https://doi.org/10.53656/math2026-1-2-fae

Science in Education
Научно-методически статии

FAST ADAPTIVE ENTROPY-DRIVEN MULTILEVEL IMAGE SEGMENTATION USING LOOK-UP TABLES ON GPU PLATFORMS

Georgi Petrov
Department of Telecommunications,
New Bulgarian University,
Sofia (Bulgaria)

Abstract. This paper proposes a fast, adaptive method for multilevel image segmentation based on Kapur's entropy. Implemented on CPU and GPU, the method achieves up to 40× acceleration using CUDA and memory optimization. Segmentation quality is preserved while enabling real-time processing of large image batches. The experiments were carried out using the university's high-performance computing (HPC) mini-cluster, demonstrating its role as an educational and research platform. The approach is scalable and suited for scientific and educational use. It builds upon earlier work on multidimensional histogram analysis, applying entropy-driven modeling to image segmentation.

Keywords: entropy; multilevel thresholding; GPU; LUT

1. Introduction

The current work builds on prior research on multidimensional image analysis and target tracking (Iliev et al., 2014a; 2014b), in which dynamic histogram-based descriptors were employed for object detection in spatio-temporal scenes. The presented multilevel Kapur's entropy on GPU implementation experiment is also part of the ongoing development of the New Bulgarian University high-performance computing (HPC) (Petrov et al. 2024) mini-cluster and serves as a practical experimental framework for the migration of existing educational and scientific source code to high-performance heterogeneous computing architectures using different

optimization strategies. Teaching activities related to CUDA technologies began at New Bulgarian University during the 2021 – 2022 academic year with the master-level course TCMM159 Seminar: Parallel Processing of Signals and Images with CUDA, which is now offered in the core bachelor curriculum under the code TCMB414 Microprocessors, Microcontrollers and Heterogeneous Computing Architectures.

Multilevel thresholding is a widely used technique in image segmentation (Sezgin & Sankur, 2004), especially in applications such as scientific imaging, biomedical diagnostics, and object recognition. Entropy-based methods, such as Kapur's criterion, are particularly effective when dealing with complex histograms, as they aim to maximize information content across segmented regions. However, traditional implementations are computationally expensive and unsuitable for real-time use.

This paper addresses that limitation by introducing a recursive entropy-based segmentation algorithm optimized with precomputed look-up tables (LUTs) and prefix sums on both CPU and GPU platforms. The method adaptively determines thresholds without prior knowledge of the number of segments. A related principle was proposed for histogram approximation by Gaussian components (Dimov et al., 2009), which similarly avoids predefined class counts and motivates recursive decomposition. Leveraging GPU parallelism and shared-memory optimization, the algorithm achieves significant speedup while preserving segmentation quality. The approach is suitable for batch processing and educational applications requiring low-latency visual feedback and continues the tradition of entropy-based image analysis in Bulgarian research (Gluhchev et al., 2021).

The remainder of this paper is structured as follows: Section 2 reviews related work. Section 3 presents the proposed method and implementation details. Section 4 discusses experimental results and performance benchmarks. Finally, Section 5 concludes the study and outlines directions for future research. Section 6 covers evaluated experimental results and error analysis.

2. Background and Related Work

Earlier work established a foundation for motion detection and target tracking in dynamic visual scenes (Iliev et al., 2007; 2014a; 2014b). Related

applications in the medical domain were demonstrated in (Mihova et al., 2015), where 3D histogram techniques and open-source frameworks were used for DICOM image preprocessing.

Kapur's entropy method (Kapur et al., 1985) maximizes the cumulative entropy across segments and is particularly effective for non-bimodal or low-contrast images, where classical methods such as Otsu's variance-based thresholding (Otsu, 1979) may fail. A more recent generalization based on non-additive Tsallis entropy was proposed by (Wang & Fan, 2024), showing improved adaptability to complex histograms.

Recent studies attempt to overcome this bottleneck using metaheuristic search methods, good example of different optimization strategies for educational use. For example, (Kumar et al., 2020) applied Whale Optimization to underwater image segmentation using Kapur's criterion, while (Maiti et al., 2021) explored hybrid multilevel thresholding for biomedical images. Similarly, (Sharma et al., 2023) proposed an improved Kapur-based algorithm for color image segmentation but did not explore hardware acceleration.

