STM JOURNALS

Journal of
Operating Systems
Development & Trends

ISSN: 2454-9355
Volume 12, Issue 1, 2025
January–April
DOI (Journal): 10.37591/JoOSDT

https://journals.stmjournals.com/joosdt

Review JoOSDT

A Hybrid Algorithm for Processor Scheduling Using Game Theory Variants

Yamuna Mundru[1], Manas Kumar Yogi[2,*]

Abstract

This study proposes a novel hybrid algorithm for processor scheduling in modern operating systems, integrating the strengths of traditional scheduling methods with game theory variants. Traditional schedulers often struggle to adapt to dynamic workload changes, leading to suboptimal performance. Our hybrid approach addresses this by treating processes as "players" in a game, where the "payoff" is CPU time. A base scheduler (e.g., Weighted Fair Queuing, Earliest Deadline First) provides a foundation for fairness and efficiency. A game theory module then refines scheduling decisions by dynamically adjusting parameters based on process utilities, reflecting their importance, resource needs, and behavior. We explore various game theory approaches, including non-cooperative, cooperative, and evolutionary games, each offering different trade-offs between individual process benefit and overall system performance. The algorithm dynamically adapts scheduling parameters (e.g., priorities, time slices) based on the game's outcome. This combined method strives to create a more effective balance between fairness, efficiency, and adaptability. We discuss the algorithm's components, including utility function design and game variant selection, and analyze its potential advantages and challenges, such as computational complexity and parameter tuning. Future research directions, including exploring machine learning for adaptation, are also outlined.

Keywords: Game theory, processor, operating system, convoy effect, scheduling

INTRODUCTION

Processor scheduling is a fundamental aspect of operating system design, responsible for allocating CPU time among competing processes. The scheduler's goal is to optimize various performance metrics, including fairness, efficiency, responsiveness, and throughput, while satisfying diverse application requirements. Traditional scheduling algorithms, such as First-Come-First-Served (FCFS), Round Robin (RR), Priority Scheduling, and their variants, have served as the backbone of operating systems for decades. However, these algorithms often struggle to adapt effectively to the dynamic and unpredictable nature of modern workloads. Contemporary systems face a complex mix of interactive applications, batch processes, real-time tasks, and background services, each with unique resource demands and performance expectations. Static scheduling policies, inherent in many traditional approaches, can lead to suboptimal resource utilization, unfair allocation of CPU time, and poor responsiveness, especially in the face of fluctuating workloads [1].

The limitations of traditional scheduling algorithms have motivated research into more

***Author for Correspondence**
Manas Kumar Yogi
E-mail: manas.yogi@gmail.com

[1]Assistant Professor, Department of Computer Science and Engineering, Artificial Intelligence & Machine Learning, Pragati Engineering College (Autonomous), Surampalem, Andhra Pradesh, India
[2]Assistant Professor, Department of Computer Science and Engineering, Pragati Engineering College (Autonomous), Surampalem, Andhra Pradesh, India

Received Date: February 21, 2025
Accepted Date: February 25, 2025
Published Date: March 18, 2025

Citation: Yamuna Mundru, Manas Kumar Yogi. A Hybrid Algorithm for Processor Scheduling Using Game Theory Variants. Journal of Operating Systems Development & Trends. 2025; 12(1): 48–56p.

intelligent and adaptive scheduling strategies. One promising direction involves the application of game theory, a mathematical framework for analyzing strategic interactions among rational agents. In the context of processor scheduling, processes can be viewed as "players" competing for the "resource" of CPU time. Each process has its own "preferences" or "utilities" regarding how much CPU time it receives, and the scheduler acts as the "mechanism" that determines the allocation. By framing the scheduling problem as a game, we can leverage game-theoretic concepts to design algorithms that dynamically adapt to the complex interplay between processes.

Game theory offers a rich set of tools for analyzing and designing scheduling algorithms. Different game variants, such as non-cooperative, cooperative, and evolutionary games, can be employed to model various types of process interactions. In a non-cooperative game, processes act selfishly to maximize their own utility, while in a cooperative game, processes can collaborate to achieve a mutually beneficial outcome. Evolutionary games model how the "strategies" of processes (e.g., their resource demands) evolve over time, and how the scheduler can adapt its policies accordingly [2].

