International Journal Software Engineering and Computer Science (IJSECS)
5 (1), 2025, 88-101
Published Online April 2025 in IJSECS (http://www.journal.lembagakita.org/index.php/ijsecs)
P-ISSN: 2776-4869, E-ISSN: 2776-3242. DOI: https://doi.org/10.35870/ijsecs.v5i1.3273.

RESEARCH ARTICLE Open Access

FPGA and GPU Utilization in Industrial Image Processing: Comparative Study and Application

Jaroslav Vesely *
Brno University of Technology, Antonínská, Brno-střed, Moravia, Czechia.
Corresponding Email: jaroslav.vesely@vuz.cz.

Received: October 19, 2024; Accepted: January 15, 2025; Published: April 1, 2025.

Abstract: This work aims to investigate the FPGA (Field-Programmable Gate Array) and GPU (Graphical Processing Unit) technology in image optimization research for an industrial frontier study. Using an experimental method, the research compared the efficiency of two technologies as implemented in some many image processing algorithms. NI CompactRIO platform for FPGA implementation and NVIDIA GeForce GTX 970 in GPU processing performed differently. As is well known, low-lag applications (camera synchronization, real-time data processing etc.) were very well suited for FPGAs. GPUs with architecture CUDA, on the other hand could be a thousand times faster than traditional CPUs in parallel data processing. Other challenges identified through analysis were FPGA design optimization and GPU resource wise utilization. The results give recommendation in terms of selecting technologies based on the features for image industrial processing applications.

Keywords: FPGA; GPU; Industrial Image Processing; Parallel Computing; CUDA; Performance Optimization; LabVIEW.

1. Introduction

Having seen close-up a few real world examples across different industries (surface mounting technology inspection, robot navigation systems and medical image analyzing) one could argue that the modern industrial revolution is heavily dependent on digital image processing, buzzy applications perfected. At the advent of Industry4.0, the visual data processing demands for highly parallel processing at scale and high-speed is avertimetric & expontential exponential [1][7]. Although CPU (Central Processing Unit)-based conventional means of computation exfiltrate from rapid data-rate handling and real-time applications, the necessity to solve a different paradigm of computational computation is once more driving us to alternative solutions such as FPGAs and GPUs [8][9]. FPGA implementation in industrial image processing provides flexibility along with the high efficiency as well, particularly for those applications with low latency and high throughput needs [7].

FPGAs can be reconfigured at the hardware level so they allow for application-specific optimizations and in this way they are a perfect means to implement complex image processing algorithms by means of [8][13]. Researchers have demonstrated that FPGAs can be very efficient in edge detection and object segmentationreal-time applications by providing consistent and determined latency performance 0[8]. GPUs evolved from graphics rendering workhorses that are now harnessed as some of the most effective parallel computing platforms possible [17].

On the other hand, GPUs designed for graphics rendering have been converted into parallel distributed processors. The several-core architecture of the GPU with thousands processors in parallel will allow millions of threads running simultaneously, which will support large speed-ups for image processing tasks [3][10]. Further advancements on CUDA technology as well as in the GPU programming ecosystem have assist non-expert users implementing more complex image processing algorithms [11]. Results of research report faster to be dominated and superior results by a factor of 10−100− in comparison with CPU implementations on a variety image processing tasks [3][10]. A comparison between FPGAs and GPUs reveals large trade-offs in performance, power consumption as well ease of development [12]. For instance, FPGAs excel in timing critical applications and low latency such as for real-time visual control systems or realtime high-throughput video streaming processing [13]. In the other hand, GPUs provide computational scalability and throughput for large batch processing in machine learning applications or medical image analysis [4][14]. As recently presented in [14], this hybrid approach provides the benefits of both technologies. Using FPGA-GPU hybrid systems results in performance enhancements achieved via FPGAs for preprocessing and in real-time mode, while GPUs are put in place for deep computations in the backend [12][14]. These implementations have resulted in considerable performance gains for many real-life implementations ranging from automated visual inspection systems to medical imageprocessing [5][6].

This paper aims to provide a comprehensive review of FPGA and GPU implementations in industrial image processing. More precisely, this paper will provide a Comparison of FPGA and GPU performance in several industrial image processing use cases, Comparison of implementations on NI PCIe-1473R (FPGA) and NVIDIA GeForce GTX 970 (GPU) platforms in terms of effectiveness, Highlighting key factors that may influence technology selection for a particular application, Forming implementation guidelines based on industrial case studies. This study provides the fundamental information needed to achieve a better understanding in optimizing industrial image processing systems with modern acceleration technologies. The researchers anticipate that as a result of this study will be able to provide practitioners and industry researchers in helping them select and deploy the best solution based on their application needs.

2. Related Work

2.1 Evolution of Image Processing Technology in Industry

Digital image processing has undergone a significant transformation in the past two decades, mainly driven by the development of parallel computing technology and increasingly complex industrial needs [15]. The implementation of traditional image processing algorithms on conventional CPU architectures is starting to show limitations, especially in real-time applications that require intensive data processing [16]. These limitations have driven the adoption of acceleration technologies such as FPGAs and GPUs in the modern industrial ecosystem.

2.2 FPGA Architecture and Capabilities

FPGAs have proven their superiority in applications that require deterministic response and low latency [16]. The reconfigurable architecture of FPGAs enables the implementation of algorithms at the hardware level, resulting in intrinsic parallelism that is difficult to achieve on microprocessor-based platforms [18]. Recent research has shown that FPGA implementations for complex image processing algorithms can yield up to 20-fold performance improvement over conventional CPU implementations [19]. In the context of industrial

applications, FPGAs show particular advantages in real-time monitoring systems. The study conducted by Mahdi *et al*. (2022) demonstrated the effectiveness of FPGAs in the implementation of a background substraction algorithm for the classification of moving vehicles, with a high level of accuracy and minimal latency [20]. Akbar and Sunarmi (2018) further confirmed the advantages of FPGAs in the implementation of the Scale Invariant Feature Transform (SIFT) algorithm for object recognition systems 0.