To address these limitations, we introduce a recursive multilevel thresholding framework based on Kapur's entropy, optimized with precomputed look-up tables (LUTs) and parallel CPU/GPU execution using CUDA and CuPy. Unlike prior work, the new adaptive method avoids heuristic search entirely, offering fast, deterministic segmentation and scalability to large batches of images. To the best of our knowledge, this is the first direct GPU implementation of entropy-based segmentation with shared-memory histogram construction and LUT-based entropy computation.

3. Methodology

The proposed segmentation method is based on recursive multilevel thresholding using Kapur's entropy criterion, with GPU acceleration and LUT optimization. The process begins by computing the normalized histogram of a grayscale image. A recursive function is then applied to iteratively divide the histogram at threshold levels that maximize the sum of entropies in the resulting subregions.

3.1. Entropy Criterion and Recursive Strategy

Kapur's entropy defines the optimal threshold as the one that maximizes the sum of Shannon entropies of the foreground and background pixel distributions. For a histogram $H = \{p_0, p_1, \dots, p_{L-1}\}$, where p_i denotes the normalized probability of gray level i, the entropy for a threshold t is given by:

$$E(t) = -\sum_{i=0}^{t} \frac{p_i}{P_1} log_2\left(\frac{p_i}{P_1}\right) - \sum_{i=t+1}^{L-1} \frac{p_i}{P_2} log_2\left(\frac{p_i}{P_2}\right)$$

where $P_1 = \sum_{i=0}^{t} p_i$ and $P_2 = \sum_{i=t+1}^{L-1} p_i$. The recursive algorithm partitions the histogram at the maximum-entropy threshold, and then applies the same procedure to each subregion until no further gain is achieved or a stopping condition is met.

3.2. Kapur with Cumulative Sums and LUT

The entropy term in Kapur's criterion involves repeated evaluation of $-p.\log_2 p$ for each histogram probability. Directly computing logarithms is expensive, especially on GPU hardware, since the classical implementation requires nested loops with quadratic complexity $O(L2)$, leading to warp divergence and repeated memory access. To avoid this, we precompute an approximation function in a look-up table (LUT) and apply prefix sums (Stokfiszewski et al., 2018). The proposed strategy removes inner loops, reduces the complexity to linear $O(L)$, and allows each threshold to be evaluated with constant-time lookups. Replacing the costly $-p.\log_2(p)$ operations with a precomputed LUT also eliminates expensive transcendental functions. Together, these improvements minimize global memory traffic and enable efficient GPU parallelization, achieving significant speedup while preserving segmentation accuracy.

Given a normalized histogram:

$H = \{p_0, p_1, \dots, p_{L-1}\}$ with $L\,(256 - 4096)$ *bins:* $\sum_{i=0}^{L-1} p_i = 1$

LUT approximation for entropy nonlinear function:

$f(p) = -p.log_2 p$

with a look-up table (LUT) the table of size $M\,(1024-4096)$ is precomputed once:

$LUT[j] = 0$, if $j = 0$

$LUT[j] = -x.log_2 x$, where $x = i/(M-1)$, for $j = 1, 2, \ldots, M-1$

Interpolation in the LUT for arbitrary $\boldsymbol{p} \in [\mathbf{0}, \mathbf{1}]$ and mapping to LUT index:

$x = p.(M-1)$

Finding the integer part and fractional offset:

$i = [x], \alpha = x - i$

Apply **linear interpolation** between LUT entries:

$f(p) \approx (1-\alpha)LUT[i] + \alpha LUT[i+1]$

Calculating prefix sums using the LUT approximation f(p), define cumulative arrays computed once per histogram:

Mass prefix sum: $S[k] = \sum_{i=0}^{k} p_i$

Entropy prefix sum: $SL[k] = \sum_{i=0}^{k} f\,(p_i)$

Left and right partitions for threshold t

For a candidate threshold $t\ (1 \le t \le L-2)$:

