

Article

Adaptive Load Balancing for Efficient Background Subtraction in Intelligent Transport Systems on Low-Cost Embedded Platforms

Brahim Tebbaa ¹ , Mohamed Ragoubi ¹ , Lhoussain El Hajjami ¹, Assia Arsalane ^{1,2} , Abdessamad Klilou ¹  and Vidas Žuraulis ^{3,*} 

¹ Microelectronic, Embedded Systems and Telecommunications (MiSET) Laboratory, Faculty of Sciences and Technologies, Sultan Moulay Slimane University, Béni Mellal 23000, Morocco; brahim.tebbaa@usms.ac.ma (B.T.); mohamed.ragoubi@usms.ma (M.R.); l.elhajjami@usms.ma (L.E.H.); assia.arsalan@usms.ma (A.A.); a.klilou@usms.ma (A.K.)

² Department of mechatronics, Higher School of Technology, Sultan Moulay Slimane University, Béni Mellal 23000, Morocco

³ Department of Automobile Engineering, Vilnius Gediminas Technical University (VILNIUS TECH), Plytines Str. 25, LT-10105 Vilnius, Lithuania

* Correspondence: vidas.zuraulis@vilniustech.lt

Abstract

Background subtraction (BS) is a fundamental technique in intelligent transport vision systems, widely used to detect and track moving objects, such as vehicles, pedestrians and obstacles, in driving environments. It plays a crucial role in advanced driver-assistance systems (ADAS) and autonomous driving by enabling scene understanding and real-time motion analysis. However, BS processing must be optimized when targeting real-time processing on resource-constrained embedded systems, which present significant challenges due to limited computational power, memory constraints, and strict real-time requirements. Among the most commonly used BS techniques, the Codebook model and Gaussian Mixture Models (GMM) are known for their higher accuracy and light-model compared to many deep learning-based BS. In this work, we propose a fully heterogeneous CPU and GPU parallel implementation of both Codebook and GMM algorithms with an auto-load balancing over the processing units. This approach has been evaluated on the low-cost Jetson Orin Nano platform from NVIDIA, enabling efficient workload balancing across heterogeneous hardware resources. The suggested solution yields significant performance improvements over the state-of-the-art, achieving 59 frames per second (FPS) for GMM and 66 FPS for the Codebook method on full-HD (1080p) video streams. The results confirm the effectiveness of the proposed method in accelerating BS and demonstrate its suitability for real-time deployment in resource-constrained embedded environments.



Academic Editors: Jan Kalivoda and Larysa Neduzha

Received: 4 May 2026

Revised: 26 June 2026

Accepted: 27 June 2026

Published: 2 July 2026

Copyright: © 2026 by the authors.

Licensee MDPI, Basel, Switzerland.

This article is an open access article distributed under the terms and conditions of the [Creative Commons Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

Keywords: intelligent transport; background subtraction; GMM; codebook algorithm; parallel implementation; heterogeneous embedded architecture; CPU/GPU load balancing

1. Introduction

1.1. Background and Motivation

Background subtraction (BS) enables the detection of moving objects by separating foreground elements from a static background in video streams, allowing the real-time identification of vehicles, pedestrians and other objects. In the field of transport engineering,

it is used in various applications, including traffic flow monitoring, congestion analysis, incident detection, and intelligent traffic light management.

In the context of modern intelligent transportation systems (ITS) and advanced road surveillance, traditional BS algorithms serve as critical primary segmentation modules. They establish the computational baseline required for subsequent feature extraction, precise vehicle classification, and complex occlusion handling under highly variable outdoor lighting conditions [1]. It is also integrated into advanced driver-assistance systems (ADAS) for collision prevention, lane departure warning, and enhanced road safety monitoring. Furthermore, these methodologies have proven highly effective when deployed on resource-constrained embedded hardware, where robust clustering and statistical BS models, such as the Codebook algorithm, successfully filter dynamic environmental noise while respecting the strict real-time operational envelopes required for edge-based traffic density analysis [2].

However, in real-world ITS deployments, BS is not a standalone task but rather a fundamental component of a multi-stage analytical pipeline. In this study, the target application is a comprehensive real-time road infraction detection framework specifically designed for vehicle speed estimation using a stereo-camera setup. To effectively manage this multi-stage processing chain on heterogeneous edge hardware without concurrent runtime resource competition, a straightforward block-by-block parallelization model can be adopted [3]. Within this architecture, each computing phase ranging from frame acquisition and stereo preprocessing to BS and final tracking is executed sequentially. The system's entire computational resources cooperate fully on one single block at a time before synchronizing and moving on to the next. Consequently, optimizing the initial BS stage through an efficient load-balancing mechanism is of paramount importance, as its throughput directly governs the end-to-end latency and real-time viability of the entire downstream enforcement pipeline.

In this context, Drygala et al. [4] proposed a background foreground segmentation framework for interior sensing, enabling reliable detection of occupants and objects within the vehicle cabin under varying illumination conditions. Nevertheless, classical background subtraction methods rely on the assumption of a static camera, which is not valid in most real-world driving scenarios. To overcome this limitation, several studies have explored the integration of ego-motion compensation techniques. In this regard, Wei et al. [5] introduced the Moving-camera Background Subtraction Network (MBS-Net), a framework that incorporates motion-aware representations to address complex background variations induced by ego-motion. Their approach highlights the importance of explicitly modeling motion when applying background subtraction in autonomous driving scenarios. On the other hand, some studies have investigated the integration of background subtraction with complementary techniques to improve robustness in dynamic environments. For instance, Beigh et al. [6] combined multiple background subtraction strategies with deep learning-based detection, using background modeling as a preprocessing step to refine the regions of interest. Despite these efforts, intelligent transport systems remain limited, particularly regarding the use of statistical algorithm approaches with fully heterogeneous CPU–GPU parallel architectures. This limitation is especially relevant in resource-constrained embedded systems, where power consumption, computational capacity, and hardware costs are significantly restricted.

1.2. Related Works

As BS forms a cornerstone of current intelligent transport systems (ITS) and traffic monitoring technologies, numerous scientific studies have focused on this particular field to enable automated vehicle detection, counting, and classification. Some of the most used BS techniques include statistical models, such as Gaussian Mixture Models (GMM) [1,7], and clustering-based techniques [8,9], which remain widely adopted due to their robustness, adaptability and background maintenance mechanisms [10]. In the context of real-time road surveillance, these traditional BS algorithms play a critical role as primary segmentation modules, serving as the foundation for subsequent feature extraction, vehicle classification, and complex occlusion handling under highly variable outdoor lighting conditions [1]. Clustering methods include K-means, sequential clustering, and the well-known Codebook model. These models are more efficient and require less processing compared to deep learning algorithms, making them ideal for deployment on resource-constrained embedded platforms used in traffic management.

The Codebook algorithm, introduced by Kim et al. [11], represents each pixel through a set of adaptive codewords based on color distortion and brightness constraints, making it particularly suitable for dynamic outdoor environments. This model has proven highly effective in practical ITS frameworks, where it is utilized for vehicle detection and traffic density analysis owing to its ability to filter environmental noise while preserving real-time operational constraints on embedded hardware [2]. Several extensions, such as the Dynamic Codebook (DCB) [12], variants using the Lab* color space [13], and multispectral extensions [14,15], have further improved robustness under varying illumination and scene dynamics.

In parallel, GMM approaches have been widely adopted in real-world applications. The method proposed by Stauffer and Grimson [7] models each pixel as a mixture of Gaussians that evolve over time, providing robust performance under noise, shadows, and gradual lighting changes. More recent studies have further optimized and validated GMM on various hardware platforms and datasets.

Due to the high computational cost of BS, particularly when executed on high-resolution video streams, many studies have explored acceleration strategies targeting embedded platforms. Both GMM and Codebook algorithm deployment has been addressed by Iszaidy et al. [16]. They proposed an implementation on the low-cost Raspberry Pi system. They achieved good FPS performances but only for videos with medium resolution. Sehairi et al. [17] have implemented and optimized BS based on the GMM algorithm on two embedded platforms, i.e., Jetson TK1 and Jetson Nano. A performances comparison was provided for HD and full-HD video streams. The obtained results are 21 and 12 FPSs for HD and full-HD resolutions respectively.

Previous research works from our team [18,19] investigated the parallel implementation of the GMM algorithm on two embedded platforms: the eight-core Texas Instruments C6678 digital signal processor (DSP) and the Nvidia Jetson Nano. We proposed three parallelization strategies: OpenMP, a manually optimized version, and a manually tuned implementation leveraging the direct memory access (DMA) controller. Moreover, we introduced OpenMP's orphan directives, which improved parallel efficiency compared to standard OpenMP scheduling. However, the obtained results for full-HD stream did not exceed 7 FPS.

