The Astrophysical Journal Supplement Series, 247:56 (26pp), 2020 April https://doi.org/10.3847/1538-4365/ab7994

Single-pulse Detection Algorithms for Real-time Fast Radio Burst Searches Using GPUs

Karel Adámek and Wesley Armour
Oxford e-Research Centre, Department of Engineering Science, University of Oxford, 7 Keble Road, Oxford OX1 3QG, UK; karel.adamek@oerc.ox.ac.uk, wes.armour@oerc.ox.ac.uk

Received 2019 October 17; revised 2020 February 19; accepted 2020 February 23; published 2020 April 1

Abstract

The detection of non-repeating or irregular events in time-domain radio astronomy has gained importance over the last decade due to the discovery of fast radio bursts. Existing or upcoming radio telescopes are gathering more and more data, and consequently, the software, which is an important part of these telescopes, must process large data volumes at high data rates. Data has to be searched through to detect new and interesting events, often in real time. These requirements necessitate new and fast algorithms that must process data quickly and accurately. In this work, we present new algorithms for single-pulse detection using boxcar filters. We have quantified the signal loss introduced by single-pulse-detection algorithms, which use boxcar filters, and based on these results, we have designed two distinct "lossy" algorithms. Our lossy algorithms use an incomplete set of boxcar filters to accelerate detection at the expense of a small reduction in detected signal power. We present formulae for signal loss, descriptions of our algorithms and their parallel implementation on NVIDIA GPUs using CUDA. We also present tests of correctness, tests on artificial data, and the performance achieved. Our implementation can process SKA-MID-like data 266× faster than real time on an NVIDIA P100 GPU and 500× faster than real time on an NVIDIA Titan V GPU with a mean signal power loss of 7%. We conclude with prospects for single-pulse detection for beyond the SKA era, nanosecond time-resolution radio astronomy.

Unified Astronomy Thesaurus concepts: Transient detection (1957); Time series analysis (1916); Astronomy data reduction (1861); Astronomy data analysis (1858); Computational astronomy (293); Astronomy software (1855); Computational methods (1965); GPU computing (1969); Algorithms (1883)

1. Introduction

The discovery of Fast Radio Bursts (FRBs) by Lorimer et al. (2007) and Rotating Radio Transients (RRATs) by McLaughlin et al. (2006) has highlighted the importance of single-pulse detection in time-domain radio astronomy. FRBs and RRATs are rare, non-repeating or irregular events; therefore, their accurate detection is of great importance if we are to understand the nature of the objects producing them. Furthermore, on modern radio telescopes, the algorithms for single-pulse detection are required to process more and more data collected by these instruments. Also, the rise of multiwavelength astronomy, which requires real time or near-real-time (Middleton et al. 2017) detections, increases the requirements put on single-pulse detection even more. All of this necessitates the need for fast and accurate algorithms designed to detect single isolated pulses. Single-pulse-detection algorithms are of importance in searches for giant pulses or irregular pulses from nulling pulsars also.

The single-pulse search was described by Cordes & McLaughlin (2003) as an exercise in matched filtering. They employed a set of boxcar filters of widths 2^n. This technique was used successfully in subsequent works (for example, Cordes et al. 2006; Deneva et al. 2009; Burke-Spolaor & Bailes 2010; Rubio-Herrera et al. 2013). The boxcar filter approach was also adopted by many software packages, like `Heimdall`,[1] `Seek`,[2] `Destroy`.[3] In some cases, it is beneficial to use more targeted matched filters as suggested by Keane et al. (2010).

The single-pulse-detection problem is also being explored using machine-learning and deep-learning techniques, for example, Wagstaff et al. (2016), Zhang et al. (2018), Connor & van Leeuwen (2018). Comparison of these techniques to traditional searches is, however, outside the scope of this work.

It is becoming increasingly important to understand the associated sensitivity and signal loss introduced by algorithms and software when considering the overall performance of a radio telescope. As such, any sensitivity loss introduced by algorithms or software must be investigated. This was discussed by Keane & Petroff (2015), where the authors compared the sensitivity of different single-pulse-detection algorithms. Single-pulse-detection algorithms can have many different forms, all with different sensitivities and also different computational costs. In the case of single-pulse detection by boxcar filters, the decrease in sensitivity can occur when a detected pulse is poorly matched by the filters used to extract it from the noise in which it is embedded.

In this work, we quantify sources of signal loss that occur when single pulses are detected using boxcar filters. Based on these, we have designed two distinct methods to distribute and create a limited set of boxcar filter widths in such a way that the signal loss is controlled. We show that the methods presented significantly increase the computational efficiency of traditional approaches and, hence, provide a significant reduction in execution time, which is of relevance to real-time processing pipelines. We present formulae for calculating the signal loss introduced by a given set and distribution of boxcar filters for an idealized model that uses rectangular pulses. These formulae allow the user to tune the signal loss of their single-pulse detection scheme. We present calculations of the computational complexity and the number of memory accesses associated with each method.

Based on our two methods, we have designed and implemented two single-pulse-detection algorithms for NVIDIA GPUs. These

---

[1] Jameson & Barsdell (2018).

[2] https://github.com/SixByNine/psrsoft

[3] https://github.com/evanocathain/destroy_gutted

algorithms rely on reusing partial sums to calculate long boxcar filters required for detection of wide single pulses. This increases computational efficiency and decreases the number of memory accesses. We present optimization techniques used in the implementation of these algorithms on GPUs. We also describe the advantages and disadvantages of these algorithms. Reusing partial sums to increase computational efficiency is well known in the radio-astronomy community. It has been successfully used in the fast folding algorithm by Staelin (1969), with the most recent implementation by Parent et al. (2018; using python). The reuse of partial sums was also successfully used in the tree de-dispersion algorithm by Zackay & Ofek (2017), who implemented tree de-dispersion using CPU but also suggested an algorithm suitable for GPUs.

This work is part of the AstroAccelerate software package Armour et al. (2019), a GPU optimized time-domain processing pipeline for radio-astronomy data. It contains GPU implementations of common signal processing tools used in time-domain radio astronomy, such as de-dispersion by Armour et al. (2012), a GPU implementation of the Fourier domain acceleration search by Dimoudi et al. (2018), and also periodicity search with a GPU implementation of the harmonic sum by Adámek & Armour (2019). Aspects of the work have been used in Karastergiou et al. (2015) and Mickaliger et al. (2018).

This paper is structured as follows. First, in Section 2, we analyze the sensitivity of the single-pulse-detection method that uses boxcar filters, and we derive formulae for quantifying the signal loss. In Section 3, we present two distinct algorithms and describe their implementation using the CUDA language extension on NVIDIA GPUs. Our results are presented in Section 4, where we present the sensitivity of our algorithm when applied to rectangular, Gaussian, and double-Gaussian pulse profiles with and without white noise. We prove the correctness of our implementation by comparing the measured signal loss with predicted values. We also present the performance of both algorithms and compare it to Heimdall, a GPU accelerated pipeline by Jameson & Barsdell (2018) and Barsdell et al. (2012). We conclude the paper in Section 5.

2. Single-pulse Detection

The aim of the single-pulse search, which represents a whole set of techniques and methods, is to find isolated pulses in input data. In this work, we focus on the step responsible for recovering the pulse, located somewhere in the input time-series. We will refer to this step as *single-pulse detection* or the *SPD* algorithm.

Single-pulse detection, in general, relies on a match filtering process, which is a convolution of the input time-series x with a response function h. Convolution in the time domain is given by

$$y[n] = \sum_{i=0}^{T-1} h[i]x[n-i] \tag{1}$$

where n is the time sample, y is the filtered time-series, and T is the length of the response function. The response function is typically designed to detect pulses of a certain or similar shape of width T. Significantly longer pulses of the same shape or of different shape require a different response function h. A matched filtering approach thus requires multiple passes through the data in order to cover the desired range of widths and shapes, and the convolution is computationally expensive and offers little opportunity to reuse data.[4]

Because of the computational expense of matched filtering, we have decided to use boxcar filters for our single-pulse-detection algorithm. If we assume that the pulse we would like to detect could be located anywhere within the time-series, be of any shape, and have a wide range of widths, the boxcar filter is a viable alternative. Boxcar filters are less sensitive than matched filters but offer two important advantages over the matched filter approach. They are independent of the pulse shape, and they allow us to reuse data and computations, which is critical when producing an algorithm for execution on modern computer architectures, especially accelerator architectures, such as GPUs.

The boxcar filter is a simple running sum, which can be expressed as

$$X_L[n] = \sum_{j=0}^{L-1} x[n+j], \tag{2}$$

where L is the boxcar width. This formalism allows us to reuse partial sums because we can form a new longer boxcar filter from an appropriate combination of shorter boxcar filters.

We can measure the strength of a sample by calculating the signal-to-noise ratio (S/N). The S/N of the nth sample from a time-series is calculated by the formula

$$\mathrm{S/N}[n] = \frac{x[n] - \mu(x)}{\sigma(x)}, \tag{3}$$

where $\mu(x)$ and $\sigma(x)$ are the mean and standard deviation of the underling noise in the initial time-series x, respectively. By applying a boxcar filter to a time-series, we are creating a new time-series with different mean $\mu(X_\mathrm{L})$ and standard deviation $\sigma(X_\mathrm{L})$. The S/N for a pulse from this time-series is then calculated as

$$\mathrm{S/N_L}[n] = \frac{X_\mathrm{L}[n] - \mu(X_\mathrm{L})}{\sigma(X_\mathrm{L})}, \tag{4}$$

where $X_L[n]$ is given by Equation (2). The value $X_L[n]$ of the new time-series is a value of the boxcar filter and the bin width, or put another way, the number of the accumulated samples from the initial time-series x is equal to the boxcar width.

We have chosen to adopt the S/N of the pulse as a figure of merit by which we measure how well any SPD algorithm detects individual pulses.

The S/N produced by the algorithm that we call *recovered S/N* (RS/N), to distinguish it from the true S/N of the pulse, is calculated using Equation (4). The value of the RS/N depends not only on the initial S/N of the injected pulse and its shape, but also on its position within the time-series and its width. This is because to localize the pulse and sum the power contained within it, we use an incomplete set of boxcar filters. That is, the boxcar filters do not cover every possible pulse width at every possible time sample. As a consequence, a pulse of the same shape could be detected with different RS/N based on its position in time and its width. A lower RS/N occurs when the injected pulse is not properly matched by these filters either in width or position in time. The highest RS/N will be

---

[4] This is because partial sums from which we can construct the output sample $y[n]$ cannot be used to construct sample $y[n+i]$, since the elements of x within these sums are weighted by the matched filter h.

produced by the boxcar with the width and position that best fit the unknown signal.

2.1. Idealized Signal Model

In order to compare algorithms, to evaluate sensitivity loss and to simplify sensitivity analysis, we have introduced an idealized model (simplified toy model) of the input signal.

The simplest form of signal that fits this role is the signal with a rectangular pulse, without any noise, where all samples except those of the pulse are set to zero. The pulse is described by its position t_s within the time-series, by its width S, and by its amplitude A. The mean (μ) and standard deviation (σ) that are required in the calculation of the S/N (Equation (3)) are set to $\mu(x) = 0$ and $\sigma(x) = 1$ to simulate the presence of white noise.

To get the value of the mean $\mu(X_\mathrm{L})$ and standard deviation $\sigma(X_\mathrm{L})$ for longer boxcar filters, we use the white-noise approximation. That is, the mean and standard deviation for time-series after application of the boxcar filter of width L are given as

$$\begin{aligned}\mu(X_\mathrm{L}) &= L\mu(x),\\ \sigma(X_\mathrm{L}) &= \sqrt{L}\,\sigma(x).\end{aligned} \tag{5}$$

For the white-noise approximation, we get $\mu(X_\mathrm{L}) = 0$ and $\sigma(X_\mathrm{L}) = \sqrt{L}$.

For the purpose of comparing the sensitivity of the SPD algorithm to different pulse widths, we normalize the amplitude (A) of the rectangular pulse to

$$A(S) = C/\sqrt{S}, \tag{6}$$

where C is a normalization constant. The normalization ensures that the RS/N produced by the boxcar filter that fits the rectangular pulse perfectly is the same regardless of the pulse width S.

2.2. Sensitivity Analysis

We quantify the sensitivity of the SPD algorithm in terms of signal loss. The *signal loss* Ψ is given as the fraction of the pulse's true S/N ($\mathrm{S/N_T}$), which was not detected by the algorithm. That is, the difference between RS/N detected by the algorithm and the true value $\mathrm{S/N_T}$ of the injected pulse divided by $\mathrm{S/N_T}$,

$$\Psi = \frac{\mathrm{S/N_T} - \mathrm{RS/N}}{\mathrm{S/N_T}} = 1 - \frac{\mathrm{RS/N}(S, L, d_s)}{\mathrm{S/N_T}}, \tag{7}$$

where the value of RS/N depends on the pulse width, the width of the boxcar filter used for detection, and on the position of the boxcar filter with respect to the pulse's position.

As the position of the signal is unknown, we must apply the boxcar filter of width L to the whole time-series x, which may or may not be applied to each and every time sample. If we are only interested in the highest RS/N detected, for fixed boxcar width L, then this reduces the sensitivity analysis to a problem of two consecutive boxcar filters separated by L_s time samples; we call this distance the *boxcar separation*. This problem is depicted graphically in Figure 1 and is then repeated throughout the whole time-series x. This allows us to avoid studying individual boxcar filters but still provides enough flexibility to evaluate sensitivity or signal loss introduced by the SPD algorithm for any signal present in the time-series.

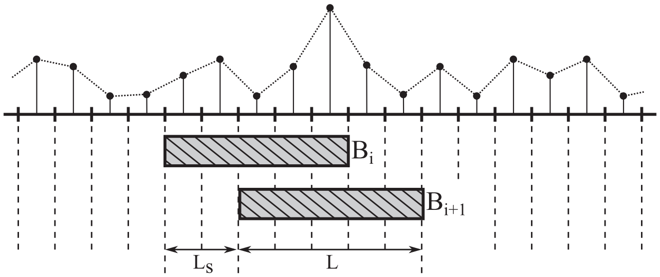

Figure 1. Layout of the boxcar filters covering a time-series x. Boxcars (B_i, B_{i+1}) are shown as hatched boxes. Depending on the value of L_s (here $L_s = 2$), multiple boxcars can intersect with themselves. Only two are shown here.

In our analysis of the sensitivity of SPD algorithms (for a detailed description, see the Appendix), we have, using the idealized signal model, identified two quantities of the RS/N, which can be used to describe the sensitivity of an SPD algorithm.

The first is the lowest RS/N detected $\mathrm{RS/N_{min}}(S)$ by the SPD algorithm under the worst conditions given the pulse width S. This acts as an infimum (greatest lower bound) for RS/N detected by the SPD algorithm for any pulse of a given width S. Along with it, we define the *worst-case-scenario signal loss* as

$$\Psi_\mathrm{w}(S) = 1 - \mathrm{RS/N_{min}}(S)/\mathrm{S/N_T}, \tag{8}$$

which is the greatest signal loss introduced by the SPD algorithm.

The second quantity is the highest possible value of RS/N ($\mathrm{RS/N_{max}}(S)$), which could be recovered by the SPD algorithm under the best possible circumstances for the given pulse width S. This acts as a supremum (least upper bound) for RS/N, which could be detected by the SPD algorithm. To accompany this quantity, we have defined the *systematic signal loss*

$$\Psi_\mathrm{l}(S) = 1 - \mathrm{RS/N_{max}}(S)/\mathrm{S/N_T}, \tag{9}$$

which is always introduced by the SPD algorithm for a pulse of width S.

2.2.1. Parameters of the SPD Algorithm

We have also determined two parameters that describe the SPD algorithm and have direct implication on the sensitivity. We characterize the SPD algorithm by the set of boxcar filter widths $\mathfrak{B} = \{L_1, L_2, \ldots, L_\mathrm{max}\}$ and by boxcar separation L_s for each boxcar width $L \in \mathfrak{B}$.

The number of different boxcar widths, which is the sparsity of the set $\mathfrak{B}$, affects the value of $\mathrm{RS/N_{max}}$. We can increase $\mathrm{RS/N_{max}}$ (decrease the systematic signal loss Ψ_l) by including more boxcar filter widths performed by the SPD algorithm. The set $\mathfrak{B}$, to a lesser degree, also affects $\mathrm{RS/N_{min}}$.

The value of the boxcar separation L_s is the most important for the value of $\mathrm{RS/N_{min}}$. In order to increase $\mathrm{RS/N_{min}}$ (decrease the worst-case signal loss Ψ_w), we have to decrease the boxcar separation L_s.

The SPD algorithm may consist of multiple iterations where both $\mathfrak{B}$ and L_s may differ. We assume that the time-series in which we want to detect pulses is completely covered by boxcar filters of a given width L.

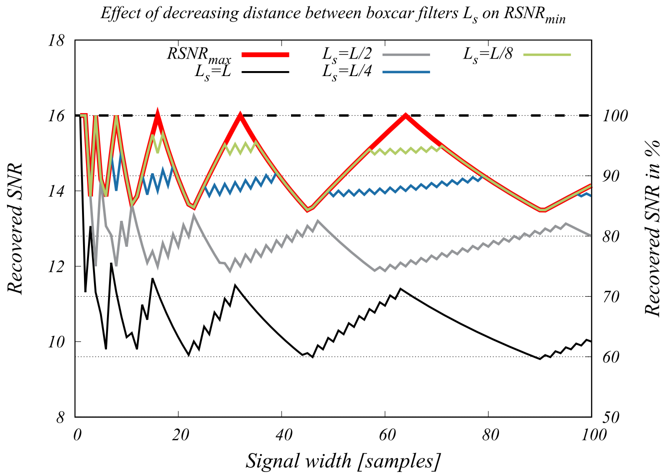

Figure 2. Behavior of $\mathrm{RS/N_{min}}(S)$ using different boxcar separation L_s. This shows that $\mathrm{RS/N_{min}}(S)$ is limited by $\mathrm{RS/N_{max}}(S)$. In order to increase $\mathrm{RS/N_{min}}(S)$ further, thus increasing the sensitivity of a SPD algorithm, we have to increase $\mathrm{RS/N_{max}}(S)$ first.

2.2.2. Importance of Maximum and Minimum RS/N

There are several reasons why $\mathrm{RS/N_{max\,/\,min}}(S)$ is important. First, from supremum and infimum properties of the $\mathrm{RS/N_{max\,/\,min}}$, we get $\mathrm{RS/N_{max}}(S) \geqslant \mathrm{RS/N_{min}}(S)$. That is, we cannot increase $\mathrm{RS/N_{min}}(S)$ above $\mathrm{RS/N_{max}}(S)$. Since $\mathrm{RS/N_{min}}(S)$ is mainly improved by decreasing boxcar separation L_s, this means that decreasing L_s may yield diminishing results. This is shown in Figure 2.

Second, the $\mathrm{RS/N_{max\,/\,min}}(S)$ gives a hint at how we can increase or decrease the sensitivity of an algorithm but also how to trade sensitivity for computational performance.

Lastly, by knowing $\mathrm{RS/N_{max\,/\,min}}(S)$ for a given SPD algorithm, we can compare them and rank them. It can also serve as a verification tool to check if an implementation of a given algorithm works as expected. In this case, the algorithm must not produce any RS/N value that would lie outside the limits given by $\mathrm{RS/N_{max\,/\,min}}(S)$.

2.2.3. Error in Detected Pulse Width and Time

The error in the detected width of the pulse for the SPD algorithm can be expressed as

$$\epsilon_{\mathrm{W}} = \frac{|S_{\mathrm{T}} - L|}{S_{\mathrm{T}}} \tag{10}$$

where S_{T} is the true pulse width and L is the boxcar width that has detected the pulse, i.e., the detected width.

