Rhodes University

Submitted in partial fulfilment
of the requirements of the degree of

Bachelor of Science (Honours)

Creating and Optimizing a Sky Tessellation Algorithm for the SKA: Literature Review

Antonio Bradley Peters

supervised by
Prof. Karen Bradshaw
Prof. Denis Pollney

project originated by
Prof. Oleg Smirnov

May 16, 2016

1 Introduction

2 Radio Astronomy

2.1 Radio Telescopes

2.1.1 Radio Telescope Design

The most common design for radio telescopes is that of the parabolic reflector antenna. The design is a large parabolic dish with a sub-reflector at the parabola's focal point channeling the input into the feed horn at the center of the dish, a diagram of this can be seen in Figure 1. While it is possible to have a single antenna as a telescope for radio astronomy, in order for them to produce meaningful results, the antenna need to be extremely large (diameter of +70m) which in most cases can be structurally infeasible especially if the antenna is made to be steerable. Instead, a series of smaller ($8 \sim 30$m) antenna are used collectively in an array to produce a more accurate signal detection. These arrays do so through radio interferometry (Cheng, 2009).

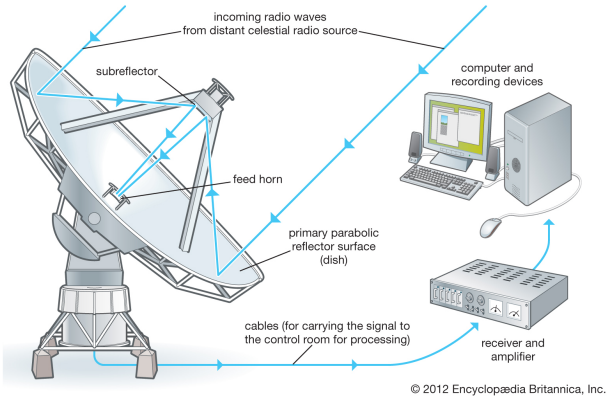

Figure 1: Parabolic Reflector Antenna Design (Britannica Online for Kids, n.d.)

2.1.2 Radio Interferometry

Radio Interferometry uses an array of antenna to detect and measure objects emitting radiation in the radio-wave frequencies. Radio waves are defined as electromagnetic radiation with wavelengths of the order of 10^{-3} to 10^{5} meters (Cheng, 2009). The antenna finds the source of these waves by detecting minor delays in the parallel ray signal transmitted by the radiating source (source) as well as the amplitude and frequency of the source to calculate the position, distance, size and intensity of the source(Thompson *et al.*, 2008).

2.1.3 The SKA

The SKA project was launched in order to create the world's largest array of radio telescopes. This will be achieved by having 197 radio telescopes situated in South Africa and Australia working together and having antenna thats' area will cover close to one square kilometer. The array is set to have a resolution of over 50 times that of the Hubble Space Telescope while still covering more massive areas of the sky (SKA (n.d.)).

2.2 Image Capturing and Processing

2.2.1 The Primary Beam

The primary beam is a mathematical function that describes the sensitivity pattern of an antenna. Naturally the beam is most sensitive in the center of the direction in which it is facing, with fringes of sensitivity radiating out as can be seen in Figure 2. The circular sensitivity present in Figure 2 is also due to the

fact that the telescope rotates in order to keep the center of the beam focused on the same area of the sky (Smirnov, 2016).

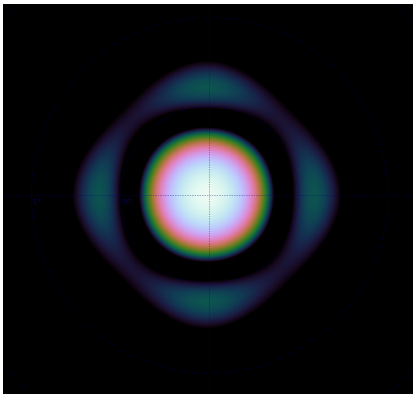

Figure 2: Primary Beam Focus Pattern (Smirnov, 2016)

2.2.2 Aperture Synthesis

The electromagnetic radiation collected by the antenna is converted into voltage differences. The data is collected and stored over time (usually $\sim$ 12h) and the resulting phase differences in the data taken in by each antenna in the telescope is Fourier transformed from the frequency domain to that of the spatial domain, to give a two dimensional image (Sault & Wieringa, 1994).

