ELEKTROTEHNIŠKI VESTNIK 93(3): 132–141, 2026
ORIGINAL SCIENTIFIC PAPER

Modeling and Experimental Evaluation of CPU-GPU Crossover Points in OpenCL-based Image Processing

Eldar Osmanović[1], Emir Skejić[1], Almir Karabegović[2]

[1]*Faculty of Electrical Engineering, University of Tuzla, Bosnia and Herzegovina*
E-mail: {eldar.osmanovic, emir.skejic}@fet.ba
[2]*Faculty of Electrical Engineering, University of Sarajevo, Bosnia and Herzegovina*
E-mail: akarabegovic@etf.unsa.ba

Abstract. GPU acceleration is widely used in image processing due to its ability to exploit massive data parallelism. However, the actual performance gain strongly depends on the problem size and hardware characteristics, particularly on integrated GPU architectures where initialization and memory overheads may dominate the total execution time. In this work, we propose a simple execution time model that decomposes the GPU runtime into initialization, memory transfer, and kernel execution components. Using the model, we experimentally evaluate CPU and GPU implementations of representative local image processing algorithms and formally define CPU-GPU crossover point N^* as the minimum problem size for which the GPU execution becomes beneficial in terms of the total runtime. The experimental analysis is conducted over a wide range of the input sizes, with repeated measurements and statistical evaluation to ensure the result stability. We show that the crossover point strongly depends on both the hardware configuration and algorithmic computational structure. Furthermore, we investigate the impact of the memory access optimization through the local (shared) memory usage within OpenCL kernels and analyze how reducing the global memory traffic affects the kernel runtime and shifts the crossover point. The results provide a quantitative insight into when the GPU acceleration is practically advantageous on integrated systems and demonstrate that the memory access patterns play a significant role in determining the effective performance boundary between the CPU and GPU execution.

Keywords: GPU acceleration, OpenCL, image processing, performance modeling, crossover point

Modeliranje in eksperimentalna analiza prelomne točke med CPE in GPE pri slikovni obdelavi z OpenCL

Pospeševanje slikovne obdelave z grafičnimi procesorji (GPE) je zaradi njihove zmožnosti izkoriščanja masivnega podatkovnega paralelizma široko uveljavljeno. Dejanski pospešek pa je močno odvisen od velikosti problema in značilnosti strojne opreme, zlasti pri sistemih z integrirano grafično enoto, kjer lahko čas inicializacije in stroški prenosa podatkov predstavljajo pomemben delež celotnega časa izvajanja. V tem delu predlagamo preprost model časa izvajanja, ki čas izvajanja na GPE razdeli na čas inicializacije, prenosa podatkov in izvajanja jedra. Na podlagi modela eksperimentalno ovrednotimo izvedbi izbranih algoritmov lokalne slikovne obdelave na CPE in GPE ter formalno določimo prelomno točko med CPE in GPE (CPU–GPU crossover point, N) kot najmanjšo velikost problema, pri kateri je izvajanje na GPE glede na skupni čas izvajanja hitrejše kot na CPE. Eksperimentalno analizo izvedemo na širokem območju velikosti vhodnih podatkov, pri čemer meritve večkrat ponovimo in statistično ovrednotimo, da zagotovimo zanesljivost rezultatov. Pokažemo, da je prelomna točka močno odvisna tako od konfiguracije strojne opreme kot tudi od računske zahtevnosti obravnavanega algoritma. Poleg tega preučimo vpliv optimizacije dostopa do pomnilnika z uporabo lokalnega (deljenega) pomnilnika v jedrih OpenCL ter analiziramo, kako zmanjšanje dostopov do globalnega pomnilnika vpliva na čas izvajanja jedra in premakne prelomno točko. Rezultati podajajo kvantitativni vpogled v pogoje, pri katerih je pospeševanje z GPE na integriranih sistemih dejansko smiselno, ter potrjujejo, da imajo vzorci dostopa do pomnilnika pomembno vlogo pri določanju učinkovite meje zmogljivosti med izvajanjem na CPE in GPE.

Received 22 May 2026
Accepted 23 June 2026

1 INTRODUCTION

Although the GPU acceleration has been widely adopted for image processing due to its ability to exploit massive data parallelism [1], its effectiveness strongly depends on the problem size and system architecture. In particular, empirical analyses have shown that the GPU execution introduces non-negligible overheads related to initialization and memory transfers, which may dominate the total execution time for small input sizes [2]. Recent work has revisited how to quantify and compare the CPU and GPU performance, introducing new metrics such as Peak Ratio Crossover (PRC) and Peak-to-Peak Ratio (PPR) to better characterize the point at which the GPU performance overtakes the CPU performance across workloads [3]. Other studies investigate how the GPU performance scales with the problem size in prac-

tical algorithmic contexts, providing insights into how the architectural characteristics and workload structuring affect the algorithmic throughput [4]. Furthermore, the problem of an optimal workload placement on heterogeneous CPU-GPU systems is highlighted as a key factor affecting the execution performance, particularly where dynamic decisions between CPU and GPU execution paths are necessary.

The execution times of the CPU and GPU implementations are modeled, and the crossover point at which the GPU acceleration becomes beneficial is experimentally identified. By decomposing the GPU execution time into initialization, memory transfer, and kernel execution components, a quantitative analysis is made of the performance trends and the impact of the algorithmic complexity and problem size on the crossover behavior on integrated GPUs is shown.

2 Related Work

Research into heterogeneous computing has extensively investigated the performance characteristics of GPUs in data-parallel workloads. Early studies primarily emphasized raw speedups over CPU baselines, highlighting the throughput potential of massively parallel architectures [5]. However, subsequent work demonstrated that the reported acceleration factors strongly depend on the workload structure, memory behavior, and architectural constraints.

