HyGrid: A CPU-GPU Hybrid Convolution-Based Gridding Algorithm in Radio Astronomy

Qi Luo[1], Jian Xiao[2(✉)], Ce Yu[1], Chongke Bi[2], Yiming Ji[1], Jizhou Sun[1], Bo Zhang[3], and Hao Wang[1]

[1] School of Computer Science and Technology, Tianjin University, 135 Yaguan Rd, Haihe Education Park, Tianjin 300350, China
[2] School of Computer Software, Tianjin University, 135 Yaguan Rd, Haihe Education Park, Tianjin 300350, China
xiaojian@tju.edu.cn
[3] National Astronomical Observatories, Chinese Academy of Sciences, 20A Datun Rd, Chaoyang District, Beijing 100012, China

Abstract. New-generation radio telescopes have been producing an unprecedented scale of data every day and requiring fast algorithms to speedup their data processing work flow urgently. The most data intensive computing phase during the entire work flow is gridding, which converts original data from irregular sampling space to regular grid space. Current methods are mainly focused on interferometers or have limitations on the resolutions due to the memory wall. Here we propose a CPU-GPU hybrid algorithm which accelerates the process of gridding. It employs multi-CPU to perform pre-ordering and GPU to speed up convolution-based gridding. Several optimization strategies are further proposed for reducing unnecessary memory access and maximizing the utilization of the heterogeneous architecture. Testing results demonstrate that the proposal is especially suitable for gridding large-scale data and can improve performance by up to 71.25 times compared to the traditional multi-thread CPU-based approach.

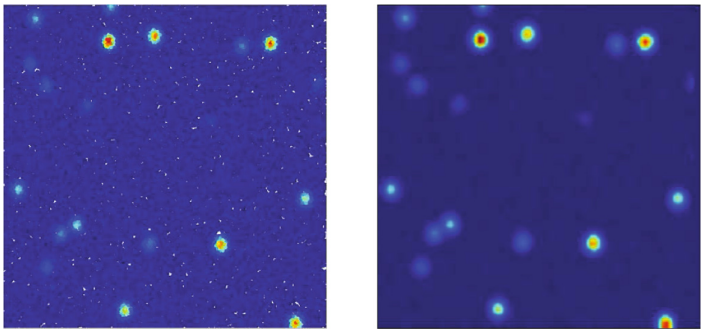
Keywords: Gridding · Heterogeneous computing · Convolution
Data pipeline · Astroinformatics

1 Introduction

In radio astronomy, the original observed data are always irregularly sampled or under-sampled because of the scan pattern. But visualizing the data on pixel-based devices requires a regular grid. Hence, gridding is applied to astronomical data for converting the data from an irregular sampling space to a regular grid space [20]. Figure 1 shows an example of original astronomical data and its output resampled data. Gridding is one of the phases in the pipeline, which involves a major data intensive computing [7,14,31]. As many new large radio telescopes have been established or are under construction, more and more observed data

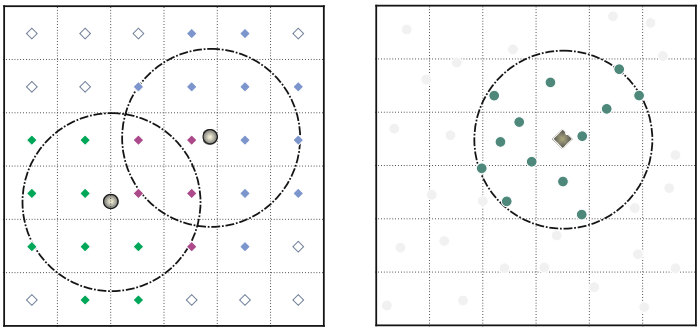
J. Vaidya and J. Li (Eds.): ICA3PP 2018, LNCS 11334, pp. 621–635, 2018.
https://doi.org/10.1007/978-3-030-05051-1_43

are being produced. So they require significant processing capability. For example, the Five-hundred-meter Aperture Spherical radio Telescope (FAST) [27,28] will generate about 40 GB to 1 TB of raw data per hour. But gridding 1 TB of raw data with a traditional multi-thread method on a 6-core CPU takes about 550 h. To reduce storage requirements and speed up observational cosmology research, how to gridding in an efficient way has become a great challenge.



(a) Original observed data (b) Regular resampled data

Fig. 1. An example of gridding irregular data to regular space.



(a) Scatter (b) Gather

Fig. 2. Two approaches of implementing convolution-based gridding. (a) Scatter: each input point (black dot) partially contributes its sampling value to all neighboring output points (rhombuses) within a kernel (dotted circle). (b) Gather: each output point (rhombus) collects its resampling value of all neighboring input points (green dots) within a kernel (dotted circle). (Color figure online)

The gridding problem has been studied extensively in both industry and academia. A number of efficient methods are developed to provide efficient performances, for example, the direct Fourier transform method [21], the back-projection method [9], the matrix inversion method [33] and so on. In radio

astronomy, the convolution-based method [29] is the most commonly adopted method [8]. It divides the sampling plane into an equally spaced linear grid, then assigns each cell (i.e., output point) a value. For each output point, its value is calculated by weighted summing all neighboring sampling points (i.e., input points). To reduce the calculation, a convolution kernel is often applied to limit these potential contributors within a relatively small range. As illustrated in Fig. 2, scatter and gather are two classical approaches to implement the convolution-based gridding. A scatter traverses each original input point and computes its contributions for all output points within the kernel. Vice verse, a gather traverses each output point, finds all neighboring input points within the kernel and calculates the final result by convolving these contributors.

