Information, Computing and Intelligent Systems, 2026, No. 8, 4 – 17
UDC 004.8:004.94 https://doi.org/10.20535/2786-8729.8.2026.356552

IMPLEMENTATION OF DIFFERENCE OF GAUSSIANS BASED ON ALL-PASS IIR FILTER WITH C-SLOWING

Artemii Vinokurov*
https://orcid.org/0009-0006-7861-5138

Anatoliy Sergiyenko
https://orcid.org/0000-0001-5965-1789

National Technical University of Ukraine
"Igor Sikorsky Kyiv Polytechnic Institute", Kyiv, Ukraine

*Corresponding author: a.vinokurov@kpi.ua

Received: 31 March 2026 / Accepted: 11 May 2026 / Published: 28 May 2026

The research analyses the computational efficiency of image processing algorithms on GPUs. The primary focus of this work is the Difference of Gaussians (DoG) stage of the scale invariant feature transform algorithm. The study aims to overcome existing computational time limitations associated with Finite Impulse Response (FIR) implementations. To solve this, both Infinite Impulse Response (IIR) with all-pass filtering applied and C-slow retiming are proposed. An experimental framework was developed and tested on NVIDIA CUDA GPUs: RTX 3090, 4090, and 5090. Thus, FIR convolution kernels were compared with the IIR implementations using various C-slowing factor values. Experimental results confirmed theoretical modelling that FIR computation time scales with the kernel size, whereas IIR computation time is independent of it. Therefore, IIR provides lower computational time and higher throughput without regression in visual quality. Further experiments revealed that C-slow retiming introduces overhead that outweighs its benefits and limits its use on modern high-end GPUs, contrary to theoretical expectations. However, using C-slow retiming on low-performance devices could yield more beneficial results. The scientific novelty lies in the experimentally validated use of all-pass IIR filters as a computationally lower alternative to traditional approaches for real-time DoG implementation on modern GPU architectures.

Keywords: synchronous data flow, c-slow retiming, graphics processing unit, scale invariant feature transform, difference of gaussians, infinite impulse response, computer vision.

1. Introduction

The rapid development of high-resolution real-time computer vision systems has introduced a substantial demand for highly efficient image-oriented processing units. Whereas modern Graphics Processing Units (GPUs) provide significant parallelisation capabilities, a contradiction between software solutions and hardware limitations remains. Specifically, real-world implementations often fail to achieve notable computational resource occupancy due to architecturally induced memory throughput bottlenecks and high register pressure [1]. Modern-world applications focus on the development of edge AI and autonomous systems. Therefore, optimising the foundational algorithm, the Scale Invariant Feature Transform (SIFT), becomes necessary for system relevance, contrary to the performance preference it appeared to have before.

The Difference of Gaussians (DoG) stage of SIFT is the most computationally expensive part of the algorithm. Existing Finite Impulse Response (FIR) implementations on NVIDIA CUDA architectures suffer from idle warp cycles and inefficient warp switching when handling tight data dependencies. As traditional software strategies focus on higher levels of abstraction, they thus limit their ability to apply more fine-grained optimisation techniques. This leaves a technical gap in the implementation of high-throughput image processing.

Consequently, systematic analysis of instruction-level optimisation and recursive filtering methods became a critical priority for the high-performance computing community. Whereas traditional parallelisation strategies yield limited benefits on modern architectures, investigating the Infinite

Impulse Response (IIR) and retiming methods is essential. These could provide a consistent, scale-independent increase in throughput in upcoming autonomous systems. The relevance of research in this field is emphasised by the need for theoretical and practical frameworks to overcome the latency limits of current hardware. Finally, the topic of this research is defined by the need to ensure that foundational image-processing algorithms can be applied in fast-paced real-time computer vision development.

2. Literature review and problem statement

The FIR filters are used in most of applications due to their properties like guaranteed stability, high performance, high parallelism level. However, IIR filters have the complexity in the number of operations in dozens times less than FIR filters have. The substantial disadvantage of the IIR filter is in the feedbacks of its algorithm which limit its parallelism. The searching of effective scheduling for the IIR algorithm is complex task. Therefore, the modulo scheduling technique [2] was proposed as an improvement to software pipelining for Coarse-Grained Reconfigurable Array machines. This technique allows to the programmer to overlap different loop iterations for memory access latency hiding. Nevertheless, for modern GPUs, which serve as massively parallel units, this approach needs to be re-evaluated.

From a mathematical perspective, there is evidence that rearranging operations so that register accesses are placed differently can optimise the clock period without altering the computation results [3]. This is crucial to support the proposed research's theoretical backing. Taking this into consideration, a practical analysis is required additionally. The implementation of C-slow retiming for Field-Programmable Gate Arrays has existed for some time [4]. The basic idea is to replace a register in the structure with C registers, enabling the interleaving of C instruction sequences that use these registers. As a result, the same structure computes C dataflows in parallel. This provides the reuse of stalling resources due to data dependencies, thereby increasing throughput.

The research comparing of C-slowing with traditional multiprocessor architectures was performed in [5]. It concluded that algorithms with tight sequential dependencies benefit more from the C-slowing technique than from a simple increase in processing core count in terms of resource utilisation. Running identical threads in terms of C-slow designs was further investigated for Single Instruction Multiple Threads (SIMT) architectures [6]. This becomes more relevant given that GPU architectures are fundamentally SIMT-based, especially in modern general-purpose architectures.

The investigated SIFT algorithm serves as a benchmark for computer vision solutions and for assessing their computational effectiveness on specific processing units. Regarding SIFT implementation on GPUs, early versions suffered from significant problems, primarily due to suboptimal memory access patterns. General parallelisation strategies proved unreliable in this context [7]. Another investigation explored tracking feature points on the GPU, with the potential to deepen understanding of the solution. Nonetheless, the author's utilisation concerns have not included the close data-dependency problems of DoG [8].

Optimisations introduced later focused on specific bottlenecks. An investigation introduced a technique based on sophisticated memory tiling that aligns with the GPU's memory hierarchy. Nevertheless, it does not focus on DoG implemented in IIR filtering, which latency issues remain unsolved [9]. More recently, highly efficient GPU-based recursive filtering frameworks have been developed to minimise memory bandwidth usage [10].

Several solutions that focus on CUDA features are also available. PopSift, a real-time implementation focusing on maximum occupancy, was introduced. However, its reliance on the hardware scheduler makes it vulnerable to the same issues as high register pressure [11]. Another work proposed efficient optimisations for parallel tasks, focusing on independent tasks, which are less relevant to the sequence of dependent operations in IIR filtering [12].

