Minor Thesis for Bachelor of Computer Science (Honours)

GPGPU Multi Object Bayesian Tracking with an embedded System on a Chip

David Webb

Supervisor: A. Prof. David Kearney
ABSTRACT

Current embedded multi object tracking system implementations are dominated by the use of Field Programmable Gate Arrays (FPGAs) and Application Specific Integrated Circuits (ASICs) as application accelerators. These offer many of the traits desirable for embedded and real time systems, including; task oriented architectures, deterministic latency and low power requirements. The drawbacks to these approaches are the difficulty of development and the high costs associated with prototyping and developing systems and hardware. Other embedded approaches feature simplified algorithms which are not robust for multiple types of targets or complex scenes.

Modern embedded system on a chip (SoC) platforms offer GPUs that are now capable of general purpose computing, enabling the efficient acceleration of parallel problems. This study focuses on multi-object trackers that are capable of tracking targets without the need for target specific customisation. Such trackers usually require computation resources that are unavailable on embedded platforms. The use of full sized computers is not viable for mobile applications which have constraints on available power and heat dissipation.

This thesis presents what is believed to be the first GPU accelerated embedded SoC based implementation of a complex Bayesian multi-object tracker. The thesis explores the applicability and optimal usage of CPU and GPGPU techniques for the tracker on both SoC and Desktop platforms. The presented SoC based implementation is a heterogeneous design, with a hybrid CPU and GPU based processing path. This implementation capable of real time tracking of four targets at 21.5 frames per second. The SoC implementation is five times more power efficient when compared with an existing FPGA accelerated CPU based implementation. It is 85% more power efficient than the same version implemented on a desktop CPU and discrete GPU. The desktop GPU accelerated implementation provides the highest throughput version of the CACTuS multi-target tracker to date. The research has shown that a SoC based platform, using the GPU and CPU implementation of the tracker is more power efficient on a frames per second per Watt basis than using the CPU alone.
DECLARATION

I declare that:

- this thesis presents work carried out by myself and does not incorporate without acknowledgment any material previously submitted for a degree or diploma in any university;

- to the best of my knowledge it does not contain any materials previously published or written by another person except where due reference is made in the text; and all substantive contributions by others to the work presented, including jointly authored publications, is clearly acknowledged.

David Webb

Adelaide, December 2015
ACKNOWLEDGMENTS

Thanks to my supervisor, A. Prof. David Kearney, for indulging me in my self-directed research. Thanks to Victor Stamatescu who has politely tolerated my interruptions and answered my many questions during this research. Special thanks to Anthony Milton, who's previous work served as a strong foundation to this research. Loving thanks to my wife and daughter for their love and support during the completion of my studies and this research.
# TABLE OF CONTENTS

ABSTRACT ......................................................................................................................... 1

DECLARATION ................................................................................................................. 2

ACKNOWLEDGMENTS ...................................................................................................... 3

LIST OF ABBREVIATIONS ............................................................................................... 6

LIST OF FIGURES ............................................................................................................ 7

1. INTRODUCTION ........................................................................................................ 8
   1.1. The Problem ........................................................................................................ 11
   1.2. Research Questions ............................................................................................ 13

2. LITERATURE REVIEW ............................................................................................. 14
   2.1. Computer Vision ................................................................................................ 14
   2.1.1. Early Computer Vision ............................................................................... 15
   2.1.2. Practical Computer Vision ......................................................................... 16
   2.2. Object Tracking .................................................................................................. 17
   2.2.1. Bayesian Techniques .................................................................................. 18
   2.2.1. Multi Object Tracking .............................................................................. 19
   2.2.2. The CACTuS Tracker .............................................................................. 19
   2.2.3. Acceleration Through Parallel Computation ............................................ 20
   2.3. GPGPU Computing ............................................................................................ 21
   2.3.1. Limitations of GPU Computing ................................................................ 22
   2.4. Embedded Systems ............................................................................................ 23
   2.4.1. Object Tracking on Embedded Systems ..................................................... 24
   2.4.2. NVIDIA Jetson TK1 Platform .................................................................... 25
   2.5. Conclusion ........................................................................................................... 26

3. RESEARCH METHOD ............................................................................................... 27
   3.1. Formalised Research Questions ................................................................. 30
   3.2. Methodology .................................................................................................... 30
   3.2.1. RQ1 CACTuS on an embedded SoC platform ......................................... 31
   3.2.2. RQ2 The impact of SoC constraints on tracking performance ............... 31
3.2.3. RQ3 Does CUDA improve the tracking implementation................................. 31
3.2.4. RQ4 Does the SoC meet real time constraints........................................... 32
3.2.5. RQ5 Power efficiency of new implementations .......................................... 32
3.3. Conclusion.................................................................................................. 32

4. EXISTING CACTuS IMPLEMENTATIONS ......................................................... 33
4.1. The MATLAB implementation...................................................................... 33
4.2. CACTuS Armadillo Implementation............................................................ 33
4.3. CACTuS Vienna Implementation................................................................. 34
4.4. Heterogeneous CACTuS Armadillo and Vienna........................................... 35
4.5. Recommendations......................................................................................... 36

5. SYSTEM IMPLEMENTATION ........................................................................ 37
5.1. High Performance Strategies for CPU targeted CACTuS................................. 37
5.2. Development and Testing Setup.................................................................... 39
5.3. CACTuS CP - an Intermediate Step............................................................ 39
  5.3.1. Processing Throughput of CACTuS CP.................................................. 42
5.4. GPGPU Acceleration - CACTuS CUDA.................................................... 44
5.5. CACTuS CUDA on Jetson............................................................................ 47
  5.5.1. CACTuS CUDA Hybrid......................................................................... 49
  5.5.2. Real time on Jetson.............................................................................. 51
5.6. Tracking Performance.................................................................................. 52
5.7. Performance per Watt.................................................................................. 54

6. CONCLUSION ................................................................................................ 57
6.1. Future Research Directions......................................................................... 58

REFERENCES.................................................................................................... 59

Appendix 1: Convolution Code ........................................................................ 61
### LIST OF ABBREVIATIONS

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Full Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>AI</td>
<td>Artificial Intelligence</td>
</tr>
<tr>
<td>ASIC</td>
<td>Application Specific Integrated Circuit</td>
</tr>
<tr>
<td>FPGA</td>
<td>Field Programmable Gate Array</td>
</tr>
<tr>
<td>FPS</td>
<td>Frames per second</td>
</tr>
<tr>
<td>FPU</td>
<td>Floating Point Unit</td>
</tr>
<tr>
<td>GPGPU</td>
<td>General Purpose computing on GPU</td>
</tr>
<tr>
<td>GPU</td>
<td>Graphics Processing Unit</td>
</tr>
<tr>
<td>IR</td>
<td>Infra-Red</td>
</tr>
<tr>
<td>SIMD</td>
<td>Single Instruction Multiple Data</td>
</tr>
<tr>
<td>SoC</td>
<td>System on a Chip</td>
</tr>
<tr>
<td>SSE</td>
<td>Streaming SIMD Extensions</td>
</tr>
<tr>
<td>TBD</td>
<td>Track Before Detect</td>
</tr>
</tbody>
</table>
## LIST OF FIGURES

<table>
<thead>
<tr>
<th>Page</th>
<th>Figure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>17</td>
<td>2.1</td>
<td>Detect Track Classify Chain</td>
</tr>
<tr>
<td>18</td>
<td>2.2</td>
<td>Bayesian Object Tracking</td>
</tr>
<tr>
<td>25</td>
<td>2.3</td>
<td>NVIDIA Jetson TK1 Platform</td>
</tr>
<tr>
<td>43</td>
<td>5.1</td>
<td>Processing throughput of CACTuS CP</td>
</tr>
<tr>
<td>46</td>
<td>5.2</td>
<td>Processing throughput of CACTuS CUDA</td>
</tr>
<tr>
<td>47</td>
<td>5.3</td>
<td>Processing throughput of CACTuS CUDA on Jetson</td>
</tr>
<tr>
<td>49</td>
<td>5.4</td>
<td>Processing throughput of CACTuS Implementations on Jetson</td>
</tr>
<tr>
<td>51</td>
<td>5.5</td>
<td>Processing throughput of CACTuS Hybrid on Jetson</td>
</tr>
<tr>
<td>55</td>
<td>5.6</td>
<td>Power efficiency of CACTuS on CPUs</td>
</tr>
<tr>
<td>56</td>
<td>5.6</td>
<td>Power efficiency of Accelerated CACTuS</td>
</tr>
</tbody>
</table>
1. INTRODUCTION

Object tracking, which is otherwise known as video tracking or visual tracking, is a sub domain of computer vision. Object tracking is concerned with the process of detecting an object (known as the target) in a scene, relative to the background, and establishing its position in subsequent frames of a video sequence (called a path).

Tracking is used in a wide range of everyday applications, including business intelligence, security, military, robotics and human computer interaction (HCI). Many approaches to tracking objects have been developed, each with specific benefits. The most pervasive trackers are designed for specific tasks, using prior knowledge of the type(s) of objects that are to be tracked. Other systems use a more generic approach, seeking to track all targets, often creating a model of the tracked object after detection. This generic approach, although more computationally complex, allows for unknown and unexpected objects to be tracked.

Traditional object tracking algorithms adhere to the Detect Track Classify paradigm; the source data (usually a single frame of video) is passed to an object detector, the output from the detector is then passed to a tracker, which generates the path. Following from the tracker, the data is optionally passed to a classifier, which assigns a type label to the object. The detect before track approach of the Detect Track Classify chain has notable weaknesses that limit its application for real world tasks.

