SDP Memo 085: PIP.IMG Gridding Algorithms

Document number	085
Document Type	MEMO
Revision	1.0
Author	Andrew Ensor
Release Date	2014-07-17
Document Classification	Unrestricted

Lead Author	Designation	Affiliation
Andrew Ensor	Director NZA	NZA - AUT
Signature & Date:	Andrew Ensor Andrew Ensor (Oct 25, 2018)	

SDP Memo Disclaimer

The SDP memos are designed to allow the quick recording of investigations and research done by members of the SDP. They are also designed to raise questions about parts of the SDP design or SDP process. The contents of a memo may be the opinion of the author, not the whole of the SDP.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

Introduction

The purpose of this document is to identify standard gridding algorithms, describe their compute requirements, and opportunities for parallelism.

References

Reference Documents

Reference Number	Reference
1	T. Cornwell and A. Bridle, "Deconvolution Tutorial," 1996. [Online]. Available: http://www.cv.nrao.edu/~abridle/deconvol/deconvol.html.
2	T. Cornwell, K. Golap and S. Bhatnagar, "The non-coplanar baselines effect in radio interferometry: The W-projection algorithm," *IEEE Journal of Selected Topics in Signal Processing,* 2008.
3	N. Mathur, "A pseudodynamic programming technique for the design of correlator supersynthesis arrays," *Radio Science,* vol. 4, 1969.
4	D. Hogg, G. MacDonald, R. Conway and C. Wade, "Synthesis of brightness distribution in radio sources," *The Astronomical Journal,* vol. 74, 1969.
5	W. Brouw, "Aperture Synthesis," *Methods in Computational Physics,* vol. 14, pp. 131-175, 1975.
6	J. O'Sullivan, "A fast sinc function gridding algorithm for Fourier inversion in computer tomography," *IEEE Transactions in Medical Imaging,* Vols. M1-4, pp. 200-207, 1985.
7	D. Slepian and H. Pollak, "Prolate Spheroidal Wave Functions, Fourier Analysis and Uncertainty – I," *Bell System Tech. Journal,* vol. 40, 1960.
8	H. Landau and H. Pollak, "Prolate Spheroidal Wave Functions, Fourier Analysis and Uncertainty – II," *Bell System Tech. Journal,* vol. 40, 1960.
9	J. Jackson, M. Meyer, D. Nishimura and A. Macovski, "Selection of a Convolution Function for Fourier Inversion Using Gridding," *IEEE Transactions on Medical Imaging,* vol. 10, 1991.
10	A. Varbanescu, A. van Amesfoort, T. Cornwell, A. Mattingly, B. Elmegreen, R. van Nieuwpoort, G. van Diepen and H. Sips, "Radioastronomy Image Synthesis on the Cell/B.E.," *Lecture Notes in Computer Science,* vol. 5168, pp. 749-762, 2008.
11	A. van Amesfoort, A. Varbanescu, H. Sips and R. van Nieuwpoort, "Evaluating Multi-Core Platforms for Data-Intensive Kernels," *Proceedings of the ACM International Conference on Computing Frontiers,* pp. 207-216, 2009.
12	R. Edgar, M. Clark, K. Dale, D. Mitchell, S. Ord, W. Wayth, H. Pfister and L. Greenhill, "Enabling a High Throughput Real Time Data Pipeline for a Large Radio Telescope Array with GPUs," *Computer Physics Communications,* vol. 181, no. 10, pp. 1707-1714, 2010.
13	A. Biem, B. Elmegreen, O. Verscheure and D. Turaga, "A Streaming Approach to Radio Astronomy Imaging," *IEEE International Conference on Acoustics Speech and Signal Processing,* pp. 1654-1657, 2010.

