Radar Detection and Tracking with GPU

Authors: Daniel Bower, Kevin Connelly
Department of Electrical and Computer Engineering, University of Arizona, Tucson AZ, 85721

***Abstract*—Our research investigates the enhancement of radar signal processing algorithms through the application of Graphics Processing Units (GPUs) and the CUDA programming model, specifically focusing on Kalman filtering and Constant False Alarm Rate (CFAR) algorithms. Given the computationally intensive nature of these algorithms, particularly in real-time radar signal processing applications, our study explores the performance improvements obtained by their parallel implementation. This paper details the process of adapting these algorithms to a parallel computing architecture, discusses the optimization strategies employed, and presents a comparative analysis of performance metrics against traditional CPU-based implementations. Additionally, the integration of both algorithms operating in conjunction enables a more robust and efficient radar signal processing pipeline. Experimental results demonstrate reductions in processing time and enhanced detection capabilities, illustrating the feasibility and benefits of GPU-accelerated radar systems. This study underscores the pivotal role of advanced computational techniques in advancing the efficacy and speed of critical defense-related signal processing tasks.**

***Index Terms*-Graphical Processing Unit, Radar Signal Processing, Kalman Filter, Constant False Alarm Rate**

I. Introduction

In modern radar systems, real-time signal processing is not just a requirement but a critical necessity for effective operation. Among the various algorithms employed, Kalman filtering and Constant False Alarm Rate (CFAR) are pivotal in enhancing the accuracy and reliability of radar detection and tracking. However, traditional implementations of these algorithms on CPUs often struggle to meet the demands of high-resolution, high-update-rate radar systems, primarily due to the inherent limitations in sequential processing capabilities. This computational gap not only hinders system performance but also limits the potential for deploying more sophisticated radar algorithms that require real-time processing. To address these challenges, this research explores the use of Graphics Processing Units (GPUs) and CUDA (Compute Unified Device Architecture) to improve the efficiency and speed of these algorithms.

The parallel processing capabilities of GPUs make them an ideal platform for executing complex radar signal processing tasks that are computationally intensive and easy to implement in parallel. By using CUDA to implement Kalman filtering and CFAR algorithms, this study demonstrates an enhancement in processing speed and efficiency. Techniques such as the utilization of shared memory and streams are specifically exploited to optimize the computation. Shared memory, due to its proximity to processor cores on the GPU, provides a much faster alternative to global memory, reducing latency and increasing throughput. Meanwhile, the use of CUDA streams allows for concurrent executions of sequences of operations, further optimizing the workflow and reducing the time to completion.

The results of this approach are quantitatively significant, showing up to a 40% decrease in computation time compared to conventional CPU-based implementations. Through a detailed comparative analysis, this study illustrates how GPU-accelerated radar processing surpasses traditional methods, offering a solution to the latency and scalability issues that plague CPU-based systems. Thus, the integration of GPU computing into radar signal processing marks a critical advancement in the field, promising enhanced system capabilities and setting a new standard for future developments.

II. Description

A. Kalman Filtering

Kalman filtering is a statistical method that provides a computational solution to the linear least squares problem. It is widely used in various applications for estimating the state of a linear dynamic system from a series of noisy measurements. Initially developed in the late 1950s and early 1960s by Rudolf E. Kalman, the filter has since become a fundamental tool in areas such as aerospace and robotics for real-time navigation and tracking systems.

The core idea behind Kalman filtering is to predict the state of a system at a given time and then update this prediction based on new measurements. The filter operates recursively, which means it processes all available measurements up to a given point, then produces estimates of current state variables, essentially blending prediction and measurement using statistical inference. The process involves two distinct phases: the prediction phase and the update phase. During the prediction phase, the Kalman filter uses the state from the previous time step to produce an estimate of the state at the current time step. This prediction is then updated as soon as the new measurement is available.

The Kalman filter is particularly effective at managing uncertainty in system models and measurements. It does this by utilizing probability distributions, specifically assuming that the noise in both the processes and measurements is Gaussian. The filter uses the mean and covariance of the state's distribution as its key variables. This allows it to provide the most likely estimate of the system's current state and to update the estimate of uncertainty each time a new measurement comes in.

