Astronomical Techniques and Instruments, Vol. 3, July 2026, 347–357

Article Open Access

GPFFC: wideband signal channelization algorithm using parallelized GPU technology

Meng Zhang[1*], Luxin Zhang[1], Yazhou Zhang[2]

[1]*School of Cyber Science and Engineering*, *Qufu Normal University*, *Qufu* 273165, *China*;

[2]*Xinjiang Astronomical Observatory*, *Chinese Academy of Sciences*, *Urumqi* 830011, *China*

*Correspondence: zhangmeng@qfnu.edu.cn

Received: August 4, 2025; Accepted: October 11, 2025; Published Online: October 13, 2025; https://doi.org/10.3724/ati2025066; https://cstr.cn/32083.14.ati2025066

© 2026 Editorial Office of Astronomical Techniques and Instruments, Yunnan Observatories, Chinese Academy of Sciences. This is an open access article under the CC BY-NC-ND 4.0 license (http://creativecommons.org/licenses/by-nc-nd/4.0/)

Citation: Zhang, M., Zhang, L. X., Zhang, Y. Z. 2026. GPFFC: wideband signal channelization algorithm using parallelized GPU technology. *Astronomical Techniques and Instruments*, **3**(4): 347−357. https://doi.org/10.3724/ati2025066.

Abstract: To meet the requirements of real-time processing of wideband or ultra-wideband signals from large aperture radio telescopes, we designed and implemented a Graphics Processing unit-based parallel polyphase Filter bank Fast Channelization algorithm, which we use to study the processes of the pulsar baseband data unpacking, channelization, and packing. Graphics Processing Unit channelization technology has been optimized. By using the Compute Unified Device Architecture multidimensional thread model, tasks are subdivided into multiple small blocks to improve parallel computing efficiency. The data processing was optimized based on Compute Unified Device Architecture asynchronous processing to reduce Graphics Processing Unit idle time, which improved the parallel computing efficiency, increasing the speed to approximately 1.4 times that of the non-optimized implementation. Baseband data for pulsar J0437–4715, recorded by the CASPER Parkes Swinburne Recorder backend of the Parkes 64-m radio telescope, were used for testing. The pulsar baseband signal was subdivided into sub-bands, and the data from each was then packaged. Finally, the Digital Signal Processing Software for Pulsar Astronomy was used to process the baseband data of multiple sub-bands, and the pulse profiles of multiple sub-bands were merged into wideband pulses, which were consistent with the pulse profiles of the raw baseband data in phase and signal-to-noise ratio, verifying the effectiveness of the Graphics Processing Unit-based parallel Polyphase Filter bank Fast Channelization algorithm. It processes 400 MHz dual-polarized data in 5.8 s with 8-bit quantization over an 8 s observation time, meeting the real-time processing requirements of pulsar baseband data.

Keywords: Astronomical methods; Astronomical techniques; Astronomy data analysis; Astronomical simulations

1. INTRODUCTION

To meet high sensitivity requirements, new radio telescopes are being developed with either larger apertures or antenna arrays, together with advances in receiver technology and system design. However, the augmentation of observational bandwidth places considerable demands on backend hardware, presenting substantial challenges for data transmission and signal processing. In the context of ultra-wide bandwidth pulsar observations, several technical obstacles must be addressed, including high-fidelity signal acquisition and transmission, accurate reconstruction across heterogeneous computing platforms, and real-time exchange and processing of large-volume astronomical data streams.

Polyphase channelization is a critical technique for real-time transmission and processing of wide and ultra-wide bandwidth signals in multifunctional digital backend systems. It decomposes high-speed input signals into multiple low-rate, adjacent narrowband sub-bands, which can be recombined to reconstruct either the full raw signal or selected portions, enabling further scientific analysis[1]. In astronomy, polyphase channelization is employed in a range of applications, such as in the offline Very Long Baseline Interferometry (VLBI) software correlator developed for the Canadian Hydrogen Intensity Mapping Experiment (CHIME)/Fast Radio Burst (FRB) outriggers project. In this system, CHIME data are channelized immediately after digitization, with baseband signals transformed into the frequency domain for slicing, allowing tracking of FRB dispersion sweeps on a per-channel basis[2]. In wideband pulsar timing, the received signals are first divided into critically sampled sub-bands that continuously span the desired frequency range. Each channel

is then processed independently, including de-dispersion, folding, and subsequent analysis[3].

With the rapid advancement of Graphics Processing Units (GPUs), their powerful parallelization, superior floating-point performance, and the introduction of the Compute Unified Device Architecture (CUDA) programming model have made GPU-based development significantly more accessible than that based on Field-Programmable Gate Arrays (FPGAs)[4]. Processing ultra-wide bandwidth signals typically requires fine-grained channelization, in which the signal spectrum is divided into narrow sub-bands on the order of a few kHz for subsequent analysis. Although FPGAs offer inherent parallelization, their computational performance and flexibility are constrained by hardware configurations and limitations in channelizer design[5]. By contrast, GPUs provide greater adaptability when performing parallelized, fine-resolution channelization tasks[6]. Real-time transmission of ultra-wide bandwidth signals involves processing large-scale, high-throughput data streams, as well as executing Fast Fourier Transforms (FFTs) and polyphase filtering. FPGA platforms are expensive to design and manufacture. Frequently, FPGAs are used to facilitate coarse channelization, before signals are further filtered to a fine frequency resolution using GPUs[7]. The implementation of a multifunctional digital backend system therefore demands a computing platform that offers high programmability, rapid development cycles, mature toolchains for debugging and optimization, and cost-effective, sustainable deployment. In combination with conventional Central Processing Units (CPUs), heterogeneous FPGA+CPU+GPU platforms have emerged as the prevailing solution for real-time pulsar signal processing in international practice[8]. Offloading computationally intensive tasks to GPUs alleviates CPU workload and substantially improves data stream processing efficiency. As digital backend systems in radio astronomy continue to evolve, the increasing demand for real-time signal processing presents significant challenges to traditional computing architectures. The massive parallelization of GPUs offers a promising pathway for handling the large-scale data flows used inherently in modern radio astronomical observations[9].

McCurry[10] implemented a Polyphase Filter Bank (PFB) using NVIDIA GPUs to channelize discrete input data into multiple frequency subchannels. Although the achieved performance fell short of expectations, the work introduced a CUDA-based approach to GPU-accelerated channelization, demonstrating the feasibility of replacing FPGAs with GPUs in astronomical signal processing systems. van der Veldt et al.[11] enhanced the digital backend of the Low-Frequency Array Radio (LOFAR) telescope by employing multicore CPUs for parallel PFB execution, followed by GPU-based optimization to support high-speed astronomical data channelization. Their results confirmed the performance advantages of NVIDIA GPU-based backend architectures. Chennamangalam et al.[12] developed a hybrid FPGA+GPU spectrometer for the Green Bank Telescope (GBT)[1], which applies polyphase filtering in the time domain. This system supports more channels and wider bandwidths than typical FPGA-only designs. However, it does not employ CUDA asynchronous streams for finer-grained segmentation of the polyphase filtering stage during GPU-based channelization. Merry[13] further implemented a GPU-based channelizer that replicates the full functionality of traditional FPGA-based systems, illustrating that GPU-accelerated architectures not only offer modular flexibility but also represent a prevailing trend for the future of radio astronomy digital backends.

