

HEGrid: A High Efficient Multi-Channel Radio Astronomical Data Gridding Framework in Heterogeneous Computing Environments

Hao Wang^a, Ce Yu^a, Jian Xiao^{a,*}, Shanjiang Tang^a, Min Long^{b,*} and Ming Zhu^{c,d}

^aCollege of Intelligence and computing, Tianjin University, No.135 Yaguan Road, Haihe Education Park, Tianjin, 300350, China

^bDepartment of Computer Science, Boise State University, Boise, ID 83725, USA

^cNational Astronomical Observatories, Chinese Academy of Sciences, 20A Datun Road, Chaoyang District, Beijing, 100101, China

^dCAS Key Laboratory of FAST, National Astronomical Observatories, Chinese Academy of Sciences

ARTICLE INFO

Keywords:

Radio astronomy
Gridding
Multi-channel
High efficient
Heterogeneous architecture

ABSTRACT

The challenge to fully exploit the potential of existing and upcoming scientific instruments like large single-dish radio telescopes is to process the collected massive data effectively and efficiently. As a "quasi 2D stencil computation" with the "Moore neighborhood pattern," gridding is the most computationally intensive step in data reduction pipeline for radio astronomy studies, enabling astronomers to create correct sky images for further analysis. However, the existing gridding frameworks can either only run on multi-core CPU architecture or do not support high-concurrency, multi-channel data gridding. Their performance is then limited, and there are emerging needs for innovative gridding frameworks to process data from large single-dish radio telescopes like the Five-hundred-meter Aperture Spherical Telescope (FAST). To address those challenges, we developed a **High Efficient Gridding** framework, **HEGrid**, by overcoming the above limitations. HEGrid is the first effort to solve the gridding of multi-channel data from the large single-dish radio telescope by multi-pipeline concurrency in the CPU-GPU heterogeneous environment. Specifically, we propose and construct the gridding pipeline in heterogeneous computing environments and achieve multi-pipeline concurrency for high performance multi-channel processing. Furthermore, we propose pipeline-based co-optimization to alleviate the potential negative performance impact of possible intra- and inter-pipeline low computation and I/O utilization, including component share-based redundancy elimination, thread-level data reuse and overlapping I/O and computation. Our experiments are based on both simulated datasets and actual FAST observational datasets. The results show that HEGrid outperforms other state-of-the-art gridding frameworks by up to 5.5x and has robust hardware portability, including AMD Radeon Instinct GPU and NVIDIA GPU.

1. Introduction

Effective and efficient data processing methods are an emerging need to fully exploit the potential of existing and upcoming scientific instruments and accelerate scientific discovery, such as data processing for the large single-dish radio telescopes FAST, Arecibo, Effelsberg and Green Bank, etc. To record sky images from a wide range of frequencies, large single-dish radio telescope receivers contain a large number of independent channels with various band coverage settings. Five-hundred-meter Aperture Spherical Telescope (FAST) [3, 7, 16, 27], the world's largest single-dish radio telescope, has been in operation since 2020. FAST receivers comprise 65,536 independent frequency channels (a significantly large but typical number for many large single-dish radio telescopes) and generate a massive volume of radio astronomical data across all frequency channels at a rate of 10-20 PB-size per year [16].

To obtain the correct sky images from such data, gridding is one of the critical steps which maps non-uniform data samples onto a uniformly distributed target grid map (referred to as **target map**) for further analysis. It is usually the most computationally intensive and time-consuming step [12, 21, 25] due to the huge size of data in multiple channels.

For those reasons, there is a great need for fast and high-performance gridding frameworks to process multi-channel radio astronomical data from large single-dish radio telescopes.

Gridding algorithm is similar to stencil computation since it iteratively updates each target cell based on neighboring points and can be treated as one type of "quasi 2D stencil computation" with "Moore neighborhood pattern"[20]. However, gridding also differs from the stencil computation in the following ways: (1) the number of selected neighboring points for each cell may not be fixed but vary significantly; (2) the number and location of neighboring points for each cell is not determined till the cell is updated. These two features pose a challenge for effective access to neighboring points contributing to the calculation. In addition, there would be much more neighboring points used in gridding than in the stencil computation. For instance, in some gridding applications with high sampling densities, the number of neighboring points could reach nearly 90,000, adding additional challenge for cells update.

A number of gridding frameworks have been developed for processing data from various types of radio telescopes. Among them, Cygrid [26] is one of the most popular and effective gridding frameworks. It supports multi-core CPU architecture and has been applied to the Effelsberg-Bonn HI Survey and the Galactic All-Sky Survey [2]. However, as discussed above, the gridding is more suitable for imple-

*Corresponding author

 xiaojian@tju.edu.cn (J. Xiao); minlong@boisestate.edu (M. Long)
ORCID(s):

mentation on GPU architectures rather than CPU, due to its features of single instruction, multiple data stream (SIMD). Our previous work, HCGrid [22], gridding framework prototype designed in CPU-GPU heterogeneous computing environments for the large single-dish radio telescope, such as the FAST, which has demonstrated promising performance in the experiments with simulated datasets. However, HCGrid does not support high-concurrency processing of multi-channel data due to its low utilization of heterogeneous resources. It is worth noting that there are other gridding algorithms used in CPU-GPU heterogeneous architectures, such as [5], [17], [19], [21], [31], but none of them were designed for and can be applied to single-dish radio telescopes.

To overcome the limitations of existing gridding frameworks, combined with our previous work, we propose **HEGrid**, a high efficient gridding framework for the multi-channel data gridding of the large single-dish radio telescope. HEGrid is the first effort to solve the multi-channel data gridding of the large single-dish radio telescope by multi-pipeline concurrency in the CPU-GPU heterogeneous environment, it can port well for different GPU architectures including NVIDIA and AMD Radeon Instinct series. Specifically, given gridding's computational correlation and data correlation, our contributions are:

- (1) We present the design of the HEGrid, including the building of the gridding pipeline and the multi-pipeline concurrency implementation.
- (2) We propose pipeline-based co-optimization to alleviate the potential negative performance impact of possible low intra- and inter-pipeline computation and I/O utilization, which includes component share-based redundancy elimination, thread-level data reuse and overlapping I/O and computation.
- (3) We port HEGrid to various GPU architectures, such as NVIDIA and AMD Radeon Instinct series, enabling HEGrid with robust hardware portability.
- (4) We are releasing our implementation as open-source¹ for further research and use in achieving efficient gridding of astronomical data for current and upcoming large single-dish radio telescopes.

The rest of the paper is organized as follows. We provide the background on gridding algorithms and the motivation for innovative methods in Section 2 and discuss the related work in Section 3. In Section 4, we describe the design of HEGrid and optimization methods we used. Section 5 compares HEGrid to other gridding frameworks by conducting various performance experiments using both simulated and actual observational data from FAST. Section 6 concludes the paper.

2. Background and Motivation

2.1. The Need of Gridding

Radio telescope consists of antenna and array receivers which detect radio signals from astronomical sources in the

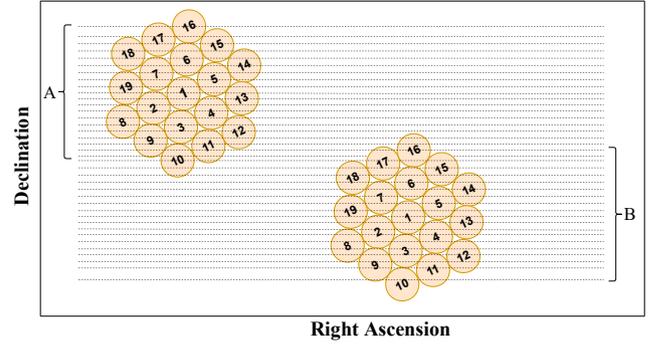


Figure 1: A schematic layout of 19-beam receiver of FAST and an example of two adjacent drift scans with areas A and B. The dotted lines show the drifting tracks of individual beams [6, 7, 28].

