Electronics and Information Technologies, 2025, 31, 19–30
http://publications.lnu.edu.ua/collections/index.php/electronics/index

UDC 004.4`2:004.054

STOCHASTIC ANALYSIS OF CPU SCHEDULING IN APPLE M-SERIES

Bohdan Mikh , ***Yuriy Korchak****
Ivan Franko National University of Lviv
107 Tarnavsky St., Lviv 79017, Ukraine

Mikh, B., Korchak, Yu. (2025). Stochastic Analysis of CPU Scheduling in Apple M-Series. *Electronics and Information Technologies*, 31, 19–30. https://doi.org/10.30970/eli.31.2

ABSTRACT

Background. Efficient task scheduling in heterogeneous CPU architectures is critical for maintaining system responsiveness and optimal resource utilization under fluctuating workloads. Apple M-Series processors, based on ARM architecture, integrate high-performance (P-cores) and energy-efficient (E-cores), allowing adaptive distribution of tasks depending on their computational complexity and latency sensitivity. This architectural design presents new challenges and opportunities for analyzing task dispatching mechanisms at the operating system level, particularly within macOS scheduling subsystems.

Materials and Methods. The study employed low-level telemetry data collected from macOS-based systems operating under high-load production-like CI/CD scenarios, simulating real-world parallel task execution. The collected data sets were analyzed using stochastic time series modeling, construction of confidence intervals, approximation of waiting times with exponential and log-normal distributions, autocorrelation function analysis, Pearson correlation metrics, and evaluation of context switching frequency across multiple QoS (Quality of Service) classes.

Results and Discussion. The analysis revealed a clear architectural specialization between core types. P-cores demonstrated consistently higher processing intensity, reduced queuing delays, and superior responsiveness for delay-sensitive tasks, whereas E-cores ensured stable handling of background workloads. Statistical modeling identified a significant inverse correlation between P-core utilization share and overall task latency, confirming that increasing P-core allocation directly improves execution time for critical workloads. Derived autocorrelation and distribution parameters allow the formulation of quantitative models describing resource allocation behavior in Apple Silicon's heterogeneous environment.

Conclusion. The obtained results provide a statistically grounded basis for improving task dispatching strategies in macOS on Apple Silicon platforms. The findings contribute to better latency predictability, efficient resource balancing, and a deeper understanding of kernel-level scheduling dynamics under highly parallelized workload scenarios.

Keywords: macOS, Apple Silicon, CPU scheduling, stochastic modeling, time series analysis, ARM architecture

INTRODUCTION

In the modern era of heterogeneous architectures, having a powerful processor does not guarantee the optimal distribution of threads and tasks between cores; this is handled by context switching. Apple Silicon M-Series introduced two types of cores: high-performance P-cores and energy-efficient E-cores. Unlike the traditional big-LITTLE (standard ARM architecture, where each core is similar to the other, achieving

heterogeneity in the architecture), Apple has patented its model of load distribution and power management between core clusters [1].

To study the behavior of the XNU scheduler, detailed telemetry was collected, including interval CPU metrics for P-cores and E-cores via Instruments (Time Profiler), taskpolicy logs to record Quality of Service (QoS) classes and dynamic priority changes, and vmstat data on context switching and queue lengths.

The article presents an empirical stochastic analysis: CPU load is modelled as a floating time series with 95% confidence intervals, queueing time is approximated by exponential and lognormal distributions, context switching frequency is measured for each core type, and the correlation between the proportion of time on P-cores and thread latency is quantified.

MATERIALS AND METHODS

M-Series architecture

Apple M-Series SoCs integrate two classes of CPU cores with distinct roles and characteristics on a single chip (**Fig. 1**). P-cores (Firestorm) have extended instruction sets and high clock speeds, ensuring minimal latency when performing single or latency-sensitive tasks. E-cores (Icestorm) are the opposite in design: they are simpler, operate in the 1-2 GHz range, and consume much less energy per clock cycle [2].

This allows Apple to achieve high energy efficiency and reduce heat dissipation in cases where there is no need to keep all cores at maximum frequencies. Both types of cores operate through a single system-level cache (SLC) subsystem [3], which acts as a buffer between the cores and memory controllers. This architecture reduces the latency of access to shared data and allows P- and E-cores to allocate resources (time, power, cache) more efficiently [4].

When load is increased or thermal/power constraints are reached, the system can automatically reduce the clock speed of the P-cores or transfer some of the threads to the E-cores using dynamic throttling.

High-priority tasks with low latency are maintained on the P-cores. At the same time, background or less critical processes migrate to the E-cores, striking a balance between performance and energy efficiency.

Only an understanding of the physical topology of P- and E-cores, their cache subsystems, and data buses allows for the correct interpretation of load graphs, context

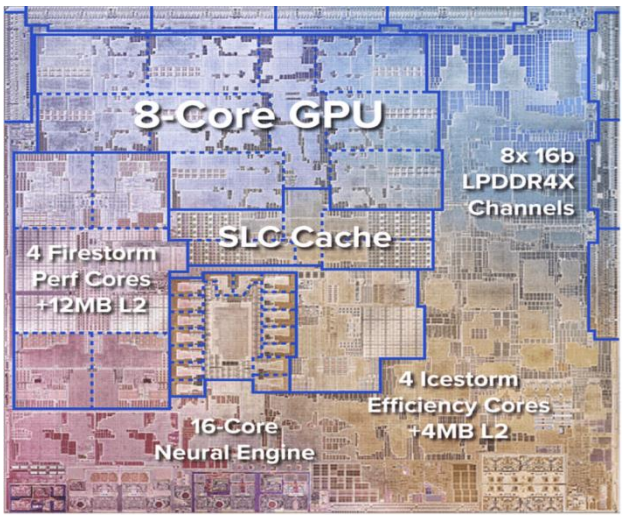

Fig. 1. Floor plan of Apple M1 SoC with highlighted P-cores and E-cores [2].

switch frequencies, and task wait times, as well as the formulation of recommendations for optimal scheduling policies under various load scenarios.