Comparative analyses of the CPU and GPU execution have shown that the performance gains are highly workload-dependent and do not scale uniformly across the problem sizes. In particular, Lee et al. [6] demonstrate that large headline speedups often diminish under careful architectural evaluation, emphasizing the importance of a balanced and methodologically rigorous comparison between platforms.

Beyond direct benchmarking, analytical performance modeling has become an important tool for understanding the GPU behavior. Hong and Kim [7] propose an analytical model capturing the memory-level and thread-level parallelism effects in the GPU architectures, illustrating how architectural factors affect the scalability. Similarly, Baghsorkhi et al. [8] develop adaptive performance modeling techniques for GPUs, combining the empirical measurement with the architectural awareness to improve the predictive capability.

A more recent research further highlights the challenge of workload placement in heterogeneous systems, where execution decisions between CPU and GPU consider both the computational intensity and the data movement overhead [9]. These studies reinforce the observation that the crossover behavior is not purely hardware-driven, but emerges from the interaction between the algorithmic complexity, memory transfers, and fixed overhead components.

While the prior work either focuses on empirical benchmarking or on architecture-aware modeling, fewer studies provide a unified parametric formulation that simultaneously models the CPU execution, GPU memory overhead, and kernel scaling under statistically grounded parameter estimation. In contrast, our work integrates a linear execution-time model for both the CPU and GPU paths and employs a weighted least squares estimation to obtain analytically derived crossover thresholds. This approach enables a quantitative operator-level comparison under identical hardware constraints and provides a predictive framework for a crossover analysis on integrated GPU platforms.

3 Performance Model and Crossover Analysis

3.1 Execution Time Model

Let N denote the number of pixels in the input image. For the sequential CPU implementation, the total execution time can be modeled as

$$T_{\text{CPU}}(N) = T_{\text{comp}}(N), \tag{1}$$

where $T_{\text{comp}}(N)$ is the computation time required to process all pixels.

For the GPU implementation, the total execution time is composed of several distinct components:

$$T_{\text{GPU}}(N) = T_{\text{init}} + T_{\text{mem}}(N) + T_{\text{kernel}}(N), \tag{2}$$

where T_{init} is the initialization overhead of the GPU execution environment, $T_{\text{mem}}(N)$ is memory transfer and buffer management costs, and $T_{\text{kernel}}(N)$ is the execution time of the GPU kernel.

Initialization term T_{init} is assumed to be largely independent of the input size, while both $T_{\text{mem}}(N)$ and $T_{\text{kernel}}(N)$ grow with the number of the processed pixels. For small input sizes, the constant overhead term can dominate the total GPU execution time, whereas for larger inputs, the kernel execution time becomes the dominant component.

3.2 Crossover Point Definition

Based on the execution time models, the CPU-GPU crossover point is defined as smallest input size N^* for which the total GPU execution time becomes lower than the corresponding CPU execution time:

$$N^* = \min \{N \mid T_{\text{GPU}}(N) < T_{\text{CPU}}(N)\}. \tag{3}$$

Crossover point N^* represents the problem size beyond which the GPU acceleration becomes beneficial in terms of the total execution time.

Value of N^* depends not only on the hardware characteristics, but also on the computational structure of the algorithm. Algorithms with a higher arithmetic complexity per pixel tend to amortize the GPU overhead

more effectively, leading to a lower crossover point, particularly for local image processing operators. In this work, two representative local image processing algorithms are considered in order to analyze how different computational patterns affect the crossover behavior.

3.3 Linear Parametric Model

To enable a quantitative estimation of the crossover point, the execution time functions are further approximated using linear models.

For the CPU implementation, the total execution time can be expressed as

$$T_{\text{CPU}}(N) = \alpha_c N + \beta_c, \tag{4}$$

where α_c is the per-pixel computational cost and β_c captures the constant overhead effects such as the function call overhead and measurement artifacts.

For the GPU implementation, each component of (2) is modeled separately. The initialization term is assumed to be constant:

$$T_{\text{init}} = \text{const}. \tag{5}$$

The memory transfer and kernel execution terms are approximated as linear functions of N:

$$T_{\text{mem}}(N) = \alpha_m N + \beta_m, \tag{6}$$

$$T_{\text{kernel}}(N) = \alpha_k N + \beta_k. \tag{7}$$

Substituting these expressions into (2), the total GPU execution time becomes

$$T_{\text{GPU}}(N) = (\alpha_m + \alpha_k)N + (T_{\text{init}} + \beta_m + \beta_k). \tag{8}$$

For compactness, we define

$$\alpha_g = \alpha_m + \alpha_k, \tag{9}$$

$$\beta_g = T_{\text{init}} + \beta_m + \beta_k, \tag{10}$$

which yields the simplified linear form

$$T_{\text{GPU}}(N) = \alpha_g N + \beta_g. \tag{11}$$

Coefficients α_c and α_g are the effective per-pixel processing cost on CPU and GPU, respectively, while β_c and β_g capture the constant overhead contributions.

Under the linear model assumptions, the crossover point can be obtained analytically by solving

$$\underbrace{\alpha_g N + \beta_g}_{T_{\text{GPU}}(N)} < \underbrace{\alpha_c N + \beta_c}_{T_{\text{CPU}}(N)}. \tag{12}$$

Rearranging the terms gives

$$N > \frac{\beta_g - \beta_c}{\alpha_c - \alpha_g}. \tag{13}$$

Therefore, the theoretical crossover point can be estimated as

$$N^* = \left\lceil \frac{\beta_g - \beta_c}{\alpha_c - \alpha_g} \right\rceil, \tag{14}$$

provided that $\alpha_c > \alpha_g$.

