Postprint: Implementation of a Dask-Based Parallel-Accelerated Gridding Method for Radio Interferometric Imaging

Authors: Li Shanshan, Luo Kaida, Wei Shoulin, Dai Wei, Liang Bo

Date: 2021-03-30T00:00:00+00:00

Abstract

The Fast Fourier Transform (FFT) offers superior algorithmic performance compared to the Fourier Transform and constitutes a fundamental algorithm in radio interferometric imaging. However, due to irregular sampling of antenna arrays, gridding algorithms are necessary to resample visibility data onto a regular grid prior to applying FFT. Convolution-based gridding computations are characterized by their computationally intensive and iterative nature, making high-performance gridding particularly crucial for accelerating the entire imaging process, especially when processing massive visibility datasets. To alleviate data processing pressure, we extend the application of the Dask parallel computing framework upon existing algorithms that process entire data blocks and support multi-core computation. This approach not only partitions data and distributes it across multiple threads to enhance numerical computation efficiency but also optimizes real-time gridding processing through dynamic distributed task scheduling strategies. Experimental results demonstrate significantly improved multi-core CPU utilization, with gridding algorithm performance further enhanced even as data volume increases. Distributed task scheduling enables elastic scaling of gridding for (single) multiple measurement sets across (single) multi-machine systems, fully leveraging the scalability advantages of clusters.

Full Text

A Distributed Gridding Implementation Method for Radio Interferometric Visibilities Based on Dask

Li Shanshan[1], Luo Kaida[1], Wei Shoulin[1],[2], Dai Wei[1],[2], Liang Bo[1],[2]
[1]Faculty of Information Engineering and Automation, Kunming University of Science and Technology, Kunming 650500, China

[2]Key Laboratory of Applications of Computer Technology of Yunnan Province, Kunming 650500, China

Abstract

Fast Fourier Transform (FFT) offers superior algorithmic performance compared to the discrete Fourier transform and serves as the foundational algorithm for radio interferometric imaging. However, due to irregular sampling in antenna arrays, gridding algorithms are required to resample visibility data onto a regular grid before FFT can be applied. Convolution-based gridding is characterized by its computationally intensive and iterative nature, particularly when processing massive visibility datasets. In such cases, high-performance gridding computation becomes crucial for accelerating the entire imaging process. To alleviate data processing pressures, we extend and apply the Dask parallel computing framework to existing algorithms that process data in monolithic blocks and support multi-core computation. This approach not only partitions data into chunks distributed across multiple threads to improve numerical computation efficiency but also leverages dynamic distributed task scheduling strategies to optimize real-time gridding processing. Experimental results demonstrate significantly improved multi-core CPU utilization, with gridding algorithm performance continuing to scale even as data volume increases. Distributed task scheduling enables elastic scaling of gridding for (single) multiple measurement sets across (single) multi-machine systems, fully exploiting the advantages of cluster scalability.

Keywords: Gridding; Interferometric imaging; Distributed computing; Parallel computing; Dask

Radio interferometric arrays produce non-uniformly sampled visibility data. Before applying Fast Fourier Transform (FFT) to image these visibilities, gridding methods must be used to resample the actual measurements onto a uniformly partitioned grid. Current gridding primarily employs convolution-based methods, which essentially involve matrix multiplication. For large datasets, this gridding computation becomes extremely time-consuming.

In recent years, astronomers have conducted extensive research to improve visibility data gridding algorithm performance. The W-projection algorithm is currently the most widely used gridding method, but since it only corrects for the W-term without accounting for direction-dependent effects (the A-term), the W-term size can become substantial when antennas are far apart, rendering the algorithm inefficient and memory-intensive [1]. The W-Stacking algorithm significantly improves gridding performance by projecting each visibility' s w-value onto neighboring w-planes, but this requires additional memory consumption [2]. When direction-dependent effects are considered, gridding computational complexity increases further. The algorithm that simultaneously corrects both the A-term and W-term is known as AW-projection [3]. In the numerical analysis domain, Barnett et al. [4] proposed the Non-uniform Fast Fourier

Transform (NUFFT) library based on a "semi-circle exponential" convolution kernel, extending FFT to discretized grid data. The Nifty-gridder algorithm, which first applied NUFFT to radio astronomy, employs shared memory and multi-threading techniques to further optimize the W-Stacking algorithm.