Left mass: $P_L(t) = S[t]$

Right mass: $P_R(t) = S[L-1] - S[t]$

Left entropy sum: $E_L(t) = SL[t]$

Right entropy sum: $E_R(t) = SL[L-1] - SL[t]$

Calculating the Kapur entropy

We take original formula to calculate Kapur entropy, expanding $\log_2 p_i / P_L = \log_2 p_i - \log_2 P_L$ and make substitutions using above terms: P_L is already calculated constant for all i, where E_L is taken from definition of f(p):

Original Kapur left entropy:

$H_L(t) = -\sum_{i=0}^{t} p_i/P_L * log_2\, p_i/P_L$, where $P_L = \sum_{i=0}^{t} p_i$ is constant

Expanding the logarithm of a ration:

$$log_2\, p_i/P_L = log_2 p_i - log_2 P_L$$

Substitute:

$$H_L(t) = (-1/P_L).\sum_{i=0}^{t} p_i\,(log_2 p_i - log_2 P_L)$$

Simplifying the sum:

$$H_L(t) = (-1/P_L).\sum_{i=0}^{t} p_i\,.log_2 p_i + (1/P_L).\sum_{i=0}^{t} p_i * log_2 P_L$$

To simplify the expression, we use two facts: $\log_2 P_L$, is constant and can be factored out, and the summation of p_i is equal to P_L.

$$(1/P_L).\sum_{i=0}^{t} p_i\, log_2 P_L = (1/P_L).\left(log_2 P_L.\sum_{i=0}^{t} p_i\right) = (1/P_L).log_2 P_L.P_L = log_2 P_L$$

Introducing entropy prefix sum:

$$\sum_{i=0}^{t} p_i\, log_2 p_i = -E_L(t)$$

Left *entropy:*

$$H_L(t) = log_2 P_L(t) + \frac{E_L(t)}{P_L(t)}, \text{ if } P_L > 0$$

Right entropy:

$$H_R(t) = log_2 P_R(t) + \frac{E_R(t)}{P_R(t)}, \text{ if } P_R > 0$$

Total entropy:

$$H(t) = H_L(t) + H_R(t)$$

In this way a better performance is achieved, significantly lowering the constant factor and enabling real-time implementations on both CPU and GPU platforms.

3.3. CPU/GPU Acceleration

The algorithm is implemented in a modular framework supporting both CPU and GPU execution. The CPU version in C/Python (NumPy) uses the same entropy LUT strategy to avoid redundant logarithmic operations, while the GPU implementation (CUDA/CuPy) achieves higher throughput through shared memory and kernel-level parallelism. The CPU version remains suitable for embedded and educational use, illustrating code portability and performance scaling across architectures. The LUT is initialized once and reused across recursive calls to ensure consistency.

A high-performance CPU version was developed in C with targeted optimizations for Intel Xeon processors. Techniques such as **loop unrolling (×4)**, cache-aware access, and precomputed entropy LUTs improved execution time by up to **77%** compared to the naïve version. Further unrolling (×8) gave no additional gain due to cache saturation. SIMD vectorization (#pragma omp simd) showed limited benefit without specialized math libraries. Similar LUT-driven acceleration has been reported in **GPU texture memory** (Zhang et al. 2013), CGH rendering (Pan et al., 2009), and **entropy-based classification** (Józefowicz & Czarnecki, 2015). The proposed method remains **deterministic, precise, and scalable** across hardware platforms.

3.4. Scalability and Determinism

Unlike metaheuristic approaches, the proposed method is deterministic and scalable to large batches of images. It does not require prior knowledge of the number of segments, and its recursive structure allows for adaptive segmentation depth depending on the input histogram complexity.