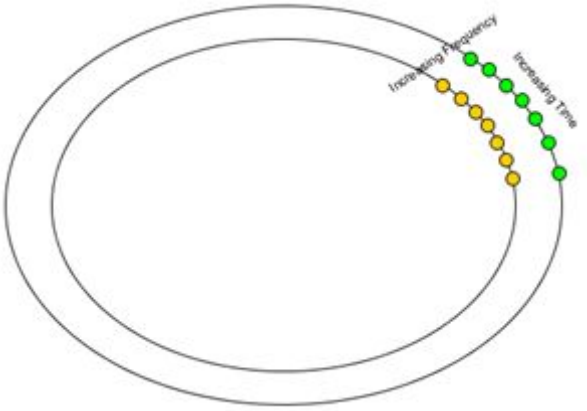
14	B. Humphreys and T. Cornwell, "Analysis of Convolutional Resampling Algorithm Performance," *SKA Memo,* vol. 132, 2011.
15	CSIRO, "tConvolveMPI," [Online]. Available: http://www.atnf.csiro.au/people/Ben.Humphreys/software/tConvolveMPI.tgz.
16	CSIRO, "tConvolveCuda," [Online]. Available: http://www.atnf.csiro.au/people/Ben.Humphreys/software/tConvolveCuda.tgz.
17	J. Romein, "An Efficient Work-Distribution Strategy for Gridding Radio-Telescope Data on GPUs," *Proceedings of the 26th ACM international conference on Supercomputing,* pp. 321-330, 2012.
18	ASTRON, "Romein Gridder," [Online]. Available: ftp://ftp.astron.nl/outgoing/romein/Gridding-0.2.tar.bz.
19	D. Muscat, "High-Performance Image Synthesis for Radio Astronomy," *University of Malta,* 2013.
20	ARXIV, "Muscat Gridder," [Online]. Available: http://arxiv.org/abs/1403.4209.
21	"Casacore," [Online]. Available: https://code.google.com/p/casacore/.
22	"Casacore-1.7.0," [Online]. Available: ftp://ftp.atnf.csiro.au/pub/software/casacore/casacore-1.7.0.tar.bz2.
23	B. Cotton, "Orbit," [Online]. Available: http://www.cv.nrao.edu/~bcotton/Obit.html.
24	B. Cotton, "Orbit SVN," [Online]. Available: https://svn.cv.nrao.edu/svn/ObitInstall/.
25	Z. Hakura and A. Gupta, "The design and analysis of a cache architecture for texture mapping," *Proceedings of the 24th annual international symposium on computer architecture,* pp. 108-120, 1997.
26	H. Schomberg and J. Timmer, "The Gridding Method for Image Reconstruction by Fourier Transform," *IEEE Transactions on Medical Imaging,* vol. 14, no. 3, pp. 596-607, 1995.
27	J. Kaiser, "Digital Filters," in *System Analysis by Digital Computer*, 1966, pp. 218-285.
28	D. Rosenfeld, "An Optimal and Efficient New Gridding Algorithm Using Singular Value Decomposition," *Magnetic Resonance Imaging,* vol. 40, no. 1, pp. 14-23, 1998.
29	H. Moriquchi and J. Duerk, "Modified block uniform resampling (BURS) algorithm using truncated singular value decomposition: fast accurate gridding with noise and artifact reduction," *Magnetic Resonance in Medicine,* vol. 46, no. 6, pp. 1189-1201, 2001.
30	R. Hoge, R. Kwan and B. Pike, "Density Compensation Functions for Spiral MRI," *Magnetic Resonance in Medicine,* vol. 38, pp. 117-128, 1997.
31	V. Rasche, R. Proksa, R. Sinkus, P. Börnert and H. Eggers, "Resampling of Data Between Arbitrary Grids Using Convolution Imterpolation," *IEEE Transactions on Medical Imaging,* vol. 18, no. 5, pp. 385-392, 1999.
32	H. Sederat and D. Nishimura, "On the Optimality of the Gridding Reconstruction Algorithm," *IEEE Transactions on Medical Imaging,* vol. 19, no. 4, pp. 306-317, 2000.

33	P. Beatty and D. Nishimura, "Rapid Gridding Reconstruction With a Minimal Oversampling Ratio," *IEEE Transactions on Medical Imaging,* vol. 24, no. 6, pp. 799-808, 2005.
34	T. Sørensen, T. Schaeffter, K. Noe and M. Hansen, "Accelerating the Nonequispaced Fast Fourier Transform on Commodity Graphics Hardware," *IEEE Transactions on Medical Imaging,* vol. 27, no. 4, pp. 538-547, 2008.

Background

Many acquisition systems ranging from radio astronomy and radar arrays to magnetic resonance imaging obtain non-uniformly sampled Fourier domain data $V(u,v)$ that must be inverse Fourier Transformed to reconstruct an image of the source. For performance an inverse Fast Fourier Transform (FFT) is typically chosen, which requires the data to be placed on a rectangular grid.