4 Experimental Setup

4.1 Hardware Platform

All experiments are conducted on a system equipped with an Intel multi-core CPU and an Intel integrated GPU. Since the integrated GPU shares the main system memory with CPU, this platform is suitable for analyzing the impact of the initialization overhead and memory-transfer/buffer-management costs on the end-to-end GPU performance.

The hardware configuration used in the experiments is summarized as follows:

- **CPU**: Intel Core i7-10610U CPU @ 1.80GHz
- **Integrated GPU**: Intel UHD Graphics (CometLake-U GT2)
- **RAM**: 32 GB DDR4 (2 × 16 GB, 2667 MT/s configured)

4.2 Software Environment

The CPU baseline implementation is written in C++ (C++17), the GPU implementation is developed using OpenCL. All measurements are performed on Ubuntu Linux. The OpenCL command queue is created with profiling enabled to allow an accurate kernel timing via OpenCL events.

The software environment is summarized as follows:

- **Operating system**: Ubuntu 24.04.3 LTS
- **Compiler**: g++ 13.3.0 (Ubuntu 13.3.0-6ubuntu2~24.04)
- **OpenCL runtime/driver**: Intel(R) OpenCL Graphics (Intel(R) Corporation, driver 23.43.027642)
- **OpenCL version**: *OpenCL 3.0*

4.3 Input Data and Problem Sizes

The evaluation is performed on grayscale images of varying resolutions. Square images with dimensions ranging from 16×16 to 4096×4096 pixels are used, covering several orders of the magnitude in problem size. The total number of the processed pixels is defined as

$$N = \text{width} \cdot \text{height}. \tag{15}$$

For each resolution, the same input image content is used (rescaled as needed) to keep the workload comparable across the problem sizes.

4.4 Algorithms

Two representative local image processing algorithms are evaluated:

- Box blur (spatial neighborhood averaging),
- Sobel edge detection (gradient-based operator).

Both algorithms are local operators that process each output pixel based on a small neighborhood in the input image, as commonly described in the classical digital image processing literature [10], making them suitable for a data-parallel execution on GPUs.

4.5 Measurement Methodology

The CPU execution time is measured using a high-resolution host-side wall-clock timer and includes the full computation time of the algorithm.
For the GPU implementation, the total execution time is decomposed according to the proposed model

$$T_{\text{GPU}}(N) = T_{\text{init}} + T_{\text{mem}}(N) + T_{\text{kernel}}(N), \qquad (16)$$

where each term is measured separately:

- T_{init}: the time taken to create the OpenCL execution environment (platform/device selection, context creation, command queue creation, and program compilation/build).
- $T_{\text{mem}}(N)$: the time taken to perform the memory-related operations (buffer creation and host-to-device/device-to-host transfers).
- $T_{\text{kernel}}(N)$: pure kernel execution time measured via OpenCL event profiling (device-side timing) [11].

The end-to-end GPU time $T_{\text{total}}(N)$ is measured using the host-side timing around the complete GPU processing routine. The residual term is computed as

$$T_{\text{other}}(N) = T_{\text{total}}(N) - T_{\text{init}} - T_{\text{mem}}(N) - T_{\text{kernel}}(N). \qquad (17)$$

It captures minor overheads and timing granularity differences between the host-side and event-based measurements.

4.6 Warm-up, Repetition, and Statistical Reporting

To reduce the impact of the cold-start effects, an initial warm-up run is executed and excluded from the reported results. For each image size and each algorithm, the experiment is then repeated K times:

$$K = 30. \qquad (18)$$

For every measured quantity (CPU time, GPU total time, and each GPU component), the sample mean and sample standard deviation are reported:

$$\mu = \frac{1}{K}\sum_{i=1}^{K} t_i, \qquad \sigma = \sqrt{\frac{1}{K-1}\sum_{i=1}^{K}(t_i - \mu)^2}. \qquad (19)$$

This provides a measure of both the typical performance and the measurement variability for each problem size. To eliminate one-time overheads, such as kernel compilation and runtime initialization, an initial warm-up execution is performed prior to the timed measurements. Each experiment is repeated multiple times, and the reported results represent the average execution time.

4.7 Parameter Estimation via Weighted Least Squares

The coefficients of the linear models are estimated using a weighted least-squares (WLS) regression. For each problem size N_i, the measured execution time is represented by its sample mean μ_i and associated standard deviation σ_i.
Since the variability of the execution time may differ across problem sizes, the weighted regression is employed in order to account for heteroscedasticity. The weight assigned to each measurement point is defined as

$$w_i = \frac{1}{\sigma_i^2}. \qquad (20)$$

The optimal model parameters are obtained by minimizing the weighted sum of the squared residuals:

$$\min_{\alpha,\beta} \sum_i w_i \left(\mu_i - (\alpha N_i + \beta)\right)^2. \qquad (21)$$

The procedure is applied separately to the CPU and GPU total execution time data in order to estimate coefficients (α_c, β_c) and (α_g, β_g), respectively.
The use of WLS ensures that measurement points with a lower variance contribute more strongly to the parameter estimation, leading to a statistically consistent estimate of the per-pixel cost and constant overhead terms.

4.8 Output Data

All measurements are stored in a CSV file containing, for each algorithm and input size, the mean and standard deviation of the CPU time, as well as the mean and standard deviation of the GPU timing components (T_{init}, T_{mem}, T_{kernel}, T_{total}, and T_{other}). The results are used in the next section to validate the time model and to determine the CPU-GPU crossover point.

