DMPedia Lecture Notes in Multidisciplinary Research,
Volume 1, pp. 219-225, 2026

Beyond Inference: Exploratory Perspectives on Leveraging NPUs for Scheduling and Optimization

Mrigansh Chaudhary, Kartikay Srivastava, Utkarsh Pant
Department of Computer Science and Engineering, Graphic Era Deemed to Be University, Clement Town, Dehradun (Uttrakhand)-248002
mriganshchaudhary@tutamail.com, kartikayomram@gmail.com, utkarshpant2005@gmail.com

ABSTRACT

Neural Processing Units (NPUs) are becoming integral components of modern computing systems, primarily by accelerating inference workloads in areas such as computer vision, natural language processing, and recommendation systems. Despite this increasing prevalence, NPUs remain underutilized in many deployments, often wasting energy while sitting idle outside of inference contexts. At the same time, CPU scheduling continues to face challenges in balancing throughput, latency, efficiency, and energy consumption under highly variable workloads. Current scheduling paradigms primarily focus on balancing computational throughput and minimizing latency. For example, the Linux kernel recently adopted the Earliest Eligible Virtual Deadline First (EEVDF) scheduler to better address latency handling, building on the earlier Completely Fair Scheduler (CFS), which emphasized fairness. Yet, in real-world scenarios, workload variance is often high, and decisions such as prioritizing latency versus throughput versus efficiency are typically made statically, based on expected conditions rather than actual runtime demands. As a result, these scheduling parameters rarely adapt in real time to dynamic workloads and real-world usage patterns.
Together, these observations highlight an opportunity to explore the potential of NPUs beyond traditional inference acceleration. By leveraging their parallel processing capabilities and low power consumption, NPUs could assist in, or even take a more active role in, dynamic scheduling, enabling adaptive resource management that responds to real-time workload demands. This paper presents an exploratory framework for investigating the integration of NPUs into scheduling. Rather than reporting empirical results, it outlines conceptual pathways and architectural possibilities.

Keywords: Neural Processing Unit, Scheduling, Optimization, CPU–NPU Cooperation, Resource Management.

1. Introduction

Neural Processing Units (NPUs) are becoming increasingly prevalent in modern computing systems, primarily for accelerating inference workloads [1], [2]. Yet outside of these contexts, NPUs often remain underutilized [3]–[5]. While several studies have examined accelerator scheduling, little work has explored the cooperative use of NPUs alongside CPUs or investigated the potential of NPUs as active schedulers for CPU tasks [6], [7].
Recent surveys of accelerator-based heterogeneous architectures, involving CPUs and GPUs/TPUs/FPGAs, highlight that scheduling policies often assume static workloads or fixed accelerator affinity, which limits adaptability under real-time constraints[6].
Current scheduling paradigms mainly focus on balancing computational throughput and minimizing latency. For instance, the Linux kernel first introduced the Completely Fair Scheduler (CFS) to improve fairness, and more recently adopted the Earliest Eligible Virtual Deadline First (EEVDF) scheduler to enhance latency handling [8], [9]. However, such approaches typically rely on static assumptions, limiting their adaptability to dynamic and diverse workload demands.
This paper explores how NPUs could contribute beyond inference acceleration to dynamic scheduling and adaptive resource management. We propose three conceptual scenarios: (A) NPUs classifying workload patterns in real time, (B) NPUs computing optimized functions

faster than CPUs, and (C) a hybrid scheduler where the CPU supervises execution while the NPU accelerates specific scheduling computations. We outline prototyping options and evaluation criteria, providing an exploratory framework for leveraging NPUs in scheduling.

2. Background and Related Works

Schedulers have been studied extensively in the context of CPUs and GPUs, with works focusing on resource allocation and fairness[10], [11].Benchmarks and surveys of NPU platforms highlight a rapidly evolving design space, with substantial variation [11] in performance, power, and memory behavior across vendors[7]. These differences complicate portability and scheduling decisions in heterogeneous systems. Prior research on heterogeneous and accelerator-enabled scheduling has demonstrated that careful workload mapping and runtime policies are critical to meeting throughput, latency, and energy objectives. Studies have also proposed making accelerators preemptible to improve latency–throughput trade-offs [4], [7].
Despite these advances, the idea of NPUs acting as active participants in CPU scheduling remains underexplored. This motivates the exploration of NPUs as cooperative schedulers, extending their role beyond inference acceleration.