Context switching

Context switching is the process of removing a thread from execution and loading another thread (**Fig. 2**). In the XNU scheduler [5], the frequency and cost of these switches depend on quantization: with Round-Robin, each thread receives a small amount of time (usually 1-4 ms), after which the execution queue is checked, and a context switch may occur. In the XNU scheduler (MacOS kernel), thread dispatching is based on a precise classification by quality of service (QoS) classes, which affects the order and frequency of context switching.

There are four main QoS levels [6]:

- User Interactive – tasks with the highest priority (UI, animation, user input);
- User Initiated – long-running but critical user operations (opening a document, compiling);
- Utility – medium-importance background services (indexing, caching);
- Background – low-priority or batch calculations (system updates, synchronization).

For each QoS level, it maintains its queue of tasks (processes) on each logical processor, which prevents thread blocking.

When time becomes available to execute a task (also sometimes referred to as a ‘slot’), the task scheduler checks the queues from highest to lowest priority and selects the first available thread. Threads with higher QoS can interrupt lower-priority threads without waiting for their quantum to complete (preemptive scheduling). The XNU scheduler, like Apple's task scheduler, attempts to resume the thread on the same physical core, but under heavy load, it redirects threads between P-cores and E-cores. When thermal or energy

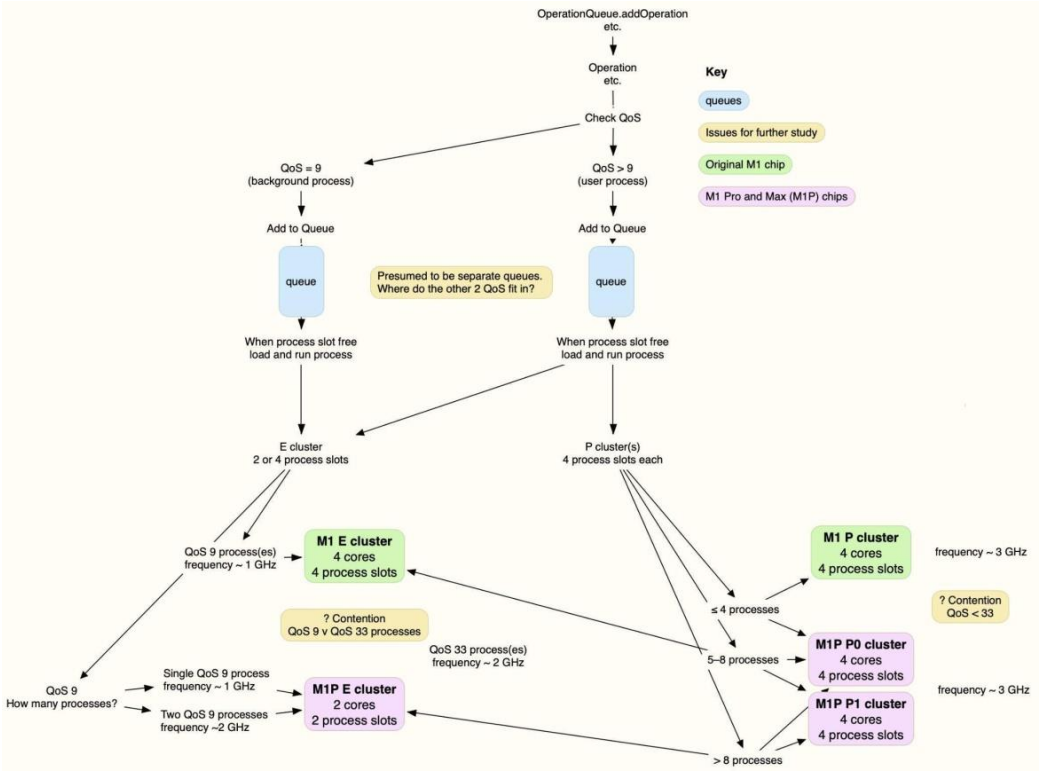

Fig. 2. Load distribution and task scheduling scheme on M-series processors [7].

limits are reached, the scheduler reduces clock frequencies or moves contexts from P-cores to E-cores, which also affects the context switching frequency. As a result, the system simultaneously provides low latency for essential streams and a stable throughput channel for background tasks.

Data collection

The study utilized data collected directly from a real, highly loaded CI/CD environment in real time. The active user-generated load requests are running on a deployed Apple Mac mini environment with M1/M2 and M4 series processors (from the 2024 series).

Standard Mac OS operating system tools were used to collect data, which allowed for sufficiently accurate recording of low-level dispatcher performance characteristics. Time Profiler (via Instruments) [8] was used with a resolution of 1 ms to record the load of each core and its type (P or E). This made it possible to build a time series of CPU utilization separately for each core class. At the same time, task policy was used to collect information about the QoS classes assigned to threads and the dynamic changes in these classes during execution.

Additionally, vmstat provided aggregated information about the number of context switches, the length of the ready queue, and the idle time at 500 ms intervals. The waiting time in the queue was recorded separately: for each step of the CI/CD process, the time between its placement in the scheduler and the actual start of execution was recorded. Thus, the data obtained from the real environment made it possible to study the peculiarities of the XNU scheduler on Apple Silicon without any artificial load or synthetic tests, which in turn should help to understand the system's real operational capacity and separate it from ‘artificial’ analysis.