An algorithm using a detect before track approach must have an object detection method that accurately discriminates foreground objects (or targets) from the background. This is difficult in all but the simplest of cases. Applying a fixed threshold to a scene is often not enough to distinguish noise from targets, especially in source data where noise is of the same magnitude as the target signal [1]. Real world visual, infrared and thermal sensors produce images that have a high level of noise, which forces a compromise. Detect before track systems must suffer either a decrease in sensitivity or an increase in the number of false detections. As a result, those incorrect detections damage robustness -a measure of how accurately the tracker follows the path of a range of objects.
For many types of targets, the appearance of the object can change over time. For example, an object entering a scene may appear smaller than a single pixel, growing to a larger portion of the scene over time [2]. Other complications arise when tracking in the presence of clutter. Occluding clutter obscures a sensor's view of the target, preventing accurate detection. Distracting clutter can be incorrectly detected as a target, diverting processing resources and potentially preventing a real target from being tracked.

The Competitive Attentional Correlation Tracker using Shape (CACTuS) is designed to overcome many weaknesses of classical trackers. CACTuS is based on CAT, a recursive Bayesian tracker, and uses a probabilistic approach to model the states of targets. This approach allows uncertainty to be propagated through the tracking chain [3]. CACTuS employs a Track Before Detect (TBD) methodology, tracking potential targets prior to detection, enabling data from multiple frames to accumulate to define a target. Further to these attributes, CACTuS is designed to track targets that change appearance over time. CACTuS uses a silhouette estimation to represent the shape of the object. This silhouette is learnt probabilistically and can change over time, as the appearance of the target changes [3]. CACTuS also models the rate of change of shape for each tracked target. These additions create a tracker with a high level of performance, especially in the presence of occluding clutter [4].

The CACTuS algorithm uses convolution operations with large filter sizes, and as a result is very computationally demanding. In previous work, this algorithm has been implemented using Field Programmable Gate Array (FPGA) devices to provide a real time solution [5]. With further work, it is possible for the FPGA accelerated solution to be integrated into an embedded system.

The use of FPGAs in embedded tracking is common as FPGAs are able to meet the processing throughput and power requirements of embedded systems. However, the costs involved with the development and production of an embedded solution using FPGAs are considerably higher than those of a system developed for System on a Chip (SoC) platforms. Furthermore, FPGA development for embedded systems often requires custom hardware designed to meet the demands of mobile environments.
General purpose computing on GPUs (GPGPU) has grown to become a mature, flexible and cost effective parallel computing alternative to costly FPGA and Application Specific Integrated Circuit (ASIC) development. Modern GPUs, with thousands of processors, are able to very effectively accelerate parallelisable problems. Computer vision is one such field where the benefit of parallel computing using GPGPU approaches is well documented.

In this research, the suitability of a SoC based platform for the implementation of a complex multi object tracker (CACTuS) has been evaluated. Utilising a SoC platform with multi-core CPU and embedded GPU, allows for an implementation that is more flexible at runtime than an FPGA or ASIC based solution. As a demonstration, a complete multi object tracking solution was implemented on a SoC based embedded platform. The embedded system was tested for power efficiency and found to outperform a desktop system significantly.

The tracking performance, processing throughput and power efficiency of the embedded solution, along with its dynamic configurability, shows that an embedded SoC with GPU approach is well suited for high complexity Bayesian multi object visual tracking.

This research produces a set of new artefacts for object tracking:

- A complex Bayesian multi-object tracker deployed on a low cost SoC platform.
- A demonstrable multi-object tracking solution with a throughput per watt ratio of 185-520% of comparable desktop implementations.
- A cross platform CACTuS implementation with GPU acceleration through CUDA.
1.1. The Problem

In the majority of existing embedded multi object tracking systems for real time video, Field Programmable Gate Array (FPGA), Digital Signal Processor (DSP) and Application Specific Integrated Circuit (ASIC) devices are used [6-9]. Of these the FPGA approach is currently the most prolific, with several embedded devices and camera platforms with integrated FPGA discussed in previous research [7-10].

The FPGA solutions have several advantages over a standard CPU based software approach, such as higher integer and bit-level operation performance. FPGAs also offer better power efficiency in many situations and can be designed with deterministic latency [11]. However, the FPGA approach is not without drawbacks; FPGAs usually require fixed point implementations for best performance [9]. The development of FPGA based systems is more complex and results in lower productivity than other development methods. Furthermore, FPGAs are most often programmed using a Hardware Design Language (HDL), such as Verilog or VHDL, a class of language a majority of software engineers are unlikely to be familiar with [11].

High cost is a further factor against the development of FPGA based embedded vision systems, with development boards for FPGAs costing thousands of dollars to over $10000 [11]. There is also corresponding cost for the software development of a FPGA based system, with additional cost and risk associated with developing the customised hardware necessary for a production ready product.

Other approaches to the real time tracking problem for embedded systems include the use of other types of hardware accelerators, such as ASICs or DSPs, designed to improve the performance of specific computer vision tasks. While these approaches are effective [6], their implementations are largely set at design time, decreasing flexibility and often leading to a sub optimal approach to processing [12]. In addition, ASIC development is more costly and involves more risk than FPGA, with the circuit set at design time.
In this research, a low cost, off the shelf SoC is used as a base. Such a system is suitable for embedding directly into a final product, thereby reducing overall development costs and time to market. The NVIDIA Jetson TK1 development platform with quad core SoC and a GPGPU capable embedded GPU has been selected as representative sample of current embedded system functionality.
1.2. Research Questions

This research focuses on evaluating the performance of a heterogeneous implementation of the CACTuS multi object tracking algorithm on a SoC based embedded platform. This research shows that SoCs with GPUs are a viable alternative for the creation of embedded multi object tracking systems. In the course of this research, the following questions were answered:

**RQ1.** Can a complex Bayesian multi-object tracker (CACTuS) be implemented on an embedded SoC platform?

**RQ2.** How is the tracking performance impacted by SoC platform computational restrictions?

**RQ3.** Can a CUDA based GPGPU solution be implemented that improves tracking performance and computational efficiency?

**RQ4.** Is the tracking algorithm realisation on the SoC platform able meet real time performance requirements?

**RQ5.** Does the SoC based solution provide better power efficiency than existing solutions?
2. LITERATURE REVIEW

This chapter focuses on the current state of, and evolution of, object tracking algorithms as a derivative of computer vision development. In section 2.3, GPGPU computing is discussed, along with its relevance and advantage to computer vision algorithms. Section 2.4 presents SoC based embedded systems and their current state of development. In section 2.5, conclusions are made, based on the presented literature, which direct the focus of this thesis.

2.1. Computer Vision

Computer vision as a field aims to build a model of a scene from an image or multiple images, describing shape, illumination and colour distributions. A model built from a visual scene enables a computer to understand the world around it. Early artificial intelligence researchers underestimated the difficulty of the "visual input" problem, believing it to be a simple step in the development of artificial intelligence systems with perceptual and cognitive intelligence [13]. While animals and humans effortlessly achieve understanding of a visual scene or image, computer vision algorithms are very error prone. Computer vision continues to be a difficult field, in part because we seek to recover multiple unknowns, given insufficient information [13]. Noise, clutter and inaccurate measurement also add to the complexity of solving computer vision problems.
2.1.1. Early Computer Vision

Since the 1960s, researchers have been developing methods for enabling computers to interpret the world through vision. Computer vision started as the visual perception component of an ambitious artificial intelligence (AI) project with the goal of replicating human intelligence. In the 1960s, researchers attached cameras to computers and started on the path of developing computer vision systems. An early computer vision project, 'The Summer Vision Project' from MIT in 1966, called for workers over the summer break to create 'a significant portion of a visual system' [14]. This project was proposed during the overly optimistic age of AI and the complexities involved with replicating a biological vision system were completely underestimated. Computer vision was considered a small step on the path to developing computers that mimic human intelligence and understand their environments [13]. Almost 50 years have passed and we are still developing methods of solving the goals of that MIT proposal, a project which initially had an estimated timeline of a month.
2.1.2. Practical Computer Vision

Despite the difficulties associated with computer vision, there are many successful practical applications.

Optical Character Recognition (OCR) is a heavily used technology that stems from advances in computer vision. This technology is used in postal systems, recognising the postal codes of typed and hand addressed mail [13]. Automatic licence plate recognition systems utilise OCR and are increasingly being used to log traffic into and out of premises, simplifying the process of identifying persons of interest following an incident.

Motion capture is a highly researched and specialised field of computer vision that achieves very good results by significantly reducing the problem domain. By utilising special markers, infrared illumination and specialised cameras, the impact of noise and inaccuracy of measurement are able to be significantly minimised [15]. These systems can vary accurately track the position and orientation of objects and people.

Video Surveillance is an industry that has successfully applied computer vision techniques (known in the security industry as video analytics) to provide a level of automated scene understanding. By simplifying the vision problem to specific target domains, automated surveillance systems utilise video analytics to monitor and report on live surveillance feeds [16]. These systems are able to detect events such as removed and left behind objects, area violations, wrong way traffic and loitering. Pan tilt zoom cameras with embedded tracking are able to autonomously follow a moving target.
2.2. Object Tracking

Object tracking is a sub-field of computer vision concerned with tracing the path of targets through a scene. A tracking algorithm is designed to detect and track one or more objects in a video sequence. The classic tracking chain can be represented diagrammatically as shown in Figure 2.1. In the chain, the Detect stage is concerned with separating objects (or targets) from the background. The track stage follows the target over the sequence to generate a path. Optionally, a classify stage can be used to identify the nature of the target [2].

![Figure 2.1: The Detect-Track-Classify representation of object tracking.](image)

The CACTuS tracking algorithm studied in this research uses a recursive Bayesian and probabilistic Track Before Detect (TBD) approach to implementing the track and detect phases.
2.2.1. Bayesian Techniques

Bayes theorem explains how ‘the relationship between causes and observations can be inverted to infer causes from observations’ [1], p892. In the Bayesian approach, the posterior belief at time $t-1$ is used to predict the prior belief at $t$, based on a model. The measurement taken at time $t$ is then combined with the prior to calculate the posterior belief at time $t$. The posterior belief in an object tracking scenario is usually a probability density function that represents the object location.

\[
bel(x_t) = \eta \, p(z_t|x_t) \int p(x_t|x_{t-1}, u_t) \, bel(x_{t-1}) \, dx_{t-1}
\]

**Figure 2.2:** Bayesian object tracking from [17].

The Bayesian technique applied to an object tracking problem is most often implemented using probability distributions to model states. This enables the system to propagate uncertainty \[3\] and reduces the effect of noise and ambiguity on the accuracy of track paths. By propagating target probabilities, a track before detect system can be created, allowing the tracker to correlate paths before targets are accurately modelled. This approach allows probabilistic trackers to track signals of the order of magnitude of the noise threshold of the sensor system, which would not be possible in a deterministic detect before track system [1].
2.2.1. Multi Object Tracking

Tracking multiple targets significantly increases the difficulty of the tracking problem. A system that needs to track multiple objects, must be able to identify each detected object in a scene as either a particular existing target, or a new target. This is complicated in scenes where the targets are occluded for several frames, more so when targets occlude each other. Multi object tracking requires that instantaneous measurements are associated with target models. In a naive Bayesian approach, this would require at least an additional estimation per pixel, creating an intractable problem [1].