Similarly, we can define the error in detected position in time relative to its true position (time localization) as a fraction of its true width S_{T} as

$$\epsilon_{\mathrm{t}} = \frac{|t_{\mathrm{T}} - t|}{S_{\mathrm{T}}} \tag{11}$$

where t_{T} is the true pulse position in time, and t is the boxcar time position.

3. Single-pulse-detection Algorithms

The single-pulse-detection algorithm may be required to scan for very long pulse widths S_{end} with related maximum boxcar filter width L_{end}. This could lead to unnecessary precision and longer execution time, as more boxcar filters than is necessary would be calculated. This is why both of our proposed SPD algorithms execute in multiple iterations with different parameters and inputs. This allows us to control precision and to increase performance. In order to decrease the sensitivity of the SPD algorithm and to lower the amount of data that are processed, we use decimation in time (by a factor D).

To distinguish quantities from different iterations, we decorate them with an index i, starting with $i = 0$. We assume, for simplicity, that the decimation factor D does not change during the execution of the algorithm, and $D^{(i)}$ means D to the power of i and not an iteration index. Thus, we have:

1. x^i input time-series for given iteration ($x^0 = x$ is the initial time-series)
2. $\mathfrak{B}^i = \{L_1^i, L_2^i, \ldots, L_{\mathrm{m}}^i, \ldots, L_{\max}^i\}$ a set of boxcar widths used in ith iteration
3. $X_L^i[n]$ is the value of the boxcar filter for width L at time sample n
4. $X^i[n] = X_{L_{\max}}^i[n]$ is the value of the boxcar filter at the end of iteration
5. S_{end} is the maximum desired pulse width to be searched for
6. L_{end} is the maximum boxcar width calculated by the SPD algorithm; L_{end} is the boxcar width of the nearest longer boxcar filter to the value of S_{end}.

The output of the SPD algorithm is the highest RS/N value detected by the boxcar filters. We have chosen to separate the output by iteration; that is, the output for ith iteration is

$$Y^i[n] = \max_{L \in \mathfrak{B}^i} (\mathrm{S/N}(X_L^i[n])), \tag{12}$$

where the S/N value is calculated using Equation (4). In addition to the S/N value, we assume that the SPD algorithm provides the width of the boxcar, which produced the highest S/N value W^i.

When designing an algorithm that is suitable for execution on GPUs, it is important to have enough parallelism to ensure high utilization of the GPU hardware. Modern GPUs can process many thousands of threads concurrently. The memory available for a single thread on the GPU is extremely limited, either in the form of registers or the amount of shared memory available. Therefore, cooperation between threads within a single threadblock[5] is important. The amount of resources (for example, the amount of local memory) consumed by a threadblock and how many threadblocks can be executed concurrently then depends on the number of threads per threadblock.

3.1. BoxDIT Algorithm

The algorithm we call BoxDIT is based on the ideal SPD algorithm. The ideal SPD algorithm is the algorithm that performs boxcar filters of all widths up to a maximum $L_{\max}$ at each element of the initial time-series and, thus, detects any rectangular pulse with $S < L_{\max}$ with its true S/N value. The ideal SPD algorithm has poor performance due to unnecessary high sensitivity. The BoxDIT algorithm trades some sensitivity for performance by introducing the decimation step, which

---

[5] Threadblock is a set of threads that can cooperate with each other.

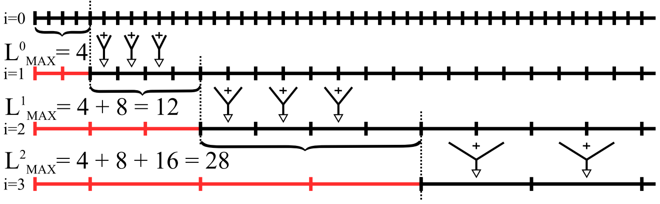

Figure 3. Decimations and maximum widths calculated at each iteration of the BoxDIT algorithm. Here, $L^i_{\text{max}} = 4$ and $D = 2$. In the red, we see parts of the time-series that are omitted by the decimations in time, i.e., total shift.

reduces the number of boxcar filter widths performed and increases boxcar separation L_s.

Algorithm 1: Pseudo-code for the BoxDIT algorithm. The algorithm performs I iterations required to reach desired boxcar width L_{end}. The quantity X^i represents values of boxcars at the end of the boxcar filter step. These values are used to built up longer boxcar filters in higher iterations. Function `Boxcars()` calculates boxcar filters up to L^i_{max} and function `Decimate()` performs decimation in time.

```
Input: $x^0$;
Output: $Y$, $W$;
$L_s^0 = 1$;
$L^0_{\text{MAX}} = 0$;
for $i = 0$ to $I$ do
    Boxcar separation for this iteration;
    $L_s^i = L_s^{i-1} D = D^{(\text{i})}$;
    Calculating boxcar filters on decimated data $x^i$;
    Boxcar ($Y^i[n]$, $W^i[n]$, $X^{i+1}$, $x^i$, $X^i$, $L^i_{\text{max}}$);
    Decimating data for next iteration;
    Decimate ($x^{i+1}$, $x^i$);
    Width of the longest boxcar calculated in this iteration;
    $L^i_{\text{MAX}} = L^{i-1}_{\text{MAX}} + D^{(\text{i})} L^i_{\text{max}}$;
end
```

The BoxDIT algorithm (Algorithm 1) is a sequence of iterations, where each iteration has two steps. The first step is to perform all boxcar filters up to L^i_{max} with respect to time-series x^i, at every point of the input data x^i. From the perspective of the initial time-series x^0, the algorithm calculates boxcar filters with a width step $D^{(i)}$ at every $D^{(i)}$ point of the initial time-series. For example, if $L^i_{\text{max}} = 4$ for all i, then: boxcar filters of width $L = 1,\ 2,\ 3,\ 4$ are calculated at every point; after decimation, boxcar filters of width $L = 6,\ 8,\ 10,\ 12$ are calculated at every second point, and so on. Thus, each subsequent iteration detects wider pulses in the input data. This step uses the previously calculated boxcar filter data to calculate longer boxcar filters.

The second step is to decimate the time-series x^i in time by a factor D to create time-series x^{i+1} for the next iteration. This is repeated until L_{end} is reached. The decimation in time that is used in this paper is given by

$$x^{i+1}[n - L^i_{\text{max}}/D] = \sum_{j=0}^{D-1} x^i[Dn + j], \tag{13}$$

for all $n - L^i_{\text{max}}/D \geqslant 0$. That is, the zeroth element of the decimated time-series is set to a position of the next time sample, which must be added to the partial sum containing the zeroth element of the initial time-series x^0. This is shown in Figure 3. Such a decimation has the advantage that the maximum width L^i_{max} performed at any iteration of the BoxDIT algorithm has to fulfill only the condition that it is divisible by D. During the decimation step, we also reduce the number of time samples to $N_i = N_{i-1}/D = N_0/D^{(i)}$; this is important, as the higher iterations work with fewer points. Thus, each BoxDIT iteration is characterized only by L^i_{max} and D.

The width of the boxcar filter calculated by iteration i is given as

$$L^i_{\text{M}} = \sum_{k=0}^{i-1} D^{(k)} L^k_{\text{max}} + D^{(i)} L^i_{\text{m}}. \tag{14}$$

This could be also used to calculate the maximum boxcar width L^i_{MAX} for given iteration i.

The value of the boxcar filter at time sample n for iteration i is given as

$$X_L[n] = X^{i-1}[n] + \sum_{j=0}^{L^i_{\text{m}}-1} x^i[n^i_{\text{s}} + j], \tag{15}$$

where $n^i_{\text{s}} = n/D^{(i)}$.

The advantage of the BoxDIT algorithm is that the sensitivity of the algorithm is easily adjusted by changing the maximum boxcar width L^i_{max} calculated by the boxcar step of the algorithm. The disadvantage of the BoxDIT algorithm is that it has higher memory bandwidth requirements since, in addition to the decimated input data, it also needs the values of the longest boxcar filter at every point, which, in effect, doubles the size of the input data.

3.1.1. BoxDIT GPU Implementation

The GPU implementation of the BoxDIT algorithm performs both steps, calculation of boxcar filters, and decimation, in one GPU kernel. The implementation must be able to extend the already calculated boxcar filters by using values from the previous iteration. Such an implementation can be used for any iteration of the BoxDIT algorithm without change.

On the input, we have the time-series x^i, values for the longest boxcar filter from the previous iteration X^{i-1}, and maximum boxcar width L^i_{max} calculated in this iteration. On the output, we expect to have the highest RS/N Y^i for every point of the input time-series, associated boxcar width W^i, values of the longest boxcar calculated X^i and decimated time-series for next iteration x^{i+1}. We have used the decimation factor $D = 2$ for the GPU implementation of BoxDIT.

Calculating all boxcars up to given maximum width L^i_{max} at a given point is equivalent to performing a prefix sum or a scan. In this case, the scan has to be performed at every point of the input time-series. We have found that using the standard prefix sum algorithm like the Hillis–Steel scan (Hillis & Steele 1986) to be slow even when used on cached data. The reason is that series independent scan operations do not cooperate by reusing data enough. Our implementation focuses on increased cooperation between independent prefixed sums within the GPU threadblock.

Algorithm 2: BoxDIT pseudo-code.

```
Input: $x^i$, $X^i$, $n$;
Output: $x^f$, $X^f$, $Y^f$, $Y^f_L$;
__Shared__float s_input [nThreads];
__Shared__float s_S/N [nThreads];
__Shared__int s_widths [nThreads];
```

(Continued)

Algorithm 2: BoxDIT pseudo-code.

```
float $B_w[w_m]$, S/N, temp;
int $w$, widths;
int t = threadIdx.x;
Calculate initial partial sums (boxcar filters) up to width $w_m$;
for $k = 0$ to $w_m$ do
    $B_w[k] = \sum_{j=0}^{k} x^i[n + t + j]$;
    Calculate S/N values corresponding to partial sums $B_w$;
    Calculate_S/N ($X^i[n + t] + B_w$, S/N, width);
    Compare S/N values and store highest S/N and its width;
    Compare_S/N (S/N, width, s_input, s_widths);
end
$B_w[w_m]$ is stored into shared memory;
s_input [t] = $B_w[w_m]$;
for $w = 4$to $B^i_{\max}$ do
    if $t - w \geqslant 0$ then
        We are changing S/N values at position $t - w$, i.e., values $B_w$ are
      used to build up higher boxcar widths at position $t - w$;
        temp = s_input [$t - w$];
        S/N = s_S/N [$t - w$];
        width = s_widths [$t - w$];
        for $k = 0$ to $w_m$ do
            $B_w[k] = B_w[k]$ + temp;
            Calculate_S/N ($X^i[n + t - w] + B_w$, S/N, width);
            Compare_S/N (S/N, width, s_input, s_widths);
        end
        s_S/N [$t - w$] = S/N;
        s_widths [$t - w$] = width;
    $w = w + 4$;
end
```

The pseudo-code for the parallel GPU implementation of the BoxDIT algorithm is given in Algorithm 2 and a graphical sketch is presented in Figure 4. This figure shows how we divide work among GPU threads and how threads cooperate on a calculation of independent prefixed sums.

As a first step, each thread calculates a much shorter prefixed sum of length w_m. That is, each thread calculates partial sums $B[n; w]$, where n is a time sample, and $w = 1, 2, \ldots, w_m$ is the length of the partial sum in the number of time samples. These partial sums $B[n, w]$ are stored into the GPU registers[6] of a given thread. These partial sums serve as accumulators used for the calculation of the longer partial sums. The value of the last partial sum $B[n; w_m]$ is also stored into shared memory,[7] because its value has to be distributed among other threads. The maximum S/N $Y^i[n]$ for a given time sample n and boxcar width $W^i[n]$ is also stored in the shared memory, as it will be modified by other threads.

After this initial phase, each thread reads the partial sum $B[n - w_m; w_m]$ from its appropriate place in shared memory. Then, using accumulators stored in its registers (partial sums $B[n; w]$, for $w = 1, 2, \ldots, w_m$), each thread is able to produce a prefixed sum up of length $w = 2w_m$. That is, partial sums $B[n; w]$, where $w = 1, 2, \ldots, 2w_m$. These partial sums are calculate as

$$B[n - w_m; w_m + w] = B[n - w_m; w_m] + B[n; w]. \qquad (16)$$

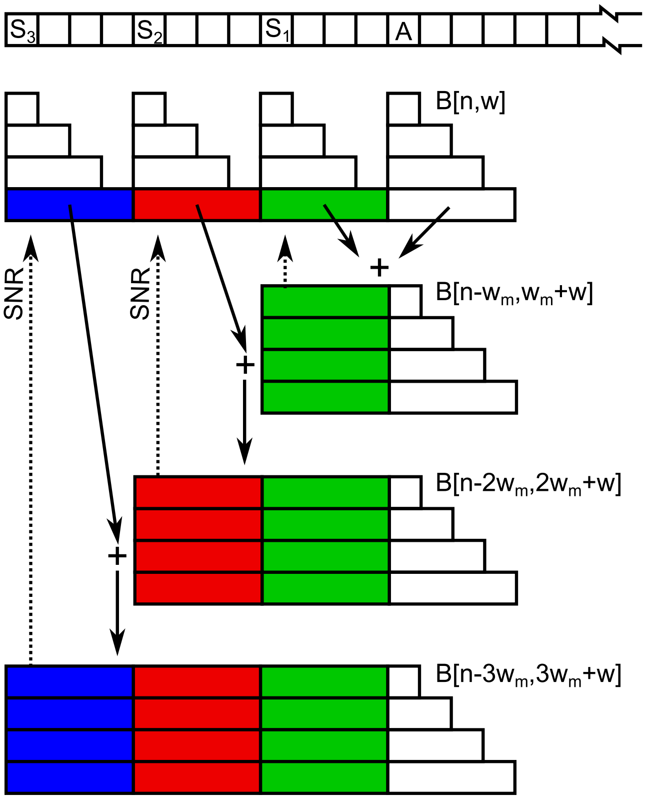

Figure 4. Graphical sketch of the BoxDIT algorithm. Each thread operates on a single sample, threads A, S_1, S_2, and S_3 are emphasized. At the beginning, each thread calculates a small scan of size w_m (here $w_m = 4$) and stores these values in the thread's registers. This is represented by rectangles of increasing size at below every fourth thread. Each thread then uses sums calculated by threads with smaller indices to calculate scan of the required size. The thread A uses the largest sums (w_m long) from threads $S_{1,2,3}$ to calculate prefixed sums of length 8, 12, and 16.

The accumulators in the threads registers are then updated to the new values representing partial sums $B[n - w_m; w_m + w]$. The thread calculates S/N value for each accumulator and compares it to the highest detected S/N for a sample $n - w_m$. If the new S/N is higher than the old S/N, the highest S/N $Y^i[n]$ is updated together with boxcar width $W^i[n]$.

With each following iteration, each thread calculates longer partial sums for time samples that are further from its starting time samples.

The number of iterations required to calculate all required boxcars is $J = L^i_{\max}/w_m$. The algorithm used in our implementation can be expressed by equation

$$X^i_{L^i_m}[n] = \sum_{k=0}^{j-1} B[n + kw_m; w_m] + B[n + jw_m; w], \qquad (17)$$

where $j = \mathrm{ceil}(L^i_m/w_m)$.

At the last step of the algorithm, the maximum values of the S/N $Y^i[n]$, the boxcar width $W^i[n]$, and the longest accumulated partial sum representing the value of the boxcar filter of width $L_{\mathrm{MAX}}[n]$ are stored into the device memory.

The intermediate partial sums produced by the algorithm are not stored to device memory as they are needed only for finding the maximum S/N for every input sample.

---

[6] GPU registers are the fastest type of GPU memory but are private to an individual thread.

[7] Shared memory is a fast user-managed cache on the GPU that all threads in a threadblock can access.

3.2. IGRID Algorithm

Our second algorithm, which we call IGRID, is based on the decimation in time SPD algorithm (DIT algorithm). The DIT algorithm has poor sensitivity, with an average signal loss of 20% and worst signal loss of 40%, but it has high performance. In terms of RS/N, both $\mathrm{RS/N_{max\,/\,min}}$ are low. The IGRID algorithm decreases signal loss at the expense of the performance.

The DIT algorithm, shown in Algorithm 3, is a sequence of decimations in time applied repeatedly on the input time-series x^0, thus, in effect producing boxcar filters of width $L^i = 2^i$, where i is the number of decimations performed so far. Therefore, we have $\mathfrak{B} = \{1, 2, 4,\ldots, 2^I\}$, where I is the total number of iterations performed. The distance between these boxcar filters is the same as the width $L_s^i = 2^i$. Based on our analysis in Section 2, we can say that large values of L_s are mostly responsible for low $\mathrm{RS/N_{min}}$ values, while sparse coverage of boxcar widths $\mathfrak{B}$ are responsible for low $\mathrm{RS/N_{max}}$ values. The IGRID algorithm corrects for these shortcomings.

Algorithm 3: Decimation in time (DIT) algorithm. Function `Calculate_S/N` calculates S/N based on boxcar filter value, boxcar width, and mean and standard deviation. Function `Compare_S/N` compares S/N values for the same sample and stores highest S/N.

```
Input: $x^i$, $n$;
Output: $Y[n]$, $W[n]$;
float S/N, temp;
int width;
int J number of iterations performed by the kernel;
S/N = 0;
for $j = 1$ to $J$ do
    Calculate_S/N ($x^j[n]$, S/N);
    width = $2^j$;
    Compare_S/N (S/N, width, $Y[n]$, $W[n]$);
    $x^{j+1}[n] = x^j[2n] + x^j[2n+1]$;
end
```

One way of decreasing L_s is to perform an additional DIT operation on the input data x^i producing additional time-series x^{i+1}, which is shifted by one sample (2^i samples from the perspective of the initial data). That is, for initial time-series, we would create one more time-series $x_1^1[n] = x^0[2n + 1] + x^0[2n + 2]$ in addition to the already existing time-series $x_0^1[n] = x^0[2n] + x^0[2n + 1]$, which results from DIT operation that was not shifted. Let's call these decimated time-series *layers* and decorate them with a subscript indicating their shift in the number of samples with regard to the initial time-series x^0. That is, x_h^i is a layer in the iteration i with the shift h. Let's also introduce *IGRID step*, which is a set of layers with different shifts but with the same decimation.

In general, if we have a *parent* layer x_h^i, which is decimated by i times (that is, each sample is a sum of 2^i samples from the initial time-series x^0) we can use it to calculate two new *child* layers, one layer x_h^{i+1} with the same shift h as the parent's and second layer x_{h+d}^{i+1}, which is shifted by $d = 2^i$ initial time samples compared to the parent. This is shown in Figure 5, where we have:

1. $i = 0$ Layer: $0 \rightarrow 0, 1$
2. $i = 1$ Layer: $0 \rightarrow 0, 2$ and $1 \rightarrow 1, 3$
3. $i = 2$ Layer: $0 \rightarrow 0, 4$ and $2 \rightarrow 2, 6$; note that layers 1 and 3 are not needed for further iterations.

