance metrics.
- Sensing variables (shared variables) are introduced to perform online monitoring and analysis of the system performance. Monitoring would be performed at distinct levels in the system; first at application thread-level; second at application level & third at system level.
- System also has “Global sensing variables” to monitor the overall system performance.
- Interrupts will be used as event trackers in the applications and will act for the parameter’s threshold value set by the user.

- Default services like scheduling, memory allocation provided by the existing OS will be disabled. And a set of new services will be activated which are included in the proposed interface.
- The conventional OS would only act as facilitator and will provide the available resources as per the requirement raised by the proposed model.
- Existing OS will just work as a resource allocator as per the policies decided by the proposed model.

2.1 Cause Effect Analysis

In the proposed model, to incorporate self-awareness, sensor channels have been employed that sense the changes in the system performance. Sensor channels are implemented as the shared variables. Sensor channels are interfaced with the performance counters that exist in the processor core. Performance counters could be easily accessed in the program using the available API's like PAPI (Jaiswal et al., 2015). PAPI interfaces shared variables to the performance counters to track the important events such as cache miss, CPU utilization, memory & bandwidth utilization etc.

Therefore, applications going to be executed in the proposed model contain a list of shared variables to balance their performance itself. Performance counters have also been assigned the responsibility to access the global system parameters (core utilization) globally. The global system variables are essentially calculated through the local sensing/performance variables available in the applications. Therefore, we have two types of sensing variables, local and global. Local variables monitor the performance of the application in isolation, and global variables measure the system performance in totality (Kanwal et al., 2022).

The variation in systems performance could be traced through the global variables. The controller from the application side and system side communicates to balance the overall performance of the system. Let's say, cache miss rate is higher at the system level, the global variable will witness the same. Accordingly, the application controller will track which application is causing the higher miss rate and the necessary action would be initiated. To monitor the systems performance loops at the application and at the system side continuously monitor the system state. The local loop also known as an inner loop will monitor the performance at the application side, whereas the Global loop is also known as the outer loop will manage the performance degradation at the system level. These loops will work for fixed intervals. Let's say after, each 1 million instructions (Kumar and Dhiman, 2021).

3. Proposed Policies to Incorporate Self-awareness

1. The application will be governed by the application parameters and the control file. Sensor variables declared inside the application will control/instruct the application to go slow or fast as per the system's performance.
2. The application consists with small components, like threads, procedures.
3. These components will report the performance of the applications to the application controller.
4. Self-service applications are allowed to allocate the required resources itself (by requesting the application controller). Applications would accomplish this task by calling the related service routines that exist in the controller.

5. Central authority (System controller/OS) in the system will only observe the applications. And would not interfere with the application activities related to resource allocation.
6. Central authority may warn the application for any false decision, it is up to the application to accept the recommendation or reject it.
7. Central authority in critical cases may forcefully stop the application to stop the activity during the execution of time critical applications.
8. There would be three modes in which the system may work- normal, application control, system control.
9. System/OS/Super-user may change the mode of the application execution, from normal mode to application control or system control mode.
10. All the applications must have full awareness about its environment, if the environment (Performance of the system) is changing applications should have to change their parameters accordingly.
11. All the applications must have a separate file which contains the list of parameters of the application.
12. All the applications should have a control file that contains the set of routines related to the performance of the application.
13. The applications may call the "parameter correlation routine" to check the adverse effect of changing the parameter value of a parameter.
14. Correlation of the parameters will be done with the help of the correlation rules, described in the parameter tuning routine. For example, no. of threads proportional to CPU utilization is one of the rules.
15. The applications can communicate to the other applications, system components (with the help of the controller).
16. If a resource is almost occupied, and is unable to serve other applications, could broadcast a message to all the applications that - "resource full" and could not accept further requests. In response, all the applications will slowly start releasing the related resource to balance the system performance.
17. The applications during releasing a resource will have to keep track, that the performance of the system is not affected in a large extent
18. If after broadcasting the message, if applications are still not releasing the resource, and do not agree for any "common solution" then the resource will itself discard the unusual applications forcefully.
19. The application will take the decision of releasing a resource, after executing their parameter correlation algorithm. If it is found that there is a possibility of performance degradation, of that application itself, it may report to the resource directly.
20. Resource allocation and de-allocation will be performed by the applications in a cooperative manner.
21. Thus, applications will be aware about co-applications resource requirements, & will assist each other in case of resource crises.
22. The decision of the preemption of the application will be taken by the system controller/resource only if applications fail to resolve it collectively.
23. The resources will periodically announce its status, as if it is free or heavily loaded to the applications, for the effective utilization of the resources.
24. The applications must maintain a table putting the communication history of resource release & possible reasons for non-release of acquired resources.