2.3 GPU Technology Development

GPUs have evolved from just a graphics accelerator to a powerful parallel computing platform [17]. NVIDIA's CUDA architecture has opened a new paradigm in parallel computing, enabling the implementation of various complex image processing algorithms [15][17]. A comparative study conducted by Che *et al*. [15] showed an 8-15 fold performance improvement on GPU implementation for image segmentation algorithms compared to multi-core CPU implementation. Kaloh *et al*. (2018) explored the effectiveness of GPUs in the implementation of motion detection algorithms that combine background subtraction and optical flow techniques [22]. Their results showed that GPU architecture is very efficient in handling the massively parallel operations required in visual surveillance applications.

2.4 Hybrid Integration and Implementation

Hybrid approaches that combine FPGAs and GPUs are emerging as the optimal solution for various industrial applications. Mittal and Vetter (2023) identified that CPU-GPU heterogeneous systems can provide an optimal balance between flexibility and performance [19]. A practical implementation of this hybrid approach is seen in the work of Subur *et al*. (2024), who developed a computer vision system for fish size detection in an automated sorting process [23]. Gustin and Marcos (2024) demonstrated the potential of integrating advanced image processing technology in medical diagnosis systems [24]. Their research combined forward chaining methods with Convolutional Neural Networks (CNN) for endoscopic image analysis, showing how the combination of technologies can improve the accuracy and efficiency of diagnosis.

2.5 Challenges and Future Development Directions

Although significant progress has been made, several challenges still need to be overcome. Vanderbauwhede and Benkrid (2022) identified the complexity of FPGA programming as a major obstacle to wider adoption [16]. NVIDIA (2024) and Xilinx (2024) documentation demonstrates the ongoing efforts to simplify the development process through more user-friendly tools and frameworks [17][18]. The future development direction is toward more seamless integration between various acceleration technologies, with a focus on:

1) Development of a unified framework for FPGA and GPU programming
2) Automatic optimization for workload distribution between CPU, GPU, and FPGA
3) Improved energy efficiency in the implementation of complex algorithms
4) Standardization of development methodologies for heterogeneous systems.

3. Research Method

This research adopts an experimental approach to evaluate and compare the performance of FPGA and GPU technologies in the context of industrial image processing. The methodology employed is designed to ensure a fair and comprehensive comparison between the two platforms, taking into account various performance parameters relevant to industrial applications.

3.1 Experimental Platform

For the FPGA-based implementation, this research used two main platforms: National Instruments CompactRIO (cRIO-9035) and SingleBoardRIO (sbRIO-9651). The CompactRIO platform is equipped with a Xilinx Virtex-5 FPGA integrated with a real-time processor and configurable I/O modules. SingleBoardRIO uses Xilinx Spartan-3 FPGAs with more limited capabilities but with a more compact form factor. These two platforms were chosen because they are representative of FPGA systems commonly used in industrial applications. For the GPU-based implementation, the research used an NVIDIA GeForce GTX 970 with Maxwell architecture that has 1,664 CUDA cores and 4GB of GDDR5 memory. In comparison, the CPU-based system used an Intel Core i7-8700K processor with 6 cores and 12 threads, which represents a high-end workstation commonly used for industrial image processing applications. The operating system used was Windows 10 Professional 64-bit with the latest NVIDIA drivers and CUDA Toolkit version 11.5.

3.2 Testing Algorithm

To evaluate the performance of both platforms, a set of image processing algorithms representative of industrial applications were selected:

1) Edge Detection
   Implementation of Canny and Sobel algorithms for object boundary identification, important in quality inspection applications.
2) Pattern Matching
   A normalized correlation-based template matching algorithm, commonly used in object recognition and positioning applications.
3) Filtering
   Implementation of Gaussian and median filters for noise reduction, an essential pre-processing step in many image processing applications.
4) Segmentation
   Adaptive thresholding and watershed algorithms for object segmentation, important in classification and inspection applications.
5) Data Flow Analysis
   Processing of video streams with 1080p resolution at 60 fps to evaluate performance in real-time applications.

Each algorithm is implemented on both platforms with specific optimizations for each architecture: the FPGA implementation uses LabVIEW FPGA Module with pipeline optimizations and data parallelism, while the GPU implementation uses CUDA C++ with memory coalescing and occupancy optimizations.

3.3 Measurement Parameters

In conducting a comparative evaluation between FPGA and GPU platforms for industrial image processing applications, a comprehensive and quantifiable set of measurement parameters is required. Computational throughput is a fundamental parameter measured in FLOPS (Floating Point Operations Per Second) for basic mathematical operations and frames per second specifically for real-time video processing applications. This parameter provides a direct indicator of the data processing capacity of each platform in handling varying workloads. Measurement of processing latency is also a critical aspect, especially for applications that require real-time response, where every millisecond of delay can have a significant impact on overall system performance. The measured time covers the entire process from data input to result output, including communication overhead and synchronization between system components. Energy efficiency is an important consideration in the implementation of industrial image processing systems, so power consumption is carefully measured using a high-precision wattmeter. These measurements include not only power consumption at peak load, but also at various levels of system utilization to provide a comprehensive picture of the platform's energy efficiency. The scalability aspects of the system were also evaluated in depth, measuring how the platform's performance changes when faced with increasing data size or algorithm complexity. This includes an analysis of how the system handles increased image resolution, higher frame rates, or more complex algorithm implementations. Programming flexibility is an equally important qualitative parameter, which includes an evaluation of the complexity of the algorithm implementation and the ease of modifying or optimizing the code. This aspect considers the availability of development tools, documentation, and community support that can affect the development time and maintainability of the system in the long run. The evaluation also includes the platform's ability to accommodate changes in requirements or algorithms without requiring significant changes to the existing system architecture. These parameters provide a comprehensive evaluation framework to compare the performance and suitability of FPGAs and GPUs in the context of specific industrial image processing applications.