2.2.3 Errors and Error Correcting

As with any real-world data input, the image capturing process of radio interferometry is subject to errors. These errors can be classified as arising from two main groups, namely Direction-independent (DI) and direction-dependent (DD) effects (Smirnov & Tasse, 2015). The DD-effects in particular arise from distortions due to interference from the ionosphere and deviations of the primary beam from the model. This distortion, D, can be corrected, but only relative to a chosen point, ξ. The correction at any given point, E_i, is dependent on the intensity at the point, $\boldsymbol{S}_i$, and the distortion at the point relative to ξ, $D(\boldsymbol{x}_i, \xi)$. It is important to note that the point-wise error is lowest at ξ, but any source at ξ will radiate out with D over the domain. It can therefore be seen that to minimize this error, every point can be made a correction seed and the image can be broken up by these points and reassembled to form an image with little to no error. However, this is computationally ineffective as there are hundreds of sources per image and also due to the fact that the image is sparely populated. We therefore seek a method which optimally compromises computational feasibility and error reduction (Smirnov, 2016).

2.3 Related Work: Naive Method

The most basic compromise is dividing the image evenly into a grid of smaller images and correcting for these from either the center of the sub-image, the point with the strongest source or the "center of mass" (average location of points) of all the points in the sub-image, either weighted by intensity or not. The problem with this method lies in the fact that either the sub-image is void and has no definite points, if ξ is set at the center, it could be far from every other point and has no substantial effect on reducing the overall error or if ξ is set at the strongest source or the center of mass, it could lie too close to the boundary of the sub-image and, again, have no overall impact on error reduction Smirnov (2016).

3 Models and Algorithms

3.1 Voronoi Tessellations

A Voronoi Diagram is a partitioning of a space S by a set of points. Given n points (seed points) the the space, $P = \{p_0, p_2, ..., p_{n-1}\}, P \subset S$, is partitioned into n regions, known as Voronoi Regions or Voronoi Cell, where every point, $s \in S_i, 0 \leq i \leq n-1$ in a region, $S_i \subset S$, is closest to a single seed point, $p_i \in P$, in terms of the space's distance measurement operation, d (Okabe & Chiu (2009)). An example of a Voronoi Diagram can be seen in Figure 3.

3.1.1 Weighted Voronoi Tessellations

3.1.2 Voronoi Tessellations in Non-Euclidean Spaces

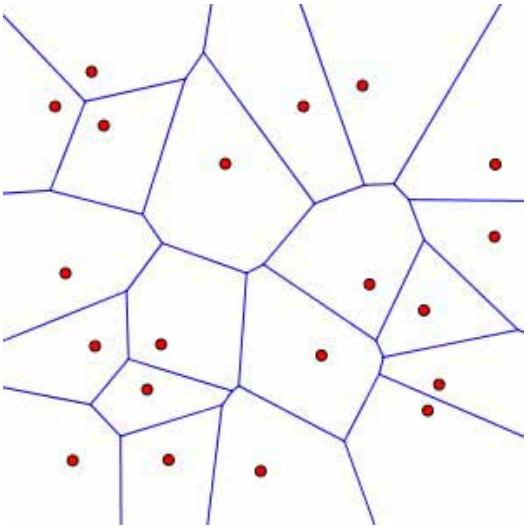

Figure 3: Voronoi Diagram(Austin (n.d.))

3.2 Voronoi Tessellation Generation Algorithms

Although Voronoi Tessellations extend to multiple dimensions, for the sake of these algorithms we will only discuss those in a two dimensional plane.

3.2.1 Incremental Algorithm

The most simplistic of the generation algorithms, the Incremental is an iterative algorithm as described by Green & Sibson (1978) and Okabe & Chiu (2009).

1. Starting from $i = 0$ and an empty plane
2. A seed point, p_i is placed into the plane.
3. The nearest neighboring seed point $p_f = p_{nn}$ is found
4. A perpendicular bisector is drawn between p_i and p_f (if it exists).
5. The bisecting line is followed in both directions until it intercepts an existing edge or the plane's boundary on both ends.

6. A new edge is defined by this segment of the bisector as part of both p_i and p_f.
7. The seed point of the polygon that shares the found edge clockwise to p_f (anticlockwise to p_i) is then set to p_f.
8. Continue from step 4 until $p_f = p_{nn}$ again.
9. Set $i = i + 1$ and repeat from step 2 until $i = n$

In it's most naive form, this algorithm achieves and efficiency of $O(n^2)$.