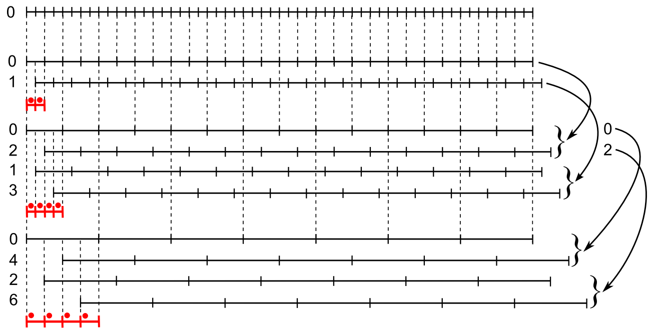

Figure 5. A graphical representation of our IGRID algorithm. The horizontal solid black lines represent the time-series where each slot is one time sample. The DIT algorithm is represented by time-series preceded by zero. Our IGRID algorithm improves the loss in sensitivity of the DIT algorithm by introducing additional layers of boxcar filters that have an additional time shift associated with them.

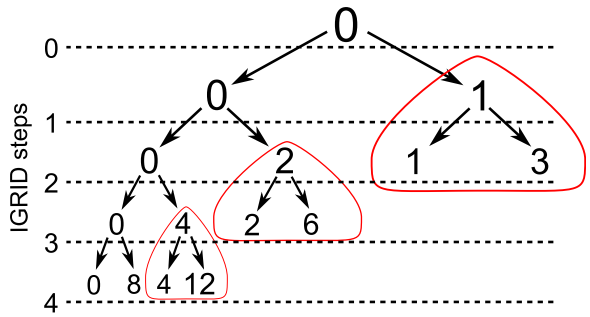

Figure 6. Representation of the IGRID algorithm as a binary tree. Areas that are enclosed in red contains layers that are not needed for the computation of the wider boxcar filters in the following IGRID steps.

In this example, we have chosen not to use layers x_1^2 and x_3^2. This means the L_s increases, but we do not need to keep these layers in memory and process them in further iterations.

Thus, by using P layers, we can decrease the distance between boxcar filter to $L_s = 2^i/P$. These layers must be shifted by $2^i/P$ samples in order to have a constant step between each boxcar filter. An uneven distribution of boxcar filters leads to increased signal loss at some parts of the input time-series.

We can look at the algorithm differently using the time shifts alone, which now represent appropriate layers. This is shown in Figure 6. The layer dependencies (from Figure 5) have the structure of a binary tree (for decimation factor $D = 2$). If we focus on the right branches (marked in red), we see that after a few IGRID steps, there are no layers dependent on layers located in these right branches of the binary tree. If we are able to keep and process all layers from these branches in local memory, then all we need to keep in the device memory are layers x_0^i.

The binary tree structure also hints at how we can deal with the second problem of the DIT algorithm, which is the scarcity of the set of boxcar filter widths $\mathfrak{B}$. By decreasing L_s we increase $\mathrm{RS/N_{min}}(S)$ but not $\mathrm{RS/N_{max}}(S)$, which could result in the situation, where $\mathrm{RS/N_{min}}(S)$ cannot increase further since $\mathrm{RS/N_{min}}(S) \leqslant \mathrm{RS/N_{max}}(S)$. This is shown in Figure 2.

We can increase the number of calculated widths if we use boxcar filters from previous IGRID steps and add them to boxcar filters from the current IGRID step. In other words, we traverse the binary tree in an upward direction toward the root. The IGRID steps contain boxcar filters of width $L = 2^i$; thus,

Table 1
Example of Possible Boxcar Widths That Could Be Calculated by Traversing the Binary Tree in an Upward Direction and Adding Partial Sums from Previous Iterations Using Equation (18)

Depth	Width								Increment
	a_0	a_3	a_2	a_3	a_1	a_3	a_2	a_3	
a	1	−1	−1	1	1	−1	1	1	
0	16								0
1	16				24				+8
2	16		20		24		28		+4
3	16	18	20	22	24	26	28	30	+2

by using previous IGRID steps, we can produce boxcar filters of width

$$L = \sum_{k=0}^{d} a_k 2^{i-k}, \tag{18}$$

where d is depth or the number of previous IGRID steps used, and $a_k = \pm 1$ are variables determining the width of the resulting boxcar filter. The values of a_k are further restricted to $a_0 = 1$ for positive L and also $a_1 = 1$ since smaller boxcars then $L = 2^i$ would be calculated in the previous IGRID step. This means that any given IGRID step, depending on depth d, can produce any boxcar width $2^i \leqslant L < 2^{i+1}$. An example of this is shown in Table 1 for a starting boxcar of width $L = 16$. However, by increasing the depth d, we increase the number of layers that the IGRID algorithm needs to access and, therefore, share between IGRID steps. The effect on $\mathrm{RS/N_{max}}(S)$ caused by increasing the set $\mathfrak{B}$ is shown in Figure 7.

The advantage of the IGRID algorithm is that it has low-memory bandwidth requirements (for lower precision), as all data required by the algorithm are contained in the decimated input data x^i. The disadvantage of the IGRID algorithm is that for higher precision, we have to use previous IGRID steps, which substantially changes the character of the algorithm (more input data, more complicated calculations of boxcar filters). This makes the IGRID algorithm hard to implement in a flexible way.

3.2.1. IGRID GPU Implementation

We have implemented the IGRID algorithm with three levels of precision; these are abbreviated as IG(d), where d is the depth. We have used three depths $d = 1, 2, 3$, which correspond to IGRID algorithms IG(1), IG(2), and IG(3), respectively. Each implementation of the IGRID algorithm also uses a different number of layers. We have chosen four layers for IG(1), eight layers for IG(2), and 16 layers for the IG(3) algorithm per IGRID step. We have implemented three different precisions because they allowed us to optimize each case and get the highest performance.

The main problem of the IGRID algorithm, especially when it is performed with a higher-depth parameter, is the amount of data that needs to be shared between IGRID steps. For IG(1), we do not need any additional data since the input layer x_h^{i-1} can also be used to calculate wider boxcar filters. However, for IG(2), we also need, in addition to the input layer x_h^{i-1}, its parent x_h^{i-2}, and finally for IG(3), we need the parents of the parents (x_h^{i-3}). This means that the size of the input data for IG(2) is twice as big as that for IG(1), and the size of the input data for IG(3) is four times bigger than that of IG(1). All of this data needs to be read and written by each IGRID step. This would slow down the code considerably and make it memory-bandwidth-bound.

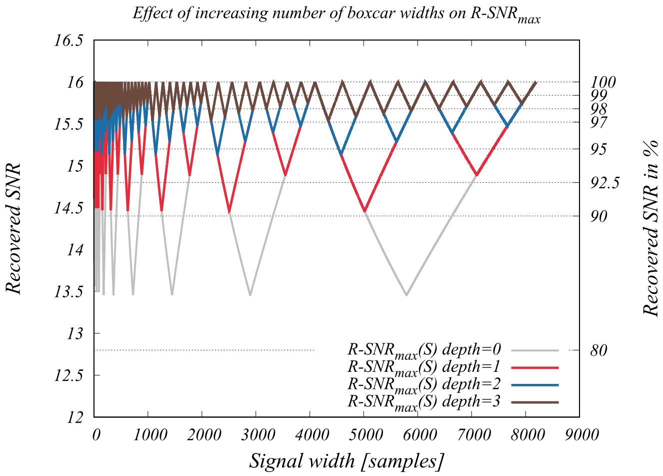

Figure 7. Effect of using the previous IGRID steps to increase the set of boxcar filter widths $\mathfrak{B}$ on $\mathrm{RS/N_{max}}(S)$.

Algorithm 4: Pseudo-code for IGRID SPD algorithm. Variable I is the number of IGRID steps required for single-pulse search, which is calculated on the maximum boxcar width L_{end}, which is user-defined, SpK is the number of IGRID steps performed by the GPU kernel, and A is the number of layers used.

```
calculate number of kernel executions $K = I/SpK$;
loop through kernel executions;
for $k = 0$ to $K$ do
    GPU Kernel begin
        loop through IGRID steps performed by kernel;
        for $s = 0$ to $SpK$ do
            loop through $A$ layers;
            for $a = 0$ to $A$ do
                calculations of layers;
            end
        end
    end
end
```

In our implementation of the IGRID algorithm, we have chosen to process more than one IGRID step per one execution of the GPU kernel. Algorithm 4 describes how the IGRID steps are split into blocks. Processing more IGRID steps per GPU kernel is beneficial in a number of different ways. If we perform more IGRID steps together, we not only remove device memory accesses between individual IGRID steps, but we also reduce the amount of data shared between consecutive GPU kernel executions, since each IGRID step reduces the amount of data by half due to decimation in time. The data shared between IGRID steps within one GPU kernel execution are stored in GPU shared memory.

The second benefit comes when we process the whole right branches that are marked in Figure 6. As discussed above, each right branch depends only on the layer with zero shift, but in order to calculate whole IGRID step, we might need layers from different branches. This is demonstrated by IGRID with a step of two in Figure 6. We see that, to calculate a whole step, we need layers with shifts $h = \{0, 2, 1, 3\}$, but layers with shifts $\{2\}$ and $\{1, 3\}$ are from different branches. If we do not

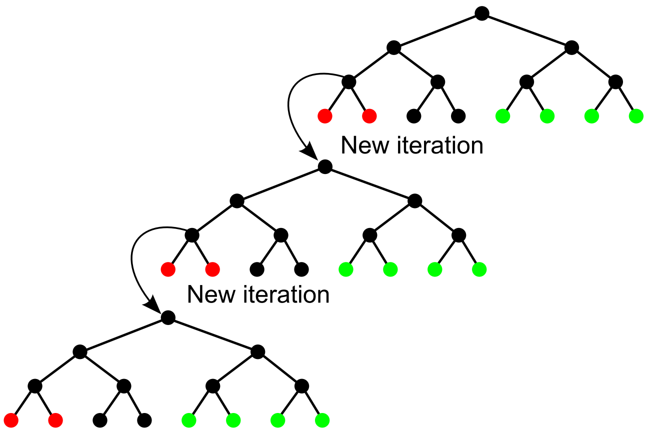

Figure 8. Representation of how IGRID kernels compute layers and progress to the next iteration. The black circles represent layers that we must calculate in order to reach the desired sensitivity. The red circles show layers that are calculated but not written to device memory (thus recalculated later by the next iteration), and the green circles are layers that are calculated but not needed; these are used to increase sensitivity.

calculate the entire right branch within one execution, we would have to save all or some of the intermediate data to the device memory.

Therefore, we have chosen to calculate whole subsections of the binary tree starting always at the layer with zero shift and going down to a chosen depth, even if it means calculating parts of the tree that are later discounted. Recalculating parts of the binary tree allows us to save device memory bandwidth and increases the performance in the process. This is shown in Figure 8. The other indirect benefit is that we can achieve higher sensitivity, since we calculate parts of the tree that would not otherwise be calculated.

The calculation of the binary tree itself inside the GPU kernel exploits the property of decimation in time, where the output-decimated time-series is half the size of the input. After decimation, we are able to fit two decimated layers into the memory space instead of one. The process of calculating these two child layers from the parent layer can be performed in one operation. If we use a running sum of size two (a boxcar filter of width two), we can calculate the two child layers in-place, but they will be mixed together: one in even samples and the second in odd samples. Figure 9 depicts how this calculation of a subsection of the binary tree is performed. As we perform more decimations, we can fit more layers into the same memory space, but also, the individual layers are more intertwined as we see in Figure 9 for later IGRID steps.

Algorithm 5: The IGRID IG(2) algorithm. Function `Calculate_S/N` calculates S/N and `Compare_S/N` compares S/N for the same sample.

```
Input: $x_h^i$, $x^{i-1}$, $n$, $i$;
Output: $x_h^{i+1}$, $x_{h+2^i}^{i+1}$, $Y^i$, $Y^i$;
B0 and Bd represent boxcar filters;
float S/N, B0, Bd;
int width;
Calculate two child layers;
$d = 2^i$;
width $= 2d$;
B0 $= x_h^i[2n] + x_h^i[2n+1]$;
Bd $= x_h^i[2n+1] + x_h^i[2n+2]$;
highest S/N is stored in initial data coordinates;
S/N = Calculate_S/N (B0, width);
Compare_S/N (S/N, width, $Y^i[2dn]$, $Y_L^i[2dn]$);
```

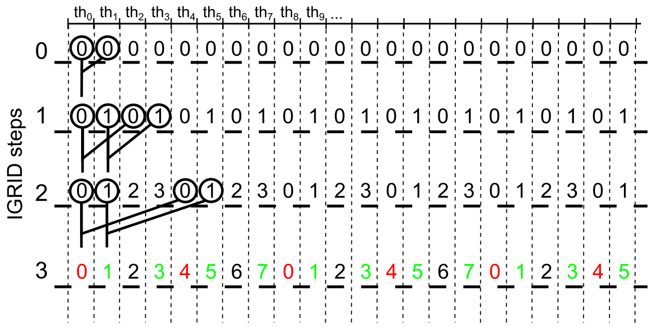

Figure 9. Visualization of IGRID GPU kernel. Iterations of the IGRID algorithm are separated by thick horizontal dashed lines. The number shows to which layer a given time sample belongs. The colored number uses the same color coding as in Figure 8. Black shows the layer we have to calculate, red shows a time sample from a layer that is calculated but not used, and green shows a layer that is calculated but is not necessary to achieve the desired sensitivity.

(Continued)

Algorithm 5: The IGRID IG(2) algorithm. Function `Calculate_S/N` calculates S/N and `Compare_S/N` compares S/N for the same sample.

```
S/N = Calculate_S/N (Bd, width);
Compare_S/N (S/N, width, $Y^i[2dn + d]$, $Y_L^i[2dn + d]$);
storing to memory;
$x_h^{i+1}[n]$ = B0;
$x_{h+d}^{i+1}[n]$ = Bd;
Traversing binary tree first depth;
width $=$ width $+ d$;
B0 $=$ B0 $+ x_h^i[2n+2]$;
Bd $=$ Bd $+ x_h^i[2n+3]$;
Calculate_S/N (...);
Compare_S/N (...);
Traversing binary tree second depth;
width $=$ width $+ d$;
B0 $=$ B0 $+ x_h^i[2n+2] + x_h^{i-1}[4n+6]$;
B0 $=$ B0 $+ x_h^{i-1}[4n+4]$;
Bd $=$ Bd $+ x_h^i[2n+3] + x_h^{i-1}[4n+8]$;
Bd $=$ Bd $+ x_h^{i-1}[6n+6]$;
Calculate_S/N (...);
Compare_S/N (...);
```

The GPU kernel uses shared memory to calculate these decimations. A thread that performs the boxcar filter of width two has to add the sample with increasing step, in order to use samples from the correct layer. Each thread always writes into the same memory address. Thus, during the calculation, a thread calculates time samples from different layers as is shown in Figure 5, where, for example, the seventh thread calculates eighth time sample from the zero shifted layer, then the fourth time sample from the layer that is shifted by one sample, and so on. Accesses to shared memory are without bank conflicts. The algorithm for IG(2) is shown in Algorithm 5.

3.3. Computational and Memory Complexity

The number of operations and number of memory accesses of one iteration for both proposed algorithms are summarized in Table 2.

4. Results

In this section, we present a selection of representative configurations for each algorithm and compare them based on

Table 2
Number of Operations and Number of Memory Accesses of Selected SPD Algorithms, where B is the Number of Boxcar Filters Performed in One Iteration

Algorithm	# Floating Point Operations	# Global Memory Accesses
Ideal boxcar filter	N^2	N^2
Decimation in time	$N \log_2 N$	$4N \log_2 N$
BoxDIT	$BN \log_2\left(\frac{N}{B} + 1\right)$	$6N \log_2\left(\frac{N}{B} + 1\right)$
IGRID(IG(1))	$N \log_2 N$	$4N \log_2 N$

performance and signal loss. Implementations presented here assume `fp32` floating-point numbers for the input.

First, we present signal loss for each algorithm and configuration for the idealized signal model (Section 2.1) using a rectangular pulse. We verify the correctness of our implementation of both algorithms using the idealized signal model by comparing the measured signal loss Ψ_{l}^{m} and Ψ_{w}^{m} ($\mathrm{RS/N_{max}}$, $\mathrm{RS/N_{min}}$) to predicted values of the signal loss Ψ_{l} and Ψ_{w}. We also present the error in the detected width of the pulse and the error in the detected position in time relative to its true position (time localization), which are introduced by each algorithm.

Next, we present RS/N and error in pulse width and time localization for the idealized signal model with Gaussian and double-Gaussian pulse profiles. These pulse shapes are a better approximation of the pulses encountered in real observations.

The RS/N and detected width for a rectangular, Gaussian, and double-Gaussian pulse shapes that are embedded in white noise are presented afterwards. For these results, we have used pulses of $\mathrm{S/N} = \{8, 5, 3\}$.

Following this, we present the performance of our algorithms on two generations of NVIDIA GPUs, the P100 and the Titan V. The performance of the algorithm is measured in the number of de-dispersion trials[8] (DM trials) that could be performed in real time for sampling time $t_{\mathrm{s}} = 64\,\mu\mathrm{s}$. We also present the dependency of this number on the maximum width performed by the algorithm.

Lastly, we present a comparison of AstroAccelerate with Heimdall and its single-pulse search algorithm. We also demonstrate how we calculate mean and standard deviation, which are then used for calculating the RS/N.

Results presented here are for three variants of the IGRID algorithm, which we have described at beginning of Section 3.2.1. To match these three IGRID configurations, we chosen three configurations of the BoxDIT algorithm, each with a different ratio of sensitivity and performance. We can change the sensitivity of the BoxDIT algorithm by changing the value of the maximum boxcar width calculated in the boxcar filter step of the algorithm. These configurations are: BD-32, which calculates a maximum width of $L_{\max}^{i} = 32$ between decimations for any i; BD-16 with maximum width of $L_{\max}^{i} = 16$; and BD-8, which uses maximum width of $L_{\max}^{i} = 8$.

4.1. Measurement of Signal Loss

To measure the signal loss of a given SPD algorithm, we have to find the RS/N produced by the algorithm and then calculate signal loss using Equation (7). The value of RS/N depends not only on the S/N of the injected pulse but also on its position within the time-series. This is because to localize the pulse and sum the power it contains, we use an incomplete set of boxcar filters. That is, the boxcar filters do not cover every possible pulse width at every possible time sample. A lower RS/N occurs when the injected pulse is not properly matched by these filters.

Therefore, to accurately measure $\mathrm{RS/N_{max\,/\,min}}$, we have to create data where a pulse of a given width will be present in all relevant positions in time. By sliding the pulse along its whole width in time, we can evaluate all of these relevant positions for a pulse of a given width. The test data used for the measurement of the $\mathrm{RS/N_{max\,/\,min}}$ consists of a set of time-series where each pulse shift is stored in its own time-series.

The test data are then processed by the SPD algorithm. From each time-series, we select the highest RS/N, and from these, we select the maximum, which is the $\mathrm{RS/N_{max}}$, and the minimum, which is the $\mathrm{RS/N_{min}}$ for a given pulse width.

To evaluate the value of the mean and standard deviation for longer boxcar filters, we have used the white-noise approximation as discussed in Section 2.1 and shown in Equation (5).

When we measure the $\mathrm{RS/N_{max}}$ and $\mathrm{RS/N_{min}}$ using our idealized signal model, the pulse will, as it slides along its width, be detected by boxcar filters placed at different times. Since we select the first maximum (minimum), we introduce a bias into our measurement of time localization, meaning that pulses are detected earlier than their true position in time.

When Gaussian noise is present in the test data, we must modify how we measure $\mathrm{RS/N_{max\,/\,min}}$. Since we select the highest S/N for each pulse position, this would prefer pulses that are enhanced by the addition of the noise. Hence, when noise is present, we use an averaged RS/N from multiple measurements of the pulse's RS/N at the same position. This way, we can mitigate the effect of the noise to some degree. Measurement made with noise will always have a tendency to get higher values of $\mathrm{RS/N_{max\,/\,min}}$, because we select the maximum S/N. For the S/N calculation, we have also used the white-noise approximation (5).

