

Towards a Realtime FRB Pipeline at the Northern Cross Radio Telescope

Adrian De Barro^{1,*†}, Alessio Magro^{1†}, Keith Bugeja^{1†}, Giovanni Naldi^{2‡}, Nicola Ragno^{2‡}, Francesco Fiori^{2‡} and Valentina Cesare^{2‡}

¹ *L-Università ta' Malta, Msida, MSD 2080, Malta*

² *Istituto Nazionale di Astrofisica, Istituto di Radio Astronomia, Via Piero Gobetti 101, I-40129 Bologna, Italy*

Abstract

Fast Radio Bursts (FRBs) are among the most intriguing astrophysical phenomena, requiring high-throughput, low-latency detection systems to capture their fleeting signatures. This paper presents the design and implementation of a real-time FRB detection pipeline tailored for the Northern Cross Radio Telescope, which is undergoing a major digital upgrade. Leveraging GPU-accelerated beamforming and transient search algorithms, the system integrates a scalable software architecture capable of processing wide-field data streams in the 400–416 MHz band. We describe the pipeline's modular components, including the data reception, the GPU-based beamforming and correlation, and a proposed orchestration framework for transmitting the synthesised beams to remote memory for downstream scientific applications. Preliminary benchmarks demonstrate significant performance over the acquisition times, enabling real-time processing across multiple synthesised beams.

Keywords

1. Introduction

Fast Radio Bursts (FRBs) are enigmatic, millisecond-duration radio transients originating from cosmological distances, first discovered by Lorimer et al. [1]. Their extragalactic nature, inferred from their large dispersion measures (DMs), exceeding the Galactic contribution, positions them as powerful probes of the intergalactic medium and cosmic baryon content [2, 3]. Despite significant observational progress, including the identification of repeating FRBs [4], the precise astrophysical origins and emission mechanism of FRBs remain largely unknown.

The transient and unpredictable nature of FRBs poses considerable challenges for their detection and characterisation. Their short durations necessitate high time resolution observations, while their large and unknown DMs require computationally intensive de-dispersion across a vast parameter space to maximise signal-to-noise ratio [5]. Furthermore, pervasive terrestrial radio frequency interference (RFI) often mimics astrophysical signals, demanding robust RFI mitigation strategies. Historically, FRB discoveries were often made through offline analysis of archival data, limiting opportunities for immediate multi-wavelength follow-up observation crucial for unravelling their progenitor systems and environments.

The advent of real-time FRB detection pipelines has revolutionised the field, enabling prompt alerts and triggered voltage data captures that provide unprecedented insight into burst microstructure and polarization properties [6]. Several observatories worldwide have successfully deployed such systems, including the realfast pipeline at the Very Large Array [7], CRAFT at the Australian SKA Pathfinder [8], ALFABURST at Arecibo [9], the CHIME/FRB instrument [10], UTMOST [11], FAST [12], and greenburst at the Green Bank Telescope [13]. These efforts underscore the critical importance of real-time processing for maximising the scientific yield from FRB observations. Next-generation facilities

ITADATA-WS 2025: The 4th Italian Conference on Big Data and Data Science – Workshops, September 9–11, 2025, Turin, Italy

*Corresponding author.

†These authors contributed equally.

✉ adrian.debarro@um.edu.mt (A. D. Barro); alessio.magro@um.edu.mt (A. Magro); keith.bugeja@um.edu.mt (K. Bugeja); gnaldi@ira.inaf.it (G. Naldi); nicola.ragno@inaf.it (N. Ragno); francesco.fiori@inaf.it (F. Fiori); valentina.cesare@inaf.it (V. Cesare)



© 2025 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).



Figure 1: Top view of the Northern Cross radio telescope with the two perpendicular arms along the East-West and North-South direction.

such as the Square Kilometre Array (SKA), the Next Generation VLA (ngVLA) and the Deep Synoptic Array 2000 (DSA-2000) are being designed with advanced transient detection capabilities, promising to dramatically increase FRB discovery rates and enable detailed studies across a wide range of frequencies and sensitivities [14, 15, 16]. The CHIME/FRB instrument also continues to be a prolific discoverer of FRBs, with ongoing efforts to further enhance its capabilities [17, 18]. Additionally, projects like the MeerTRAP at MeerKAT aim to further expand our observational reach for these elusive phenomena [19, 20].

The Northern Cross Radio Telescope, a T-shaped radio interferometer (see Figure 1) operating at a central frequency of 408 MHz, possesses a large collecting area ($\sim 30,000 \text{ m}^2$) and a wide field of view, making it well suited for FRB detection [21]. Following significant upgrades, the Northern Cross has recently demonstrated its capability in FRB observations, including the detections of repeating sources [21]. The instrument is now being fully refurbished, including mechanical and analogue restoration and upgrades, and the installation of digital systems and a High Performance Computing (HPC) compute cluster that will be capable of performing FRB searches utilising the full instrument. To enable this, the development of a dedicated real-time FRB detection pipeline is essential.

In this paper, we highlight ongoing efforts to implement a digital backend pipeline that supports real-time FRB detection at the Northern Cross Radio Telescope. Our contributions are threefold. First, we propose an improved data acquisition (DAQ) system, building on the design in [22], capable of handling higher packet throughputs via a multi-threaded, multi-producer/multi-consumer architecture optimized for low-latency data ingestion. Second, we present a GPU-accelerated beamforming and correlation pipeline, including a beamforming kernel that achieves throughput exceeding $3\times$ the real-time acquisition rate, and a correlation kernel that operates at $23\times$ real-time. Third, we introduce the architecture for a distributed orchestration framework designed to synchronize multiple GPU pipelines with downstream FRB detection services. This orchestration layer leverages RDMA-based data movement and a publisher/subscriber model to ensure batch-level synchronization between components across heterogeneous compute environments.

The remainder of this paper is organised as follows:

- Section 2 describes the Northern Cross Radio Telescope and the ongoing hardware and software upgrades;
- Section 3 presents improvements to the DAQ library used in the Data Reception layer;
- Section 4 details the GPU pipeline for beamforming and correlation computation;
- Section 5 proposes an orchestration framework for transmitting synthesised beams downstream;
- Section 6 summarises the work and discusses planned future developments.

2. The Northern Cross Radio Telescope

The Northern Cross Radio Telescope, also known as "Croce del Nord", is a T-shaped radio interferometer located near Medicina, Bologna, Italy. Inaugurated in 1964, it was designed to observe cosmic radio waves at central frequencies of 408 MHz with an initial bandwidth of 2.5 MHz, later widened to 16 MHz for pulsar research. Its vast collecting area makes it one of the world's largest transit radio telescopes and the largest UHF-band antenna in the Northern Hemisphere. The telescope consists of two perpendicular arms: an East-West (E-W) arm, a single cylindrical-parabolic reflector 564 m long and 35 m wide, and a North-South (N-S) arm composed of 64 smaller cylindrical-parabolic antennas, totalling 640 m in length. As a transit instrument, it is mechanically steerable only in declination, observing celestial sources as they cross the local meridian.

The Northern Cross has a rich history in radio astronomy. In recent years it has undergone significant upgrades to enhance its capabilities, particularly for transient phenomena like FRBs. The "Northern Cross Fast Radio Burst Project" is a dedicated initiative to leverage this instrument for FRB Research

2.1. The Northern Cross Fast Radio Burst Project

Up until 2024, the Northern Cross underwent a significant refurbishment, particularly to parts of the N-S arm. An initial refurbishment of eight parabolic cylindrical reflectors of the N-S arm was carried out between 2006 and 2009 as a technological demonstrator for the SKADS-FP6 project, testing new technologies like RF over fibre links for signal transport and software beamforming. Subsequently, the entire N-S arm underwent refurbishment, following similar technological developments. Since 2000, the N-S arm has been dedicated in parallel to the detection and monitoring of FRBs and Space Debris [23, 24]. This involved the installation of Low Noise Amplifiers (LNAs) on the focal line of the N-S cylinders, with signals transmitted via RF over fibre to acquisition boards for digitisation and channelisation. The system utilised an analogue beamformer, grouping 16 dipoles (one receiver) together within each cylinder to produce four beams per cylinder. The FPGA-based channelisation provided a 16 MHz bandwidth with a 781.25 kHz channel width, leading to a digital beam. A second channelisation further refined the data to a final time resolution of 134 μs and 14 kHz channel resolution. Pilot observations of pulsars, such as B0329+54, were conducted to characterise the system's performance and forecast FRB detectability [21].