Many previous studies have shown that GPU-based gridding would significantly outperform the CPU-based counterpart [1,10,17,24,32]. They also found the data writing race were unavoidable when parallel a scatter, since multiple input points may contribute to the same output points (e.g., purple rhombuses in Fig. 2(a)) [23]. Although atomic operations can serialize the updates, the performance of a scatter is still fundamentally bounded by the memory wall. On the other hand, a gather will never suffer the race condition, because all operations of an output point are executed by a single thread. Nevertheless, irregular input points result in the access pattern of each output point being less predictable. To reduce unnecessary memory access of non-contributing input points, a gather always requires a pre-ordering process. The pre-ordering will generate a lookup table by hashing the input points to regular bins. Then each output point can quickly determine the bins falling within its kernel and access its neighboring input points. The Hierarchical Equal Area isoLatitude Pixelation (HEALPix) [13] is a widely adopted spherical division scheme in observational cosmology [2,11,15,36].

However, existing proposals are not designed for radio interferometers or have not well considered large-scale data situation such as FAST. To address these problems, this paper presents a CPU-GPU **Hy**brid convolution-based **Grid**ding algorithm, called **HyGrid**. Our main contributions are summarized as following:

- Firstly, a pre-ordering algorithm is presented to build an efficient HEALPix-based lookup scheme on CPU with a $O(N \log(N))$ time complexity and a low memory consumption.
- Then, a gather is further accelerated on GPU with a data layout strategy. Moreover, thread coarsening is adopted on the situation of large output resolution.
- Finally, several cases are feed into HyGrid to verify its validity and feasibility. The results show that HyGrid can solve the convolution-based gridding quickly and efficiently. Compared to the traditional multi-thread approach, it can achieve a speedup by up to 125.81 times on pre-ordering and a speedup by up to 18.63 times on gather. For thread coarsening, the performance can achieve 4.64 times improvement by using it compared to the counterpart.

This paper is structured as follows. In Sect. 2, several related works are briefly discussed. in Sect. 3, there are details about preliminaries and problem descrip-

tion. in Sect. 4, the HyGrid algorithm is described in detail. In Sect. 5, the test data and result are provided. Finally in Sect. 6, it is the conclusions and further work.

2 Related Work

Previous studies related to the convolution-based gridding are introduced and reviewed on three aspects: (1) GPU-based scatters, (2) pre-ordering and gathers, and (3) optimization strategies.

Due to the intrinsic data race, proposed scatters mainly focused on avoiding updating device memory simultaneously. Van Amesfoort et al. [1] developed a scatter on GPU by assigning a small private grid to each input point. Although it can avoid data race fundamentally. It limits the resolution of the output grid and cannot be applied to the latest large radio telescopes. Based on memory copy, Humphreys and Cornwell [5,6,17] also implemented a scatter for the Australian Square Kilometre Array Pathfinder (ASKAP) [16] on GPU. To reduce device memory access, Romein [32] designed anther GPU-based scatter. For each antenna pair in a radio telescope interferometer, its sampling coordinates changes slowly in the sampling space. Hence, using a dedicated thread to iterate over the sampling trajectories will lead to tiny shifts in the kernel. So the thread can accumulate weighted sums in its register as long as possible and write the sums to device memory when necessary. Merry [24] further improved Romein's algorithm by using thread coarsening [26]. The improved version significantly reduces the addressing overhead of device memory. However, both methods heavily depend on the spatial coherence of the interferometers' data and cannot be adopted by single dishes.

Proposed gathers are mainly aiming at reducing unnecessary memory access of non-contributing input points. Edgar et al. [10] designed a gather for the Murchison Widefield Array (MWA) [37] on GPU. It uses Thrust [3] to pre-order input points into 24×24 bins. Then each output point can only search the neighboring input points within the bin it falls in and its adjacent eight bins. Although the pre-ordering leads to an efficient device memory access, the coarse binning strategy makes that a big part (i.e., 8/9) of searched input points still cause unnecessary memory addressing overheads and convolving calculations. Gai et al. [12] presented a gather for magnetic resonance imaging data [18] on GPU by pre-ordering data with a compact binning method. Winkel et al. [40] proposed another compact binning method based on HEALPix. Their implementation is on CPU, called cygrid. It pre-orders astronomical data by using the C++ STL vector [30] to convert an input-output hash map to an output-input hash map. For large-scale data, the data layout and the pre-ordering strategy will significantly increase time overhead and memory consumption. Hence, it is not suitable for gridding large-scale data.

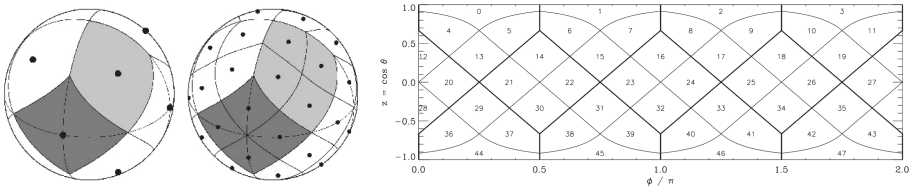
In addition, researches also focused on improving GPU's efficiency and allowing synergies between the CPU and GPU. Amesfoort's scatter [1] accesses the convolution kernel as a 1D texture and loads common data into shared memory

for all threads in a block only once. It can efficiently utilize memory bandwidth. But it is not appropriate for neither large kernels nor varying kernels, because different input points may use an identical convolution kernel. Merry [24] adopted a trade-off method between thread coarsening and parallelism efficiency. While Gai et al. [12] improved load balance by partitioning the gridding operators evenly between GPU and CPU.