In the case of radio astronomy the visibilities $V(u,v)$ are measured by correlating the signals from each baseline pair of receivers. Each receiver provides an X polarization and a Y polarization signal, so there are four correlated results for each baseline, in each frequency. Their positions (u,v) trace out an ellipse over time as the Earth rotates, with longer baselines (receivers further apart) and higher frequencies giving ellipses with larger axes.



The receivers are usually positioned to give good coverage of the UV plane, but clearly the points (u,v) do not fall on a grid and there will be gaps in the uv plane. The relationship between the visibilities $V(u,v,w)$ in three dimensions and the intensity distribution on the sky $I(l,m)$ for direction cosines l and m is given by the van Cittert-Zernike theorem:

$$V(u,v,w) = \iint \frac{I(l,m)}{\sqrt{1-l^2-m^2}} e^{-i2\pi\left(ul+vm+w\left(\sqrt{1-l^2-m^2}-1\right)\right)} dldm$$

In cases when the field of view is small, the spheroid form of the observed sky is close to planar. If the receivers are also coplanar the effect of the w term becomes insignificant and the equation simplifies to:

$$V(u,v) = \iint I(l,m)\, e^{-i2\pi(ul+vm)} dldm$$

showing that the sky image is a 2D inverse Fourier transform of the visibilities. The imaging process obtains the intensities $I(l,m)$ from the visibilities $V(u,v)$ by gridding the data and performing an inverse FFT. The resulting $I(l,m)$ is called the *dirty image* as the visibilities $V(u,v)$ are only sampled at certain (u_s,v_s), so actually are the product of the (measured and unmeasured) visibilities with a delta sampling function $S(u,v) = \Sigma_s \delta(u-u_s, v-v_s)$. Hence the dirty image is the convolution of the true sky image with the *point spread function* (inverse Fourier transform of $S(u,v)$). The sky image is then obtained by an iterative process of

deconvolution, such as the Högbom CLEAN algorithm, or one of its variant due to Clark or Cotton-Schwab (requiring repeated gridding and inverse FFTs). More detail on deconvolution can be found in [1].

For wide-field imaging the w term cannot be ignored. Using a delta function the visibility equation can be written as:

$$V(u,v,w) = \iiint \frac{I(l,m)\delta\left(n-\sqrt{1-l^2-m^2}\right)}{n} e^{-i2\pi(ul+vm+w(n-1))} dldmdn$$

showing that $I(l,m)$ can be obtained by an inverse 3D Fourier transform, requiring 3D gridding if the FFT is then used. Instead, it can be written as:

$$V(u,v,w) = \iint \frac{I(l,m)}{\sqrt{1-l^2-m^2}} G(l,m,w) e^{-i2\pi(ul+vm)} dldm$$

where $G(l,m,w) = e^{-i2\pi w\left(\sqrt{1-l^2-m^2}-1\right)} \simeq e^{i\pi w(l^2+m^2)}$, which is used in the *W-projection algorithm* to correct for the w term by deconvolving the visibilities $V(u,v,0)$ with the inverse Fourier transform $\frac{i}{w} e^{-i\pi(u^2+v^2)/w}$ of G. This is discussed in [2] along with alternatives to W-projection, including W-snapshots and W-stacking.

When the primary beam is not centred on the phase centre the antenna primary beam $A(l,m)$ can be included as a multiplicative factor in $G(l,m,w)$. This results in the W-projection convolution function $G(l,m,w)$ being smoothing and slightly extended by convolution with the inverse Fourier transform of $A(l,m)$, and is called *aw-projection*. In practice, the resulting aw-projection convolution function is determined by numerical approximation.

Gridding

A basic interpolation between sampled $V(u,v)$ data onto a rectangular grid results in undesirable artifacts for the inverse Fourier transform, which can be reduced by more sophisticated interpolation approaches. Early approaches to gridding radio astronomy data simply added each data point $V(u,v)$ to an accumulator at its nearest grid point [3], or averaged those values that fall nearest to each grid point [4]. The technique commonly known as the gridding method was first developed by [5] and utilized a weighted sum based on distance between the sample point (u,v) and the grid point, such as a Gaussian average. In [6] convolution with a sinc function was shown to provide optimal gridding. However since a sinc function has infinite extent convolution kernels C with finite support are instead used.