2.2. Digital and Software Backend

The FRB search effort at the Northern Cross prior to the latest upgrades (as discussed in Section 2.3) were primarily enabled by a dedicated digital and software backend. The data acquisition system processed signals from the N-S arm. The core of the FRB search was performed using an adaptation of the SPANDAK pipeline [25]. The pipeline leveraged `rfifind` from the PRESTO software suite [26] for real-time RFI masking. When needed for individual filterbanks with high RFI, the Inter-Quartile Range Mitigation (IQRM) real-time adaptive RFI masking was also employed [27]. Following RFI mitigation, `Heimdal1` [28] was used to dedisperse the data across a wide range of DMs, typically 20 to 3000 pc cm^{-3} , for single pulses. A fine DM sampling was adopted, with approximately 25,000 DMsteps, and searches were conducted for bursts with a maximum width of around 70 ms. Candidates identified by `Heimdal1` with a signal-to-noise ratio (SNR) greater than 7 were then subjected to further selection criteria based on their SNR, DM, burst width ($\Delta t \leq 141.6$ ms), the maximum number of allowed candidates within a time window, and the minimum number of distinct boxcar/DM trial clusters into a candidate. Plausible FRB candidates were then validated using `FETCH` [29], a convolutional neural network trained to distinguish between RFI and genuine astrophysical signals. Finally, every candidate that passed all selection steps underwent a deep investigation with human inspection [24, 21].

2.3. Next Generation Croce del Nord

The Northern Cross is currently undergoing a significant upgrade under the "Next Generation Croce del Nord" (NG-Croce) program. The upgrade aims to greatly improve the overall capabilities of the radio telescope, enhancing its field-of-view, sensitivity and resolution. The core of the upgrade lies in the new digital backend and HPC cluster. A total of 672 analogue inputs (256 from the N-S arm and 416 from the E-W arm) will be digitised by the FPGA boards. Real-time (multi)beam forming and FRB searches will be performed by an 18-node HPC cluster, with each node equipped with dual multi-core CPUs, two NVIDIA L40s GPUs, 2 TB of RAM, and SSDs. This powerful computing infrastructure allows for advanced signal processing. 8 PB of disk storage is also available for immediate or short-term processing. The software system for the upgraded Northern Cross builds upon previous efforts while incorporating new capabilities. It is designed to handle the increased data rates and complexity from the enhanced hardware. While NG-Croce upgrades both arms, this paper focuses specifically on the N-S arm and its 256 antennas.

2.4. Digital and Software processing

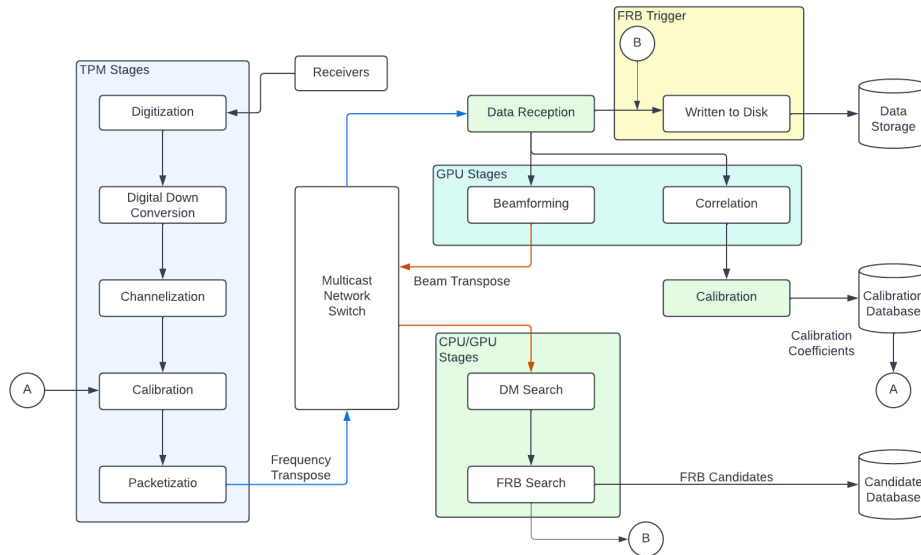


Figure 2: Overall orchestration diagram of the planned real-time FRB pipeline

The system architecture is designed to support flexible and scalable signal processing for multiple Key Science Projects (KSPs), including FRB detection and Space Debris (SD) monitoring. A high-level architecture diagram (see Figure 2) illustrates the clear separation between the beamforming/correlation servers and the downstream FRB/SD/other KSP servers. The beamformer/correlator subsystem generates multiple beams that can be simultaneously distributed to all active KSP pipelines. While SD observations require a transmitter and thus cannot piggyback on FRB operations, FRB detection can leverage SD observational data. The architecture is modular, allowing for the integration of additional KSP pipelines in the future.

The digital backend comprises 21 FPGA-based boards, each capable of processing signals from 32 receivers across two FPGAs per board. Incoming analog signals are digitised at 800 MSPS, digitally down-converted to 50 MSPS, and channelised into 4096 frequency channels. These channelised raw voltages are packetised using the SPEAD protocol [30], with each packet containing 128 frequency channels from one spectrum for 16 antennas. Data are transmitted over 40 Gbps links to a central 100 GbE switch.

Channelised antenna voltages are then routed to the beamforming/correlation servers, which are

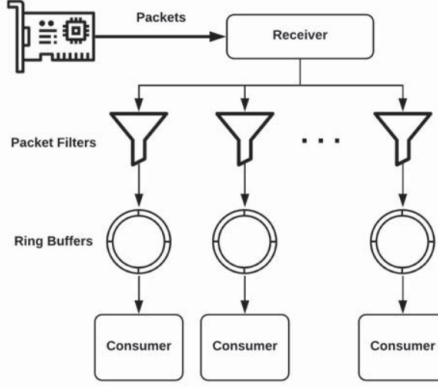


Figure 3: High-level architecture design of the DAQ system. Packets received are pushed onto a consumer ring buffer or filtered. Non-filtered packets are aggregated and passed to the consumer for processing. From [22].

partitioned across frequency bands. Each server is responsible for beamforming and correlating its assigned frequency channels. These servers also implement a transient buffer to store raw voltages temporarily. Upon detection of an FRB or SD event, the buffer can be triggered to persist the relevant raw data for detailed offline analysis.

For FRB detection, a subset of the beams is forwarded to dedicated servers running Heimdall, a real-time transient detection pipeline. This setup enables efficient and scalable FRB searches across the telescope’s field of view.

3. Data Reception

The Data Acquisition (DAQ) library [22] was originally developed for the Aperture Array Verification System (AAVS) in the SKA-Low technology pathfinder. It ingests raw Ethernet frames at high throughput via Linux’s `PACKET_MMAP` interface and provides a modular API for dynamically loaded, consumer-specific logic.

Consumer logic is fully decoupled from the DAQ core. Each consumer is compiled as a shared library, loaded at runtime, and encapsulates both packet-filtering and processing callbacks. The DAQ spawns Network Receiver (NR) threads to pull raw frames from the kernel’s `PACKET_MMAP` buffer, and Packet Consumer (PC) threads—one per consumer—to retrieve filtered packets. PCs aggregate packets for each time interval, invoke the user-defined callback on the full batch, then clear their buffer and resume consumption. Multiple NRs may work on the same ring buffer, synchronising via a fine-grained test-and-test-and-set spin lock with exponential back-off to minimise cache coherence traffic and contention. Figure 3 illustrates this high-level flow.