3 System Model

3.1 Preliminaries

When applying the FITS World Coordinate System (WCS) [25,39], HEALPix is suitable to bin the spherical sampling space [4]. Given a grid resolution (denoted by $N_{side} \in \{1, 2, 4, 8, \cdots\}$), HEALPix hierarchically subdivides the sphere surface into $12N_{side}^2$ equal-area pixelization. As illustrated in Fig. 3, all pixel centers are placed on $4N_{side} - 1$ rings of constant latitudes, and are equidistant in azimuth (on each ring). Then, these pixels can be simply indexed by moving down from the north to the south pole along each ring.



(a) Subdivided sphere surface (b) Pixels index in RING scheme

Fig. 3. The HEALPix partition [13]. (a) It is hierarchically subdivided with resolution $N_{side} = 1$ and 2 respectively; (b) $12N_{side}^2$ pixels are located at $4N_{side} - 1$ isolatitude rings with resolution $N_{side} = 2$.

HEALPix software libraries[1] support spherical transformations. For instance, the WCS coordinates (denoted by (α, β) with α being the longitude and β being the latitude) of any pixel center (denoted by pix) can be computed by $(\alpha, \beta) \leftarrow Pix2Loc(pix, N_{side})$; the reverse transformation is $pix = Loc2Pix(\alpha, \beta, N_{side})$; the ring (denoted by rix) to which pix belongs can be calculated by $rix = Pix2ring(pix, N_{side})$; the lowest numbered pixel which is located at rix can be calculated by $pix = Ring2Start(rix, N_{side})$.

3.2 Problem Description

Suppose N input points $\mathbb{S} = \{s_1, s_2, \cdots, s_N\}$ are distributed on a sampling plane, where each input point $s_n \in \mathbb{S}$ is located at coordinates (α_n, β_n) with a

[1] http://healpix.sourceforge.net/.

value $V[s_n]$ representing its sampling value. After dividing the plane into $I \times J$ regular grid cells (denoted by $\mathbb{G} = \{g_{1,1}, g_{1,2}, \cdots, g_{I,J}\}$), where the center of each grid cell $g_{i,j} \in \mathbb{G}$ is located at coordinates $(\alpha_{i,j}, \beta_{i,j})$, convolution-based gridding will assign $g_{i,j}$ an resampling value (denoted by $V[g_{i,j}]$) and an overall weighting value (denoted by $W_{i,j}$).

Given a weighting function (denoted by w), the convolution kernel of $g_{i,j}$ will be,

$$V[g_{i,j}] = \frac{1}{W_{i,j}} \sum_n V[s_n] w(\alpha_{i,j}, \beta_{i,j}; \alpha_n, \beta_n), \tag{1}$$

where

$$W_{i,j} = \sum_n w(\alpha_{i,j}, \beta_{i,j}; \alpha_n, \beta_n). \tag{2}$$

Here, s_n be a contributed input point, $w(\alpha_{i,j}, \beta_{i,j}; \alpha_n, \beta_n)$ be a weight that s_n contributes to $g_{i,j}$, $W_{i,j}$ be the overall weight of all contributors. $W_{i,j}$ be used to conserve flux density, because different output points will be influenced by different contributors in varying degrees. In most cases, w be a radially symmetric function, for example, a Gaussian kernel with standard deviation σ. Such that,

$$w(\alpha_{i,j}, \beta_{i,j}; \alpha_n, \beta_n) = \exp\left[\frac{-d^2(\alpha_{i,j}, \beta_{i,j}; \alpha_n, \beta_n)}{2\sigma^2}\right], \tag{3}$$

where d be the shortest distance between s_n and $g_{i,j}$ measured along the surface of the spherical sky [35,38]. Hence, partial contributions denoted by each contributor can be concentrated towards the center of the kernel. The chosen of the radial-symmetric function is beyond the scope of the paper, more detailed analysis can be found at [19].

4 HyGrid Algorithm

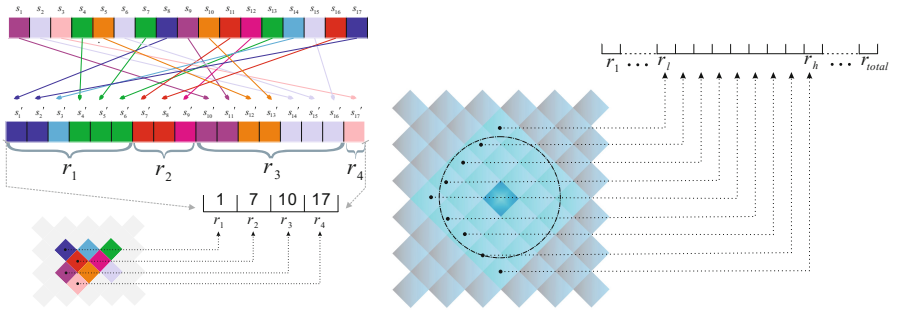
In this section, we present a CPU-GPU hybrid gridding algorithm. To limit the search space, the algorithm first pre-orders the input points using multi-CPU. Then the proposal parallelizes the convolution by assigning each output point a dedicated thread on GPU. It further optimizes gridding using data layout and thread coarsening.