3.2.2 Divide and Conquer Algorithm

The Divide and Conquer algorithm was first proposed by Shamos & Hoey (1975) and also described in Okabe & Chiu (2009). It is a recursive algorithm which improves on the Incremental algorithm by having a construction time of $O(n \log n)$.

1. If the contains only one point, return it with the entire plane as it's voronoi region.
2. Divide the space, S containing the set of seed points, P with n points, into two subspaces, S_L and S_R, such that S_L and S_R contain $n/2$ seed points and every seed point of P_L lies to the left of every seed point of P_R (this is made easier if P is ordered).
3. Recursively compute the voronoi tessellations for P_L in S_L and P_R in S_R; V_L and V_R respectively.
4. A polygonal line, Q, must now be found such that Q merges V_L and V_R into a single voronoi tessellation, V:
   (a) Starting with the polygon of V_R which contains the top-left corner of S_R, p_R and the polygon of V_L which contains the top-right corner of S_L, p_L. Since p_L must lie to the left of p_R, they must overlap when V_L and V_R are extended into S.
   (b) A perpendicular bisector is drawn between p_L and p_R and segmented between its two closest edge intercepts from the shortest distance between p_L and p_R and add this segment to Q.
   (c) If the lower intercepting edge of the is in V_R then p_R is set to the seed point polygon which shares this edge and similarly if the edge is in V_L.
   (d) Continue from step 4b until the bottom of S is reached.
5. Remove all line segments of V_L to the right of Q and all those of V_R to the left of Q to form V.
6. Return V recursively until the full voronoi tessellation is complete.

Part of achieving this efficiency is assuming P is co-lexicographically ordered, meaning for all $p_i, p_j \in P$, $0 \leq i < j < n$; $x_i > x_j$ or ($x_i = x_j$ and $y_i > y_j$). This speeds up the partitioning of P into P_R and P_L at each level of recursion.

3.2.3 Fortune's Algorithm (Sweep-Line Method)

Fortune (1987) describes an algorithm where the tessellations are found by a line "sweeping" over the space and solving the problem at each step of the sweep. This can be problematic for Voronoi tessellations as the line may intercept the Voronoi Region of a seed point before it intercepts the point. Therefore the Voronoi Tessellation is not computed directly, but through a geometric transform. The transform $\phi(x(s), y(s))$ works such that for any point, $s \in S$ with coordinates $(x(s), y(s))$,

$$\phi(x(s), y(s)) = (x(s) + r(s), y(s)) \tag{1}$$

Where $r_i(s)$ is defined as the distance to the seed point $p_i \in P$ and $r(s) = min\{r_i(s)|1 \leq i \leq n-1\}$, is the distance to the closest seed point to s. This transform can then easily be reversed to re-obtain S and its set of Voronoi tessellations. Now, for the transform of S, $\phi(S)$ denoted by Φ, the left-most point of each Voronoi Region is its seed point (except the left-most seed point), this is essential for the algorithm. It is important to note that the perpendicular bisectors of seed points in S, through the transform, become hyperbolas in Φ (provided they are not horizontal in S). For $p_i, p_j \in P$, the hyperbola is denoted as h_{ij} which can be split into h_{ij}^+ and h_{ij}^- as the upper and lower half-hyperbolas about the left-most point respectively. Set Q is denoted as the set of all event points in the algorithm. Q is initially populated with the seed points (in co-lexicographical order) but the edge interceptions will be added as they are found. The algorithm, as described by Okabe & Chiu (2009) goes as follows:

1. Add P to Q.
2. Choose and delete the leftmost seed point, p_i from Q
3. Create a list, L containing the transformed voronoi region of p_i, $\phi(V_i)$.
4. While Q is not empty do the following:
   (a) Choose and delete the leftmost element, w of Q.
   (b) If w is a seed point:
      i. Set $p_i = w$.
      ii. Find the region, $\phi(V_j)$, containing p_i.
      iii. Replace $\phi(V_j)$ in L with $(\phi(V_j), h_{ij}^-, \phi(V_i), h_{ij}^+, \phi(V_j))$
      iv. The half-hyperbola intercept(s) with any other hyperbolas are found, if they exist, and are appended to the front of Q.
      v. Repeat from 4.
   (c) If w is a half-edge:
      i. Set $\phi(q_t) = w$ where $\phi(q_t)$ is the intercept of $h_{ij}^\pm$ and $h_{jk}^\pm$.
      ii. Replace all sequences of the form $(h_{ij}^\pm, \phi V_j, h_{jk}^\pm)$ on L with $h = h_{ik}^+$ or $h = h_{ik}^-$ appropriately.
      iii. Remove from Q any intersections of $h_{ij}^\pm$ and $h_{jk}^\pm$ with other half-hyperbolas.
      iv. Move any intersections of h in L to Q
      v. Mark $\phi(q_t)$ as a Voronoi vertex incident to $h_{ij}^\pm$, $h_{jk}^\pm$ and h.
      vi. Repeat from 4.
5. Return the half-hyperbolas on L, the set of marked intersections from step 4(c)v and the relations among them.