2.2.2. The CACTuS Tracker

The Competitive Attentional Correlation Tracker using Shape (CACTuS) has been selected as the target for this study. CACTuS was developed at the University of South Australia by Sebastien Wong, it provides a solution to the multi object tracking problem. The CACTuS algorithm is an extension of the work by Strens and Gregory on their Competitive Attentional Tracker (CAT) [1]. CACTuS refines CAT by adding a shape silhouette to the object model, allowing the tracker to more accurately correlate objects in clutter and track a target as its shape changes.

In CACTuS, multiple single object trackers are run simultaneously, each constrained to their own portion of the image by a competitive mechanism [3]. These single object trackers, known as Shape Estimating Filters (SEFs), are each capable of describing the position, velocity and shape of a single target.

A comparison of CACTuS against other trackers shows how CACTuS’ shape tracking improves the tracking performance [4]. In the testing, CACTuS significantly outperformed other tested trackers in the presence of occluding clutter.
2.2.3. Acceleration Through Parallel Computation

Many of the low level tasks in computer vision are embarrassingly parallel and are easily accelerated using parallel processing techniques. In many vision processing tasks operations are applied to each individual pixel, or groups of pixels, independently of others. Multi core CPUs offer an advantage to these operations and by utilising multi core CPUs, these low level vision processing operations gain benefit from each additional core.

Due to their parallelism, FPGAs and GPUs are well suited to accelerating the application of these algorithms. Many object tracking algorithms, including CACTuS, heavily rely on two key operations, correlation and convolution. The operations are shift-invariant and linear, meaning the same operations are applied to each output pixel and those operations are simple linear combinations. These properties allow for simple implementations on parallel hardware, providing a significant, near linear, benefit for each available processing unit.

Previous examples of object tracking acceleration using parallel architectures include the use of FPGAs [5, 9, 12]. The use of desktop GPUs for acceleration of specific computer vision tasks, including convolution and FFT has also been well researched [18].
2.3. GPGPU Computing

Modern GPUs take the multi core concept to an extreme. GPUs are built for accelerating graphics rendering and are designed to process shader programs on pixels and vertices, these operations happen for each pixel and vertex in a 3D scene. The pixel and vertex shading operations are most often independent and greatly benefit from a parallel implementation. As such GPUs are designed with a massively parallel architecture, with current devices containing thousands of processors.

GPU performance progressed in the late 1990s and early 2000s, funded by the booming computer gaming industry. These efforts were focused on improving the visual quality of 3D rendered scenes. As GPUs and gaming progressed, the fixed graphics pipeline quickly became a bottleneck to the improvement of scene rendering. The introduction of programmable shader processors alleviated that problem and greatly increased the flexibility of graphics rendering. The concept of the pixel and vertex shaders quickly shaped GPU technology. GPUs were being built with tens of pixel shaders and several vertex shaders, each capable of running a targeted shader program. Researchers in parallel computation recognised the potential benefit to repurposing the GPU for general purpose computing [19].

Early GPGPU attempts include BrookGPU, a C like language that can be run on OpenGL or DirectX compatible devices [20]. Prior to this, developers wanting to take advantage of the parallel nature of the GPU were required to utilise graphics targeted APIs [18]. BrookGPU still relies on using graphics APIs to perform the calculations, but this detail is abstracted away from the application programmer. Despite BrookGPU’s abstraction from graphics APIs, developers were still required to be mindful of the limitations present in those APIs, especially the limits imposed on memory usage [21].

In 2006, NVIDIA introduced a unified processor architecture in their G80 GPU, the first processors designed to run both vertex and shader programs. In this new architecture each processor’s allocation to vertex or shader programs is dynamic during runtime. The G80 GPUs were built with up to 128 of these unified processors, in place of separate processing units dedicated to vertex or pixel processing. Following from this, in 2007, NVIDIA released the CUDA platform, which offered a way to perform general purpose
computing directly on an NVIDIA GPU. CUDA was the first commercially available platform for performing general purpose computing on a GPU. CUDA differed significantly from previous attempts at GPGPU processing as CUDA provides a method for directly programming general purpose computing tasks for execution on a GPU. With CUDA, NVIDIA provides a specialised driver (and driver mode) that accesses the hardware directly, bypassing the graphics APIs, increasing efficiency and capability. The CUDA programming language is a domain specific version of C, enabling easy transition of algorithms to the platform [21].

Significant progress in GPGPU computing has occurred over the past decade, with many problems re-engineered to take advantage of GPU platforms [22].

2.3.1. Limitations of GPU Computing

Although GPUs allow for massively parallel operations, there are limitations on the performance that can be gained by utilising GPU architectures. Memory bandwidth and transfers between host (CPU) and device (GPU) memory are the most significant bottleneck to GPU computing. To gain an advantage using the GPU, the problem must have a high computation to memory access ratio or arithmetic intensity [20]. This is where the time penalty involved with the transfer of memory to and from the device is outweighed by the speed with which the device is able to perform the computation.
2.4. Embedded Systems

Embedded systems are all around us, often hidden in everyday objects. Cars, TVs, entertainment systems, media players and mobile phones are all examples of embedded systems. Designed to perform specific tasks, they have specific advantages over standard computing platforms. In most cases, they are designed to be small and energy efficient. Embedded systems are usually based on RISC architectures, such as ARM, as these architectures enable smaller, more energy efficient cores [23].

Embedded systems have gained greatly from the introduction and popularity of the smart phone. As competitors in the market have fought to gain market share, by delivering consumers better products with more features and better performance. The corresponding increase in capability, energy efficiency and processing power have followed. These advances led to the creation of the system on a chip (SoC), as its name implies a complete system integrated into a single chip. A single chip solution is of great benefit for low power and mobile embedded computing as transfer of data off-chip requires significantly more power than internal data transfer [24]. The on-chip integration of the main CPU and peripheral controllers allows for a performant and highly efficient system.

In recent years, SoC systems have seen dramatic increases in GPU capabilities, including the release of embedded SoC platforms with GPUs that enable GPGPU computing [22]. The obvious advantage of the inclusion of these GPUs is that SoCs are now able to utilise mature GPGPU algorithms to increase processing performance and in many cases increase energy efficiency.

Smart Cameras are an example of an embedded computer vision system which includes a complete image sensor and processor in one package [10]. Smart Cameras most often utilise FPGAs, ASICs or other specialised processors for low level computer vision processing. The low level processing tasks generally perform the same operation over each pixel in the entire image and are inherently parallel and thus gain a large benefit from highly parallel architectures.
2.4.1. Object Tracking on Embedded Systems

Object tracking can be described as one of the most computationally intensive tasks more so than most other computer vision processes [12]. As such, embedded systems require special techniques, different from those suitable on desktop platforms, to achieve real time processing for computer vision applications [10].

Current systems follow one of two distinct approaches to achieving real time object tracking on embedded platforms. The first approach is the most obvious: reduce the complexity of the tracker to achieve the desired performance on the platform. While this approach is clearly effective [25], several drawbacks to this approach are apparent. Reducing complexity has the clear problem of reducing the performance of the tracker. This performance problem can be offset by defining the problem more strictly, reducing the application domain, and utilising predetermined object models. This creates a tracker that is capable of tracking only certain kinds of objects, with restrictions on the type of scene. There are tracking systems created by those that define a lower level of performance as meeting the real time aspect, with several systems meeting a performance of 10-15 frames per second (fps) of processing [6, 7, 26]. In many scenarios, this lower rate of processing is acceptable. However, in many scenarios, such as those involving fast moving objects, that rate of processing is inadequate. In this thesis, a real time processing target of 20fps has been set, to enable tracking of faster moving objects.

The second commonly used approach is to improve the processing capability of the embedded system, utilising additional processing hardware. Embedded hardware accelerators may be in the form of ASIC, FPGA or DSP [8], with the specific implementation tailored to the required computational tasks. In this approach, the accelerator is able to provide a high level of performance for specific portions of the tracking algorithm, augmenting the overall performance of the system. FPGAs are well suited to this and are the most commonly implemented, they are both computationally powerful and flexible. FPGAs provide benefit to computations that are easily described as a series of parallel operations or as a pipeline of operations, such as those required for low level computer vision processing. That functionality becomes embedded in the extra hardware, demanding far less processing from the CPU.
2.4.2. NVIDIA Jetson TK1 Platform

This research aims to show modern embedded SoCs with GPGPU capability are a viable alternative to FPGA and ASIC base development. As such, the NVIDIA Jetson TK1 has been selected as the target platform for this research. The Jetson platform was selected for the following reasons:

1. The Jetson contains a high performance quad core CPU.
2. The Jetson contains an integrated Kepler architecture GPU.
3. The platform targets a 10W power envelope.
4. The platform runs Linux.

![Figure 2.3: The NVIDIA Jetson TK1 Development Platform.](image)

The Jetson TK1 has a NVIDIA Tegra K1 CPU, a 4-Plus-1™ Quad Core ARM A15 device with a maximum clock speed of 2.3GHz and an embedded Kepler GPU with 192 CUDA cores, 2GB DDR3L-1866 RAM, USB 3.0, Gigabit Ethernet and a 16GB SanDisk eMMC [27]. The Jetson TK1 platform is targeted for low power and as such is suitable for a range of mobile applications such as automotive, marine or aviation. The CUDA capable Kepler GPU provides an efficient means of parallel computation.