Research Methedology

- **NPU Variability Across Vendors**

NPUs are specialized accelerators, but their architecture, workload handling, and precision support differ significantly across vendors. Each company employs distinct software stacks and optimization strategies, which results in highly variable behavior even for the same computational task.[2] Due to this heterogeneity, direct cross-vendor comparisons are avoided in this study. Instead, the focus is placed on conceptual exploration. Schedulers are commonly evaluated based on properties such as fairness, efficiency, adaptability, and scalability [11]. Within this context, NPUs are regarded as a distinct class of devices, defined by shared attributes including high degrees of parallelism, energy efficiency, and workload specialization, which in turn influence the design and evaluation of their scheduling strategies.

- **NPU Emulation Environment**

To explore potential roles of NPUs in scheduling without relying on specific vendor hardware, emulation provides a flexible and controlled environment. Several frameworks exist, such as gem5 for system-level architectural exploration [12], mNPUsim for evaluating multi-core NPU performance under memory contention [13], and VPU-em for exploring accelerator scheduling in heterogeneous CPU–VPU platforms [14]. For this study, we select ONNXim [15], a fast, cycle-level multi-core NPU simulator, illustrates how simulation can balance modeling detail and performance. By enabling exploration of NPUs at varying performance levels, ONNXim offers insight into how differences in compute capacity and memory handling may influence scheduling integration. Such frameworks allow preliminary investigation without being constrained by proprietary hardware limitations.

- **Scheduler Prototyping with sched_ext**

The Linux kernel's sched_ext framework[16], introduced in version 6.7, allows custom schedulers to be written in eBPF and dynamically loaded at runtime. This capability makes it possible to experiment with non-traditional scheduling policies without altering kernel source code. In this study, sched_ext serves as the prototyping environment for describing how policies influenced by NPU-generated hints could be implemented. While it does not provide direct access to NPU hardware, it offers an effective vehicle for assessing how such hints might influence CPU scheduling decisions.

- **Co-simulation Framework**

Since neither ONNXim nor sched_ext provides a native interface for direct NPU–scheduler integration, a co-simulation approach is considered. In this setup, ONNXim functions as the NPU emulator, generating classification results or optimization outputs. These results would then be communicated to the scheduler prototype running under sched_ext, which applies corresponding task placement and prioritization strategies. Although this coupling introduces additional latency, it provides a practical and flexible pathway for exploring how NPU-assisted scheduling could impact dynamic workload conditions.

Scope and Future Evaluation

This paper is exploratory and does not include empirical experimentation. It outlines a conceptual framework for integrating NPUs into CPU scheduling, identifies potential scenarios, and highlights suitable test environments such as emulation (ONNXim) and scheduler prototyping (sched_ext).

Future evaluation could consider metrics such as scheduling latency, adaptability to workload variability, CPU utilization, and overall system efficiency. In addition to classification-based hints (Scenario A), metrics could capture the effectiveness of NPU-accelerated optimization functions (Scenario B), including improvements in task-to-core mapping, load balancing, and DVFS decisions. The hybrid scenario (Scenario C) could be evaluated for cooperative CPU–NPU decision-making, measuring how the integration of classification and optimization outputs affects system performance while maintaining fairness and thermal safety.

These studies, performed on emulated or real NPU hardware, would provide deeper insight into the feasibility, performance impact, and potential benefits of the proposed framework across a variety of dynamic workload conditions. Building on this methodological foundation, the next section outlines exploratory scenarios that illustrate how NPUs could be positioned within scheduling and algorithm optimization.