Although the DAQ remained reliable under moderate load, high-throughput, multi-threaded scenarios exposed several bottlenecks:

- Callbacks block packet processing, creating backpressure;
- Fine-grained spin-locks degrade performance under contention;
- The multi-producer single-consumer (MPSC) ring buffer, limits scalability;
- Only data incoming through a single network interface is supported.

To address these bottlenecks, we applied targeted architectural enhancements to improve scalability and throughput. The following subsections detail each enhancement. Figure 4 illustrates the upgraded DAQ subsystem.

3.0.1. MCS Lock-Based Ring Buffer

Although the test-and-test-and-set spin-lock handled moderate loads, it faltered under heavy contention. When many NR threads race a single, slow-consuming Packet Consumer, lock acquisition spikes.

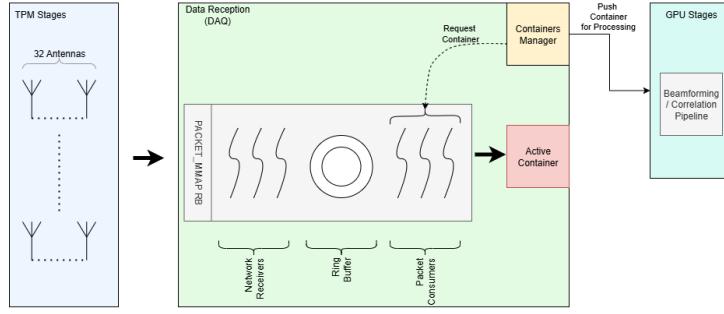
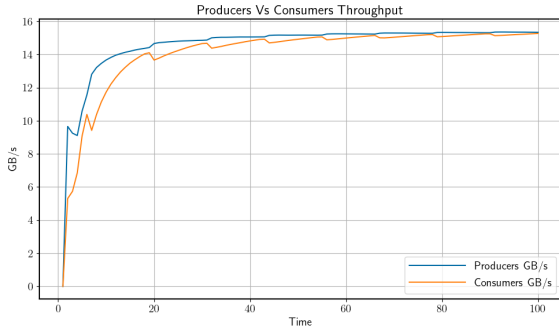


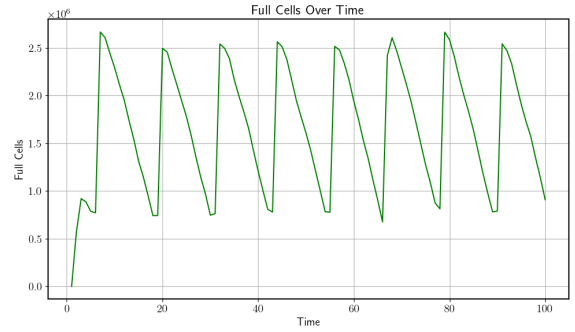
Figure 4: High-level architecture of the improved DAQ subsystem. The upgraded DAQ supports multiple NR and PC threads that share a ring buffer. NRs enqueue packets on the ring buffer, and the PCs process them and sort them into containers. The ContainersManager dispatches full containers to the consumer’s callback.

Repeated reads of the same cache line inflate coherence traffic, and the lack of fairness guarantees risks thread starvation. To solve these issues, we adopted the Mellow–Crummey and Scott (MCS) lock [31].

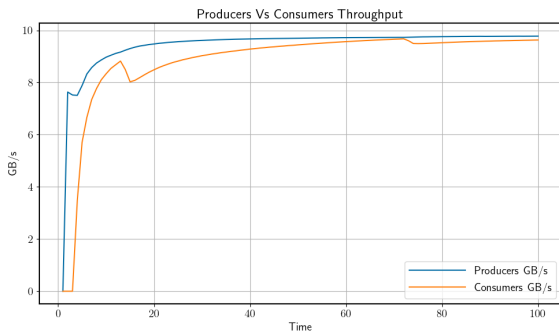
In an MCS lock, each thread spins on its own queue node, dramatically cutting memory-bus contention. The queue-based protocol enforces FIFO ordering—eliminating starvation—and preserves $O(1)$ handoff under contention, thus maintaining throughput and bounding latency.



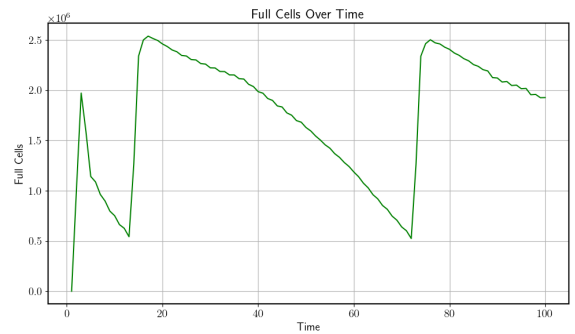
(a) Total Pushed vs Pulled GB/s



(b) Full Cells Over Time



(c) Total Pushed vs Pulled GB/s



(d) Full Cells Over Time

Figure 5: Comparative benchmarks for MPMC ring buffer utilising the original spinlock vs the newly implemented MCS lock, MCS shows smoother throughput and regular buffer cycling under high contention. (a): MCS memcpy throughput scaling; (b): MCS ring buffer occupancy. (c),(d): Spin lock equivalents.

3.1. Multi-Producer Multi-Consumer Ring Buffer

Originally, the DAQ’s ring buffer supported a multi-producer single-consumer model. To meet higher throughput demands, we extended it to a multi-consumer design. Since multiple producers were already supported, the enqueue (push) logic remains unchanged.

In the updated design, each PC thread maintains a thread-local variable that tracks the last cell from which it successfully retrieved data. When a PC thread attempts to pull a new packet, it first copies the current value of the global tail pointer into its local variable. The thread then attempts to acquire a lock on the corresponding cell and retrieve its contents. If the cell is found to be empty, the lock is released, and the thread proceeds to acquire a lock on the global tail pointer. If there are still occupied cells (i.e., the number of full cells is greater than zero), the tail is incremented, and the thread retries the pull operation.

This design ensures thread-safe access to the ring buffer and allows multiple consumers to operate concurrently without data corruption or race conditions. Pseudocode for the pull procedure is provided in Algorithm 1.

Algorithm 1 pull()

```

1: while true do
2:   local_consumer ← current consumer index
3:   acquire lock on cell at local_consumer
4:   if cell is empty (size == 0) then
5:     release cell lock
6:     acquire lock on consumer index
7:     if there are full cells then
8:       advance consumer index (wrap around)
9:     else
10:      sleep briefly
11:    end if
12:    release consumer lock
13:    continue
14:  else
15:    break
16:  end if
17: end while
18: data_size ← cell size at local_consumer
19: data_ptr ← pointer to cell data
20: return data_size

```

The extended MPMC ring buffer was implemented using both the original DAQ spinlocks and the newly introduced MCS locks. Both implementations were benchmarked with 8 producers and 8 consumers. The cell size was set to 8192 bytes to simulate the size of expected payload for packets received from the digital backend, and each test was conducted over a duration of 100 seconds. Figure 5 presents the performance of the MCS-based ring buffer against the original spinlock design. The MCS lock achieves and sustains higher and more balanced producer-consumer throughput (a), while also maintaining regular, cyclical buffer usage (b). In contrast, the spinlock implementation exhibits degraded and unstable performance under higher thread contention. The adoption of the MCS lock results in a performance improvement of approximately 60%.

3.1.1. Container Pool with Asynchronous Callbacks

In the original design, the PC thread was required to wait for the consumer callback to finish processing the aggregated data before consuming more packets from the ring buffer. This prevents overwriting any data that the consumer has not processed yet. Under high-packet ingress scenarios, this latency led to packet drops. To decouple the two operations, we introduce a container pool. The container pool maintains a number of packet aggregations (of an arbitrary length) which we refer to as containers. Each container maintains a four-state lifecycle (EMPTY → WRITING → PROCESSING → READY), managed via an atomic counter. The ContainersManager maintains a list of available containers and supplies

them to PC on request.

