GPU-Based Incoherent Dedispersion Algorithm Postprint

Authors: Tohti Nur, Zhang Hailong, Wang Jie, Ye Xinchen

Date: 2020-05-15T10:44:39+00:00

Abstract

To address the real-time dedispersion processing requirements for pulsar signals, a GPU-based incoherent dedispersion algorithm has been implemented. Employing high-performance parallel computing methods, an in-depth investigation into the multi-threaded processing of the incoherent dedispersion algorithm was conducted, and a parallelization acceleration scheme was proposed, thereby solving the problem that the high computational load of the dedispersion algorithm prevents real-time processing. By analyzing the computationally intensive portions of the algorithm and efficiently utilizing the hierarchical memory architecture of the GPU, GPU resource utilization was improved, computation time was reduced, and the computational performance of the incoherent dedispersion algorithm was significantly enhanced.

Full Text

GPU-Based Incoherent Dedispersion Algorithm

Toktonur[13], Zhang Hailong[123]*, Wang Jie[13], Ye Xinchen[13]
[1]Xinjiang Astronomical Observatory, Chinese Academy of Sciences, Urumqi 830011, China
[2]Key Laboratory of Radio Astronomy, Chinese Academy of Sciences, Nanjing 210008, China
[3]National Astronomical Data Center, Beijing 100101, China

Abstract

To meet the requirements of real-time dedispersion processing for pulsar signals, we have implemented a GPU-based incoherent dedispersion algorithm. Using high-performance parallel computing methods, we conducted an in-depth study of multi-threaded processing for the incoherent dedispersion algorithm

and proposed a parallel acceleration scheme that solves the problem of excessive computational load preventing real-time processing. By analyzing the compute-intensive portions of the algorithm and efficiently utilizing the hierarchical memory structure of GPUs, we improved GPU resource utilization, reduced computation time, and significantly enhanced the computational performance of the incoherent dedispersion algorithm.

Keywords: De-dispersion; GPU; Parallel Computing; Real-time Processing

---

The interstellar medium (ISM) in space contains substantial ionized gas clouds, neutral dust particles, and free electrons. When pulsar signals propagate through space, their velocity is reduced by the dispersive effects of the ISM. High-frequency radio waves travel faster than low-frequency waves, causing a time delay between high- and low-frequency electromagnetic waves arriving at the radio telescope. This dispersion results in energy spreading of the pulsar signal, broadening the pulse profile, reducing the signal-to-noise ratio (SNR), and potentially causing the pulse signal to disappear entirely.

To address this dispersion problem, astronomers have studied dispersion removal methods [1,2] and high-speed dedispersion processing techniques. Using a multi-channel filter bank [3], pulsar signals are divided into multiple narrowband signals. Time-delay processing is applied to each narrowband signal, with the delay for each channel calculated using the dispersion formula. Finally, all narrowband channels are accumulated to obtain a high-SNR pulsar signal.

Incoherent dedispersion is the most commonly used dispersion correction method in pulsar observation data processing, offering advantages of simple implementation, high speed, and flexible post-processing. In pulsar searches, as the dispersion measure (DM), number of data channels, and sampling points increase, the computational load of the incoherent dedispersion algorithm grows rapidly, making it difficult for general-purpose computing platforms to achieve real-time processing. In recent years, the programmability and parallel processing capabilities of GPUs [4,5] have improved dramatically, expanding their application scope. In CPU+GPU hybrid computing systems, the addition of GPUs significantly enhances overall data processing capability. High-performance GPU cluster systems can provide powerful computing resources to meet the real-time processing requirements of massive astronomical data, thereby solving the problem that the enormous computational load of pulsar dedispersion algorithms prevents real-time processing.

1 Incoherent Dedispersion

The incoherent dedispersion algorithm calculates the delay time for each channel based on the dispersion formula, applies the respective delays to each channel, and sums all channels together to eliminate dispersive effects. The principle of incoherent dedispersion is illustrated in Figure 1 [Figure 1: see original paper].

Incoherent dedispersion is implemented using a multi-channel filter bank. The dedispersion process mainly includes: (1) Channel division: using a filter bank to divide the total bandwidth of the observed astronomical signal into several independent narrow channels; (2) Time delay compensation: calculating the time delay for each channel based on the dispersion formula, shifting each channel according to its delay to align the pulse signals from all narrowband channels at the same moment; and (3) Channel accumulation: superimposing the time series from all channels.