The Jetson TK1 runs Linux For Tegra, an Ubuntu derivative and has access to the ARM packages maintained by Canonical and the Ubuntu community. The use of a well-supported, open source operating system has advantages for development; tools such as gcc and make are readily available to support compilation directly on the platform. Common libraries are also available pre-compiled for the ARM targeted, Ubuntu based, Linux For Tegra. In addition, NVIDIA provides a Jetson targeted and optimised version of OpenCV, the open source computer vision library.
2.5. Conclusion

Current research in embedded object tracking has not pursued the merits of SoCs incorporating GPUs. Previous embedded tracking efforts have been focussed on the more expensive and difficult to develop FPGAs. Current research points to significant performance improvement with GPGPU approaches to tracking problems. There is little research demonstrating the benefits of utilising the GPU of these newer mainstream SoC systems. The increasing GPGPU capabilities of current SoC platforms offer an obvious benefit to computer vision applications. The research presented in this thesis shows that current embedded GPUs enable the creation of viable embedded object tracking systems.
3. RESEARCH METHOD

This work focuses on evaluating the feasibility and performance of a heterogeneous computing realization of the CACTuS tracking algorithm on a system on a chip (SoC) based platform. When developing a methodology, we categorize the research question into one of the five question types detailed by [28] and shown in table 3.1.

<table>
<thead>
<tr>
<th>Type of Question</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Method or means of development</td>
<td>How can we do/create X? What is a better way to do/create X?</td>
</tr>
<tr>
<td>Method for analysis</td>
<td>How can I evaluate the quality/correctness of X? How do I choose between X and Y?</td>
</tr>
<tr>
<td>Design, evaluation or analysis of a particular instance</td>
<td>What is a (better) design of implementation for application X? What is property X of artefact/method Y? How does X compare to Y? What is the current state of X/practice of Y?</td>
</tr>
<tr>
<td>Generalisation or characterisation</td>
<td>Given X, what will Y (necessarily) be? What, exactly, do we mean by X? What are the important characteristics of X? What is a good formal/empirical model for X? What are the varieties of X, how are they related?</td>
</tr>
<tr>
<td>Feasibility</td>
<td>Does X even exist, and if so what is it like? Is it even possible to accomplish X at all?</td>
</tr>
</tbody>
</table>

Table 3.1: Research questions in software engineering from [28]

Following the formalisation of a question, a result must be produced. Results are the specific outputs of research. Table 3.2 details common types of research results.
<table>
<thead>
<tr>
<th>Type of Result</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Procedure or technique</td>
<td>New or better way to do some task, such as design, implementation, measurement, evaluation, selection from alternatives. Includes operational techniques for implementation, representation, management, and analysis, but not advice or guidelines.</td>
</tr>
<tr>
<td>Qualitative or descriptive model</td>
<td>Structure or taxonomy for a problem area; architectural style, framework, or design pattern; non-formal domain analysis. Well-grounded checklists, well-argued informal generalizations, guidance for integrating other results.</td>
</tr>
<tr>
<td>Empirical model</td>
<td>Empirical predictive model based on observed data</td>
</tr>
<tr>
<td>Analytic model</td>
<td>Structural model precise enough to support formal analysis or automatic manipulation.</td>
</tr>
<tr>
<td>Notation or tool</td>
<td>Formal language to support technique or model (should have a calculus, semantics, or other basis for computing or inference). Implemented tool that embodies a technique.</td>
</tr>
<tr>
<td>Specific solution</td>
<td>Solution to application problem that shows use of software engineering principles – may be design, rather than implementation. Careful analysis of a system or its development. Running system that embodies a result; it may be the carrier of the result, or its implementation may illustrate a principle that can be applied elsewhere.</td>
</tr>
<tr>
<td>Answer or judgement</td>
<td>Result of a specific analysis, evaluation, or comparison.</td>
</tr>
<tr>
<td>Report</td>
<td>Interesting observations, rules of thumb.</td>
</tr>
</tbody>
</table>

**Table 3.2: Research results in software engineering from [28]**

For a research result to be considered useful, the result must be validated. The technique selected for validation of a research result must give ‘clear and convincing evidence that the result is sound’ [28]. Common types of result validation techniques are given in Table 3.3.
<table>
<thead>
<tr>
<th>Type of Validation</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analysis</td>
<td>I have analysed my result and find it satisfactory through...(formal analysis) ... rigorous derivation and proof (empirical model) ... data on controlled use(controlled carefully designed statistical experiment) experiment</td>
</tr>
<tr>
<td>Experience</td>
<td>My result has been used on real examples by someone other than me, and the evidence of its correctness / usefulness / effectiveness is ...(qualitative model) ... narrative(empirical model, ... data, usually statistical, on practice (notation, tool) ... comparison of this with similar results in (technique) actual use</td>
</tr>
<tr>
<td>Example</td>
<td>Here’s an example of how it works on (toy example) ... a toy example, perhaps motivated by reality (slice of life) ... a system that I have been developing</td>
</tr>
<tr>
<td>Evaluation</td>
<td>Given the stated criteria, my result... (descriptive model) ... adequately describes the phenomena of interest ... (qualitative model) ... accounts for the phenomena of interest... (empirical model) ... is able to predict ... because ..., or ... gives results that fit real data ... Includes feasibility studies, pilot projects</td>
</tr>
<tr>
<td>Persuasion</td>
<td>I thought hard about this, and I believe... (technique) ... if you do it the following way, ... (system) ... a system constructed like this would ... (model) ... this model seems reasonable Note that if the original question was about feasibility, a working system, even without analysis, can be persuasive</td>
</tr>
<tr>
<td>Blatant assertion</td>
<td>No serious attempt to evaluate result</td>
</tr>
</tbody>
</table>

Table 3.3: Validation techniques in software engineering from [28]

Following from these definitions, the questions, methods and validation strategies for this research can be formalised.
3.1. Formalised Research Questions

In this research, a set of questions is formalised to enable the evaluation of GPGPU capable SoCs for embedded multi object tracking. The research questions:

**RQ1.** Can a complex Bayesian multi-object tracker (CACTuS) be implemented on an embedded SoC platform?

**RQ2.** How is the tracking performance impacted by SoC platform computational restrictions?

**RQ3.** Can a CUDA based GPGPU solution be implemented that improves tracking performance and computational efficiency?

**RQ4.** Is the tracking algorithm realisation on the SoC platform able meet real time performance requirements?

**RQ5.** Does the SoC based solution provide better power efficiency than existing solutions?

In remainder of this chapter we develop the methodology to answer these questions and validate the results.

3.2. Methodology

By categorising the research questions, strategies can be formulated for the generation of results. The following table denotes the types of question, result type and validation technique for each research question.

<table>
<thead>
<tr>
<th>Question</th>
<th>Type</th>
<th>Result</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>RQ1</td>
<td>Feasibility</td>
<td>Specific solution</td>
<td>Example</td>
</tr>
<tr>
<td>RQ2</td>
<td>Evaluation of a particular instance</td>
<td>Answer or judgement</td>
<td>Experience</td>
</tr>
<tr>
<td>RQ3</td>
<td>Design and evaluation of a particular instance</td>
<td>Procedure or technique</td>
<td>Example</td>
</tr>
<tr>
<td>RQ4</td>
<td>Evaluation of a particular instance</td>
<td>Answer or judgement</td>
<td>Evaluation</td>
</tr>
<tr>
<td>RQ5</td>
<td>Evaluation of a particular instance</td>
<td>Answer or judgement</td>
<td>Experience</td>
</tr>
</tbody>
</table>

*Table 3.4: Research question methodology classification.*
3.2.1. RQ1 CACTuS on an embedded SoC platform

As discussed in the previous chapter, most existing multi object tracking solutions for embedded systems target FPGAs. SoC based systems are more limited in computing resources than desktop systems. As such, the aim is develop techniques that allow CACTuS to run on the embedded platform. The result will be a complete embodied system, a SoC based system with a functional implementation of CACTuS. As embedded development can be challenging, a desktop implementation will be created first as an aid and a benchmark. The desktop version can provide a solution on a known platform ensuring correct implementations are created for the SoC platform.

3.2.2. RQ2 The impact of SoC constraints on tracking performance

The computing constraints and specific hardware implementation of SoC devices present possible sources of error for probabilistic tracking. To ensure an acceptable level of tracking performance, an accepted set of tracking metrics is used for evaluation. The tracking output from both the desktop and SoC implementations can be run through a MATLAB based tracking metrics evaluation framework. The use of a single evaluation framework enables direct comparison of tracking performance metrics between platform specific implementations.

3.2.3. RQ3 Does CUDA improve the tracking implementation

It is widely accepted that GPGPU techniques improve the computational efficiency of many parallelisable computing algorithms. CACTuS has been implemented utilising OpenCL for GPGPU previously. It is not known is whether CUDA can provide a performance benefit over OpenCL on a desktop platform, nor is it known if utilising the GPU of the SoC will provide a performance benefit. This will require the development of a technique for implementing specific portions of the CACTuS algorithm on the GPU. Further to the technique and implementation, the impact on processing and tracking performance must be evaluated in order to determine the impact of CUDA on the tracking system.
3.2.4. RQ4 Does the SoC meet real time constraints

A significant aim of this research is to achieve real time performance of CACTuS on the SoC platform. The system must be able to process in real time for use in real world applications. As stated previously, the aim is to meet or exceed a processing throughput of 20 frames per second (fps). The implemented system’s throughput will be characterised by timing the processing of representative data sets.