Moreover, the software implementation of DoG was thoroughly investigated on multi-core CPUs. A SIFT analysis was performed, providing significant gains through thread parallelism using the Single

Instruction – Multiple Data approach. Nonetheless, it confirmed that reduced occupancy and memory bandwidth remain crucial bottlenecks [13]. Furthermore, there is a significant difference with GPUs: CPU implementations rely heavily on caching, whereas GPUs have a more limited size for this kind of memory.

Furthermore, a promising synchronisation method that involves both the CPU and the GPU was proposed to reduce offload latency, but it offered no solution for intra-kernel stalls [14]. As follows, hardware latency hiding has its limitations. Restructuring dependency patterns has been proven in GPU-oriented software to recover idle cycles [15]. However, the proposed solution lacked support for modulo scheduling of the complex algorithms.

Nevertheless, software pipelining has received more attention in recent work. One provided development for modern NVIDIA Tensor Core architectures utilising warp specialisation. It suggests greater benefits from fine-grained instruction scheduling [16]. Another paper establishes methods for software pipelining on a GPU, including a C-slowing theoretical basis [17]. While these works provide a baseline for structural rescheduling, a contradiction remains: the theoretical benefits of software pipelining cannot be fully utilised. This is due to a lack of specialised apparatus to overcome latency arising from high-dependency feedback loops in a massively parallel environment. Therefore, a substantial performance plateau is observed in the DoG stage of the SIFT algorithm, with hardware resources not fully utilised despite the availability of scheduling instruments and theories. Thus, resolving the mismatch between instruction-level scheduling and algorithmic recursion is critical for the progress of high-throughput computer vision. Hence, creating a necessity for the development of a C-slow retiming framework on the GPU.

3. The aim and objectives of the study

The purpose of our study is to improve the computational efficiency of the DoG stage of the SIFT algorithm. It is done by developing and evaluating a custom all-pass IIR filter implementation and applying the C-slow retiming. This makes it possible to minimise computational complexity and maximise hardware resource utilisation on modern GPU architectures without compromising feature detection accuracy.

To achieve the goal, the following tasks are set:

– to formalise the mathematical model of all-pass IIR filtering for the DoG stage to establish a theoretical foundation for its reduced computational complexity;

– to develop and experimentally evaluate a high-performance implementation of all-pass IIR filtering for the DoG stage, utilising C-slow retiming to optimise instruction-level parallelism on massively parallel architectures.

4. Materials and methods for high-throughput IIR-based DoG implementation on GPUs

4.1. The object, subject, and hypothesis of the study

The following study is based on the critical analysis of hardware bottlenecks and dependency constraints specified in the previous sections. The research parameters are defined to ensure a rigorous evaluation of the proposed framework.

The object of the research is the process of computing the DoG stage of the SIFT algorithm on massively parallel hardware architectures.

The subject of the research includes the mathematical models of all-pass IIR filtering. The scope of the subject also includes software scheduling mechanisms based on C-slow retiming for NVIDIA CUDA-enabled graphics processing units.

The research hypothesis posits that the structural rescheduling of recursive instructions through C-slow retiming, integrated with the low-complexity nature of all-pass IIR filters, will effectively mask instruction-level latency. It is expected that this approach will bypass the hardware stalls caused

by tight data dependencies in recursive algorithms. Thereby achieving higher warp occupancy and superior throughput compared to traditional FIR convolution kernels.

4.2. Algorithmic requirements of the SIFT-based DoG stage

The SIFT method is based on mapping a frame into a scale space [18]. Scale space is a space where Feature Points (FP) are present at different scales, while keeping their relative position. In the SIFT method, it is constructed from an image that's blurred with a Gaussian filter with different blurring parameter σ [19]. The two neighbouring images are then included in the difference calculation, and the process is later repeated with a rescaled version of each image.

During the initial stage of the SIFT method, the image frame is filtered using a Gaussian kernel $G(x, y, \sigma)$

$$L(x, y, \sigma) = G(x, y, \sigma) * X(x, y), \tag{1}$$

where $*$ denotes the convolution operator, σ is a blurring parameter, and $X(x, y)$ represents the pixel intensity at coordinates (x, y). This resulting image becoming blurred. The basic kernel of Gaussian filtering is two-dimensional

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}. \tag{2}$$

To eliminate the complex computation required for this kernel, another approach is used. It is based on the separability of the Gaussian kernel (2)

$$G(x, y, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{x^2}{2\sigma^2}} \cdot \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{y^2}{2\sigma^2}} = G(x, \sigma) \cdot G(y, \sigma). \tag{3}$$

The filtered image is derived in two steps. Firstly, the image is convolved along one of the axes

$$\begin{aligned} L_v(x, y, \sigma) &= G(y, \sigma) * X(x, y), \\ L(x, y, \sigma) &= G(x, \sigma) * L_v(x, y, \sigma). \end{aligned} \tag{4}$$

where $L_v(x, y, \sigma)$ represents the intermediate image after the vertical convolution with the kernel $G(y, \sigma)$, and the final image is the result of horizontal convolution with the kernel $G(x, \sigma)$.

This separation of the kernel (3) provides a significant improvement in computational time. As a two-dimensional kernel has size $N \cdot N$, a direct convolution has complexity $O\left(N^2\right)$ for each pixel. On the other hand, two one-dimensional convolutions have a combined complexity of $O\left(2N\right)$ for each pixel. This creates a substantial time savings.

The next step of the algorithm is calculating the difference of Gaussians itself

$$D(x, y, \sigma_i) = L(x, y, \sigma_i) - L(x, y, \sigma_{i+1}), \tag{5}$$

where σ_i and σ_{i+1} are blurring parameters of the neighbouring scales. In other words, σ_i determines the blurring parameter in the i-th scale.

The Gaussian blurring parameter σ differs for each index of the image in the octave. However, σ values for different octave images with the same indexes in the octave were equal. Thus, these parameters must be calculated only once and then accessed repeatedly for each pass. The equation represents actual values

$$\sigma = 1.6 \cdot 2^{k/3}, \tag{6}$$

where the index k represents the scale progression within the octave, constrained by the base kernel size limit $k \in [0, K_B]$, and K_B is the number of derived blurred images.

As seen in the (5), a pixel in the resulting difference image can be computed as the literal difference between two pixels of the incoming blurred images, making this a computationally inexpensive part of the DoG algorithm.

