GPU Accelerated Nonlinear Optimization in Radio Interferometric Calibration

Sarod Yatawatta
ASTRON, the Netherlands Institute for Radio Astronomy,
Postbus 2, 7990 AA,
Dwingeloo,
The Netherlands.
yatawatta@astron.nl

Sanaz Kazemi
Kapteyn Astronomical Institute, University of Groningen,
P.O. Box 800, 9700 AV
Groningen,
The Netherlands.
kazemi@astro.rug.nl

Saleem Zaroubi
Kapteyn Astronomical Institute, University of Groningen,
P.O. Box 800, 9700 AV
Groningen,
The Netherlands.
saleem@astro.rug.nl

ABSTRACT

We present the GPU based acceleration of two well known nonlinear optimization routines: Levenberg-Marquardt (LM) and Limited Memory Broyden-Fletcher-Goldfarb-Shanno (LBFGS) in radio interferometric calibration. Radio interferometric calibration is a heavily compute intensive operation where the same nonlinear optimization problem has to be solved over many time intervals, with different data. We achieve a speedup of about 3 times compared with conventional multi-core CPU based optimization by using GPU accelerated linear algebra routines (CULAtools,CUBLAS). We present details of our GPU accelerated optimization algorithms as well as timing comparisons with non-GPU based multi-core CPU routines.

1. INTRODUCTION

Nonlinear optimization has diverse applications in science, engineering, economics and industry. Such problems are mainly solved using iterative techniques and are therefore demanding in computational cost and thus, they are ideal candidates for acceleration using Graphics Processing Units (GPU). Until recently, few such attempts have been made. For instance, in [1], a conjugate gradient method based nonlinear optimization is considered for image modeling. In [2], a nonlinear optimization method for mesh refinement is proposed. Medical imaging applications based on the GPU accelerated Levenberg-Marquardt (LM) method is considered in [3, 4].

Note that the general procedure for solving any nonlinear optimization problem is by iteratively solving linear problems multiple times. Linear algebra routines based on GPU acceleration has grown significantly over the past few years. The CULA toolkit [5] provides accelerated versions of the standard Linear Algebra Package (LAPACK) [6]. Moreover, CUBLAS [7] provides the equivalent Basic Linear Algebra Subroutines (BLAS) for the GPU.

In this paper, we consider GPU acceleration of radio interferometric calibration. As shown in Fig. 1, data collected by a radio interferometer has to be corrected for the corruptions introduced by the atmosphere (ionosphere, troposphere) as well as the receiver (beam). These corruptions vary over time and therefore, for a long observation lasting many hours, these corruptions have to be estimated many times (and over many frequencies). This is a highly compute intensive optimization problem and some ways of minimizing the computational cost are presented in [8, 9, 10].

In [8, 9], the Space Alternating Generalized Expectation Maximization (SAGE) algorithm is presented for radio interferometric calibration. The *maximization* step of this algorithm is in fact a nonlinear optimization problem.We shall use two optimization routines for this purpose, and we will present the motivation behind using two instead of one later. The Levenberg-Marquardt [11, 12] method is a nonlinear optimization routine with an implicit least squares cost function. The core of this method is based on solving a full system of linear equations. On the other hand, the Limited Memory Broyden-Fletcher-Goldfarb-Shanno (LBFGS) [13] algorithm is an optimization problem which can take any (smooth) cost function. Moreover, this method does not solve a full system of equations and therefore, is more memory efficient. We present the efficient implementation of both algorithms with combined GPU acceleration as well as multi-core CPU BLAS routines [14].

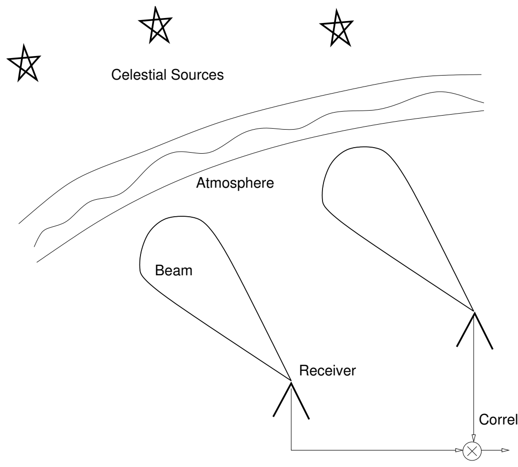


Figure 1: A basic radio interferometer that correlates the signals received from far away celestial sources. The signals are corrupted by the earth's atmosphere as well as by the receiver beam pattern.

We summerize the novelty of this paper: First, as discussed in [8, 9], the calibration approach presented in this paper has linear complexity with respect to the number of celestial sources being solved for, unlike traditional approaches that have quadratic or cubic complexity. Second, we present a hybrid optimization approach, unlike existing approaches that basically rely on the LM algorithm. Thirdly, we present GPU based acceleration of the proposed calibration approach.