3.5. Pseudocode: GPU-based 2D/3D Kapur Entropy Segmentation

```
Input: grayscale or RGB image I, number of bins B
1. Transfer image I to GPU memory.
2. Launch CUDA kernel hist2d_combined:
       For each pixel (x,y) in I:
           v1 = intensity(x,y)
           If x < width-1:
                v2 = intensity(x+1,y)
```

```
                atomicAdd(H[v1,v2], 1)
            If y < height-1:
                v3 = intensity(x,y+1)
                atomicAdd(H[v1,v3], 1)
   Output: joint histogram H of size B×B
3. Compute row-wise entropy (parallel on GPU):
       For each row r in H:
           S_left  = cumulative_sum(H[r,:])
           L_left  = cumulative_sum(H[r,:] * log2(H[r,:]))
           For each threshold t:
               H_left           =      log2(S_left[t])      -
L_left[t]/S_left[t]
               H_right = log2(S_total-S_left[t]) - (L_total-
L_left[t])/(S_total-S_left[t])
               H_row[r,t] = H_left + H_right
4. Compute column-wise entropy in same manner (on H^T):
       H_col[c,t] = ...
5. Combine: H_final = H_row + H_col
6. Locate thresholds:
       For each row: argmax(H_row[r,:])
       For each col: argmax(H_col[:,c])
       Global: argmax(H_final)
7. (Optional) visualize results:
       - 2D heatmaps and 3D surface plot of histogram and
entropy maps

Output: entropy maps (H_row, H_col, H_final), optimal
thresholds
```

4. Results and Discussion

The proposed method was evaluated on a dataset of grayscale images with varying levels of histogram complexity, ranging from bimodal medical scans to low-contrast aerial surveillance frames. Performance was compared across three implementations: a baseline NumPy version on CPU, an optimized C version for Intel Xeon E-2176M, and a GPU-accelerated CUDA version using CuPy. The proposed GPU-accelerated segmentation achieved a consistent 8× to 9× speedup over the CPU implementation, reducing average processing time from ~160 ms to ~20 ms across multiple LUT sizes and segmentation levels.

4.1. Segmentation Quality

Visual inspection confirmed that the recursive entropy-based approach successfully captured perceptual zones across all test cases. The method does

not require prior knowledge of the number of thresholds, which makes it robust to content variability.

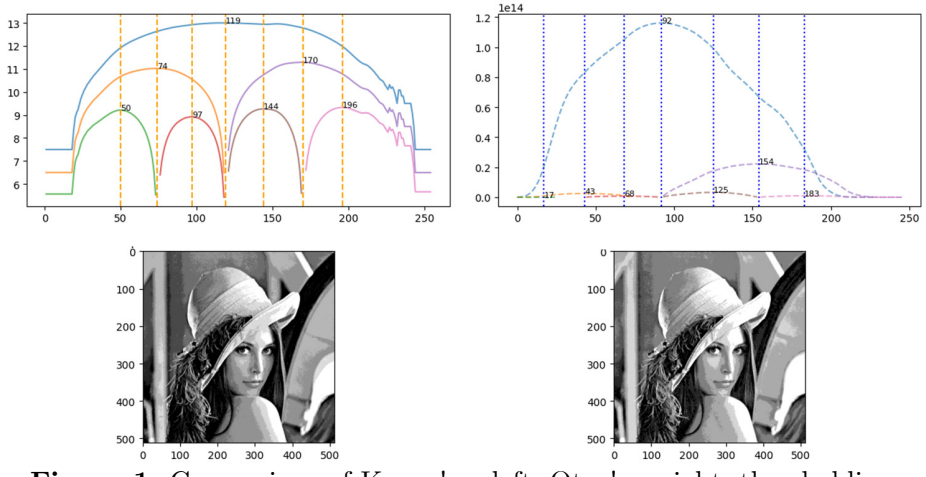

Figure 1. Comparison of Kapur's – left, Otsu's – right, thresholding across 8 level segmentation to the standard "Lena" image

4.2. Execution Time and Speedup

Table 1. CPU Execution Time with and without LUT

LUT size (KiB)	Without LUT (ms)	With LUT ×4 (ms)	Speed Gain	Δ (Entropic Error)
2	446.2	98.7	77.8	0
4	446.7	98.9	77.8	0
8	448.1	99.0	77.9	0
16	451.1	99.0	78.1	0
32	453.2	99.1	78.2	0

The GPU implementation outperformed both CPU variants in runtime. On a GeForce RTX 3060, it achieved up to 40× acceleration over the NumPy baseline and 17× speedup compared to the optimized C version. The histogram and entropy LUT computations were fully parallelized using CUDA shared memory and warp-level optimization. The C implementation on an Intel Xeon E-2176M, though sequential, achieved 77% faster execution

through loop unrolling, cache-aware access, and LUT reuse. For educational purposes, both optimization strategies – CPU and GPU-based acceleration—are **demonstrated as complementary examples in training sessions on HPC programming**.

