UNIVERSITY OF CAPE TOWN
IYUNIVESITHI YASEKAPA • UNIVERSITEIT VAN KAAPSTAD

A GPU based X-Engine for the MeerKAT Radio Telescope

by

Gareth Mitchell Callanan

A dissertation submitted in partial fulfilment of the requirements for the degree of

Master of Science in Engineering

in the Department of Electrical Engineering,
Faculty of Engineering and the Built Environment

at the

University of Cape Town

15 September 2020

Supervisor: Dr. Simon Winberg

Plagiarism Declaration

I know that plagiarism is wrong. Plagiarism is to use another's work and pretend that it is my own.

I have used the IEEE convention for citation and referencing. In this dissertation, all contributions to, and quotations from the work(s) of other people have been cited and referenced.

This dissertation is my own work. I have not allowed, and will not allow, anyone to copy my work.

Gareth Callanan

15 September 2020

Abstract

The correlator is a key component of the digital backend of a modern radio telescope array. The 64 antenna MeerKAT telescope has an FX architecture correlator consisting of 64 F-Engines and 256 X-Engines. These F- and X-Engines are all hosted on 128 custom designed FPGA processing boards. This custom board is known as a SKARAB. One SKARAB X-Engine board hosts four logical X-Engines. This SKARAB ingests data at 27.2 Gbps over a 40 GbE connection. It correlates this data in real time.

GPU technology has improved significantly since SKARAB was designed. GPUs are now becoming viable alternatives to FPGAs in high performance streaming applications. The objective of this dissertation is to investigate how to build a GPU drop-in replacement X-Engine for MeerKAT and to compare this implementation to a SKARAB X-Engine. This includes the construction and analysis of a prototype GPU X-Engine.

The 40 GbE ingest, GPU correlation algorithm and the software pipeline framework that links these two together were identified as the three main sub-systems to focus on in this dissertation. A number of different tools implementing these sub-systems were examined with the most suitable ones being chosen for the prototype.

A prototype dual socket system was built that could process the equivalent of two SKARABs worth of X-Engine data. This prototype has two 40 GbE Mellanox NICS running the SPEAD2 library and a single Nvidia GeForce 1080Ti GPU running the xGPU library. A custom pipeline framework built on top of the Intel Threaded Building Blocks (TBB) library was designed to facilitate the flow of data between these sub-systems.

The prototype system was compared to two SKARABs. For an equivalent amount of processing, the GPU X-Engine cost R143 000 while the two SKARABs cost R490 000. The power consumption of the GPU X-Engine was more than twice that of the SKARABs (400W compared 180W), while only requiring half as much rack space. GPUs as X-Engines were found to be more suitable than FPGAs when cost and density are the main priorities. When power consumption is the priority, then FPGAs should be used.

When running eight logical X-Engines, 85% of the prototype's CPU cores were used while only 75% of the GPU's compute capacity was utilised. The main bottleneck on the GPU X-Engine was on the CPU side of the server. This report suggests that the next iteration of the system should offload some CPU side processing to the GPU and double the number of 40 GbE ports. This could potentially double the system throughput. When considering methods to improve this system, an FPGA/GPU hybrid X-Engine concept was developed that would combine the power saving advantage of FPGAs and the low cost to compute ratio of GPUs.

Acknowledgements

I would like to thank:

My supervisor, Dr Simon Winberg, for helping turn my idea from a collection of disjointed concepts into a coherent dissertation, as well as for always being available to assist during the last few weeks before submission.

Dr Jason Manley for being an endless source of information on correlator design and for always being willing to stop what he was doing to enthusiastically answer my technical questions.

Martin Slabber for being able to provide me with the necessary hardware required.

David MacMahon for responding to my emails across time zones to assist me with configuring xGPU.

Dr Bruce Merry for patiently answering all my questions related to 40 GbE NICs even after the email chain had reached beyond 30 messages.

Dr Andrew van der Byl for providing valuable insight into the world of academia.

To the South African Radio Astronomy Organisation (SARAO) DSP team for being an endless source of amusement, inspiration and support.

To the entire SARAO organisation. SARAO is filled with people who were always willing to provide any assistance I needed. This ranged from technical support, to assisting with locating archived quotations, to legal advice as to what information I could publicly disclose.

I would like to thank the National Research Foundation (NRF) and Peralex Electronics for providing funding for this dissertation.

Finally, but most importantly, I would like to thank my parents for laying a strong foundation. I would not be where I am today without their constant support. An additional thanks to my father for proofreading all my work in the last few days before submission.

Table of Contents

List of Figures

List of Tables

Glossary

1k Mode: This is a MeerKAT mode of operation where the antenna data has undergone a 1024 point Fourier transform.

32k Mode: This is a MeerKAT mode of operation where the antenna data has undergone a 32 768 point Fourier transform.

40 GbE: 40 Gigabit Ethernet (40 GbE) is a network interconnect designed to support full duplex communication between connected nodes.

4k Mode: This is a MeerKAT mode of operation where the antenna data has undergone a 4096 point Fourier transform.

ALMA: The Atacama Large Millimeter Array (ALMA) is a 66 antenna radio telescope located in the Chilean Andes.

ASIC: An Application Specific Integrated Circuit (ASIC) is an integrated circuit that is designed and optomised for a specific function.

ASKAP: The Australian Square Kilometre Array Pathfinder (ASKAP) is a 36 antenna radio telescope located in the Australian Mid West.

Baseline: A baseline is the output from a correlator. Each baseline is produced by correlating and Fourier transforming two antenna signals.

Boilerplate Code: In computer programming, boilerplate code refers to code that has to be repeated with minimal modification in many locations throughout a program.

Breakthrough Listen: Breakthrough Listen is a large research program with the goal of finding evidence of intelligent life beyond Earth.

CHIME: The Canadian Hydrogen Intensity Mapping Experiment (CHIME) array is a 1024 antenna radio telescope array located in British Columbia, Canada.

Device: In a GPU system, the device refers to the GPU that is connected to the server.

F-Engine: An F-Engine is the part of the correlator that is responsible for performing a Fourier transform on time domain data.

FAST: The Five-hundred-meter Aperture Spherical radio Telescope (FAST) is the worlds largest single dish antenna located in Guizhou, China.

FPGA: A Field Programmable Gate Array (FPGA) is an integrated circuit that is designed to be configured at the gate level during run-time.

GPGPU: A General Purpose Graphics Processing Unit (GPGPU) is a GPU that is designed specifically for performing computation. GPGPUs generally lack the video interfaces present on a normal GPU.

GPU: A Graphics Processing Unit (GPU) is an integrated circuit originally designed to process computer graphics that can also perform complex multi-threaded computation.

HERA: The Hydrogen Epoch of Reionisation Array (HERA) is a 350 antenna radio telescope located in the Karoo desert in South Africa..

Host: In a GPU system, the host refers to the server that the GPU is connected to..

HPC: High Performance Computing (HPC) is a field of computing focused on building extremely large computers (or collections of computers) for the purposes of accelerating complex computational problems.

ICD: An Interconnect Document (ICD) is a document that describes a set of standards that all nodes on a network must adhere to when transmitting and receiving network data.

Interconnect: An Interconnect is a system that provides a means for other systems to transfer data to one another. In the context of MeerKAT, this generally refers to a 40 GbE network.

L-Band: L-band refers to the operating frequency range of 1–2 GHz in the radio spectrum. With reference to MeerKAT, L-band refers to the frequecny range 0.856 - 1.712 GHz.

LEDA: The Large-Aperture Experiment to Detect the Dark Ages (LEDA) telescope is a 256 antenna radio telescope located in New Mexico, USA.

MeerKAT: MeerKAT is a 64 antenna L-band radio telescope built and operated by SARAO. It is located in the Karoo desert in South Africa.

Multicast: Multicast in an IP network refers for the ability of a network to transmit the same data to multiple end nodes without transmitting redundant packets.

MWA: The Murchison Widefield Array (MWA) is a 128 antenna radio telescope located in the Australian Mid West.

NIC: A Network Interface Card (NIC) is a hardware device that allows a computer to connect to a network.

NUMA: Non-Uniform Memory Access (NUMA) refers to systems where different CPUs have different latencies when accessing different memory banks..

Nvidia: Nvidia is one of the two major manufacturers of GPUs in the world.

OOP: Object Orientated Programming (OOP) is a software development paradigm where related pieces of data are encapsulated into a single object that provides an interface to create, access and manipulate this data.

PAPER: The Precision Array for Probing the Epoch of Re-ionisation (PAPER) is a 128 antenna radio telescope located in the Karoo desert in South Africa. It is a precursor to HERA.

PFB: A Polyphase Filter Bank (PFB) is an efficient implementation of a FIR filter followed by an FFT. The PFB performs windowing on time series data before Fourier transforming it in order to reduce side-lobes in the frequency domain.

Pipeline: A pipeline in a data streaming application is a software framework that facilitates moving data between processing threads.

Polymorphism: Polymorphism is the ability for an object in an OOP inheritance hierarchy to take on the form of any object in this hierarchy. This allows related objects of different types to be accessed through the same interface.

SARAO: The South Africa Radio Astronomy Observatory (SARAO) is facility funded by the National Research Foundation that is responsible for managing all radio astronomy activity in South Africa. This includes the MeerKAT radio telescope.

SKA: The Square Kilometre Array (SKA) is an intergovernmental radio telescope project to be built partly in South Africa and partly in Australia. It aims to be the largest radio telescope in the world and is scheduled to begin operation in the late 2020s..

SKA-Low: SKA-Low refers to the 50 MHz to 350 MHz bandwidth portion of the Square Kilometer Array to be constructed in Australia.

SKA-Mid: SKA-Mid refers to the 350 MHz to 14 GHz bandwidth portion of the Square Kilometer Array to be constructed in South Africa.

SKARAB: The Square Kilometre Array Reconfigurable Application Board (SKARAB) is an FPGA system designed specifically to be used in the MeerKAT correlator.

SPEAD: SPEAD is a streaming protocol implemented on top of UDP that is used in the MeerKAT network. It requires data to be organised into logical heaps. Each heap is transmitted as a series of packets and then reassembled at the destination node.

Stokes Parameters: Stokes Parameters are a set of values that describe the polarisation of received radio signals.

Synchronisation Epoch: In MeerKAT, a Synchronisation Epoch refers to collection of correlated data that is accumulated together by the MeerKAT X-Engines. The end of an epoch is the last time stamp accumulated before the accumulated data is sent to the next stage in the signal pipeline.

TBB: Intel's Threaded Building Blocks (TBB) library is a C++ tool designed to make it easier to develop multithreaded programs.

UML: Unified Modeling Language (UML) is a standardized set of tools for specifying and visualising software programs.

VLA: The Very Large Array (VLA) is a 28 antenna radio telescope located in New Mexico, USA.

X-Engine: An X-Engine is the part of the correlator that is responsible for performing frequency domain correlation between every antenna pair.

1. Introduction

This past decade has seen an explosion in the complexity and capability of modern radio telescopes. The era of single antenna telescopes has come to an end. Modern radio telescopes are built out of multiple smaller antennas made to appear as a single larger antenna. The collection of techniques used to do this is called interferometry and an array of dishes is known as an interferometer. Current interferometers often consist anywhere from tens of dishes (such as MeerKAT[1] and ASKAP[2]) to hundreds or even thousands of dishes (HERA[3] and CHIME[4]). Future telescope designs (including SKA-Mid and SKA-Low[5]) will continue to increase both the number of antennas per interferometer and the data rates per antenna.

The digital back-ends servicing these arrays are forced to grow and improve in order to keep up with the required data rates. Modern digital back-ends do not run on a single server, instead they consist of many racks of servers running in a data centre. Solving modern radio astronomy problems increasingly involves entering the field of high performance computing (HPC).

The core focus of this dissertation is on the design of a GPU based drop-in replacement X-Engine for the MeerKAT radio telescope. X-Engines make up part of this digital back-end and the results from this dissertation can be used to inform future telescope design decisions.

This chapter introduces the reader to the problem, presents the project objectives, and outlines the structure to be followed by the rest of this dissertation.

1.1. Problem Background

The MeerKAT radio telescope is a 64 Antenna interferometer located in the Karoo desert in South Africa. The management and operation of MeerKAT is overseen by the South African Radio Astronomy Observatory(SARAO). [1]

The MeerKAT correlator was built using a custom FPGA platform called the Square Kilometre Array Reconfigurable Astronomy Board (SKARAB). The correlator consists of 64 SKARABs performing channelisation (referred to as F-Engines) and 64 SKARABs performing correlation (referred to as X-Engines) all connected over 40 Gigabit-Ethernet (40 GbE). A more detailed explanation of the correlator will be presented in Chapter 3.

The SKARABs were designed in 2013 and will reach their designated end of life in 2022. A replacement correlator is currently being designed. Due to the advancement in technology since 2013, it is expected that the new correlator will be able to perform the same amount of

computation for a far lower cost. This can be achieved due to the performance per component increasing resulting in fewer components being required. A number of new platforms are being considered for this upgrade, including FPGA and GPU platforms.

This dissertation will focus on the development of a GPU X-Engine prototype for the correlator. This includes reviewing the current state of the art of GPU X-Engines as well as designing, building, and testing a prototype.

1.2. Research Focus

The research focus of this dissertation was determined after a consultation with various stakeholders.

The objective of this dissertation is to answer a number of questions about designing and implementing a GPU based X-Engine for the MeerKAT correlator.

In order to thoroughly answer these questions, a prototype platform needed to be built. As such a functional prototype GPU X-Engine is a core deliverable of this dissertation.

The research questions lead to some high level system requirements for the prototype. These requirements were derived after further discussions with stakeholders. These high level requirements will be translated into formal system requirements in Section 4.1.

The research questions and the corresponding high level prototype system requirements are listed below:

- **Research Question 1:** *How does one design and build a GPU drop-in replacement X-Engine system for MeerKAT?* A number of telescopes have implemented GPU based X-Engines. This dissertation needs to explore how to build a GPU based X-Engine that processes MeerKAT specific data. The bulk of this dissertation will be dedicated to determining the best approach to design this system. The different design approaches and tools available to build this system are explored in Chapter 3 .The design of this GPU X-Engine is discussed in Chapter 4 and the verification of this design is presented in Chapter 5. MeerKAT X-Engines receive and process data at 27.2 Gbps. This data is formatted according to the MeerKAT packet standard. A drop-in replacement would thus need to conform to the following high level requirements:
  - **High Level Requirement 1:** *The GPU X-Engine prototype shall ingest MeerKAT F-Engine data at 28 Gbps over a 40 GbE port.* A SKARAB X-Engine has a single 40 GbE port that subscribes to 4 MeerKAT F-Engine multicast streams. Each multicast stream consists of data from 64 F-Engines with a combined data rate of just under 7 Gbps per stream. The GPU X-Engine prototype will need to process 4 of these

multicast streams per port.

- **High Level Requirement 2:** *The GPU X-Engine prototype shall accept and transmit data according to the format specified in the MeerKAT Functional Interface Control Document (M1000-0001-020).* MeerKAT Ethernet packets must conform to a standard called SPEAD. This standard groups packets into a single logical collection of data called a heap. The format of the X-Engine input and output heaps is defined in the *MeerKAT Functional Interface Control document (ICD) for the Correlator-Beamformer Visibilities and Tied Array Data.* A GPU X-Engine conforming to this standard will be indistinguishable from a SKARAB X-Engine.
- **High Level Requirement 3:** *The GPU X-Engine needs to correlate the data from all 64 antennas in real time.* The MeerKAT system is configured such that each X-Engine always receives data at 7 Gbps per multicast stream. For correlation, the amount of computation is not only a function of input data rate, it is also a function of number of antennas and polarisation requirements. The output data rate is also function of the accumulation time. Thus to fully describe the computational requirements, the GPU X-Engine shall adhere according to the following:
  - Correlate data from 64 antennas.
  - Perform full stokes correlation.
  - Support accumulation times between 0.5 and 2 seconds.

  The above requirements will be explained in more depth in Section 3.1

- **Research Question 2:** *How much rack space will this system require?* The SKARABs were designed in 2013. Since then, computing technology has continued to march inexorably forward. More and more computing capacity is able to fit in the same physical space. Chapter 6 explores the possibility of designing a more compact GPU X-Engine. The physical constraints on the system lead to another requirement on the prototype:
  - **High Level Requirement 4:** *The GPU X-Engine implementation must use rack space equal to or less than the SKARAB X-Engines for the same amount of compute.* The correlator has a specific maximum rack space allocated in the MeerKAT server room. The SKARABs are 1U high and process 4 multicast streams each. The GPU X-Engine must process at least 4 multicast streams per U but it is not required that the X-Engine be 1U high. For example, a 4U high GPU X-Engine would need to process 16 multicast streams.
- **Research Question 3:** *How does the system cost, performance and power consumption*

differ between the current FPGA X-Engine and the GPU X-Engine replacement? There are differences between FPGA and GPU systems. These differences need to be analysed to determine if it makes financial and operational sense to use a GPU system over an FPGA system in future correlators. This analysis is performed in Chapter 6.

Following these research questions and high level requirements, a hypothesis for this dissertation was formulated:

HYPOTHESIS:

It is possible to design a GPU based X-Engine that will perform to the same specifications as the SKARAB X-Engine: for the same level of performance, this GPU system will have a cheaper purchase price, require less rack space, and consume more power than a SKARAB equivalent.

1.3. Dissertation Overview

This section provides a brief outline of the seven chapters of this research project.

Chapter 1 has introduced the topic. It has given a brief outline of the topic as well as presented the hypothesis, research question and requirements. It has given a broad overview of the structure of the project.

Chapter 2 will outline the dissertation methodology. This chapter will present the strategy required to take this dissertation from a collection of broad requirements into a functioning prototype that can be used to test the hypothesis.

Chapter 3 is the literature review. This review will establish the scientific and engineering context of the dissertation, explore the technologies that can be used to build a GPU X-Engine and examine how other telescopes have implemented similar systems. This chapter will determine which technologies are to be used in the prototype.

Chapter 4 will provide a detailed design for the system. This includes providing more detailed requirements and implementing the suitable technologies identified in the literature review according to these requirements. The prototype will be verified in Chapter 5 to ensure that it meets all high level requirements. All software that is designed will be stored on a publicly accessible GitHub repository: https://github.com/gcallanan/GPU_XEngine.

In Chapter 6 the prototype will be analysed to further the understanding of the system.

The dissertation will be concluded in Chapter 7. This conclusion will provide answers to all research questions, discuss the hypothesis and give recommendations for future research to be carried out.

2. Methodology

Section 1.2 presented the research questions for this dissertation, as well as high level requirements for a GPU X-Engine prototype. This section will describe the methodology used to take this dissertation from research questions to a complete body of research with a functioning prototype that can test the hypothesis completely.

The objective of this dissertation is to test the hypothesis and answer the research questions. In its current form, the project consists of research questions and high-level requirements. This gives the project a broad focus while having no well developed prototype capable of testing the hypothesis. Over the course of this dissertation, the focus of the project will narrow, while the depth of understanding will increase. This will take place in three main stages: Concept Development, System Design and System Analysis. This is illustrated in Figure 2.1.

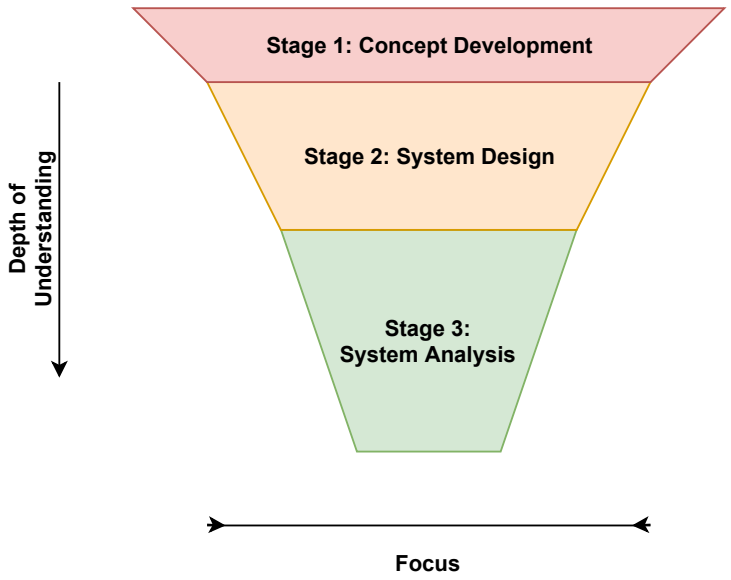


Figure 2.1: Diagram showing the different stages of the research project.

By the end of the system analysis stage, the research questions must be answered and the hypothesis must be tested.

SARAO follows the Systems Engineering approach to system design[6]. The concept development and system design stages are core parts of the Systems Engineering system acquisition process. The Systems Engineering approach to system design has informed Stage 1 and Stage 2 of this methodology.[7]

2.1. Stage 1: Concept Development

The concept development stage begins before the dissertation topic has even been formalised. It involves taking the dissertation from a collection of thoughts and turning it into a well defined topic. From this topic a robust set of requirements needs to be formulated which tightly directs the design of the prototype platform. The generation of these requirements needs to be informed by exploring the current state of the art of the relevant technology - this exploration is a core part of the concept development.

Concept development takes place in three steps. These steps are illustrated in Figure 2.2

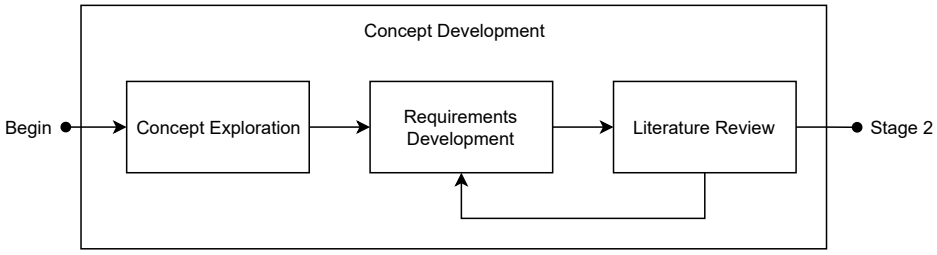


Figure 2.2: Diagram showing the progression of the concept development stage.

The first step is called concept exploration. This involves formalising the project topic. It requires meeting with all stakeholders to determine what different outcomes are expected from this project. From these meetings, the hypothesis and research questions are formulated. The results of this step are given in Section 1.2.

This dissertation requires a prototype platform. This platform needs to be built according to specific requirements. The requirements are developed from the hypothesis and research questions. Requirements development follows concept exploration. The high level requirements are defined in Section 1.2.

The high level requirements provide direction to the dissertation and prototype. However there exists a broad range of possible solutions that can meet these requirements. The third step of this stage is the literature review. This review will examine different X-Engines and supporting infrastructure to build up an understanding of the current state of the art of GPU X-Engine technology. From this examination, informed decisions must be made as to which hardware and software is suitable for the prototype platform. The literature review is presented in Chapter 3.

The results of the literature review are then used to develop a more detailed set of requirements which specify the hardware and software to be used in the prototype. These detailed requirements are laid out in Section 4.1. The literature review and detailed requirements are developed

iteratively; reviewing the literature leads to more requirements being developed and these requirements guide which literature must be next reviewed. Chapter 3 and Section 4.1 are the final results of this iterative process.

At the end of this stage, it should be very clear exactly what is required from the prototype and what technologies are to be used to build it.

2.2. Stage 2: System Design

In stage 2, the GPU X-Engine prototype is designed and assembled according to the detailed requirements. Figure 2.3 describes the logical flow of stage 2.

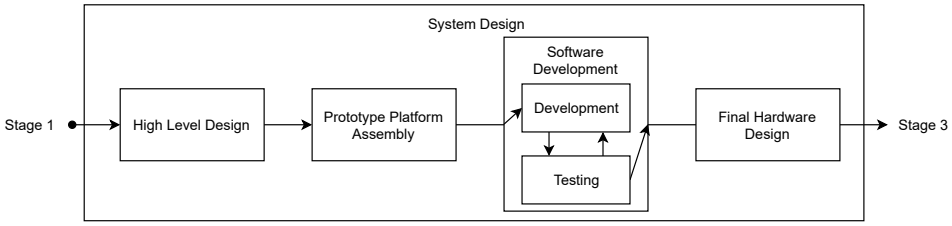


Figure 2.3: Diagram showing the progression of the system design stage.

The first step in this stage is a high level design that takes place in Section 4.2 and Section 4.3. This is where the high level architecture of the system is determined. This high level design takes the constraints and tools specified in the requirements and determines how to link all this together into a single cohesive system. Both hardware and software need to be considered in the step of this stage. The high level design does not require any development time or procurement, but small mistakes made in this stage can severely compromise the system further on in development with time and cost implications. As such this high level design needs to be undertaken thoroughly.

Once the high level architecture is determined, detailed designs for both hardware and software are required. The following has to be considered when designing the system:

1. Hardware is expensive to purchase and time consuming to make changes to once assembled.
2. Software is more forgiving to develop and can be modified easily. It can easily be ported from one hardware system to another.
3. It is difficult to determine where a bottleneck will occur that will prevent the system meeting its requirements without doing some level of experimentation.

Due to the above three points, it was decided that an initial prototype hardware platform should be assembled from available hardware before detailed hardware design would take place.

This prototype platform will be used to develop the software to be run on the final platform. Developing on this system allows for bottlenecks to be determined before expensive procurement takes place. This will then be used to inform the final hardware design, de-risking a large portion of the design.

Once the high level architecture and functional blocks are defined, the detailed software design strategy will take place. Software design follows an iterative approach. A software algorithm is coded and then tested. If it fails to perform as expected it is tweaked or rewritten. This approach is suitable for software as changing functional blocks of code is a simple and cheap process. The iterative software design method is a standard approach when following AGILE software development methodologies[8]. This iterative design is documented in Section 4.4, 4.5, 4.6, and 4.7. All software will be stored in a publicly accessible GitHub repository[1].

With the required system software running on the preliminary prototype platform, the performance of the system can be evaluated. From these results, the final prototype that will satisfy all the requirements can be designed. This design is presented in Section 4.8.

This stage will be complete when the final prototype has been assembled.

2.3. Stage 3: System Analysis

By stage three, the final prototype system should exist. This stage will focus on analysing the system; ensuring that the prototype meets the requirements, the research questions are answered and the hypothesis accepted or rejected. The logical flow of this stage is shown in Figure 2.4.

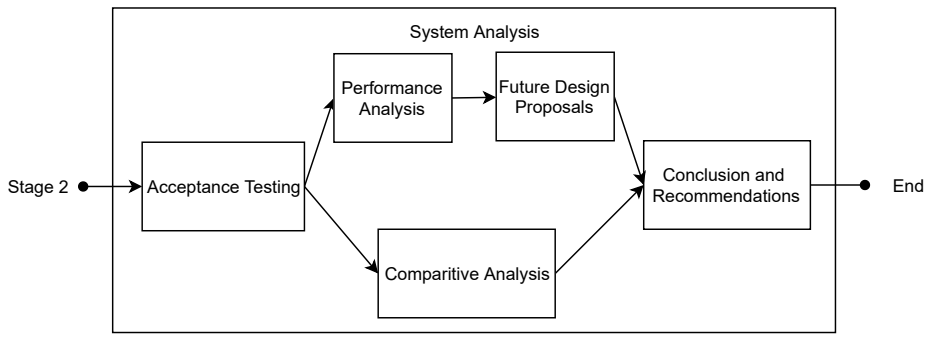


Figure 2.4: Diagram showing the progression of the system analysis stage.

The first step of this stage involves verifying that the final prototype GPU X-Engine meets all the high level requirements. This takes the form of acceptance tests. These tests are a series of binary tests where the system is measured against each requirement and either passes or fails

[1] https://github.com/gcallanan/GPU_XEngine

the test. These tests are presented in Chapter 5. No attempts to quantify the performance beyond the requirements will be made in this step.

Once the system is confirmed to be working, a serious analysis of the system will take place in order to assist with the answering of the research questions. This analysis follows two separate paths:

1. The performance of the system will be analysed. This analysis will determine bottlenecks, shortcomings and advantages in the system. This analysis will be used in answering **Research Question 1** and **Research Question 2** as these questions are open ended and cannot be answered with binary acceptance tests. This performance analysis takes place in Section 6.2. Once the performance has been analysed, future designs that can improve the system throughput, and reduce costs will be suggested. These designs will be discussed in Section 6.3
2. In order to answer **Research Question 3**, this GPU X-Engine system needs to be compared to the SKARAB X-Engine. This step involves testing the SKARAB and GPU systems under the same conditions and verifying that the output of these two systems is the same. Further comparisons will be performed in order to quantify the costs of the two systems. This comparison is done in Section 6.1

By this point in the project, all research questions should be answered. This will mean that sufficient information should be available to accept or reject the hypothesis. This is presented in the conclusions in Chapter 7.

Finally, the insights gained throughout this project will be used to predict where correlator technology is heading in the future and suggestions for future research will be made. This is all detailed in Section 7.2.

2.4. Methodology Conclusion

This chapter has explained the methodology of every stage in the life-cycle of this project.

The process to take these research questions and turn them into detailed system requirements after a thorough examination of the literature has been explained (Section 2.1).

The method followed to design the system and verify this design have also been put forward (Section 2.2).

Finally, the reader has also been presented with the steps taken to analyse the prototype to answer the research questions (Section 2.3). From these answers, the hypothesis can be accepted and rejected, and recommendations for future research can be made.

3. Literature Review and Theory Development

This chapter will explore the available literature and theory relevant to this project. As discussed in Section 2.1, this chapter must not only present the relevant information, it must also evaluate the existing research in order to determine the detailed requirements for the prototype. As such this chapter has the following objectives:

1. Establish the engineering and scientific context for this project.
2. Present the most prominent and relevant literature available on GPU X-Engines.
3. Compare and contrast the different solutions found in the literature.
4. Evaluate which solutions are most suitable for this thesis.

Figure 3.1 is a graphical representation of the logical outline of this review.

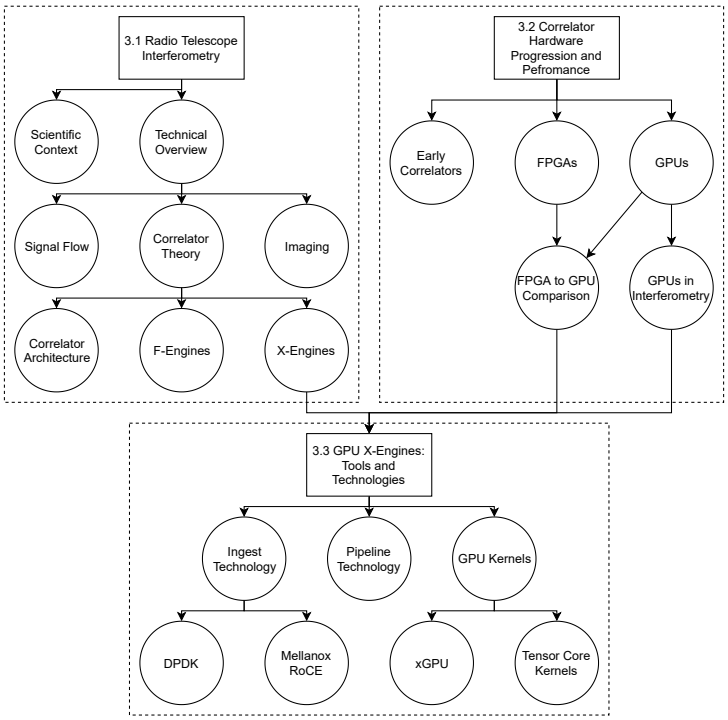


Figure 3.1: Diagram showing the logical flow of the Literature Review.