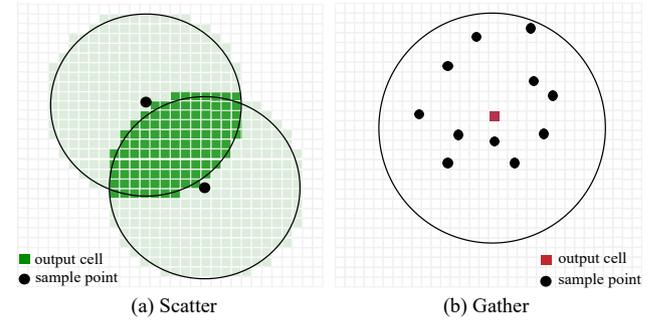


Figure 2: Input-oriented scatter (left) and output-oriented gather (right) gridding methods. Circles represent input data points and squares represent output cells.

sky. Given the size of the large single-dish radio telescope, deploying the telescope in a "drift scan" is usually needed as a feasible and near-optimum sky survey strategy. The "drift scan" means moving the telescope's receivers to a target azimuth and then fixing the telescope. Since the earth rotates once in 24 hours, various celestial objects enter the receiver's field of view once, recorded in a coordinate of two directions: right ascension and declination.

Figure 1 shows the layout of 19-beam receivers of FAST and the "drift scan" strategy of FAST's survey [7]. FAST adopts a fixed rotation angle of the beam pattern toward a certain declination on the sky, and the drifting drags the receiver along the right ascension direction. By rotating the array by 23.4° , the most uniform coverage in the declination direction and super-Nyquist sampling can be achieved [16]. After a 24-hour scan of a certain declination, a new declination is taken for new surveys of objects [16, 28]. The received continuous data streams are stored in a multi-dimensional array according to values of channel number, right ascension, declination, forming a multi-dimensional datacube. However, this strategy can cause a problem, the coverage of the raw recorded data has a much denser grid resolution in the right ascension direction than in the decli-

¹<https://github.com/HPCAstroAtTJU/HEGrid>

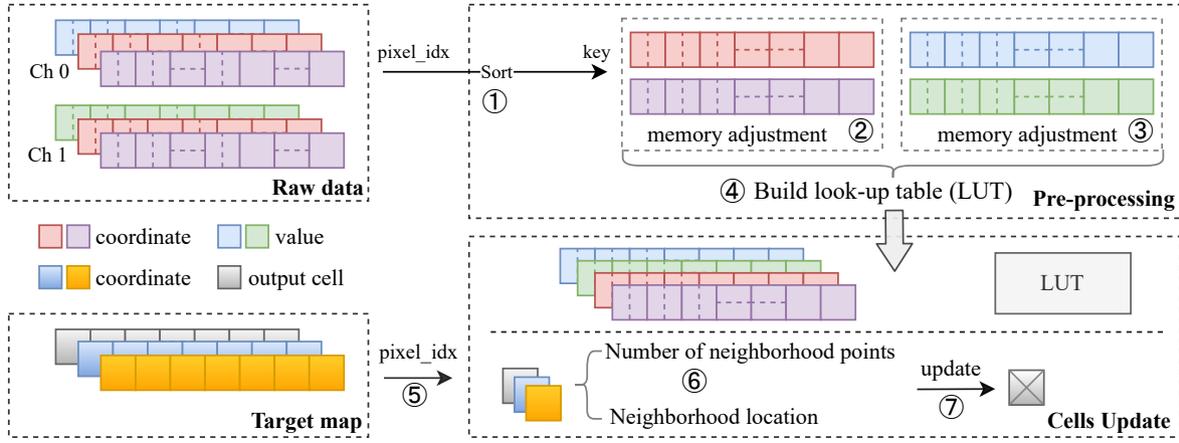


Figure 3: The overview of the HEGrid pipeline. The step of pre-processing runs on CPU. It first computes and sorts the `pixel_idx` of the raw data points, then adjusts the raw data memory, building a look-up table. The step of cells update runs on GPU, which loads the target map, raw data, and LUT, then computes the contribution region and updates the target cell. `ch0` and `ch1` represent different frequency channels.

nation direction [9, 10], limiting itself to be directly used for scientific analysis, which requires a uniform interval in the two spatial directions of right ascension and declination to obtain high quality of sky image. Like X-ray computed tomography [4], gridding can be seen as an image reconstruction method in radio astronomy, which solves the problem of uneven distribution of raw data points in the sampling space.

As with stencil computation, the kernel of a gridding computation typically contains a weighted sum of neighboring points for each target cell, and applies two methods of scatter or gather to the calculation [29]:

- (1) Scatter: The scatter method traverses each input data point and broadcasts its sampling value to other output cells within the convolution kernel. Then the sampling value is weighted and updated to each target output cells, as shown in Figure 2 (a).
- (2) Gather: The gather method traverses each output cell and searches for neighboring points within its convolution kernel. Then it adds up weighted values and updates to the output cell, as shown in Figure 2 (b).

2.2. The Gridding Algorithm

The gather-based gridding algorithm is as follows. After determining a target grid map, the output value for each targeted grid cell is calculated as the weighted sum of all neighboring samples. Let $\mathbb{S} = \{s_1, s_2, \dots, s_N\}$ denote N discrete, non-uniformly spaced input samples in the right ascension-declination plane (ra-dec). Each sample $s_n \in \mathbb{S}$, ($n \in \{1, 2, \dots, N\}$) has equatorial coordinates (α_n, δ_n) (i.e., right ascension and declination) and a sampled value of $V[s_n]$. For the output grid map, the ra-dec plane is divided into a regular, uniform grid with $I \times J$ cells as $\mathbb{G} = \{g_{1,1}, g_{1,2}, \dots, g_{I,J}\}$. For any cell $g_{i,j} \in \mathbb{G}$ with central coordinates $(\alpha_{i,j}, \delta_{i,j})$, its re-sampled value $V[g_{i,j}]$ equals the weighted sum of raw data \mathbb{S} related to $g_{i,j}$.

$$V[g_{i,j}] = \frac{1}{W_{i,j}} \sum_n V[s_n] w(\alpha_{i,j}, \delta_{i,j}; \alpha_n, \delta_n). \quad (1)$$

$s_n \in \mathbb{S}$ represents any raw input sample with a weighted contribution to $g_{i,j}$; $w(\alpha_{i,j}, \delta_{i,j}; \alpha_n, \delta_n)$ is a convolution kernel (weighting function) depending on positions of the output cell and raw data points, usually related to distances between input and output coordinates; and $W_{i,j} = \sum_n w(\alpha_{i,j}, \delta_{i,j}; \alpha_n, \delta_n)$ is the normalisation coefficient.

3. Related Work

Gridding is one of the most critical tasks in processing radio astronomical data such as pulsar data, spectral line data and so on. Several gridding frameworks have been developed and customized for such data processing in the field of radio astronomy. Cygrid [26] is one of the state-of-the-art works and runs only on the CPU platforms, which has been applied to studies like the Effelsberg-Bonn HI Survey and the Galactic All-Sky Survey[2]. However, its gridding performance is limited by the CPU architecture because it can't handle well the main features of gridding: single-instruction and multiple data (SIMD) stream. Modern parallel processors, such as GPU instead of CPU should be able to provide a better platform for such SIMD operations.

HCGrid [22] is designed for gridding data from single-dish radio telescopes. It is based on CPU-GPU heterogeneous architecture and can achieve good performance for single-channel data. However, it does not support the high concurrency processing of multi-channel data due to its low utilization of heterogeneous resources.

Image-Domain Gridding [21] implemented the gridding on both CPU and GPU, and has been deployed to the LO-FAR (Low-Frequency Array) Central Processing center. It utilizes CUDA stream and related mathematics library to optimize the gridding. However, it cannot be applied to single-dish radio telescopes, and the portability of the algorithm is not desirable.