5 Results and Discussion

5.1 Validation of the Execution Time Model

The first objective of the experimental evaluation is to validate proposed execution time model 2 by analyzing the measured timing components across different input sizes.
Initialization Time: Figure 1 shows measured mean initialization time T_{init} as a function of problem size N. The results indicate that the initialization time remains approximately constant across all tested resolutions, with only minor fluctuations within the range of measurement variability. The standard deviation remains small relative to the mean value, confirming that T_{init} can be reasonably modeled as a constant term independent of N.
This observation empirically supports the modeling assumption introduced in Section 3.1 and confirms that the initialization overhead does not scale with the problem size on the tested integrated GPU platform.
Although slight variations of T_{init} can be observed for different input sizes, no monotonic trend or systematic growth with respect to N is present. The observed fluctuations remain within the confidence interval defined by one standard deviation and are likely attributable to

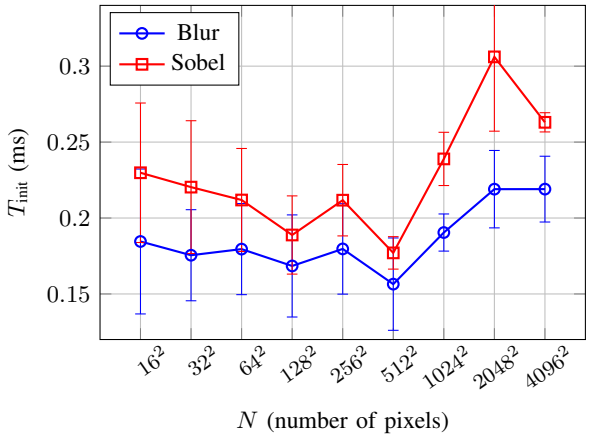

Figure 1. Mean initialization time T_{init} as a function of problem size N (error bars indicate one standard deviation).

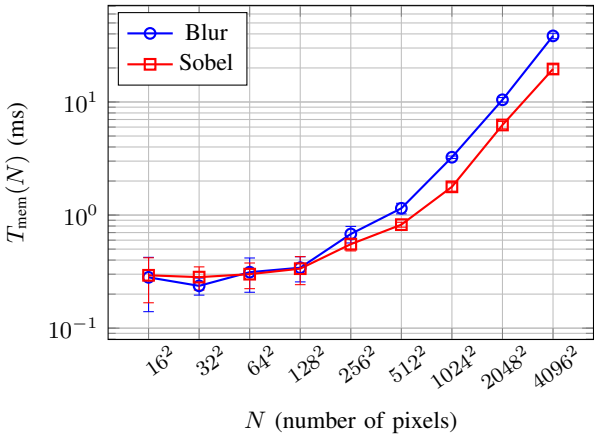

Figure 2. Mean memory-related time $T_{\text{mem}}(N)$ as a function of problem size N (error bars indicate one standard deviation).

operating system scheduling effects, driver-level behavior, and runtime variability in OpenCL context creation and program compilation.

Importantly, the magnitude of the variations is significantly smaller than the scaling observed for $T_{\text{mem}}(N)$ and $T_{\text{kernel}}(N)$, which exhibits a clear growth with the increasing problem size. Therefore, from the modeling perspective, T_{init} can be treated as a constant term without introducing any significant approximation error.

To further quantify the dependence between the problem size and initialization time, the Pearson correlation coefficient between N and T_{init} is computed. The obtained values are $r_{\text{blur}} \approx 0.54$ and $r_{\text{sobel}} \approx 0.61$, indicating only a weak to moderate linear association.

Despite this mild positive correlation, the absolute variation of T_{init} remains confined to a narrow interval (approximately 0.16-0.31 ms) and does not exhibit a proportional growth with respect to N. In contrast, both $T_{\text{mem}}(N)$ and $T_{\text{kernel}}(N)$ demonstrate a clear scaling behavior. Therefore, from the modeling perspective, the initialization time can still be reasonably approximated as a constant term without significantly affecting the accuracy of the execution time model.

Memory and Kernel Scaling: Figures 2 and 3 present the measured mean memory transfer time $T_{\text{mem}}(N)$ and kernel execution time $T_{\text{kernel}}(N)$ as functions of N.

Both components exhibit an approximately linear growth with respect to the number of the processed pixels. For the small problem sizes, the memory component dominates the GPU execution time, while for the larger images the kernel execution time becomes increasingly significant.

To quantitatively validate the linear assumption, the weighted least squares regression is applied to the measured data. The resulting fits show a very high goodness-of-fit (R^2 close to 1), confirming that both $T_{\text{mem}}(N)$ and $T_{\text{kernel}}(N)$ can be reliably approximated by the first-

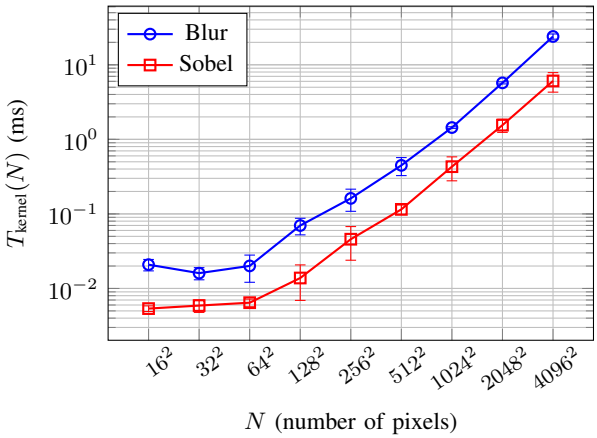

Figure 3. Mean kernel execution time $T_{\text{kernel}}(N)$ as a function of problem size N (error bars indicate one standard deviation).

order linear models:

$$T(N) \approx aN + b, \tag{22}$$

where a is the effective per-pixel processing cost and b captures the constant overhead contributions.

A more detailed inspection of the curves reveals that the memory-related component exhibits a stronger scaling than the kernel execution time for larger input sizes. In particular, $T_{\text{mem}}(N)$ grows steadily across several orders of the magnitude in N, indicating that the buffer management and data movement constitute a significant portion of the total GPU execution time, even on an integrated architecture where CPU and GPU share the system memory.