25. Each application will maintain a table for self-healing. Whenever an application faces a particular performance issue, the self-healing table will be referred to. Self-healing table will essentially store a set of performance issues and possible solutions in the form of routine.
26. Overall system will be governed by the two optimization strategies: first self-optimization at application level, second at system level.

4. Proposed Model

To overcome the performance issues of multi-core systems incorporating self-awareness principles, an interface between multicore systems and their application has been assumed/proposed (Kumar and Singh, 2021). Further, a hypothesis has been established as "the proposed layer being an interface between applications and the operating system would mitigate the multicore performance issues". Moreover, to leverage the potential of the proposed model, applications must be transformed and become self-aware. To prove the given hypothesis an Application Self-control Engine (**ASCE**) has been designed and kept between application and OS. ASCE further includes two independent modules named "application control engine" (ACE) and system control engine (SCE). ACE and SCE includes the necessary services to enable an application to be transformed as self-aware. The services have been incorporated from the autonomic principles such as self-healing, self-protection, and self-control. These services are nothing but routines and related interfaces. The purpose of ACE is to interact with applications, whereas SCE interacts with the OS and low-level services. The distinct modules of the proposed model are detailed below (Park et al., 2019).

4.1 ACE Modules

(i) Profilers

Profilers are the programs that profiles and characterize the application behavior. Profiles help in making various scheduling and memory allocation decisions for the applications.

(ii) Application Thread Flow Graph

This module is responsible for creating the thread flow graph. The thread flow graph includes the information about which thread is using which memory object. Flow graph also shows which thread is dependent on other threads. It helps in knowing different options for the thread execution during the shared resources crisis in the system.

(iii) Application Self-control Services

This module provides the appropriate control & management services to the applications. The services related with autonomic aspects like self-healing, self-optimizing, self-protecting will be available here.

(iv) Application Thread to Core Mapping Services

These services would help the application to acquire the suitable core among a set of candidate cores.

(v) Application Threads to Memory Object Mapping Services

These services take support from the application thread flow graph to know how many threads are pointing for a memory object. It helps in solving the shared resource contention problems.

(vi) Application History Sensitive Data and Knowledge

This module contains the application's historical data related to the thread-to-core mapping. Information related to the application like a priority, how many instructions cause the memory references. The historical information helps for the application-to-core allocation & memory allocations.

4.2 SCE Modules

(a) Application Online Monitoring and Analysis Module

Online monitoring of performance is performed by the System control engine. The online monitoring module is responsible for finding the hot spot (the portion of applications causing the performance degradation) in the application, also monitoring the parameter tuning process. Application run time manager (ARM) is a program that runs inside the SCE to support the online monitoring and analysis module.

(b) Application Runtime Manager

This module fulfills all the requirements of the application during its execution. Application runtime manager is the agent that coordinates with the application controller engine (ACE) for the availability of appropriate services to the application. It helps applications to take performance-related decisions at runtime.

(c) ARM Plan Activation

ARM when sensing that an application's performance is decreasing, will select the most appropriate plan from the knowledge repository and will instruct the Thread run time module (TRM). Here the plan is a parameter value increase/decrease for the performance stabilization.

(d) Parameter Threshold Value Monitor (PTVM)

PTVM is a service that monitors applications. PTAM works as an event tracker in the applications. The main task of PTAM is to keep track of the occurrence of specific activity related to the performance in the application. For example, a routine is taking more time for execution as expected. Additionally, one of the threads creating contention for the shared cache is also an event needed to observe. The event server does the task of tracking with a threshold table that contains the threshold value of each parameter of the application. The event server matches the expected value with the threshold value and then passes the events to the Run time manager. Other tasks performed by the module are, monitoring performance decreasing, resource conflict (Politou et al., 2021).

(e) Migrator

It is a sub-module of the application online analysis & control module. It decides at run time which component of the application will run in which processor core.

(f) Application Tuning Rules

Application tuning rules contain the restrictions and parameter tuning configuration. Rules like data locality are inversely proportional to the shared cache contention. Other Rules like increasing the cache size will increase the hit rate. Increasing the I/O activity will choke the interconnection network.

(g) Application Self-healing Plans

Self-healing plans are the group of rules to address some performance problem issues like contention. The difference between rule and plan is that rules are atomic & plans are the set of rules. For a particular performance problem, there could be various plans & each plan may have a different set of rules (Salami et al., 2020).

(h) Application Self-optimization

Application self-optimization is also the set of rules to minimize the application execution time. Self-optimization rules are used by the applications to optimize their threads performance.