Other methods including [5], [17], [19] and [31]. For instance, [19] developed a work-distribution scheme for grid-

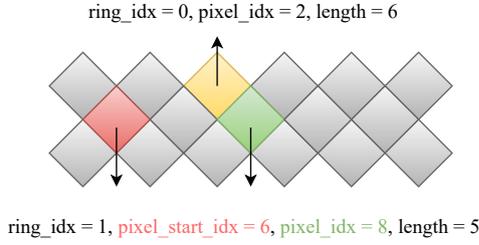


Figure 4: Raw data points are partitioned into different pixels. Each pixel has its index information, including pixel_idx, ring_idx, ring length, etc.

ding using GPU, which can reduce the memory access time of computing device, and map observed samples onto a grid with high efficiency. [17] optimized the algorithm using thread coarsening strategy, which makes each thread to handle multiple samples simultaneously. However, like Image-Domain Gridding, all those existing methods are only designed for radio telescope array but not fully applicable to the large single-dish radio telescopes.

4. Design of HEGrid

This section introduces the design and implementation of our multi-channel radio astronomical data gridding framework, HEGrid. We first explain the design of the HEGrid pipeline and present the capabilities of multiple pipeline concurrency on heterogeneous architecture. Then, the details of pipeline-based co-optimization strategies were given. Furthermore, we port HEGrid to heterogeneous computing environments with different GPU architectures.

4.1. HEGrid Pipeline

Figure 3 shows the pipeline of HEGrid. First, an efficient look-up table (LUT) is built in the pre-processing step to accelerate the contribution point acquisition process. Second, to accelerate the most time-consuming cell update step, we achieve cell update parallelization by using SIMT instruction-level parallelism on GPU.

4.1.1. Building the Efficient LUT

As discussed in Section 1, uncertainty (location, number) of contribution points brings challenges to the cell update. We design an efficient lookup table with the help of HEALPix²[11]. With HEALPix, the raw data points on the celestial surface are partitioned into different pixels with different indexes, as shown in Figure 4.

Figure 5 gives an example for illustrating the process of lookup table build in Figure 3. First, the 17 raw data points $S_i (i = 1, 2, \dots, 17)$ are partitioned into 9 pixels (A ~ I), and then the pixel_idx was sorted as shown in Figure 5 (step ① in Figure 3). The Block Indirect sort algorithm is utilized in our work, its average time complexity

²HEALPix is a software package for hierarchical equal-area isolatitude pixelation on spherical surfaces in astronomy, which makes fast, accurate statistical or astrophysical analysis of massive all-sky datasets.

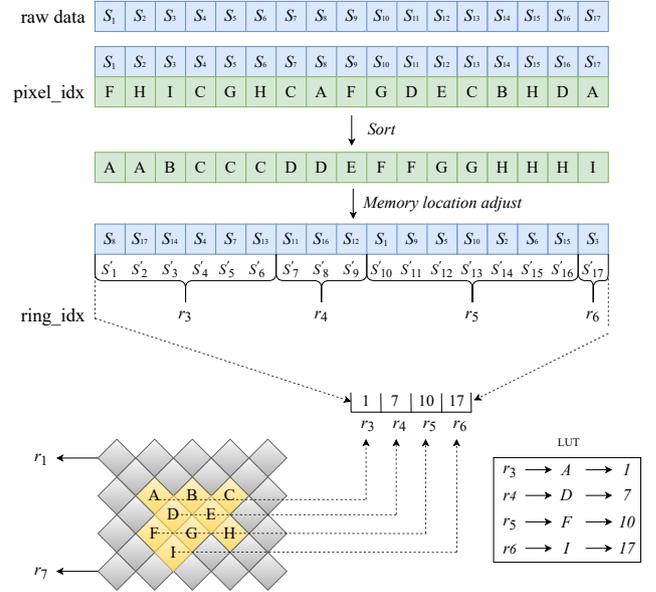


Figure 5: The schematic diagram of pre-processing. Top: Partition 17 raw data points $S_i (i = 1, 2, \dots, 17)$ into 9 pixels from "A" to "I". Middle: Adjust memory location of the raw data points based on the sorted pixel_idx. Bottom: Build a lookup table based on the partitioned pixels.

is $\mathcal{O}(N \log N)$. Second, the location of coordinates and sampling value in memory for the raw data points was adjusted according to their pixel_idx (steps ②, ③). Third, after computing the ring_idx of the latitude ring where different pixels are located, the lookup table is built based on the ring_idx, pixel_idx, and sampling points index (step ④). The pre-processing step runs on CPU because there is a series of logic operations.

4.1.2. Parallelizing Cell Updates

Vectorization is a common technique employed in parallel processors. Cell update step has computational characteristics of single instruction multiple data streams. In HEGrid, we manually vectorize the cell update on GPU in a SIMT manner.

Algorithm 1 shows the workflow of the cell update step. After loading the data points and lookup table from the host, we first compute the pixel_idx of the target cell, and determine the region (including the range of the ring, the starting pixel index on the contribution ring, and the offset between different contribution rings) of the contribution points for the target cell. Then, with the help of the developed lookup table in Section 4.1.1, the contribution points are loaded ring-by-ring from device memory to the streaming multiprocessor (SM)³, and their weights contributed to the target cell is computed. When a thread finishes its task for one target cell, we cache the temporary results to register memory.

As the smallest unit of SM execution and GPU resource

³We mainly use terminology from NVIDIA hardware, such as SM, block, and warp, equivalent to CU, workgroup, and wavefront in AMD terminology.

Algorithm 1: Cell Update Workflow

```

Input: sorted data, LUT, target map
Result: updated cells
1 for target_cell[0] to target_cell[n] do
2   Compute the pixel_idx of the target cell;
3   Compute the min contribution ring ring_min;
4   Compute the max contribution ring ring_max;
5   for ring_min to ring_max do
6     Compute the min contribution pixel
       pixel_min;
7     Compute the max contribution pixel
       pixel_max;
8     Compute the min indices i of raw data in
       pixel_min;
9     Load the contribution points raw_data[];
10    while pixel_idx of
        raw_data[i]  $\leq$  pixel_max do
11      if  $d(\text{target\_cell}[], \text{raw\_data}[i]) \leq R$ 
        then
12        Compute the weight sum;
13        Compute the weighted value;
14      end
15      i = i + 1
16    end
17  end
18  Normalize the weighted value;
19  Update cell;
20 end
    
```

scheduling for NVIDIA GPU and AMD GPU, the thread warp is the key to achieve efficient cell update. In a warp, all threads execute in a single-instruction, multiple-thread (SIMT) manner [14]. Target cells on the same row have the same contribution ring and the difference is that its contribution points may locate in different regions of the contribution ring. Further, as shown in Figure 6, the contribution points on the same contribution ring for adjacent target cells have overlapping contribution regions. To enable HEGrid could port to the computing environments with different GPU architectures and obtain a high cache hit rate on GPU, we propose an efficient organization strategy for parallel threads on GPU. The detail is that we use one thread block as a vector, and each of the threads in the thread block is responsible for one target cell. In addition, we assign the parallel threads along the longitude direction, and each thread warp is responsible for the computational tasks of the consecutive target cells. Figure 7 shows the parallel threads assignment in HEGrid. The warp number in each row equals $\text{cell_num_one_row} / 32$ (64)⁴. By organizing parallel threads in this way, using warp as the task parallelization unit, we can strengthen the portability of the HEGrid across different GPU architectures. Furthermore, our thread assignment strategy also considers the data reusability of inter-threads. The threads responsible for adjacent target cells can

⁴The warp size (wavefront) in AMD GPU is 64.

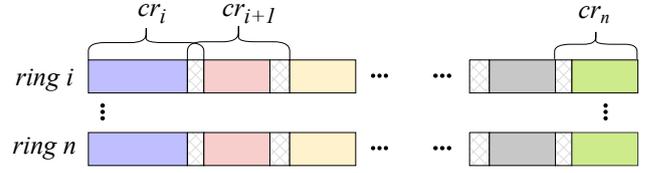


Figure 6: The location in memory of the contribution points for adjacent target cells on different rings. Different colors rectangles represent the contribution regions of different target cells, and shaded rectangles are the contribution points shared by the adjacent target cells.

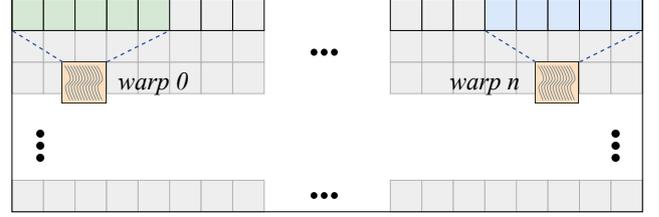


Figure 7: HEGrid parallel threads assignment in GPU. $n \times \text{warp}$ will be responsible for one-row target cells.

