

Assessing Performance Portability of Hybrid Video Encoders

Maciej Okoń

0009-0005-7427-6784

Maria Curie-Skłodowska University (UMCS)

ul. Pl. Marii Curie-Skłodowskiej 5, 20-031 Lublin, Poland

Email: maciej.okon@mail.umcs.pl

Beata Bylina

0000-0002-1327-9747

Maria Curie-Skłodowska University (UMCS)

ul. Pl. Marii Curie-Skłodowskiej 5, 20-031 Lublin, Poland

Email: beata.bylina@mail.umcs.pl

Abstract—While the Java Virtual Machine ensures functional code portability, achieving consistent execution efficiency across hardware generations remains a significant challenge for compute-intensive multimedia applications. This paper evaluates performance portability (PP) in a hybrid video encoder, focusing on the interaction between algorithmic complexity and parallelization granularity. We contrast a legacy fine-grained model, utilizing macroblock-level task scheduling, with a coarse-grained architecture based on Group of Pictures (GOP) parallelism. The study quantifies how these two paradigms respond to increasing core counts and generational shifts in server-grade hardware.

The evaluation utilizes three configurations: an intra-frame baseline (F_B), a hybrid model with integer-pixel motion estimation (F_N), and a sub-pixel hybrid model (F_H). Stability is assessed across four server generations (C1–C4, with 12 to 32 cores per socket) using a framework that combines harmonic portability metrics (ϕ) with arithmetic efficiency averages ($\bar{\Phi}$) and cascade visualizations. This approach identifies the exact points where task synchronization and memory latency limit hardware utilization in managed execution environments.

The results demonstrate that the GOP-parallel implementation significantly enhances performance portability, achieving architectural portability metrics (ϕ_{arch}) between 0.83 and 0.87. In contrast, the legacy macroblock-level model exhibits severe efficiency degradation on modern high-density processors, with portability scores dropping as low as 0.41. Visualization through efficiency cascades and platform fingerprints confirms that the coarse-grained GOP model effectively mitigates synchronization overheads and latency sensitivities. The study demonstrates that fine-grained parallelization strategies are insufficient for maintaining performance portability on modern many-core architectures within managed environments, whereas a coarse-grained GOP-level approach provides a more robust alternative for achieving consistent execution efficiency.

I. INTRODUCTION

VIDEO encoding is a compute-intensive process that necessitates efficient utilization of parallel hardware resources. While general-purpose processors provide the flexibility required for modern compression standards, maintaining execution efficiency across different hardware generations remains a significant challenge for maintaining execution efficiency. This issue is particularly prominent in applications targeting managed runtime environments, such as the Java Virtual Machine (JVM). In these environments, the abstraction layers that ensure functional portability can introduce unpredictable synchronization overheads, which often prevent

the application from scaling linearly as the core density and architectural complexity of modern server platforms increase. This discrepancy necessitates a formal performance portability metric to quantify how consistently an application utilizes hardware potential across diverse platforms.

By applying the formal performance portability framework, this paper evaluates the performance portability of a hybrid video encoder by contrasting two fundamental parallelization paradigms: a legacy fine-grained model utilizing macroblock-level task scheduling and a proposed coarse-grained architecture based on Group of Pictures based parallelism. The study quantifies how task granularity affects execution stability across four server-grade platforms with varying core counts. By applying formal performance portability metrics [8], [9], the work identifies specific scalability limits inherent in managed execution environments.

The evaluation utilizes three encoder configurations: an intra-frame baseline (F_B), a hybrid model with integer-pixel motion estimation (F_N), and a sub-pixel hybrid model (F_H). The results show that while the legacy model exhibits significant efficiency degradation on high-density architectures, the GOP-parallel approach maintains high hardware utilization across the tested platforms. The analysis indicates that coarse-grained task isolation is necessary to mitigate the synchronization penalties and communication costs that otherwise limit the scalability of fine-grained models.

