The Astrophysical Journal Supplement Series, 279:8 (21pp), 2025 July https://doi.org/10.3847/1538-4365/adc89e
© 2025. The Author(s). Published by the American Astronomical Society.
OPEN ACCESS

Pulscan: Binary Pulsar Detection Using Unmatched Filters on NVIDIA GPUs

Jack White[1], Karel Adámek[1], Jayanta Roy[2], Scott M. Ransom[3], and Wesley Armour[1]
[1] Oxford e-Research Centre, Department of Engineering Science, University of Oxford, 7 Keble Road, Oxford OX1 3QG, UK; wes.armour@oerc.ox.ac.uk
[2] National Centre for Radio Astrophysics (NCRA-TIFR), Pune 411 007, India
[3] National Radio Astronomy Observatory, Charlottesville, VA 22903, USA
Received 2024 June 21; revised 2025 February 26; accepted 2025 February 27; published 2025 June 20

Abstract

The Fourier domain acceleration search (FDAS) and Fourier domain jerk search (FDJS) are proven matched-filtering techniques for detecting binary pulsar signatures in time-domain radio astronomy data sets. Next-generation radio telescopes such as the SPOTLIGHT project at the Giant Metrewave Radio Telescope (GMRT) produce data at rates that mandate real-time processing, as storage of the entire captured data set for subsequent offline processing is infeasible. The computational demands of FDAS and FDJS make them challenging to implement in real-time detection pipelines, requiring costly high-performance computing facilities. To address this, we propose Pulscan, an unmatched-filtering approach that achieves order-of-magnitude improvements in runtime performance compared to FDAS while being able to detect both accelerated and some jerked binary pulsars. We profile the sensitivity of Pulscan using a distribution ($N = 10{,}955$) of synthetic binary pulsars (simulated post–radio-frequency interference mitigation) and compare its performance with FDAS and FDJS. Our implementation of Pulscan includes an OpenMP version for multicore CPU acceleration, a version for heterogeneous CPU/GPU environments such as NVIDIA Grace Hopper, and a fully optimized NVIDIA GPU implementation for integration into an AstroAccelerate pipeline, which will be deployed in the SPOTLIGHT project at the GMRT. Our results demonstrate that unmatched filtering in Pulscan can serve as an efficient data reduction step prior to FDAS or FDJS, selecting data sets for further analysis and focusing subsequent computational resources on likely binary pulsar signatures.

Unified Astronomy Thesaurus concepts: Binary pulsars (153); Millisecond pulsars (1062); Radio pulsars (1353); Pulsars (1306); Radio astronomy (1338); GPU computing (1969); Computational methods (1965)

1. Introduction

Detecting pulsars in binary systems is crucial because they serve as unique laboratories for testing theories of gravity. Their presence in extreme gravitational environments, combined with their function as extremely precise celestial clocks, allows for highly accurate timing measurements of their dynamics.

Detecting solitary pulsars is possible by searching the frequency spectrum of a time-domain signal captured by a radio telescope for peaks corresponding to the fundamental and higher harmonic frequencies of the spin frequency of the pulsar. However, in binary pulsar systems the relative motion between the pulsar and the observer, due to the orbit of the pulsar around its companion, leads to a Doppler shift in the observed frequency of the pulsar's signal. The Doppler shift spreads the frequency response of the pulsar across a range of frequencies, making detection more complex, due to the reduced local prominence caused by having the same total power spread across multiple frequency bins, in contrast to all the power being in a single frequency bin.

The Fourier domain acceleration search (FDAS) was proposed in S. M. Ransom (2001). It is based on matched filtering that operates on the frequency spectrum of the observed radio signature and uses the fast Fourier transform (FFT) to accelerate the convolutions required for the matched-filtering process.

While the FFT is an $O(N\log N)$ algorithm for calculating the discrete Fourier transform (DFT), and hence provides favorable performance compared to directly calculating the DFT using an $O(N^2)$ algorithm, the key computational challenge lies in the convolution process, with typical searches including over 100 matched filters.

In this paper we investigate replacing the matched filters in FDAS with boxcar filters and examine whether this new approach retains the ability to detect binary pulsars with the significant advantage of reduced execution time, allowing greater volumes of data to be processed in real time. We present multiple implementations of the algorithm, optimized for both CPU and GPU hardware. The implementations of Pulscan discussed in this paper are available in our freely accessible repository.[4]

1.1. SPOTLIGHT Project at the GMRT

SPOTLIGHT[5] is a commensal survey instrument at the Giant Metrewave Radio Telescope (GMRT; funded by the National Supercomputing Mission, Government of India) aiming to discover a large population of fast radio bursts (FRBs) associated with its host galaxies and pulsars. While the GPU-based petaflop-scale high-performance computing (HPC) facility is primarily designed for a real-time FRB search over ∼2000 beams, the facility will also be equipped with the capacity to perform a matched-filtering-based pulsar search using FDAS/Fourier domain jerk search (FDJS) over 10% of

Original content from this work may be used under the terms of the Creative Commons Attribution 4.0 licence. Any further distribution of this work must maintain attribution to the author(s) and the title of the work, journal citation and DOI.

[4] Repository: https://github.com/jack-white1/pulscan. A copy of the initial release is deposited to Zenodo (J. White 2025).
[5] https://spotlight.ncra.tifr.res.in/

the real-time data set. The output from the real-time multibeam pipeline based on the AstroAccelerate-compatible GPU version of Pulscan will be used as the sifting metric to determine which beams and dispersion measure (DM) trials will be recorded for subsequent processing. Pulscan will also be used to accelerate the FDAS/FDJS process by reducing the fraction of Fourier bins that need to be searched. We present the results of testing Pulscan in a synthetic environment designed to be relevant to this application in Sections 4 and 5.

2. Searching for Binary Pulsars

In this section we will explore a more detailed description of the fundamental algorithms that are used in each approach.

2.1. Matched Filtering

A matched filter is a template of the exact signal of interest. In the context of searching for binary pulsars, searching a data set with matched filtering involves convolving the complex frequency spectrum of a dedispersed time-domain signal collected on a radio telescope with a range of matched filters. Each filter corresponds to a pulsar signature that has spread across a certain number of frequency bins, also referred to as r-bins. When there is a match between the frequency spectrum and the matched filter, there will be a spike in the response at the spin frequency of the pulsar. Essentially, the filter consolidates the power spread across multiple Fourier bins back into a single output bin. The number of bins over which the signal has drifted during the course of the observation is referred to as z. The parameters (r and z) of the template that leads to a match can be used to infer details of the parameters of the orbital system that led to the creation of the signal in the frequency spectrum. Typically, there will also be a detectable response at integer multiples of r and z, which correspond to the higher harmonics of the signal, particularly in the case of low-duty-cycle pulsars. The higher harmonics are a by-product of the frequency decomposition of low-duty-cycle periodic signals.

In a standard FDAS, the matched filters correspond to the frequency smearing pattern of a linear chirp with the form

$$s(t) = A\cos(\phi_0 + \pi(2f_0 t + ct^2)), \tag{1}$$

where ϕ_0 is the initial phase, f_0 is the starting frequency, and c is the chirp rate. This creates the implicit assumption that FDAS will only lead to strong responses in cases where the pulsar signature exhibits a linear change in frequency during the observation and therefore closely resembles the matched filter. This linear approximation is only valid over short sections of a sinusoid, which is formalized in the following equation taken from S. M. Ransom (2001)

$$T_{\rm obs} \lesssim \frac{P_{\rm orb}}{10}, \tag{2}$$

where $T_{\rm obs}$ is the total integration time of the observation and $P_{\rm orb}$ is the period of the orbit of the binary system. The parameter that varies between the templates is referred to as z, which is a positive or negative number that corresponds to the direction and number of frequency bins that the signal drifted between during the observation. The astronomer can use the measured value of z to calculate the time derivative of frequency $\dot{f} = \frac{z}{T^2}$, which in turn leads to a physical measurement for the acceleration of the pulsar in $\mathrm{m\,s^{-2}}$ via

$$\text{Acceleration} = \frac{\dot{f}\,c}{f}, \tag{3}$$

where c is the speed of light, f is the fundamental spin frequency of the pulsar, and T is the integration time of the observation.

A standard FDAS can be extended into an FDJS (B. C. Andersen & S. M. Ransom 2018), where for each z-value extra templates are tested that include a linear change in acceleration. The goal of an FDJS is to increase the upper bound on the $T_{\rm obs}/P_{\rm orb}$ range of sensitivity from 0.10 to 0.15, so that systems with a shorter $P_{\rm orb}$ become detectable with a fixed $T_{\rm obs}$.

This leads to an extra dimension in which the templates can vary, referred to as $w = \dot{z}$, which is the number of z-bins that drifted during the course of the observation. Then, w can be used to calculate the rate of change of $\dot{f}$, which can then be used to calculate a physical "jerk" value for the pulsar in units of $\mathrm{m\,s^{-3}}$ using

$$\text{Jerk} = \frac{wc}{fT^3}. \tag{4}$$

While an FFT-based search for approximately periodic pulsars is a 1D search across r frequency bins, using FDAS extends this to a 2D search across r frequency bins and $2 \times z_{\rm max} - 1$ (to include positive and negative values of z) z-bins. A jerk search extends this further to a 3D search across an r, z, w volume.

Since the jerk search can be thought of as consisting of multiple FDAS searches at varying values of w, if the astronomer wants to search a wide range of z- and w-values, or wants to perform an FDAS on a high-resolution time series (large r), the computational cost can become impractical to perform in real time.

2.1.1. Matched Filtering with PRESTO

The de facto standard implementation of matched filtering for binary pulsar searching on CPUs is implemented in PRESTO (S. M. Ransom et al. 2003).

PRESTO is a toolkit for processing time-domain radio astronomy data, but in this work we are particularly concerned with the `accelsearch` program. This program implements both FDAS and FDJS, which the user can select by passing nonzero arguments to the `-zmax` and `-wmax` flags, respectively.

In this work we use PRESTO as a gold standard for quantifying the sensitivity and execution time of a CPU-based, FDAS/FDJS matched-filtering approach.

2.1.2. Significance Calculation in FDAS/FDJS

For this analysis we will consider zero-mean, unit-variance additive white Gaussian noise (AWGN). The complex FFT spectrum, denoted as $\boldsymbol{X}$, can be formalized as follows, where r

represents the number of frequency bins in the FFT spectrum:

$$\begin{aligned} & \boldsymbol{X} = [x_0,\ x_1, \ldots, x_{r-1}], \\ \text{where:}\quad & x_i = a_i + b_i j, \\ & j^2 = -1, \\ & a \sim \mathcal{N}(0,\ 1), \\ & b \sim \mathcal{N}(0,\ 1). \end{aligned} \tag{5}$$

Let the individual complex matched filters, denoted as $\boldsymbol{M}$, where z indicates the number of Fourier bins drifted by the signal, be defined by

$$\begin{aligned} & \boldsymbol{M} = [m_0,\ m_1, \ldots, m_{z-1}], \\ \text{where:}\quad & m_i = c_i + d_i j. \end{aligned} \tag{6}$$

For each combination of r_i and z, a matched filter corresponding to the drift z is applied at the frequency bin r_i. The result is a linear combination of samples from a Gaussian distribution:

$$\boldsymbol{Y}_{i,z} = [x_i \cdot m_0 + x_{i+1} \cdot m_1 + \ldots + x_{i+z_{\max}-1} \cdot m_{z_{\max}-1}]. \tag{7}$$

Consequently, $\boldsymbol{Y}$ remains a complex array, with both its real and imaginary components being Gaussian random variables.

When the output of FDAS/FDJS is searched for candidates, for each value $y_{i,z}$ in the complex $\boldsymbol{Y}$ array, the sum of squares of the real and imaginary components is calculated as $|y_{i,z}|^2$. Since the sum of squares of k independent standard normal random variables follows the χ^2 distribution $\chi^2(2)$ (with 2 degrees of freedom), one can calculate a p-value for each point using the survival function of the $\chi^2(2)$ distribution. This p-value must be adjusted using the Bonferroni correction for the number of independent trials, as detailed in Equation (6) in B. C. Andersen & S. M. Ransom (2018). The p-value can then be converted into a Gaussian-equivalent sigma using the survival function for the normal distribution.

2.2. Boxcar Filtering

Instead of using a matched filter, Pulscan implements a unit-amplitude boxcar filter. This describes the profile of a group of frequencies appearing together with the same magnitude.

By applying this processing to the real-valued magnitude squared of the FFT spectrum, information about phase has been discarded, so it is impossible to tell from this approach the exact nature of the frequencies that are grouped together.

In practice, this means that the astronomer cannot tell whether the pulsar is accelerating toward or away from the observer without further processing the result with a phase-sensitive technique, an example of which would be FDAS/FDJS.

2.2.1. Magnitude versus Phase

Figure 1 shows the fundamental peak of a 1.69 ms pulsar J1227−4853 discovered at the GMRT. This particular observation shows a strong response at a z-value of −10 and a w-value of −10 when processed with FDJS, implying that the signal is moderately accelerated and lightly jerked. Each plot is drawn from the complex FFT spectrum of a dedispersed time series collected at the GMRT, the details of which are provided in Section 4.4. Figures 1(a) and (b) show the real and imaginary components of the complex FFT spectrum surrounding and including the fundamental peak. As described in Section 2.1, these are the data that a complex matched filter will operate on. It can be seen that the overlaid FDJS matched filter is a good approximation to the data in both the real and imaginary components of the complex data. When the data from Figures 1(a) and (b) are combined to produce the power spectrum in Figure 1(c), it is possible to compare the unit-amplitude boxcar filter used by Pulscan with the matched filter used by FDJS. The FDJS filter is a significantly better approximation to the profile of the fundamental peak, and this is reflected in the higher significance of the detection that was made by FDJS, shown in Table 3. However, while we acknowledge that the sensitivity drop caused the application of boxcar filters to this detection problem, the motivation behind Pulscan is to maximize the overall number of detections for a constrained compute budget. Specifically, the goal of Pulscan is to maximize the amount of data that can be processed in real time. The benefit of the unmatched filter in Pulscan is the symmetric, uniform structure of the filter, which we have exploited using the methods described in Section 2.2.4 to maximize the throughput of a Pulscan-based data reduction step.

2.2.2. Boxcar Filtering in Pulscan

Boxcar filtering has been implemented in many software packages for many purposes, including PRESTO and Astro-Accelerate, where it is used to search dedispersed time series for single pulses, such as those caused by FRBs.

Pulscan recursively computes a complete set of boxcar-filtered FFT magnitude squared spectra, using the same input data as PRESTO's `accelsearch`, which are .fft files.

2.2.3. Significance Calculation in Pulscan

The input data to Pulscan are a complex FFT spectrum. Each component, real and imaginary, of the spectrum is normalized independently to ensure that they each follow a zero-mean unit-variance Gaussian distribution.

