Black Sea Journal of Engineering and Science
doi: 10.34248/bsengineering.1988909

Open Access Journal
e-ISSN: 2619 – 8991

Research Article
Volume 9 - Issue 5: 2473-2481 / September 2026

CUDA-BASED PARALLEL INTRA PREDICTION AND TRANSFORM ARCHITECTURE FOR HEVC INTRA-FRAME IMAGE CODING

Mücahit KAPLAN[1]*, Ali AKMAN[2]

[1]*Mudanya University, Faculty of Engineering, Architecture and Design, Department of Computer Engineering, 16940, Bursa, Türkiye*
[2] *İstanbul Ticaret University, Faculty of Engineering, Architecture and Design, Department of Computer Engineering 34000, Istanbul, Türkiye*

Abstract: Computations related to intra prediction and transform in High Efficiency Video Coding are highly computational demanding, especially when multiple intra prediction modes have to be computed for block-based image encoding. In this paper, we propose a parallel CUDA implementation of HEVC intra frame prediction and integer transform operations on an embedded GPU platform. The presented implementation parallelizes HEVC intra prediction modes (35 in total, divided into planar, DC, and angular modes) and implements an efficient transform computation utilizing shared memory allocation. The proposed implementation was tested on NVIDIA Jetson Xavier NX platform using standardized test images and various quantization parameter settings. The results demonstrate that our solution accelerates the computations by up to 105.17× in terms of kernel computations compared to the sequential CPU implementation. Taking into account CPU–GPU memory transfer and synchronization overheads, the acceleration factor for the end-to-end process is 1.62×. In terms of the reconstructed image quality, there is a small decrease in the average PSNR value equal to about 0.65 dB. It is mainly caused by implementation-related differences in parallel computation, rounding, and quantization procedures. These results show that intra-frame prediction and transform stages in HEVC video codec can be efficiently parallelized using CUDA; however, there are still memory transfer and synchronization overheads limiting end-to-end performance. Our implementation can serve as a building block of a fast HEVC encoder but not as a full-fledged video encoder itself. Source code and dataset for this research are freely available on Zenodo with the DOI: https://doi.org/10.5281/zenodo.17286247.

Keywords: HEVC intra prediction, CUDA, Integer DCT, Embedded GPU, Intra-frame image coding, Block-based image compression

***Corresponding author:** Mudanya University, Faculty of Engineering, Architecture and Design, Department of Computer Engineering, 16940, Bursa, Türkiye
E mail: mucahit.kaplan@mudanya.edu.tr (M. KAPLAN)
Mücahit KAPLAN https://orcid.org/0000-0001-5712-3093
Ali AKMAN https://orcid.org/0000-0002-9486-519X

Received: July 07, 2026
Accepted: August 14, 2026
Published: September 15, 2026

Cite as: Kaplan, M., & Akman, A. (2026). CUDA-based parallel intra prediction and transform architecture for HEVC intra-frame image coding. *Black Sea Journal of Engineering and Science,* 9(5), 2473-2481.

1. Introduction

HEVC has provided very efficient methods for block-based prediction and transforms for visual data compression. In particular, intra prediction and integer transforms are very crucial tools, as they are used extensively in the process of coding blocks, where various prediction modes have to be evaluated (Bross et al., 2023). Even though HEVC was initially designed as a video encoding technique, the intra-frame coding scheme provided by HEVC can offer a good environment for investigating block-based image coding schemes. This is why the current work will concentrate only on the acceleration of HEVC intra-frame prediction and transforms using CUDA, and not HEVC video encoder itself.

Recently, the Discrete Cosine Transform has been identified as the most computationally intensive component of the High Efficiency Video Coding standard. Based on this fact, Akman and Cekli (2023) proposed new architectures for the 2D DCT using approximate compressors with low power and hardware requirements. The proposed design has high computational efficiency with minimal quality degradation. This shows the potential benefits of hardware optimization in the implementation of the High Efficiency Video Coding standard. The rapid advancement in the design of Graphics Processing Units has introduced new frontiers in video encoding techniques using the concept of General-Purpose GPU computing. The NVIDIA Compute Unified Device Architecture has introduced ease in programming the GPU (NVIDIA, 2023). This has brought about the efficient implementation of parallel video encoding techniques. The intra prediction of large coding blocks has attracted significant attention in the field of parallel processing. However, the implementation of intra prediction in the GPU has brought about several challenges in terms of alignment difficulties in the intra prediction process. The parallelized calculation of SAD and SSD has been demonstrated to be efficiently performed in the GPU. The implementation of motion estimation in the GPU has brought about significant improvements in the motion

estimation process. Khemiri et al. (2018) demonstrated the parallelization of motion estimation in the GPU. The parallelization has brought about an improvement of 56% in the motion estimation process compared to the CPU implementation for large block sizes. Additionally, Jingjing and Kehua (2019) proposed a fast algorithm for intra picture prediction that reduces computational complexity by simplifying prediction modes analyzed according to how often they are applied, making intra prediction more efficient for video coding platforms.

Tariq (2020) proposed a high-performance accelerator that combines statistical inference and optimal stopping theory to accelerate intra-mode selection in the HEVC standard. This approach is fundamentally based on CPU-based modeling and aims to reduce coding complexity. In this study, the same principles are extended to GPU parallel architecture, significantly increasing real-time applicability in heterogeneous CPU–GPU environments.

In addition to traditional GPU-accelerated HEVC approaches, similar parallel processing concepts have recently been applied to other subfields of visual computing. For example, Pietri et al. (2025) proposed a GPU-based system for real-time compression of dynamic 3D meshes in cloud-based games. This system utilizes CPU–GPU co-processing on the client side to enable low-latency data transfer.

Zhu et al. (2023) proposed a frame-based fusion network (FFFN) that utilizes a feedback mechanism for video super-resolution (VSR). This method enables high-quality video reconstruction by iteratively combining the spatiotemporal information obtained from aligned frames. Although the focus is on super-resolution, the proposed feedback-based parallel fusion structure shares similar acceleration principles with the GPU-based parallel implementation of intra prediction in HEVC in this work.

An essential step in video compression standards, including HEVC, is the Discrete Cosine Transform (DCT), valued for its high-energy efficiency capabilities (Singhadia et al., 2019). To reduce computational overhead, HEVC employs integer DCT instead of real-valued DCT. However, even with integer DCT, the process remains computationally intensive because of the large number of multiplication operations. Quantization and transform processes, along with analysis of rate-distortion, can account for nearly 25% of the overall encoding time in the HEVC reference software (Akman and Cekli, 2023; Ryoo et al., 2008). Therefore, optimizing these processes is crucial for any attempt at enhancing encoding efficiency and saving energy. In the work by Juncheng et al. (2015), parallel intra programming was explored with respect to HEVC encoding and decoding on GPU + CPU platforms. It was shown that the combined computational resources provided by heterogeneous platforms can be effectively utilized to achieve faster encoding speeds with no compromise in terms of efficiency. Another work by Radicke et al. (2016) proposed a novel intra prediction technique specifically designed to be implemented on heterogeneous platforms with CPU + GPU architecture. This demonstrates the potential benefits of exploiting the computational resources provided by heterogeneous platforms to achieve faster encoding speeds with HEVC encoders. In the context of parallel processing techniques, the demand for efficient computation with respect to video encoding has led to the implementation of parallel execution techniques. These techniques have been implemented either by exploiting GPU resources or by exploiting the combined resources provided by CPUs and GPUs. It has been shown that the transform block of the HEVC encoding scheme can be efficiently implemented by exploiting parallel processing techniques, where the transform block is efficiently implemented by exploiting the shared memory addressing capabilities provided by GPUs. Moreover, the exploration of design alternatives technique was employed by Bloch et al. (2023) with respect to GPU + CPU common system platforms to optimize task partitioning and scheduling. This was implemented by exploiting the Tabu search technique. Another work by James et al. (2024) evaluated GPU-based HEVC encoders, i.e., Kvazaar and x265, with respect to embedded platforms and concluded that these encoders can be efficiently implemented to achieve real-time encoding with superior speed and quality balance. Several more recent researches have also dealt with the issue of computational complexity of HEVC both in terms of algorithms and hardware acceleration. For example, Wang et al. (2025) have suggested a fast HEVC intra-coding mode selection algorithm based on image texture direction and correlation between various prediction modes. As a result of this algorithm implementation, the authors managed to achieve more than 46% reduction in the encoding time on target resource-limited embedded devices. Li et al. (2025) have proven that texture-based characteristics, neighboring coding modes and rate-distortion data may be used together to decrease the complexity of intra-prediction process in scalable HEVC systems. With regard to hardware-software co-design approach, Lopez-Granado et al. (2025) have shown that the overall acceleration of HEVC system is significantly dependent not only on the efficiency of accelerated computational core but also on the hardware-software integration overhead. More recently, Rathee et al. (2026) have suggested HEVC architecture implemented on the FPGA platform with dynamic-programming optimization of intra-frame prediction process. In a recent work by Imen et al. (2022), intra-mode complexity was addressed by exploiting the potential benefits provided by the LeNet-5 and AlexNet deep learning architectures. This was done by replacing the exhaustive RDO search with CNN prediction for CU partitioning. This demonstrates the importance of optimizing intra prediction with respect to HEVC encoding.