The rest of the paper is organized as follows: In section 2, we present an overview of radio interferometric calibration and the underlying nonlinear optimization problems. Next, in section 3, we provide details of the nonlinear optimization routines and aspects of GPU acceleration. Later, in section 4, we provide performance comparisons. Finally, we draw our conclusions in section 5.

Notation: Matrices are denoted in bold uppercase letters (e.g. $\mathbf{A}$) and (column) vectors in bold lowercase letters (e.g. $\mathbf{v}$). The sets of Complex and Real numbers are denoted as $\mathbb{C}$ and $\mathbb{R}$ respectively. The norm of a matrix or a vector is given by $\|\mathbf{A}\|$ or $\|\mathbf{v}\|$ respectively. The identity matrix is given by $\mathbf{I}$ and the transpose, Hermitian transpose and conjugation by $(.)^T$, $(.)^H$ and $(.)^\star$, respectively. The Kronecker product is given by $\otimes$ while $vec(.)$ reorders a matrix to a vector.

2. RADIO INTERFEROMETRIC CALIBRATION

In this section we give a brief overview of radio interferometry and calibration. Consider a pair of stations p and q, forming an interferometer as in Fig. 1. The correlated signal of the two stations is given by

$$\mathbf{V}_{pq} = \sum_{i=1}^{K} \mathbf{J}_{pi}\mathbf{C}_{pqi}\mathbf{J}_{qi}^H + \mathbf{N}_{pq} \tag{1}$$

where $\mathbf{V}_{pq}$ is a 2×2 matrix of complex numbers $\mathbb{C}^{2\times2}$ representing the observation (also called the *visibility* matrix) [15]. This signal is a superposition of electromagnetic radiation emanating from K distinct celestial sources in the sky. In (1), $\mathbf{C}_{pqi}$ ($\in \mathbb{C}^{2\times2}$) represents the source coherency for the i-th source, seen from the interferometer (or baseline) p-q. The values of $\mathbf{C}_{pqi}$ are well known (and stable) and can be calculated for any given p, q, i. As shown in Fig. 1, the electromagnetic radiation from these celestial sources are corrupted by the atmosphere as well as the instrument. These corruptions are represented by Jones matrices [15] given by $\mathbf{J}_{pi}$,$\mathbf{J}_{qi}$ ($\in \mathbb{C}^{2\times2}$) in (1). We also have the noise matrix $\mathbf{N}_{pq}$ ($\in \mathbb{C}^{2\times2}$). We provide an image of the sky made using a radio interferometric simulation in Fig. 2. The image in Fig. 2 (a) has no corruptions in the data while in Fig. 2 (b), the data is corrupted. The artifacts in Fig. 2 (b) are due to the corruptions in the data and these corruptions have to be removed by calibration.

Consider having N stations (or receivers) at distinct locations on earth. Then, we can form $N(N-1)/2$ interferometers which give data as in (1). Calibration is estimating $\mathbf{J}_{pi}$ for all possible p and i given the data $\mathbf{V}_{pq}$. Due to the Earth's rotation as well as due to intrinsic variations, the elements in (1) change both with time as well as with frequency. However, we make a fundamental assumption that within τ time samples, the values of $\mathbf{J}_{pi}$ remain constant.

We reformulate the calibration problem as an optimization problem. As in [8, 9], we first rewrite (1) in vector form as

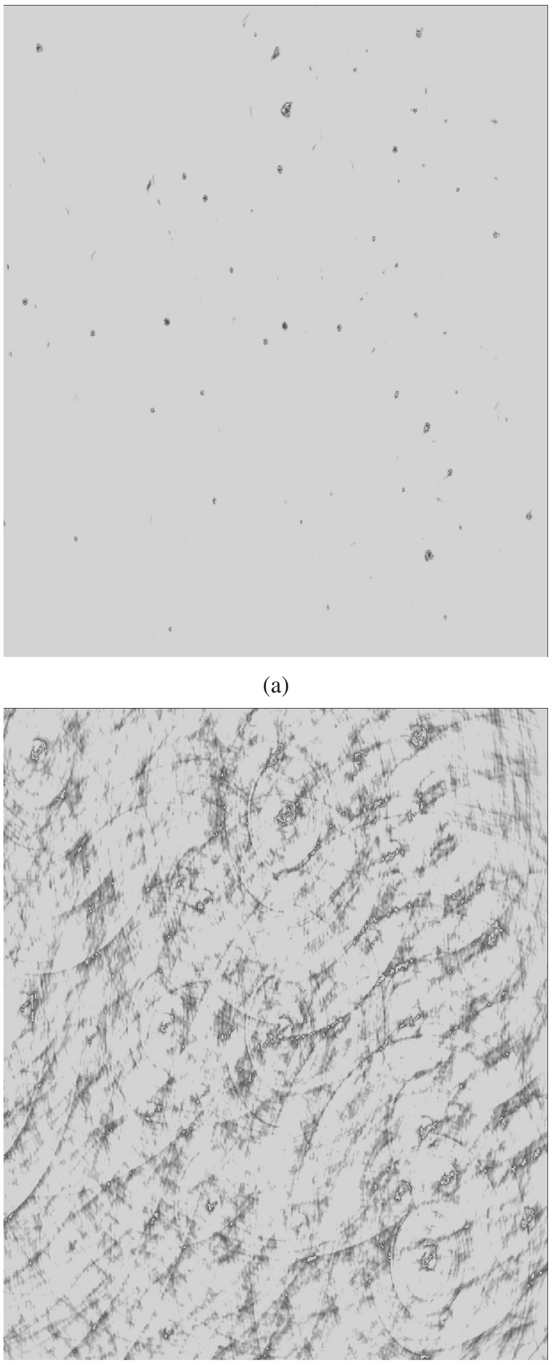
$$\mathbf{v}_{pq} = \sum_{i=1}^{K} \mathbf{s}_{pqi} + \mathbf{n}_{pq} \tag{2}$$