4.1 Pre-ordering

4.1.1 Ring-Based Lookup Strategy

$\forall s_n \in \mathbb{S}$, given $S_{pix}[n] = Loc2Pix(\alpha_n, \beta_n, N_{side})$ be the pixel to which s_n belong, $rix_n = Pix2ring(S_{pix}[n], N_{side})$ be the ring at which $S_{pix}[n]$ be located. Different pixels may located at the same ring, i.e., $rix_i = rix_j$ with $i \neq j$. Let $rix_1 = r_1$ and $rix_N = r_{total}$, we have $\mathbb{R} = \{r_1, r_2, \cdots, r_{total}\}$ be a set of rings at which S_{pix} be located.

Since guaranteeing the output grid image resolution requires a considerable number of pixels. The rearranged input points (denoted by $\mathbb{S}'$) will be stored in



(a) Pre-ordering strategy (b) Searching strategy

Fig. 4. Ring-based lookup table. (a) It sorts input points by their pixels (i.e., boxes in different colors) and generates a lookup table based on their rings; (b) it searches the neighboring input points for a given pixel by using the lookup table. (Color figure online)

a compact way on GPU's device memory. Hence, we cannot afford to directly search a pixel among $\mathbb{S}'$. To address the problem, we propose an efficient ring-based lookup table (denoted by R_{start}). It records the lowest indexed pixels which are located at each ring. Specifically, $S_{pix}[R_{start}[k]]$ be the lowest indexed pixel in S_{pix} which also be located at ring $r_k \in \mathbb{R}$. Such that $R_{start}[k] = n$ if and only if $rix_{n-1} < r_k$ and $rix_n \geq r_k$.

Therefore, given pixel pix with $rix = Pix2Ring(pix, N_{side})$, the problem of finding its contributors within a kernel radius (denoted by rad) becomes how to calculate the rings which are rad distant from rix, where $r_l = Loc2Pix(\alpha_{i,j}, \beta_{i,j} - rad, N_{side})$ and $r_h = Loc2Pix(\alpha_{i,j}, \beta_{i,j} + rad, N_{side})$ denote lowest and highest indexed rings, respectively. Then, for each r_k between r_l and r_h, we can binary search the lowest indexed neighboring pixel within the range $[R_{start}[k], R_{start}[k+1]]$ in S_{pix}. Due to the compact relationship among pixels, the following elements in S_{pix} will be continuously contributed to pix. Such that, if $S_{pix}[n]$ and $S_{pix}[n+2]$ are contributors and $rix_n = rix_{n+2}$, then $S_{pix}[n+1]$ will also be a contributor with $rix_{n+1} = rix_n$. So performing lookup only once, we can find all contributors located at rix_n and access them by sequential scanning $\mathbb{S}'$.

Figure 4(a) shows an example of generating R_{start}. In this example, 17 input points are binned into 9 pixels and rearranged according to the pixels' indexes in RING scheme. Then we have $R_{start}[1] = 1$, $R_{start}[2] = 7$, $R_{start}[3] = 10$ and $R_{start}[4] = 17$. Figure 4(b) illustrates an example of lookup with R_{start}.

4.1.2 Pre-ordering Algorithm on Multi-CPU

Given N_{side} and S_{pix}. The HEALPix-based Pre-ordering Algorithm (denoted by *HpxPreOrdering*) applies a parallel block indirect sort algorithm[2] to rearrange input points. The sort algorithm is represented by

[2] https://boost.org.

$BlockIndirectSort(S_{pix}, S_{val}, N)$ with $S_{val}[n] = n$ being the sort value. In addition, our early attempts have shown that the CPU-based block indirect sort algorithm outperformed GPU-based Thrust [3] on sorting large-scale data.

Details of *HpxPreOrdering* are described in Algorithm 1. There are two steps: sorting (Line 1–4) and lookup-table-generating (Line 5–13). In step 1, *BlockIndirectSort* is performed on multi-core CPU to sort S_{val} by S_{pix} in parallel (Line 1), then we rearrange $\mathbb{S}$ to $\mathbb{S}'$ according to the ordered S_{val} (Line 2–4). In step 2, the algorithm first computes r_1 and r_{total} (Line 5–6), then updates R_{start} by iterating over $\mathbb{R}$ and doing binary search in S_{pix} (Line 7–13). Here, given a non-decreasing array (denoted by $Array$), $BiSearchLastPosLessThan(Array, left, right, key)$ will binary search key within the range of $[left, right]$ in $Array$ and return an index i satisfying $Array[i] < key$ and $Array[i+1] \geq key$.

The complexity of *HpxPreOrdering* is decided by the block indirect sort algorithm, which is $O(N \log(N))$ at the worst time and $O(N)$ at the best time.

4.2 Gather Algorithm on GPU

Algorithm 2 illustrates the GPU-based Gridding Gather (denoted by *GPUGridGather*). The entire computation of $V[g_{i,j}]$ is executed by a single thread. The thread iterates from r_l to r_h for finding all contributors of $g_{i,j}$ and performing Eq. 1 to obtain $V[g_{i,j}]$ and $W_{i,j}$ (Line 1–17). For each ring (denoted by $cntRix$), the thread calculates its latitude β (Line 2–3) to obtain the lowest and the highest indexed neighboring pixels (denoted by p_l and p_h, respectively) located at $cntRix$ (Line 4–6), where $tanRad$ is the great-circle distance between $(\alpha_{i,j}, \beta)$ and p_l (or p_r). Secondly, the thread binary searches p_l in S_{pix} with the search space restricted by R_{start} (Line 7–9). Finally, the thread can sequential scan $\mathbb{S}'$ and apply Eq. 1 (Line 10–16).