For the smaller problem sizes, the memory component clearly dominates the kernel execution time. However, as N increases, $T_{\text{kernel}}(N)$ begins to grow more rapidly, especially for the blur operator, whose arithmetic workload per pixel is higher than that of the Sobel operator. This

difference reflects the distinct computational intensity of the evaluated algorithms.
The estimated linear model parameters obtained via the weighted least-squares regression are summarized in Table 1.

Table 1. Estimated linear model coefficients obtained via the weighted least-squares (WLS) regression for BLUR and SOBEL operators.

Component	α **(ms/px)**	β **(ms)**	R^2
	BLUR		
CPU	1.718764×10^{-4}	0.254418	0.998120
Memory	2.531334×10^{-6}	0.436511	0.990245
Kernel	1.361912×10^{-6}	0.020988	0.998202
GPU Total	3.893246×10^{-6}	0.643348	–
	SOBEL		
CPU	7.287088×10^{-5}	0.154956	0.999181
Memory	1.298430×10^{-6}	0.432453	0.973792
Kernel	3.974216×10^{-7}	0.004875	0.989093
GPU Total	1.695852×10^{-6}	0.664847	–

In both algorithms, the weighted coefficient of the determination remains above 0.97, confirming a strong linear dependence between the execution time and input size within the analyzed range.
These observations empirically validate the proposed linear approximation model and confirm that the dominant scaling behavior of the GPU execution time is driven by the per-pixel processing cost and memory traffic, rather than hidden constant overhead terms.
The estimated coefficients in Table 1 provide several important insights into the performance characteristics of both operators.
First, for both blur and Sobel, the weighted R^2 values exceed 0.97, confirming that the execution time exhibits a strong linear dependence on the number of the processed pixels within the analyzed range. This validates the linear parametric model assumed in Section 3.3.
Comparing the per-pixel coefficients, it can be observed that CPU slope α_c is significantly larger than corresponding GPU slope α_g in both cases. For the blur operator, α_c is nearly two orders of the magnitude larger than α_g, indicating a substantially higher per-pixel computational cost on the CPU. A similar trend is observed for Sobel, although the absolute CPU slope is lower due to the reduced arithmetic intensity.
The intercept terms reveal a different behavior. While CPU intercept β_c remains relatively small, GPU intercept β_g is dominated by memory-related overhead (β_m) and initialization cost. This explains why GPU execution is less efficient for small problem sizes despite its significantly lower per-pixel cost.
Furthermore, the difference between blur and Sobel highlights the impact of the algorithmic structure. The blur operator exhibits a higher CPU slope, reflecting its heavier computational workload per pixel. As a result, the GPU advantage becomes more pronounced compared to Sobel, where the lower arithmetic intensity reduces the relative performance gap.
Overall, the results confirm that the GPU acceleration primarily improves the asymptotic per-pixel processing cost, while constant overhead terms determine the crossover behavior for small input sizes.

Consistency of the Decomposition: To further validate the proposed timing model, the component-wise estimate

$$T_{\text{init}} + T_{\text{mem}}(N) + T_{\text{kernel}}(N)$$

is compared against the directly measured end-to-end GPU execution time $T_{\text{total}}(N)$. The residual term is defined as

$$T_{\text{other}}(N) = T_{\text{total}}(N) - T_{\text{init}} - T_{\text{mem}}(N) - T_{\text{kernel}}(N). \tag{23}$$

For small problem sizes, $T_{\text{other}}(N)$ remains close to zero, indicating that the additive decomposition captures the dominant contributions well. In this regime, any small non-zero residual can be attributed to the measurement granularity and instrumentation overhead, in particular the difference between the host-side timing (CPU wall-clock) and the device-side event timing, as well as unavoidable synchronization and driver/runtime overheads that are difficult to isolate perfectly.
However, as the image size increases, a clear systematic trend emerges: the magnitude $|T_{\text{other}}(N)|$ grows with N, and the residual becomes increasingly negative. This behavior implies that

$$T_{\text{init}} + T_{\text{mem}}(N) + T_{\text{kernel}}(N) \;>\; T_{\text{total}}(N)$$

for larger problem sizes. Such an outcome cannot be explained by a simple timing noise alone, and instead points to a limitation of the strictly additive model. The most plausible explanation is increasing the overlap between the GPU activities.
Modern GPU runtimes can pipeline the work such that the memory transfers and kernel execution partially overlap (e.g., through asynchronous copies, DMA engines, and concurrent execution) [12], especially when the workload is large enough to fill the pipeline. In that case, the effective end-to-end time is no longer the sum of transfer and compute times, but closer to the time of the critical path. A more realistic model for large N is therefore

$$T_{\text{total}}(N) \approx T_{\text{init}} + \max\big(T_{\text{mem}}(N),\, T_{\text{kernel}}(N)\big) + \delta(N), \tag{24}$$

where $\delta(N)$ are smaller secondary effects (e.g., the launch overhead, occasional synchronization, and measurement artifacts). Under an increasing overlap, the additive estimate overcounts the work that happens concurrently, which naturally produces a negative residual

whose absolute value grows with N. Such a max-based formulation is consistent with the critical-path performance modeling commonly used in the GPU performance analysis [13].

In summary, the decomposition is approximately additive for small problem sizes, confirming that T_{init}, $T_{\text{mem}}(N)$, and $T_{\text{kernel}}(N)$ are the dominant components. For larger images, the growing negative residual provides evidence of the pipelining/overlap, and motivates the use of a critical-path (max-based) model rather than a purely additive one.

The detailed numerical results supporting this analysis are presented in Table 2. For large problem sizes, the rows corresponding to the highest input resolutions (highlighted in the table) clearly show that the additive model of (2) increasingly overestimates the measured execution time for both the blur and Sobel operators. In these cases, the model prediction $T_{\text{init}} + T_{\text{mem}}(N) + T_{\text{kernel}}(N)$ exceeds the directly measured $T_{\text{total}}(N)$ by a growing margin, consistent with the negative residual values discussed above.