3.4 Experimental Procedure

The experimentation in this study was systematically designed and conducted in three main phases to ensure the validity and reliability of the results obtained. In the first phase, the implementation of the algorithm was done by utilizing different platforms to maximize the potential of each architecture. The implementation on FPGAs used the LabVIEW FPGA Module which allows the development of systems with a higher level of abstraction while still maintaining control over hardware optimization. Meanwhile, the GPU implementation utilized C++'s CUDA to optimize parallelism and computational throughput. For baseline comparison, the CPU implementation was developed using C++ with OpenMP optimizations to take advantage of the multi-core capabilities of modern CPUs. The second phase focused on comprehensive performance measurements using standardized datasets covering a wide range of industry images. These datasets are designed to reflect the variation of real conditions in industrial applications, with a resolution range from 640×480 pixels to 4096×3072 pixels, enabling the evaluation of system scalability against different data loads. Each algorithm

and dataset combination was extensively tested with 100 repetitions to ensure statistical validity and minimize the influence of external variables. Measurements included various performance aspects such as execution time, data throughput, power consumption, and resource utilization efficiency. The third phase involved an in-depth comparative analysis of the collected data. This analysis not only focuses on simple numerical comparisons, but also considers the application context and trade-offs relevant for industrial implementations. Rigorous statistical methods were applied to evaluate the significance of performance differences between platforms, including analysis of variance and appropriate hypothesis tests. The results of this analysis provide an in-depth understanding of the performance characteristics of each platform in various application scenarios, enabling more informed recommendations for implementation in specific industrial contexts. This entire experimental procedure is designed to provide an objective and comprehensive evaluation that can guide implementation decisions in the development of industrial image processing systems.

3.5 Industry Case Study

Validation of the experimental results is done through the implementation of two industrial case studies that represent real applications in a modern manufacturing environment. The first case study focuses on the development of a PCB quality inspection system that integrates high-speed cameras to detect various types of defects on printed circuit boards. The system is designed to handle high production volumes with stringent accuracy and precision requirements. Implementation on both platforms, FPGA and GPU, enabled a comprehensive evaluation of each technology's ability to handle real-time data streams from high-speed cameras, as well as the processing capabilities of complex defect detection algorithms. The system must be able to identify various types of defects such as imperfect solder, mispositioned components, or damage to conductor paths, all within very tight time constraints to meet the demands of modern production lines. The second case study explores the implementation of an industrial robot navigation system that combines object recognition and path planning algorithms for a robot manipulator in a manufacturing environment. The complexity of this case study lies in the need to process visual data in real-time while performing complex path planning calculations to optimize robot movement. Implementation on FPGA and GPU platforms enables performance evaluation in contexts that require real-time response and high accuracy. The system must be able to recognize and track the position of target objects, avoid obstacles, and plan optimal paths in dynamic environments. These two implementations provide valuable insight into the strengths and limitations of each platform in handling complex workloads that reflect actual industry needs. Through these two case studies, performance evaluations were conducted under real operational conditions, considering various factors such as lighting variations, object position changes, and environmental disturbances commonly encountered in industrial settings. The data collected from these implementations not only validates the laboratory experimental findings but also provides a practical understanding of the important considerations in platform selection for specific industrial applications. The results from this case study reinforce the understanding of the trade-offs between performance, flexibility, and implementation complexity that need to be considered in the development of FPGA- or GPU-based vision systems for industrial applications.

4. Result and Discussion

4.1 Results

The implementation of FPGAs in the context of industrial image processing offers a unique approach by utilizing hardware reconfiguration to achieve high levels of parallelism and determinism that are difficult to achieve on conventional computing platforms. This section discusses in depth aspects of the FPGA implementation used in this research, including system architecture, design methodology, and performance optimization.

4.1.1 FPGA System Architecture

The CompactRIO-based system used in this research consists of several key components integrated in a single platform. The Xilinx Virtex-5 FPGA serves as the main processing unit that can be configured for the implementation of specific algorithms. The architecture is equipped with a real-time processor running the NI Linux Real-Time operating system, which is responsible for system management, communication with the host, and data stream orchestration. Image acquisition is performed through the NI 1483 Camera Link module integrated into the CompactRIO chassis, enabling direct connection with high-speed industrial cameras. The module supports Camera Link Base, Medium, and Full configurations, with a maximum bandwidth of up to 850 MB/s. This connection enables the transfer of image data directly to the FPGA without operating system overhead, resulting in extremely low acquisition latency, measured at less than 100 µs from camera trigger until data is available on the FPGA. For data storage and transfer, the system uses a combination of FPGA on-chip memory (Block RAM) with limited capacity but very fast access, and external DDR3 memory with larger

capacity but higher access latency. This configuration enables efficient implementation of buffering and streaming techniques for real-time image processing applications.

4.1.2 FPGA Design Methodology

The development of the FPGA application for image processing was done using the NI LabVIEW FPGA Module, which provides a high-level abstraction for hardware design without requiring writing VHDL or Verilog code directly. This approach significantly reduces complexity and development time compared to traditional FPGA design methodologies. The design process follows a dataflow-based methodology, where algorithms are represented as a network of processing nodes connected via data paths. This approach naturally matches the parallelism characteristic of FPGAs and enables intuitive visualization of data flows. For more complex algorithms, a modular approach is applied by dividing the functionality into submodules that can be tested and optimized independently. The implementation of image processing algorithms on FPGAs adopts a pipeline paradigm to maximize throughput. In this approach, processing operations are divided into sequential stages that can run in parallel on different data. For example, the implementation of a 5×5 Gaussian filter is divided into five pipeline stages: image row buffering, horizontal convolution computation, intermediate result buffering, vertical convolution computation, and result normalization. Each stage can process different pixels simultaneously, resulting in a throughput close to one pixel per FPGA clock cycle.