To minimise the computational complexity, the scale space is divided into octaves. In the adjacent octaves j and $j+1$ the parameters are related as $\sigma_{i,j+1}/\sigma_{i,j} = 2$. The Gaussian image in higher octave is derived by decimation in 2 times of the Gaussian image in lower octave, which shortens the convolution cores twofold. However, this simplification makes the next FP localisation function rather complex. The resulting Gaussian images $L(x, y, \sigma)$ constitute an image pyramid.

The DoG algorithm serves as a less computationally intensive alternative and approximation for the Laplacian of Gaussians function. Additionally, it provides the immunity to the image noise. That's a two-dimensional second derivative of the image, inherently finding edges. Another consideration is that FIR and IIR are interchangeable for Gaussian blurring. In other words, the recursive implementation is equivalent to the classical approach in that context [20–22]. However, the IIR filter utilisation practice has shown that the filtering needs the high-precision calculations providing the stable filtering. A comprehensive survey of such algorithms is available in the literature [23].

4.3. Experimental environment and implementation stages

The research hypothesis was validated within a standard high-performance computing environment, ensuring reproducibility of the discovered results. The primary hardware platform used is the NVIDIA RTX 3090 GPU (Ampere architecture). The secondary platforms used are both NVIDIA: RTX 4090 and RTX 5090. All software elements were compiled using the NVIDIA CUDA Compiler with unified optimization flags, thereby eliminating the possibility of compiler-specific factors. Time measurements are performed on a warmed-up device via precise CUDA-compatible libraries, eliminating external influences.

The experimental framework is organised into three separate implementation stages, ensuring systematic validation of the research hypothesis and result repeatability:

1. baseline benchmarking (FIR) involves the traditional FIR filter is used to ensure the proposed framework's gains are persistent;
2. recursive implementation (IIR) evaluates the proposed IIR all-pass filter solution to analyse the computational complexity of sequential data dependencies and the reduction in the processing latency;
3. C-slowing optimisation applies intra-thread rescheduling to the IIR all-pass filter to analyse latency-hiding techniques on modern GPUs.

Input data for the experiments consisted of grayscale versions of the image. To maintain a stable result without black borders, the pre-computed out-of-the-border values were introduced based on the actual near-border values. Besides the mentioned point, it also formed a warm-up distance for the IIR solutions, ensuring the predictability of the resulting measurements. To comprehensively analyse different algorithms, a set of four images containing the same visual information is used. However, they feature different resolutions: input0 (0.5 MPix), input1 (2.5 MPix), input2 (5.0 MPix), and input3 (20.0 MPix).

4.4. All-pass IIR filtering implementation method

The FIR Gaussian implementation serves as a baseline for this research, as it is the default choice of the algorithm at this stage of SIFT. The finite approach has a computational disadvantage even for one-dimensional separated Gaussian convolution. The number of pixels involved in the calculation of the resulting blurred pixel is derived by equation

$$K_{FIR} \approx 5\sigma, \tag{7}$$

where K_{FIR} denotes the one-dimensional kernel length.

Since full Gaussian convolution consists of two separate one-dimensional passes, the total multiplication amount for a single pixel could be estimated as follows

$$N_{FIR}(\sigma) \approx 2 \cdot K_{FIR} = 10\sigma, \tag{8}$$

where $N_{FIR}(\sigma)$ denotes an approximate number of multiplications required to calculate one resulting pixel in the blurred image. So, this number is equal to 16–32 in a single octave depending on σ.

It could be stated from (8) that the computational complexity in the implementation grows proportionally with the σ parameter value. With the GPU architecture used, this scaling increases both instruction and memory pressure, thereby highlighting inherent bottlenecks and resulting in FIR-based DoG solutions being memory-bound in any implementation.

On the contrary, the IIR-based DoG is independent on the σ value. Instead, the logic is incorporated into the calculation of the updated pixel and subsequent pixels, forming a recursive sequence of dependencies. To compensate for the one-sided effect of blur, the anti-causal pass is added to the IIR for each vertical or horizontal causal pass. Among many IIR filter algorithms the all-pass based filters are distinguished in high stability, minimised complexity, high precision [24]. For these features this kind of filters is selected for our purposes.

An implicit all-pass filter response compensates phase distortion in these; as a result, a zero-phase filter is achieved. In more detail, a single pixel's calculation is performed as follows

$$\begin{aligned} q[n] &= x[n] - c \cdot q[n-1] - b \cdot q[n-2], \\ y[n] &= (b \cdot q[n] + c \cdot q[n-1] + q[n-2] + x[n-1])/2, \end{aligned} \tag{9}$$

where $q[n]$ is an intermediate value, $y[n]$ is a resulting value of the blurred pixel, and b and c are coefficients. Note that the (3) and (4) are substituted to the equations

$$\begin{aligned} L_v(x, y, \sigma) &= G_v^*(y, \sigma) * G_v(y, \sigma) * X(x, y), \\ L(x, y, \sigma) &= G_h^*(x, \sigma) * G_h(x, \sigma) * L_v(x, y, \sigma), \end{aligned} \tag{10}$$

where $G_v^*(y, \sigma)$ and $G_v(y, \sigma)$ are kernels of IIR filters (9) traversing the image columns up and down, $G_h(x, \sigma)$ and $G_h^*(x, \sigma)$ are ones traversing the rows forward and backward respectively. So, it has to be considered that

$$G_v^*(y, \sigma) * G_v(y, \sigma) \approx G(y, \sigma), \tag{11}$$

when selecting the coefficients b and c in (9).

The coefficients b and c are the slope coefficient and cut-off frequency control coefficient, respectively. Their values are adjusted such that the resulting IIR-blurred image matches the FIR-generated blurred image as closely as possible. The coefficients b and c are selected using some optimisation process which provides the equality (11). Considering the relations in [24], the approximated coefficients can be derived as follows

$$\begin{aligned} t &= \tan(0.95\sigma^{-1.5}), \\ b &= (1 - t)/(1 + t), \\ c &= -\cos(0.98/\sigma) \cdot (1 + b). \end{aligned} \tag{12}$$

The coefficients are calculated once, so they add practically no computational overhead to the long-running DoG task. Thus, an estimation for the IIR per-pixel computational complexity could be expressed as

$$N_{IIR} = 2 \cdot 2 \cdot 2 + 1 = 9. \tag{13}$$

The final term of the (13) is derived from the need for the IIR result to be multiplied by a scaling factor to match the saturation of the Laplacian of Gaussian result approximated by DoG. Comparing (13) to (8) one can reason that IIR filter DoG implementation is preferrable when $\sigma > 0.9$. Note that the estimated computational complexity focuses on multiplication due to fused multiply-add hardware instructions, which remove the overhead of addition by merging it with multiplication in the CUDA architecture.