Exploratory Scenarios

This section presents three exploratory scenarios for integrating NPUs into scheduling and algorithm optimization. These scenarios are not prescriptive implementations but rather conceptual possibilities, intended to highlight potential roles NPUs may play beyond inference acceleration.

- **Scenario A: NPU-Assisted Real-Time Workload Classification**

In this scenario, the NPU functions as a real-time classifier, analyzing workload patterns and generating hints for the CPU scheduler. It takes input data from /proc/stat (CPU usage statistics), /proc/loadavg (system load averages), /proc/[pid]/stat (per-process state), and

/sys/class/power_supply/BAT0/power_now (instantaneous power draw), among other available system metrics, to determine workload characteristics [17], [18].

Based on this information, the NPU categorizes tasks (for example, latency-sensitive vs. throughput-oriented, interactive vs. background) and communicates these classifications as hints to the scheduler. The scheduler operates in adaptive intervals: if the NPU does not provide a classification within a given interval, the CPU scheduler proceeds without modification. When hints are available, scheduling policies can be adjusted dynamically, such as prioritizing throughput-heavy workloads or applying energy-saving strategies under high power draw.

This design ensures robustness by keeping the CPU scheduler as the default authority while the NPU provides guidance only when beneficial. The adaptive structure reduces overhead, mitigates risks from misclassification, and allows exploration of more responsive and energy-aware scheduling strategies.

- **Scenario B: NPU-Accelerated Optimization Functions**

In this scenario, the NPU is employed to directly compute optimization functions that guide scheduling policies. Unlike Scenario A, which focuses on classification and providing hints, the role of the NPU here is to execute computation-heavy optimization routines that would be impractical for the CPU to run continuously [19].

Examples of such optimizations include:

- Calculating optimal task-to-core mappings under multi-core or heterogeneous environments.
- Performing real-time load balancing by minimizing queue imbalance across CPU cores.
- Evaluating energy-performance trade-offs to determine dynamic voltage and frequency scaling (DVFS) settings.
- Estimating migration costs for moving processes between cores and computing the most efficient migration plan.

Because NPUs excel at parallel computation, they can evaluate multiple candidate policies or solve optimization problems more efficiently than a CPU. The results of these computations are then applied directly by the scheduler to adjust task placement, frequency scaling, or migration policies.

This scenario envisions the NPU as a mathematical optimizer integrated into the scheduling loop, capable of accelerating tasks that involve repeated, resource-intensive computation. Rather than producing hints, the NPU delivers actionable optimization results, enabling the scheduler to adapt more intelligently to dynamic workload conditions.

- **Scenario C: Hybrid CPU–NPU Cooperative Scheduling**

In this scenario, the CPU and NPU operate as cooperative entities, combining the strengths of both classification (Scenario A) and optimization (Scenario B). The NPU performs real-time workload classification while simultaneously executing optimization functions, such as task-to-core mapping, load balancing, and DVFS tuning. The CPU remains the supervisory authority, validating NPU outputs, enforcing system-wide policies, and handling global constraints such as fairness and thermal safety.

For example, the NPU could classify incoming tasks as latency-sensitive, throughput-oriented, or background, while also computing optimal task distributions across cores to minimize queue imbalance and reduce migration costs. The CPU scheduler then integrates these results, applying them selectively while ensuring that fairness policies and thermal limits are respected. This hybrid approach provides flexibility: classification enables adaptive awareness of workload patterns, while optimization supplies concrete scheduling parameters. The CPU ensures reliability and policy compliance, while the NPU accelerates the computationally expensive parts of decision-making. Scenario C thus envisions a system where NPUs act as active partners, supporting dynamic and energy-aware scheduling, without taking over critical global control responsibilities.

Result and Discussion

Although no empirical experiments are conducted in this study, conceptual analysis of the three proposed scenarios allows us to explore the potential impact of NPUs in scheduling and algorithm optimization. The discussion focuses on expected advantages, limitations, and trade-offs inherent to each scenario. Table 1 provides the summary of the Scenarios discussed.