Often a prolate spheroidal function is chosen in radio astronomy, as they are band limited and useful for recovering time limited functions from their Fourier transform [7], [8].

As the (u,v) are not uniformly distributed using a sum at each grid point for the convolutions of each $V(u,v,w)$ give densely sampled regions in the uv plane more weight, which can be corrected by dividing the $V(u,v,w)$ by the convolution of $S(u,v)$ with the convolution function $C(u,v)$. Also, as the convolution function may attenuate the sides of the resulting image, the image can be divided by the inverse Fourier transform of the convolution function. If the gridder

is denoted by $III(u,v) = \Sigma_{x,y}\delta(u-x)\,\delta(v-y)$ then the gridding algorithm can be written mathematically as:

$$I(l,m) = \frac{F^{-1}\left(\left(\frac{V(u,v)\cdot S(u,v)}{S(u,v)*C(u,v)} * C(u,v)\right)\cdot III(u,v)\right)}{F^{-1}(C(u,v))}$$

This is described in [9] which also compares different convolution functions: cosine, Gaussian, prolate spheroidal functions, and a simpler to compute Kaiser-Bessel function alternative.

A naïve approach to gridding would simply apply a chosen convolution kernel to each visibility in each of its channels. One basic efficiency that can be made is that the same convolution entries can be used for the visibilities in each polarization pair XX, XY, YX, YY:

GENERAL GRIDDING

Initialize $grid_{XX}$, $grid_{XY}$, $grid_{YX}$, $grid_{YY}$ accumulators to 0
for each visibility sample V(u,v,w) **do**
    for each channel f **do**
        x = round(u), y = round(v) calculate nearest position on grid
        C = getConvolutionKernel(u,v,w) in general kernel might not be fixed
        for each kernel column i **do** i goes from -(width-1)/2 to (width-1)/2
            for each kernel row j **do** j goes from -(height-1)/2 to (height-1)/2
                Δi = x+i-u, Δj = y+j-v
                $grid_{XX}(x+i,y+j) \mathrel{+}= V_{XX}(u,v,w) \cdot C(\Delta i, \Delta j)$
                $grid_{XY}(x+i,y+j) \mathrel{+}= V_{XY}(u,v,w) \cdot C(\Delta i, \Delta j)$
                $grid_{YX}(x+i,y+j) \mathrel{+}= V_{YX}(u,v,w) \cdot C(\Delta i, \Delta j)$
                $grid_{YY}(x+i,y+j) \mathrel{+}= V_{YY}(u,v,w) \cdot C(\Delta i, \Delta j)$

Note that unlike usual image processing convolutions the samples at (u,v) are not on the grid points (x,y) so the convolution is utilized at non-integer fractional positions (Δi, Δj), complicating a matrix representation of the convolution. To avoid calculating the kernel at arbitrary fractional values during gridding each of Δi and Δj might be rounded to the nearest multiple of 0.125 (for 8 times oversampling), and 64 versions of the kernel C precalculated before gridding. The version of the kernel chosen for each visibility depends on the number -0.375, -0.25, -0.125, 0, 0.125, 0.25, 0.375, 0.5 to which each of u-x and v-y is closest.

In the case of the W-projection algorithm the convolution kernel varies with w, and 64 oversampling is commonly used:

W-PROJECTION GRIDDING

Initialize $grid_{XX}$, $grid_{XY}$, $grid_{YX}$, $grid_{YY}$ accumulators to 0
for each visibility sample V(u,v,w) **do**
    for each channel f **do**
        x = round(u), y = round(v) calculate nearest position on grid
        a = round(8*(u-x)), b = round(8*(v-y)) classify fractional parts of u and v
        $C_{a,b}$ = getConvolutionKernel(a,b,w) w-projection kernel depends on a,b,w
        for each kernel row i **do** i goes from -(height-1)/2 to (height-1)/2
            for each kernel column j **do** j goes from -(width-1)/2 to (width-1)/2
                $grid_{XX}(x+i,y+j) \mathrel{+}= V_{XX}(u,v,w) \cdot C_{a,b}(i,j)$

$$grid_{XY}(x+i,y+j) \mathrel{+}= V_{XY}(u,v,w) \cdot C_{a,b}(i,j)$$
$$grid_{YX}(x+i,y+j) \mathrel{+}= V_{YX}(u,v,w) \cdot C_{a,b}(i,j)$$
$$grid_{YY}(x+i,y+j) \mathrel{+}= V_{YY}(u,v,w) \cdot C_{a,b}(i,j)$$

The size of the convolution kernel grows with the value of w and with the field of view, and might range from between 7x7 or 9x9 (for w=0) and 70x70 (for the largest value of w) [2] or even 129x129. The convolution kernel is often calculated at around 33-65 regularly spaced values to reduce aliasing effects, resulting in GB memory requirements for representing the full convolution kernel.