Kalman filters are particularly valuable in situations where data is uncertain or incomplete, and where it is necessary to estimate variables dynamically over time. They are optimal under conditions of Gaussian noise and when the system dynamics and measurement models are linear. For non-linear models, extensions such as the Extended Kalman Filter (EKF) and the Unscented Kalman Filter (UKF) have been developed to handle the non-linearities.

B. CFAR

The CFAR algorithm is used to determine a higher than average value in a specific cell under test (CUT). Typically CFAR algorithms are used to determine the presence of a target in a radar return. In Figure 1 you can see a basic implementation of a CFAR algorithm. The result of the algorithm is usually a binary 1 or 0, corresponding to a detected target (1) or no detected target (0). There are many varieties of CFAR algorithms, with each one performing differently depending on the expected radar return data. The Cell Averaging CFAR (CA-CFAR) algorithm was chosen for its high reliance on training cells and simplicity of execution. This allows for a better determination of performance for a given optimization. These results will be explored later in section IV.

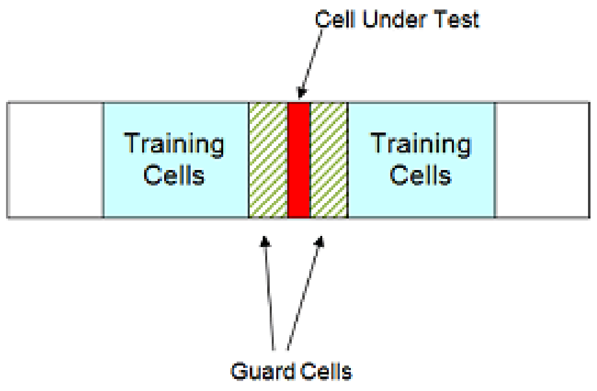

Fig. 1. A basic CFAR algorithm. Courtesy of Mathworks.

III. Related Work

A. Kalman Filtering

One study that delves into the optimization of Kalman filtering through GPU technology is titled 'A GPU-based Kalman Filter for Track Fitting[1].' This research focuses on the significant enhancements GPU acceleration can bring to the computational efficiency of Kalman filtering, especially within track fitting processes in high-energy physics. Although the specific application discussed in this study differs from ours, the underlying algorithm remains the same. The authors detail their rationale for employing CUDA and GPUs in the context of Kalman filtering. However, they do not elaborate on the specific implementation details of the algorithm nor do they provide quantitative performance metrics. Another significant contribution to this field is detailed in the study titled 'Kalman Filter Tracking on Parallel Architectures[[2].' This paper elaborates on how leveraging GPU acceleration can improve the computational efficiency of Kalman filtering, particularly emphasizing its application in track fitting utilized within high-energy physics. While the authors do not go into specific implementation details they do provide the results of their GPU implementation. In particular, they mention that the total time of the program tends to be dominated by the setup time of the GPU.

B. CFAR

There are several works that explore the efficiency of using a GPU to compute CFAR algorithms. The paper on Ordered-Statistic CFAR[3] analyzes the efficiency of an ordered-statistic CFAR (OS-CFAR) instead of a CA-CFAR. While the algorithms differ it highlights the fact that CA-CFAR produces a very reliable detection rate when given a homogeneous set of input data. The data used to test the CFAR algorithm were generated in Matlab using several built in functions and as such output highly homogeneous and random data. The paper focuses on the effect of varying the training bin size, but highlights the performance gain of using a GPU over a CPU. The paper also explores the power efficiency of the solutions, which provides an excellent point of comparison between the solutions. This is especially important for battery dependent systems like small drones or electric cars.

A study on various CFAR algorithms was also conducted by the Kaman Sciences Corporation[4] that explores the performance of each algorithm and their efficacy relative to each other. This was used as the basis for determining the type of data to input to the CFAR kernel as a set of data that was "homogeneous, Gaussian, independent and identically distributed".

IV. Methodology

A. Kalman Filtering

We aimed to evaluate the performance of three distinct implementations of GPU based Kalman filtering, benchmarked against a serial implementation to establish a baseline for performance enhancements through parallel computing. Our experimental setup utilized a Slurm script configured to automate the execution of 30 separate tests for each algorithm version within a high-performance computing environment. The first implementation was serial, focusing on foundational performance metrics. Subsequent versions were optimized for multi-threading and GPU acceleration using CUDA, designed to address and alleviate computational bottlenecks identified in the initial serial implementation.