According to the literature review of the relevant works, GPU acceleration has been extensively researched for various stages of HEVC encoding such as intra prediction,

transform calculation, task scheduling, and heterogeneous CPU-GPU operation. Yet, the joint work of HEVC intra prediction mode evaluation and the optimal integer transform calculation on the embedded GPU system deserves more experimentation regarding thread mapping, shared memory utilization, and CPU-GPU coordination overheads. Furthermore, recent research has proven that acceleration achieved at the computational kernel level does not always result in the same amount of acceleration in the overall system because the process of data transfer and synchronization could become the main bottleneck (Lopez-Granado et al., 2025). Hence, the joint consideration of kernel execution efficiency and total processing time becomes crucial for heterogeneous embedded systems.

In our work, we are not introducing any novel standards for HEVC, or developing a full video encoder; however, our primary contribution is the design and implementation of the CUDA architecture that performs parallel evaluation of all 35 HEVC intra prediction modes, as well as the optimal integer transform computation process. Our proposed architecture is experimentally tested on the NVIDIA Jetson Xavier NX platform.

Recently, transformer-based visual processing frameworks such as the Efficient Attention Pyramid Transformer (EAPT) have demonstrated the potential of multi-level attention and global patch communication for high-performance image processing tasks (Lini et al., 2023).

In this work, we present a highly parallelized implementation intra prediction and DCT for HEVC on GPUs. The proposed method effectively balances memory bandwidth and access patterns, maximizing both resource utilization and execution speed.

The key contributions of this research are listed below: First, a CUDA-based parallel architecture is designed to evaluate all 35 HEVC intra prediction modes on an embedded GPU platform. Second, intra prediction is combined with integer transform, quantization, inverse quantization, and inverse transform procedures within the framework of intra-frame block-based coding. Third, two improved transformation procedures based on row-wise multiplication without transposition and symmetry are implemented using CUDA. Fourth, the proposed architecture is tested on the NVIDIA Jetson Xavier NX using separate kernel execution speed and full execution time results to clarify the impact of CPU–GPU memory transfer and synchronization overheads. Fifth, the image reconstruction quality is estimated using PSNR.

The remainder of this paper is organized as follows: Section 2 describes the mathematical and procedural foundations of HEVC intra prediction and transform phases. Section 3 presents the detailed design of the proposed GPU-based parallel computing architecture, thread mapping model, and shared memory configurations. Experimental methodologies, execution benchmarks, and resource bottleneck analyses are evaluated in Section 4. Finally, Section 5 concludes the paper and highlights strategic directions for future work.

2. Materials and Methods

HEVC block structure describes hierarchical visual data representation into coding units, prediction units, and transform units. In the current study, only the intra frame part of HEVC coding process is discussed. All input test images are treated as intra frames and segmented into coding units (block-based) for assessment of intra prediction and transformation. Consequently, the suggested approach

may be considered as CUDA-acceleration of HEVC intra frame computational modules, but not as a full-fledged HEVC video encoder. The general structure of HEVC intra prediction process is shown in Figure 1.

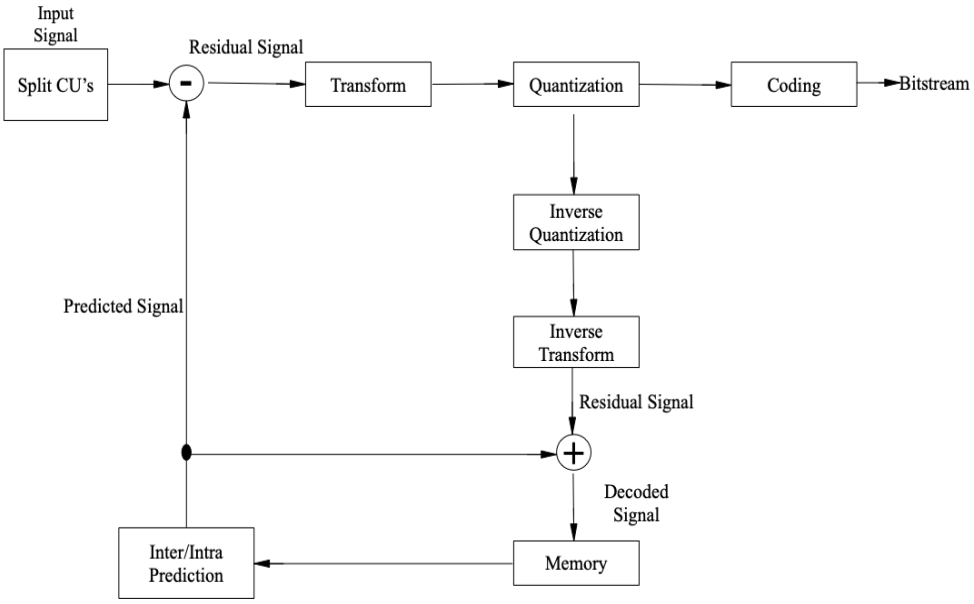


Figure 1. Basic Structure of the HEVC Intra Prediction Implementation.

In HEVC, the visual data is segmented into Coding Tree Units (CTUs) that provide flexible block structure for different degrees of spatial detail description. The CTU can be further split into Coding Units (CU) that consist of Prediction Units (PU) and Transform Units (TU). The TU size is represented by the formula N × N, where N equals to 4, 8, 16 or 32 (Bross et al., 2023). In the current research, 32 × 32 coding units are used in the experimental implementation in order to map the intra prediction and transformation into the CUDA thread/block structure.

2.1. Intra Prediction

HEVC's intra prediction framework consists of three basic steps aimed at increasing coding efficiency and reducing the computational load on the encoder and decoder: preparation of reference samples, creation of prediction samples, and post-processing (Bross et al., 2023). In this process, each block is predicted using its reconstructed upper and left neighboring samples. HEVC supports 35 different intra prediction modes to achieve high compression ratios: 33 of them are angular, one is planar, and one is DC (mean-based) mode. An example of reference samples used by HEVC intra prediction for an N x N block is presented in Figure 2. General block diagram of parallel calculation of intra prediction model is presented in Figure 3.