4.2. Signal Loss for Idealized Signal

The measured $\mathrm{RS/N_{max}}$ and $\mathrm{RS/N_{min}}$ together with measured signal loss (Equation (7)), error in detected width (Equation (10)), and error in time localization (Equation (11)), of the rectangular pulse using our idealized signal model for our chosen configurations of our BoxDIT algorithm are shown in Figure 10 and for our IGRID algorithm in Figure 11. Both errors are expressed as a percentage of the true pulse width.

The errors in the time localization shown in Figures 10 and 11 are biased for earlier pulse detection as discussed above.

The comparison to predicted signal loss, which is calculated by Equations (A3) and (31) are shown in Figures 10 and 11 as gray and black lines. The measured systematic signal loss Ψ_{l}^{m} must be higher or equal to the predicted Ψ_{l}, while measured worst signal loss Ψ_{w}^{m} must be lower or equal to the predicted worst signal loss Ψ_{w} for a given pulse width. In essence any measured value of the RS/N must lie be between values for Ψ_{l} and Ψ_{w}.

Lastly, in Figures 10 and 11, we also present the cumulative average signal loss $\langle \Psi_X(S_0) \rangle$ up to a given pulse width S_0, which is calculated as

$$\langle \Psi_X(S_0) \rangle = \frac{\sum_{s=1}^{S_0} \Psi_X(s)}{S_0 - 1}. \tag{19}$$

This represents the mean signal loss $\langle \Psi_X(S_0) \rangle$ one can expect for all pulses with width $S < S_0$.

---

[8] A DM trial is a time-series that is the output of the de-dispersion transform algorithm. It corrects for the effects of interstellar medium.

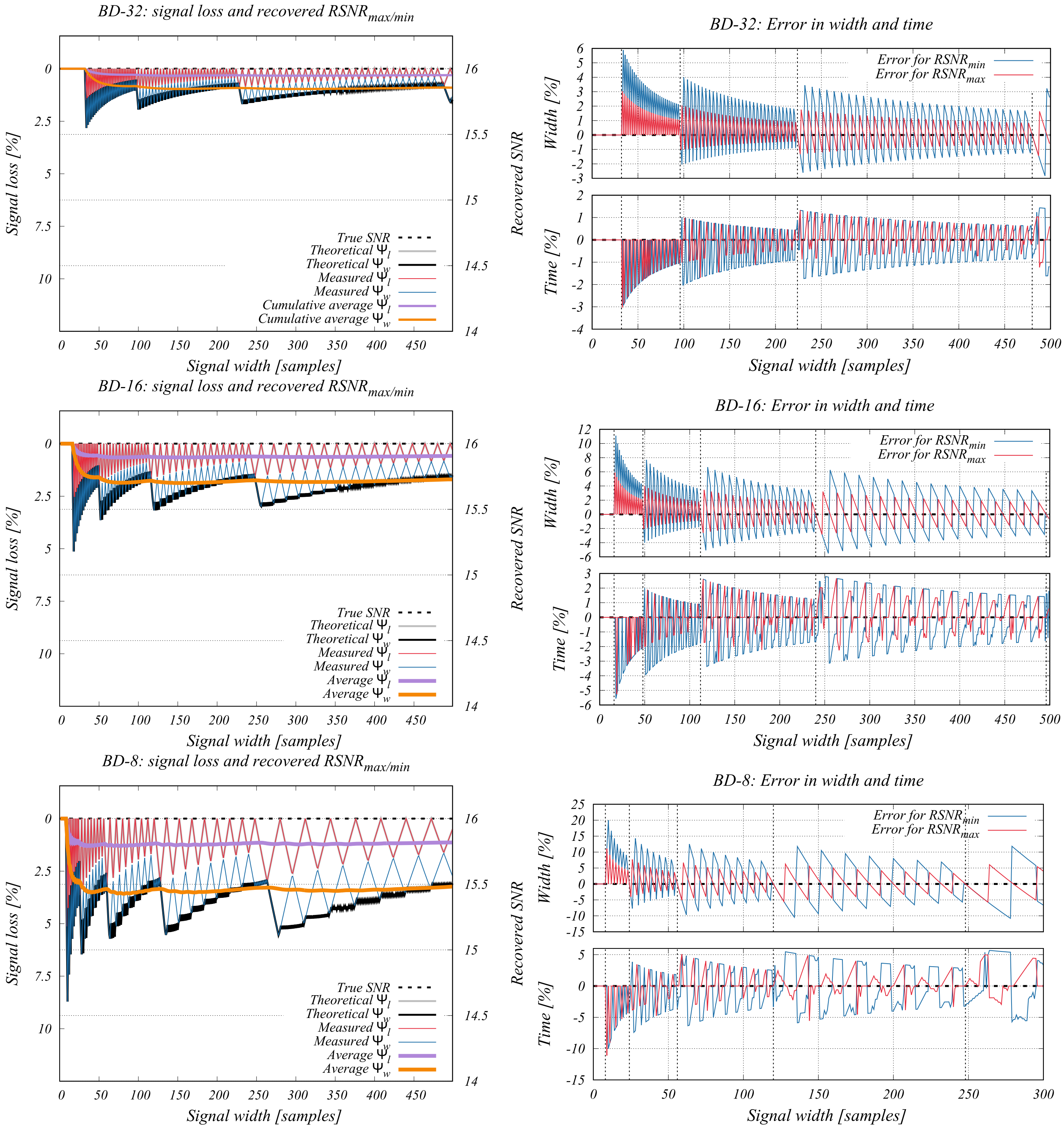

Figure 10. BoxDIT: measured signal loss Ψ_l and Ψ_w and associated $\mathrm{RS/N}_{\mathrm{max\,/\,min}}$ (left panel) and error in measured width (as a percentage of the real pulse width), time difference relative to the true position of the pulse (as a percentage of the real pulse width; right panel). These results are for our idealized signal model for three different configurations of our BOXDIT algorithm. A positive value of the error for the detected width indicates that the detected width is longer than the real width, and a negative value indicates that it is shorter than the real width. For localization in time, negative values indicate that the pulse was detected at an earlier time than the actual time at which the pulse occurred, and a positive value indicates that the pulse was detected later. Together with measured signal loss (red and blue), we also show the predicted values of systematic signal loss Ψ_l in gray (located under red line) and worst signal loss Ψ_w in black. Lastly, we also show the cumulative average signal loss in violet and orange.

4.2.1. Discussion

The averaged signal loss for our chosen configurations of our BoxDIT algorithm (Figure 10) ranges from 1% for configuration BD-32 to 3% for configuration BD-8. For IGRID, the averaged signal loss starts at 2% for configuration IG(3) and increases to 12.5% for IG(1). The sudden decreases in RS/N (increases in signal loss), which occur at different places for different configurations are caused by decimation in time. The

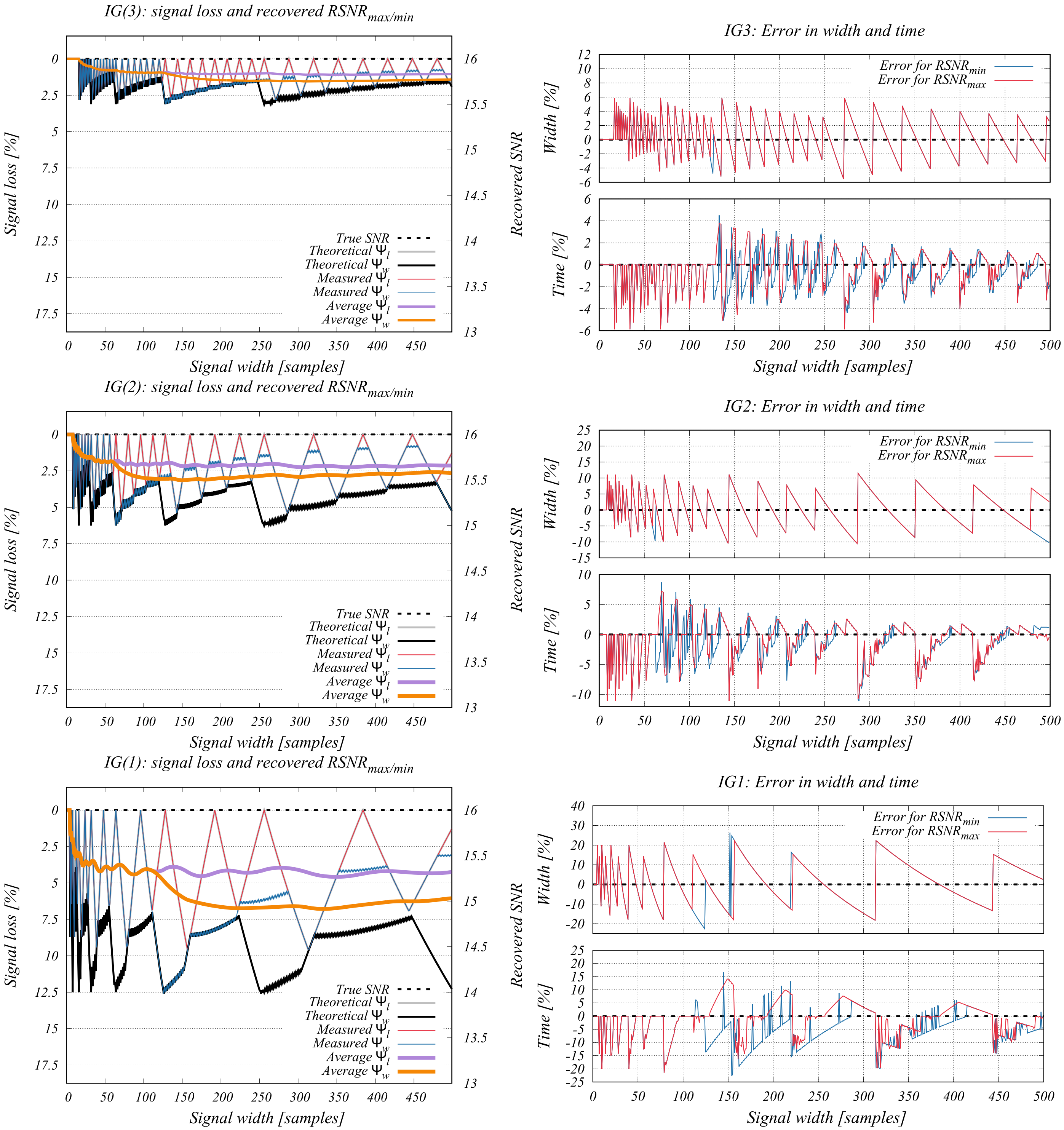

Figure 11. IGRID: measured signal loss $\Psi_{\rm l}$ and $\Psi_{\rm w}$ and associated RS/N$_{\rm max\,/\,min}$ (left panel) and error in measured width (as a percentage of the real pulse width), time difference relative to the true position of the pulse (as a percentage of the real pulse width; right). These results are for our idealized signal model for three different configurations of our IGRID algorithm. A positive value of the error for the detected width indicates that the detected width is longer than the real width, and a negative value indicates that it is shorter than the real width. For localization in time, negative values indicate that the pulse was detected at an earlier time than the actual time at which the pulse occurred, and a positive value indicates that the pulse was detected later. Together with measured signal loss (red and blue), we also show the predicted values of systematic signal loss $\Psi_{\rm l}$ in gray (located under red line) and worst signal loss $\Psi_{\rm w}$ in black. Lastly, we also show the cumulative average signal loss in violet and orange.

decimation in time increases boxcar separation L_s, which, in the case of width and time localization, determines the time resolution we are able to achieve. The boxcar separation represents the minimum increment in the measured width and the index of the time sample. Thus, it is more significant for shorter pulse widths, as it is a proportionally bigger part of the pulse's width. This is why the error in the width and time index decreases with pulse width.

Our IGRID algorithm behaves in a similar way (Figure 11). Our IGRID algorithm has, in general, higher signal loss and a higher error in width and time localization than our BoxDIT algorithm. This is because IGRID places boxcar filters more sparsely than BoxDIT.

A comparison of the measured signal loss Ψ_l^m and Ψ_w^m values to the predicted values of the signal loss Ψ_l and Ψ_w for both algorithms is shown in the left panel of Figure 10 for the BoxDIT algorithm and in Figure 11 for IGRID algorithm. Both algorithms give the same values for Ψ_l, as is predicted. However, both algorithms have better worst signal loss Ψ_w than what is predicted by Equation (31). This can be explained for each case. In the case of the IGRID algorithm, this is because we calculate more layers than are necessary (Section 3.2.1). This results in a smaller signal loss for some pulse widths. For the BoxDIT algorithm, the higher $\mathrm{RS/N_{min}}$ is cause by boxcar filters of a shorter width that cover the pulse completely (best case), which has higher RS/N and, thus, lower signal loss than the boxcar filters considered in our sensitivity analysis for the worst case.

4.3. Other Pulse Shapes

The profile of a real signal is usually far from rectangular, and this is why we have included results from studies of two other pulse profiles: a Gaussian shape and a combination of two Gaussians. We have measured RS/N, the error in detected width, and the error in time localization using the idealized signal model. For these alternative pulse shapes, we cannot use Equation (7) to calculate signal loss, because these pulses have a nonuniform distribution of power between samples of the pulse. Although the pulse is still normalized to the constant $\mathrm{S/N_T}$, the nonuniform distribution means we can sum most of the pulse power with much shorter boxcar filters that overestimate the S/N of the pulse, resulting in RS/N higher than $\mathrm{S/N_T}$.

These pulse shapes are inserted, in a similar fashion, to the rectangular pulse shape. The input time-series is zeroed—all elements are set to zero. The pulse is then injected at the required position.

To produce these pulses, we have sampled the normal distribution given by

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right), \qquad (20)$$

where the mean is set to $\mu = S/2$ and the variance, σ^2, depends on the shape type and width of the pulse. For a Gaussian-like shape, we have used $\sigma^2 = S/(8 \ln 2)$ (Brandt 2014), and for the double-Gaussian profile, we have used $\sigma^2 = (S/4)/(8 \ln 2)$ and $\sigma^2 = (S/5)/(8 \ln 2)$. The Gaussian profiles are then sampled at discrete intervals to produce values that are then injected into the idealized data set. These pulses are shown in Figure 12. The signal is normalized such that when all of its samples are summed together, i.e., a boxcar of signal width is applied, it will produce $\mathrm{RS/N} = C = 16$.

The results for the BoxDIT algorithm in the most sensitive configuration BD-32 are presented in Figure 13. We present results for the fastest configuration of our IGRID algorithm (IG (1)) in Figure 14. We also show an alternative figure of merit to the signal loss in the form of the recovered fraction of a signal's power in Figure 15 for both algorithms.

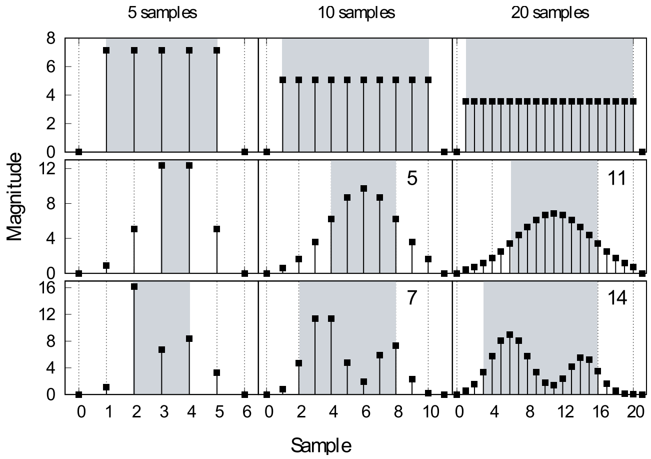

Figure 12. The three different pulse shapes used in this article. The pulses are shown with resolutions of 5, 10, and 20 samples. The shaded region in each graph shows the samples of the pulse that were summed by the boxcar filter, which produces the highest RS/N. The number of samples covered by the shaded area is at top right corner.

4.3.1. Discussion

Figures 13 and 14 show that the RS/N of pulse shapes other than rectangular are higher than the maximum for the rectangular pulse. This is due to the nonuniform distribution of power throughout these pulse shapes, where the power is concentrated into fewer samples. This means that a shorter boxcar filter can sum most of the power contained in the pulse, which results in a higher RS/N. The fluctuations in RS/N for short widths is cause by the poor discreet representation of pulse shapes, shown in Figure 12.

The error in measured width and position in time is presented in Figures 13 and 14. We see that boxcar filters systematically under-estimate pulse width when applied to a non-rectangular pulse. The error in the measured width is consistently about 40% of the true pulse width for the single Gaussian and about 30% for double-Gaussian. These findings are consistent with the fact that Gaussian-like pulse profiles tend to be detected with shorter boxcar filters than their true width due to the nonuniform distribution of power in the pulse.

The error in time localization is presented in Figures 13 and 14. Time delay in detection is again expressed as a percentage of the true pulse width. Positive values indicate that the pulse is detected later in time, while negative values indicate that the pulse was detected earlier in time. These results show that Gaussian pulses are detected later than their true position in time. This is also shown in Figure 12, which highlights the portion of the pulse that results in the highest RS/N detection. If we consider a Gaussian pulse that is 20 samples wide, we see that it is detected by the boxcar filter with width of 11 samples, centered about the peak. Meaning that the pulse is detected with a time delay of four or five samples, which represents 20% or 25% time delay compared to the true pulse width of 20 samples.

The recovered power by the BoxDIT (BD-32) and IGRID (IG(1)) algorithms for pulses of Gaussian-like shapes are presented in Figure 15. We see that to produce the highest RS/N for Gaussian-like pulses, the algorithm sums on average 85% of the signal's power or 94% for the double-Gaussian pulse profile. These results show that boxcar filters that sum all of the power contained within the pulse have lower RS/N and, thus, are not selected as candidates.

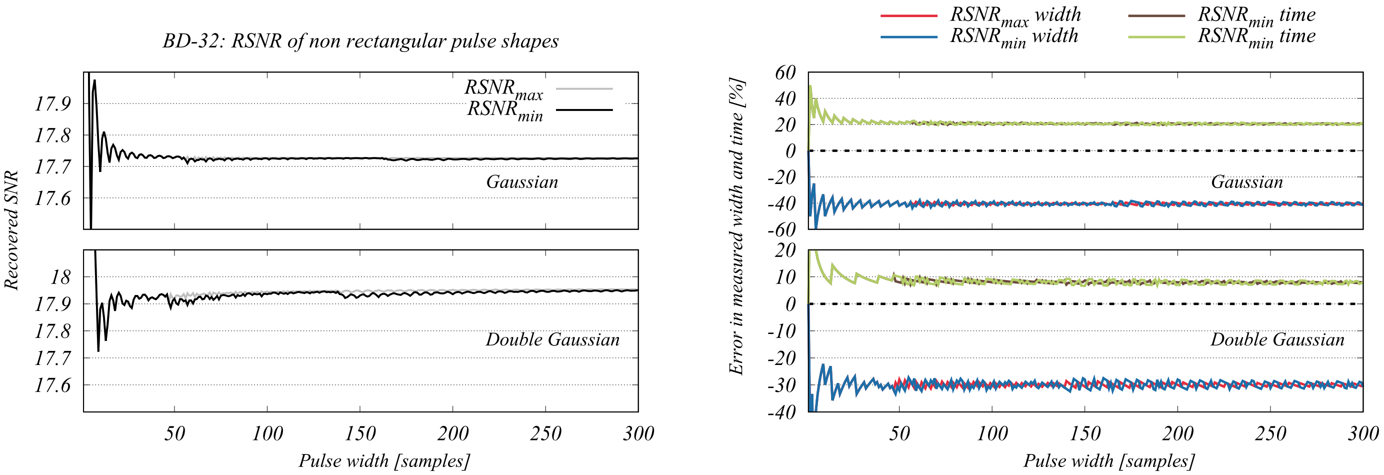

Figure 13. Measured RS/N values and error in pulse width and time localization for different pulse shapes for BoxDIT BD-32 configuration.