Figure 1 illustrates the processing procedure of incoherent dedispersion. The left half shows the signal before dedispersion, while the right half shows the result after processing. Before dedispersion, the pulse width is broadened after channel accumulation, and the output signal SNR decreases. After dedispersion, a pulsar profile with significantly improved SNR can be obtained.

Incoherent dedispersion has been widely applied in pulsar and fast radio burst (FRB) searches [5]. For pulsar data processed by incoherent dedispersion, dispersion delays within each sub-channel still exist, preventing acquisition of the true pulse profile. As the number of spectral channels increases, the bandwidth of each channel becomes smaller, and the intrachannel dispersion effect can be correspondingly reduced. The propagation time difference between low-frequency signal f_1 and high-frequency signal f_2 in the intergalactic medium is:

$$\Delta t = \frac{e^2}{2\pi m_e c} \times DM \times \left(\frac{1}{f_1^2} - \frac{1}{f_2^2} \right)$$

where c is the speed of light in vacuum, e is the electron charge, DM is the dispersion measure, and m_e is the electron mass. The dispersion measure can be expressed as:

$$DM = \int_0^d n_e \, dl$$

where n_e is the electron density and d is the actual path traveled by the electromagnetic wave.

In pulsar dedispersion processing, the time delay of a frequency channel relative to a reference channel (typically the center frequency of the observation bandwidth) can be calculated using the dispersion formula:

$$\Delta t_{chan} = K_{DM} \times DM \times \left(\frac{1}{f_{ref}^2} - \frac{1}{f_{chan}^2} \right)$$

where K_{DM} is the dispersion constant, DM is the dispersion measure of the observed pulsar signal (in units of cm^{-3} pc), and frequency is in MHz. The dispersion constant is:

$$K_{DM} = \frac{e^2}{2\pi m_e c} \approx 4.148808 \times 10^3 \text{ MHz}^2 \text{ pc}^{-1} \text{ cm}^3 \text{ s}$$

In actual observations, the observing frequency f is often much greater than the channel bandwidth Δf. When $f \gg \Delta f$, the delay time of a frequency channel can be approximated as:

$$\Delta t \approx 8.3 \times 10^3 \times DM \times \frac{\Delta f}{f^3} \text{ ms}$$

This equation shows that channel bandwidth is proportional to dispersion delay time. To obtain smaller dispersion delays, it is necessary to divide very fine narrowband channels to minimize the signal bandwidth of a single channel.

2 GPU Implementation of the Incoherent Dedispersion Algorithm

GPUs are common devices in modern PCs, designed as highly parallelized, multi-threaded, multi-core processors for executing complex mathematical and geometric calculations to achieve high-speed graphics rendering. General-purpose computing on GPU (GPGPU) has become a research hotspot in high-performance parallel computing. Due to their numerous cores, GPUs are more suitable for compute-intensive operations. GPU cores are equivalent to multiple threads on a CPU, which can run in parallel to complete designated computational tasks, with numerical computation speeds far exceeding those of CPUs. The thread structure of a GPU is shown in Figure 2 [Figure 2: see original paper].

GPU development utilizes CUDA [6] programs. CUDA is a general-purpose parallel computing platform and programming model introduced by NVIDIA that enables GPUs to solve complex computational problems. CUDA provides direct hardware access interfaces, independent of graphics APIs, and employs a novel computing architecture to utilize GPU hardware resources. CUDA uses extensions to standard C as its programming language, providing numerous high-performance computing instructions that allow implementation of more efficient intensive data computation algorithms based on the powerful computing capabilities of GPUs.

Algorithm Analysis and Parallelization Strategy

The computational complexity of the incoherent dedispersion algorithm is $O(N_d \times N_s \times N_c)$, where N_d is the total number of DMs, N_c is the number of channels, and N_s is the number of samples. Although the mathematical model involves substantial computation, excellent speedup can be achieved through GPU processing. In the GPU algorithm, parallelization is applied to the dispersion measure and time dimensions of incoherent dedispersion, while frequency channel accumulation employs a serial processing method, using a

single GPU thread to implement the summation of N_c channels. Throughout the dedispersion process, the CPU is responsible for system initialization and input data reading, the GPU handles parallel dedispersion processing, and the CPU receives the dedispersed time series and outputs it to a data file. The CUDA program flowchart for the incoherent dedispersion algorithm is shown in Figure 3 [Figure 3: see original paper].