The remainder of this article is organized as follows: Section II describes the transition from intra-frame to hybrid encoding, details the parallel execution architectures, and defines the evaluated encoder configurations. Section III introduces the formal performance portability framework and the specific metrics used for assessment. Section IV outlines the experimental environment and presents the computational results, including the performance portability evaluation and the cascade analysis of hardware sensitivity. Finally, Section V concludes the paper with an analysis of the impact of task granularity on execution stability across tested platforms.

II. PARALLEL EXECUTION ARCHITECTURES FOR HYBRID ENCODING

The transition from a purely intra-frame encoding model, as discussed in [1], to a hybrid predictive model requires

a fundamental reassessment of parallelization strategies. In hybrid architectures, the compression process relies heavily on inter-picture prediction to exploit temporal redundancy. This process utilizes a prediction loop where the signal for the current block is derived from prior coded pictures that serve as reference frames [3].

These temporal dependencies inherently limit the applicability of simple frame-level parallelism. Because each predicted frame (P-VOP) requires the full reconstruction of its predecessor to perform accurate motion estimation and compensation, frames cannot be processed as independent entities [7]. Consequently, the encoder must exploit parallelism either at a finer granularity within the frame or at a coarser granularity across independent segments of the video stream.

A. Fine-grained Parallelism: Macroblock-level Executor

The first approach evaluated is a fine-grained model that parallelizes the workload at the macroblock (MB) level within a single frame. The macroblock serves as the basic overhead unit for transform and prediction processes [6]. In this implementation, a frame is partitioned into 16×16 pixel blocks, which are then queued as individual tasks. This model is implemented using the `ExecutorService` framework provided by the Java standard library [4].

B. Coarse-grained Parallelism: GOP-level Implementation

To mitigate the synchronization bottlenecks of the fine-grained model, a coarse-grained architecture based on Group of Pictures (GOP) parallelism was implemented. This strategy leverages the fact that GOPs can be treated as independent units, allowing multiple video segments to be encoded simultaneously across different threads [2].

The video stream is segmented into independent GOPs of up to 30 frames, each treated as an atomic task submitted to a thread pool managed by the `ExecutorService`. This model provides dependency isolation, as frames within a single task execute sequentially, allowing each thread to maintain its own local reconstructed reference frame without inter-thread locking or shared-memory synchronization. Additionally, an integrated Scene Change Detection module finalizes GOP tasks upon detecting significant temporal shifts to reset the prediction chain according to content characteristics.

To optimize the computational kernel, the implementation incorporates a P-SKIP mechanism. For each macroblock, the encoder evaluates the motion vector and residual energy; if the block is static and the prediction error falls below a predefined threshold, the subsequent transform, quantization, and entropy coding stages are bypassed.

C. Encoder Variants

To evaluate the interaction between parallel execution and algorithmic complexity, the study investigates three selected encoder configurations. These variants represent escalating levels of computational intensity and varying hardware sensitivities, enabling a granular analysis of performance portability across platforms.

- **Baseline Variant (F_B):** An implementation operating exclusively in Intra-frame mode (I-VOP). This version utilizes DCT transform, quantization, and entropy coding for every block. It represents a highly regular, *compute-bound* workload with no temporal dependencies, serving as the reference point for raw arithmetic throughput.
- **No-HalfPel Variant (F_N):** A hybrid predictive implementation that introduces inter-frame coding. It includes Integer-pixel Motion Estimation (IME) and a P-SKIP mechanism. While IME adds search-related overhead, the P-SKIP logic allows the encoder to bypass the transform and entropy stages for static blocks.
- **Hybrid Variant (F_H):** The full-featured implementation including Half-Pel motion estimation refinement. In addition to the IME used in F_N , it performs bilinear interpolation to achieve sub-pixel precision. This variant is the most complex, combining search operations with a compute-intensive interpolation kernel, primary architectural stress test.