This study proposes a novel hybrid algorithm for processor scheduling that combines the strengths of traditional scheduling methods with the adaptive power of game theory. Our approach integrates a base scheduler, which provides a foundation for fairness and efficiency, with a game theory module that dynamically refines scheduling decisions. The base scheduler can be any well-established algorithm, such as Weighted Fair Queuing (WFQ) for fairness-based allocation or Earliest Deadline First (EDF) for real-time systems. The game theory module then analyzes the current state of the system, considering factors like process priorities, resource consumption, deadlines, and recent CPU burst lengths, and adjusts the scheduling parameters accordingly. This dynamic adaptation allows the scheduler to respond effectively to changes in the workload and optimize for various performance metrics [3].

The key innovation of our hybrid algorithm lies in its ability to dynamically adapt to the "strategic interactions" between processes. By treating processes as players in a game, the scheduler can learn about their behavior and adjust its policies to achieve a better balance between individual process needs and overall system performance. For example, if a high-priority process frequently exhibits short CPU bursts, indicating interactive behavior, the scheduler can temporarily boost its priority to improve responsiveness, even if its initial priority was lower. This type of dynamic adjustment, based on observed process behavior, is difficult to achieve with traditional static scheduling algorithms.

BACKGROUND

Processor scheduling is a critical function within operating systems, responsible for allocating the central processing unit (CPU) among the various processes competing for its attention. The scheduler acts as a traffic controller, determining which process gets to run and for how long. The primary objective of a scheduler is to optimize system performance according to a set of predefined criteria. These criteria can include maximizing throughput (the number of processes completed per unit time), minimizing turnaround time (the total time taken to complete a process), ensuring fairness (giving each process a reasonable share of CPU time), and providing responsiveness (quickly reacting to user input or real-time events). However, achieving these goals simultaneously, especially in dynamic environments, presents a significant challenge.

Traditional scheduling algorithms, while effective in certain scenarios, often fall short when dealing with the complexities of modern workloads. These algorithms can be broadly categorized into several types:

- *First-Come-First-Served (FCFS):* Processes are executed in the order they arrive. While simple to implement, FCFS can lead to long waiting times for short processes if a long process arrives first, a phenomenon known as the "convoy effect". For example, if a long batch job arrives before several interactive processes, the interactive processes will experience significant delays [4].

- *Round Robin (RR):* Each process is given a fixed time slice (quantum) of CPU time. If a process does not complete within its time slice, it is pre-empted and moved to the back of the queue. RR provides better responsiveness than FCFS, especially for interactive processes. However, it can introduce overhead due to frequent context switching. If the quantum is too small, the overhead becomes excessive; if it is too large, RR behaves more like FCFS [5].
- *Priority Scheduling:* Processes are assigned priorities, and the scheduler runs the highest-priority process. Priorities can be static (assigned at process creation) or dynamic (adjusted during execution). While priority scheduling can be effective for important tasks, it can lead to starvation if low-priority processes are never given a chance to run. For example, a critical system process might be given a very high priority, potentially starving other essential background tasks.
- *Shortest Job First (SJF):* The scheduler runs the process with the shortest estimated execution time. SJF is optimal in terms of minimizing average turnaround time. However, it requires knowledge of the future execution time, which is often not available. Preemptive versions of SJF, like Shortest Remaining Time First (SRTF), can improve responsiveness but still require estimations of remaining execution time.
- *Multilevel Queue Scheduling:* This approach divides the ready queue into multiple queues, each with its own scheduling algorithm. Processes are assigned to queues based on their characteristics (e.g., interactive vs. batch). This allows for more specialized scheduling strategies, but it can be complex to configure and manage.
- *Real-Time Scheduling:* These algorithms are designed to meet strict deadlines imposed by real-time applications. Examples include Earliest Deadline First (EDF) and Rate Monotonic Scheduling (RMS). Real-time scheduling is crucial in systems where timing constraints are critical, such as industrial control or robotics.