To the best of our knowledge, the work by Mabrouk et al. [20] is the closest to our research study. They propose an efficient strategy to implement compressive sensing and a mixture of Gaussians on heterogeneous CPU–GPU computing platforms. They proposed an optimal data distribution cursor (ODDC), in order to manage simultaneously the heterogeneous processors and ensure an automatic workload balancing by estimating

the optimal data chunk size that must be assigned to each processor. The contribution of this paper is slightly similar to Mabrouk et al. [20]. However, our portioning strategy is different and more optimized. It was developed from scratch and is based on just one parameter, which is the “ratio”, instead of the three parameters used by Mabrouk et al. [20], which reduce the runtime overhead. Furthermore, our proposed method adjusts dynamically the horizontal partition boundary row. Finally, our proposed approach was deployed and validated on a resource-constrained embedded platform instead of powerful platforms such as workstations.

1.3. Contributions

This study proposes fully heterogeneous parallel implementations of enhanced Codebook and GMM algorithms for real-time transport vision systems, deployed on the low-cost NVIDIA Jetson Orin Nano platform, which integrates a six-core Arm CPU and an NVIDIA Ampere architecture GPU with 1024 CUDA cores. The proposed framework is specifically designed to meet the stringent computational and latency requirements of intelligent transport systems.

The key contributions of this paper are:

- A multi-core CPU optimization using efficient OpenMP scheduling strategies.
- CUDA GPU kernels for accelerating both Codebook and GMM algorithms.
- An adaptive CPU–GPU load-balancing strategy enabling simultaneous execution on both processors to maximize throughput and ensure real-time performance under automotive constraints.
- A validation of all proposed approaches on low-cost and low-power embedded platforms in the context of automotive vision applications, demonstrating its suitability for dynamic and challenging driving environments.

1.4. Manuscript Structure

The remainder of this paper is organized as follows. Section 2 reviews the Codebook and GMM algorithms and describes the experimental setup. Section 3 presents the proposed heterogeneous implementations and GPU optimizations. Section 4 reports and discusses the experimental results. Section 5 concludes this paper and outlines future directions.

2. Materials and Methods

2.1. Codebook Algorithm

The Codebook algorithm used in this work is a refined and optimized version of the model proposed in our previous works [21]. Each pixel is represented by a compact set of codewords whose number and structure have been optimized to reduce computational cost while preserving segmentation accuracy. Furthermore, to improve the adaptability of the model, a Cache model was integrated to represent transient foreground objects entering or leaving the scene. Codewords associated with such transient elements are initially stored in the cache and are promoted to the permanent background model only if they persist for a predefined period. However, permanent codewords that are not observed for a specified duration are removed from the model. In addition, a filtering mechanism is applied to the cache to limit noise and short-lived fluctuations. Figure 1 illustrates the modeling process of the proposed Codebook algorithm. Owing to its compact memory footprint and predictable computational behavior, the refined Codebook model is particularly well suited for heterogeneous parallel implementations.

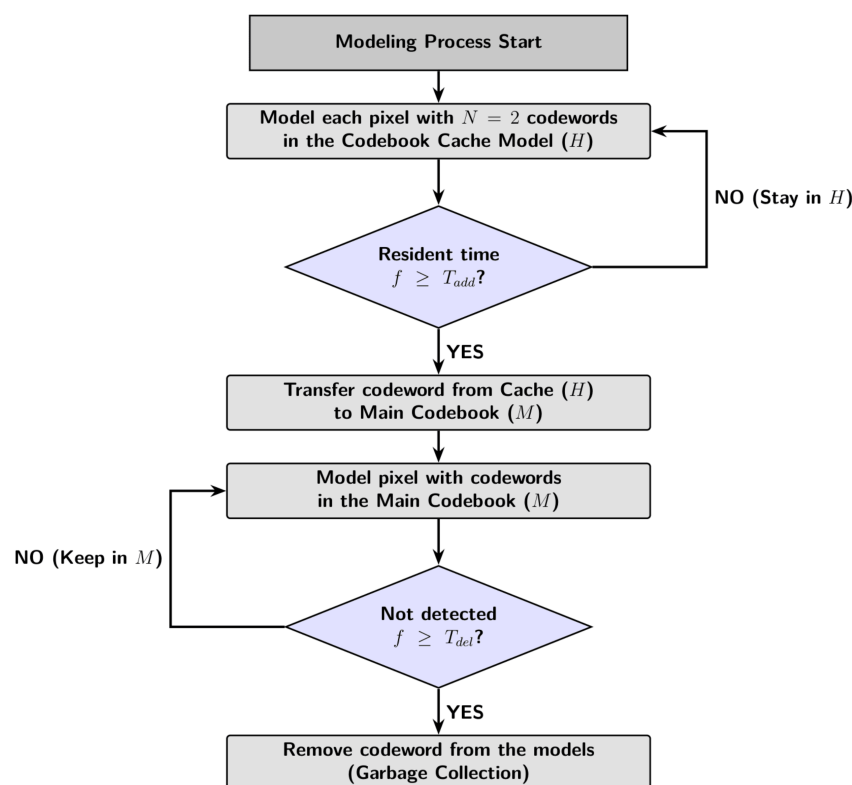


Figure 1. Modeling process of Codebook algorithm for the background subtraction.

2.2. Background Subtraction Based on GMM Algorithm

GMM constitutes one of the most widely used statistical approaches for BS in video analysis applications. Early formulations, proposed by Friedman and Russell [22], modeled each pixel using a fixed mixture of three Gaussian components. Although this approach provides high accuracy in relatively simple scenarios, it lacks the adaptability required for real-world environments characterized by dynamic backgrounds, illumination changes, and complex scene variations.

A major advancement was introduced by Stauffer and Grimson [7], who proposed a fully adaptive GMM framework for background modeling. In this approach, each pixel is represented by a mixture of K Gaussian distributions (with $K = 3$ in this work), where each Gaussian encodes an observed appearance mode of the pixel. As new frames are processed, the parameters of the mixture—namely the means, variances, and weights—are updated using a learning rate, enabling the model to adapt continuously over time. This adaptive mechanism allows GMM to cope with illumination variations, background motions, sensor noise, and gradual changes in scene appearance. Background and foreground classification are obtained by sorting the Gaussians according to their weight-to-variance ratio, which serves as a measure of their stability. The most persistent distributions form the background model, while pixels not matching any background Gaussian are labeled as foreground. The possibility linked to the present pixel value is formulated as shown in (1), which emphasizes the methodology's intrinsic probabilistic basis:

$$P(x_t) = \sum_{i=1}^k \omega_{i,t} * \eta(x_t, \mu_{i,t}, \Sigma_{i,t}) \quad (1)$$

where

$\omega_{i,t}$: The i th Gaussian's weight at time t .

$\mu_{i,t}$: Mean value.

$\Sigma_{i,t}$: Covariance matrix.

$\eta(X_t, \mu_{i,t}, \Sigma_{i,t})$: density function for Gaussian probability evaluating how well pixel X_t fits the distribution.

Each new pixel X_t is evaluated against the K existing Gaussians to see if it lies within 2.5 standard deviations of any distribution.

This study considers the classical adaptive GMM with specific refinements oriented toward efficient heterogeneous parallelization. The recursive update rules, matching conditions, and component reordering operations make the GMM particularly compatible with data-parallel execution, a key property exploited in the proposed CPU and GPU implementation.

2.3. Experimental Platforms Overview

For the real-world deployment of the studied BS algorithms, the Jetson Orin Nano developer kit from NVIDIA was used (Figure 2). This platform includes 1024 GPU CUDA cores in Ampere architecture with a six-core Arm CPU and 8 GB of 128-bit LPDDR5 RAM, making it suitable for real-time object detection and computer vision workloads on low-power edge devices. The module supports high-bandwidth memory access (up to 102 GB/s), multiple high-speed I/O interfaces, camera inputs, and Gigabit Ethernet connectivity, enabling seamless integration of sensors and peripherals in robotics and embedded vision systems. It is compatible with the NVIDIA Jetson Linux software, Ubuntu 22.04 LTS version, stack and supports CUDA, cuDNN, and TensorRT, enabling accelerated parallel processing and optimized deep learning inference. The typical power consumption is configurable between approximately 7 W and 15 W, offering an energy-efficient solution for embedded and robotics applications requiring on-device AI and computer vision processing [23].