To achieve this normalization, the input data are divided into blocks. For each block, the median serves as a robust estimator of the central location, and the median absolute deviation (MAD) is used as a robust scale estimator. However, since the MAD alone is not a consistent estimator for the standard deviation of a Gaussian distribution, it is scaled by the factor $1/(\phi^{-1}(3/4))$, where ϕ^{-1} is the quantile function of the standard Gaussian distribution. This adjustment ensures that the transformed data have unit variance under the assumption of Gaussianity.

The formula for the normalized data point, after considering the aforementioned adjustments, is

$$\text{Normalized Data} = \frac{\text{Data} - \text{Median}}{\text{MAD} \times \frac{1}{\phi^{-1}\left(\frac{3}{4}\right)}}. \tag{8}$$

In contrast to FDAS and FDJS, the next step after normalization in Pulscan involves calculating the square of the magnitude of the FFT spectrum. In the case of AWGN, this reduces the input data to a real-valued χ^2 distribution with $k = 2$ degrees of freedom for each frequency bin r_i.

Following this step, $Z_{\max}$ unit-amplitude boxcar filters of incremental widths in the range $[2, 3, \ldots, z_{\max} + 1]$ are applied to the FFT magnitude spectrum. The user can select the step size to determine which boxcar-filtered spectra are searched for candidates. However, all must be calculated owing to the recursive implementation of the algorithm in Pulscan, detailed in Section 2.2.5. As each boxcar-filtered output is essentially a sum of $z_i + 1$ independent χ^2 random variables (each

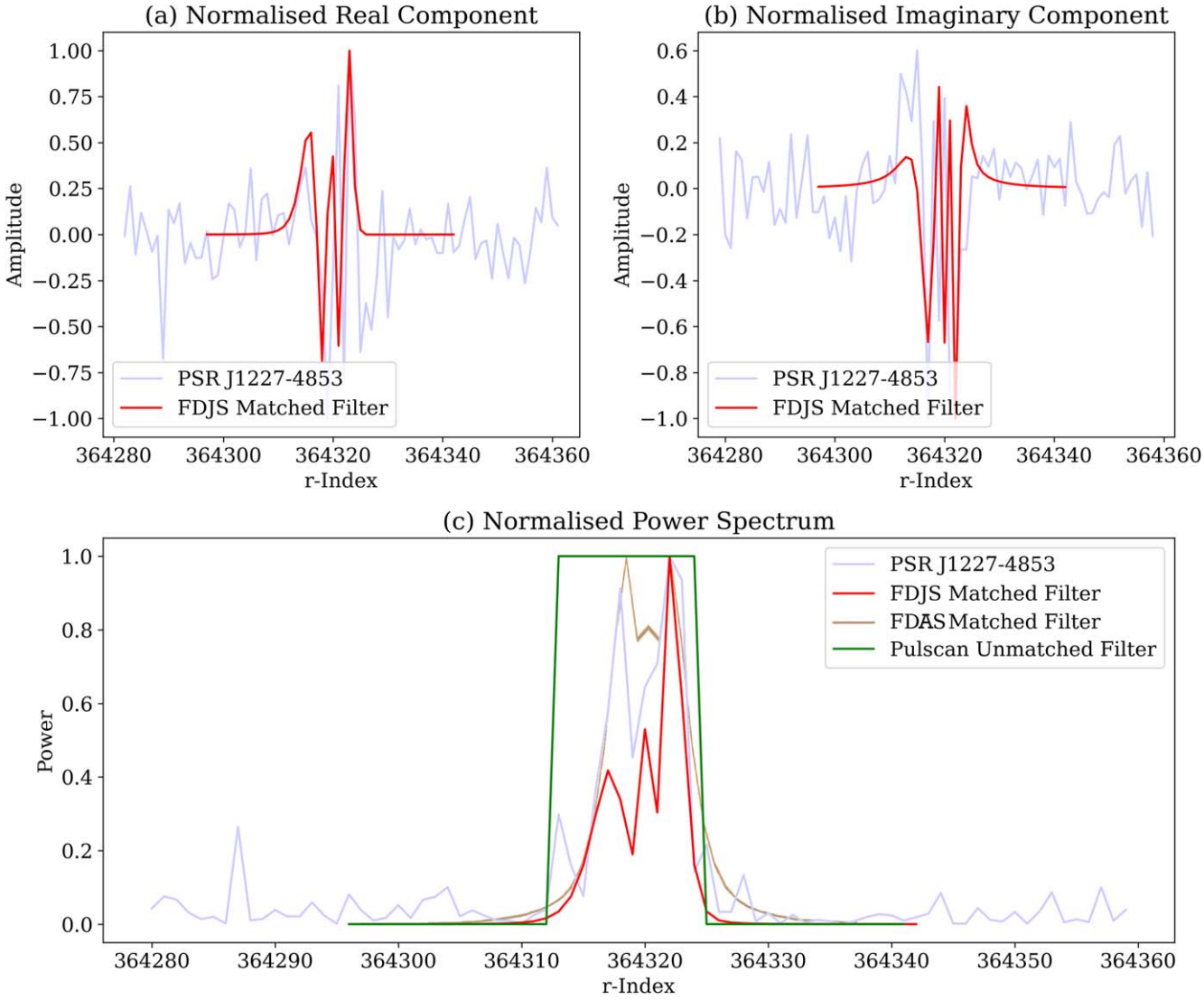


Figure 1. Fundamental peak section of the FFT spectrum of a dedispersed observation of PSR J1227−4853, collected with the GMRT. Overlaid are the matched and unmatched filters that led to the highest-significance candidate from FDJS, FDAS, and Pulscan. All overlaid filters are normalized to a peak magnitude of 1 to facilitate comparison of the profile, rather than the exact magnitude scale that would have led to the detection response at that location in the frequency spectrum. Both FDAS and FDJS have distinct profiles in the real and imaginary space; we present only FDJS to avoid overcrowding of panels (a) and (b).

individually having $k = 2$ degrees of freedom), the output follows a χ^2 distribution with $2 \times (z_i + 1)$ degrees of freedom:

$$\text{Filtered Output}_{r_i, z_i} \sim \chi^2(2 \times (z_i + 1)). \quad (9)$$

To evaluate the significance of any detected peaks, one can compute a p-value for each data point using the survival function of the corresponding χ^2 distribution. These p-values must then be adjusted using the Bonferroni correction to account for the multiple comparisons made across different boxcar widths and frequency bins. Finally, the adjusted p-values can be converted into a Gaussian-equivalent sigma, providing a normalized measure of significance for each candidate event.

2.2.4. Execution Time Efficiency of Boxcar Filtering

Boxcar filtering offers computational benefits that, when combined, potentially make it more time efficient than FDAS. These advantages are derived from four primary factors:

1. *Computational economy through recursion*. The algorithmic design of boxcar filters can maximize computational efficiency by employing recursion to calculate successive filters. This optimization minimizes the total number of calculations required.
2. *Data volume reduction*. Unlike FDAS, which operates on the complex FFT spectrum, the boxcar filters in Pulscan operate on the real-valued squared magnitude of the same spectrum. This effectively reduces the input data volume to the search by half.
3. *Reduction in required number of filters*. FDAS has separate (although related) filters for positive and negative accelerations, when the pulsar is accelerating toward or away from the observer. A boxcar-filter-based approach uses a single filter to gather the frequencies spread out by both positive and negative accelerations, reducing the number of template trials by a factor of 2.

4. *Parallelization suitability*. Both boxcar filtering and FDAS/FDJS are well suited for parallel implementation, particularly on GPUs. Prior research (S. Dimoudi et al. 2018) has demonstrated significant speed improvements in GPU-based implementations of FDAS, and subsequently FDJS in K. Adámek et al. (2020). Other work designing single pulse detection codes based on boxcar filtering for GPUs (K. Adámek & W. Armour 2020) has also resulted in implementations that run many times faster than real time.

2.2.5. Recursively Calculating Boxcar Filters in Pulscan

As mentioned, one of the advantages of employing boxcar filtering as opposed to matched filtering lies in the computational efficiency gained through recursive calculations. Specifically, an array filtered with a boxcar of width N can be incrementally computed by filtering with boxcar widths ranging from 2 to $N-1$, followed by the final boxcar filter of width N.

Let $\boldsymbol{X} = [x_1, x_2, \ldots, x_N]$ be the input array. Then, the unit-amplitude boxcar-filtered array $\boldsymbol{Y}_k$ for boxcar width k can be formulated as follows:

1. For $k = 1$:
$$\boldsymbol{Y}_1 = \boldsymbol{X}.$$
2. For $k = 2$ (output length $N-1$):
$$\boldsymbol{Y}_2 = [x_1 + x_2, x_2 + x_3, \ldots, x_{N-1} + x_N].$$
3. For $k = 3$ (output length $N-2$):
$$\boldsymbol{Y}_3 = [x_1 + x_2 + x_3, x_2 + x_3 + x_4, \ldots, x_{N-2} + x_{N-1} + x_N].$$

Noticeably, the array $\boldsymbol{Y}_3$ can be efficiently calculated using the array $\boldsymbol{Y}_2$ as follows:

$$\boldsymbol{Y}_3 = \boldsymbol{Y}_2[:\text{end} - 1] + \boldsymbol{X}[3:\text{end}].$$

In general, the array $\boldsymbol{Y}_k$ for boxcar width k can be recursively computed using the previous boxcar-filtered array $\boldsymbol{Y}_{k-1}$ as

$$\boldsymbol{Y}_k = \boldsymbol{Y}_{k-1}[:\text{end} - 1] + \boldsymbol{X}[k:\text{end}].$$

This recursive relation allows for substantial computational savings, particularly beneficial for large input data and extensive searches.

This is a computational advantage over FDAS/FDJS because it is not possible with the matched-filtering approach to reuse the result for a particular r, z, w to calculate any other; they must all be calculated from scratch.

2.3. Harmonic Sum

To increase the detection significance of low-duty-cycle pulsars where the power of the signal is spread across multiple integer harmonics of the fundamental frequency, one can use a harmonic sum on the r–z plane after performing FDAS. Successive harmonics will be detected with increasing integer multiples of r and z and can be summed accordingly. The goal is to produce a single peak with a higher significance than the individual constituent peaks corresponding to each harmonic. Since the distribution of power between the harmonics is not known a priori, sums over different numbers of harmonics can be performed to increase the likelihood of detection.

The approach to harmonic summing implemented in Pulscan involves a decimation and summing step on the input frequency magnitude spectrum.

Formally, assuming Gaussian input, the original frequency magnitude spectrum is represented as

$$X_1(r_1) \sim \chi^2(2), \quad \text{for } r_1 \in \left[0, \frac{N_{\text{samp}}}{2}\right].$$

The decimation employed in Pulscan involves summing adjacent nonoverlapping pairs, triples, quartets, etc., of frequency magnitudes to produce a magnitude spectrum decimated by factors of 2, 3, 4, etc., respectively.

The decimated magnitude spectrum required to align the Hth peak in the harmonic spectrum with the fundamental peak would be

$$X_H(r_H) \sim \chi^2\left(\sum_{n=1}^{H} 2n\right), \quad \text{for } r_H \in \left[0, \frac{N_{\text{samp}}}{2H}\right],$$

where $\sum_{n=1}^{H} 2n = H(H+1)$ degrees of freedom.

For example, to align the second harmonic with the fundamental, we take our input frequency magnitude spectrum, which is χ^2 distributed with $k = 2$, and decimate it by a factor of 2.

We then sum the overlapping sections of each decimated spectrum (aligned so the zeroth bin of each spectrum is summed) and discard the nonoverlapping parts. Then, we proceed to boxcar-filter this input as previously described, resulting in an output spectrum that is distributed following a χ^2 distribution with $k = H(H+1)(z_i+1)$ degrees of freedom. The significance of local maxima (candidates) on these output spectra can be calculated according to this distribution.

2.4. Approximate r-bin Location

Pulscan is intended to be used in conjunction with proven, highly sensitive techniques such as FDAS and FDJS. In this scenario, the goal is to use Pulscan to filter through data, ranking candidates by their significance and location on the r–z plane. Under these circumstances, it is possible to save only an approximate version of the exact frequency (r) location that Pulscan has identified for each candidate. The frequency spectrum is processed in blocks in all versions of Pulscan to enhance cache compatibility by ensuring that all the data that are being operated on at any given time can be stored in the smallest and fastest cache, such as the L1 cache in a CPU. By simply saving the block index (an approximate frequency location) of candidates rather than the exact r-bin they came from, it is possible to accelerate the process further. In the subsequent investigations of performance, the inclusion of this optimization will be referred to as "Pulscan R-Block," to reflect the approximate r block index that is saved. The block width is a user-defined parameter that governs the width (in r-bins) of the processing blocks. For the experiments in this investigation, the default value of 32,768 was used for all Pulscan R-Block tests. This value was chosen empirically as the default because it achieves good performance across a range of CPUs, as it is approximately the size of the L1 cache on the CPUs being tested in this investigation.

In a resource-constrained survey, high-sigma candidates from a Pulscan R-Block search could be passed to a follow-up FDJS to confirm detections. The FDJS would only have to operate on a small section of the frequency spectrum equal to

the block width, which would reduce the computational burden compared to an FDJS on the entire FFT spectrum.

2.5. Median and MAD Normalization on GPUs

In both the CPU and hybrid CPU/GPU implementations of Pulscan, normalization is performed on parallel chunks of the input spectrum on the CPU using the Quickselect algorithm to determine the median and MAD of each chunk. This process is trivially data-parallel by distributing chunks across CPU cores, and therefore it achieves meaningful speedup when run with more OpenMP threads.

For the GPU-native implementation of Pulscan, we preferred a vector-parallel approach to maximize throughput and avoid running conditional, branching code on the GPU. We chose to estimate the median of a chunk of the frequency spectrum using the median-of-medians algorithm. This can be implemented as a parallel reduction on the GPU, where threads recursively find the median of four nonoverlapping elements.

2.6. Significance (χ^2)log(p)) Calculation on GPUs

As can be seen in Sections 2.3 and 2.2.5, to ensure that the power distributions of candidates follow χ^2 statistics, the resulting powers for high-z candidates will be the sum of many constituent χ^2 powers. The significance calculation is critical, as it allows us to discriminate between candidates that have larger powers owing to the presence of underlying signal and those candidates that have high powers solely because they are being drawn from a χ^2 distribution with a high number of degrees of freedom and contain no signal.

In the CPU and hybrid CPU/GPU implementations of Pulscan, the cumulative distribution functions required for significance calculation are processed on the CPU after the search has been completed. To achieve higher throughput in the GPU-native version, we used linear regression to fit a function to approximate the output required. This resulted in the following formula:

$$\log(p) = \frac{\text{power} \cdot f(x)}{A_1}, \tag{10}$$

where $f(x)$ is defined as

$$f(x) = A_2 \times x^{16} + A_3 \times x^{15} + \ldots + A_{16} \times x^2 + A_{17} \times x + A_{18}, \tag{11}$$

x is defined as

$$x = \frac{A_1 \times \text{dof}}{\text{power}}, \tag{12}$$

and A_1, ..., A_{18} are constants derived from the linear regression process and are provided in Appendix B.