Each algorithm's implementation was tested 30 times to mitigate any anomalies caused by system performance fluctuations or external variables, with the script recording precise execution times for each run. This extensive testing procedure ensured that the data collected—execution times across all trials—was reliable. Statistical analysis was then performed to calculate the average execution times, providing a quantitative basis for comparing the performance enhancements of GPU-accelerated versions against the serial baseline.

B. CFAR

The main testing method for the CFAR algorithm was using the built in CUDA function cudaEventRecord(). This allowed for precise timing of the kernel execution time. Various input data sizes were generated with serial code and then fed to the specified kernel to execute. The final implementation of the CFAR kernel was based on repeated testing with this method to achieve lower execution times. A global memory version of the CFAR algorithm was first written to serve as a baseline for further optimizations.

V. Evaluation And Validation

A. Kalman Filtering

The critical metric for evaluating the Kalman filtering algorithm in this study was the execution time, especially considering its deployment in real-time tracking systems where delays can lead to significant data losses. To gauge the effectiveness of GPU acceleration, the Kalman filtering process was executed across three different kernel configurations: Basic, Shared Memory, and Streaming, each optimized for GPU execution.

Initially, a baseline dataset was generated using serial code to simulate typical radar tracking scenarios. This dataset was then processed through the Kalman filter implemented in the CUDA environment under various block and thread configurations to optimize performance. Each configuration was run 30 times to ensure statistical reliability and to mitigate variations caused by transient system behaviors or external interruptions.

Performance data was collected and analyzed, comparing execution times across the three GPU configurations against the serial implementation to quantify the improvement. The configurations tested included combinations of 4 and 8 blocks with 256 and 512 threads respectively. This testing aimed to identify the most effective GPU setup for minimizing the execution time of the Kalman filter, thereby enhancing the throughput and responsiveness of the entire radar processing system.

B. CFAR

The most important aspect of the CFAR kernel is execution time. A CFAR algorithm is generally implemented as a single stage in a larger radar processing system. This means that functions before and after the algorithm are dependent on the output and any delay could cause downstream issues. Some systems implement a ”reset or discard” approach where any delay in the system will dump the entire frame or set of data and immediately move to the most current. Therefore ensuring the CFAR algorithm completes in as little time as possible is beneficial to the system as a whole.

To test the full functionality of the CFAR algorithm an initial data set was generated in serial code before executing the kernel. The dataset is assumed to be an m x n matrix that represents range and relative speed respectively. An arbitrary number was chosen as the base, in this case 20, and then a random number from 0-20 was added on to each element in the dataset. This serves as the homogeneous background noise that the target will sit in. A target was then placed manually at a specified location. The target has a higher value than the background noise by a factor of 2. Adjusting the level of the target relative to the background noise will still result in a detection but introduces more variance in the output. A value of 90 was chosen to provide a definitive detection point.

VI. Experimental Results and Analysis

A. Kalman Filtering

The evaluation process began with the generation of a dataset designed to test the efficiency and effectiveness of the parallelized Kalman filtering algorithms. The dataset was first processed using a serial implementation of the algorithm in MATLAB, establishing a baseline for performance metrics such as execution time. The average of the 10 executions equaled 0.0056 seconds. This baseline provided a reference point against which all subsequent enhancements could be quantitatively compared.

Following the establishment of a baseline, the first version of the algorithm was implemented using CUDA, specifically utilizing only global memory to handle data within the GPU environment. This implementation was designed to explore the performance gains from merely offloading computations to a GPU, without any advanced memory management optimizations. The algorithm was executed 30 times with 4 different configurations, and the execution times were recorded and averaged to assess the mean performance. This average execution time was then compared directly to the serial MATLAB implementation to quantify the improvements or drawbacks of using a basic GPU acceleration approach.