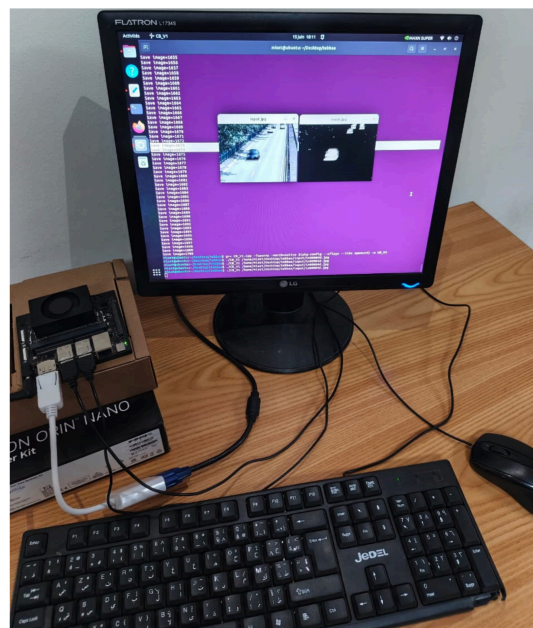


Figure 2. Experimental setup using Jetson Orin Nano embedded platform.

2.4. Metrics for Quality Evaluation

Mask quality assessment was based on standard metrics that have been widely adopted in the literature, namely *Precision*, *Recall*, and *F-measure*, explained in Equations (1)–(3). The use of these metrics allows consistent evaluation of segmentation accuracy across platforms.

The findings suggest that the proposed hardware implementation maintains the quality of detection while achieving higher speedups.

$$Recall = \frac{TP}{TP + FN} \quad (2)$$

$$Precision = \frac{TP}{TP + FP} \quad (3)$$

$$F - Measure = \frac{2 * (Precision * Recall)}{Precision + Recall} \quad (4)$$

where

TP: True Positive is the foreground pixels that are detected correctly as foreground.

TN: True Negative is the background pixels that are detected correctly as background.

FP: False Positive is the pixels that are actually background but wrongly detected as foreground.

FN: False Negative is the pixels that are actually foreground but detected as background.

3. Proposed Parallelization Approaches

3.1. OpenMP Parallelization Methods

OpenMP is a widely adopted API for shared-memory architecture that accelerates processing on such architectures [18,19,24,25]. Through compiler directives, developers specify which portions of code are executed concurrently and define how the workload is distributed among the available threads. A key directive for loop parallelism is the schedule clause, which defines how loop iterations are assigned to threads and therefore has a significant impact on overall performance. This study benchmarks five scheduling techniques on the Jetson Orin Nano platforms, i.e., static, dynamic, guided, runtime, and OpenMP Orphan Directive (OOD) [26].

3.2. GPU Parallelization Techniques

Modern GPUs are highly parallel, multi-core processors capable of delivering substantial computational throughput, particularly when combined with CUDA, which provides a unified programming environment for launching thousands of parallel threads through GPU kernels. In heterogeneous CPU and GPU systems, efficient task distribution is essential: the CPU manages image acquisition, memory allocation, and data transfers over the PCIe bus, while the GPU performs massively parallel processing. Real-time performance strongly depends on effective memory management. In this context, CUDA-enabled devices expose multiple memory spaces, including global, shared, texture, and registers where shared memory offers low-latency, high-bandwidth on-chip storage. To reduce memory bottlenecks caused by slow global memory, frequently accessed data, such as video frames or background model parameters, are often copied into shared memory for faster computation [27].

Furthermore, additional optimizations can be achieved through single instruction multiple data (SIMD) intrinsic instructions, which replace standard mathematical operations with faster, hardware-specific intrinsic functions. While the compiler option `-use_fast_math` can automatically apply these substitutions, precision may be affected. A selective replacement strategy has been proposed. In our implementation, intrinsic functions such as `_fdividef(x,y)` and `_powf(x,y)` were replaced with standard division and exponentiation instructions, yielding noticeable speedups without any impact on the accuracy [28].

The performance of edge computing applications is often bottlenecked by host-device transfers rather than core computations. To optimize CPU-to-GPU data transfers, data are buffered into the host system's RAM using page-locked (pinned) memory (cudaHostAl-

loc) [27]. This prevents the host operating system from paging the memory, allowing the GPU's direct memory access (DMA) engine to saturate the internal bus bandwidth and achieve maximum transfer throughput.

The proposed GPU implementation is presented in Algorithm 1.

Algorithm 1: Parallel implementation of BS on GPU

```

1  FUNCTION Main [CPU]
2  //Allocate GPU memory
3  allocate input_frames[] → GPU global memory
4  allocate output_masks[] → GPU global memory
5  allocate models (Codebook | GMM) → GPU global memory
6  InitializeModels(models)
7  //Configure kernel launch
8  blockSize = (16, 16)
9  gridSize = (frame.width/blockSize.x, frame.height/blockSize.y)
10 //Main processing loop
11 foreach frame in video_stream:
12   GetCurrentFrame(input_frame)
13   CopyToDevice(input_frame, CPU → GPU)
14   //Kernel invocation
15   BackgroundSubtractor <<<gridSize, blockSize>>>(input_frame, output_mask,
        models)
16   CopyToHost(output_mask, GPU → CPU)
17   DisplayResult(output_mask)
18   KERNEL BackgroundSubtractor (input_frame, output_mask, models,
        params) [GPU]
19   //Thread-to-pixel mapping
20   x = threadIdx.x + blockIdx.x * blockDim.x
21   y = threadIdx.y + blockIdx.y * blockDim.y
22   pixelIdx ← y * frame.width + x // 2D → 1D
23   modelIdx ← GetModelIndex(pixelIdx)
24   //1 thread ↔ 1 pixel
25   //Load to shared memory
26   LoadShared(input_frame, global → shared)
27   LoadShared(models, global → shared)
28   SyncThreads()
29   //Per-pixel model update (Codebook or GMM)
30   pixel ← shared_frame[pixelIdx]
31   if mode == CODEBOOK:
32     output_mask[pixelIdx] ← CodebookMatch(pixel, models[modelIdx])
33   else if mode == GMM:
34     output_mask[pixelIdx] ← GMMMatch(pixel, models[modelIdx])

```

3.3. Optimal CPU–GPU Workload Partitioning

The heterogeneous implementation leverages both CPU and GPU resources to accelerate the background subtraction pipeline. Each input frame is segmented into two regions processed concurrently: low-complexity or irregular tasks are handled by multi-core CPU threads using OpenMP, while compute-intensive pixel-wise operations are executed on the GPU through highly parallel CUDA kernels with optimized memory access.

The CPU and GPU workload ratio is defined at runtime to balance execution times and avoid bottlenecks. By using asynchronous data transfers and overlapping CPU and GPU execution, the system minimizes idle time and overlap communication latency. This combined method significantly improves throughput compared to CPU-only or GPU-only implementations, yielding substantial FPS gains and enabling real-time performance.

To formalize this dynamic balancing process, the workload distribution relies on a continuous performance monitoring loop executed at the end of each frame processing cycle, avoiding any invasive or computationally heavy offline profiling step. The framework records the absolute runtime execution latencies of both the CPU and GPU components, with the primary objective of minimizing the temporal divergence between the two processing domains. It is important to note that, from an algorithmic standpoint, pixel-level background subtraction methods, such as GMM or Codebook, execute a deterministic sequence of mathematical instructions per pixel regardless of scene content, consistently calculating color distortions and updating background models. However, from a hardware execution perspective, spatial anomalies and sudden traffic variations introduce severe runtime variations. When a substantial number of pixels transition from background to foreground, it triggers differing memory access paths, structural code branches for code-word maintenance, and localized non-coalesced memory reads on the GPU or cache misses on the multi-core CPU. This hardware-level execution lag directly translates spatial scene complexity into temporal asymmetry. The adaptive scheduler immediately registers this execution imbalance through the measured processing times. By dynamically adjusting the horizontal partition boundary row, the system re-allocates the workload to compensate for these hardware execution delays, maintaining an optimal end-to-end throughput under varying real-world traffic profiles.

Consequently, the frame partitioning boundary is updated dynamically for the subsequent frame according to the following mathematical formulation:

$$H_{CPU} = H_{Total} - H_{GPU} \quad (5)$$

where H_{CPU} is the number of rows assigned to the CPU. H_{GPU} is the number of rows assigned to the GPU. H_{Total} is the total number of rows in the input image.

Optimal performance is achieved when both devices complete their respective workloads at approximately the same time:

$$T_{GPU}(H_{GPU}) \approx T_{CPU}(H_{CPU}) \quad (6)$$

where T_{GPU} is the times processing of H_{GPU} rows assigned to the GPU. T_{CPU} is the times processing of H_{CPU} rows assigned to the CPU.