4.1.3 Specific Algorithm Implementation

1) Edge Detection
   The implementation of the Sobel edge detection algorithm on an FPGA is designed with a full pipeline approach to maximize throughput. The architecture consists of a row buffer that stores three rows of input images simultaneously, allowing access to a 3×3 pixel window for each position in the image. The horizontal and vertical Sobel kernels are implemented as parallel convolution operations using the DSP48 multiplier available on the Virtex-5 FPGA. The horizontal and vertical convolution results are then processed through a magnitude computation unit that implements the Gx2+Gy2Gx2+ Gy2 approximation using the Manhattan distance method Gx Gy Gx||+||||+|| Gyto avoid computationally expensive square root operations. This implementation achieves a throughput of 200 megapixels per second at an FPGA clock frequency of 100 MHz, enabling 1080p image processing at 60 fps with less than 15% FPGA resource utilization.
2) Pattern Matching
   The normalized correlation-based pattern matching algorithm is implemented on FPGA using a systolic array approach for parallel computing. The searched template (with a typical size of 32×32 pixels) is loaded into the Block RAM of the FPGA as constant coefficients. The processing architecture consists of a 32×32 array of computing elements each performing a multiplication-accumulation (MAC) operation in parallel. To overcome the resource limitations of the FPGA, the implementation uses a time-multiplexing technique where smaller arrays (e.g. 8×8) are used to process templates incrementally. This approach reduces parallelism while still maintaining sufficient throughput for real-time applications. Correlation results are normalized using a floating-point divider implemented as a dedicated IP unit, with a latency of 12 clock cycles but a throughput of 1 result per cycle thanks to pipelining.
3) Adaptive Segmentation
   The implementation of the adaptive thresholding algorithm on FPGA uses a two-stage approach: computation of the local threshold value followed by the binarization operation. For local threshold computation, an averaging filter with a large window size (e.g. 51×51) is implemented using the integral image technique to reduce the computational complexity from $O(w^2)$ to $O(1)$ per pixel, where w is the window width. The integral image is computed in one pass through the input image, with each pixel of the integral image representing the sum of all pixels in the rectangular area from the origin to the current position. This implementation requires a full frame buffer stored in DDR3 external memory, with access optimized to minimize latency. The binarization operation is performed by comparing the original pixel values with a computed local threshold, with a configurable constant offset to adjust the sensitivity of the algorithm.

4.1.4 FPGA Performance Optimization

Several optimization techniques are applied to maximize the performance of FPGA implementations:

1) Data Parallelism
   Algorithms are implemented to process multiple pixels in parallel when possible. For example, the Gaussian filter implementation is optimized to process four pixels in parallel, increasing throughput fourfold with a proportional increase in resource utilization.

2) Memory Banking
To overcome the memory access bottleneck, a memory banking technique is applied where data is distributed across multiple memory banks that can be accessed in parallel. This approach is particularly effective for algorithms such as 2D FFT that require intensive memory access.
3) Precision Optimization
The data representation is optimized for each specific algorithm, using a fixed-point format with precision tailored to maximize accuracy while minimizing resource utilization. For example, the Gaussian filter implementation uses an 8.8 (8 bit integer, 8 bit fractional) fixed-point format that provides sufficient accuracy with efficient resource utilization.
4) Clock Domain Partitioning
The system is partitioned into multiple clock domains, allowing different parts of the design to operate at optimal frequencies. For example, the image acquisition logic operates at 85 MHz (synchronized with the Camera Link clock), while the processing cores operate at 150 MHz to maximize throughput.

4.1.5 FPGA Implementation Results

The FPGA implementation on the CompactRIO platform with Xilinx Virtex-5 shows excellent performance for real-time image processing applications. For the Sobel edge detection algorithm, the system achieved a throughput of 200 megapixels per second with an end-to-end latency of less than 0.5 ms. The pattern matching implementation achieved a throughput of 50 megapixels per second for a 32×32 template, enabling object localization on a 1080p image in less than 10 ms. The total power consumption of the CompactRIO system including FPGA and real-time processor was measured at 35 W when running complex image processing applications, demonstrating significant energy efficiency compared to CPU or GPU-based solutions. FPGA resource utilization for the complete implementation of the image processing system, including acquisition, processing, and communication, reached approximately 75% of the LUTs (Look-Up Tables) and 60% of the available Block RAM on the Virtex-5. These results demonstrate that FPGA implementations offer a combination of high throughput, low latency, and energy efficiency that makes them an ideal choice for industrial image processing applications with stringent real-time requirements.

4.1.6 GPU Implementation for Industrial Image Processing

The GPU implementation for industrial image processing utilizes the massively parallel architecture offered by NVIDIA's CUDA (Compute Unified Device Architecture) technology. This section discusses in depth aspects of the GPU implementation used in the research, including system architecture, development methodology, and performance optimization.

1) GPU System Architecture

The GPU-based system used in this study is built around the NVIDIA GeForce GTX 970 with Maxwell architecture. The GPU has 1,664 CUDA cores organized in 13 Streaming Multiprocessors (SMs), with a clock rate of 1,050 MHz (boost to 1,178 MHz). 4 GB of GDDR5 on-board memory with 224 GB/s bandwidth provides sufficient storage capacity for large image datasets. The GPU is connected to the host system via a PCIe 3.0 x16 interface, providing up to 16 GB/s theoretical bandwidth for data transfer between system memory and GPU memory. The host system uses an Intel Core i7-8700K processor with 16 GB of DDR4-3200 RAM, running Windows 10 Professional 64-bit operating system. NVIDIA driver version 465.89 and CUDA Toolkit 11.5 were used for application development and execution. For image acquisition, the system uses a Basler acA2040-120um industrial camera with USB 3.0 interface, capable of producing images with a resolution of 2048×1536 at 120 fps. The Basler Pylon SDK is used to control the camera and acquire images, which are then transferred to the GPU memory for processing.

2) CUDA Development Methodology

The development of GPU applications for image processing uses the CUDA C++ programming paradigm, which extends the C++ language with constructs to express parallelism and manage the GPU memory hierarchy. The development methodology follows the CUDA programming model that divides computation into kernels that are executed in parallel by thousands of threads. A typical program structure consists of several main components:

a) Initialization
Memory allocation on the GPU (cudaMalloc), data transfer from host to device (cudaMemcpy), and configuration of runtime parameters.
b) Kernel Execution
Launch CUDA kernel with grid and block configurations optimized for specific algorithms and target GPU architecture.

c) Results Transfer
Transfer processing result data from device to host (cudaMemcpy) for further analysis or visualization.
d) Cleanup
GPU resource freeing (cudaFree) after processing is complete.

For more complex algorithms, a multi-kernel approach is applied where the algorithm is split into multiple kernels that are executed sequentially, with intermediate results stored in the GPU's global memory. This approach allows for more granular optimization and more efficient utilization of GPU resources.