The limitations of these traditional algorithms stem primarily from their static nature. They often rely on fixed parameters or policies that do not adapt well to the dynamic changes in workload characteristics. Modern systems, however, are characterized by a diverse mix of applications with varying resource demands and performance requirements. Workloads can fluctuate significantly over time, making it difficult for static schedulers to maintain optimal performance. For instance, a system might experience a surge in interactive user activity during peak hours, followed by a period of heavy batch processing overnight. A static scheduler might be unable to efficiently handle these shifts in workload, leading to either poor responsiveness during peak hours or underutilization of resources during off-peak hours.

Traditional schedulers often treat processes as independent entities, ignoring the potential for cooperation or competition between them. In reality, processes can interact with each other in various ways, sharing resources, communicating, or competing for access to shared data. A scheduler that ignores these interactions may make suboptimal decisions, leading to inefficiencies and unfairness. For example, a set of related processes might benefit from being scheduled together to minimize inter-process communication overhead. A traditional scheduler that treats these processes independently might miss this opportunity for optimization [6].

The need for more intelligent and adaptive scheduling strategies has led researchers to explore the application of techniques from other fields, such as artificial intelligence and game theory. Game theory, in particular, offers a powerful framework for analyzing strategic interactions among rational agents, making it a natural fit for modeling the competition and cooperation between processes in a scheduling context as shown in Table 1.

NOVEL PROPOSITION

Core Idea: The algorithm leverages game theory to dynamically adjust scheduling parameters based on the "strategic interactions" between processes vying for CPU time. This allows the scheduler to adapt to changing workload characteristics and prioritize processes based on their importance and behavior.

Table 1. Taxonomy of Processor scheduling algorithms [7–9].

Processor Scheduling Algorithm	Design Aspect	Merits	Demerits	Real-Time Application Domains
Single-core Scheduling	Traditional scheduling on a single CPU core	Simple implementation	Inefficient for multi-core processors	Legacy systems, Basic embedded systems
		Predictable performance	Not scalable	
Symmetric Multiprocessing (SMP) Scheduling	Evenly distributes processes across multiple cores	Efficient CPU utilization	High overhead in managing multiple cores	General-purpose OS, Web servers, Gaming systems
		Load balancing	Potential cache coherence issues	
Asymmetric Multiprocessing (AMP) Scheduling	Master-slave model where one core handles scheduling	Reduces scheduling overhead	Underutilization of slave processors	Embedded systems, Real-time avionics
		Simplifies synchronization	Less flexible	
Gang Scheduling	Groups related processes (threads) to run simultaneously on multiple cores	Reduces context switching overhead	Requires job synchronization	Parallel computing, Supercomputers
		Improves performance for parallel applications	Can lead to CPU underutilization	
Affinity Scheduling (Soft and Hard Affinity)	Tries to keep a process on the same CPU core	Improves cache performance	Can lead to load imbalance	Databases, Cloud computing, Virtualization
		Reduces context switching	Less flexibility in scheduling	
Completely Fair Scheduler (CFS)	Uses a red-black tree to schedule tasks fairly	Provides fairness between tasks	High computational overhead	Modern Linux-based OS, Cloud workloads
		Lowers starvation issues	May not be optimal for real-time tasks	
Deadline Scheduling (Earliest Deadline First (EDF))	Prioritizes tasks with the earliest deadline	Guarantees deadline adherence for real-time tasks	High scheduling overhead	Hard real-time systems, Autonomous vehicles, Medical systems
			Inefficient under overload conditions	
Proportional Share Scheduling (Lottery Scheduling, Fair-Share)	Assigns execution chances based on resource allocation shares	Ensures fair CPU usage	Less predictable in real-time tasks	Cloud computing, Virtual machines
		Good for multi-user environments	Randomness may cause delays	
Two-Level Scheduling	Combines long-term and short-term scheduling	Efficient for handling large numbers of processes	Requires more memory for swapped-out processes	Time-sharing systems, Batch processing
O(1) Scheduler (Legacy Linux)	Uses priority arrays for fast scheduling decisions	Low scheduling latency	Poor interactivity for desktop applications	Embedded systems, Low-latency applications
		Efficient for real-time workloads		