3.2.5. RQ5 Power efficiency of new implementations

Most embedded systems are constrained by the power availability and dissipation capacity of their environment. For this reason, it is necessary to characterise power consumption

3.3. Conclusion

This chapter has explored the development of a methodology to evaluate the proposed research questions. The methods and scientific principles that apply to this research have been explained. With this knowledge it is now possible to create run the experiment and verify the results.
4. EXISTING CACTuS IMPLEMENTATIONS

In this section, existing CACTuS implementations are reviewed for their applicability for the evaluation of multi object tracking on the NVIDIA Jetson TK1 platform. At the commencement of this research, three implementations of the CACTuS algorithm existed; one in MATLAB, and two in C++. The MATLAB implementation is based on the original algorithm design. The first C++ implementation, CACTuS Armadillo, was designed to closely match the MATLAB code. The second C++ version, CACTuS Vienna, is based on CACTuS Armadillo, targeted at OpenCL acceleration using ViennaCL.

4.1. The MATLAB implementation

The implementation of CACTuS in MATLAB is a ‘gold standard’ version. This means the MATLAB version implements the CACTuS algorithm in a standard manner to which all other versions are compared. MATLAB is a language very suitable for development of algorithms, especially those which contain a large amount of matrix manipulation. It provides a vast library of matrix operations, and expansion packages that increase those, in a well optimised, easy to use system. This allows developers to focus on the algorithm, rather than issues with implementation. Although there are versions of MATLAB for Linux based systems, currently there exists no MATLAB for an ARM based Linux environment. This prevents the direct implementation of MATLAB CACTuS on the Jetson platform.

4.2. CACTuS Armadillo Implementation

CACTuS Armadillo was designed with the goal of matching the MATLAB code and designed to run on a desktop CPU. It suffers from several problems when viewed as the starting point for an embedded SoC system implementation.

CACTuS Armadillo has significant library dependencies, including the Armadillo math library, Boost C++ library, Qt, FFTW and several support libraries. The Armadillo library was selected for the development of CACTuS Armadillo for several reasons, most prominently that the library was designed with the goal of quickly converting MATLAB code to production environments [29]. This simplified the conversion process as the API for Armadillo is deliberately similar to MATLAB.
The Armadillo library has notable faults, especially in this instance where the system is to be used in demanding scenarios with low processing resources and real time performance constraints. The Armadillo library is a template library in a template metaprogramming style [29]. As it is heavily templated, it is very difficult to accurately debug issues that arise during the compilation process. The use of Armadillo version 3.2, on the development system, resulted in a compiled program with a high number of temporary matrix objects. For each temporary object, at minimum one additional array copy was required. The cumulative result of the temporary object and copies is detrimental to processing performance and memory usage. There are additional effects of template programming that lead to less than optimal performance, especially for matrix operations [30]. The nature of templates also means that errors in the functions that utilise library templates can result in compiler errors, which are very difficult to find and debug. This type of error was encountered when attempting to update the Armadillo library to a later version.

### 4.3. CACTuS Vienna Implementation

CACTuS Vienna, an OpenCL GPU targeted variant of CACTuS Armadillo, utilises version 1.3.1 of the ViennaCL library. During the conversion from Armadillo to ViennaCL, the default matrix representation for ViennaCL was utilised. By default, the ViennaCL library uses row major ordering of matrices, whereas MATLAB and Armadillo both use column major ordering. The creators of the GPU version implemented several OpenCL kernels for specific functions, which are beneficial for performance. However, an attempt to update the ViennaCL to version 1.4.1 showed that several operations relied on functions not available in the newer version of ViennaCL.
4.4. Heterogeneous CACTuS Armadillo and Vienna

In order to create a heterogeneous C++ version of CACTuS, an evaluation of merging the Armadillo and Vienna variants was considered. As the Vienna variant was derived from the Armadillo implementation, it was, wrongly, assumed that it would be an easy task to merge the two versions. The issues that prevent such a combined version are discussed here.

Each version uses the default matrix representation of their respective math library. The choice to follow each libraries standard matrix representation is one that causes several issues when attempting to create a heterogeneous implementation. In order for operations to be efficiently computed across CPU and GPU, the matrix representations must match. The overhead involved with the necessary array manipulation outweighs the performance benefit gained from GPU implementation. Additionally the row major ordering from the Vienna variant makes for inefficient interoperation with MATLAB based systems. This interoperation is likely to be desirable in the ongoing development of CACTuS.

CACTuS Vienna uses custom OpenCL kernels which, due to the use of Row major ordering, are not suitable for a combined CPU/GPU implementation.

CACTuS Vienna uses pre-processor directives heavily. While the use of pre-processor directives is standard practice for C/C++ development, especially for platform specific compatibility, the way they are used in the existing code creates a significant problem for the creation of a heterogeneous version. The pre-processor directives are used to select the target platform of the code, between CPU and GPU (in the Vienna variant). Each version uses the native matrix type of the specific maths library and no provision was been made to allow for these platforms to be combined in a single executable. Being tightly coupled with each specific library placed a significant disadvantage on using this code as a basis for a heterogeneous solution.
4.5. Recommendations

The above observations lead to a conclusion that it is beneficial for the CACTuS algorithm to undergo a new conversion from MATLAB to C/C++. This conversion should focus on reducing the number of dependent libraries, creating a matrix class designed to be used for both CPU and GPU operations and simplifying the ability to add platform targeted code for performance. To improve interoperation and allow for system testing, the new version’s matrix storage format and matrix operations should closely match MATLAB’s standards.

Further to these recommendations, and the associated complications, the existing C++ implementations of CACTuS were not considered suitable as the basis for development of the SoC based implementation.
5. SYSTEM IMPLEMENTATION

The previous chapter presented the previous CPU and GPU targeted implementations of CACTuS. A number of shortcomings of those previous versions were identified that make them unsuitable for use in the evaluation of the SoC based Jetson platform. Several of the identified issues also unnecessarily reduce the processing frame rate achievable with that code.

In this chapter, new versions of CACTuS are introduced that were developed during the course of this research. These new versions included the design ideas identified during the evaluation of the previous versions. The new implementations, CACTuS CP and CACTuS CUDA are the highest performing C++ CPU and GPU targeted versions of CACTuS to date. CACTuS CP and CACTuS CUDA are also the first C++ versions compatible with both desktop and SoC platforms.

This chapter is organised as follows; in the first section the design ideas included in new high performance CACTuS implementations are presented. The next section describes the CPU targeted CACTuS Cross Platform (CP), the first implementation of CACTuS capable of running on both desktop and SoC platforms. In the next section, the re-introduction of GPU acceleration into CACTuS using CUDA is discussed.

The remainder of the chapter provides a comparative evaluation of these newly developed CACTuS implementations and compares with existing implementations. The evaluation takes a multi-faceted approach, covering the processing rate (or frame rate), the tracking performance and power efficiency of each entire system.

5.1. High Performance Strategies for CPU targeted CACTuS

The strategies taken to create a new, higher performance, implementation of CACTuS are those suggested in the previous chapter. Those strategies are; reduction in the number of external libraries used, reduction of pre-processor directives and the introduction of a single matrix class for CPU and GPU targeted code. Further performance targeted strategies for the development of a new CACTuS version include the use of OpenMP for parallelisation of the main loop and an investigation of the
applicability of SIMD instructions to improving the performance of the matrix math operations and the convolution function.

The reduction of external libraries as a method of performance enhancement is often be ignored by many. Testing has shown libraries targeted for simplified development with performance as a secondary goal are often incompatible with truly high performance computing. By creating a new, CACTuS targeted matrix class, we remove the need to use Armadillo and ViennaCL libraries. Our own uses an optimised memory storage strategy that is compatible with both CPU and GPU targeted functions, which ensures an aligned memory allocation, enabling highest performance from both SSE instructions and FFTW.

OpenMP provides a very simple method for enabling programmers to take advantage of multi core environments without needing to explicitly create worker threads. OpenMP is a suitable method of enabling CACTuS to take advantage of multi core CPUs. OpenMP hides the complexity associated with creating and running a pool of threads. In CACTuS, for loops are used to run each SEF’s operations in turn. These for loops are parallelised using OpenMP compiler directives. This is especially effective due to the highly independent nature of each SEF. As OpenMP is designed as a multi-platform solution and is available for gcc on ARM, the benefit will translate from the desktop to the Jetson implementation.

Matrix operations are often easily parallelised and will likely benefit from SIMD instructions. Convolution is a readily parallelisable operation, it could potentially benefit from SIMD instructions. To that end, SSE targeted instructions should be created to improve the performance of desktop targeted code.

Use of SIMD on the desktop was investigated and matrix functions using x86 SSE intrinsic instructions were included a desktop targeted CACTuS. This includes a SSE targeted version of the convolution function.

In practice the SSE instructions had no measurable benefit for the matrix math operations. The convolution did benefit from the use of SSE, reducing processing time by almost 7%. However, due to the limited benefit and high effort needed to port to
platform specific instructions, the use of SIMD was abandoned for the SoC targeted versions.

5.2. Development and Testing Setup

In order to develop, test and verify a SoC targeted version of CACTuS for the Jetson platform, a desktop system for development was constructed. By utilising a desktop system, a baseline implementation was able to be compared with that of the Jetson platform. This desktop was used as a test bed for development, enabling much faster development than the platform alone could provide. The system was selected to have a GPU with a level of CUDA capability that mirrors but exceeds the Jetson platform. The NVIDIA GTX 960 was selected as it met the CUDA capability requirement. By creating a development environment shared by desktop and platform, the influence of various factors, including OS and system architecture, on performance can be examined. On the desktop, a dual boot configuration was used, with Windows 7 as the primary development platform, simplifying the use of a MATLAB based test suite. The secondary OS for the desktop was Ubuntu 14.04, enabling the use of the CUDA Development kit with Jetson parity. The Jetson TK1 was loaded with the Linux For Tegra R21.4, an Ubuntu 14.04 environment with CUDA and ARM targeted extensions from NVIDIA.

Visual Studio 2013 with the NVIDIA Nsight plugin was used for development of the new CACTuS implementations. By performing the bulk of the development on Windows, the CACTuS implementation was easily analysed for tracking performance using the Windows based MATLAB tracking metrics framework. The use of an IDE, with full support for debugging and performance profiling tools greatly increased the ease and speed of development. Desktop based development brings additional advantages, including faster build times and simplified debugging.