4.3. Memory Usage and Scalability

The GPU version achieved 10–20× acceleration over the CPU with identical segmentation quality. Its shared-memory kernel and deterministic entropy computation make it suitable for embedded and high-throughput imaging. In the **training context**, the focus is not only on performance but also on memory optimization in GPU architectures, demonstrating efficient reuse of buffers and LUTs for scalable, real-time processing.

4.4. Comparison with Related Work

Compared to heuristic metaoptimization approaches (e.g., Whale Optimization by (Kumar et al., 2020; Sharma et al., 2023), our method is significantly faster and fully deterministic. Previous GPU-based entropy models (Zhang et al., 2013; Pan et al., 2009) focused on non-visual domains (molecular simulations or CGH), while our work is specialized for adaptive image segmentation. CPU-based approximations such as those by (Józefowicz & Czarnecki, 2015) apply entropy simplification, whereas we achieve speedup without compromising entropy accuracy.

4.5. Performance Comparison on CPU Using OpenMP, SIMD, and LUT Optimization

Table 2. Loop x8 and OpenMP Optimization on CPU

LUT size	**Loop x8 (ms)**	**OpenMP (ms)**	**Speed Gain**
2048	44.21	29.08	+51.9%
8192	55.54	22.46	+109%

Loop-unrolled C tests achieved up to **78% speedup** with no loss of accuracy, confirming the efficiency of LUT + unroll on scalar CPUs. Comparative tests with OpenMP, SIMD, and LUT approaches showed the LUT method as fastest and most stable. These experiments are used in **training sessions on CPU optimization techniques**, illustrating practical gains from parallelization, cache-aware access, and loop-level

tuning, **resulting in more energy-efficient and resource-optimized code**.

5. Experimental results and Error Analysis

To evaluate the proposed image segmentation, many extensive tests were conducted on a diverse dataset of grayscale and color images with varying textures, content, and resolutions. The dataset included standard test images (e.g., *Lena*, *Cameraman*), natural photographs, and high-resolution scenes ranging from 256×256 up to 4128×3096 pixels from video sequences. Figure 2 illustrates the error curve as a function of linear LUT size.