A difficulty with parallelizing this approach to gridding is that the grid accumulators will be updated by multiple visibilities, so processing visibilities in parallel requires some synchronized access to the grid accumulators. In the case of the Survey telescope, prior to rebaselining and with a dump time of at most 3s, there are at least 218880 visibilities being produced each second, each with 2^{18} channels (across 500MHz), so a 12 hour observation must grid at least 2.4×10^{15} points. If a separate $2^{14} \times 2^{14}$ grid is used for each 2^{10} consecutive channels (a 2MHz sub-band for continuum imaging) with a 7x7 kernel size then on average each grid accumulator will be updated about 1.7 million times during gridding. Synchronizing access to the grid accumulators could easily become a bottleneck, although the visibilities can be partitioned for parallel gridding on multiple copies of the grid and the results later merged (added) together into a single grid.

Varbanescu et al. Gridder

In [10] the W-projection was implemented using Cell BE processors. Each Cell BE processor consists of a 64-bit PowerPC PPE and eight 128-bit SPE cores that have 256KB local storage each, connected together by a high-bandwidth bus that allowed asynchronous DMA transfers. This work notes the typically irregular access pattern to grid points and to kernel entries when visibilities are processed serially along any baseline. It has the PPE hold a copy of the grid for each SPE so the SPE can all convolve visibilities in parallel and each SPE holds a piece of its grid copy in its local store, performing DMA transfers to and from its local store when it needs a different piece of the grid to work on. The PPE coordinates distribution of each visibility to an SPE to convolve, and reduces the frequent DMA transfer of pieces of the grid by distributing adjacent visibilities to the same SPE so the SPE can perform more accumulations on the same part of the grid and only DMA transferring the portion of the grid back when it receives visibilities requiring a different region. Once completed the PPE can itself merge together the results from each SPE copy of the grid.

Van Amesfoort et al. Gridder

In [11] gridding strategies for quad core Xeon, the Cell BE and GPU are compared. On the Xeon pthreads write to their own private grids to avoid synchronized access, and a coordinating thread increases data locality by placing jobs in a worker thread's queue that have semi-overlapping grid and convolution entries, and SIMD intrinsics are used. The Cell BE

approach is similar to the Varbanescu et al. gridder with the PPE scheduling the SPE work, and utilizing vector operations, loop unrolling and DMA multibuffering to improve performance. A CUDA implementation is used in the GPU approach where write conflicts are avoided by having each block of threads write to its own private grid, with the block of threads working on a row at a time (but no discussion of accommodating texture tile sizes).

Edgar et al. Gridder

For the MWA [12] a GPU gridder was implemented in CUDA using a (24x24) convolution function intended to be different for each visibility. Rather than iterating through the visibilities (with each grid point accumulating convolutions across around 60 visibilities) this implementation uses a gather approach where a different thread is responsible for each grid point, avoiding the need for synchronized access to the grid accumulators. Each thread searches the 130000 visibilities for those visibilities that are within the convolution size of its grid point. To reduce the search space for the threads, the visibilities are partitioned into 24x24 sized grid regions (the same size as the kernel) so that a thread only need check the visibilities within its own grid region and the eight neighbouring regions (a search through about 540 visibilities for the approximately 60 that affect the grid point). Furthermore, visibilities within a grid region could be loaded into shared memory for common access by threads within the same grid block.

The paper also discussed a potential alternative GPU approach using the OpenGL rendering pipeline and exploiting GPU hardware acceleration for rasterization and FBO blending, but despite the approach working well, the CUDA approach was developed with the intention of more broad maintainability.