5.3. CACTuS CP - an Intermediate Step

The first move towards a CACTuS implementation tailored for the Jetson TK1 platform was the development of an intermediate, CPU only version. This version, designed for code portability, enables direct comparison across platforms with a single code base.
CACTuS is a Bayesian tracker and as such follows a four step approach, augmented with an additional competition step. The first step is prediction, this involves four convolutions. Then next phase, observation involves two further convolutions, one of which is very large and measures the velocity of the target.

Previous CACTuS versions utilise FFTW to perform an approximation of the largest convolution (on the CPU) as this reduces the computational complexity. FFTW has been shown to be the fastest implementation of FFT for CPUs. FFTW is well documented, available precompiled for many platforms and open source, as such, its use has been retained in CACTuS CP.

The new matrix class and its operations were designed with SSE intrinsic calculations that improve the per-cycle performance on the x86 CPU. Such optimisations could potentially be converted to work with other CPU specific instruction sets. To simplify testing, debugging, and to increase the portability of the code, standard C++ implementations of each SSE based operation were created. By utilising standard C++ and open source, cross platform libraries, the code can be compiled across many platforms, simplifying future development.

This early version of CACTuS CP was compiled and verified to work correctly on both the desktop and Jetson platforms. Following validation of the tracking output of CACTuS CP, a performance review of the code was undertaken using profiling tools. The output from the performance review enabled the determination of areas of code that would benefit most from refactoring. This refactoring process focused on the elimination of buffer copies, the removal of unnecessary memory initialisation and ensuring the readability of the code. By utilising this targeted approach to refactoring, effort is matched with results.

The heavily used matrix class needs to be well optimised for CACTuS CP to work efficiently, especially on the SoC platform. The matrix class refactoring focused specifically on memory use reduction and reduction of processing.

In the matrix class, each instantiation led to memory allocation, followed by initialisation to zeros. Initialisation is generally a good practice, it places known values
in each block of allocated memory. However, in high performance situations or those with limited compute resources, initialisation should be restricted to where it is explicitly necessary. Several matrix functions, the copy constructors, arithmetic operations and matrix conversion methods create matrices only to follow on and fill each location with a specific value. By eliminating the initialisation in these operations, computation is reduced.

During the refactoring process, several new functions for the efficient operation of CACTuS were created. These functions target operations that involve multiple steps and the creation of temporary objects. As an example, in preparation for an FFT operation, the matrix undergoes a rotation, then a conversion from real floating point representation to complex. The initial implementation used a rotation operation which creates a result matrix, followed a conversion function which copies the values from that result matrix into a new complex matrix. The conversion to complex representation can take place during the rotation, utilising a new function which combines those operations, saving the allocation of a temporary matrix. This not only speeds execution but decreases memory usage.

Function parameters were optimised to reduce memory consumption and remove unnecessary computation. The removal of strings as parameters from the MATLAB style functions and the use of more informative and restrictive enumerations provide not only less memory usage but also allow compilation time validity checking.

The convolution function was originally tailored for x86 SSE, to provide high performance on that platform. x86 SSE instructions are processor family specific and thus are not available on the ARM based Jetson and would need to be converted to NEON if support was desired. As convolution is the highest used operation in CACTuS, it is important that a known good implementation is used. To that end, the code from GNU Octave was used as a base for the convolution function in CACTuS CP. The GNU Octave code provides an implementation compatible with that of MATLAB and is standard C++. (See Appendix 1, Convolution code)
5.3.1. Processing Throughput of CACTuS CP

Here the processing throughput of CACTuS CP on the desktop and Jetson platforms is compared. The tests are configured the same in all scenarios and consist of a test run using synthetic video. For these tests, 16 SEFs has been selected as a representative multi object tracker, capable of tracking 16 objects. The specifications for the Desktop test platform and the NVIDIA Jetson TK1 platform are detailed in table 5.1.

<table>
<thead>
<tr>
<th>System</th>
<th>CPU</th>
<th>RAM</th>
<th>GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Desktop</td>
<td>Intel Core i7 4970K – Quad core @ 4.0GHz nominal</td>
<td>16GB DDR3-1866</td>
<td>GeForce GTX960 1024 CUDA Cores 2GB GDDR5 RAM</td>
</tr>
<tr>
<td>NVIDIA Jetson TK1</td>
<td>Tegra TK1 4-Plus-i™ Quad Core ARM A15 @ 2.3GHz Maximum</td>
<td>2GB DDR3L-1866</td>
<td>Embedded Kepler GPU 128 CUDA Cores Shared Memory</td>
</tr>
</tbody>
</table>

Table 5.1: Test system specifications.

CACTuS is tested for processing throughput using a standard set of tests, run with 16 SEFs. These tests use synthetic video that enables accurate ground truth data to be compared with the tracking outputs. As an additional test, the benefit to throughput by utilising multi core processing with OpenMP is also quantified by running a set of tests on builds with OpenMP disabled. Figure 5.1 shows the throughput result, measured in frames per second (fps), and higher is better.
Figure 5.1: The Processing Throughput of CACTuS CP

Here we can see a significant benefit from OpenMP on the desktop platform of over 5x. The throughput increase on the Jetson platform is also very good, the quad core achieving over 3.6x the single core. Using Open MP, it is possible to assign a number of threads. Testing confirmed that one thread per SEF gives the highest throughput for CACTuS CP on the desktop compared to one thread per CPU core for the Jetson.
5.4. GPGPU Acceleration - CACTuS CUDA

After code review and performance profiling of CACTuS CP, the obvious candidate for GPGPU acceleration was the heavily used convolution operation. The CPU version of this function is based on the specifications of MATLAB’s Conv2 function and implemented using the code from an early version of GNU Octave. This function was successfully converted to a CUDA kernel.

```c
// CUDA Kernel for Conv2 - adapted from Octave conv2
__global__ void conv2_d(float * output, int out_rows, int out_cols, float * input, int in_rows, int in_cols, float * kernel, int k_rows, int k_cols, int edg_x, int edg_y) {
    int oi = threadIdx.x + blockIdx.x * blockDim.x;
    if (oi> out_rows - 1) { return; } // Outside of output bounds.
    int oj = threadIdx.y + blockIdx.y * blockDim.y;
    if (oj> out_cols - 1) { return; } // Outside of output bounds.
    float sum = 0;
    for (int bj = k_cols - 1 - max(0, edg_y - oj),
         aj = max(0, oj - edg_y);
         bj >= 0 && aj < in_cols;
         bj++, aj--) {
        int bi = k_rows - 1 - max(0, edg_x - oi);
        int ai = max(0, oi - edg_x);
        const float* Ad = input + ai + in_rows*aj;
        const float* Bd = kernel + bi + k_rows*bj;
        for (; bi >= 0 && ai < in_rows;
             bi--, Bd--, ai++, Ad++) {
            sum += (*Ad) * (*Bd);
        }
    }
    *(output + (oj * out_rows + oi)) = sum;
}
```

**Code Excerpt:** CUDA Kernel for convolution.

This CUDA kernel computes the values for each output pixel in an independent thread on the GPU. This provides a very high level of parallelisation and a corresponding boost to performance.

The initial CUDA implementation involved the replacement of only the convolution with an equivalent CUDA function. This provided a significant performance gain on the desktop. The Jetson platform however, saw a detriment to processing. This is due to the overhead involved with each buffer copy from CPU (host) allocated memory to GPU (device) allocated memory. The overhead on the platform is significantly higher due to the lower memory bandwidth, the platform has 13GB/s of shared memory bandwidth whereas the desktop GPU has 112GB/s of GPU memory bandwidth, a PCI-e bandwidth of 15GB/s and a main memory bandwidth of 29GB/s.
Throughput was further increased for CACTuS CUDA on both platforms, by the reduction of the number of host to device and device to host memory copies. The entire prediction phase of the CACTuS algorithm was implemented in such a way that each matrix used was only copied from host to device once and copies from device to host were only made if necessary. By using application specific knowledge of the way each matrix is used in these calculations, a significant performance increase was gained by defining a specific memory management and CUDA kernel launch sequence, as detailed below.

In the prediction phase, four convolutions are run in succession. The first convolution’s output is used in the second convolution as an input. This pattern is followed by the successive convolutions, with the later relying on results from previous convolutions. By overlapping kernel invocations and asynchronous host to device and device to host memory copies, we can reduce the time spent waiting for data, increasing throughput. There is also some benefit for the generic convolution filter matrices to be copied only once per frame, this is accomplished by copying the data outside the loop that invokes prediction processing on each SEF.

In this research, cuFFT, NVIDIA’s FFT implementation for CUDA was tested, both for its performance merit and effect on tracking accuracy. FFT is used for approximating the largest convolution. As discussed earlier, FFTW is used in CACTuS CP, using cuFFT in place of FFTW is a simple exercise as NVIDIA provide a wrapper API.

There are several specific implementations of CACTuS CUDA and CACTuS CP included in the comparison. All versions use OpenMP to provide multi-threading for the main (SEF) loops. CACTuS CP was augmented with cuFFT and included in the comparison to measure its impact over FFTW. Table 5.2 details the configuration of the tested versions.
Table 5.2: Configurations of CACTuS.

<table>
<thead>
<tr>
<th>Version</th>
<th>FFTW</th>
<th>cuFFT</th>
<th>Convolution On CPU</th>
<th>Convolution On GPU</th>
<th>Optimised Prediction Phase</th>
</tr>
</thead>
<tbody>
<tr>
<td>CACTuS CP</td>
<td>Y</td>
<td></td>
<td>Y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cuFFT</td>
<td></td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Convolution Prediction</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Convolution + cuFFT</td>
<td></td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Prediction + cuFFT</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The results for the performance of the CUDA versions are detailed in figure 5.2.