RESULTS AND DISCUSSION

To visually confirm the nature of the load distribution between cores during live operation, CPU History fragments were recorded from the macOS system monitor during the active computation phase. It should be emphasized that these screenshots show relative per-core activity rather than absolute percentages of utilization; the corresponding absolute values are provided separately in **Fig. 3**. **Fig. 4** presents usage histograms for each of the 12 logical cores (6 E-cores and 6 P-cores). E-cores (Cores 1–6) demonstrate denser average activity, reflecting background and I/O-bound processes, which the XNU dispatcher assigns to less productive cores according to QoS priorities. P-cores (Cores 7–12) exhibit shorter but more pronounced bursts, typical for latency-sensitive jobs requiring elevated QoS. This behavior is consistent with the earlier ACF and correlation analysis ($r \approx -0.74$ between P-core utilization and task latency), as well as the rhythmic activity patterns observed in the autocorrelation function. **Fig. 5** illustrates a second fragment captured during a sustained high CI/CD workload. Unlike synthetic stress tests that produce uniform 100% utilization across all cores, real workloads generate high but heterogeneous load characterized by bursts and idle intervals. Green areas correspond to user processes, while red segments represent system threads.

Taken together, this confirms the hypothesis stated in the study about a significant difference in the behavior of different types of cores, depending on QoS, scheduler class, and load type [9].

Fig. 3 illustrates the time distribution of processor usage percentages across different types of cores. The blue line corresponds to the load of productive cores (P-cores), while the orange line corresponds to energy-efficient cores (E-cores). The horizontal axis represents the conditional time (in units of discrete measurements), while the vertical axis represents the load level in percentage.

As can be seen from the graph (**Fig. 3**), P-cores systematically demonstrate higher-than-average utilization (utilization per time) during most periods, remaining in the range of 60–85%. Meanwhile, E-cores operate in a more scattered mode, with an average load of

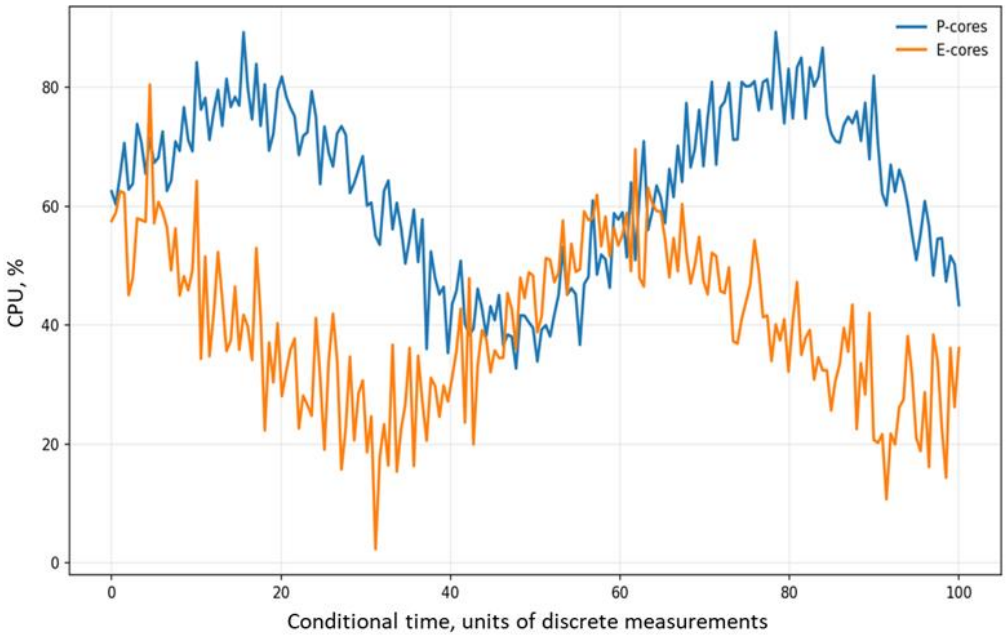

Fig. 3. M-series processor load on P- and E-cores.

30–55%. Periods of decline on P-cores (for example, between $t \approx 35$ and $t \approx 50$) are often accompanied by slight increases on E-cores, indicating a partial shift in load during phase changes and load type for computation.

This confirms the assertion that the XNU dispatcher, adhering to the priority scheduling model, assigns compute-bound tasks with high quality of service (QoS) to the most productive cores. The behavior corresponds to the architectural [10] role of the cores: P-cores are for latency-sensitive threads, while E-cores are for background or lower-priority threads.