In contrast, the max-based formulation

$$T_{\text{total}}(N) \approx T_{\text{init}} + \max\big(T_{\text{mem}}(N),\, T_{\text{kernel}}(N)\big) + \delta(N)$$

provides a substantially closer approximation in this regime, as it reflects the dominance of the critical path under increasing overlap between memory transfers and kernel execution. This observation further supports the interpretation that the discrepancy is structural rather than stochastic, and that the concurrent GPU activity becomes progressively more significant as N grows.

5.2 CPU vs GPU Performance Comparison

Figure 4 compares the total execution time of CPU and GPU implementations for both evaluated algorithms.

For the small input sizes, the CPU implementation consistently outperforms the GPU due to the initialization and memory overhead associated with the GPU execution. However, as the problem size increases, the GPU becomes increasingly advantageous due to its ability to exploit data-level parallelism.

The intersection point between the CPU and GPU performance curves corresponds to the experimentally observed crossover region.

5.3 Detailed Timing Breakdown

All reported results are presented as mean values obtained over repeated experimental runs. The accompanying standard deviation quantifies the natural variability of execution time arising from system-level factors, such as operating system scheduling, memory hierarchy effects, runtime fluctuations, and background processes. Table 2 provides a comprehensive breakdown of the measured quantities. In addition to the CPU execution time, the table reports the individual GPU components (initialization, memory transfer, and kernel execution),

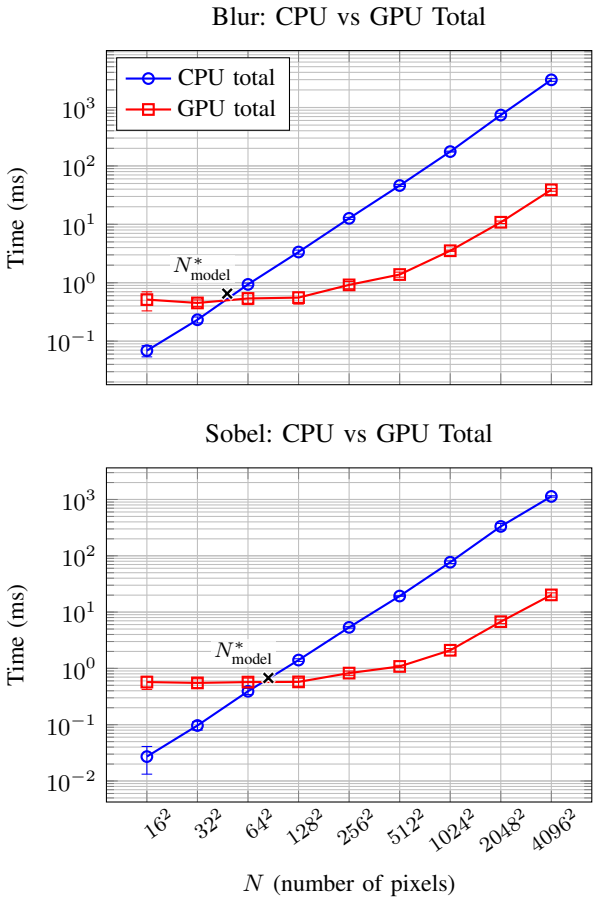

Figure 4. Total execution time comparison between the CPU and GPU implementations as a function of problem size N for (top) blur and (bottom) Sobel. Error bars indicate one standard deviation.

the model-based aggregate estimate (sum of components), the directly measured total GPU time, and residual term $T_{\text{other}}(N)$. This detailed presentation enables a direct comparison between the additive model and the measured end-to-end execution time, and allows the scaling behavior and deviation patterns to be examined quantitatively across all tested input sizes.

5.4 Crossover Point Estimation

CPU-GPU crossover point N^* is defined as the smallest input size for which $T_{\text{GPU}}(N) < T_{\text{CPU}}(N)$.

For the box blur operator, the measurements show that the crossover occurs between $N = 32^2$ and $N = 64^2$ pixels. For the Sobel operator, the crossover region is shifted toward larger input sizes, occurring between $N = 64^2$ and $N = 128^2$ pixels.

This difference indicates that the crossover point depends not only on the hardware characteristics (fixed GPU overheads such as initialization and transfers), but also on the algorithmic work per pixel. In our measurements, blur exhibits a higher GPU kernel time

Table 2. Detailed timing breakdown (mean ± std, in ms). "Model" denotes $T_{\text{init}} + T_{\text{mem}} + T_{\text{kernel}}$, while "GPU Total Measured" is the directly measured GPU execution time.