The use of cuFFT in place of FFTW utilises the GPU to perform the FFT, and gains a small advantage over the CPU only CACTuS CP. By replacing the CPU targeted convolution function with the CUDA based equivalent, a throughput increase of 360% was gained. Further gains came from the use of the customised prediction phase where the use of overlapped memory transfer and CUDA kernel invocation delivers an increased throughput of a further 5.4 fps. CACTuS CP compared with CACTuS CUDA Prediction + cuFFT, corresponds to a frame rate increase from 7.75fps to 41.25fps. Interestingly, when the optimised prediction phase is used, cuFFT’s benefit is within the margin of error for the measurement.
5.5. CACTuS CUDA on Jetson

![Processing Throughput of CACTuS CUDA](image)

**Figure 5.3:** The processing throughput of CACTuS CUDA on Jetson.

As can be seen in Figure 5.3, implementing CACTuS CUDA on the Jetson platform resulted in a decrease in performance in all but the Prediction + cuFFT tests. The cause for this detrimental performance is the throughput of processing the convolution on the GPU on the Jetson is slower than the CPU. The integrated Kepler GPU and its 192 CUDA cores are not powerful nor parallel enough to process the convolution significantly faster than the quad core CPU. Further overhead is introduced when copying the matrices from CPU (host) allocated memory to GPU (device) allocated memory. It is likely the memory copy overhead that leads to the significantly lower throughput.

Despite the poor performance demonstrated in convolutions by the Jetson GPU, the use of cuFFT (which processes FFT on the GPU) provides a 32% boost to throughput, compared to FFTW in CACTuS CP. This result is surprising considering the minor difference cuFFT makes on the desktop. It is very likely that the pre-compiled FFTW library suffers from poor performance on the ARM based Jetson platform, despite being well optimised on the x86 based desktop.
NVIDIA’s CUDA supports many memory management models, including a ‘Zero-Copy’ global memory mode. This global mode allows memory to be allocated to be accessed by both the CPU and GPU simultaneously. By utilising this global memory model, there is no need to copy from CPU allocated memory to GPU allocated memory. This could potentially benefit the convolution on the GPU. However, an attempt to utilise the Zero-Copy feature of CUDA resulted in a significant performance penalty, taking an average of 2.8 seconds to process each frame. The performance detriment is very likely caused by inefficient caching of global memory in both GPU and CPU. Anecdotal evidence from NVIDIA's Developer Zone agrees with this hypothesis; on April 6th 2015 Shervin Emami of NVIDIA wrote:

Zero-copy removes the delay of transferring memory between CPU & GPU, but in Tegra K1 the zero-copy memory won’t be cached as well as regular GPU memory. So zero-copy is more likely to be faster in small simple kernels that don't access the same group of pixels more than once, while the traditional method is more likely to be faster in large complex kernels that access the same pixels many times. [31]

The CUDA kernel used for convolution is not large, but accesses each input pixel value a number of times, dependent on the size of the convolution filter matrix. The results from this study show that the reduced caching of CPU and GPU shared data contributes to a significant detrimental performance impact for this CUDA kernel.

Despite a large benefit for all the CUDA versions on the desktop, the Jetson platform gains negligible performance benefit from CACTuS CUDA (Prediction + cuFFT) when compared with CACTuS CP. As will be shown in the proceeding section, it is possible to run convolution on the GPU and CPU simultaneously, by taking advantage of the design of the matrix class.
5.5.1. CACTuS CUDA Hybrid

The CACTuS CUDA Hybrid mode takes advantage of the CPU and GPU simultaneously to accelerate a portion of the prediction phase convolutions by splitting the load between the CPU and GPU. Conceptually, CACTuS CUDA Hybrid runs half the SEFs in CACTuS CP + cuFFT mode and the remaining half in the CACTuS CUDA Prediction + cuFFT mode.

This provides a significant boost to throughput, gaining 40% improvement over the next fastest version (5.55 fps for CACTuS CP with cuFFT vs 7.78 fps for CACTuS CUDA hybrid mode) as shown in figure 5.4.

![Processing Throughput of CACTuS Implementations](image)

**Figure 5.4**: The processing throughput of CACTuS implementations on Jetson.

The hybrid mode provides the best throughput of any CACTuS implementation on the Jetson platform. The hybrid version is faster as it effectively utilises unused capacity on the GPU, compared to the CACTuS CP + cuFFT version, to run prediction convolutions for half of the SEFs. Several processing allocations between GPU and CPU were attempted. It was found experimentally that the highest efficiencies are obtained by running half the predictions on the GPU and half on CPU.
<table>
<thead>
<tr>
<th>Implementation</th>
<th>Prediction (ms)</th>
<th>Prediction per SEF (ms)</th>
<th>Observation (ms)</th>
<th>Overall (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>cuFFT – Jetson</td>
<td>129.5</td>
<td>8.1</td>
<td>41.4</td>
<td>180.3</td>
</tr>
<tr>
<td>Prediction + cuFFT - Jetson</td>
<td>135.2</td>
<td>8.4</td>
<td>80.6</td>
<td>235.2</td>
</tr>
<tr>
<td>Hybrid – Jetson</td>
<td>73.8</td>
<td>4.6</td>
<td>43.8</td>
<td>128.5</td>
</tr>
<tr>
<td>cuFFT – Desktop</td>
<td>96.8</td>
<td>6.1</td>
<td>22.8</td>
<td>123.0</td>
</tr>
<tr>
<td>Prediction + cuFFT - Desktop</td>
<td>10.9</td>
<td>0.68</td>
<td>10.1</td>
<td>24.24</td>
</tr>
</tbody>
</table>

**Table 5.3:** Processing rates of CACTuS implementations.

Table 5.3 gives a clear insight as to how the Jetson platform benefits from the hybrid mode. The timing for prediction in the CPU driven prediction phase of CACTuS CP is very close to the timing for the CUDA based prediction phase on the Jetson platform. It is this similarity that provides an optimal result when the predictions are equally shared by the CPU and GPU. Further to this, the observation phase runs fastest when the convolutions are processed on the CPU and the FFT on GPU. By splitting the prediction phase between CPU and GPU, running the FFT on the GPU and the remaining convolutions on the CPU, the hybrid mode is able to provide the highest throughput. The minor throughput drop between the observation phase of the hybrid mode and the CACTuS CP + cuFFT mode is likely due to caching in efficiencies as some of the data is not accessed by the CPU prior to the observation phase.

Table 5.3 also outlines why the desktop does not benefit from splitting the prediction between CPU and GPU. The desktop GPU is able to process the entire 16 SEF’s prediction phase faster than the CPU can perform two of the SEF’s predictions. There is no split of predictions between CPU and GPU that can match the performance of the Prediction + cuFFT implementation. This is due to the inefficiencies of caching and threading issues for running a single prediction, which is inherently serial, on multiple CPU cores.
5.5.2. Real time on Jetson

Despite the performance of the CACTuS CUDA Hybrid exceeding the performance of the desktop CPU only CACTuS CP, it does not provide a real time solution for 20fps video. In order to reduce the processing requirements of CACTuS, it can be run with fewer SEFs. This change reduces the number of objects that CACTuS is able to track, but does not change the type of objects nor its Track Before Detect (TBD) capability.

To that end, a series of tests using lower numbers of SEFs was undertaken. As the competitive mechanism starts the SEFs in a grid and the input image is square, it was decided that square allocations of SEFs would be tested. 9 SEFs and 4 SEFs were tested and the results are shown in figure 5.5.

![Processing Throughput of CACTuS CUDA Hybrid](image)

**Figure 5.5:** The processing throughput of CACTuS CUDA Hybrid on Jetson.

Figure 5.5 shows the throughput can be improved by reducing the number of SEFs. With 4SEFs, CACTuS is able to track 4 targets. By running CACTuS CUDA with 4SEFs, it is possible to meet the throughput target necessary for real time operation with 20 fps video on the Jetson platform.
5.6. Tracking Performance

Our lab has created a standard set of synthetic videos for the purpose of comparing tracking implementations. These videos are generated such that the exact location of each object is known for every frame, this set of data is called ground truths. Each of the CACTuS implementations generate MATLAB compatible output files for each frame. This enables a MATLAB based script to compare the algorithm outputs to ground truths for each tested synthetic video file. By running this set of standardised tests on the output of each tracking implementation, the effect of implementation on tracking performance can be evaluated. It is important to note that new faster versions of CACTuS must have similar tracking performance to be considered as viable alternatives.

These tests are performed using a standard setting of 16 SEFs, with an image size of 127 x 127 pixels. The synthetic video has 160 frames.

The performance of the tracker is measured using 3 key performance metrics; Track Completeness, Mean Track Matching Error, and Mean Track Latency. The track completeness is a measure, in percentage, of the distance that the tracker accurately follows the target. The Track Matching Error is calculated as the difference, in pixels, of the detected object’s centroid vs the ground truth centroid. The track latency is a measure of the number of frames elapsed before the tracker has acquired the target. As each SEF tracks multiple probable targets, the acquisition is defined as the point at which a particular target becomes the primary target in a SEF.

<table>
<thead>
<tr>
<th>Implementation</th>
<th>Mean Track Completeness (%)</th>
<th>Mean Track Matching Error (Pixels)</th>
<th>Mean Track Latency (Frames)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CACTuS CP – Desktop</td>
<td>85</td>
<td>1.53</td>
<td>14.5</td>
</tr>
<tr>
<td>CACTuS CP – Jetson</td>
<td>82</td>
<td>1.52</td>
<td>21</td>
</tr>
<tr>
<td>CACTuS CUDA – Desktop</td>
<td>84</td>
<td>1.47</td>
<td>13.5</td>
</tr>
<tr>
<td>CACTuS CUDA – Jetson</td>
<td>83</td>
<td>1.47</td>
<td>16</td>
</tr>
<tr>
<td>CACTuS CP + cuFFT Jetson</td>
<td>83</td>
<td>1.49</td>
<td>17</td>
</tr>
</tbody>
</table>

Table 5.4: Tracking performance results.