Once a container is ready for processing, the callback function is called asynchronously. The container is passed to the callback function using a `MetaData` structure that is initialised using a smart pointer. On exiting the callback, the structure instance is automatically destroyed, changing the container's state to `READY` automatically. This indicates to the `ContainerManager` that the container is ready for clearing. Containers recycling procedure is handled by a secondary thread executing within the `ContainersManager`. The containers recycler periodically loops through all containers and checks for containers in `READY` state. These containers are cleared and introduced back on the list of availability.

3.1.2. Multi-Consumer Container Caching

For multiple PC to operate concurrently, container retrieval has to be thread-safe; ensuring that all PC write packets within the same arbitrary range of time samples to the same container. The `ContainerManager` is extended to implement a thread-safe map of active containers. When a PC encounters a timestamp outside the range of its current active container, it requests a new container from the `MultiConsumerContainerManager` via the `getContainer` function.

Internally, the manager maintains a cache of active containers. If the cache cannot fulfil a PC request, a new one is acquired from the `ContainerPool`. This container is wrapped in a cache context structure, which stores metadata including the timestamp range it covers, the number of currently active threads (AT), and the total number of threads that have requested this container (TT). The new container is also added to the cache. Upon switching containers, the PC decrements the AT counter of the previously used container. Thread safety for this cache is enforced via a lock that serializes access to the cache. This mechanism ensures that only the first request would invoke a new container, while all other PC retrieve the same instance from the cache. The pseudocode for this process is presented in Algorithm 2.

Containers in the cache are marked ready for processing when their AT count reaches zero. A secondary thread has been introduced in the new container manager to periodically check active containers cache for AT state equal to 0. Once found, these containers are pushed for processing.

Algorithm 2 Acquire or Create Container

```
1: procedure GETCONTAINER( $ts, out$ )
2:   Lock containers_mtx
3:    $base \leftarrow (ts \div N) \times N$ 
4:   for all  $(k, ctx)$  in active_map do
5:     if  $k = base$  then
6:        $out \leftarrow ctx$ 
7:        $ctx.at \leftarrow ctx.at + 1$ 
8:        $ctx.tt \leftarrow ctx.tt + 1$ 
9:       return SUCCESS
10:    end if
11:  end for
12:   $idx \leftarrow \text{NEXT}$ 
13:  if  $idx < 0$  then
14:    return NO_CONTAINER
15:  end if
16:   $c \leftarrow \text{GET}(idx)$ 
17:  SWAPSTATE( $c$ , EMPTY, WRITING)
18:   $ctx \leftarrow \text{INITNEWCONTEXT}$ 
19:  active_map[base]  $\leftarrow ctx$ 
20:   $ctx.at \leftarrow 1, \quad ctx.tt \leftarrow 1$ 
21:   $out \leftarrow ctx$ 
22:  return NEW
23: end procedure
```

4. The Beamforming/Correlator Process

Once an aggregate of packets triggers the callback function, the data is transferred to the beamforming pipeline. The beamforming pipeline and the callback operate asynchronously in an effort to reduce the latency of releasing the container back to the DAQ system.

4.1. Beamforming Pipeline

The beamforming pipeline is composed from five main classes:

1. **CudaContext**
2. **PipelineMemoryPool**
3. **BeamformingPipelineContext**
4. **BeamformingPipelineManager**
5. **PipelineAllocationDetails**

The CudaContext handles low-level CUDA initialisation, setting up the environment for GPU operations. The BeamformingPipelineContext, is responsible for initialising and coordinating all components of the pipeline. All interactions from the consumer side go through the BeamformingPipelineContext, which serves as the main interface for consumers.

The PipelineMemoryPool manages memory allocations used to store both container input data and the resulting beamformed output. It functions similarly to the ContainerPool, maintaining a list of reusable buffers for efficient memory management.

The BeamformingPipelineManager controls the execution of the beamforming process. The pipeline runs on a dedicated thread managed by this class. Data intended for processing is pushed onto an atomic queue, which the beamforming thread polls routinely for new jobs. Upon retrieving a job, the thread processes it asynchronously.

An instance of `PipelineAllocationDetails` stores pointers for the allocation used in GPU beamforming including both host and device buffers. The host buffers are pinned memory allocations, required because the pipeline executes asynchronously. As such, the data residing in the containers (as populated by the DAQ) is first copied into host-side buffers and then asynchronously transferred to the device buffers. Additionally, three CUDA events are initialised per instance to synchronise GPU operations across three distinct CUDA streams.

4.2. Asynchronous Kernel Execution

Execution of the kernel is distributed across three different CUDA streams to amortise the cost of copying data between the host and device. Synchronisation between the streams is orchestrated using CUDA events associated with the memory allocations (represented as E1, E2, and E3 in Figure 6). Once the output results have been successfully copied to the host, a callback is triggered to handle transferring the data to the required destination.

4.3. Beamforming Kernel

The beamforming process is implemented as a matrix-vector multiplication involving N_a antennas and a coefficient matrix of size $N_B \times N_a$, producing N_b beams per frequency channel. Each entry in the coefficient matrix combines a per-antenna, per-frequency calibration coefficient that corrects for instrumental effects with a phase delay term that determines the beam pointing. The phase delay term is obtained by passing a frequency-independent phase delay for each antenna and beam pointing, from which the kernel computes the corresponding frequency-dependent phase delays. These frequency-dependent delays are then combined with the per-antenna calibration coefficients, to generate the full coefficient matrix.

In this kernel, we adapt and extend previous approaches [30, 32] to improve the performance of beam synthesis. Computation is parallelised in three dimensions: time samples (x-dimension), frequency channels (y-dimension), and beams (z-dimension). With the availability of larger device memory on modern GPUs, we can scale parallelization further.

We restrict each computation block to process 32 time samples for the current maximum of 256 antennas, for a single channel and a single beam. The kernel supports unordered input data as provided by the data reception component (Tile/TimeSample/Channel/Antenna), eliminating latency arising from data reordering prior to beamforming.

This configuration requires 8 warps per block, with each warp leveraging warp-level intrinsics to compute partial beams. Threads within a block collaboratively compute each beam in a monotonic fashion, where each thread is responsible for the contribution of a single antenna across multiple time samples. This eliminates the need for shared memory to store antenna coefficients, which are instead held in thread-local registers. Shared memory is reserved solely for storing partial sums. The partial sums for all 32 beams (8 partial sums per beam)—are then reduced by 8 threads. Each warp is assigned the reduction of 4 beams using `__shfl_down_sync` operations for intra-warp communication.

Where applicable, fused multiply-add (FMA) instructions were employed to optimize floating-point throughput and minimize numerical error during accumulation. The pseudocode for the beamforming kernel is provided in Algorithm 3.

4.3.1. Beamforming Performance

Experimental benchmarks for the beamforming kernel were run inline with the requirements defined for the first milestone, that is, 512 channels, 256 antennas and 256 beams. The number of time samples for each channel were varied between 2048 and 32768 time samples. In all scenarios the kernel sustains a $3\times$ real-time speed-up. The results have been tabulated in Table 2. A detailed summary of the kernel's performance metrics is provided in Table 1.