In summary, improvements and refinements to gridding algorithms require computing more convolution kernels, with convolution calculations constituting the primary computational overhead. While multi-core CPUs and GPUs [5] enable parallel computation to improve algorithm performance, Python-based implementations of the aforementioned gridding methods are primarily limited to NumPy multi-dimensional array operations, making them ill-suited for the demands of massive data volumes and real-time processing. The recent introduction of Dask.Array has opened new avenues for numerical computation with ultra-large matrices. Jamie Farnes et al. [8] employed the Dask parallel framework [9] combined with pipeline technology to test LOFAR datasets, reducing imaging pipeline execution time from 11 hours to 8 minutes—a dramatic reduction in interferometric imaging time. This paper proposes a Dask-parallel-accelerated convolutional gridding method for radio interferometric visibility data. Building upon parallel computation, our approach also considers system elastic scalability. The key features include using Dask.Array' s matrix chunking storage and computation as the core, encapsulating the Python interface provided by the Nifty-gridder convolutional gridding algorithm, employing data chunking and lazy evaluation to improve numerical computation efficiency, and implementing distributed scheduling strategies to enable migration of the gridding algorithm from single machines to clusters.

2. Gridding

The radio interferometric measurement equation establishes that visibility data V represents the Fourier transform of sky brightness distribution B, expressed mathematically as:

$$K_{pq} = e^{-2\pi i(u_{pq}l+v_{pq}m+w_{pq}(n-1))} \tag{1}$$

$$V_{pq} = G_p \left(\iint K_{pq} A_p B A_q^H \, dl \, dm \right) G_q^H \tag{2}$$

where (l, m, n) are direction cosines of the observation, (u_{pq}, v_{pq}, w_{pq}) are baseline coordinates for antennas p and q, and the G and A terms represent direction-independent and direction-dependent effects parameterized by Jones matrices, respectively. Under small-field approximations, the $w_{pq}(n-1)$ term in the exponent approaches zero, making the relationship between visibility and sky brightness approximately a two-dimensional Fourier transform. Due to irregular baseline uv tracks, visibility data are not uniformly sampled, making direct Fourier inversion of the measurement equation computationally expensive. To

apply FFT imaging, visibility data must be resampled onto a regular Cartesian grid (Gridding).

In the imaging pipeline [Figure 1: see original paper], visibility data measured at different spectral frequencies (i.e., image channels) can be processed independently. An image channel typically corresponds to one or more data channels. Imaging generally begins iteratively from a blank sky model. After gridding and inverse Fourier transform operations, one or more bright sources may obscure nearby faint sources. The CLEAN algorithm extracts bright point sources into the sky model. The reverse process of gridding involves performing a fast Fourier transform on the sky model to calculate model visibilities, known as degridding. Subtracting model visibilities from measured visibility data enables further extraction of faint sources. Gridding and degridding are repeated until the sky model converges.

Radio interferometric imaging is a critical step in radio astronomy. Simple interpolation of visibility data to neighboring grid points causes severe artifacts, particularly image aliasing (where strong sources outside the field of view, even noise, alias into the field). To suppress these side effects, visibilities are typically convolved with a gridding function (convolution kernel) before resampling onto the grid, providing anti-aliasing effects. Due to the window function characteristics of the convolution kernel, image cropping errors at boundaries are several orders of magnitude higher than at the center, producing a large dirty image. This requires multiplication by a correction function to offset errors introduced by the convolution kernel and obtain correct flux density. This correction function is typically the inverse of the Fourier transform of the convolution kernel. Compared to the W-Stacking algorithm, Nifty-gridder improves convolution kernel computation accuracy through: (1) gridding all visibility data along the w-axis into three-dimensional uvw space rather than projecting each visibility' s w-value onto neighboring w-planes; and (2) an improved correction function that minimizes differences between FFT and DFT of the dirty image (DFT being lossless), thereby obtaining higher-precision dirty images [10].