III. PERFORMANCE PORTABILITY

Assessing the impact of the previously discussed parallelization models and algorithmic variants requires a methodology that accounts for performance stability across diverse hardware. For this purpose, we adopt the performance portability (PP) framework [8], which defines the core concepts as follows:

- **Platform (i):** A specific execution environment consisting of a hardware architecture, operating system, and the complete software stack (compilers, drivers, and runtime libraries).
- **Application (a):** Any software package capable of processing a given problem p (e.g., video encoding) and producing a correct, verifiable output. In this study, the distinct GOP-level and macroblock-level models are considered separate implementations of the same application.
- **Performance:** Any measurable property of an application's correct execution on a platform. Here, performance is quantified as throughput in Frames Per Second (FPS).
- **Portability:** A binary property describing the ability of an application to execute a problem correctly on a given set of platforms H . An application is portable for H only if it runs correctly on every platform in the set.

A. Performance Portability Metric (ϕ)

Performance Portability (ϕ) is a measurement of an application's performance efficiency for a problem executed correctly on all platforms in a set H . Formally, for a problem p , it is defined as the harmonic mean of the efficiencies:

$$\phi(a, p, H) = \begin{cases} \frac{|H|}{\sum_{i \in H} \frac{1}{e_i(a, p)}} & \text{if } i \text{ is supported } \forall i \in H \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

where $e_i(a, p)$ represents the performance efficiency on platform i , provided that the platform executed the problem p

correctly and produced verifiable output. This efficiency can be expressed in two primary forms:

- 1) **Architectural Efficiency:** The fraction of peak theoretical hardware performance (e.g., memory bandwidth or compute throughput) utilized by the application.
- 2) **Application Efficiency:** The performance achieved by the application normalized relative to the best-known performance of any implementation on the same platform.

The specific procedures for calculating these efficiency values based on throughput and core counts in our experiments are detailed in Section IV.

B. Refinements: Arithmetic Performance Portability ($\bar{\Phi}$)

To address the high sensitivity of the harmonic mean to performance outliers on specific platforms, an arithmetic approach to performance portability is adopted [9]. This metric, denoted in this study as $\bar{\Phi}$, is defined as the arithmetic mean of performance efficiencies observed across the set of supported platforms $S \subseteq H$:

$$\bar{\Phi}(a, p, S, H) = \begin{cases} \frac{\sum_{i \in S} e_i(a, p)}{|S|} & \text{if } |S| > 0 \\ 0 & \text{otherwise} \end{cases} \quad (2)$$

By evaluating this metric within a single architecture class, we can precisely quantify how well an implementation maintains its efficiency across the tested CPUs.

IV. PERFORMANCE ANALYSIS AND TEST ENVIRONMENT

The performance portability of the developed GOP-parallel encoder was evaluated across four heterogeneous server platforms. These systems were selected to represent different microarchitectures, core densities, and memory controller generations, providing a baseline for assessing execution efficiency:

- **C1:** 2 × Intel Xeon E5-2670 v3 (12 cores / 24 threads, Haswell-EP)
- **C2:** 2 × Intel Xeon Gold 5218R (20 cores / 40 threads, Cascade Lake)
- **C3:** 2 × Intel Xeon Gold 6342 (24 cores / 48 threads, Ice Lake-SP)
- **C4:** 2 × Intel Xeon Platinum 8358 (32 cores / 64 threads, Ice Lake-SP)

The evaluation focuses on the performance gap between the legacy *Macroblock-level* parallelization (utilizing `ExecutorService` for individual blocks) and the proposed *GOP-Parallel* model. The raw throughput (FPS) results are summarized in Tables I and II.