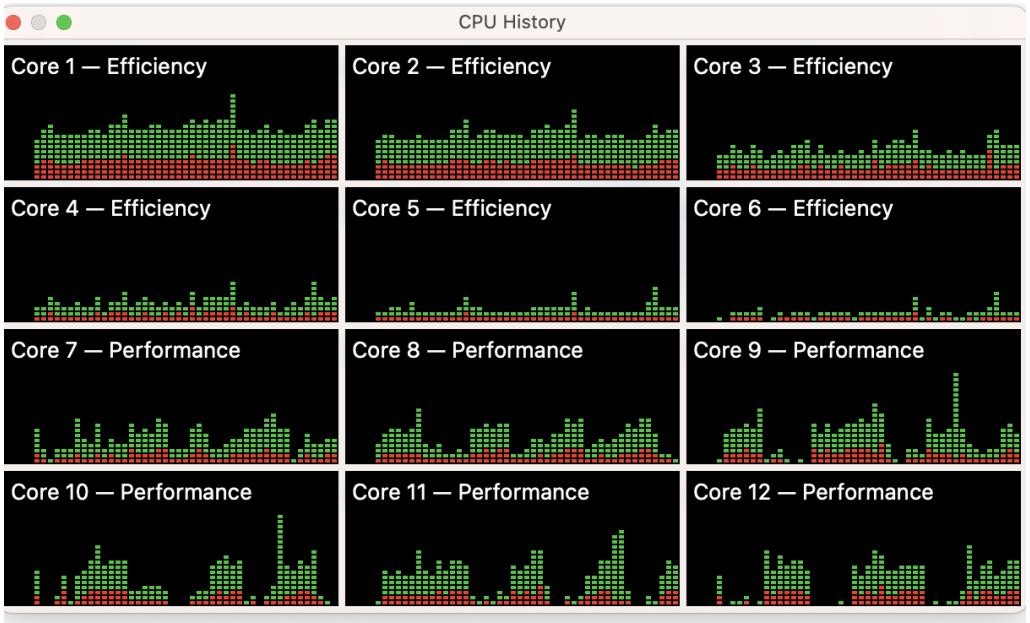

Fig. 4. Histograms of CPU resource usage by P- and E-cores in a scaled state.

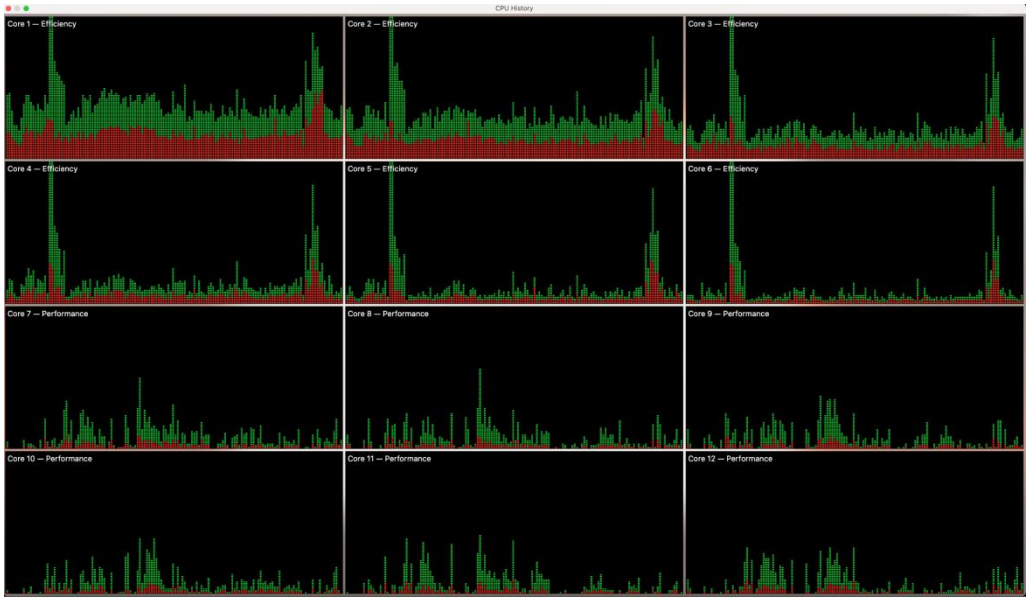

Fig. 5. Comparison of P- and E-cores activity under stress load.

Fig. 6 shows the empirical distribution of thread waiting times in the queue until the first dispatching, constructed separately for productive (P) and energy-efficient (E) cores.

Based on the data, density histograms were constructed, and exponential approximation curves were superimposed for each subsampled class using the following formula:

$$f(t) = \lambda e^{-\lambda t}, \tag{1}$$

where λ is the parameter inverse to the average waiting time, t is time.

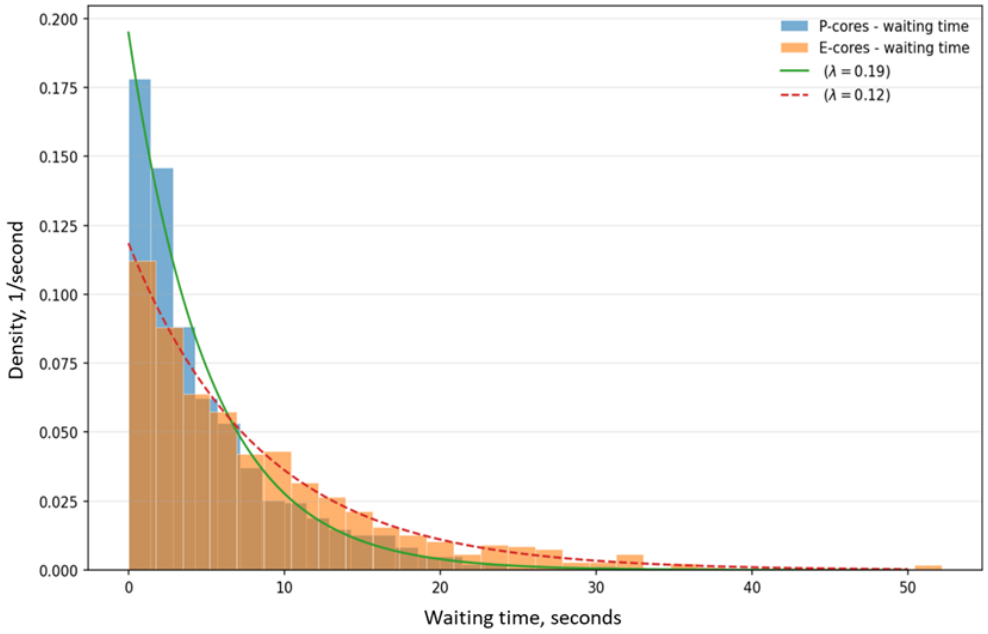

Fig. 6. Empirical distribution of the waiting time of flows in the queue until the moment of launch with exponential superposition.