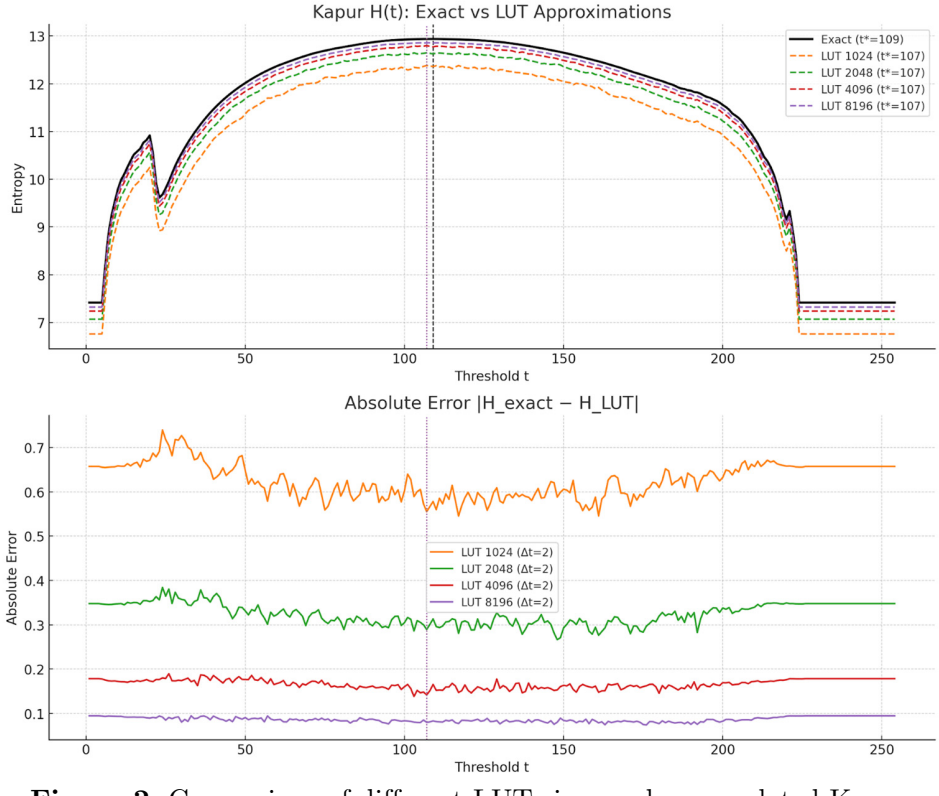

Figure 2. Comparison of different LUT sizes and accumulated Kapur entropy errors, Exact was calculated in clasic math.h $\log_2$ operations

Table 3. Comparison of exact $\log_2(p)$ and LUT with sizes = {512-8196} under different pixel count {1-1,000M}

LUT size	**Average error (for N ≈ 50Mpix)**	**Speedup**
512	~0.000480	~+49%
1024	~0.000530	~+47%
2048	~0.000010	~+46%
4096	~0.000010	~+46.5%
8192 (float32)	~0.000005	~+46%
16384 half precision (float16)	~0.000009	~+46%

When using single-precision floats, numerical instability occurs at the edges of the entropy function due to very small probabilities. To mitigate this, the entropy is computed symmetrically from the left half to the center and from the right half to the center. The results show that the **Classic** and **LUT** methods remain nearly constant in execution time (~32 ms and ~51 ms, respectively) on CPU and GPU, regardless of the number of segments, since they repeatedly recompute entropy for each threshold.

In contrast, the **Cumulative Prefix Sums** method demonstrates significantly lower execution times for small numbers of segments (only 2.36ms at 16 levels), see Fig. 3. Performance was tested on images of varying resolutions, from 256×256 up to 8192×8192 pixels, and under different segmentation depths (2–128 regions). In real applications, segmentation with over 64 levels is impractical.

The results demonstrated that the GPU implementation consistently outperforms the CPU baseline, achieving speedups between 12× and 70×, with the highest gains observed for high-resolution images.

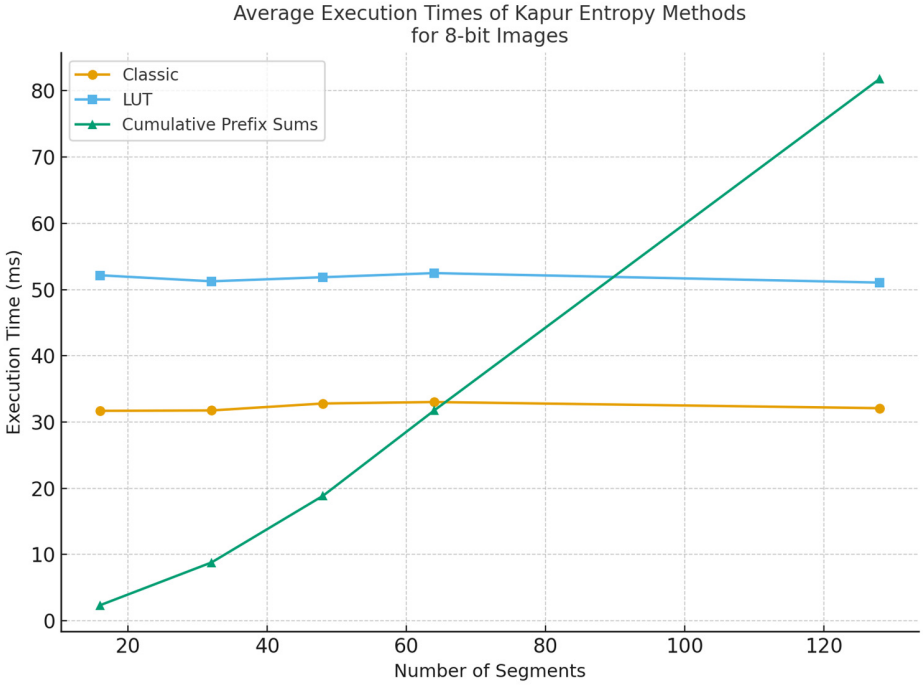

Figure 3. The graph illustrates the average execution times for Classic, LUT-based, and Cumulative Prefix Sums versions