Alg.	N	CPU	GPU Components				GPU Total Measured	T_{other}
		Mean ± Std	T_{init}	T_{mem}	T_{kernel}	Model	Mean ± Std	Mean ± Std
Blur	16^2	0.069±0.015	0.185±0.048	0.281±0.141	0.021±0.004	0.487	0.514±0.185	0.028±0.019
	32^2	0.232±0.027	0.176±0.030	0.237±0.041	0.016±0.003	0.429	0.451±0.057	0.023±0.009
	64^2	0.937±0.051	0.180±0.030	0.313±0.105	0.020±0.008	0.513	0.538±0.116	0.026±0.019
	128^2	3.357±0.264	0.168±0.034	0.342±0.086	0.070±0.018	0.580	0.557±0.126	-0.024±0.027
	256^2	12.653±0.513	0.180±0.030	0.680±0.114	0.162±0.053	1.022	0.919±0.131	-0.103±0.059
	512^2	46.175±1.671	0.156±0.030	1.148±0.123	0.448±0.121	1.752	1.382±0.137	-0.370±0.128
	1024^2	175.86±3.18	0.190±0.012	3.246±0.079	1.440±0.036	4.876	3.529±0.084	-1.347±0.040
	2048^2	742.45±53.32	0.219±0.025	10.458±0.488	5.725±0.184	16.402	10.853±0.519	-5.548±0.178
	4096^2	2976.81±173.17	0.219±0.022	38.407±2.135	23.985±1.920	62.611	38.988±2.149	-23.623±1.921
Sobel	16^2	0.027±0.014	0.230±0.046	0.293±0.126	0.005±0.0005	0.528	0.572±0.148	0.043±0.011
	32^2	0.097±0.017	0.220±0.044	0.283±0.066	0.006±0.001	0.509	0.552±0.101	0.043±0.014
	64^2	0.392±0.048	0.212±0.034	0.300±0.077	0.006±0.001	0.518	0.571±0.100	0.053±0.019
	128^2	1.405±0.077	0.189±0.026	0.336±0.093	0.014±0.007	0.539	0.576±0.094	0.037±0.019
	256^2	5.350±0.162	0.212±0.024	0.553±0.071	0.046±0.022	0.811	0.825±0.071	0.014±0.028
	512^2	19.271±0.131	0.177±0.011	0.823±0.041	0.115±0.017	1.115	1.082±0.051	-0.032±0.020
	1024^2	76.909±4.152	0.239±0.018	1.777±0.149	0.430±0.152	2.446	2.098±0.149	-0.348±0.161
	2048^2	332.204±42.76	0.306±0.049	6.261±0.472	1.551±0.311	8.118	6.755±0.517	-1.363±0.315
	4096^2	1133.12±33.08	0.263±0.006	19.572±1.759	6.073±1.776	25.908	20.194±1.753	-5.714±1.783

per pixel than Sobel, which allows the GPU overhead to be amortized earlier and thus yields a smaller crossover threshold, whereas Sobel requires larger inputs before the GPU becomes faster overall.

To obtain a theoretical estimate of the crossover point, we use the linear models introduced in Section 3.3:

$$T_{\text{CPU}}(N) = \alpha_c N + \beta_c,\ T_{\text{GPU}}(N) = \alpha_g N + \beta_g, \quad (25)$$

with the crossover condition

$$N^*_{\text{model}} = \left\lceil \frac{\beta_g - \beta_c}{\alpha_c - \alpha_g} \right\rceil, \qquad \text{for } \alpha_c > \alpha_g. \quad (26)$$

Box Blur: Using the estimated parameters from Table 1, the models for the blur operator are:

$$T_{\text{CPU}}(N) = 1.718764 \times 10^{-4} N + 0.254418, \quad (27)$$
$$T_{\text{GPU}}(N) = 3.893246 \times 10^{-6} N + 0.643348. \quad (28)$$

Substituting these values into the analytical expression (26) yields

$$N^*_{\text{model}} \approx 2316 \text{ pixels}. \quad (29)$$

This corresponds approximately to an image of size $\sqrt{2316} \approx 48 \times 48$ pixels.

Experimentally, the measured crossover occurs between $32^2 = 1024$ and $64^2 = 4096$ pixels, with the first observed speedup at 64×64. Therefore, the analytical prediction lies within the measured transition region, confirming a good agreement between the linear model and experimental results.

Sobel: For the Sobel operator, the estimated models are:

$$T_{\text{CPU}}(N) = 7.287088 \times 10^{-5} N + 0.154956, \quad (30)$$
$$T_{\text{GPU}}(N) = 1.695852 \times 10^{-6} N + 0.664847. \quad (31)$$

Applying the same formula (26) gives

$$N^*_{\text{model}} \approx 7164 \text{ pixels}. \quad (32)$$

This corresponds approximately to $\sqrt{7164} \approx 85 \times 85$ pixels.

Experimentally, the crossover occurs between $64^2 = 4096$ and $128^2 = 16384$ pixels, with the first consistent speedup observed at 128×128. Again, the analytical estimate falls within the empirically observed transition interval.

Overall, the close correspondence between N^*_{model} and the measured crossover region demonstrates that the proposed linear model captures both the asymptotic per-pixel cost and the constant overhead contributions with a sufficient accuracy for a practical crossover prediction.

5.5 Impact of Memory Transfers

An analysis of the relative contribution of the memory transfer time to the total GPU execution time reveals that the memory operations represent a dominant fraction of the total runtime for intermediate problem sizes.

Although the integrated GPU shares the memory with CPU, the buffer management and synchronization overhead still introduce measurable costs. As the problem size increases, the kernel execution time gradually becomes the primary contributor to the total GPU runtime. These findings suggest that optimization strategies aimed at reducing the global memory traffic, such as the use

of a local (shared) memory within the OpenCL kernels, have the potential to shift the crossover point toward smaller input sizes.

6 Limitations

Although the proposed modeling framework provides a structured and analytically grounded view of the CPU-GPU crossover behavior, several limitations should be acknowledged.

First, the experimental evaluation is conducted on a single hardware platform featuring an integrated GPU. While this setup is representative of many contemporary heterogeneous systems, the derived parameter values and crossover thresholds are architecture-dependent. Discrete GPUs with a dedicated memory and higher bandwidth may exhibit substantially different overhead characteristics and scaling behavior.

Second, the execution-time model assumes linear scaling with respect to the number of the processed pixels. Although the weighted least-squares (WLS) fitting demonstrates a high goodness-of-fit for the considered input range, non-linear effects may emerge for extremely small or extremely large problem sizes due to the cache behavior, memory contention, or scheduling effects.

Third, the analysis focuses on two representative image processing operators (box blur and Sobel). While these operators differ in the computational intensity and provide an insight into the algorithm-dependent crossover shifts, the results cannot be directly generalized to all classes of the GPU workloads, particularly those with irregular memory access patterns or complex control flow.