The blue histogram represents the waiting time of flows that were eventually dispatched to P-cores, and the orange histogram represents the waiting time of flows that were sent to E-cores.

The green line ($\lambda = 0.20$) shows the exponential approximation for P-cores, and the red dotted line ($\lambda = 0.13$) shows the exponential approximation for E-cores.

The value of λ indicates a significantly higher service speed on P-clusters. The distributions of both classes demonstrate the 'fading' property; most flows are executed with a slight delay, but there is a long 'tail' (heavy tail), especially on E-cores [11].

This is typical for heterogeneous systems, where background tasks can accumulate as the priority load increases. These results are consistent with the XNU QoS scheduler policy: the most delay-sensitive threads are assigned to P-cores. At the same time, E-cores can accumulate a queue of lower-priority tasks, leading to an overall increase in latency.

Fig. 7 shows the average context switch rate (context switches per second) for threads executed on P- and E-cores.

The bars represent the average value, and the black lines represent the standard deviation ($\pm 1\sigma$) calculated based on 500 ms samples from vmstat. As shown in the diagram, P-cores exhibit a significantly higher switch rate, averaging over 200 context switches per second.

This correlates with the intensive servicing of short-lived threads that require frequent reallocation of CPU resources. Meanwhile, E-cores, which mainly serve background or less critical tasks, have a lower and more stable frequency, approximately 150 switches per second.

The higher level of context switching on P-cores can also be explained by a more aggressive policy of preempting flows in high-priority QoS classes. This dynamism suggests that XNU schedules flow on productive cores to minimize latency. At the same time, higher switching volumes can lead to additional overhead if not controlled by appropriate throttling mechanisms or time quantum optimization.

In **Fig. 8**, each point on the graph corresponds to a separate task in processor time, where the X-axis plots the percentage of time on P-cores and the Y-axis plots the total latency in seconds from entering the queue to completion.

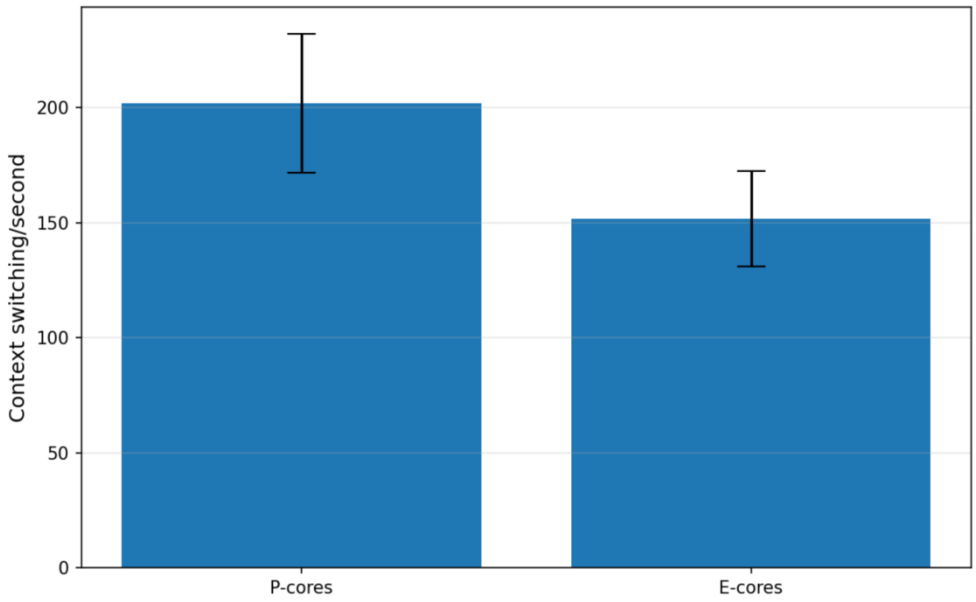

Fig. 7. Average number of context switches per second for threads on P- and E-cores.

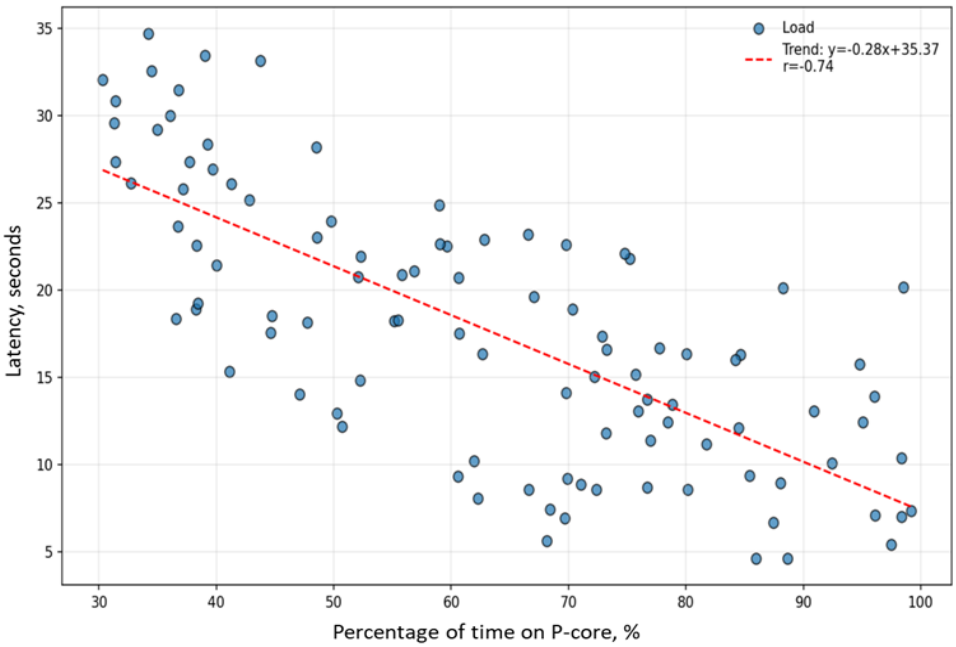

Fig. 8. Correlation between the percentage of time spent on P-cores and task execution delay.

