Algorithm Optimization and Parallelization of PRESTO Single Pulse Search: Postprint

Authors: Fu Zhiming, Chen Zonghao, Liang Nan, Yu Xuhong, Guangsheng Shao, Zhang Bin, Xie Xiaoyao

Date: 2023-01-17T00:00:00+00:00

Abstract

Since the discovery of aperiodic celestial objects such as Fast Radio Bursts (FRBs) and Rotating Radio Transients (RRATs), single-pulse search has garnered widespread attention among researchers. Concurrently, with the continual refinement of radio telescope facilities, the volume of observational data generated by higher resolutions and broader observational ranges has increased dramatically, making accelerated data processing imperative. This paper presents the single-pulse search methodology in PRESTO (PulsaR Exploration and Search Toolkit), optimizes the detrending algorithm in single-pulse search using the Cython programming approach, and achieves parallelization of single-pulse search on Central Processing Units (CPUs) through the Ray framework. Experimental results demonstrate that the parallelized single-pulse search following algorithm optimization significantly improves search program performance and substantially reduces data processing time. Furthermore, this parallelization strategy is CPU-only, enabling high-performance data processing in pure CPU environments without requiring code modifications.

Full Text

Algorithm Optimization and Parallelization of PRESTO Single Pulse Search

Fu Zhiming[1], Chen Zonghao[1], Liang Nan[1], Yu Xuhong[1], Shao Guangsheng[2], Zhang Bin[1], Xie Xiaoyao[1]*

[1]Guizhou Provincial Key Laboratory of Information and Computing Science, Guizhou Normal University, Guiyang 550001, China
[2]Library of Guizhou Normal University, Guizhou Normal University, Guiyang 550001, China

Abstract: Since the discovery of celestial objects without obvious periodicity such as Fast Radio Bursts (FRBs) and Rotating Radio Transients (RRATs), single pulse search has attracted extensive attention from researchers. Meanwhile, with the continuous improvement of radio telescope equipment, the observation data generated by higher resolution and wider observation space has increased dramatically, making it urgent to accelerate the processing of observation data. This paper introduces the single pulse search in PRESTO (PulsaR Exploration and Search Toolkit), optimizes the detrending algorithm in single pulse search using Cython programming, and achieves parallelization of single pulse search on Central Processing Units (CPUs) through the Ray framework. Experimental results show that the parallelized single pulse search with optimized algorithm can significantly improve the performance of the search program and shorten data processing time. Moreover, this parallel strategy is based solely on CPUs, enabling high-performance data processing in a pure CPU environment without code modification.

Keywords: single pulse search; pulsar; parallelization

1 Introduction

Pulsars are a special class of neutron stars with extremely strong magnetic fields and highly stable rotation periods. Based on this stable periodicity, current pulsar detection primarily employs Fast Fourier Transform (FFT) and Fast Folding Algorithm (FFA) [1] for periodic searches in the time domain. However, detection studies have revealed that some pulsars do not exhibit obvious periodicity, such as nulling pulsars [2-3], intermittent pulsars, as well as the Fast Radio Bursts discovered in [4] and the Rotating Radio Transients discovered in [5]. These celestial objects cannot be found through periodic search methods and require single pulse search for detection.

Since the beginning of the 21st century, radio telescope equipment has become increasingly sophisticated. Higher-resolution observations have caused pulsar search data volumes to grow rapidly at rates reaching TB, PB, or even EB scales [6]. On January 11, 2020, the Five-hundred-meter Aperture Spherical radio Telescope (FAST) was officially commissioned. With FAST's efficient operation, it is estimated that observation data collected through the L-band 19-beam receiver can reach 1.6 TB per hour, with annual data volumes of 10-20 PB [7]. This surge in data volume poses severe challenges for storage and processing. Additionally, predictions suggest there are approximately 150,000 potential pulsars in the Milky Way, of which about 30,000 are detectable [8]. However, the Australia Telescope National Facility (ATNF) pulsar catalog currently contains no more than 3,500 pulsars (https://www.atnf.csiro.au/research/pulsar/psrcat/). Therefore, many pulsars in the Milky Way remain undetected. Efficient processing of this data has become extremely urgent; otherwise, it will constrain FAST' s scientific discoveries of pulsars.