Components

1. *Base Scheduler:* A traditional scheduling algorithm (e.g., Weighted Fair Queuing (WFQ), Earliest Deadline First (EDF) with some modifications) serves as the foundation. This provides a baseline level of fairness and efficiency. WFQ can be used for fairness based on assigned weights to processes, or EDF for real-time systems.
2. *Game Theory Module:* This module employs a game-theoretic approach to refine the scheduling decisions made by the base scheduler. It treats processes as "players" in a game where the "payoff" is CPU time.

3. *Utility Function:* Each process is assigned a utility function that reflects its importance, resource requirements, and behavior. This function can consider factors like:
   - *Priority:* User-assigned or system-determined priority.
   - *Resource Consumption:* CPU, memory, I/O usage.
   - *Deadline:* For real-time processes.
   - *Recent CPU Burst Lengths:* To predict future needs.
   - *Waiting Time:* How long the process has been waiting.
4. Game Variants: Several game theory variants can be employed:
   - *Non-Cooperative Game:* Processes act selfishly to maximize their own utility. The scheduler acts as a "referee", allocating CPU time based on the Nash Equilibrium or other solution concepts. This might be suitable for general-purpose systems where processes are independent.
   - *Cooperative Game:* Processes can cooperate (e.g., through inter-process communication) to improve the overall system performance. The scheduler can facilitate cooperation by providing mechanisms for resource sharing and negotiation. This could be useful in specialized systems with well-defined process interactions.
   - *Evolutionary Game:* The scheduler dynamically adjusts the "rules of the game" (e.g., weights assigned to processes) based on the observed behavior of the processes. This allows the system to adapt to long-term changes in the workload.
5. *Hybrid Approach:* The base scheduler makes initial scheduling decisions. The game theory module then analyzes the current state of the system (process utilities, resource availability) and adjusts the scheduling parameters (e.g., priorities, time slices) to optimize the overall system performance according to the chosen game-theoretic approach.

Example (Non-Cooperative)

Imagine using WFQ as the base scheduler. Each process has a weight. The game theory module could observe recent CPU burst lengths. If a high-priority process has consistently short bursts, it might indicate interactive behavior. The game theory module could temporarily increase its weight to improve responsiveness, even if its initial weight was lower. This is a dynamic adjustment based on observed behavior, not a static priority.

ADVANTAGES

Adaptability

Modern workloads are dynamic, with varying mixes of interactive, batch, and real-time processes. A key strength of the game-theoretic approach is its inherent adaptability. The scheduler is not bound by static rules. The game theory module continuously monitors process behavior (CPU bursts, I/O requests, memory usage) and adjusts scheduling parameters (priorities, time slices, weights) in response to these changes [10]. For example, if a surge of I/O-bound processes occurs, the scheduler can dynamically favor them to maximize overall system throughput. Conversely, if CPU-intensive processes dominate, the scheduler can adjust to ensure fairness and prevent starvation. This dynamic adaptation allows the system to maintain optimal performance even under fluctuating workloads.

Fairness

While game theory can model selfish behavior, it does not preclude fairness. The base scheduler provides a foundation of fairness, ensuring that all processes receive a minimum share of CPU time. This could be implemented using algorithms like Weighted Fair Queuing (WFQ), where each process is assigned a weight representing its relative importance. The game theory module then refines this baseline fairness. It might temporarily deviate from strict weight-based allocation to improve overall system performance or responsiveness, but the underlying fairness provided by the base scheduler acts as an anchor, preventing extreme imbalances. For example, even if a high-priority process is favored due to its interactive nature, the base scheduler ensures that lower-priority processes eventually receive their fair share of CPU time.