The calculation of Function GPUGridGather$g_{i,j}$ is actually linear to the number of contributors, although there exist double nested loops.

4.3 CUDA Optimization Strategy

We further propose two optimization strategies while implementing HyGrid in CUDA 8.0 [34]: (1) using different device memory types to optimize the data layout; (2) using thread coarsening to accelerate gather in the situation of the larger output resolution.

To optimize memory access, the data layout strategy manipulates $\mathbb{S}$ and $\mathbb{G}$ in the structure of an array format, accesses the lookup table as a 1D texture memory, as well as accesses global parameters (such as N, I, J, N_{side} and so on) as constant memory. Furthermore, partial convolving results of a thread are maintained in a register and only flushed to global memory once after all calculation.

In the situation of a large output resolution, although the same scale of threads could be generated by GPU processors. But these threads cannot be

Algorithm 1. HEALPix-based Pre-ordering Algorithm: *HpxPreOrdering*

Parameter: N_{side}
input: $\mathbb{S}$, S_{pix}, S_{val}
output: $\mathbb{S}'$, S_{pix}, R_{start}, r_1, r_l, r_h
Initialize: $\mathbb{S}' \leftarrow \varnothing$, $R_{start} \leftarrow \varnothing$

▷ sort input points

1: $BlockIndirectSort(S_{pix}, S_{val}, N)$
2: **for** $n = 1$ **to** N **do**
3: $\quad \mathbb{S}'[n] \leftarrow \mathbb{S}[S_{six}[n]]$
4: **end for**

▷ generate lookup table

5: $r_1 = Pix2Ring(S_{pix}[1], N_{side})$
6: $r_{total} = Pix2Ring(S_{pix}[N], N_{side})$
7: $R_{start}[0] \leftarrow 1$, $idx = 1$
8: **for** $cntRix = r_1 + 1$ **to** r_{total} **do**
9: $\quad pix = Ring2Start(cntRix, N_{side})$
10: $\quad cntIdx = BiSearchLastPosLessThan(S_{pix}, idx, N, pix)$
11: $\quad R_{start}[cntRix - r_1] = cntIdx + 1$
12: $\quad idx = cntIdx$
13: **end for**

Algorithm 2. GPU-based Gridding Gather: *GPUGridGather*

Parameter: N_{side}, rad
input: $\mathbb{S}'$, S_{pix}, r_1, R_{start}, r_1, r_l, r_h
output: $W_{i,j}$, $V[g_{i,j}]$
Initialize: $W_{i,j} = 0$, $V[g_{i,j}] = 0$
1: **for** $cntRix = r_l$ **to** r_h **do**

▷ iterate each neighboring ring

2: $\quad pix = Ring2Start(cntRix, N_{side})$
3: $\quad (\alpha, \beta) \leftarrow Pix2Loc(pix, N_{side})$
4: $\quad tanRad = Rad2Tan(\beta_{i,j}, \beta, rad)$
5: $\quad p_l = Loc2Pix(\alpha_{i,j} - tanRad, \beta, N_{side})$ ▷ lowest pixel
6: $\quad p_h = Loc2Pix(\alpha_{i,j} + tanRad, \beta, N_{side})$ ▷ highest pixel
7: $\quad lIdx = R_{start}[cntRix - r_1]$
8: $\quad rIdx = R_{start}[cntRix - r_l + 1]$
9: $\quad n = BiSearchLastPosLessThan(S_{pix}, lIdx, rIdx, p_l) + 1$

▷ calculate Equation 1

10: $\quad$ **while** $S_{pix}[n] \leq p_h$ **do**
11: $\quad\quad$ **if** $d(\alpha_{i,j}, \beta_{i,j}; \alpha_n, \beta_n) \leq rad$ **then**
12: $\quad\quad\quad W_{i,j} = W_{i,j} + w(\alpha_{i,j}, \beta_{i,j}; \alpha_n, \beta_n)$
13: $\quad\quad\quad V[g_{i,j}] = V[g_{i,j}] + w(\alpha_{i,j}, \beta_{i,j}; \alpha_n, \beta_n) * V[s'_n]$
14: $\quad\quad$ **end if**
15: $\quad\quad n = n + 1$
16: $\quad$ **end while**
17: **end for**

executed simultaneously due to the limited processing capability on each GPU processor. We also notice that output points located at the same ring usually have similar neighboring rings. Accordingly, we apply thread coarsening to collaborate the computations of several continuous output points located at the same ring into one thread. In particular, each thread executes *GPUGridGather* only once and its corresponding output points share the same p_l, determined by the "first" output point. Furthermore, partial convolving results of different output points would be remained on the different registers. The performance of the thread coarsening strategy will depend on the number of output points calculated by each thread, called coarsening factor (denoted by γ).