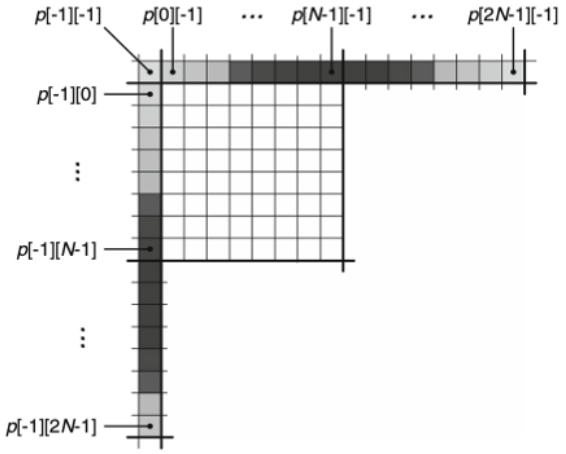


Figure 2. An example of reference samples used by HEVC intra prediction for an N x N block (Bross et al., 2023).

HEVC uses 33 intra prediction modes for angular prediction that represent various directional structures in a block, whereas the other two modes are for planar and DC prediction. Angular prediction is the process through which each of the predicted samples is obtained via interpolation of reference samples based on the angular direction chosen. The value of the pixel is obtained through interpolation of two adjacent samples denoted as R[i] and R[i+1]. This is shown in equation 1.

$$P(x,y) = \frac{(32-w)\,R[i] \;+\; wR[i+1]}{32} \tag{1}$$

In this case, the notation P(x,y) refers to the interpolated sample, while R[i] and R[i+1] refer to the neighboring reference samples. The interpolation weight w is defined according to the selected angular prediction mode. The denominator is equal to 32 and relates to HEVC angular interpolation normalization.

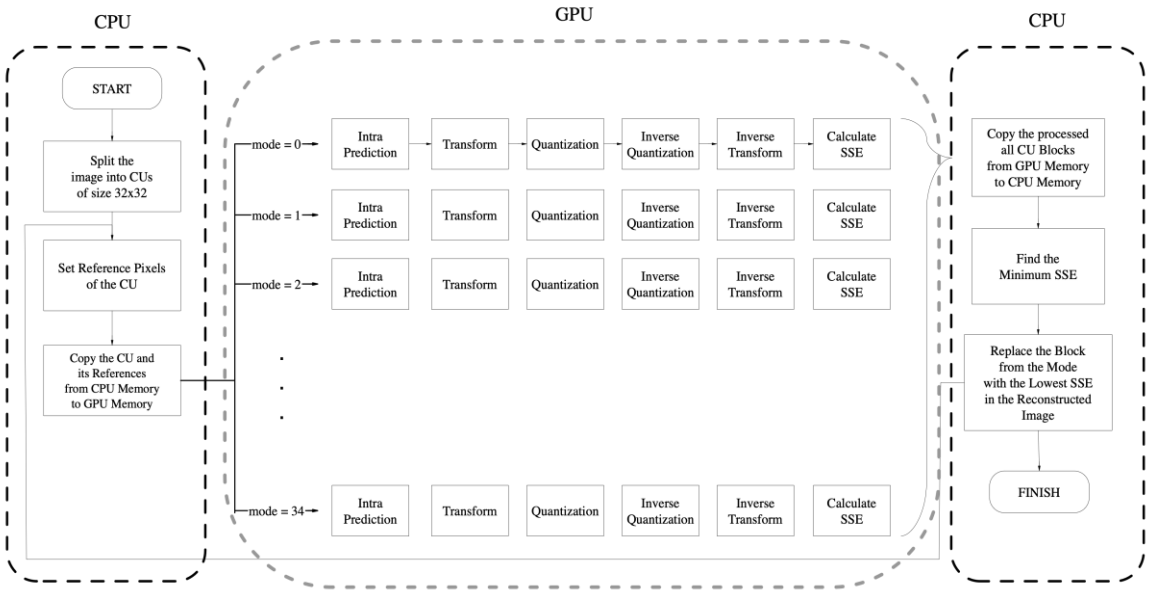


Figure 3. General block diagram of parallel calculation of intra prediction model.

In the DC estimation case, the estimated sample values are filled with a fixed value representing the average of the reference samples immediately to the left and above the block to be estimated.

Planar mode is an estimation method used to provide smooth transitions between image blocks. This mode makes estimates based on both the top and left edge pixels and determines the value of each pixel by taking the average of these two estimates.

The estimated pixel value is calculated as follows equation 2:

$$P(x,y) = (P_h(x,y)+P_v(x,y)+N) / 2N \quad (2)$$

Here, $P_h(x,y)$ is horizontal (top edge) prediction, $P_v(x,y)$ is vertical (from the left edge) estimate. N is block size (e.g., 4, 8, 16). These two sub-estimates are obtained as follows (equations 2a and 2b):

$$P_h(x,y)= (N-1-x)\cdot p(-1,y)+(x+1)\cdot p(N,-1) \quad (2a)$$
$$P_v(x,y)=(N-1-y)\cdot p(x,-1)+(y+1)\cdot p(-1,N) \quad (2b)$$

That is, $p(-1,y)$ is reference pixels on the left edge, $p(x,-1)$ is reference pixels on the top edge, $p(N,-1)$ is top-right corner pixel, $p(-1,N)$ is bottom-left corner pixel.

These expressions establish a linear transition between reference pixels horizontally and vertically, depending on the pixel position. Taking the average of these two estimates yields a smooth surface within the block and prevents sharp transitions at edges, ensuring higher visual quality in flat regions.

In HEVC, intra prediction is performed at the transform block (TB) level, and the block size can be selected from 4×4 to 32×32. However, for the prediction to be performed, the necessary reference samples must have been previously encoded and reconstructed from the relevant neighboring blocks. This creates a dependency chain that determines the order of computation and imposes a constraint on parallelization.

The encoder calculates the sum of the squares of the differences between the predicted block and the original block to evaluate the accuracy of each prediction mode. This operation is carried out by taking the square of the difference between the original and the predicted values at each pixel position and summing them across the whole block. This measure is referred to as the sum of squared errors (SSE), and the prediction mode with the lowest value is chosen for the next steps in the encoding process.

The prediction mode with the lowest SSE value is chosen for the next steps in the encoding process. This selection has a direct influence on the efficiency of the compression and the complexity of the encoding. From a parallel processing viewpoint, it can be seen as an advantage.

2.2. Transform and Inverse Transform

The direct application of the transform on the predicted block is not performed in the HEVC intra-coding scheme. The transform process is done on the residual block that is formed after subtracting the predicted block from the original input block. Hence, in this research work, the input matrix X in the transform formula is the residual block and not the original or the predicted block. This transformation has the purpose of concentrating the energy in certain components and attaining a more efficient data structure for the quantization step. It is allowed for the transformation blocks (Transform Units – TU) to have different sizes: 4×4, 8×8, 16×16, and 32×32.

HEVC uses an integer-based DCT instead of floating-point-based DCT. The two-dimensional DCT operation is expressed as a two-step matrix multiplication along the row and column directions (equation 3):

$$[C] = [D] [X] [D]^T \quad (3)$$

In the above case, X stands for the input residual block, D stands for the N × N integer transform matrix, and C stands for the transform coefficient matrix. Once the inverse quantization and the inverse transform process are completed, the residual block is added to the predicted block.

Inverse transform side, this process is reversed to obtain the estimated block (equation 4):

$$[X] = [D]^T [C] [D] \quad (4)$$

During this transformation process, various scaling factors are used to maintain numerical accuracy. These factors for the forward and inverse stages of the transformation are defined as follows:

After the first forward transformation: $ST_1 = 2^{(-B+M-9)}$

After the second forward transformation: $ST_2 = 2^{-(M+6)}$

After the first inverse transformation: $SIT_1 = 2^{-6}$

After the first inverse transformation: $SIT_2 = 2^{-(21-B)}$

Here, B denotes the bit depth of the input/output signal (e.g., 8 bits), and M denotes the base-2 logarithm of the transformation size: $M=\log_2 N$.