Table 4. Average processing time per 1 MPx (CPU/GPU)

Regions	**CPU (ms/MPx)**	**GPU (ms/MPx)**	**Speedup**	**Diff (%)**
2	23.31	1.41	16.5	0.0001
4	23.58	1.45	16.3	0.0003
8	24.98	1.58	15.8	0.0047
16	25.63	1.65	15.5	0.0894
32	26.58	1.86	14.3	0.1209
64	28.22	2.20	12.8	0.1636

Table 5. Percentage speedup depending on image size

Size	Regions	CPU Time Exact (ms)	GPU Time Exact (ms)	GPU Time LUT (ms)	Speedup EXACT – LUT (%)
8192×8192	64	2456.91	78.41	35.88	96 – 98
4096×4096	64	634.00	34.40	16.43	94 – 97
4096×4096	8	385.00	12.90	6.25	96 – 98
2048×2048	8	94.00	5.20	3.12	94 – 96
1024×1024	8	24.00	2.40	1.89	90 – 92

The results also indicate that the **LUT approximation can be flexibly used depending on the hardware constraints**, although it slightly increases computation time compared to the exact GPU logarithm (1–2%).

Alongside LUT and prefix-sum acceleration, alternative strategies were tested on Arduino-based boards. The **bit-trick approximation** (Schraudolph, 1999) exploits IEEE-754 structure for fast $\log_2(p)$ estimation with minimal hardware, while **CORDIC** (Volder, 1959) offers iterative, shift-based computation suitable for low-power FPGA and edge systems. These experiments emphasize **training on optimizing energy-intensive mathematical operations** for IoT platforms.

6. Conclusions

This paper presented a fast, adaptive framework for multilevel image segmentation based on Kapur's entropy criterion with recursive thresholding and precomputed LUTs.

The method has strong educational value, illustrating optimization techniques for both CPU and GPU platforms and serving as a teaching tool in courses on parallel and heterogeneous computing. It proved to be an effective example particularly relevant for energy-efficient IoT systems.

Future work includes extending the framework to hybrid entropy models. The deterministic and scalable nature of the algorithm supports its use in both research and academic training, and it may

be combined with deep architectures such as U-Net and their fuzzy extensions (Kirichev et al., 2021) to balance processing speed and segmentation accuracy.

REFERENCES

Dimov, D. T., Todorov, M. D. & Christov, C. I. (2009). Histogram Optimal Multi-thresholding. *AIP Conference Proceedings*, 1186, 187 – 194. Melville, NY: American Institute of Physics. DOI: 10.1063/1.3265354

Gluhchev, G., Dimov, D.T. & Ouzounov, A. (2021). Pattern Recognition Development in Bulgarian Academy of Sciences. *Research in Computer Science in the Bulgarian Academy of Sciences, Studies in Computational Intelligence*, vol. 965. Springer, Cham, 105 – 123. DOI: 10.1007/978-3-030-72284-5_7

Iliev, P., Tzvetkov, P. & Petrov, G. (2014a). Multi-Dimensional Dynamic Scene Analysis Using 3D Image Histogram and Entropy Sequences Analysis. *International Journal of Computing*, *6*(1), 35 – 43. DOI: 10.47839/ijc.6.1.422

Iliev, P., Tzvetkov, P. & Petrov, G. (2014b). Multi-Dimensional Dynamic Scene Analysis: Multidimensional Image Object Segmentation and Target Tracking. *International Journal of Computing*, *6*(3), 30 – 37. DOI: 10.47839/ijc.6.3.448