The data reveals a fundamental difference in how both models scale with the workload size and hardware complexity. In the *GOP-Parallel* implementation (Table I), throughput increases significantly with the number of processed frames. For instance, on the C4 platform, the F_H variant scales from 1.79 FPS at 250 frames to 11.34 FPS at 2000 frames. This trend confirms that processing larger sequences effectively amortizes the initial overhead of thread pool management and bitstream serialization, enabling higher utilization of available

hardware resources and demonstrating sustained scalability with increasing workload volume.

In contrast, the legacy *Macroblock-level* model (Table II) demonstrates an initial throughput advantage for small workloads but quickly hits a scaling wall. While the fine-grained implementation utilizes hardware threads more effectively for sequences up to 500 frames, it fails to translate larger workloads into higher performance. As the number of frames increases, the legacy model's throughput remains nearly stagnant. This confirms that the massive management overhead of millions of individual macroblock tasks and the subsequent synchronization of `Future` objects create a fixed bottleneck. Even as hardware resources and workload volume grow, this architectural limitation prevents the legacy model from scaling upward, whereas the coarse-grained approach effectively amortizes its initial costs to achieve significantly higher sustained throughput.

Table I
COMPUTATIONAL PERFORMANCE (FPS) FOR ENCODER VARIANTS USING GOP-PARALLELISM.

Frames	C1			C2			C3			C4		
	F_H	F_N	F_B	F_H	F_N	F_B	F_H	F_N	F_B	F_H	F_N	F_B
250	1.16	3.58	1.14	1.57	4.96	1.56	1.84	5.68	1.89	1.79	5.52	1.83
500	2.15	2.77	2.24	2.89	3.55	2.88	3.54	4.13	3.76	3.43	4.01	3.65
750	2.92	3.82	3.09	3.90	4.98	3.94	5.18	6.16	5.47	5.13	6.01	5.44
1000	3.42	4.74	3.24	4.90	6.55	4.81	6.89	8.18	7.25	6.83	7.99	7.03
1250	3.72	5.59	3.66	5.51	7.51	4.77	8.48	9.88	9.01	8.43	9.63	8.77
1500	3.49	6.11	2.77	5.91	8.54	5.49	9.06	11.62	7.89	9.96	11.53	10.47
1750	3.26	6.68	3.03	6.15	9.39	6.09	8.98	13.22	8.90	11.27	13.35	11.99
2000	3.47	7.23	3.44	6.76	10.08	6.56	9.50	14.85	9.62	11.34	15.07	10.97

* F_H : Hybrid, F_N : No-HalfPel, F_B : Baseline

Table II
COMPUTATIONAL PERFORMANCE (FPS) FOR ENCODER VARIANTS USING MACROBLOCK-LEVEL PARALLELISM.

Frames	C1			C2			C3			C4		
	F_H	F_N	F_B	F_H	F_N	F_B	F_H	F_N	F_B	F_H	F_N	F_B
250	2.43	4.11	2.66	3.54	5.92	4.22	4.63	7.39	5.84	4.84	7.19	6.02
500	2.43	3.69	2.68	3.68	5.53	4.28	4.88	7.13	5.94	5.10	7.06	6.16
750	2.40	3.33	2.70	3.70	5.10	4.28	4.93	6.61	5.94	5.06	6.56	6.13
1000	2.45	3.55	2.71	3.83	5.27	4.30	5.13	6.81	5.94	5.28	6.76	6.10
1250	2.48	3.34	2.71	3.85	5.02	4.34	5.09	6.55	5.95	5.26	6.56	6.12
1500	2.34	3.44	2.72	3.85	5.15	4.32	5.13	6.68	5.95	5.27	6.72	6.11
1750	2.48	3.44	2.70	3.82	5.16	4.34	5.14	6.70	5.98	5.29	6.63	6.14
2000	2.47	3.42	2.71	3.83	5.16	4.34	5.13	6.73	5.96	5.31	6.65	6.13

* F_H : Hybrid, F_N : No-HalfPel, F_B : Baseline

Based on these scaling characteristics, the GOP-Parallel implementation is adopted as the reference for best-known performance when calculating application efficiency (e_i)

A. Performance Portability Evaluation

To derive meaningful performance portability metrics, the application efficiency (e_i) must first be calculated for each platform. These values are calculated for the 2000-frame workload to ensure the assessment reflects sustained throughput across all systems. Using the GOP-parallel implementation

Table III
APPLICATION EFFICIENCY (e_{app}) COMPARISON: BASELINE VS. GOP
PARALLEL.