Section 3.1 will outline both the scientific context and technical theory relevant to radio telescope arrays. It will explain both where and how the X-Engine fits into the greater array system. This section will specifically focus on the MeerKAT radio telescope.

Section 3.2 will explore the change in correlator hardware trends that has led to the topic of this thesis. It will also discuss SARAO's reasons for moving away from FPGA technology. Furthermore, it will briefly survey some of the existing GPU X-Engines that are being used in the field.

Section 3.3 will examine the different tools that exist for building GPU correlators. This section will draw on the information presented in Section 3.1 and Section 3.2 to evaluate which of these tools are best suited for a MeerKAT specific GPU X-Engine implementation.

Finally Section 3.4 will collect and summarise all the research that has been presented in this review.

The radio astronomy instrumentation community is relatively small. Each telescope generally has a few engineers working on the digital back-end. Often funding and scheduling constraints limit the amount of time these engineers can dedicate to writing papers. The most up to date information is often best acquired by contacting the telescope engineers directly. These conversations will be mentioned where relevant.

3.1. Radio Telescope Interferometry

Radio astronomy is the study of celestial objects at radio frequencies. In order to observe the radio spectrum, astronomers require instruments known as radio telescopes.

South Africa has one of these telescopes located in the Karoo desert. This was originally a 7 dish telescope array called KAT7 (Karoo Array Telescope)[9], but this has been superseded by a 64 dish telescope called MeerKAT (Meer meaning 'more' in Afrikaans)[1]. The MeerKAT antennas are shown in Figure 3.2 below. The design, construction and operation of MeerKAT is managed by the South African Radio Astronomy Observatory (SARAO). At the time of writing, MeerKAT is one of the most sensitive L-band radio telescope in the world[10]. In the future, MeerKAT will form the centre of the multi-national next generation radio telescope known as the Square Kilometre Array (SKA)[11].

This section will provide the reader with a summary of the science being performed by MeerKAT (Section 3.1.1). Once this scientific context has been established, Section 3.1.2 will present a technical overview of the digital back-end of radio telescopes. This will explain where the correlator fits in the MeerKAT system. Section 3.1.3 will discuss in detail the technical aspects of the MeerKAT correlator system with Section 3.1.4 providing MeerKAT specific X-Engine details.

Figure 3.2: Photograph of the MeerKAT antennas in the field.[1]

3.1.1. Scientific Context

MeerKAT has two Priority Group 1 Science projects. These projects occupy the majority of the telescopes time. The one project is known as LADUMA and the other is MeerKAT's contribution to the International Pulsar Timing array project.[12]

MeerKAT is also gathering data for a number of Priority Group 2 Science projects. These projects receive far less telescope time than the Group 1 projects.[12]

Astronomical Surveys and LADUMA

An astronomical survey is an image of the sky that does not have a specific object of interest. The lack of a specific target allows statistical analysis to be performed on the many celestial objects surveyed. Figure 3.3 is a visible light image of the Hubble Ultra-Deep Field Survey. This image was produced by the Advanced Camera for Surveys (ACS) mounted on the Hubble telescope. This camera was able to look at the visible light spectrum.[13]

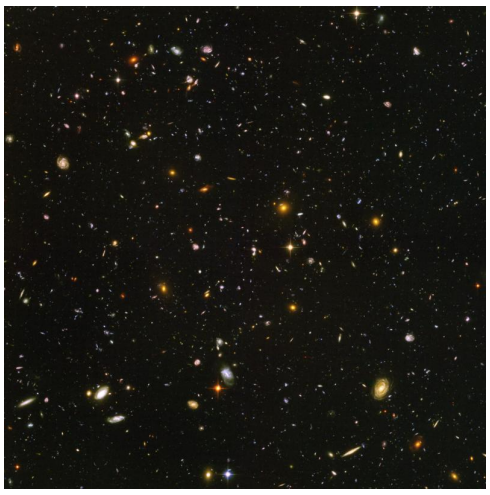


Figure 3.3: IMage from the Hubble Ultra Deep Field survey.[2]

[1]Image source: https://www.sarao.ac.za/wp-content/uploads/2019/01/2018-MeerKAT-3-1030x509.jpg

The MeerKAT telescope is performing its own survey called 'Looking at the distant universe with the MeerKAT Array'(LADUMA). LADUMA aims to detect neutral atomic hydrogen with the goal of improving human understanding of galaxy formation and assembly. Unlike the Hubble Ultra Deep Field, this observation takes place in the L-band part of the radio spectrum. [14]

In order to perform these surveys, MeerKAT needs to produce images similar to Figure 3.3. The process of turning raw telescope data into images is shown in Section 3.1.2.

Pulsars and the International Pulsar Timing array project

A pulsar is generally accepted to be a rapidly rotating neutron star that emits narrow beams of electromagnetic radiation from its polls. These beams rotate with the rotation of the star and due to this, a pulsar appears to be flickering when observed from a telescope. Pulsars can undergo a full rotation in the order of milliseconds - pulsars exhibiting this behaviour are known as millisecond pulsars. [15]

The International Pulsar Timing array project is a project that observes pulsars in the Northern and Southern hemispheres with the goal of detecting ultra-low frequency gravitational waves. This is done using a technique called pulsar timing where the expected frequency of a pulsar pulse is compared to its actual frequency. The deviations in expected and actual frequencies indicate that un-modelled effects are present. This can then be used to infer the presence of a gravitational wave. MeerKAT will contribute to this international project.[16]

MeerKAT Priority Group Two Science Projects

MeerKAT is scheduled to undertake eight Priority Group 2 science projects. These include:

1. MESMER - A project to search for carbon dioxide to investigate the roll of molecular hydrogen in the early universe.[17]
2. MHONGOOSE - An investigation into different types of galaxies and the roll of dark matter in their formation. [18]
3. ThunderKAT - A search for high power radio transients in order to better understand and describe physics that operates at energy levels that cannot be observed elsewhere. [19]

The rest of the projects are described in documentation available on the Square Kilometre Array South Africa (SKA SA) public website.[12]

[2]Image source: https://www.nasa.gov/sites/default/files/styles/side_image/public/thumbnails/image/hs-2004-07-a-large_web.jpg

3.1.2. Technical Overview

Most modern radio telescopes are no longer built as single large dishes, they are built as arrays of smaller dishes. As mentioned in Chapter 1, this is known as interferometry with an array of dishes being called an interferometer. MeerKAT is a 64 dish interferometer.

Arrays are used to increase the resolution of telescopes. For a single dish telescope, the resolution is given by $\frac{\lambda}{D}$ where D represents the antennas diameter and λ represents the wavelength of the received signal. An interferometer has a resolution of $\frac{\lambda}{B}$ where B represents the longest distance between antennas in the array. Smaller dishes are less sensitive than larger dishes, but it is more practical to construct an interferometer than a single dish when it comes to improving resolution of radio telescopes as separate antennas are cheaper and easier to manufacture.[20]

The process to take the separate antenna signals and produce a single image is well established in radio astronomy. This sub-section outlines the entire signal chain required to form images.

Signal Flow

Figure 3.4 shows the basic signal flow of an interferometer. N represents the total number of antennas.

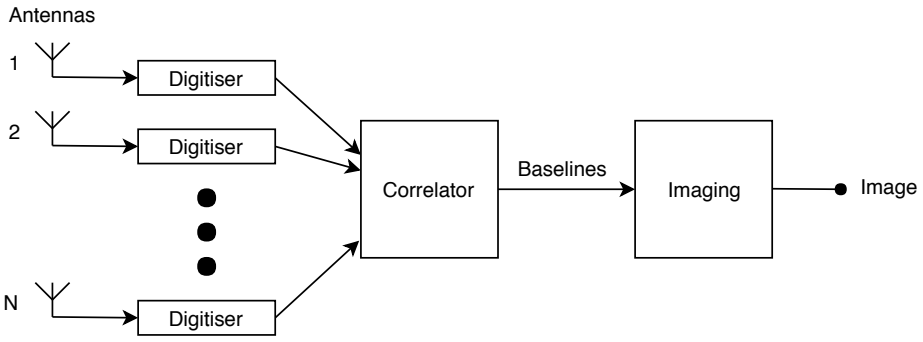


Figure 3.4: Diagram showing the signal flow through an interferometer from antennas to the final output image.

Analogue data from the antennas is converted to digital data at the digitisers. The MeerKAT digitisers are mounted directly on the antennas. Each MeerKAT antenna produces separate vertically and horizontally polarised streams. These streams are processed independently as if each were a separate antenna. These seperate polarisations means that MeerKAT should be looked at as if $N = 128$ where N is number of antennas.

This data is then fed into the correlator which correlates the signal from each antenna with every other antenna. The data produced from each antenna pair is known as a baseline. The correlator also performs a Fourier transform on each antenna signal. The correlator consists of

X-Engines and F-Engines. The F-Engines perform the Fourier transform and the X-Engines perform correlation. The correlator will be described in much more detail in Section 3.1.3. [20]

The baselines are then fed into the Imaging block which converts them into an image of the sky. Figure 3.5 shows an extract from the MeerKAT First Light image that was produced using this process[21]. Figure 3.5 shows 1% of the entire first light image. The object of focus is a massive black hole in the distant universe.

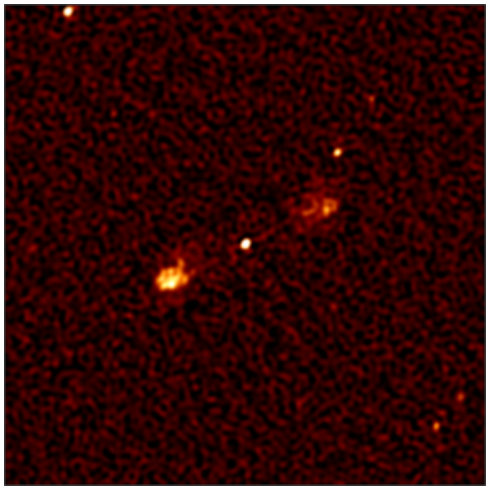


Figure 3.5: Extract from the MeerKAT First Light image showing a massive black hole in the distant universe.[21]

Image Formation

The function of the imaging block in Figure 3.4 will be discussed here as it provides valuable insight as to the purpose of the correlator baselines.

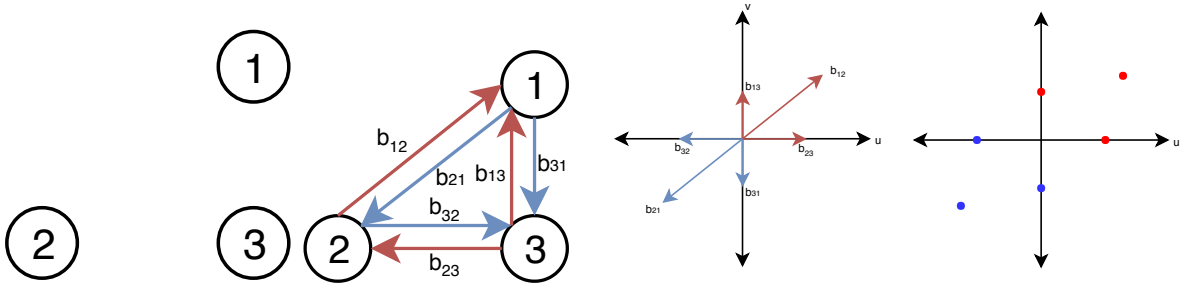
The baselines are used to populate the *uv* plane. The *uv* plane is, in one sense, a 2D Fourier transform of the sky. By performing an IFFT on the *uv* plane, the image of the sky can be generated.

The baselines need to be mapped to the *uv* plane. A baseline consists of amplitude and phase values for many frequency channels. A *uv* plane for each frequency channel will be produced. As such, the process below will need to be applied to each frequency channel individually.

Figure 3.6 shows the steps taken to generate the values on the *uv* plane. Figure 3.6a shows a hypothetical array. A baseline vector is defined as a vector representing the physical distance between two antennas. In Figure 3.6b, these vectors are plotted. There is a vector for each

[2]Image source: https://www.sarao.ac.za/wp-content/uploads/2016/07/2016_meerkat_fli_04-300x300.png

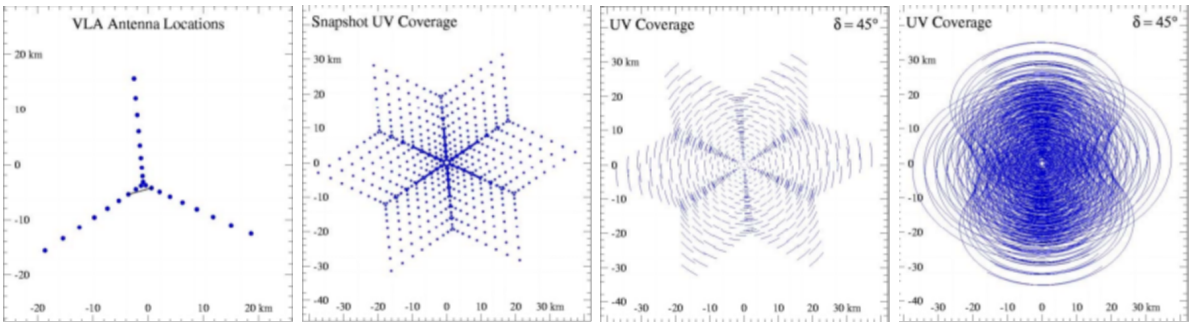
direction between two antennas. These vectors are then placed on the origin of the *uv* plane in Figure 3.6c. The points these vectors end on, are the points in the *uv* plane that the baseline value represents (Figure 3.6d). For this 3 element array, six points in the *uv* plane are known. This collection of known points is called the sampled *uv* plane. This is not the complete *uv* plane as most of the information is missing.



(a) Antenna positions *(b) Baseline vectors* *(c) Vectors to plane* *(d) Sampled uv plane*

Figure 3.6: Diagrams of the steps taken to generate a uv plane from the correlator baselines.

The sampled *uv* plane is relatively sparse. In order to find more points on the plane, long observations are performed where the rotation of the earth is leveraged in order to generate different baseline vectors. This is demonstrated in Figure 3.7 which shows the sampled *uv* plane for the Very Large Array (VLA) over time. Figure 3.7a shows the location of the 27 antennas in the VLA. Figure 3.7b shows the instantaneous point spreader function. Figure 3.7c and Figure 3.7d show how the *uv* coverage expands while observing for 1 and 12 hours respectively.[22]



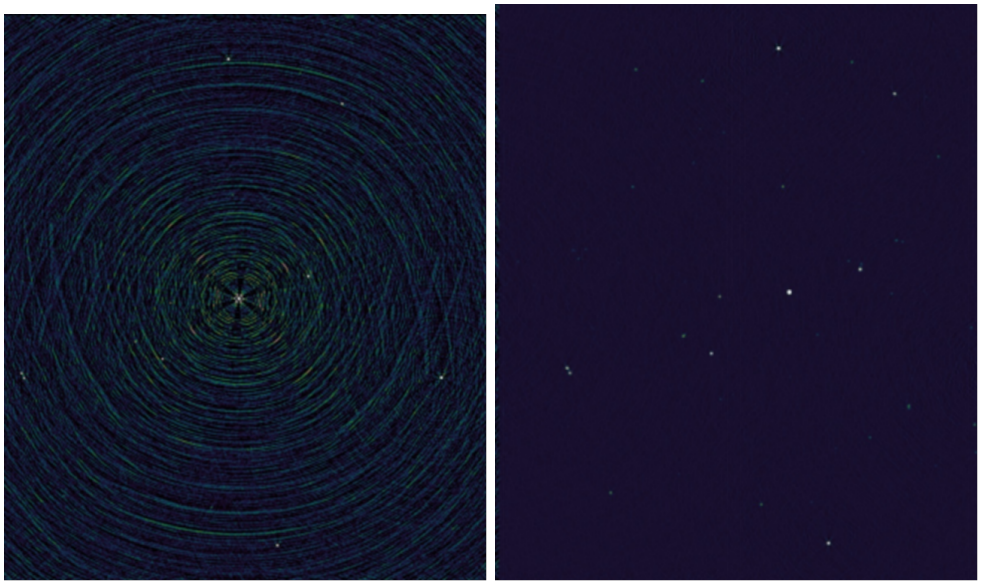
(a) Antenna positions *(b) Snapshot uv coverage* *(c) 1 Hour uv Coverage* *(d) 12 Hour uv Coverage*

Figure 3.7: Plots showing how the earths rotation can be exploited to fill the uv plane.[3]

This sampled *uv* plane in Figure 3.7 then undergoes an inverse 2D Fourier transform to produce an image of the sky. Because the sampled *uv* plane is not complete, a distorted image is produced

[3]Image source: http://www.hartrao.ac.za/synthesis_school/Miod_Array_Design.pdf

called a dirty image. A dirty imaged produced by the VLA is shown in Figure 3.8a. This dirty image can be thought of as the sky convolved with the point spreader function (PSF) of the array. The dirty image undergoes post-processing using the CLEAN algorithm which is an iterative process that attempts to estimate the values of the missing points in the *uv* plane. This 'CLEANed' image is shown in Figure 3.8b. The CLEAN algorithm is processing intensive and is not applied in real time on MeerKAT. The CLEAN algorithm will not be discussed here.[23]



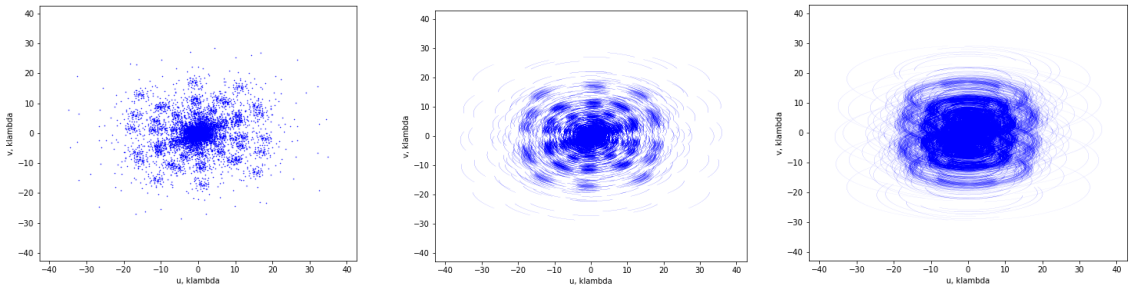
(a) Dirty image *(b) Clean image*

Figure 3.8: Images showing the raw dirty image of the sky and the clean image of the sky after the CLEAN algorithm is applied.[4]

The above process applies to a single frequency channel. However, a baseline is a vector of many frequency channels. This means that instead of a single *uv* plane, a vector of *uv* planes is produced with each *uv* plane belonging to a different frequency channel. Due to the expansion of the universe, the further away a source of light is, the more it is red-shifted and thus will show up in a different bin relative to a source of light with a similar frequency that is closer. This means that for some observations, the frequency bins give a sense of depth to the area of sky under observation. This is a very simple explanation but it is useful for conceptual purposes.

The MeerKAT *uv* coverage plot is shown in Figure 3.9. This plot was produced internally at SARAO.[24]

[4]Image source: https://github.com/ska-sa/tutorials/blob/77fc2b80f3dc7432b97b847c026723d0cfd64a36/3_Interferometry_Workshop/5_Imaging/imaging.pdf



(a) Instantaneous *(b) 2 hour observation* *(c) 8 hour observation*

Figure 3.9: Plots showing the filling in of the MeerKAT uv plane after different observation lengths.[24]

This section on imaging provides a few valuable insights into radio telescope array design:

1. The bandwidth of the telescope affects the depth of sky that can be observed due to red-shifting affects.
2. Baseline length affects resolution - longer baselines extend further out into the *uv* plane. This leads to an improved resolution after an inverse 2D Fourier transform is performed.
3. Antenna placement affects the shape and coverage of the sampled *uv* plane.

The above is a very brief overview of imaging. A simple summary of imaging can be found in [22] and [23]. A detailed look at the interferometry signal chain is explained in [20].

3.1.3. Correlator Theory

Section 3.1.2 explained where the correlator fits into the greater interferometer system. This section will unpack the theory behind correlators.

The definitive guide for correlator theory is found in the Interferometry and Synthesis in Radio Astronomy textbook by Thompson, Moran and Swenson[20]. However, more implementation specific information can be found by studying correlator designs such as [25][26] and [27].

A correlator correlates signals between antennas. It also performs a Fourier transform on the correlated signals. This can be described mathematically in Equation (3.1). In this equation, $a_x(t)$ represents a digitised antenna signal and $A_x(\omega)$ is its Fourier transform. This is a complex signal, as such $\overline{A_x(\omega)}$ represents the complex conjugate of this signal. In the MeerKAT case, $x, y \in [0, 63] \cap \mathbb{Z}$

$$\mathcal{F}[a_x(t) \otimes a_y(t)] = A_x(\omega)\overline{A_y(\omega)} \tag{3.1}$$

The input into the correlator is the digital antenna signals while the output is the Fourier

transformed correlation function for each antenna pair. The output is known as a baseline.

The complexity of correlator systems is introduced due to the massive data rates from each antenna. For MeerKAT, each antenna outputs data at 35 Gbps (17.5 Gbps per polarisation). With 64 antennas, this means that 2.24 Tb needs to be processed by the MeerKAT correlator each second. This requires well designed networks and computing systems that make efficient use of all resources.

As a baseline is required for each antenna pair, a correlator will produce $\frac{N(N+1)}{2}$ baselines where N is the number of antennas. This puts the correlation portion of an interferometer directly into the N^2 scaling regime. With antenna numbers expected to increase in future radio telescopes, the design of the correlator will become a greater challenge. This N^2 scaling results in a massive increase in data rates, however the correlator also accumulates data over a period of time. This accumulation serves to reduce data rates. Furthermore, by accumulating data, correlated signals beneath the noise floor rise to detectable levels.

Correlator Mathematical Description

Equation (3.1) mathematically described a correlator in its simplest form. This section will expand upon this equation in order to provide a more complete description of the function performed by the MeerKAT correlator.

Equation (3.1) is rewritten below in Equation (3.2) to show the formula for a single un-accumulated baseline. $\hat{B}_{x,y}(\omega)$ is equal to $\overline{\hat{B}_{y,x}(\omega)}$. x represents the antenna index.

$$\hat{B}_{x,y}(\omega) = A_x(\omega)\overline{A_y(\omega)} \tag{3.2}$$

The Fourier transform takes place over a certain window in time. In order to express these windows mathematically, $A_x(\omega)$ can be re-written as $A_x(\omega, n)$ where n represents the index of the window to be used.

Equation (3.3) shows the value of a baseline after accumulation. This takes place in the digital domain and as such is written as a summation. The summation takes place over M windows.

$$B_{x,y}(\omega) = \sum_{n=0}^{M} A_x(\omega, n)\overline{A_y(\omega, n)} \tag{3.3}$$

One final addition is that the antennas have horizontally(h) and vertically(v) polarised signals. Both polarisations of antenna x need to be correlated with both polarisations of antenna y. This

means that for two antennas, there are 4 correlation products. To account for these, $B_{x,y}(\omega)$ can be written as $B_{x,y,pp}(\omega)$ where pp represents the polarisation product. $A_n(\omega, n)$ will be written as $A_{n,p}(\omega, n)$ where p represents the polarisation. The set of equations completely describing all baseline products for a single antenna pair can be found in Equations (3.4), (3.5), (3.6) and (3.7).

$$B_{x,y,hh}(\omega) = \sum_{n=0}^{M} A_{x,h}(\omega, n)\overline{A_{y,h}(\omega, n)} \tag{3.4}$$

$$B_{x,y,hv}(\omega) = \sum_{n=0}^{M} A_{x,h}(\omega, n)\overline{A_{y,v}(\omega, n)} \tag{3.5}$$

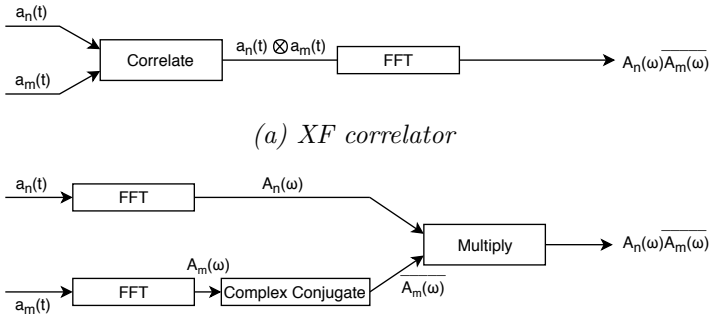
$$B_{x,y,vh}(\omega) = \sum_{n=0}^{M} A_{x,v}(\omega, n)\overline{A_{y,h}(\omega, n)} \tag{3.6}$$

$$B_{x,y,vv}(\omega) = \sum_{n=0}^{M} A_{x,v}(\omega, n)\overline{A_{y,v}(\omega, n)} \tag{3.7}$$

These equations provide a mathematical description of the MeerKAT correlator. The physical realisation of the correlator will be described in the subsequent sections.

FX and XF Correlators

Equation (3.1) gives insight into the two distinctive types of correlators that are commonly implemented. The first type is known as an XF correlator and is shown in Figure 3.10a. The XF correlator, takes in two antenna signals, performs conventional correlation and then performs a Fourier transform on the correlated signal. The second type of correlator is known as an FX correlator as shown in Figure 3.10b. The FX correlator takes advantage of one of the properties of Fourier transforms: frequency domain convolution is a multiplication instead of an integral. In an FX correlator, the antenna signals are first Fourier transformed and then multiplied together to produce the baselines. Most modern architectures conform to the FX correlator style as the interconnect is generally simpler to implement for larger array sizes.[27][28]



(a) XF correlator

(b) FX correlator

Figure 3.10: Diagrams showing the different signal paths taken when using an XF versus FX correlator.

FX Correlator Architecture

The MeerKAT correlator has an FX architecture, as such only this architecture will be examined further. A simple diagram for an FX correlator is shown in Figure 3.11.

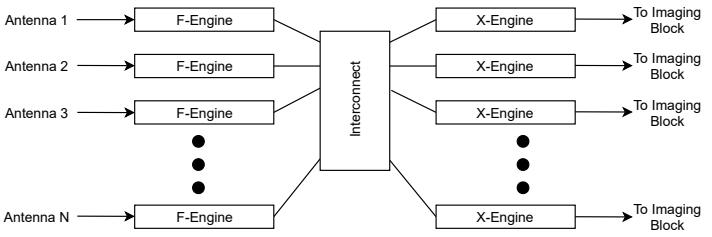
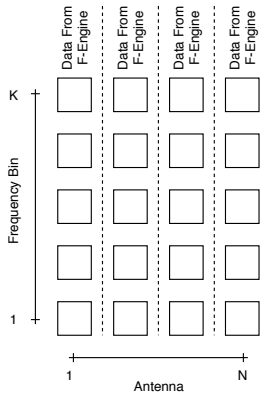


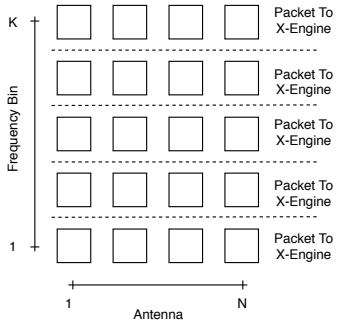
Figure 3.11: Diagram showing the arrangement of F-Engines and X-Engines in a modern FX correlator architecture.

Each antenna sends a signal to an F-Engine. The F-Engine performs the Fourier transform on the antenna signals.

Once the F-Engine has processed the antenna data, it then transmits it to the X-Engines over some form of interconnect. The F-Engines transmit the entire frequency band for a single antenna. Each X-Engine requires a unique portion of the frequency band from every F-Engine. This requires the data to go through a matrix transpose operation known as a corner turner. The corner turning operation takes place in two stages. The F-Engine transposes the spectrum data so that data bound for each X-Engine is grouped together contiguously. The interconnect then transmits the data and ensures that each X-Engine receives the correct potion of the band from each F-Engine. Figure 3.12a shows the data that is produced from each F-Engine before the transpose is applied and Figure 3.12b shows the data as it needs to be sent to each X-Engine. The corner turn is a computationally intensive operation. The interconnect is used to perform corner-turning and data transfer concurrently.



(a) F-Engine pre-transpose data out



(b) X-Engine data in

Figure 3.12: Figures showing how the corner turn operation logically organises data from the F-Engines to the X-Engines.

Once the X-Engine receives the data, it multiplies the baseline vectors of the different antennas together to perform correlation. In order to prevent the output data rate from expanding to unmanageable sizes, the X-Engine also accumulates data which vastly reduces this rate from the order of gigabytes per second to megabytes per second. The X-Engine then transmits data along the signal chain to the imaging part of the telescope.

Depending on the telescope implementation, each X- and F-Engine can be a separate process on a single server (for very small correlators) or a separate hardware platform (in the case of large correlators). [29]

For MeerKAT, the F-Engines and X-Engines are implemented on a custom reconfigurable FPGA board called SKARAB (shown in Figure 3.13)[30][31]. A single SKARAB board hosts two logical F-Engines, one for each polarisation of the same antenna. Four logical X-Engines are hosted on a single SKARAB board. There are four logical X engines for each F Engine. This means that 64 SKARAB boards are used as F-Engines and another 64 SKARAB boards are used as X-Engines. The interconnect is a 40 GbE folded Clos network. Each X-Engine produces the baselines for every antenna pair but only for a portion of the frequency channels. If required, the baselines are reassembled into a single spectrum further along in the signal chain[32].

Figure 3.13: SKARAB FPGA platform with Virtex 7 FPGA[5]

F-Engine

The block diagram for the F-Engine is shown in Figure 3.14. The input is a digital signal from a single antenna. This signal is buffered to accumulate enough data for a Fourier transform window. It is then fed into a filter bank that performs windowing to reduce spectral leakage that occurs during a Fourier transform. The filtered signal is then Fourier transformed using an FFT. This filter bank and FFT is often combined into a single more resource efficient block called a Polyphase Filter Bank (PFB). The output from the PFB is then passed into the packetiser and corner turner block which buffers frequency domain data and splits it into collections of channels to be sent to multiple different X-Engines

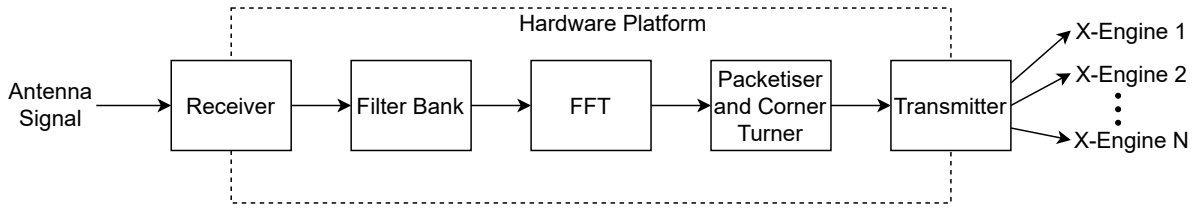


Figure 3.14: Block diagram for the MeerKAT F-Engine.[32]

F-Engines for high bandwidth radio astronomy interferometers are generally implemented on FPGA platforms, as it was traditionally too challenging to perform this high speed Fourier processing in real time on a GPU[29].