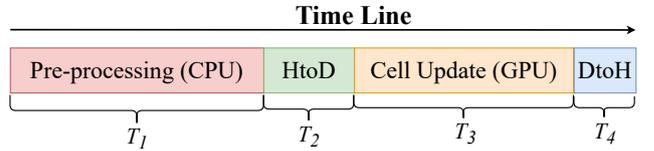


Figure 8: The experimental timeline of the HEGrid pipeline. HtoD and DtoH: "Host to Device" and "Device to Host".

reuse the data cached in the GPU L1/L2 cache.

4.2. Multi Pipeline Concurrency

As discussed in the Section 1, because receivers typically cover a wide range of frequencies, the sky survey of large single-dish radio telescopes collect sky data using a large number of, independent frequency channels. Thus, the data processing in those channels are naturally independent to each other. Combining multi-channel radio astronomical data characteristics, we explore the process-level parallelization and implementation of multi-channel gridding in this section.

4.2.1. Profiling

GPU supports multi-stream parallel execution [8, 13, 23], which can facilitate the HEGrid to realize process-level parallelization on GPU, by dispatching the cell update for different channels to different streams.

We analyzed the time spent at each stage of the HEGrid pipeline. Figure 8 shows our experimental results. The length of the rectangle represents the duration. It can be seen that $T_1 > T_3 > T_2 > T_4$.

The prerequisite of using GPU streams in the grid-

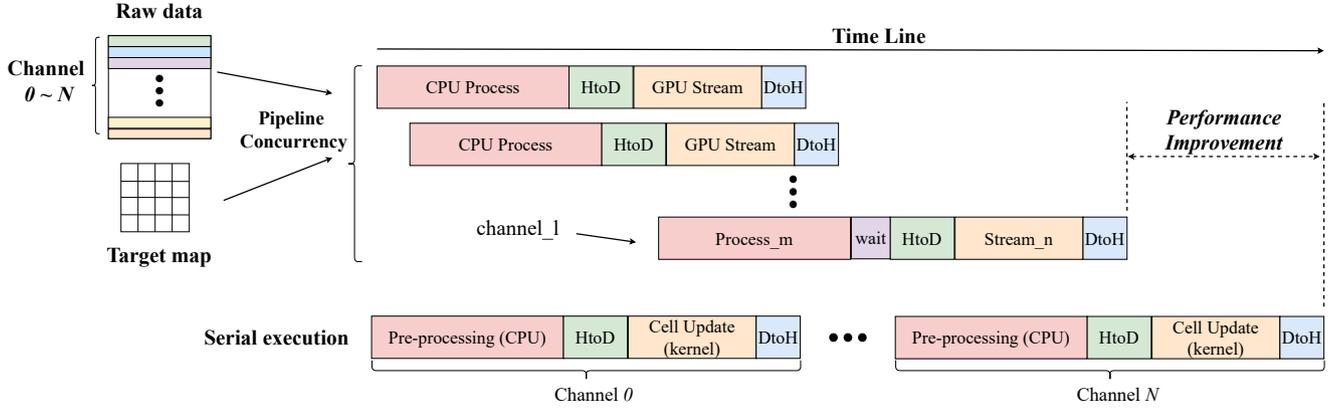


Figure 9: Multi pipeline concurrency on heterogeneous architecture. "wait" or idle stream waiting for data is mainly determined by the GPU's hardware transfer mechanism, where when two adjacent streams in the same direction (e.g., from the host to the device) are requesting to transfer data, the stream which requests first could block the other stream.

ding computation is that each stream executed concurrently should have access to data processed from the CPU, i.e., the CPU has to provide sufficient channels of data to each stream. Therefore, while the CPU is still processing data from multiple channels using sequential execution, multiple GPU streams can execute concurrently only if $T1 + T2 < T3$. Otherwise there will be idle streams waiting for data and degenerated to serial execution. However, the timeline in the HEGrid ($T1 + T2 > T3$) is exactly opposite to the prerequisites (i.e., $T1 + T2 < T3$), which prevented performance improvement using GPU streams.

4.2.2. Pipeline Concurrency and Scheduling

As we analyzed, partial parallelization of the HEGrid pipeline can only achieve in some instances. Otherwise, it will degrade to serialize. In short, if there is concurrency in the pre-processing and data transfer, different GPU kernels can start asynchronously. To achieves process-level parallelization of gridding, we propose multi pipeline concurrency on heterogeneous architectures, shown in Figure 9.

By combining CPU multi-process and GPU multi-stream, we achieve multi pipeline concurrency. The pipeline scheduling needs to be considered at both inter-and intra-pipeline levels. We found through experimental analysis that the data processing time at each stage, such as pre-processing, cell update, and the overall, is similar for different channels. Therefore, the optimal two-level scheduling policy followed by pipeline should be FIFO. As shown in Figure 9, the data from channel_1 will load to the idle process_m, and the idle stream_n will process the data from process_m.

4.3. Pipeline-based Co-optimization

4.3.1. Component Share-based Redundancy Elimination

As mentioned in Section 4.1, data points in different channels with the same coordinate correspond to the same HEALPix pixels. In HEGrid, each pipeline can build up its lookup table and load it from host to device. It's not hard to

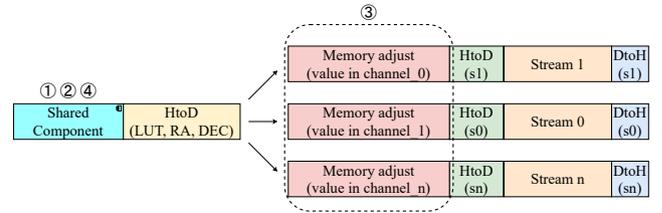


Figure 10: Component share-based redundancy elimination. The shared component is responsible for pixel_idx computation and sort, adjustments of coordinates storage in memory and lookup table construction.

see that this causes the existence of redundant computations and redundant data transfers. We design a "shared component" mechanism to eliminate the duplicate construction of the lookup table. The pre-processing steps plotted in Figure 3 are divided into stages as shown in Figure 10. Steps ①, ②, ④ are assigned to shared components because they can be reused in all pipelines. Furthermore, a fixed size memory was allocated in the device, and the LUT, coordinates, and sampling value were loaded only once from the host to the device. In the concurrency pipeline, the data from the shared component will be broadcasted to the cell update kernel of each pipeline.

4.3.2. Asynchronous Data Transfer and Computation

As the number of concurrency pipelines increases, the number of data exchanges between CPU and GPU will also increase accordingly. In order to reduce the cost of memory allocation, we implement a memory pool that can be reused by the gridding kernel. Each GPU stream can access its fraction of the memory pool through its stream id. We then allocate pinned memory on the host side to obtain the highest bandwidth. Since the host CPU feeds cell update tasks to GPU asynchronously, the computation and data transfer can be overlapped.

4.3.3. Thread-level Data Reuse

In some gridding applications with high sampling densities, the number of neighboring points for one target cell can achieve 90,000, which challenges data transfer between the SM and device memory. From experiment analysis, adjacent cells on the same row have the similar contribution region at high output resolution. To increase the data reusability between different threads, we assign threads to be responsible for the computational tasks of multiple adjacent cells. In addition, the adjacent cells corresponding to the same thread will use the same intermediate results in the computation, such as the contributing rings and starting contributing points, etc.

4.4. Porting to different GPU architectures

With the advent of numerous accelerators with different architectures, hardware portability is becoming a feature that should be present in typical scientific applications. As a data-driven astronomical application, gridding should have portability on different GPU architectures. ROCm⁵ is AMD's open-source software platform for GPU-accelerated high-performance computing and machine learning [1, 15, 18]. An efficient thread assignment scheme is designed for the cell update step on GPU, in Section 4.1.2, to enable HEGrid could adapt to AMD GPU and obtain high performance. Through the ROCm platform, we successfully port HEGrid to the AMD GPU architecture and run it on the Instinct MI50 GPU. In addition, HEGrid can also run on the GPU with AMD's latest CDNA architecture. There are challenges in porting HEGrid to the AMD GPU. First, ROCm does not support texture memory bound, we allocate global memory instead. Second, the performance profiling tools integrated with ROCm are under development and are not yet perfect. Currently, we manually tune the size of thread blocks to get better performance and have not yet done further profiling and optimization of performance based on architecture, which is part of our future work.

