The Design and Development of a GPU-accelerated Radar Simulator for Space Debris Monitoring

Mogamat Yaaseen Martin
Department of Electrical Engineering, University of Cape Town, South Africa

Simon Lucas Winberg
Department of Electrical Engineering, University of Cape Town, South Africa

Mohammed Yunus Abdul Gaffar
Department of Electrical Engineering, University of Cape Town, South Africa

David Macleod
Advanced Computer Engineering Lab, Centre for High Performance Computing, South Africa

ABSTRACT

The problem of space debris represents a major topic of concern in astronomy as the threat of space junk continues to grow, and the accuracy of its tracking is greatly restricted by the insufficiency and limitations of current surveillance sensors. This article presents the development of an open-source, high-performance, signal-level radar simulator to assist in modelling the detection and tracking of space debris from terrestrial radar stations, including multistatic installations where the transmitter and receiver may be separated by many kilometers. This tool is expected to aid astronomers and researchers in space situational awareness, supporting the modelling of radar interactions in this context and simulation-based exploration of radar designs for space surveillance. It makes use of an accelerated orbit propagation technique with measured two-line element datasets being used to define space debris objects. The software has been named the Space Object Astrodynamics and Radar Simulator – or SOARS – and both the transmitted and received signals generated by the application have been shown to agree with theoretical expectations. Additionally, SOARS is presently undergoing continued development, extension and optimization for heterogeneous computing platforms, enabling the use of the NVIDIA® Compute Unified Device Architecture (CUDA) interface. Results have demonstrated promising speed-ups in simulation runtimes when using the CUDA version of the application over the original sequential version, even on lower-end graphics processors. It is anticipated that the developed application will be used for the design and testing of radar sensors for space situational awareness applications, as well as for use in research, teaching and training environments.

CCS CONCEPTS

• **Software and its engineering** → Software creation and management; Designing software; Software design engineering; • **Computing methodologies** → Modeling and simulation; Simulation types and techniques; Massively parallel and high-performance simulations.

KEYWORDS

Space debris, radar simulation, orbital modelling, CUDA

ACM Reference Format:
Mogamat Yaaseen Martin, Simon Lucas Winberg, Mohammed Yunus Abdul Gaffar, and David Macleod. 2021. The Design and Development of a GPU-accelerated Radar Simulator for Space Debris Monitoring. In *2021 5th High Performance Computing and Cluster Technologies Conference (HPCCT'21), July 02–04, 2021, Qingdao, China.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3497737.3497741

HPCCT'21, July 02–04, 2021, Qingdao, China

ACM ISBN 978-1-4503-9013-2/21/07…$15.00
https://doi.org/10.1145/3497737.3497741

1 INTRODUCTION

Orbital space debris – or "space junk" – comprises artificial space objects such as spent rocket stages and inactive satellites. These are often produced as the result of in-orbit collisions, explosions, or the natural decay of objects through solar wind and other forces. Due to the increasing number of these objects in the Low Earth Orbit (LEO), there is accordingly a growing threat of valuable spacecraft potentially colliding with these debris [1]. In fact, Kessler and Cour-Palais [2] posited that the continued exponential generation of space junk could eventually create a debris "shell" in LEO, which would prevent any further space missions and operations from being conducted in this orbital space.

LEO is currently at the greatest risk of over-population due to space debris [3], and recent trends have already demonstrated the predicted near-exponential growth in space junk numbers. The implications of this are already being experienced, as discussed in [4] – which summarizes the extreme increase in catalogued Resident Space Objects (RSOs) since 1956.

For this to be remedied, space situational awareness methods are employed using advanced sensing instrumentation to detect, measure and track space debris. However, the sheer size of the growing debris population – particularly smaller objects that are harder to track – poses a significant challenge. Typically, debris populations are monitored by highly sensitive sensors such as terrestrial or space-based radars as well as optical and laser instruments [5, 6]. In particular, radar offers many advantages over optical and laser systems for these kinds of purposes, such as being able to operate during the day or at night under various conditions, operating using a wider beam even at very large ranges, and providing improved orbit determination for space-based targets [7]. The primary aim of this paper is therefore to design and develop a complete space

debris and radar simulator that could be used to design and test radar sensors for space monitoring applications.

Additionally, the program could be used by researchers for conceptual pre-studies or simulation-based investigations for proposed installations, through which they may ascertain what could happen in certain situations under various conditions. This would allow studies and experiments to be conducted even in situations where the required infrastructure does not exist, and this could be naturally extended to teaching or training environments. Theoretically, this software could also be extended for planning satellite evasive maneuvers and spacecraft launches, as the program is intended to simulate large populations of space debris and predict their propagation paths over extended periods.

In 2013, similar work was done by McCall [8], who proposed using thermal modelling for characterizing space debris objects via infrared signals, and specially aimed to focus on the simulation of debris in LEO. However, this focused solely on the thermal model of the debris and thus did not priorities the detection, tracking or imagine of such objects through radar. More recent related work was conducted by Kastinen *et al.* [9] in 2019, which presented the development of a radar simulator to be used for monitoring space debris through a tracklet approach. However, this specifically served to estimate the sizes of debris populations through tracking and was thus not flexible enough for the objectives in this work.

Presently, no simulator exists for use in designing space-oriented radar systems. This paper aims to remedy this through the development of a radar simulator to be used in designing space surveillance radars on an international scale. In particular, it is desired that such a simulator should simulate the raw return signal that would be received by a physical radar receiver, factoring in the system's design and its surrounding propagation environment. This structure agrees with the definition of a *signal-level* radar simulator, which allows users to process the resulting return signal(s) for the purposes of target detection, imaging and tracking. An example of detection processing is shown in [10]. Signal-level simulators are vital for various applications, including pulsed systems testing [11], Synthetic Aperture Radar (SAR) techniques [12], and for modelling specific systems such as the MARSIS radar [13].

Previously, three basic signal-level simulators were developed by Golda [14], Lengenfelder [11], and Brooker [15] at the University of Cape Town in 1997, 1998, and 2008 respectively. The most capable of these is Brooker's [15] Flexible Extensible Radar Simulator (FERS) – an open-source signal-level simulator used to design and test various general radar configurations. This represents a viable, modern candidate for use as a software basis in this work, which aims to modify FERS towards a more space-oriented approach with built-in support models for astronomy applications.

The baseline version of the developed software has been aptly named the Space Object Astrodynamics and Radar Simulator (SOARS). The work in this paper introduces the following original contributions with respect to the established literature:

- The developed software represents a *signal-level* radar simulator that is specialised towards space monitoring and includes a built-in model for environmental noise as well as dynamic propagation for target orbits.
- A software implementation of a galactic and sky noise model has been developed for use in radar simulations.

This paper thus aims to document the development of SOARS as an open-source radar simulator as well as its use for space-oriented experiments and system design and testing.

2 BACKGROUND

As it is desired that SOARS should be able to accurately model space objects and their movements, it is important to understand the dynamics involved in orbital propagation. This section therefore summarizes some of the key ideas from the literature surrounding the field of orbital dynamics and space monitoring.

The motion of orbiting space objects is often described by the Keplerian coordinate system, which defines both the position and velocity of RSOs. This framing system makes use of six classical Keplerian elements, namely the semi-major axis a, eccentricity e, right-ascension of the ascending node Ω, orbital inclination i, argument of the perigee ω, and the true anomaly ν. Together, these describe the space object's orbit size, shape, and orientation relative to the Earth's equator, as well as the observed object's position and movement within the orbit. The relationships between these Keplerians are illustrated in Figure 1 (adapted from [16]), depicting a typical RSO's elliptical orbit relative to the Earth.

These Keplerian elements are stored inside Two-Line Element (TLE) datasets, constructed from raw measured data after minor corrections; most notably, certain periodic variations are removed from the data for simplicity. As such, it is critical that these periodic variations are carefully reconstructed when using the TLE sets in practice. Many existing space-monitoring radars make use of TLE sets to predict when RSOs could cross the radar's beam. One example of this is the European Incoherent SCATter facility at Troms Φ, which was previously used to compare measured object data against the theoretical propagation prediction from a TLE dataset in [17].

TLE sets are typically used to propagate the objects they represent along their extrapolated orbits. This is often achieved using Simplified General Perturbations #4 (SGP4) theory [18] – an analytical orbit propagation scheme that predicts the effect of dynamic perturbations on a space object's orbit due to the Earth's shape, drag, and radiation, as well as due to gravitational forces from the sun and the moon [19]. Since these are only predictions, the use of TLE sets can typically result in semi-major axis position errors up to a few kilometers per day of propagation.

A comparative study by Shuster [20] demonstrated the speed of SGP4 over alternative methods on a Central Processing Unit (CPU). In LEO, SGP4 was shown to operate over 500 times faster than both the Runge-Kutta and Nyström-Lear numerical integrators – with the same position accuracy. The study thus concluded that SGP4 is more reliable in its accuracy than most alternatives. This makes it perfectly suited for use in this work.