TABLE I
Basic Kalman Filtering Execution Times

Serial Implementation	.0056 ms
4 Blocks 256 Threads	55.5 μs
4 Blocks 512 Threads	54.9 μs
8 Blocks 256 Threads	57.7 μs
8 Blocks 512 Threads	51.4 μs

Next, a second version of the CUDA code was developed, this time incorporating the use of shared memory within the GPU. The use of shared memory aimed to reduce the latency and bandwidth issues associated with global memory access by allowing faster access to frequently used data within the confines of each block of threads. Similar to the previous GPU implementation, this version was also executed 30 times, with 4 different configurations, for statistical reliability, and the resulting data was averaged and analyzed. The performance metrics from this iteration were then compared not only to the baseline serial implementation but also against the first CUDA implementation to gauge the incremental benefits of using shared memory in GPU computations.

TABLE II
Kalman Filtering With Shared Memory Execution Times

Serial Implementation	.0056 ms
4 Blocks 256 Threads	57.9 μs
4 Blocks 512 Threads	52.5 μs
8 Blocks 256 Threads	53.1 μs
8 Blocks 512 Threads	58.5 μs

Finally, the algorithm underwent a third enhancement, integrating CUDA streams to allow overlapping of computations and data transfers within the GPU. This approach was intended to maximize the utilization of the GPU's resources by reducing idle times and increasing parallelism even further. Again, this version was subjected to the same rigorous testing—30 executions across 4 configurations with averaged timings. The evaluation of this version focused on comparing its performance against all previous versions (serial MATLAB, CUDA with global memory, and CUDA with shared memory), providing a comprehensive view of how each modification impacts overall efficiency.

TABLE III
KALMAN FILTERING WITH SHARED MEMORY USING STREAMS EXECUTION TIMES

Serial Implementation	.0056 ms
4 Blocks 256 Threads	54.8 μs
4 Blocks 512 Threads	52.1 μs
8 Blocks 256 Threads	52.9 μs
8 Blocks 512 Threads	56.2 μs

Below is a graph representing a comparison of the execution times for different kernel configurations using Basic, Shared Memory, and Streaming approaches. The graph illustrates the differences in performance across these configurations. Notably, the variations in execution times are minor and sometimes counterintuitive, suggesting that the improvements from more sophisticated memory management techniques such as shared memory and streaming are not as pronounced as expected. This subtlety in the results likely indicates that the overhead associated with setting up and managing the GPU kernels predominates the total execution time. This observation underscores the importance of considering overhead costs when evaluating the effectiveness of different GPU optimization strategies in real-world applications.

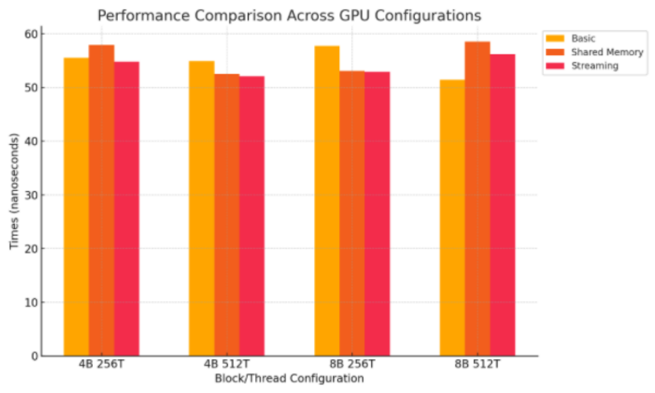

Fig. 2. Comparision of executions times for each kernel.

B. CFAR

Table 2 shows the execution time for the three versions of the CFAR algorithms. The streaming kernel implementation was unable to provide a noticeable time reduction and will be discussed at the end of this section. The shared memory version reduced the execution time by about 35%.

One important aspect of the CFAR algorithm that was tested was how well the kernel performed with increasing amounts of data. The hypothesis was that increasing the amount of data (corresponding to increasing the Z-Depth of the Range-Doppler Matrix) would increase the execution time in a linear manner[5]. As can be seen in Figure 2, increasing the Z-Depth led to a linear increase in the execution time. At 60 frames per second, a radar system will need to process 3,600 frames in 1 minute. Any increase in execution time could therefore cascade through the system and cause issues later in the signal processing path.