The MeerKAT F-Engine (implemented on a SKARAB platform) performs either 1 024, 4 096 or 32 768 bin FFTs. This is known as 1k, 4k and 32k mode respectively. The PFB size is 8 taps per frequency channel for the 1k and 4k modes and 4 taps for the 32k mode.

[5]Image source: https://github.com/ska-sa/skarab_docs/blob/master/peralex/CasperSkarab2017_v1_presentation.pdf

3.1.4. X-Engine

Section 3.1.3 explored the technical workings of FX correlators and their place in the greater telescope signal chain. This sub-section will perform an in-depth examination of the functioning X-Engines with a specific focus on the MeerKAT X-Engine.

The main reference material for Section 3.1.4 is based on the KAT7 and MeerKAT designs presented by Dr Jason Ryan Manley in his PHD thesis[25], an internal SARAO document on the Correlator design[32] and conversations with SARAO engineers directly involved with the design of the correlator.

General Operation

An X-Engine receives and processes data from the F-Engines. The block diagram for an X-Engine is shown in Figure 3.15.

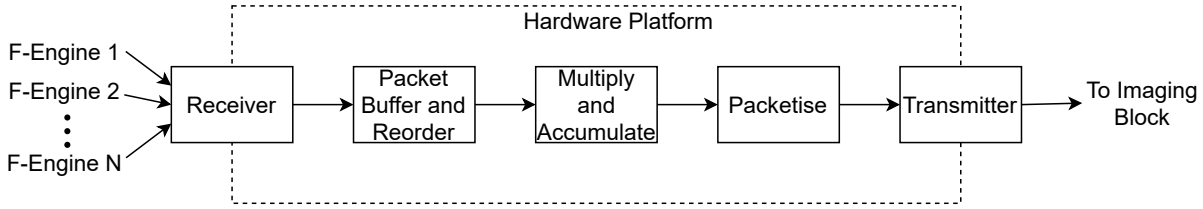


Figure 3.15: Block diagram for the MeerKAT X-Engine.[32]

The input data is received from multiple F-Engines. Each X-Engine receives a portion of the FFT bins from each F-Engine. For example, if an F-Engine performs a 4 096 point FFT and there are 256 X-Engines in the system, then the first X-Engine will receive frequency bins 0 to 15 from all antennas, the second X-Engine will receive frequency bins 16 to 31 from all antennas, etc.

Packets are buffered until the data from all antennas corresponding to the same point in time is received. Depending on the correlator interconnect that is used, the data may be received out of order and thus it needs to be reordered. If the X-Engine does not receive a packet, then a strategy needs to be decided upon on how to proceed. The simplest strategy would be to zero the data for that specific antenna, but this can lead to unwanted statistical properties occurring in the data. Another method would be to re-use the last known good data for that specific antenna.

The data is then fed into the multiply and accumulate block. This is the most computationally intense portion of the X-Engine. Every sample received from each F-Engine is multiplied with each sample received from all the other F-Engine on a per channel basis. Figure 3.16 illustrates the interaction of the signals from different antennas. As discussed before, for N inputs, there

are $\frac{N(N+1)}{2}$ correlation pairs. This is an $O(N^2)$ expansion. The multiplied data is accumulated over many thousands of samples to reduce the data rates. Each accumulated correlation pair is known as a baseline. These baselines are the output of the correlator.

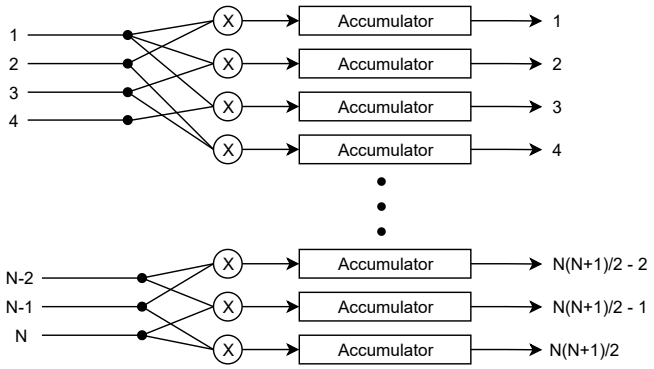


Figure 3.16: Diagram showing the implementation of the X-Engine multiply and accumulate operation.

The main challenge with X-Engine design is in how to process the massive amounts of data required by the N^2 scaling.

MeerKAT Specific Operation

The MeerKAT F-Engines transmit their data on the network interconnect. Each frequency group is transmitted on the same Ethernet multicast address by all the F-Engines. This multicast group will be referred to as a stream. Each stream transmits at 6.8 Gbps.

The SKARAB X-Engine block diagram is shown in Figure 3.17. A SKARAB X-Engine contains 4 logical X-Engine cores and a single 40 GbE receiver. The SKARAB platform will subscribe to 4 streams (1 for each X-Engine core), receiving a combined 27.2 Gbps over its 40 GbE port. The SKARAB firmware then directs the data from these 4 streams to the correct logical X-Engine. The logical X-Engine will then handle the multiplication and accumulation.

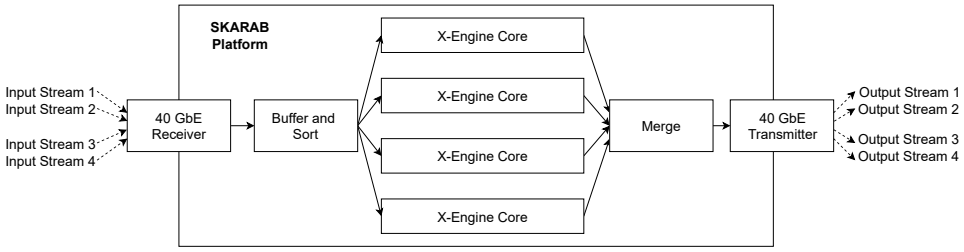


Figure 3.17: Diagram showing the mapping of four logical X-Engines to a single SKARAB host

The X-Engines accumulate data between 0.5s and 2s. The amount of time spent accumulating

reduces the output data rate from 27.2 Gbps to a few tens of Mbps to be transmitted back into the network on a new multicast address.

With $N = 64$, there are 2080 different correlation pairs in the single polarisation case. Due to the antennas being vertically and horizontally polarised there are 4 baselines per correlation pair. This makes for a total of 8 320 baselines.

Inputs and Output Data Format

Data is exchanged over the MeerKAT network by following the Streaming Protocol for the Exchange of Astronomy Data commonly referred to as SPEAD. This is a one way best effort streaming protocol. It is built on top of the UDP protocol. SPEAD groups logically similar sets of data together into a structure called a heap. A heap can be much larger than an ethernet packet, and as such a heap is constructed out of multiple ethernet packets. A SPEAD receiver is responsible for receiving packets from the network and reassembling them into heaps. The receiver is also responsible for handling heaps with missing packets[33]. The MeerKAT software team has already developed software that can receive SPEAD data over 40 GbE which will be discussed in Section 3.3.1.

The data format for the heaps is defined in the MeerKAT CBF ICD[34]. The format of the X-Engine input heaps is shown in Figure 3.18. The output heap format is shown in Figure 3.19. For the 4k correlation mode in the tables below, $n\_chans = 4096$, $n\_xengs = 256$ (4 logical X-Engines on 64 boards), $xeng\_acc\_len = 256$ and $n\_bls = 8320$. From this point on in the dissertation, only the 4k mode will be considered as only one mode is required to prove the concept. The X-Engine input data will be described more fully in the paragraph below. For brevity, the X-Engine output data format will not be discussed further.

SPEAD Item ID	SPEAD Item Name	Format	Shape	Description
Hardware Heap				
	heap ID	('u', spead.ADDRSIZE)	1	All packets from an F-engine destined for the same X-engine with the same timestamp have the same unique heap ID.
	heap size	('u', spead.ADDRSIZE)	1	Used to track how much data to expect. This will be determined by the number of x-engines and number of frequency channels.
	heap offset	('u', spead.ADDRSIZE)	1	The heap offset will be used to position the current frequency data in relation to others for the current SPEAD heap.
	payload size	('u', spead.ADDRSIZE)	1	Helps to keep track of amount of data received.
0x1600	timestamp	('u', spead.ADDRSIZE)	1	A number to be scaled by an appropriate scale factor, provided as a KATCP sensor, to get the number of Unix seconds since epoch of the first time sample used to generate data in the current SPEAD heap. Consult CAM ICD [10] for the appropriate sensor.
0x4101	feng_id	('u', spead.ADDRSIZE)	1	Uniquely identifies the F-engine source for the data. A sensor can be consulted to determine the mapping of F-engine to antenna input.
0x4103	frequency	('u', spead.ADDRSIZE)	1	Identifies the first channel in the band of frequencies in the SPEAD heap. Can be used to reconstruct the full spectrum.
0x4300	feng_raw	nominally('i', 8)	[n_chans/n_xengs, xeng_acc_len, 2, 2]	Channelised complex data from both polarisations of digitiser associated with F-engine. Real comes before imaginary and input 0 before input 1. A number of consecutive samples from each channel are in the same packet.

Figure 3.18: F-Engine output data format extracted from the MeerKAT CBF ICD.[34]

SPEAD Item ID	SPEAD Item Name	Format	Shape	Description
			Hardware Heap	
	heap ID	('u', spead.ADDRSIZE)	1	The baseline streams of each X-engine are output as independent SPEAD streams. The heap ID is unique for each stream and will be different to those of other X-engines. It is thus possible to subscribe to all baselines for a portion of the band.
	heap size	('u', spead.ADDRSIZE)	1	Used to track how much data to expect. This will be determined by the number of frequency channels and antennas.
	heap offset	('u', spead.ADDRSIZE)	1	The heap offset of this packet within the SPEAD stream.
	payload size	('u', spead.ADDRSIZE)	1	Helps to keep track of amount of data received.
0x4103	frequency	('u', spead.ADDRSIZE)	1	Identifies the first channel in the band of frequencies in the SPEAD heap. Can be used to reconstruct the full spectrum.
0x1600	timestamp	('u', spead.ADDRSIZE)	1	A number to be scaled by an appropriate scale factor, provided as a KATCP sensor, to get the number of Unix seconds since epoch of the first time sample used to generate data in the current SPEAD heap. Consult CAM ICD [10] for the appropriate sensor.
0x1800	xeng_raw	nominally('i', 32)	[n_chans/n_xengs, n_bls, 2]	Integrated Baseline Correlation Products; packed in an order described by the KATCP sensor *bls_ordering*. Real values are before imaginary. The bandwidth and centre frequencies of each sub-band are subject to the granularity offered by the X-engines.

Figure 3.19: X-Engine output data format extracted from the MeerKAT CBF ICD.[34]

Each X-Engine input sample is 32 bits wide. This 32-bit sample contains two complex numbers, one for each polarisation in a particular frequency bin. Each complex sample is 16 bits wide, containing an 8-bit real and an 8-bit imaginary number. According to the equation in Figure 3.18 and the information in the above paragraph, each 4k mode heaps contains 16384 (i.e. $\frac{4096}{256} \times 256 \times 2 \times 2$) bytes of data. This is equivalent to 16 KiB. The data packing is shown in Figure 3.20 below. Each heap contains 256 time samples for a single frequency channel and 16 frequency channels. The data is grouped first by time then by frequency channel. Each 256 grouping of samples from a single frequency channel is 1 KiB wide. Each heap contains multiple packets, and each frequency channel is a single packet. This means a 16 KiB heap will consists of 16 packets.

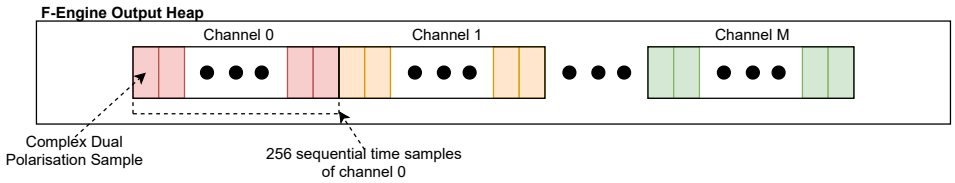


Figure 3.20: Diagram showing the data format of the F-Engine output heaps.

Every second, each F-Engine produces 1 632 heaps per logical X-Engine when in 4k mode. Each X-Engine will receive packets from every F-Engine. A single logical X-Engine will receive 104 492 heaps per second. This is a single stream. With 4 streams, every X-Engine server needs to receive 417 968 heaps per second. Each channel in the heap is transmitted as a 1 KiB packet. 16 packets are transmitted per heap which equates to 6 687 500 packets received per second per stream. Four streams of 6 687 500 packets per second where each packet is 1 KiB in size equals the measured input data rate of 28 Gbps per SKARAB X-Engine.

In 32k mode, the heap size increase by a factor of 8, but the number of heaps per F-Engine per

second decreases by a factor of 8. Furthermore, the packet size remains the same. This means that the number of packets each X-Engine receives remains the same in 1k, 4k and 32k modes. As such the ingest rate remains constant across modes.

3.1.5. Summary

Section 3.1 has contextualised the X-Engine. It has described the science behind radio astronomy and the signal chain required to do this science. The function of the correlator in the signal chain and of the X-Engine in the correlator has been explained.

From Section 3.1.4 an in-depth explanation of the SKARAB X-Engine has been provided including the expected input and output data formats as well as the computation that will be performed. This information will be used to inform the requirements for the GPU replacement system.

3.2. Correlator Hardware Progression and Performance

This section will examine the changes in trends in correlator hardware over time. Furthermore, Section 3.2.3 will give an overview of the most relevant GPU correlators operating at the time of writing.

3.2.1. Early of Correlator Hardware

Some of the earliest interferometers made use of Application Specific Integrated Circuits (ASICs) to perform correlation. One such instrument is the Very Large Array. The VLA is a 27 dish array that was completed in 1980 in New Mexico, USA[35][36]. It was perhaps the most well known interferometer of its time.

3.2.2. FPGA Technology

ASICs are expensive, inflexible, and have long development and production times. These disadvantages led to the radio astronomy community switching to Field Programmable Gate Array (FPGA) Technology.

The switch from ASICs to FPGAs in the ALMA correlator design is a prime example of this trend. ALMA is a 66 antenna array in Chile. ALMA produced its first science in 2011 using an FPGA correlator, but initial designs produced in 1997 showed the original plan was to use ASICs for correlation[37][38].

FPGAs solutions are generally cheaper, more flexible and quicker to design than ASICs. The main downside of an FPGA is that their clock rates are much lower than an equivalent ASIC implementation.

Basic FPGA Architecture

An FPGA is an integrated circuit designed to be configurable at a gate level after manufacture. To do this, an FPGA consists of many Configurable Logic Blocks (CLBs) connected together by an interconnect. The configuration of these logic blocks and the manner with which they are connected to each other can be changed by the designer and this configuration is what determines the function performed by the FPGA. Figure 3.21 shows a simplified FPGA layout, the red lines in this figure are part of the interconnect and the blue blocks are the CLBs.

For an examination of CLB architecture for the latest Xilinx (a major FPGA manufacturer) devices refer to [39]. For a white-paper examining the progression in technology leading to next generation FPGAs refer to [40].

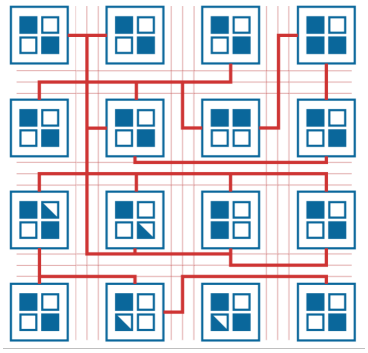


Figure 3.21: Diagram showing a simplified FPGA layout including CLBs and Interconnects.[6]

The configuration that is loaded onto an FPGA is typically called an image. The automatic process of building an image can take anywhere between half an hour to half a day for bigger designs. This depends on the compute resources used, design complexity and clock speeds that are set.

Unlike ASICs, FPGAs are flexible, with an image development cycle that spans days or weeks instead of years. They occupy a middle ground between ASIC and traditional software design.

FPGAs in Radio Astronomy

In radio astronomy, the CASPER collaboration was formed to develop hardware and open source firmware specifically to ease the development of X- and F-Engines. CASPER supports a number of hardware platforms including ROACH, ROACH2, SKARAB, and SNAP. CASPER supported FPGA hardware has been adopted by over 45 radio astronomy instruments world wide including

[6] Image Source: https://www.xilinx.com/support/documentation/white_papers/wp434-ultrascale-smarter-systems.pdf

industry leading telescopes such as MeerKAT, FAST and the Greenbank telescope[41]. CASPER provides a graphical interface via MATLAB Simulink for FPGA firmware development. All SKARAB firmware for the MeerKAT X- and F-Engines has been developed using CASPER tools.

Traditionally, FPGAs have required specialist tools to program, however the ease of use of the CASPER tools and pre-designed hardware has made FPGAs very accessible.

The CASPER collaboration is not the only user of FPGA hardware in radio astronomy. One such example is the Australian Square Kilometre Array Pathfinder (ASKAP). ASKAP has custom firmware on custom designed Redback-3 FPGA boards[42][43].

FPGAs are fairly ubiquitous in modern radio telescopes as they are able to handle the high data rates required by these instruments. Furthermore they generally interface quite easily with the ADCs required to digitise radio signals.

3.2.3. GPU Technology

While FPGAs have been firmly entrenched as the technology of choice in Radio Astronomy instrumentation, GPU technology has improved dramatically over the last decade. This technology has now reached a stage where it is starting to become a viable alternative to FPGAs in certain high throughput streaming applications.

Basic GPU Architecture

The Graphics Processing Unit (GPU) has in recent years become a preferred alternative to the CPU for high performance computing applications. Modern GPUs are able to perform an order of magnitude more operations than current CPUs. Nvidia's V100 server GPUs can process up to 14 TeraFLOPS.[44][45]

Figure 3.22 shows a diagram of a Nvidia GeForce 8800 GTX GPU (The high level architectures across Nvidia GPU platforms are relatively similar). This GPU consists of a collection of streaming processors (SP) within multiprocessor blocks. Each SP can be seen as a single thread. Modern GPUs have many thousands of these SPs with the Nvidia P100 GPU having 3 584 SPs.[46]

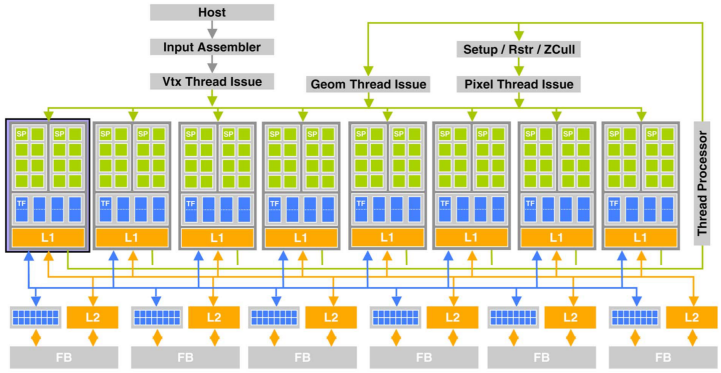


Figure 3.22: Diagram showing the system structure of the NVIDIA GeForce 8800 GTX GPU.[7]

Refer to [47] for an in-depth exploration of GPU architectures, performance and applications. Refer to [48] for a look at the evolution of GPU architectures.

GPUs in Radio Astronomy

There are a number of telescopes with FX architectures similar to MeerKAT that use GPU X-Engines.

The earliest of these include the MWA and LEDA arrays in Western Australia and New Mexico, USA respectively[26][49]. These telescopes are pictured in Figure 3.23a and Figure 3.23b.

More recently, the CHIME radio telescope in British Columbia, Canada was completed. This telescope is pictured in Figure 3.23c. At the time of writing, CHIME boasts the largest correlator in the world.[50]

At the same site as MeerKAT, the HERA telescope is under under construction. This telescope is an upgrade of the already existing PAPER telescope, both of which make use of GPU X-Engines[3].

[7] Image Source: https://escholarship.org/uc/item/0cv1p1nc

(a) MWA

(b) LEDA

(c) CHIME

Figure 3.23: Images of telescopes that have GPU X-Engines.[8]

Table 3.1 shows the key parameters between the 4 telescopes mentioned above. All these telescope have dual polarisation feeds increasing the number of baselines by a factor of 4. This has been factored into the calculations in the table. All four of these telescopes follow the approach of building many cheap antennas in contrast to MeerKAT which has fewer but more expensive and sensitive dishes. Due to the N^2 scaling of correlation, arrays with higher numbers of antennas are more likely to be in the compute bound problem domain if all other variables were controlled. GPUs seem to be more suitable in this problem domain as they have a high theoretical maximum computational capacity while having limited IO bus speeds.

Telescope	**Year Built**	**Number of Antennas**	**Bandwidth (MHz)**	**Number of Baselines**	**Number of Channels**
MWA Phase 1	2013	128	30.6	33024	3072
LEDA	2013	256	57.5	131584	2398
CHIME	2017	1024	400.0	2099200	1024
HERA	Under Construction	350	100.0	245700	1024

Table 3.1: Table of key performance parameters for different telescopes that have GPU correlators.

[8] MWA Image Source: http://www.mwatelescope.org/images/RotatorImages/Site/IMG_0881_small.jpg
LEDA Image Source: http://www.tauceti.caltech.edu/leda/wp-content/uploads/2015/07/lwa1.jpg
CHIME Image Source: https://chime-experiment.ca/images/gallery/Full-chime-2_nearlycomplete.jpg

Feasibility studies have been performed for the next generation SKA-Low telescope. This telescope is expected to consist of about 1 000 stations and over 250 000 frequency channels. It has been concluded that GPUs are a suitable platform for the X-Engines in the SKA-Low telescope.[51]

3.2.4. GPUs and FPGAs

Much research has gone into comparing FPGAs and GPUs[52][53]. The opinions on the issue can often be contentious. In general it is better to evaluate which platform most suitable an application by application basis. There are however a few generally accepted differences:

- Development Time - FPGA images take many hours to build whereas GPU programs take a few seconds or minutes to compile.
- Power - FPGAs generally consume less power than GPUs when performing similar functions.
- Cost - GPUs typically have a much lower initial cost than FPGAs. However, the running costs of FPGAs are lower due to their lower energy requirements.
- Throughput - FPGAs generally have much higher IO bandwidth than GPUs.
- Latency - GPUs have unpredictable latencies. FPGAs precisely control how many clock cycles an operation takes. Thus FPGAs are better suited for low latency environments.

At SARAO, the main disadvantage that has been experienced with the CASPER tools and FPGA development in general is the time taken to develop a well tested, bug free FPGA image. The MeerKAT FPGA designs take many hours to build (up to six hours on some of the bigger designs). The time between discovering and fixing a bug can often take many days. Another specific disadvantage with the CASPER tools is the lack of a testing framework.

SARAO hopes that moving development to GPUs would alleviate both of these problems. This move could also potentially decrease hardware purchasing costs with the drawback of increasing power consumption.

3.2.5. Summary

Section 3.2 has provided a summary of the hardware used in correlators. Section 3.2.1 described the early days of correlator hardware where ASICs were prominent, Section 3.2.2 described FPGA hardware and its place in the radio astronomy community and Section 3.2.3 described how GPUs have begun to rise in prominence.

Section 3.2.3 also described the different instruments that use GPU correlators, examining how they are firmly entrenched in the compute bound problem domain compared to MeerKAT which

has a lower compute to IO ratio.

Section 3.2.4 provided a general comparison between FPGAs and GPUs. It also listed the specific problems SARAO has encountered when it comes to developing systems on FPGAs. These problems include the time taken to fix a bug and the lack of a testing framework.

3.3. GPU X-Engines: Tools and Technologies

This section will investigate the tools and software libraries that are available to be integrated into the GPU X-Engine.

A GPU system is not a GPU operating independently. It is an entire server consisting of, in addition to the GPU, a Network Interface Card (NIC) to receive data from the network and a host CPU with a motherboard and RAM to manage the transfer of data from the NIC to the GPU. Figure 3.24 presents a simplified diagram of a GPU server.

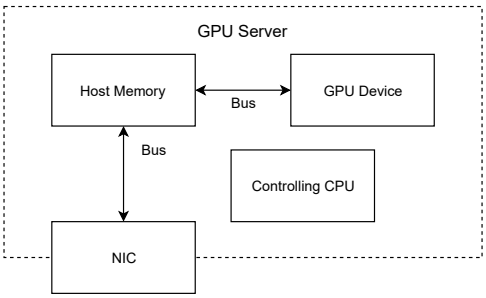


Figure 3.24: Simplified system diagram of a GPU server showing all relevant hardware components and data buses.

From Figure 3.24, it becomes clear that the tools needed to operate this server must accomplish three main functions. The first function is to offload data from the NIC into system memory, which will be discussed Section 3.3.1. Secondly a GPU kernel is needed which will handle the processing of the data on the GPU. These kernels will be examined in Section 3.3.2. The third function, analysed in Section 3.3.3, is a pipeline framework that manages the transfer of data between these separate devices. This framework will link all these tools together.

3.3.1. Ingest Technology

The X-Engine is required to connect to a 40 GbE network. The MeerKAT 40 GbE network has Mellanox switches as the backbone. All the MeerKAT servers have Mellanox ConnectX-5 40 GbE network cards installed[54].

Issues with the Linux Network Stack

When moving into the high speed networking realm (from 10 Gbps upwards), the traditional Linux networking stack fails to keep up with the network data rate. Traditionally, to process a packet from a network card, the Linux kernel copies the data into kernel space. It then needs to copy the data from kernel space to user space. This is signalled via an interrupt. The interrupt forces a context switch from user space to kernel space. The kernel manages the transfer of data from kernel memory to main memory. Once in user space the data is available to the user.

This redundant copy between user and kernel space and context switch mean that a large potion of the time the CPU spends performing a receive operation is actually spent context switching and performing the redundant memory transfer.

The DPDK library

The solutions to this require bypassing the kernel completely. One of the main libraries to do this is Intels Data Plane Development Kit (DPDK). The DPDK solution involves moving the driver to user space. This driver interacts with the NIC over UserIO (UIO), bypassing the kernel completely. This removes the need for a context switch and second memory copy. This driver uses a single CPU in polling mode to manage the transfers removing the need for an interrupt and reducing spinlock issues. Furthermore, DPDK pre-allocates memory into a ring buffer reducing allocation overhead[55]. CHIME makes use of the DPDK library to process data over 10 GbE at 6.4 Gbps[56].

Mellanox iberbs

Mellanox performs functions similar to DPDK with its ibverbs tools. This is implemented using the Mellanox OFED driver and ibverbs kernel bypass API. This implementation is similar to DPDK, except that ibverbs allows for either interrupt mode or poll mode. Interrupt mode means that the CPU waits for an IRQ from the NIC instead of continuously polling it. The OFED driver allows the standard linux network stack and the ibverbs stack to operate side by side as shown in Figure 3.25.

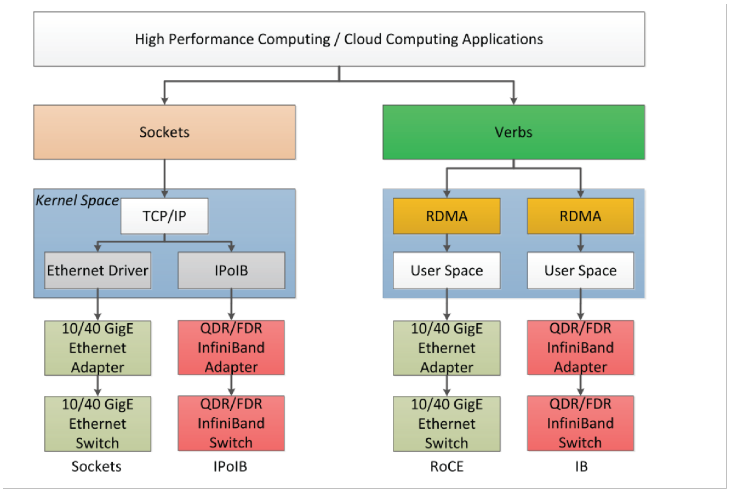


Figure 3.25: Diagram showing the Mellanox network protocol stack including the separation between the kernel socket stack and the ibverbs stack.[9]

Comparisons

The ibverbs API is used by the HERA and the Breakthrough-Listen teams in polling mode. Breakthrough-Listen is busy developing an ibverbs based receiver to subscribe to MeerKAT F-Engine streams. This receiver will be used in their servers that are attached to the MeerKAT network. The SARAO team has recently developed their own custom IRQ mode ibverbs library called SPEAD2 to receive SPEAD data heaps. The data rates received by the different implementations are shown in Table 3.2. The information in Table 3.2 on these ingest libraries is not widely documented and had to be obtained by contacting engineers on the various projects directly.

Implementation	**Interconnect**	**API**	**Mode**	**Speed (Gbps)**
CHIME	10 GbE	DPDK	Poll	6.4
HERA	40 GbE	ibverbs	Poll	36.0
Breakthrough-Listen	40 GbE	ibverbs	Poll	28.0
SARAO SPEAD2	40 GbE	ibverbs	Interrupt	TBD

Table 3.2: Table showing the ingest implementation and performance of different GPU X-Engines.

MeerKAT uses Mellanox networking equipment with the network receiver software implementing using ibverbs. In order to keep the technology in the telescope consistent, only an ibverbs solution was considered.

The Breakthrough-Listen and HERA ibverbs implementations will be able to process the required data rates. However the source code for these implementations is not open source or easily

[9]Image source: http://jitongchen.com/papers/HOTI12.pdf

accessible. After conversations with the creator of SPEAD2, it was determined that while the library has not been tested at the required rates, it is likely to be able process at this rate as long as each 6.8 Gbps stream is processed on a separate CPU core. This coupled with available code, existing documentation and the ability of the SPEAD2 development team to provide assistance led to the SPEAD2 ibverbs implementation to be the ingest library chosen for the prototype.

3.3.2. GPU Kernels

Development in GPU X-Engine kernels has been going on as far back as 2008[57]. Only a few of these kernels are in use in radio telescopes today. The main kernel in use is called xGPU. The CHIME team has produced the only alternative open source kernel. At the time of writing, some research has been released on new kernels that make use of Nvidia's tensor core technology. These three kernels will all be considered below.

The CHIME-x Kernel

The CHIME correlator team have developed and make use of a kernel called CHIME-x[10] that performs correlation on AMD GPUs[58]. This kernel manages to sustain GPU performance at 82% of maximum. It is very efficient but will not be considered further for a number of reasons:

1. The kernel only operates on 4-bit data where 8-bit operations are required by MeerKAT.
2. SARAO has no AMD GPUs for test and development.
3. The CHIME team is moving away from CHIME-x and is instead researching tensor core kernels on GPUs.