Variant	Implementation	C1	C2	C3	C4
Hybrid (F_H)	Baseline	0.712	0.567	0.540	0.468
	GOP Parallel	1.000	1.000	1.000	1.000
No-HalfPel (F_N)	Baseline	0.473	0.512	0.453	0.441
	GOP Parallel	1.000	1.000	1.000	1.000
Baseline (F_B)	Baseline	0.788	0.662	0.620	0.559
	GOP Parallel	1.000	1.000	1.000	1.000

as the reference for best-known performance, Table III summarizes these efficiency values across the architectural set $H = \{C1, C2, C3, C4\}$.

The application efficiency e_{app} represents the fraction of the best observed performance achieved by an implementation on a given platform. For instance, the efficiency values for the F_H variant on the C1 and C4 platforms at the 2000-frame mark are derived as follows:

$$e_{app,C1} = \frac{2.47}{3.47} \approx 0.712, \quad e_{app,C4} = \frac{5.31}{11.34} \approx 0.468 \quad (3)$$

In addition to implementation-specific metrics, architectural efficiency (e_{arch}) is established to evaluate how effectively the hardware resources are utilized across different processor generations. In this study, e_{arch} is quantified as the throughput per physical core, normalized against the peak observed efficiency within the architectural set H :

$$e_{arch,i} = \frac{FPS_i/N_i}{\max_{j \in H}(FPS_j/N_j)} \quad (4)$$

where N_i denotes the number of available physical cores on platform i .

This distribution reveals a critical downward trend: the legacy implementation's efficiency significantly declines as the hardware becomes more modern and parallel. While the macroblock-level executor remains somewhat competitive on C1 (reaching up to 0.78 for F_B), its efficiency drops by nearly half on the high-density Ice Lake-SP architecture (C4).

A more nuanced observation can be made by comparing the F_H and F_N variants. On the C1 platform, the compute-heavy F_H variant achieves a higher efficiency (0.712) than the faster F_N variant (0.473). This indicates that the legacy model's task management overhead becomes even more dominant when the actual encoding work per task is reduced. In the case of F_N , the overhead of the `ExecutorService` and `Future` objects is disproportionately high relative to the simplified integer-only motion estimation, effectively bottlenecking the system regardless of raw hardware power. Across all variants, the consistent degradation of e_i towards C4 underscores that fine-grained task submission in Java cannot scale with increasing core counts, making the coarse-grained GOP-parallel approach essential for maintaining performance portability.

The final portability score ϕ , calculated as the harmonic mean of efficiencies following the formula 1 is presented in

Table IV
ARCHITECTURAL EFFICIENCY (e_{arch}) COMPARISON BASED ON
THROUGHPUT PER TOTAL PHYSICAL CORES.

Variant	Implementation	C1	C2	C3	C4
Hybrid (F_H)	Legacy	0.520	0.484	0.540	0.419
	GOP Parallel	0.731	0.854	1.000	0.895
No-HalfPel (F_N)	Legacy	0.461	0.417	0.453	0.336
	GOP Parallel	0.974	0.814	1.000	0.761
Baseline (F_B)	Legacy	0.563	0.541	0.619	0.478
	GOP Parallel	0.715	0.818	1.000	0.855

Table V. The results indicate that the GOP-Parallel implementation achieves high architectural portability, with ϕ_{arch} values between 0.83 and 0.87. This suggests that the GOP-level parallelization strategy effectively abstracts hardware differences, maintaining consistent utilization across the tested processor generations and memory subsystems.