In the standard computational approach, the transpose of the intermediate matrix resulting from the D·X multiplication must be taken. However, transposing the intermediate matrix increases memory access costs and hinders parallel computation. Therefore, the row-based multiplication method proposed by Kaplan and Akman (2024) eliminates the transpose operation, reducing the computational load and facilitating parallelization. This method allows the two-dimensional DCT to be computed directly, eliminating the need for an intermediate transpose.

Another improvement method proposed in the transformation phase is to simplify matrix operations by utilizing the symmetry and inverse symmetry properties of HEVC transformation matrices. In HEVC integer DCT matrices: Even-numbered rows are symmetric, Odd-numbered rows are inverse symmetric. Thanks to this structure, the transformation process can be performed with fewer operations as follows (equation 5):

$$\begin{bmatrix} C_c \\ C_t \end{bmatrix} = \begin{bmatrix} D_{N/2\,x\,N/2} & 0 \\ 0 & M_{N/2\,x\,N/2} \end{bmatrix} \cdot \begin{bmatrix} a_{n/2} \\ b_{n/2} \end{bmatrix} \quad (5)$$

represent the transformation coefficients in even and

odd rows, respectively, D and M represent the simplified transformation matrices.

This structure reduces the computational load by removing matrix elements multiplied by zero from the calculation. Such simplified structures provide significant advantages for parallel computation, especially in 32×32 transformation blocks.

2.3. Quantization and Inverse Quantization

Quantization is the process of scaling the coefficients obtained because of the transformation process using a specific quantization parameter (QP) and representing them with smaller integer values with limited bit width. This step forms the basis of compression, aiming to minimize file size while preserving perceptual image quality. The HEVC standard has incorporated a frequency component-based quantization approach to make this process more efficient. In this approach, low frequency component quantization is done with a smaller quantization step size, while high frequency component quantization is done with a larger quantization step size. This is done to ensure the quality of the image is not compromised, especially in the low frequency regions where the human eye is more sensitive to changes.

For the designed architecture, the quantization step will be applied to observe the reconstruction performance of the intra-frame pipeline processing design. At the end of the quantization step, the coefficients will be reordered into 1D ZigZag scan format and will be encoded through the Exponential-Golomb (Exp-Golomb) coding block. The Exp-Golomb coding block will be used to compute the output coefficient size as well as S.O. value from the experiment results. For the current research, the Exp-Golomb block will be used for coefficient coding purpose only and not to generate the HEVC bitstream.

The process of quantization is performed in an inverse manner during inverse quantization, whereby the quantized values are scaled back to represent their corresponding amplitudes. As such, inverse quantization forms part of the intra-frame coding process by assisting in the reconstruction of the residual block and quantization effects. Although this process is not entirely lossless, near-lossless quality can be achieved by choosing an appropriate QP value.

For this investigation, QP levels of 17, 22, 27, 32, 37, and 42 are used in order to test the performance of the proposed CUDA implementation in terms of its stability under different levels of quantization. Smaller QP levels correspond to better reconstruction quality with more non-zero transform coefficients, while greater QP levels imply a greater level of quantization and, correspondingly, worse reconstruction quality.

2.4. CUDA-Based Parallel Implementation and Memory Organization

GPU is extensively used as a highly parallel processor to perform visual data processing jobs which are highly compute-intensive. In HEVC intra-frame encoding process, operations like intra prediction, transformation, quantization, inverse quantization, and inverse transformation need to be performed at block level repeatedly. Thus, these operations can be considered as good choices to perform with CUDA parallelism, especially in case of multi-mode intra prediction.

In GPU-enabled visual computing pipelines, the calculation can be done on a per-frame, per-block, or batch basis according to the dependency model of the algorithm. In our solution, the source image is broken down into coding units, and the 35 modes of HEVC intra prediction are mapped to parallel CUDA kernels for each block. However, data transfers between the CPU and GPU, especially in heterogeneous accelerator architectures, can be a significant factor in the total processing time. Depending on the level of the GPU used, these transfer delays can reach up to 26% on a mid-range GPU and up to 73% on a high-end GPU. In this study, to reduce this latency, data blocks are transferred asynchronously from main memory (CPU – host) to the GPU (device) in equal-sized batches. This allows for more efficient use of bandwidth and optimizes processing time (Harris, 2019).

CUDA (Compute Unified Device Architecture) is a platform developed by NVIDIA that enables general-purpose parallel programming on the GPU. The CUDA architecture allows tasks to be divided into small pieces and executed simultaneously by multiple threads. In the CUDA architecture, the part of the code to be executed on the GPU is defined as a kernel function. These functions are marked with the __global__ statement, called by the CPU, and executed in parallel on the GPU (NVIDIA, 2019).

When each kernel is executed, threads are created. These threads are grouped into blocks, and the blocks are placed into larger structures called grids. Both blocks and threads can be configured in 1D, 2D, or 3D configurations. Threads are defined by indices such as threadIdx.x, threadIdx.y, threadIdx.z, while blocks are defined by indices such as blockIdx.x, blockIdx.y (Ghorpade et al., 2012; Thurley and Danell, 2012).

This hierarchical memory structure minimizes memory access conflicts (bank conflicts), thereby enhancing the performance of parallel processing. In CUDA, shared memory is divided into sections called banks, and consecutive addresses are assigned to different banks. However, when multiple threads attempt to read data from the same bank at the same time, a conflict occurs. In this research, an empty (unused) column was added to the shared memory to prevent this issue, and this empty space was accounted for in the transformation matrices (Figure 4).

In this method, image blocks are first transferred from the CPU to the GPU's global memory and then copied to shared memory. Each thread processes only one pixel within this block, and operations such as transformation and quantization are executed in full parallel. This approach significantly reduces processing time and maximizes the GPU's high parallelism and bandwidth advantages.

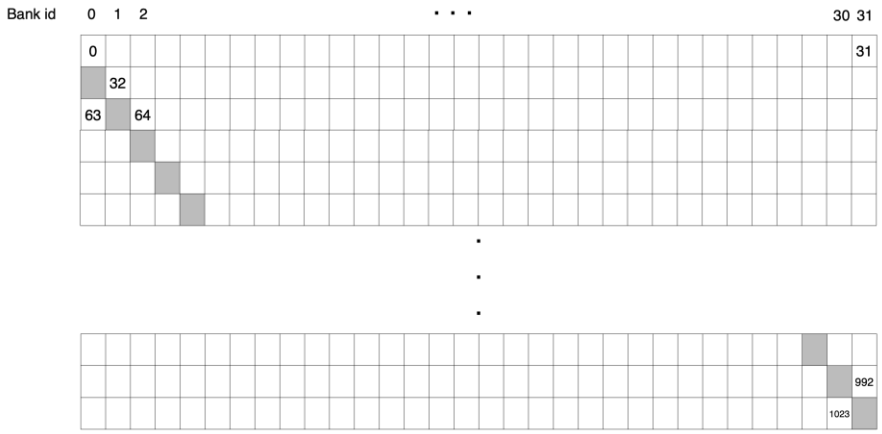


Figure 4. Space Usage in Shared Memory (Gray Shaded Areas represent gaps).

2.5. Implementation and Evaluation

The following section presents the implementation of the proposed HEVC intra-frame computational model using the CUDA framework. The work concentrates on the acceleration of intra prediction, residual generation, integer transform, quantization, inverse quantization, inverse transform, and reconstruction procedures for block-based image compression algorithm. The block scheme given in Fig. 3 illustrates the general flow of the proposed approach.

In this case, all 35 HEVC intra prediction modes for each coding unit are tested simultaneously on the GPU. For each HEVC intra mode, there is a CUDA execution block assigned, while the pixels of the coding unit are processed in parallel CUDA threads. In such a way, the HEVC intra prediction procedure, which is very resource-intensive operation in HEVC intra coding, is executed in the parallel GPU environment. Data preparation, memory management, and mode selection tasks are done on the CPU, while block-level prediction and transformation are performed by GPU.