The xGPU Library

The most popular GPU library for X-Engine correlators is xGPU[11] which was developed in 2011. xGPU is an open source CUDA software library. It is being used in a number of major correlators around the world and is compatible with all CUDA capable Nvidia GPUs[59].

xGPU was originally designed to work with data formatted as 8-bit dual polarisation complex values that would be cast to a 32-bit float. The latest version of the library has the option to make use of Nvidia's DP4A instructions. These instructions allow for four 8-bit integers to be packed into a single 32-bit word and then multiplied and accumulated independently in a single clock cycle. This greatly increases xGPU's throughput.

[10] CHIME-x Source Code: https://github.com/radiocosmology/CHIME-x
[11] xGPU Source Code: https://github.com/GPU-correlators/xGPU

xGPU is a self contained software package that manages the transfer of data to GPU memory, execution of the multiplication and accumulation kernel, and the transfer of data back to system memory. All these transfers are pipelined and the pipeline is managed by the xGPU library. The user only needs to provide a pointer to correctly formatted input data in system memory and the software package will do the rest. This makes xGPU very easy to use. Furthermore it is very efficient, being able to sustain 79% of peak GPU performance in the original tests performed by the xGPU authors. These tests were performed on a Nvidia GTX 480 GPU with xGPU in floating point mode.

Figure 3.26 shows how the performance of xGPU increases as the number of antennas increase. xGPU makes better use of processing resources for higher number of antennas. However due to the N^2 nature of the problem, orders of magnitude less processing is required for a 64 antennas correlator compared to a 1024 antenna correlator.

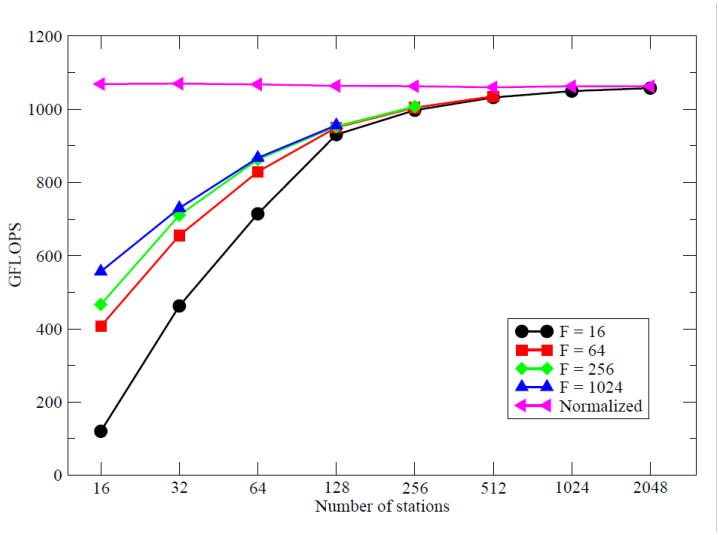


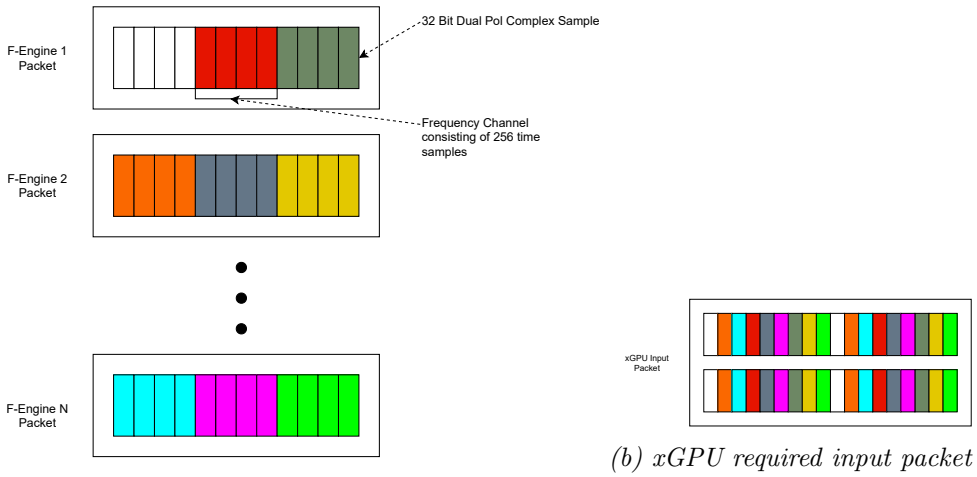
Figure 3.26: Plot of xGPU performance for different array sizes when run on a Nvidia GTX 480 GPU in floating point mode.[12]

xGPU does have two drawbacks. The first drawback is that xGPU does not allow the user to access the data while it is in GPU memory. This means that any other kernels that a user would want to execute on the same data would require the data to be re-transferred to the GPU. In IO limited environments this can lead to a reduction in throughput. In radio astronomy it is common to beamform F-Engine output data on GPUs. Using xGPU means that these two functions cannot be co-located on the same GPU. This co-location is not a requirement in this dissertation but should be considered in other designs.

The second drawback of xGPU is that the data order needs to be changed, with the most granular

[12]Data source: https://arxiv.org/pdf/1107.4264.pdf

ordering required to be the F-Engine ID, then frequency channel, then time. This reorder will be referred to as the transpose operation for the rest of this dissertation. The normal X-Engine input data format is shown in Figure 3.27a and the required format is shown inFigure 3.27b. Because each F-Engine packet is received as a whole, having to interleave these packets is a very fine grained, system memory bandwidth heavy operation. Due to xGPU requiring input data to be in system memory, this reorder has to be done on the CPU. Using CPUs at such high data rates will require significant processing power and development time.



(a) X-Engine input packet structure

(b) xGPU required input packet structure

Figure 3.27: Diagrams showing the difference between the format of antenna data as received from the F-Engines compared to that required by xGPU.

HERA, LEDA and MWA Phase 1 all make use of xGPU. They produce 4 bit complex data from their F-Engines but convert it to 8-bit complex data during the transpose operation.

Tensor Core Kernels

A potential new paradigm shifting technology in GPU correlation are the Tensor cores that Nvidia has added to their Turing architecture GPUs. These cores have specialised 4×4 matrix multiplication compute units. They have been designed to meet the growing demand for accelerated machine learning. They are theoretically 12 times faster than regular GPU matrix operations. Nvidia reports actual speedups of 9.3 for large matrix sizes.[60]

The correlation algorithm can be rewritten as a matrix operation to make it suitable for Tensor core processing.

The Dutch Institute for Radio Astronomy (ASTRON) has developed experimental kernels that perform the correlation operation[61]. They report these kernels to be 6 times faster than their current GPU kernels.

The ASTRON kernel is in the early stages of development. An early copy of the kernel was obtained for performance testing, but it is not yet available for use and distribution. Furthermore Tensor cores are experimental and there is no guarantee that they will be available in Nvidia's next generation of GPUs. This technology is thus high risk and should only be used if no other options are suitable.

Kernel Comparisons

These kernels were evaluated experimentally using a Nvidia RTX 2080 Ti GPU. 8-bit data was generated in system RAM, transferred to the GPU, and then correlated and accumulated. The correlation kernels were configured to operate on data sizes of 256 time samples and 64 channels. 1000 of these transfers from RAM and correlation kernels were run to simulate a single accumulation epoch. The time taken to perform this accumulation was measured and from this the data throughput was calculated. Figure 3.28 shows the throughput achieved from each kernel for different array sizes(each antenna has two polarisations) in the initial round of tests:

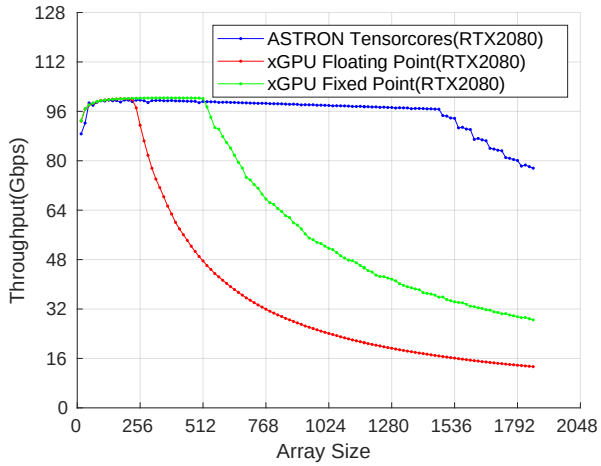


Figure 3.28: Plot showing the different GPU kernel throughput rates versus interferometer array size.

When the throughput is at 100 Gbps this indicates that the PCIe bus to the GPU has reached its maximum throughput rate and the problem is IO bound. The maximum theoretical throughput rate for a PCIe 3.0 x16 bus is 128 Gbps, but after some experimental testing it became clear that only 100 Gbps could be practically achieved.

The ASTRON kernel performs better when it comes to raw computational power, remaining IO bound for a larger array size compared to xGPU. For 64 antennas these kernels are both IO bound and throughput the same amount of data. They can process over 14 MeerKAT streams

per GPU. This is over 3 times the throughput achieved by a single SKARAB.

As both of these kernels are suitable, it was decided that xGPU should be used instead of the the ASTRON kernel because it can work on a broader range of Nvidia GPUs with a lower risk of obsolescence.

3.3.3. Pipeline Frameworks

There are a number of radio astronomy pipeline frameworks that have been developed to ease the writing of GPU pipelines. These include:

1. Hashpipe - A C based framework that has been used in the PAPER, HERA and Breakthrough-Listen pipelines[62]
2. Bifrost - A Python based framework that has been used for the LEDA project.[63]
3. Kotekan - A C++ framework developed for the CHIME correlator.[64]

After some examination, it became clear that none of these frameworks were immediately suitable for the prototype. Hashpipe and the SARAO SPEAD2 library could not be built into the same project, Kotekan had no supporting documentation and tutorials at the time of the original examination, and Bifrost is written in Python which would require significant further work to get working with the C based xGPU library.

All these frameworks share a few similar functions. They allow for the creation of separate stages that each operate as an individual thread. They enable the user to pass data from one thread to the next and they hide the locks and synchronisation required to do this from the user.

Intel has a library called Threaded Building Blocks[13] (TBB) that allows for the construction of graphs which contain a series of nodes connected with edges. Each node operates in a different thread (and is thus comparable to a stage). The TBB library is compatible with all the required technologies to be used in the prototype, is well documented and supported by a well known company. As such, while it was not designed specifically for radio astronomy pipelines, it is a suitable base library for building a pipeline to be used in the X-Engine prototype.

3.4. Literature Review Conclusion

Section 3.1 has put the X-Engine in its context, allowing the reader to identify where and how the X-Engine fits in the greater telescope system. Furthermore it has presented the expected input and output data rates, packet formats and accumulation times necessary to inform the requirements for the prototype.

[13]Intel TBB Documentation: https://software.intel.com/en-us/node/506212

Section 3.2 has explored the technology behind X-Engines. It has explained how the CASPER tools and hardware has led to FPGAs being the dominant platforms used in Radio astronomy. The lack of a suitable test framework and long debug times with FPGAs were highlighted as the main reasons to switch to GPU technology. Furthermore the GPU X-Engines that are used in the field were mentioned. These X-Engines have many more antennas than MeerKAT and operate in the compute bound problem domain, MeerKAT have a lower number of antennas and as such is more likely to be IO bound.

Section 3.3 examined the different tools and frameworks that can be used to develop a GPU X-Engine. It was determined that the SPEAD2 library built on top of the Mellanox ibverbs API would be suitable for handling the ingest data as the MeerKAT network data already conforms to the SPEAD protocol. Furthermore the xGPU kernel should be used for the GPU processing portion of the prototype as it is efficient, well tested, and relies on mature technology. It will perform as required in the IO limited domain that MeerKAT sits in. Finally the Intel TBB framework should be used to link all the different processing stages together as none of the radio astronomy specific frameworks can be used without extensive modification.

This review has provided the context of and justification for this dissertation. Furthermore it has examined all the tools used in GPU X-Engine design. This has paved the way for designing, building and analysing a suitable prototype platform to test the hypothesis.

4. System Design

The objective of this chapter is to take the detailed requirements and constraints, and develop a fully functional GPU X-Engine prototype. System design is the second stage of this dissertation. The process behind this stage is described in Section 2.2.

The input to this stage is the system's requirements and constraints. These have been discussed and mentioned in previous chapters, but they will be coalesced and formalised in Section 4.1.

The high level design of this system will be presented in Section 4.2. This involves examining the signal chain and determining the major sub-systems that will form part of this chain. Both the hardware and software components of each sub-system will be identified. The high level design of the system software, including UML diagrams, will be expanded upon in Section 4.3.

The design of these different sub-systems will then be detailed in Section 4.4, 4.5, 4.6 and 4.7. These sections will discuss the Pipeline Framework, Correlation stage, Ingest stage and Transpose stage sub-systems respectively.

Finally the design of the hardware platform that integrates all these sub-systems will be discussed in Section 4.8.

By the end of this chapter, the reader will have been presented with the high level overview of the system including how all the major sub-systems interact, the detailed operations of the major components, and the final hardware design. This section will fully describe the GPU X-Engine prototype.

4.1. Requirements Overview and Design Constraints

The requirements for this MeerKAT GPU X-Engine prototype have been gradually developed over the previous few chapters. This section will formalise these requirements into a single list. In a professional engineering environment, the system requirements document is exhaustive - often spanning hundreds of pages. For the purposes of this dissertation, only the most relevant requirements will be listed.

In this dissertation, Rx will stand for Requirement number x.

Section 1.2 described some high level requirements for this project. Section 3.1.4 described how the MeerKAT X-Engine fits into the greater correlator system. This informs the requirements on the system interfaces. Finally Section 3.3 identified the relevant tools to be used in the system. These tools to be used constrain the design of the system.

The following requirements arise from Section 1.2 and Section 3.1.4:

R1 The GPU X-Engine prototype shall ingest four multicast streams over a single 40 GbE port. Each multicast stream receives 1 KiB packets at just under 6.8 Gbps per stream for a combined data rate of 27.2 Gbps.

R2 The input and output data packets shall conform to the format specified in the MeerKAT CBF ICD. These formats are shown in Figure 3.18 and Figure 3.19 and both follow the SPEAD protocol.

R2.1 The GPU Prototype shall buffer and reorder the data according to packet SPEAD heap timestamps as the order the packets are received in is not guaranteed.

R3 The GPU X-Engine prototype shall perform correlation on the MeerKAT F-Engine data.

R3.1 Pairwise correlation shall be performed on all 64 antennas to produce 2 080 baselines.

R3.2 Full stokes correlation shall be performed - for each antenna pair, the vertical and horizontal polarisations must be correlated, producing 4 baselines per correlation pair.

R3.3 The correlator shall support accumulation times between 0.5 and 2 seconds.

R3.4 The GPU X-Engine shall have a mechanism to begin accumulation on a specific timestamp in order to synchronise with other X-Engines in the correlator.

R4 The GPU X-Engine shall process all the required data in real time.

R5 The GPU X-Engine shall have at least one 40 GbE port per 1U of server space.

From Section 3.3 the following constraints apply to the prototype design:

R6 Use the SPEAD2 library described in Section 3.3.1 to receive data from the network.

R6.1 Use a Mellanox NIC.

R6.2 Allocate one CPU core to the ingest portion of each input stream.

R7 Use the xGPU software library described in Section 3.3.2 to perform GPU correlation.

R7.1 In the CPU domain, perform a transpose on the input data so that it is in the format required by xGPU (shown in Figure 3.27).

R7.2 Use a Nvidia GPU for all GPU processing.

R8 Use the Intel TBB library described in Section 3.3.3 to link all parts of the system software together.

It is good practice to verify all system requirements. This is done in Chapter 5 with Table 5.3 matching each requirement to a specific verification test.

4.2. High Level Design: Signal and Data Flow

The first part of this design involved describing the flow of data through the system. The signal goes through a series of logical steps - each step will be referred to as a processing stage. The collection of stages that the signal flows through is known as a pipeline. As per R1 multiple multicast streams will be received by the X-Engine. As such multiple copies of the same pipeline should be created, with each pipeline processing a different multicast stream.

A pipeline stage either requires light or heavy processing resources. Light processing occurs when minimal manipulation of the signal is required or if the rate of data through a stage is low. Heavy processing occurs when the signal data needs to be copied or transformed at high data rates. Each stage identified will be classified as light or heavy according to these rules.

Each multicast stream will follow the same logical path. This logical path is shown in Figure 4.1. This figure will be explained in the paragraphs below.

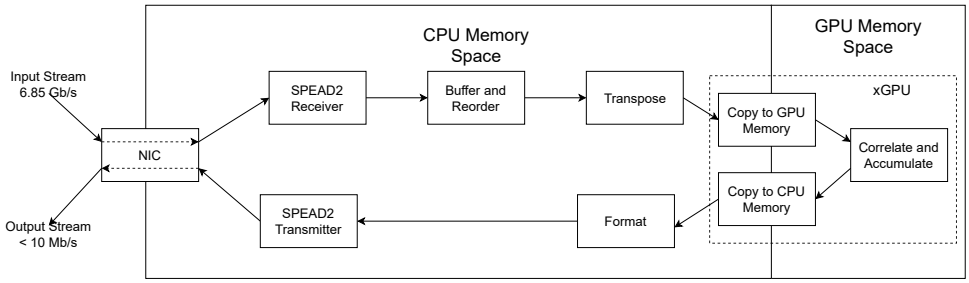


Figure 4.1: Diagram showing the high level logical flow of data through the GPU server.

The input data stream is received at a 40 GbE network port at 6.8 Gbps. It is formatted according to the SPEAD standard. This data must be copied from the network into system memory (R1 and R2). The first stage, referred to as the SPEAD2 Receiver Stage, will use the SPEAD2 library (R6) to receive data off a 40 GbE Mellanox NIC (R6.1). The copy of data from the NIC to system memory at high data rates makes this a processing heavy stage.

The data received from the NIC is not guaranteed to be in the correct order. The next stage known as the Buffer and Reorder stage manages the heaps copied into memory from the NIC. It waits for packets with the same timestamp to arrive from each antenna. It then collects this data logically and passes it on to the next stage in chronological order of timestamp satisfying R2.1. This stage also determines when a heap is considered as lost due to packets being dropped on the network. This stage does not actually manipulate or transform data. It simply checks

header and passes meta-data on to the next stage. As such it is a processing light stage.

The data is now received and in the correct order. In order to use xGPU, the data needs to be transposed as described in Section 3.3.2 (Satisfying R7.1). This transpose takes place at the full data rate and requires a 32-bit granularity reordering of data. As such it is processing heavy stage.

The next stage performs the actual correlation of the signal data (R3). This requires copying the data to the GPU, performing pairwise full-stokes correlation (R3.1 and R3.2) and then copying the data back to system memory. This correlation is performed by the xGPU library (R7) on a Nvidia GPU (R7.2). This data is accumulated for many thousands of samples (R3.3) significantly reducing the data rate. This stage synchronises accumulation according to the signal timestamps (R3.4). This stage combines a number of GPU functions and will be referred to as the GPUWrapper stage. This stage performs both a memory copy to the GPU and pairwise correlation. As such it is processing heavy.

The final two stages can also be combined into a single stage. They involve taking the data that has been correlated and accumulated and transferring it back on the network. This stage happens at significantly reduced data rates due to the data accumulation (A heap is transferred every 0.5 to 2 seconds). As such they are processing light. These stages will not be discussed in further detail as they do not present any significant design challenges.

The three major processing stages are the SPEAD Rx, Transpose and GPU Correlation stages. The data flow through these stages is mapped to the system hardware in Figure 4.2. This diagram shows the interconnects of a typical multi-socket server.

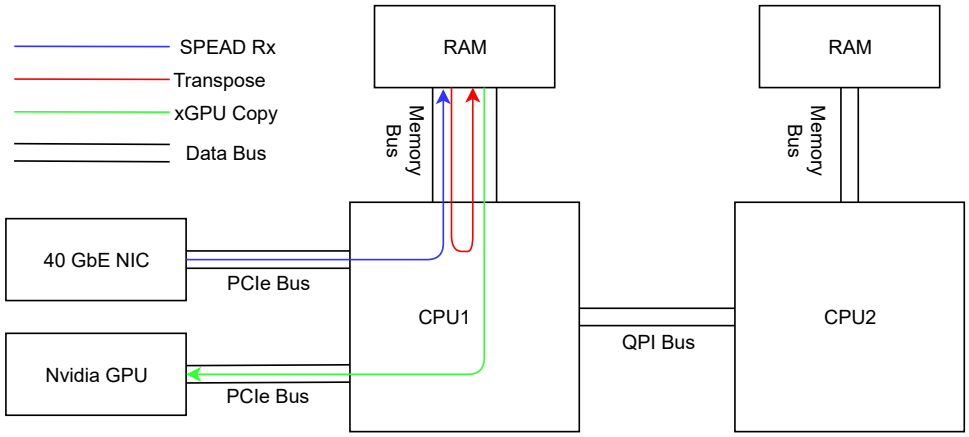


Figure 4.2: Diagram mapping the logical flow of data through the X-Engine to the GPU server hardware.

The transfer speeds through RAM and across the PCIe bus directly affect which hardware components will be used in this design. As such the three stages listed in Figure 4.2 are the major sub-systems of the GPU X-Engine prototype. An in-depth examination of these sub-systems will be undertaken in this chapter (Section 4.5, 4.6 and 4.7).

4.3. High Level Design: Software Structure

Section 4.2 identified and described the logical flow of data through the system. It defined a stage as a discrete function that operates on the signal. This section will translate the signal flow description into a software UML description. This software is designed according to standard Object Orientated Design principles, allowing for the design of the system to be broken up into discrete objects that are independent of each other. This section does not describe the actual framework that manages transferring data between stages as this framework is a major technical sub-system that will be examined in Section 4.4.

In the GPU X-Engine, a pipeline stage is a sub-system that takes in an input packet, processes this packet and then generates an output packet. This definition naturally leads to two base classes for this system. The one base class is the PipelineStage class that defines a single stage to process packets and the other base class is the PipelinePacket that defines the packets that will be passed between stages. Each of these base classes have sub-classes that inherit from them. The sub-classes take the generic functions specified in the base-class and map them to the functionality required in Section 4.2. The UML diagrams for these class hierarchies are shown in Figure 4.3 and Figure 4.4.

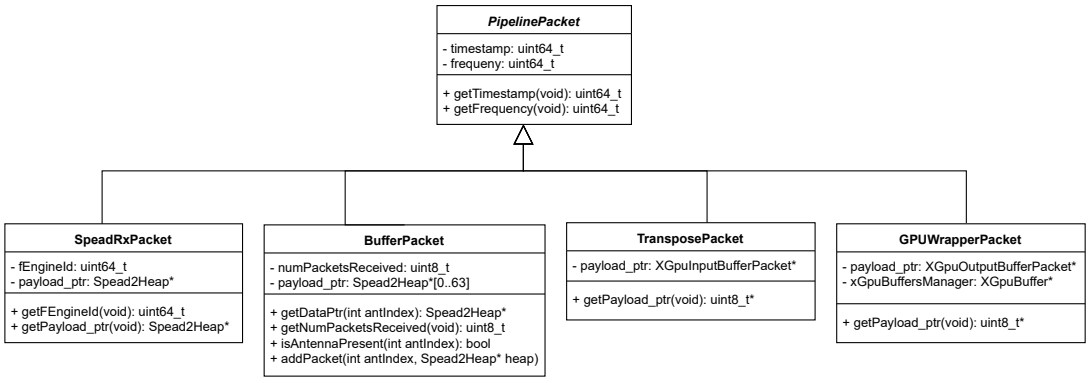


Figure 4.3: UML diagram of the pipeline packets used to transmit information between different pipeline stages in the GPU X-Engine.

Figure 4.3 describes the different pipeline packets used in this system. The base class, PipelinePackets, defines *frequency* and *timestamp* parameters as all packets have these values. The sub-classes define some member specific functions. All sub-classes will have a pointer

to a data payload (they do not contain the payload themselves). Each payload will be of a different data type and format. The SpeadRxPacket will point to a received Spead2 heap. The BufferPacket will then contain a pointer to 64 Spead2 heaps all with the same timestamp. This is then combined into a larger single packet by the transpose stage.

In Figure 4.3, all pointers are given in the standard C pointer syntax, in practice these will be implemented as a C++ shared_ptr<> as this is the preferred way to manage pointers in C++.

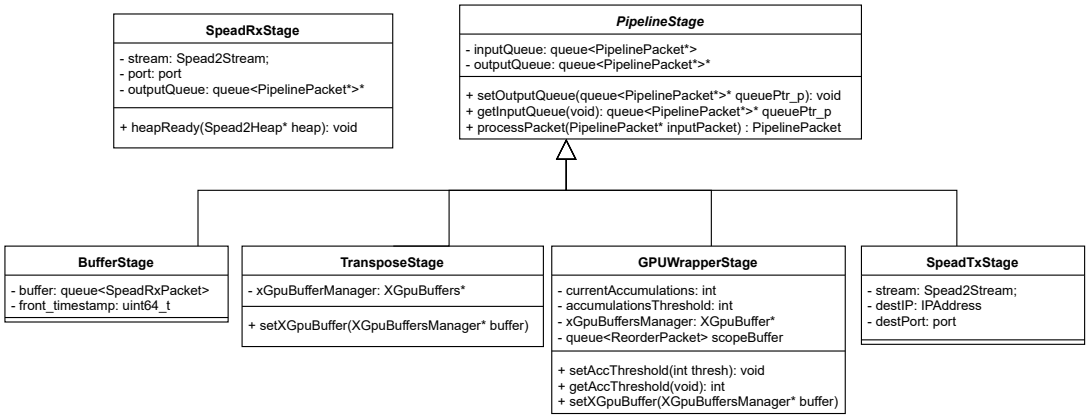


Figure 4.4: UML diagram of the pipeline stages in the GPU X-Engine.

Figure 4.4 shows the different stages in this system. The base class PipelineStage defines an input queue for the stage as well as a pointer to the output queue. These queues take packets of base type PipelinePacket. The framework that manages the linking of input queues in one stage to output queues of the preceding stage is described in the next section. The *processPacket()* function in a PipelineStage object performs the transformation required to take an input packet and produce an output packet. This *processPacket()* function is an abstract function in the base PipelineStage class and must be defined in each child PipelineStage class.

Each child class of PipelineStage has its own specific function and parameters. The BufferStage has a queue of SpeadRxPackets as it needs to store, sort and reorder these packets before it can transmit the data onwards. The GPUWrapper stage has methods for setting and monitoring the number of input packets received and accumulated. The SpeadTx stage has parameters indicating where the processed data needs to be transmitted on the network.

The transpose stage and the GPUWrapper stage both have pointers to a XGpuBufferManager object which manages the memory shared between them.

The SpeadRxStage does not inherit from the PipelineStage class. The Spead2 C++ receiver library has a unique implementation that launches its own threads to process data received by

the network card. As such it does not conform to the same structure as the other stages. When a Spead2 thread receives a heap, it calls the *heapReady()* function which will place a pointer to the received Spead2 heap in the SpeadRxPacket. This heap will then be put onto the output queue of the Spead2Rx stage (which is the input queue to the Buffer stage).

This UML diagram has not defined the behaviour of each stage. However it has defined a common interface that allows each sub-system to be developed independently while still being able to integrate to the rest of the GPU X-Engine.

4.4. Sub-System Design: Pipeline Framework

At this point, all the sub-systems have been identified and the interfaces between them have been defined. However, there is no actual sub-system yet in place to manage the stages and the passing of packets between these stages.

The Pipeline Framework is the sub-system that is responsible for managing this. In order to make use of modern multi-core CPUs, the framework needs to allow multiple different stages to process data in different threads simultaneously. The expected operation of the framework is shown in Figure 4.5. This figure shows multiple stages executing concurrently, furthermore it shows the packets being passed from one thread to another.

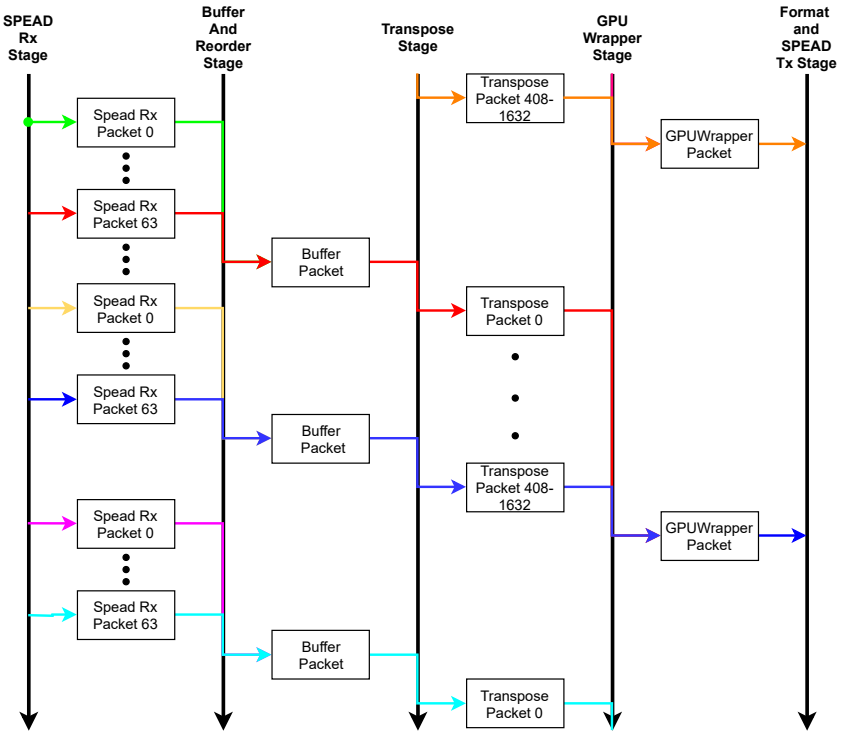


Figure 4.5: Diagram showing the interaction between pipeline packets and pipeline stages over time in the GPU X-Engine.

It is necessary to handle the passing of packets between each stage in a thread-safe manner because each stage operates in a different thread.

Table 4.1 shows the number of pipeline packets transferred between stages per second. The Buffer and Reorder stage will combine 64 SpeadRx input packets with the same timestamp and produce a single BufferPacket output (this stage does not copy any data, it performs its function by managing pointers to the different packets). The Transpose stage produces one output TransposePacket for every input BufferPacket. The GPUWrapper stages accumulates data between 0.5 to 2s which results in between 408 and 1 632 Transpose packets being accumulated to produce a single GPUWrapperPacket. This accumulation results in extremely low transfer rates in the Format and SpeadTx stages which led to them being combined into a single stage as the thread synchronisation overhead between them would incur more processing cost than combining them.

Stage	**Packets received per second per stream**	**Packets received per second per 4 streams**	**Memory Copy**	**Processing Intensive**
SPEAD Rx	52 224	208 896	Yes	Yes
Buffer and Reorder	52 224	208 896	No	No
Transpose	816	3 264	Yes	Yes
GPU Wrapper	816	3 264	Yes	No
Format and SPEAD Tx	0.5 - 2	2 - 4	Yes	No

Table 4.1: Table summarising the key characteristics of the different pipeline stages.

In Section 3.3.3 the TBB library was identified as suitable base for this pipeline framework (R8). The TBB library provides a mechanism for linking stages (which it refers to as nodes) together via edges. Each node operates in a thread and TBB links the input and output queues together when an edge is created. The packets between the threads are passed by reference not value, reducing communication overhead. TBB manages all synchronisation required to be thread safe. Furthermore TBB also determines when the *processPacket()* function in each thread is called.