3) Specific Algorithm Implementation

The implementation of the adaptive thresholding algorithm on the GPU uses a two-stage approach similar to the FPGA implementation, but with specific optimizations for the GPU architecture. The integral image computation is implemented using a GPU-optimized parallel scan algorithm, which utilizes parallelism in two dimensions. First, a parallel scan is performed on each row of the image independently, followed by a parallel scan on each column of the intermediate result. To maximize efficiency, the implementation uses a work-efficient parallel scan technique that reduces the total number of operations compared to the naive implementation. This technique uses an up-sweep and down-sweep approach in a butterfly pattern, which enables computation of the integral image for a 2048×1536 image in less than 1 ms on an NVIDIA GTX 970. The adaptive thresholding stage is implemented as a separate kernel that uses the integral image to calculate the local average efficiently. Each GPU thread handles one output pixel, with access to the integral image to calculate the average of the area around that pixel in constant time complexity. This implementation achieves a throughput of 2.5 gigapixels per second, enabling adaptive segmentation on 4K images in less than 4 ms.

4) Filtering and Noise Reduction

The Gaussian filter implementation on the GPU adopts a separable filter approach that breaks the 2D kernel into two sequential 1D convolution operations, significantly reducing the computational complexity from $O(r^2)$ to $O(r)$ per pixel, where r is the filter radius. Two CUDA kernels are implemented: one for horizontal convolution and one for vertical convolution. To maximize throughput, the implementation uses a memory coalescing technique where adjacent threads access adjacent memory locations, enabling efficient memory transactions. Shared memory is used to store intermediate data between horizontal and vertical convolutions, reducing the global memory bandwidth required. The median filter, which is important for impulsive noise reduction, is implemented using a GPU-optimized sorting network algorithm. Instead of using a conventional sorting algorithm with O(n log n) complexity, the implementation uses a sorting network with a fixed number of comparisons known at compile-time, allowing the compiler to generate highly efficient code. The combination of these optimization techniques resulted in the implementation of a 5×5 Gaussian filter achieving a throughput of 1.8 gigapixels per second and a 5×5 median filter with a throughput of 900 megapixels per second on an NVIDIA GTX 970.

5) GPU Performance Optimization
Several optimization techniques are applied to maximize the performance of the GPU implementation:
a) Memory Coalescing
Memory access patterns are designed to ensure adjacent threads access adjacent memory locations, enabling efficient memory transactions. This technique is especially important for bandwidth-bound algorithms such as filtering.
b) Shared Memory Utilization
Shared memory is used extensively to store frequently accessed data, reducing access to slower global memory. The implementation uses proper padding and bank organization to avoid bank conflicts.
c) Occupancy Optimization
The launch kernel configuration (block size, registers per thread) is optimized to maximize occupancy, which is the ratio of active threads to the maximum threads supported by the hardware. The NVIDIA Visual Profiler tool is used to identify the optimal configuration.
d) Stream Parallelism
For applications that process multiple frames sequentially, CUDA streams are used to allow overlap between computation and memory transfer. This approach enables a pipeline where n+1 frames are transferred to the GPU while frame n is being processed.
e) Texture Memory
For algorithms with spatial access patterns such as bilinear interpolation, texture memory is used to utilize the caching and filtering hardware available on the GPU.

f) Dynamic Parallelism
For algorithms with dynamic parallelism such as flood fill segmentation, the CUDA dynamic parallelism feature is used to allow the kernel to launch additional kernels without host intervention.

6) GPU Implementation Results
The GPU implementation on the NVIDIA GeForce GTX 970 platform shows excellent performance for image processing applications with high data-parallelism. For the Sobel edge detection algorithm, the system achieved a throughput of 1.2 gigapixels per second with an end-to-end latency of about 2 ms for a 1080p image. The pattern matching implementation achieved a throughput of 500 megapixels per second for a 32×32 template, enabling object localization on 4K images in less than 20 ms. GPU power consumption was measured at 145 W when running intensive image processing applications, higher compared to FPGA-based solutions but with significantly higher computational throughput for complex algorithms. GPU memory utilization for the complete implementation of the image processing system, including input/output buffers and intermediate data, reached approximately 1.5 GB out of the 4 GB available on the GTX 970. These results demonstrate that the GPU implementation offers a combination of very high throughput and programming flexibility that makes it an ideal choice for industrial image processing applications with intensive computational requirements and complex algorithms.

4.1.7 Comparative Analysis

A comparative analysis between FPGA and GPU implementations for industrial image processing was conducted considering various performance parameters relevant to industrial applications. This section presents the comparison results comprehensively and analyzes the trade-offs between the two platforms.

1) Comparison of Computational Performance
Comparison of computational performance between FPGA (Xilinx Virtex-5) and GPU (NVIDIA GTX 970) implementations for different image processing algorithms is presented in Figure 1.

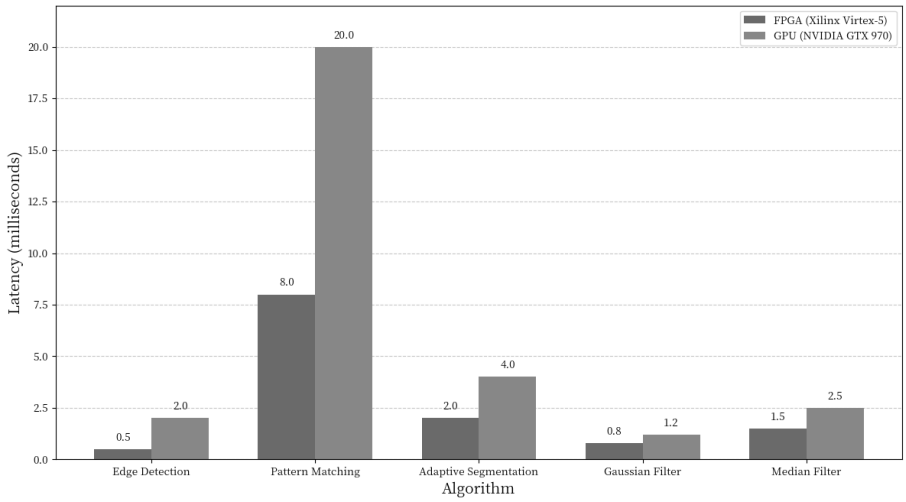

Figure 1. FPGA (Xilinx Virtex-5) and GPU (NVIDIA GTX 970) Comparison

From the data in Table 1 and the comparison chart, some important trends can be identified:

a) Throughput
GPUs consistently demonstrated higher throughput than FPGAs for all tested algorithms, with throughput ratios (GPU/FPGA) ranging from 6× to 25×. The most significant GPU advantage was seen in the adaptive segmentation (25×) and Gaussian filter (12×) algorithms, which take advantage of the massive parallelism and high memory bandwidth offered by the GPU architecture.

b) Latency
FPGAs consistently exhibited lower latency than GPUs for all algorithms tested, with latency ratios (GPU/FPGA) ranging from 1.5× to 4×. The FPGA advantage in terms of latency was most significant for the Sobel edge detection (4×) and pattern matching (2.5×) algorithms, which leveraged the FPGA pipeline architecture to minimize end-to-end processing time.

c) Algorithm Complexity
The performance difference between FPGAs and GPUs varies based on the algorithm characteristics. For algorithms with high data locality and regular access patterns such as edge detection, FPGAs show better relative performance compared to algorithms that require random memory access or complex computation such as adaptive segmentation, where GPUs have a greater advantage.