IIR filtering is independent on the kernel-size σ parameter. However, the dependency on the previous element is introduced, thereby limiting the GPU's natural latency hiding.

Because typical GPU algorithms implement image processing via a one-thread-per-pixel approach, one might infer that the same holds for the FIR implementation used in this paper. However, the FIR implementation relies on one-thread-per-row (one-thread-per-column), as the paper's main concern is resource efficiency, not the raw time for a single image. This led to decreased shared-memory halo overhead typical of tile-based filtering, enabling instruction-level parallelism, increasing performance, and reducing thread occupancy [25].

4.5. C-slow retiming as a thread coarsening strategy

As the IIR filter consists of a dependent sequence of operations, including arithmetic operations and register accesses, this logic could be represented as a Synchronous Dataflow (SDF) graph. SDF is a directed graph consisting of operations as nodes and values/variables as edges. Additionally, an edge could store a set of data, meaning that it contains a FIFO buffer storing values from the previous iterations. The FIFO depth is equal to the number of bars marking the edge. The properties mentioned make it an efficient instrument for mapping the loop algorithms with close dependencies. Thus, the principle of C-slowing is directly related to an SDF graph, which provides a visually clear representation and a logically correct form of the underlying transformations.

Fig. 1 illustrates an SDF graph for the basic one-sided IIR Gaussian pass which executes equations (9). It can be noted that every resulting pixel value depends on the input values of the same iteration and a previous iteration. The intermediate values of the previous one and the one before the previous iteration are part of the dependency, too. These values are passed between iterations; thus, they must be fast-access data and stored in registers. Note that this algorithm needs only 2 multiplications per a single result.

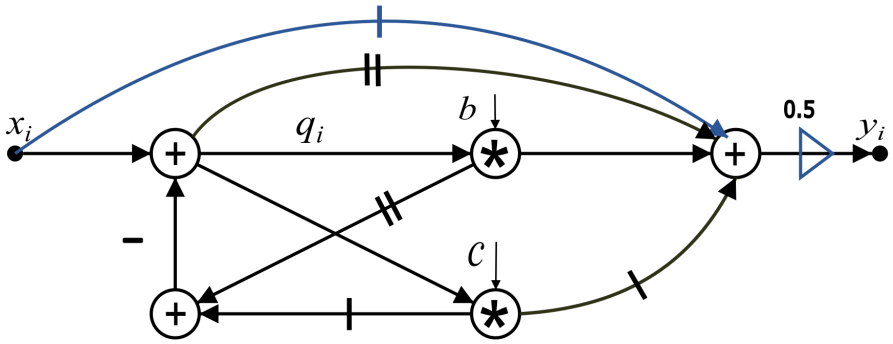

Fig. 1. SDF representation of the DoG pipeline

In this context, a principle of C-slow retiming consists of multiplying the number of all registers (bars) within the SDF graph by a factor of C. As classic C-slowing implementations state, this transformation forms a folded SDF that combines C instances of the original loops into a single loop without introducing any additional computational logic [26].

Looking at this process from a hardware perspective, C threads that are independent on each other share the same datapath. However, it is artificially elongated by the instruction count in a single thread being C times higher. In terms of software scheduling for GPUs, it is matched by explicit interleaving,

as independent DoG streams are running within a single execution kernel. These separate streams are neighbouring rows or columns in the current implementation.

By applying instruction-level modulo scheduling, the compiler generates a sequence of instructions in which operations are optimised to maximise arithmetic logic unit (ALU) occupancy. It is done by placing computational operations of Stream 2 immediately after access to memory is initiated in Stream 1. Thus, no time is stalled, after Stream 2 goes to Stream 3 and so on, until Stream C. This eliminates idle cycles in the ALU while waiting for memory or other long-running instructions by overlapping them with computation.

GPUs have different specifications, including the number of threads, the size of the warp, the clock frequency, the memory bandwidth, and other properties. For some units, C-slowing may provide more benefits than for others; it could be generalised as follows. Let t_0 denote the baseline per-pixel execution time of the IIR-based DoG kernel, where Δt denotes the overhead added by applying C-slowing. Resulting overhead is calculated as

$$\varepsilon = \frac{\Delta t}{t_0}. \tag{14}$$

According to the (14), the GPU specifications increase t_0 and decrease, while the other variable Δt is constant. Thus, it could be expected that lower-capability GPUs could gain more from interleaving and eliminating memory stalls, surpassing C-slowing overhead.

For C-slowing, a range of values was tested for $C \in [2, 8]$. They were used as a factor for the scale of row/column interleaving.

In terms of mapping the introduced algorithm to CUDA, the described C-slowing approach could be represented as thread coarsening, so that instead of dividing the load among threads, it is collected into a single thread. This provides several benefits:

1. decreased indexing overhead by processing multiple rows at once, no need for additional indexes inside a loop;
2. allocation amount reduction caused by C rows being processed in the same thread, as memory could be allocated only once, reducing the number of allocation operations. The additional effect is the cache miss minimisation due to the data localisation;
3. warp coalescing preservation, as even after row/column interleaving, the accessed data remains adjacent, leaving room for easier memory access management and caching.

5. Results of investigating the computational efficiency of the IIR-based DoG implementation on GPUs

5.1. Theoretical results of all-pass IIR complexity formalisation

The first stage of the investigation focused on the mathematical formalisation of the all-pass IIR filtering. As a result of the analysis, the computational complexity of the recursive approach was confirmed to be independent on the σ parameter of the Gaussian blur. This represents a significant result, establishing a theoretical limit of processing time, in contrast to the traditional FIR convolution kernel. The latter experiences complexity that scales linearly $O(N)$ with the kernel diameter. The theoretical proof of the method's viability enables the transition to the practical implementation phase to ensure support of the hypothesis by the experimental outcome.

5.2. Experimental performance evaluation of IIR and C-slow implementation

As DoG is an image-processing algorithm, its results should be visually confirmed to ensure correctness. In this case, FIR and IIR should produce visually consistent resulting images with the chosen parameters. Fig. 2 shows the original image and the corresponding results from both the IIR and FIR pipelines. It can be noted that both output images share similarities in general structure and in their sensitivity to the elements.