3.3.1. Measurement of Processing Speeds

To characterize device performance, the average processing time per pixel is measured for both the GPU and the CPU during several testing frames.

$$T'_{GPU} = \frac{T_{GPU}}{N_{GPU}} \quad (7)$$

$$T'_{CPU} = \frac{T_{CPU}}{N_{CPU}} \quad (8)$$

$$N_{GPU} = H_{GPU} \times W \quad (9)$$

$$N_{CPU} = H_{CPU} \times W \quad (10)$$

with T'_{GPU} denotes the time processing per pixel for GPU, T'_{CPU} denotes the time processing per pixel for CPU, N_{GPU} and N_{CPU} denote the number of pixels assigned to the GPU and CPU respectively, and W denotes the number of pixels per row in the input image.

These measurements allow a continuous estimation of the computational cost per pixel on each processing unit, i.e., CPU and GPU.

3.3.2. Derivation of the Optimal Partition Ratio

Using the Equations (6)–(8), balancing CPU and GPU processing charge can be explained in Equation (11):

$$T'_{GPU} \times H_{GPU} = T'_{CPU} \times (H_{\text{total}} - H_{GPU}) \quad (11)$$

Solving Equation (11) for H_{GPU} yields the optimal partition:

$$H_{GPU} = \frac{T'_{CPU}}{T'_{CPU} + T'_{GPU}} H_{\text{total}} \quad (12)$$

Similarly, for H_{CPU} , we get:

$$H_{CPU} = H_{\text{total}} - H_{GPU} \quad (13)$$

Equations (12) and (13) provide a direct method for calculating the number of rows allocated to each device based only on their measured performance per pixel.

3.3.3. Dynamic Online Adaptation

Since the CPU load and kernel execution time may vary during runtime (e.g., due to system activity or varying scene complexity), the optimal partition is recalculated at every frame in order to maintain balanced device utilization and ensure steady real-time performance. Figure 3 and Algorithm 2 describe the proposed method for dynamic load balancing between CPU and GPU.

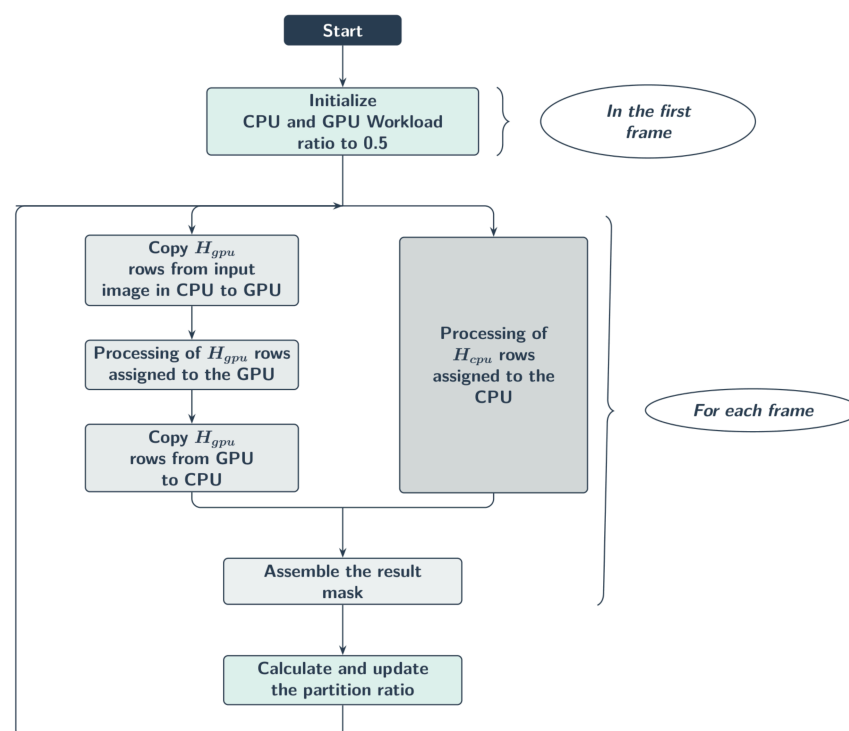


Figure 3. Adaptive load balancing when BS algorithm is running.

Algorithm 2: Adaptive Load Balancing Pseudocode

```

1  FUNCTION Main
2  //Allocate memory & init models
3  allocate input_frame, output_mask, models → CPU & GPU
4  InitializeModels(models)
5  //CUDA streams
6  cudaStreamCreate(stream_gpu)
7  //Compute initial CPU/GPU partition
8  ratio_gpu ← 0.5 //will be updated dynamically
9  H_gpu ← ratio_gpu × frame.height; H_cpu ← frame.height − H_gpu
10 while video_stream has frames:
11     ReadFrame(input_frame)
12     //Async GPU transfer + kernel launch
13     cudaMemPrefetchAsync(input_frame[0..H_gpu], CPU→GPU, stream_gpu)
14     LaunchKernel <<<grid, block, stream_gpu>>> (input_frame, output_mask,
models, H_gpu)
15     //CPU OpenMP — concurrent with GPU
16     BSModel_CPU(input_frame[H_gpu..H_total], models)
17     //Sync & update ratio (EMA  $\alpha = 0.8$ )
18     cudaEventSynchronize()
19     ratio_gpu ← t_cpu/(t_cpu + t_gpu)
20     H_gpu ← clamp(ratio_gpu × H_total, 1, H_total−1)
21     H_cpu ← H_total − H_gpu
22     DisplayResult(output_mask)
23 pm.stop()
24 pm.print(measured, avg_sync)

```

3.3.4. CPU–GPU Data Transfer Optimization

Minimizing latencies of data transfer between the host (CPU) and the device (GPU) is critical to improve computing performances in CPU–GPU heterogenous architecture. To this end, data transfer must be overlapped with the processing instead of making processing and data transfer in a sequential manner, as illustrated in Figure 4. CUDA streams [28] tools have been employed to manage this parallelism scenario, i.e., while the GPU processes frame F_t , input data for the next frame $F_{(t+1)}$ are transferred from the host to the device, and the output data of the last frame $F_{(t-1)}$ are back-transferred from the device to the host. This stream-based strategy allows simultaneous memory transfers and kernel running.

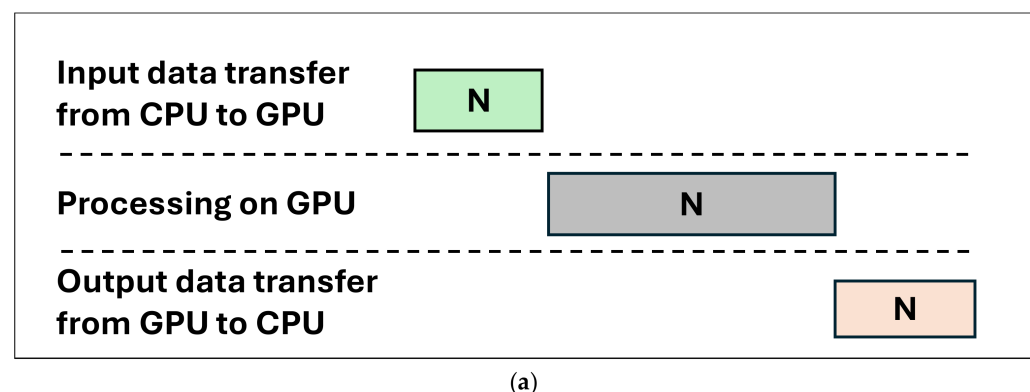


Figure 4. Cont.

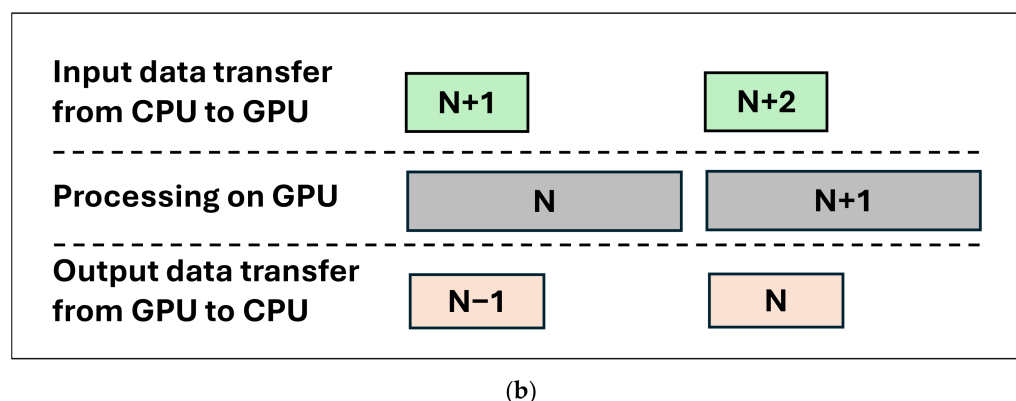


Figure 4. Overlapping CPU–GPU data transfers and processing: (a) without Cuda streams; (b) with Cuda streams.