The Xinjiang Qitai 110-m aperture radio telescope (QTT) is designed to operate over a frequency range of 150 MHz to 115 GHz and supports ultra-wideband receivers. It offers a variety of observation modes, including total power, polarimetry, and VLBI[14]. Among its multiple ultra-wide bandwidth receivers, the low-frequency system (704–4032 MHz) utilizes an FPGA to divide the 3328 MHz input into 26 sub-bands, each 128 MHz wide. To enable quasi-real-time processing of ultra-wide bandwidth signals, the digital backend system needs to process 1 GB s^{-1} of incoming data[2]. In response to the growing demand for real-time data transmission and preprocessing in next-generation large-aperture radio telescopes and wideband or ultra-wide bandwidth array systems, we present the design and implementation of a GPU-based parallel Polyphase Filter bank Fast Channelization algorithm (GPFFC). By incorporating asynchronous CUDA streams to reduce GPU idle time and GPU-based unpacking to bypass CPU overhead, our proposed approach yields improved computational efficiency while enabling real-time subdivision of wideband signals into finer narrowband components than those achievable with FPGA-based approaches. Our algorithm offers a viable solution for real-time preprocessing step in applications such as high-precision pulsar timing[15], radio interferometric imaging[16], molecular spectral line observations[17], and FRB-related studies[18,19].

2. DESIGN AND IMPLEMENTATION OF THE GPFFC

The polyphase channelization algorithm enables the decomposition of radio astronomy wideband signals into multiple narrowband signals, facilitating subsequent scientific analysis. A typical polyphase channelizer consists of three key components: phase shifting and downsampling, polyphase filtering, and parallel FFT, the essence of

1. *https://casper.berkeley.edu/wiki/Tutorial_5: _Heterogeneous_Instrumentation*

2. Bandwidth 128 MHz, dual-polarized, 16-bit complex data per second, $128 \times 2 \times 2 \times 16/8 = 1$ GB s^{-1}.

which is the combination of parallel branching filtering and FFT. Fundamentally, it integrates multi-branch Finite Impulse Response (FIR) filtering and FFT processing, which involves intensive multiply-accumulate operations and can be accelerated using the GPU parallel computing techniques[6]. The processing flow of the GPFFC is illustrated in Fig. 1. Two operations are performed on the host (CPU). First, the prototype polyphase filter coefficients are transferred to the device (GPU). Specifically, the sub-filter coefficients obtained from the polyphase decomposition of the prototype filter are sequentially transmitted for each sub-band. The baseband data are then transferred in blocks, where the number of divisible blocks is determined from the header information of the data to be processed. Each block is copied to the device sequentially. On the device side, three primary tasks are executed: unpacking the baseband data, performing channelization, and transferring the processed data back to the host. Finally, the CPU performs packetization of the channelized data and outputs it.

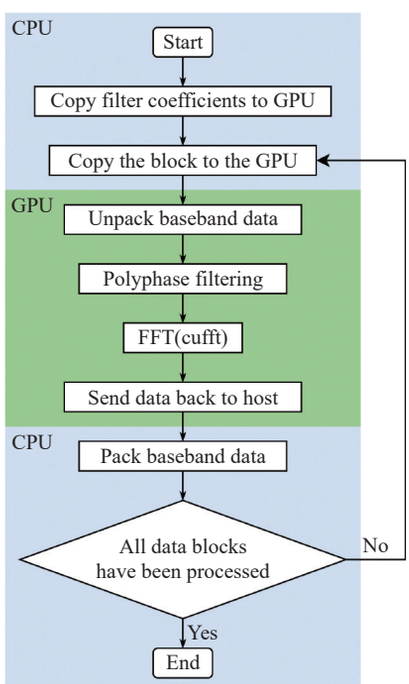


Fig. 1. Flow chart for the GPFFC algorithm.

2.1. Unpacking Pulsar Baseband Data

The full pulsar baseband dataset generated by observations is too large to be copied to the device in a single operation. According to the data bandwidth, the data must be divided into multiple data blocks at the host. The divided baseband data is then copied block-by-block to the device for batch processing, to reduce the number of memory accesses. CUDA stream asynchronous parallelization is combined to reduce the delay caused by memory access.

For the Data Acquisition and Distribution Architecture (DADA)[3] data recorded by the Collaboration for Astronomy Signal Processing and Electronics Research (CASPER) Parkes Swinburne Recorder (CASPSR) backend of the Parkes 64-m radio telescope, information including the bandwidth and central frequency of the observed pulsar signal are stored in a 4096-byte header by default. The data section consists of interleaved dual-polarization baseband samples, where data from the two polarizations alternate every four real-valued samples, with each polarization represented using 8-bit precision[20], as shown in Fig. 2A. During data parsing, the interleaved polarizations are separated and transferred individually to the device. Channelization is performed independently on each polarization stream, after which the processed data are re-interleaved and stored in DADA format.

When processing pulsar baseband data with the polyphase channelizer, the dual-polarization data must first be unpacked and converted into high-precision floating-point format. Subsequently, a large volume of unpacked data is transferred to the device for further processing. To reduce overall latency and support real-time processing, the data transfer pipeline between the host and device is optimized. Instead of performing unpacking on the CPU, the raw baseband data is copied directly into GPU memory, thereby minimizing CPU-side overhead and reducing memory transfer time. The dual-polarized streams are unpacked and converted to floating-point values in parallel using GPU threads. Moreover, asynchronous data transfer and computation are achieved via CUDA streams, effectively overlapping communication and computation to further mitigate the impact of data movement on channelization latency. The index mapping before and after unpacking is given by

$$i = j \,\%\, npad + \mathrm{round}\left(\frac{j}{npol \cdot npad}\right) \times npad, \qquad (1)$$

where j and i are the indices before and after unpacking, respectively. The expression inside the brackets indicates rounding to the nearest integer. $npol$ is the number of polarizations, and $npad$ represents the number of interleaved samples per polarization (with $npad = 4$ for the baseband data recorded by CASPSR backend). After unpacking, the polarization data are stored independently, and i indexes the data arrays corresponding to each polarization, as shown in Fig. 2A.