While the beamforming kernel achieves faster-than-real-time execution for the current configuration, increasing the timesample count to 65536, while maintaining the other parameters constant results in

Algorithm 3 Beamforming Kernel Pseudocode

```
1: Determine:
2:  $beam\_idx \leftarrow$  beam index
3:  $chan\_idx \leftarrow$  channel index
4:  $ts\_block \leftarrow$  time block index
5:  $ant\_idx \leftarrow$  antenna index
6:  $tile\_idx \leftarrow$  tile index

7: Compute:
8:  $weight \leftarrow delay\_coeff \cdot calibr\_coeff$ 

9: Collaborative Beamforming:
10: for all local time samples  $t$  in block do
11:   Get antenna Input  $input$ 
12:    $contrib \leftarrow weight \cdot input$ 
13:   Perform warp-level reduction to sum  $contrib$ 
14:   if thread is warp leader then
15:     Store result in shared memory at warp index
16:   end if
17: end for
18: Synchronize all threads
19: for all thread group of size 8 do
20:   Load partial sums from shared memory
21:   Perform intra-group reduction using shuffle
22:   if thread is group leader then
23:     compute  $beam\_power$ 
24:      $output \leftarrow clamp(beam\_power)$ 
25:     Write  $output$  to output buffer
26:   end if
27: end for
```

Test Device: NVIDIA L40S	
Kernel compute utilisation	85%
Warp efficiency	100%
Shared memory throughput	170.27 GB/s
Major issue stall	No eligible warps (51.22%)
Issued instructions per cycle	1.95
Shared memory efficiency	50.1%
L2 Hit Rate	55.78%
L1/TEX Hit Rate	85.80%
FLOPS	81.77 TFLOPS/s

Table 1
Beamforming kernel performance analysis on NVIDIA L40S

GPU memory exhaustion. This limits the extent of to which we are able to increase the data volume per GPU for improved utilisation. Therefore, although the beamformer operates with a performance margin, memory constraints dictate the maximum manageable workload per GPU. This suggests that further optimisation of the beamforming kernel, particularly in terms of memory efficiency, may enable processing of larger time windows per GPU, thereby improving hardware utilisation. Exploring alternative tiling strategies or memory layouts could help mitigate current memory constraints and allow for increased throughput without exceeding GPU limits.

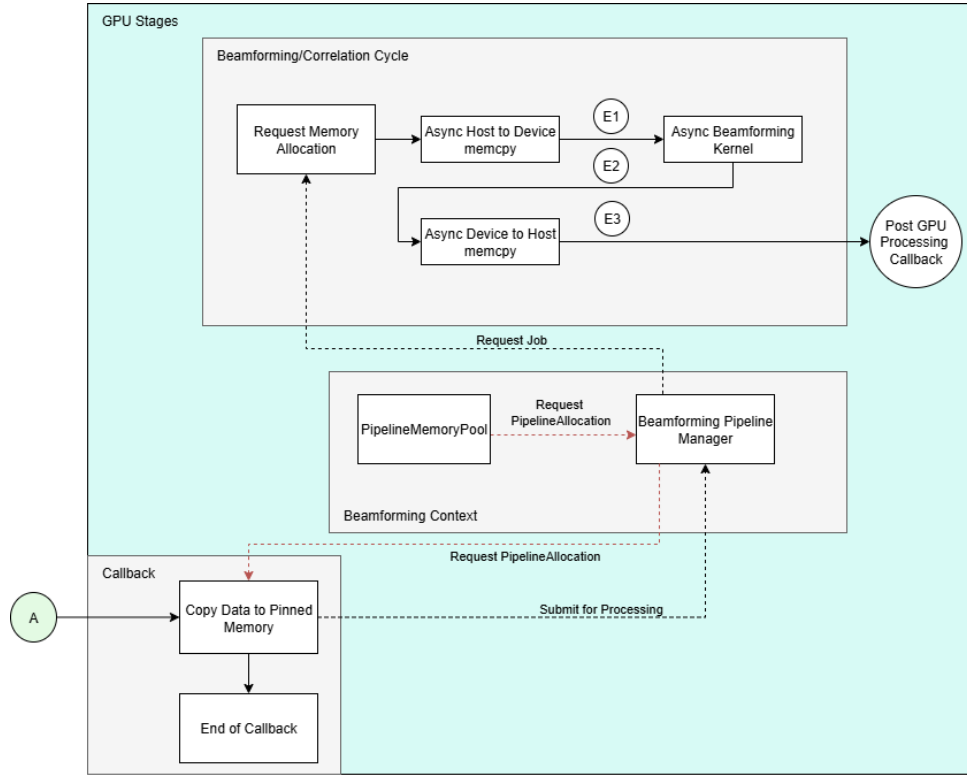


Figure 6: Beamforming/Correlation Cycle: The pipeline periodically checks for available data to beamform, if found, the PipelineAllocation details are processed for beamforming. **Callback triggered process:** The callback function (A) is triggered by the DAQ, which in turn passes a container. The callback function requests a memory allocation from the PipelineMemoryPool and copies the data on the host memory allocation. The allocation details are then submitted to the BeamformingPipelineManager for processing.

Time-samples per Channel	Duration (ms)	Acquisition Time (s)	Speedup
2048	214	0.66	3×
4096	427	1.3	3×
8192	859	2.64	3×
16384	1737	5	3×
32768	3513.28	10.58	3×

Table 2

Beamforming kernel performance for 256 antennas, 256 beams and 512 channels at varied time-sample counts per channel. In each case the kernel runs at a consistent 3× speedup over acquisition time.

4.4. Correlation Kernel

Cross-correlation is inherently a triangular computation problem, as it involves computing the outer product of signals received from multiple antennas, resulting in a symmetric matrix. This symmetry can be exploited to reduce redundant calculations, enabling various performance optimisations. A range of hardware platforms, including FPGAs and GPUs, have been employed for cross-correlation tasks [33, 34]. With the introduction of Tensor Cores, it is now possible to leverage high-throughput matrix operations for this purpose [35]. However, utilising Tensor Cores typically requires reordering input data into a matrix-compatible format, which can introduce pre-processing overheads that may offset the performance benefits in some applications.

In our implementation, we adopt the hardware-managed strategy used by [36], which explicitly leverages matrix symmetry to halve the number of required computations. Since only the upper (or lower) triangular portion of the matrix needs to be computed, a primary challenge lies in determining

which products to evaluate and which to omit—without introducing warp divergence during execution. To address this, we implement a two-level tiling strategy. The computation is decomposed into tiles and register-level products: tiles define the 2D thread block structure, while registers manage the subset of matrix entries computed by each thread. This approach ensures balanced workload distribution while preserving high memory throughput. Additional implementation details can be found in [36].

We also developed a shared-memory-based version of the kernel that incorporates fused multiply-add (FMA) operations and utilizes 3D textures to load the input data. This design allows for efficient memory access and requires less arithmetic operations when determining and retrieving the antenna indices. However, this imposes a limitation on the number of time samples that can be processed due to the hardware limit of 3D textures (maximum dimensions of $16384 \times 16384 \times 16384$). Consequently, this variant currently supports a maximum of 16,384 time samples per cross-correlation pass. The shared-memory implementation has not yet been fully benchmarked or optimised for the NVIDIA L40S GPU, and this remains a subject of ongoing work.

4.4.1. Cross-Correlation Performance

The performance of the cross-correlation kernel was evaluated for various tile and register size combinations. For every experimental run the kernel processed 16384 time samples, for 512 channels and 256 antennas. Each configuration was executed 10 times, and the average execution time was recorded. The highest performance was observed with a tile size of 8×8 threads per block, where each thread computed a single cross-correlation product. Under this configuration the kernel takes ~ 245 ms, which is 23x faster than the expected data throughput for 256 antennas. A detailed summary of the kernel’s performance metrics is provided in Table 3.

Test Device: NVIDIA L40S	
Kernel compute utilisation	>90%
Warp efficiency	100%
Memory throughput	45.43%
Major issue stall	No eligible warps (51.68%)
Issued instructions per cycle	1.93
L2 Cache Hit Rate	99.93%
L1/TEX Hit Rate	29.29%
FLOPS	289.3 TFLOPS/s

Table 3

Correlation kernel metrics for when profiled on the L40S

While the cross-correlation kernel executes approximately $23\times$ faster than the real-time data rate for 256 antennas and 512 channels, this performance headroom allows flexibility in processing larger antenna arrays or higher channel resolutions, or alternatively, reducing the number of GPUs/servers required. However, this trade-off is constrained by the total incoming data bandwidth, which must be carefully balanced against GPU compute utilisation. Further analysis is needed to determine the optimal configuration that minimizes hardware usage while satisfying real-time constraints.

5. Transmission of Beams

The GPU stages outlined in Figure 2 operate across multiple servers (two GPUs per server), with each GPU responsible for processing a subset of the total channel count. The synthesised beams must then be transmitted downstream to the FRB pipeline for processing.