3.3 Clustering Algorithms

3.3.1 K-Means Algorithm

3.4 Related Work

3.4.1 Standard Voronoi Faceting

In Tasse (2014), Smirnov & Tasse (2015) and van Weeren *et al.* (2016), a series of simulated extragalactic points are clustered into facets using a voronoi tessellation algorithm, the seed points for these facets are set as the brightest points in each facet. Another example of this can be seen in Figure 4 where the facets are superimposed over the source image from which they are derived.

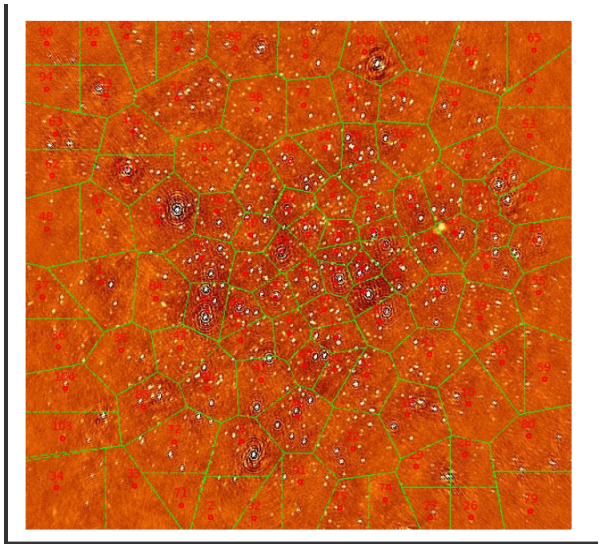

Figure 4: Example of Voronoi Faceting to group Extragalactic points for DD-calibration (Smirnov, 2016)

4 GPU Architecture and Concepts

4.1 Parallelism

One of the main means of optimizing processing is through parallelism. The two main forms of parallelism are task and data parallelism. Task parallelism can be seen as running multiple processes concurrently where communication between the processes is explicitly defined to avoid race conditions. Data parallelism is the distribution of a data set over a number of identical processes each of which performs operations on a unique subset of the data. Race conditions occur when parallel processing streams access data or perform operations out of the intended order, leading to errors or incorrect output being produced. A combination of task and data parallelism can lead to an ideal speed-up, but both have their limits depending on the task and the data being operated on (Subhlok & Gross (1993)).

The increased need for parallelism came about in 2005, when CPU frequency peaked at 4 GHz due to heat dissipation issues. However, Moore's Law still holds, and is still expected to hold until 2025; that is, that the number of transistors for a computer will double every two years. This leads to a problem where the speed at which an operation is done cannot be increased (due to the frequency limit), but the number of concurrent operations can still increase. This means that the only way to speed up an operation is to change it from a sequential to a parallel process (Rajan (2013)).

4.2 GPU Optimization

Graphical Processing Units (GPUs) were originally designed for rendering pixels and vectors in games. They were especially designed for this since CPU's are optimized to run sequential instructions as fast as possible; whereas pixel and vector calculations are inherently parallel. With NVIDIA's release of CUDA in 2006, general purpose GPU (GPGPU) programming became common place as a way to accelerate data processing through data parallelism and task parallelism through the simultaneous execution of similar tasks (NVIDIA (n.d.a)).

The power of GPUs comes from its architecture which is optimized for a special case of SIMD (Single Instruction Multiple Data) processing known as SIMT (Single Instruction Multiple Threads). SIMD allows a central processor to distribute a set of instructions to multiple simple processors which then act on the data simultaneously. SIMT is more generalized as each warp of the GPU can perform different tasks given the same set of instructions. This is due to the way in which the GPU handles branching at the thread level. By exploiting these processes, and this instructional architecture, some instructions can be computed faster than a CPU (Vuduc & Choi (2013)).