where $\mathbf{s}_{pqi} = (\mathbf{J}_{qi}^\star \otimes \mathbf{J}_{pi})vec(\mathbf{C}_{pqi})$ and $\mathbf{v}_{pq} = vec(\mathbf{V}_{pq})$, $\mathbf{n}_{pq} = vec(\mathbf{N}_{pq})$.

The parameters in our optimization problem are the elements of the Jones matrices and since they are complex numbers, we take the real and imaginary parts separately.

$$\boldsymbol{\theta} = [real(\mathbf{J}_{11}[1,1]), imag(\mathbf{J}_{11}[1,1]), \ldots] \tag{3}$$

Therefore, given N stations and K directions, the number of parameters becomes $N \times K \times 4 \times 2$ and this is the length of the vector $\boldsymbol{\theta}$ ($\in \mathbb{R}^{8NK}$). Considering τ time samples (and assuming $\boldsymbol{\theta}$ to be invariant over this time), stacking up (2) with real and imaginary


(a)

(b)

Figure 2: An image of the sky (about one square degree) made using a simulated interferometric observation (a) without corruptions (b) with corruptions. The artifacts due to these corruptions are clearly visible in (b). The causes of these corruptions are mainly due to the atmosphere (ionosphere, troposphere) and the instrument (beam shape).

parts separately, we get

$$\mathbf{s}_i(\boldsymbol{\theta}) = [real(\mathbf{s}_{12i}^T), imag(\mathbf{s}_{12i}^T), real(\mathbf{s}_{13i}^T), \ldots]^T \quad (4)$$
$$\mathbf{y} = [real(\mathbf{v}_{12}^T), imag(\mathbf{v}_{12}^T), real(\mathbf{v}_{13}^T), \ldots]^T$$

where $\mathbf{s}_i(\boldsymbol{\theta})$ and $\mathbf{y}$ are vectors of size $N(N-1)/2\times 8\times\tau$ ($\in \mathbb{R}^{4\tau N(N-1)}$).

To sum up: We need to estimate $8NK$ parameters using $4\tau N(N-1)$ observed data points according to the data model given by (1) and (2). Exact values for N,K, and τ will vary depending on the radio interferometer used and other observational parameters including frequency, sky coverage, total duration of the observation etc. We give a detailed example in section 4 and typically, both N and K will not be greater than a few hundred at the most. The value for τ depends on the rapidness of the variations of the corruptions over time. The value of τ can be chosen to be just a few to a few hundred. It should be noted that a single observation takes data at a few hundred different frequencies (also called as channels) and the duration of the observation can last a few hours. Therefore, for a full observation, the same set of parameters $\boldsymbol{\theta}$ has to be estimated using data taken at different time and frequency intervals.

We reformulate the estimation of $\boldsymbol{\theta}$ as a nonlinear optimization problem. For the LM optimization routine, we define the cost function as

$$\mathbf{f}(\boldsymbol{\theta}) = \sum_{i=1}^{K} \mathbf{s}_i(\boldsymbol{\theta}) \quad (5)$$

where $\mathbf{f}(\boldsymbol{\theta})$ is a mapping from $\mathbb{R}^{8NK}$ to $\mathbb{R}^{4\tau N(N-1)}$. The LM routine will give the solution for $\boldsymbol{\theta}$ that minimizes the cost $||\mathbf{y}-\mathbf{f}(\boldsymbol{\theta})||^2$. The additional function required for LM is the Jacobian of $\mathbf{f}(\boldsymbol{\theta})$

$$\mathbf{J}(\boldsymbol{\theta}) = \frac{\partial}{\partial\boldsymbol{\theta}}\mathbf{f}(\boldsymbol{\theta}) \quad (6)$$

where $\mathbf{J}(\boldsymbol{\theta})$ is a matrix of size $4\tau N(N-1)\times 8NK$. This can be calculated in closed form using (1) and (2).