The SGP4 propagation model generates RSO ephemerides in a reference frame defined as the True Equator, Mean Equinox (TEME) frame [21] – a special type of the Earth-Centered Inertial (ECI) frame classification. This is useful for representing objects in space, as the frame does not rotate with the Earth's surface. However,

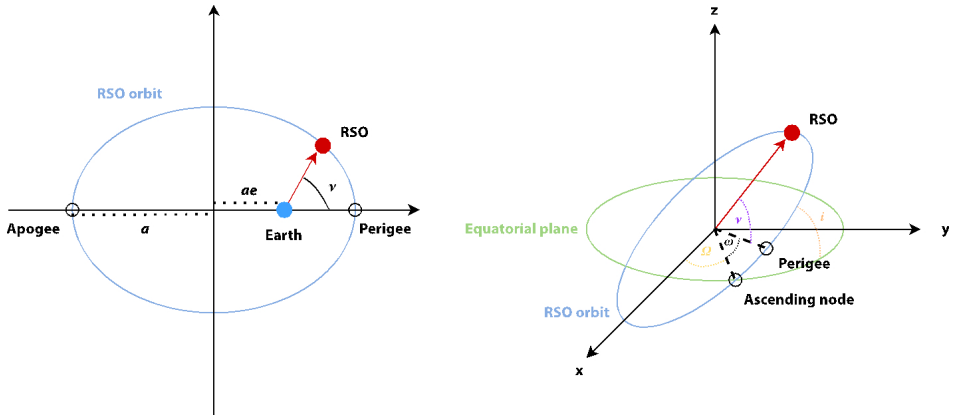

Figure 1: An elliptical Kepler orbit showing the classical elements.

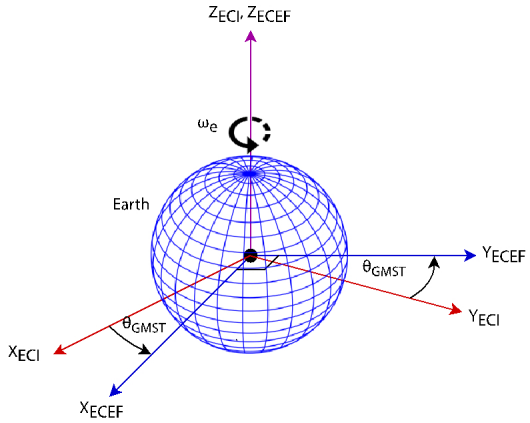

Figure 2: Flowchart illustrating the integration of CUDA into the SGP4 algorithm within SOARS.

the ECI frame presents significant problems when also considering terrestrial objects, such as ground-based radars.

Ideally, a radar's coordinates should be fixed relative to other objects in a simulation environment. This is to ensure that target ranges and Doppler shifts can be reliably predicted and computed. Ideally, a radar's coordinates should be fixed relative to other objects in a simulation environment. This is to ensure that target ranges and Doppler shifts can be reliably predicted and computed. Terrestrial objects are thus best represented in the Earth-Centered, Earth-Fixed (ECEF) frame – a coordinate system that remains fixed with the Earth's surface during its rotation. This would thus serve as the most useful reference frame for use in SOARS, as all TEME coordinates could be converted directly into ECEF coordinates following the SGP4 processing. Figure 2 shows the relationship between the ECI frame and the ECEF frame, as adapted from [22].

This illustrates the rotation angle θ_{GMST} (measured in radians) and the rate of the Earth's rotation about its axis ω_e (measured in radians per second). Here, θ_{GMST} is defined as the Greenwich Mean Sidereal Time angle derived from the Julian date at the RSO's epoch. This represents the angle through which the ECEF frame is rotated about the z-axis, as measured from the initial x-axis direction in the ECI frame. These quantities are thus vital in the transformation from ECI to ECEF.

Due to inherent limitations in any radar system's observation angles and field of view, as well as the obstruction of the Earth itself, there is currently no single space surveillance sensor that is able to monitor every object in LEO. For the surveillance of space junk to be improved, a more global approach is required – one that is not centralized in the Northern Hemisphere but is instead spread throughout various locations on an international scale [23].

This shows that there is a lack of space-oriented sensors currently stationed in the Southern Hemisphere. This gives rise to a need for more space-oriented instrumentation below the equator. With the development of MeerKAT and the SKA [24], Agaba [25] proposed a design for a bistatic, space-oriented radar to assist with this problem. In theory, this type of system could provide the desired global surveillance required to observe more of the debris population. These types of systems could thus serve to help populate international databases for space objects, making the testing of such systems through simulation significantly useful.

3 METHODOLOGY

The first step in designing the simulator was to review the relevant literature associated with space debris, Keplerian orbital dynamics and radar simulation, as covered in the previous sections. The next step was thus to develop a design plan and define a methodology through which the software design could be approached. It is thus important to determine the desired operation of the SOARS program and how it could be used to address the problems described in the previous sections.

It is desired that the SOARS software should operate using a single simulation definition file as the main input, specifying all simulation properties; the simulator should then output the raw received signal(s) computed at each simulated receiver. This is in keeping with the signal-level radar simulator classification and will enable users to post-process the output signal(s) in various ways. The accessibility of the application should also be improved via a Graphical User Interface (GUI) for SOARS. This should allow users

to design and output simulations into a definition file using a user-friendly application window, as well as to visualize their simulated environments through an interactive 3-D rendering.

Due to the inherent limitations of commodity computers, it is also desired that Graphics Processing Unit (GPU) acceleration should be used via the NVIDIA® Compute Unified Device Architecture (CUDA) model [26]. This is expected to provide noticeable performance improvements to some modules of the software once the sequential versions have been tested, but also to futureproof the program for future expansion; namely, SOARS is expected to be extended through the implementation of a ray-tracing algorithm using NVIDIA®'s OptiX™ [27] in a future iteration of SOARS.

With these objectives in mind, the primary aims of this research can be broken down into the following sequence of sub-objectives:

- Explore and review the relevant literature associated with space junk, orbital modelling, radar simulation and high-performance computing to better understand the space debris problem and its challenges.
- Design and develop the radar simulator with appropriate hardware models, system properties, an orbit propagation scheme, and noise considerations. A user interface should also be developed to improve usability and accessibility.
- Integrate CUDA into the program for GPU acceleration and futureproofing purposes (namely for the inclusion of a ray-tracing algorithm in the next iteration of SOARS). Any CUDA speed-ups should be benchmarked.
- Verify the operation of the software by comparing output data against established radar theory and performing software benchmarks. These verification tests should namely cover the new additions to the existing programs, as will be detailed later in this section.

This paper thus serves to document the development and testing of the baseline SOARS software, as well as identifying the capabilities and limitations of the present build of the application while additional features are still in development. The version of SOARS explored in this paper is thus limited so as to establish a working system, but it will later be improved through the implementation of a ray-tracing algorithm using NVIDIA®'s OptiX™ software [27]. This is expected to improve simulation accuracy, but benchmarks will need to be conducted to assess the potential trade-off between accuracy and speed.

4 DESIGN AND DEVELOPMENT

After conducted a review of the relevant literature, it was decided that the development of a signal-level radar simulator would be appropriate to meet this work's objectives. The design of this simulator has progressed incrementally using the spiral development model [28], and design tools were specifically chosen to meet the desired software capabilities – as described by a list of user requirements for this software.

4.1 Software Requirements

The establishment of the user requirements was focused on after the literature review. This was separated into non-functional and functional requirements. The non-functional requirements focused on portability, speed, accuracy, usability, and modularity – and prioritizing aspects of reliable and extensibility of the program.

Regarding functional requirements, the baseline version of SOARS needed to satisfy these technical aspects:

- **Measured datasets**: The software should natively support measured datasets as inputs for maximum accuracy in target representations, such as TLE sets (described further in [29]).
- **Keplerian dynamics**: The software should consider the effects of orbital dynamics and perturbations as modelled by TLE sets and the well-documented SGP4 theory.
- **Radar systems theory**: The software should accurately adhere to established radar theory, such as the modelling of object Radar Cross Section (RCS), path loss, gain patterns, noise considerations, and Doppler computation.
- **Noise sources**: The simulator should account for the effects of internal thermal system noise as well as external sources of noise such as natural and man-made phenomena in the simulation environment.

With these software requirements, the expected capabilities of the software have been fully defined. Thusly, the development tools and design criteria required to fully realize SOARS can be selected.