Algorithm 1: Pseudo-code for GPU-based HEVC intra prediction.

START
// Divide the input image into 32 × 32 coding units
blocks = splitImageIntoBlocks(image, 32, 32)
// Initialize reconstructed output image
reconstructedImage = initializeImageLike(image)
// Process each coding unit
for originalBlock in blocks:
// Upload the original block and its reference samples to GPU memory
copyOriginalBlockAndReferencesToGPU(originalBlock)
// Initialize arrays for storing mode-dependent results
reconstructedBlocks = []
sseValues = []
// Evaluate all 35 HEVC intra prediction modes in parallel on the GPU
for mode in parallel(35 HEVC intra prediction modes):
predictedBlock = intraPrediction(originalBlock, mode)
residualBlock = originalBlock - predictedBlock
transformedResidual = applyTransform(residualBlock)
quantizedResidual = quantize(transformedResidual)
inverseQuantizedResidual = inverseQuantize(quantizedResidual)
reconstructedResidual = applyInverseTransform(inverseQuantizedResidual)
reconstructedBlock = predictedBlock + reconstructedResidual
sse = calculateSSE(originalBlock, reconstructedBlock)
storeInGPU(mode, reconstructedBlock, sse)
// Copy reconstructed blocks and SSE values back to the CPU
reconstructedBlocks, sseValues = copyModeResultsFromGPUToCPU()
// Select the best prediction mode based on the minimum SSE
bestMode = findMinimumSSEMode(sseValues)
// Write the best reconstructed block to the reconstructed image
replaceBlockInImage(reconstructedImage, originalBlock.position, reconstructedBlocks[bestMode])
END

Algorithm 1 summarizes the processing steps of the suggested implementation. This pseudocode aims to clarify the computational flow instead of representing a full HEVC-compliant encoder. Specifically, the algorithm demonstrates how every original block gets predicted, how the residual block is derived, how transform and quantization operations apply to the residual, and how the reconstructed block is produced for SSE-based mode evaluation.

According to Algorithm 1:

splitImageIntoBlocks: The initial picture gets split into blocks that measure thirty-two by thirty-two pixels. Every single block then runs through its own separate step inside the suggested process for frames taken within one scene.

copyOriginalBlockAndReferencesToGPU: For every coding unit, the initial block and needed reference pixels from adjacent rebuilt areas move into GPU storage. Those reference pixels serve the HEVC intra prediction methods.

parallel(35 HEVC intra prediction modes): For every

initial block, all 35 HEVC intra prediction modes are assessed simultaneously on the GPU. These modes include 33 angular types, one planar type, and one DC type.

intraPrediction(originalBlock, mode): For the chosen intra prediction mode, a predicted block is created by applying the matching reference samples.

residualBlock: The residual block comes from subtracting the predicted block from the original one. That residual block serves as the input to the transform stage.

applyTransform leads into quantize which goes to inverseQuantize then applies inverseTransform: here the integer transform acts on the residual block. After that the transformed residual gets quantized, inverse quantized, and inverse transformed so we get the reconstructed residual block back.

reconstructedBlock: The rebuilt block comes from putting the recovered residual block onto the forecasted block again.

calculateSSE: The SSE value is computed between the initial block and the rebuilt block for every intra prediction mode. That figure is utilized to compare the prediction modes.

storeInGPU: The rebuilt block and SSE value matching every mode get put into temporary GPU storage space.

copyModeResultsFromGPUToCPU: All 35 mode reconstructed blocks and SSE values get copied back onto the CPU.

findMinimumSSEMode: The CPU picks the optimal intra prediction mode by finding the one that has the lowest SSE number.

replaceBlockInImage: The reconstructed block matching the best mode gets placed inside the output reconstructed image right where the original coding unit sits.

This structure significantly speeds up the mode selection process, one of the most computationally intensive stages of HEVC, by making it fully parallel on the GPU. The block/thread structure of the CUDA architecture allows each prediction mode to be processed independently.

In addition, two optimization techniques proposed by Kaplan and Akman (2024) are used in the transformation operations. The first is a row-based multiplication method that eliminates matrix transposition. This method simplifies memory access complexity, thereby facilitating faster computation of transformation calculations. The second method involves a simplified transformation structure that does not include zero elements based on symmetry properties of DCT matrices. This makes the transformation operations more compatible with and faster on the GPU architecture.

Differences between the two DCT optimization methods lie mostly in the different techniques used in minimizing computation and memory access overheads. First, Method 1 of the DCT performs the transformation of the input into two dimensions through matrix multiplication based on the rows of the matrix. As such, the primary benefit of this method is that the transpose step of the intermediate matrix is eliminated. The second DCT method further takes advantage of the symmetry and antisymmetry characteristics of the integer transform matrix of HEVC. This is done by re-arranging the transform equations into even and odd parts and minimizing unnecessary multiplications of zero-value matrix elements. While Method 1 focuses on improving the memory access aspect, the latter method reduces the memory overhead and unneeded computations.

Shared memory was used effectively throughout all operations, data sharing between threads was optimized, and intra-block synchronization was achieved using the __syncthreads() command. This is particularly critical for data-dependent operations such as transformation and quantization.

As a result, the proposed parallel application structure enables intra prediction and mode selection operations to be performed in less time, with fewer memory accesses, and with higher accuracy; while maintaining HEVC's coding efficiency, performance has been significantly improved.

4. Results

The proposed CUDA-based parallel implementation was experimentally evaluated for the HEVC intra-frame prediction and transform operations in this study. Four 512 × 512 reference images were used for experiments: Lena, Airplane, Boat and Barbara. This set of reference images was chosen due to the fact that they possess varied spatial properties which could be used to assess image processing algorithms such as smoothness, directional edges, contours of objects, and texture. This means that the chosen images represent a set of varied test cases for testing whether the proposed parallel approach is computationally stable depending on the nature of the image content. All images were processed using 512 × 512 resolution in order to create an equal computational load. Each image was treated as an intra frame and split into coding units of 32 * 32. The experimental platform is the developer kit of the NVIDIA Jetson Xavier NX. The implementation is developed by CUDA programming model. The main objective of the experiments was to measure the computational speedup offered by parallel intra prediction and optimized integer transform computation, not the performance of a full HEVC video encoder. Development environment is shown in Figure 5.

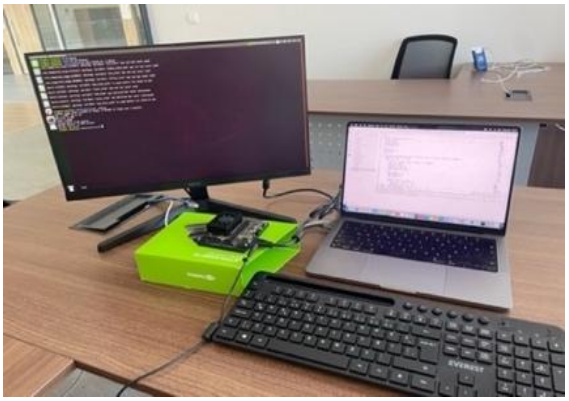


Figure 5. Development environment.

For a fair device-level baseline, the sequential CPU implementation was run natively on the onboard CPU of the NVIDIA Jetson Xavier NX platform with a 6-core NVIDIA Carmel ARMv8.2 64-bit CPU. CPU code was compiled with GNU Compiler Collection with optimization option -O3. This baseline was compared with the proposed GPU implementation under the same constraints of the embedded platform.

The workload was split in 32 × 32 coding units. The 33 angular modes, one planar mode and one DC mode were used to test all 35 HEVC intra prediction modes for each coding unit. In the CUDA implementation, each prediction mode was assigned to a parallel execution block and the CUDA threads processed the pixels within the coding unit.