(i) Knowledge Stores

A knowledge repository is a table that stores the set of already executed plans. The set of plans is the parameter tuning rules. Rules like when the interconnection network is full then decrease the parallelism parameter. Rules have one another interpretation set of possible combinations of

parameters and their values. These plans (rules) are made based on the different performance problems which may come in the system. The knowledge repository will be updated from time to time after a successful solution to a performance problem comes into the system. New plans will be updated in the table for the successful heuristic optimization solution (Vaishnav et al., 2021).

(j) Policy Engine

The policy engine is the brain of the ACSE Model. The policy engine contains the overall rules/regulations, protocols that need to be followed by the modules. The policy engine contains the policies related to resource allocation, processor allocation, contention solution, and synchronization issues in the system, concurrency policies. Other issues like which process will get higher priority, application communication policies, voting policies all are stored in the policy engine. Policies are the set of atomic rules that need to follow all the applications running in the system (Yadav et al., 2022).

4.3 Working of Proposed Model

(a) Screening of the Application

Application will first go for the initial investigation. Investigations related to the type of application, verification for malware. Application tunable parameters will also be gathered by the ACE.

(b) Self resource Allocation

When an application requires the resources, it will request them from the application controller engine. The application will call the resource requirement estimation service (RRES) that is available at the ACE. That service will estimate the resource requirement of that application. Application may use existing service, or it may use their own available in their source code. Applications will put their requirements to the ACE; ACE will show the exact status of the resources like available main memory, cache etc., at that time. The application will then call the performance estimation after allocation service (PEAS). This service will calculate the effect on the performance of this application will be submitted to the system. It will allocate the resource to the application. If (PEAS) show that after submission of the application performance of the system will hamper or the application will take more time for their execution, the application may wait for some time. It may call the PEAS service routine for the appropriate time to apply. Application if it is a critical one may enter in the system without considering the performance problem. For understanding the concept, we are taking the example of sorting some numbers. The request of the resource allocation will first go to the ACE; ACE will transfer to the SCE. ACE cannot go to the system area directly; it will go through the SCE only. The application will use resource allocation related algorithms.

(c) Self-aware Application Processor Scheduling

Application after getting the required resources is ready for execution. The application will now call for the most appropriate processor scheduling algorithm from the knowledge engine. Knowledge engines will keep a list of scheduling algorithms ranked as per their past performance; they are also bifurcated as per the type of application. The algorithm for the image processing is grouped, the same as the algorithm for real-time application. Scheduling algorithms are categorized as I/O sensitive applications that may need a slow core and CPU incentive algorithm may require the fast core. Scheduling algorithms if they schedule the applications as per the characteristics of the applications it may solve various issues like shared cache contention.

The scheduling policy is part of the policy engine. Scheduling policy contains one important point, those threads which take more time for their execution will be allocated to the fixed processor core. The phenomenon of putting a thread to the same core all the time in the

scheduling process is called processor core affinity. The application will also prefer the core which has slightly less load as shown in Figure 1.

(d) Self-aware Resource Allocation and Parameter Tuning

Performance tuning will be accomplished through the loops. The application executes its thread for an iteration & collects the initial value for the tuning parameters, after that, it calculates the next values for the threads through executing self-aware parameter tuning having criteria such as throughput or the CPU utilization or main memory utilization. The online tuning will be done by getting the initial value of the parameters through executing for one iteration. As we know that, the application parameter tuning in our proposed model is through the cooperative manner, each the application needs to check the other application & system performance also before making any decision.

Suppose we have two applications A1 & A2 and we have some resources like R1 and R2 and both have only one instance. Application A1 has taken R1 resources only. Now if A2 wants to get R1 resources, it needs to claim it from A1 by making a request or A1 may itself release some percentage of resources for R1 to allocate to A2. Here each application (Old) needs to be aware of the new events, new applications entering the system. New applications would be aware of the old application's resources.

Here, there might be a possibility that both the applications are waiting and are not making sufficient progress. The said issue could be solved by identifying the behavior and characteristics of the application. Suppose a new application just entered in the system is time-critical, it would request an old application (memory-bound) to release the required resources. As per the policy and ranking of the applications both the applications will decide in a cooperative manner who will release what number of resources. There could be a concept of survivability, it means an application will release a few resources such that it can survive itself and make progress. Survivability will ensure that the application will not sit completely ideal. It will execute slowly and progress.