4.2 Design Considerations

Based on the software requirements and the discussions in previous sections, the FERS software was selected as the simulator upon which the design of SOARS would be based. This is because FERS is coded in standardised C++ – an efficient, compiled programming language supported on a wide variety of operating systems with support for CUDA and a variety of fast built-in algorithms and data structures. For these reasons, the use of C++ provides the portability and efficient performance as specified by the first two non-functional user requirements. FERS has also been verified against established theory and validated against measured data, as documented in [15] for SAR processing and pulse-Doppler testing.

Additionally, FERS is already a well-established signal-level radar simulator that offers support for the pulse-Doppler radar scheme – which SOARS is also anticipated to use – and is capable of executing simulations comprised of multiple targets and multistatic radar systems. The software has also been extensively used for designing, testing and training purposes since its inception, and it features built-in models for propagation loss, Doppler, phase shifts, pulse waveforms, accurate clock timings, and various antenna gain patterns. FERS is thus a logical choice for use in this work, providing the foundation for all the required features of a standard radar simulator. Maintaining and expanding upon the existing FERS also addresses the third functional software requirement by accounting for all the necessary radar systems theory.

However, FERS also has limitations that could be improved upon; for instance, FERS acts as a generalized radar simulator, whereas SOARS should be specialised towards space monitoring. FERS also does not include built-in models for environmental noise sources and dynamic movement for targets – instead, FERS only models internal noise and uses hard-coded target coordinates at predefined time instances. SOARS should thus make use of measured TLE sets to model targets in simulation and propagate them dynamically over a defined time period.

Achieving this in the base FERS software would be significantly slow since each TLE target would need to be propagated manually by the user and in a serial fashion, and then translated into Cartesian coordinates and written into a FERS input file; instead, SOARS will use CUDA with an integrated SGP4 algorithm to do all the TLE propagation independently. This integration of CUDA will also futureproof the application for when ray tracing is added using OptiX™ in the near future.

Based on these design decisions, the following libraries constitute the external dependencies that are required to compile SOARS:

- **HDF5**: The Hierarchical Data Format version 5 (HDF5) library [30] enables reading and writing of the popular HDF5 data storage format. This is used in SOARS to store simulated received signals in their in-phase and quadrature forms via an efficient, portable storage structure.
- **Boost**: A formally reviewed C++ package providing a set of efficient functions for use alongside native libraries [31]. This grants access to multiple additional features in SOARS, including the use of CPU threading.
- **TinyXML/PugiXML**: FERS makes use of the TinyXML [32] package to read and write simulation scripts stored in Extensible Markup Language (XML) files, whereas SOARS makes use of the PugiXML library [33] for improved efficiency. This is further discussed later.
- **FFTW3**: The FFTW3 library [34] provides access to an efficient implementation of the discrete Fourier transform in the SOARS C++ code.
- **Python**: Enables the use of the Python programming language [35] for optional functionalities within FERS, such as importing antenna gain patterns from Python files.
- **SGP4**: An official C++ implementation of the SGP4 model developed by Vallado et al. [21]. This is used to propagate the orbits of TLE targets within the SOARS framework.
- **CMake**: An open-source tool [36] used to build SOARS and link its source code against the required libraries; also supports compilation of CUDA code through the NVCC compiler [37], as developed by NVIDIA®.

These dependencies are all freely available as open-source software and are widely used in C++ development; many of them have been chosen due to their native use in FERS, while additional packages have been selected for implementation specifically in SOARS.

4.3 Simulator Structure

The SOARS software has largely been based upon the structure of FERS, which was designed to be modular in nature and thus extensible for various purposes. SOARS aims to follow a similar design approach, whereby the code should be easily readable through comments and adaptable for possible future expansion. It is also desired that the structure of SOARS should be made clear to the end user to encourage further development and iteration.

Based on the software requirements defined earlier, the block diagram for the structure of the SOARS software is presented in Figure 3. This depicts SOARS as being comprised of five core software elements, namely the environment model, the signal renderer, the ray-tracing simulator, the SGP4 algorithm, and the CUDA integration. The ray-tracing simulator is currently being worked upon using the OptiX™ engine, which natively requires CUDA integration. It is expected that this will form part of a near-future iteration of the SOARS program upon completion.

As illustrated in Figure 3, the SOARS software application requires an input file in a specific XML format – a SOARSXML file – comprehensively describing the full simulation environment with all systems, antennas, signals, and targets. These inputs are read into the software using an XML parser (PugiXML), stored in memory, and then passed into an environment model. This environment model is responsible for processing the full radar simulation as well as the radar equation computations, after which the signal renderer outputs the raw return signal in the efficient HDF5 file format.

Each HDF5 output file is generated to contain the raw return signal samples for one simulated radar receiver, stored in their original in-phase and quadrature form. Results from the environment model are then produced separately in additional XML files, documenting the receiver response data for every receiver in the environment.

4.4 Noise Implementation

Each receiver in FERS simulates a low-noise amplifier to model the hardware between the antenna and down-mixer. This generates thermal noise that is picked up in the received signal due to the resistance of the amplifier. This component is thus responsible for the effect of internal system noise on the signal, where the noise power generated from this – assuming a resistance of $R_n = 1\ \Omega$ – is represented by a series voltage source with magnitude V_n.

The system's noise power P_n is thus expressed as:

$$P_n = kT_sB \tag{1}$$

where k is Boltzmann's constant, B is the system bandwidth, and T_s is the system noise temperature. This is then used to generate white Gaussian noise.

Aside from system thermal noise, the SOARS simulator should also account for *external* noise sources such as galactic noise and the Cosmic Microwave Background (CMB) [38]. Such environmental noise sources are often estimated as non-linear functions of frequency, such as noise from atmospheric gases, the Earth's surface, and the Sun. These sources are often defined by a brightness temperature, which relates to the actual effective antenna temperature, T_a, through the convolution of the antenna pattern and the brightness temperature of the sky and ground [39].

The data in [39] depicts the frequency-dependent antenna noise temperature curves for carrier wave frequencies between 100 MHz and 100 GHz, where F_a is defined as:

$$F_a = 10\log(f_a) \tag{2}$$

and f_a is the external noise factor. This is related to the effective antenna temperature, T_a, through the equation:

$$f_a = \frac{T_a}{T_0} \tag{3}$$

with T_0 taken as the reference temperature of 290 K.

When designing radio links, it is often useful to account for the worst-case scenario in terms of received noise and interference. As a result of this, SOARS is designed to only account for the effects of noise contributions from nearby man-made sources, galactic noise, variable noise emitted due to quiet Sun conditions, atmospheric sky noise, and black body radiation from the CMB.

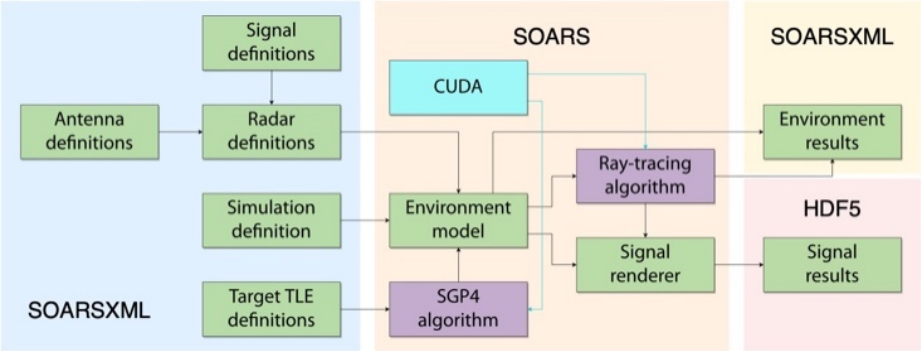

Figure 3: Block diagram showing the overall software structure and information flow of the SOARS software.

The aforementioned external noise additions are implemented in SOARS using linear interpolation between approximated points along each temperature curve; the user's input carrier frequency is then to be used to find the exact interpolated noise temperatures at the specific frequency. The user should be allowed to disable any or all of these external noise sources if they believe the sources to be irrelevant to their simulation experiment. Overall, this noise implementation serves as a new addition to the existing FERS program and accounts for many of the most common radioastronomy noise sources. This also addresses the fifth functional software requirement by accounting for the modelling of internal and external noise sources.

4.5 Orbit Propagation

Targets in FERS need to be manually defined by the user and then propagated through various position waypoints, which also need to be defined by the user at specific time steps. SOARS, however, aims to automate this process and dynamically propagate targets. This is achieved by using TLE datasets to define targets, and then employing an appropriate propagation algorithm to compute the target's time-varying position based on its orbital properties.

Based on the previous discussions on Keplerian dynamics, the SGP4 model was selected as the propagation scheme to be used in SOARS, while the TEME reference frame is to be used to define target coordinates relative to fixed radar system positions. However, the desired use case for SOARS is to simulate extraordinarily large numbers of targets simultaneously. It would thus be very computationally expensive to serially compute the SGP4-propagated paths of thousands of targets, particularly when the paths are computationally independent.