This polynomial equation can be executed in a vector-parallel approach across all candidates on the GPU instead of the highly branched cumulative distribution functions that the CPU version uses to calculate a Gaussian-equivalent sigma value. The $\log(p)$ value can be used as an input to a precomputed threshold that the user selects for their application.

3. Methodology

In this section we will describe and justify our experimental design.

Table 1
Configuration Arguments for SIGPROC `fake` Command to Generate a Binary Pulsar Signal

Parameter	Range	Units	Distribution
Spin period	[1, 10]	ms	log-uniform
Pulse width	[5, 50]	%	log-uniform
Input SNR	[0.01, 0.1]	⋯	log-uniform
Dispersion measure	[100, 1000]	cm^{-3} pc	log-uniform
Bits per sample	8	⋯	⋯
Number of channels	1024	⋯	⋯
Sampling time	128	μs	⋯
Observation time	600	s	⋯
First channel frequency	1550	MHz	⋯
Channel offset	−0.292968752	MHz	⋯
Additional flags	−binary	⋯	⋯
Orbital period	[1, 10]	hr	log-uniform
Starting orbital phase	[0, 1.0]	⋯	uniform
Pulsar mass	[0.5, 5.0]	$M_\odot$	log-uniform
Companion mass	[0.5, 5.0]	$M_\odot$	log-uniform
Orbital eccentricity	[0, 1.0]	⋯	uniform

Note. [a, b] represents the closed interval inclusive of the bounds. The input SNR values are per pulse and were chosen empirically to span the range of detectable to undetectable systems. This goal was achieved, demonstrated by the fact that only a subset of the data resulted in detectable binary pulsar signatures. Further details are provided in Section 4.1.

3.1. Setting the Parameter Space

We wanted to understand how Pulscan would compare to the ability of FDAS to detect highly accelerated binary pulsar systems, so we chose a parameter space that would lead to the creation of a range of millisecond pulsars in close orbits around their companion.

The distribution sampling information and parameter limits are contained in Table 1. Parameters that can span orders of magnitude (e.g., spin period, dispersion measure) were sampled using a log-uniform distribution to ensure a balanced representation across their ranges. Parameters confined to a specific range (e.g., orbital eccentricity) were sampled uniformly to evenly cover all possible values within that range. The parameters were sampled continuously within the specified ranges using the distributions indicated and 32-bit floating-point numerical precision.

It is important to note that the parameter range intentionally goes further than the constraint under which FDAS would be expected to maintain sensitivity, $T_{\text{obs}} < P_{\text{orb}}/10$. We chose a lower bound on the orbital period of 1 hr, which, compared with the constant 600 s observation time, demonstrates that we are violating this constraint.

We know from Z. Pan et al. (2023) that systems with an orbital period shorter than this are physically plausible, and this extreme edge of the parameter space is an important one to include in our data set.

Some combinations of parameters may not be physically realizable, but it is important that we understand how the signal recovery performance degrades when searching for systems beyond the boundaries of currently known binary pulsars.

To achieve sufficient covering of this high-dimensional space, we initially aimed to generate a sample size of $N_{\text{total}} = 10{,}000$ synthetic pulsars. During preliminary testing of our HPC setup, we generated an additional 955 pulsars. These extra samples were included in the final data set, bringing the total to 10,955 pulsars. Including these additional

pulsars enhanced the robustness of our analysis without compromising the integrity of the data set.

For some experiments, we chose to remove extremely bright pulsars that were detectable with a periodicity search, which does not account for the frequency smearing effect. The goal of removing pulsars detectable by a periodicity search is to focus our data set on a population of pulsars that are considered to be only potentially detectable by some form of acceleration search. Further details are provided in Section 4.1.

3.2. Generating Synthetic Data

We used the `fake` program in SIGPROC to generate synthetic binary pulsar signatures against a background of AWGN.

It is important to note that this will generate data with a fixed pulse profile (square pulses) in both time and frequency and an amplitude defined by the `snrpeak` parameter.

The data lack any undesirable sources of non-Gaussian background, such as radio-frequency interference (RFI). This enables us to make repeatable measurements of detection significance, as our assumptions of the χ^2 background noise from which our peaks are being drawn will be valid across the entire data set.

We dedispersed the filterbank files using the `prepsubband` program in PRESTO to the corresponding DM of the synthetic pulsar. We then generated the frequency spectrum using the `realfft` program in PRESTO.

This gave us a series of .fft files, each of which contained a pulsar signature with a known rest-frame spin frequency. This information is stored in a metadata file in the same subdirectory for subsequent processing.

3.3. Summary Statistics of the Synthetic Data

In order to understand how our simulated data set spans the parameter space of acceleration (z) and jerk (w) as measured by FDJS, we analyzed the output from FDJS across our full synthetic data set. In Appendix A, Figures 14 and 15 show the distribution of the z- and w-values measured by FDJS for our entire data set. This provides a sense of how the simulated pulsars are distributed in terms of these measured parameters, ensuring that the parameter space relevant to binary pulsar searches is well sampled. We acknowledge that we did not explicitly sample z and w uniformly, as our aim was to generate a distribution of binary pulsar parameters that spanned extreme combinations of orbital and observational parameters, rather than imposing arbitrary sampling that could bias our results.

3.4. Comparing Results from Different Pipelines

It is essential to ensure that results from separate pipelines are being compared fairly. In this subsection we will cover the key considerations to ensure that our experiments did not favor one technique over another.

3.4.1. Sensitivity

Both pipelines generate a Gaussian-equivalent sigma for each output candidate. These can be directly compared, as they have been treated to ensure that they represent the equivalent confidence level that the particular corresponding candidate has not arisen from a random Gaussian noise process.

To ensure that we are making a fair comparison of the ability of FDAS/FDJS and boxcar filtering to extract individual binary pulsar fundamental peaks from noise, we initially disabled harmonic summing in PRESTO using the `-numharm 1`, `-noharmremove`, and `-noharmpolish` flags. In subsequent tests evaluating the performance of relative harmonic summing approaches, we used the `-numharm 4` setting in both `accelsearch` and Pulscan.

In the sensitivity experiments, we defined detection as the ability of a particular method to produce a candidate above a certain confidence threshold within 1% of the fundamental harmonic frequency, i.e., the spin frequency of the pulsar we knew a priori that was present in the file being processed. The highest-sigma candidate meeting this criterion was isolated from the text-formatted output candidate lists from both `accelsearch` and Pulscan. Section 4 addresses the considerations necessary to use this detection criterion meaningfully.

3.4.2. CPU Implementation Performance

All figures quoted are averages over 16 runs, and error bars represent the measured sample standard deviation of the execution times collected.

As we are testing multiple (CPU, CPU/GPU hybrid, GPU-native) implementations of a boxcar-based acceleration search, we designed experiments to profile their performance in appropriate settings.

For the CPU version of Pulscan, we use the bash command `time`'s `real` output to time the end-to-end wall clock time of executing each program, `pulscan` and `accelsearch`. Notably, in addition to searching the data, this includes the overhead of reading the input file, normalizing the data, and writing the output candidate list. In the experiment where we disable the harmonic sum, we are isolating the minimum viable functionality to complete a binary pulsar search and maximizing the fraction of the execution time that is spent on the filtering operations. We focus on end-to-end speedup, as this is the most relevant figure for an astronomer who wants to determine whether the trade-offs of boxcar filtering are worth the speedup on CPUs.

We profiled the execution time of the code over a variety of input data sizes, each of which covers various combinations of observation integration time and time sampling rate.

We also profiled the execution time of the code when we varied the `-zmax` parameter. We adhered to the upper limit of 1200 that is present in the PRESTO implementation of FDAS/FDJS, although there is no upper limit on `-zmax` in our CPU implementation of Pulscan.

For the CPU/GPU hybrid version of Pulscan, we measure the total execution time of the program using the self-reported timing statements. This is to ensure that we do not include a measurement of the first GPU initialization process, which would not be incurred in an always-on real-time setting.

3.4.3. GPU Implementation Performance

Since the GPU implementation of Pulscan is intended as a data flagger to be integrated into an existing GPU-native data pipeline (AstroAccelerate), the way we have measured its performance differs.

The GPU version does not calculate an exact Gaussian-equivalent sigma for each candidate. This is because there is a large arithmetic overhead with calculating the relevant forward

and inverse cumulative distribution functions, so instead the GPU version records the log(p) value that would have been used as an input to the Gaussian-equivalent sigma function. Since the GPU version is intended as a binary data flagger, these log(p) values can then be compared with precomputed threshold values that have been chosen as detection criteria.

Additionally, as the GPU implementation of Pulscan will be integrated into a pipeline, the code will not suffer the overhead of data transfers, as it will pick up where the previous stage in the pipeline finished with preprepared data in preallocated memory locations. Therefore, we define the execution time of the GPU version as solely the execution time of the GPU kernels relevant to Pulscan, ignoring the overhead of data movement to and from the GPU. When making comparisons to the CPU/GPU hybrid version, we apply the same logic and only measure the core execution time of the Pulscan operations, not reading or writing the relevant files.

4. Results: Sensitivity

In this section there are two main metrics that we are interested in quantifying for each approach:

1. The sensitivity of the detection method, i.e., the number of pulsar fundamentals that can be detected against background noise. A true matched filter achieves optimal signal recovery (when measured as signal-to-noise ratio (SNR)) against a Gaussian noise background, so for all techniques we are measuring how well the filters being used approximate the binary pulsar signals we have synthesized.
2. The ability of the detection method to accurately measure the physical parameters of the binary system, primarily the spin frequency (r-bin) and acceleration (proportional to z-bin) of the pulsar. Accurate measurements are beneficial, as they may lead to the ability to reduce the search space for follow-up confirmation and timing measurements using other techniques of the pulsar peaks initially detected by Pulscan.

We have generated 10,955 synthetic filterbank files ($T_{\rm obs} = 600$ s, $T_{\rm samp} = 128$ μs) and processed the dedispersed frequency spectra separately with PRESTO `accelsearch` using the `-zmax 0` option for periodicity results, up to `-zmax 1200` for FDAS results and `-zmax 1200` and `-wmax 2000` for jerk search results. For boxcar results we have used Pulscan with up to `-zmax 1200`. These upper bounds were chosen because they are the upper limits in PRESTO.

Our entire data set contained 10,955 synthetic filterbanks. As previously mentioned, due to the stochastic sampling of our binary system parameters, only a subset of these led to a binary pulsar signature that was undetectable by a periodicity search. As will be further discussed in Section 4.5, the data set was simulated without RFI, consistent with the assumption that RFI mitigation precedes the binary pulsar detection step.

4.1. Detection Significance

In this section we will compare the number of pulsar fundamentals that each technique was able to detect above a Gaussian-equivalent sigma of 6 with the harmonic sum disabled. After excluding those detectable by a periodicity search above a sigma of 2 (default value in `accelsearch`), there were 5751 simulated pulsars remaining in the data set.

This conservative threshold ensures that we exclude extremely bright pulsars that do not pose a challenge for detection. We felt that it was justifiable to use an overly strict (lower-sigma) threshold for excluding periodicity detections than our definition for detection in other experiments, as pulsars detectable by a periodicity search are not the focus of our investigation.

The size of our data set is primarily governed by the computational cost of performing these large-scale simulations and subsequent searches over synthetic data. We found that each FDJS data point required a node with 128 GB of memory reserved for 2 hr. Based on these values, to re-create our data set of 10,955 data points would require 2,804,480 GB (RAM) hours of reserved computing capacity, which comes with a significant associated environmental and financial cost.

As a reference, we performed a comprehensive FDJS using `-zmax 1200` and `-wmax 2000`. This detected the fundamental of 3127 pulsars above a Gaussian-equivalent sigma of 6.

To understand why FDJS did not detect all simulated pulsars above the 6σ threshold, we analyzed the properties of the missed pulsars. We found that the pulsars either not detected by FDJS (producing no relevant candidate) or detected with a sigma below 6 had an average input SNR value of 0.018. In comparison, those detected by FDJS with a sigma above 6 had an average input SNR value of 0.047. As can be seen in Appendix A (comparing Figures 12 and 16), our log-uniform sampling of the input SNR value has intentionally biased our range toward the low end, to cover the parameter space of when pulsars transition from being highly visible to becoming undetectable.

A detection was logged as the highest-sigma candidate with a reported frequency within 1% of the true rest-frame spin frequency of the pulsar that the data contained (known a priori). The implications of this detection criterion are explored further in Section 4.2. We chose to vary `-zmax` for both FDAS and Pulscan and measure the number of total and unique detections by each approach.

It can be seen from Figure 2 that both FDAS and Pulscan detect fewer than 3127 pulsar fundamentals above a sigma of 6, regardless of the `-zmax` setting. This demonstrates the value of the FDJS as a highly sensitive technique that will detect pulsars that are undetectable by these less computationally extensive methods. It is important to note that these results will be highly dependent on the underlying distribution of pulsar signatures in the data set being searched. It can be seen in Figure 6 that the comprehensive FDJS detected pulsar fundamentals spanning the full range of z-bin $0 \rightarrow 1200$, so it is expected that neither technique detected all the pulsars in the data set when the `-zmax` setting was significantly under 1200. There may also be pulsars present in the data set with a true z-value greater than 1200 (indicated by the collection of data points on the z-bin = 1200 vertical axis), but due to this value being the upper limit of `zmax` in PRESTO, we were not able to quantify the number that FDJS would have detected with this experiment.

Figure 2 shows that there are two key regions in which the relative performance of each technique differs, and each region demonstrates the benefits and drawbacks of matched and unmatched filtering.

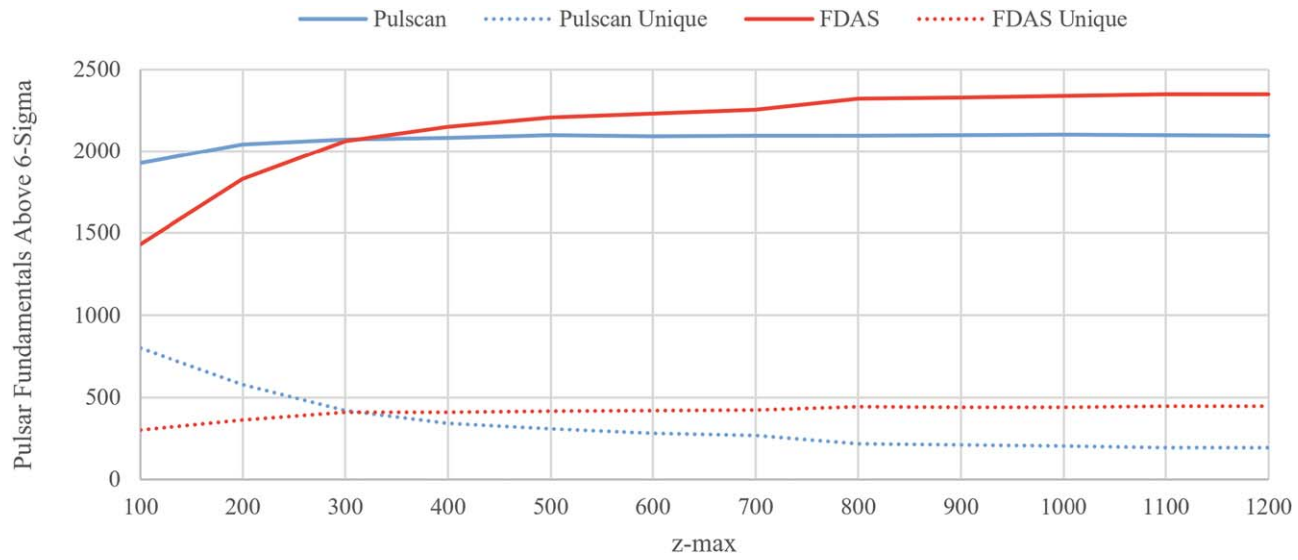


Figure 2. Step size for z-max is 100. Shown are the number of pulsar fundamental peaks (harmonic sum disabled) detected within 1% of their rest-frame spin frequency above the 6σ threshold against varying `zmax` values, after removing candidates that were detectable by a periodicity (`zmax 0`) search using `accelsearch`. The "unique" data series represent those that were not detected by the other detection method.