5. Conclusion

Multicore systems are rapidly changing and demanding robust policies to optimize performance. Conventional approaches have shown a definite performance improvement. To have exponential growth in performance to support the upcoming computing applications there is a dire need to explore the novel paradigms and integrate them into the Multicore systems. In this research article, a well-known computing paradigm that works in the principles of self-awareness has been proposed. Integrating autonomic computing in Multicore systems would certainly help to gain the required performance from the multicore systems. The Self-aware execution environment termed SAE2 has been proposed. The effectiveness of the model has been prepositioned for the self-aware resource allocation and self-aware application-to-core mapping. The cooperative behavior of applications to enhance the system performance has been also discussed. The set of protocols in terms of policies have been elaborated. The proposed model has covered all the contemporary performance issues of Multicore systems and their possible solutions. The implementation aspects that have not been covered in this research article could be extended in future work.

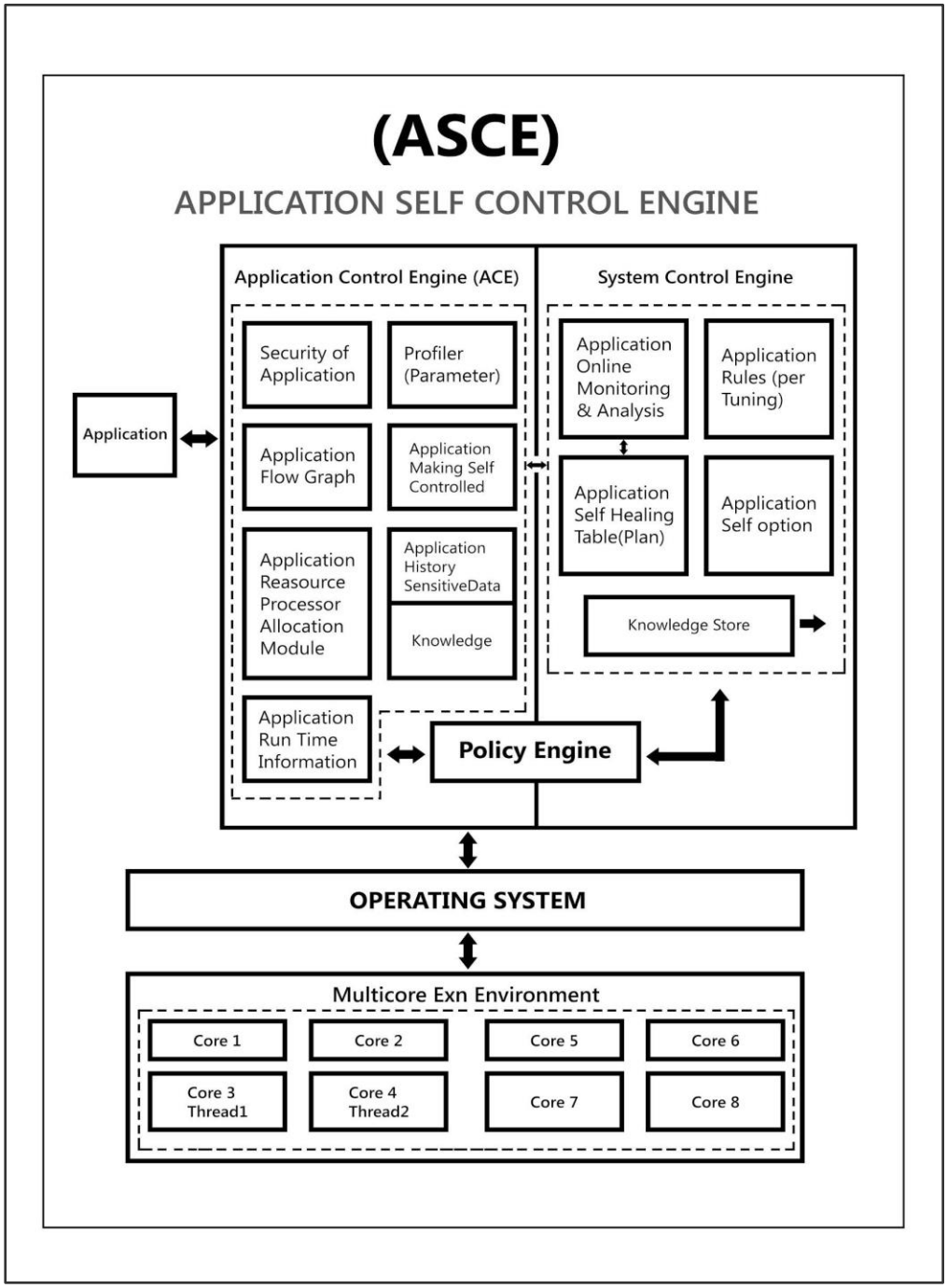

Figure 1: Proposed SAE2 Model for Performance Improvement.