A layer was built on top of the TBB framework in the PipelineStage base class. The process packet function was modified to take in multiple input PipelinePackets and produce multiple output PipelinePackets. This method made use of the polymorphism property of OOP to create a generalised and flexible framework that allowed for multiple stages to be created without having to write additional boilerplate code.

Thread Overhead and the Packet Armortiser

An issue that arose with the TBB framework is that each stage processes a single packet in the *processPacket()* function, and gets suspended before processing the next packet. When the pipeline was profiled it became clear that the thread overhead consumed a significant portion of available processing time. Figure 4.6a quantifies this overhead. The *sched_yield* function is the function that is called on completion of the *processPacket()* function and this call consumes 66% of all CPU time. *sched_yield* is meant to facilitate better time sharing between different TBB threads by giving control of the CPU other threads. It introduces a thread context switch and a system call which is extremely costly on a per packet level.

To reduce the number of times that this function is called, a packet armortiser was introduced in the PipelineStage base class. This armortiser makes each stage buffer a number of input PipelinePackets and then pass them to the next PipelineStage in a single transfer. The next stage then processes all these stages in a single *processPacket()* function. This armortiser is set to store 100 packets which reduces the amount of times *sched_yield* is called by 100. The pipeline was tested with this armortiser implemented and produces the results shown in Figure 4.6b. These results show that the armortiser reduces the pipeline overhead to an insignificant portion of processing time.

Both tests in Figure 4.6 were run for 30 seconds. When more than 30 seconds of CPU time is reported, it means that more than 1 CPU was processing the data in the different stages.

Function	Module	CPU Time
sched_yield	libc.so.6	79.592s
recvmmsg	libc.so.6	8.440s
Transpose::operator()	x_gpu_pipelined	7.904s
func@0x25020	libtbb.so.2	4.834s
syscall	libc.so.6	3.000s
[Others]		15.810s

*N/A is applied to non-summable metrics.

(a) No armortiser

Function	Module	CPU Time
recvmmsg	libc.so.6	6.670s
Transpose::operator()	x_gpu_pipelined	5.912s
std::ostream::flush	libstdc++.so.6	1.382s
sched_yield	libc.so.6	1.194s
__memmove_avx_unaligned_erms	libc.so.6	1.162s
[Others]		5.980s

*N/A is applied to non-summable metrics.

(b) Armortiser implemented

Figure 4.6: Extracts from the Intel VTune profiler showing the different amounts of CPU time required when running the pipeline for 30 seconds with and without packet amortising.

4.5. Sub-System Design: GPU Correlation

Correlation takes place in the GPUWrapper stage. This stage will fulfil R3. This involves synchronising the X-Engine with the rest of the system, handing data over to the xGPU library and managing the data being stored in RAM. The tasks performed by this stage are explained in Section 4.5.1, 4.5.2 and 4.5.3.

4.5.1. The xGPU Software Library

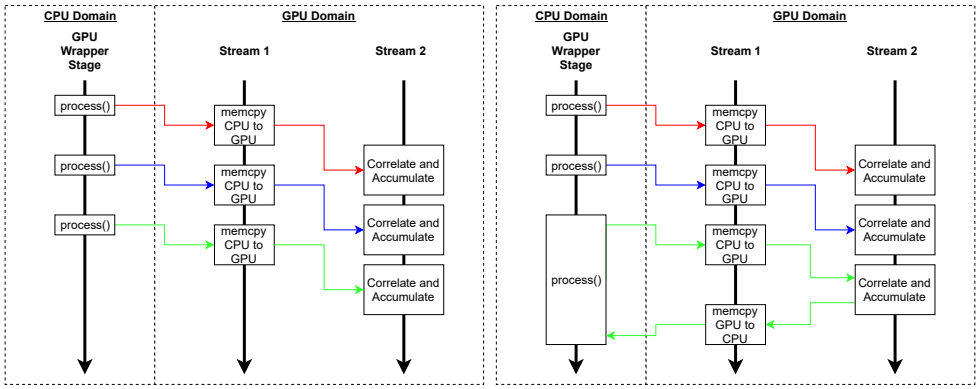
According to R7, the xGPU library must be used to perform correlation and accumulation. This library performs full stokes correlation on every antenna pair satisfying R3.1 and R3.2.

xGPU correlates the different antennas and then adds the output values to an accumulation buffer. The accumulation buffer will accumulate over what is known as an epoch. As per R3.3 this epoch is required to be between 0.5 and 2s long. This equates to between 408 to 1 632 TransposePacket payloads being accumulated per stream.

Once a TransposePacket is received in the GPUWrapper thread, the xGPU library function $xgpuCudaXengine()$ is called. This function launches two asynchronous calls on the GPU. The *memcpyCPUtoGPU* operation (To copy the packet to GPU memory) and the shared2x2 kernel (To correlate and accumulate the packet). These calls are queued on two separate CUDA streams (The GPU equivalent of stage). Once the *memcpyCPUtoGPU* operation finishes processing, it sends a signal to the stream handling the *shared2x2* kernel calls indicating that it can now start processing. These two streams can operate concurrently - a *memcpyCPUtoGPU* for one packet can be executing while a *shared2x2* kernel for another packet is running. The xGPU library uses CUDA's signalling mechanisms to prevent a *shared2x2* kernel and *memcpyCPUtoGPU* operation for the same packet from executing concurrently - this does not need to be managed by the GPUWrapper thread. The floating point version of xGPU was used as this requires a coarser transpose than fixed point mode and as such is expected to reduce the requirements on the transpose stage of the pipeline. Floating point mode is expected to be sufficient for MeerKATs problem size.

At the end of an epoch, $xgpuCudaXengine()$ is called with an argument indicating this it is the last accumulation in an epoch. This call launches the *memcpyCPUtoGPU* operation and *shared2x2* kernel. It will then launch an additional operation called *memcpyGPUtoCPU* which will wait for the other two streams to finish executing and then copy the contents of the accumulation buffer into CPU RAM.

Figure 4.7 demonstrates when different functions are executed in the CPU and GPU domain both during and at the end of an epoch.



(a) Middle of an epoch *(b) End of an epoch*

Figure 4.7: Diagrams showing the interaction between the CPU and GPU for the xGPU library both during and at the end of an epoch.

4.5.2. Managing Data Scope in the GPUWrapper Stage

From, Figure 4.7 it is clear that the xGPU library is non-blocking. Once the xGPU library has queued the kernels, it returns to the program flow. The kernels will be executed when they reach the front of their queues without any additional CPU intervention - this is all managed by the GPU. This operation is shown in Figure 4.7a. The kernel execution outlives the *processPacket()* function that launches it in all cases except for the last accumulation in an epoch. In this last accumulation $xgpuCudaXengine()$ blocks the CPU until the data is returned as shown in Figure 4.7b. This blocking occurs for less than a millisecond every second and it is not a source of any bottlenecks.

The data in RAM to be transferred to the GPU is managed by a C++ smart pointer. If this pointer goes out of scope, the data will be de-allocated by the smart pointers destructor. The asynchronous nature of xGPU means that if the pointer is released when the *processPacket()* function ends, there is no guarantee that the data has finished copying to GPU memory. This location could then be allocated to another pointer overwritten mid transfer. The pointers to the packets being transferred must be stored by the GPUWrapper class until the end of an epoch when they can all be released and de-allocated at the same time. For a maximum accumulation time of 2s for a single stream, 1.57 GB (13.53 Gb) of data will be held per epoch. This is not a significant amount considering modern servers easily have upwards of 100 GB of RAM.

4.5.3. Evaluating the throughput of the GPUWrapper Stage

The GPUWrapper stage works when tested experimentally on a single multicast stream. However, multiple multi-cast streams need to be run simultaneously and the density of requirements of 4 stream per 1U need to be met (R5). In order to determine the achievable density, the maximum compute of the GPU needed to be tested. The card that was tested was the Nvidia GTX 1080

Ti. This GPU is the most commonly used GPU by SARAO.

The utilisation was tested experimentally and the results were recorded in Table 4.2. The utilisation is very bursty, but averages a maximum of 10% per stream. Assuming linear scaling per stream, the Nvidia GTX 1080Ti can process 8 streams at 80% utilisation. This means that a single Nvidia GTX 1080Ti can be used in a 2U server box to process data coming in from two separate 40 GbE ports. This reduces the GPU costs by half. The Nvidia GTX 10 series has been superseded by the Nvidia GTX 16 and 20 series GPUs. Any GPU that needs to be used must have a processing power equal to or greater than the Nvidia GTX 1080Ti.

Number of Streams	**Input Data Rate(Gbps)**	**Minimum GPU Utilisation(%)**	**Maximum GPU Utilisation(%)**
1	7	6	12
2	14	13	21
3	21	24	31
4	28	26	39

Table 4.2: Table showing GPU utilisation per number of F-Engine input streams.

4.6. Sub-System Design: Ingest Stage

As discussed in Section 3.3.1 and according to R6 the SPEAD2 library must be used in conjunction with a 40 GbE Mellanox NIC for receiving F-Engine multicast streams. The SPEAD2 library manages receiving packets off the NIC, reassembling it into a heap (of the type specified in Figure 3.18) and handing this heap over to the pipeline to be processed.

The SPEAD2 library has three important aspects that must be understood. Firstly SPEAD2 creates a memory pool. This is a dedicated location in memory where heaps are assembled and stored. The second is a thread pool. This is a collection of dedicated processing threads that copy data from the NIC to the memory pool when an interrupt from the NIC is received. The final concept is the interface with the SPEAD2 library and the rest of the pipeline. A thread in the thread pool will asynchronously call a function called *heapReady(Spead2Heap * heap)* where **heap* is a pointer to the heap in the memory pool. This function must be overridden by the user. The user must ensure this function passes the heap pointer on to the next stage in the pipeline. For this application, the *heapReady(...)* function processes the heap meta data, wraps the heap payload in a SpeadRxPacket and then transmits it to the next stage in the pipeline.

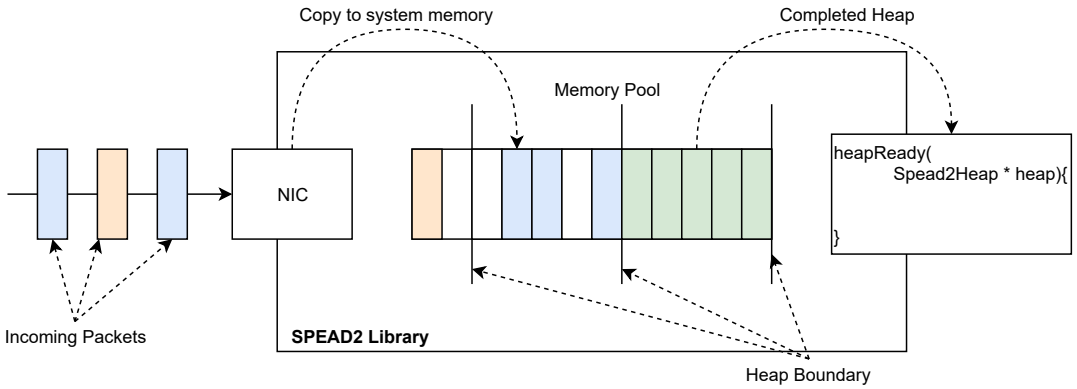


Figure 4.8: Diagram of a high level view of the SPEAD2 library.

Figure 4.8 illustrates the assembly of the SPEAD heaps in the SPEAD2 library. The packets belonging to a heap are copied into a contiguous section in the memory pool. Once all packets in the heap have been received a pointer to the heap is passed on to the rest of the pipeline.

Interfacing with the SPEAD2 library is not a significant challenge. The main challenge with integrating SPEAD2 is ensuring that the correct system environment is set up to enable fast packet capture. This requires configuring the NUMA boundaries (Section 4.6.1), IRQ settings (Section 4.6.2), and operating system buffer sizes(Section 4.6.3).

4.6.1. NUMA Boundaries

As illustrated in Figure 4.2, modern servers often have a multiple processor sockets. Each socket has its own dedicated memory bank. Each socket can also access another sockets memory over a QPI bus. This bus is many orders of magnitude slower than accessing its own dedicated memory bank. This is known as Non-Uniform Memory Access (NUMA). If NUMA boundaries are not managed, memory can be allocated across NUMA domains resulting in increased memory access times.

NUMACTL[1] and hwloc-bind[2] are two tools that can force a program to run on a designated CPU and in a specific memory bank.

NUMACTL was used to test the impact of the NUMA boundaries had on performance. Four input streams were transmitted to the GPU X-Engine at a combined rate of 27.2Gb/s. The different NUMA boundary settings were tested and the maximum rate the receiver was able to ingest packets was plotted in Figure 4.9 below. From these results it is clear that not managing NUMA boundaries will result in fewer heaps are reassembled.

[1]NUMACTL Linux man page: https://linux.die.net/man/8/numactl
[2]hwloc-bind Linux man page: https://linux.die.net/man/1/hwloc-bind

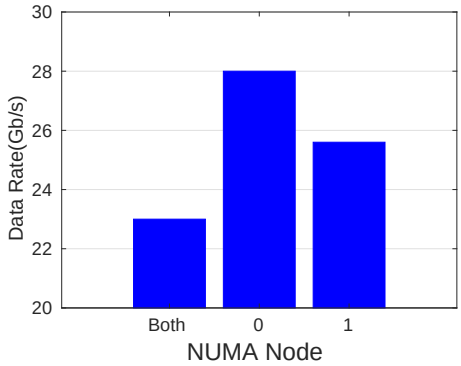


Figure 4.9: Plot of data rates the X-Engine pipeline is able to process data at when assigned to different NUMA nodes. The input data is fed to the system at a constant 28 Gbps.

When examining throughput rates in Figure 4.9 it becomes apparent that node 1 does not manage to process at the full data rate while node 0 does. It was initially thought this was due to node 0 not being attached to the same PCIe bus as the NIC. However, switching the network card to the PCIe bus on Node 1 did not boost performance while node 0 was still able to process at the full rate.

It was discovered that the IRQs of the system were mapped to node 1 and these IRQs were causing a significant slowdown on that node. The techniques used to manage these IRQs are discussed below.

4.6.2. IRQ Management

Certain linux distributions map all system hardware and software interrupts to a single NUMA node by default. At the full data rate this node can receive up to half a million interrupt requests(IRQs) per second from the NIC. By default each IRQ has significant processing overhead and interrupts the CPU threads managing the packet streams. This contention for the CPU prevented NUMA node 1 from processing as much data as required. In order to manage these IRQs, two solutions were employed:

1. Interrupt Coalescing - Linux provides tools to combine multiple interrupts from a network card into a single IRQ [3]. When enabling IRQ coalescing, a threshold for the number of frames to be received before generating an interrupts is set. This reduces the number of interrupts per second. Interrupt coalescing reduced the number of IRQs from over 500 000 per second to less than 2 000 per second for 4 streams.

2. Allocating IRQs to specific CPU cores - Linux provides tools for mapping IRQs to certain CPU cores[4]. For this system, the IRQs from the NICs were mapped to a single core per node and the rest of the threads were allocated to processing the streams. This prevents

[3] https://community.mellanox.com/s/article/understanding-interrupt-moderation
[4] https://community.mellanox.com/s/article/what-is-irq-affinity-x

the IRQs from interfering with the processing threads.

The NUMA boundary performance was tested with IRQ coalescing enabled and the results were plotted in Figure 4.10. Both node 0 and node 1 are able to process at the full data rate. This resolves the issue that was present in Figure 4.9.

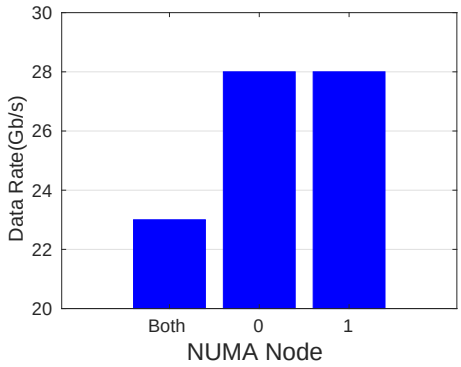


Figure 4.10: Plot of data rates the X-Engine pipeline is able to process data at when assigned to different NUMA nodes. IRQs are coalesced in this test. The input data is fed to the system at a constant 28 Gbps.

4.6.3. System Buffer Size

Linux sets very restrictive buffer sizes for UDP packets. These buffers are by default a few tens of kilobytes wide and overflow often at 40 GbE data rates. The size of these buffers was increased significantly (to at least 1 6MB) to stop packets being dropped due to overflow.

4.6.4. Hardware Throughput

By increasing buffer sizes, both NUMA nodes are capable of processing at the full data rate of 27.2 Gbps. This means that two separate NICs can be attached to the server and each process at the full data rate. As such, a 2U server with two 40 GbE NICs (or a single dual port NIC) would satisfy the density requirements laid out in R5. Furthermore by ingesting at 27.2 Gbps, R1 is satisfied.

4.7. Sub-System Design: Transpose

As per R7.1 and discussed in Section 3.3.2, a granular transpose operation needs to be performed in system memory to get the ingest data into the format xGPU required by xGPU. The design of this sub-system was not a simple task. Section 4.7.1 describes memory access patterns that needed to be considered for this transpose. The different implementations of this system

are discussed from Section 4.7.2 to Section 4.7.4. These implementations are compared in Section 4.7.5. The results from this are then used to inform the system DRAM requirements in Section 4.7.6.

4.7.1. Memory Access Patterns and Considerations

Server memory consists of RAM and multiple levels of cache located at gradually decreasing distances to the CPU. Each of these different levels of memory has different latencies and sizes. The structure of this cache is detailed in the Intel 64 and IA-32 Architectures Optimization Reference Manual[65] and summarised in Table 4.3 below:

Memory Type	**Location**	**Latency (Clock Cycles)**	**Size (KB)**
L1 Cache	On a single core	4	32
L2 Cache	On a single core	12	256
L3 Cache	Shared by multiple cores on a single chip	44	13 750
DDR4 RAM	DIMM slots on motherboard	>100	>1 000 000

Table 4.3: Table showing key performance metrics at different levels in the memory hierarchy.

Cache is both much faster and far smaller than RAM. The memory access latency of DDR4 RAM is highly system and access pattern dependant. A conservative estimate of 200 clock cycles has been used for the purposes of this design. However an access latency between 100 and 200 clock cycles is likely.

When data is read from RAM, a cache line is read. This cache line is 64 bytes wide - when a single 32-bit sample is loaded into cache, the next 15 samples are also loaded into cache. This reduces the access time of those subsequent samples significantly (to 4 cycles). A similar process occurs for a write. If the data is not in cache when it is accessed, it will be take the full access time to RAM to complete the instruction.

The interaction between RAM, Cache and the CPU was considered when designing the transpose sub-system.

4.7.2. Naive Implementation

The naive algorithm, loads a single input sample and then stores it in the correct location in main memory. This is done by sequentially iterating through the samples in each F-Engine packet. This algorithm was the first attempt at the problem where no effort was made into optimising the system performance.

Figure 4.11 shows the typical memory access pattern to main memory. Every time a sample is loaded from main memory, the next 15 samples are also loaded into L1 cache. Effectively, only 1 in 16 samples needs to be read from RAM. In all the transpose algorithms described, reading the data from RAM is always a cache efficient memory operation.

However, when in 4K mode, in the naive case, every store operation misses L1 cache. This means that every store takes the full >100 clock cycles to load data from RAM or the 44 cycles from L3 cache in the ideal case. This is the processing intensive part of the naive transpose algorithm.

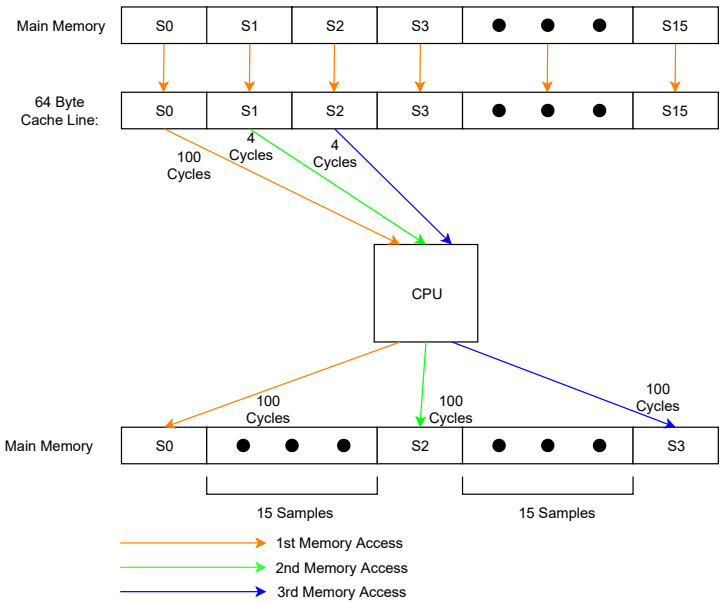


Figure 4.11: Diagram showing the memory access pattern when implementing a naive transpose operation.

Assuming the processor being used has 16 cores and each 32-bit transfer requires a write to L3 cache, then the clock rate of each core in order to transfer the data into RAM for all four streams needs to be:

$$Clock\ Rate = \frac{0.852\ GT/s}{16\ cores} \times 44\ Clock\ Cycles\ per\ transfer = 2.343\ GHz\ per\ processor$$

This calculation assumes the memory writes will dominate and ignores the memory reads. Furthermore it is a simplification as memory access patterns are very difficult to model. This implementation is achievable but requires significant CPU resources.

4.7.3. Block Transpose Implementation

An optimisation of this transpose is known as a block transpose. This method increases the number of writes to L1 and L2 cache while decreasing the number of writes to RAM.

Figure 4.12 demonstrates how this algorithm is implemented. Instead of loading a single F-Engine packet into cache at a time, N F-Engines packets are accessed in the same loop iteration and stored in cache memory. The same sample index of all N F-Engine packets is accessed sequentially and then written into RAM. These samples will be stored sequentially in the output memory location and as such are all stored in an L1/L2 cache line. Only when this cache line is full will a write to RAM occur. When N is too large multiple cache lines will be written per iteration which can reduce efficiency. As such N needs to be tuned experimentally.

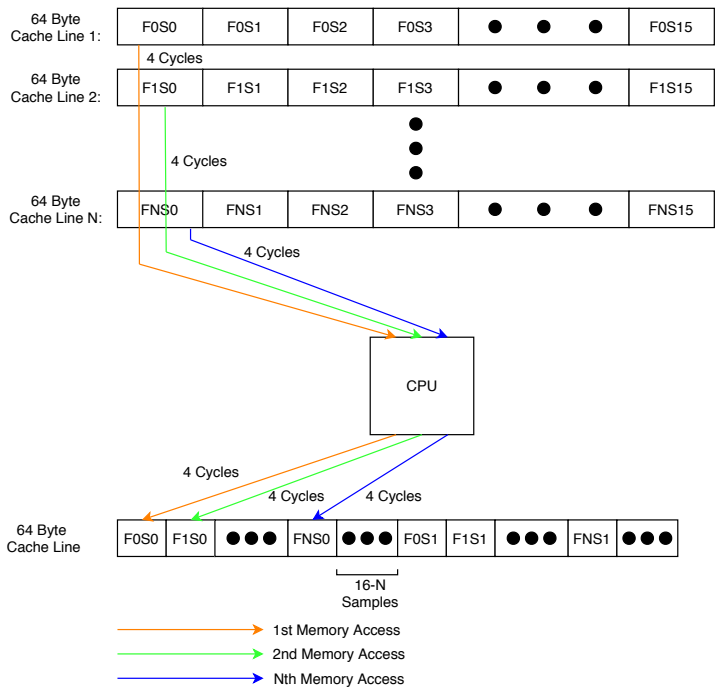


Figure 4.12: Diagram showing the memory access pattern when implementing a block transpose operation.

4.7.4. Intel SSE Implementation

The block transpose algorithm can be improved by making use of the Intel SSE instructions. This implementation is very similar to the tiling algorithm, except instead of writing the output data to a cache line, it is written to the SSE registers on the CPU. SSE instructions are specific instructions for operating on these registers. The SSE registers are referred to as the *xmm*, *ymm* and *zmm* registers. These registers are 128, 256 and 512 bits wide respectively. This corresponds to $N = 4, N = 8$ and $N = 16$ in the block transpose algorithm. These SSE registers are located on the CPU reducing write latency to a single clock cycle. On every Nth write the register will be full and is then written to RAM. SSE registers are explicitly managed unlike cache operations which need to be inferred.

4.7.5. Comparison of Different Implementations

The different transpose algorithms were all tested on a single stream. The measure of performance for these algorithms was CPU utilisation as this is the simplest metric to measure. Lower CPU utilisation means that the algorithm is able to complete the transpose with fewer clock cycles and is thus performing better. A CPU percentage greater than 100% means more than a single core is required to perform the transpose. The results for the different algorithms are shown in Figure 4.13. This figure shows both the Intel Intrinsics and Block tiling implementation. The naive implementation is plotted in the block tiling graph where $N = 1$.

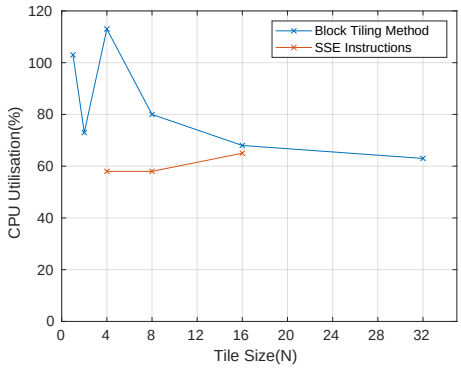


Figure 4.13: Plot showing the CPU utilisation for the different transpose algorithm implementations.

Figure 4.13 confirms that SSE instructions perform better than the tiling approach and use nearly half as many clock cycles as the naive implementation to perform the same approach.

For $N = 4$ there is a spike in the CPU utilisation of the block tiling algorithm. It is suspected that this is due to the input reorder data not being aligned to cache boundaries resulting in an additional fetch from main memory.

For the SSE instructions, when $N = 16$ the transpose operation uses more resources than for $N = 4$. The SSE instruction do not allow for writing data to the 512-bit register in a single write. Instead two 256-bit writes are required. This add extra overhead to $N = 16$ that is not present in the $N = 8$ SSE implementation.

The SSE transpose algorithm with $N = 4$ or $N = 8$ will be used to perform the transpose operation as it results in the required performance while consuming the fewest CPU clock cycles.

4.7.6. Memory Channels and Hardware Considerations

Each server supports a number of memory channels per socket. Populating more channels with RAM modules results in a greater memory bandwidth as more simultaneous reads and writes to RAM can take place. This means that more multicast streams can be transposed simultaneously. A number of configurations were tested to see if they could reorder the full number of streams. The results were recorded in Table 4.4.

Number of Channels	**Pass/Fail Result**			
	1 Stream	**2 Streams**	**3 Streams**	**4 Streams**
1	✓	✗	✗	✗
2	Not Testable			
3	✓	✓	✓	✗
4	✓	✓	✓	✓
5	Not Testable			
6	✓	✓	✓	✓

Table 4.4: Table showing the number of streams the GPU server is able to process when tested on different RAM channel configurations.

From Table 4.4 it was determined that 4 channels per socket was the optimal amount. 3 Channels is too few. 6 channels is acceptable but requires more RAM modules. The 4 channel test was performed with RAM clocked at 2400 MHz.

4.8. Integration and Final Prototype

At this stage of the design, the software pipeline has been fully developed and some of the hardware relevant to each pipeline stage has been specified. This section will determine the last few hardware components to be used in the design (Section 4.8.1) and integrate them into the final prototype system (Section 4.8.2). The motherboard and chassis of modern servers are often integrated - they cannot be purchased separately.

4.8.1. Hardware Components Specification

As per Section 3.3 a GPU streaming server consists of a GPU, NIC, Host Memory and CPU all connected by a motherboard.

Section 4.5.3 determined that a Nvidia GTX 1080Ti GPU should be used, R6.1 determined that a Mellanox ConnectX-5 40 GbE NIC should be used and Section 4.7.6 determined that 4 channels of RAM would be required with a minimum clock rate of 2400 MHz. The CPU and the motherboard are the only components that still need to be determined.

Examining the pipeline CPU utilisation revealed that each stream required two cores to process the required data rates (one core for the SPEAD2 receiver and the other for the transpose

operation). For 4 streams, 8 cores are required. Additional cores are required to handle the NIC IRQs (Section 4.6.2). The 10 core Intel Xeon Silver 4114T CPU[5] will be able to handle the required throughput. At the time of writing it is one of the cheapest Intel Scalable processors on the market that will be able to handle the processing requirements. Figure 4.15 shows the single-socket system architecture:

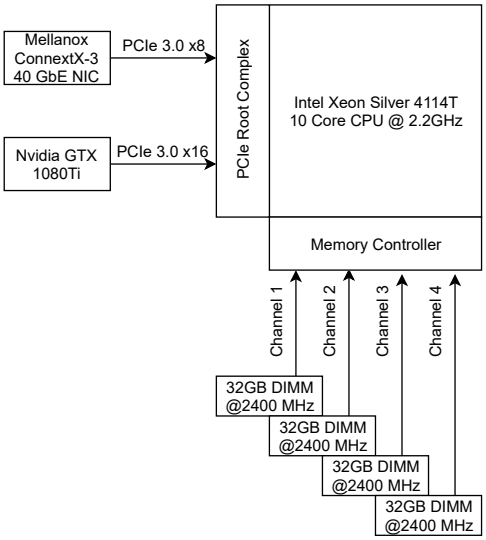


Figure 4.14: Diagram of the single-socket prototype GPU server architecture.

The motherboard and chassis still need to be determined. This takes place in Section 4.8.2 below.

4.8.2. Density Considerations

The single socket system is capable of processing 4 F-Engine streams just like the SKARAB X-Engine. As discussed in Section 4.5.3, the Nvidia GTX 1080 Ti is capable of processing 8 streams - the single socket system wastes half of the GPUs processing capacity. If the server design is changed to a dual socket system with one 40 GbE NIC per socket, then each socket can process 4 streams while sharing the same GPU. This dual socket system halves the number of power supplies, hard drives and other peripheral components required per stream reducing the system cost further. Additionally, a dual socket system can be 2U wide (due to R5) increasing the number of server chassis products that will meet the requirements. This new architecture is shown in Figure 4.15.