To maintain correct data alignment and timing, all components in the processing pipeline must operate synchronously: the FRB pipeline should begin processing only after it has received beam data from all beamforming pipelines. Transmission between the GPU pipelines and the FRB pipeline will occur via RDMA, a zero-copy mechanism that allows direct memory access between remote nodes, minimizing transfer latency and CPU overheads.

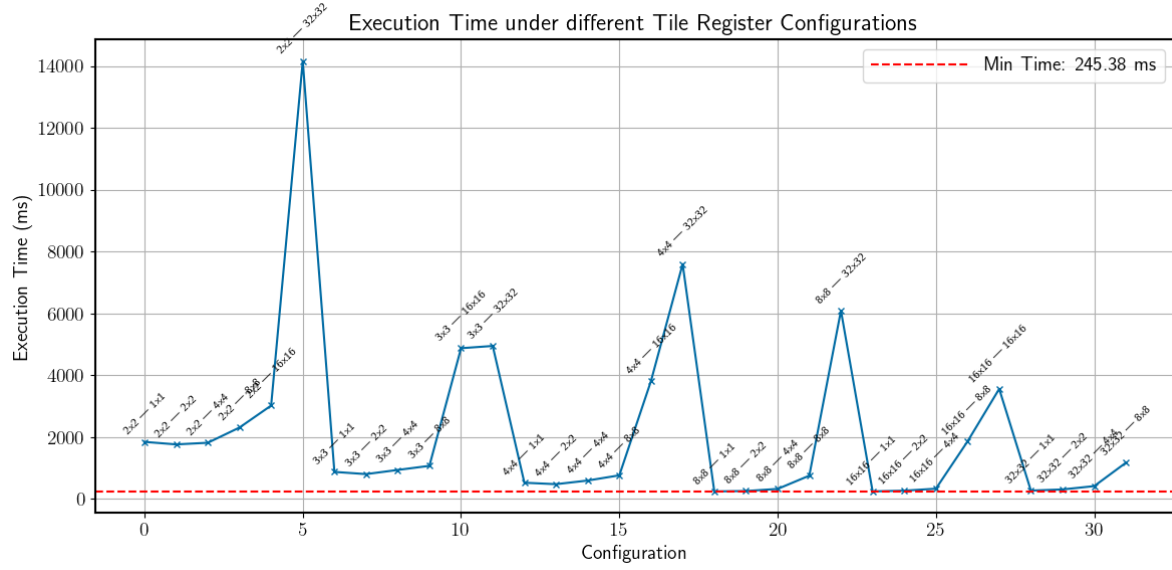


Figure 7: Experimental runs for cross-correlation performance for different tile and register combinations for 256 antennas, 512 channels and 16384 time samples. Best performance achieved for tile size of 8×8 and a register size of 1×1 .

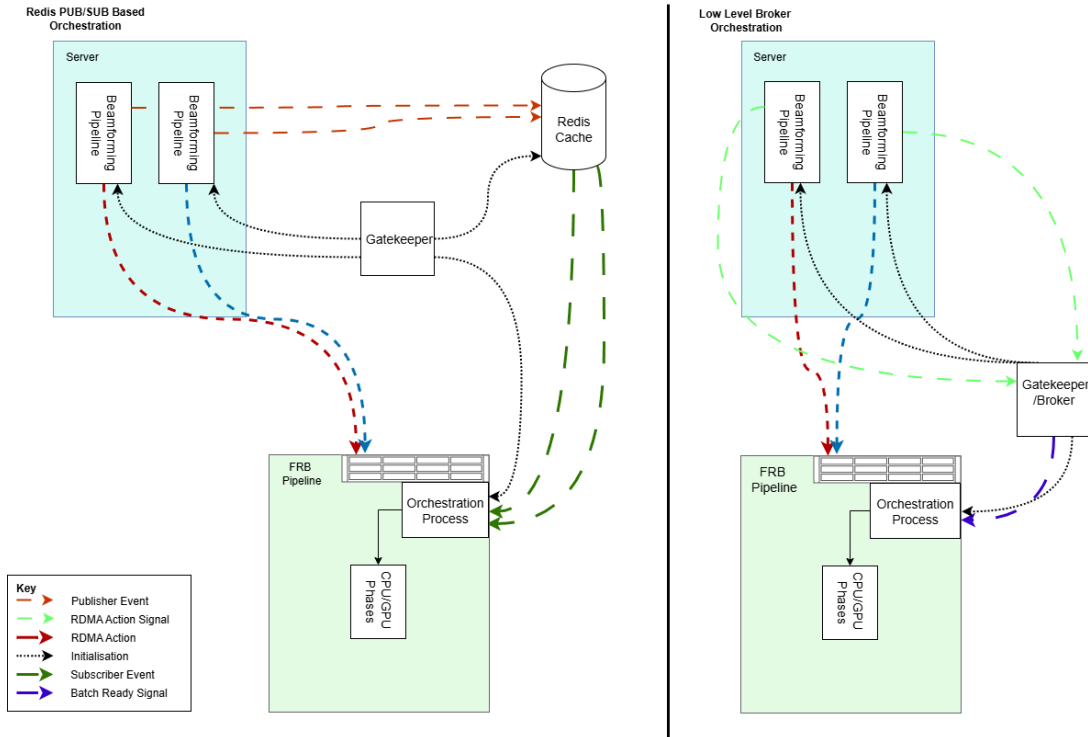


Figure 8: The envisaged orchestration between individual beamforming pipelines and the FRB pipeline operating on different compute environments. **Left:** The Central Gatekeeper initialises the system setup, distributing configuration and RDMA metadata. Each beamforming pipeline broadcasts its RDMA results and publishes a completion event. The FRB pipeline's orchestration process subscribes to these events and trigger processing once all required data has been received, subsequently pushing the data to the FRB pipeline. **Right:** Orchestration without Redis. The Gatekeeper directly coordinates between the GPU clusters and the FRB pipeline, acting as a lightweight broker.

This tight synchronisation requires orchestration between the GPU stages and the FRB process. Specifically, GPU pipelines must be aware of the memory allocations on the FRB pipeline (e.g., remote buffer addresses and RDMA keys), while the FRB pipeline must be notified precisely when a complete set of beams for a given batch (identified by a timestamp) has arrived and is ready for processing.

To manage this, an orchestration mechanism will be implemented using a publisher/subscriber (PUB/SUB) coordination model, where each GPU acts as a publisher and the FRB pipeline as the sole subscriber. A central orchestration process, referred to as the Gatekeeper, will oversee this synchronisation. The gatekeeper will handle Redis-based PUB/SUB setup across all GPU pipelines, as well as auxiliary tasks such as RDMA remote key distribution and session initialisation. The orchestration flow detailed above is summarised in the left diagram of Figure 8.

For the RDMA transport layer we will employ Libfabric [37], which provides a high-performance messaging and memory access API suitable for our low-latency needs. Initially, Redis will be used to manage PUB/SUB event notifications between the GPUs and the FRB pipeline. However, if Redis introduces unacceptable latency or synchronisation overheads, we will extend the gatekeeper's role to direct coordination between the parties involved. This orchestration flow is summarised in the right diagram of Figure 8. In this case, Redis will be replaced by a custom, low-level messaging mechanism, such as, RDMA-based signaling or tagged messages to efficiently manage readiness tracking and processing triggers.

One important consideration is network saturation or congestion which can affect message delivery latency. To mitigate this, care must be taken to isolate control signaling from the RDMA network channels.

6. Conclusion

We presented the ongoing development of a high-performance, real-time FRB detection pipeline for the upgraded Northern Cross Radio Telescope. This work details key architectural components, including a scalable and efficient data acquisition system, a GPU-accelerated beamforming and correlation backend, and suggests a transmission orchestration framework based on RDMA and Redis PUB/SUB synchronisation.