This allowed for a highly efficient parallel implementation.

QP values of 17, 22, 27, 32, 37, and 42 were considered for analysis of the proposed implementation under various degrees of quantization strength. The lower the value of QP, the better the accuracy of reconstruction. On the other hand, a higher QP value means greater quantization strength. In such an environment, Peak Signal-to-Noise Ratio was employed for assessing the quality of reconstructed images. However, since the present study does not implement a complete HEVC video encoder, the results focus on computational acceleration, Exp-Golomb-based S.O. values, and reconstructed image quality. The obtained PSNR values revealed the effect of the DCT methods used in the transformation step on image quality (Figure 6).

The performance gain on the GPU was evaluated using two main criteria:

Kernel execution time: This refers to the time it takes to run kernel functions on the GPU and perform intra prediction operations on blocks. This measurement was made excluding memory transfers and image segmentation operations.

Total processing time: This includes all steps from the start of the transformation process, including memory transfers and image processing.

The time measurements have been recorded with the help of NVIDIA Nsight Profiler. Two timing results were obtained: kernel execution time and total execution time. Kernel execution time is only concerned with the GPU execution phase and ignores the data transfer phase, image segmentation and CPU side mode choice. Total execution time considers CPU-GPU memory transfer, synchronization and collection phases. The performance achieved by the proposed GPU approach on average is 105.17× in terms of kernel execution when the DCT Method 2 and shared memory layout are employed. Nevertheless, in terms of total execution, the average speedup attained is 1.62× (Figure 6).

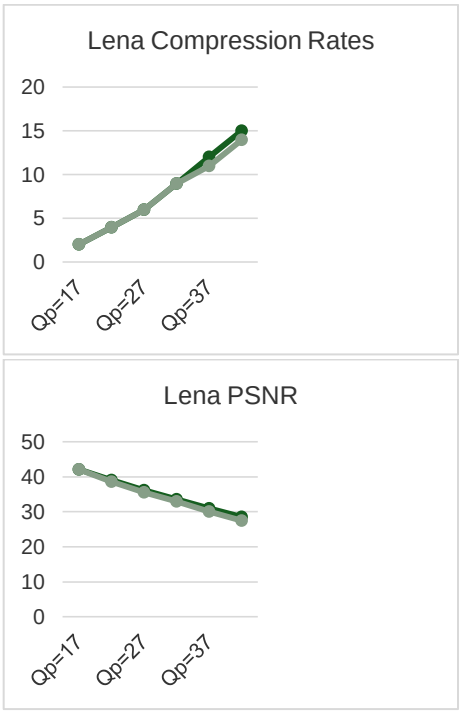


Figure 6. Comparison S.O. and PSNR Values of the Reference Image in CPU Serial and GPU Parallel Implementation

Significant improvements were also observed in total processing time. Compared to the CPU-based sequential system, the GPU parallel implementation was 1.3× faster, while shared-memory-based DCT methods achieved speedups of up to 1.62×. These results indicate that CPU–GPU data transfer remains a performance bottleneck despite the considerable kernel-level acceleration. Leaving extra space in the shared memory area in kernel processes running on the GPU prevented bank conflicts that could occur during memory access, further reducing the processing time. These results show that shared memory optimization improves kernel-level performance, while end-to-end acceleration remains limited mainly by CPU–GPU memory transfer and synchronization overheads.

These data demonstrate that optimizing shared memory directly affects computational speed at the kernel level and enables more efficient use of GPU resources through rapid access to repeatedly accessed data. This block-to-block cycle of memory transfers (CPU-to-GPU memory

transfer, evaluation on the GPU, GPU-to-CPU memory transfer, and selection of the minimum on the CPU side) is a carefully crafted trade-off dictated by the constraints of memory resources on low-power edge systems. For the NVIDIA Jetson Xavier NX platform, simultaneously caching the set of blocks and residual modes on edge for a whole high-resolution grid is beyond the capability limits of the device in terms of its global memory space, resulting in critical errors due to Out-of-Memory runtime execution problems. Offloading the minimum SSE evaluation task to the host CPU allows for safe operation and accurate spatial placement at the cost of additional time due to bus synchronization.

Moreover, the parallel DCT approaches proposed in this study, when assessed in terms of the intra prediction operations, ensured fair utilization of the GPU cores and fair allocation of the workload among the threads. In addition, the utilization of blocks for each mode proves the capability of the system in terms of scalability, which is important in embedded intra-frame coding.

The study proves the significant benefits of the proposed parallel architecture in terms of the processing time and quality in the implementation of the HEVC intra prediction process. This is because of the management of the memory, the synchronization, and the utilization of the appropriate algorithmic structures for the CUDA architecture.

The future direction of work will be to minimize CPU/GPU data transfer bottlenecks, such as using the CUDA unified memory model and asynchronous data streams.

In addition, the implementation of the parallelization of the execution threads, synchronization, and memory bank optimization will further enhance the system's efficiency in real-time encoding applications. These improvements will allow the widespread adoption of GPU-based parallel processing methods in high-density encoding standards such as HEVC.

The high-speed processing gains achieved in GPU-based parallel processing applications are a significant step forward in the development and implementation of real-time processing capabilities, particularly in computationally intensive applications such as video encoding. For high-efficiency coding standards such as HEVC, the intra prediction process consumes a significant amount of time. However, the capability to execute the transformation process and the entire intra prediction process in parallel has become a crucial factor in the system's efficiency. In this study, separating the prediction processes at the block level for each mode and executing these blocks in parallel through separate threads provided significant advantages in terms of scalability and concurrency.

Furthermore, the fact that the DCT methods proposed were tested on GPUs with different configurations demonstrates that parallel systems can be effectively used in comparative tests of different algorithms. While methods 1 and 2 used in the transformation steps yielded similar PSNR results, they exhibited different performance in terms of processing times. In particular, the fact that method 2 achieved higher speed gains with shared memory optimizations highlights that in such systems, not only the mathematical complexity of the algorithm but also its suitability for the GPU architecture must be considered.

Although data transfer times are high, this can be reduced with CUDA's asynchronous and stream-based architecture. Parallel architectures based on GPUs can be very beneficial for HEVC coding in terms of performance if computation-heavy tasks are efficiently implemented on parallel hardware, but the final performance is highly dependent on memory access patterns and synchronization between the CPU and GPU (Lopez-Granado et al., 2025). Proper memory management and synchronization improve performance. As shown in Table 1, the use of shared memory reduces the need for repeated data transfers, resulting in a notable decrease in processing time and an overall improvement in performance.

When the execution times are analyzed, an average 105.17-fold acceleration occurred in the core time, that is, in the part where the prediction takes place and parallelization takes place, and 1.62-fold acceleration occurred in the whole time. Table 2 gives the comparative context regarding the acceleration values between the proposed implementation and selected studies on acceleration values concerning HEVC intra prediction or parallel coding methods. These values should not be considered a benchmarking method due to differences in hardware architecture, input data, resolution, software implementation, and scope of experiments in the compared studies. For this reason, Table 2 is provided just to put the proposed implementation in the comparative perspective of the acceleration values presented in the literature. The experimental comparison in this study will be done between the sequential CPU implementation and the proposed GPU implementation on the same NVIDIA Jetson Xavier NX platform.

The PSNR, kernel execution time, total execution time, and reconstruction-related output metrics of the reference images at various QP values are shown in Table 3. A comparison of the sequential CPU version and the proposed GPU approach using DCT Method 2 and shared memory structure is made on equal platform settings. The obtained results demonstrate that the average PSNR difference between the CPU and GPU methods is about 0.651 dB.