Among the variants, the No-HalfPel (F_N) configuration demonstrates the highest portability ($\phi_{arch} = 0.876$). This stability is attributed to its balanced computational profile. In comparison, the full Hybrid variant (F_H) shows slightly lower portability (0.859), indicating that the sub-pixel interpolation kernels introduce a higher degree of sensitivity to specific JIT-compiler optimizations and instruction set extensions.

The legacy implementation exhibits significantly lower portability, with ϕ_{arch} ranging from 0.41 to 0.55. As the harmonic mean is heavily weighted by the lowest efficiency values, the low e_i scores observed on high-core-count platforms like C4 pull down the final portability results. The application-level metrics, also summarized in Table V, further quantify the performance gap. The GOP-Parallel implementation maintains a constant value of 1.000 for both ϕ_{app} and $\bar{\Phi}_{app}$, as it serves as the best-known performance reference. In contrast, the legacy model achieves significantly lower application portability, with ϕ_{app} ranging from 0.468 to 0.647. A comparison between the harmonic mean (ϕ) and the arithmetic average ($\bar{\Phi}$) provides a numerical measure of architectural sensitivity; for the legacy F_H variant, the harmonic portability (0.559) is lower than the arithmetic portability (0.572). This divergence confirms the presence of a performance outlier - specifically the C4 platform—which disproportionately penalizes the harmonic mean. The convergence of both metrics at peak levels for the GOP-parallel model demonstrates that coarse-grained task isolation effectively eliminates the scalability wall encountered by the legacy executor.

B. Cascade Analysis and Hardware Sensitivity

To gain a deeper understanding of the performance portability characteristics, we utilize the visualization framework established in [10]. Figure 1 presents the efficiency cascades (lines) and the corresponding platform fingerprints (barcodes) for all encoder variants. This approach provides a more granular view of performance portability by illustrating how ϕ degrades as the platform set H is progressively expanded. Unlike a static table, this method reveals the sensitivity of

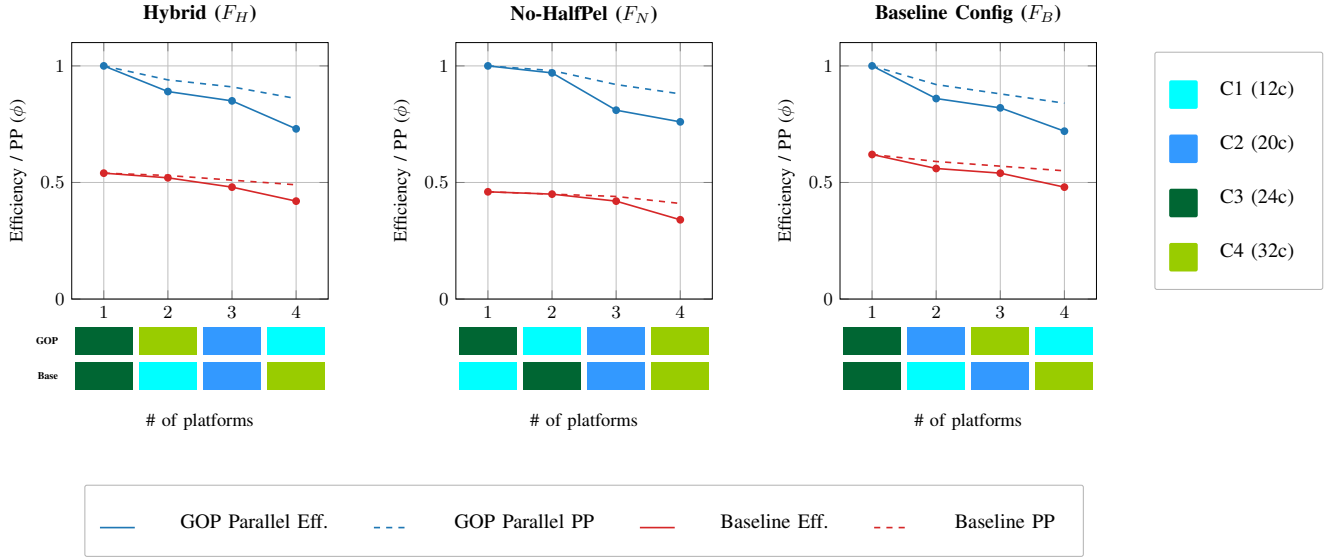
Figure 1. Efficiency cascades and performance portability (ϕ) for three encoder configurations.