As a result, it was decided that CUDA would be used in SOARS, providing GPU-based parallelism on NVIDIA® graphics hardware. This interface facilitates the communication between a host (i.e., a CPU) and a device (i.e., a GPU), making use of kernel functions that can run concurrently across multiple CUDA threads on the device, resulting in a high throughput for many large-scale use cases.

However, for small simulation sizes, the use of CUDA may actually reduce program efficiency due to the overhead required in its setup. It is thus important to consider this trade-off when designing a program around this interface, and GPU acceleration should only be employed in situations where large numbers of independent mathematical operations are being computed. Benchmarking should therefore always be used to assess the potential trade-offs in using a CUDA-based software instead of its serial equivalent. Such results are shown later in the paper.

In SOARS, CUDA is intended to be used for two express purposes, namely:

- accelerating the ray-tracing algorithm used during computation of the radar equation (part of future work)
- parallelizing the SGP4 algorithm when propagating target RSOs along their orbits

The first purpose involves using the GPU to maximize throughput of the ray-tracing mechanism. Since ray tracing is an inherently parallel technique, it is well suited for acceleration across thousands of CUDA threads - with each thread computing the path of a single radar ray. This ties in well with the radar equation computation currently used in FERS (and hence also SOARS), which computes the return power of a received signal and is highly dependent on target range, RCS, and other factors. This is to be implemented using the OptiX™ engine in a future iteration of SOARS.

The second purpose for CUDA in this work relates to the parallelization of Vallado's [40] SGP4 software, providing an additional boost to performance while also futureproofing for the coming ray-tracing model. The desired functionality of the SGP4 library is clear: the software needs to convert TLE sets into ECEF vectors of the form (x, y, z) at time t. The implementation of this algorithm with CUDA is reflected in Figure 4.

As depicted in Figure 4, the process works as follows:

- A TLE set is parsed from the SOARSXML file and is processed to produce a satrec structure and create a new target in the simulation environment.
- The satrec structure is then stored in an array on the host side. This is repeated for every TLE set serially, and a fully populated satrec array if formed.
- A time array is produced on the host, which contains the time t for various instances. This accounts for every sample between the simulation start and end in increments of t_i, the target sampling interval.

This concludes the orbit propagation procedure and enables further processing to be done on the target position arrays.

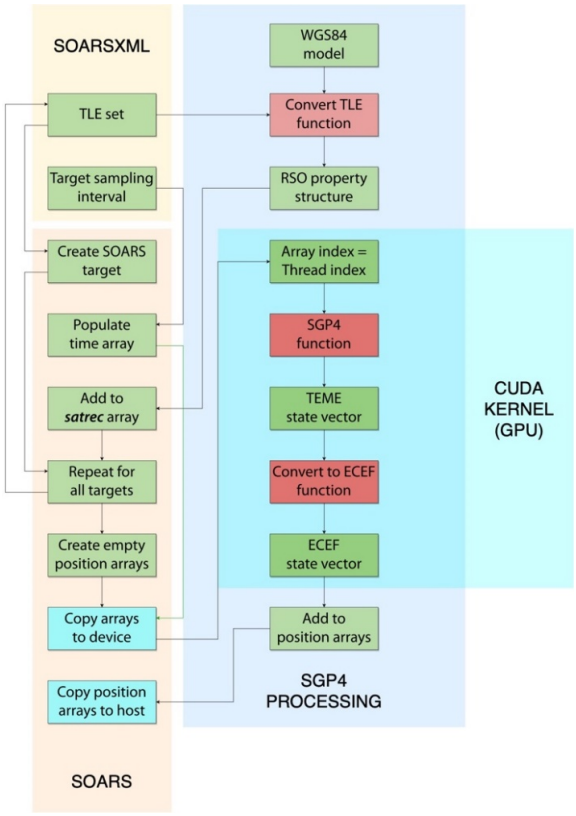

Figure 4: Flowchart illustrating the integration of CUDA into the SGP4 algorithm within SOARS.

4.6 Radar Computation

The original FERS application accounts for most of the basic radar computations to be expected in any basic radar simulator, but improvements can be made to some of these aspects to increase their accuracy and efficiency.

In both FERS and SOARS, the effect of propagation attenuation occurring between a radar and a target is inherently computed by the multistatic radar equation. This is defined as shown in Equation 4 [41, 42] for the q_{th} receiver (up to Q receivers) the j_{th} transmitter (up to $\mathcal{J}$ transmitters):

$$P_{R_q} = \frac{P_{T_j} G_{T_j} G_{R_q} \lambda^2 \sigma_{RCS}}{(4\pi)^3 R_{T_j}^2 R_{R_q}^2} \tag{4}$$

where P_R and P_T are the powers of the received and transmitted signals respectively (measured in Watts), G_R and G_T are the respective gains of the receive and transmit antennas, σ_{RCS} is the RCS of the target (measured in square meters), λ is the wavelength of the radar, and R_R and R_T represent the respective ranges of the receive and transmit antennas from the target. This equation is thus used to compute the return power of the received signal after the effects of attenuation, RCS, and signal-target interactions have been factored into the transmit signal.

In the case of a monostatic radar, this is simplified to the equation:

$$P_{R_q} = \frac{P_{T_j} G_{T_j} \lambda^2 \sigma_{RCS}}{(4\pi)^3 R_T^4} \tag{5}$$

as the transmitter and receiver are collocated and thus $G_R = G_T$ and $R_R = R_T$. In this setup, there is only one antenna available and it will thus toggle between transmit and receive modes. After transmitting the pulse, the antenna switches to receive mode and waits for return echoes for a specified period of time; once the next PRI starts, the process repeats. It is thus pertinent that the echoes are not received while the antenna is still in transmit mode.

In the case of FERS and SOARS, this is avoided using a windowing method to skip specified lengths of the received signal. This is essentially treated as dead time that is specified as a fraction of the Pulse Repetition Interval (PRI) to allow the radar to transmit a pulse and then safely switch to receive mode. In this way, the simulator avoids immediate, potentially dangerous feedback that could be picked up at the receiver. Antennas in SOARS will thus only operate in receive mode during the window's "on" period – usually taken as the remainder of the PRI.

It is also important to consider the time delay τ in receiving the observed signal after initial transmission, computed as follows:

$$\tau = \frac{R_T + R_R}{c} \tag{6}$$

where c is the speed of light, taken as 299,792,458 m/s. This accounts for the round-trip time between a signal being transmitted, propagating towards a target, and then propagating from the target to a receiver.

Another computation to consider is that of the Doppler frequency f_D. Before this can be computed, it is important to consider the geometry of the simulated bistatic radar configuration. This is reflected in Figure 5, as adapted from [43].

Using the information highlighted in this geometry setup, the bistatic Doppler shift can be related to the velocity of a target V (as projected onto the bistatic plane) using Equation 7 [44].

$$f_D = \frac{2V \cos(\delta) \cos\left(\frac{\beta}{2}\right)}{\lambda} \tag{7}$$

where f_D corresponds to the Doppler shift, L is the bistatic baseline, β is the bisecting angle between the R_T and R_R vectors, and δ is the angle measured from the bistatic bisector to the target propagation vector. Alternatively, Equation 8 [45] can be used to relate the Doppler shift to a radial velocity component V_r (i.e., the projection of V onto the initial radar-to-target line-of-sight vectors):

$$f_D = f_c \left(\frac{1 + \frac{V_r}{c}}{1 - \frac{V_r}{c}} - 1 \right) \tag{8}$$

In a three-dimensional space, it is vital that a target velocity vector V_{tgt} (computed as the change in target coordinates over time) is first projected onto the bistatic plane as vector V. Only after this projection should V be projected onto the transmitter and receiver line-of-sight vectors. Both Equations 7 and 8 represent more accurate implementations of the Doppler than the version currently found in FERS; as such, both of these are to be tested and compared within SOARS.

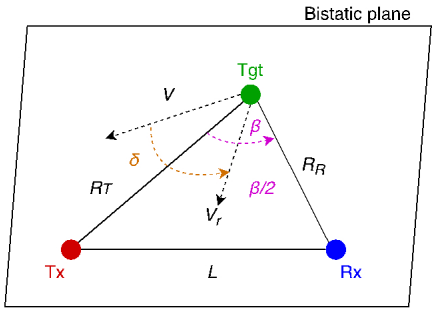

Figure 5: Bistatic radar geometry showing the bistatic plane, a transmitter Tx, receiver Rx, and target Tgt.