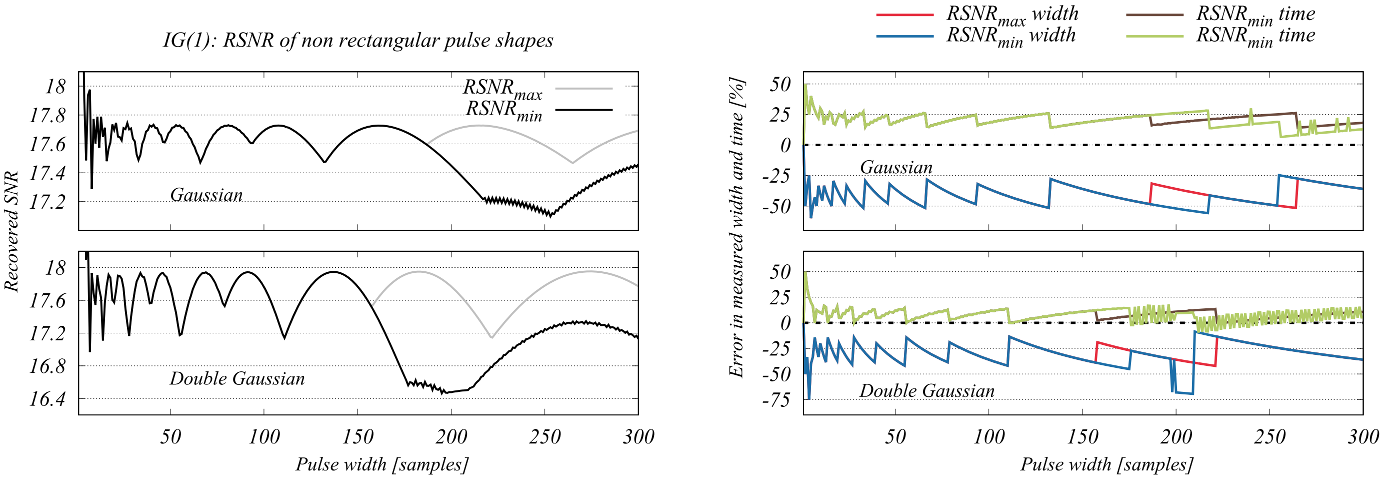

Figure 14. Measured RS/N values and error in pulse width and time localization for different pulse shapes for IGRID IG(1) configuration.

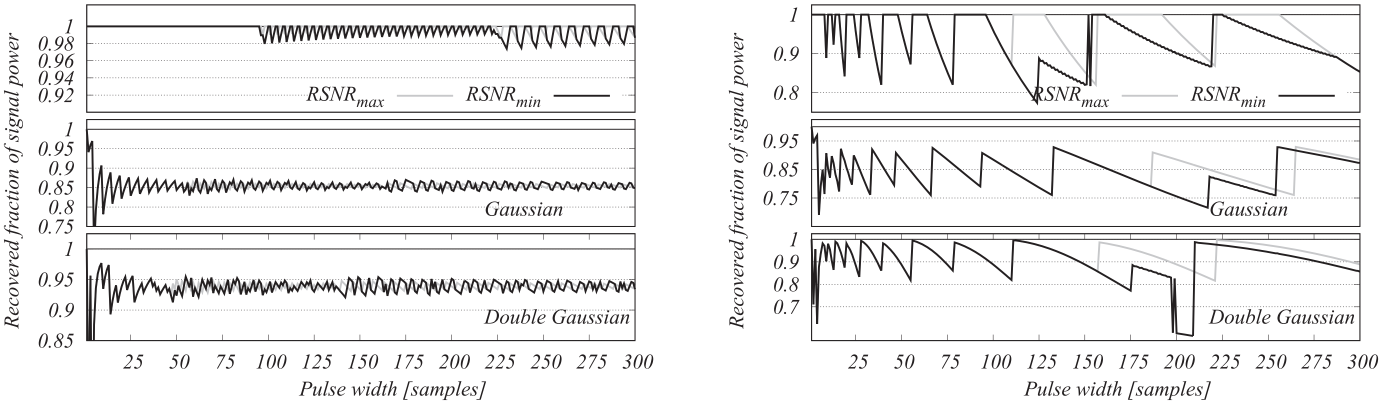

Figure 15. Fraction of power recovered by the both algorithms. BoxDIT algorithm in configuration BD-32 (left panel) and IGRID algorithm in configuration IG(1) (right panel).

The noticeable increase in the width error visible in Figure 14 for the double-Gaussian pulse profile as well as the drop in the fraction of recovered power in Figure 15 is because the IGRID algorithm (IG(1)) detects only the first Gaussian out of two that are present in the pulse. The error in time localization, which is unchanged, indicates that it is the first Gaussian that is detected. For other configurations of our IGRID algorithm, this was not observed. This is because these configurations have a denser set $\mathfrak{B}$ as well as shorter boxcar separation L_s, and they are able to better match the entire pulse.

4.4. Time-series with White Noise

To verify the results of our SPD algorithms when there is noise present in the input data, we have inserted Gaussian noise (mean $\mu = 0$, standard deviation $\sigma = 1$) into the idealized

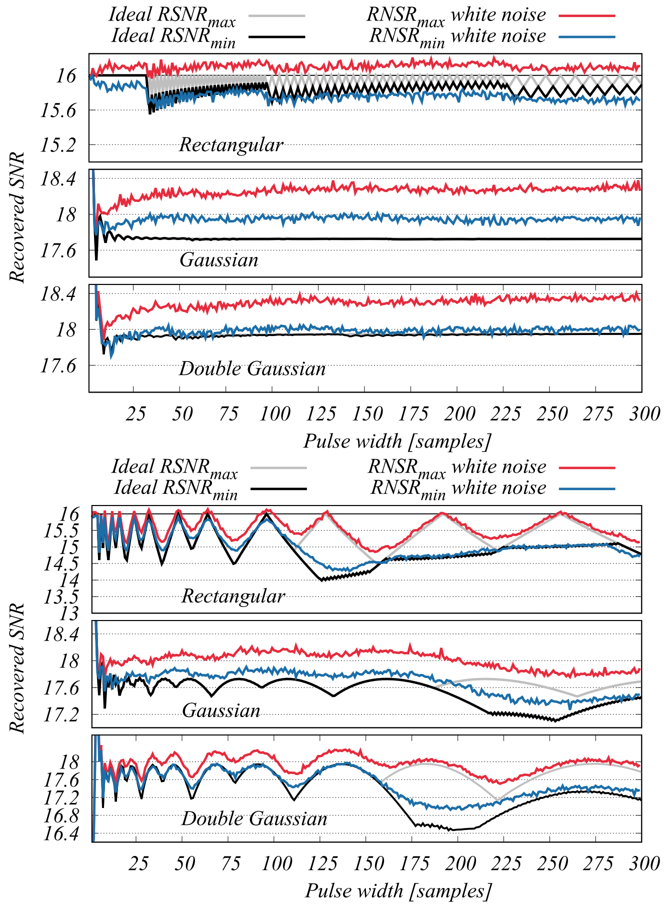

Figure 16. Comparison of $\mathrm{RS/N}_{\max/\min}$ for our idealized signal without noise (gray and black) with $\mathrm{RS/N}_{\max/\min}$ for signals with Gaussian white noise (red and blue).

signal and measured $\mathrm{RS/N}_{\max/\min}$. To measure $\mathrm{RS/N}_{\max/\min}$, we have used the method described in Section 4.1. The measured values of $\mathrm{RS/N}_{\max/\min}$ for BoxDIT in configuration BD-32 and IGRID in configuration IG(1) are presented in Figure 16. The injected pulses have S/N = 16. The effect of the noise on the measured $\mathrm{RS/N}_{\max/\min}$ is higher for pulses with lower S/N, as expected.

We have also used a time-series with white noise to test S/N extraction. Examples for each pulse shape are presented in Figure 17 for the rectangular pulse, and in Figure 18 for the Gaussian and double-Gaussian pulse. The pulses have S/N = {8, 5, 3}, width $S = 20$ samples, and they start at sample 100. For the rectangular pulse, the detection should occur at sample 110 by a boxcar filter of width 20 samples. For other pulse shapes, the boxcar width is shorter as shown in Figure 12. The addition of the while noise means that single pulses shown in these figures do not represent overall sensitivity of our algorithms.

4.4.1. Discussion

The values of RS/N reported by our BoxDIT algorithm in configuration with BD-32 and by our IGRID IG(1) algorithm are presented in Figure 16. The introduction of Gaussian white noise into our idealized model with rectangular pulse had only a marginal effect on the detected RS/N by both algorithms. This is, for the most part, due to the initial S/N of the injected pulse, which was $\mathrm{S/N_T} = 16$. We also see that RS/N produced by both SPD algorithms fluctuates about $\mathrm{RS/N_{min}}$ and often goes below $\mathrm{RS/N_{min}}$. These variations are caused by fluctuations in the detection boxcar's width and its position in time.

Studying other pulse shapes, we see that the RS/N reported by BoxDIT is higher than in case without noise. This is most visible for the Gaussian pulse shape. This, again, is due to the pulse shapes having a nonuniform distribution of power in their profile; hence, they are detected with shorter boxcar filters than the actual pulse width. This makes it easier for the added noise to change the behavior of the SPD algorithm by changing the distribution of power throughout the pulse. For example, by decreasing the peak in the Gaussian pulse shape, the boxcar that detects the pulse will be wider, as it needs to accumulate power from more samples. In combination with our normalization, this will decrease RS/N closer to 16. On the other hand, by slightly increasing the peak in pulse profile, the pulse will be detected by a shorter boxcar, producing higher RS/N.

We see that in the case of the rectangular pulse (Figure 17), the detection occurs at the correct position in time for S/N = 8 and S/N = 5. For pulse width S/N = 3, the highest RS/N is detected at time sample 116, but the detected width underestimates the actual pulse width. This detection is mostly due to noise. The pulse itself is detected correctly at the time sample 110 with width 20, but it has lower RS/N. With decreasing pulse S/N, we see that noise has a stronger effect on the detected width and position of the pulse, as would be expected. Figure 17 also shows same data in a wider perspective with all candidates above a given threshold. For S/N = 3, we see that the injected pulse is detected correctly; however, it is hard to distinguish from background noise.

4.5. Performance

We have evaluated the performance of our algorithms and their implementation in different configurations by measuring the execution time. As a derived metric, we have calculated the number DM trials that can be searched in real time using sampling time $t_s = 64\,\mu$s. Processing data in real time means performing all of the required operations on the data faster than we acquire the data. The number of DM trials processed in real time N_{DM} is calculated as ratio of the number of processed time-samples per second by the SPD algorithm N_{P} and the number of acquired time-samples per second $N_{\mathrm{S}} = 1/t_s$. That is,

$$N_{\mathrm{DM}} = \frac{N_{\mathrm{P}}}{N_{\mathrm{S}}}. \tag{21}$$

The number of DM trials processed in real time depends on the telescope sampling time, and for a different sampling time other then $t_s = 64\,\mu$s, the number of DM trials searched in real time will be different. For example, for $t_s = 128\,\mu$s, the number of DM trials processed in real time will be multiplied by factor of two $N_{\mathrm{DM}}^{(128)} = 2N_{\mathrm{DM}}^{(64)}$.

The number of DM trials searched in real time also depends on the maximum width of the boxcar filter L_{end} used for detection of the pulses. In the following results, we have used $L_{\mathrm{end}} = 8192$ samples, which, for a sampling time $t_s = 64\,\mu$s, means a boxcar filter that is approximately half a second long. When searching for shorter pulses, the performance of the SPD algorithm can be increased by using shorter L_{end}.

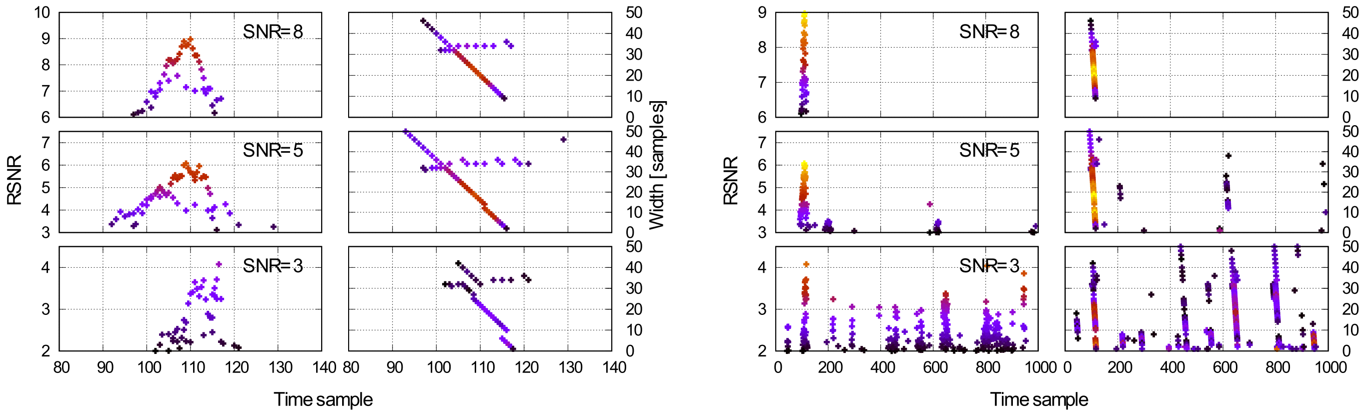

Figure 17. Results for the rectangular pulse detections with different S/N embedded in white noise. The left graph shows a injected rectangular pulse of different initial S/N. It is split to show the RS/N values on the left and boxcar widths on the right. The points are color coded by the RS/N. There might be more than one S/N value per sample because the SPD algorithm runs in multiple iterations and the results of all iterations are shown. Same results are shown in wider perspective in the graph on the right-hand side. We see that the S/N = 3 pulse would be hard to distinguish from background noise. This result is for BoxDIT algorithm with maximum search width $w = 8000$ samples.

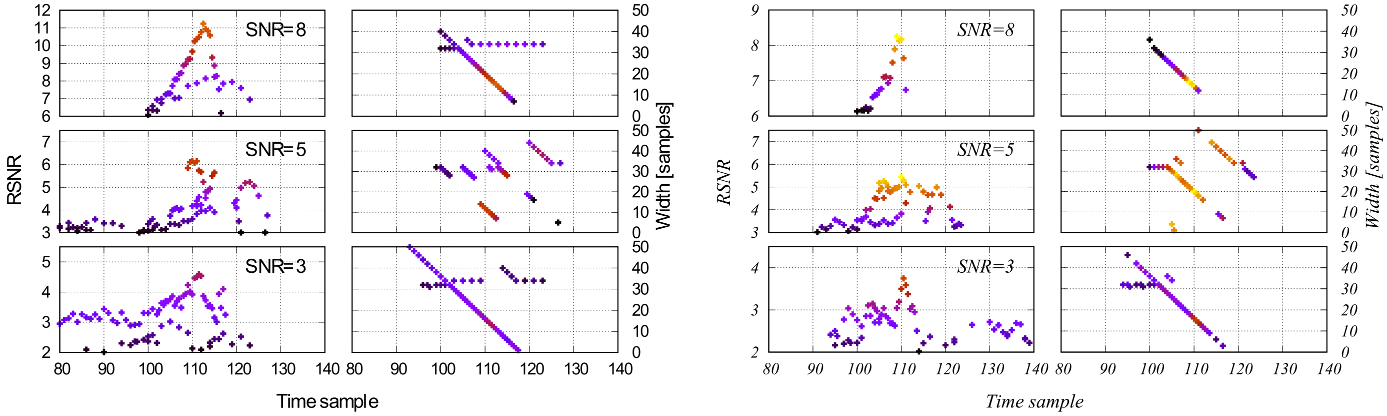

Figure 18. Results for the detections for the Gaussian pulse with different initial S/N embedded in white noise is shown on the left set of graphs, and for double-Gaussian on the right set of graphs. Each graph shows RS/N on the left and detected widths on the right. The points are color coded by the RS/N. Results are for BoxDIT algorithm with maximum search width $w = 8000$ samples.

Table 3
GPU Card Specifications

	P100	Titan V
Total CUDA Cores	3584	5120
Streaming Mul. (SMs)	56	80
Core Clock	1303 MHz	1455 MHz
Memory Clock	1406 MHz	850 MHz
Device m. bandwidth	720 GB s^{-1}	652 GB s^{-1}
Shared m. bandwidth	9121 GB s^{-1}	14550 GB s^{-1}
FP32 performance	9.3 TFLOPS	12.3 TFLOPS
GPU memory size	16 GB	12 GB
TDP	250 W	250 W
CUDA version	10.1	10.1
Driver version	418.39	415.27

Note. The shared memory bandwidth is calculated as BW (bytes/s) = (bank bandwidth (bytes)) × (Clock Frequency (Hz)) × (32 Banks) × (# multiprocessors).

We present the performance of our implementations on two NVIDIA GPUs. The first is the TESLA P100, which is a scientific card from the Pascal generation. The second is the Titan V, which is a high-end prosumer card from the Volta generation. We have not included any card from the current Turing generation since this generation is not designed for scientific tasks and workloads. The hardware specifications of these two GPUs are summarized in Table 3.

Execution time and how it scales with an increasing number of time-samples (per one DM trial) and execution time scaling with increasing number of DM trials for the P100 and the Titan V are presented in Figure 19. The calculated number of DM trials processed in real time for different sampling times is presented in Figure 20.

The comparison of the two algorithms with respect to their signal loss and achieved performance is presented in Figure 21. This figure combines results presented in Figures 10 and 11 with the number of DM trials processed in real time using a sampling time $t_s = 64\ \mu$s. This figure allows us to compare the algorithms signal loss/performance ratio.

Performance as a function of maximum boxcar width searched by the SPD algorithm is presented in Figure 22. In the case of IGRID algorithm, these results depend on how many IGRID steps we perform per GPU threadblock.

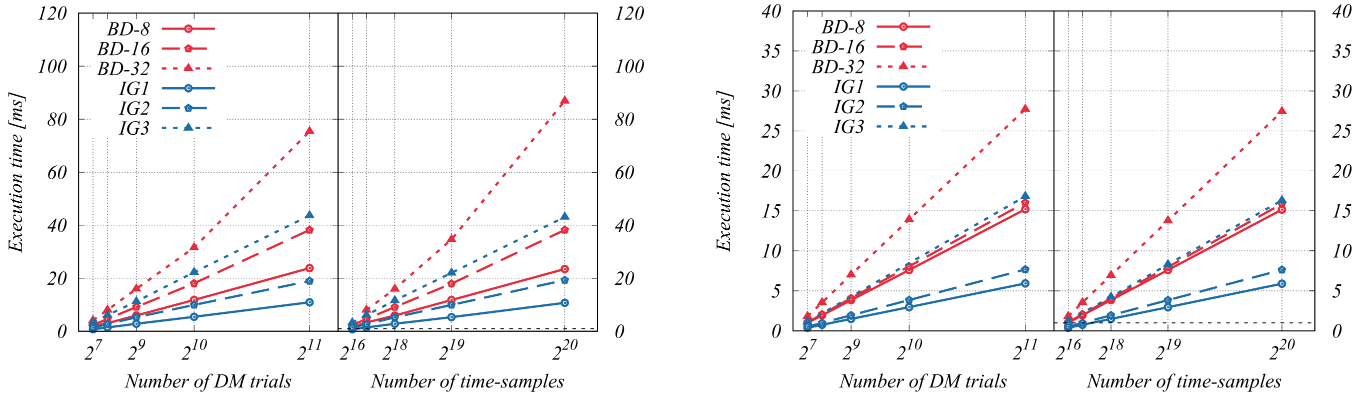

Figure 19. Measured execution time (P100 and Titan V) for both BoxDIT and IGRID algorithms for increasing number of DM trials with 131072 time-samples per DM trial is shown in the left panel. Execution time for a varying number of time-samples per DM trial for 256 DM trials is shown in the right panel. All results are for a maximum boxcar width $L_{\mathrm{end}} = 8192$ samples. In both cases, our algorithms scale linearly.