3.1 Parallel Reading and Chunking of Measurement Sets

Compared to NumPy' s `ndarray`, Dask.Array offers several advantages: (1) it supports splitting ultra-large arrays into many small NumPy `ndarray` chunks; and (2) it employs blocked algorithms to enable multi-core parallel computation on arrays larger than memory. Additionally, we utilize Xarray to achieve consistent chunking (`chunksize`), allowing relevant field data to be easily converted to Dask.Array types. For (single) multiple measurement set files, we uniformly place path information into a list object and use Dask.Bag for distributed loading, then group by the `FIELD_{ID}` and `DATA_{DESC}_{ID}` fields in the measurement sets to enable parallel loading.

In this experiment, the entire dataset is divided into four sub-datasets: (0_0, 0_1, 0_2, and 0_3). Taking sub-dataset 0_1 as an example, some important

fields and data types are shown in Table 1 .

3.2 Parallel Implementation of Gridding Methods

Distributed computing is an effective approach for massive data processing. The Dask parallel computing framework provides flexible and versatile distributed scheduling mechanisms. Since Dask' s task scheduling is decoupled from user-defined algorithms, switching scheduling modes enables elastic scaling of algorithms across (single) multi-machine systems using multi-(thread)process execution. However, appropriate scheduling strategies must be selected based on algorithm characteristics to achieve optimal performance. This paper employs the most sophisticated `dask.distributed` scheduler to execute the Nifty-gridder algorithm across two machine nodes. The task scheduling algorithm [Figure 2: see original paper].a adopts a multi-process execution approach:

Physical separation of multiple MS files facilitates parallel dataset reading using multi-process execution. MS files typically contain multiple radio sources (i.e., multiple sub-datasets), and task scheduling based on sub-datasets further refines the parallel granularity of the entire Nifty-gridder gridding workflow. Using the higher-order function `Dask.Array.blockwise` to encapsulate and invoke Nifty-gridder' s Python interface enables parallel computation based on sub-blocks and coordinates aggregation and concatenation operations of sub-blocks [Figure 2: see original paper].b. To avoid transmission costs of Dask.Array between processes, numerical computation of dirty images employs multi-threaded execution. The Nifty-gridder algorithm execution proceeds as follows:

1. Determine N_w sampling planes along the w-axis and distribute them uniformly across the w-axis (from w_0 to w_{N_w-1});
2. Grid visibility data onto w-planes along the w-axis;
3. Initialize a zero matrix $\mathcal{I}$ of size $N_x \times N_y$, and for each w = w_i plane:
   a) Perform uv-gridding on each w-plane, then execute a two-dimensional inverse Fourier transform;
   b) Crop the outer half of the iFFT image, then multiply by $e^{2\pi i w_i(\sqrt{1-l^2-m^2}-1)}$;
   c) Accumulate the result into matrix $\mathcal{I}$;
4. Multiply matrix $\mathcal{I}$ by the correction function to obtain the final dirty image.

4. Experimental Setup

Dataset 1 originates from an 8-hour observation of supernova remnant G055.7+3.4 by the Karl G. Jansky Very Large Array on August 23, 2010. The array was in D-configuration with an observation frequency range of 1-2 GHz, covering all available L-band frequencies. The experimental hardware environment consists of two high-performance servers: Intel Xeon CPU E5-2660 v4 @ 3.4 GHz processors (56 cores), 512 GB RAM. Data results were validated using Common Astronomy Software Applications (CASA 5.6.2).

4.2 Dask Parallel Acceleration and Experimental Results

Using four sub-datasets as an example, with `chunksize` set to 20,000 rows, dirty images were generated through gridding. Dataset volume was measured by the number of visibility rows, and execution times (in seconds) of Dask.Array and NumPy versions of the Nifty-gridder algorithm were compared in the same hardware/software environment. Results are shown in Table 2 .

Table 2 shows that for the Dask.Array-improved Nifty-gridder algorithm, CPU Time exceeds Wall Time in all cases, demonstrating that for compute-intensive problems, multi-core parallelization yields significant effects and substantially reduces program execution time. Comparing sub-datasets 0_0 and 0_1: even when visibility data volume increases by 10.5× (413,696/39,274), Wall Time only increases by 1.85× (4.95/2.68), with speedup ratio further improving. However, Dask.Array adds a layer of complexity atop NumPy; for smaller data volumes (sub-dataset 0_1 occupies approximately 40 MB), NumPy may be the appropriate choice. Conversely, this precisely demonstrates Dask.Array' s suitability for numerical computation with ultra-large matrices.