In this paper, it is worth noting all results are computed using a parabolic gain pattern for each antenna in the simulations. This is implemented using the following equations, as per [46]:

$$X_p = \left(\frac{\pi d}{\lambda}\right) \sin\left(\theta_{OB}\right) \tag{9}$$

$$G_{max} = \left(\frac{\pi d}{\lambda}\right)^2 \tag{10}$$

$$G\left(\theta_{OB}\right) = \underset{\max\left(\frac{2J_1(X_p)}{X_p}\right)}{G^2} \tag{11}$$

where G_{max} the maximum possible gain of the antenna, G is the overall antenna gain as a function of the off-boresight angle θ_{OB}, and J_1 is the Bessel function of the first kind of order one. The variable $X_p$is an interim quantity used in the calculation of the overall antenna gain G and is dependent on θ_{OB}. This angle corresponds to the angle measured between the vector from the antenna to a target and the vector from the antenna to its beam center. This is computed directly in SOARS.

With all of the aforementioned equations defined, the return signal is described by Equation 12 in SOARS.

$$y_q\left[n\right] = \sum_{j=0}^{J}\left(\frac{A_{jq}n\, d_{jq}\left[n\right]}{f_s} + \sum_{p=0}^{P}\frac{A_{jpq}n\, d_{jpq}\left[n\right]}{f_s}\right) \tag{12}$$

where q is the receiver index, j is the transmitter index, p is the target index, n is the sample number, f_s is the sampling rate, J is the number of transmitters in the simulation, and P represents the total number of targets being simulated. A_{jq} and d_{jq} correspond to the amplitude function and delayed samples for the direct path from a transmitter j to a receiver q, respectively. Thus, A_{jpq} and d_{jpq} correspond to the amplitude function and delayed samples for the path measured from j to target p to q, respectively. Further detail is provided in [47].

5 RESULTS

Having detailed the design and development of the application, this section aims to investigate the capabilities and limitations inherent in the baseline version of the developed software. This includes testing the software against the intended implementations detailed earlier, verifying its operation against established theory, and benchmarking the application in some aspects of its performance.

Since the operation of FERS has already been verified against radar systems theory and validated against real-world radar data in [15], this section aims to focus on results produced by the new features implemented in SOARS. Namely, this section details the comparison of simulated received signals against expected results based on the equations defined previously, benchmarks for the SGP4 algorithm and the developed CUDA alternative, as well as runtime improvements to file parsing.

5.1 Simulator Verification

The previous section covered the detailed design and development of the SOARS application, explaining in particular the underlying radar simulation approach and models that were implemented. This section presents the testing and analysis of some of the results from this baseline version of SOARS.

All simulation results were verified using the theoretical space-monitoring radar designed in a study by Agaba [25] in 2017. This design proposed using the MeerKAT radio telescope [24] as the receiver in a bistatic radar configuration. This simulation represents a short unit test performed using the SOARS simulator for verification purposes, where many of the parameters were chosen for convenience and the simulation was fully designed using the aforementioned GUI designed for SOARS.

The tested simulation makes use of a bistatic radar and a target that was propagated through nine manual positions at time-steps between t = 0 ms and t = 80 ms in intervals of 10 ms. The target was initially placed at the approximate coordinates (x, y, z) = (5208049.329, 1999182.830, -3485881.148) m such that it transited close enough to both the transmit and receive antennas, which were placed at (x, y, z) = (4923738.121, 1821610.792, -3622037.530) m and (x, y, z) = (5103916.070, 2004714.383, -3257572.688) m respectively. This ensured that the target position would not introduce any range ambiguities.

All noise sources were also disabled for these initial results (including the 20 K amplifier noise temperature), but these will be enabled in a later experiment to verify the noise implementation. The full list of the simulation parameters is shown in Table 1. Note that the receiver position vector used in this experiment corresponds to the nominal array centre position of the full 64-antenna MeerKAT configuration.

Upon executing this simulation in SOARS, the target's orbit is dynamically propagated using SGP4 while the radar antennas are created at the designated position coordinates. Thereafter, pulses are spawned and propagated from the transmitter, through the simulation environment, and towards the TLE-based target. These pulses then illuminate and reflect off the target, after which they are collected at the receiving antenna. The signal information is then computed using the equations defined earlier in the paper, such as time and phase delays, Doppler, and signal power. The results are then written to the SOARS output files after the process has been repeated for every target and each transmitter-receiver pair.

For this small unit-test simulation, the transmit and receive signals are illustrated in Figure 6 after post-processing the results and plotting their measured power against time. This depicts both the full transmitted signal and the raw received signal, where the latter signal is stored via its samples in a HDF5 file and separated into its

Table 1: Simulation parameters used to test verification aspects of SOARS

Parameter	Value	Units
Number of pulses	12	-
Carrier frequency	1.35	GHz
Sampling frequency	25	MHz
Transmit power	10	kW
Radar bandwidth	10	MHz
Pulse width	1	ms
Pulse Repetition Interval	6.667	ms
Receive window length	4.667	ms
Receive skipping period	2	ms
System noise temperature	20	K
Parabolic transmit antenna diameter	10	m
Parabolic receive antenna diameter	13.5	M
Target RCS	0.0001	m^2

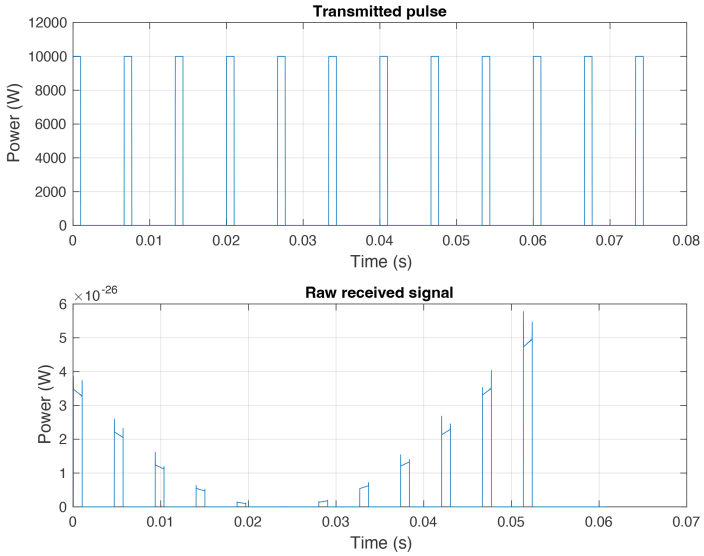

Figure 6: Transmitted and received signals simulated in SOARS using the MeerKAT radar.

in-phase and quadrature components – along with a scaling factor that needs to be applied before plotting the results. These two signal components are each measured in Volts, and in assuming an ideal resistance of 1 Ω, the power is computed by squaring the signal magnitude.

It should be noted that this received signal only shows the signal samples observed at the receiver while it is in operation, i.e., the receive window skip period is not inherent in the recorded HDF5 file, hence why the signal ends before 80 ms. This plot was obtained by plotting the raw signal against sample number and then scaling the horizontal axis by the reciprocal of the sampling frequency.

When applying a matched filter to the received signal and enlarging the first pulse in each signal, the result in Figure 7 is obtained.

As shown in Figure 7, the transmit signal has a power of 10 kW, a pulse width of 1 ms, as expected. As for the first received pulse, the receive skip period of 2 ms needs to be accounted for and thus the return echo is seen to be received starting at 0.04408 ms + 2 ms = 2.04408 ms. This agrees with the result calculated based on Equation 6 (computed as 2.04409 ms based on the propagation distance of $R_T + R_R = 612.802$ km). The matched-filtered signal also demonstrates the near-exact same time delay at 0.04404 ms. Additionally, the width of the return pulse is calculated to be exactly 1.00028 ms, demonstrating a slight change in the transmitted 1 ms pulse width due to the falling edge of the pulse illuminating the target at a slightly different position than the leading edge.

Finally, the received signal's power can be computed using the monostatic radar equation presented in Equation 5. This requires knowledge of the overall transmitter and receiver gains, each of which is calculated using Equation 11. This yields transmitter and receiver gains of $G_T = 1.198$ and $G_R = 9.644$ respectively. Applying Equation 5, P_R is thus computed as 3.481 x 10^{-26} W, closely agreeing with the power result of 3.480 x 10^{-26} W depicted in the first receive

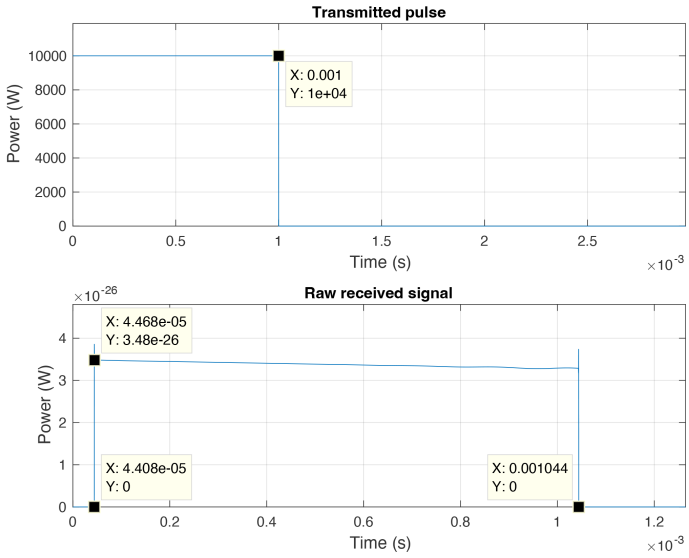

Figure 7: Enlarged view of the first transmit and receive pulses using the MeerKAT radar design in SOARS.

pulse in Figure 7. This confirms that both the transmitted and received signals are being generated as expected, verifying that the implementation of power and range computation.