In the `-zmax` $<$ 300 region, Pulscan detects more pulsar fundamentals above the sigma threshold than FDAS and shows only a marginal improvement when the number of templates is increased by increasing the `-zmax` parameter from 100 to 200 and then to 300. In contrast, FDAS detects fewer pulsar fundamentals but shows a more marked increase in the number of pulsar fundamentals detected with the number of templates tested.

It is important to consider the results in the `-zmax` <300 region of Figure 2 alongside the results presented in Table 3 and Figure 6. Table 3 demonstrates that when either coherent technique (FDAS or FDJS) employs a sufficiently wide range of z-value or w-value templates to precisely match the pulsar's underlying frequency profile, those techniques achieve higher signal recovery (sigma) than Pulscan's unmatched boxcar filters. The results in Figure 6 demonstrate that FDJS detects pulsars spanning the range of z-magnitudes up to $z = 1200$ in our data set. Therefore, the results in the `-zmax` $<$ 300 region of Figure 2 indicate that the only circumstance under which Pulscan detects more binary pulsars than FDAS is when FDAS is underresourced and unable to perform sufficient matched-filter trials to match the pulsar signatures present in the data.

We confirmed this by calculating the average magnitude of z and w measured in the FDJS for the 777 "Pulscan unique" detections at `-zmax 100`. The average z-value for the FDJS detection using `-zmax 1200` and `-wmax 2000` in cases where Pulscan made a unique detection was 396. The average w-value for the FDJS detection in cases where Pulscan made a unique detection was 410. This demonstrates that by limiting FDAS to a `-zmax 100` search with `-wmax 0`, we are severely underresourcing the matched-filtering approach to fully cover the underlying distribution of binary pulsars in the synthetic data set. This is further emphasized by the rise in the number of detections corresponding to the rise in `-zmax` for FDAS.

However, the goal of Pulscan is to enable binary pulsar searching in resource constrained settings, and the key insight from these results is that in our experiments unmatched boxcar filters retain acceptable sensitivity under conditions of significant undersampling of the z-axis to cover the pulsar distribution of our data set. Further investigation is provided in Sections 4.1.3 and 4.3.

In the `-zmax` $>$ 300 region, FDAS detects more pulsar fundamentals above the sigma threshold than Pulscan. This once again demonstrates the value of matched filtering in situ ations lacking computational constraints, as they more closely match the underlying signal they are trying to detect and lift more fundamental peaks above the 6σ threshold required to be classified as a detection in this experiment. Figure 2 demonstrates that increasing `-zmax` in Pulscan results in diminishing returns in the high `-zmax` regime.

The majority of the pulsar fundamentals detected by each technique were the same, as shown by the low overall fraction of the total detection area covered by the area under the dotted lines for the unique detections. Notably, at all `-zmax` values Pulscan does detect some unique pulsar fundamentals. This suggests that if the cost of running Pulscan was low enough, it could still be included as an addition to an FDAS-based pipeline, to detect the remaining pulsar fundamentals that FDAS might have missed.

4.1.1. Unique Detections

At all the z-max settings tested (up to z-max $= 1200$), it can be seen in Figure 2 that both Pulscan and FDAS make a varying amount of unique detections. This represents the number of pulsar fundamentals that were not detected by the other method.

To minimize the fraction of pulsar fundamentals that were not detected owing to an insufficient z-max setting, we focused on the unique detections at z-max $= 1200$ for both FDAS ($N = 479$) and Pulscan ($N = 170$). Using the FDJS in PRESTO, we gathered the measured w-value for each unique detection and plotted a histogram of the values, presented in Figures 3 and 4.

It can be seen that the w-values of the unique detections of FDAS are distributed closely around $w = 0$. This is expected and demonstrates the primary goal of performing an FDJS. In contrast, the unique detections made by Pulscan have a broader range of w-values, spanning the range of $-2000 < w < 2000$.

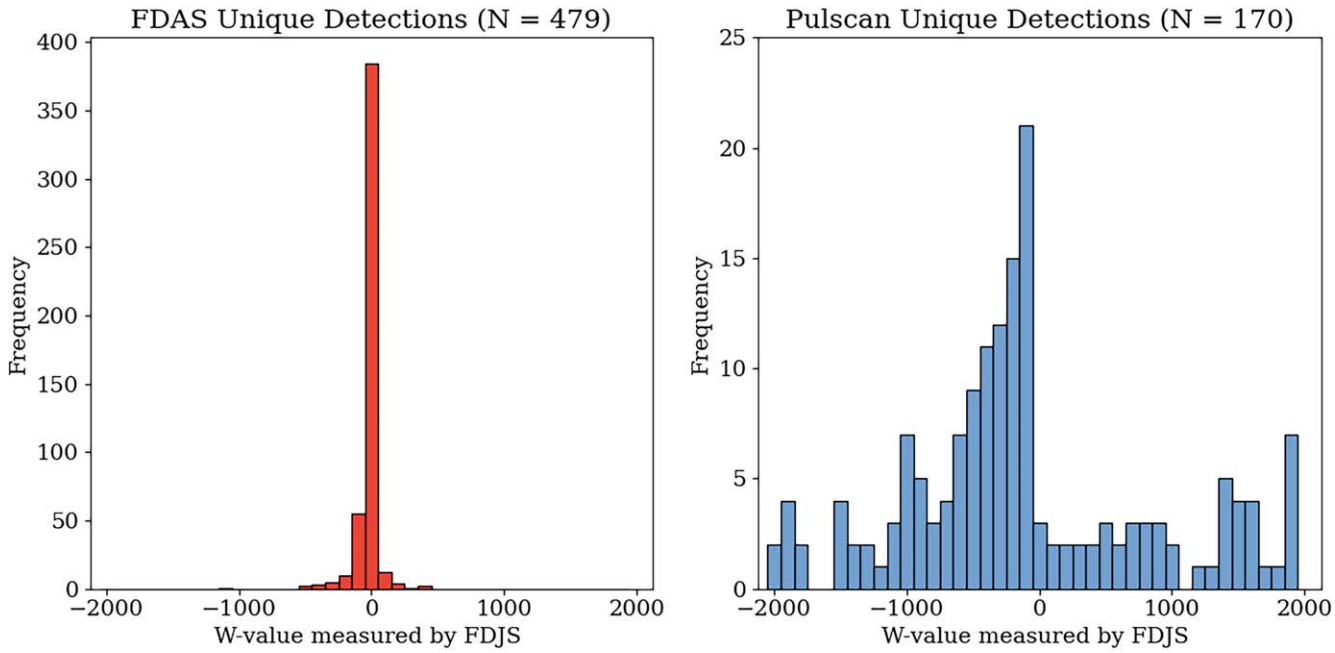


Figure 3. Histograms of w-values (measured by FDJS) of the unique detections by FDAS and Pulscan when run over the synthetic data set with a z-max setting of 1200. Unique detections were those for which each search technique (FDAS and FDJS or Pulscan and FDJS) reported a candidate above 6σ and the other search technique combination did not. All candidates detected by either FDAS or Pulscan or both were also detected by FDJS.

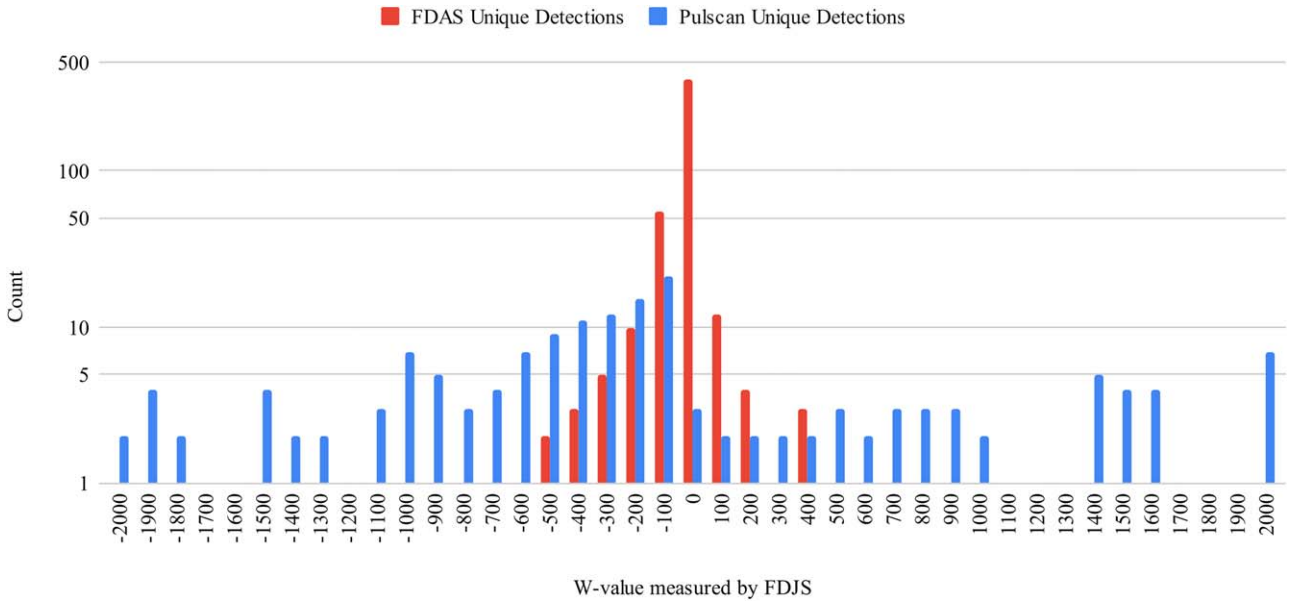


Figure 4. Same data as Figure 3, plotted on the same axes with a log y-scale. Shown are histograms of w-values (measured by FDJS) of the unique detections by FDAS and Pulscan when run over the synthetic data set with a z-max setting of 1200. Unique detections were those for which each search technique (FDAS and FDJS or Pulscan and FDJS) reported a candidate above 6σ and the other search technique combination did not. All candidates detected by either FDAS or Pulscan or both were also detected by FDJS.

This suggests that Pulscan can be used to detect jerked signals that FDAS would not be able to detect, assuming the same fixed 6σ threshold for each approach.

When run with z-max = 1200 over the synthetic data set, FDAS makes 479 unique detections, in contrast to the 170 unique detections made by Pulscan.

The ability of each approach to make their own unique detections could justify running both Pulscan and FDAS in parallel in surveys where sufficient computational capacity is available.

4.1.2. Harmonic Sum

We reperformed this experiment by excluding any files that contained a pulsar detectable above 6σ by a periodicity search with harmonic sum of up to four harmonics enabled (`-zmax 0` and `-numharm 4`). This left 5684 pulsars.

We then ran FDAS and Pulscan over the remaining data with `-zmax` settings of [50, 100, ..., 400] and the `-numharm 4` flag enabled, to finely sample the region where the two techniques were comparable in the previous experiment.

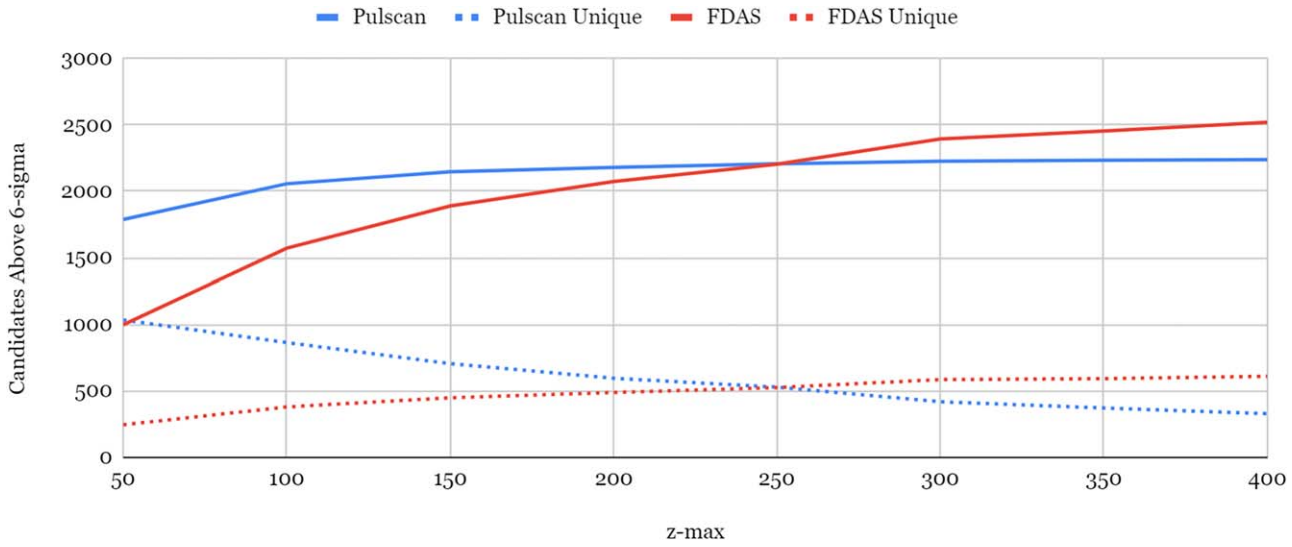


Figure 5. Number of candidates detected (harmonic sum enabled) within 1% of their true spin frequency above the 6σ threshold against varying `z-max` values, with `-numharm 4` setting enabled in both Pulscan and FDAS, after removing any candidates detectable by a periodicity (`-zmax 0 -numharm 4`) search using accelsearch. The "unique" data series represent those that were not detected by the other detection method.

Conclusions can be drawn from Figure 5 that are similar to those from Figure 2. At low `-zmax` settings, Pulscan makes more detections per filter than FDAS; however, this effect does not scale with an increased number of boxcar filters. As more matched filters are tested by increasing `-zmax` in FDAS, the number of pulsars detected above 6σ continues to rise.