[5]Intel Xeon 4114T specifications: https://ark.intel.com/content/www/us/en/ark/products/126153/intel-xeon-silver-4114t-processor-13-75m-cache-2-20-ghz.html

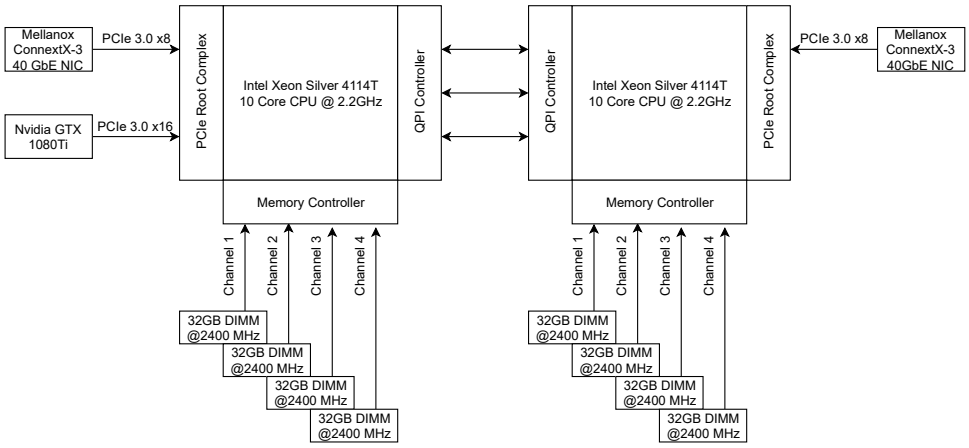
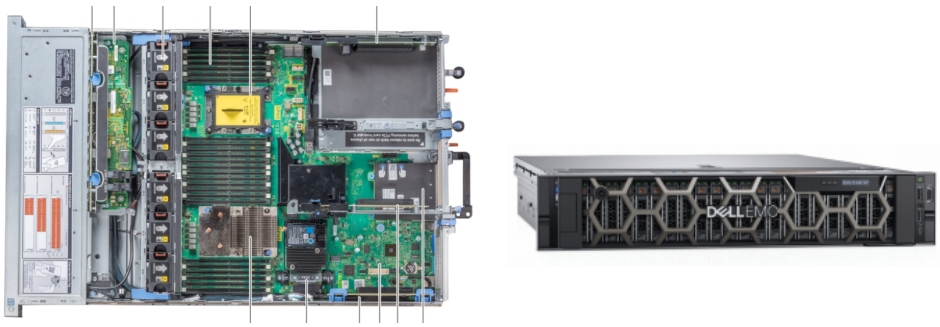


Figure 4.15: Diagram of the dual-socket prototype GPU server architecture.

Each socket is almost identical except for the GPU attached to one socket. The transfer across the QPI boundaries to the GPU did not impact performance. It is suspected that this is due to the data copies across the QPI being asynchronous bulk data copies instead of fine grained copies such as the transpose operation or SPEAD2 heap assembly.

One caveat with a dual port system is that two NICs connected to the same network need some additional configuration in order to receive data correctly. This usually involves a process called network bonding where the two ports are made to look like one. However this was not tested with the prototype due to concerns that it would cause network instability in the telescope data centre during operation. The dual socket system was instead tested by plugging each port into a completely separate network and streaming data in from F-Engine simulators.

These components are all housed in a Dell EMC PowerEdge R740 2U server pictured in Figure 4.16 [66]. The R740 is the definitive Dell 2U server available on the market at the time of writing. It is trusted to perform well thermally and it comes with a comprehensive set of monitoring tools that will make testing the power and thermal performance of this system simple.



(a) Internal layout *(b) External chassis*

Figure 4.16: Images of the Dell EMC Poweredge R740 server.[6]

The R740 is a relatively expensive server. There are cheaper options available that should be able to house the same number of components. However the thermal performance of these cheaper servers should be tested thoroughly. SARAO does have plans to evaluate these cheaper options, but this is beyond the scope of this project. Cost estimates using a cheaper Tyan server[7] and the new AMD Epyc CPUs[8] will be done in Chapter 6. This TYAN and AMD configuration will not be purchased or tested but based on the product specifications, it is expected to equal the performance of the Intel CPU and Dell chassis configuration for a reduced cost.

4.9. Design Conclusion

This chapter has systematically gone through the design process for this prototype.

It began with Section 4.1 which collated and formalised all the system requirements that had been identified in previous chapters.

These requirements were then used to formulate the high level system design in Section 4.2. This design took the form of a pipeline consisting of data being passed from one stage to the next via packets. Each stage was identified and described. The stages requiring the most processing capacity were highlighted. Section 4.3 presented a formalised UML software design of the different stages within this pipeline. These stages were divided into sub-classes of a PipelineStage class. Furthermore a PipelinePacket class and sub-classes were defined for communication between stages.

The complex sub-systems were all identified and the design of these sub-systems was then presented.

The Pipeline Framework sub-system design took place in Section 4.4. The pipeline framework was built on top of the Intel TBB library. This layer made use the polymorphism principle to reduce the amount of boilerplate code. Furthermore, packets were amortised to reduce inter-thread synchronisation.

The implementation of the GPU correlation stage was discussed in Section 4.5. This design involved integrating the xGPU library into the pipeline framework. The achievable throughput on the Nvidia GTX 1080 Ti was measured and quantified.

The inner workings of the ingest stage were examined in Section 4.6. The system configuration that was required to ingest at the required rates were identified. This including managing NUMA

[6] Image source: https://i.dell.com/sites/csdocuments/Shared-Content_data-Sheets_Documents/en/aa/PowerEdge_R740_R740xd_Technical_Guide.pdf

[7] Tyan Transport HX GA88-B8021 server specifications: https://www.tyan.com/Barebones_GA88B8021_B8021G88V2HR-2T-RM-N

[8] AMD EPYC 7002 CPU specifications: https://www.amd.com/en/processors/epyc-7002-series

boundaries, enabling interrupt coalescing and increasing system buffer sizes.

Section 4.7 discussed the transpose operation. This including going through different possible transpose implementations, comparing their performance and determining the number of memory channels required to transpose at the required rate. An SSE implementation making use of 128-bit registers with 4 channels of 2 400 MHz RAM was the optimal implementation of this operation.

In Section 4.8 the hardware platform that these sub-systems would operate in was designed. This design was of a dual socket system with two 40 GbE NICs and a single GPU. This double NIC system shared many common components reducing the overall system cost per stream.

From this chapter, a prototype GPU X-Engine was built. The next chapter will go through the process of testing this prototype and verifying it against its requirements.

5. Verification

The GPU X-Engine has now been designed. The design of this system was according to certain requirements and constraints. The success of the design is measured by its ability to meet all its requirements. This chapter will verify that all requirements have been met and constraints have been adhered to.

Some of these requirements are intrinsically verifiable. These will be discussed in Section 5.1

Other requirements need a more in-depth verification process. In this system, the in-depth verification will be primarily focused on confirming that the data can be processed at the required rate and ensuring that the output data is correct. These requirements will be examined in more detail in Section 5.2 and Section 5.3

5.1. Intrinsically Verifiable Requirements

Intrinsically verifiable requirements are requirements that are realised in the construction of the system. A simple inspection of the system will confirm that was has been specified in the requirements is present in the system. No in-depth verification tests need to be designed.

R5 specified that there needs to be one 40 GbE network port per hardware NIC. The prototype is a 2U system with two 40 GbE ports (discussed in Section 4.8.2), thus satisfying R5.

R6, R7 and R8 specified some of the hardware and software components that must be used in the system. This includes using xGPU with a Nvidia GPU, SPEAD2 with a Mellanox NIC and TBB as the base of the software pipeline. This has all been designed into the system (discussed in Section 4.5, 4.6 and 4.4 respectively), as such these requirements have been met.

R1 to R3 cannot be verified by simple inspection. More extensive tests need to be performed in order to verify these two requirements.

5.2. Output Data Verification

In order to satisfy R2 and R3, the data coming out of the pipeline needs to be verified. To verify that the output data is correct, the input data needs to be carefully controlled.

To produce controlled F-Engine data, an F-Engine packet simulator was created. This simulator transmitted heaps in the F-Engine SPEAD format over 40 GbE in unicast mode. A single simulator would generate heaps for all 64 antennas in a single stream. Due to limits in the simulator hardware, while the heap size was kept the same, the size of the packets in each heap

was increased from 1 KiB to 4 KiB. Each heap consisted of four 4 KiB packets instead of sixteen 1 KiB packets. The input data rate remained the same regardless of packet size.

The data in the simulated heaps was set to a ramp pattern over the frequency channels for specific antennas and zeros for other antennas. Setting data to zero for most antennas made it simple to determine if data meant for one baseline was bleeding into another baseline.

This ramp pattern was changed every synchronisation epoch. This epoch change allowed one to easily see if heaps were being processed out of order.

The expected output for each baseline product is based on Equation (3.4), Equation (3.5), Equation (3.6) and Equation (3.7) in Section 3.1.3. This data was averaged over each accumulation epoch to produce smaller values that were simpler to analyse. The input data per channel was held constant over the entire epoch, so the average, if correct, would be equal to a single un-accumulated baseline value. This averaged output equation is shown in Equation (5.1).

$$B_{x,y,p0p1}(\omega) = \frac{\sum_{n=0}^{M} A_{x,p0}(\omega, n)\overline{A_{y,p1}(\omega, n)}}{M} = A_{x,p0}(\omega, 0)\overline{A_{y,p1}(\omega, 0)} \tag{5.1}$$

5.2.1. Zero Test

The most basic test that was performed was to insert all zero input data. If any output data produced was not zero then this would indicate that something was fundamentally wrong with how the pipeline produced data. When tested, all zero input data produced all zero output data.

The second part of this test was to generate non-zero input data for certain antenna pairs and zero data for other pairs. This ensured that input data did not bleed into the wrong baseline. When this test was run, no bleed through was detected.

These tests were each run for 30 minutes.

5.2.2. Autocorrelation Test

Once the zero test was complete, the output values produced by each antenna pair needed to be verified. This was done by performing an autocorrelation test. Input data for a single antenna was generated and fed into the GPU X-Engine. The baselines values for each polarisation pair out of the pipeline was then be compared to the expected output.

The results for one of these tests is shown in Table 5.1. Table 5.1 and Equation (5.1) correspond with $x = y = 17$, $p0p1 \in \{hh, vh, hv, vv\}$, $M = 408$ and $\omega = Chan$.

Due to the properties of autocorrelation and complex numbers, it is expected that $\overline{h} \times h$ and $\overline{v} \times v$ will produce real numbers and $\overline{v} \times h$ will be equal to the conjugate of $\overline{h} \times v$. All these expected properties occur in this table.

Chan	h Value	v Value	$\overline{h} \times h$	$\overline{v} \times h$	$\overline{h} \times v$	$\overline{v} \times v$	Valid
0	-8+2j	-1-2j	68	4-18j	4+18j	5	✓
1	-2+4j	1-j	20	-6+2j	-6-2j	2	✓
2	-1+6j	2+j	37	4+13j	4-13j	5	✓
3	1+8j	4+2j	65	20+30j	20-30j	20	✓
4	2-8j	6+4j	68	-20-56j	-20+56j	52	✓
5	4-2j	8+6j	20	20-40j	20+40j	100	✓
6	6-j	-8+8j	37	-56-40j	-56+40j	128	✓
7	8+j	-2-8j	65	-24+62j	-24-62j	68	✓
8	-8+2j	-1-2j	68	4-18j	4+18j	5	✓
9	-2+4j	1-j	20	-6+2j	-6-2j	2	✓
10	-1+6j	2+j	37	4+13j	4-13j	5	✓
11	1+8j	4+2j	65	20+30j	20-30j	20	✓
12	2-8j	6+4j	68	-20-56j	-20+56j	52	✓
13	4+4j	-2+8j	32	24-40j	24+40j	68	✓
14	6-j	-8+8j	37	-56-40j	-56+40j	128	✓
15	8+j	-2-8j	65	-24+62j	-24-62j	68	✓

Table 5.1: Table showing the results from an autocorrelation test.

Figure 5.1 plots the data in Table 5.1.

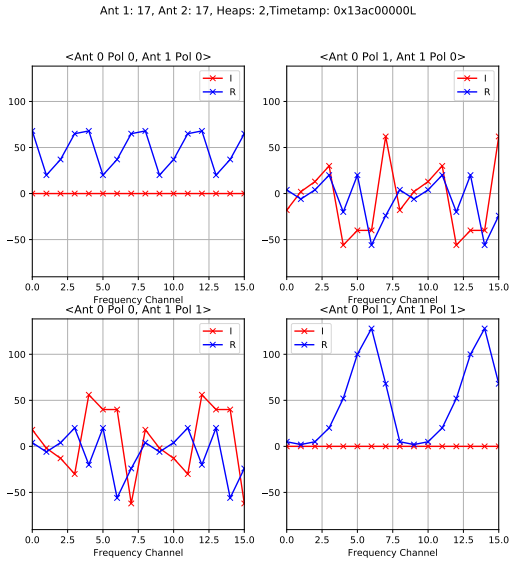


Figure 5.1: Plot showing the results of the autocorrelation test. The zero phase of the hh and vv baselines, and the complex conjugate relationship between the hv and vh baselines is visible.

The autocorrelation tests produce results that are identical to what is expected. This proves that the X-Engine is producing the correct data for each autocorrelation baseline.

5.2.3. Correlation Test

With the auto correlation function verified, it was then necessary to verify that the correlation output data between antenna pairs is as expected. This test was performed in a very similar fashion to the autocorrelation test, except that data for two different antennas was generated.

The results for one of these tests is shown in Table 5.2. Table 5.2 and Equation (5.1) correspond with $x = 50$, $y = 17$, $p0p1 \in \{hh, vh, hv, vv\}$, $M = 408$ and $\omega = Chan$

Chan	$A_{17,h}$ Value	$A_{17,v}$ Value	$A_{50,h}$ Value	$A_{50,v}$ Value	$\overline{A_{17,h}} \times A_{50,h}$	$\overline{A_{17,v}} \times A_{50,h}$	$\overline{A_{17,h}} \times A_{50,v}$	$\overline{A_{17,v}} \times A_{50,v}$	Valid
0	-8+2j	-1-2j	1+8j	6+4j	8-66j	-17-6j	-40-44j	-14+8j	✓
1	-2+4j	1-j	2-8j	8+6j	-36+8j	10-6j	8-44j	2+14j	✓
2	-1+6j	2+j	4-2j	-8+6j	-16-22j	6-8j	44+42j	-10+20j	✓
3	1+8j	4+2j	6-j	-2+8j	-2-49j	22-16j	62+24j	8+36j	✓
4	2-8j	6+4j	8+j	-1-2j	8+66j	52-26j	14-12j	-14-8j	✓
5	4-2j	8+6j	-8+2j	1-j	-36-8j	-52+64j	6-2j	2-14j	✓
6	6-j	-8+8j	-2+4j	2+j	-16+22j	48-16j	11+8j	-8-24j	✓
7	8+j	-2-8j	-1+6j	4+2j	-2+49j	-46-20j	34+12j	-24+28j	✓
8	-8+2j	-1-2j	1+8j	6+4j	8-66j	-17-6j	-40-44j	-14+8j	✓
9	-2+4j	1-j	2-8j	8+6j	-36+8j	10-6j	8-44j	2+14j	✓
10	-1+6j	2+j	5-2j	-8+8j	-17-28j	8-9j	56+40j	-8+24j	✓
11	1+8j	4+2j	6-j	-2-8j	-2-49j	22-16j	-66+8j	-24-28j	✓
12	2-8j	6+4j	8+j	-1-8j	8+66j	52-26j	62-24j	-38-44j	✓
13	4+4j	-2+8j	-8+2j	1-j	-24+40j	32+60j	-8j	-10-6j	✓
14	6-j	-8+8j	-2+4j	2+j	-16+22j	48-16j	11+8j	-8-24j	✓
15	8+j	-2-8j	-1+6j	4+2j	-2+49j	-46-20j	34+12j	-24+28j	✓

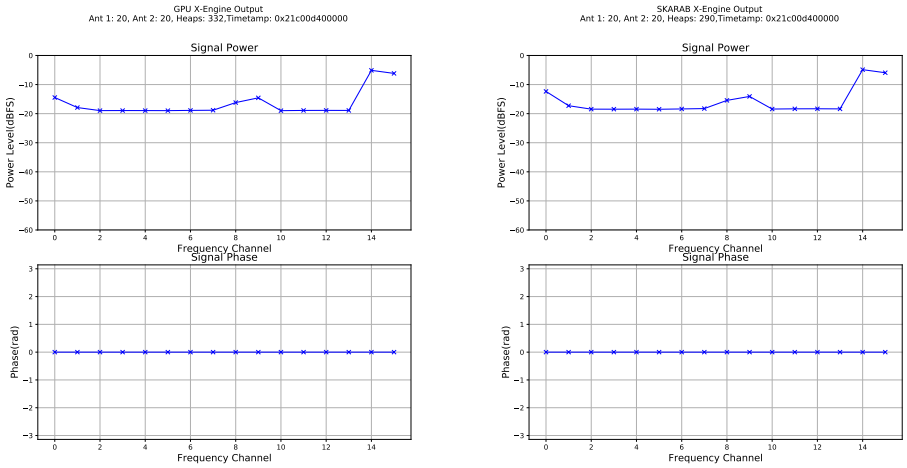
Table 5.2: Table showing the results from a correlation test.

The correlation tests produce results that are identical to what is expected. This proves that the X-Engine is producing the correct data for each baseline.

5.2.4. Real Data Test

A final test that was done was to compare the output of SKARAB X-Engine with the GPU X-Engine. The equipment used to test this was a spare server on the MeerKAT site, not the designed server in the SARAO laboratory. This server was limited in its capability - it only had a Mellanox ConnectX-3 NIC, unlike the Mellanox ConnectX-5 installed in the prototype. Furthermore, its RAM was clocked at 1600 MHz instead of the 2666 MHz specified. As such, it

could only process one stream while dropping up to 15% of the heaps. The results for this test are shown in Figure 5.2 with Figure 5.2a showing the GPU X-Engine output and Figure 5.2b showing the SKARAB X-Engine output. Due to the packet drops,the SKARAB and GPU results are not identical. These results show that the GPU implementation produces an output at least similar to what is expected.



(a) GPU X-Engine pipeline output *(b) SKARAB pipeline output*

Figure 5.2: Plots showing the similarities of the output spectra produced by the SKARAB and GPU X-Engines.

5.2.5. Output Data Verification Results

All these tests have confirmed that the data generated is correct. The pipeline operates on the correct data, data does not bleed between antennas and the correlation output is correct. This satisfies R2 and R3.

5.3. Data Rate Verification

As per R1, the GPU X-Engine needs to be able to ingest 4 streams of data per port. Furthermore, according to R4, this data needs to be correlated in real time. The final dual socket system has two ports. This means that it needs to process data received at a sustained 54.4 Gbps.

A rate test was designed. This test was performed by using the F-Engine simulator that was mentioned in Section 5.2. Two servers were connected to the GPU X-Engine via 40 GbE. Each of these servers transmitted four simulated streams at a minimum data rate of 6.8 Gbps per stream. Confirming that this data was processed correctly involved measuring the RX rate at the NIC (Section 5.3.1) and then at every stage in the pipeline (Section 5.3.2).

5.3.1. Data Rate Verification at the NIC

Measuring the NIC RX rate was done using using the Linux utility iftop[1]. The input data rates for a single 40 GbE port is shown in Figure 5.3.

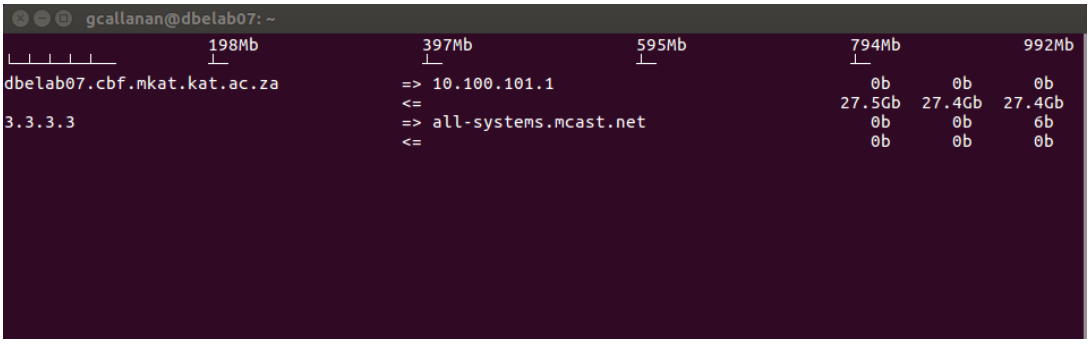


Figure 5.3: Image showing the measured rates of data ingested by the GPU X-Engine.

The input data rates are shown in the top-right section of Figure 5.3 with the three columns representing the average rate over 2, 10 and 40 seconds respectively. All these rates are reported to be above 27.4 Gbps. This image shows that data rates were sustained for at least 40 seconds. These tests were run for an hour. This tool was used on both 40 GbE ports but only the figure for one was shown as they both reported similar results. Figure 5.3 verifies that the data was successfully received at the NIC at the required rate.

5.3.2. Data Rate Verification Through the Pipeline

Once the NIC input rate had been verified, the rate of data through the pipeline needed to be measured. This was done by adding counters to every stage in the pipeline. Every ten seconds the number of packets processed per stage would be printed to the terminal.

Figure 5.4 is an image of the terminal output of the counter values. The most informative values in this image are the *Normalised Diff* values. This is the difference in the number of packets received at the start of the ten second recording and at the end. As the name implies, these counters have been normalised to account for packet aggregation that takes place in the pipeline. The *HeapsReceived*, *SpeadRx* and *Buffer* counters occur before any accumulation takes place - as such the reported value is equal to the actual value. The *Transpose* and *GPUWrapper* counters occur after 64 packets have been aggregated into one - the reported value is the stage counter multiplied by 64. The *SpeadTx* counter increments after accumulation on the GPU occurs (408 packet accumulations in this case). The normalised value for *SpeadTx* is thus the original counter value multiplied by 64 and then multiplied by 408. According to Table 4.1, the

[1]iftop Linux man page: https://linux.die.net/man/8/iftop

number of packets processed at each stage needed to be above 52 224 packets per second per stream when normalised. These counters are all above 53 000 indicating that the packets are processed at the required rate.

```
Complete Heaps Rate  : 72.21 Gbits received in 10.00 seconds. Data Rate: 7.22 Gbps
Incomplete Heaps Rate: 72.21 Gbits received in 10.00 seconds. Data Rate: 7.22 Gbps
Processed Heaps Rate : 72.27 Gbits received in 10.00 seconds. Data Rate: 7.23 Gbps
HeapsReceived                :   17365234 Normalised Diff: 550882
SpeadRx       Packets Processed:   17365232 Normalised Diff: 550882
Buffer        Packets Processed:   17365232 Normalised Diff: 550882
Transpose 0   Packets Processed:     271047 Normalised Diff: 551360
GPUWrapper    Packets Processed:     271001 Normalised Diff: 550400
SpeadTx       Packets Processed:        665 Normalised Diff: 537600
Incomplete Heaps: 2 heaps out of 17365234. Drop Rate Inst/Tot: 0.0000/0.0000 %
Heaps Too Late: 7. Diff: 0. Instantaneus Percentage Late:0.0000%
```

Figure 5.4: Image of the packet counter values at different stages of the GPU X-Engine pipeline for a single multicast stream. The GPU X-Engine has been fed simulated data.

These counters in Figure 5.4 tick over correctly when all 8 stream are running. This satisfies all data rate requirements. One caveat is that there is some instability when all 8 pipelines are running. When the 8th pipeline is started there is a chance that it will stall. This always occurs on one pipeline assigned to specific cores. It is suspected that this is due to some other process operating on those cores which causes a bottleneck in the pipeline. When this bottleneck occurs, a bug in the *Buffer* stage prevents the pipeline recovering from this stall. Due to time constraints this specific problem was not diagnosed and fixed for all cases but it occurs in a minority of tests and only for the full 8 multicast streams.

Figure 5.4 also reports on the data rates into and out of the pipeline, and the number of missing packets. The *Incomplete Heaps Rate* label is misleading. This label actually reports the rate of the packets in the system after the incomplete heaps have been removed from the pipeline. The input and output processing rates reported in this Figure are above the required levels. Negligible packets are dropped in this system when testing simulated data with 4 KiB packet sizes.

5.3.3. Data Rate Verification on Site

As mentioned earlier, the limit of the simulator was that it could only produce 4 KiB packets instead of 1 KiB packets. The prototype platform could not be sent to the MeerKAT telescope site in the Karoo to run on actual data, however the on-site server mentioned in Section 5.2.4 reported a high heap loss rate of 15% when processing actual telescope data. The pipeline counters for this test are shown in Figure 5.5. The packet counters reach their expected value at the transpose stage as the dropped packets are replaced with packets with zero value by the buffer stage, indicating the pipeline can handle lost heaps.

```
Complete Heaps Rate  : 57.38 Gbits received in 10.00 seconds. Data Rate: 5.74 Gbps
Incomplete Heaps Rate: 60.55 Gbits received in 10.00 seconds. Data Rate: 6.05 Gbps
Processed Heaps Rate : 67.95 Gbits received in 10.00 seconds. Data Rate: 6.79 Gbps
HeapsReceived                  :    1846403 Normalised Diff: 461935
SpeadRx       Packets Processed:    1749969 Normalised Diff: 437798
Buffer        Packets Processed:    1749969 Normalised Diff: 437798
Transpose 0   Packets Processed:      32401 Normalised Diff: 518400
GPUWrapper    Packets Processed:      32401 Normalised Diff: 519680
SpeadTx       Packets Processed:         79 Normalised Diff: 512000
```

Figure 5.5: Image of the packet counter values at different stages of the GPU X-Engine pipeline for a single multicast stream. The GPU X-Engine has been fed real telescope data.

The packet loss rate is likely significantly less than the heap loss rate, as when a packet is lost, the entire heap of 16 packets is dropped by the SPEAD2 library. This loss rate was independent of the number of streams running seeming to indicate it is a limitation of the SPEAD2 library when it comes to processing higher numbers of packets.

This server at the MeerKAT site was an older server with significantly slower RAM than the designed prototype (1666 MHz compared to 2400 MHz), so it cannot be conclusively stated that the reduced packet size results in this increased drop rate.

After some discussion with the SPEAD2 author, it was concluded that these packets drops would be difficult to avoid with 1 KiB packets as often these drops are occurring at the NIC before the CPU is even able to process them. As such it is recommended that in order to use a GPU X-Engine at MeerKAT data rates, the F-Engines should be modified in future iterations to use 4 KiB packets. This will reduce heap drops to near zero.

5.3.4. Data Rate Verification Summary

Apart from some minor instability on a single stream when all 8 streams are running, the X-Engine is able to process all data in real time, resulting in R4 being verified.

The input rate requirements (R1) could only be partially verified. It is expected that the smaller 1 KiB packet size could lead to a higher packet drop rate. This cannot be cheaply fixed by modifying the X-Engine, but is quite simple to fix by increasing the F-Engine packet size, thereby eliminating this problem.

5.4. Verification Conclusion

This chapter has gone through all the system requirements outlined in Section 4.1 in order to determine if they have been met.

The intrinsic properties of the system were the simplest to verify. This was discussed in Section 5.1.

Verifying that the output data was correct and processed at the required rate was a more

complicated process.

Section 5.2 was an exhaustive testing process that fed a variety of different simulated input patterns into the system and measured the output data to confirm that it was correct. This section also compared the GPU X-Engine data to the SKARAB X-Engine data and showed that the results were as expected.

Section 5.3 fed simulated data into the system, and measured the data rate at each stage in the pipeline. This section verified that the pipeline was able to process 8 streams simultaneously. Instability on the 8th stream at start-up was observed and reported upon. Furthermore, high packet drop rates were observed when running the pipeline on older systems with 1 KiB packet sizes. In order eliminate this, the F-Engine output packet size should be increased to at least 4 KiB.

Table 5.3 matches all the requirements to a specific section in this chapter and indicates if they have been verified.

Requirement	**Verification Section**	**Verified**
R1	5.3	∼
R2	5.2	✓
R2.1	5.2	✓
R3	5.2	✓
R3.1	5.2	✓
R3.2	5.2	✓
R3.3	5.2	✓
R3.4	5.2	✓
R4	5.3	✓
R5	5.1	✓
R6	5.1	✓
R6.1	5.1	✓
R6.2	5.1	✓
R7	5.1	✓
R7.2	5.1	✓
R7.1	5.1	✓
R8	5.1	✓

Key:
✓: Verified
∼: Partially verified
✗: Not verified

Table 5.3: Table listing the different system requirements, the location of the tests performed to verify each requirement and the verification status of the tested requirement.

This chapter has verified that most system requirements have been met. R1 could only be partially met but a workaround was found and proposed for future iterations of the correlator. The next chapter will perform an in-depth analysis of the system that will go beyond just meeting the requirements.

6. Analysis and Discussion

This chapter seeks to further the understanding of the system operation and performance. It will be focussed in such a way as to answer the dissertation research questions and test the hypothesis presented in Section 1.2.

Verification (Chapter 5) can be seen as the start of system analysis. However where verification needs to prove the requirements are met, this chapter looks beyond just the requirements instead seeking to gain insight into the system's strengths and weaknesses. It will also determine which areas of research should be focused on in the future.

Research question 3 required that the differences in the SKARAB and GPU X-Engine implementation to be analysed. Section 6.1 examines both the hardware costs and energy consumption of both implementations to assist with answering this question.

Research question 1 asked how to design a drop-in replacement for a SKARAB X-Engine. Chapter 5 proved that it was possible to design this system. However, it is useful to have a more in-depth understanding of the system performance to identify bottlenecks and opportunities to improve. Research question 2 was concerned with the density of this system: will more modern technology allow for the creation of a more dense X-Engine than SKARAB? Section 6.2 examines the system to provide information to answer these questions. Section 6.3 then uses this insight to make some suggestions as to how this system can be improved.

6.1. Comparative Analysis

This section will compare the cost and power consumption of the GPU X-Engine prototype to that of the SKARAB X-Engine. The system cost can divided into two sections: initial hardware costs (Section 6.1.1) and energy costs (Section 6.1.2). Both of these costs will be quantified and measured. Two SKARABs are required for every GPU X-Engine, as such all SKARAB costs will be doubled. The final system cost will be discussed in Section 6.1.3.

6.1.1. Hardware Cost

The SKARABs were custom designed for MeerKAT. The initial contract was for 288 SKARABs and cost R39.5 million[67]. Each SKARAB therefore cost just under R140 000. A recent purchase of 17 SKARABs in February 2019 has seen the cost rise to R215 000 per SKARAB[68]. This is a common risk with bespoke systems: the cost for replacement components from the supplier can often be higher than the initial cost of these components. The most recent price of R215

000 per SKARAB will be used. Additionally each SKARAB is populated with three Hybrid Memory Cube (HMC) mezzanine cards. These modules were designed by SARAO and cost approximately R10 000 per module. The total hardware cost per SKARAB is thus equal to R245 000. The cost for two SKARABs is equal to R490 000.

The cost of the GPU server is the sum of the individual components. The prototype system was developed using a Dell R720 and two Intel Xeon Silver Z4114T CPUs. If this system were to be deployed at scale it is more likely that it will be built using the Tyan server and AMD CPU mentioned in Section 4.8.2. A single AMD ROME 7402 CPU is about equal in performance and cost to two of the Intel Z4114T CPUs. The Tyan server is cheaper than the Dell server. The costs of the system was determined using quotes obtained by SARAO[69] [70]. The prices were valid in October 2019. The component costs are shown in Table 6.1:

Component	**Cost Per Unit(R)**	**Quantity**	**Cost(R)**
TYAN B8021G88V2HR-2T-RM-N 1U Server	49 359	1	49 359
AMD ROME 7402 CPU	25 239	1	25 239
Nvidia GeForce 1080Ti GPU	13 000	1	13 000
Mellanox ConnectX-5 NIC	17 675	2	35 350
32GB DDR4-2666 2RX4 LP ECC REG DIMM	2 515	8	20 120
Total:			143 068

Table 6.1: Table showing the hardware cost of the GPU system at the component level.