Biem et al. Gridder

The paper [13] uses a IBM middleware called Infosphere Streams (previously termed System S) to help automate the distribution of gridding streaming visibility data across multiple compute nodes. It distributes the gridding task by partitioning blocks of channels to be processed in parallel on separate local copies of the grid, and for each visibility computes an index that points to the appropriate convolution entry and an index that points to the grid coordinates before convolving the visibility on the local grid, and later aggregates the results together on a final grid. For testing on dual-core Xeon CPU it uses 10^4 simulated ASKAP visibilities for W-projection with a 45x45 convolution matrix, 8 times oversampling, and 65 w-planes, with up to 2^{14} channels.

Humphreys/Cornwell Gridder

The W-projection gridding requirements for ASKAP are discussed in [14] with performance tests given on a CPU implementation in C++ with MPI communication and on a GPU implementation in CUDA. The implementations exploit that ASKAP receivers have a three-axis

design that allow the PAF and feed legs to stay rotationally fixed relative to the sky (fixed parallactic angle and no change in aperture blockage due to the feed legs). This means the convolution functions do not change during the course of the observation so can be precalculated beforehand. The testing used a 129x129 convolution matrix, 8 times oversampling, 33 w-planes, a single spectral channel and a 4096x4096 grid (although tests were also conducted with 4 times oversampling, up to 129 w-planes, and grid sizes up to 12288x12288 to verify those parameters impact the implementation's memory requirements rather than performance). The results on Intel Nehalem and Nvidia C2070 conclude that for smaller 33x33 convolution matrices memory latency was significant whereas for larger 257x257 convolution matrices memory bandwidth was significant. The paper also compares gridding power requirements for Intel Nehalem, Nvidia C2070 and Blue Gene/P PowerPC systems.

The CPU C++ implementation tConvolveMPI is available from [15] with the gridding performed in the Benchmark::gridKernel function and an approximate convolution kernel calculated in Benchmark::initC. For testing the visibilities are assigned random (u,v,w) (so possibly not allowing the CPU to exploit caching) and the grid point (samples[i].iu and samples[i].iv) and convolution offset (samples[i].cOffset) are precalculated in Benchmark::initCOffset. BLAS routines are used when available for the convolution otherwise the code simply iterates through convolution kernel multiplying one entry at a time in a local grid. MPI is only used for barriers in timing, there is actually no interprocess messaging during testing and no final grid merging.

The GPU implementation tConvolveCuda is available from [16] and compares an elementary C++ gridder in tConvolveCuda::gridKernel with a CUDA gridder CudaGridKernel that also has the grid points (iu[i] and iv[i]) and convolution offsets (cOffset) precalculated. The function cuda_gridKernel uses a single grid and iterates through the visibility data, collecting together up to 32 consecutive visibilities whose convolution kernels do not overlap grid points, and creates a thread for each visibility and each convolution point so that the collection of visibilities can be convolved at each grid point in parallel by d_gridKernel which essentially performs one complex multiply-accumulate at a single grid point (with a 2D thread block giving the convolution rows and collection size, and one thread per convolution column) .

Romein Gridder

A different GPU implementation of W-projection is described in [17]. Instead of allocating one OpenCL or CUDA thread per grid point it considers a single grid to be conceptually partitioned into subgrids each the same size nxn as the convolution kernel, and allocates n^2 threads in total, with each thread responsible for exactly one grid point within each subregion (at the same relative position within every subregion). In this way regardless of where in the single grid a convolution is applied it each thread will be responsible for exactly one of the grid points. If consecutive visibilities being gridded are closely spaced on the grid (as they would usually be along a single baseline and across nearby frequency channels) then many of the threads will be accumulating at their same subgrid point over multiple visibilities. This enables an OpenCL or CUDA kernel to just keep a simple accumulator for each of the XX, XY, YX, YY polarization pairs, and only needs to add to the common grid whenever the convolution kernel has moved so that the thread must now accumulate for a grid point in a different subregion. In addition for

greater parallelization a separate thread block is created for each baseline and any updates to the common grid are synchronized.

The GPU implementation in OpenCL is available from [18] and has C++ host Gridding.cc program and OpenCL kernel Kernel.cl, which keeps track whether the new_grid_point for the next visibility it will grid is different from the previous grid_point used for the previous visibility, and uses OpenCL extensions for atomic operations to update the grid.