5 Benchmarking

Let output resolution (denoted by O_r) be the width of each output point. For a given sampling field, a small output resolution indicates a big I (or J). For instance, let $O_r = 200''$ and the sampling field be $5^\circ \times 5^\circ$, we have $I \times J = (5^\circ/200'') \times (5^\circ/200'') = 90 \times 90$. Several experiments are conducted to evaluate the performance of HyGrid with different N and O_r. To better illustrate the performance of HyGrid, we compare it against cygrid [40], which is one of the fastest gridding implementations to the best of our knowledge. The experimental environment is shown in Table 1. Specifically, cygrid and *HpxPreOrdering* are executed on a 6-core CPU with 12 threads per core. *GPUGridGather* is executed by $I \cdot J/\gamma$ threads on GPU with each thread being responsible for γ output points.

Table 1. Experimental environment

	Brand	Model	Cores	SMs	Tflops	RAM (GB)	Freq. (GHz)	Cache (MB)	Thread pre core
GPU	NVIDIA	Tesla K40	2880	15	1.4	12	-	-	-
CPU	Intel	Xeon E5-2620	6	-	-	-	2.5	15	12

5.1 Performance vs. Input Size

Figure 5(a) illustrates the performance of gridding as a function of N, where the sampling field is $5^\circ \times 5^\circ$ with $O_r = 600''$ (i.e., $I \times J = 90 \times 90$) and $\gamma = 1$. The results indicate that HyGrid outperforms the multi-thread cygrid with a speedup of 125.81 times on pre-ordering, a speedup of 18.63 times on gather and a speedup of 71.25 times on the gridding process overall. As N increasing, the pre-ordering strategy of cygrid is strictly limited by the memory wall. The number of contributors and convolving computations of each thread are also significantly increased. Thereby, *GPUGridGather* can obtain higher performance and data throughput, especially when $N > 10^8$.

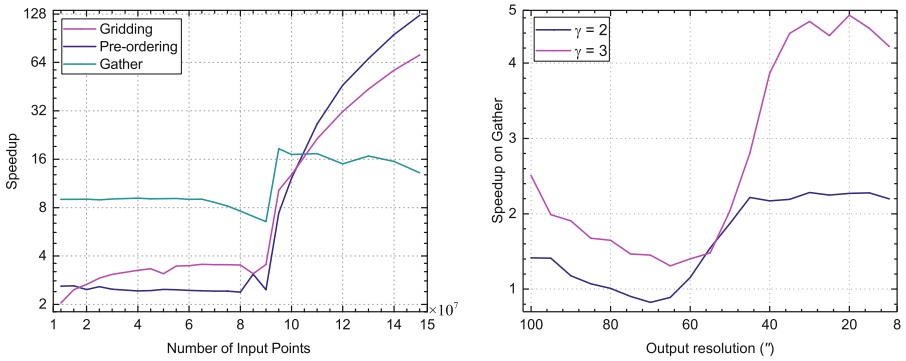


Fig. 5. The speedup of performance. (a) Gridding a different number of input points to a $5^\circ \times 5^\circ$ sampling filed with $O_r = 200''$ ($I \times J = 90 \times 90$); (b) gridding 1.5×10^8 input points to a $5^\circ \times 5^\circ$ sampling filed with different O_r.

5.2 Performance vs. Thread Coarsening

To analyze the efficiency and the feasibility of the thread coarsening strategy, we select a benchmark model of a fixed N and different O_r. Figure 5(b) shows the performance as a function of O_r with 1.5×10^8 input points and a $5^\circ \times 5^\circ$ sampling field. The results show that the strategy can achieve significant performance improvement even if I (or J) is very large. Compared to executing *GPU-GridGather* with $\gamma = 1$, we find that thread coarsening can achieve a speedup of 2.28 times with $\gamma = 2$, $O_r = 30''$ ($I \times J = 600 \times 600$) and a speedup of 4.64 times with $\gamma = 3$, $O_r = 20''$ ($I \times J = 900 \times 900$).

We also found that the trade-off between thread coarsening and parallel efficiency partly depends on the output resolution and the hardware. Hence, γ should be well-chosen in actual use. For example, a good γ should be 2 when gridding onto 327×327 ($O_r = 55''$) output grid cells, while 514×514 ($O_r = 35''$) output grid cells prefer $\gamma = 3$. In addition, because the shape of the convolution kernel is actually rectangular in thread coarsening strategy, the experiments with $\gamma = 4$ and 5 may result in "bad" output images. Such that original circular astronomical sources (as showed in Fig. 1) become rectangular sources after the gridding process. Therefore, we do not recommend a big γ, although it can bring about a high speedup of performance.

5.3 Performance vs. Order Degree

In practice, radio astronomical observations are usually taken by using the On-The-Fly (OTF) observing technique [22], where sampling value and position information are recorded continuously. Hence, some original astronomical data should be partial ordered. Some experiments are conducted to demonstrate the performance at such situation. Figure 6 shows the time overhead ratio of pre-ordering to gather, when respectively gridding random input points and ordered

input points. The random data are from the above experiments. The ordered data is constructed by simulating the OTF observing technique. For random input data, the results indicate that *HpxPreOrdering* consumed more running time compared to *GPUGridGather* as a function of N or O_r. While for ordered input data, the most time-consuming process is *GPUGridGather*. Thereby, the order degree of original data has a big impact on load balance between CPU and GPU.