2.2. Design of the GPFFC

The core of the polyphase filter bank channelization algorithm consists of two main components: polyphase filtering and parallel FFT. To implement polyphase filtering, the prototype filter coefficients and the raw baseband data need to be polyphase decomposed, which rearranges the coefficients and data in the raw order. Fig. 3 shows the arrangement of the filter coefficients and data for a 3-channel example. After decomposition, the coefficients are orga-

3. *https://sourceforge.net/projects/psrdada/*

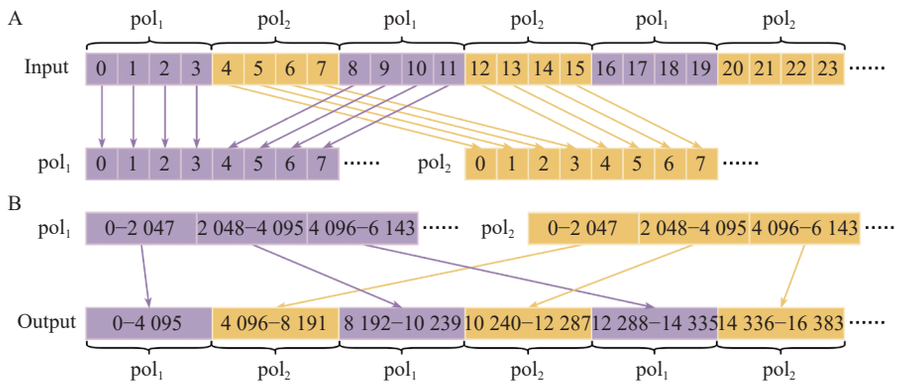


Fig. 2. (A) Structure and unpacking process of DADA baseband data recorded by the CASPSR backend of the Parkes 64-m radio telescope. The data stream consists of interleaved dual-polarization samples with 8-bit precision, alternating every four real-valued samples. During processing, the two polarizations are separated and transferred independently to the device for channelization. (B) Packing of DADA baseband data. The dual-polarized data are stored alternately at 2048 complex sample intervals.

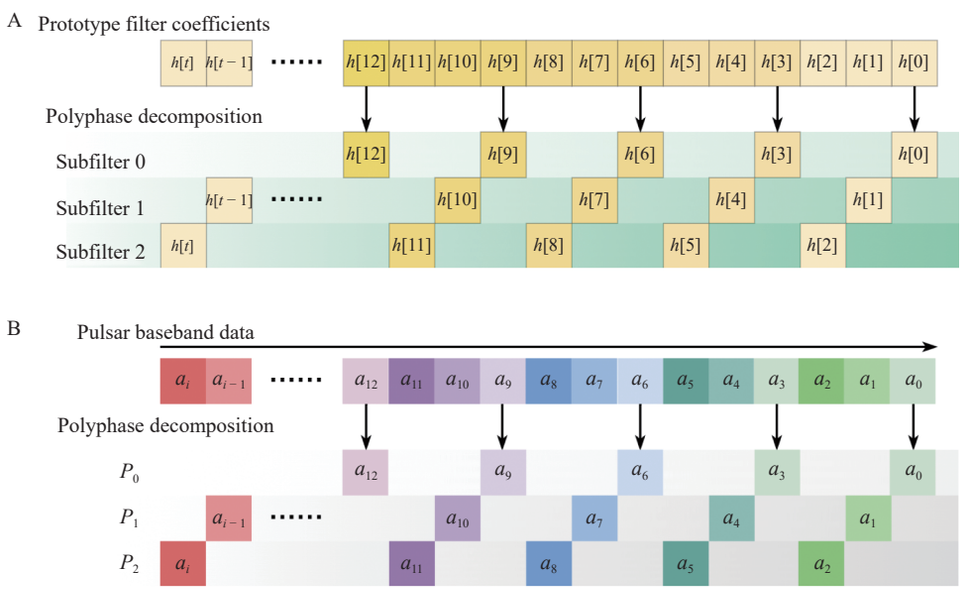


Fig. 3. (A) Arrangement of filter coefficients after polyphase decomposition for a 3-channel example. (B) Arrangement of input data after polyphase decomposition for a 3-channel example.

nized into three sub-filter groups, and the input data are correspondingly divided into three segments, each aligned with one sub-filter. Specifically, a coefficient with index s_i is assigned to sub-filter $s_i \bmod n_s$ and appears at position s_i/n_s within that sub-filter, where $n_s = num_{chans}$, n_s denotes the total number of sub-filters, num_{chans} denotes the number of channels after channelization, including those later discarded during real-to-complex FFT processing.

To facilitate subsequent calculations, all indices s_i, n_s, and num_{chans} are zero-based. Similarly, an input sample at index d_i is filtered by the sub-filter at index $num_{chans} - d_i \bmod num_{chans} - 1$, starting from the position d_i/num_{chans}. As shown in Fig. 3, the decomposed data and coefficients form two-dimensional matrix-like structures, which make their structure and relationship more visually intuitive. However, explicitly storing them as matrices would require two memory accesses (row and column indexing) per data element during FIR filtering. For large-scale data, such repeated memory access significantly degrades processing efficiency. In the implementation of the GPFFC, neither the filter coefficients nor the baseband data are stored in matrix form. The GPFFC uses efficient index-based access through one-dimensional arrays.

The polyphase filtering operation is essentially a linear convolution between the polyphase data and the impulse response of the filter. This involves a large number of addition and multiplication operations, making it well-suited for parallel acceleration using GPU threads. The main workflow of the polyphase filtering kernel function on the GPU architecture is as follows:

(1) A unique thread identifier *tid* is assigned using *threadIdx.x*, *blockDim.x*, and *blockIdx.x*. This identifier is used to determine the array indices of the input data and corresponding filter coefficients involved in the convolution, ensuring accurate computation. The same *tid* is also

used as the index for writing to the output array, where the channelized results are stored sequentially by a sub-band.

(2) Based on the specified data block size and the number of channels, the total number of data points to be processed in each sub-band is computed. The corresponding sub-band is determined for each output data point, with its index within that sub-band. This information is then used to identify the sub-band to which the relevant filter coefficients belong, along with the corresponding array boundaries. Similarly, the index range of the input data involved in the convolution can be inferred.

(3) The convolution is carried out using the previously computed variables.