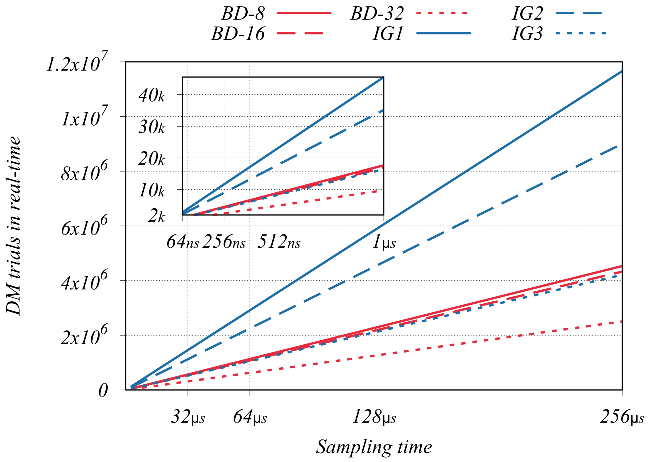

Figure 20. This figure shows the number of DM trials that can be processed in real time for a range of common sampling times.

4.5.1. Discussion

The execution time for both algorithms is presented in Figure 19. Both algorithms scale linearly with the number of DM trials as well as with the number of time-samples. This is because the SPD problem is separable into independent parts at the level of DM trials as well as within a single DM trial. Thus, adding more time samples or DM trials is equivalent.

The calculated number of DM trials processed in real time for the Titan V GPU is presented in Figure 20. We see that for a sampling time of $t_{\mathrm{s}} = 64\ \mu$s, the IG(1) algorithm is capable of processing three million DM trials in real time, and the BD-32 algorithm is capable of processing almost 600,000 DM trials in real time. When considering radio astronomy beyond the SKA era, for extremely short sampling times such as $t_s = 256$ ns, the real-time performance of our IG(1) algorithm is about 10,000 DM trials in real time. This, however, has to be taken in the context of a whole pipeline in which the SPD algorithm is used.

The comparison of both SPD algorithms based on their real-time performance for $t_{\mathrm{s}} = 64\ \mu$s and their cumulative averaged signal loss is presented in Figure 21. When considering the trade-off between signal loss and performance, both algorithms perform well. For roughly a two-time increase in signal loss, the performance also increases by roughly two times. There are some exceptions to this rule. For example, BD-8 running on Titan V has almost the same performance as BD-16. Also note that IG(1) on Titan V offers a performance increase of only 1.4× when compared to IG(2).

Table 4
Compute/Memory Utilization of Used GPU's per Configuration as Reported by the NVIDIA Visual Profiler

	P100	Titan V
BoxDIT BD-32	90%/15%	85%/45%
BoxDIT BD-16	80%/35%	80%/85%
BoxDIT BD-8	80%/55%	45%/85%
IGRID IG(3)	75%/25%	85%/45%
IGRID IG(2)	85%/35%	90%/75%
IGRID IG(1)	85%/55%	65%/85%

Figure 21 presents the different behavior of the GPUs used and also differences between both algorithms. The GPU resource utilization, as reported by the NVIDIA visual profiler for selected configurations of the BoxDIT and IGRID algorithms, is summarized in Table 4.

When we compare utilization of both algorithms, we see that they are similar on both GPUs. On the P100 GPU, both algorithms are limited by computation performance. On the Titan V GPU, they are limited by global memory bandwidth for higher signal loss configurations (BD-8, IG(1)), and by computation performance for lower signal loss configurations (BD-32, IG(3)). Algorithms with lower signal loss have to calculate a denser set of boxcar filters (more different widths), which are closer together. This requires more computations, but they use the same amount of input data. Despite similar GPU utilization, IGRID is the better-performing algorithm.

There are three main reasons why the IGRID algorithm is faster than BoxDIT. First, the IGRID algorithm requires a smaller number of boxcar filters than the BoxDIT algorithm because IGRID decimates the input data after each IGRID step. If we compare BoxDIT BD-8 with the IGRID IG(2) algorithms, the number of boxcar filters required by the IGRID IG(2) algorithm is half[9] compared to the number of boxcar filters required by the BoxDIT BD-8. On the P100 GPU where both algorithms are limited by computation, we see that the IG(2) configuration has a lower signal loss but a higher

---

[9] The BD-8 performs 10 iterations with decimation after each step and in each step performs eight boxcar filters. This gives $8 \cdot 2 \cdot N$ boxcar filters. The IGRID performs 14 IGRID steps of which the first six are in full resolution and all subsequent IGRID steps are decimated. This gives $6N + 2N = 8N$ boxcar filters.

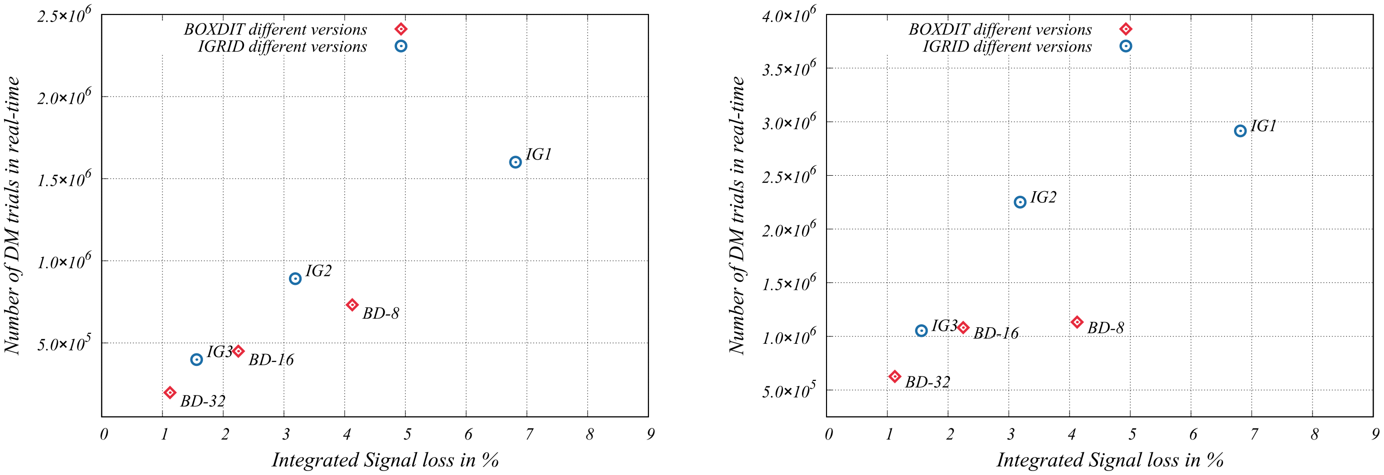

Figure 21. Dependence of performance on the maximum width computed by the SPD algorithm used. Results are for P100 (left panel) and TITAN V (right panel).

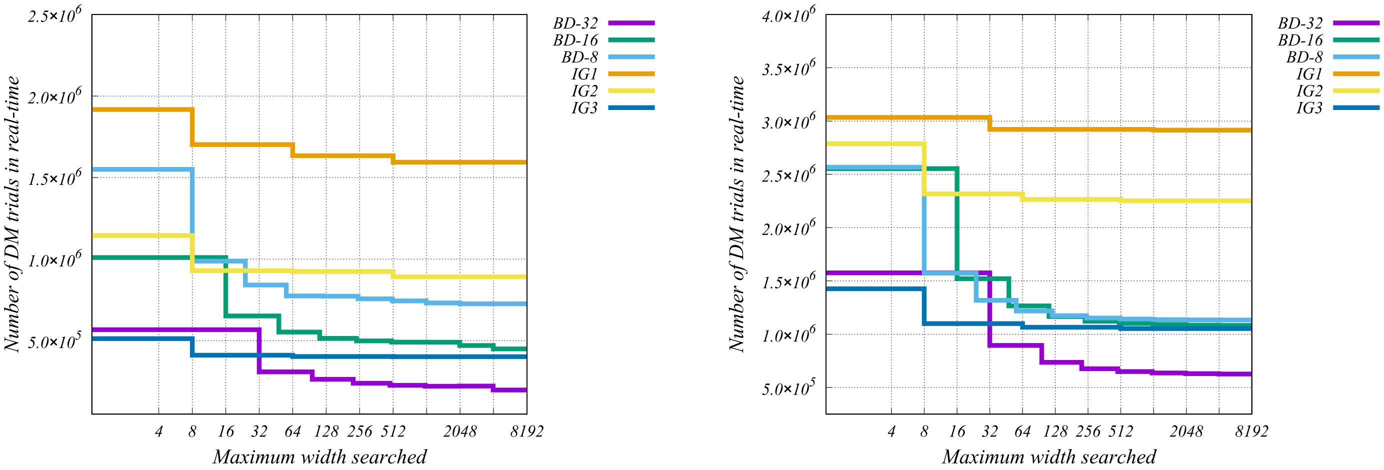

Figure 22. Dependence of performance on the maximum width computed by the SPD algorithm used. Results are for P100 (left panel) and TITAN V (right panel).

performance, while compute utilization is almost the same for both algorithms. This shows that IGRID is more economical with the number of boxcar filters required to achieve a similar signal loss.

The IGRID algorithm also requires smaller input data. This is because in the IGRID algorithm, the input time-series x^i serves two functions. First, its values represent the values of the previously calculated boxcar filters that are used to built even longer boxcar filters. Second, it serves as sample values, i.e., elements that are added to the existing boxcar filters to build up longer boxcars. In the case of the BoxDIT algorithm, we have to store these quantities in separate input arrays.

On Titan V, both algorithms in their less-sensitive variants are limited by device memory bandwidth. When we compare IGRID IG(2) with BoxDIT BD-8, we again see that IGRID is the better-performing algorithm with lower signal loss, while both algorithms are limited by device memory bandwidth.

The IGRID algorithm has some disadvantages as well. For high efficiency, each configuration needs to be implemented individually and the complexity and size of the input required by IGRID increases with decreasing signal loss. Therefore, extending IGRID to even higher sensitivities (lower signal loss) might be difficult. Furthermore, the signal loss of the IGRID algorithm depends on parameters of the implementation. Even though they do not decrease below the theoretically predicted worst signal loss Ψ_{w}, the analysis of the data processed by the same configuration of IGRID algorithm (same depth) but with different parameters might cause issues.

This problem is not shared by the BoxDIT algorithm. All presented configurations of our BoxDIT algorithm are served by a single implementation of the BoxDIT algorithm that only takes as an input the number of boxcar filters to be performed (or maximum size of the boxcar filter) per iteration. Furthermore, these might change from iteration to iteration, which enables the user to decrease signal loss for certain pulse widths while increasing it for others.

Figure 22 shows how performance depends on the maximum boxcar filter width L_{end} used. We see that IGRID algorithms are much less sensitive to increases in L_{end}. This is due to lower number of GPU kernel execution needed by the IGRID algorithm to get to the desired boxcar width.

Lastly, we present present fractions of real time for SKA-like data. For SKA-like data, we assume that the single-pulse-detection pipeline will process 6000 DM trials per second with sampling time $t_s = 64\,\mu$s. The BoxDIT algorithm in configuration BD-32 is capable of processing SKA-like data 33× faster than real time and 106× faster than real time on Titan V. The IGRID algorithm in IG(1) configuration is

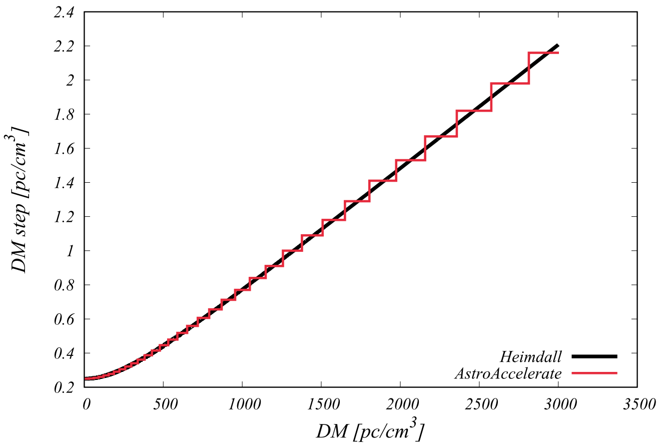

Figure 23. Comparison between DM steps used by Heimdall's and AstroAccelerate.

capable of processing SKA-like data 266× faster than real time on P100 GPU and 500× faster than real time on Titan V GPU. If the final SKA single-pulse-detection pipeline runs exactly in real time with no spare processing time, the IG(1) algorithm would take 0.4% of processing time on P100 GPU and 0.2% on Titan V GPU.

4.6. Heimdall Pipeline

In order to compare RS/N acquired by our SPD algorithms that are part of AstroAccelerate with another signal processing pipeline, we have chosen to use Heimdall. Heimdall is a GPU accelerated transient detection pipeline developed by Jameson & Barsdell (2018) and Barsdell et al. (2012).

AstroAccelerate's single-pulse search pipeline contains the following steps: (1) the data are de-dispersed; (2) the mean and standard deviation are calculated; (3) the SPD algorithm is calculated; and (4) candidates are selected using a peak-finding algorithm. Since RS/N by the pipeline depends also on other steps, we will briefly describe them as well.

4.6.1. De-dispersion Transform

The de-dispersion transform used in AstroAccelerate was developed by Armour et al. (2012). This implementation of the de-dispersion transform requires that the DM trials are calculated in groups with a constant step in DM. In contrast, the de-dispersion transform used in the Heimdall pipeline has an adaptive step and so can have a varying step in DM for every DM trial. This means that with AstroAccelerate, we cannot fully match the de-dispersion scheme used by Heimdall (the Heimdall scheme should be more accurate).

We have based the AstroAccelerate de-dispersion scheme on the one used by Heimdall, a comparison of the two presented in Figure 23. Furthermore, the input time-series for each DM trial can be decimated in time. This decimation step is matched for both codes apart from the first group of DM trials, which, in the case of AstroAccelerate, is performed at full time resolution. Heimdall, in contrast, only performs the first DM trial at full time resolution.

4.6.2. Calculation of Mean and Standard Deviation

The correct calculation or estimation of the mean and the standard deviation of the base level noise in the input data is essential because these values are used to calculate the RS/N.

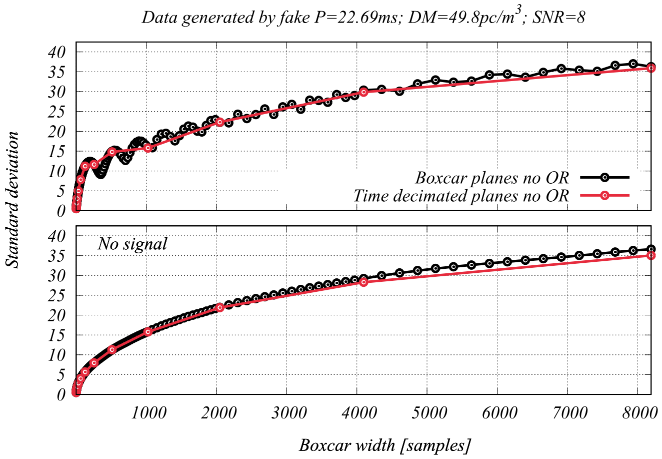

Figure 24. Difference between stdev values calculated from time-series after applying a boxcar of given width and standard deviations calculated from time-series after successive decimations in time. Data were generated by SIGPROC "fake." Top line: a time-series that contains a strong signal. Bottom line: a time-series without a signal present.

Only those detections with RS/N above a user-defined number of standard deviations are considered significant and potential candidate detections. To ensure we have a fast, flexible, and robust GPU code to calculate mean and standard deviation, we have chosen to implement a streaming algorithm by Chan et al. (1983) in CUDA for NVIDIA GPUs. This algorithm has several advantages. It is numerically stable and it is more precise than conventional methods. This algorithm is also very well suited to a parallel implementation.

We estimate the true value of the standard deviation of the underlying base level noise by point-wise outlier rejection. Each point is compared with the current estimate of the mean and standard deviation and rejected if it is significantly different (the threshold is user-defined). This is repeated until convergence.

Each application of a boxcar filter on the DM trial changes its mean and standard deviation. Since we cannot calculate the standard deviation after each boxcar filter, because it would be too computationally expensive, we need to approximate changes in the standard deviation introduced by application of boxcar filters. Therefore, we calculate values of the standard deviation only for time-series that are decimations of the input time-series in time, i.e., we calculate the standard deviation only for boxcar filters of width that are equal to powers of two. We then interpolate values of standard deviation for all intermediate boxcar widths. An example of the results produced by this method is presented in Figure 24 where we have used SIGPROC[10] "fake" to generate input data.

4.6.3. Comparison to Heimdall Pipeline

We have compared the performance, reported S/N values, and reported widths produced by AstroAccelerate to those produced by Heimdall. Both codes' standard output provides S/N values and pulse widths. The execution time of Heimdall's SPD algorithm is measured using Heimdall's internal benchmark. To measure the execution time of the SPD algorithm within AstroAccelerate, we have used the NVIDIA visual profiler.

---

[10] https://sourceforge.net/projects/sigproc/

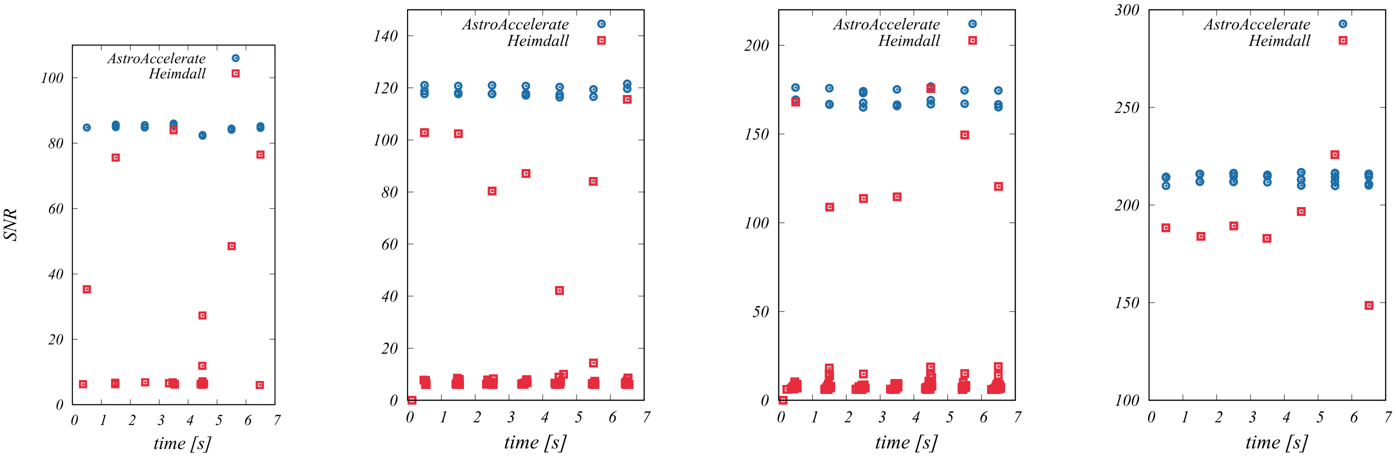

Figure 25. Comparison RS/Ns by Heimdall and Astroaccelerate in a single-pulse case. Pulses from left to right are 100, 200, 400, and 800 samples wide.