Another quantity to be tested is that of target Doppler, i.e., verifying the Doppler/velocity information computed in SOARS for a SPG4-propagated target. For this to be tested, the target's velocity needs to be computed by considering the change in its coordinates over a specified time period. This velocity vector must then be projected onto the bistatic plane, which is formed using the initial coordinates of the transmitter, receiver and target – denoted Tx, Rx, and Tgt respectively.

The projected velocity component – denoted V in Equation 7 – can then be used to calculate the Doppler shift once the angles β and δ are determined; however, SOARS instead uses Equation 8 to get the Doppler result. This represents a more accurate method compared to that of FERS, which could produce unreliable Doppler frequencies in some situations. The implementations in Equations 7 and 8 have thus been tested to compare their accuracy. This experiment used the same target position set-up as before, with the target being propagated over a 1 ms interval from its initial position to the coordinates $(x,\ y,\ z)$ = (5208045.021, 1999183.745, -3485875.509) m. This yields a velocity vector V_{tgt} with a magnitude of 7155.231 m/s, which is then projected onto the bistatic plane as V.

This is approximated in the geometry shown in Figure 8.

This depicts the bistatic bisector, the object coordinates, angles δ and β, the target velocity and its projection, and the bistatic plane. Using the bistatic plane, the angle β is calculated as 91.553 deg. The bisector is simply found by normalizing and adding the line-of-sight vectors, i.e., the bisector is the resultant of the vector sum of the range vectors from the target to the transmitter and receiver. Determining V and δ, however, requires normalising the normal vector of the bistatic plane and then projecting the final coordinates of the target onto this plane. Taking the vector subtraction

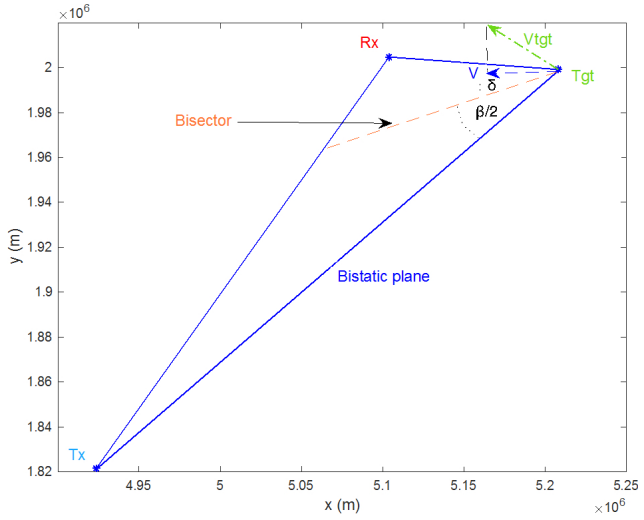

Figure 8: Approximation of the geometry in the tested bistatic configuration.

of the initial target position vector from its final position vector, and then dividing it by the propagation time of 1 ms, yields the projected velocity vector V. Along with this, the angle δ can finally be computed as the angle between the bisector and V.

This is determined as 37.554°, leading to a calculated Doppler of 34903.930 Hz using Equation 7; the corresponding output from Equation 8 was found to be 34904.382 Hz. Based on the theory of special relativity, the latter result was found to be slightly more accurate than Equation 7 based on the extensive derivation in [45], ultimately accounting for a greater number of minor Doppler contributions. It was for this reason that Equation 8 was chosen for

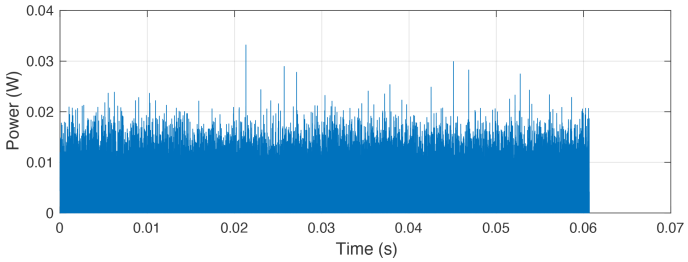

Figure 9: Matched-filtered return signal observed by the MeerKAT radar when all noise sources are enabled.

implementation in SOARS, as opposed to the more equation used for Doppler computation performed in FERS.

One final verification experiment is to test the noise implementation. When accounting for the noise sources described in [39], including all external and internal sources, the result in Figure 9 is obtained after matched-filtering the received signal.

This demonstrates the matched-filtered return signal for the same simulation depicted previously – but with all noise sources enabled. This accounts for the internal amplifier temperature of 20 K, the CMB, and all other aforementioned noise sources for a carrier frequency of 1.35 GHz. The result shows that the noise peaks far higher than the original target return shown in Figure 6, showing that the target echo is drowned out in the interference. This demonstrates the importance of carefully selecting the noise and system design parameters, as well as how feint the target returns from space debris will typically be relative to environmental interference.

With this, the simulator has been fully tested from the simulation definition files to the final output results. The results have shown that both the simulated transmit and receive signals have met theoretical expectations, and thus the operation of the simulator has been verified against the theory. The raw received signal can thus be reliably used for post-processing purposes, such as for target tracking or detection [10], signal-to-noise ratio comparisons, et cetera.

Realistically, SOARS is designed for much larger-scale, dynamic simulations with upwards of 100,000 targets being simulated at once. In general, the program is capable of being used for multiple antennas with large fields of view, multiple noise sources, and a huge number of space objects. The unit tests conducted in this section have thus only served to highlight some of the verification methods used to ensure that the simulator has been accurately implemented against established theory.

5.2 Orbit Propagation Performance

With SOARS having been verified against established radar theory, this subsection aims to consider the performance of the adapted SGP4 model with the added CUDA support.

The target propagation and file parsing performance of the simulator were tested using a full simulation with the parameters as presented in Table 2. Most of these parameters were selected for convenience to showcase the operation of SOARS in its simplest form – using a monostatic radar with a single pulse, a single target, and generic parameters values offering predictable results for a typical space-oriented radar experiment. Some parameters were selected such that benchmarking could be more easily conducted.

The simulation makes use of a monostatic radar configuration and a single TLE-based target that was propagated through nine time-steps between $t = 0$ ms and $t = 80$ ms in intervals of 10 ms. The transmit pulse is defined by a linear chirp with a bandwidth of 10 MHz and a width of 1 ms. The range to the target was computed as 463.405 km at $t = 0$ ms. All noise sources were disabled for simplicity.

Using the CUDA SGP4 algorithm (based on the original serial code developed and verified in [40]), every target at every time instance can be propagated simultaneously using SGP4 theory and CUDA; however, there are drawbacks to this approach. The most notable drawbacks are the additional memory requirements for the arrays on the host side (which could become quite enormous for large numbers of targets or long simulation times), as well as the additional overhead introduced into the software's runtime by using CUDA. The latter entails overhead from having to initialize the GPU, using the CUDA compiler NVCC [36], and the process of copying data to and from the device.

It is thus important to consider the possibility that the use of CUDA may actually *degrade* code performance as opposed to accelerating it as desired. Runtime benchmarking tests were thus conducted for this purpose, pitting the serial version of SGP4 in SOARS against its CUDA alternative. This entailed conducting several ordinary SOARS simulations with a varying number of targets (ranging from 1 to 1,000,000) – each of which was propagated through nine time-steps – and then measuring the time taken to process every target via SGP4. This was done for both the serial and CUDA versions of the code.

All benchmarking tests were conducted on a heterogeneous compute node equipped with 132 GB of high-speed memory, an Intel®Xeon®E5-2695 v3 CPU clocked at 2.30 GHz (sporting 28 CPU cores), and a NVIDIA® Tesla V100 GPU [48]. The V100 GPU sports 32 GB of GPU memory and 5,120 CUDA cores. The results of these benchmarks are illustrated in Figure 10, showing a ratio of the serial software's runtime to the CUDA version's runtime for a varying number of propagated targets. All runtimes were initially measured in ms.