Table 1: Overall Summary of the Scenarios

Scenario	**NPU Role**	**Output Type**	**CPU Supervision**	**Potential Benefits**	**Challenges**
A: NPU-Assisted Real-Time Classification	Classifies workloads using system metrics	Hints for scheduler	CPU retains full control	Adaptive scheduling, energy-awareness, low overhead	Dependence on classification accuracy; partial workload coverage
B: NPU-Accelerated Optimization Functions	Computes optimization routines (task-to-core mapping, load balancing, DVFS, migration costs)	Direct actionable optimization results	CPU applies results, validates global constraints	Faster decision-making, improved system efficiency	Integration overhead, latency sensitivity, NPU resource contention
C: Hybrid CPU–NPU Cooperative Scheduling	Combination of classification and optimization	Hints + optimization results	CPU supervises, enforces fairness and thermal limits	Combines responsiveness with optimized decision-making; supports energy- and performance-aware scheduling	Complexity in CPU–NPU coordination; risk of conflicting outputs; NPU utilization management

- **Expected Insights**

1. **Scenario A** demonstrates how NPUs can augment existing CPU schedulers with minimal integration effort. By providing classification hints, the scheduler can adapt dynamically to changing workloads, improving responsiveness without introducing significant computational overhead.

2. **Scenario B** highlights the potential of NPUs to accelerate resource-intensive computations that would otherwise burden CPUs. Optimization routines such as task-to-core mapping, load balancing, and DVFS tuning can be executed faster, enabling more frequent and fine-grained adjustments.
3. **Scenario C** illustrates a cooperative framework combining the benefits of both A and B. While the CPU supervises and enforces system-wide policies, the NPU actively contributes to classification and optimization. This hybrid approach can improve overall efficiency and adaptability but introduces complexity in coordination and validation.

▪ **Potential Limitations**

- NPU hardware heterogeneity may limit portability of strategies across vendors.
- Integration latency between the NPU and CPU may reduce real-time responsiveness in scenarios B and C.
- The effectiveness of classification and optimization depends on workload predictability and quality of NPU-generated outputs.
- Ensuring fairness and thermal safety in hybrid scheduling may require additional safeguards.

▪ **Conceptual Takeaways**

- NPUs have the potential to move beyond inference workloads, supporting dynamic and energy-aware scheduling.
- The scenarios illustrate a spectrum of integration strategies: from lightweight hint generation to full optimization acceleration.
- Hybrid approaches may offer maximal benefit but require careful coordination and validation to avoid adverse effects on system stability.

Conclusion

This paper presented an exploratory perspective on how Neural Processing Units (NPUs), traditionally deployed for accelerating inference workloads, might be repositioned as active participants in CPU scheduling and optimization. Motivated by the observation that NPUs often remain underutilized, we outlined three conceptual scenarios: NPU-assisted classification of workloads, NPU-accelerated optimization routines, and a hybrid cooperative model where CPUs and NPUs jointly contribute to scheduling decisions.
Through conceptual analysis, these scenarios highlight a spectrum of integration strategies, ranging from lightweight guidance of CPU schedulers to deeper involvement in optimization and cooperative decision-making. While Scenario A emphasizes responsiveness and energy awareness with minimal complexity, Scenario B illustrates the potential for NPUs to accelerate resource-intensive scheduling computations. Scenario C combines the advantages of both approaches but raises additional coordination and validation challenges.
The overall takeaway is that NPUs, with their parallelism and efficiency, may hold promise beyond their conventional role. However, realizing this potential will require addressing hardware heterogeneity, integration latency, and fairness guarantees. Future research should extend these concepts into practical validation, employing emulation and real hardware experiments to quantify gains in adaptability, efficiency, and system stability.