Algorithm 1 Polyphase filtering algorithm

```
function polyphaseFilter(coeffs, inputData, outputData, col)
  for all thread tid in block do
    if tid < DATA_BLOCK_SIZE then
      chan_out Idx ← tid % col
      chan_out ← ⌊tid/col⌋
      chan_in ← num_chans − chan_out Idx − 1
      chan_in fIdx ← chan_in + chan_out Idx · num_chans
      outputData[tid] ← complex(0,0)
      for tap = 0 to num_taps − 1 do
        if chan_out Idx − tap = 0 then
          input ← inputData[chan_in fIdx − tap · num_chans]
          coeff ← coeffs[chan_out · num_taps + tap]
          product ← input · coeff
          outputData[tid] ← outputData[tid] + product
        end if
      end for
    end if
  end for
end function
```

The polyphase filtering is implemented within a single kernel function, with its pseudocode presented in Algorithm 1. The sub-band corresponding to the current thread is then given by

$$chan_{\text{out}} = \lfloor tid/col \rfloor, \tag{2}$$

where $chan_{\text{out}}$ is the output sub-band assigned to the current thread, and col is the length of each set of data after polyphase decomposition. The index of the thread within the output sub-band is calculated by

$$chan_{\text{out}}Idx = tid \bmod col, \tag{3}$$

where $chan_{\text{out}}Idx$ is the index of the thread within the output sub-band. The input sub-band required for the computation is then calculated as

$$chan_{\text{in}} = num_{\text{chans}} - chan_{\text{out}}Idx - 1, \tag{4}$$

where $chan_{\text{in}}$ denotes the input sub-band needed for the calculation, and num_{chans} is the total number of output sub-bands. Finally, the index of the first element (i.e., subscript 0) within that input sub-band is given by

$$chan_{\text{in}}fIdx = chan_{\text{in}} + chan_{\text{out}}Idx \cdot num_{\text{chans}}, \tag{5}$$

where $chan_{\text{in}}fIdx$ denotes the starting index of the input sub-band corresponding to the thread.

Fig. 4 shows the convolution process in polyphase filtering. After polyphase decomposition, each set of data is convolved with the coefficients of a corresponding sub-filter. Thread m performs the computation starting from position a_j in P_2 and proceeds forward from this point. Each thread performs at least one Multiply-Accumulate (MAC) operation, with the maximum number determined by the sub-filter length num_{taps}. The number of MAC operations executed by each thread is computed based on $chan_{\text{out}}Idx$ and num_{taps}. As shown in Fig. 4, the calculation of each output sample is as follows: The first output sample of the current subband is given by

$$v\left(a_{j-11}\right) = h(0) \cdot x\left(a_{j-11}\right), \tag{6}$$

where $v(n)$ is the output, $x(n)$ is the raw baseband input, and $h(m)$ is the coefficient of the sub-filter. For the second sample,

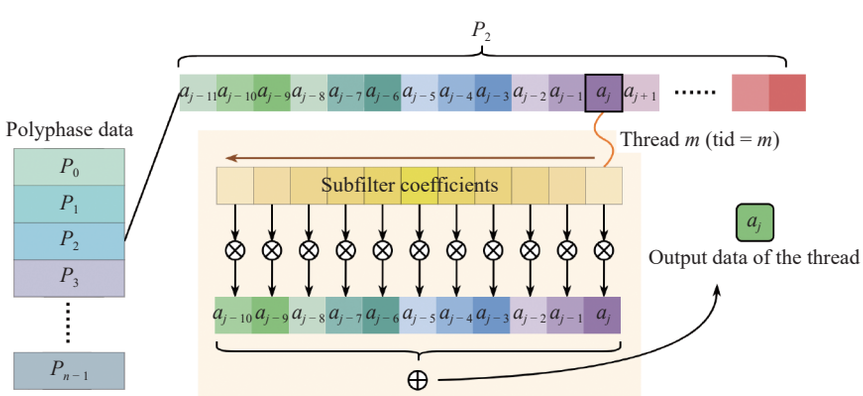


Fig. 4. Parallel polyphase filtering. After polyphase decomposition, each data segment is convolved with the coefficients of its corresponding sub-filter, with each GPU thread performing the assigned MAC operations in parallel. The process effectively maps the serial convolution in Equations (6), (7) and (8) onto thread-level parallel execution, enhancing computational efficiency.

$$v\left(a_{j-10}\right)=h(0)\cdot x\left(a_{j-10}\right)+h(1)\cdot x\left(a_{j-11}\right). \tag{7}$$

The current thread performs a similar convolution operation on the remaining samples, using the corresponding filter coefficients. The convolution is computed as

$$\begin{aligned} v\left(a_j\right)=&\,h(0)\cdot x\left(a_j\right)+h(1)\cdot x\left(a_{j-1}\right)\\ &+\cdots+h\left(num_{\text{taps}}-1\right)\cdot x\left(a_{j-num_{\text{taps}}+1}\right), \end{aligned} \tag{8}$$

where num_{taps} is the length of the sub-filter. Each thread m in the GPU performs the MAC operations starting from its assigned position a_j, executing a number of operations determined by num_{taps} and the thread's ordinal position within the sub-band. If the thread's ordinal position is smaller than num_{taps}, it executes fewer MAC operations accordingly. The original serial execution of the data computation process is parallelized using GPU threads. For every input sample received by the device, a corresponding GPU thread is launched to compute the output. However, the threads corresponding to a kernel invocation do not necessarily run simultaneously, and are instead limited to the number the device can support. All accumulations within the polyphase filter are performed using single-precision complex data.

After polyphase filtering, a parallel FFT is applied to the filtered sub-band data. In this process, the results of the polyphase filtering are stored in a two-dimensional array with a column length denoted by col. For each column, data from all sub-bands are processed using a num_{chans}-point parallel FFT. Specifically, the elements at indices 0, col, $2col$, … ,$(num_{\text{chans}}-1)col$ form one FFT group; indices 1, $col+1$, … , $(num_{\text{chans}}-1)col+1$ form the next; and so on, with the FFTs executed column by column in parallel.

The Fourier transform is performed using CUDA Fast Fourier Transform (cuFFT), configured to execute a one-dimensional inverse FFT with a length equal to the number of channels. Since the input pulsar baseband data are real-valued, only half of the FFT output needs to be retained, owing to the conjugate symmetry property of the transform. For complex-to-complex transforms, cuFFT forward and inverse FFTs to be distinguished from each other using execution parameters. However, when performing a real-to-complex transform, converting the FFT output to the corresponding inverse FFT result requires an additional normalization step. Specifically, each output element must be divided by the FFT length. As such, the FFT results of each channel are divided by num_{chans} prior to output. This normalization is applied during the pulsar baseband data packaging process.

In the original batch processing workflow, all data transferred to the device in a single operation are processed using kernel functions in a strictly sequential manner. Data reading, transmission, and computation occur one after another. This serial pipeline fails to fully exploit the GPU's parallel processing capabilities. Additionally, data transfers between the host and device incur latency, during which the GPU remains idle as it waits for new data. These inefficiencies become pronounced when processing large data volumes, leading to reduced algorithmic performance in the GPFFC implementation. The input consists of dual-polarization baseband data, which is interleaved. During unpacking, the two polarizations are separated and processed independently using identical polyphase filtering logic, followed by re-packing into baseband format. We optimize this procedure using CUDA stream technology. By assigning separate streams for data upload, computation, and download, the GPU's copy engines can operate concurrently with the Streaming Multiprocessors (SMs). Fig. 5A shows a processing flow where dual-polarization data are handled without CUDA stream separation. In this case, each processing node must wait for the completion of the dual polarization data processing to synchronize the processing progress of the dual polarization data. In contrast, Fig. 5B shows an optimized approach where polarization data is handled in separate CUDA streams, enabling asynchronous and parallel processing. Data transfer and filtering operations within each stream are executed in sequence, but they can proceed concurrently across different streams. This decoupling significantly enhances throughput by minimizing idle GPU time and improving overall data handling efficiency.