Currently, commonly used single pulse search tools include PRESTO (PulsaR Exploration and Search Toolkit) (https://www.cv.nrao.edu/~sransom/presto/), SEEK in SIGPROC (https://sourceforge.net/projects/sigproc/), and HEIMDALL (https://sourceforge.net/projects/heimdall-astro/), a GPU-accelerated single pulse search software. PRESTO is a large pulsar search and analysis software suite developed by Scott Ransom [9] and is one of the most commonly used software packages for pulsar searches, having participated in the discovery of over 700 pulsars (https://baijiahao.baidu.com/s?id=1689281702839604685&wfr=spider&for=pc). FAST observations primarily use PRESTO for pulsar search [10]. However, PRESTO was developed relatively early, and its search pipeline mostly employs single-core serial execution. Faced with massive pulsar observation data, PRESTO' s search speed is still inadequate. Applying parallel programming techniques to fully utilize multi-core processor computing resources will significantly improve data processing performance and accelerate the pace of scientific discovery. This paper first introduces single pulse search technology, analyzes the current single pulse search process and its characteristics, optimizes the detrending algorithm in PRESTO' s single pulse search, and parallelizes the optimized single pulse search. Experimental results demonstrate that algorithm optimization and CPU parallelization significantly reduce data processing time and improve single pulse search performance.

2 Single Pulse Search

The single pulse search algorithm was first proposed in [11]. Considering various factors that affect the strength, shape, and width of received pulses, the approximate signal model is described as:

$$() = ()()[() * () * () * ()] + (), \quad (1)$$

where the source signal () is modulated by modulation factors () and () corresponding to refractive and diffractive scintillation, while being convolved (denoted by asterisks) with several factors including dispersion smearing (), pulse broadening caused by multipath propagation (), and averaging by the receiver and data acquisition system (). () is additive receiver noise. The dispersion smearing () related to () can be deconvolved from the signal. However, if an inaccurate Dispersion Measure (DM) value is used, there may be residual dispersion smearing.

The width of collected signal pulses is described as:

$$W = \sqrt{W_s^2 + W_{DM}^2 + W_{t_{samp}}^2 + W_{t_{DM}}^2}, \quad (2)$$

where W_s is the scattering time; $W_{t_{samp}}$ is the sampling time; $W_{t_{DM}}$ is the intrinsic pulse width; and W_{DM} is the frequency-dependent smearing caused by

dispersion measure, defined as ():

$$W_{DM} = 8.3 \times 10^6 \cdot \mathrm{DM} \cdot \Delta f_{\mathrm{MHz}} \cdot f_{\mathrm{GHz}}^{-3} \text{ ms}, \quad (3)$$

In equation (3), Δf is the total bandwidth and f is the observation frequency. $W_{t_{DM}}$ is the smearing caused by using an incorrect dedispersion, defined as ():

$$W_{t_{DM}} = 8.3 \times 10^6 \cdot \Delta\mathrm{DM} \cdot f_{\mathrm{GHz}}^{-3} \cdot \frac{B}{N_{\mathrm{chan}}} \text{ ms}, \quad (4)$$

In equation (4), N_{chan} is the number of frequency channels and ΔDM is the error between the trial DM value and the true DM value.

According to the single pulse search theory proposed in [11], single pulse search in PRESTO is mainly divided into three stages: data preparation, single pulse search, and result diagnosis. The single pulse search flowchart is shown in [Figure 1: see original paper], where the purple part represents the data preparation stage, the green part represents the single pulse search stage, and the blue part represents the result diagnosis stage. The data preparation stage primarily involves marking interference signals, setting dedispersion plans, and dedispersing observation data to obtain dat files corresponding to different DM values. The single pulse search stage mainly performs detrending on time series, matched filtering, screening candidate information, and generating search result plots based on candidate information. The result diagnosis stage further determines whether the candidate information contains celestial signals.

Numerous optimization and improvement efforts for PRESTO have achieved good results, such as GPU-based dedispersion acceleration in [12], parallelization of dedispersion, FFT, and acceleration search on CPUs in [13], a parallel computing task scheduling system developed in [14] that divides tasks based on individual observation data files, optimization of PRESTO pulsar candidate image output for FAST-specific parameter space in [15], and optimization of acceleration search in PRESTO on GPUs using parallelization techniques in [16].

Regarding optimization of the single pulse search stage in PRESTO, [17] used Python' s multiprocessing module to achieve CPU-based parallel acceleration of single pulse search, but the paper did not introduce the specific implementation or acceleration effect. This paper will provide a detailed introduction and experimental reproduction.