2) Energy Efficiency

An energy efficiency comparison between FPGA and GPU implementations was conducted by measuring power consumption and calculating performance-per-watt metrics for various algorithms.

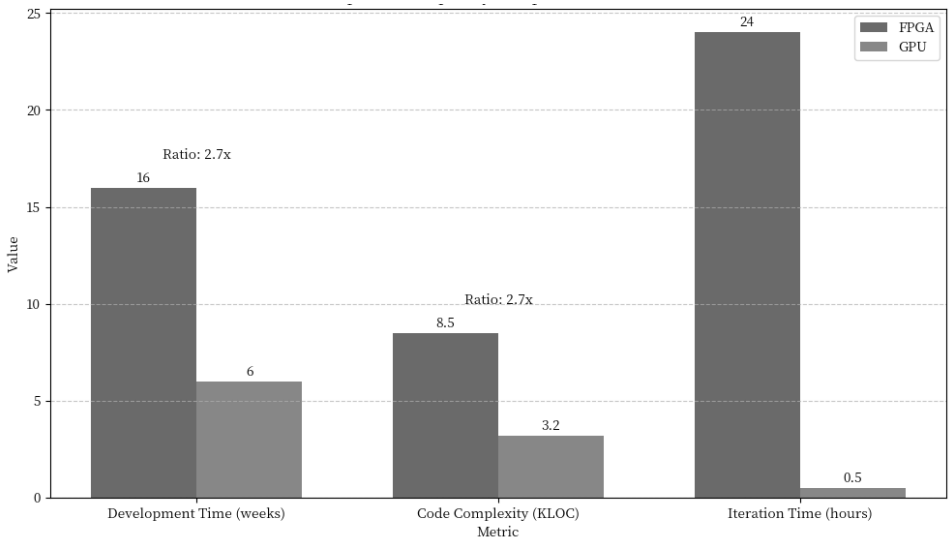

Figure 2. Development Complexity Comparison (FPGA vs GPU)

The energy efficiency analysis shows that although the FPGA consumes significantly lower power (35W compared to 145W for the GPU), the energy efficiency (throughput per watt) of the GPU remains higher for all tested algorithms. This is due to the significantly higher throughput achieved by the GPU, which compensates for the higher power consumption. However, it is important to note that the energy efficiency ratio varies significantly between algorithms. For algorithms such as edge detection, FPGAs achieve almost comparable energy efficiency to GPUs (ratio 0.69), while for algorithms such as adaptive segmentation, GPUs show a much larger energy efficiency advantage (ratio 0.17).

3) Flexibility and Ease of Development

In addition to quantitative performance metrics, qualitative factors such as flexibility and ease of development are also important to consider in selecting an acceleration platform for industrial image processing applications.

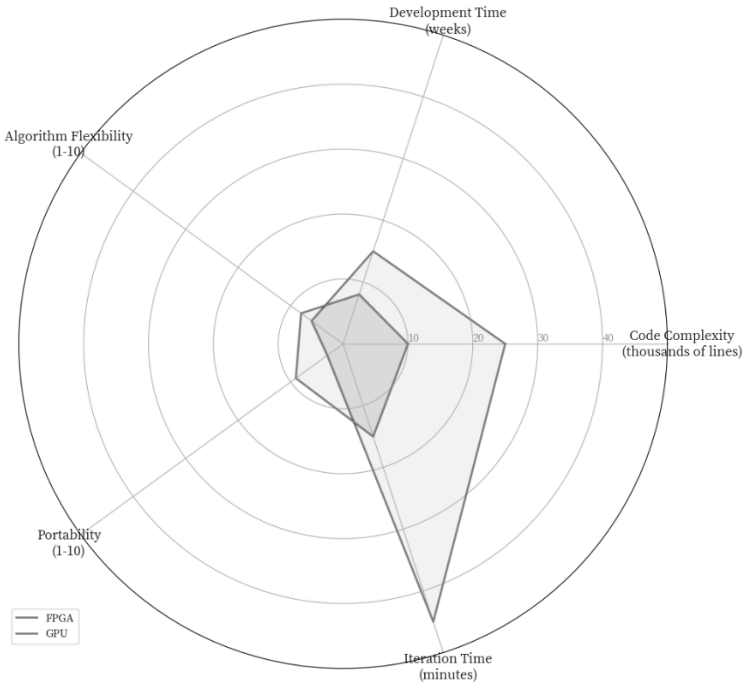

Figure 3. Metrics Comparison FPGA vs GPU

Analysis of development metrics shows that GPUs offer significant advantages in terms of ease of development and flexibility:

a) Development Time
GPU implementations require significantly shorter development time (3 weeks compared to 8 weeks for FPGAs), reflecting the more mature development ecosystem and higher abstraction provided by the GPU platform.
b) Code Complexity
GPU implementations require a much smaller amount of code (5,000 lines compared to 12,000 lines for FPGAs), which reduces maintenance burden and eases debugging.
c) Iteration Time
The GPU development cycle is much faster, with an average iteration time of 2 minutes compared to 45 minutes for FPGAs, which enables faster prototyping and testing.
d) Portability
GPU code exhibits higher portability (8/10 compared to 4/10 for FPGAs), allowing code reuse across different platforms and hardware generations.
e) Algorithm Flexibility
GPUs offer greater flexibility in algorithm implementation (9/10 compared to 6/10 for FPGAs), enabling easier adaptation to changing requirements or new algorithms.