The total cost of the GPU X-Engine is R143 068. This is just over a third of the cost of the two SKARABs. Additionally, the GPU X-Engine is manufactured from off the shelf components - replacement components should be readily available (and will likely reduce in price over time). These are both advantages over the SKARAB system.

6.1.2. Energy Consumption

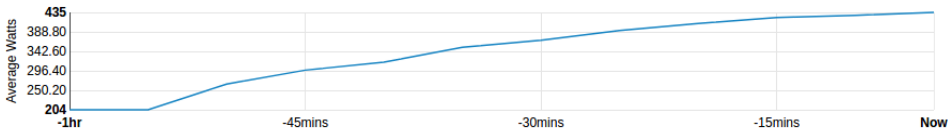
Before comparing the SKARAB and GPU power consumption, these values need to be measured.

Dell servers have an integrated management processor for system control and monitoring known as an iDRAC[1]. The iDRAC is accessible via a separate 1 GbE Ethernet port and provides an easy to use web interface where the power can be monitored. The iDRAC monitors the power drawn from the ATX power supply.

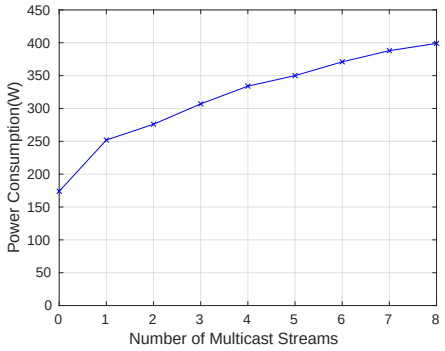
In order to measure the power, the system was left to idle for 5 minutes. After 5 minutes a single F-Engine stream was fed into the system, after a further 5 minutes, another stream was fed

[1]iDRAC home page: https://www.delltechnologies.com/en-za/solutions/openmanage/idrac.htm

into the system. This was done until the system was processing the full 8 streams. The power consumption plot produced by the iDRAC web interface while these tests were being performed is shown in Figure 6.1a. The results produced are measured over time. Figure 6.1b matches the power consumption level with the number of streams being processed. The prototype server had an additional accelerator card installed that consumed 30W. This 30W was subtracted from the recorded power when producing Figure 6.1b



(a) Measured power consumption



(b) Adjusted power consumption

Figure 6.1: Plots showing the GPU X-Engine system power consumption versus number of multicast streams being processed.

Figure 6.1 shows that the system has high base power consumption with each stream requiring a small amount of additional power on top of this base consumption (27 W per stream). Furthermore as per Figure 6.5 the GPU only consumes 85 W for the full 8 streams. This indicates that the supporting server infrastructure contributes quite significantly to the overall power consumption. Future designs should attempt to process even more streams per server to distribute this idle load across more streams and reduce overall system power consumption.

SKARAB also has an ATX power supply with current and voltage sensors on all the supply rails. These currents were monitored when the SKARAB X-Engine was idle, processing 2 streams and processing 4 streams. The power consumption from the SKARAB was 70 W, 80 W and 90 W respectively.

Figure 6.2 plots the SKARAB and GPU X-Engine power consumptions vs number of streams. Two SKARABs were tested as both were required to equal the performance of a single GPU

server - when 4 multicast streams were tested, each SKARAB processed 2 streams. As expected the two SKARABs require far less power than the GPU server when processing all the required data. In this case it is less than half.

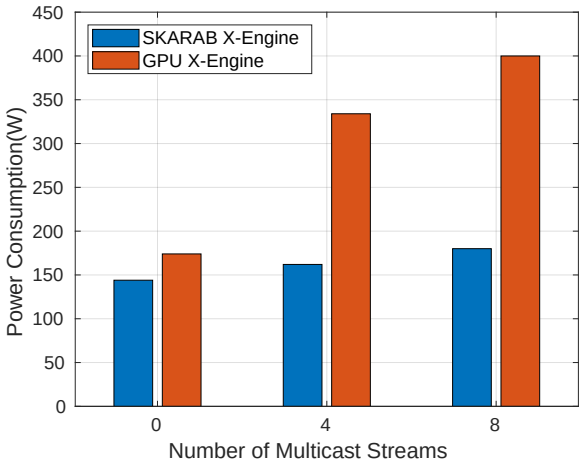


Figure 6.2: Plot comparing the power consumption of a single GPU X-Engine and two SKARAB X-Engines versus number of multicast streams being processed.

According to the latest South African electricity tariffs, it will cost no more than R2.00 per kWh[71]. The inflationary increases in electricity prices will not be considered in these calculations.

Based on the measured power consumption, the monthly energy cost for two SKARABs are:

$$0.18\ kWh/h \times 24\ hours \times 30\ days \times R2.00/kWh \approx R260.00/month$$

The monthly energy cost of the GPU X-Engine is:

$$0.4\ kWh/h \times 24\ hours \times 30\ days \times R2.00/kWh \approx R576.00/month$$

The GPU X-Engine costs more than twice as much per month to run than two SKARABs.

The power consumption of the system does not just impact the system cost. It has an impact on the data centre in terms of cooling requirements and power infrastructure. This is because while 200W its not much in a single system, 200W over 64 GPU systems (the number required by MeerKAT) starts to generate a significant amount of heat. A server room equipped with GPUs instead of FPGAs will need more advanced cooling solutions such as water or immersion cooling compared to the current air cooling system. Discussing these implications further is beyond the scope of this dissertation.

6.1.3. System Cost

The lifecycle costs of the GPU and SKARAB systems are plotted in Figure 6.3. This assumes a 10 year lifespan. While this is an optimistic estimate for commercial integrated circuits under continuous use, it does serve to illustrate that the cost over the entire life-cycle of the GPU system is far less than the cost of the SKARAB due to the major difference in initial hardware costs. The maintenance costs have been excluded as it is assumed that these costs will be similar across both systems.

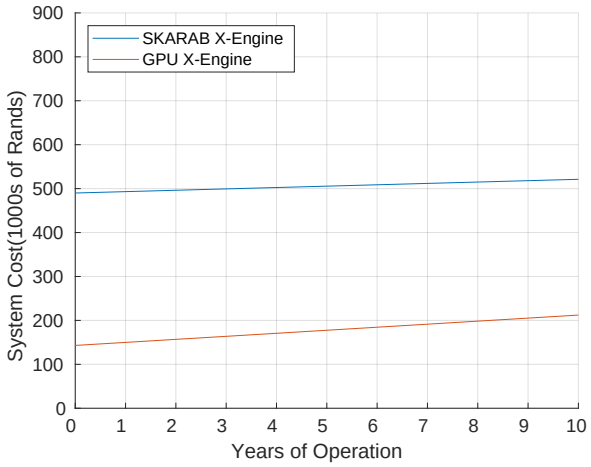


Figure 6.3: Plot of GPU and SKARAB X-Engine procurement and energy costs over 10 years.

From a cost perspective it is clear that the GPU server described in Table 6.1 is significantly cheaper than the two SKARABs it would replace.

6.2. System Performance Analysis

While the actual correlation algorithm takes place on the GPU, a significant amount of processing takes place in the CPU domain. This is because both the transpose and ingest operations are processing intense and require a large number of the CPU cycles each.

Figure 6.4 shows the CPU utilisation of the system when ingesting 8 multicast streams. This figure was obtained from the Linux utility htop[2] running on the GPU X-Engine server. The first 16 cores are assigned to the 8 streams. Core 17 manages the systems IRQs, core 1 and 4 manage other system functions and core 18 is idle. While core 17 is loaded at 100%, this does not affect the processing of the streams. It is clear from this figure that the CPUs have very little spare processing capacity. This indicates that the number of CPU cores was correctly specified. The instability that periodically occurs when all 8 streams are being processed is a sign that the

[2]htop Linux man page: https://hisham.hm/htop/

CPU side of the system is almost fully loaded.

From Figure 6.4, it is also clear that far too much system memory has been installed to the server (Using 34.8 GB out of a possible 377 GB) - the server was initially filled with as much RAM as possible (12 DIMMs with 32 GB capacity per DIMM). The size of each DIMM module can be reduced to a 8 GB to reduce costs. The number of DIMMs and DIMM clock speed must remain the same to preserve the memory bandwidth.

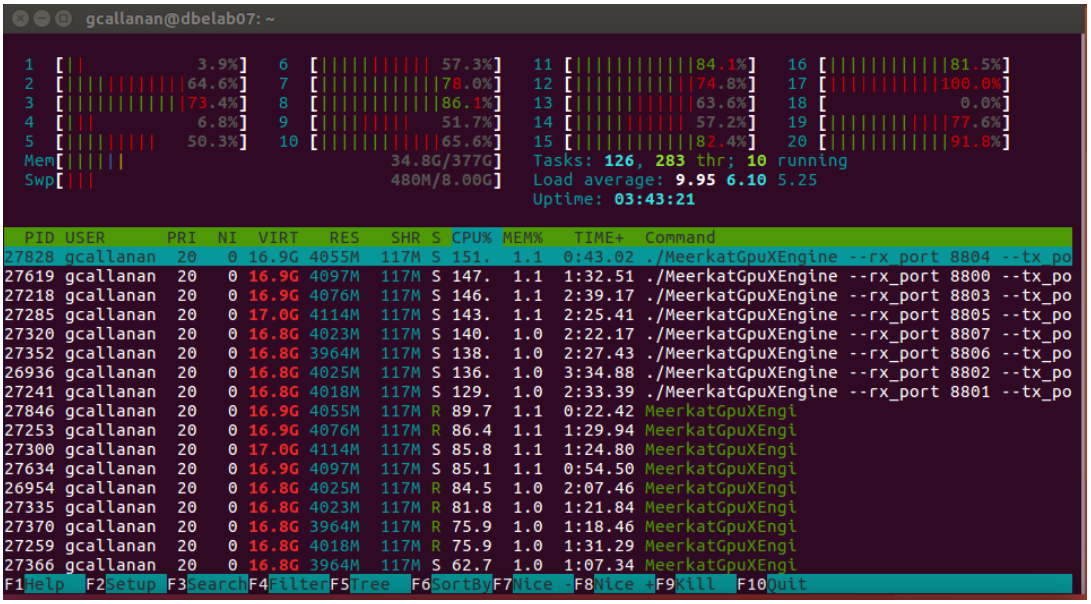


Figure 6.4: Image showing the RAM and CPU utilisation by the host server when the GPU X-Engine is processing 8 multicast streams.

Once all the CPU processing had been performed, the data was copied to the GPU for correlation. Figure 6.5 shows the GPU utilisation obtained when running the Nvidia System Management interface[3] on the GPU X-Engine. The GPU utilisation was significantly variable ranging between 50% and 80% of the total GPU utilisation. Furthermore, the floating point version of xGPU was used in this prototype. If the fixed point version was used, half as much GPU compute would be required (see Figure 3.28). Less than 10% of the GPU memory is used.

[3]Nvidia SMI home page: https://developer.nvidia.com/nvidia-system-management-interface

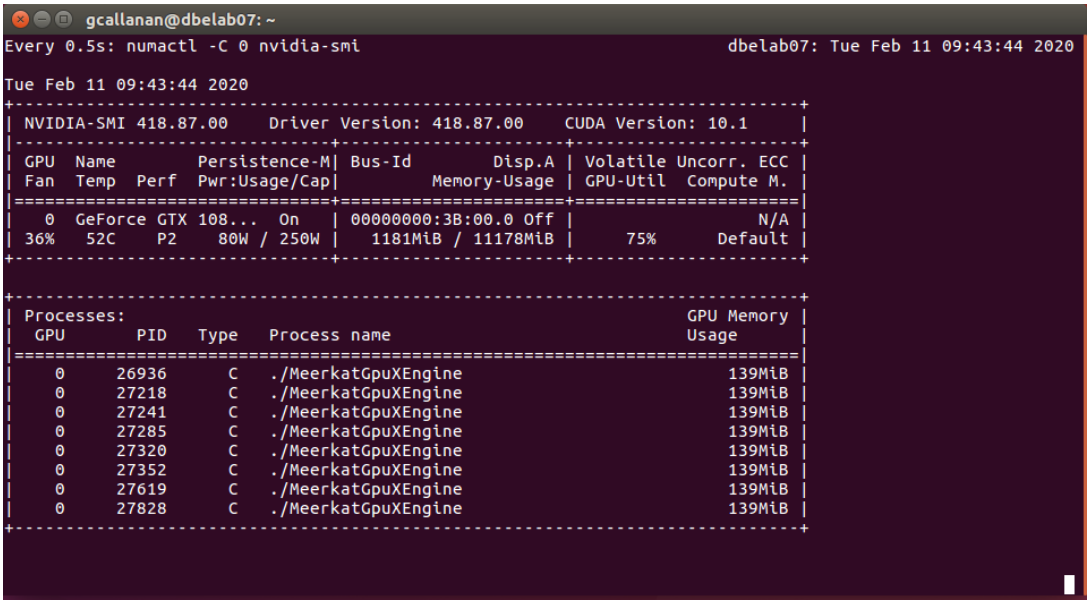


Figure 6.5: Image showing the resource utilisation of the GPU when the GPU X-Engine is processing 8 multicast streams.

Figure 6.4 and Figure 6.5 show that the main system bottleneck lies in the CPU side of the processing pipeline. The CPUs are nearly at full capacity while the previous generation Nvidia GPU sits at 75% compute utilisation.

The GPU X-Engine system meets all its design constraints, therefore this near bottleneck is not cause for concern. It must only be considered when looking at improving the system in the future.

6.3. Upgrade Path and Future Designs

The GPU X-Engine is able to process 8 streams per server. When using the 1U Tyan server, this equates to a density twice that of SKARAB. However future designs may be able to support an increased number of 40 GbE ports per server, reducing power and costs due to increased sharing of common resources.

6.3.1. An Iterative Improvement

The CPU bottleneck is the primary concern when looking at improving this system. After testing on a number of different systems, it became clear that adding more CPUs would not resolve this bottleneck. This is because the CPU side has many different interacting processes that are all contending for the same resources. This bottleneck is due to memory bus contention, TLB contention and IRQ rates among other issues. While RAM clock speeds and the number of memory channels supported by motherboards continues to increase, it is difficult to quantify and measure how these improvements will affect the throughput rate.

In future designs, it would be simpler to reduce the CPU and RAM load by moving the transpose stage onto the GPU. GPUs have higher memory bandwidths and should be able to perform the transpose for minimal processing capacity. If the GPU reaches capacity, another GPU should be slotted into a spare PCIe slot for minimal additional cost (The Dell R720 and Tyan server both have extra PCIe slots). The high level data flow diagram for this updated system is shown in Figure 6.6. This is an updated version of Figure 4.1.

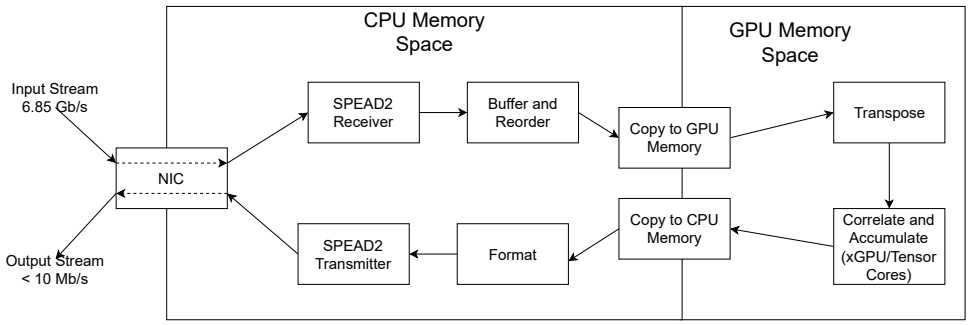


Figure 6.6: Diagram showing the logical flow of antenna data through the proposed next generation GPU X-Engine.

To build this system, xGPU would have to be reverse engineered in order to separate the PCIe transfers and kernel operations of the library. This would allow for a transpose operation to be inserted between these two operations. Alternatively, the ASTRON Tensor Core kernels could be used instead of xGPU as they are already independent from the bus transfer. Two dual port NICs can then be used instead of two single port NICs. This would allow the system to support up to four 40 GbE ports compared to the current two 40 GbE ports.

This improved X-Engine may require larger packet sizes than 4 KiB to handle all the additional ingest data. The packet size required will need to be determined when benchmarking this system.

6.3.2. A Complete Architecture Shift

Another option is to not take the traditional approach of using the NIC and CPU to receive data. Xilinx has released an FPGA board called an Alveo Accelerator card that is designed to slot into a PCIe connector on a server[72]. The U280 Alveo card has two 100 GbE ports and a PCIe 3.0 x16 interface. This card is priced at only $6 000 and is very cost competitive when compared to both GPU servers and FPGAs. Some preliminary calculations indicate that the Alveo card has sufficient resource to run 2 SKARAB equivalent F-Engines and still have additional resources available. These two F-Engines would receive digitiser data over one of the 100 GbE ports. This would leave a single extra 100 GbE port. This port could be used to receive X-Engine data

which could be transposed in the FPGA fabric (FPGAs are better suited for this reorder than either GPUs or CPUs). This data could then be transferred directly from the PCIe bus to the GPU for processing - the CPU and NICs would be bypassed entirely. This system would be capable of receiving 10 streams - 2.5 times that of a single SKARAB. In total this system would do the work of 2 SKARAB F-Engines and 2.5 SKARAB X-Engines for a cost very similar to the current GPU X-Engine. This is a significant reduction in hardware costs. This will potentially use less power than the GPU X-Engine as well. Figure 6.7 shows a basic diagram of this system architecture. The X-Engine signal chain follows the red path on this diagram.

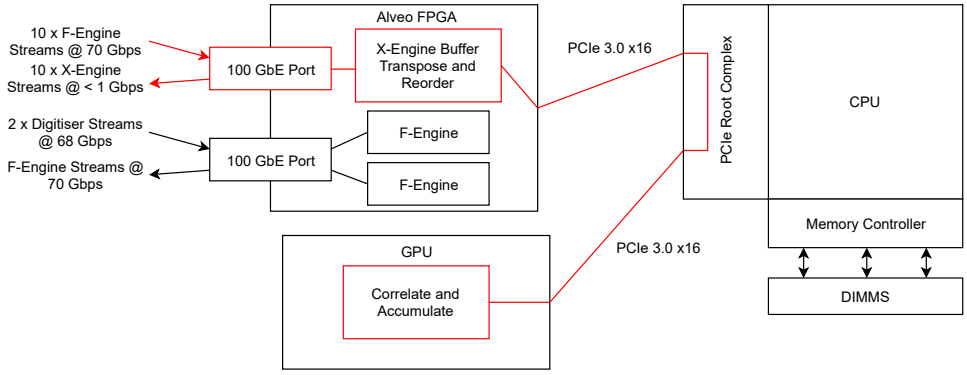


Figure 6.7: Diagram showing a proposed computationally dense hybrid FPGA/GPU system that will make use of a Xilinx Alveo FPGA to reduce the GPU X-Engine host CPU and RAM resource requirements. The Alveo FPGA will also host two F-Engines.

The FPGA/GPU hybrid is highly conceptual and would require months of research before implementation. The biggest risk in this concept is the Alveo to GPU direct PCIe transfer. Some traditional NICs have this functionality, indicating that it is possible to implement, although it is not an advertised function of the Alveo card. It is likely that this feature would need to be developed from scratch.

Another advantage of using the Alveo card as a NIC is that it will be able to handle packets of any size without dropping them.

This FPGA/GPU hybrid concept was developed at SARAO during formal discussions with the signal processing engineers on the potential design of the next generation correlator. It is still being examined for feasibility and is likely to change as more of the technical aspects are understood.

6.3.3. Analysis Conclusion

This chapter has conducted an in-depth analysis of the cost, performance and energy requirements of the GPU X-Engine. This has included comparing the GPU implementation to the SKARAB

implementation, diagnosing bottlenecks and suggesting techniques to improve the design.

Section 6.1 compared the GPU X-Engine to the SKARAB X-Engine. Section 6.1.1 looked at the hardware costs of each system, finding that the SKARAB solution cost nearly three time as much as the GPU solution. Section 6.1.2 then examined the power consumption of these two systems. The GPU solution required more than twice as much energy compared to the SKARAB solution. It was mentioned that this will have implications on cooling when deployed at scale. Section 6.1.3 calculated the lifecycle costs of the system where it was determined that the GPU system is far cheaper over the entire system life-cycle even when considering that the energy costs are twice that of SKARAB.

Section 6.2 examined the GPU X-Engine performance. The current system performance was analysed, which identified that the CPU portion of the server was reaching capacity while the GPU portion was underutilised. Section 6.3 then used this information to suggest future versions of the system. One version involved moving the transpose operation to the GPU to ease the CPU side bottleneck. This would allow twice as many 40 GbE ports per X-Engine, doubling the system processing capacity. This next generation version is the logical improvement of the current design and can be implemented with minimal additional research.

Another more radical future implementation was proposed. This implementation would be a FPGA/GPU hybrid system. This system would be able to do the work of 4 SKARABS, with reduced hardware and energy costs. However this suggestion is in the early concept stages and would require significant amounts of research before becoming viable.

The GPU X-Engine design has now been fully analysed. The analyses has looked at the current state of the system, compared it to its competitors and suggested where it can be improved after future research. Sufficient information exists to answer the research questions posed in Section 1.2. This will be undertaken in the next chapter.

7. Conclusions and Recommendations

Chapter 1 introduced the topic of this dissertation. This involved briefly describing the MeerKAT project, detailing the reasons for looking into GPU systems, listing the research questions and presenting the hypothesis.

The research questions and hypothesis all centred around the design of a drop-in GPU replacement for the MeerKAT X-Engine. A prototype platform needed to be developed in order to answer these questions.

A Systems Engineering approach to system design was adopted for this project. It was outlined in Chapter 2

As indicated in the methodology, the development of the prototype was a two step process. Step one (Chapter 3) was to understand the system context and examine all tools available for constructing an X-Engine. Section 3.1 looked at the scientific and technical context of the X-Engine; including examining the MeerKAT X-Engine input and output data formats and data rates. Section 3.2 examined the different radio astronomy instruments that use GPU X-Engines, while Section 3.3 investigated the possible software and hardware tools that were available to build the prototype. From this review, it was decided to use Nvidia GPUs running the xGPU library, Mellanox NICs using the SPEAD2 library and the Intel TBB library to create a software pipeline that connected all these different tools together.

Step two of this design (Chapter 4) focused on building a functional prototype using the information presented in Chapter 3. The system requirements were formalised in Section 4.1. A high level system design took place in Section 4.2: the system was divided into a number of signal processing stages that would operate in a multi-threaded pipeline. Section 4.3 presented the formal software design for this pipeline. Section 4.4 to Section 4.7 examined the design for the various subsystems highlighting all relevant major technical concerns and resource constraints. Section 4.8 then provided a final design for a hardware platform - a dual socket system with two 40 GbE NICs. This system shared the GPU and server infrastructure between the sockets.

Chapter 5 tested the prototype to confirm that it conformed to its requirements. Section 5.1 confirmed that the intrinsically verifiable requirements were met. Section 5.2 verified that the baselines being produced by the X-Engine were correct for all different polarisation products. Section 5.3 confirmed that this data was processed at the required rate for the full 8 multicast streams. High heap drop rates were observed when ingesting 1 KiB packets. It was recommend that the F-Engine output packet size be changed to 4 KiB to eliminate this problem.

Chapter 6 analysed the system. Section 6.1 found that the GPU X-Engine hardware was a third of the price of a SKARAB equivalent solution but consumed over twice the power. Section 6.2 determined that the CPU side of the system would be the main bottleneck in future designs; the GPU was used at only 65% of its capacity while the CPU portion was at about 85% utilisation. Section 6.3 suggested methods to avoid this bottleneck in future designs. Suggestions included offloading processing from the CPU to the GPU or creating a FPGA/GPU hybrid system that could bypass the CPU and NIC entirely.

This chapter concludes this dissertation. Section 7.1 makes use of information gathered from the previous chapters to answer the research questions and discuss the hypothesis. Section 7.2 recommends future research required to develop the next generation of platforms.

7.1. Research Questions Answers

The research questions and hypothesis were presented in Section 1.2. This section will draw on the information obtained from this research project to provide a response to these questions and the hypotheses.

7.1.1. Research Question 1: Response

The first research question was as follows:

> **Research Question 1:** *How does one design and build a GPU drop-in replacement X-Engine system for MeerKAT?*

This dissertation examined previous implementations of GPU X-Engines in radio telescopes (Section 3.2.3) as well as the general architecture of GPU servers (Section 3.3). From this, it was determined that the most important sub-systems to consider when designing this X-Engine were the ingest stage, GPU correlation stage and the pipeline framework connecting these two stages.

The tools available to implement these sub-systems were examined in Section 3.3.1, Section 3.3.2 and Section 3.3.3. The most suitable tool for each sub-system was then chosen.

The final GPU X-Engine prototype ingested data over two Mellanox 40 GbE NICs using the SPEAD2 library (Section 4.6) and correlated this data on a Nvidia GPU using the xGPU correlation library (Section 4.5). The only part of this system that had to be designed from the ground up was the pipeline framework (Section 4.4). This was all combined into a dual socket server. Each socket had ten CPU cores and four RAM channels (Section 4.8). This system could perform the function of two SKARAB X-Engines.

This dissertation has proven that it is possible to design a GPU X-Engine for MeerKAT. Most individual sub-systems already have existing implementations - the main development effort is in combining all these tools into a single system and correctly specifying the hardware components that can handle the required data rates. The CPU domain data reorder required by xGPU was an additional challenge due to the high reorder data rates.

The CPUs on the implemented prototype are nearly fully utilised (17 out of 20 CPU cores are used) while the GPU has excess capacity (75% of the GPU is used). This suggests that the main bottleneck of the implemented prototype is in the ingest and reorder stages. Future designs should consider improving the ingest stage and moving the reorder stage to the GPU to achieve higher data rates.

It was found that small packet sizes(1 KiB) led to high packet and heap loss rates. Packet sizes of 4 KiB and above eliminated this problem.

7.1.2. Research Question 2: Response

The second research question was as follows:

> **Research Question 2:** *How much rack space will this system require?*

The prototype design using a Dell R720 server has one 40 GbE port per U (Section 4.8.2). This is the same density as a SKARAB X-Engine.

The suggested Tyan server will have a density of two 40 GbE ports per U. This is twice that of a SKARAB.

With additional research and development, the transpose operation can be moved to the GPU allowing for a potential four 40 GbE ports per U (Section 6.3.1).

The GPU system requires twice as much power per 40 GbE port when compared to the SKARAB (Section 6.1.2). The Tyan server implementation will consume four times the energy per U when compared to a SKARAB while the suggested four ports per U implementation will consume 8 times the energy.

In order to reduce the power consumption, a hybrid FPGA/GPU system could possibly be developed that would have the same density of four 40 GbE ports per U but a lower power consumption due to the CPU portion of the design being offloaded to the lower power FPGA (Section 6.3.2). More research would have to be done to verify that this suggestion can be implemented.

Newer technology has allowed the physical size of the correlator to shrink significantly, while

drastically increasing power consumption. Additional research would have to be done to determine how to cool this GPU system if it were to be deployed in the MeerKAT server room on scale. This would most likely involve moving from air cooling to water cooling - replacing all heat sinks on the GPUs and processors with custom designed water cooling blocks.

7.1.3. Research Question 3: Response

The third research question was as follows:

> **Research Question 3:** *How does the system cost, performance and power consumption differ between the current FPGA X-Engine and the GPU X-Engine replacement?*

This system was compared to the SKARAB in Section 6.1.

The GPU implementation has the benefit of costing less than half of a SKARAB implementation over the entire system life-cycle (Section 6.1.3). The GPU system uses off the shelf components, making it easier to repair and replace malfunctioning parts than it would be in the case of the custom designed SKARAB board. Additionally, the GPU X-Engine was far quicker to develop and debug than the SKARAB X-Engine.

The main disadvantage is that the GPU system consumes twice as much power per multi-cast stream than a SKARAB consumes (Section 6.1.2).

When considering whether to use a GPU or FPGA X-Engine, a GPU X-Engine should be used in order to reduce hardware costs. If power consumption or cooling is a high priority, then an FPGA solution should be considered.

7.1.4. Hypothesis Results

The hypothesis was as follows:

HYPOTHESIS:

> *It is possible to design a GPU based X-Engine that will perform to the same specifications as the SKARAB X-Engine: for the same level of performance, this GPU system will have a cheaper purchase price, require less rack space, and consume more power than a SKARAB equivalent.*

This investigation found the hypothesis to be mostly valid.

A GPU X-Engine performing most of the same functions as a SKARAB X-Engine was demonstrated (Chapter 5). The GPU system requires half as much rack space when compared to a SKARAB (Section 4.8.2), costs a third of the price (Section 6.1.3) while consuming more

than double the power (Section 6.1.2).

The GPU X-Engine was unable to process 1 KiB packets. To install a GPU X-Engine in the place of a SKARAB X-Engine, the MeerKAT F-Engines need to be modified so that they output 4 KiB packets.

7.2. Recommendations

This section lists recommendations for future work. These recommendations will encompass considerations to be made when designing a new correlator (Section 7.2.1) and research required in order to improve upon this X-Engine's design (Section 7.2.2 and 7.2.3).

7.2.1. Correlator Design Recommendations

The following recommendations should be considered when a new correlator is being designed:

1. The correlator X-Engines should be implemented on GPUs instead of FPGAs when trying to reduce system cost and increase density. FPGA X-Engines should be used if minimising power consumption is a high priority.
2. When using GPU X-Engines, careful consideration should go into designing the correlator cooling system as significantly more power is consumed compared to an equivalent FPGA system.
3. When using a GPU X-Engine, the packet sizes produced by the F-Engines should be set to at least 4 096 bytes to reduce ingest processing requirements at the NIC. This will lead to significantly lower packet drop rates.

7.2.2. X-Engine Research Recommendations: Iterative Improvement

The next logical upgrade to this system was discussed in Section 6.3.1. This proposed X-Engine should be able to process 16 multi-cast streams from four 40 GbE ports. In order to create this system the following steps are recommended:

1. Separate the xGPU library into a correlation kernel and memory copy.
2. Migrate the transpose operation to the GPU.
3. Add a second GPU to the system.
4. Change the single port 40 GbE NICs to dual port 40 GbE NICs.
5. Measure the achievable ingest rate when connecting all four NICs to the network.

7.2.3. X-Engine Research Recommendations: Architectural Shift

Section 6.3.2 suggested a design for an FPGA/GPU hybrid system. This design should greatly reduce the system power requirements. This hybrid system is still in the concept phase and would require more research before it is ready for use. In order to develop this system, it is recommended that the following research be undertaken:

1. Investigate methods to DMA data from the Alveo card to the GPU over the PCIe bus, bypassing the CPU and system memory completely.
2. Evaluate the FPGA resources required to perform the X-Engine buffer, reorder and transpose operations in order to verify that it will fit on an Alveo card running two logical F-Engines.
3. Upgrade the SARAO 40 GbE FPGA core that is used for SKARAB to a 100 GbE core that can be used on the Alveo card.
4. Investigate 100 GbE network architecture - using the Alveo cards will require the entire 40 GbE network to be converted to a 100 GbE network. The network switches and architecture available for 100 GbE would need to be evaluated for suitability.