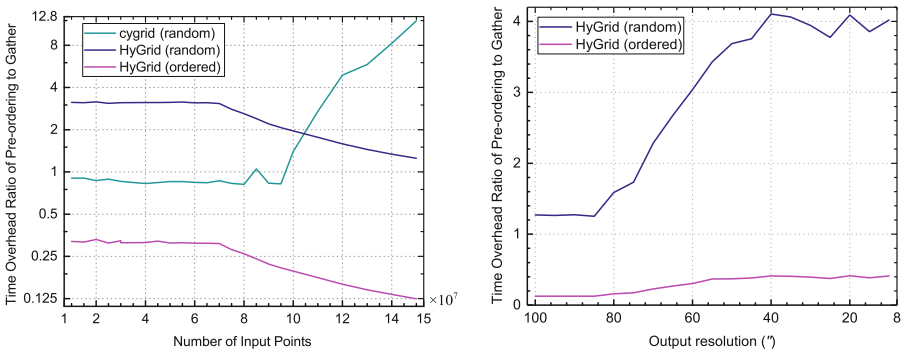


Fig. 6. The time overhead ratio of pre-ordering to gather. (a) Gridding a different number of input points to a $5^\circ \times 5^\circ$ sampling filed with $O_r = 200''$ ($I \times J = 90 \times 90$); (b) gridding 1.5×10^8 input points to a $5^\circ \times 5^\circ$ sampling filed with different O_r.

5.4 Performance in Practice

In addition, the time overhead of using HyGrid (with $\gamma = 3$) to gridding 1.7 GB of ordered (or random) data (nearly $N = 1.5 \times 10^8$) onto a 900×900 grid image is within 8 s (or 32 s). Approximately, gridding 1 TB of ordered (or random) data takes about 1.34 h (or 5.35 h). Accordingly, executing HyGrid on a single workstation is qualified for small-scale data, while a small cluster is enough for large-scale data by running multiple HyGrid simultaneously. In order to achieve a best trade-off between performance and power saving, the number of CPU and GPU should be well-chosen according to the order degree of input data.

6 Conclusion and Future Work

Due to the irregularity and the large-scale astronomical data, simply transferring the convolution-based gridding to GPU cannot obtain a good performance. Here we propose a hybrid approach. First an efficient lookup table based on HEALPix is introduced in the pre-ordering step. Then a GPU-accelerated gather is implemented by utilizing data layout and thread coarsening in a proper manner.

Experiments have shown that our proposal provides a significant improvement in performance, and can be applied in practical data pipelines for both large interferometer and single dish.

However, current implementation has not yet supported auto-tuning to find the optimized coarsening factor. Also, there need some improvements in load balance between CPU and GPU. Future research will mainly focus on extending the gridding approach to multiple GPUs platform, adapting it to real-time data stream, as well as integrating it into the data pipeline of FAST.

Acknowledgments. The authors would like to thank Benjamin Winkel for providing the Cython code of the *cygrid* method.

This work is supported by the Joint Research Fund in Astronomy (U1731125, U1531111) under a cooperative agreement between the National Natural Science Foundation of China (NSFC) and Chinese Academy of Sciences (CAS). This work is also supported by the Young Researcher Grant of National Astronomical Observatories, Chinese Academy of Sciences.

References

1. van Amesfoort, A.S., Varbanescu, A.L., Sips, H.J., van Nieuwpoort, R.V.: Evaluating multi-core platforms for HPC data-intensive kernels. In: Proceedings of the 6th ACM conference on Computing frontiers, CF 2009, pp. 207–216. ACM, New York (2009)
2. Baron, F., Kloppenborg, B., Monnier, J.: Toward 5D image reconstruction for optical interferometry, vol. 8445. Amsterdam, Netherlands (2012)
3. Bell, N., Hoberock, J.: Thrust: a productivity-oriented library for CUDA. In: Hwu, W.W. (ed.) GPU Computing Gems. Applications of GPU Computing Series, Jade edn, pp. 359–371. Morgan Kaufmann, Boston (2012)
4. Calabretta, M.R., Roukema, B.F.: Mapping on the healpix grid. Mon. Not. R. Astron. Soc. **381**(2), 865–872 (2007)
5. Cornwell, T.J., Golap, K., Bhatnagar, S.: W projection: a new algorithm for wide field imaging with radio synthesis arrays. In: Astronomical Data Analysis Software and Systems XIV. Astronomical Society of the Pacific Conference Series, vol. 347, p. 86 (12 2005)
6. Cornwell, T.J., Golap, K., Bhatnagar, S.: The noncoplanar baselines effect in radio interferometry: the W-projection algorithm. IEEE J. Sel. Top. Signal Process. **2**(5), 647–657 (2008)
7. De, K., Gupta, Y.: A real-time coherent dedispersion pipeline for the giant metrewave radio telescope. Exp. Astron. **41**(1), 67–93 (2016)
8. Dickey, J.M.: Spectral line advanced topics. In: Single-Dish Radio Astronomy: Techniques and Applications. Astronomical Society of the Pacific Conference Series, vol. 278, pp. 209–225 (2002)
9. Dudgeon, D.E., Mersereau, R.M.: Multidimensional Digital Signal Processing. Prentice Hall Signal Processing Series. Prentice-Hall (1984)
10. Edgar, R., et al.: Enabling a high throughput real time data pipeline for a large radio telescope array with GPUs. Comput. Phys. Commun. **181**(10), 1707–1714 (2010)