3.1 Algorithm Optimization for Single Pulse Search

In a hardware environment with an AMD Ryzen 7 2700x eight-core processor*16, 64GB memory, and Ubuntu 18 operating system, performance analy-

sis of PRESTO' s single pulse search program single_{pulse}*{search}.py using FAST observation data file FH20201014*{00C10}.fits revealed that the detrending algorithm accounts for approximately 60% of the total time overhead of single_{pulse}*{search}.py. The time consumption of each step in the single*{pulse}_{search}.py program is shown in . If the performance of the detrending algorithm can be improved, the overall performance of PRESTO' s single pulse search algorithm will be significantly enhanced.

Detrending suppresses power spectrum fluctuations caused by long-term observations during signal acquisition to achieve optimal detection performance. The original program implements the detrending algorithm by calling the detrend method in the signal processing module of the Python mathematical computing library Scipy. The core of this method uses least squares to calculate the parameters of the fitted line. However, in the computation flow of this method, there are operations for reconstructing and restoring data shapes, as well as calculations of redundant parameters. The original detrend algorithm flow is shown in Figure 2: see original paper. This paper optimizes this method by redesigning the detrending computation flow, removing redundant calculations in the original method, and reducing unnecessary overhead. The optimized detrending algorithm flow is shown in Figure 2: see original paper: (1) In the original method, data shape reconstruction and type conversion are performed to obtain a suitable data type for computation. In the optimized algorithm, time series data is directly converted to a list type supported by Cython, reducing computational overhead from data shape reconstruction and type conversion. (2) In the original method, the least squares method calculates regression coefficients (coef), residual sum of squares (resids), independent variable rank (rank), and independent variable singular values (s). However, for fitting a straight line, only regression coefficients need to be calculated. In the optimized algorithm, the least squares method is redesigned to calculate only regression coefficients (coef) and is implemented based on Cython programming, reducing overhead from other parameter calculations.

3.2 Parallelization Based on Optimized Detrending Algorithm

FAST pulsar search scenarios require processing numerous small files with frequent read and write operations. GPU-based parallelization struggles to achieve high performance in search scenarios with large numbers of small files [18]. Additionally, the search algorithm itself has excessive data dependencies or control dependencies, which greatly affects acceleration effectiveness. Therefore, this paper chooses CPU parallelization.

In theory, a computer can run as many processes simultaneously as it has CPU cores. However, current single pulse search programs are still designed based on single-core serial execution, which clearly no longer conforms to the mainstream

of parallel algorithms in the multi-core era. Therefore, it is necessary to parallelize the serially executed single pulse search algorithm from a parallelization perspective, thereby fully utilizing the parallel capabilities of multi-core CPUs to improve program performance.

Analysis of the single pulse search flow reveals that when the single pulse search program searches different dat files, it outputs different singlepulse files. There is no data coupling between different dat files, and the search for each dat file is independent. Therefore, single pulse search parallelization can be achieved simply by assigning different dat files to different processes for individual searching.

In the original single pulse search stage, search results from each dat file need to be written to a global list for storage. If the single pulse search flow is parallelized without modification, each process writing search results to the global list will incur some inter-process communication overhead, affecting acceleration effectiveness. The schematic diagram of direct parallelization of the original single pulse search is shown in Figure 3: see original paper. To reduce inter-process communication overhead, the single pulse search flow is improved by returning the search results of each dat file and writing them to the global list only after all dat file searches are completed. The schematic diagram of the improved single pulse search parallelization is shown in Figure 3: see original paper.

3.2.1 Parallelization Based on multiprocessing Module

Python' s standard library multiprocessing is a multi-process parallel data processing solution that helps developers easily convert programs from single-process execution to multi-process parallel execution with low technical complexity.

For different pulsar observation files and different parameters used in DM plan step size planning, the number of generated dat files varies greatly. If the parallel program creates and destroys processes for each dat file search, it will incur overhead that affects overall program performance. Therefore, when introducing the multiprocessing module for single pulse search parallelization, ProcessPoolExecutor is used to manage multi-process tasks to reduce overhead from process creation and destruction. The process pool also improves program stability by preventing crashes caused by creating too many processes.