* * *

This research project has informed the design of the next generation MeerKAT correlator. SARAO is actively investigating certain recommendations made in this dissertation.

References

[1] J. L. Jonas, "The MeerKAT radio telescope," *Proceedings of Science*, vol. 277, 2016. [Online]. Available: https://pos.sissa.it/cgi-bin/reader/contribution.cgi?id=PoS(MeerKAT2016)001

[2] D. R. DeBoer, R. G. Gough, J. D. Bunton, T. J. Cornwell, R. J. Beresford, S. Johnston, I. J. Feain, A. E. Schinckel, C. A. Jackson, M. J. Kesteven, A. Chippendale, G. A. Hampson, J. D. O'Sullivan, S. G. Hay, C. E. Jacka, T. W. Sweetnam, M. C. Storey, L. Ball, and B. J. Boyle, "Australian SKA Pathfinder: A High-Dynamic Range Wide-Field of View Survey Telescope," *Proceedings of the IEEE*, vol. 97, no. 8, pp. 1507–1521, aug 2009. [Online]. Available: http://ieeexplore.ieee.org/document/5164981/

[3] D. R. DeBoer, A. R. Parsons, J. E. Aguirre, P. Alexander, Z. S. Ali, A. P. Beardsley, G. Bernardi, J. D. Bowman, R. F. Bradley, C. L. Carilli, C. Cheng, E. d. L. Acedo, J. S. Dillon, A. Ewall-Wice, G. Fadana, N. Fagnoni, R. Fritz, S. R. Furlanetto, B. Glendenning, B. Greig, J. Grobbelaar, B. J. Hazelton, J. N. Hewitt, J. Hickish, D. C. Jacobs, A. Julius, M. Kariseb, S. A. Kohn, T. Lekalake, A. Liu, A. Loots, D. MacMahon, L. Malan, C. Malgas, M. Maree, Z. Martinot, N. Mathison, E. Matsetela, A. Mesinger, M. F. Morales, A. R. Neben, N. Patra, S. Pieterse, J. C. Pober, N. Razavi-Ghods, J. Ringuette, J. Robnett, K. Rosie, R. Sell, C. Smith, A. Syce, M. Tegmark, N. Thyagarajan, P. K. G. Williams, and H. Zheng, "Hydrogen Epoch of Reionization Array (HERA)," *Publications of the Astronomical Society of the Pacific*, vol. 129, no. 974, p. 045001, apr 2017. [Online]. Available: http://stacks.iop.org/1538-3873/129/i=974/a=045001?key=crossref.fdc8d59fdfae8cafc3f3ec26e4ba6300

[4] M. Amiri, K. Bandura, P. Berger, M. Bhardwaj, M. M. Boyce, P. J. Boyle, C. Brar, M. Burhanpurkar, P. Chawla, J. Chowdhury, J.-F. Cliche, M. D. Cranmer, D. Cubranic, M. Deng, N. Denman, M. Dobbs, M. Fandino, E. Fonseca, B. M. Gaensler, U. Giri, A. J. Gilbert, D. C. Good, S. Guliani, M. Halpern, G. Hinshaw, C. Höfer, A. Josephy, V. M. Kaspi, T. L. Landecker, D. Lang, H. Liao, K. W. Masui, J. Mena-Parra, A. Naidu, L. B. Newburgh, C. Ng, C. Patel, U.-L. Pen, T. Pinsonneault-Marotte, Z. Pleunis, M. R. Ravandi, S. M. Ransom, A. Renard, P. Scholz, K. Sigurdson, S. R. Siegel, K. M. Smith, I. H. Stairs, S. P. Tendulkar, K. Vanderlinde, and D. V. Wiebe, "The CHIME Fast Radio Burst Project: System Overview," *The Astrophysical Journal*, vol. 863, no. 1, p. 48, 2018. [Online]. Available: https://iopscience.iop.org/article/10.3847/1538-4357/aad188

[5] R. Ekers, "The History of the Square Kilometre Array (SKA)," in *Resolving The Sky*. Manchester: Proceedings of Science, 2012. [Online]. Available: https://arxiv.org/ftp/arxiv/papers/1212/1212.3497.pdf

[6] T. Kusel and R. Lord, "SARAO Systems Engineering Management Plan," SSA0000-0000-017 MP rev 1, South African Radio Astronomy Observatory, Cape Town, South Africa, Tech. Rep., 2018.

[7] INCOSE, *Systems Engineering Handbook: A Guide for System Life Cycle Processes and Activities*, 4th ed. John Wiley & Sons, Inc, 2015.

[8] C. Larman, *Agile and Iterative Development: A Manager's Guide*, 1st ed. Addison-Wesley Professional, 2003.

[9] A. R. Foley, T. Alberts, R. P. Armstrong, A. Barta, E. F. Bauermeister, H. Bester, S. Blose, R. S. Booth, D. H. Botha, S. J. Buchner, C. Carignan, T. Cheetham, K. Cloete, G. Coreejes, R. C. Crida, S. D. Cross, F. Curtolo, A. Dikgale, M. S. de Villiers, L. J. du Toit, S. W. Esterhuyse, B. Fanaroff, R. P. Fender, M. Fijalkowski, D. Fourie, B. Frank, D. George, P. Gibbs, S. Goedhart, J. Grobbelaar, S. C. Gumede, P. Herselman, K. M. Hess, N. Hoek, J. Horrell, J. L. Jonas, J. D. Jordaan, R. Julie, F. Kapp, P. Kotzé, T. Kusel, A. Langman, R. Lehmensiek, D. Liebenberg, I. J. Liebenberg, A. Loots, R. T. Lord, D. M. Lucero, J. Ludick, P. Macfarlane, M. Madlavana, L. Magnus, C. Magozore, J. A. Malan, J. R. Manley, L. Marais, N. Marais, S. J. Marais, M. Maree, A. Martens, O. Mokone, V. Moss, S. Mthembu, W. New, G. D. Nicholson, P. C. van Niekerk, N. Oozeer, S. S. Passmoor, A. Peens-Hough, A. B. Pińska, P. Prozesky, S. Rajan, S. Ratcliffe, R. Renil, L. L. Richter, D. Rosekrans, A. Rust, A. C. Schröder, L. C. Schwardt, S. Seranyane, M. Serylak, D. S. Shepherd, R. Siebrits, L. Sofeya, R. Spann, R. Springbok, P. S. Swart, V. L. Thondikulam, I. P. Theron, A. Tiplady, O. Toruvanda, S. Tshongweni, L. van den Heever, C. van der Merwe, R. van Rooyen, S. Wakhaba, A. L. Walker, M. Welz, L. Williams, M. Wolleben, P. A. Woudt, N. J. Young, and J. T. Zwart, "Engineering and science highlights of the KAT-7 radio telescope," *Monthly Notices of the Royal Astronomical Society*, vol. 460, no. 2, pp. 1664–1679, aug 2016. [Online]. Available: https://academic.oup.com/mnras/article/460/2/1664/2608970

[10] F. Camilo, "African star joins the radio astronomy firmament," *Nature Astronomy*, vol. 2, no. 7, p. 594, jul 2018. [Online]. Available: https://doi.org/10.1038/s41550-018-0516-y

[11] K. Grainge, B. Alachkar, S. Amy, D. Barbosa, M. Bommineni, P. Boven, R. Braddock, J. Davis, P. Diwakar, V. Francis, R. Gabrielczyk, R. Gamatham, S. Garrington, T. Gibbon,

D. Gozzard, S. Gregory, Y. Guo, Y. Gupta, J. Hammond, D. Hindley, U. Horn, R. Hughes-Jones, M. Hussey, S. Lloyd, S. Mammen, S. Miteff, V. Mohile, J. Muller, S. Natarajan, J. Nicholls, R. Oberland, M. Pearson, T. Rayner, S. Schediwy, R. Schilizzi, S. Sharma, S. Stobie, M. Tearle, B. Wang, B. Wallace, L. Wang, R. Warange, R. Whitaker, A. Wilkinson, and N. Wingfield, "Square Kilometre Array: The radio telescope of the XXI century," *Astronomy Reports*, vol. 61, no. 4, pp. 288–296, apr 2017.

[12] SKASA, "MeerKAT Science." [Online]. Available: http://public.ska.ac.za/meerkat/meerkat-large-survey-projects

[13] R. Garner, "Hubble Space Telescope – Advanced Camera for Surveys," 2016. [Online]. Available: https://www.nasa.gov/content/hubble-space-telescope-advanced-camera-for-surveys

[14] B. W. Holwerda, S. L. Blyth, A. J. Baker, and t. L. Team, "Looking at the distant universe with the MeerKAT Array (LADUMA)," *Proceedings of the International Astronomical Union*, vol. 7, no. S284, pp. 496–499, sep 2011. [Online]. Available: http://www.journals.cambridge.org/abstract_S1743921312009702

[15] D. R. Lorimer, "Binary and millisecond pulsars," vol. 11, no. 1, pp. 1–90, nov 2008. [Online]. Available: http://www.livingreviews.org/lrr-2008-8http://astro.wvu.edu/people/dunc

[16] G. Hobbs, A. Archibald, Z. Arzoumanian, D. Backer, M. Bailes, N. D. R. Bhat, M. Burgay, S. Burke-Spolaor, D. Champion, I. Cognard, W. Coles, J. Cordes, P. Demorest, G. Desvignes, R. D. Ferdman, L. Finn, P. Freire, M. Gonzalez, J. Hessels, A. Hotan, G. Janssen, F. Jenet, A. Jessner, C. Jordan, V. Kaspi, M. Kramer, V. Kondratiev, J. Lazio, K. Lazaridis, K. J. Lee, Y. Levin, A. Lommen, D. Lorimer, R. Lynch, A. Lyne, R. Manchester, M. McLaughlin, D. Nice, S. Oslowski, M. Pilia, A. Possenti, M. Purver, S. Ransom, J. Reynolds, S. Sanidas, J. Sarkissian, A. Sesana, R. Shannon, X. Siemens, I. Stairs, B. Stappers, D. Stinebring, G. Theureau, R. van Haasteren, W. van Straten, J. P. W. Verbiest, D. R. B. Yardley, and X. P. You, "The International Pulsar Timing Array Project: Using Pulsars as a Gravitational Wave Detector," *Classical and Quantum Gravity*, vol. 27, no. 8, nov 2010. [Online]. Available: http://dx.doi.org/10.1088/0264-9381/27/8/084013

[17] I. Heywood, R. P. Armstrong, R. Booth, A. J. Bunker, R. P. Deane, M. J. Jarvis, J. L. Jonas, M. E. Jones, H.-R. Kloeckner, J.-P. Kneib, K. K. Knudsen, F. Levrier, D. Obreschkow, D. Rigopoulou, S. Rawlings, O. M. Smirnov, A. C. Taylor, A. Verma, J. Dunlop, M. G. Santos, E. R. Stanway, and C. Willott, "MESMER: MeerKAT Search

for Molecules in the Epoch of Reionization," *Proceedings of Science*, mar 2011. [Online]. Available: http://arxiv.org/abs/1103.0862

[18] W. J. G. de Blok, E. A. K. Adams, P. Amram, E. Athanassoula, I. Bagetakos, C. Balkowski, M. A. Bershady, R. Beswick, F. Bigiel, S. L. Blyth, A. Bosma, R. S. Booth, A. Bouchard, E. Brinks, C. Carignan, L. Chemin, F. Combes, J. Conway, E. C. Elson, J. English, B. Epinat, B. S. Frank, J. Fiege, F. Fraternali, J. S. Gallagher, B. K. Gibson, G. Heald, P. A. Henning, B. W. Holwerda, T. H. Jarrett, H. Jerjen, G. I. Józsa, M. Kapala, H. R. Klöckner, B. S. Koribalski, R. C. Kraan-Korteweg, S. Leon, A. Leroy, S. I. Loubser, D. M. Lucero, S. S. McGaugh, G. R. Meurer, M. Meyer, M. Mogotsi, B. Namumba, S.-H. Oh, T. A. Oosterloo, D. J. Pisano, A. Popping, S. Ratcliffe, J. A. Sellwood, E. Schinnerer, A. C. Schröder, K. Sheth, M. W. L. Smith, A. Sorgho, K. Spekkens, S. Stanimirovic, K. van der Heyden, W. van Driel, L. Verdes-Montenegro, F. Walter, T. Westmeier, E. Wilcots, T. Williams, O. I. Wong, P. A. Woudt, and A. Zijlstra, "An Overview of the MHONGOOSE Survey: Observing Nearby Galaxies with MeerKAT," *Proceedings of Science*, sep 2017. [Online]. Available: http://arxiv.org/abs/1709.08458

[19] R. Fender, P. A. Woudt, R. Armstrong, P. Groot, V. McBride, J. Miller-Jones, K. Mooley, B. Stappers, R. Wijers, M. Bietenholz, S. Blyth, M. Bottcher, D. Buckley, P. Charles, L. Chomiuk, D. Coppejans, S. Corbel, M. Coriat, F. Daigne, W. J. G. de Blok, H. Falcke, J. Girard, I. Heywood, A. Horesh, J. Horrell, P. Jonker, T. Joseph, A. Kamble, C. Knigge, E. Koerding, M. Kotze, C. Kouveliotou, C. Lynch, T. Maccarone, P. Meintjes, S. Migliari, T. Murphy, T. Nagayama, G. Nelemans, G. Nicholson, T. O'Brien, A. Oodendaal, N. Oozeer, J. Osborne, M. Perez-Torres, S. Ratcliffe, V. Ribeiro, E. Rol, A. Rushton, A. Scaife, M. Schurch, G. Sivakoff, T. Staley, D. Steeghs, I. Stewart, J. Swinbank, K. van der Heyden, A. van der Horst, B. van Soelen, S. Vergani, B. Warner, and K. Wiersema, "ThunderKAT: The MeerKAT Large Survey Project for Image-Plane Radio Transients," *Proceedings of Science*, nov 2017. [Online]. Available: http://arxiv.org/abs/1711.04132

[20] A. R. Thompson, J. M. Moran, and G. W. Swenson, *Interferometry and Synthesis in Radio Astronomy*, ser. Astronomy and Astrophysics Library. Cham: Springer International Publishing, 2017. [Online]. Available: http://link.springer.com/10.1007/978-3-319-44431-4

[21] SARAO, "MeerKAT joins the ranks of the world's great scientific instruments through its First Light image - Media Statement, 16 July," 2016. [Online]. Available: http://www.ska.ac.za/media-releases/meerkat-joins-the-ranks-of-the-worlds-great-scientific-instruments-through-its-first-light-image/

[22] A. Mioduszewski, "Array Configuration," 2008. [Online]. Available: http://www.hartrao.ac.za/synthesis_school/Miod_Array_Design.pdf

[23] G. Foster, "Imaging and Deconvolution," p. 73, 2014. [Online]. Available: https://github.com/ska-sa/tutorials/blob/77fc2b80f3dc7432b97b847c026723d0cfd64a36/3_Interferometry_Workshop/5_Imaging/imaging.pdf

[24] O. Smirnov, private communication, 2019.

[25] J. R. Manley, "A Scalable Packetised Radio Astronomy Imager," Ph.D. dissertation, University of Cape Town, 2014.

[26] S. M. Ord, B. Crosse, D. Emrich, D. Pallot, R. B. Wayth, M. A. Clark, S. E. Tremblay, W. Arcus, D. Barnes, M. Bell, G. Bernardi, N. D. R. Bhat, J. D. Bowman, F. Briggs, J. D. Bunton, R. J. Cappallo, B. E. Corey, A. A. Deshpande, L. DeSouza, A. Ewell-Wice, L. Feng, R. Goeke, L. J. Greenhill, B. J. Hazelton, D. Herne, J. N. Hewitt, L. Hindson, N. Hurley-Walker, D. Jacobs, M. Johnston-Hollitt, D. L. Kaplan, J. C. Kasper, B. B. Kincaid, R. Koenig, E. Kratzenberg, N. Kudryavtseva, E. Lenc, C. J. Lonsdale, M. J. Lynch, B. McKinley, S. R. McWhirter, D. A. Mitchell, M. F. Morales, E. Morgan, D. Oberoi, A. Offringa, J. Pathikulangara, B. Pindor, T. Prabu, P. Procopio, R. A. Remillard, J. Riding, A. E. E. Rogers, A. Roshi, J. E. Salah, R. J. Sault, N. Udaya Shankar, K. S. Srivani, J. Stevens, R. Subrahmanyan, S. J. Tingay, M. Waterson, R. L. Webster, A. R. Whitney, A. Williams, C. L. Williams, and J. S. B. Wyithe, "The Murchison Widefield Array Correlator," *Publications of the Astronomical Society of Australia*, vol. 32, p. e006, mar 2015. [Online]. Available: https://www.cambridge.org/core/product/identifier/S1323358015000053/type/journal_article

[27] J. Bunton, "ALMA Memo 342 - An Improved FX Correlator," no. December, 1999. [Online]. Available: http://legacy.nrao.edu/alma/memos/html-memos/abstracts/abs342.html

[28] J. D. Bunton, "Ska Correlator Advances," in *The Square Kilometre Array: An Engineering Perspective*. Springer-Verlag, jan 2006, vol. 17, pp. 251–259. [Online]. Available: https://link.springer.com/chapter/10.1007/1-4020-3798-8_23

[29] J. Kocz, L. J. Greenhill, B. R. Barsdell, G. Bernardi, A. Jameson, M. A. Clark, J. Craig, D. Price, G. B. Taylor, F. Schinzel, and D. Werthimer, "A Scalable Hybrid FPGA/GPU FX Correlator," *Journal of Astronomical Instrumentation*, vol. 03, no. 01, p. 1450002, mar 2014. [Online]. Available: http://www.worldscientific.com/doi/abs/10.1142/S2251171714500020

[30] G. Teague, "SKARAB Motherboard: Hardware Description Document," HDD-123702, Peralex Electronics, Cape Town, South Africa, Tech. Rep., 2015.

[31] Peralex, "SKARAB Agile Extreme-Scale Networked Compute Platform." [Online]. Available: https://github.com/ska-sa/skarab_docs/blob/master/peralex/CasperSkarab2017_v1_presentation.pdf

[32] S. Dennehy, "Correlator-Beamformer Design Document," M1200-0000-003, South African Radio Astronomy Observatory, Cape Town, South Africa, Tech. Rep., 2014.

[33] J. R. Manley, M. Welz, A. Parsons, S. Ratcliffe, and R. van Rooyen, "SPEAD: Streaming Protocol for Exchanging Astronomical Data," 2015. [Online]. Available: https://casper.ssl.berkeley.edu/astrobaki/images/9/93/SPEADsignedRelease.pdf

[34] T. van Balla, "MeerKAT Functional Interface Control Document for Correlator Beamformer Visibilities and Tied Array Data," M1000-0001-020, South African Radio Astronomy Observatory, Cape Town, South Africa, Tech. Rep., 2016.

[35] M. P. Rupen, "Cross Correlators & New Correlators," *Eleventh Synthesis Imaging Workshop*, 2008. [Online]. Available: http://www.hartrao.ac.za/synthesis_school/Correlators_2008_Rupen.pdf

[36] P. J. Napier, "The EVLA Project: Ten Times More Capability for the VLA Peter," *Revealing the Molecular Universe: One Antenna is Never Enough, ASP Conference Series*, vol. 356, pp. 65–71, 2006. [Online]. Available: http://adsabs.harvard.edu/full/2006ASPC..356...65N

[37] R. P. Escoffier, "MMA Memo 166: The MMA Correlator," 1997. [Online]. Available: http://legacy.nrao.edu/alma/memos/html-memos/alma166/memo166.html

[38] R. P. Escoffier, G. Comoretto, J. C. Webber, A. Baudry, C. M. Broadwell, J. H. Greenberg, R. R. Treacy, P. Cais, B. Quertier, P. Camino, A. Bos, and A. W. Gunst, "The ALMA correlator," *Astronomy & Astrophysics*, vol. 462, no. 2, pp. 801–810, feb 2007. [Online]. Available: http://www.aanda.org/10.1051/0004-6361:20054519

[39] Xilinx, "UltraScale Architecture Configurable Logic Block User Guide (UG574)," Xilinx, Tech. Rep., 2017. [Online]. Available: https://www.xilinx.com/support/documentation/user_guides/ug574-ultrascale-clb.pdf

[40] S. Leibson and N. Mehta, "Xilinx UltraScale: The Next-Generation Architecture for Your Next-Generation Architecture," Xilinx, Tech. Rep., 2013. [Online]. Available: https://www.xilinx.com/support/documentation/white_papers/wp435-Xilinx-UltraScale.pdf

[41] J. Hickish, Z. Abdurashidova, Z. Ali, K. D. Buch, S. C. Chaudhari, H. Chen, M. Dexter, R. S. Domagalski, J. Ford, G. Foster, D. George, J. Greenberg, L. Greenhill, A. Isaacson, H. Jiang, G. Jones, F. Kapp, H. Kriel, R. Lacasse, A. Lutomirski, D. MacMahon, J. Manley, A. Martens, R. McCullough, M. V. Muley, W. New, A. Parsons, D. C. Price, R. A. Primiani, J. Ray, A. Siemion, V. Van Tonder, L. Vertatschitsch, M. Wagner, J. Weintroub, D. Werthimer, and o. b. o. t. C. Collaboration, "A Decade of Developing Radio-Astronomy Instrumentation using CASPER Open-Source Technology," *Journal of Astronomical Instrumentation*, vol. 05, no. 04, p. 1641001, dec 2016. [Online]. Available: http://www.worldscientific.com/doi/abs/10.1142/S2251171716410014

[42] "ASKAP PAF ADE advancing an L-band PAF design towards SKA," 2012. [Online]. Available: https://www.researchgate.net/publication/261309860

[43] G. Hampson, A. Brown, J. D. Bunton, S. Neuhold, G. Hampson, A. Brown, J. Bunton, S. Neuhold, R. Chekkala, T. Bateman, and J. Tuthill, "ASKAP Redback-3 - An Agile Digital Signal Processing Platform," 2016. [Online]. Available: https://www.researchgate.net/publication/301413339

[44] Nvidia, "Cuda C Programming Guide," Tech. Rep., 2018. [Online]. Available: https://docs.nvidia.com/pdf/CUDA_C_Programming_Guide.pdf

[45] ——, "Tesla V100 PCIe GPU Accelerator Product Brief," Tech. Rep., 2017. [Online]. Available: https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/tesla-product-literature/Tesla-V100-PCIe-Product-Brief.pdf

[46] ——, "Tesla P100 PCIe Data Sheet," Tech. Rep., 2016. [Online]. Available: https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/tesla-product-literature/nvidia-tesla-p100-PCIe-datasheet.pdf

[47] J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone, J. C. Phillips, and J. C. Phillips GPU, "GPU Computing," *Computing. Proceedings of the IEEE*, vol. 96, no. 5, pp. 879–899, 2008. [Online]. Available: https://escholarship.org/uc/item/0cv1p1nc

[48] C. McClanahan, "History and Evolution of GPU Architecture A Paper Survey," *Computer Science*, 2011. [Online]. Available: https://www.semanticscholar.org/paper/History-and-Evolution-of-GPU-Architecture-A-Paper-McClanahan/247980e834f1c8f684d85067402f950930e6af91

[49] J. Kocz, L. J. Greenhill, B. R. Barsdell, D. Price, G. Bernardi, S. Bourke, M. A. Clark, J. Craig, M. Dexter, J. Dowell, T. Eftekhari, S. Ellingson, G. Hallinan,

J. Hartman, A. Jameson, D. MacMahon, G. Taylor, F. Schinzel, and D. Werthimer, "Digital Signal Processing Using Stream High Performance Computing," *Journal of Astronomical Instrumentation*, vol. 04, no. 01n02, p. 1550003, jun 2015. [Online]. Available: https://doi.org/10.1142/S2251171715500038

[50] N. Denman, M. Amiri, K. Bandura, L. Connor, M. Dobbs, M. Fandino, M. Halpern, A. Hincks, G. Hinshaw, C. Hofer, P. Klages, K. Masui, J. Mena Parra, L. Newburgh, A. Recnik, J. R. Shaw, K. Sigurdson, K. Smith, and K. Vanderlinde, "A GPU-based correlator X-engine implemented on the CHIME Pathfinder," in *2015 IEEE 26th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. IEEE, jul 2015, pp. 35–40. [Online]. Available: https://doi.org/10.1109/ASAP.2015.7245702

[51] A. Magro, K. Adami, and S. Ord, "Suitability of NVIDIA GPUs for SKA1_LOW," Institute of Space Science and Astronomy, University Of Oxford, Oxford, Tech. Rep., 2014. [Online]. Available: https://arxiv.org/abs/1407.4698

[52] S. Che, J. Li, J. W. Sheaffer, K. Skadron, and J. Lach, "Accelerating Compute-Intensive Applications with GPUs and FPGAs," in *2008 Symposium on Application Specific Processors*. IEEE, jun 2008, pp. 101–107. [Online]. Available: https://ieeexplore.ieee.org/document/4570793/

[53] C. Cullinan, C. Wyant, T. Frattesi, and X. Huang, "Computing Performance Benchmarks among CPU , GPU , and FPGA," MathWorks, Tech. Rep., 2012. [Online]. Available: http://hgpu.org/?p=7484

[54] Mellanox, "ConnectX-5 EN Card," Mellanox, Tech. Rep., 2018. [Online]. Available: http://www.mellanox.com/related-docs/prod_adapter_cards/PB_ConnectX-5_EN_Card.pdf

[55] D. Scholz, D. Raumer, and F. Wohlfart, "A Look at Intel's Dataplane Development Kit," *Network*, vol. 115, 2014. [Online]. Available: https://doi.org/10.2313/net-2014-08-1_15

[56] A. Renard, "Building a High Performance DSP Framework for GPUs and x86 Hardware: A look at the design of the CHIME X-Engine and Beamformer." in *The Collaboration for Astronomy Signal Processing and Electronics Research Annual Conference*, Boston, 2019.

[57] C. Harris, K. Haines, and L. Staveley-Smith, "GPU Accelerated Radio Astronomy Signal Convolution," *Experimental Astronomy*, vol. 22, no. 1-2, pp. 129–141, 2008. [Online]. Available: https://link.springer.com/content/pdf/10.1007%2Fs10686-008-9114-9.pdf

[58] P. Klages, K. Bandura, N. Denman, A. Recnik, J. Sievers, and K. Vanderlinde, "GPU Kernels for High-Speed 4-Bit Astrophysical Data Processing," in *IEEE 26th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, Toronto, 2015, pp. 164–165. [Online]. Available: https://doi.org/10.1109/ASAP.2015.7245729

[59] A. Baudry, L. Blackburn, B. Carlson, G. Crew, S. Doeleman, R. Esco, L. Greenhill, D. Herrera, J. Hickish, R. Lacasse, R. Primiani, M. Rupen, A. Saez, J. Weintroub, and A. Young, "ALMA Memo 607 - Digital Correlator and Phased Array Architectures for Upgrading ALMA," Smithsonian Astrophysical Observatory, Tech. Rep. December, 2017. [Online]. Available: https://library.nrao.edu/public/memos/alma/main/memo607.pdf

[60] Nvidia, "Nvidia Tesla V100 GPU Architecture: The World's Most Advanced Data Center GPU," Tech. Rep. August, 2017. [Online]. Available: http://www.nvidia.com/content/gated-pdfs/Volta-Architecture-Whitepaper-v1.1.pdf

[61] J. Romein and B. Veenboer, "Extreme Signal-Processing Performance Using Tensor Cores: Astronomical Imaging on GPUs," in *GPU Technology Conference*, 2019. [Online]. Available: https://developer.download.nvidia.com/video/gputechconf/gtc/2019/presentation/s9306-extreme-signal-processing-performance-using-tensor-cores-and-astronomical-imaging-on-gpus.pdf

[62] C. Schollar, "Correlator Data Transport Benchmark Testing of FPGA/GPU Correlators using the HASHPIPE Software and 10Gbit Ethernet," Tech. Rep., 2012. [Online]. Available: https://casper.ssl.berkeley.edu/wiki/images/c/cc/Hashpipe_throughput.pdf

[63] M. D. Cranmer, B. R. Barsdell, D. C. Price, J. Dowell, H. Garsden, V. Dike, T. Eftekhari, A. M. Hegedus, J. Malins, K. S. Obenberger, F. Schinzel, K. Stovall, G. B. Taylor, and L. J. Greenhill, "Bifrost: a Python/C++ Framework for High-Throughput Stream Processing in Astronomy," *Journal of Astronomical Instrumentation*, vol. 6, no. 4, aug 2017. [Online]. Available: https://doi.org/10.1142/S2251171717500076

[64] A. Recnik, K. Bandura, N. Denman, A. D. Hincks, G. Hinshaw, P. Klages, U.-L. Pen, and K. Vanderlinde, "An efficient real-time data pipeline for the CHIME Pathfinder radio telescope X-engine," in *26th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. Toronto: IEEE, jul 2015, pp. 57–61. [Online]. Available: https://doi.org/10.1109/ASAP.2015.7245705

[65] Intel, *Intel® 64 and IA-32 Architectures Optimization Reference Manual*, 2016. [Online]. Available: https://www.intel.co.uk/content/dam/www/public/us/en/documents/manua

ls/64-ia-32-architectures-optimization-manual.pdf

[66] Dell, "Dell EMC PowerEdge R740 and R740xd Technical Guide," 2019. [Online]. Available: https://i.dell.com/sites/csdocuments/Shared-Content_data-Sheets_Documents/en/aa/PowerEdge_R740_R740xd_Technical_Guide.pdf

[67] I. Fielding, "Quote: SKA ROACH01/0102014 R10000-000-000," South African Radio Astronomy Observatory(SARAO), Cape Town, Tech. Rep., 2014.

[68] A. Okecha, "Tender Bid Award: NRF SARAO SCBF 001 2018," South African Radio Astronomy Observatory(SARAO), Cape Town, Tech. Rep., 2019. [Online]. Available: https://www.sarao.ac.za/wp-content/uploads/2019/11/Notice-of-Bid-Award.pdf

[69] N. Jacobs, "Estimate SIM3705," Nazreen Jacobs IT Services and Distribution, Cape Town, Tech. Rep., 2019.

[70] C. Brown, "Quotation ISQ28674," Infinetix Connect, Midrand, Tech. Rep., 2019.

[71] Eskom, "Tariffs & Charges 2019/2020," Eskom, Tech. Rep., 2019. [Online]. Available: http://www.eskom.co.za/CustomerCare/TariffsAndCharges/Documents/CompleteTariff2019web1.pdf

[72] Xilinx, "Breathe New Life into Your Data Center with Alveo Adaptable Accelerator Cards(WP499)," Xilinx, Tech. Rep., 2018. [Online]. Available: https://www.xilinx.com/support/documentation/white_papers/wp499-alveo-intro.pdf