Table V
COMPARISON OF PERFORMANCE PORTABILITY (ϕ) AND ($\bar{\Phi}$) ACROSS
ARCHITECTURAL AND APPLICATION LEVELS.

Implementation	Variant	Architectural PP		Application PP	
		ϕ_{arch}	$\bar{\Phi}_{arch}$	ϕ_{app}	$\bar{\Phi}_{app}$
Legacy	Hybrid (F_H)	0.486	0.491	0.559	0.572
	No-HalfPel (F_N)	0.410	0.417	0.468	0.470
	Baseline (F_B)	0.546	0.550	0.647	0.657
GOP Parallel	Hybrid (F_H)	0.859	0.870	1.000	1.000
	No-HalfPel (F_N)	0.876	0.887	1.000	1.000
	Baseline (F_B)	0.835	0.847	1.000	1.000

each implementation to specific architectures and identifies the outliers that disproportionately affect the overall portability score.

The efficiency cascades reveal that the GOP-Parallel implementation maintains significantly higher absolute efficiency across all variants (F_H, F_N, F_B), despite exhibiting a steeper rate of decline compared to the legacy model. This steeper slope for the GOP model is a consequence of its role as the performance reference ($e_i = 1.0$); however, its final portability score (ϕ) remains consistently above 0.83. In contrast, the legacy model's flatter trajectory at a much lower efficiency level (0.34 - 0.48) indicates a persistent inability to utilize the available hardware potential on any platform in the set.

The platform barcodes under the X-axis identify the architectural sensitivity of each implementation by ranking the platforms from most to least efficient. A significant divergence in hardware utilization is observed:

- **Inversion of Efficiency in Legacy Model:** In the F_N variant, the legacy implementation achieves its highest relative efficiency on the C1 platform (cyan). As more modern architectures are added, specifically the C4 (light

green), the efficiency drops, placing the most powerful hardware at the end of the ranking. This confirms that the macroblock-level executor is best suited for lower-core-count architectures and fails to scale with increasing hardware density.

- **Sustained Performance in GOP Model:** The GOP-Parallel implementation effectively utilizes the modern C3 and C4 platforms, which frequently occupy the top positions in its efficiency barcodes. The fact that the dashed ϕ lines for the GOP model stay significantly higher than those of the legacy model demonstrates that the coarse-grained approach is the only strategy capable of maintaining high performance portability across the tested heterogeneous environment.

V. CONCLUSION

This work evaluated the performance portability of a multithreaded video encoder in Java, contrasting a legacy fine-grained parallelization strategy against a proposed coarse-grained GOP-parallel architecture. By isolating the impact of algorithmic complexity across three variants (F_B, F_N, F_H), we demonstrated how task granularity dictates execution stability on modern high-density processors. The experiments, conducted on four generations of server-grade platforms (C1-C4), provided a quantitative basis for assessing performance stability using formal portability frameworks and efficiency cascade visualizations.

The results show that the GOP-parallel implementation achieved higher throughput and execution stability than the legacy model. The GOP-level approach yielded architectural portability (ϕ_{arch}) between 0.83 and 0.87, showing consistent hardware utilization across the tested platform set. The No-HalfPel (F_N) variant resulted in the highest portability score ($\phi_{arch} = 0.876$), as it maintained efficiency levels across

platform generations. In comparison, the legacy macroblock-level implementation achieved lower portability ($\phi_{arch} \approx 0.41 - 0.55$).