References

[1] Kang, D., Oh, J., Choi, J., Yi, Y., & Ha, S. (2020). Scheduling of deep learning applications onto heterogeneous processors in an embedded device. *IEEE Access*, *8*, 43980-43991..
[2] Millar, J., Huang, Y., Sethi, S., Haddadi, H., & Madhavapeddy, A. (2025, November). Benchmarking ultra-low-power μNPUs. In *Proceedings of the 31st Annual International Conference on Mobile Computing and Networking* (pp. 1060-1074).
[3] Feng, E., Feng, D., Du, D., Xia, Y., & Chen, H. (2024, June). snpu: Trusted execution environments on integrated npus. In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)* (pp. 708-723). IEEE.
[4] Xue, Y., Liu, Y., Nai, L., & Huang, J. (2023, June). V10: Hardware-assisted npu multi-tenancy for improved resource utilization and fairness. In *Proceedings of the 50th Annual International Symposium on Computer Architecture* (pp. 1-15).
[5] Xue, Y., Liu, Y., Nai, L., & Huang, J. (2023, June). V10: Hardware-assisted npu multi-tenancy for improved resource utilization and fairness. In *Proceedings of the 50th Annual International Symposium on Computer Architecture* (pp. 1-15).
[6] Zou, A., Xu, Y., Ni, Y., Chen, J., Ma, Y., Li, J., ... & Jin, Y. (2025). A Survey of Real-time Scheduling on Accelerator-based Heterogeneous Architecture for Time Critical Applications. *arXiv preprint arXiv:2505.11970*.
[7] Choi, Y., & Rhu, M. (2020, February). Prema: A predictive multi-task scheduling algorithm for preemptible neural processing units. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)* (pp. 220-233). IEEE.
[8] Rusling, D. A. (1999, January). *The linux kernel*.
[9] "CFS scheduler — The Linux kernel documentation," *Kernel.org*. [Online]. Available:
[10] Singh, R., Suyal, H., Shivhare, A., & Malviya, L. (2024, November). A Stochastic Hill Climbing Approach for Power Efficiency in Cloud-Based Systems. In *2024 International Conference on Cybernation and Computation (CYBERCOM)* (pp. 46-51). IEEE.
[11] González-Rodríguez, Miguel, Lorena Otero-Cerdeira, Encarnación González-Rufino, and Francisco Javier Rodríguez-Martínez. "Study and evaluation of CPU scheduling algorithms." *Heliyon* 10, no. 9 (2024).
[12Lowe-Power, J., Ahmad, A. M., Akram, A., Alian, M., Amslinger, R., Andreozzi, M., ... & Zulian, É. F. (2020). The gem5 simulator: Version 20.0+. *arXiv preprint arXiv:2007.03152*.
[13] Hwang, S., Lee, S., Kim, J., Kim, H., & Huh, J. (2023, October). mNPUsim: Evaluating the effect of sharing resources in multi-core NPUs. In *2023 IEEE International Symposium on Workload Characterization (IISWC)* (pp. 167-179). IEEE.
[14] Qi, C., Wang, Y., Wang, H., Lu, Y., Subramanian, S. S., Cahill, F., ... & Mathaikutty, D. (2023). VPU-EM: An event-based modeling framework to evaluate NPU performance and power efficiency at scale. *arXiv preprint arXiv:2303.10271*.
[15Ham, H., Yang, W., Shin, Y., Woo, O., Heo, G., Lee, S., ... & Kim, G. (2024). Onnxim: A fast, cycle-level multi-core npu simulator. *IEEE Computer Architecture Letters*.
[16] "Extensible scheduler class — The Linux kernel documentation (sched_ext)," *Kernel.org*, 2025. [Online].
Available:[https://www.kernel.org/doc/html/next/scheduler/schedext.html](https://www.kernel.org/doc/html/next/scheduler/sched-ext.html)
[17] "proc(5) Linux manual page," *man7.org*. [Online]. Available:https://man7.org/linux/man-pages/man5/proc.5.html
[18] "Linux power supply class — The Linux kernel documentation," *Kernel.org*, 2025. [Online]. Available:
https://www.kernel.org/doc/html/latest/power/power_supply_class.html
[19] Xu, D., Zhang, H., Yang, L., Liu, R., Huang, G., Xu, M., & Liu, X. (2025, March). Fast on-device LLM inference with npus. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1* (pp. 445-462).