However, as SIFT algorithms serve as object detectors, DoG should highlight the inherent feature points of the image. As an approximation of the Laplacian of Gaussian, DoG should be capable of finding edges of the object in two-dimensional space, similar to the derivative it approximates. As shown in Fig. 3, theoretical expectations are matched with the practical measurements.

a *b* *c*

Fig. 2. Qualitative comparison of full DoG outputs: *a* – Input image; *b* – DoG (FIR); *c* – DoG (IIR)

a *b* *c*

Fig. 3. High-resolution fragment comparison: *a* – Input fragment; *b* – FIR detail extraction; *c* – IIR detail extraction

Whereas FIR is used as a baseline for the IIR comparison, some of its configurations could also be changed. The final kernel size is calculated as the product of the base kernel size and a value. Since the latter is derived from the image index within an octave, the former could be treated as a *base blurring parameter*.

Fig. 4 shows the dependencies of the measured execution time on the base kernel size for the FIR algorithm. It was measured for four different image sizes. Reported values are median runtimes (P50, ms) per input image. The charts clearly show that computational time scales with the base kernel size, resulting in a near-linear trend across all four images. This proves our theoretical estimations.

Whereas FIR computation time varies with the base kernel size, IIR computation time varies with the C-slowing factor, with the default implementation corresponding to a factor of one.

As FIR and IIR DoG outputs were confirmed to be visually equivalent in the context of edge localisation, another comparison should be conducted: a quantitative one by comparing measurement results.

Fig. 5 compares FIR and IIR computation times, enabling a strict comparison of the methods' computational effectiveness. The version of the FIR algorithm used in this comparison has a base kernel radius of three. IIR blur charts represent different C-slowing factor values. All four images are considered. Measured times represent median values. As can be seen, the default IIR implementation has the lowest computation time, whereas the C-factor application with different values does not reduce the measured time. For most of the images, C-slowing demonstrates similar times to the bare IIR filter. However, that's not applied to the largest image, where the C-slowing factor's growth significantly increases the measured time.