Muscat Gridder

The Masters thesis [19] available from [20] modifies the Romein Gridder with a pre-gridding preparation phase. This is done to improve performance by removing flagged visibilities and points that lay off the grid, and performing a form of “compression” where visibilities along a baseline are summed that are found to have the same convolution and grid position, providing statistics on gridding rates and the level of compression found. It uses two different kernels for convolutions, one optimized for convolutions smaller than 31x31 (with 256 threads per thread block) and the other for larger convolutions (with 1024 threads per thread block to maximize shared memory usage).

Casa

Casacore is a collection of C++ libraries derived from aips++ upon which the Common Astronomy Software Applications (CASA) is built. Casacore is described at [21] and its code is available from [22].

Its gridding is performed by dividing the gridding plane into small patches, maintaining a cache for one patch in memory and gridding time-sorted visibilities on that patch until another needs to be swapped into memory (essentially performing memory paging on the grid). It supports BOX (Box-car), SF (Prolate Spheroidal Function), GAUSS (Gaussian), and GJINC (Gaussian Jinc using a first order Bessel function) convolution kernels, with support for PB (Primary Beam) planned. The actual gridding is performed by a Fortran gridder (wprojgrid.f and faccumulateToGrid.f, both located in the code/synthesis/fortran folder) and called by GridFT.cc (located in the code/synthesis/TransformMachines folder).

Orbit

Orbit is a collection of C/C++ software packages for processing and handling radio astronomy data, largely developed by Bill Cotton. It is described at [23] and its code is available from [24]. The OrbitUVGrid.c function OrbitUVGridReadUV creates a pool of threads, one per processor, that each grid a subset of visibilities on their own grid, and then the grids are accumulated together (and FFTs performed using the FFTW library).

ASKAPsoft

ASKAPsoft is a C++ package developed since 2007 by CSIRO for the eventual 36 dish ASKAP precursor to the SKA1 Survey array. It is available from CSIRO upon request with a GNU General Public License. The gridder.rst document provides the following description of the gridders available in ASKAPsoft:

Gridder	Accounts for w-term	Mosaicing	Convolution functio	Notes
Box	No	No	Box function	Very crude nearest neighbour gridder
SphFunc	No	No	Prolate spheroidal function	The simplest usefu gridder (just optima aliasing rejection)
WProject	Yes (projection)	No	Spheroidal function + FT of the w-term	Uses W-projection to account for the w term in the Fourier transform
WStack	Yes (stacking)	No	Spheroidal function	Uses stacking in image space to account for the w term in the Fourier transform
AWProject	Yes (projection)	Yes	Spheroidal function + FT of the w-term + autocorrelation of antenna illumination	In addition to WProject, takes int account primary beam
AProjectWStack	Yes (stacking)	Yes	Autocorrelation of antenna illumination	In addition to WStack, takes into account primary beam

The main code for gridding is in the Components/Synthesis/synthesis/current/gridding folder. The gridders BoxVisGridder, SphFuncVisGridder and its subclasses WProjectVisGridder (with subclass AWProjectVisGridder) and WStackVisGridder (with subclass AProjectWStackVisGridder) within the namespace askap.synthesis extend the class TableVisGridder which in turn extends the abstract base class IVisGridder. The class SphFuncVisGridder includes a comment by Tim Cornwell that it uses the same prolate

spheroidal function as used in AIPS and AIPS++, and that that choice should be revisited in future.

GPU Rasterizers

Modern GPU processors have very high performance hardware accelerated *rasterizers* which grid geometry onto a rectangular grid as fragments. They typically use a form of supersampling (such as multisample anti-aliasing) to reduce the aliasing effect of rasterizing (essentially the same as the oversampling approach in W-projection). This can be used with point sizes to have a point rasterized over a defined radius of the grid region, including with texture coordinates or point sprite alpha values to change the weighting of the point depending on distance from its centre, and a vertex shader can be used to manipulate its position and attributes.

Texture sampling is also highly optimized, with arbitrary software interpolations possible but having built-in hardware support for linear (and bilinear, trilinear) interpolations between texels, which as noted in [17] may improve the sampling of the convolution kernel used in gridding.

Another hardware acceleration feature of GPUs is their frame buffer operations, which are specifically designed for highly parallel and concurrent updates at frame buffer entries. Operations such as the OpenGL glBlendFunc with GL_FUNC_ADD allow a bound frame buffer colour or depth texture to be used for accumulating rasterized values that can optionally be manipulated in a fragment shader.

More recently, 1D and 2D convolutions also have direct hardware support.