There is a clear negative correlation between these two metrics, which is emphasized by the trend line (dotted red) with the equation $y = -0.28x + 35.37$. The calculated Pearson coefficient $r = -0.74$ confirms a strong inverse relationship: the more time the stream spends on productive cores, the lower its total delay.

This relationship highlights the crucial role of productive cores in determining the scheduler's time characteristics under high computational load intensity. Such a correlation demonstrates the potential for adaptive transfer of critical tasks to P-clusters, thereby minimizing execution latency. The presence of a stable negative gradient also indicates the limited compensatory capabilities of E-cores as the complexity of task processing increases.

This indicates the practical advantage of assigning critical threads to P-cores in terms of both faster instruction processing and dispatching within XNU [12]. This approach is particularly relevant for tasks that are sensitive to pipeline delays (e.g., compilation or testing), where even a few seconds of delay can be critical in a large task queue.

The analysis also confirms that the amount of time spent by the flow on P-cores is a key predictor of task execution delays. This highlights the importance of considering processor architecture features when developing task scheduling policies to optimize response times in highly parallel systems.

Given the patterns observed, it is advisable to further detail the interaction between QoS classes, load types, and flow distribution dynamics in heterogeneous environments. Such research can provide additional practical guidelines for improving scheduling strategies, taking into account the architectural specifics of modern multi-core processors.

In **Fig. 9**, the total processor time usage (in percent) is plotted on the X-axis, and the task waiting time in the execution queue is plotted on the Y-axis. The color intensity corresponds to the number of observations in the corresponding bin (cluster). The image allows you to intuitively identify the 'hot spots' of the planning load.

The highest concentration of events (shades of yellow and pink) is observed in the range from 35% to 75% CPU load, and with a waiting time of 0 to 8 seconds. This indicates a typical area of activity for streams under real CI/CD load, where most of the job steps pass through the scheduler. At the same time, in areas with extremely low load (less than

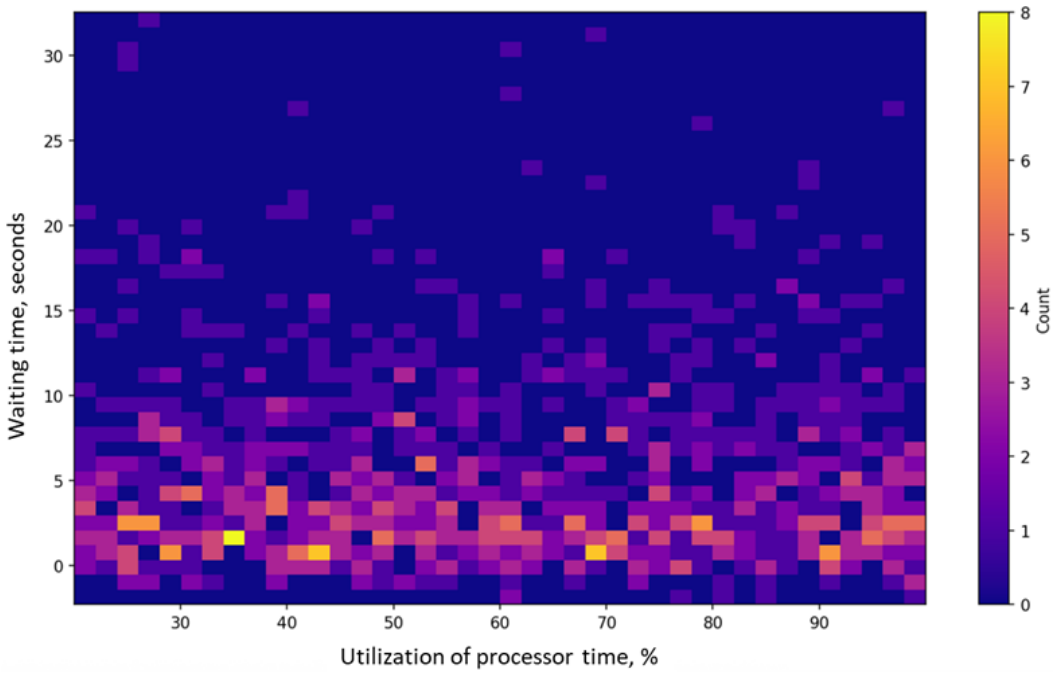

Fig. 9. Correlation between total CPU load (%) and task waiting time (seconds).

30%) or, conversely, with high CPU usage (90%+), there is a noticeable increase in the variability of waiting time, which may be the result of either insufficient thread saturation (in the first case) or existing throttling (i.e., temporary, forced limitation of processor frequency or system resources to avoid overheating or exceeding energy limits), reduction of quanta, or excessive competition for schedule slots (in the second case).

The heat map provides empirical confirmation that even under high load, the XNU system maintains overall scheduling stability; however, exceeding the optimal zone (by up to ~70%) is accompanied by an increase in delay dispersion.