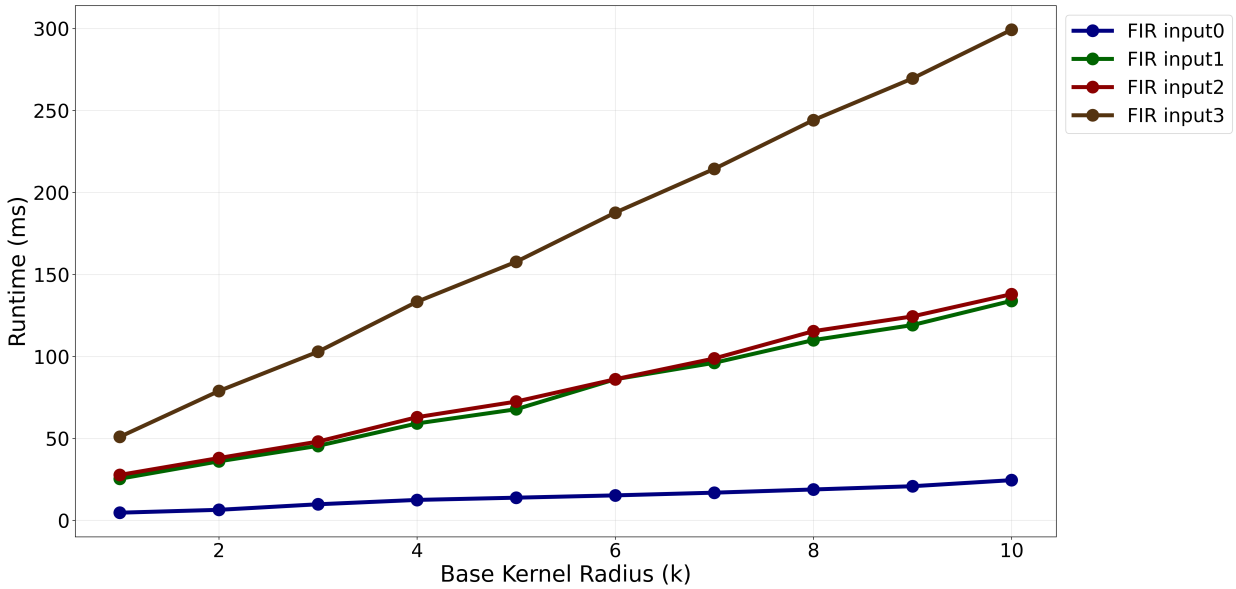

Fig. 4. FIR Gaussian blur runtime versus base kernel radius K

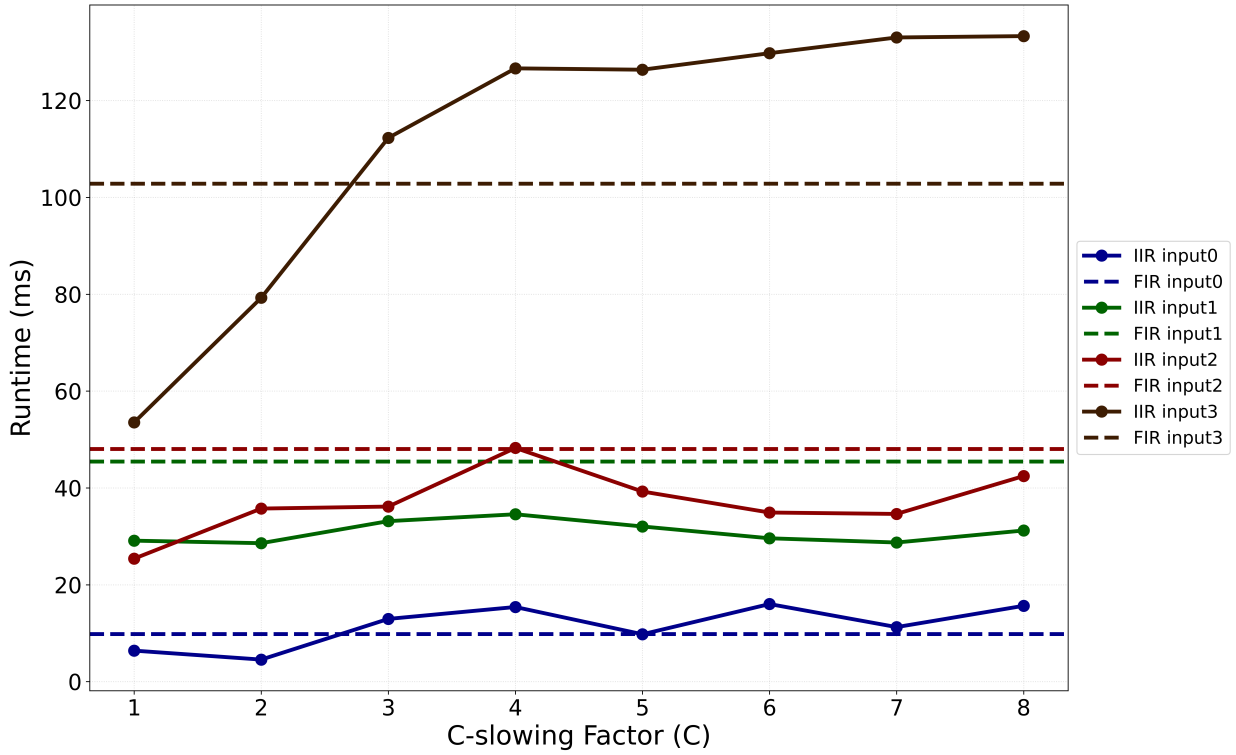

Fig. 5. Median runtime comparison: FIR vs IIR

Comparing the FIR and IIR filtering for the smallest image, the results differ for virtually every point on the chart. Then it performs worse for practically all the C values on medium-sized images (input1, input2). However, FIR improves, starting with $C = 4$ for the largest image (input3). Despite that, IIR still has a lower runtime for every image processing, confirming that basic IIR is computationally lighter than FIR in the DoG context.

Even though previous experiments confirmed basic IIR as a dominant strategy for the DoG pipeline, a caveat still undermines the derived results. GPU load was uneven across different image sizes or algorithms, as some implementations may have left resources idle. To eliminate the possibility that

the mentioned factors would influence the research outcome, another test was performed. As our previous setups had some blurs in an octave set $K = 5$, it is an optimal number for detecting edges. At the same time, it is suboptimal to load the GPU to its maximum capacity. This experiment uses $K = 25$, ensuring a higher level of GPU load. With this parameter change, a stable load of 90% or higher on the threads is achieved. In practice, it means that out of 100 threads, at least 90 were either performing actual computation or waiting for register memory to free up. This setup is a purely experimental workaround, as five blurs are considered enough. However, it could still be interpreted as a set of five images being processed simultaneously. Another change in this configuration is that, instead of computation time, computed throughput is measured (in MPix/s).

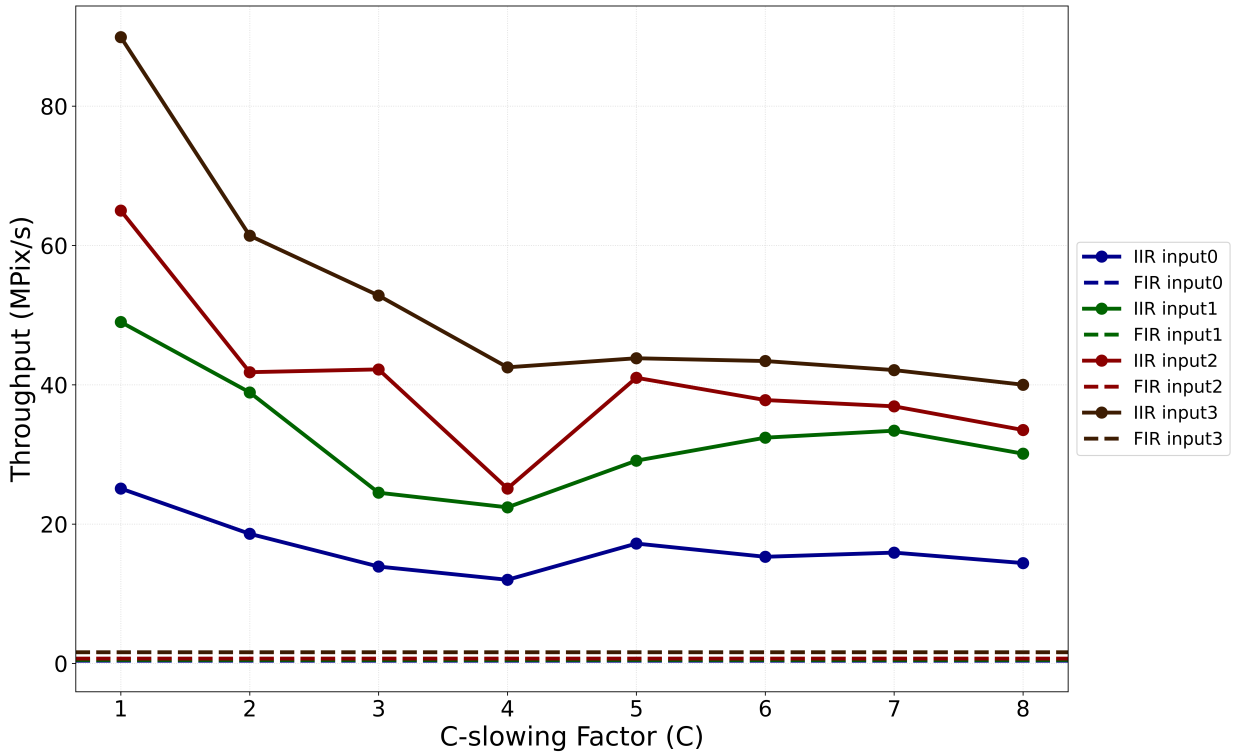

Fig. 6. Computed throughput comparison: FIR vs IIR

Fig. 6 shows results that are practically similar for the IIR implementation. The main difference is that FIR has low throughput due to its dependence on σ, which scales with each subsequent blur. However, the basic IIR outperforms all other algorithms, as in the previous experiments. Therefore, an IIR all-pass filter's superiority in the DoG context is experimentally confirmed in the different conditions.

6. Discussion of results concerning recursive DoG implementation and the effectiveness of C-slow optimisation on GPUs

The performance gap between the FIR and IIR implementations is primarily driven by the transition from linear $O(N)$ complexity to a constant $O(1)$ per-pixel load. Thereby, decoupling processing latency from the Gaussian kernel radius σ. The recursive nature of the IIR reinforces the received mathematical advantage, where significantly fewer coefficients are needed in contrast to the large spatial convolutions of the FIR. In the context of the hardware, this memory footprint minimisation substantially reduces register and memory bandwidth pressure. Therefore, a stable performance ceiling is achieved, limiting the amount of stalling due to data retrieval.