Performance

The hybrid algorithm offers flexibility in optimizing for different performance metrics. By adjusting the utility functions and selecting appropriate game variants, the scheduler can be tuned to prioritize specific goals. For instance, if the primary objective is to maximize throughput, the utility functions can be designed to reward processes that utilize the CPU efficiently. If responsiveness is more critical, the utility functions can prioritize interactive processes with short CPU bursts. Different game variants also offer different trade-offs. Non-cooperative games might be suitable for maximizing individual process performance, while cooperative games can be used to optimize for overall system throughput by encouraging processes to share resources [11]. This adaptability allows the algorithm to be tailored to the specific needs of the system.

Responsiveness

Interactive processes, characterized by short CPU bursts and frequent user input, require quick response times to provide a good user experience. The game-theoretic approach can prioritize these processes through dynamic weight adjustments. The game theory module can monitor the CPU burst lengths of processes and identify those exhibiting interactive behavior. It can then temporarily increase the weights or priorities of these processes, giving them preferential access to the CPU. This dynamic adjustment ensures that interactive processes receive the attention they need, even if they have lower static priorities. For example, a user typing commands in a terminal will experience a more responsive system because the shell process will be given preferential treatment when it needs to process user input.

CHALLENGES

Computational Complexity

Game theory, while powerful, can be computationally expensive. Solving complex games, especially those with a large number of players (processes in our case), can require significant processing power and time. As the number of processes increases, the complexity of the game grows exponentially, potentially making real-time scheduling decisions infeasible. Therefore, efficient approximation algorithms are crucial. Instead of seeking the exact Nash Equilibrium or optimal solution, the algorithm might need to settle for a near-optimal solution that can be computed quickly. Techniques like heuristic search, genetic algorithms, or reinforcement learning could be employed to find good solutions within a reasonable timeframe. The trade-off between solution quality and computational cost needs to be carefully considered.

Utility Function Design

The utility function is a critical component of the game-theoretic approach. It quantifies the "value" or "preference" of each process with respect to CPU time. Designing appropriate utility functions that accurately reflect process importance and system goals is a significant challenge. These functions need to consider a variety of factors, such as process priority, resource consumption (CPU, memory, I/O), deadlines (for real-time tasks), and recent process behavior. Determining the relative weights of these factors and combining them into a single utility function can be complex and requires careful consideration of the specific system requirements. Furthermore, the utility functions might need to be dynamically adjusted over time as the system's goals or workload characteristics change [12].

Parameter Tuning

The hybrid algorithm typically has several parameters that need to be tuned for optimal performance. These parameters might include the weights assigned to different factors in the utility functions, the parameters of the game-solving algorithm, and the frequency with which the game theory module is invoked. Finding the optimal values for these parameters can be a complex optimization problem, often requiring extensive experimentation and simulation. Automated parameter tuning techniques, such as machine learning algorithms, could be employed to simplify this process.

Overhead

The introduction of the game theory module adds computational overhead to the scheduling process. The module needs to collect information about the processes, solve the game, and adjust the scheduling parameters. This overhead needs to be carefully managed to ensure that it does not outweigh the benefits of the adaptive scheduling strategy. The frequency with which the game theory module is invoked is a key factor [13]. Invoking it too frequently can lead to excessive overhead; while invoking it too infrequently can reduce the algorithm's adaptability. Finding the right balance is essential. Furthermore, efficient implementation of the game theory module and the data collection mechanisms is crucial to minimize overhead.

FUTURE RESEARCH

Developing Efficient Algorithms for Solving the Games

The computational complexity of solving game-theoretic problems, particularly with a large number of processes, poses a significant challenge. Future research should focus on developing efficient algorithms for approximating solutions to these games. This could involve exploring various techniques [14–17]:

- *Heuristic Search:* Algorithms like genetic algorithms, simulated annealing, or tabu search can be employed to find near-optimal solutions within a reasonable timeframe. These methods can explore the solution space efficiently without guaranteeing the absolute best outcome.
- *Reinforcement Learning:* Reinforcement learning agents can learn to approximate the optimal strategies for the processes by interacting with the scheduling environment. This approach can be particularly useful in dynamic environments where the game's characteristics change over time.
- *Distributed Algorithms:* For large-scale systems, distributed algorithms can be used to solve the game across multiple processors or nodes, reducing the computational burden on a single entity.