For the LBFGS routine, we have a scalar cost function

$$f(\boldsymbol{\theta}) = ||\mathbf{y} - \sum_{i=1}^{K}\mathbf{s}_i(\boldsymbol{\theta})||^2 \quad (7)$$

where $f(\boldsymbol{\theta})$ is a mapping from $\mathbb{R}^{8NK}$ to $\mathbb{R}$. Moreover, the extra ingredient needed for LBFGS is the gradient of $f(\boldsymbol{\theta})$,

$$\nabla f(\boldsymbol{\theta}) = [\frac{\partial f(\boldsymbol{\theta})}{\partial\theta_1}, \frac{\partial f(\boldsymbol{\theta})}{\partial\theta_2}, \ldots]^T \quad (8)$$

where $\nabla f(\boldsymbol{\theta})$ is a vector of size $8NK$ and θ_i is the i-th element in $\boldsymbol{\theta}$. Once again, this can be calculated in closed form using (1) and (2).

As shown in [8, 9], it is possible to consider the elements in $\boldsymbol{\theta}$ corresponding to one direction (say i) to be independent of all other elements in $\boldsymbol{\theta}$. In other words, the mapping $\mathbf{s}_i(\boldsymbol{\theta})$ in (5) can be considered to only be a function of the i-th subset of parameters, which we denote by $\boldsymbol{\theta}_i$. Using this information and considering $\mathbf{s}_i(\boldsymbol{\theta}) = \mathbf{s}_i(\boldsymbol{\theta}_i)$, we define the Jacobian for the i-th parameter subset as

$$\mathbf{J}_i(\boldsymbol{\theta}_i) = \frac{\partial}{\partial\boldsymbol{\theta}_i}\mathbf{s}_i(\boldsymbol{\theta}_i). \quad (9)$$

The function $\mathbf{s}_i(\boldsymbol{\theta}_i)$ is a mapping from $\mathbb{R}^{8N}$ to $\mathbb{R}^{4\tau N(N-1)}$ while the Jacobian $\mathbf{J}_i(\boldsymbol{\theta}_i)$ is a matrix of size $4\tau N(N-1)\times 8N$. This enables us to use the SAGE and Expectation Maximization (EM) algorithms to reduce the computational as well as memory cost in our optimization, as shown in [8, 9].

3. NONLINEAR OPTIMIZATION ROUTINES

In this section, we present details of our GPU accelerated calibration algorithm as well as details of the LM and LBFGS algorithms. We emphasize that it is possible to select the hardware configuration to get the maximum performance of a given algorithm, for example by using more CPUs than GPUs. However, in this work, we select the converse: In other words, we tune our algorithm to get the best performance from a given hardware configuration. That way, the same hardware can be used in other aspects of radio interferometry with equally optimized algorithms, such as in imaging. Moreover, we have used GPU equivalents of BLAS and LAPACK routines (CUBLAS and CULA) as much as possible, rather than reimplementing them from scratch. The motivation for selecting CUBLAS and CULA was mainly due to their similarity to classic BLAS and LAPACK interfaces.

In listing 1, we present our calibration algorithm suitable for a computer with 2 GPUs. We have selected the parts of the algorithm that require most amount of computations to be run on the GPUs. Some computations require significant data transfer between the host and the GPUs and we have chosen to execute them on the host.

Listing 1 Calibration

```
Require: Data $\mathbf{y}$, mappings $\mathbf{f}(\boldsymbol{\theta})$,$f(\boldsymbol{\theta})$,$\mathbf{s}_i(\boldsymbol{\theta}_i)$,..., Jacobians
    $\mathbf{J}_1(\boldsymbol{\theta}_1)$,..., gradient $\nabla f(\boldsymbol{\theta})$
 1: Initialize $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}^0$
 2: $k \leftarrow 1$
 3: while $k <$ max EM iterations do
 4:   Residual $\mathbf{r} = \mathbf{y} - \mathbf{f}(\boldsymbol{\theta})$
 5:   Set of directions $D = \{1, 2, \ldots, K\}$
 6:   while $D$ is not empty do
 7:     $(m, n) \leftarrow$ remove two directions from $D$
 8:     $\mathbf{y}_1 \leftarrow \mathbf{s}_m(\boldsymbol{\theta}_m) + 0.5\mathbf{r}$
 9:     $\mathbf{y}_2 \leftarrow \mathbf{s}_n(\boldsymbol{\theta}_n) + 0.5\mathbf{r}$
10:     Update residual $\mathbf{r} \leftarrow \mathbf{r} + \mathbf{s}_m(\boldsymbol{\theta}_m) + \mathbf{s}_n(\boldsymbol{\theta}_n)$
11:     LM min$||\mathbf{y}_1 - \mathbf{s}_m(\boldsymbol{\theta}_m)||^2$ over $\boldsymbol{\theta}_m$ with $\mathbf{s}_m(\boldsymbol{\theta}_m)$ and $\mathbf{J}_m(\boldsymbol{\theta}_m)$
12:     LM min$||\mathbf{y}_2 - \mathbf{s}_n(\boldsymbol{\theta}_n)||^2$ over $\boldsymbol{\theta}_n$ with $\mathbf{s}_n(\boldsymbol{\theta}_n)$ and $\mathbf{J}_n(\boldsymbol{\theta}_n)$
13:     Update $\boldsymbol{\theta}$ with new values for $\boldsymbol{\theta}_m$ and $\boldsymbol{\theta}_n$
14:     Update residual $\mathbf{r} \leftarrow \mathbf{r} - \mathbf{s}_m(\boldsymbol{\theta}_m) - \mathbf{s}_n(\boldsymbol{\theta}_n)$
15:   end while
16:   $k \leftarrow k + 1$
17: end while
18: LBFGS min $f(\boldsymbol{\theta})$ over all $\boldsymbol{\theta}$, using current $\boldsymbol{\theta}$ as starting point
19: return $\boldsymbol{\theta}$
```