Table 1. Test data of reference images

Duration	Measurement Shared Memory Space Utilization	Method	Lena		Airplane		Boat		Barbara	
			Execution Time (s)	Acceleration	Execution Time (s)	Acceleration	Execution Time (s)	Acceleration	Execution Time (s)	Acceleration
All Time	-	CPU serial	1.53	1.00	1.58	1.00	1.54	1.00	1.54	1.00
	YES	GPU Parallel	1.07	1.43	1.13	1.40	1.23	1.25	1.37	1.12
		GPU Parallel DCT Method 1	1.14	1.34	1.11	1.42	1.03	1.50	1.12	1.38
		GPU Parallel DCT Method 2	0.94	1.63	0.94	1.68	1.02	1.51	0.92	1.68
	NO	GPU Parallel	1.13	1.35	1.26	1.26	1.30	1.18	1.28	1.21
		GPU Parallel DCT Method 1	1.11	1.37	1.31	1.21	1.21	1.27	1.25	1.23
		GPU Parallel DCT Method 2	1.11	1.38	1.13	1.40	1.12	1.37	1.13	1.37
Core Time	-	CPU serial	0.84	1.00	0.87	1.00	0.87	1.00	0.88	1.00
	YES	GPU Parallel	0.01	57.61	0.01	70.02	0.01	62.55	0.01	66.48
		GPU Parallel DCT Method 1	0.01	66.83	0.01	71.76	0.01	79.77	0.01	72.53
		GPU Parallel DCT Method 2	0.01	103.10	0.01	105.80	0.01	104.70	0.01	107.10
	NO	GPU Parallel	0.02	34.37	0.01	62.92	0.01	59.97	0.01	65.49
		GPU Parallel DCT Method 1	0.02	35.55	0.02	52.62	0.01	67.93	0.01	69.65
		GPU Parallel DCT Method 2	0.02	36.32	0.01	76.84	0.01	79.77	0.01	78.36

Table 2. Contextual comparison of reported acceleration values in related studies

	Maximum Acceleration	Minimum Acceleration	Average Acceleration
Proposed method	1.82	1.33	1.62
Mate et al., 2020	-	-	1.22
Jingjing and Kehua. 2019	1.21	1.18	1.19
Juncheng et al., 2015	1.62	1.45	1.52
Radicke et al., 2016	1.54	1.52	1.52

Table 3. PSNR, Execution Time and Exp-Golomb-Based S.O. Variation of Reference Images

		CPU Serial			GPU Parallel DCT-2. Method Shared Memory Space Utilization				Comparison			
	QP	PSNR (dB)	Core Time (s)	Overall Time (s)	S.O	PSNR (dB)	Core Time (s)	Overall Time (s)	S.O	ΔPSNR (dB)	Core-Time Speedup (×)	Overall-Time Speedup (×)	ΔS.O
Lena	17	42.11	0.8292	1.519	2	42.11	0.0081	0.976	2	0	102.37	1.56	0.00
	22	39.09	0.8376	1.503	4	38.58	0.0084	0.956	4	0.511	99.71	1.57	0.00
	27	36.08	0.8505	1.57	6	35.57	0.0083	0.998	6	0.511	102.47	1.57	0.00
	32	33.50	0.8245	1.516	9	32.94	0.0079	0.876	9	0.561	104.37	1.73	0.00
	37	31.05	0.8307	1.529	12	30.06	0.0079	0.879	11	0.986	105.15	1.74	1.00
	42	28.58	0.8399	1.539	15	27.41	0.0081	0.946	14	1.170	103.69	1.62	1.00
Airplane	17	45.12	0.876	1.608	2	43.35	0.0077	0.902	2	1.761	113.77	1.78	0.00
	22	40.34	0.8787	1.597	3	40.34	0.0086	0.928	3	0	102.17	1.72	0.00
	27	37.33	0.8837	1.606	5	36.66	0.0079	0.883	5	0.669	111.86	1.82	0.00
	32	34.15	0.8365	1.532	8	33.50	0.0093	1.127	8	0.644	89.95	1.36	0.00
	37	31.40	0.8706	1.571	11	30.57	0.0079	0.920	11	0.837	110.20	1.71	0.00
	42	28.73	0.8645	1.560	16	27.71	0.0082	0.884	15	1.023	105.42	1.52	1.00
Boat	17	42.11	0.8791	1.580	1	42.11	0.0075	0.956	1	0	117.21	1.65	0.00
	22	38.58	0.8595	1.536	3	38.13	0.0079	1.002	2	0.457	108.80	1.53	1.00
	27	34.90	0.8772	1.542	4	34.32	0.0083	1.010	4	0.58	105.69	1.53	0.00
	32	31.79	0.8839	1.539	7	31.14	0.0089	1.022	7	0.655	99.31	1.51	0.00
	37	29.04	0.8685	1.531	11	28.26	0.0081	1.003	10	0.782	107.22	1.53	1.00
	42	26.48	0.8492	1.496	15	25.36	0.0093	1.122	14	1.122	91.31	1.33	1.00
Barbara	17	42.11	0.8662	1.564	1	42.11	0.008	0.905	1	0	108.28	1.73	0.00
	22	38.13	0.8909	1.574	2	38.13	0.0076	0.864	2	0	117.22	1.82	0.00
	27	34.70	0.8872	1.561	4	33.98	0.0083	0.911	3	0.725	106.89	1.71	1.00
	32	31.40	0.8958	1.555	6	30.72	0.0082	1.083	6	0.682	109.24	1.43	0.00
	37	28.58	0.8713	1.528	10	27.79	0.0087	0.870	9	0.791	100.15	1.76	1.00
	42	25.98	0.8543	1.505	13	24.84	0.0084	0.905	12	1.145	101.70	1.66	1.00
Average										0.65	105.17	1.62	0.38

5. Conclusion

A parallel architecture for accelerating the HEVC intra-frame computation based on CUDA has been implemented in this study. The proposed implementation uses parallel processing for evaluating 35 different HEVC intra prediction modes, including 33 angular modes, 1 planar mode, and 1 DC mode, for 32 × 32 coding units. Thus, the HEVC intra mode evaluation process, which requires considerable computational resources, has been moved to the GPU platform.

The experimental results conducted on the NVIDIA Jetson Xavier NX platform demonstrate that the proposed implementation achieves significant acceleration at the GPU kernel level. With the use of DCT Method 2 and shared memory optimizations, the average kernel acceleration is 105.17× compared to the sequential CPU implementation. However, in case of considering the CPU–GPU data transfer, synchronization and result collection operations, the average end-to-end acceleration rate reaches 1.62×. Therefore, one can conclude that data transfer and synchronization overheads remain the main limitations of the proposed solution.

The image quality was measured by means of PSNR metric. It has been demonstrated that the average PSNR difference is about 0.651 dB between the CPU implementation in the serial mode and the GPU implementation in the parallel mode. Such small difference in PSNR values is determined by implementation-related factors including rounding, quantization, and parallel computation order. Thus, the proposed method allows achieving a good tradeoff between computation speedup and image quality in the context of HEVC intra-frame computation on GPU platforms.

In the comparison of the two GPU-based DCT implementations, it can be stated that both methods allow increasing the execution performance. The highest efficiency is provided by DCT Method 2 in combination with the shared memory optimization. Thus, one can see

that not only the structure of the algorithm itself but also its adaptation to GPU memory structure has a direct influence on the algorithm performance.

It is important to emphasize that the proposed architecture should be viewed as an accelerated intra-frame computation module but not as an entire HEVC video encoder.

A disadvantage of the current experimental study is that the image set used contains four reference images of a constant size of 512 × 512 pixels. The controlled experiment setting was chosen to allow for the proper comparison of CPU and GPU performance on the same embedded platform. Still, evaluation through use of a greater number of images and different resolutions would provide some additional information about the performance and scalability of the suggested architecture. Therefore, it is one of the important areas for future research.