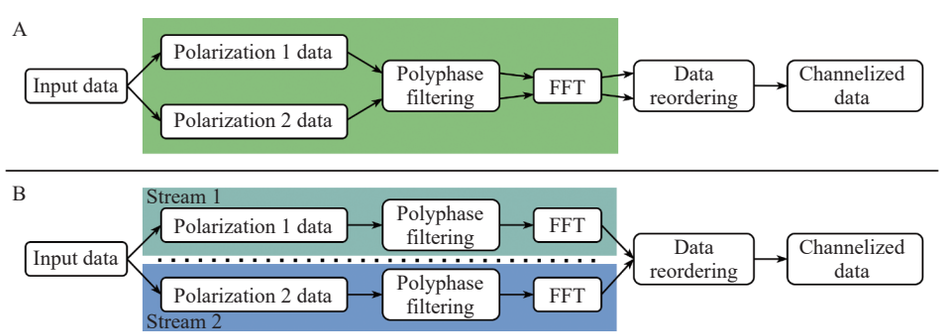


Fig. 5. Diagram of dual-polarization data processing with CUDA stream parallelization.

The complete workflow of the GPFFC implementation, using GPU parallelization to enhance computational efficiency, is shown in Fig. 6. The process begins with the transfer of dual-polarization baseband data to the device.

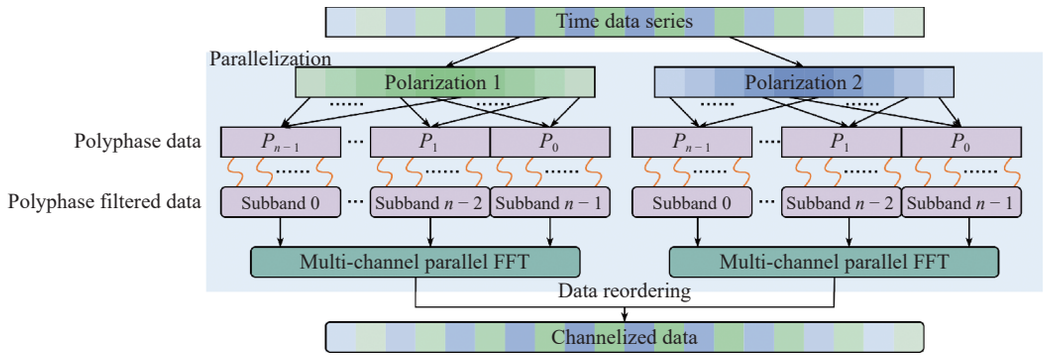


Fig. 6. Complete workflow of the GPFFC implementation, with GPU parallelization. The process begins with the transfer of dual-polarization baseband data to the GPU, where data transmission and computation are executed concurrently, using CUDA streams. Each polarization stream is polyphase decomposed and convolved with its corresponding sub-filter coefficients in parallel. Subsequently, a multi-channel FFT is applied to each polarization output, and the results are rearranged and repackaged to generate the final channelized data.

By employing CUDA-based parallel techniques, data transmission and computation are executed concurrently. Once the two polarization streams are independently transferred to the device, the polyphase-decomposed data of each polarization are convolved with the corresponding sub-filter coefficients. The polyphase filtering of all sub-bands is performed in parallel, followed by a multi-channel FFT applied separately to the output of each polarization. After the FFT, the dual-polarization data are rearranged and repackaged to form the final channelized output.

2.3. Pack Pulsar Baseband Data

To facilitate subsequent processing with Digital Signal Processing Software for Pulsar Astronomy (DSPSR), the data from each sub-band are encapsulated into DADA baseband files. The header of each output DADA file is configured based on the header of the input pulsar baseband data and the number of sub-bands produced by the channelizer. Parameters such as bandwidth, center frequency, quantization bit precision, data dimensions, and sampling interval must be recalculated and updated according to the specifications of the polyphase channelization scheme. The default header length is 4096 bytes.

The output of the GPFFC is in the form of single-precision complex floating-point data. To reduce the volume of data transferred from the device to the host and to improve transfer efficiency, 16-bit quantization is performed in parallel on the GPU. The quantized data from each sub-band are then encapsulated into the DADA baseband format, as illustrated in Fig. 2B. For each polarization, the data are interleaved every 2048 complex samples (equivalent to 4096 real values), and each sample is encoded using offset binary coding[20], given by

$$out = input \bigoplus 0x8000. \tag{9}$$

Both data rearrangement and offset encoding are executed concurrently on the GPU. The processed data are subsequently copied back to the host, where the header is updated and the output is packaged into the DADA baseband format. The complete packing procedure is shown in Fig. 2B.

3. EXPERIMENTS

The prototype filter parameters of the GPFFC are fully configurable, including the FFT length num_{chans} (i.e., the number of frequency channels), the number of taps per channel, and the choice of window function. This flexibility allows the polyphase channelizer to be adapted to different frequency resolutions and stopband attenuation requirements. Here, we employ a Hamming window in the design of the prototype low-pass FIR filter, yielding a total filter length of 512 taps. The resulting coefficients are modulated and subsequently polyphase decomposed into num_{chans} polyphase sub-filters. The amplitude response s of all sub-filters exhibit flat passbands and strong stopband attenuation, meeting the design specifications.

To evaluate the effectiveness of the GPFFC, tests were conducted on a GPU server with the configuration listed in Table 1. The GPU implementation is based on CUDA 11.8. Global memory is used for storing input data, filter coefficients, and output results. The test data consists of DADA baseband data from the CASPSR backend of the Parkes 64-m radio telescope, with pulsar J0437-4715 as the source. The baseband data are dual-polarized real-valued signals with a sampling interval of 0.00125 s and a total file size of 12.8 GB. The frequency coverage spans from 1582 MHz to 1182 MHz, corresponding to a total bandwidth of 400 MHz centered at 1382 MHz. For testing, the GPFFC divides the baseband signal into 8 sub-bands, with each sub-filter comprising 64 taps (num_{taps}). The raw 400 MHz bandwidth is evenly split into 8 channels, each with a bandwidth of 50 MHz. The sub-band frequency ranges are 1182–1232 MHz, 1232–1282 MHz, 1282–1332 MHz, 1332–1382 MHz, 1382–1432 MHz, 1432–1482 MHz, 1482–1532 MHz, and 1532–1582 MHz.