First, a buffer is allocated in CPU memory to write the data to be processed. The samples temporarily stored in the buffer all contain dispersion, requiring shift and accumulation operations for each frequency channel. As the time delay increases, the channel shift also increases, with greater shift amounts for low-frequency channels. To reduce the impact of frequent data transfers between CPU and GPU on algorithm performance, a global memory buffer is also set up in the GPU to copy large amounts of data in a single transfer, reserving sufficient storage space for data writing and reading. During GPU kernel function execution, shared memory is used to reduce global memory latency and improve kernel function execution speed.

To improve algorithm parallelization, the computation is divided into two parts. The first part, executed on the CPU, calculates channel delays:

$$t_{chan} = 4.15 \times 10^6 \times \left(\frac{1}{f_{low}^2} - \frac{1}{f_{high}^2} \right)$$

where f_1 and f_2 are in MHz. The calculation of t_{chan} is independent of the DM value and is stored in GPU constant memory. The second part of the computation is then executed on the GPU:

$$t_{DM} = t_{samp} \times \Delta t_{chan}$$

where t_{samp} is the sampling time in ms. Based on the characteristics of the incoherent dedispersion algorithm, two memory buffers are defined in the GPU kernel function: shared memory and constant memory. Shared memory stores the results of integration operations, while constant memory is used to store DM_{shift}. To enable fast channel accumulation, shared memory is used to hide global memory access latency. All threads within a thread block obtain the DM shift values and store them in shared memory. For each DM value, integration operations are performed in shared memory, and the dedispersion processing results are written to the GPU's global memory.

3 Experimental Analysis

The experimental platform used an Intel Xeon E5-1620 CPU, TITAN V GPU, CUDA 10.0, and Ubuntu 18.04. The TITAN V is a high-end GPU from the NVIDIA GeForce series, featuring 5120 CUDA cores and a memory access bandwidth of 652.8 GB/s.

To verify the performance of the GPU parallel algorithm, we simulated and generated pulsar data with 32, 64, 128, 256, 512, and 1024 channels. The data had a center frequency of 420 MHz, a bandwidth of 6 MHz, a sampling interval of 165 s, and a pulsar signal period of 0.1 s. Each data block was then individually read and loaded into the GPU for processing. The experimental results are shown in Tables 1 , 2 , 3 , and 4 .

Table 1. Incoherent dedispersion processing time when the number of samples is fixed (time: s, samples: 131072)

Table 2. Incoherent dedispersion processing time when the number of samples is fixed (time: s, samples: 131072)

Tables 1 and 2 show the processing time consumed by the GPU parallel algorithm and CPU serial algorithm for incoherent dedispersion when the number of samples is 131072 (single-channel sampling) and both the number of channels and DMs vary. When the number of channels is fixed, the data processing time for both GPU and CPU increases linearly with the number of DMs, but the GPU dedispersion processing time is far less than the CPU computation time. When the number of DMs is fixed, the processing time required by both CPU and GPU increases with the number of data channels, but the CPU computation time is much greater than that of the GPU.

Table 3. Incoherent dedispersion processing time when the number of channels is fixed (time: s, channels: 1024)

Table 3 shows the processing time consumed by the GPU parallel algorithm and CPU serial algorithm for incoherent dedispersion when the data has 1024 channels and both the number of samples and DMs vary. When the number of samples is fixed, the data processing time for both GPU and CPU increases with the number of DMs. When the number of DMs is fixed, both parallel and serial algorithms require more execution time as the number of samples increases. However, the GPU dispersion processing time is less than the CPU computation time, with a difference of several hundred times in computational speed. The table clearly demonstrates that the GPU significantly reduces the execution time of incoherent dedispersion.

Table 4. Incoherent dedispersion processing time when the number of DMs is fixed (time: s, DMs: 5120)

Table 4 shows the processing time consumed by the GPU parallel algorithm and CPU serial algorithm for incoherent dedispersion when the number of DMs is 5120 and both the number of samples and channels vary. When the number of samples is fixed, the data processing time for both GPU and CPU increases with the number of data channels. When the number of channels is fixed, both parallel and serial algorithms require more execution time as the number of samples increases. In all cases of varying channel numbers or sample numbers, the GPU operation time is less than that of the CPU, with a significant difference in time consumption.

Figure 4 [Figure 4: see original paper] shows the GPU algorithm speedup when the number of samples is 131072. Compared with the CPU, the GPU incoherent dedispersion data processing capability achieves a speedup of several hundred times, demonstrating significant acceleration advantages.