From the results depicted in Figure 10, it is clear that there is an advantage to using the CUDA version for large-scale simulations. Based on the data, the CUDA version of SGP4 executes faster than the serial version for large simulation sizes and requires nearly the same amount of processing time regardless of the number of

Table 2: Simulation parameters used to test performance aspects of SOARS

Parameter	Value	Units
Number of pulses	8	-
Carrier frequency	1	GHz
Sampling frequency	25	MHz
Transmit power	10	kW
Radar bandwidth	10	MHz
Pulse width	1	ms
Pulse Repetition Interval	10	ms
Receive window length	8	ms
Receive skipping period	2	ms
System noise temperature	0	K
Parabolic antenna diameter	13.5	m
Target RCS	0.0001	m^2

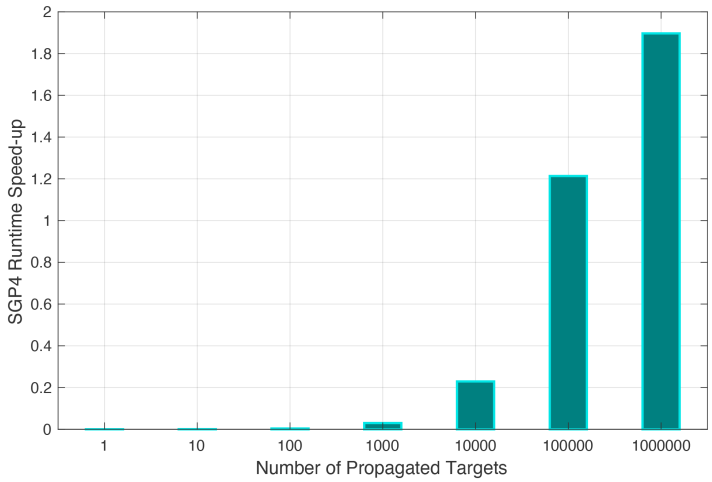

Figure 10: Speed-up in processing times for propagating a varying number of targets using SGP4 functions serially and with CUDA.

targets – up until around 10,000 targets. Thereafter, the results scale differently based on the size of the simulation. This is due to the larger array sizes that require copying to and from the device, which increases the runtime non-linearly.

For both the cases of 100,000 and 1,000,000 targets, the CUDA version demonstrates faster runtimes compared to the serial version of the software. In cases where fewer targets are used, the serial version of the software provides improved speeds over the CUDA version. This is attributed to the large processing overhead and memory access patterns required for CUDA usage. Overall, the results demonstrate that the simulator is well suited towards large-scale use, as per the software requirements. Additionally, the main purpose of CUDA in SOARS is for futureproofing its use with OptiX^{TM} – a program that requires CUDA integration. The speed-ups demonstrated in this subsection are thus an additional benefit to the software on top of the futureproofing provided by CUDA.

5.3 File Processing Performance

This subsection aims to consider the performance changes provided by the file processing methods used to read and write inputs and outputs in SOARS.

In particular, while FERS was originally developed to use the TinyXML package for reading and writing XML files, modern implementations of XML parsing mechanisms have offered significant speed-ups over this library. It was for this reason that PugiXML was chosen to replace TinyXML for file processing. To verify any potential performance improvements, both parsers were benchmarked via timing tests observing their read/write speeds for SOARS inputs and outputs.

Using the same hardware and simulation setup described earlier, the results of the file read/write tests are demonstrated in Figure 11, illustrating the speed-up provided by PugiXML over TinyXML as a ratio of their performance, i.e., TinyXML's runtime to PugiXML's runtime. All runtimes were initially measured in ms.

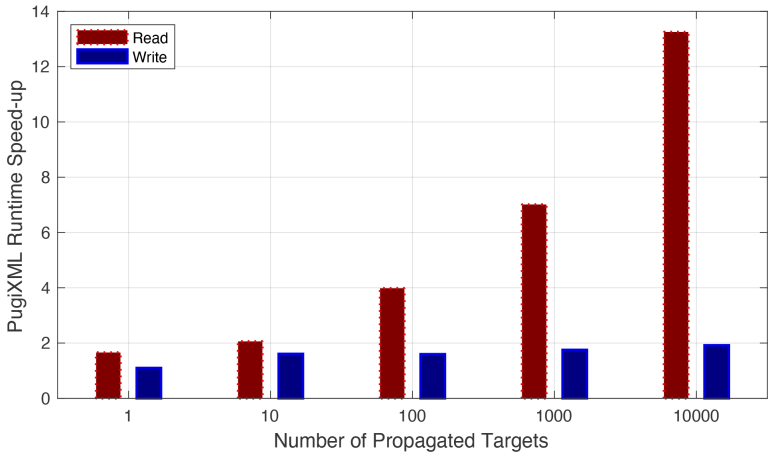

Figure 11: Speed-up in read/write times using PugiXML over TinyXML for SOARSXML simulation files with a varying number of targets.

Each parser was tested using the parameters in Table 2 with a varying number of targets, each of which is a duplicated TLE set propagated through nine time-steps. The recorded timings in Figure 11 correspond to the periods of time taken to fully parse a complete SOARSXML file into the SOARS software, illustrating how the use of PugiXML provided significant performance improvements over TinyXML for almost every simulation tested – both in terms of read and write speeds. The speed-ups in reading times show that PugiXML parsed the input file at up to 13x the speed of TinyXML. The speed-ups in writing times also show that PugiXML was up to 1.9x faster than TinyXML.

Another consideration is the difference in output values generated by the serial and CUDA versions of SGP4, as there are slight discrepancies in the computed values of the target position coordinates produced by each method. These are attributed to the unavoidable precision differences resulting from floating-point calculations, which are typically computed slightly differently on a CPU compared to a GPU.

This is particularly apparent with CUDA, where functions that are compiled for the GPU make use of NVIDIA®'s CUDA math library, while functions compiled for the CPU use the host compiler's math library [49]. These libraries will often produce slightly different output values due to the different implementations of mathematics functions built into the libraries.

6 CONCLUSIONS

This paper has highlighted the threat of the growing space debris population and the potential damage it may cause to existing and future spacecraft. This represents a major topic of concern, particularly with the insufficiency and limitations of available space-monitoring sensors. This paper has thus presented the design and development of a baseline version of SOARS – an open-source, signal-level radar simulator intended to be used in the design and testing of space surveillance radar systems to improve global coverage of the space debris population. It is anticipated that SOARS could be used by researchers for conceptual investigations and systems design, as well as in teaching or training environments.

The developed program serves as an extension and combination of a newly developed software model for radioastronomy noise sources, the existing FERS application, the established SGP4 propagation model, and NVIDIA® CUDA. Improvements have been made to the original FERS software and SGP4 model, including the improvement of the former's Doppler accuracy and the use of GPU-based parallelism for more efficient target processing. Additionally, this work has added a new noise model implementation to account for galactic, sky, manmade and internal noise, the integration of CUDA and SGP4 into the simulator, and the introduction of a more modern XML file parser, PugiXML. A list of software requirements, both functional and non-functional, has also been presented and met by the developed software.

The results have shown that both the transmitted and received signals, as generated by SOARS, agree with theoretical expectations. This includes the computed target range and the received signal's pulse width, time delay, and power amplitude. This served to verify the operation of the simulator in basic radar computation. The introduction of CUDA into the SGP4 algorithm has also been presented, where the serial version of SGP4 showed software runtimes scaling almost linearly in proportion to the number of targets in the simulation. However, for 100,000 targets or more, the CUDA version of the software proved faster than the serial version – making the CUDA version ideal for large-scale use.

The accuracy of the simulator will need to be further investigated by comparing the raw simulated return against measured radar data from a space surveillance experiment. As part of future work, the application is also expected to be augmented with a ray-tracing model through the OptiX™ software. This should improve simulation accuracy and allow for more complex space object properties to be simulated, such as size, shape, and reflectivity parameters. This represents the primary reason for the use of CUDA in SOARS, while also providing the advantage of speed-ups to the SGP4 algorithm for large simulations. However, benchmarking tests will

need to be conducted to assess the trade-offs between the theoretical improvements in accuracy and the changes in computational efficiency brought about by OptiX™.

An extensible baseline version of SOARS has thus been established through this paper, which was planned around facilitating further expansion of the application. In particular, the provision of the aforementioned additions should result in a feature-complete iteration of the software which should be completed in the future.

ACKNOWLEDGMENTS

This work was supported through funding by the Council for Scientific and Industrial Research (CSIR) in South Africa. All results in this paper were made possible through the use of GPU computing hardware provided by the Centre for High Performance Computing – a subsidiary of the CSIR.