4.3 The NVIDIA GeForce GTX 750 Ti

4.3.1 GM107 Maxwell Architecture

The NVIDIA GeForce GTX 750 Ti GPU was released on the 18th of February 2014. It boasts 640 CUDA cores, 1020 MHz base clock speed, 1305.6 GFLOPs and a memory bandwidth of 86.4 GB/sec. It is NVIDIA's first-generation Maxwell architecture, designed for high performance at relatively low power consumption (60 W) and has the codename 'GM107'. The GPU uses PCI Express 3.0 to interface with the host machine through the GigaThread engine. The first-generation Maxwell (from now simply referred to as Maxwell) is made up of one Graphics Processing Cluster (GPC) on which the processing occurs. It also contains a large L2 cache at 2048 KB and two 64-bit memory controllers to access the 2048 MB global memory. This design can be seen in Figure 5 (NVIDIA (n.d.c), NVIDIA (n.d.d), NVIDIA (n.d.e)).

4.3.2 Streaming Multiprocessors

The GPC is further broken down into five streaming multiprocessors(SMs) which are further divided into four processing blocks. The processing blocks each contain an instruction buffer, a scheduler and 32 CUDA cores as seen in Figure 6 (Nathan Kirsch (n.d.)). These warps are set in a lock step, meaning each core in a warp executes the same set of commands at the same time, with different valued variable.(NVIDIA (n.d.b)

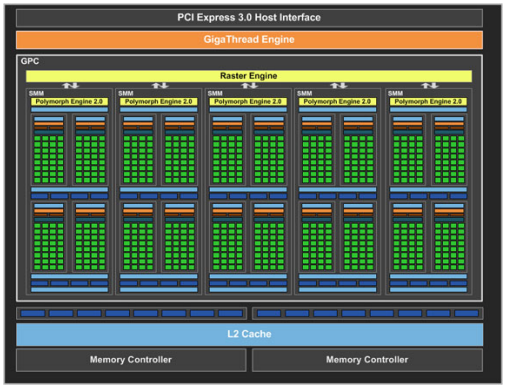

Figure 5: NVIDIA Maxwell Architecture(George Cella (n.d.))

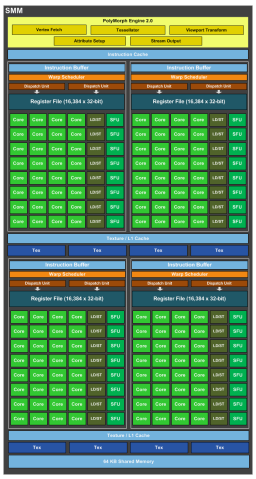

Figure 6: Maxwell Streaming Multiprocessor(Nathan Kirsch (n.d.))

4.3.3 GPU Memory Hierarchy

4.4 CUDA

CUDA is a parallel programming language created by NVIDIA for the purpose of running on their brand of GPU's. CUDA was modeled as a C-like language with some C++ features. Its main feature is the way in which it separates CPU and GPU code. The CPU code is labeled as "host" code and the GPU's as "device" code. Device code is called by the host through a special case of a method, known as a kernel. The basic structure of a kernel is as follows:

```
kernel0<<<grid, block>>>(params);
```

In this instance `kernel0` would be the name of the kernel being called, `grid` is the three dimensional value of the number of blocks to be assigned, `block` being similar to `grid` is a three dimensional value of the number of threads needed and `params` is simply the parameters needed by the kernel to execute (similar to those of a method) (NVIDIA (n.d.b)).

4.4.1 Threads

The thread is the smallest processing unit of the GPU. GPU threads are designed to be cheap and lightweight compared to those of a CPU so that it can be easily created, run it's small task and be destroyed to make place for the next thread. Threads are arranged into three dimensional blocks with each thread having a unique 3 dimensional ID within that block, namely an x, y and z ID. Generally the thread ID is used as the means of determining the difference in the task process of each thread (NVIDIA (n.d.b)).

4.4.2 Blocks

Each block may have a maximum of 2056 threads in total and 1024 for any single dimension, hence why they are bundled into a larger, three dimensional grid structure. Similarly to threads, blocks have a unique three dimensional ID in the grid. Blocks exist such that each step of the processes execute simultaneously. This is done as, more often than not, blocks exchange data within their threads and if this precaution is not taken, race conditions could ensue to break the code. Each block, when executing, must occupy a whole number of warps (rounded up). This is done as warps are constantly in lock step. Threads within a block share a fast memory, located in the L1 cache of the streaming multiprocessor. This shared memory must be preallocated when the kernel is called as a third parameter within the kernel launch (parameters within the triple angle brackets) (NVIDIA (n.d.b)).