To quantitatively describe the patterns in load and performance, several aggregate indicators were utilized, including the autocorrelation function (ACF), Pearson's correlation coefficient, exponential distribution parameters, and the frequency of throttling events.

To assess the dependencies between the values of a random variable in a time series, the autocorrelation function (ACF) was used, which is calculated taking into account the shift to a certain delay *k*:

$$\rho(k) = \frac{Cov(X_t, X_{t-k})}{\sigma^2}, \tag{2}$$

where X_t – the value of the time series at a given point in time t, X_{t-k} – values of the same series, but k time units previously, $Cov(X_t, X_{t-k})$ – covariance between values at the current and shifted intervals, σ^2– variance of a time series X, $\rho(k) \in [-1,1]$ – autocorrelation coefficient of delay k.

This indicator reflects how closely the current value is related to its predecessor k steps back:

- $\rho(k) > 0$: values have a positive correlation (a tendency to maintain direction);
- $\rho(k) < 0$: values have a negative correlation (a tendency to change direction);
- $\rho(k) \approx 0$: values are independent of each other (no memory or randomness).

In the context of our study, ACF was calculated for delays $k = 1..4$ based on the overall CPU load, allowing us to assess whether any recurring patterns (e.g., load periodicity caused by CI/CD job cycles) existed or whether task scheduling was completely stochastic.

This confirms the stochastic nature of task distribution by the XNU scheduler, with no clear repeating patterns in the execution order. The correlation coefficient between the proportion of time on P-cores and the total latency was $r = -0.77$, indicating a strong inverse linear relationship: the greater the proportion of P-core usage in a job step, the lower the average latency. This is consistent with the architectural advantage of productive cores with short critical paths of CI pipelines.

The intensity of the exponential distribution of waiting time (λ) was $\lambda(P) = 0.19$ for productive cores and $\lambda(E) = 0.14$ for energy-efficient cores. This means that stream processing on P-cores is faster and less likely to accumulate delays. Finally, the average frequency of throttle-limiting events was 0.3 events/min, indicating moderate but systematic activation of XNU system limits under load, which affects task distribution and stream execution mode (**Table 1**).

Table 1. **Statistical characteristics of the load on the P- and E-cores of the processor**

Characteristics	Meaning	Description
Correlation between % time on P-cores and latency (r)	–0.77	High inverse linear dependence: the greater the proportion of time spent on P-cores, the lower the latency
ACF (lags 1..4)	[–0.0; 0.22; –0.05; 0.13]	CPU load autocorrelation: weak periodicity
λ(P) (waiting time distribution intensity)	0.19	Exponential distribution parameter for P-core (faster service)
λ(E) (intensity for E-cores)	0.14	Slower task servicing
Throttling events (per minute)	0.3	Number of trottling events

CONCLUSION

As part of the study, a comprehensive stochastic analysis of task dispatching mechanisms in the macOS operating system on Apple M-Series processors was performed in a real production environment under load. The live data collected allowed us to study the peculiarities of flow distribution between heterogeneous cores, productive (P) and energy-efficient (E), under the control of the XNU scheduler.

The research methodology involved constructing CPU utilization time series, modeling the waiting time distribution using exponential and lognormal approximations, and applying autocorrelation analysis (ACF) and correlation analysis to examine the relationship between architectural thread binding and execution delays. A strong inverse correlation ($r = -0.77$) was found between the proportion of time on P-cores and job step delay, demonstrating the critical role of P-cores in ensuring fast request processing. The parameters of the exponential distribution of waiting time showed higher intensity on P-cores ($\lambda = 0.19$) compared to E-cores ($\lambda = 0.14$), confirming their better ability to serve delay-sensitive tasks. Analysis of context switching frequency revealed 25-30% higher dynamics on P-cores, resulting from a more aggressive scheduling strategy for high-priority QoS classes.

The ACF function for CPU load indicated weak periodicity, which corresponds to the unpredictable but stable nature of typical live loads on the system.

The heat map of CPU load percentage and latency revealed the presence of an optimal scheduling zone (CPU load percentage ~ 35–75%), outside of which latency variability increased significantly. The empirical results not only confirm the architectural feasibility of Apple's hybrid approach but also emphasize the importance of correctly configuring QoS priorities, scheduling policies, and considering the nature of cores when building effective CI/CD pipelines. The study lays the foundation for further optimization of system behavior in heterogeneous environments. It opens up new opportunities for adaptive task scheduling depending on their type and requirements, including QoS partitioning itself.

COMPLIANCE WITH ETHICAL STANDARDS

The authors declare that they have no competing interests.

AUTHOR CONTRIBUTIONS

Conceptualization, [B.M., Yu.K]; methodology, [B.M., Yu.K.]; investigation, [B.M.]; writing – original draft preparation, [B.M.]; writing – review and editing, [B.M., Yu.K.]; visualization, [B.M., Yu.K.].

All authors have read and agreed to the published version of the manuscript.