Dask enables task-based parallelism across clusters by submitting Python functions to generate numerous Future objects that can be monitored, controlled, and computed. For complex processing workflows, dynamic visualization monitoring helps identify algorithmic performance bottlenecks. Real-time performance monitoring during experimental execution is shown in Figure 3 [Figure 3: see original paper].

To demonstrate Dask' s superiority, we increased measurement set input volume while limiting memory allocation per machine, ensuring consistent experimental environments. System resource utilization was analyzed and compared between Dask.Array-based and NumPy-based Nifty-gridder algorithms. As shown in Figure 4 [Figure 4: see original paper], Dask.Array-based gridding achieves significantly lower peak and average CPU utilization and memory occupancy compared to the NumPy version, yet obtains faster gridding execution times (see Table 2). This occurs because Dask.Array employs chunked loading and lazy evaluation: sub-blocks not ready for computation remain on disk to conserve system resources, while computable sub-blocks are loaded into memory for parallel execution. In contrast, NumPy arrays must be fully loaded into memory, resulting in higher memory occupancy.

To further verify code correctness, we used standard CASA software to image this dataset[2], generating a dirty image (Figure 5 left) for comparison with our experimental results (Figure 5 right). The gray-white points in both grayscale images represent observed sources, confirming correct identification of radio source distribution positions.

High-performance distributed parallel computing has become essential for radio interferometric imaging to handle massive data from high-resolution and wide-field interferometric arrays. Visibility data gridding and degridding consti-

tute critical components of imaging, and parallel acceleration of gridding is undoubtedly significant for improving overall imaging speed. This paper employs the open-source Dask distributed computing framework combined with Nifty-gridder to implement distributed loading and parallel gridding acceleration of measurement sets, fully leveraging cluster scalability and improving multi-core CPU utilization. The interferometric imaging process involves multiple complex processing workflows, all involving matrix numerical computations. Since Dask.Array can efficiently handle multi-dimensional ultra-large matrix operations, future work will consider parallel acceleration of degridding, calibration, and imaging algorithms based on Dask.

References

[1] Cornwell T J, Golap K, Bhatnagar S. W projection: A new algorithm for wide field imaging with radio synthesis arrays[C]//Astronomical Data Analysis Software and Systems XIV. 2005, 347: 86.

[2] Cornwell T J, Voronkov M A, Humphreys B. Wide field imaging for the square kilometre array[C]//Image Reconstruction from Incomplete Data VII. International Society for Optics and Photonics, 2012, 8500: 85000L.

[3] Bhatnagar S, Cornwell T J, Golap K, et al. Correcting direction-dependent gains in the deconvolution of radio interferometric images[J]. Astronomy & Astrophysics, 2008, 487(1): 419-429.

[4] Barnett A H, Magland J, af Klinteberg L. A parallel nonuniform fast fourier transform library based on an "exponential of semicircle"kernel[J]. SIAM Journal on Scientific Computing, 2019, 41(5): C479-C504.

[5] Veenboer B, Petschow M, Romein J W. Image-domain gridding on graphics processors[C]//2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS). IEEE, 2017: 545-554.

[6] Merry, B. Faster GPU-based convolutional gridding via thread coarsening[J]. Astronomy & Computing, 2016, 16: 140-145.

[7] Feng Yong, Wang Feng, Deng Hui, et al. Implementation of Gridding Algorithm for Radio Interferometric Imaging Based on OpenCL[J]. Astronomical Research & Technology, 2019, 16(01): 8-15.

[8] Farnes J, Mort B, Dulwich F, et al. Science pipelines for the square kilometre array[J]. Galaxies, 2018, 6(4): 120.

[9] Rocklin M. Dask: Parallel computation with blocked algorithms and task scheduling[C]//Proceedings of the 14th python in science conference. Austin, TX: SciPy, 2015, 126.

[10] Ye H, Gull S F, Tan S M, et al. Optimal gridding and degridding in radio interferometry imaging[J]. Monthly Notices of the Royal Astronomical Society, 2020, 491(1): 1146-1159.

[2] https://casaguides.nrao.edu/index.php?title=VLA_{CASA}_{Imaging}-CASA5.7.0

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