REFERENCES

[1] Jer-Chyi Liou. 2016. Orbital Debris Challenges for Space Operations. In *Second International Civil Aviation Conference (ICAO)*. NASA, 1–12.
[2] Donald J Kessler and Burton G Cour-Palais. 1978. Collision frequency of artificial satellites: The creation of a debris belt. *Journal of Geophysical Research: Space Physics* 83, A6 (1978), 2637–2646.
[3] Heiner Klinkrad. 2006. *Space debris: Models and Risk Analysis*. Springer Science & Business Media.
[4] National Aeronautics and Space Administration. 2020. Orbital Debris Quarterly News. Retrieved February 1, 2021 from https://www.orbitaldebris.jsc.nasa.gov/quarterly-news/pdfs/odqnv24i1.pdf.
[5] John A Kennewell and Ba-Ngu Vo. 2013. An overview of space situational awareness. In *FUSION*. 1029–1036.
[6] Osman Kalden and Christian Bodemann. 2011. Building space situational awareness capability. In *2011 5th International Conference on Recent Advances in Space Technologies (RAST)*. IEEE, 650–654.
[7] Ian Ritchie. 2013. Remote control southern hemisphere SSA observatory. In *Proceedings of the 18th international workshop on laser ranging, Fujiyoshida, Japan*. 1–9.
[8] Paul D McCall. 2013. *Modeling, simulation, and characterization of space debris in low-Earth orbit*. Ph.D. Dissertation. Florida International University.
[9] D Kastinen, Juha Vierinen, Johan Kero, S Hesselbach, Tom Grydeland, and Holger Krag. 2019. Next-generation Space Object Radar Tracking Simulator: SORTS++. In *1st NEO and Debris Detection Conference*. European Space Agency, 1–8.
[10] D Cutajar, A Magro, J Borg, K Adami, G Bianchi, G Pupillo, A Mattana, G Naldi, C Bortolotti, F Perini, *et al.* 2020. PyBIRALES: A Radar Data Processing Backend for the Real-Time Detection of Space Debris. *Journal of Astronomical Instrumentation*, Vol. 9, No. 1.
[11] Rolf Lengenfelder. 1998. *The design and implementation of a radar simulator*. Master's thesis. University of Cape Town.
[12] Giorgio Franceschetti, Maurizio Migliaccio, Daniele Riccio, and Gilda Schirinzi. 1992. SARAS: a synthetic aperture radar (SAR) raw signal simulator. *IEEE Transactions on Geoscience and Remote Sensing* 30, 1 (1992), 110–123.
[13] J-F Nouvel, A Herique, W Kofman, and A Safaeinili. 2003. Marsis radar signal simulation. In *IGARSS 2003. 2003 IEEE International Geoscience and Remote Sensing Symposium. Proceedings (IEEE Cat. No. 03CH37477)*, Vol. 4. IEEE, 2756–2758.
[14] Peter John Golda. 1997. *Software simulation of synthetic aperture radar*. Master's thesis. University of Cape Town.
[15] Marc Brooker. 2008. *The design and implementation of a simulator for multistatic radar systems*. Ph.D. Dissertation. University of Cape Town.
[16] Jill Tombasco. 2011. *Orbit Estimation of Geosynchronous Objects Via Ground-Based and Space-Based Optical Tracking*. Ph.D. Dissertation. University of Colorado.
[17] Markus Landgraf, R Jehn, and W Flury. 2004. Comparison of EISCAT radar data on space debris with model predictions by the master model of ESA. *Advances in Space Research* 34, 5 (2004), 872–877.
[18] David Vallado and Paul Crawford. 2008. SGP4 orbit determination. In *AIAA/AAS Astrodynamics Specialist Conference and Exhibit*. 6770.
[19] Felix R Hoots, Ronald L Roehrich, and TS Kelso. 1980. Spacetrack report no. 3. *Project Spacetrack Reports, Office of Astrodynamics, Aerospace Defense Center, ADC/DO6, Peterson AFB, CO* 80914 (1980), 14.
[20] Simon P Shuster. 2017. *A Survey and Performance Analysis of Orbit Propagators for LEO, GEO, and Highly Elliptical Orbits*. Master's thesis. Utah State University.
[21] David A Vallado, Paul Crawford, Richard Hujsak, and TS Kelso. 2006. Revisiting spacetrack report #3: Rev 2. In *AIAA/AAS Astrodynamics Specialist Conference and Exhibit*. 6753.
[22] Morten Breivik. 2003. *Nonlinear maneuvering control of underactuated ships*. Master's thesis. Norwegian University of Science and Technology.
[23] Glen Shepherd. 2018. Space Surveillance Network. Retrieved September 21, 2020 from http://climateviewer.org/img/gallery/ssa2shepherd.pdf.
[24] SKA Africa. 2017. SKA SA Project. Science & Technology Parliamentary Portfolio Committee. Retrieved September 10, 2021 from http://www.ska.ac.za/wp-content/uploads/2017/06/presentation parliament 2017.pdf.
[25] Doreen Agaba. 2017. *System design of the MeerKAT L-band 3D radar for monitoring near earth objects*. Ph.D. Dissertation. University of Cape Town.
[26] David Kirk *et al.* 2007. NVIDIA® CUDA software and GPU parallel computing architecture. In *ISMM*, Vol. 7. 103–104.
[27] Steven G Parker, James Bigler, Andreas Dietrich, Heiko Friedrich, Jared Hoberock, David Luebke, David McAllister, Morgan McGuire, Keith Morley, Austin Robison, *et al.* 2010. OptiX™: a general purpose ray tracing engine. In *ACM transactions on graphics (tog)*, Vol. 29. ACM, 66.
[28] Barry W Boehm. 1988. A spiral model of software development and enhancement. *Computer* 21, 5 (1988), 61–72.
[29] Emilian-Ionu Croitoru and Gheorghe Oancea. 2016. Satellite Tracking Using NORAD Two-Line Element Set Format. *Scientific Research and Education in the Air Force-AFASES*, Vol. 1, 423–431.
[30] Mike Folk, Gerd Heber, Quincey Koziol, Elena Pourmal, and Dana Robinson. 2011. An overview of the HDF5 technology suite and its applications. In *Proceedings of the EDBT/ICDT 2011 Workshop on Array Databases*. ACM, 36–47.
[31] Boris Schäling. 2011. *The Boost C++ libraries*.
[32] Lee Thomason. 2015. TinyXML. Retrieved February 16, 2021 from http://www.grinninglizard.com/tinyxml/.
[33] Arseny Kapoulkine. 2019. A light-weight C++ XML processing library. Retrieved June 27, 2021 from http://pugixml.org.
[34] Matteo Frigo and Steven G Johnson. 2005. The design and implementation of FFTW3. *Proc. IEEE* 93, 2 (2005), 216–231.
[35] Python™ Software Foundation. 2019. Python™. Retrieved September 10, 2020 from https://www.python.org/.
[36] Ken Martin and Bill Hoffman. 2010. *Mastering CMake: a cross-platform build system*. Kitware.
[37] Vinod Grover and Yuan Lin. 2012. Compiling CUDA and other languages for GPUs. In *GPU Technology Conference (GTC)*. 1–59.
[38] George Smoot. 2007. Cosmic Microwave Background Radiation anisotropies: their discovery and utilization. *Bulletin of the American Physical Society* 52 (2007).
[39] ITU. 2016. Recommendation ITU-R P. 372-13. Radio Noise. *ITU-R P Series, Radiowave Propagation* (2016).
[40] David Vallado. 2018. Astrodynamics Software. Retrieved May 21, 2020 from https://celestrak.com/software/vallado-sw.php.
[41] K Milne. 1977. Principles and concepts of multistatic surveillance radars. *Radar-77*. 46–52.
[42] Victor S Chernyak. 1998. Fundamentals of multisite radar systems: multistatic radars and multistatic radar systems. CRC press.
[43] Terje Johnsen and Karl E Olsen. 2006. *Bi-and multistatic radar*. Technical Report. NORWEGIAN DEFENCE RESEARCH ESTABLISHMENT KJELLER.
[44] Nicholas J Willis. 2005. *Bistatic radar*. Vol. 2. SciTech Publishing.
[45] Mark A Richards, James A Scheer, William A Holm, Brent Beckley, *et al.* 2010. *Principles of Modern Radar - Volume 1: Basic Principles*. SciTech Publishing.
[46] Robert M Gagliardi. 2012. *Satellite communications*. Springer Science & Business Media.
[47] Marc Brooker and Michael Inggs. 2011. A signal level simulator for multistatic and netted radar systems. *IEEE Trans. Aerospace Electron. Systems* 47, 1 (2011), 178–186.
[48] NVIDIA® Corporation. 2018. NVIDIA® Tesla® V100 GPU Accelerator. Retrieved February 01, 2021 from https://images.nvidia.com/content/technologies/volta/pdf/tesla-volta-v100-datasheet-letter-fnl-web.pdf.
[49] Nathan Whitehead and Alex Fit-Florea. 2011. Precision & performance: Floating point and IEEE 754 compliance for NVIDIA® GPUs. *rn (A+ B)* 21, 1 (2011), 18749–19424.