4. Experimental Results

4.1. Datasets and Metrics Quality Evaluation

The video sequences used in this evaluation were obtained from two widely adopted benchmark datasets: the Change Detection 2014 dataset (CDnet 2014) [29] and the Background Models Challenge (BMC) 2012 workshop [30], comprising the “Street” (sequence 112) and “Rotary” (sequence 122) synthetic sequences. Both datasets are extensively used in the background subtraction literature in order to validate algorithm precision across a broad range of real-world conditions, including dynamic illumination changes, camera motion, and intermittent object motion [10,16,19,31].

The Codebook and GMM algorithms were implemented in C/C++ using OpenMP and CUDA and applied to the full set of available sequences. Figure 5 illustrates representative foreground masks produced by both algorithms on the highway sequence from CDnet 2014. Visual inspection confirms that the generated masks closely match the dataset ground truth, providing an initial qualitative indication that parallelization does not alter the fundamental behavior of either algorithm.



Figure 5. Qualitative results of CD net 2014 highway frame 1626. From left: (a) original frame, (b) ground truth, (c) Codebook algorithm and (d) GMM algorithm.

To quantify this observation, the quality of the produced masks was assessed using the standard recall, precision, and F-measure metrics introduced in Section 2.4. The values obtained were compared against the reference scores provided by each dataset as well as results reported in selected prior works. The results are summarized in Tables 1 and 2 for the Codebook and GMM algorithms respectively.

Table 1. Quantitative evaluation of Codebook implementation.

Datasets	Change Detection 2014 Dataset			112 Synthetic Video from BMC 2012 “Street”			122 Synthetic Video from BMC 2012 “Rotary”		
Methods	Recall	Precision	F-measure	Recall	Precision	F-measure	Recall	Precision	F-measure
CD-NET references values [29]	0.71	0.90	0.76	n/a	n/a	n/a	n/a	n/a	n/a
Iszaidy et al. [16]	n/a	n/a	n/a	0.91	0.04	0.07	n/a	n/a	n/a
This work	0.78	0.83	0.79	0.81	0.97	0.88	0.81	0.95	0.88

Table 2. Quantitative evaluation of GMM implementation.

Datasets	Change Detection 2014 Dataset			112 Synthetic Video from BMC 2012 “Street”			122 Synthetic Video from BMC 2012 “Rotary”		
Methods	Recall	Precision	F-measure	Recall	Precision	F-measure	Recall	Precision	F-measure
CD-NET and BMC references values [29,30]	0.69	0.70	0.65	0.92	0.86	0.89	0.92	0.88	0.90
Bariko et al. [18]	0.62	0.81	0.69	0.93	0.82	0.87	0.95	0.81	0.88
This work	0.86	0.56	0.68	0.9	0.84	0.87	0.89	0.86	0.88

The recall metric reflects the proportion of true foreground pixels correctly identified by the detector and is directly related to the sensitivity of the background model. The GMM method demonstrates the highest sensitivity across all three datasets, achieving recall values of 0.86, 0.90, and 0.89 on CDnet 2014, BMC “Street”, and BMC “Rotary” respectively. The Codebook algorithm produces lower but consistent recall values of 0.78, 0.81, and 0.81 on the same sequences, reflecting a more conservative foreground classification strategy. On the synthetic BMC sequences, where illumination is controlled and scene complexity is reduced, the gap between the two methods narrows, suggesting that the Codebook model is particularly well-suited to stable environments.

The precision metric quantifies the reliability of detections, representing the proportion of pixels classified as foreground that are genuinely so. In this respect, the Codebook method yields markedly stronger results, achieving precision values of 0.83 on CDnet 2014 and as high as 0.97 and 0.95 on the BMC “Street” and “Rotary” sequences. The GMM method, while more sensitive, produces a substantially lower precision of 0.56 on CDnet 2014, indicating a higher rate of false positives on real-world sequences affected by dynamic backgrounds and illumination variation. On the BMC datasets, however, GMM precision improves considerably, reaching 0.84 and 0.86, and closely approaches the reference values of 0.86 and 0.88. These observations confirm a consistent precision–recall trade-off between the two algorithms: the GMM prioritizes sensitivity at the cost of specificity, while the Codebook achieves the inverse balance.

Taken together, these results demonstrate that the proposed parallelization strategies do not compromise the accuracy of the background subtraction operation. No single algorithm dominates unconditionally across all conditions: the Codebook method achieves superior overall quality on real-world sequences, while the GMM offers greater sensitivity and competitive performance on synthetic data. This complementarity motivates the config-

urable algorithm selection strategy embedded in the proposed heterogeneous framework, which allows either method to be deployed according to application requirements without modifying the underlying CPU–GPU partitioning logic.

4.2. Sequential Implementation

Table 3 presents the obtained performance results for medium-resolution input frames using the sequential implementation of both algorithms, as well as the improvements achieved through different compiler optimization levels. These results show that the optimization level -O3, which applies software pipelining, achieves the highest execution efficiency. Table 4 presents the FPS obtained using the -O3 optimization level for both GMM and Codebook algorithms across different input frame resolutions. These measurements enable the estimation of performance acceleration when applying the proposed parallel implementation on multiple CPUs and GPUs, as described in the following sections.

Table 3. The achieved FPS using different compiler level optimizations of the sequential codes for the medium frame resolution.

Compiler Optimization Level	FPS	
	Codebook	GMM
OFF	27	27
1	152	100
2	175	111
3	196	143

Table 4. The achieved FPS using the optimized sequential codes for different frame resolutions.

Image Resolution	FPS	
	Codebook	GMM
SD (640 × 480)	196	143
HD (1280 × 720)	67	50
Full-HD (1920 × 1080)	30	23

4.3. Acceleration Results Using Multiple CPUs

Five OpenMP scheduling strategies have been evaluated: static, dynamic, guided, runtime, and OpenMP Orphan Directive (OOD). The experimental results (Figure 6) indicate that the guided scheduling strategy achieves the best FPS performances for both the Codebook and GMM algorithms. These results highlight the importance of tailoring scheduling techniques to the characteristics of each algorithm.

Figure 7 shows the experimental results of the parallel implementation of both the GMM and Codebook BS algorithms on the multiple CPUs of the Jetson Orin Nano, using the OpenMP API to manage the scheduling across all available cores. As can be seen, the parallel implementation of the GMM algorithm reaches a maximum of 53 FPS on full-HD frames when executed across four CPU threads using the guided scheduling strategy. The optimized Codebook implementation demonstrates a slightly higher performance, achieving 69 FPS using the same scheduling technique. This performance difference can be explained by the Codebook's ability to handle data parallelism more efficiently across multiple CPU cores.

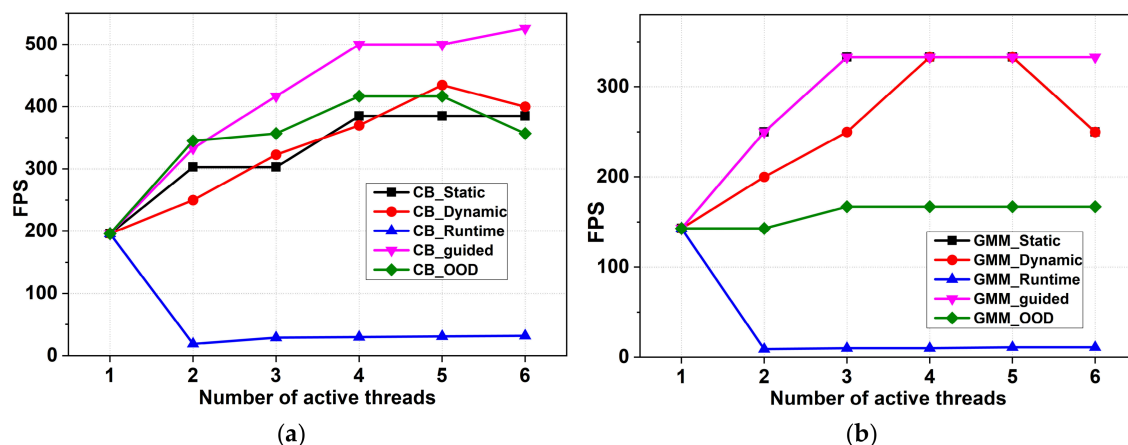


Figure 6. Test of OpenMP scheduling strategies for medium resolution on Jetson Orin Nano CPUs: (a) using Codebook; (b) using GMM.