When using the multiprocessing module for parallelized search, the number of processes in the pool is first set, and the process pool manages the search tasks for these dat files. When a new task request is submitted to the process pool, if the parallel tasks in the pool are not full, a new process is created to execute the new task request. If the parallel tasks in the pool are full, the task will wait until a process in the pool becomes idle, at which point it immediately responds to the task request.

3.2.2 Parallelization Based on Ray Framework

The parallelization implemented by the multiprocessing module above does not achieve high CPU utilization due to communication between child processes and the main process, failing to reach or approach 100%. To address this, this paper uses the higher-performance parallel framework Ray [19] to achieve parallelization. Ray is a distributed computing framework developed by the RISE Lab at UC Berkeley that can flexibly run any compute-intensive Python workload.

The schematic diagram of single pulse search parallelization using Ray is shown in [Figure 4: see original paper]. When the main process executes, it creates a global scheduler and a local scheduler (the global scheduler is only enabled in cluster deployment mode and is not discussed here). When the main process calls a remote function, it submits the task to the local scheduler, which then assigns the task to worker processes in the local machine. The worker processes complete the computation and return the results.

3.3 Experimental Results Analysis

The hardware environment for comparative experiments on algorithm optimization and parallelization of the single pulse search program consists of an AMD Ryzen 7 2700x eight-core processor*16, 64GB memory, and Ubuntu 18 operating system. This paper tests 3,500 dat files generated after dedispersion of FAST observation data files (FH20201014_{00C10}, with data parameters shown in). The experimental data parameters are shown in , and the dedispersion scheme for the experimental data is shown in .

With the same experimental environment and data, the single pulse search program before and after detrending algorithm optimization was run 10 times, with each run time recorded. The experimental results are shown in [Figure 5: see original paper], where the horizontal axis represents run number and the vertical axis represents time consumed (in seconds). single_{pulse}*{search}*{raw} is the line chart of time consumption for the original program, single_{pulse}*{search}*{rm}*{rad} is the line chart after removing redundant calculations from the original program, single*{pulse}*{search}*{Cy}*{raw} is the line chart of time consumption after reimplementing the original detrend algorithm using Cython, and single*{pulse}*{search}*{Cy}*{rm}*{rad} is the line chart of time consumption after redesigning the detrend algorithm and optimizing it with Cython programming. [Figure 5: see original paper] shows that redesigning the detrend algorithm and optimizing it with Cython programming improves performance by approximately 1.8 times compared to the original program.

Based on Cython programming optimization of the detrend algorithm, parallelization was implemented using the Ray framework. The original single pulse

search program, the multiprocessing-based parallelized program, and the Ray-based parallelized program were each run 10 times, with each run time recorded. The experimental results are shown in [Figure 6: see original paper], where the horizontal axis represents run number and the vertical axis represents time consumed (in seconds). raw is the line chart of time consumption for the original program, Cy_{detrend} is the line chart after detrend algorithm optimization, multiprocessing is the line chart after detrend algorithm optimization and CPU parallelization using the multiprocessing module, and Ray is the line chart after detrend algorithm optimization and CPU parallelization using the Ray framework. [Figure 6: see original paper] shows that using the Ray framework to improve the single pulse search program achieves tremendous performance improvement, with a speedup of approximately 10 times compared to the original single pulse search program, and also shows improvement over the multiprocessing module parallelization version, with a speedup of approximately 3 times. Overall, the optimization and parallelization of the program provide certain performance improvements.

Comparison of search result plots before and after optimization shows that the parallelized single pulse search algorithm based on optimized detrending proposed in this paper can maintain the search effectiveness of the serial algorithm without causing inconsistent search results due to algorithm optimization and parallelization. The search result plots before and after program optimization are shown in [Figure 7: see original paper], where (a) is the search result plot generated by the original serial program, (b) is the search result plot generated after detrend algorithm optimization, (c) is the search result plot generated by multiprocessing module parallelization, and (d) is the search result plot generated by Ray framework parallelization.

---

4 Summary and Outlook

This paper introduced and analyzed single pulse search technology in PRESTO. Based on the characteristics of single pulse search technology, Cython programming, the multiprocessing parallel module, and the Ray parallel framework were used to optimize the original single pulse search program in PRESTO, significantly improving single pulse search program performance while ensuring search effectiveness. The algorithm optimization and parallelization work in this paper are based purely on CPU environments, requiring no dependence on CUDA and GPUs, and enabling high-performance single pulse search in CPU environments without code modification.