We make additional comments about listing 1 as follows:

- Line 1: We take initial values such that $\mathbf{J}_{pi}$ matrices are equal to $\mathbf{I}$. Over time, we initialize them by the solutions obtained from the previous time interval.
- Line 4: The residual calculation is done on the host.
- Line 7: We select two directions here because we have two GPUs. If only one GPU is available, we only select one direction here. Also, if K is odd, for one update, we use only one GPU. It is straightforward to extend this selection to more than two in the case we have more GPUs. It is also possible to schedule multiple LM runs on the same GPU, provided that it has enough memory and compute capability.
- Lines 8-9: This is actually calculating the conditional mean and is computed on the host.
- Line 10 and 14: The residual is updated as the parameters $\boldsymbol{\theta}$ are updated by LM, on the host.

- Lines 11-12: Each LM optimization is run concurrently on a different GPU (see listing 2).

- Line 18: LBFGS uses both GPUs (see listing 3).

Listing 2 Levenberg-Marquardt [16, 17]

```
Require: Data $\mathbf{y}$, mapping $\mathbf{f}(\boldsymbol{\theta})$, Jacobian $\mathbf{J}(\boldsymbol{\theta})$
 1: $k \leftarrow 0; \nu \leftarrow 2; \boldsymbol{\theta} \leftarrow \boldsymbol{\theta}^0$
 2: $\mathbf{A} \leftarrow \mathbf{J}(\boldsymbol{\theta})^T\mathbf{J}(\boldsymbol{\theta}); \mathbf{e} \leftarrow \mathbf{y} - \mathbf{f}(\boldsymbol{\theta}); \mathbf{g} \leftarrow \mathbf{J}(\boldsymbol{\theta})^T\mathbf{e}$
 3: found $\leftarrow (\|\mathbf{g}\|_\infty < \epsilon_1); \mu \leftarrow \tau \max \mathbf{A}_{ii}$
 4: while (not found) and ($k <$ max iterations) do
 5:     $k \leftarrow k + 1$; Solve $(\mathbf{A} + \mu\mathbf{I})\mathbf{h} = \mathbf{g}$
 6:     if $\|\mathbf{h}\| < \epsilon_2(\|\boldsymbol{\theta}\| + \epsilon_2)$ then
 7:         found $\leftarrow$ true
 8:     else
 9:         $\boldsymbol{\theta}_{new} \leftarrow \boldsymbol{\theta} + \mathbf{h}$
10:         $\rho \leftarrow (\|\mathbf{e}\|^2 - \|\mathbf{y} - \mathbf{f}(\boldsymbol{\theta}_{new})\|^2)/(\mathbf{h}^T(\mu\mathbf{h} + \mathbf{g}))$
11:         if $\rho > 0$ step acceptable then
12:             $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}_{new}$
13:             $\mathbf{A} \leftarrow \mathbf{J}(\boldsymbol{\theta})^T\mathbf{J}(\boldsymbol{\theta}); \mathbf{e} \leftarrow \mathbf{y} - \mathbf{f}(\boldsymbol{\theta}); \mathbf{g} \leftarrow \mathbf{J}(\boldsymbol{\theta})^T\mathbf{e}$
14:             found $\leftarrow (\|\mathbf{g}\|_\infty \leq \epsilon_1)$
15:             $\mu \leftarrow \mu \max(1/3, 1 - (2\rho - 1)^3); \nu \leftarrow 2$
16:         else
17:             $\mu \leftarrow \mu\nu; \nu \leftarrow 2\nu$
18:         end if
19:     end if
20: end while
21: return $\boldsymbol{\theta}$
```

In listing 2, we present the LM algorithm, similar to as presented in [16, 17]. The following additional comments can be made about listing 2:

- $\mathbf{A}$,$\mathbf{e}$,$\mathbf{g}$,$\mathbf{h}$,$\boldsymbol{\theta}_{new}$ as well as a copy of $\boldsymbol{\theta}$ are stored in the GPU.