Conflict of Interest
The authors confirm that there is no conflict of interest to declare for this publication.

Acknowledgments
We sincerely thank to the Department of Computer Science and Engineering of Graphic Era Deemed to be University, Dehradun for providing the facility of various multicore system for the performance improvement during execution of data.

References

Abbasi, S. I., Kamal, S., Gochoo, M., Jalal, A., & Kim, K. (2021). Affinity-Based Task Scheduling on Heterogeneous Multicore Systems Using CBS and QBICTM. *Applied Sciences*, *11*(12), 5740.

Alferaidi, A., Yadav, K., Alharbi, Y., Viriyasitavat, W., Kautish, S., & Dhiman, G. (2022). Federated Learning Algorithms to Optimize the Client and Cost Selections. *Mathematical Problems in Engineering*, *2022*.

Asbeutah, A. M., AlMajran, A. A., Brindhaban, A., & Asbeutah, S. A. (2020). Comparison of radiation doses between diagnostic full-field digital mammography (FFDM) and digital breast tomosynthesis (DBT): a clinical study. *Journal of Medical Radiation Sciences*, *67*(3), 185-192.

Dhiman, G., Rashid, J., Kim, J., Juneja, S., Viriyasitavat, W., & Gulati, K. (2022). Privacy for Healthcare Data Using the Byzantine Consensus Method. *IETE Journal of Research*, 1-12.

Dinakarrao, S. M. P. (2021). Self-aware power management for multi-core microprocessors. *Sustainable Computing: Informatics and Systems*, *29*, 100480.

Götzinger, M., Rahmani, A. M., Pongratz, M., Liljeberg, P., Jantsch, A., & Tenhunen, H. (2016, September). The role of self-awareness and hierarchical agents in resource management for many-core systems. In *2016 IEEE 10th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSOC)* (pp. 53-60). IEEE.

Gupta, V. K., & Rana, P. S. (2019a). Activity assessment of small drug molecules in estrogen receptor using multilevel prediction model. *IET Systems Biology*, *13*(3), 147-158.

Gupta, V. K., & Rana, P. S. (2019b). Toxicity prediction of small drug molecules of androgen receptor using multilevel ensemble model. *Journal of Bioinformatics and Computational Biology*, *17*(05), 1950033.

Gupta, V. K., Shukla, S. K., & Rawat, R. S. (2022). Crime tracking system and people's safety in India using machine learning approaches. *International Journal of Modern Research*, *2*(1), 1-7.

Jaiswal, N., Gupta, V. K., & Mishra, A. (2015, March). Survey paper on various techniques of recognition and tracking. In *2015 International Conference on Advances in Computer Engineering and Applications* (pp. 921-925). IEEE.

Kanwal, S., Rashid, J., Kim, J., Juneja, S., Dhiman, G., & Hussain, A. (2022). Mitigating the Coexistence Technique in Wireless Body Area Networks By Using Superframe Interleaving. *IETE Journal of Research*, 1-15.

Kumar, R., & Dhiman, G. (2021). A comparative study of fuzzy optimization through fuzzy number. *International Journal of Modern Research*, *1*(1), 1-14.

Kumar Gupta, V., & Singh Rana, P. (2021). Ensemble technique for toxicity prediction of small drug molecules of the antioxidant response element signalling pathway. *The Computer Journal*, *64*(12), 1861-1875.

Park, S., Song, D., Jang, H., Kwon, M. Y., Lee, S. H., Kim, H. K., & Kim, H. (2019, September). Interference analysis of multicore shared resources with a commercial avionics RTOS. In *2019 IEEE/AIAA 38th Digital Avionics Systems Conference (DASC)* (pp. 1-10). IEEE.

Politou, E., Alepis, E., Virvou, M., & Patsakis, C. (2021). *Privacy and Data Protection Challenges in the Distributed Era*. Springer International Publishing AG.

Salami, B., Noori, H., & Naghibzadeh, M. (2020). Fairness-aware energy efficient scheduling on heterogeneous multi-core processors. *IEEE Transactions on Computers*, *70*(1), 72-82.

Vaishnav, P. K., Sharma, S., & Sharma, P. (2021). Analytical review analysis for screening COVID-19 disease. *International Journal of Modern Research*, *1*(1), 22-29.

Yadav, K., Alshudukhi, J. S., Dhiman, G., & Viriyasitavat, W. (2022). iTSA: an improved Tunicate Swarm Algorithm for defensive resource assignment problem. *Soft Computing*, 1-9.