5. EXPERIMENTS

In this section, we perform detailed evaluations of HEGrid, in aspects of overall performance, the performance impact of the optimization scheme, the portability on different GPU architectures, and the accuracy of HEGrid. We used both simulated datasets and actual observational datasets from FAST in the experiment and compared the HEGrid to other gridding frameworks.

5.1. Experimental Setup

5.1.1. Hardware Configurations

Experiments are conducted on two servers with different GPU architectures, namely Server_V with Xeon Gold 6151 CPU and NVIDIA V100 GPU, and Server_M with Xeon E5-2620 CPU and AMD MI50 GPU. Their hardware configurations are shown in Table 1.

⁵<https://rocmdocs.amd.com/en/latest/index.html>

⁶<https://github.com/ROCm-Developer-Tools>

5.1.2. Datasets

As performance could be data dependent, we use two different datasets for the performance evaluation, as shown in Table 2. The first is a simulated datasets generated with the observation parameters of FAST. The second is an actual observational datasets collected by FAST. It is worth noting that the simulated dataset differs from the actual dataset with a much larger data size at each channel (10^7 vs 10^6). This is because FAST has not yet completed a full survey of the sky, and it requires multiple repeat scans of the same target sky area to get as complete a picture of the sky as possible. Additionally, FAST will complete a more comprehensive survey in the coming period, and more complete data are not yet available for performance analysis, so we use simulated data with high sampling density (i.e., the datasets with large data sizes) for the performance analysis in some experiments.

5.1.3. Performance Metrics

We use speedup as our main metric to measure the performance changes. In Section 5.2 and 5.4, the speedup is the ratio of the baseline running time to the running time of HEGrid. A state-of-the-art gridding framework with the shortest running time (i.e., the best) would be selected as the baseline. In Section 5.3, the speedup is the ratio of the running time of the non-optimized HEGrid (i.e., baseline) to the running time of the optimized HEGrid.

5.2. Overall Performance

The comparison between HEGrid and other state-of-the-art gridding frameworks is shown in Table 3. It can be seen that HEGrid outperforms other frameworks by up to 5.5x performance speedup in all experiments. This demonstrated the advantage of HEGrid in radio astronomy data gridding.

Specifically, on the simulated dataset, we evaluate the dependence of performance on data size per channel. HEGrid has the best performance compared to Cygrid and HCGrid. It's 5.5x faster than Cygrid. This benefits from our design of HEGrid running on CPU-GPU heterogeneous architectures and the high process-level parallelism in the gridding.

On the FAST's observed data, we evaluate the dependence of performance on the number of frequency channels. HEGrid also outperforms Cygrid and HCGrid. It's 4.3x faster than HCGrid. This demonstrates the effectiveness of our parallelization strategy of multi pipeline concurrency and other optimization techniques.

5.3. Analysis of Performance Optimizations

5.3.1. Redundancy Elimination

Our optimization scheme on the CPU eliminates the duplicate computations in the pre-processing step. We now measure the performance of HEGrid under the optimization of component share-based redundancy elimination. Figure 11 and Figure 12 show the overall performance improvements brought by our scheme on the simulated dataset and FAST's observed data respectively.

First, under the simulated datasets with different data sizes, the average performance improvement brought by the

Table 1

Hardware configurations for experiments.

Server	Server_V		Server_M	
	CPU	GPU	CPU	GPU
Processor				
Model	Xeon Gold 6151	Tesla V100	Xeon E5-2620	Instinct MI50
Transistors	14nm	12nm	22nm	7nm
# Cores	16	5120	32	3840
Base Frequency (MHz)	3000	1245	2100	1200
Max Frequency (MHz)	3400	1380	3000	1746
Device Memory Type	-	HBM2	-	HBM2
Device Memory Clock (MHZ)	-	876	-	1000
Device Memory Bandwidth (GB/s)	-	897	-	1024
Device Memory Size (GB)	-	16	-	16
Host Memory Size (GB)	128	-	128	-

Table 2

Datasets for Experiment.

Dataset	Simulated	Observed (by FAST)
File Format	HDF5	HDF5
Beam Size	180''	180''
Map Size	60° × 20°	60° × 20°
Map Center	(30°, 41°)	(30°, 41°)
Points Num	1.50E + 07 ~ 1.90E + 07	2.83E+06
Channels Num	50	10 ~ 50
File Size	2.91 ~ 3.68 GB	3.32 GB

redundancy elimination is 3.2x. Specifically, compared with Figure 12, performance improvement of redundancy elimination scheme is more obvious for large datasets. This is because the duplicate lookup table construction and the duplicate data loading from the host to the device will be one of the major performance challenges in processing the large datasets. Therefore, the performance benefits of the redundancy elimination strategy will be more obvious. Second, the performance improvement is also evident in the data observed by FAST with different frequency channels, demonstrating the effectiveness of our strategy for multiple pipeline concurrency. In addition, the performance improvement exhibited with a channel count of 50 is slightly lower than that of Figure 11, which further proves the performance advantage of the redundancy elimination strategy for large datasets, the scale of data from FAST will need to handle in the future.

5.3.2. Threads Concurrency on GPU

HEGrid vectorizes the cell update on GPU in a SIMT manner. We evaluate the HEGrid performance by varying the size of the thread blocks on NVIDIA V100 GPU. The data used in this section is the simulated dataset with 1.5×10^7 and 1.9×10^7 data sizes, respectively. Figure 13 shows the HEGrid running time under different thread block sizes. We can observe that before reaching the optimal thread organization configuration (e.g., near 352), the performance improves as the size of the thread block increases. This is because more threads are being scheduled to ex-

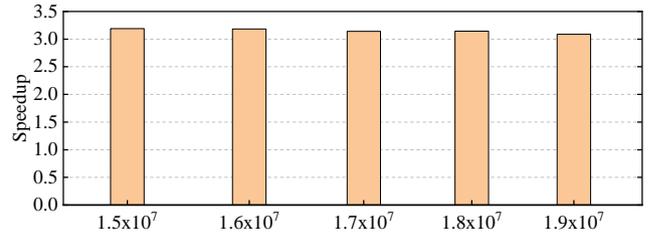


Figure 11: Performance improvement of redundancy elimination scheme under the simulated datasets with different data sizes.

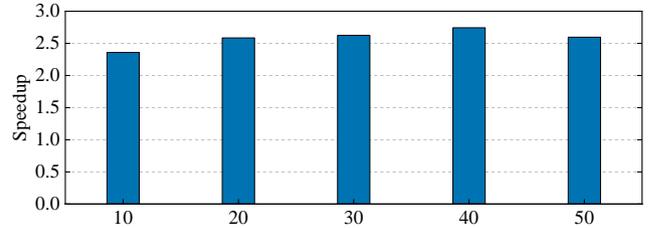


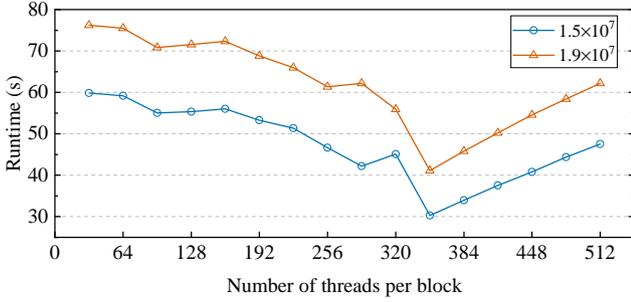
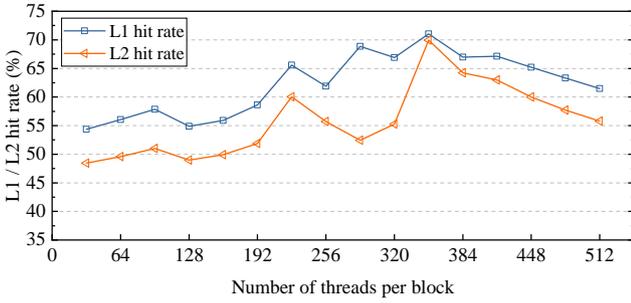
Figure 12: Performance improvements of redundancy elimination scheme under the FAST's observed data with different frequency channels.