- The initial memory transfers from host to GPU include data $\mathbf{y}$, initial parameters $\boldsymbol{\theta}$ and additional data required to calculate $\mathbf{f}(\boldsymbol{\theta})$ and $\mathbf{J}(\boldsymbol{\theta})$. This includes source coherencies $\mathbf{C}_{pqi}$ in (1).

- Line 2: We evaluate both $\mathbf{f}(\boldsymbol{\theta})$ and $\mathbf{J}(\boldsymbol{\theta})$ on the GPU.

- Lines 2 and 13: We evaluate $\mathbf{J}(\boldsymbol{\theta})^T\mathbf{J}(\boldsymbol{\theta})$ using `DGEMM`, $\mathbf{J}(\boldsymbol{\theta})^T\mathbf{e}$ using `DGEMV` routines provided by CULAtools [5].

- Line 5: For solving $(\mathbf{A} + \mu\mathbf{I})\mathbf{h} = \mathbf{g}$, we use three different approaches (i) Cholesky Factorization, (ii) QR Factorization and (iii) the singular value decomposition (SVD), all provided by CULAtools. The addition of μ to the diagonal of $\mathbf{A}$ is done using a CUDA [18] kernel. Because $\mathbf{A}$ is modified while solving the system of equations, we keep a copy of $\mathbf{A}$ on the GPU. While using the SVD, we use a CUDA kernel to eliminate singular values close to zero.

- Line 21: The final memory transfer from GPU to host is mainly the updated value of $\boldsymbol{\theta}$.

- All vector operations and norm calculations are done by using CUBLAS.

- All conditionals and loops (the `while` loop on line 4, the `if` conditionals on lines 3, 6, 11, 14) are run on the host, using references to GPU memory locations.

The implementation of LBFGS is given in listing 3, extracted from [13]. Note that because of memory limitations, we never explicitly construct $\mathbf{H}_k$, instead we calculate this by M pairs of vectors $\mathbf{u}_k$ and $\mathbf{v}_k$. For additional information, the reader is referred to [13], chapter 9.

Listing 3 LBFGS [13]

```
Require: Data $\mathbf{y}$, mapping $f(\boldsymbol{\theta})$ gradient $\nabla f(\boldsymbol{\theta})$, starting point $\boldsymbol{\theta}^0$,
    memory $M > 0$
 1: $k \leftarrow 0$
 2: repeat
 3:     Choose $\mathbf{H}_k^0$
 4:     $\mathbf{p}_k \leftarrow -\mathbf{H}_k\nabla f(\boldsymbol{\theta})$
 5:     $\boldsymbol{\theta}_{new} \leftarrow \boldsymbol{\theta} + \alpha_k\mathbf{p}_k$
 6:     Where $\alpha_k$ chosen using line search
 7:     if $k > M$ then
 8:         Discard the vector pair $\mathbf{u}_{k-M}$, $\mathbf{v}_{k-M}$ from storage
 9:     end if
10:     Compute and save $\mathbf{u}_k \leftarrow \boldsymbol{\theta}_{new} - \boldsymbol{\theta}; \mathbf{v}_k \leftarrow \nabla f(\boldsymbol{\theta}_{new}) - \nabla f(\boldsymbol{\theta})$
11:     $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}_{new}$
12:     $k \leftarrow k + 1$
13: until convergence or $k >$ max iterations
14: return $\boldsymbol{\theta}$
```

Additional comments about listing 3 are as follows:

- We run all code on the host, using multi-core GOTOBLAS routines [14], except the calculation of gradient $\nabla f(\boldsymbol{\theta})$.

- The initial memory transfers from host to GPU include data $\mathbf{y}$, initial parameters $\boldsymbol{\theta}$ and additional data required to calculate $\nabla f(\boldsymbol{\theta})$. This includes source coherencies $\mathbf{C}_{pqi}$ in (1).

- Line 4: The gradient calculation is done using both GPUs. The cost function (7) can be considered as an inner product of a vector (of length $4\tau N(N-1)$). This inner product is divided into two inner products of length $2\tau N(N-1)$ and this is calculated using the two GPUs. For each parameter in (8), we have to calculate this summation (after evaluating the derivative of each element of the vector in closed form).

- Line 4: The memory transfer from the GPU to the host is a scalar, one for each parameter in (8). However, for each iteration, we need to transfer the updated value of $\boldsymbol{\theta}$ from the host to both GPUs.

- Lines 3, 4 and 6: More detail about these steps can be found in [13], chapter 9.