For further research, the reduction of the overhead of data transfer from CPU to GPU memory is planned to be done with the help of asynchronous memory transfers, CUDA streams, unified memory, and mode selection at the GPU side. Moreover, the extension of the proposed framework towards other HEVC components including inter prediction and entropy coding is considered.

Author Contributions

The percentages of the authors' contributions are presented below. All authors reviewed and approved the final version of the manuscript.

	M.K.	A.A.
C	60	40
D	60	40
S	10	90
DCP	80	20
DAI	70	30
L	70	30
W	70	30
CR	40	60
SR	70	30
PM	60	40

C= concept. D= design. S= supervision. DCP= data collection and/or processing. DAI= data analysis and/or interpretation. L= literature search. W= writing. CR= critical review. SR= submission and revision. PM= project management.

Conflict of Interest

The authors declared that there is no conflict of interest.

Ethical Consideration

Ethics committee approval was not required for this study because of there was no study on animals or humans.

References

Akman, A., & Cekli, S. (2023). Efficient 2D DCT architecture based on approximate compressors for image compression with HEVC intra-prediction. *Journal of Real-Time Image Processing, 20*(22). https://doi.org/10.1007/s11554-023-01261-3

Bloch, A., Casale-Brunet, S., & Mattavelli, M. (2023). Design space exploration for partitioning dataflow programs on CPU + GPU heterogeneous systems. *Journal of Signal Processing Systems, 95*. 1219–1229. https://doi.org/10.1007/s11265-023-01884-6

Bross, B., Han, W.-J., Sullivian, G. J., Ohm, J.-R., & Wiegand, T. (2023). *High Efficiency Video Coding (HEVC) text specification draft 9*. http://phenix.itsudparis.eu/jct/doc_end_user/current_document.php?id=6803 (accessed on 15 June 2023).

Ghorpade, J., Parande, J., Kulkarni, M., & Bawaskar, A. (2012). GPGPU processing in CUDA architecture, *arXiv preprint arXiv:1202.4347*.

Harris, M. (2019). How to optimize data transfers in CUDA C/C++. *NVIDIA Developer Blog*. https://devblogs.nvidia.com/how-optimize-data-transfers-cuda-cc/

Imen, W., Amna. M., Fatma. B., Ezahra. S. F., & Masmoudi. N. (2022). Fast HEVC intra-CU decision partition algorithm with modified LeNet-5 and AlexNet. *Signal. Image and Video Processing. 16*(7). 1811–1819. https://doi.org/10.1007/s11760-022-02139-w

James, R., Abo-Zahhad. M., Inoue, K., & Sayed, M. S. (2024). Performance evaluation of all intra Kvazaar and x265 HEVC encoders on embedded system NVIDIA Jetson platform. *Journal of Real-Time Image Processing. 21*(67). https://doi.org/10.1007/s11554-024-01429-5

Jingjing, Y., & Kehua, J. (2019). A fast intra prediction algorithm with simplified prediction modes based on utilization rates. *2019 IEEE/ACIS 18th International Conference on Computer and Information Science (ICIS)*. 225–229. Beijing. China.

Juncheng, M., Falei, L., Shanshe, W., Nan. Z., & Siwei. M. (2015). Parallel intra coding for HEVC on CPU plus GPU platform. *2015 Visual Communications and Image Processing (VCIP)*, 1–4.

Kaplan, M., & Akman, A. (2024). Parallel implementation of Discrete Cosine Transform (DCT) methods on GPU for HEVC. In M. N. Seyman (Ed.). *2nd International Congress of Electrical and Computer Engineering: ICECENG 2023*. EAI/Springer Innovations in Communication and Computing, Springer. https://doi.org/10.1007/978-3-031-52760-9_20

Khemiri, R., Kibeya, H., Sayadi, F. E., Bahri, N., Atri, M., & Masmoudi, N. (2018). Optimization of HEVC motion estimation exploiting SAD and SSD GPU-based implementation, *IET Image Processing, 12*(2), 243–253. https://doi.org/10.1049/iet-ipr.2017.0474

Li, W., Ou, W., Wang, H., Wu, Y., & Zhong, Y. (2025). Jointly probability-based fast intra prediction algorithm for spatial SHVC. *EURASIP Journal on Image and Video Processing*, 2025, 20. https://doi.org/10.1186/s13640-025-00683-2

Lini X., Sun, S., Huang, W., Sheng, B., Li, P., & Feng, D. D. (2023). EAPT: Efficient attention pyramid transformer for image processing. *IEEE Transactions on Multimedia, 25*, 50–61. https://doi.org/10.1109/TMM.2021.3120873

Lopez-Granado, O., Migallon, H., Alcocer Espinosa, E., Gutierrez, R., Van Wallendael, G., & Malumbres, M. P. (2025). Performance, limitations, and design issues of the integration of a hardware-based IME module with HEVC video encoder software. *IEEE Access*, 13, 113390–113406.

https://doi.org/10.1109/ACCESS.2025.3581961

Mate, Č., Alen, D., Leon, D., Igor, P., & Mario, K. (2020). Performance engineering for HEVC transform and quantization kernel on GPUs. *Automatika, 61*(3), 325–333. https://doi.org/10.1080/00051144.2020.1752046

NVIDIA. (2019). CUDA refresher: The CUDA programming model, *NVIDIA Developer Blog*. https://developer.nvidia.com/blog/cuda-refresher-cuda-programming-model/ (accessed on 18 July 2023).

NVIDIA. (2023). *NVIDIA Jetson Xavier NX developer kit*. https://openzeka.com/wpcontent/uploads/2020/03/Jetson_Xavier_NX_Developer_Kit-One-Pager.pdf (accessed on 23 July 2023).

Pietri, N., Nocent, O., Robert, P., & Schmutz, A. (2025). Real-time live compression of dynamic 3D meshes for client-side GPU cloud gaming using skinning decomposition, *The Visual Computer*, Advance online publication. https://doi.org/10.1007/s00371-025-04112-3

Radicke, S., Hahn, J., Wang, Q., & Grecos, C. (2016). A parallel HEVC intra prediction algorithm for heterogeneous CPU+GPU platforms. *IEEE Transactions on Broadcasting. 62*(1), 103–119.

Rathee, N., Sharma, T., Yadav, J., et al. (2026). High-performance FPGA implementation of HEVC with dynamic programming for optimized intra-frame prediction. *Scientific Reports*, 16, 21241. https://doi.org/10.1038/s41598-026-51672-y

Ryoo, S., Rodrigues, C., Baghsorkhi, S, S., et al. (2008). Optimization principles and application performance evaluation of a multithreaded GPU using CUDA. *Proceedings of the 13th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP)*, 73–82. ACM.

Singhadia, A., Bante, P., & Chakrabarti, I. (2019). A novel algorithmic approach for efficient realization of 2-D-DCT architecture for HEVC, *IEEE Transactions on Consumer Electronics, 65*(3), 264–273.

Tariq, J. (2020). High-performance intra-mode accelerator for HEVC, *The Visual Computer, 36*(8), 1603–1617. https://doi.org/10.1007/s00371-019-01764-w

Thurley, M, J., & Danell, V. (2012). Fast morphological image processing open-source extensions for GPU processing with CUDA, *IEEE Journal of Selected Topics in Signal Processing*, 849–855.

Wang, S., Liu, J., Zhang, H., & Hu, Y. (2025). A fast algorithm of HEVC intra mode decision. *Journal of Shanghai University (Natural Science Edition)*, 31(3), 561–570. https://doi.org/10.12066/j.issn.1007-2861.2639

Zhu, J., Zhang, Q., Fei, L., Cai, R., Xie, Y., Sheng, B., & Yang, X. (2023). FFFN: Frame-by-frame feedback fusion network for video super-resolution, *IEEE Transactions on Multimedia, 25*, 6821–6834. https://doi.org/10.1109/TMM.2022.3214776