ecute on the SM. After that, when the number of threads per block is greater than 352, the running time begins a linear increase again. The reason is that the V100 has a total number of 65,536 registers for each SM, while the HEGrid's kernel uses 88 registers as we analyzed using the nsight-compute tool [24, 30]. It means that the maximum (also optimal) of block sizes that can be scheduled to execute on the SM would be 352, which could schedule two blocks to SM (use $2 \times 352 \times 88 = 61,952$ registers, less than 65,536), that is 2×352 parallel threads execution on the SM. However, even one more warp is added, i.e., the block size is $352+32 = 384$, there would be no more blocks (e.g., two blocks required registers would be $2 \times 384 \times 88 > 65,536$.) to be scheduled to execute on the SM, that is only 384 parallel threads execu-

Table 3

Comparison of the performance of gridding frameworks (Running Time (s)).

Dataset	Simulated					Observed (by FAST)					
	Datasize / Channel num	1.50E+07	1.60E+07	1.70E+07	1.80E+07	1.90E+07	10	20	30	40	50
Cygrid		165.87	171.37	178.99	187.31	194.6	77.77	79.01	80.12	80.97	84.76
HCGrid		173.25	178.82	189.01	196.43	206.28	25.5	52.02	79.58	113.12	137.1
HEGrid		30.21	32.77	35.27	38.2	40.94	7.15	12.77	18.29	24	29.6
Speedup (HEGrid)		5.49	5.23	5.07	4.90	4.75	3.57	4.07	4.35	3.37	2.86


Figure 13: Performance as a function of the size of the thread blocks on NVIDIA V100 GPU.

Figure 14: L1 and L2 hit rate as functions of the size of the thread blocks on NVIDIA V100 GPU.

tion on the SM, and the performance will degrade.

In addition, the GPU L1/L2 cache hit rate changes with the thread block sizes also demonstrate our thread parallelization scheme's effectiveness on GPU. We assign threads on the GPU in a way that takes into account the reusability of data between different threads. As depicted in Figure 14, before reaching the optimal thread organization configuration (e.g., near 352), the hit rates of L1 and L2 increase with the increase of the thread block size. These results show that our thread organization scheme meets our expectations. That is, improving the GPU L1/L2 cache hit rate by organizing parallel threads through improving the inter-thread data reuse rate responsible for adjacent target cells.

5.3.3. Streams Concurrency on GPU

Now, we analyze the performance improvements from the multi-streams concurrency on GPU. Here we have expanded the experimental dataset, i.e., added two sizes of ob-

served sky fields, $5^\circ \times 5^\circ$ and $10^\circ \times 10^\circ$, and two beam widths, $180''$ and $300''$ (A small beam width represents a high output resolution and more target cells on the map). The data size of the extended dataset range from 1.5×10^5 to 1.5×10^7 . Figure 15 shows the performance improvements using different streams compared to the default stream. Here we use "R*-S*" to represent a specific output resolution and sampling density of the experimental data, for example, RH-SH for an output resolution of $180''$ and a sampling size of 1.5×10^7 , RL-SM for an output resolution of $300''$ with a sample size of 1.5×10^6 , and the rest of the cases and so on. Based on the results presented in the figure, one can make the following observations. First, multiple streams can yield significant performance improvements compared to using the default stream, up to 55% in the current experiments, which benefits from the overlapping between different streams. Second, the performance improvement brought by the multiple concurrent streams is more pronounced in the low output resolution and small observation field cases. On the contrary, the performance improvement percentage tends to decrease. This is because at low output resolutions or small observation fields, the number of cells on the output map is smaller, allowing more streams to be concurrent and a higher concurrency between different streams. On the contrary, the computation will change to compute-intensive in the higher output resolutions or larger observation fields, and the concurrency of streams will decrease due to the resource limitation of the GPUs. Third, the performance improvement from multiple concurrent streams is more pronounced at low sample sizes. On the contrary, the computation will change to memory-intensive in the larger sample size, and the concurrency of streams will decrease due to the I/O limitation. In addition, performance improvements tend to flatten out after a threshold number of streams, which is determined by the resources of the device.

Overall, we could conclude that multi-stream concurrency could significantly improve the performance in most cases. Specifically, the optimal concurrent stream configuration needs to be tuned based on the observations, output resolution, and device resources, and dynamically adjusting the stream configuration for optimal performance is part of our future work.

5.3.4. Thread-level Data Reuse

We also propose a thread-level data reuse scheme targeting the high output resolution and large data size. Fig-

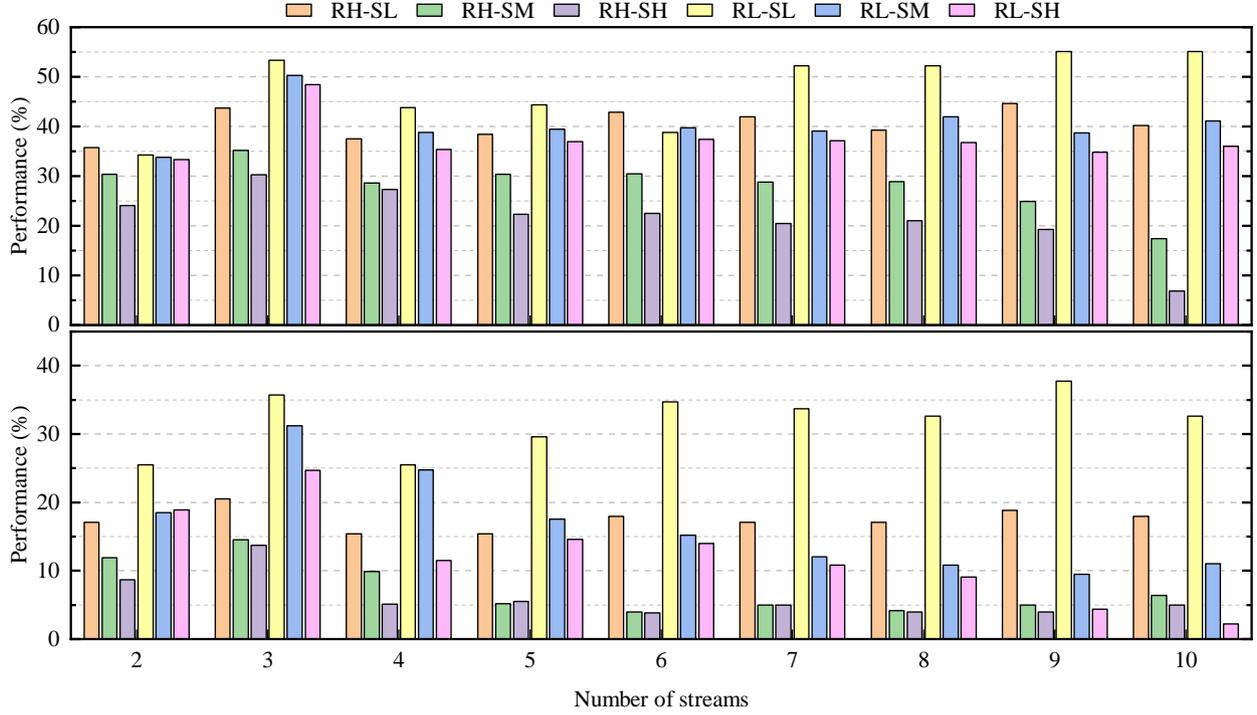


Figure 15: Performance of varied number of streams. The top and bottom represent the results of the experimental analysis for the gridding of two observation sky fields of size $5^\circ \times 5^\circ$ and $10^\circ \times 10^\circ$, respectively. The vertical axis represents the performance improvement with different streams compared to using the default stream (one stream).