To wrap up this section, we discuss the motivation behind using two optimization algorithms in our calibration approach. The dimension of the Jacobian for the full problem in (6) is $4\tau N(N-1) \times 8NK$, that is prohibitively expensive for GPUs (and even for a normal computer) for large values of N, K, and τ. Therefore, we choose the EM algorithm in combination with the LM optimization to significantly cut down the storage requirement. In the proposed approach, this memory requirement is only $4\tau N(N-1) \times 8N$ and is feasible for a GPU. The LM algorithm is a combination of both steepest descent as well as Newton nonlinear optimization methods. Therefore, when we are far away from the desired solution, the combination of steepest descent and Newton methods enable us to reach a point close to the optimal solution, albeit only for a subset of the parameters at a time. At this point, we switch to LBFGS which is more memory efficient and uses the full parameter space. In this way, we overcome the limitation of slow convergence of the

LBFGS algorithm. As mentioned before, since we have to solve the same optimization problem over many time and frequency intervals, we have to keep the computational time spent on a single calibration fixed. In other words, we have to keep the number of iterations fixed, regardless of convergence in our calibration. Therefore, by the proposed scheme, it is possible to reach a solution that drives the error below a desired threshold with a fixed number of iterations.

4. PERFORMANCE EVALUATION

We consider the performance of the proposed algorithm in a computer with 8 (2 CPUs with 4 cores each) Intel Xeon E5520 cores (with hyperthreading) and two NVIDIA Tesla M1060 GPUs. The host memory is 12 GB and each GPU has 4 GB memory. For comparison, we have our algorithm implemented without GPU acceleration and only using GOTOBLAS [14] low level BLAS and LAPACK routines (using 16 threads). We calibrate a simulated interferometric observation with $N = 50$ stations while varying the number of directions K that we calibrate. For each calibration, we use $\tau = 120$ time samples (corresponding to 20 minutes of data) and we find the average compute time (wall clock) over different time and frequency intervals. All computations are done using double precision arithmetic.

For the optimization we chose the following parameters, mainly to reach the required level of accuracy desired with minimum amount of computation: For the EM stage, we choose 3 EM iterations, with 4 LM iterations within each EM iteration. For the LBFGS stage, we choose 10 iterations with memory size (size of the past history used in calculating the step direction) of $M = 7$. We have also tested three linear system solvers in the LM stage: (i) Cholesky factorization, (ii) QR factorization, and (iii) the SVD.

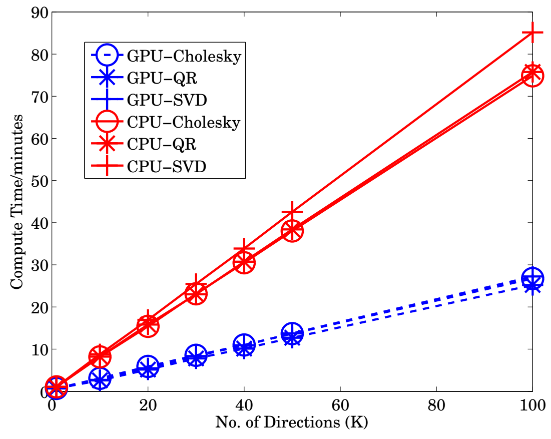


Figure 3: Timing comparison between GPU accelerated calibration (dashed-blue lines) and calibration without GPU acceleration (solid-red lines). Both versions have three different linear solvers in the LM stage. The GPU accelerated version is about three times faster.

In Fig. 3, we have given the timing results for the full algorithm. The time of the LM and LBFGS stages are given in Figs. 4 and 5, respectively. We make a few observations based on these results: First, the GPU accelerated version is about three times faster than the one without GPU acceleration. Moreover, the bulk of the compute time is spent on the LM algorithm, where we see a factor of 3 speedup. Since we only calculate the derivative using the GPUs in the LBFGS stage, the speedup is not significant but the total time spent on LBFGS is much smaller. Secondly, both CPU and GPU versions have linear complexity with the number of directions being calibrated. A more subtle fact is that while the SVD based linear solver takes more time in the non-GPU based version (as expected), all three solvers take the same amount of time in the GPU accelerated version. Generally, the SVD based linear solver is the most robust of the three and we can use this solver in the LM optimization without any additional cost. It should also be noted that conventional calibration has quadratic (and even cubic) complexity and the approach proposed in [8, 9] is much faster.

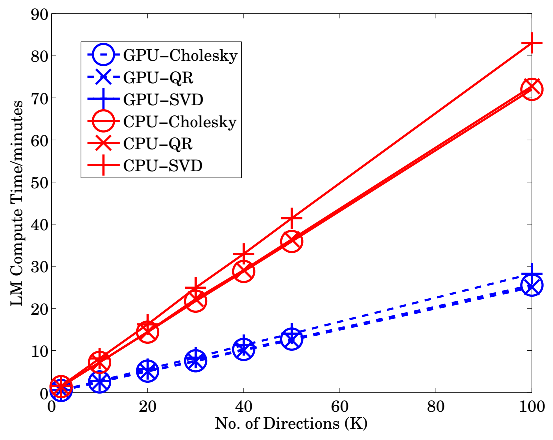


Figure 4: Timing comparison of LM with GPU acceleration (dashed-blue lines) and without GPU acceleration (solid-red lines). Both versions have three different linear solvers.