Finally, the study focuses on the execution time as the primary performance metric. The energy consumption, thermal constraints, and multi-threaded CPU scaling are not evaluated and may affect the practical decision of workload placement in real-world systems.

Despite these limitations, the proposed modeling approach remains applicable as a general framework for a structured crossover analysis under similar architectural constraints.

7 Conclusion

The work presents a structured performance modeling framework for analyzing the CPU-GPU crossover behavior in image processing workloads on integrated heterogeneous systems. By decomposing the GPU execution time into initialization, memory transfer, and kernel components, and by modeling both CPU and GPU execution paths using a linear parametric formulation, the study enables analytical derivation of crossover thresholds rather than relying solely on empirical observation.

The parameter estimation via weighted least squares provides statistically grounded scaling coefficients with a high goodness-of-fit, validating the suitability of the linear execution-time model across the examined input sizes. The experimental evaluation demonstrates that the crossover behavior is strongly affected by the algorithmic intensity: the box blur operator exhibits an earlier transition to the GPU advantage compared to the Sobel operator, reflecting differences in the computational workload per pixel.

The results highlight that the crossover points are not purely hardware-defined constants, but instead emerge from the interaction between the fixed overheads, memory behavior, and algorithmic complexity. Consequently, the performance-aware workload placement on heterogeneous systems requires structured modeling rather than relying exclusively on the peak throughput metrics or isolated benchmarks.

Overall, the proposed approach provides a predictive and analytically interpretable framework for crossover estimation, offering insights into the performance dynamics of integrated CPU-GPU platforms and supporting informed decisions in heterogeneous computing environments.

References

[1] Kirk, D. B. & Hwu, W.-M. W. *Programming Massively Parallel Processors: A Hands-on Approach*. 3rd ed. Morgan Kaufmann (2016)

[2] Fang, J., Varbanescu, A. L. & Sips, H. A Comprehensive Performance Comparison of CUDA and OpenCL *International Conference on Parallel Processing*. pp. 216–225 (2011)

[3] Huebler, G., et al. Actionable reporting of CPU-GPU performance comparisons: Insights from a CLUBB case study. *EGUsphere Preprint*, pp. 1–24 (2025)

[4] Yu, X., Jiang, T., Cheng, R., Jin, Y. & Tan, K. C. Scaling Behaviors of Evolutionary Algorithms on GPUs: When Does Parallelism Pay Off? *arXiv preprint arXiv:2601.18446*, pp. 1–28 (2026)

[5] Owens, J. D., Houston, M., Luebke, D., Green, S., Stone, J. E. & Phillips, J. C. GPU computing. *Proceedings of the IEEE*. **96**(5), pp. 879–899 (2008)

[6] V. W. Lee, C. Kim, J. Chhugani, M. Deisher, et al., "Debunking the 100X GPU vs. CPU myth: An evaluation of throughput computing on CPU and GPU," *ACM SIGARCH Computer Architecture News*, vol. 38, no. 3, pp. 451–460, 2010.

[7] S. Hong and H. Kim, "An analytical model for a GPU architecture with memory-level and thread-level parallelism awareness," *Proceedings of the 36th Annual International Symposium on Computer Architecture (ISCA)*, pp. 152–163, 2009.

[8] S. S. Baghsorkhi, M. Delahaye, S. J. Patel, W. D. Gropp, and W.-M. Hwu, "An adaptive performance modeling tool for GPU architectures," *ACM SIGPLAN Notices*, vol. 45, no. 5, pp. 105–114, 2010.

[9] Carvalho, M. N. L., Simitsis, A., Queralt, A., & Romero, O. Workload placement on heterogeneous CPU-GPU systems. *Proceedings of the VLDB Endowment*. **17**(12), pp. 4241–4244 (2024)

[10] Gonzalez, R. C. & Woods, R. E. *Digital Image Processing*. 4th ed., Pearson (2018).

[11] Gaster, B. R., Howes, L., Kaeli, D. R., Mistry, P. & Schaa, D. *Heterogeneous Computing with OpenCL*. Morgan Kaufmann (2012)

[12] NVIDIA Corporation, *CUDA C++ Programming Guide*, NVIDIA Developer Documentation, 2023. [Online]. Available: https://docs.nvidia.com/cuda/

[13] S. Williams, A. Waterman, and D. Patterson, "Roofline: An Insightful Visual Performance Model for Multicore Architectures," *Communications of the ACM*, vol. 52, no. 4, pp. 65–76, Apr. 2009.

Eldar Osmanović received his B.Sc. degree in Electrical Engineering from the Faculty of Electrical Engineering, University of Tuzla, Bosnia and Herzegovina, in 2025. He is currently pursuing his M.Sc. degree in Electrical Engineering at the same institution. He works as an Embedded Engineer at HTEC Group, where he is involved in the development of compilers and low-level system software. His research interests include compilers, low-level software systems, quantum computing, and GPU computing.

Emir Skejić received his B.Eng., M.Sc. and Ph.D. degrees in Electrical Engineering from the Faculty of Electrical Engineering, University of Tuzla, Bosnia and Herzegovina, in 2000, 2003 and 2007, respectively. Since 2001, he has been employed with the same faculty, where he is currently an Associate Professor in the field of Computer and Information Science. His research interests are in computer graphics, human-computer interaction, and image processing and analysis.

Almir Karabegović received his Ph.D. in technical sciences from the University of Sarajevo, Faculty of Electrical Engineering, in 2009. He is a Full Professor at the same Faculty and Leading Researcher at the GAUSS Center for Geospatial Research Sarajevo. He is a Senior IEEE Member, as well as ACM and AIS. His professional interests include Geoinformatics (GIS), Artificial Intelligence, Geostatistics, Information Systems Analysis and Design, and Database Systems.