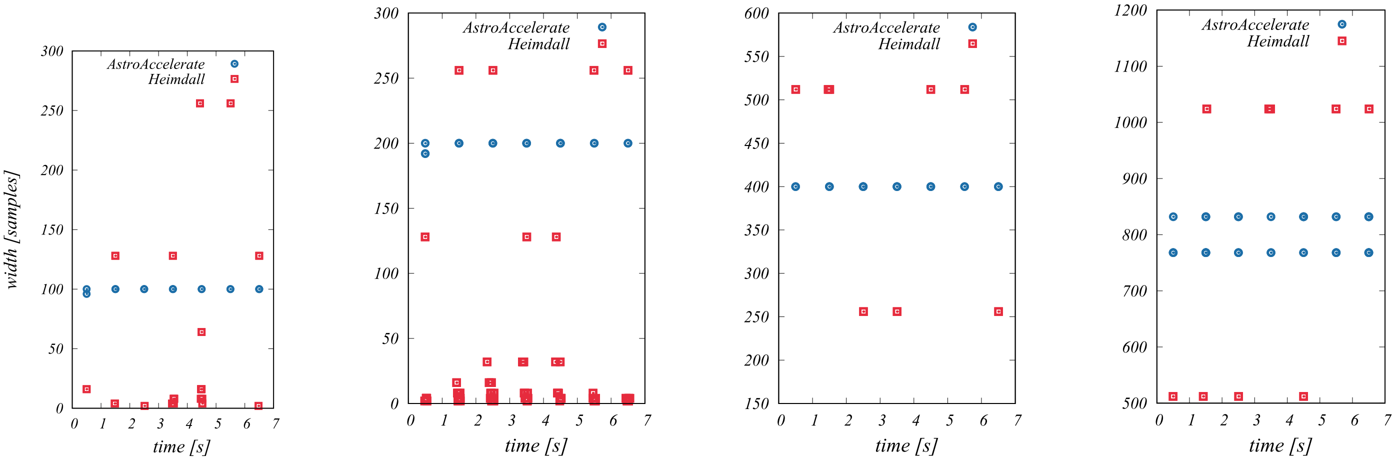

Figure 26. Comparison reported pulse width by Heimdall and Astroaccelerate in a single-pulse case. Pulses from left to right are 100, 200, 400, and 800 samples wide.

The output of the Heimdall pipeline was compared to the AstroAccelerate S/N output for two different scenarios. The first scenario focused on the detection of single pulses with fixed initial S/N and increasing pulse width. The second scenario focused on the response of both pipelines to pulses of fixed width but decreasing initial S/N of the pulses. For both scenarios, SIGPROC fake was used to generate an observation file containing a fake pulsar. AstroAccelerate's output contains many more data points per detection when compared with Heimdall's output. This is because we compare the full Heimdall pipeline with Astroaccelerate's basic candidate selection (peak-finding). In order to keep figures simple and readable, we do not display most of the points produced by AstroAccelerate, and instead, we show only the highest S/N candidates.

In the first scenario, we have compared both codes using four different pulse widths (created by changing the duty cycle parameter in SIGPROC fake) with fixed initial S/N. The SIGPROC fake file used was generated using period $P = 1000$ ms, DM $= 90$ pc cm^{-3}, S /N$_{\mathrm{init}} = 8$, and varying duty cycle. RS/N for each case is presented in Figure 25, and the reported widths of the pulses are shown in Figure 26. Our implementation delivers S/N values that have a consistent S/N value for all peaks, independent of the position of the pulse within the input data. AstroAccelerate also reports more precisely the pulse width. We see that the Heimdall SPD algorithm, which uses a sequence of boxcar filter widths that are equal to powers of two (and, hence, cannot accurately represent all pulse widths), reports varying RS/N. This is because in some cases, the boxcar filter width is shorter (longer) that the actual width of the injected pulse or the injected pulses sit in between adjacent boxcars used by Heimdall's SPD algorithm or both. This is shown in Figure 26 where Heimdall reports noticeably longer or shorter pulse widths. In both cases where the SPD algorithm does not sum all samples of the pulse (shorter boxcar filter width) or adds too much of the noise samples (longer boxcar filter width), the reported S/N will be lower than the true value. In the case where the position of the pulse in the time-series straddles adjacent boxcar filters, a similar effect to that above will occur, even if those boxcars have a width equal to the pulse.

The second scenario compares the response of the SPD algorithm to different initial S/N values of the pulsar by processing an SIGPROC fake file that contained a pulsar with parameters $P = 100$ ms, DM $= 50$ pc cm^{-3}, and a variable initial S/N. The pulse width was set to 32 samples, which is an ideal width for the SPD algorithms in both pipelines. The results are presented in Figure 27. Our implementation delivers consistent results that are similar to the results from the Heimdall pipeline. The values of RS/N for initial pulsar

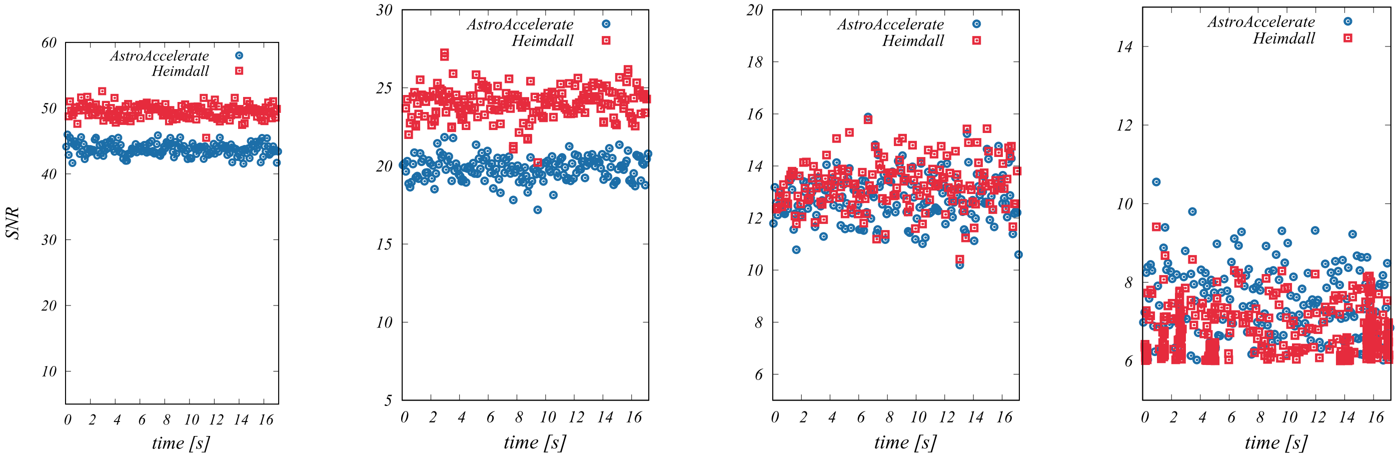

Figure 27. Comparison RS/Ns by Heimdall and Astroaccelerate in pulsar case. Initial S/N of the pulsar is from left 8, 4, 2, and 1.

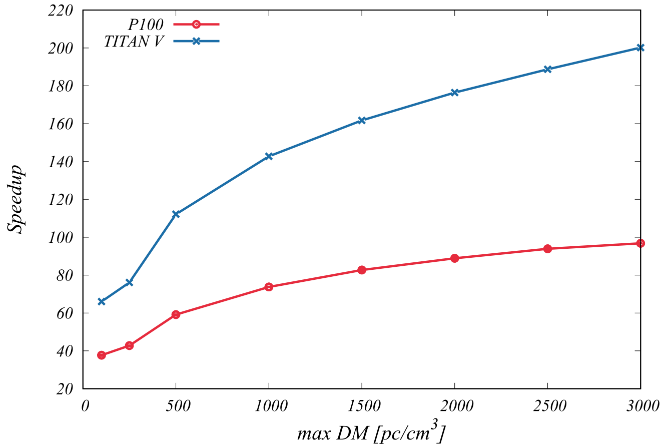

Figure 28. How performance depends on the maximum width computed by the SPD algorithm used.

$S/N_{\rm init} = 8$ and $S/N_{\rm init} = 4$ are higher for the Heimdall pipeline, but for lower initial pulsar S/N values, the RS/N reported by both pipelines are similar. The values of RS/N depend on the correct estimation of the mean and the standard deviation of the underlying noise, where the two pipelines differ. To get these results, we have used an approximation of the mean and standard deviation with outlier rejection where samples with $\sigma > 2$ were rejected.

For our performance comparison, we have used the same two NVIDIA GPU cards (P100 and Titan V). We present speed-up for different maximum DM values searched, i.e., we present the ratio of Heimdall's SPD execution time over BoxDIT execution time in configuration BD-16. Our code is faster by an order of magnitude, achieving almost 100× speed-up for P100 and almost 200× speed-up for Titan V. This is shown in Figure 28. For lower values of maximum DM searched, our speed-up is lower, because the first group of DM trials in case of AstroAccelerate is processed in full time resolution in contrast to Heimdall, which process only the first DM trial in full time resolution.

5. Conclusions

In this work, we focused on single-pulse detection using an incomplete set of boxcar filters. We have quantified properties of these SPD algorithms and quantified errors introduced by them.

We have described the signal loss of the SPD algorithm in terms of the systematic signal loss, and in terms of the worst-case signal loss, these change with pulse width and parameters of the SPD algorithm. The systematic signal loss represents the loss in signal S/N, which will occur even under best possible circumstances. This signal loss is different for different pulse widths, and it occurs because the SPD algorithm uses an incomplete set of boxcar filters. The worst-case signal loss represents the largest signal loss possible for a given pulse width and SPD algorithm. These two quantities represent upper and lower bounds on possible signal loss for a given pulse width and SPD algorithm.

We have also identified two governing parameters of the single-pulse-detection algorithm. These are the set of boxcar widths used for detection and the boxcar separation L_s, which is the distance between two neighboring boxcar filters of the same width. The systematic signal loss depends only on how dense (lower signal loss) or sparse (higher signal loss) the set of boxcar filter widths are. The worst-case signal loss depends strongly on boxcar separation L_s and weakly on density of set of the boxcar filter widths.

Based on the importance of these two parameters, we have designed two distinct single-pulse-detection algorithms. We have also presented our parallel implementation of these algorithms on many-core architectures, namely NVIDIA GPUs. We have verified the correctness of these implementation and discussed their performance.

Our first algorithm is called BoxDIT. The BoxDIT algorithm calculates boxcar filters at every point of the input time-series, which is progressively decimated. Decimation steps increase performance, but they also increase the signal loss. This algorithm is designed to be flexible and to minimize systematic signal loss Ψ_1. It is more suited for situations were high sensitivity (low signal loss) is required, where it offers good performance. This makes it optimal for processing large amounts of archival data. The disadvantage of the algorithm is that with increasing signal loss, performance does not increase accordingly.

The second algorithm is called IGRID, and it is based on a distribution of boxcar filters using a binary tree structure. This algorithm has higher performance than our BoxDIT algorithm at the cost of higher signal loss, but for low signal loss, this algorithm is slower. IGRID is best suited for real-time search scenarios where performance is critical. The disadvantage of the algorithm is that decreasing signal loss increases the memory footprint of the algorithm, which leads to lower performance.

Using IGRID in its less-sensitive configuration, we are able to process SKA-MID-like data 266× faster than real time on P100 GPU and 500× faster than real time on Titan V GPU. Even when considering the era beyond SKA of ultra-high-time-resolution radio astronomy, decreasing sampling time to nanosecond scales ($t_s = 512\,\mathrm{ns}$), the IGRID algorithm on Titan V GPU would be able to process 22000 DM trials in real time.

BoxDIT algorithm is part of the AstroAccelerate software package (Armour et al. 2019) and the AstroAccelerate with BoxDIT algorithm was used by Mickaliger et al. (2018).

This work has received support from an STFC Grant (ST/R000557/1). The authors would like to acknowledge the use of the University of Oxford Advanced Research Computing (ARC) facility in carrying out this work. The authors would also like to express thanks to Aris Karastergiou for valuable discussions and input.

Appendix
Sensitivity Analysis

Here, we present a detailed description of our derivation of sensitivity analysis for a given SPD algorithm. We use S/N as defined in Equations (3) and (4) as a figure of merit for pulse detection by a given SPD algorithm. To distinguish S/N produced by the SPD algorithm from the true S/N of the injected pulse, we use RS/N. To simplify the sensitivity analysis, we have used an idealized signal model as defined in Section 2.1, which includes the normalization of the injected rectangular pulse as given by Equation (6). Furthermore, we assume that the SPD algorithms return only the highest RS/N for a given sample; that is, if multiple boxcar filters of different widths are applied to a given sample, only the boxcar filter that has produced the highest RS/N is returned by the SPD algorithm.

Using the idealized signal, we can simplify the S/N calculation (4) and define RS/N of the boxcar filter, which acts on a rectangular pulse. When a boxcar filter encounters the pulse, it will give a value $X_L = d_s A$, where d_s is the portion of the pulse (in the number of time samples) that has been summed by the boxcar filter or *pulse coverage*. The pulse coverage is an integer value between zero and the pulse width, $0 \leqslant d_s \leqslant S$. Using the formula for S/N (4) together with white-noise approximation (5), mean ($\mu(x) = 0$), and standard deviation ($\sigma(x) = 1$) for the idealized signal, and with normalization of the pulse's amplitude $A(S)$ (6), we can write the RS/N by the boxcar filter of width L as

$$\mathrm{RS/N}(S, L, d_s) = \frac{d_s A(S)}{\sqrt{L}} = \frac{d_s C}{\sqrt{L}\sqrt{S}}. \tag{A1}$$

The RS/N by the boxcar filter in general depends on signal width S, boxcar width L, and on the portion of the pulsed covered by the boxcar filter d_s. The dependence on the pulse coverage d_s is another way of saying that the RS/N depends on the pulse's position.

A boxcar filter that does not intersect with the injected rectangular pulse will sum up to zero ($X_L = 0$ from Equation (2)); that is, $d_s = 0$, thus, $\mathrm{RS/N}(S, L, d_s) = 0$ as well.

To quantify $\mathrm{RS/N}_{\max}(S)$ and $\mathrm{RS/N}_{\min}(S)$, we first have to define the worst-case detection event and the best-case detection event.

A.1. Best-detection Case

By investigating the best-detection case and the signal loss of the SPD algorithm under the best possible circumstances, we are also investigating the minimal signal loss, which is introduced by the SPD algorithm.

The maximum $\mathrm{RS/N}((S, L)$ for a fixed S and L is when a boxcar filter covers the whole pulse or as much of the pulse as is possible if the pulse is wider than the boxcar detecting it. There are three possible configurations, depending on the size of the pulse S with respect to the size of the boxcar filter L. These are $S < L$, $S > L$, and $S = L$. This situation is summarized in Figure A1, which only shows the first two cases.

The values of pulse coverage d_s and RS/N, using Equation (A1) for all three possible cases are:

1. $S < L$ then $d_s = S$ and $\mathrm{RS/N}_{S<L} = (\sqrt{S}\,C)/\sqrt{L}$
2. $S = L$ then $d_s = S = L$ and $\mathrm{RS/N}_L = C$
3. $S > L$ then $d_s = L$ and $\mathrm{RS/N}_{S>L} = (\sqrt{L}\,C)/\sqrt{S}$.

As the SPD algorithm may consist of many different boxcar filter widths, we first determine the response $\overline{\mathrm{RS/N}_{\max}}(S; L)$ of a single boxcar filter of width L to a different pulse width S. We

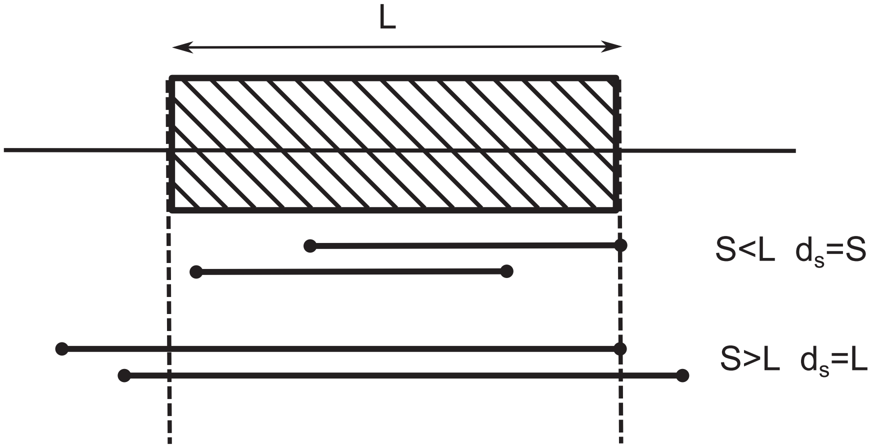

Figure A1. Best-case detection event. The pulse is shown as lines with circles at the ends, and the boxcar is shown as a hatched square.

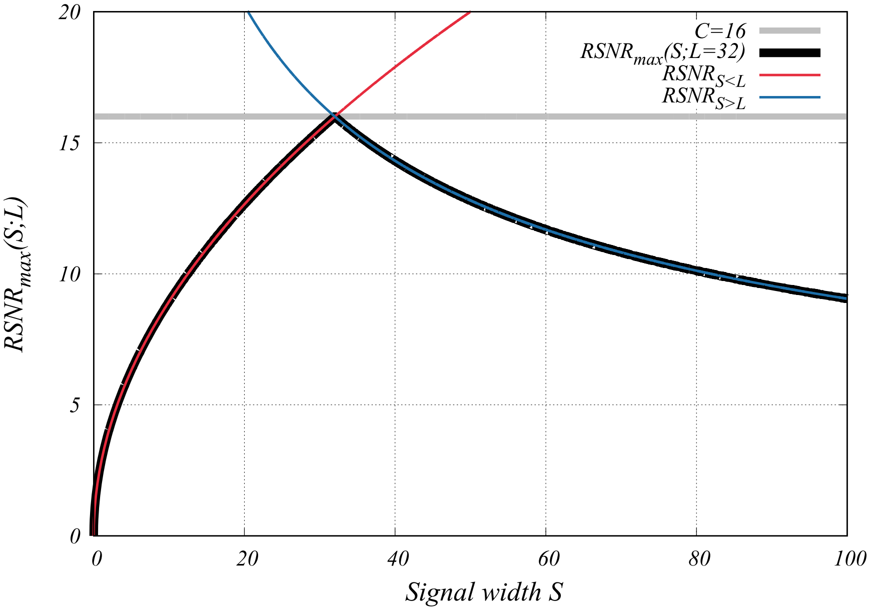

Figure A2. Response of the boxcar filter of with $L = 32$ to different pulse widths, which shows the behavior of $\overline{\mathrm{RS/N}}_{\mathrm{max}}(S;\, L)$. The peak is located at $L = S$, where the boxcar filter adds up all of the pulse power.