This hardware acceleration is graphics-specific so typically available on GPU via DirectX or OpenGL rather than directly via Cuda or OpenCL. It appears that apart from a brief mention in [12] and the use of texture interpolation for convolution kernels in [17] many of the hardware features of GPU have not yet been exploited in radio astronomy gridders. Furthermore, some existing GPU gridding implementations appear to not consider the Z-ordering (Morton code) and/or tiling design of textures which can significantly affect performance [25].

Medical Imaging Gridders

Medical imaging techniques such as magnetic resonance imaging and X-ray computed tomography obtain non-uniformly sampled Fourier domain (called k-space) data, from which an image must be constructed. A little differently to the elliptical visibility arcs in radio astronomy, medical imaging often obtains sampled data along radial or spiral arms. Early work in medical imaging adopted the gridding approach pioneered in radio astronomy but since the late 1990s the medical imaging community has undertaken a lot of research themselves in improved techniques. Some of the more heavily cited papers include:

- [26] builds on radio astronomical approach to gridding described by [5] but states the supposed optimality of using a prolate spheroidal function for minimizing aliasing error is

not entirely relevant and for gridding in computed tomography and MRI instead recommends using the Kaiser-Bessel window as described in [27].

- [28] also refers to using the Kaiser-Bessel window for MRI gridding. It presents a new gridding algorithm called Block Uniform Resampling (BURS) which decomposes the gridding task into solving a system of linear equations at each grid point via singular value decomposition, and claimed to be computationally more efficient than conventional gridding with high quality results. A modified BURS algorithm is presented in [29] that is less sensitive to noise and reduces computational requirements.
- [30] discusses a density compensation function that reduces artifacts resulting from unevenly spaced sampling trajectories. [31] describes an approach for obtaining the sampling density function for an arbitrary sampling pattern by determining the Voronoi diagram via a Delaunay triangulation, and using the area of the Voronoi polygon around each sample point as the measure of its density compensation function. [32] shows that gridding is an approximation to least squares approximation and provides a method for calculating density compensation factors to minimize approximation error.
- [33] provides informative results for reducing the memory requirements of gridding by using a minimal oversampling ratio (1.125-1.375 instead of oversampling the grid points by a factor of 2), and evaluates the effect of presampling the Kaiser-Bessel convolution kernel, concluding that linearly interpolating between presampled kernel values (as mentioned earlier for which GPU have hardware acceleration) adds little error and requires far less presampling than using nearest-neighbour kernel values (currently common practice in W-projection). The paper also provides an equation for calculating a near optimal Kaiser-Bessel kernel that has near-lowest maximum aliasing amplitude rather than the usual lowest average aliasing energy across the entire image which is claimed deposits much of the aliasing energy around the edges of the image.
- [34] discuses gridding and the inverse FFT (termed non-equispaced FFT, NFFT) and provides a GPU implementation in CTM (a predecessor to Cuda and OpenCL), and discuses several implementation strategies, including a naïve parallelization strategy where each processor is responsible for gridding a subset of visibilities, having each processor responsible for a single grid point (as later adopted by the Edger et al Gridder), and a hybrid approach where a processor is responsible for a neighbourhood of grid points and the samples are organized depending on the neighbourhoods.

SKA1 Gridding Parameters

Following rebaselining and the cost control project as of 2018 the Mid array consists of 133 SKA1 + 64 MeerKAT = 197 dishes, so up to 19,306 baselines. Each produces 65,536 ± 20% channels with four polarisation pairs XX, XY, YX, YY, and has a minimum dump time of 0.14s. This can result in up to 36.15×10^9 visibilities per second for the Mid array (or 36.5×10^9 including autocorrelations). The use of up to 16 subarrays would lower the total number of baselines so give fewer visibilities but would increase the number of grids. At present each visibility is represented by 80 bits (excluding SPEAD and UDP packet overheads).

Following rebaselining and the cost control project as of 2018 the Low array consists of 512 stations, so up to 130,816 baselines. Also with 65,536 ± 20% channels and four polarisation

pairs, but with a longer minimum dump time of 0.9s the Low array can produce up to 38.1×10^9 visibilities per second (or 38.25×10^9 including autocorrelations). Using subarrays brings down the data rate, and although using substations increases the number of baselines it is required to not increase the visibility dump rate to the SDP.