Designing Robust Utility Functions

The utility function is a cornerstone of the game-theoretic approach, representing the preferences and goals of each process. Future research should investigate methods for designing robust utility functions that accurately capture process importance and system objectives. This could involve:

- *Adaptive Utility Functions:* Utility functions should be able to adapt to changing system conditions and process behavior. Machine learning techniques could be used to learn and adjust the parameters of the utility functions based on observed data.
- *Multi-Objective Optimization:* Scheduling often involves multiple conflicting objectives (e.g., fairness, responsiveness, throughput). Research should explore multi-objective optimization techniques to design utility functions that balance these competing goals.
- *User-Defined Utilities:* Allowing users or system administrators to define or customize utility functions could provide greater flexibility and control over the scheduling process.

Exploring Different Game Theory Variants

Different game theory variants offer different trade-offs in terms of complexity, fairness, and efficiency. Future research should explore the suitability of various game-theoretic models for processor scheduling:

- *Evolutionary Games:* These games can model how process behavior and scheduler policies evolve over time. Research should investigate how evolutionary game theory can be used to design adaptive scheduling algorithms that respond to long-term changes in the workload.
- *Mechanism Design:* This area of game theory focuses on designing the "rules of the game" (i.e., the scheduling mechanism) to achieve desired system outcomes. Research should explore how mechanism design can be applied to processor scheduling to create algorithms that incentivize processes to behave in a way that benefits the overall system.
- *Coalition Formation:* Processes might form coalitions to improve their collective payoff. Research should investigate how coalition formation can be incorporated into scheduling algorithms to optimize resource allocation among groups of related processes.

- *Evaluating Performance in Various Workload Scenarios:* Thorough performance evaluation is essential to assess the effectiveness of the hybrid algorithm.

Future research should involve [18]:

- *Realistic Workload Models:* Using realistic workload models that reflect the characteristics of modern applications and systems. This could involve analyzing real-world workload traces or generating synthetic workloads that capture important statistical properties.
- *Performance Metrics:* Evaluating the algorithm's performance using a variety of relevant metrics, including throughput, turnaround time, waiting time, responsiveness, and fairness.
- *Comparative Studies:* Comparing the performance of the hybrid algorithm with traditional scheduling algorithms under different workload scenarios.

Investigating Machine Learning for Adaptation

Machine learning offers powerful tools for learning and adapting to complex environments. Future research should explore the use of machine learning techniques to enhance the hybrid scheduling algorithm:

- *Learning Utility Functions:* Machine learning algorithms can learn the optimal parameters for the utility functions based on observed process behavior and system performance.
- *Adaptive Game Parameters:* Machine learning can be used to dynamically adjust the parameters of the game-solving algorithm or the frequency with which the game theory module is invoked.
- *Predictive Scheduling:* Machine learning models can be used to predict future workload characteristics and proactively adjust the scheduling policies to optimize performance.

CONCLUSION

This study has presented a hybrid processor scheduling algorithm that leverages the power of game theory to enhance traditional scheduling methods. By treating processes as strategic players, the algorithm dynamically adapts to changing workload characteristics, optimizing for fairness, efficiency, and responsiveness. The integration of a base scheduler with a game theory module allows for a balanced approach, combining the established strengths of traditional methods with the adaptability of game-theoretic strategies. Different game variants, such as non-cooperative, cooperative, and evolutionary games, offer flexibility in addressing diverse system requirements and process interactions. While challenges remain, including the computational cost of game solving, the design of effective utility functions, and the complexities of parameter tuning, the potential benefits of this hybrid approach are significant. Future studies will aim to tackle these challenges by developing efficient approximation algorithms and exploring how machine learning can be leveraged to automate the adjustment of utility functions and game parameters. Ultimately, this hybrid algorithm represents a promising step towards more intelligent and adaptive processor scheduling in modern operating systems, paving the way for improved resource utilization and enhanced user experience.