The original single pulse search program was developed based on Python. Reimplementing the program in C or C++ could achieve better acceleration effects, but would require a long development cycle. To further improve single pulse search program performance, the next step could attempt to completely reimplement PRESTO' s single pulse search program in C or C++ and use the Ray

framework to build cluster deployment for distributed parallel computing, to better meet the rapid search requirements for ever-growing pulsar observation data.

Acknowledgments

This work was completed based on FAST (Five-hundred-meter Aperture Spherical radio Telescope) data. FAST is a national major scientific facility operated and managed by the National Astronomical Observatories, Chinese Academy of Sciences. We thank the Guizhou Provincial Key Laboratory of Information and Computing Science for providing data support.

References

[1] STAELIN D H. Fast folding algorithm for detection of periodic pulse trains[J]. Proceedings of the IEEE, 1969, 57(4): 724-725.

[2] BACKER D C. Pulsar nulling phenomena[J]. Nature, 1970, 228(5266): 42-43.

[3] WANG N, MANCHESTER R N, JOHNSTON S. Pulsar nulling and mode changing[J]. Monthly Notices of the Royal Astronomical Society, 2007, 377(3): 1383-1392.

[4] LORIMER D R, BAILES M, MCLAUGHLIN M A, et al. A bright millisecond radio burst of extragalactic origin[J]. Science, 2007, 318(5851): 777-780.

[5] MCLAUGHLIN M A, LYNE A G, LORIMER D R, et al. Transient radio bursts from rotating neutron stars[J]. Nature, 2006, 439(7078): 817-820.

[6] NORRIS R P. Data challenges for next-generation radio telescopes[C]// Proceedings of the Sixth IEEE International Conference on e-science Workshops. 2011:21-24.

[7] LI D, WANG P, QIAN L, et al. FAST in space: considerations for a multi-beam, multipurpose survey using China' s 500-m aperture spherical radio telescope (FAST)[J]. IEEE Microwave Magazine, 2018, 19(3): 112-119.

[8] SWIGGUM J K, LORIMER D R, MCLAUGHLIN M A, et al. Arecibo pulsar survey using ALFA. III. precursor survey and population synthesis[J]. The Astrophysical Journal, 2014, 787(2): 137.

[9] RANSOM S M. New search techniques for binary pulsars[C]// Bulletin of the American Astronomical Society. 2001: 99-102.

[10] PAN Z C, QIAN L, YUE Y L. Pulsar searching techniques and their applications to FAST pulsar search[J]. Astronomical Research & Technology, 2017, 14(1): 8-16.

[11] CORDES J M, MCLAUGHLIN M A. Searches for fast radio transients[J]. The Astrophysical Journal, 2003, 596(2): 1142.

[12] YOU S P, WANG P, YU X H, et al. A GPU based single-pulse search pipeline (GSP) with database and its application to the Commensal Radio Astronomy FAST Survey (CRAFTS)[J]. Research in Astronomy and Astrophysics, 2022, 21(12): 314.

[13] YU Q Y, PAN Z C, QIAN L, et al. A PRESTO-based parallel pulsar search pipeline used for FAST drift scan data[J]. Research in Astronomy and Astrophysics, 2020, 20(6): 091.

[14] ZHANG H, XIE X Y, LI D, et al. A data processing acceleration method and system for FAST Petabyte pulsar data processing[J]. Astronomical Research & Technology, 2021, 18(1): 129-137.

[15] ZHANG X, LIU Z J, WANG P, et al. Improvement of candidate recognition accuracy in PRESTO pulsar search[J]. Journal of Guizhou Normal University (Natural Sciences), 2018, 36(3):84-88.

[16] JI C X, YU X H, LIU Z J. Binary pulsar system acceleration search method and software improvement[J]. Astronomical Research & Technology, 2022, 19(02):103-110.

[17] YOU S P. Research and application of parallel acceleration of pulsar search data processing[D]. Guizhou: Guizhou University, 2021.

[18] YU Q Y, PAN Z C, QIAN L, et al. A PRESTO-based parallel pulsar search pipeline used for FAST drift scan data[J]. Research in Astronomy and Astrophysics, 2020, 20(6): 091.

[19] MORITZ P, NISHIHARA R, WANG S, et al. Ray: a distributed framework for emerging {AI} applications[C]//Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18). 2018: 561-577.

Note: Figure translations are in progress. See original paper for figures.