The dynamic spectrum and spectrogram of pulsar J0437–4715, generated directly from the recorded data using the DSPSR software, are shown in Fig. 7. To verify the channelization, each output channel was subjected to de-dispersion, integration, and folding[21]. The resulting

Table 1. Server configuration

Key	Value
CPU	Intel Xeon Gold 5317
CPU memory	512 GB
CPU clock rate	3.00 GHz
CUDA	CUDA 11.8
GPU	NVIDIA GeForce RTX 3090
GPU memory	24 GB
Max threads per block	1024
Max thread dimensions	(1024, 1024, 64)

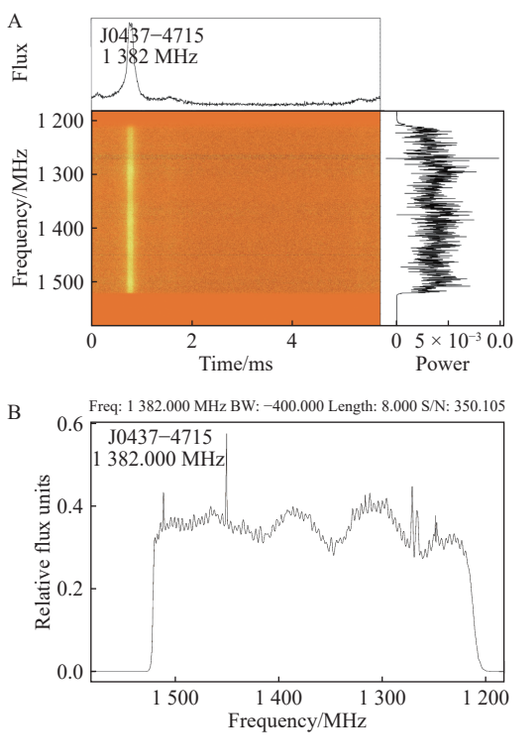


Fig. 7. (A) Dynamic spectrum of pulsar J0437–4715, generated directly from the recorded data, using the DSPSR software. (B) Radio spectrum of pulsar J0437–4715, generated directly from the recorded data, using the DSPSR software.

profiles from 16 sub-bands are presented in Fig. 8, demonstrating clearly identifiable pulsar profiles in each sub-band. Because of conjugate symmetry, only half of the channels are retained, yielding a total of 8 output channels. Fig. 9 shows the pulse profiles and spectrogram after recombining the 8 output sub-bands. A comparison between Fig. 7A and Fig. 9A shows that the pulse profiles and phases obtained after channelization are consistent with those derived from the raw data. This confirms that the raw signal can be recovered from the channelized output, validating the effectiveness of the GPFFC proposed in this paper. Fig. 7B and Fig. 9B also show that the spectral information represented by the 8 retained sub-bands is sufficient to reconstruct the full-band spectrum. The recombined spectrum closely matches the original, except for mild edge attenuation and spectral leakage at sub-band boundaries in Fig. 9B, which are inherent to the use of critical sampling[22].

To implement the GPFFC, the input data is processed in chunks. Due to the baseband data packing format, where dual-polarization samples are alternately stored every 2048 complex values, the chunk size must be an integer multiple of 2048 times the number of channels. In this study, the chunk size is configured from 16 × 2048 to 496 × 2048. If the size exceeds this range, the program may fail to execute properly. The minimum chunk size is set to 16 × 2048 (corresponding to 16-channel channelization), i.e., 32768 samples. Under this configuration, one chunk from each polarization is processed per iteration, and 2048 complex samples per polarization are written to the output file of each channel. If the chunk size is smaller than 32768, the processed results cannot be correctly output. The maximum allowable chunk size depends on the available GPU memory.

We configured the thread block size to 256, which was chosen based on the recommendation for optimal register reuse and to avoid excessive register pressure. We also varied the data chunk size as integer multiples of 2048. The GPFFC execution time was measured as the average over multiple runs. When the channelization process was executed on an RTX 3090 GPU, the maximum allowable chunk size was 1015808 samples (496 × 2048). The least execution time occurred at a chunk size of 786432 samples (384 × 2048), with the total time for data unpacking, transfer, processing, and repacking being 29.94 s.

To further analyze the optimal performance case, we evaluated the execution time of the GPFFC for a data block size of 786432 samples under different thread block sizes, as shown in Fig. 10A. According to NVIDIA's official documentation, the RTX 3090 GPU supports a maximum thread block size of 1024, within which optimal performance and stability are achieved. We tested thread block sizes ranging from 32 to 1024 and observed only minor differences in execution time, with the minimum time of 29.37 s achieved with a thread block size of 768. Since the data block size is fixed, the number of thread blocks is determined by both the thread block and data block sizes. Consequently, the total number of active threads remains constant for a given data block, regardless of how threads are grouped. Larger thread blocks may lead to increased idle threads within some blocks, resulting in inefficient use of computational resources. Therefore, within the valid range (up to 1024), increasing the thread block size does not necessarily lead to faster execution.

In practical applications, the GPFFC is typically applied after the raw pulsar baseband data have been unpacked, and the channelized output is transferred directly to subsequent processing stages without repacking. To evaluate the core performance of the algorithm, we measured the time required for data transfer from CPU to GPU and the channelization process during multiple executions, as shown in Fig. 10B, where the horizontal axis represents the data block size in multiples of 2048. As the block size increases, the proportion of processing