Józefowicz, R. & Czarnecki, W. M. (2015). Fast Optimization of Multithreshold Entropy Linear Classifier. *Studia Informatica*, *36*(2), 49 – 64. DOI: 10.4467/20838476SI.14.005.3022

Kapur, J. N., Sethi, I. C. & Sujan, K. C. (1985). A new method for gray-level picture thresholding using the entropy of the histogram. *Computer Vision, Graphics, and Image Processing*, *29*(3), 273 – 285. DOI: 10.1016/0734-189X(85)90125-2

Kumar, A., Mishra, S., Singh, A. & Singh, P. (2020). Kapur's Entropy for Underwater Multilevel Thresholding Image Segmentation Based on Whale Optimization. *International Journal of Advanced Trends in Computer Science and Engineering*, *9*(3), 2850 – 2857.

lKirichev, M., Slavov, T. & Momcheva, G. (2021). Fuzzy U-Net Neural Network Architecture Optimization for Image Segmentation. *IOP Conference Series: Materials Science and Engineering*. 1031. 012077. 10.1088/1757-899X/1031/1/012077.

Laskov, L. & Dimov, D. (2007). Color Image Segmentation for Neume Note Recognition. *Proceedings of the International Conference A&I'07*, Sofia, 3 – 6 October 2007. III.37 – 41.

Mihova, P., Petrov, G. & Andonov, F. (2015). Applications of Open Source Frameworks for Advanced Medical Image Processing. *Bulgarian Journal of Public Health*, *7*(1), 69 – 77.

Otsu, N. (1979). A threshold selection method from gray-level histograms. *IEEE Transactions on Systems, Man, and Cybernetics*, *9*(1), 62 – 66. DOI: 10.1109/TSMC.1979.4310076

Pan, Y., Xu, X., Solanki, S., Liang, X., Tanjung, R., Tan, C. & Chong, T. C. (2009). Fast CGH computation using SLUT on GPU. *Optics Express*, *17*(21), 18543 – 18555. DOI: 10.1364/OE.17.018543

Petrov G. K., Pasarelski R. I. & Angelov K. K. (2024), University parallel computing micro-cluster build with second-hand computer systems. *Electrotechnica & Electronica (E+E)*, *59*(2 – 4), 46 – 53, ISSN: 0861-4717 (Print), 2603-5421 (Online)

Sezgin, M. & Sankur, B. (2004). Survey over image thresholding techniques and quantitative performance evaluation. *Journal of Electronic Imaging*, *13*(1), 146 – 165. DOI: 10.1117/1.1631315

Sharma, R., Kumar, D. & Sharma, V. (2023). Improved Kapur's Entropy Algorithm for Color Image Segmentation. *Proceedings of the Amity International Conference on Artificial Intelligence and Machine Learning (AICAIML)*, India.

Schraudolph, N. N. (1999). A Fast, Compact Approximation of the Exponential Function. *Neural Computation*, *11*(4), 853 – 862, 1999.

Stokfiszewski, K., Puchala, D., & Yatsymirskyy, M. (2018). Effectiveness of GPU Realizations of Parallel Prefix-Sums Computation Algorithms. *2018 IEEE 13th International Scientific and Technical Conference on Computer Sciences and Information Technologies (CSIT)*, *1*, 436 – 439.

Volder, J. E. (2009). The CORDIC trigonometric computing technique. *IRE Transactions on electronic computers*, (3), 330 – 334.

Wang, S. & Fan, J. (2024). Image thresholding method based on Tsallis entropy correlation. *Multimedia Tools and Applications*, *84*, 9749 – 9785. DOI: 10.1007/s11042-024-19332-3

Zhang, Q., García, J. M., Wang, J., Hou, T. & Pérez-Sánchez, H. (2013). A GPU-Based Conformational Entropy Calculation Method. *International Work-Conference on Bioinformatics and Biomedical Engineering (IWBBIO)*, Granada, 18 – 20 March 2013, 735 – 743.

✉ **Dr. Georgi Petrov, Assoc. Prof.**
ORCID iD: 0000-0001-8735-4677
Department of Telecommunications
New Bulgarian University
21, Montevideo Blvd.
1618 Sofia, Bulgaria
E-mail: gpetrov@nbu.bg