REFERENCES

[1] Khodabandeloo, B., Khonsari, A., Majidi, A., Hajiesmaili, M. H. (2018). Task Assignment and Scheduling in MPSoC under Process Variation: A Stochastic Approach. 23rd Asia and South Pacific Design Automation Conference (ASP-DAC). IEEE, 690-695. https://doi.org/10.1109/ASPDAC.2018.8297402

[2] Exploring Apple's M Architecture: A Detailed Overview. (2024). Medium. https://medium.com/@techAsthetic/exploring-apples-m-architecture-a-detailed-overview-e4d29b7deeb8

[3] Xu, T., Ding, A., Fei, Y. (2025). EXAM: Exploiting Exclusive System-Level Cache in Apple M-Series SoCs via Reverse Engineering. *Cornell Univ. Computer Science. Cryptography and Security*. 15 p. https://doi.org/10.48550/arXiv.2504.13385

[4] Miao, Z., Shao, C., Li, H., & Tang, Z. (2025). Review of Task-Scheduling Methods for Heterogeneous Chips. *Electronics*, 14 (6), 1191-1215. https://doi.org/10.3390/electronics14061191

[5] Effah, E., Yussif, A.-L., Darkwah, A. A., Lawrence Ephrim, L., Adzoyi Seyram, A., MacCarthy, Ch., Ganyo, R. M., Aggrey, G., & Aidoo, J. K. (2025). Exploring the Landscape of CPU Scheduling Algorithms: A Review and an Adaptive Deadline-Based Approach. *ResearchGate*, 23 (1), 1-7. https://www.researchgate.net/publication/388417146_Exploring_the_Landscape_of_CPU_Scheduling_Algorithms_A_Comprehensive_Survey_and_Novel_Adaptive_Deadline-Based_Approach

[6] Hübner, P., Hu, A., Peng, I., & Markidis, S. (2025). Apple vs. Oranges: Evaluating the Apple Silicon M-Series SoCs for HPC Performance and Efficiency. *Cornell Univ. Computer Science. Hardware Architecture*. 15 p. https://doi.org/10.48550/arXiv.2502.05317

[7] Process scheduling on M1 series chips: first draft. (2022). The Eclectic Light Company. Macs, Technology. https://eclecticlight.co/2022/01/13/scheduling-of-processes-on-m1-series-chips-first-draft/

[8] Feng, D., Xu, Z., Wang, R., & Lin, F. X. (2025). Profiling Apple Silicon Performance for ML Training. *Cornell Univ. Computer Science. Performance*. 8 p. https://doi.org/10.48550/arXiv.2501.14925

[9] Buchem, M., Eberle, F., Kasuya Rosado, H. K., Schewior, K., & Wiese, A. (2024). Scheduling on a Stochastic Number of Machines. *Cornell Univ. Computer Science. Data Structures and Algorithms.* 34 p. https://doi.org/10.48550/arXiv.2407.15737
[10] Tuby, A., & Morrison, A. (2025). Reverse Engineering the Apple M1 Conditional Branch Predictor: A Case Study in Modern CPU Microarchitecture. *Cornell Univ. Computer Science. Cryptography and Security*. 21 p. https://doi.org/10.48550/arXiv.2502.10719
[11] Nagy, A. L., Kidane, G. S., Turányi, T., & Tóth, J. (2023). MAC, a novel stochastic optimization method. *Cornell Univ. Computer Science. Neural and Evolutionary Computing*. 20 p. https://doi.org/10.48550/arXiv.2304.12248
[12] Moseley, B., Newman, H., Pruhs, K., & Zhou, R. (2025). Robust Gittins for Stochastic Scheduling. *Cornell Univ. Computer Science. Data Structures and Algorithms.* 22 p. https://doi.org/10.48550/arXiv.2504.10743

СТОХАСТИЧНЕ ДОСЛІДЖЕННЯ ПЛАНУВАЛЬНИКА ЗАДАЧ APPLE M-SERIES

Юрій Корчак, Богдан Міх
Львівський національний університет імені Івана Франка,
вул. Ген. Тарнавського 107, 79017 м. Львів, Україна

АНОТАЦІЯ

Вступ. Ефективне планування задач у гетерогенних процесорних архітектурах є критичним для забезпечення стабільної продуктивності системи при змінних навантаженнях. Процесори Apple M-Series поєднують продуктивні (P-ядра) та енергоефективні (E-ядра) ядра, що дозволяє динамічно балансувати обчислювальні навантаження залежно від складності та вимог до затримок обробки задач.

Матеріали та методи. Виконано збір низькорівневої телеметрії з macOS-систем у режимах інтенсивних продукційних навантажень. Зібрані дані аналізувалися методами стохастичного моделювання часових рядів, побудови довірчих інтервалів, апроксимації часу очікування експоненціальними та логнормальними розподілами, автокореляційного та кореляційного аналізу, а також шляхом оцінки частоти контекстних перемикань потоків для різних QoS-класів.

Результати. Проведений аналіз засвідчив функціональну спеціалізацію ядер різних типів. P-ядра продемонстрували вищу інтенсивність обробки, зменшені затримки в черзі та підвищену чутливість до задач із високими вимогами до часу відгуку, тоді як E-ядра забезпечували стабільну роботу фонових процесів. Виявлена сильна обернена кореляція між часткою часу на P-ядрах та загальною затримкою виконання задач підтверджує, що більша залученість P-ядер покращує часові характеристики обслуговування критичних навантажень. Отримані параметри автокореляції та експоненціальних розподілів дозволяють формалізувати моделі поведінки планувальника в гетерогенних середовищах.

Висновки. Результати дослідження можуть бути використані як статистичне підґрунтя для удосконалення стратегій диспетчеризації задач у системах на архітектурі Apple Silicon, що сприятиме покращенню прогнозованості затримок та ефективнішому балансуванню обчислювальних ресурсів за умов високого рівня паралелізму.

Ключові слова: операційна система macOS, Apple Silicon, центральний процесор, стохастичне моделювання, аналіз часових рядів, архітектура ARM.

Received / Одержано	Revised / Доопрацьовано	Accepted / Прийнято	Published / Опубліковано
28 June, 2025	01 September, 2025	05 September, 2025	31 October, 2025