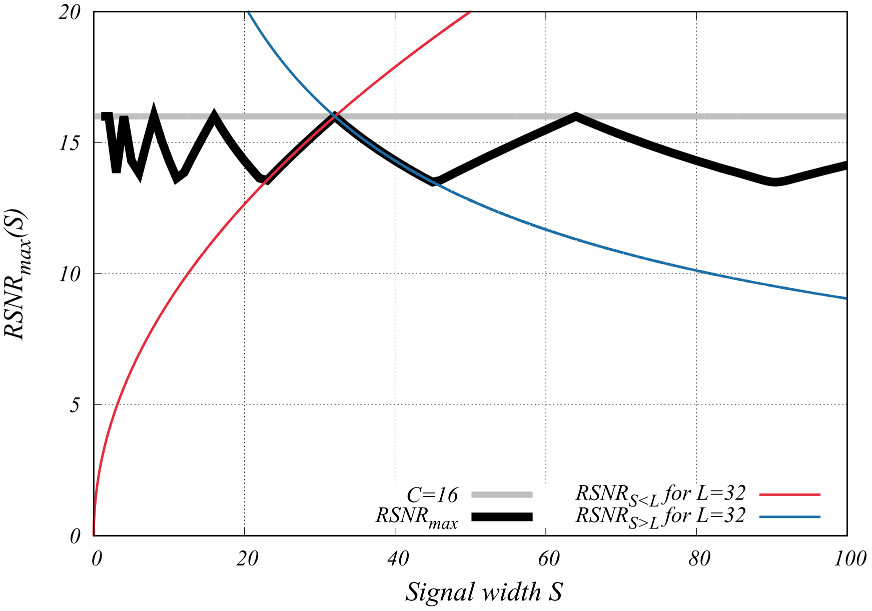

Figure A3. Response (behavior of $\mathrm{RS/N}_{\mathrm{max}}(S)$) of an SPD algorithm, where widths of boxcar filters are powers of two, that is $\mathfrak{B} = \{1, 2, \ldots, 2^t\}$, and boxcar separation is $L_s = L$. The $\overline{\mathrm{RS/N}}_{\mathrm{max}}(S;\, L)$ for $L = 32$ is emphasized to illustrate how $\mathrm{RS/N}_{\mathrm{max}}(S)$ is constructed.

define $\overline{\mathrm{RS/N}}_{\mathrm{max}}(S;\, L)$, a function of S and parameterized by L, as a combination of the cases we have discussed above, as

$$\overline{\mathrm{RS/N}}_{\mathrm{max}}(S;\, L) = \begin{cases} \frac{\sqrt{S}}{\sqrt{L}}C & : S < L \\ \frac{\sqrt{L}}{\sqrt{S}}C & : S \geqslant L. \end{cases} \tag{A2}$$

This quantity gives us the highest RS/N possible for a given boxcar of width L for any pulse of width S. The behavior of $\overline{\mathrm{RS/N}}_{\mathrm{max}}(S;\, L)$ for a fixed boxcar filter of width L is shown in Figure A2.

To find the response of the whole SPD algorithm to a pulse of width S, we need to find the maximum value of $\overline{\mathrm{RS/N}}_{\mathrm{max}}(S;\, L)$ produced by boxcar filters of every boxcar width L used by the SPD algorithm. That is:

$$\mathrm{RS/N}_{\mathrm{max}}(S) = \max_{L \in B} (\overline{\mathrm{RS/N}}_{\mathrm{max}}(S;\, L)). \tag{A3}$$

An example of the behavior of $\mathrm{RS/N}_{\mathrm{max}}(S)$ is shown in Figure A3. We see that the form of $\mathrm{RS/N}_{\mathrm{max}}(S)$ depends on the number of boxcar widths we combine together into the SPD algorithm. Thus, in order to increase (decrease) $\mathrm{RS/N}_{\mathrm{max}}(S)$, we need to add (remove) more boxcar filters of different widths to (from) set $\mathfrak{B}$.

The associated systematic signal loss Ψ_l is calculated using Equation (7). This represents minimal signal loss, which is always introduced by the SPD algorithm for given pulse width S.

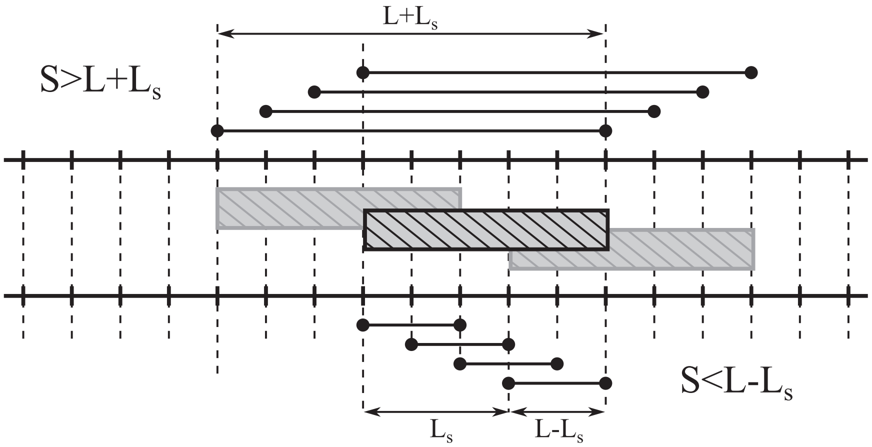

Figure A4. Pulses of width $S \leqslant L - L_s$ and $S \geqslant L + L_s$ are shown as lines ending with circles, and boxcar filters are shown as gray hatched rectangles. The pulses shift in time from one boxcar to the next. The boxcar that is always covered by the pulse for $S \leqslant L + L_s$ or always covered by the pulse width $S \geqslant L - L_s$ is emphasized in black. Parameters are $L = 5$, $L_s = 3$ with pulses of width $S = 8$ and $S = 2$, respectively.

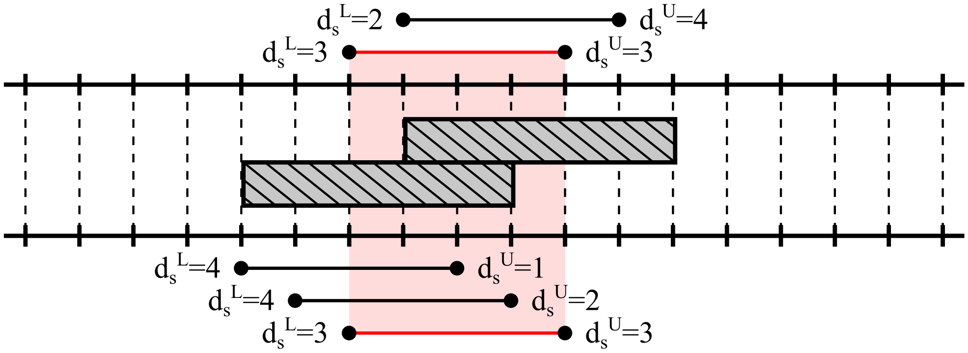

Figure A5. Pulses are shown as lines ending with circles, and boxcar filters are shown as hatched rectangles. A pulse of width $L - L_s < S < L + L_s$ starts at the beginning of the lower boxcar, and as we shift the pulse in time, it moves to the starting point of the upper boxcar. The worst case is emphasized in red. It also shows pulse coverage for both lower and upper boxcars. Parameters are $L = 5$, $L_s = 3$, and $S = 4$.

A.2. Worst-detection Case

To find the worst-detection case, we need to find a position of the pulse that minimizes $\mathrm{RS/N}(S, L, d_s)$. Let's suppose that we have a rectangular pulse of arbitrary width S, which is detected by a boxcar filter of fixed width L. Let's also suppose that the starting time of our rectangular pulse is increasing; that is, the pulse is sliding forward in time. We assume that the boxcar filters of given width are distributed evenly throughout the time-series with boxcar separation L_s. As the pulse slides forward in time, it crosses the beginning of the next boxcar filter covering the time-series every L_s steps. When the signal crosses the beginning of the next boxcar filter, the situation is the same as it was L_s steps before. This means that the only shifts that matter happen between two consecutive boxcar filters on the span of L_s samples. Thus, for the investigation of the worst-case detection, we need to consider two consecutive boxcar filters.

Depending on the pulse width, we can have three cases:

$S \geqslant L + L_s$: the pulse is wider than the extent of two consecutive boxcar filters. There is no way to shift the pulse, so the pulse would not cover at least one boxcar filter. The pulse coverage is $d_s = L$.

$S \leqslant L - L_s$: the pulse is shorter than the overlap of two consecutive boxcar filters. Wherever the pulse is, the pulse will always be covered by at least one boxcar filter with pulse coverage $d_s = S$.

$L - L_s < S < L + L_s$: the pulse lies between two consecutive boxcar filters.

The first two cases are shown in Figure A4, and the last case is shown in Figure A5.

We see that from these three cases, the worst case can only occur when $L - L_s < S < L + L_s$. Our goal is then to find the position of the pulse that minimize RS/N.

Let's denote the boxcar that starts earlier in time as the *lower boxcar* with pulse coverage d_s^L and boxcar next to it as the *upper boxcar* with d_s^U. To continue our example with the pulse shifting in time, let us inject our rectangular pulse so that its beginning coincides with the beginning of the lower boxcar. As the pulse shifts in time, the pulse coverage of the lower boxcar d_s^L will decrease while pulse coverage of the upper boxcar d_s^U will increase. Let's now suppose that the shift is continuous instead of discrete. In the continuous case, there will be a time when $d_s^L = d_s^U$; this is shown in Figure A5. If we shift the pulse in either direction, one of the pulse coverages will increase at the expense of the other. When performing single-pulse detection, we are interested only in the highest RS/N, for fixed S and L, meaning the highest pulse coverage d_s. In the worst-case scenario, we are looking for the lowest maximum produced by the SPD algorithm. This is achieved when both

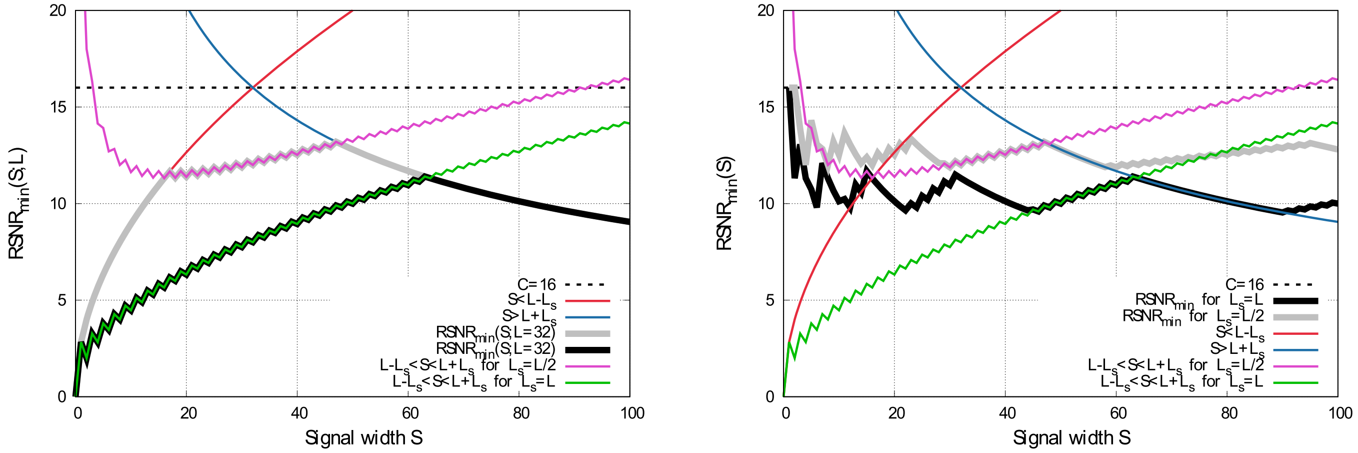

Figure A6. Left panel: the worst-case response $\overline{\mathrm{RS/N}_{\min}}(S; L)$ of the boxcar filter of width $L = 32$ for two values of $L_s = L$ (black) and $L_s = L/2$ (gray). We see that a change in L_s has a noticeable effect on the value and behavior of $\overline{\mathrm{RS/N}_{\min}}(S; L)$; thus to increase (decrease) $\overline{\mathrm{RS/N}_{\min}}(S; L)$, we need to decrease (increase) the value of L_s. Right panel: the response of the whole SPD algorithm $\mathrm{RS/N}_{\min}(S)$, where boxcar widths are powers of two $B = \{1, 2, \ldots, 2^t\}$, and $L_s = L$ is in black and $L_s = L/2$ is in gray.

the lower and upper boxcars produce the same RS/N. Therefore, the condition for the worst-case detection is

$$d_s^L = d_s^U. \tag{A4}$$

For clarity, it is useful to split the range $L - L_s < S < L + L_s$ into two further cases $L - L_s < S < L$ and $L \leqslant S < L + L_s$. For range $L - L_s < S < L$, we can express pulse coverage for both boxcars as

$$\begin{aligned} d_s^L &= S - \alpha L_s, \\ d_s^U &= L - L_s + \alpha L_s, \end{aligned} \tag{A5}$$

where $0 \leqslant \alpha \leqslant 1$ is a shift of the pulse in time. Using condition (A4) together with Equation (A5), we can get the expression for the parameter α. Substituting α into Equation (A5) will give pulse coverage

$$d_s^< = S - \left\lfloor \frac{S - L + L_s}{2} \right\rfloor, \tag{A6}$$

where ⌊ ⌋ represents the floor function that rounds the argument down to nearest lower integer. Rounding is necessary because we are working with discrete data. This is depicted in Figure A5.

For the second case, where $L \leqslant S < L + L_s$, we can express pulse coverage for both boxcars as

$$\begin{aligned} d_s^L &= L - \alpha L_s, \\ d_s^U &= S - L_s + \alpha L_s. \end{aligned} \tag{A7}$$

Solving Equation (A4) and using Equation (A7) for α and then substituting into d_s^L gives

$$d_s^> = L - \left\lfloor \frac{L + L_s - S}{2} \right\rfloor. \tag{A8}$$

We can get the same expression using d_s^U.

To summarize, we have the following cases:

1. $S \leqslant L - L_s$ then $d_s = S$ and $\mathrm{RS/N} = (\sqrt{S}C)/\sqrt{L}$
2. $L - L_s < S < L$ then $d_s^< = S - \lfloor (S - L + L_s)/2 \rfloor$, and $\mathrm{RS/N} = d_s C/\sqrt{LS}$
3. $L \leqslant S < L + L_s$ then $d_s^> = L - \lfloor (L + L_s - S)/2 \rfloor$, and $\mathrm{RS/N} = d_s C/\sqrt{LS}$
4. $S > L + L_s$ then $d_s = L$ and $\mathrm{RS/N} = (\sqrt{L}C)/\sqrt{S}$.

After the investigation into all possible cases, we can define the worst-case response of the boxcar filter of width L to a rectangular pulse of different widths $\overline{\mathrm{RS/N}_{\min}}(S; L)$ as

$$\overline{\mathrm{RS/N}_{\min}}(S; L) = \begin{cases} \frac{\sqrt{S}}{\sqrt{L}}C & : S \leqslant L - L_s \\ d_s^< \frac{C}{\sqrt{LS}} & : L - L_s < S < L \\ d_s^> \frac{C}{\sqrt{LS}} & : L \leqslant S < L + L_s \\ \frac{\sqrt{L}}{\sqrt{S}}C & : S \geqslant L + L_s \end{cases} \tag{A9}$$

where $d_s^<$ and $d_s^>$ are given by Equations (A6) and (A8).

The worst-case response of the whole SPD algorithm $\mathrm{RS/N}_{\min}(S)$ is then given by

$$\mathrm{RS}/\mathrm{N}_{\min}(S) = \max_{L \in \mathfrak{B}} (\overline{\mathrm{RS/N}_{\min}}(S; L). \tag{A10}$$

The behavior of $\overline{\mathrm{RS/N}_{\min}}(S; L)$ and $\mathrm{RS/N}_{\min}(S)$ for two different values of L_s is shown in Figure A6. We see that $\mathrm{RS/N}_{\min}(S)$ is also influenced by how many different widths of boxcar filters are used. Furthermore, we see that it is perhaps even more influenced by the distance between boxcar filters L_s.

ORCID iDs

Karel Adámek https://orcid.org/0000-0003-2797-0595
Wesley Armour https://orcid.org/0000-0003-1756-3064

References

Adámek, K., & Armour, W. 2019, in ASP Conf. Ser. 523, Astronomical Data Analysis Software and Systems XXVIII, ed. P. J. Teuben et al. (San Francisco, CA: ASP), 489

Armour, W., Adámek, K., Dimoudi, S., et al. 2019, AstroAccelerate v1.7.10, Zenodo, doi:10.5281/zenodo.2556573

Armour, W., Karastergiou, A., Giles, M., et al. 2012, in ASP Conf. Ser. 461, Astronomical Data Analysis Software and Systems XXI, ed. P. Ballester, D. Egret, & N. P. F. Lorente (San Francisco, CA: ASP), 33

Barsdell, B. R., Bailes, M., Barnes, D. G., & Fluke, C. J. 2012, MNRAS, 422, 379

Brandt, S. 2014, Analysis: Statistical and Computational Methods for Scientists and Engineers (4th ed.; Cham: Springer)

Burke-Spolaor, S., & Bailes, M. 2010, MNRAS, 402, 855

Chan, T. F., Golub, G. H., & LeVeque, R. J. 1983, Am. Stat., 37, 242

Connor, L., & van Leeuwen, J. 2018, AJ, 156, 256
Cordes, J. M., Freire, P. C. C., Lorimer, D. R., et al. 2006, ApJ, 637, 446
Cordes, J. M., & McLaughlin, M. A. 2003, ApJ, 596, 1142
Deneva, J. S., Cordes, J. M., McLaughlin, M. A., et al. 2009, ApJ, 703, 2259
Dimoudi, S., Adamek, K., Thiagaraj, P., et al. 2018, ApJS, 239, 28
Hillis, W. D., & Steele, G. L., Jr. 1986, Commun. ACM, 29, 1170
Jameson, A., & Barsdell, B. 2018, Heimdall—A Transient Detection Pipeline (La Jolla, CA: Slashdot Media), https://sourceforge.net/projects/heimdall-astro/
Karastergiou, A., Chennamangalam, J., Armour, W., et al. 2015, MNRAS, 452, 1254
Keane, E. F., Ludovici, D. A., Eatough, R. P., et al. 2010, MNRAS, 401, 1057
Keane, E. F., & Petroff, E. 2015, MNRAS, 447, 2852
Lorimer, D. R., Bailes, M., McLaughlin, M. A., Narkevic, D. J., & Crawford, F. 2007, Sci, 318, 777
McLaughlin, M. A., Lyne, A. G., Lorimer, D. R., et al. 2006, Natur, 439, 817
Mickaliger, M. B., Jankowski, F., Rajwade, K., et al. 2018, ATel, 11606, 1
Middleton, M. J., Casella, P., Gandhi, P., et al. 2017, NewAR, 79, 26
Parent, E., Kaspi, V. M., Ransom, S. M., et al. 2018, ApJ, 861, 44
Rubio-Herrera, E., Stappers, B. W., Hessels, J. W. T., & Braun, R. 2013, MNRAS, 428, 2857
Staelin, D. H. 1969, IEEEP, 57, 724
Wagstaff, K. L., Tang, B., Thompson, D. R., et al. 2016, PASP, 128 084503
Zackay, B., & Ofek, E. O. 2017, ApJ, 835, 11
Zhang, Y. G., Gajjar, V., Foster, G., et al. 2018, ApJ, 866, 149