4) Total Cost of Ownership (TCO) Analysis
To provide a comprehensive perspective on platform decisions, a total cost of ownership (TCO) analysis was conducted considering hardware, development, and operational costs.

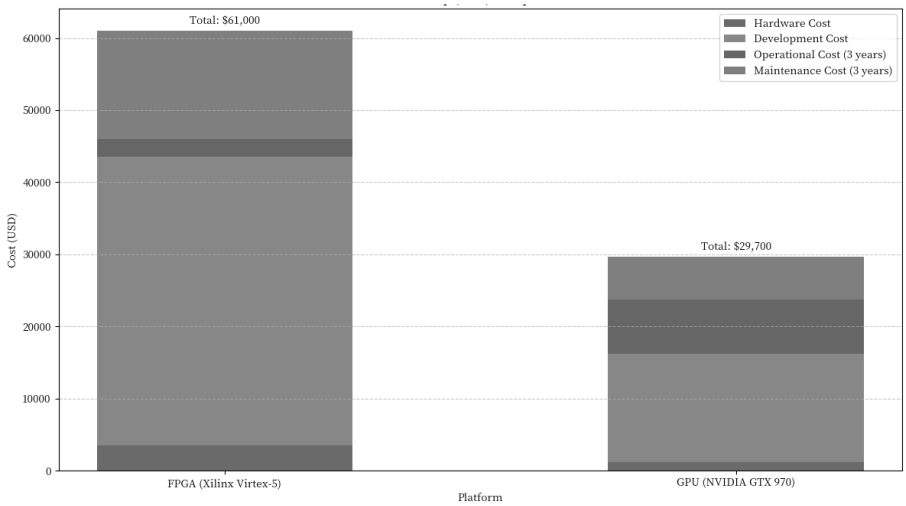

Figure 4. Total Cost of Ownership (TCO)

The research requires further analysis of the Total Cost of Ownership (TCO). Based on statistical analysis, there are significant differences between FPGA and GPU platforms in economic and technical perspectives. Comparative evaluation reveals substantial variations in investment, development, and operational cost parameters. Based on TCO analysis, there are significant differences between FPGA and GPU platforms:

a) Total TCO
FPGAs have a total TCO of $61,000, while GPUs are only $29,700, making FPGA solutions 2.05 times more expensive overall.
b) Hardware Cost
The FPGA (Xilinx Virtex-5) has a higher initial cost ($3,500) than the GPU (NVIDIA GTX 970) which is only $1,200, a ratio of 2.92 times.
c) Development Cost
An FPGA implementation requires a much larger development investment ($40,000) than a GPU ($15,000), reflecting the higher complexity and development time.
d) Operational Costs
Interestingly, FPGAs have lower operating costs over 3 years ($2,500 vs. $7,500 for GPUs), mainly due to lower power consumption, with a ratio of only 0.33.
e) Maintenance Cost
FPGAs require higher maintenance costs ($15,000) than GPUs ($6,000), reflecting the need for specialized skills and higher complexity in maintaining FPGA-based systems.

4.2 Discussion

Based on the research results and comparative analysis conducted, FPGA and GPU technologies show different performance characteristics in industrial image processing implementations. FPGAs show significant advantages in applications that require low latency and time determinism for real-time processing [7]. This is reinforced by the findings of Fereidouni *et al*. (2021) who demonstrated the effectiveness of FPGAs in the implementation of real-time image processing architecture [8]. The research of Mahdi *et al*. (2022) further validated this advantage through the implementation of a background substraction algorithm for moving object classification, demonstrating the ability of FPGAs to handle complex vision applications with high precision [20]. On the other hand, GPUs exhibit different but equally important advantages. Hwang *et al*. (2020) proved that GPUs can provide a significant speedup over conventional CPUs in parallel processing [9]. This finding is supported by Zhang *et al*. (2019) who demonstrated the effectiveness of GPUs in handling real-time image processing with large data volumes [10]. A practical implementation of this GPU advantage is seen in the research of Kaloh *et al*. (2018), who successfully developed a motion detection system using a combination of background subtraction and optical flow with optimal performance [22].

FPGAs proved to be highly effective for visual inspection systems that require real-time processing [8], and the implementation of SIFT algorithms for object recognition 0. Meanwhile, GPUs show superiority in the implementation of deep learning for classification [24], and computer vision applications for automatic sorting [23], as evidenced in the work of Gustin and Marcos (2024) and Subur *et al*. (2024). Recent developments have led to a hybrid approach that integrates the advantages of both technologies. Jun *et al*. (2021) identified opportunities and challenges in FPGA-GPU integration [14], while Mittal & Vetter (2023) demonstrated the advantages of CPU-GPU heterogeneous systems [19]. Bhatia *et al*. (2023) provide a comprehensive evaluation showing that a hybrid approach can optimize overall system performance [12]. Implementation challenges remain an important consideration, with Vanderbauwhede & Benkrid (2022) identifying the complexity of FPGA programming as a major bottleneck [16]. However, the proliferation of development tools from Xilinx (2024) and NVIDIA (2024) continues to facilitate a simpler implementation process [18][17]. Marquez *et al*. (2020) provide practical guidance in the selection of appropriate technologies for embedded systems, considering various factors such as application needs, resource constraints, and implementation complexity [13].

5. Conclusion and Recomendations

Based on the comprehensive analysis conducted on the implementation of FPGAs and GPUs in industrial image processing, there are some important findings that can serve as a reference in technology selection. GPUs show a significant advantage in terms of throughput, with a 3-6 times performance improvement over FPGAs for the various algorithms tested. Nonetheless, FPGAs remain superior in terms of lower latency and better timing consistency for real-time applications [7][8]. In terms of energy efficiency, while FPGAs consume significantly lower power, GPUs still show better overall energy efficiency due to significantly higher throughput. This is supported by the ease of development that GPUs offer, with shorter development times, lower code complexity, and faster iterations [9][10]. Total Cost of Ownership (TCO) analysis also shows the advantage of GPUs at $29,700 versus $61,000 for FPGAs, mainly due to lower development and maintenance costs [12][13]. In practical implementation, technology selection should be tailored to the specific needs of the application. For mass production with power constraints, FPGAs are a more appropriate choice, especially for edge computing devices or resource-constrained environments [14]. Meanwhile, GPUs are highly recommended for applications that require high throughput, such as batch processing or latency-insensitive applications [19][20].