4.1.3. Logarithmic Sampling of the z-axis

Due to the recursive nature of the boxcar calculation as it is implemented in Pulscan, regardless of the z-max setting, it is essential for every width of boxcar up to and including z-max to be calculated. Notably, a significant proportion of the time is spent searching each subsequent boxcar-filtered spectrum for candidates, and therefore we designed Pulscan GPU to calculate all boxcar filter widths but only search logarithmically spaced (i.e., $z = 1, 2, 4, 8, \ldots$ z-max) boxcar filter widths for candidates. This is possible in Pulscan owing to the versatility of boxcar filters demonstrated in the previous results.

To isolate the impact on sensitivity of this change, we implemented it as an option in the CPU version of Pulscan. Then, we ran Pulscan with the option both enabled and disabled over the full data set of synthetic pulsars.

With the logarithmic sampling option disabled, Pulscan detected 7202 candidates above a sigma threshold of 6, while detecting 7111 candidates with the option enabled. The value of this modification is a 4.1× speedup (average of eight repeats) when run with a z-max of 1024 and summing four harmonics when run on a single CPU code of an Intel Xeon Gold 6342. This speedup justified the inclusion of logarithmic z-sampling in the GPU version, as it is designed for the performance-sensitive real-time pipeline described in Section 1.1.

4.2. Pulsar Spin-frequency Measurement

We are evaluating the performance of Pulscan as a data reduction technique to pass candidates to a phase-sensitive matched-filtering technique such as FDJS. It is essential to establish that (1) there is no false-positive problem and (2) the degree of agreement between the reported r-bin location of candidates from Pulscan and FDAS allows us to pass a reduced r-range derived from the output of Pulscan as the input to FDJS.

4.2.1. False Positives

The synthetic filterbanks generated for the data set contain only AWGN and a single binary pulsar signature generated with square frequency and time profiles. They are not contaminated by any sources of RFI that we would experience using real data; therefore, we can reasonably assume that any significant candidate produced away from the integer harmonics of the binary pulsar spin frequency is a false positive.

In the previous section, we looked for a candidate within ±1% of the rest-frame spin frequency of the pulsar. This is convenient because it would correspond to a 50× reduction in the amount of data that would need to be passed to an FDJS to confirm a detection made by Pulscan.

To ensure that we were not ignoring a false-positive problem, we analyzed the output of performing Pulscan with `-zmax 1200 and -numharm 1` across the entire data set of 10,955 simulated pulsars. We blindly calculated the mean absolute percentage error (MAPE) of the frequency location of the highest-sigma candidate above the 6σ threshold, without using our a priori knowledge of the rest-frame frequency location of the underlying pulsar signal to select a candidate. The MAPE of the frequency location of the 3256 candidates above 6σ compared to the a priori known rest frequency location was 0.936%. There were 28 outliers with an error of between 99.5% and 100.5%, indicating that the first harmonic of the fundamental had been identified rather than the fundamental. The MAPE of the remaining 3228 candidates, not including the 28 outliers, was 0.0771%, and the maximum percentage error of the remaining 3228 was 1.19%. This suggests that over the synthetic data set Pulscan reliably detected either the first or second harmonic as the highest-sigma candidate.

4.2.2. r-bin Accuracy

After establishing that Pulscan is not generating spurious candidates at all frequencies to saturate the detection criteria

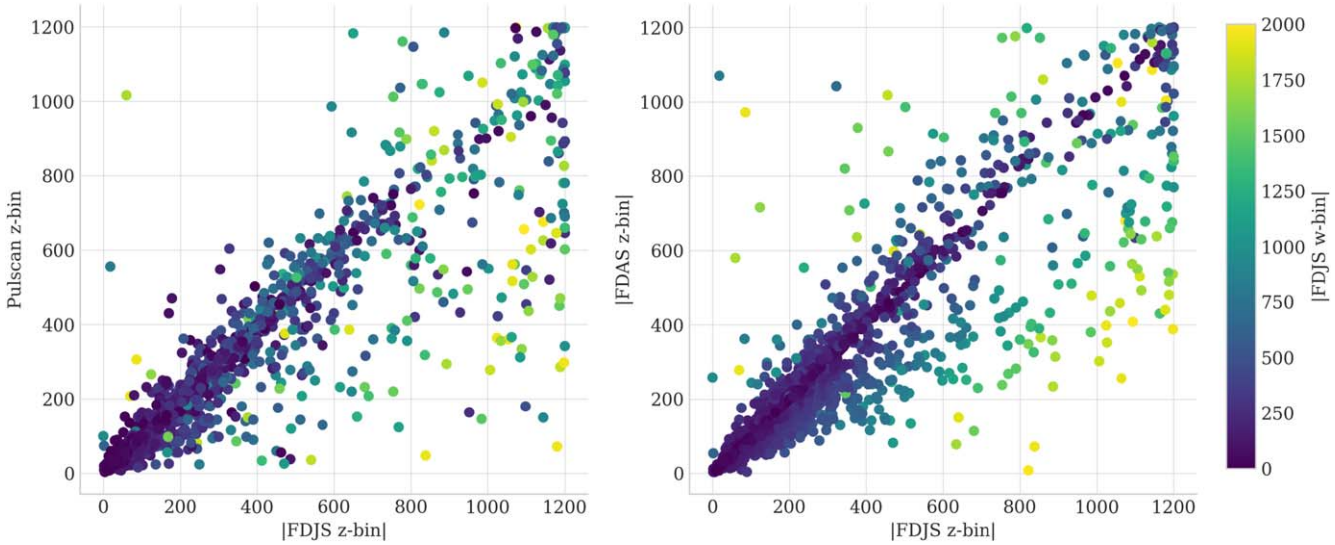


Figure 6. Comparison of the measured z-values for Pulscan and FDAS against the reference values measured by FDJS.

Table 2
Comparison of Linear Regression Parameters for the z-bin Measurements by Pulscan and FDAS

Metric	Pulscan	FDAS
Gradient	0.783	0.852
Intercept	23.3	13.9
R^2 value	0.827	0.841

with candidates, the next evaluation metric is how well the r-bin reported by Pulscan and FDAS for a given binary pulsar matches the r-bin reported by FDJS. To test this, we ran all three detection methods with harmonic summing disabled across the entire data set of 10,955 synthetic pulsars, using `-zmax` = 1200 for all methods and `-wmax` 2000 for FDJS. We selected the 6551 pulsar signatures where all of Pulscan, FDAS, and FDJS produced at least one candidate each above 6σ. We blindly selected the candidate with the highest significance from each of Pulscan and FDAS and compared the r-bin location with the highest-significance candidate reported by FDJS.

It is important to compare against the r-bin reported by FDJS rather than the predicted average frequency of the signature, as we are not evaluating Pulscan as a stand-alone detection method, and instead we want candidates from Pulscan to be a subset of those detected by a follow-up method, such as FDJS.

There were 6551 cases where Pulscan, FDAS, and FDJS all produced a candidate above 6σ.

In 6411 cases, the highest-significance candidate in FDAS was within 1200 r-bins of the highest-significance candidate in FDJS. Of these, the mean number of r-bins between the candidates reported by FDAS and FDJS was 7.01 and the maximum was 982. In all 140 cases where there was a discrepancy greater than 1200 r-bins, the discrepancy was manually verified to be a case of FDAS and FDJS detecting different harmonics of the same signal as the highest-sigma candidate. In all 140 cases, the r-bin reported by FDAS was within 0.011% of the frequency of a harmonic of the signal reported by FDJS.

This compares to 6373 cases where the highest-significance candidate from Pulscan was within 1200 r-bins of the candidate produced by FDJS. The mean number of r-bins between the candidates reported by Pulscan and FDJS was 9.08, and the maximum was 711. In the 178 cases where there was a greater discrepancy than 1200 r-bins between the Pulscan and FDJS r-bin, the maximum relative distance of the r-location of a Pulscan candidate to the r-location of a harmonic of the FDJS candidate was 0.0482%.

These results demonstrate that either FDAS or Pulscan could be used to reduce the size of the r-axis by providing candidates to be subsequently confirmed by FDJS.

4.3. Pulsar Acceleration Measurement

Using the same complete data set of 10,955 data points processed with Pulscan, FDAS, and FDJS as in the previous section, Figure 6 shows the measured z-bin values from Pulscan and FDAS, compared with those measured by the FDJS.

We rely on the z-value magnitude obtained from FDJS as our reference. As before, it is important to compare the value from Pulscan with the value from FDJS, as we intend to use Pulscan before using FDJS as a data reduction method.

The limitation of Pulscan to only discern magnitudes rather than signs of z-values stems from its symmetric unit-amplitude boxcar filters, which are phase insensitive and only have width as a tunable parameter.

We assess the performance of Pulscan and FDAS based on their ability to approximate this reference value using the data in Table 2. Neither Pulscan nor FDAS perfectly re-creates the results measured by FDJS. Although, FDAS demonstrates superior performance in replicating the z-values measured by FDJS, which is expected given that the set of matched filters in FDJS is a superset of those in FDAS.

In Figure 6, it can be seen that the degree of mismatch between the z-values reported by FDAS and FDJS correlates strongly with the measured value of w, represented by the color scale. This shows that FDAS is more likely to overestimate or underestimate the z-value in the presence of highly jerked signals. In the Pulscan results, the highly jerked signals are distributed over the chart area, indicating that the accuracy of a Pulscan z-bin measurement is less strongly dependent on the jerk level of the underlying signal.

Table 3
Highest-significance Candidate Identified by Each Search Method on Real Data from the GMRT

Technique	Hardware	zmax	wmax	Significance	r-bin	z-bin	w-bin	Execution Time (s)
Pulscan GPU	GPU	256	N/A	$\log(p) = -171.341$	364288	8	N/A	0.015[a]
Pulscan hybrid	CPU (24c) + GPU	256	N/A	$\sigma = 18.795$	364318	12	N/A	1.182
Pulscan R-block	CPU (24c)	256	N/A	⋯	360448	12	N/A	2.123
Pulscan	CPU (24c)	256	N/A	⋯	364318	12	N/A	2.328
Pulscan R-block	CPU (1c)	256	N/A	⋯	360448	12	N/A	4.75
Pulscan	CPU (1c)	256	N/A		364318	12	N/A	14.046
FDAS	CPU (1c)	256	N/A	$\sigma = 20.53$	364319	−9	N/A	28.291
FDJS	CPU (1c)	256	20	$\sigma = 20.76$	364319	−10	−10	N/A
FDJS	CPU (1c)	1200	2000	N/A	364319	−10	−10	N/A

Notes. The CPU being tested was an Intel Xeon Gold 6342, and the GPU was an NVIDIA H100 PCIe. The significance values are given in Gaussian-equivalent sigma (σ) units, except for the Pulscan GPU version, where significance is reported as $\log(p)$ $\log(p)$ owing to the implementation of the algorithms described in Section 2.6.
[a] Execution time not including the time taken to transfer the data to and from the GPU.

Neither Pulscan nor FDAS demonstrates the ability to reduce the range of z with a high degree of confidence for a subsequent FDJS around a candidate.

4.4. Real Data

To validate our findings using actual observations, we analyzed data from PSR J1227−4853, a known pulsar with a spin frequency of 592.988 Hz and an orbital period of 0.2788 days. Our data set comprises an observation of 614.4 s duration, equivalent to 60,000,000 samples. These data were obtained from the GMRT with a sampling time, $T_{\rm samp}$, of 10.24 μs over an observing frequency span of 550–750 MHz.

The initial processing involved dedispersion to a DM of 43.4 using PRESTO's `prepsubband`. Subsequent steps included the removal of powerline interference with `zapbirds` and a dereddening step using `rednoise`.

The `.fft` file generated was processed using FDAS in PRESTO and Pulscan.

The top candidates identified by each search method are presented in Table 3. The parameters listed in the table align with the fundamental peak of PSR J1227−4853. The results highlight that all the employed techniques list the significant binary pulsar signal as the highest-sigma candidate in the output file.

4.5. RFI Discussion

Our analysis of PSR J1227−4853 demonstrates that Pulscan can detect real binary pulsar signals in observational data. However, we acknowledge that the synthetic data set does not include RFI, which is present in real observations and can affect detection performance. A significant historical and recent literature exists in the field of mitigating RFI from time-domain radio astronomy data sets. Some recent examples are X. Zhang et al. (2024), H. Shan (2023), H. Shan et al. (2022), and J. M. Ford & K. D. Buch (2014). Hence, we assume that periodic RFI mitigation has been handled before the Pulscan step in the data pipeline. Testing Pulscan in situations where insufficient RFI mitigation has been performed is beyond the scope of this work, and we encourage investigation on a case-by-case basis.

5. Results: Execution Time

In this section we are primarily concerned with establishing the relative execution times of each approach. We present how long it takes to perform a variety of searches on various hardware configurations. This is key when planning surveys, as it will govern the boundaries of the search space that can be performed in time allocated on a shared cluster, or in real time on a dedicated cluster.

We perform separate investigations to benchmark the performance of Pulscan on CPU-based hardware and in GPU-accelerated environments.

Based on the relative balance between data production rate and available facilities to process the data, Pulscan is targeting situations where the data production rate is too high to store and subsequently do an archival search.

In these settings, there is evidently a strict requirement to process the data in less than real time, and we envisage that astronomers involved in these applications will be highly motivated to reduce the execution time of their desired search to less than the observation time collected.

There are four primary benefits to a faster search technique:

1. It can be run on more data (more dispersion measures, higher-resolution time series) in a real-time setting.
2. A more extensive search can be performed in real time (such as higher z-max).
3. The upfront cost of hardware required for a planned survey can be reduced.
4. The ongoing environmental and financial cost (due to the energy usage of HPC facilities) of running a given survey can be reduced.

5.1. FDAS versus Pulscan Execution Time

The performance measurements for the CPU versions of Pulscan and PRESTO FDAS were conducted serially on a high-end workstation equipped with an AMD Ryzen Threadripper PRO 3995WX 64-core CPU, 512 GB of RAM, and PCIe SSD storage. This enabled a comprehensive assessment of their capabilities in a resource-intensive setting.

In this subsection, we present a selection of experiments that profile the execution time of each approach.