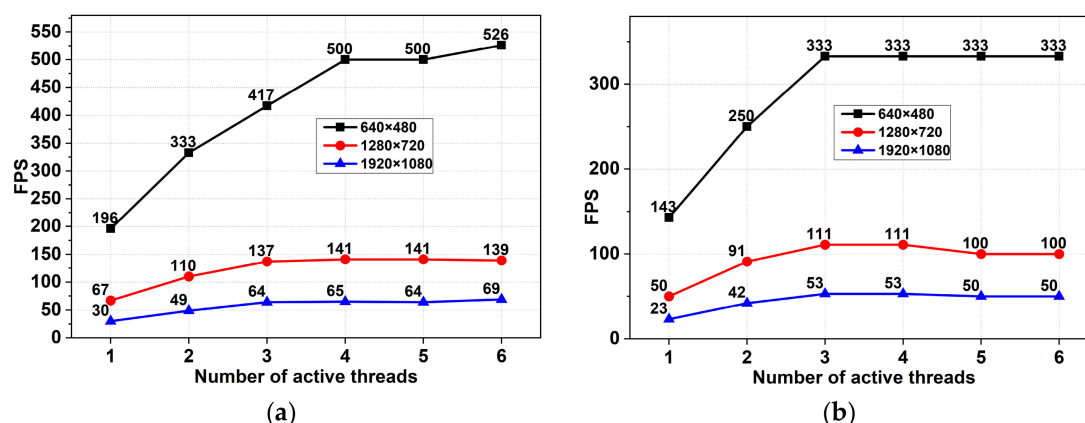


Figure 7. OpenMP results on CPUs: (a) using Codebook algorithm; (b) using GMM algorithm.

4.4. Acceleration Results Using GPUs

Figure 8 presents the obtained results of the parallel mapping of the studied BS algorithms on multiple GPU cores using the different strategies described in Section 3.2. The initial CUDA-based parallelization method shows that GMM implementation clearly outperforms the Codebook one, i.e., 116 FPS and 8 FPS for GMM and Codebook, respectively, for full-HD frame resolution.

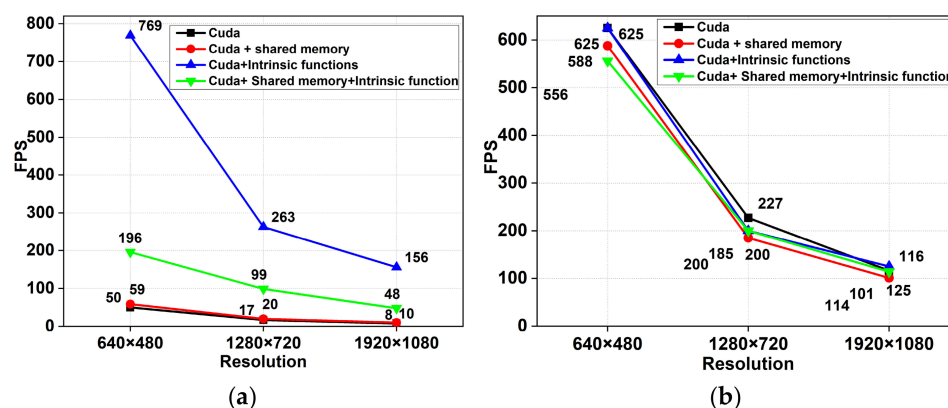


Figure 8. FPS results using proposed parallelization strategies on GPUs: (a) using Codebook algorithm; (b) using GMM algorithm.

Combining the CUDA-based parallelization method with SIMD instructions for both algorithms produces a significant acceleration (approximately $\times 15$) for Codebook and a smaller improvement for GMM. This large difference can be explained by the higher number of vectorizable operations in the Codebook pipeline compared to GMM. As a result, both algorithms achieve comparable real-time performances, i.e., 125 FPS for GMM and 156 FPS for Codebook. However, introducing shared memory and Cuda stream optimization methods provides limited acceleration (i.e., $\times 1.2$). This is primarily due to the limited shared memory capacity on the Jetson Orin Nano GPU [27,28].

4.5. Heterogeneous Implementation Result

A further performance gain was achieved using the proposed dynamic heterogeneous CPU–GPU parallelization strategy presented in Section 3.3. This approach combines, for each BS algorithm, the OpenMP scheduling technique that delivers the best CPU performances with the most effective GPU optimization method. This cooperative approach provides a significant increase in frame throughput, reaching 125 FPS for GMM and 167 FPS for Codebook for full-HD (1080p) video streams (Figure 9). These results highlight the effectiveness of the proposed adaptive workload distribution technique across CPU and GPU computing devices.

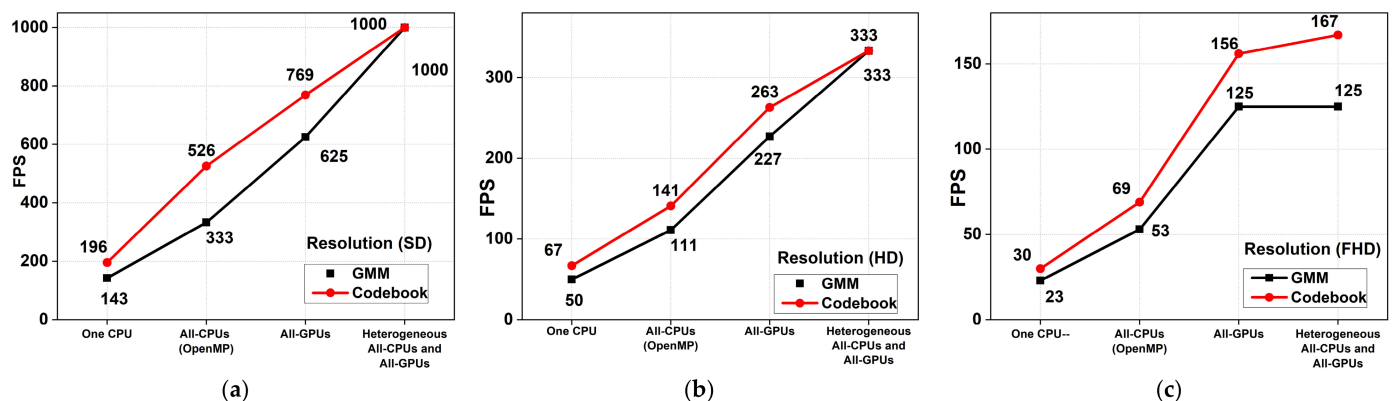


Figure 9. FPS results using the proposed CPU and GPU auto-load balancing strategy. (a) SD resolution, (b) HD resolution, (c) full-HD resolution.

4.6. Results of Power Consumption and Energy Efficiency Evaluation

Figure 10 illustrates the FPS performance for full-HD frame resolution of the four mapping proposed strategies (CPU Sequential, CPUs+OpenMP, GPU+Cuda+SIMD, and Heterogeneous CPU&GPU) across the four power modes supported by the Jetson Orin Nano platform, i.e., 7 W, 15 W, 25 W, and MAX_N. The Heterogeneous CPU&GPU strategy consistently achieves the highest throughput across all power modes, peaking at 133 FPS for GMM at 15 W power mode and 167 FPS for codebook at MAX_N power mode. Notably, there are performance plateaus beyond MAX_N, indicating that larger data volumes allow better utilization of available compute resources.

Regarding energy efficiency (FPS/Watt), the obtained results for full-HD frame resolution are presented in Figure 11. For GMM, GPU+Cuda+SIMD and Heterogeneous CPU&GPU reach their peak efficiency at 15 W, achieving 17 and 16 FPS/W respectively, identifying this power mode as the optimal operating point for HD processing. For Codebook, the peak efficiency shifts to 25 W, where both implementations attain 19 FPS/W, suggesting that Codebook workloads require a higher power budget to fully leverage available parallelism before encountering efficiency saturation. A slight decline was observed at MAX_N for both implementations, which confirms that operating at maximum power does not yield proportional performance gains.

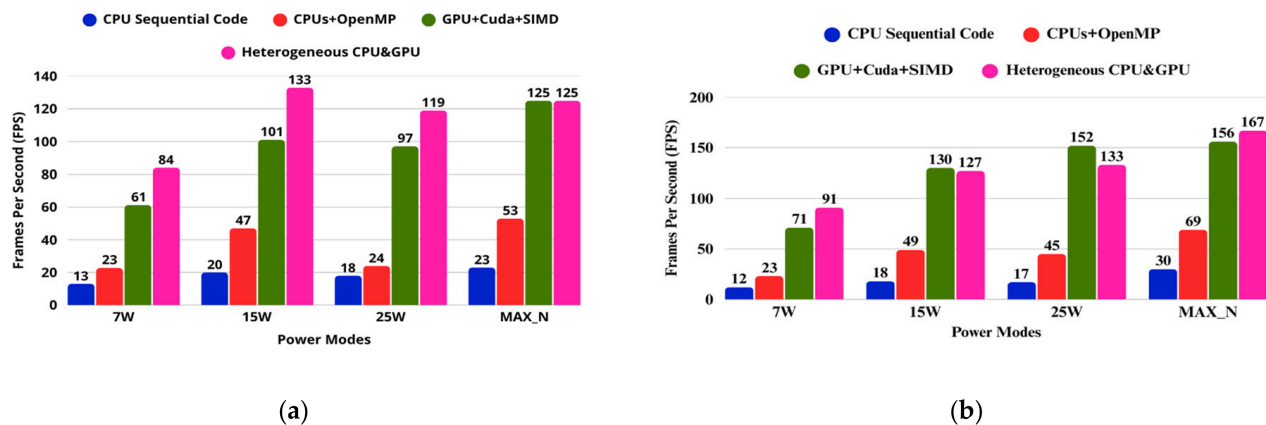


Figure 10. FPS performance for full-HD resolution in four power modes. (a) For GMM algorithm; (b) for Codebook algorithm.