The speedup performance of the GPU parallel algorithm is shown in Figures 4 [Figure 4: see original paper], 5 [Figure 5: see original paper], and 6 [Figure 6: see original paper]. Figure 4 shows the speedup of the GPU parallel algorithm when the number of samples is 131072 and both the number of channels and DMs vary. The speedup improves as the number of DMs increases, reaching a maximum when the number of DMs is 2560, with TITAN V achieving a speedup of 538 times that of the CPU, after which the speedup begins to decrease. As shown in Figure 5, the greater the number of channels, the larger the GPU algorithm speedup and the better the acceleration performance. The speedup is highest when the number of channels is 1024 and lowest when it is 32.

Figure 5 [Figure 5: see original paper] shows the speedup of the GPU parallel algorithm when the number of DMs is 5120 and both the number of channels and samples vary. The speedup decreases slightly with increasing samples but still reaches over 550 times. TITAN V achieves its highest speedup when the number of channels is 1024. When the number of channels is 1024 and the number of samples is 8192, the GPU parallel algorithm is more than 780 times faster than the CPU algorithm.

Figure 6 [Figure 6: see original paper] shows the speedup of the GPU parallel algorithm when the number of channels is 1024 and both the number of samples and DMs vary. The number of DMs has the greatest impact on the GPU incoherent dedispersion algorithm speedup. The speedup increases rapidly with the number of DMs. As shown in Figure 6, larger DM values result in greater computational time consumption, and the speedup is maximized when the number of DMs is 5120.

Conclusion

The experimental results demonstrate that the number of samples, number of channels, and number of DM values are all key factors affecting GPU algorithm acceleration performance. During incoherent algorithm execution, the GPU demonstrates powerful parallel processing advantages, saving substantial repetitive computation time and improving computational efficiency.

This paper studies a GPU-based incoherent dedispersion algorithm and proposes a GPU parallelization acceleration scheme. By analyzing the compute-intensive parts of the algorithm, researching GPU multi-threaded task allocation, management, and memory hierarchy optimization methods, GPU resource utilization is improved and algorithmic computational performance is significantly enhanced. During GPU algorithm implementation, the main factors affecting parallel algorithm performance were thoroughly analyzed, the CUDA program was optimized, and algorithm execution time was greatly reduced. The GPU

dedispersion algorithm achieves a speedup of nearly 700 times, solving the problem that the algorithm's huge computational load on CPU prevents real-time processing. The algorithm's data processing time consumption and speedup were analyzed through experiments, validating the performance of the parallel algorithm.

References

[1] Spitzer Jr L. Physical processes in the interstellar medium[M]. John Wiley & Sons, 2008.
[2] Barsdell B R, Bailes M, Barnes D G, et al. Accelerating incoherent dedispersion[J]. Monthly Notices of the Royal Astronomical Society, 2012, 422(1): 379-392.
[3] Huang Yuxiang, Wang Min, Hao Longfei, Li Zhixuan, Xu Yonghua. Comparative Study between the Coherent De-dispersion and the Incoherent De-dispersion of Pulsar Signal[J]. Astronomical Research & Technology, 2019, 16(01): 16-24.
[4] Toktonur, Zhang Hailong, Wang Jie. A Design of Polyphase Filter Bank for Radio Astronomy Based on CUDA[J]. Astronomical Research & Technology, 2017, 14(01): 117-123.
[5] Su Tonghua, Li Dong, Li Songze. CUDA Programming: A Developer's Guide to Parallel Computing with GPUs[M]. China Machine Press, 2014.
[6] Rob Farber. CUDA Application Design and Development[M]. China Machine Press, 2013.
[7] Petroff E, Oostrum L C, Stappers B W, et al. A fast radio burst with a low dispersion measure[J]. Monthly Notices of the Royal Astronomical Society, 2019, 482(3): 3109-3115.
[8] Cheng J, Grossman M, McKercher T. Professional CUDA C programming[M]. John Wiley & Sons, 2014.

Funded by: National Natural Science Foundation of China (11873082, 11803080); National Key R&D Program of China (2018YFA0404704); Youth Innovation Promotion Association of Chinese Academy of Sciences; "Light of West China" Program (2019-XBQNXZ-B-018); National Astronomical Data Center, CAS Scientific Data Center System.

Received: 2020-00-00; Revised: 2020-00-00

Author Introductions: Toktonur, male, engineer. Research interests: digital backends and GPU parallel computing. Email: nuer@xao.ac.cn
Corresponding Author: Zhang Hailong, male, senior engineer. Research interests: data-intensive research. Email: zhanghailong@xao.ac.cn

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