REFERENCES

1. Dirdal DO, Vo D, Feng Y, Davidrajuh R. Developing a Platform Using Petri Nets and GPenSIM for Simulation of Multiprocessor Scheduling Algorithms. Appl Sci. 2024 Jun 29; 14(13): 5690.
2. Papp PA, Anegg G, Karanasiou A, Yzelman AJ. Efficient Multi-Processor Scheduling in Increasingly Realistic Models. In Proceedings of the 36th ACM Symposium on Parallelism in Algorithms and Architectures. 2024 Jun 17; 463–474.
3. Acharya B, Panda S, Sivakumar E. An analytical study of multiprocessor scheduling using metaheuristic approach. SN Comput Sci. 2022 Sep 29; 3(6): 497.
4. Lima G, Massa E, Regnier P. Practical considerations in optimal multiprocessor scheduling. In Handbook of real-time computing. Singapore: Springer Nature Singapore; 2022 Aug 9; 193–231.
5. Liu X, Li W. Approximation algorithms for the multiprocessor scheduling with submodular penalties. Optim Lett. 2021 Sep; 15: 2165–80.

6. González-Rodríguez M, Otero-Cerdeira L, González-Rufino E, Rodríguez-Martínez FJ. Study and evaluation of CPU scheduling algorithms. Heliyon. 2024 May 15; 10(9): e29959.
7. Omar HK, Jihad KH, Hussein SF. Comparative analysis of the essential CPU scheduling algorithms. Bull Electr Eng Inform. 2021 Oct 1; 10(5): 2742–50.
8. Gandhi N, Yadav M, Senthil A. Multiprocessor Scheduling Algorithm based on 2D Cellular Automata. International Conference on Innovative Science & Engineering Technology (ICISET-11). 2011.
9. Ali SM, Alshahrani RF, Hadadi AH, Alghamdi TA, Almuhsin FH, El-Sharawy EE. A review on the cpu scheduling algorithms: Comparative study. International Journal of Computer Science & Network Security (IJCSNS). 2021; 21(1): 19–26.
10. Chen J, Wang Y, Zhang Y, Lu Y, Li Q, Shu Q. Non-Preemptive Scheduling of Periodic Tasks with Data Dependencies in Heterogeneous Multiprocessor Embedded Systems. ACM Trans Des Autom Electron Syst. 2025 Feb 7; 30(2): 1–25.
11. Tian S, Ren W, Deng Q, Zou S, Li Y. A predictive energy consumption scheduling algorithm for multiprocessor heterogeneous system. IEEE Trans Green Commun Netw. 2021 Nov 30; 6(2): 979–91.
12. Gong M, Goebel R, Lin G, Miyano E. Improved approximation algorithms for non-preemptive multiprocessor scheduling with testing. J Comb Optim. 2022 Aug; 44(1): 877–93.
13. Gong M, Lin G. Improved approximation algorithms for multiprocessor scheduling with testing. In International Workshop on Frontiers in Algorithmics. Cham: Springer International Publishing; 2021 Aug 16; 65–77.
14. Sotskov YN, Mihova EI. Scheduling multiprocessor tasks with equal processing times as a mixed graph coloring problem. Algorithms. 2021 Aug; 14(8): 246.
15. Sudvarg M, Gill C, Baruah S. Improved implicit-deadline elastic scheduling. In 2024 IEEE 14th International Symposium on Industrial Embedded Systems (SIES). 2024 Oct 23; 50–57.
16. Acharya B, Panda S, Ray NK. Multiprocessor task scheduling optimization for cyber-physical system using an improved salp swarm optimization algorithm. SN Comput Sci. 2024 Jan 10; 5(1): 184.
17. Zhao Q, Qu M, Gu Z, Zeng H. Minimizing stack memory for partitioned mixed-criticality scheduling on multiprocessor platforms. ACM Trans Embed Comput Syst. 2022 Mar 4; 21(2): 1–30.
18. Agarwal G, Gupta S, Ahuja R, Rai AK. Multiprocessor task scheduling using multi-objective hybrid genetic Algorithm in Fog–cloud computing. Knowl-Based Syst. 2023 Jul 19; 272: 110563.