Although the complete end-to-end pipeline is still under integration, several core components have been successfully implemented and benchmarked. These include the high-throughput data reception and buffering system, which supports multi-producer, multi-consumer packet processing with low latency, and GPU kernels for beamforming and correlation that achieve $3\times$ and $23\times$ speedups over real-time acquisition rates, respectively, for current configuration targets.

The hypothesised orchestration layer should enable synchronised data delivery across heterogeneous compute environments, ensuring that FRB pipeline processes valid data batches. A central gatekeeper process facilitates RDMA resource allocation and coordination, laying the groundwork for reliable and scalable system orchestration.

Future work will focus on completing the integration of the entire processing chain, scaling the system to full instrument bandwidth and antenna count, verification of output, and incremental commissioning once the instrument is available, and the digital backend is ready.

Acknowledgments

The research activities described in this paper are carried out with the contribution of the NextGenerationEU funds within the National Recovery and Resilience Plan (PNRR), Mission 4 - Education and Research, Component 2 - From Research to Business (M4C2), Investment Line 3.1 - Strengthening and creation of Research Infrastructures, Project IR0000026 – Next Generation Croce del Nord." This research was also funded by the University of Malta Research Fund 2023.

Declaration on Generative AI

The authors have not employed any Generative AI tools.

References

- [1] D. R. Lorimer, M. Bailes, M. A. McLaughlin, D. J. Narkevic, F. Crawford, A bright millisecond radio burst at l-band, *Science* 318 (2007) 777–780. doi:10.1126/science.1147532.
- [2] D. Thornton, B. W. Stappers, M. Bailes, B. Barsdell, S. Bates, N. D. R. Bhat, M. Burgay, S. Burke-Spolaor, P. Coster, N. D’Amico, A. Jameson, S. Johnston, M. J. Keith, M. Kramer, L. Levin, A. G. Lyne, S. Milia, C. Ng, A. Possenti, B. van Straten, C. Tiburzi, A population of fast radio bursts at cosmological distances, *Science* 341 (2013) 53–56. doi:10.1126/science.1236789.
- [3] J.-P. Macquart, J. X. Prochaska, M. McQuinn, S. D. Ryder, R. M. Shannon, C. K. Day, A. T. Deller, R. D. Ekers, C. W. James, J. Kocz, A. Mannings, K. W. Bannister, S. Bhandari, M. Caleb, C. Flynn, L. Keating, P. Kumar, S. Osłowski, C. J. Phillips, D. C. Price, A. Rane, S. Simha, D. Sprenger, J. Staff, C. Wolf, A census of baryons in the universe from localized fast radio bursts, *Nature* 581 (2020) 391–395. doi:10.1038/s41586-020-2300-2.
- [4] S. Chatterjee, C. J. Law, R. S. Wharton, S. Burke-Spolaor, J. W. T. Hessels, G. C. Bower, J. M. Cordes, V. M. Kaspi, P. Demorest, B. J. Butler, A. Seymour, P. Scholz, S. Bogdanov, R. P. Eatough, L. E. Kasian, R. S. Lynch, E. C. Madsen, M. A. McLaughlin, D. Michilli, E. Parent, S. M. Ransom, L. G. Spitler, B. W. Stappers, S. P. Tendulkar, J. van Leeuwen, W. W. Zhu, Direct localization of a fast radio burst and its host galaxy, *Nature* 541 (2017) 58–61. doi:10.1038/nature20797.
- [5] A. Magro, A. Karastergiou, S. Salvini, B. Mort, F. Dulwich, K. Zarb Adami, Real-time, fast radio transient searches with gpu de-dispersion, *Monthly Notices of the Royal Astronomical Society* 417 (2011) 2642–2650. doi:10.1111/j.1365-2966.2011.19426.x.
- [6] W. Farah, M. Bailes, A. Jameson, E. D. Barr, T. Bateman, S. Bhandari, M. Caleb, C. Flynn, A. J. Green, E. F. Keane, J.-P. Macquart, S. Osłowski, A. Parthasarathy, V. Plant, V. Ravi, R. M. Shannon, B. E. Tucker, V. Venkatraman Krishnan, C. Wolf, Frb microstructure revealed by the real-time detection of frb170827, *Monthly Notices of the Royal Astronomical Society* 478 (2018) 1209–1218. doi:10.1093/mnras/sty1122.
- [7] C. J. Law, G. C. Bower, S. Burke-Spolaor, B. J. Butler, P. Demorest, A. Halle, S. Khudikyan, T. J. W. Lazio, M. Pokorny, J. Robnett, M. P. Rupen, realfast: Real-time, commensal fast transient surveys with the very large array, *The Astrophysical Journal* 236 (2018) 128. doi:10.3847/1538-4365/aab77b.
- [8] J.-P. Macquart, M. Bailes, N. D. R. Bhat, G. C. Bower, J. D. Bunton, S. Chatterjee, T. Colegate, J. M. Cordes, L. D’Addario, A. Deller, et al., The commensal real-time askap fast-transients (craft) survey, *Publications of the Astronomical Society of Australia* 27 (2010) 272–282. doi:10.1071/AS09082.
- [9] G. Foster, A. Karastergiou, G. Golpayegani, M. Surnis, D. R. Lorimer, J. Chennamangalam, M. McLaughlin, W. Armour, J. Cobb, D. H. E. MacMahon, X. Pei, K. Rajwade, A. P. V. Siemion, D. Werthimer, C. J. Williams, Alfaburst: a commensal search for fast radio bursts with arecibo, *Monthly Notices of the Royal Astronomical Society* 474 (2017) 3847–3856. doi:10.1093/mnras/stx3038.
- [10] M. Amiri, K. Bandura, P. Berger, M. Bhardwaj, P. J. Boyle, C. Brar, P. Chawla, T. Chen, J.-F. Cliche, A. Cook, J. Crafard, D. Cubranic, X. Deng, N. Denman, M. Dobbs, M. Fandino, E. Fonseca, B. M. Gaensler, K. Gourdji, M. Halpern, A. Hill, T. Hincks, V. M. Kaspi, R. Kothes, T. L. Landecker, M. Lang, D. Li, H. Lin, K. W. Masui, R. Mckinven, D. Michilli, S. Mirhosseini, C. Ng, C. Patel, U.-L. Pen, L. Pinsonneault, Z. Pleunis, M. Rafiei-Ravandi, S. M. Ransom, A. Renard, P. Scholz, R. Shaw, J. Shin, J. Sievers, K. M. Smith, I. H. Stairs, K. Stovall, S. P. Tendulkar, P. Tretyakov, R. Van Der Meer, K. Vanderlinde, P. Wang, D. Wulf, H. Yu, A second source of repeating fast radio bursts, *Nature* 566 (2019) 235–238. doi:10.1038/s41586-018-0864-x.