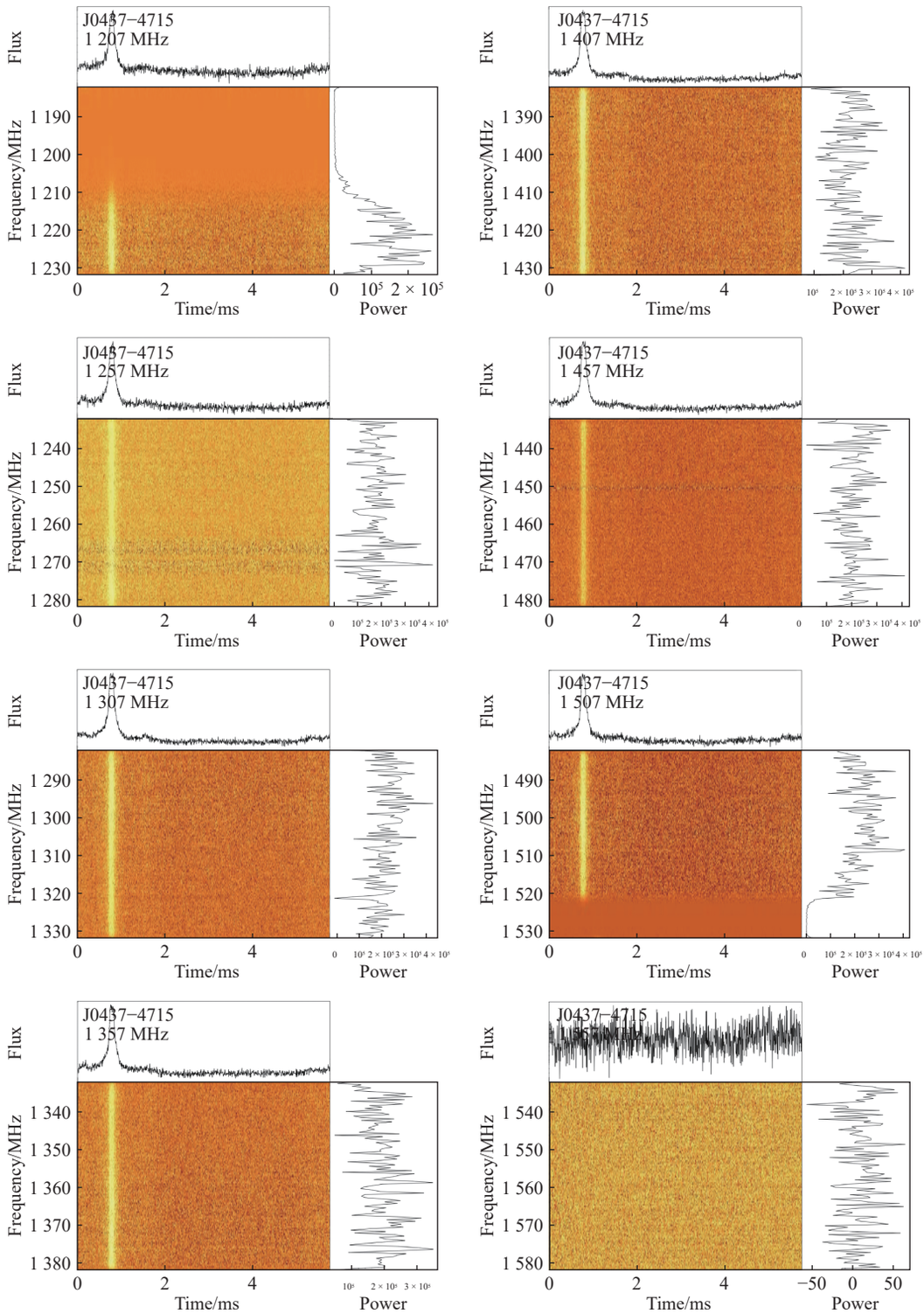


Fig. 8. The sub-band dynamic spectrum of pulsar J0437−4715, recovered from the eight channels of the GPFFC output after data processing.

time gradually surpasses that of data transfer, and the overall execution time decreases. When using the RTX 3090 GPU, the maximum block size is 1 015 808 samples (496 × 2 048). However, Fig. 10B shows that execution time does not decrease monotonically with increasing block size. For small blocks, the total number of transfers increases due to the limited data volume per block, resulting in transfer time dominating the overall runtime. Once the block size exceeds 229 376 samples (112 × 2 048), the advantages of GPU parallelization become more evident. The polyphase filtering stage is further subdivided, and data from the two polarizations are processed asynchronously, allowing data transfer and computation to proceed in parallel. At a block size of 819 200 (400 × 2 048)

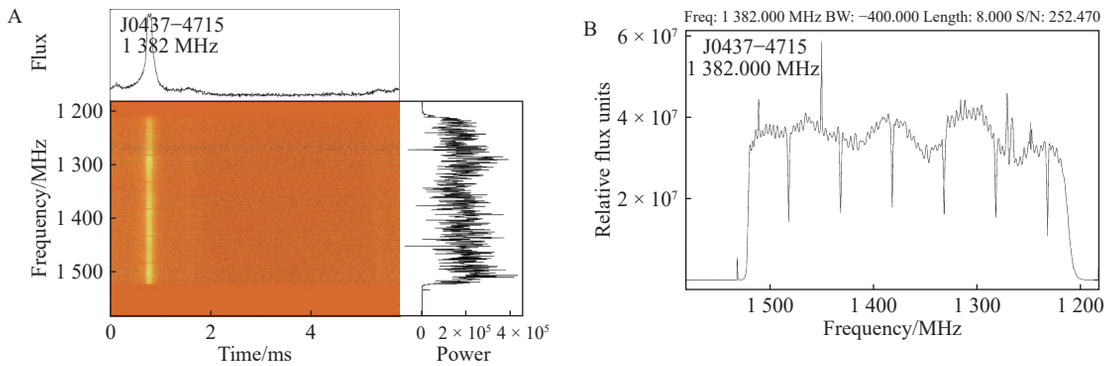


Fig. 9. (A) Dynamic spectrum of the eight-channel data synthesized from the GPFFC output. (B) The pulsar J0437−4715 bandpass spectrum of the GPFFC output after recombining the eight subchannels.

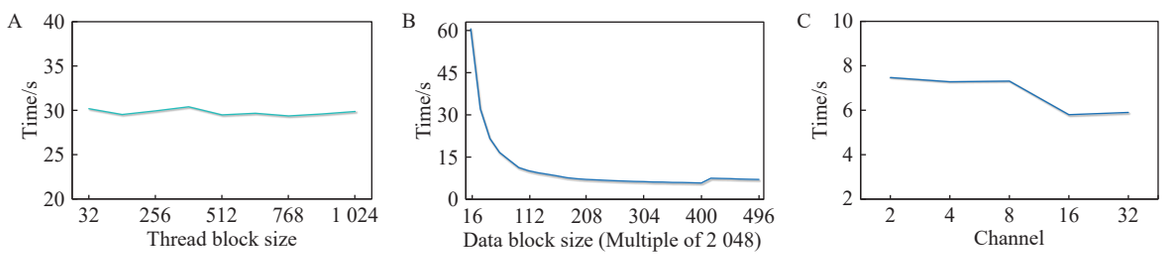


Fig. 10. (A) Total execution time of data unpacking, transmission, processing, and packing process for a data block size of 786 432 and different thread block sizes. (B) Time for the GPFFC to perform the channelization process when the thread block size is 768 and different data block sizes. (C) Time for the GPFFC to perform the channelization process for different numbers of channels.

and thread block size of 768, the system processes 8 s of baseband data with 400 MHz bandwidth in approximately 5.8 s, satisfying the quasi-real-time processing requirements of future QTT ultra-wide bandwidth low-frequency receiver systems. However, further increasing the block size (416 to 496 × 2 048) leads to heavy use of limited GPU resources, which constrains performance and results in increased execution time.

To evaluate the effect of channel count on GPFFC execution time, we conducted tests using the same input data and identical sub-filter order for all configurations. The number of output channels varied between 2, 4, 8, 16, and 32, and the execution time was averaged over multiple runs. The results are shown in Fig. 10C. As the number of channels increases, the parallelization of the algorithm improves, leading to a reduction in execution time.