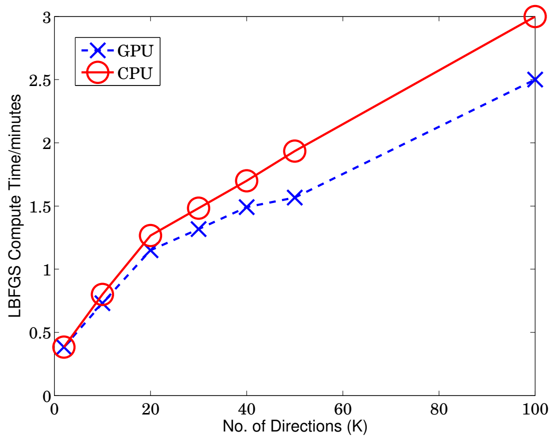


Figure 5: Timing comparison LBFGS with GPU acceleration (dashed-blue line) and without GPU acceleration (solid-red line).

5. CONCLUSIONS

We have demonstrated the feasibility of accelerating radio interferometric calibration using GPUs. We have done this by optimizing the underlying optimization routines for the GPU. As both GPU hardware and low level software improves, we hope to increase the throughput gained by GPU acceleration in the future.

6. ACKNOWLEDGEMENTS

We thank the members of the LOFAR Epoch of Reionization key science project for valuable comments and suggestions as well as for providing the GPU computing cluster as a testbed. We also thank the anonymous reviewers for a careful and insightful review.

References

[1] K. E. Hillesland, S. Molinov, and R. Grzeszczuk, "Nonlinear optimization framework for image based modeling on programmable graphics hardware," *in proc. ACM SIGGRAPH*, 2003.

[2] S. Dong and M. Garland, "Iterative methods for improving mesh parameterizations," *in proc. IEEE Shape Modeling International Conference*, 2007.

[3] B.R. Smith, G. Hamarneh, and A. Saad, "Fast GPU fitting for kinetic models for dynamic PET," *in proc. International Workshop on High-Performance Medical Image Computing for Image-Assisted Clinical Intervention and Decision-Making (MICCAI HP)*, 2010.

[4] B. Li, A. Young, and B. Cowan, "GPU accelerated non rigid registration for the evaluation of cardiac function," *in proc. Medical image computing and computer assisted intervension*, vol. 5242, pp. 880–887, 2008.

[5] J.R. Humphrey, D.K. Price, K.E. Spagnoli, A.L. Paolini, and E.J. Kelmelis, "CULA: Hybrid GPU accelerated linear algebra routines," *in proc. SPIE Defense and Security Symposium (DSS)*, 2010.

[6] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, *LAPACK Users' Guide, Third Ed.*, Philadelphia USA: SIAM, 1999.

[7] CUBLAS, `http://developer.nvidia.com/cublas`.

[8] S. Yatawatta, S. Zaroubi, G. de Bruyn, L. Koopmans, and J. Noordam, "Radio interferometric calibration using the SAGE algorithm," *in proc. 13th IEEE DSP workshop*, pp. 150–155, Jan. 2009.

[9] S. Kazemi, S. Yatawatta, and S. Zaroubi, "Clustered radio interferometric calibration," *in proc. IEEE Statistical Signal Processing Workshop (SSP), Nice, France*, Mar. 2011.

[10] S. Kazemi, S. Yatawatta, S. Zaroubi, P. Labropoluos, G. de Bruyn, L. Koopmans, and J. Noordam, "Radio interferometric calibration using the SAGE algorithm," *MNRAS*, vol. 414, no. 2, pp. 1656–1666, June 2011.

[11] K. Levenberg, "A method for the solution of certain non linear problems using least squares," *The Quarterly Jnl. of App. Math.*, vol. 2, pp. 164–168, 1944.

[12] D. Marquardt, "An algorithm for least squares estimation of nonlinear parameters," *SIAM Jnl. of App. Math.*, vol. 11, pp. 431–441, 1963.

[13] J. Nocedal and S. J. Wright, *Numerical Optimization*, New York USA:Springer, 1999.

[14] K. Goto and R. van de Geijn, "Anatomy of high-performance matrix multiplication," *ACM Transactions on Mathematical Software*, vol. 34-3, pp. 1–25, 2008.

[15] J. P. Hamaker, J. D. Bregman, and R. J. Sault, "Understanding radio polarimetry, paper I," *Astronomy and Astrophysics Supp.*, vol. 117, no. 137, pp. 96–109, 1996.

[16] M.I.A. Lourakis, "levmar: Levenberg-Marquardt nonlinear least squares algorithms in C/C++," `http://www.ics.forth.gr/~lourakis/levmar/`, 2004.

[17] K. Madsen, H.B. Nielsen, and O. Tingleff, "Methods for non-linear least squares problems," *Lecture Notes: Technical University of Denmark*, 2004.

[18] CUDA, `http://www.nvidia.com/cuda`.