4.4.3 CUDA Optimization

4.5 Related Works

5 Summary

References

Austin, David. *Furthest Point Voronoi Diagrams: Voronoi Diagrams*. http://www.ams.org/samplings/feature-column/fcarc-voronoi. Accessed on: 26/02/2016.

Britannica Online for Kids. *Telescope: Radio Telescope*. http://kids.britannica.com/comptons/art-145514. Accessed on: 16/05/2016.

Cheng, Jingquan. 2009. Radio Telescope Design. *The Principles of Astronomical Telescope Design*.

Fortune, Steven. 1987. A sweepline algorithm for Voronoi diagrams. *Algorithmica*, **2**(1-4), 153–174.

George Cella. *MSI GTX 750 Ti Gaming Video Card Review*. http://www.hitechlegion.com/reviews/graphics/38752-msi-gtx-750-ti-gaming-video-card-review?start=2. Accessed on: 27/04/2016.

Green, Peter J, & Sibson, Robin. 1978. Computing Dirichlet tessellations in the plane. *The Computer Journal*, **21**(2), 168–173.

Nathan Kirsch. *NVIDIA GeForce GTX 750 Ti 2GB Video Card Review*. http://www.legitreviews.com/nvidia-geforce-gtx-750-ti-2gb-video-card-review_135752/2. Accessed on: 27/04/2016.

NVIDIA. *CUDA*. http://www.nvidia.com/object/cuda_home_new.html. Accessed on: 22/02/2016.

NVIDIA. *CUDA Toolkit Documentation*. http://docs.nvidia.com/cuda/cuda-c-programming-guide/#axzz47aOWTI3j. Accessed on: 03/05/2016.

NVIDIA. *GeForce GTX 750 Ti*. http://www.geforce.com/hardware/desktop-gpus/geforce-gtx-750-ti. Accessed on: 26/04/2016.

NVIDIA. *GeForce GTX 750 Ti Specifications*. http://www.geforce.com/hardware/desktop-gpus/geforce-gtx-750-ti/specifications. Accessed on: 26/04/2016.

NVIDIA. *GeForce GTX 750 Ti Whitepaper*. http://international.download.nvidia.com/geforce-com/international/pdfs/GeForce-GTX-750-Ti-Whitepaper.pdf. Accessed on: 26/04/2016.

Okabe, Boots, Sugihara, & Chiu. 2009. *Spatial tessellations: concepts and applications of Voronoi diagrams*. Vol. 501. John Wiley & Sons.

Rajan, Krishna. 2013. *Informatics for materials science and engineering: data-driven discovery for accelerated experimentation and application*. Butterworth-Heinemann.

Sault, RJ, & Wieringa, MH. 1994. Multi-frequency synthesis techniques in radio interferometric imaging. *Astronomy and Astrophysics Supplement Series*, **108**, 585–594.

Shamos, Michael Ian, & Hoey, Dan. 1975. Closest-point problems. *Pages 151–162 of: Foundations of Computer Science, 1975., 16th Annual Symposium on*. IEEE.

SKA. *The SKA Project*. http://www.ska.ac.za/about/project.php. Accessed on: 21/02/2016.

Smirnov, Oleg. 2016. Personal Communication. SKA Chair at Rhodes Centre for Radio Astronomy Techniques & Technologies.

Smirnov, OM, & Tasse, Cyril. 2015. Radio interferometric gain calibration as a complex optimization problem. *Monthly Notices of the Royal Astronomical Society*, **449**(3), 2668–2684.

Subhlok, Stichnoth, O'hallaron, & Gross. 1993. Exploiting task and data parallelism on a multicomputer. *Pages 13–22 of: ACM SIGPLAN Notices*, vol. 28. ACM.

Tasse, Cyril. 2014. Applying Wirtinger derivatives to the radio interferometry calibration problem. *arXiv preprint arXiv:1410.8706*.

Thompson, A Richard, Moran, James M, & Swenson Jr, George W. 2008. *Interferometry and synthesis in radio astronomy*. John Wiley & Sons.

van Weeren, RJ, Williams, WL, Hardcastle, MJ, Shimwell, TW, Rafferty, DA, Sabater, J, Heald, G, Sridhar, SS, Dijkema, TJ, Brunetti, G, *et al.* 2016. LOFAR facet calibration. *arXiv preprint arXiv:1601.05422*.

Vuduc, Richard, & Choi, Jee. 2013. A Brief History and Introduction to GPGPU. *Pages 9–23 of: Modern Accelerator Technologies for Geographic Information Science*. Springer.