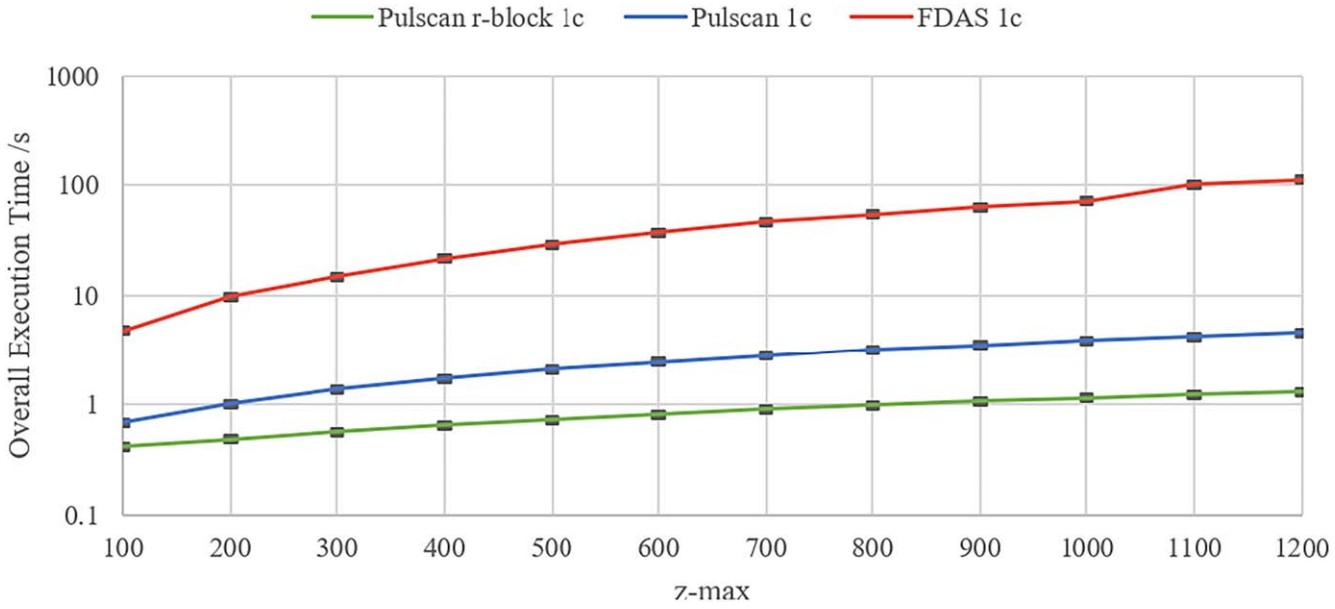


Figure 7. Comparison of total end-to-end execution time on a single CPU core of Pulscan and PRESTO FDAS (accelsearch) against various `-zmax` settings. The execution time was that required to process the same input data file, a PRESTO-compatible `.fft` file with 8388608 (2^{23}) samples. Error bars represent sample standard deviation over 16 repeats.

Since all observation files being processed across all experiments were 10 minutes long, it is possible to convert any execution time into a fraction of real time by dividing by 600 s.

The results of the first experiment in Figure 7 profile the execution time of Pulscan and PRESTO's implementation of FDAS under changing the `-zmax` setting on a single CPU core. This parameter represents the highest level of Fourier bin drift that the search is aiming to find over the course of the observation.

At zmax = 100, which is the least extensive search setting profiled, Pulscan using the approximate R-Block optimization took on average 0.42 s to process the data; disabling the R-Block optimization increased the execution time to 0.69 s. For the same `-zmax` setting, PRESTO's implementation of FDAS took 4.8 s.

At zmax = 1200, the most extensive search possible with PRESTO's implementation of FDAS, Pulscan took on average 1.32 s with the R-Block optimization enabled and 4.61 s with the optimization disabled. PRESTO's implementation of FDAS took 112.8 s using the same `zmax` setting. This demonstrates the reduced computational cost of searching extra boxcar filters by using a recursive algorithm. Figure 6 demonstrates the value of being able to search up to very high z-max values when measuring the parameters of highly accelerated systems.

At low `-zmax` values, both techniques are bottlenecked by nonfiltering operations such as loading the file and normalizing the data. These operations have to be done once regardless of the extent of the search and so put a minimum execution time on each approach, when the operations are run on a CPU.

The second experiment, presented in Figure 8, was to vary the input data size and measure the end-to-end execution time of each approach with a zmax = 1200 search on a single CPU core. Pulscan maintained a $>10\times$ speedup over PRESTO's implementation of FDAS across the entire range of input sizes tested.

Both Pulscan and PRESTO's `accelsearch` employ OpenMP for runtime parallelization. As demonstrated in Figure 9, Pulscan demonstrates notable improvement as the number of CPU threads increases.

Due to a known bug in the code, no measurable speedup was observed for FDAS and FDJS when using the `-ncpus` option. This issue is not critical, considering that processing single `.fft` files in isolation is not a typical use case for surveys, and the inherent data parallelism of processing one DM trial per CPU core can be leveraged as a workaround.

5.2. GPU Acceleration of Pulscan

In this subsection, we will compare the performance of our implementations of Pulscan on different hardware.

The first GPU-accelerated version of Pulscan has been referred to as the hybrid CPU/GPU version. This is because parts of the pipeline are processed on the CPU, while the core boxcar filtering section is performed on the GPU. The benefit of this approach is that it re-creates the CPU version of Pulscan, with a true Gaussian sigma significance value calculated and normalization using exact median calculations. In the CPU-only version of Pulscan, the bottleneck on performance is the boxcar filtering step, and so this was our first target to offload to the GPU.

The results in Figure 10 demonstrate the value of this approach on larger data sets (those with short sampling times at $T_{\rm obs} = 600$ s). As can be seen, the time spent searching on the GPU version is a small fraction of the overall execution time. This leads to a $5.5\times$ overall speedup of the processing of the data at a sampling time of 32 μs. The graph also demonstrates how the benefit of a hybrid CPU/GPU approach diminishes on smaller data sets. At a sampling time of 1024 μs, there is only a $1.2\times$ speedup when switching from the CPU-only code to the hybrid CPU/GPU implementation. This is an example of Amdahl's law, demonstrating that as the high-latency serial CPU operations become the dominant cost of the overall execution time, the benefit of parallelizing only a section of the pipeline diminishes.

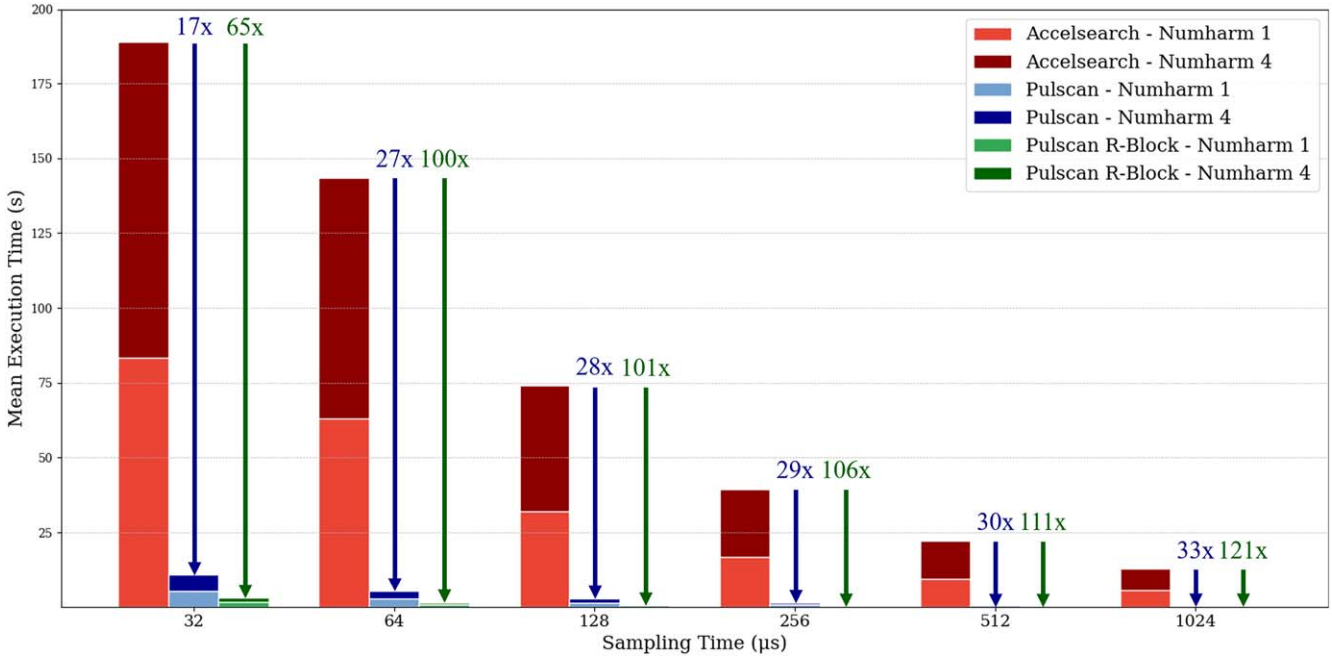


Figure 8. Comparison of end-to-end execution times of Pulscan and FDAS (accelsearch) on a single CPU core (AMD 3995WX) across different input data sizes. We generated a synthetic filterbank ($T_{obs} = 600$ s) with varying T_{samp} values, followed by dedispersion and Fourier transformation via PRESTO to produce `.fft` files. These files were subsequently analyzed using both accelsearch and Pulscan, performing a search with `zmax` = 1200. The reported execution times are average values over 16 trials.

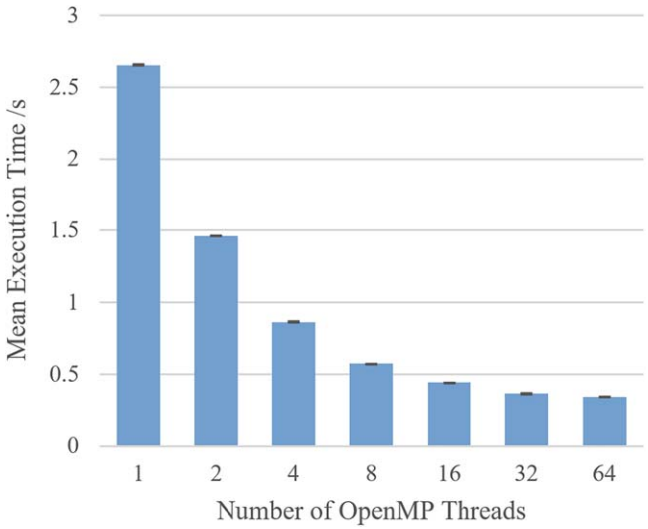


Figure 9. Comparison of end-to-end execution time of the CPU implementation of Pulscan against various `-ncpus` settings. The execution time was that required to perform a zmax = 1200 search with four harmonics summed on a PRESTO-compatible `.fft` file, derived from a 600 s T_{obs} filterbank, with T_{samp} of 128 μs. Error bars represent sample standard deviation over 16 repeats. The CPU being tested was an AMD Ryzen Threadripper Pro 3995WX.

This motivates the development of a fully GPU-native pipeline, which maximizes utilization of the available GPU hardware. By using the techniques described in Sections 2.6 and 2.5 (with algorithms detailed in Appendix B), Pulscan GPU is able to run all operations on the GPU. This minimizes the overhead of high-latency serial operations that run on the CPU. In addition, as Pulscan GPU is designed to be integrated into AstroAccelerate, which is a fully GPU-based pipeline, the GPU version will not have to wait for data to be transferred from the CPU memory to the GPU memory. As before, for the following benchmarks, we exclude the execution time of loading the input data from disk to memory. Further, as the output of Pulscan GPU will be passed to another section of the pipeline that also runs on the GPU, we can exclude the execution time of writing the candidates to disk.

Figure 11 demonstrates the increased speedup achieved by performing all operations on the GPU. Both the CPU and GPU versions of Pulscan perform a boxcar acceleration search with median-based normalization, four harmonics summed up to a z-max of 256, and significance calculation, but the NVIDIA H100 (GH200) GPU implementation is between 70× and 250 × faster than the Pulscan CPU with 24 OpenMP threads on an Intel Xeon Gold 6342. This is a result of the combination of higher memory bandwidth being available on the GPU, higher computational throughput (FLOP/s), and GPU-optimized algorithms we designed for normalizing the data, performing the boxcar search, and calculating the significance of candidates. The key utilization statistics of the GPU version are detailed in Table 4. The longest executing kernel, corresponding to the boxcar filtering search step, achieves high (approximately 80%) utilization of the GPU, demonstrating how effectively the algorithm maps to the GPU hardware. The other kernels show moderate utilization but account for a smaller fraction of the overall pipeline execution time.

6. Conclusions

The analysis presented in this work helps us to understand the performance and capabilities of an accelerated pulsar search based on boxcar filtering. The following points are the key summaries from each section:

1. *Sensitivity and detection.* FDJS demonstrates the value of a matched-filtering approach, achieving greater sensitivity by detecting more pulsars above a given Gaussian-

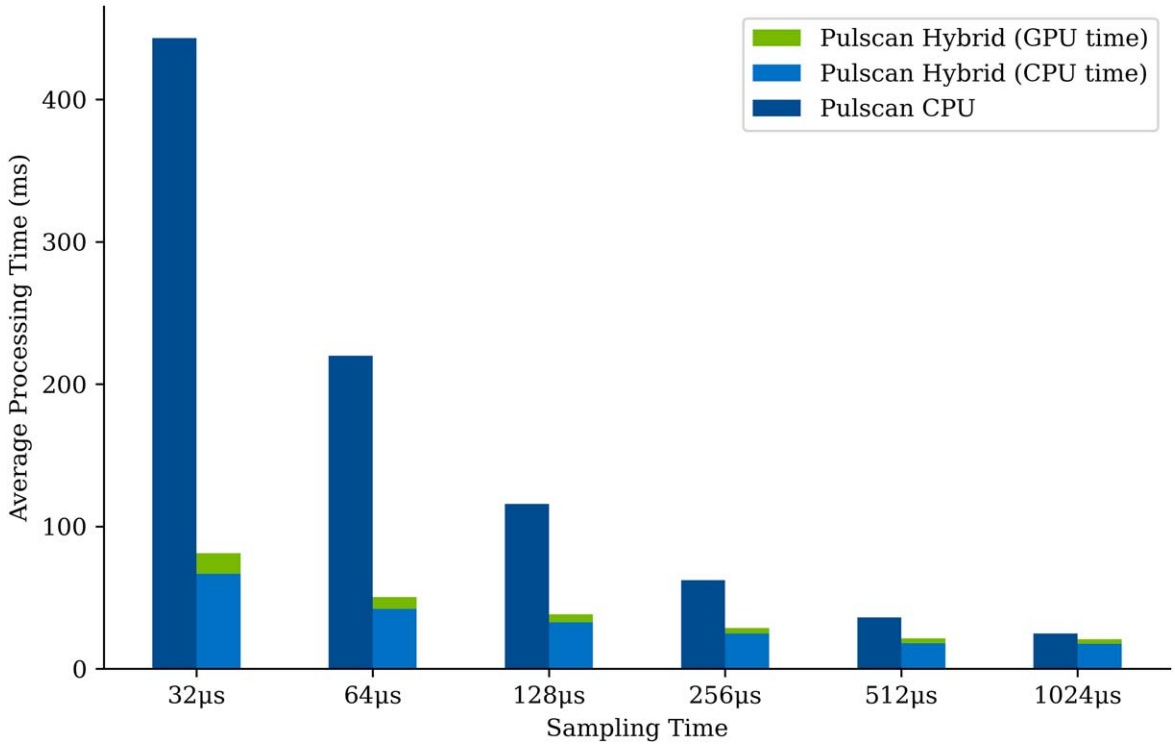


Figure 10. Comparison of processing times of Pulscan CPU vs. Pulscan hybrid CPU/GPU. The boxcar filtering was offloaded to the GPU in Pulscan hybrid. The CPU being tested was an Intel Xeon Gold 6342 (using 24 OpenMP threads), and the GPU was an NVIDIA H100 PCIe. We generated a synthetic filterbank ($T_{\rm obs} = 600$ s) with varying $T_{\rm samp}$ values, followed by dedispersion and Fourier transformation via PRESTO to produce `.fft` files. These files were subsequently analyzed using Pulscan, performing a search with `zmax` $= 256$ and `numharm` $= 1$. The reported processing times are average values over 16 trials, excluding the time taken to read the data into memory and write the output to disk.