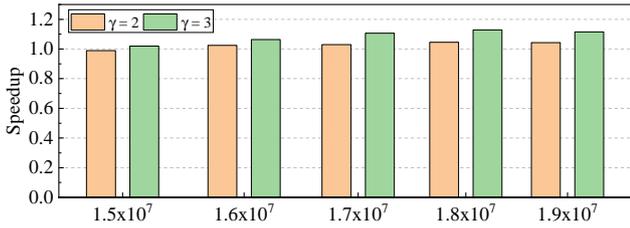


Figure 16: Performance improvement of thread-level data reuse scheme. $\gamma = 2$ and $\gamma = 3$ presents γ adjacent cells shared same contribution region.

ure 16 shows the performance improvement. γ is the reuse factor, representing each thread responsible for γ adjacent grid cells. We can observe that, for large data sizes, thread-level data reuse scheme can achieve up to 1.2x performance speedup. The major reason is that it reduces the workload of the contribution points searching on the host and overhead of data loading between device memory and SM. For example, the computation complexity changes from $\mathcal{O}(N)$ to $\mathcal{O}(N/\gamma)$, N is the number of cells in the target map.

5.4. Performance Portability

To demonstrate the performance portability of HEGrid, using the simulated datasets and FAST actual observation datasets, we also evaluate HEGrid on Server_M with AMD Instinct MI50 GPU. The comparison of HEGrid, and Cygrid is shown in Table 4. Cygrid-16 and Cygrid-32 rep-

resent Cygrid experiments using 16 and 32 CPU cores. We can observe that running on Server_M, HEGrid also presents promising performance, outperforming Cygrid by up to 3.8x performance speedup, which demonstrates the performance portability potential of HEGrid under different GPU architectures. It is undeniable that HEGrid running on Server_M exhibits a performance gap compared to running on Server_V with V100 GPU. The reasons behind this include. First, the limited hardware resources of the MI50 GPU compared to the V100 GPU results in low concurrency of the HEGrid's pipelines. Experimental analysis reveals that for HEGrid, thread blocks can only schedule up to 128 parallel threads concurrently on the MI50 GPU SM to get "relatively better" performance. Second, as introduced in Section 4.4, the performance profiling tools integrated with ROCm are not yet complete, and we currently manually tune the size of thread blocks to obtain better performance. Therefore, besides organizing parallel threads to obtain high performance, no further performance analysis and optimizations have been done based on the architecture, which is part of our future work.

5.5. Accuracy

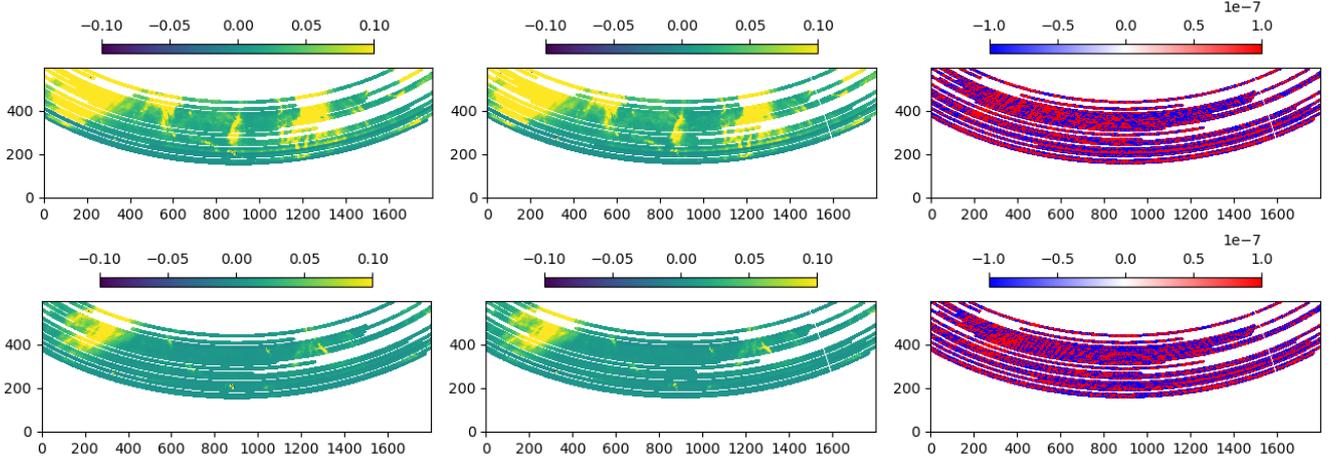
Along with the performance analysis, we evaluate the accuracy of the gridding results by comparing Cygrid and HEGrid results using actual FAST observational data.

Figure 17 shows the real sky images of one of the FAST surveys obtained from the gridding of HEGrid (left) and Cygrid (middle), respectively, and their differences (right) for comparing the accuracy of HEGrid and Cygrid. We can ob-

Table 4

Comparison of the performance of Cygrid and HEGrid (Running on Server_M, Running Time (s)).

Dataset	Simulated					Observed (by FAST)					
	Datasize / Channel num	1.50E+07	1.60E+07	1.70E+07	1.80E+07	1.90E+07	10	20	30	40	50
Cygrid-16		163.15	171.17	177.85	177.95	185.43	86.31	83.16	88.11	85.79	87.04
Cygrid-32		161.95	169.96	178.21	185.98	187.21	83.62	85.35	85.45	87.57	91.24
HEGrid		70.75	75.5	77.42	80	85.62	21.7	50.88	78.16	99.56	125.87
Speedup (HEGrid)		2.29	2.25	2.30	2.22	2.17	3.85	1.63	1.09	0.86	0.71

**Figure 17:** FAST sky images for accuracy comparison. The top and bottom represent two gridding results from two different frequency channels. The gridding results from HEGrid (left), Cygrid (middle) and their difference (right) are presented.

serve that all-sky details can be clearly reconstructed and resolved in both cases and the difference between HEGrid and Cygrid, which is mainly caused by the different hardware architectures, is almost negligible. Overall, we can conclude that HEGrid retains high accuracy but better performance and is a better option for radio astronomical data gridding for large single-dish radio telescopes.

6. Conclusions

Effective and efficient data processing methods are an urgent need to fully exploit the potential of current and upcoming scientific instruments. Gridding is the most computationally intensive step in the data reduction pipeline for data from multiple frequency channels collected by radio telescopes. Fast and high-performance gridding frameworks of multi-channel radio astronomical data for large single-dish radio telescopes are expected to address the challenges.

In this paper, we develop a high efficient and scalable gridding framework, HEGrid, for multi-channel radio astronomical data of the large single-dish radio telescopes. Specifically, we propose and construct the gridding pipeline in CPU-GPU heterogeneous environments and achieve multi-pipeline concurrency. Furtherly, we propose pipeline-based co-optimization strategy to alleviate the potential negative performance impact of possible low intra- and inter-pipeline computation. Our experiments are based

on both simulated datasets and FAST's actual observed datasets. The results show that HEGrid shows very competitive performance compared with other state-of-the-art works.

Our future work plan for the optimization and application of HEGrid includes the following aspects. First, for the ROCm version of HEGrid, we plan to perform detailed performance profiling and optimization based on AMD's GPU architecture, including the latest CNDA architecture, to improve performance further. Second, we plan to achieve architecture-aware optimization, enabling HEGrid could automatically adapt to different heterogeneous architectures (CPU and GPU with different architectures) and obtain high pipeline concurrency. Third, we plan to scale HEGrid to the cluster with multiple GPU accelerators to handle larger-scale datasets. Such as developing a more efficient resource scheduler in HEGrid for processing different batches of observations with varying sampling densities and sky area sizes. In addition, introducing HEGrid into the data reduction pipeline of FAST as a user toolkit.

7. Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

8. Acknowledgments

This work is sponsored by the Joint Research Fund in Astronomy (grant Nos.U1731125, U1731243, U1931130) under a cooperative agreement between the National Natural Science Foundation of China (NSFC) and the Chinese Academy of Sciences, NSFC grant No.11903056; as well as the National Natural Science Foundation of China (grant Nos.61972277).