4. CONCLUSION

To meet the real-time processing requirements of multi-functional digital backends for ultra-wide bandwidth signals in radio astronomy, we designed and implemented the GPFFC algorithm, which enables fine-grained real-time channelization of wideband data. We validated the GPFFC using 8-s dual-polarization baseband data of pulsar J0437–4715 with a 400 MHz bandwidth. The input signal was divided into eight sub-bands using the GPFFC, and each sub-band was stored in the DADA baseband data format. Subsequent processing with the standard pulsar software DSPSR yielded correct pulse profiles for each sub-band. When combined, the wideband pulse profile was found to be consistent in phase and amplitude with that obtained directly from the raw baseband data, confirming the correctness of the GPFFC. To evaluate its performance, we fixed the thread block size at 256 and determined the execution time across multiple runs. The shortest total time, comprising data unpacking, transfer, processing, and repacking, was 29.94 s for a data block size of 786 432 samples (384 × 2 048). Further analysis revealed that setting the thread block size to 768 reduced the total time slightly to 29.37 s under the same data block configuration. In practical applications, the GPFFC is not expected to start from data unpacking, nor require repacking after channelization. Consequently, we measured the time required solely for data transfer from CPU to GPU and the channelization process. As shown in our tests, increasing the number of output channels to 2, 4, 8, 16, and 32 enhanced algorithm parallelization and reduced execution time. Overall, the results demonstrate that the GPFFC effectively accelerates wideband signal channelization. For 8-s baseband data with a 400 MHz bandwidth, the channelization process requires approximately 5.8 s, thus enabling real-time processing of data with up to 400 MHz bandwidth on an NVIDIA GeForce RTX 3090 GPU, satisfying the quasi-real-time processing requirements for multi-channel data in future ultra-wide bandwidth low-frequency receiving systems such as QTT. For bandwidths exceeding 400 MHz, a single GPU may not be able to process the data because of memory capacity limitations, but multi-GPU or heterogeneous distributed platforms provide a feasible solution. In addition to pulsar observation, the fine-grained channelization of the

GPFFC is also suitable for high-precision pulsar timing, radio interferometric imaging, molecular spectral line observations and FRB detection.

ACKNOWLEDGEMENTS

This work is supported by the Natural Science Foundation of Shandong Province (ZR2024QA219).

AI DISCLOSURE STATEMENT

AI-assisted technology is not used in the preparation of this work.

AUTHOR CONTRIBUTIONS

Meng Zhang provided the research topic, conceived the ideas, and designed the study. Luxin Zhang wrote and tested the algorithm. Yazhou Zhang participated in discussions on program writing issues and provided critical revisions. All authors read and approved the final version of the manuscript.

DECLARATION OF INTERESTS

The authors declare no competing interests.

REFERENCES

[1] Harris, F., Dick, C., Rice, M. 2003. Digital receivers and transmitters using polyphase filter banks for wireless communications. *IEEE Transactions on Microwave Theory and Techniques*, **51**(4): 1395−1412.

[2] Leung, C., Andrew, S., Masui, K. W., et al. 2024. A VLBI software correlator for fast radio transients. *arXiv: 2403.05631*.

[3] Curyło, M., Pennucci, T. T., Bailes, M., et al. 2023. Wideband timing of the Parkes pulsar timing array UWL data. *The Astrophysical Journal*, **944**(2): 128.

[4] Liu, W., Meng, Q., Wang, C., et al. 2021. An efficient channelization architecture and its implementation for radio astronomy. *Journal of Instrumentation*, **16**(8): P08047.

[5] Zhang, W. X., Yao, Y. S., Zhao, Z. K., et al. 2019. Design and FPGA implementation of a novel efficient FRM-based channelized receiver structure. *IEEE Access*, **7**: 114778−114787.

[6] Sha, Z., Feng, H., Shi, Y., et al. 2018. Phasesensitive optical time domain reflectometer with ultrafast data processing based on GPU parallel computation. *Applied Optics*, **57**(10): 2679−2685.

[7] Price, D. C. 2021. Spectrometers and polyphase filterbanks in radio astronomy. In the WSPC Handbook of Astronomical Instrumentation: Volume 1: Radio Astronomical Instrumentation. World Scientific, 159–179.

[8] Girish, B., Reddy, S. H., Sethi, S., et al. 2023. Progression of digital-receiver architecture: From MWA to SKA1-Low, and beyond. *Journal of Astrophysics and Astronomy*, **44**(1): 28.

[9] Zhang, Y. Z., Zhang, H. L., Wang, J., et al. 2024. UWLPIPE: Ultra-wide bandwidth low-frequency pulsar data processing pipeline. *Research in Astronomy and Astrophysics*, **24**(7): 075011.

[10] McCurry, M. 2011. CUDA based polyphase filter. REU Program, MIT Haystack.

[11] van der Veldt, K., van Nieuwpoort, R. V., Varbanescu, A. L., et al. 2012. A polyphase filter for GPUs and multi-core processors. In Proceedings of the 2012 workshop on High Performance Computing for Astronomy Date.

[12] Chennamangalam, J., Scott, S., Jones, G., et al. 2014. A GPU-based wide-band radio spectrometer. *Publications of the Astronomical Society of Australia*, **31**: e048.

[13] Merry, B. 2023. Efficient channelization on a graphics processing unit. *Journal of Astronomical Telescopes, Instruments, and Systems*, **9**(3): 038001.

[14] Wang, N., Xu, Q., Ma, J., et al. 2023. The Qitai radio telescope. *Science China Physics, Mechanics & Astronomy*, **66**(8): 289512.

[15] Wang, Y. B., Zhou, X., Kurban, A., et al. 2024. Bounding the photon mass with ultrawide bandwidth pulsar timing data and dedispersed pulses of Fast Radio Bursts. *The Astrophysical Journal*, **965**(1): 38.

[16] Li, W., Pober, J. C., Hazelton, B. J., et al. 2018. Comparing redundant and sky-model-based interferometric calibration: A first look with Phase Ⅱ of the MWA. *The Astrophysical Journal*, **863**(2): 170.

[17] Jaeggli, S. A., Daw, A. N. 2024. Molecular hydrogen line identifications in solar flares observed by IRIS: Lower atmospheric structure from radiometric analysis. *The Astrophysical Journal*, **976**(1): 18.

[18] Siemion, A. P., Bower, G. C., Foster, G., et al. 2011. The Allen Telescope Array Fly's eye survey for fast radio transients. *The Astrophysical Journal*, **744**(2): 109.

[19] Mckinven, R., Michilli, D., Masui, K., et al. 2021. Polarization pipeline for fast radio bursts detected by CHIME/FRB. *The Astrophysical Journal*, **920**(2): 138.

[20] Zhang, H. L., Zhang, Y. Z., Zhang, M., et al. 2023. Research on a coherent dedispersion algorithm for pulsar baseband data. *Research in Astronomy and Astrophysics*, **23**(1): 015023.

[21] Zhang, Y. Z., Zhang, H. L., Wang, J., et al. 2024. PSRDP: A parallel processing method for pulsar baseband data. *Research in Astronomy and Astrophysics*, **24**(1): 015025.

[22] Zhang, M., Zhang, H. L., Zhang, Y. Z., et al. 2023. Research on channelization techniques of radio astronomical wideband signal with oversampled polyphase filter banks. *Research in Astronomy and Astrophysics*, **23**(8): 085012.