- [11] W. Farah, C. Flynn, M. Bailes, A. Jameson, T. Bateman, D. Campbell-Wilson, C. K. Day, A. T. Deller, A. J. Green, V. Gupta, R. Hunstead, M. E. Lower, S. Osłowski, A. Parthasarathy, D. C. Price, V. Ravi, R. M. Shannon, A. Sutherland, D. Temby, V. V. Krishnan, M. Caleb, S.-W. Chang, M. Cruces, J. Roy, V. Morello, C. A. Onken, B. W. Stappers, S. Webb, C. Wolf, Five new real-time detections of fast radio bursts with utmost, *Monthly Notices of the Royal Astronomical Society* 488 (2019) 2989–3002. doi:10.1093/mnras/stz1748.
- [12] T. An, D. Li, P. Wang, C.-H. Niu, Y.-K. Zhang, Y. Feng, C.-W. Tsai, H.-X. Chen, Y.-H. Zhu, An overview of fast real-time fast radio burst searching system, *Research in Astronomy and Astrophysics* 23 (2023) 095023. doi:10.1088/1674-4527/ace518.
- [13] P. Kumar, M. Bhardwaj, C. J. Law, E. Fonseca, V. M. Kaspi, D. Michilli, Z. Pleunis, J. Shin, I. H. Stairs, S. P. Tendulkar, P. Chawla, J. Crafard, D. C. Good, K. Gourdji, M. Halpern, J. W. T. Hessels, A. Hill, R. Kothes, T. L. Landecker, K. W. Masui, R. Mckinven, C. Ng, L. Pinsonneault, S. M. Ransom, A. Renard, P. Scholz, K. M. Smith, P. Tretyakov, K. Vanderlinde, D. Wulf, H. Yu, Initial results from a real-time frb search with the gbt, *Monthly Notices of the Royal Astronomical Society* 497 (2020) 352–363. doi:10.1093/mnras/staa2001.
- [14] P. Chandra, G. C. Anupama, K. G. Arun, S. Iyyani, K. Misra, D. Narasimha, A. Ray, L. Resmi, S. Roy, F. Sutaria, Explosive and radio-selected transients: Transient astronomy with ska and its precursors, *Journal of Astrophysics and Astronomy* 37 (2016). doi:10.1007/s12036-016-9408-7.
- [15] E. J. Murphy, et al., The ngvla science case and associated science requirements, in: E. J. Murphy (Ed.), *Science with a Next-Generation VLA*, ASP, San Francisco, CA, 2018. ArXiv:1810.07524.
- [16] G. Hallinan, V. Ravi, J. Kocz, C. J. Law, D. A. Frail, A. Horesh, A. W. Hotan, S. R. Kulkarni, C. Lynch, J.-P. Macquart, K. P. Mooley, J. Nativi, D. A. Perley, R. A. Perley, D. Petry, M. Salvato, F. K. Schinzel, A. Varghese, H. Vedantham, P. K. G. Williams, A. Zic, The dsa-2000 – a radio survey camera, *arXiv e-prints* (2019). arXiv:1907.07648.
- [17] The CHIME/FRB Collaboration, A catalog of local universe fast radio bursts from chime/frb and the kko outtrigger, *arXiv e-prints* (2025). arXiv:2502.11217.
- [18] The CHIME/FRB Collaboration, Updating the first chime/frb catalog of fast radio bursts with baseband data, *The Astrophysical Journal* 969 (2024) 145. doi:10.3847/1538-4357/ad464b.
- [19] M. C. Bezuidenhout, E. Barr, M. Caleb, L. N. Driessen, F. Jankowski, M. Kramer, M. Malenta, V. Morello, K. Rajwade, S. Sanidas, B. W. Stappers, M. Surnis, Meertrap: 12 galactic fast transients detected in a real-time, commensal meerkat survey, *Monthly Notices of the Royal Astronomical Society* 512 (2022) 1483–1496. doi:10.1093/mnras/stac579.
- [20] S. Sanidas, M. Caleb, L. Driessen, V. Morello, K. Rajwade, B. W. Stappers, Meertrap: A pulsar and fast transients survey with meerkat, volume 13, 2017, p. 406–407. doi:10.1017/S1743921317009310.
- [21] D. Pelliciari, G. Bernardi, M. Pilia, et al., The northern cross fast radio burst project: Iv. multi-wavelength study of the actively repeating frb 20220912a, *Astronomy Astrophysics* 690 (2024) A219. doi:10.1051/0004-6361/202450271.
- [22] A. Magro, K. Bugeja, R. Chiello, A. DeMarco, A high-performance, flexible data acquisition library for radio instruments, in: 2019 IEEE-APS Topical Conference on Antennas and Propagation in Wireless Communications (APWC), 2019, pp. 069–074. doi:10.1109/APWC.2019.8870490.
- [23] D. Pelliciari, G. Bernardi, G. Naldi, et al., The northern cross fast radio burst project – i. overview and pilot observations at 408 mhz, *Monthly Notices of the Royal Astronomical Society* 493 (2020) 2243–2252. doi:10.1093/mnras/staa813. arXiv:2002.04944.
- [24] D. Pelliciari, G. Bernardi, M. Pilia, et al., The northern cross fast radio burst project – ii. monitoring of repeating frb 20180916b, 20181030a, 20200120e, and 20201124a, *Monthly Notices of the Royal Astronomical Society* 513 (2022) 1858–1866. doi:10.1093/mnras/stac1031.
- [25] V. Gajjar, et al., Searching for broadband pulsed beacons from 1883 stars using neural networks, *The Astrophysical Journal* 932 (2022). doi:10.3847/1538-4357/ac6dd5.
- [26] S. Ransom, PRESTO: PulsAR Exploration and Search TOOLkit, *Astrophysics Source Code Library*, record ascl:1107.017, 2011.
- [27] V. Morello, K. M. Rajwade, B. W. Stappers, Iqrm: real-time adaptive rfi masking for radio transient and pulsar searches, *Monthly Notices of the Royal Astronomical Society* 510 (2021) 1393–1403.

doi:10.1093/mnras/stab3493.

- [28] B. R. Barsdell, M. Bailes, D. J. Barnes, M. J. Keith, Heimdall: A gpu-accelerated single-pulse search pipeline, *Monthly Notices of the Royal Astronomical Society* 422 (2012) 379–387. doi:10.1111/j.1365-2966.2012.20461.x. arXiv:1201.2965.
- [29] D. Agarwal, K. Aggarwal, S. Burke-Spolaor, D. R. Lorimer, N. Garver-Daniels, Fetch: A deep-learning based classifier for fast transient classification, *Monthly Notices of the Royal Astronomical Society* 497 (2020) 1661–1674. doi:10.1093/mnras/staa1856.
- [30] A. Magro, J. Borg, R. Chiello, D. Cutajar, K. Z. Adami, J. A. González-Esparza, E. A. Mascote, E. A. Rodríguez, J. C. M. Ambriz, P. Villanueva, Digitizing mexart – system overview and verification, 2021. URL: <https://arxiv.org/abs/2109.11329>. arXiv:2109.11329.
- [31] J. M. Mellor-Crummey, M. L. Scott, Algorithms for scalable synchronization on shared-memory multiprocessors, *ACM Trans. Comput. Syst.* 9 (1991) 21–65. URL: <https://doi.org/10.1145/103727.103729>. doi:10.1145/103727.103729.
- [32] A. Magro, A real-time, GPU-based, non-imaging back-end for radio telescopes, Doctoral dissertation, University of Malta, 2013. URL: <https://www.um.edu.mt/library/oar/handle/123456789/100862>.
- [33] R. V. van Nieuwpoort, J. W. Romein, Using many-core hardware to correlate radio astronomy signals, in: *Proceedings of the 23rd International Conference on Supercomputing, ICS '09*, Association for Computing Machinery, New York, NY, USA, 2009, p. 440–449. URL: <https://doi.org/10.1145/1542275.1542337>. doi:10.1145/1542275.1542337.
- [34] A. Parsons, D. Backer, A. Siemion, H. Chen, D. Werthimer, P. Droz, T. Filiba, J. Manley, P. McMahon, A. Parsa, D. MacMahon, M. Wright, A scalable correlator architecture based on modular fpga hardware, reuseable gateware, and data packetization, *Publications of the Astronomical Society of the Pacific* 120 (2008) 1207. URL: <https://dx.doi.org/10.1086/593053>. doi:10.1086/593053.
- [35] K. Fujita, T. Yamaguchi, Y. Kikuchi, T. Ichimura, M. Hori, L. Maddegedara, Calculation of cross-correlation function accelerated by tensorfloat-32 tensor core operations on nvidia's ampere and hopper gpus, *Journal of Computational Science* 68 (2023) 101986. URL: <https://www.sciencedirect.com/science/article/pii/S1877750323000467>. doi:<https://doi.org/10.1016/j.jocs.2023.101986>.
- [36] M. A. Clark, P. C. L. Plante, L. J. Greenhill, Accelerating radio astronomy cross-correlation with graphics processing units, 2011. URL: <https://arxiv.org/abs/1107.4264>. arXiv:1107.4264.
- [37] O. I. W. Group, libfabric: Openfabrics interfaces (ofi) framework, <https://ofiwg.github.io/libfabric/>, 2024. Version 1.20 or latest accessed version.