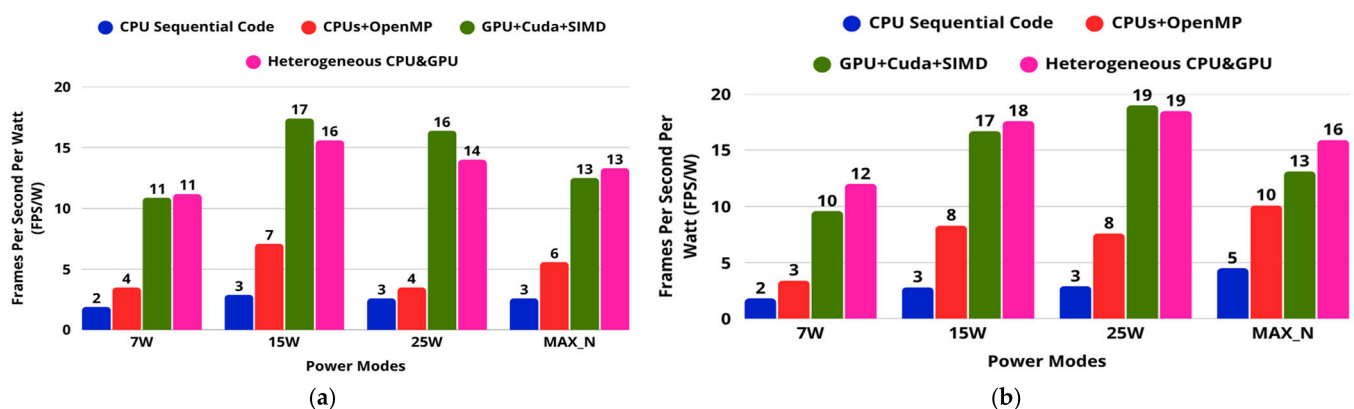


Figure 11. Performance per watt for full-HD resolution in four power modes. (a) For GMM algorithm; (b) for Codebook algorithm.

4.7. End-to-End Performance

Figure 12 provides the obtained results of the end-to-end performance, taking into consideration three intermediate stages, i.e., frame acquisition, processing, and system overhead. As can be observed, the FPS measured only for the processing step is highly impacted by intermediate stages. The end-to-end FPS obtained is 59 for GMM and 66 for Codebook.

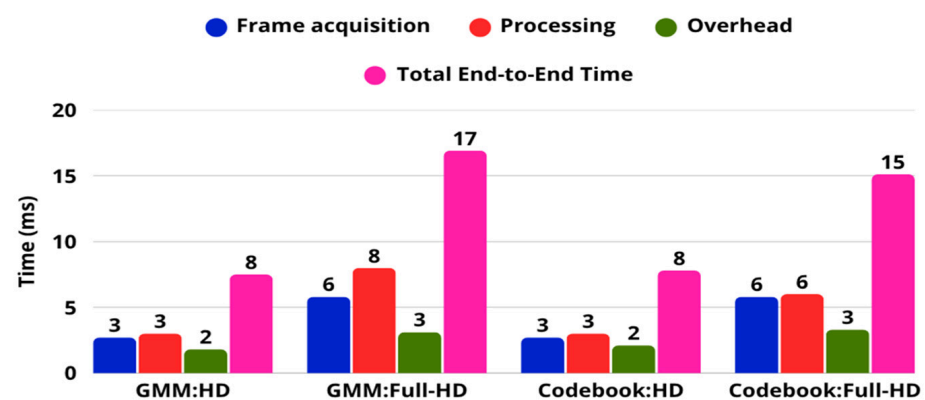


Figure 12. Time spent across different steps for HD and full-HD resolutions.

4.8. Comparison Against State-of-the-Art

Table 5 compares the obtained results in terms of end-to-end processing time with previously published works employing various platforms and parallelization techniques for BS

algorithm acceleration. As shown, the proposed heterogeneous implementation surpasses the best reported results in the literature for both Codebook and GMM, demonstrating its suitability for real-time deployment on low-power embedded systems.

Table 5. Comparative analysis of our obtained performances and related works.

Frame Resolution	Works	Platforms	FPS for GMM	FPS for Codebook
SD	Iszaidy et al. [16]	Raspberry PI	34	40
	Bariko et al. [18,19]	TI C6678 DSP	73	n/a
	This work	Jetson Orin Nano CPU&GPU	278	313
HD	Bariko et al. [18,19]	TI C6678 DSP	17.4	n/a
	Sehairi et al. [17]	Jetson TK1 Jetson Nano GPU	8 21	n/a n/a
	This work	Jetson Orin Nano CPU&GPU	133	128
Full-HD	Bariko et al. [18,19]	TI C6678 DSP	7.3	n/a
	Sehairi et al. [17]	Jetson TK1 Jetson Nano GPU	4 12.07	n/a n/a
	This work	Jetson Orin Nano CPU&GPU	59	66

4.9. Discussion and Limitations

The present work focuses on real-time background subtraction for intelligent transport vision systems on embedded hardware with limited resources. This is achieved through a heterogeneous CPU/GPU implementation of Codebook and GMM algorithms with auto-load balancing on the Jetson Orin Nano embedded platform. The main contribution was the proposition of an adaptive CPU–GPU load-balancing strategy enabling simultaneous running of both CPU and GPU processors to maximize throughput and ensure real-time performance under automotive constraints. Deep learning algorithms could provide better precision performance than that of GMM and Codebook. Future work will concern the adaptation of the proposed heterogeneous CPU/GPU implementation methods for these algorithms.

5. Conclusions

This paper addresses the challenge of achieving high-performance real-time video processing in the context of computationally intensive BS algorithms. Specifically, it investigates the limitations of conventional sequential implementations and explores the need for efficient parallelization strategies tailored to modern heterogeneous computing platforms. The main objective is to design and evaluate optimized implementations of GMM and Codebook algorithms by leveraging both multi-core CPUs and GPUs within a unified framework. To this end, a heterogeneous parallelization approach combining OpenMP and CUDA is proposed to fully exploit the complementary capabilities of CPU and GPU architectures.

The experimental results obtained on full-HD video sequences demonstrate that the proposed framework achieves significant performance gains, reaching 59 FPS for the GMM algorithm and 66 FPS for the Codebook method. These findings confirm that the proposed implementations are scalable, hardware-aware, and capable of meeting strict real-

time constraints, making them particularly suitable for video surveillance and embedded system applications.

Future work will focus on integrating advanced machine learning- and deep learning-based tracking techniques for vehicle speed estimation on low-power embedded platforms.

Author Contributions: Conceptualization, B.T., A.K. and L.E.H.; methodology, A.A. and V.Ž.; software, B.T. and M.R.; validation, A.K., A.A., L.E.H. and V.Ž.; formal analysis, A.A. and V.Ž.; writing—original draft preparation, B.T., A.K. and A.A.; writing—review and editing, M.R., L.E.H. and V.Ž.; supervision, A.K. and A.A. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The original contributions presented in this study are included in the article. Further inquiries can be directed to the corresponding author.