The analysis confirms that task granularity and synchronization overhead are the primary factors shaping performance portability in managed runtime environments. The divergence between the harmonic portability and the arithmetic average ($\bar{\Phi}$) in the legacy model quantifies the performance penalty incurred on high-density architectures. While the fine-grained executor reached a scaling wall, the coarse-grained GOP-parallel approach maintained consistent efficiency.

These findings validate the need to assess application behavior through dedicated performance portability metrics, moving beyond isolated speedup scenarios. In production server environments where CPUs must take on end-to-end responsibility for video processing without GPU acceleration, software-level architectural choices dictate how well a system scales.

Future research may explore extending this performance portability study to energy-to-solution metrics or hybrid CPU-GPU architectures. Nevertheless, this work provides a grounded perspective on achieving hardware-agnostic efficiency in CPU-bound multimedia pipelines and offers a valuable reference for porting parallel applications across heterogeneous server environments.

ACKNOWLEDGMENT

During the preparation of this work, the authors used Gemini (Google) for editorial refinement. The authors take full responsibility for the content of this publication.

REFERENCES

- [1] B. Bylina and M. Okoń, "A Multithreaded Java-Based Video Encoder for Multicore Systems," in *Proc. 20th Conf. Computer Science and Intelligence Systems (FedCSIS)*, ACSIS, vol. 43, 2025, pp. 659–664. doi: 10.15439/2025F1886.
- [2] S. Sankaraiah, L. H. Shuan, C. Eswaran, and J. Abdullah, "Performance Optimization of Video Coding Process on Multi-Core Platform Using GOP Level Parallelism," *International Journal of Parallel Programming*, vol. 42, no. 6, pp. 931–947, Dec. 2014. doi: 10.1007/s10766-013-0267-4.
- [3] T. Wiegand, G. J. Sullivan, G. Bjøntegaard, and A. Luthra, "Overview of the H.264/AVC Video coding standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 7, pp. 560–576, 2003. doi: 10.1109/TCSVT.2003.815165.
- [4] Oracle, "Concurrency Utilities," *Java Platform, Standard Edition 8 API Specification*, 2014. [Online]. Available: <https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/package-summary.html>.
- [5] G. J. Sullivan, J. Ohm, W. Han, and T. Wiegand, "Overview of the High Efficiency Video Coding (HEVC) Standard", in *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, no. 12, pp. 1649–1668, Dec. 2012. doi: 10.1109/TCSVT.2012.2221191.
- [6] I. E. G. Richardson, *H.264 and MPEG-4 Video Compression: Video Coding for Next-generation Multimedia*. Chichester, U.K.: Wiley, 2003.
- [7] V. Sze, M. Budagavi, and G. J. Sullivan, *High Efficiency Video Coding (HEVC): Algorithms and Architectures*. Cham, Switzerland: Springer, 2014. doi: 10.1007/978-3-319-06895-6.
- [8] S. J. Pennycook, J. D. Sewall, and V. W. Lee, "A Metric for Performance Portability," arXiv preprint, 2016. [Online]. Available: <https://doi.org/10.48550/arXiv.1611.07409>.
- [9] A. Marowka, "Portability Efficiency Approach for Calculating Performance Portability," arXiv preprint, Nov. 2024. [Online]. Available: <https://doi.org/10.48550/arXiv.2407.00232>.
- [10] J. Sewall, S. Pennycook, D. Jacobsen, T. Deakin, and S. McIntosh-Smith, "Interpreting and Visualizing Performance Portability Metrics," in *Proc. Int. Workshop Perform. Portability Product. HPC (P3HPC)*, Nov. 2020, pp. 14–24. doi: 10.1109/P3HPC51967.2020.00007.