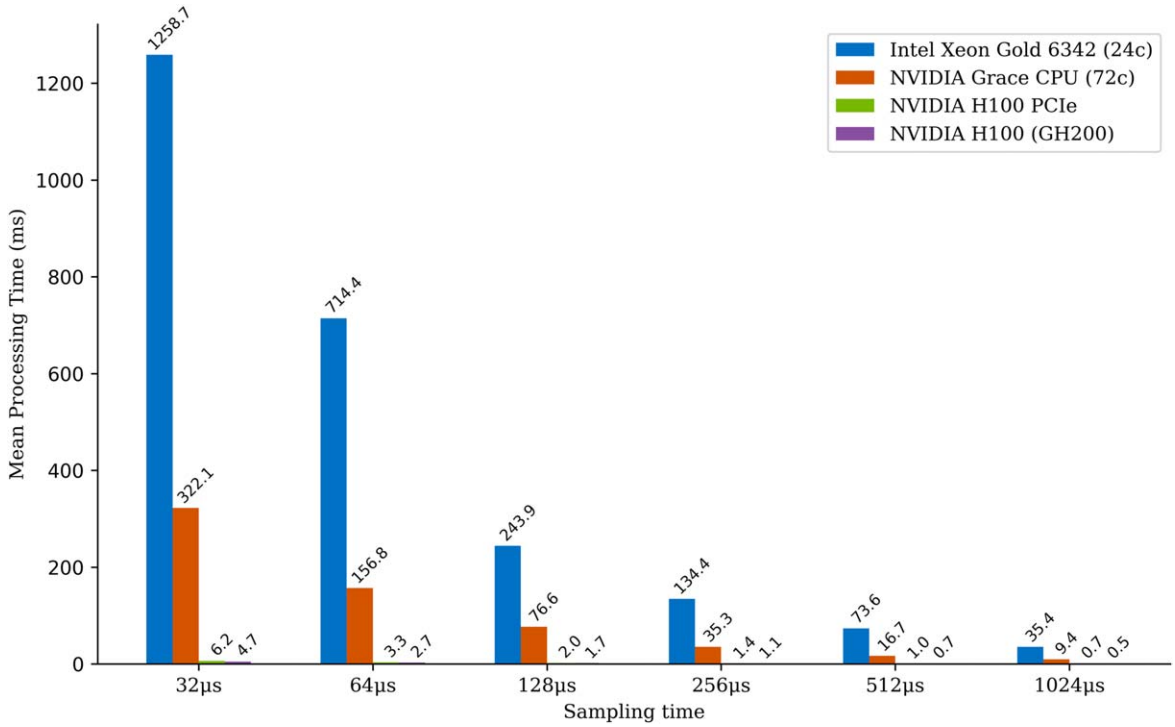


Figure 11. Comparison of processing times of Pulscan CPU on an Intel Xeon Gold 6342 (using 24 OpenMP threads) vs. Pulscan CPU on an NVIDIA Grace CPU (using 72 OpenMP threads) vs. Pulscan GPU on NVIDIA H100 PCIe vs. Pulscan GPU on NVIDIA H100 (GH200) across different input data sizes. We generated a synthetic filterbank ($T_{\rm obs} = 600$ s) with varying $T_{\rm samp}$ values, followed by dedispersion and Fourier transformation via PRESTO to produce `.fft` files. These files were subsequently analyzed using Pulscan, performing a search with `zmax` $= 256$, `numharm` $= 4$, and `chunkwidth` $= 256$. The z-axis was logarithmically sampled in both the CPU and GPU versions. The reported processing times are average values over eight trials, excluding the time taken to read the data into memory and write the output to disk.

Table 4
Performance Metrics of Relevant Pulscan GPU Kernels Reported by NVIDIA NSight Compute on an NVIDIA GH200 Grace Hopper Superchip When Processing a $T_{obs} = 600$ s, $T_{samp} = 128$ μs Observation Frequency Spectrum with z-max = 256

Kernel	Duration (μs)	Compute Throughput (%)	Memory Throughput (%)
SeparateComplexComponents	11.46	20.03	40.85
MedianOfMediansNormalisation	14.4	38.61	39.67
MagnitudeSquared	11.3	22.37	40.24
DecimateHarmonics	9.31	36.63	47.19
BoxcarFilterArray	420.26	79.34	83.35
CalculateLogP	23.26	51.7	2.91

Note. When maximizing overall throughput on the GPU, it is important that the majority of execution time is spent on kernels with high utilization of available compute and memory resources.

equivalent sigma compared to the other methods tested. However, the execution time of FDJS is significantly greater than that of FDAS or Pulscan, which motivates the development of a hybrid approach in resource-constrained applications.

2. *Pulsar parameter measurement.* All three techniques—Pulscan, FDAS, and FDJS—have demonstrated reliable accuracy in measuring the fundamental spin frequency of an accelerated pulsar when a detection is made. FDAS outperforms Pulscan in approximating the reference z-bin values obtained from FDJS.
3. *Real data validation.* The real-world data from the GMRT of compact binary pulsar J1227−4853 reaffirmed the utility of all techniques and give an indication that our findings on synthetic data will generalize to real data. Every method detected the significant binary pulsar signal as the candidate with the highest significance.
4. *Execution time.* In all tests, across all configurations, Pulscan takes less time to execute than the implementation of FDAS in PRESTO when run with equivalent settings. The advantage becomes greater as a greater fraction of the overall processing time is spent on the search, rather than pre- or post-processing.
5. *GPU acceleration.* The parallel nature of boxcar filtering can be exploited by implementation on GPUs, showing further speedups.

In conclusion, this study provides a comparison of FDAS and Pulscan, emphasizing the strengths and limitations of each. The findings underscore the importance of selecting the right approach based on the specific goals and constraints of a given astronomical survey and give an early indication that Pulscan could be used in conjunction with existing techniques such as FDAS or FDJS to find potentially exotic binary pulsar systems.

Acknowledgments

W.A. and K.A. would like to acknowledge support received from STFC grant (ST/W001969/1). J.W. would like to acknowledge support from EPSRC (2595728). The authors would like to acknowledge the use of the University of Oxford Advanced Research Computing (ARC) facility in carrying out this work (A. Richards 2015).

The National Radio Astronomy Observatory is a facility of the National Science Foundation operated under cooperative agreement by Associated Universities, Inc. S.M.R. is a CIFAR Fellow and is supported by the NSF Physics Frontiers Center awards 1430284 and 2020265.

Appendix A
Additional Figures

Appendix A is provided to add detail to the distribution of pulsars in our synthetically generated data set, beyond the scope of our existing discussion. Figures 12 and 13 represent the distributions of synthetic input SNR and synthetic period which were provided to the SIGPROC fake program. Figures 14, 15, and 16 represent the distribution of measured values, using the Fourier domain jerk search technique to analyse the same data set.

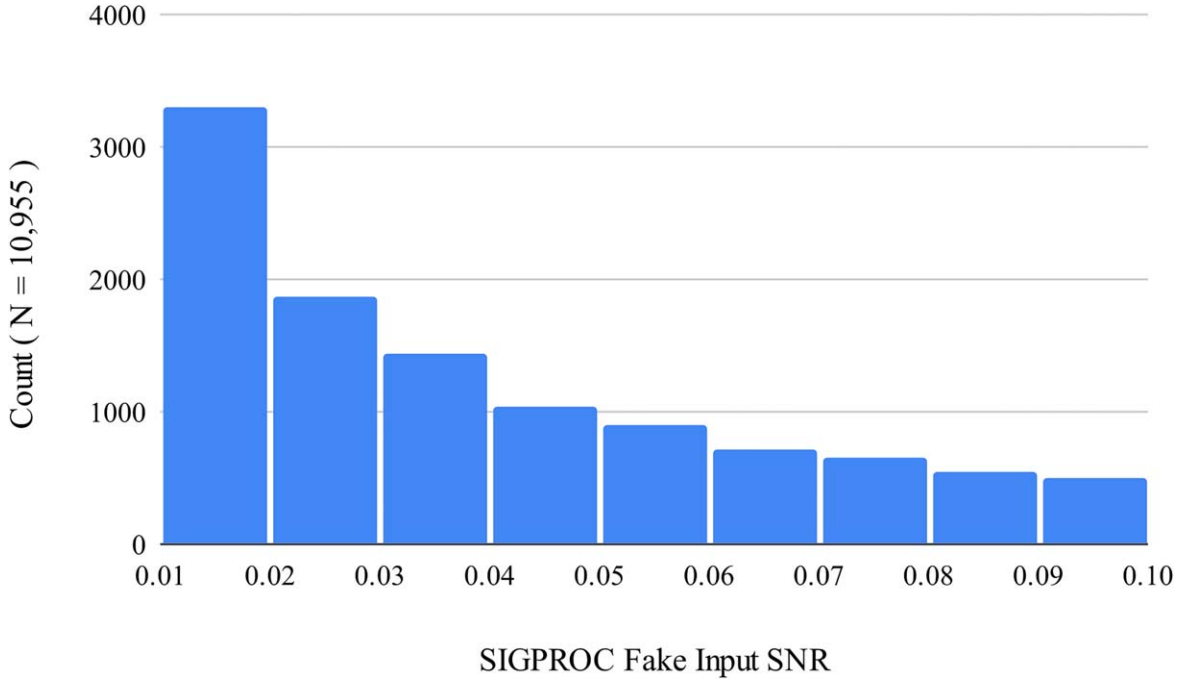


Figure 12. Histogram reporting the distribution (log-uniform sampling) of SIGPROC input SNR values for the total synthetic data set ($N = 10{,}955$).

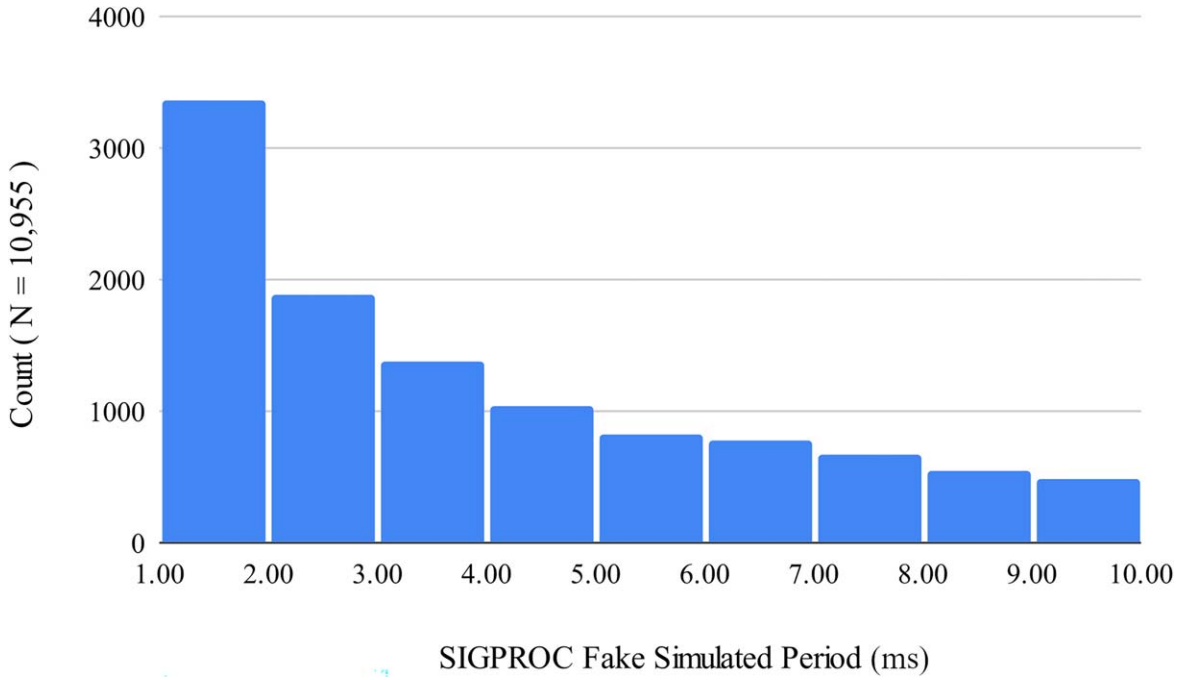


Figure 13. Histogram reporting the distribution (log-uniform sampling) of SIGPROC period values used for the total synthetic data set ($N = 10{,}955$).

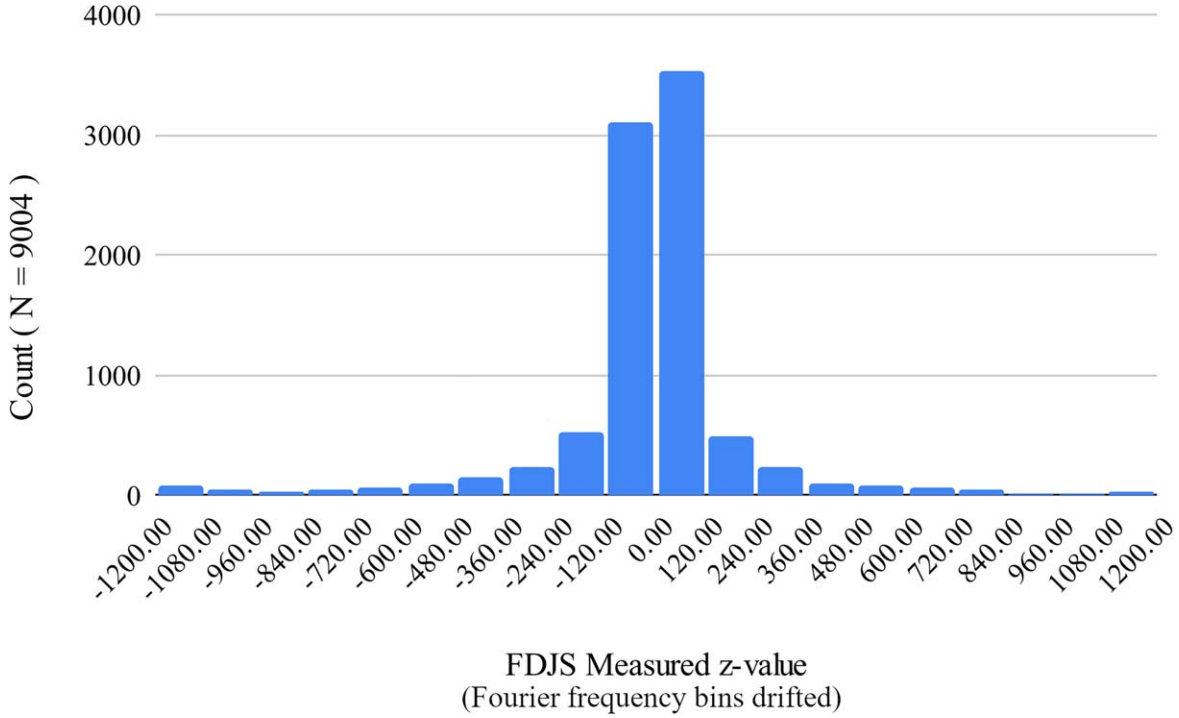


Figure 14. Histogram reporting the distribution of z-values measured by FDJS from PRESTO of the entire data set where a detection was made within 1% of the true period of the pulsar.