TABLE IV
CFAR ALGORITHM EXECUTION TIME

Basic Kernel	241.8 μs
Shared Memory Kernel	155.6 μs
Streaming and Shared Kernel	155.6 μs

This also shows the increase in the shared memory version compared to the global memory implementation (labeled as "Basic" in Figure 2). The difference between the two versions increases linearly with increased Z-Depth. Because the CFAR algorithm is expected to be run in real-time for the duration of the radar, this reduction in execution time is meaningful for the signal processing of the radar system as a whole.

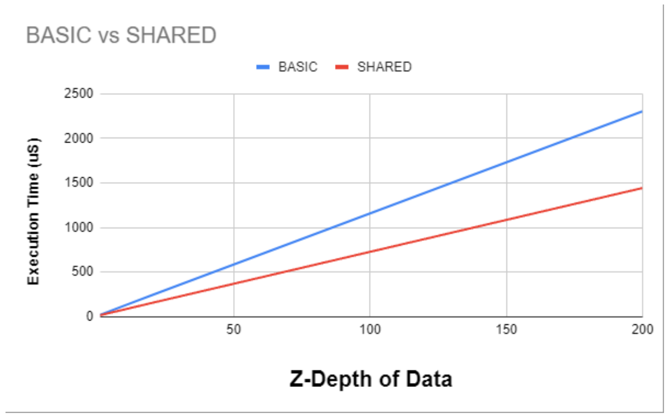

Fig. 3. Execution time for various Z-Depths of data.

Along with the execution time, the run to run variance of the kernel was also profiled. Using a Z-Depth of 200, the average execution time was 1444μs. The variance was $\pm 5\mu$s, which amounts to a standard deviation of only 1.84. This compares to the basic kernel that had $\pm 12\mu$s with a standard deviation of 3.32. This result shows that decreasing the execution time can also improve the variance of the kernel, which is beneficial for systems that rely on many individual steps to provide a solution.

Along with the Basic and Shared Memory kernel, Streaming was also attempted to reduce the execution time of the code. This would enable several streams of data to each grab a frame and process them in parallel. Ideally this would reduce the execution time further and allow for larger data sets. Unfortunately the implementation did not achieve a reduction in execution time, and in fact produced no statistically significant difference. This indicates that the streaming solution was not working correctly and is a future area of investigation and improvement.

VII. Conclusion

In conclusion this study on using a GPU to calculate complex radar processing algorithms shows that with modern GPUs the possibility to process real-time radar data is feasible. While this study focused on two significant algorthims involved in the signal processing tree for a radar system, similar performance gains should be possible along the entire radar processing chain. Both algorithms focused on reducing the execution time of their respective kernels.

The Kalman filtering was able to utilize shared memory and streaming to achieve a reduction in execution time, although this only held for a specific block size. The CFAR algorithm was able to achieve a significant performance reduction utilizing shared memory.

This study holds significant potential for future study. A significant avenue of study could be implementing a complete signal processing system in CUDA and compare the performance relative to current solutions. The efficiency in terms of power usage could also be explored in more detail and highlight the importance of low-power devices and solutions in today's technological atmosphere.

References

[1] X. Ai, G. Mania, H. M. Gray, M. Kuhn, and N. Styles, "A gpu-based kalman filter for track fitting," *Computing and Software for Big Science*, 2021.

[2] G. Cerati, P. Elmer, S. Krutelyov, S. Lantz, M. Lefebvre, K. McDermott, D. Riley, M. Tadel, P. Wittich, F. Wurthwein, and A. Yagil, "Kalman filter tracking on parallel architectures," *EPJ Web of Conferences*, 2016.

[3] M. Bales, T. Benson, R. Dickerson, D. Campbell, and R. Hersey, "Real-time implementations of ordered-statistic CFAR," in *Sensors and Electromagnetic Applications Laboratory, Georgia Tech Research Institute*, Atlanta, Georgia, 2015.

[4] Kaman Sciences Corporation, "Intelligent use of CFAR algorithms," Griffiss Air Force Base, New York, Tech. Rep., 1993.

[5] X. Zhao, P. Liu, B. Wang, and Y. Jin, "Gpu-accelerated signal processing for passive bistatic radar," *Remote Sensing*, 2023.