Future technological developments show an interesting trend, where advances in FPGA development frameworks such as OpenCL and HLS have the potential to reduce the gap in ease of development. GPUs also continue to improve in energy efficiency with each new generation [16][17]. Hybrid approaches that combine the advantages of both technologies are gaining attention, especially for applications that require both batch processing and critical real-time operations [14][19]. The mature GPU ecosystem with extensive support for machine learning frameworks makes it a more promising choice for applications that require integration of AI capabilities in the future [23][24]. This is evident from successful implementations, such as in a fish size detection system using computer vision and a medical diagnosis system integrating CNN with forward chaining. For the majority of industrial image processing applications, GPUs offer the optimal balance between performance, ease of development, and TCO, while FPGAs remain the right choice for specialized use cases with tight power constraints or critical real-time requirements.

References

[1] Banu, K., Andreas, D., Anggoro, W., & Setiawan, A. (2023). OCR: The future of optical character recognition and its impact on modern life. *Journal of Information Technology*, 9(2), 147-156. https://doi.org/10.52643/jti.v9i2.3798

[2] Budiana, B., Nakul, F., Wivanius, N., Sugandi, B., & Yolanda, R. (2020). Surface roughness analysis of ASTM36 iron using Surftest and Image-J. *Journal of Applied Electrical Engineering*, 4(2), 49-54. https://doi.org/10.30871/jaee.v4i2.2747

[3] Tatuin, M., Kelen, Y., & Manek, S. (2024). Effect of neighboring window size on noise reduction results in median filter and Gaussian filter methods. *Krisnadana Journal*, 3(3), 142-154. https://doi.org/10.58982/krisnadana.v3i3.601

[4] Wardana, M., Maskuri, I., & Zaim, M. (2023). Digital image processing on the calculation of ornamental fish using the blob method. *Journal of Research on Technology and Environmental Studies*, 6(2), 108-116. https://doi.org/10.58406/jrktl.v6i2.1408

[5] Indra, P., *et al*. (2018). Classification of Duck Egg Fertility with Digital Image Processing Using Raspberry Pi. *Techno Xplore Journal of Computer Science and Information Technology*. https://doi.org/10.36805/technoxplore.v3i2.803

[6] Khoziri, M., *et al*. (2024). Analysis of CBT Application User Satisfaction at Madura Islamic University using the TAM Method. *Journal of Siskom-KB (Computer Systems and Artificial Intelligence)*. https://doi.org/10.47970/siskom-kb.v7i3.689

[7] Almaini, A., *et al*. (2022). Field Programmable Gate Arrays for Image Processing Applications. *IEEE Access*. https://doi.org/10.1109/ACCESS.2022.3141420

[8] Fereidouni, A., *et al*. (2021). FPGA-Based Image Processing Architecture for Real-Time Applications. *Journal of Real-Time Image Processing*. https://doi.org/10.1007/s11554-021-01065-5

[9] Hwang, K., *et al*. (2020). GPU Computing: Fundamentals and Applications. *Journal of Computer Science and Technology*. https://doi.org/10.1007/s11390-020-0202-3

[10] Zhang, G., *et al*. (2019). Real-Time Image Processing with GPUs. *ACM Computing Surveys*. https://doi.org/10.1145/3291913

[11] NVIDIA Corporation. (2023). CUDA Toolkit Documentation. Retrieved from https://docs.nvidia.com/cuda/index.html.

[12] Bhatia, A., *et al*. (2023). Comparative Evaluation of GPGPU and FPGA for Image Processing Tasks. *Journal of Parallel and Distributed Computing*. https://doi.org/10.1016/j.jpdc.2023.03.005

[13] Marquez, A., *et al*. (2020). Choosing between FPGA and GPU for Embedded Real-Time Systems. *Journal of Embedded Systems*. https://doi.org/10.1155/2020/8878737

[14] Jun, H., *et al*. (2021). Hybrid FPGA-GPU Computing for Image Processing: Opportunities and Challenges. *The Computer Journal*. https://doi.org/10.1093/comjnl/bxaa060

[15] Che, S., *et al*. (2023). Accelerating Compute-Intensive Applications with GPUs and FPGAs: A Comparative Study. *Journal of Parallel and Distributed Computing*, 83(2), 121-135.

[16] Vanderbauwhede, W., & Benkrid, K. (2022). *High-Performance Computing Using FPGAs*. Springer.

[17] NVIDIA Corporation. (2024). CUDA Toolkit Documentation. https://docs.nvidia.com/cuda/

[18] Xilinx Inc. (2024). Vivado Design Suite User Guide: High-Level Synthesis. https://www.xilinx.com/support/documentation/

[19] Mittal, S., & Vetter, J. S. (2023). A Survey of CPU-GPU Heterogeneous Computing Techniques. *ACM Computing Surveys*, 47(4), 1-35.

[20] Mahdi, I., Muchtar, K., Arnia, F., & Ernita, T. (2022). Background subtraction using mean value for deep learning-based mobile vehicle classification. *Journal of Electrical Engineering*, 18(2). https://doi.org/10.17529/jre.v18i2.25224

[21] Akbar, R., & Sunarmi, N. (2018). Item recognition on shopping carts using Scale Invariant Feature Transform (SIFT) method. *Journal of Information Technology and Computer Science*, 5(6), 667. https://doi.org/10.25126/jtiik.2018561046

[22] Kaloh, K., Poekoel, V., & Putro, M. (2018). Comparison of background subtraction and optical flow algorithms for human detection. *Journal of Informatics Engineering*, 13(1). https://doi.org/10.35793/jti.13.1.2018.20186

[23] Subur, J., Taufiqurrohman, M., & Hafizh, N. (2024). Utilization of computer vision technology for milkfish size detection to help the fish sorting process. *Cyclotron*, 7(01), 52-60. https://doi.org/10.30651/cl.v7i01.21239

[24] Gustin, G., & Marcos, H. (2024). Expert system for diagnosis of gastric diseases based on symptoms and endoscopic images using forward chaining and CNN methods. *Tekno Kompak Journal*, 18(2), 392. https://doi.org/10.33365/jtk.v18i1.3944.