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

BS	Background Subtraction
ADAS	Advanced Driver-Assistance Systems
GMM	Gaussian Mixture Models
FPS	Frame Per Second
DCB	Dynamic Codebook
OpenMP	Open Multi-Processing
API	Application Programming Interface
CPU	Central Processing Units
GPU	Graphics Processing Units
RAM	Random Access Memory
CUDA	Compute Unified Device Architecture
cuDNN	CUDA Deep Neural Network
OOD	OpenMP Orphan Directive
SIMD	Single Instruction Multiple Data

References

1. Moutakki, Z.; Ouloul, I.M.; Afdel, K.; Amghar, A. Real-Time System Based on Feature Extraction for Vehicle Detection and Classification. *Transp. Telecommun. J.* **2018**, *19*, 93–102. [CrossRef]
2. Moutakki, Z.; Ouloul, I.M.; Amghar, A.; Motahhir, S.; Afdel, K. Embedded and Real Time Vehicle Classification System with Occlusion Handling. *Multimed. Tools Appl.* **2023**, *82*, 24407–24423. [CrossRef]
3. Rodríguez, A.; Navarro, A.; Asenjo, R.; Vilches, A.; Corbera, F.; Garzarán, M. Parallel Pipeline on Heterogeneous Multi-Processing Architectures. In *2015 IEEE Trustcom/BigDataSE/ISPA*; IEEE: Piscataway, NJ, USA, 2015; Volume 3, pp. 166–171.
4. Drygala, C.; Rottmann, M.; Gottschalk, H.; Friedrichs, K.; Kurbiel, T. Background-Foreground Segmentation for Interior Sensing in Automotive Industry. *J. Math. Ind.* **2022**, *12*, 13. [CrossRef]
5. Wei, J.; Jiang, J.; Yilmaz, A. MBS-NET: A MOVING-CAMERA BACKGROUND SUBTRACTION NETWORK FOR AUTONOMOUS DRIVING. *Int. Arch. Photogramm. Remote Sens. Spat. Inf. Sci.* **2021**, *XLIII-B2-2021*, 69–76. [CrossRef]
6. Beigh, T.M. Enhanced Surveillance: Triple Background Subtraction with YOLO V8. *Int. J. Intell. Syst. Appl. Eng.* **2024**, *12*, 3843–3852.
7. Stauffer, C.; Grimson, W.E.L. Adaptive Background Mixture Models for Real-Time Tracking. In *Proceedings. 1999 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (Cat. No PR00149)*; IEEE: Piscataway, NJ, USA, 1999; pp. 246–252.
8. Prakoso, P.B.; Sari, Y. Vehicle Detection Using Background Subtraction and Clustering Algorithms. *TELKOMNIKA Telecommun. Comput. Electron. Control* **2019**, *17*, 1393. [CrossRef]
9. Shen, J.; Hao, X.; Liang, Z.; Liu, Y.; Wang, W.; Shao, L. Real-Time Superpixel Segmentation by DBSCAN Clustering Algorithm. *IEEE Trans. Image Process.* **2016**, *25*, 5933–5942. [CrossRef] [PubMed]

10. Huang, Y.-S.; Guo, J.-M.; Yang, Y.-X. SeqFeedNet: Sequential Feature Feedback Network for Background Subtraction. In *2026 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*; IEEE: Piscataway, NJ, USA, 2026.
11. Kim, K.; Chalidabhongse, T.H.; Harwood, D.; Davis, L. Real-Time Foreground–Background Segmentation Using Codebook Model. *Real-Time Imaging* **2005**, *11*, 172–185. [\[CrossRef\]](#)
12. Krungkaew, R.; Kusakunniran, W. Foreground Segmentation in a Video by Using a Novel Dynamic Codebook. In *2016 13th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON)*; IEEE: Piscataway, NJ, USA, 2016; pp. 1–6.
13. Aung, S.S.; War, N. Foreground Objects Segmentation in Videos with Improved Codebook Model. In *2019 International Conference on Advanced Information Technologies (ICAIT)*; IEEE: Piscataway, NJ, USA, 2019; pp. 161–166.
14. Liu, R.; Ruichek, Y.; El Bagdouri, M. Extended Codebook with Multispectral Sequences for Background Subtraction. *Sensors* **2019**, *19*, 703. [\[CrossRef\]](#) [\[PubMed\]](#)
15. Liu, R.; Ruichek, Y.; El Bagdouri, M. Multispectral Dynamic Codebook and Fusion Strategy for Moving Objects Detection. In *Image and Signal Processing*; El Moataz, A., Mammass, D., Mansouri, A., Nouboud, F., Eds.; Lecture Notes in Computer Science; Springer International Publishing: Cham, Switzerland, 2020; Volume 12119, pp. 35–43. ISBN 978-3-030-51934-6.
16. Iszaidy, I.; Ngadiran, R.; Ahmad, R.B.; Ramli, N.; Jais, M.I.; Vijayarveswari, V. An Analysis of Background Subtraction on Embedded Platform Based on Synthetic Dataset. *J. Phys. Conf. Ser.* **2021**, *1755*, 012042. [\[CrossRef\]](#)
17. Sehairi, K.; Chouireb, F. Implementation of Motion Detection Methods on Embedded Systems: A Performance Comparison. *Int. J. Technol.* **2023**, *14*, 510. [\[CrossRef\]](#)
18. Bariko, S.; Arsalane, A.; Klilou, A.; Abounada, A. Efficient Parallel Implementation of Gaussian Mixture Model Background Subtraction Algorithm on an Embedded Multi-Core Digital Signal Processor. *Comput. Electr. Eng.* **2023**, *110*, 108827. [\[CrossRef\]](#)
19. Bariko, S.; Klilou, A.; Abounada, A.; Arsalane, A. Improved Automated Parallel Implementation of GMM Background Subtraction on a Multicore Digital Signal Processor. *Int. J. Reconfigurable Embed. Syst.* **2024**, *13*, 552. [\[CrossRef\]](#)
20. Mabrouk, L.; Huet, S.; Houzet, D.; Belkouch, S.; Hamzaoui, A.; Zennayi, Y. Efficient Adaptive Load Balancing Approach for Compressive Background Subtraction Algorithm on Heterogeneous CPU–GPU Platforms. *J. Real-Time Image Process.* **2020**, *17*, 1567–1583. [\[CrossRef\]](#)
21. Brahim, T.; Klilou, A.; Bariko, S.; Arsalane, A. Efficient Implementation of Codebook Algorithm for Background Subtraction. In *2024 3rd International Conference on Embedded Systems and Artificial Intelligence (ESAI)*; IEEE: Piscataway, NJ, USA, 2024; pp. 1–5.
22. Friedman, N.; Russell, S. Image Segmentation in Video Sequences: A Probabilistic Approach. *arXiv* **2013**, arXiv:1302.1539.
23. Jetson-Orin-Datasheet-Nano-Developer-Kit-3575392-R2.Pdf. Available online: <https://nvdam.widen.net/s/zkfqjmtds2/jetson-orin-datasheet-nano-developer-kit-3575392-r2> (accessed on 11 June 2026).
24. Dagum, L.; Menon, R. OpenMP: An Industry Standard API for Shared-Memory Programming. *IEEE Comput. Sci. Eng.* **1998**, *5*, 46–55. [\[CrossRef\]](#)
25. How Good Is OpenMP-Mattson-2003-Scientific Programming-Wiley Online Library. Available online: https://onlinelibrary.wiley.com/doi/abs/10.1155/2003/124373?__cf_chl_f_tk=zsFBRjg2WT3UoXZDYxGx2CefSPRK.KJwMDJUUS1H5xE-1782962776-1.0.1.1-eGuqvIhMeh0ctEZxPksLkR6abRvJBjbZt.VfGrZYjo (accessed on 26 June 2026).
26. Chapman, B.; Jost, G.; van der Pas, R. *Using OpenMP: Portable Shared Memory Parallel Programming*; Scientific and Engineering Computation; MIT Press: Cambridge, UK, 2008; ISBN 978-0-262-53302-7.
27. CUDA C++ Programming Guide. Available online: <https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html> (accessed on 27 October 2024).
28. CUDA C++ Best Practices Guide. Available online: <https://docs.nvidia.com/cuda/cuda-c-best-practices-guide/index.html> (accessed on 25 December 2024).
29. Wang, Y.; Jodoin, P.-M.; Porikli, F.; Konrad, J.; Benezeth, Y.; Ishwar, P. CDnet 2014: An Expanded Change Detection Benchmark Dataset. In *2014 IEEE Conference on Computer Vision and Pattern Recognition Workshops*; IEEE: Piscataway, NJ, USA, 2014; pp. 393–400.
30. Vacavant, A.; Chateau, T.; Wilhelm, A.; Lequière, L. A Benchmark Dataset for Outdoor Foreground/Background Extraction. In *Computer Vision—ACCV 2012 Workshops*; Park, J.-I., Kim, J., Eds.; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2013; Volume 7728, pp. 291–300. ISBN 978-3-642-37409-8.
31. Porr, W.; Easton, J.; Tavakkoli, A.; Loffredo, D.; Simmons, S. GPU Accelerated Non-Parametric Background Subtraction. In *Advances in Visual Computing*; Bebis, G., Boyle, R., Parvin, B., Koracin, D., Turek, M., Ramalingam, S., Xu, K., Lin, S., Alsallakh, B., Yang, J., et al., Eds.; Lecture Notes in Computer Science; Springer International Publishing: Cham, Switzerland, 2018; Volume 11241, pp. 629–639. ISBN 978-3-030-03800-7.

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.