The paradox of C-slowing ineffectiveness on high-end GPU architectures stems from a conflict between manual instruction-level interleaving and the scheduling of modern hardware. The described

inherent logic allows for high intrinsic warp occupancy, leaving minimal idle cycles to recover. Whereas manual rescheduling adds instructions that result in overhead higher than the theoretical benefits of instruction interleaving. Therefore, on modern architectures that ensure high warp saturation, hardware scheduling remains a superior mechanism for latency hiding in the sequentially dependent algorithms.

In addition to the described throughput gains at constant pixel complexity, the IIR implementation is constrained by several factors. Firstly, persistent state variables increase register pressure, limiting warp occupancy. Secondly, strict sequential dependency chains impose limitations on the level of parallelisation by reducing the number of independent instructions. This feedback loop contradicts SIMT logic, which is fundamental to GPUs. Therefore, a limit on recursive filter optimisations is introduced as a trade-off between mathematical efficiency and hardware-level concurrency.

Despite the data presented, several points should be noted to clarify the details of the research. Firstly, this paper focuses only on the DoG stage; other stages of SIFT are not analysed here. Secondly, the measurements were executed for the images of defined sizes; different image resolutions may lead to the result deviations. However, the general trend does not indicate a significant effect. Finally, the GPUs used were based on the related CUDA architecture and programming framework. As a result, differences in architecture or applied framework could bring microarchitectural or compiler differences into play, leading to results less aligned with those described in this work.

Future work will focus on automated scale definition for interleaving, integrated hybrid-based DoG pipelining, and further C-slowing analysis on resource-constrained GPUs. There is an idea to investigate C-slowed IIR filtering on the Raspberry Pi GPU, which is distinguished by moderate hardware capabilities.

Conclusion

Based on the research conducted, the following conclusions are drawn.

The mathematical model of all-pass IIR filtering for the DoG stage has been formalised. It has established an analytical proof that the computational complexity is reduced from linear $O(N)$ to constant $O(1)$ with respect to the blur parameter (radius) σ. The scientific novelty of this result lies in the verification of a two-way cascaded structure that ensures zero-phase shift and numerical stability, providing a theoretical framework for scale-space construction without compromising feature localisation accuracy.

The software implementation and experimental evaluation on NVIDIA CUDA architectures confirm that the IIR-based DoG stage consistently outperforms traditional FIR convolution across all investigated resolution tiers (from 0.5 Mpix to 20.0 MPix). It is quantitatively identified that applying C-slow retiming is counterproductive for high-end GPUs, as the instruction overhead outweighs the benefits of manual latency hiding. The practical value of the study lies in identifying bare recursive all-pass IIR filters as the optimal lightweight alternative for high-resolution real-time computer vision pipelines.

The future investigations are needed to prove the C-slow retiming effectiveness at the assembly language level providing the genuine software pipelining. Also, other feature extraction algorithms like speeded up robust features (SURF) can be improved by substituting the FIR filters by IIR ones.

Acknowledgements

Declaration on the use of Artificial Intelligence. The authors used generative AI tools (Grammarly Pro 1.162.0.0, Gemini Pro 3) during the preparation of this manuscript for language editing and document formatting. All AI-generated content was carefully reviewed and edited by the authors. The authors take full responsibility for the accuracy, integrity, and originality of the article.

Funding. This research was supported in part by the National Research Foundation of Ukraine (NRFU), grant No 2025.06/0100.

References

[1] S. Williams, A. Waterman, and D. Patterson, "Roofline: An insightful visual performance model for multicore architectures," *Communications of the ACM*, 2009. https://doi.org/10.1145/1498765.1498785.

[2] C. Tirelli, J. Sapriza, R. R. Álvarez *et al.*, "SAT-based exact modulo scheduling mapping for resource-constrained CGRAs," *ACM Journal on Emerging Technologies in Computing Systems*, 2024. https://doi.org/10.1145/3663675.

[3] C. E. Leiserson and J. B. Saxe, "Retiming synchronous circuitry," *Algorithmica*, 1991. https://doi.org/10.1007/BF01759032.

[4] N. Weaver, Y. Markovsky, Y. Patel, and J. Wawrzynek, "Post-placement c-slow retiming for the Xilinx Virtex FPGA," in *Proc. FPGA*, 2003. [Online]. Available: https://doi.org/10.1145/611817.611845

[5] M. A. Akram, A. Khan, and M. M. Sarfaraz, "C-slow technique vs. multiprocessor in designing low area customized instruction set processor for embedded applications," *International Journal of Computer Applications*, 2011. https://doi.org/10.48550/arXiv.1204.1179.

[6] T. Strauch, "Running identical threads in c-slow retiming based designs," *arXiv preprint arXiv:1502.01237*, 2015. https://doi.org/10.48550/arXiv.1502.01237.

[7] J.-B. Carluer, L. Chauvin, J. Luo, W. M. Wells, I. Machado, R. Harmouche, and M. Toews, "GPU optimization of the 3d scale-invariant feature transform algorithm and a novel BRIEF-inspired 3d fast descriptor," *arXiv preprint arXiv:2112.10091*, 2021. https://doi.org/10.48550/arXiv.2112.10258.

[8] S. N. Sinha, J.-M. Frahm, M. Pollefeys, and Y. Genc, "Feature tracking and matching in video using programmable graphics hardware," *Machine Vision and Applications*, vol. 22, pp. 207–217, 2011. https://doi.org/10.1007/s00138-007-0105-z.

[9] M. S. Mohammadi and M. Rezaeian, "Towards affordable computing: SiftCU a simple but elegant GPU-based implementation of SIFT," *International Journal of Computer Applications*, 2014. https://doi.org/10.5120/15587-4329.

[10] D. Nehab and H. Hoppe, "GPU-efficient recursive filtering and summed-area tables," *ACM Transactions on Graphics (TOG)*, 2011. https://doi.org/10.1145/2070781.2024210.

[11] C. Griwodz, L. Calvet, and P. Halvorsen, "PopSift: A faithful SIFT implementation for real-time applications," in *Proceedings of the 9th ACM Multimedia Systems Conference*, 2018. [Online]. Available: https://doi.org/10.1145/3204949.3208136

[12] Z. Li, H. Jia, Y. Zhang, S. Liu, S. Li, X. Wang, and H. Zhang, "Efficient parallel optimizations of a high-performance SIFT on GPUs," *Journal of Parallel and Distributed Computing*, 2019. https://doi.org/10.1016/j.jpdc.2018.10.012.