References

- [1] Abdelfattah, A., Barra, V., Beams, N., Bleile, R., Brown, J., Camier, J.S., Carson, R., Chalmers, N., Dobrev, V., Dudouit, Y., et al., 2021. Gpu algorithms for efficient exascale discretizations. *Parallel Computing* 108, 102841.
- [2] Bekhti, N.B., Flöer, L., Keller, R., Kerp, J., Lenz, D., Winkel, B., Bailin, J., Calabretta, M., Dedes, L., Ford, H., et al., 2016. Hi4pi: a full-sky h i survey based on ebhis and gass. *Astronomy & Astrophysics* 594, A116.
- [3] Bigot-Sazy, M.A., Ma, Y.Z., Battye, R.A., Browne, I.W., Chen, T., Dickinson, C., Harper, S., Maffei, B., Olivari, L.C., Wilkinson, P.N., 2015. Hi intensity mapping with fast. *arXiv preprint arXiv:1511.03006*.
- [4] Blas, J.G., Abella, M., Isaila, F., Carretero, J., Desco, M., 2014. Surfing the optimization space of a multiple-gpu parallel implementation of a x-ray tomography reconstruction algorithm. *Journal of Systems and Software* 95, 166–175.
- [5] Cárcamo, M., Román, P.E., Casassus, S., Moral, V., Rannou, F.R., 2018. Multi-gpu maximum entropy image synthesis for radio astronomy. *Astronomy and computing* 22, 16–27.
- [6] Carrad, G., Sykes, P., Moorey, G., 2006. A cryogenically cooled seven beam 21 cm wavelength receiver front end for the arecibo radio telescope, in: *Proc. Workshop Applications Radio Science*, pp. 15–17.
- [7] Dunning, A., Bowen, M., Castillo, S., Chung, Y.S., Doherty, P., George, D., Hayman, D.B., Jeganathan, K., Kanoniuk, H., Mackay, S., et al., 2017. Design and laboratory testing of the five hundred meter aperture spherical telescope (fast) 19 beam l-band receiver, in: *2017 XXXIInd General Assembly and Scientific Symposium of the International Union of Radio Science (URSI GASS)*, IEEE. pp. 1–4.
- [8] Durrani, S., Chughtai, M.S., Hidayetoglu, M., Tahir, R., Dakkak, A., Rauchwerger, L., Zaffar, F., Hwu, W.m., 2021. Accelerating fourier and number theoretic transforms using tensor cores and warp shuffles, in: *2021 30th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, IEEE. pp. 345–355.
- [9] Fabello, S., Catinella, B., Giovanelli, R., Kauffmann, G., Haynes, M.P., Heckman, T.M., Schiminovich, D., 2011. Alfalfa h i data stacking–i. does the bulge quench ongoing star formation in early-type galaxies? *Monthly Notices of the Royal Astronomical Society* 411, 993–1012.
- [10] Giovanelli, R., Haynes, M.P., Kent, B.R., Perillat, P., Catinella, B., Hoffman, G.L., Momjian, E., Rosenberg, J.L., Saintonge, A., Spekkens, K., et al., 2005. The arecibo legacy fast alfa survey. ii. results of precursor observations. *The Astronomical Journal* 130, 2613.
- [11] Gorski, K.M., Hivon, E., Banday, A.J., Wandelt, B.D., Hansen, F.K., Reinecke, M., Bartelmann, M., 2005. Healpix: A framework for high-resolution discretization and fast analysis of data distributed on the sphere. *The Astrophysical Journal* 622, 759.
- [12] Griffin, A., Ensor, A., 2018. End-to-end modelling of the imaging pipeline in radio astronomy, in: *2018 IEEE 10th Sensor Array and Multichannel Signal Processing Workshop (SAM)*, IEEE. pp. 480–484.
- [13] Jain, T., Cooperman, G., 2020. Crac: checkpoint-restart architecture for cuda with streams and uvm, in: *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, IEEE. pp. 1–15.
- [14] Jung, J., Park, D., Jo, G., Park, J., Lee, J., 2021. Snurhac: A run-time for heterogeneous accelerator clusters with cuda unified memory, in: *Proceedings of the 30th International Symposium on High-Performance Parallel and Distributed Computing*, pp. 107–120.
- [15] Leinhauser, M., Widera, R., Bastrakov, S., Debus, A., Bussmann, M., Chandrasekaran, S., 2022. Metrics and design of an instruction roofline model for amd gpus. *ACM Transactions on Parallel Computing* 9, 1–14.
- [16] Li, D., Wang, P., Qian, L., Krco, M., Dunning, A., Jiang, P., Yue, Y., Jin, C., Zhu, Y., Pan, Z., et al., 2018. Fast in space: considerations for a multibeam, multipurpose survey using china's 500-m aperture spherical radio telescope (fast). *IEEE Microwave Magazine* 19, 112–119.
- [17] Merry, B., 2016. Faster gpu-based convolutional gridding via thread coarsening. *Astronomy and Computing* 16, 140–145.
- [18] Otterness, N., Anderson, J.H., 2020. Amd gpus as an alternative to nvidia for supporting real-time workloads, in: *32nd Euromicro Conference on Real-Time Systems (ECRTS 2020)*, Schloss Dagstuhl-Leibniz-Zentrum für Informatik. pp. 10:1–10:23.
- [19] Romein, J.W., 2012. An efficient work-distribution strategy for gridding radio-telescope data on gpus, in: *Proceedings of the 26th ACM international conference on Supercomputing*, pp. 321–330.
- [20] Träff, J.L., Hunold, S., Mercier, G., Holmes, D.J., 2021. Mpi collective communication through a single set of interfaces: A case for orthogonality. *Parallel Computing*, 102826.
- [21] Veenboer, B., Petschow, M., Romein, J.W., 2017. Image-domain gridding on graphics processors, in: *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, IEEE. pp. 545–554.
- [22] Wang, H., Yu, C., Zhang, B., Xiao, J., Luo, Q., 2021a. Hcgrid: a convolution-based gridding framework for radio astronomy in hybrid computing environments. *Monthly Notices of the Royal Astronomical Society* 501, 2734–2744.
- [23] Wang, J., Zhang, X., Li, Y., Lin, Y., 2021b. Exploring hw/sw co-optimizations for accelerating large-scale texture identification on distributed gpus, in: *50th International Conference on Parallel Processing*, pp. 1–10.
- [24] Wang, P., Wang, J., Li, C., Wang, J., Zhu, H., Guo, M., 2021c. Grus: Toward unified-memory-efficient high-performance graph processing on gpu. *ACM Transactions on Architecture and Code Optimization (TACO)* 18, 1–25.
- [25] Wang, R., Tobar, R., Dolensky, M., An, T., Wicencenc, A., Wu, C., Dulwich, F., Podhorski, N., Anantharaj, V., Suchyta, E., et al., 2020. Processing full-scale square kilometre array data on the summit supercomputer, in: *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, IEEE. pp. 1–12.
- [26] Winkel, B., Lenz, D., Flöer, L., 2016. Cygrid: a fast cython-powered convolution-based gridding module for python. *Astronomy & Astrophysics* 591, A12.
- [27] Yue, Y., Li, D., Nan, R., 2012. Fast low frequency pulsar survey. *Proceedings of the International Astronomical Union* 8, 577–579.
- [28] Zhang, K., Wu, J., Li, D., Krčo, M., Staveley-Smith, L., Tang, N., Qian, L., Liu, M., Jin, C., Yue, Y., et al., 2019. Status and perspectives of the crafts extra-galactic hi survey. *Science China Physics, Mechanics & Astronomy* 62, 1–9.
- [29] Zhao, T., Basu, P., Williams, S., Hall, M., Johansen, H., 2019. Exploiting reuse and vectorization in blocked stencil computations on cpus and gpus, in: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–44.
- [30] Zhou, K., Krentel, M.W., Mellor-Crummey, J., 2020. Tools for top-down performance analysis of gpu-accelerated applications, in: *Proceedings of the 34th ACM International Conference on Supercomputing*, pp. 1–12.
- [31] Zhu, Y., Hou, J., Song, Y., Zheng, Y., Huang, T., Wu, H., 2020. Processing data of correlation on gpu, in: *Big Data in Astronomy*. Elsevier, pp. 139–163.