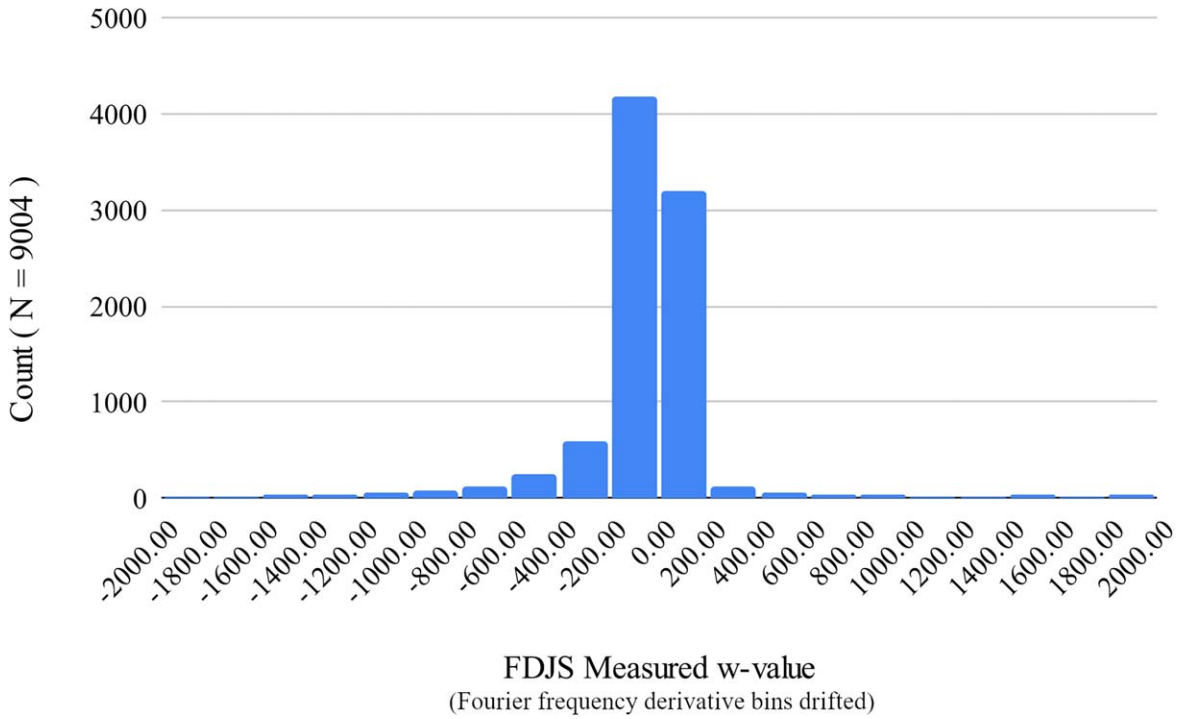


Figure 15. Histogram reporting the distribution of w-values measured by FDJS from PRESTO of the entire data set where a detection was made within 1% of the true period of the pulsar.

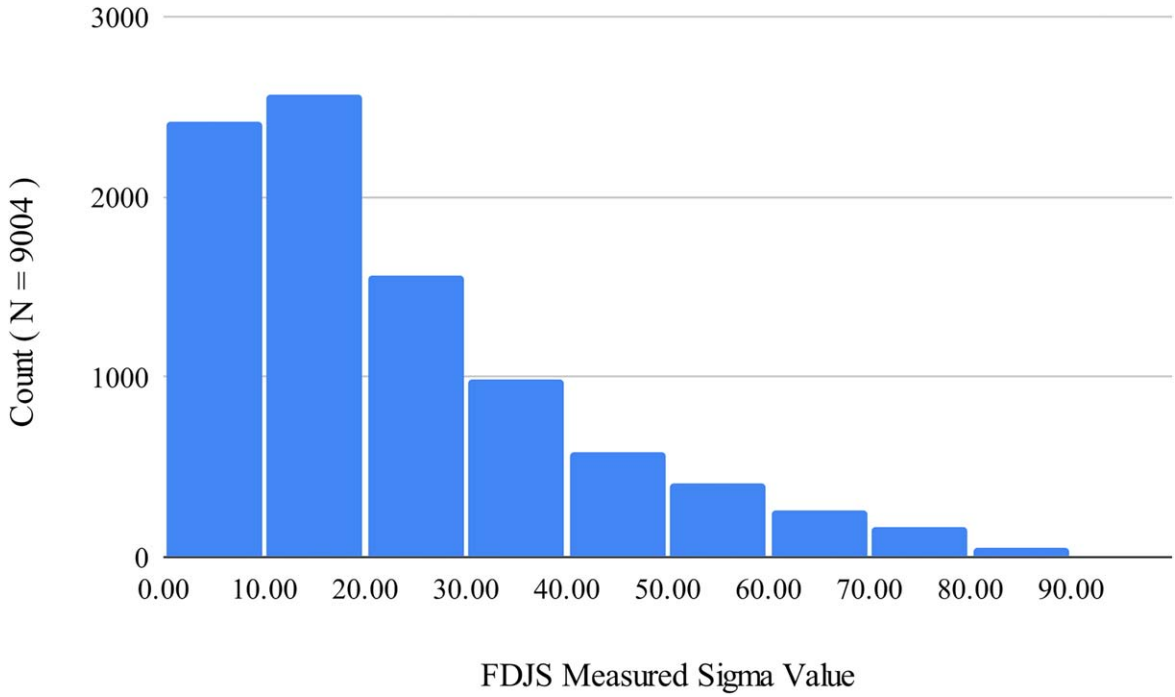


Figure 16. Histogram reporting the distribution of sigma values measured by FDJS from PRESTO of the entire data set where a detection was made within 1% of the true period of the pulsar.

Appendix B
Algorithm Pseudocode

Appendix B is provided to add detail on the exact structure of our implementations of the algorithms relevant to Pulscan on the GPU.

B.1. GPU Median + MAD Normalization

Algorithm 1: Median of Medians Normalization

Device function `local median(a, b, c, d)`:
  $min \leftarrow \min(\min(\min(a, b), c), d)$
  $max \leftarrow \max(\max(\max(a, b), c), d)$
  $median \leftarrow (a + b + c + d - min - max)/2$
  return median
Global function `median of medians (data array)`:
  shared *sdata_array*[4 × blockDim]
  // load sdata_array from data_array
  // then reduce over sdata_array
  for $strides \leftarrow blockDim, blockDim/4, \ldots, 1$ **do**
    if *threadIdx* < *stride* **then**
      $a \leftarrow sdata$[threadIdx]
      $b \leftarrow sdata$[threadIdx + *stride*]
      $c \leftarrow sdata$[threadIdx + 2 × *stride*]
      $d \leftarrow sdata$[threadIdx + 3 × *stride*]
      $sdata$[threadIdx] ← local_median(a, b, c, d)
  return *sdata*[0]
Procedure:
  foreach *chunkofspectrum* **do**
    $median \leftarrow$ median_of_medians(*chunk*)
    $MAD \leftarrow$ median_of_medians($chunk - median$)
    $scale_factor \leftarrow 1.4826$
    $normalised_chunk \leftarrow (chunk - median)/(scale_factor \times MAD)$

B.2. GPU Harmonic Sum Decimation

Algorithm 2: Harmonic Sum Decimation Process

Global function *decimation*
  (*data_len*, *in_data_array*, *dec_by_2_array*, *dec_by_3_array*, *dec_by_4_array*):
  float $1a$, $2a$, $2b$, $3a$, $3b$, $3\,c$, $4a$, $4b$, $4\,c$, $4d$
  $1a \leftarrow in_data_array$[threadIdx]
  $2a \leftarrow in_data_array$[2 × threadIdx]
  $2b \leftarrow in_data_array$[2 × threadIdx + 1]
  //Assignments for 3a, 3b, 3 c, 4a, 4b, 4 c, 4d would be similar
  if *threadIdx* × 2 + 1 < data_len **then**
    $dec_by_2_array$[threadIdx] ← $1a + 2a + 2b$
  if *threadIdx* × 3 + 2 < data_len **then**
    $dec_by_3_array$[threadIdx] ← $1a + 2a + 2b + 3a + 3b + 3c$
  if *threadIdx* × 4 + 3 < data_len **then**
    dec_by_4_array
  [threadIdx] ← $1a + 2a + 2b + 3a + 3b + 3\,c + 4a + 4b + 4\,c + 4d$

B.3. GPU Significance Calculation

Algorithm 3: Power to Log Probability Calculation

Device function `power_to_logp`(*pow*, *dof*):
  if $dof \geqslant pow \times 1.05$ **then**
    $logp \leftarrow 0.0$
  else
    $x \leftarrow 1500 \times dof/pow$
    $f(x) \leftarrow -4.4604059 \times 10^{-46} \times x^{16} + 9.4927863 \times 10^{-42} \times x^{15}$
      $-9.1470451 \times 10^{-38} \times x^{14} + 5.2810851 \times 10^{-34} \times x^{13}$
      $-2.0376166 \times 10^{-30} \times x^{12} + 5.5480334 \times 10^{-27} \times x^{11}$

(Continued)

$-1.0973877 \times 10^{-23} \times x^{10} + 1.5991804 \times 10^{-20} \times x^{9}$
$-1.7231488 \times 10^{-17} \times x^{8} + 1.3660070 \times 10^{-14} \times x^{7}$
$-7.8617952 \times 10^{-12} \times x^{6} + 3.2136336 \times 10^{-9} \times x^{5}$
$-9.0466418 \times 10^{-7} \times x^{4} + 0.00016945948 \times x^{3}$
$-0.021494231 \times x^{2} + 2.9515954 \times x - 755.24091$
logp ← *pow* × $f(x)/1500$
return *logp*

B.4. GPU Boxcar Filter

Algorithm 4: Boxcar Filtering Process

Global function *boxcar_filter*(*data_array*, *chunk_size*, *z_max*, *z_step*):
dynamic shared *lookup_array*[chunk_size + *z_max*]
dynamic shared *sum_array*[chunk_size]
dynamic shared [*float*, *int*]*search_reduce_array*[blockDim]
numIterationsToCoverChunk ← (chunk_size + blockDim − 1)/blockDim
numIterationsToFillLookup ← (chunk_size + *z_max* + blockDim − 1)/blockDim
// Fill lookup_array with data
for *i* ← 0 **to** *numIterationsToFillLookup* **do**
local_index ← threadIdx + *i* × blockDim
global_index ← blockIdx × chunk_size + *local_index*
if *local_index* < chunk_size + *z_max* **then**
lookup_array[*local_index*] ← *data_array*[*global_index*]
// Set sum_array to 0
for *i* ← 0 **to** *numIterationsToCoverChunk* **do**
local_index ← threadIdx + *i* × blockDim
if *local_index* < chunk_size **then**
sum_array[*local_index*] ← 0
// Search array using boxcar filters
for *z* ← 0 **to** *z_max* **do**
// Apply the next boxcar filter
for *i* ← 0 **to** *numIterationsToCoverChunk* **do**
local_index ← threadIdx + *i* × blockDim
if *local_index* < *chunk_size* **then**
sum_array[*local_index*] += *lookup_array*[*local_index* + *z*]
// This condition on z can be replaced with a check for if the current z is a power of 2 for logarithmic sampling
if remainder(*z*/*z_step*) = 0 **then**
// Scan across the array for max
local_max_power ← 0
local_max_index ← 0
for *i* ← 0 **to** *numIterationsToCoverChunk* **do**

(Continued)

local_index ← threadIdx + *i* × blockDim
if *sum_array*[*local_index*] > *local_max_power* **then**
local_max_power ← *sum_array*[*local_index*]
local_max_index ← chunk_size × blockIdx + *local_index*
search_reduce_array[threadIdx]. *power* ← *local_max_power*
search_reduce_array[threadIdx]. *index* ← *local_max_index*
// Followed by block wide max reduction
for *stride* ← blockDim/2, blockDim/4, …, 1 **do**
if threadIdx < *stride* **then**
if *search_reduce_array*[threadIdx + *stride*]. *power* > *search_reduce_array*[threadIdx]. *power* **then**
search_reduce_array[threadIdx] ← *search_reduce_array*[threadIdx + *stride*]
if threadIdx = 0 **then**
// Write out local candidate to global candidate array

ORCID iDs

Jack White https://orcid.org/0000-0003-2690-6858
Karel Adámek https://orcid.org/0000-0003-2797-0595
Jayanta Roy https://orcid.org/0000-0002-2892-8025
Scott M. Ransom https://orcid.org/0000-0001-5799-9714
Wesley Armour https://orcid.org/0000-0003-1756-3064

References

Adámek, K., & Armour, W. 2020, ApJS, 247, 56
Adámek, K., Novotný, J., Dimoudi, S., & Armour, W. 2020, in ASP Conf. Ser. 527, Astronomical Data Analysis Software and Systems XXIX, ed. R. Pizzo et al. (San Francisco, CA: ASP), 671
Andersen, B. C., & Ransom, S. M. 2018, ApJL, 863, L13
Dimoudi, S., Adamek, K., Thiagaraj, P., et al. 2018, ApJS, 239, 28
Ford, J. M., & Buch, K. D. 2014, in 2014 IEEE Geoscience and Remote Sensing Symp. (Piscataway, NJ: IEEE), 231
Pan, Z., Lu, J., Jiang, P., et al. 2023, Natur, 620, 961
Ransom, S. M. 2001, PhD thesis, Harvard Univ.
Ransom, S. M., Cordes, J. M., & Eikenberry, S. S. 2003, ApJ, 589, 911
Richards, A. 2015, University of Oxford Advanced Research Computing, v1, Zenodo, doi:10.5281/zenodo.22558
Shan, H. 2023, ApJ, 952, 70
Shan, H., Yuan, J., Wang, N., & Wang, Z. 2022, ApJ, 935, 117
White, J., 2025 jack-white1/pulscan: Initial Release, Zenodo, doi:10.5281/zenodo.15151296
Zhang, X., Cognard, I., & Dobigeon, N. 2024, A&C, 47, 100822