[13] H. Feng, E. Q. Li, Y. Chen, and Y. Zhang, "Parallelization and characterization of SIFT on multi-core systems," in *2008 IEEE International Symposium on Workload Characterization (IISWC)*, 2008. [Online]. Available: https://doi.org/10.1109/IISWC.2008.4636087

[14] D. Lustig and M. Martonosi, "Reducing GPU offload latency via fine-grained CPU-GPU synchronization," in *2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*, 2013. [Online]. Available: https://doi.org/10.1109/HPCA.2013.6522336

[15] B. Liu, W. Qiu, L. Jiang, and Z. Gong, "Software pipelining for graphic processing unit acceleration: Partition, scheduling and granularity," *International Journal of High Performance Computing Applications*, 2016. https://doi.org/10.1177/1094342015585845.

[16] R. Soi, R. Yadav, f. Kjolstad, A. Aiken, M. M. Dehnavi, M. Garland, and M. Bauer, "Optimal software pipelining and warp specialization for tensor core GPUs," *arXiv preprint*, 2025. https://doi.org/10.48550/arXiv.2512.18134.

[17] A. Vinokurov and A. Sergiyenko, "Method for software pipelining on graphical processing units," *Journal of Applied Computer Systems*, 2025. https://doi.org/10.20535/2786-8729.6.2025.331193.

[18] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision*, 2004. https://doi.org/10.1023/B:VISI.0000029664.99615.94.

[19] T. Lindeberg, "Scale-space theory: A basic tool for analysing structures at different scales," *Journal of Applied Statistics*, vol. 21, no. 2, pp. 225–270, 1994. https://doi.org/10.1080/757582976.

[20] R. Saputra, S. Wahyuni, and M. I. Zulfa, "Comparison of LoG (Laplacian of Gaussians) and DoG (Difference of Gaussians) algorithms in the measurement of the nerve quality," *Journal of Applied Informatics and Engineering Applications*, 2025. https://doi.org/10.59934/jaiea.v5i1.1670.

[21] P. D. Luca and S. Cuomo, "Recursive filter based GPU algorithms in a data assimilation scenario," *Journal of Computational Science*, 2021. https://doi.org/10.1016/j.jocs.2021.101339.

[22] L. Zhang, H. Wang, and J. Liu, "Blurred lesion image segmentation via an adaptive scale thresholding network," *Applied Sciences*, 2025. https://doi.org/10.3390/app15179259.

[23] P. Getreuer, "A survey of gaussian convolution algorithms," *Image Processing On Line*, 2013. https://doi.org/10.5201/ipol.2013.87.

[24] P. A. Regalia, S. K. Mitra, and P. P. Vaidyanathan, "The digital all-pass filter: a versatile signal processing building block," *Proceedings of the IEEE*, vol. 76, no. 1, pp. 19–37, 1988. https://doi.org/10.1109/5.3286.

[25] Z. Lin, M. Mantor, and H. Zhou, "GPU performance vs. thread-level parallelism: Scalability analysis and a novel way to improve TLP," *ACM Transactions on Architecture and Code Optimization (TACO)*, 2018. https://doi.org/10.1145/3177964.

[26] N. V. Shenoy, "Retiming: Theory and practice," *Integration, the VLSI Journal*, 1997. https://doi.org/10.1016/S0167-9260(97)00002-3.

УДК 004.8:004.94

РЕАЛІЗАЦІЯ РІЗНИЦІ ГАУССІАНІВ НА ОСНОВІ ФАЗОВОГО НІХ-ФІЛЬТРА З С-СПОВІЛЬНЕННЯМ

Артемій Вінокуров
https://orcid.org/0009-0006-7861-5138

Анатолій Сергієнко
https://orcid.org/0000-0001-5965-1789

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського», Київ, Україна

Отримано: 31.03.2026р. / Прийнято: 11.05.2026р. / Опубліковано: 28.05.2026р.

У цій науковій роботі проведено детальний аналіз обчислювальної ефективності алгоритмів обробки цифрових зображень на графічних процесорах. Центральним об'єктом дослідження обрано етап побудови різниці гауссіанів (*DoG, Difference of Gaussians*) у межах алгоритму масштабної інваріантної трансформації ознак. Дослідження спрямоване на подолання критичних обмежень часу обчислення, притаманних традиційним реалізаціям фільтрів із скінченною імпульсною характеристикою (СІХ). Для вирішення цієї проблеми обґрунтовано та запропоновано застосування фільтрів із нескінченною імпульсною характеристикою (НІХ) на основі всепропускної фільтрації в поєднанні з методом *C*-сповільнення. Розроблена експериментальна база була розгорнута та протестована на сучасних обчислювальних архітектурах NVIDIA CUDA, представлених графічними процесорами RTX 3090, RTX 4090 та RTX 5090. Метод дослідження ґрунтується на порівняльному аналізі продуктивності стандартних СІХ-фільтрів із розробленими модифікаціями НІХ-фільтрів за різних значень фактора *C*-сповільнення. Експериментальні дані повністю підтверджують результати теоретичного моделювання, згідно з якими час обчислення за підходом використання СІХ-фільтрів масштабується лінійно залежно від розміру ядра згортки, тоді як час виконання алгоритму НІХ-фільтраії не залежить від цього параметра. Таким чином, використання НІХ-фільтрів забезпечує менші часові витрати та вищу пропускну здатність системи, зберігаючи при цьому візуальну якість обробленого зображення. Подальші експерименти показали, що метод *C*-сповільнення створює додаткові обчислювальні витрати, які перевищують його переваги, що суперечить початковим теоретичним очікуванням. Проте використання *C*-сповільнення на пристроях із низькою продуктивністю може забезпечити більш позитивні результати завдяки приховуванню затримок. Наукова новизна роботи полягає у використанні фазових НІХ-фільтрів як альтернативи традиційним підходам для реалізації етапу *DoG* на сучасних архітектурах графічних процесорів, ефективність яких підтверджена експериментально Практичне значення результатів полягає в можливості значного прискорення систем комп'ютерного зору в реальному часі шляхом оптимізації операцій масштабно-просторової фільтрації.

Ключові слова: синхронний потік даних, програмна конвеєризація, С-сповільнення, графічний процесор, інваріантне до масштабу перетворення ознак, різниця гауссіанів.