The results are consistent when CPU vs GPU implementations are considered. The FFT algorithm implementation is different between CACTuS CP and CACTuS CUDA, with the CPU version using FFTW and CACTuS CUDA utilising cuFFT.
The results show that CACTuS benefits from the accuracy of the cuFFT implementation of CACTuS CUDA when compared against the FFTW implementation in CACTuS CP, with better track latency and lower matching errors. The Jetson platform implementation suffers an increase in track latency of 44% on the CPU targeted CACTuS CP and an increase of only 18% with CACTuS CUDA. The track matching accuracy and completeness of the Jetson implementations is within 3% of the desktop versions. The increase in track latency and matching error is most likely to be caused by the inaccuracies of FFTW on the ARM when compared to the desktop CPU FFTW implementation for CACTuS CP. CACTuS CUDA likely suffers from reduced floating point arithmetic precision of the ARM FPU.
5.7. Performance per Watt

In order to quantify the efficiency of the implementation, each solution was measured for average processing time per frame and average power usage in watts. The combined result allows for a frames per second per watt for each platform to be calculated, allowing for direct comparison.

For this test, a common setting of 16 SEFs was used, with one thread per SEF allocated using OpenMP. The measurement setup was devised such that the power usage for the entire platform, including 240V power supply, was measured. An off the shelf power monitor was used, in line with the 240V power connection. To test the efficiency of each platform, a series of tests were undertaken, with the results averaged. The efficiency of both CACTuS CP and CACTuS CUDA versions were tested. These new versions are compared on the desktop and platform and are compared to the existing CACTuS FPGA implementation. As an additional comparison, the server used for CACTuS FPGA was also benchmarked with CACTuS CP and CACTuS CUDA for power efficiency comparison.

<table>
<thead>
<tr>
<th>System</th>
<th>CPU</th>
<th>RAM</th>
<th>GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Desktop</td>
<td>Intel Core i7 4970K – Quad core @ 4.0GHz nominal</td>
<td>16GB DDR3-1866</td>
<td>GeForce GTX960 1024 CUDA Cores 2GB GDDR5 RAM</td>
</tr>
<tr>
<td>NVIDIA Jetson TK1</td>
<td>Tegra TK1 4-Plus-i™ Quad Core ARM A15 @ 2.3GHz Maximum</td>
<td>2GB DDR3L-1866</td>
<td>Embedded Kepler GPU 128 CUDA Cores Shared Memory</td>
</tr>
<tr>
<td>Server</td>
<td>Intel Xeon X5677 Quad core @ 3.47GHz</td>
<td>6GB DDR3</td>
<td>NVIDIA Quadro 4000 256 CUDA Cores 2GB GDDR5 RAM</td>
</tr>
</tbody>
</table>

*Table 5.5: Test system specifications.*
Figure 5.6: The Power Efficiency of CPU based CACTuS Implementations.

<table>
<thead>
<tr>
<th>Version</th>
<th>Platform</th>
<th>Processing Rate (fps)</th>
<th>Average Power (W)</th>
<th>Fps/watt</th>
</tr>
</thead>
<tbody>
<tr>
<td>MATLAB</td>
<td>Desktop</td>
<td>6.8</td>
<td>123</td>
<td>0.055</td>
</tr>
<tr>
<td>CACTuS CP</td>
<td>Desktop</td>
<td>7.75</td>
<td>142</td>
<td>0.055</td>
</tr>
<tr>
<td>CACTuS CP</td>
<td>Jetson</td>
<td>4.18</td>
<td>14.5</td>
<td>0.288</td>
</tr>
<tr>
<td>CACTuS CP</td>
<td>Server</td>
<td>4.81</td>
<td>308</td>
<td>0.015</td>
</tr>
</tbody>
</table>

Table 5.6: CACTuS power efficiency test results. (16 SEFs)

Here we can see over 5 times efficiency increase per Watt for the Jetson multi core CPU implementation. This result demonstrates the mobile targeted ARM processor’s increased energy efficiency, which is substantial, despite comparing a 28nm ARM processor to a 22nm Intel Core processor.

The newer Core i7-4790 significantly outperforms the older Xeon X5677, on both raw performance and performance per watt. Of note is the performance increase of over 60%, despite the core clock increase of only 30%. The Core i7 based desktop exhibits over 3 times the power efficiency of the Xeon based server.

The next set of tests compare the accelerated versions of CACTuS, comparing the highest performing CACTuS CUDA (Prediction + cuFFT) on the desktop, the CACTuS CUDA Hybrid on the Jetson and previous versions of CACTuS.
Table 5.5: Accelerated CACTuS power efficiency test results. (16 SEFs)

*(Prediction + cuFFT)

<table>
<thead>
<tr>
<th>Version</th>
<th>Platform</th>
<th>Processing Rate (fps)</th>
<th>Average Power (W)</th>
<th>Fps/watt</th>
</tr>
</thead>
<tbody>
<tr>
<td>CACTuS</td>
<td>Desktop</td>
<td>41.25</td>
<td>160</td>
<td>0.258</td>
</tr>
<tr>
<td>CUDA *</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CACTuS</td>
<td>Jetson</td>
<td>7.78</td>
<td>16</td>
<td>0.486</td>
</tr>
<tr>
<td>CUDA Hybrid</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CACTuS</td>
<td>Server</td>
<td>16.75</td>
<td>300</td>
<td>0.056</td>
</tr>
<tr>
<td>CUDA *</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CACTuS Vienna</td>
<td>Server</td>
<td>8.77</td>
<td>300</td>
<td>0.029</td>
</tr>
<tr>
<td>CACTuS FPGA</td>
<td>Server</td>
<td>23.9</td>
<td>295</td>
<td>0.081</td>
</tr>
</tbody>
</table>

There is still a clear advantage for the Jetson platform in this test, however not as pronounced as the results for the CACTuS CP test. The Xeon server gains significantly from the CUDA implementation, compared to both CACTuS CP and CACTuS Vienna, however it still is not able to compete with the energy efficiency of the Core i7 based desktop or SoC based Jetson platforms.
6. CONCLUSION

The example system, as demonstrated in this thesis, shows that it is possible to implement a complex multi object tracker on a SoC based platform.

The SoC based Jetson platform shows a clear benefit for power consumption with the CACTuS CP showing a fivefold increase in performance per watt on the ARM CPU over the desktop’s Core i7 CPU.

CACTuS CP demonstrates the applicability of OpenMP to the CACTuS tracker and the results for both the desktop and Jetson platform reflect a very good increase in performance for a naïve multi-threaded implementation.

The performance increase from implementing the convolution operation in CUDA was significant on the desktop. On such a platform, GPGPU is a stable, mature solution for acceleration of a visual tracking algorithm. CACTuS CUDA (Prediction + cuFFT) shows significant throughput increase from CACTuS CP. It surpasses the throughput of CACTuS Vienna, when used on the same hardware, and is the highest performing desktop version of CACTuS to date. This brings a throughput level that enables real time tracking on 30 frame per second video.

The CACTuS CUDA Hybrid on Jetson provides the best power to throughput ratio of any multi object tracking algorithm tested. This demonstrates the power efficiency of current embedded SoC and GPU designs.

The Jetson platform is capable of running CACTuS at respectable frame rates with 16 SEFs, when utilising the hybrid CPU and GPU implementation. At 7.7 fps, it compares favourably to the MATLAB and CACTuS CP desktop implementations, achieving a similar processing and performance level, but significantly better power efficiency. Utilising a reduced SEF count of 4 enables the platform to achieve a processing rate of 21.5 fps, achieving throughput adequate for real time processing of 20 fps video.
6.1. Future Research Directions

On both platforms, CACTuS performance may benefit from additional performance profiling. Specifically, there is likely to be performance gained by implementing the observation phase using an overlapped kernel invocations and memory transfers.

On the Jetson platform, use of FFT to perform the medium sized convolution operations to increase throughput, by the reduction of complexity, could be explored. The exploration of parallel implementations utilising multiple Jetson platforms could allow for highly energy efficient systems compatible with both high frame rate (>15fps) and larger numbers of SEFs (9+).

The structure of the CACTuS CP and, consequently, CACTuS CUDA implementations is easily adaptable to take advantage of new hardware platforms as they emerge. An exploration of CACTuS' performance on the upcoming NVIDIA Jetson TX1 platform is recommended for a future study.
REFERENCES


Appendix 1: Convolution Code

```cpp
// Portable version of conv2 – code from
/*
 * conv2: 2D convolution for octave
 *
 * Copyright (C) 1999 Andy Adler
 * This code has no warranty whatsoever.
 * Do what you like with this code as long as you
 *     leave this copyright in place.
 */

mat mat::cpu_conv2(const mat& filter, ConvShape shape) const {
    int filt_rows = filter.rows();
    int filt_cols = filter.cols();

    int out_rows = 0, out_cols = 0, edg_x = 0, edg_y = 0;
    switch (shape) {
    case SHAPE_FULL:
        out_rows = n_rows + filt_rows - 1;
        out_cols = n_cols + filt_cols - 1;
        edg_x = filt_rows - 1;
        edg_y = filt_cols - 1;
        break;
    case SHAPE_SAME:
        out_rows = n_rows;
        out_cols = n_cols;
        // Matlab seems to arbitrarily choose this convention for
        // 'sn_rowse' with even length R, C
        edg_x = (filt_rows - 1) / 2;
        edg_y = (filt_cols - 1) / 2;
        break;
    case SHAPE_VALID:
        out_rows = n_rows - filt_rows + 1;
        out_cols = n_cols - filt_cols + 1;
        edg_x = edg_y = 0;
        break;
    }

    mat output(out_rows, out_cols);

    for (int oi = 0; oi < out_rows; oi++) {
        for (int oj = 0; oj < out_cols; oj++) {
            float sum = 0;

            for (int bj = filt_cols - 1 - std::max(0, edg_y - oj),
                aj = std::max(0, oj - edg_y);
                bj >= 0 && aj < n_cols;
                bj--, aj++) {
                int bi = filt_rows - 1 - std::max(0, edg_x - oi);
                int ai = std::max(0, oi - edg_x);
                float* Ad = this->data + ai + n_rows*aj;
                float* Bd = filter.data + bi + filt_rows*bj;

                for (; bi >= 0 && ai < n_rows;
                    bi--, Bd--, ai++, Ad++) {
                    sum += (*Ad) * (*Bd);
                }
            }
        }
    }

    return output;
}
```