11. Fernique, P., Durand, D., Boch, T., Oberto, A., Pineau, F.: HEALpix based cross-correlation in astronomy. In: Astronomical Data Analysis Software and Systems XXII. Astronomical Society of the Pacific Conference Series, vol. 475, p. 135 (2013)
12. Gai, J., et al.: More IMPATIENT: a gridding-accelerated Toeplitz-based strategy for non-Cartesian high-resolution 3D MRI on GPUs. J. Parallel Distrib. Comput. **73**(5), 686–697 (2013)
13. Górski, K.M., et al.: HEALPix: a framework for high-resolution discretization and fast analysis of data distributed on the sphere. Astrophys. J. **622**(2), 759 (2005)
14. Giovanelli, R., Haynes, M.P., Kent, B.R., et al.: The arecibo legacy fast ALFA survey: I. Science goals, survey design, and strategy. Astrophys. J. **130**(6), 2598 (2005)
15. Hong, Z., Yu, C., Wang, J., Xiao, J., Cui, C., Sun, J.: Aquadexim: highly efficient in-memory indexing and querying of astronomy time series images. Exp. Astron. **42**(3), 387–405 (2016)
16. Hotan, A.W., et al.: The Australian square kilometre array pathfinder: system architecture and specifications of the boolardy engineering test array, vol. 31, p. e041. Publications of the Astronomical Society of Australia (2014)
17. Humphreys, B., Cornwell, T.: SKA memo 132: analysis of convolutional resampling algorithm performance (2011)
18. Hwu, W.M.W., et al.: Accelerating MR image reconstruction on GPUs. In: 2009 IEEE International Symposium on Biomedical Imaging: From Nano to Macro, pp. 1283–1286 (2009)
19. Jackson, J.I., Meyer, C.H., Nishimura, D.G., Macovski, A.: Selection of a convolution function for Fourier inversion using gridding. IEEE Trans. Med. Imaging **10**(3), 473–478 (1991)
20. Léna, P., Rouan, D., Lebrun, F., Mignard, F., Pelat, D., Lyle, S.: Observational Astrophysics. Astronomy and Astrophysics Library, 3rd edn. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-21815-6
21. Maeda, A., Sano, K., Yokoyama, T.: Reconstruction by weighted correlation for MRI with time-varying gradients. IEEE Trans. Med. Imaging **7**(1), 26–31 (1988)
22. Mangum, J.G., Emerson, D.T., Greisen, E.W.: The on the fly imaging technique. A&A **474**(2), 679–687 (2007)
23. McCool, M., Reinders, J., Robison, A.: Structured Parallel Programming: Patterns for Efficient Computation, 1st edn. Morgan Kaufmann Publishers Inc., San Francisco (2012)
24. Merry, B.: Faster GPU-based convolutional gridding via thread coarsening. Astron. Comput. **16**, 140–145 (2016)
25. Mink, D.: WCSTools 4.0: Building Astrometry and Catalogs into Pipelines. In: Astronomical Data Analysis Software and Systems XV. Astronomical Society of the Pacific Conference Series, vol. 351, p. 204 (2006)
26. Muscat, D.: High-performance image synthesis for radio interferometry (2014)
27. Nan, R.: Five hundred meter aperture spherical radio telescope (fast). Sci. China Ser. G **49**(2), 129–148 (2006)
28. Nan, R., et al.: The five-hundred-meter aperture spherical radio telescope (fast) project. Int. J. Mod. Phys. D **20**(06), 989–1024 (2011)
29. O'Sullivan, J.D.: A fast sinc function gridding algorithm for Fourier inversion in computer tomography. IEEE Trans. Med. Imaging **4**(4), 200–207 (1985)
30. Plauger, P., Lee, M., Musser, D., Stepanov, A.A.: C++ Standard Template Library, 1st edn. Prentice Hall PTR, Upper Saddle River (2000)
31. Reynolds, C., Paragi, Z., Garrett, M.: Pipeline Processing of VLBI Data. Physics (2002)

32. Romein, J.W.: An efficient work-distribution strategy for gridding radio-telescope data on GPUs. In: Proceedings of the 26th ACM International Conference on Supercomputing, ICS 2012, pp. 321–330. ACM, New York (2012)
33. Rosenfeld, D.: An optimal and efficient new gridding algorithm using singular value decomposition. Magn. Reson. Med. **40**(1), 14–23 (1998)
34. Sanders, J., Kandrot, E.: CUDA by Example: An Introduction to General-Purpose GPU Programming, 1st, edn. Addison-Wesley Professional, Boston (2010)
35. Sinnott, R.W.: Virtues of the Haversine, vol. 68, p. 158 (1984)
36. Sum, J., Leung, C.S., Cheung, R.C.C., Ho, T.Y.: HEALPIX DCT technique for compressing PCA-based illumination adjustable images. Neural Comput. Appl. **22**(7), 1291–1300 (2013)
37. Tingay, S.J., et al.: The Murchison widefield array: the square kilometre array precursor at low radio frequencies. Publications of the Astronomical Society of Australia, vol. 30, no. 30, pp. 109–121 (2013)
38. Vincenty, T.: Direct and inverse solutions of geodesics on the ellipsoid with application of nested equations. Surv. Rev. **23**(176), 88–93 (1975)
39. Wells, D.C., Greisen, E.W.: Fits: a flexible image transport system, vol. 44, p. 363 (1981)
40. Winkel, B., Lenz, D., Flöer, L.: Cygrid: a fast cython-powered convolution-based gridding module for python. Astron. Astrophys. **591**, A12 (2016)