# Optimizing a calibration software for radio astronomy

Souley Madougou
IvI, University of Amsterdam
Amsterdam, The Netherlands
Email: S.Madougou@uva.nl

Ana L. Varbanescu
IvI, University of Amsterdam
Amsterdam, The Netherlands
Email: A.L.Varbanescu@uva.nl

Rob van Nieuwpoort
Netherlands eScience Center
Amsterdam, The Netherlands
Email: R.vanNieuwpoort@esciencecenter.nl

***Abstract*—With the turn to multicore in chip design and manufacturing, consumer and high performance applications likewise can benefit from ubiquitous hardware parallelism. In this paper, we present the parallelization of a model fitting algorithm implementation used in calibrating telescope observation data in radio astronomy. Despite some strict limitations in code modification, we show how classical "bound-and-bottleneck" analysis and optimization using multicore architectures help achieving up to 2.3x speedup compared to original sequential implementation.**

## I. Introduction

Nowadays, all scientific domains use some kind of computational modeling and/or simulation giving raise to what is now known as computational science. Advances in both computing hardware and scientific experiments devices have made scientific applications even more demanding in terms of computational and storage resources. Thus, the world's most powerful supercomputers are being built for and operated by scientific institutes and organizations. The increasing use of computing infrastructure has considerably changed the practice of science in many areas.

An interesting case is radio astronomy. Traditional radio telescopes needed to be big in order to improve their resolving power and sensitivity. So, to probe a deeper universe, radio astronomers used to build bigger and more expensive telescopes. As this trend cannot be pursued, a shift is done towards building synthesis arrays consisting of simpler hardware back-ended with complex software processing. To address this challenge, high performance computing (HPC) infrastructure is used. In the case of The Netherlands based LOw Frequency ARray (LOFAR) international telescope, a Blue Gene/P (BG/P) [1] connected with high-bandwidth networks to the antenna sites is used. Signals from individual antennas are first gathered and preprocessed on on-site processing cabinets (called stations) from where they are sent to the central BG/P.

Telescopes are essentially imaging devices just like our eyes or a CCD camera. They concentrate radio signals from the sky onto receivers that amplify the signals before converting them from analog to digital. Signals are then filtered, beamformed, cross-correlated and written to a temporary storage space as raw `visibilities`. This data is later read by different pipelines corresponding to different key projects such as the transients key project concerned with the discovery of transients. Before doing any useful processing with that data, it needs to be *calibrated*. `Calibration` consists in removing from the measured visibilities imperfections due to environmental and instrumental effects. To this aim, a parameterized model of those effects, which implements the `measurement equation` (ME) [2], is built. Measured visibilities are then used to `fit` model parameters. Practically, calibration boils down to a model fitting problem. Although research in numerical algorithms has been very successful in addressing fitting problems, the nonlinearity of the parameters with regard to the model prevents from using the myriad of simpler linear methods. Only nonlinear methods can satisfactorily handle the calibration case.

In this paper, we describe one of the most used calibration software package in LOFAR processing pipelines, the BlackBoard Selfcal (*a.k.a.* BBS). We will primarily focus on the model fitting aspects. Because of the complexity and size of the LOFAR software to which BBS is tightly coupled, it has been decided that the code modification should be minimal and yet optimally use the available computing resources. Using code analysis, profiling and data dependencies in BBS, we show it can be enhanced for performance using several optimization approaches including explicit parallelism via multithreading, use of optimized linear algebra libraries and accelerator technologies. Combining those approaches leads to up to 2.3x speedup compared to the original sequential code. The remainder of this paper is structured as follows: in section II, we give some background about BBS, briefly describing the physics and mathematics behind it and also its design and performance issues. We present our optimization approaches in section III and then, detail their implementation and evaluation using real data from recent LOFAR observations and discuss the results in section IV. We relate our contribution to previous work and discuss about alternative optimization methods in section V before section VI concludes the paper and draws future research directions.

## II. Background of BBS

BBS is used to perform off-line reduction of the large volumes of raw visibilities for further processing such as imaging. Calibration and simulation of LOFAR data are 2 important operations for which BBS is designed. From this perspective, the core functionality of BBS can be split in 2 parts: simulation of visibilities given a parameterized ME and estimating improved values given a set of simulated and a set of observed visibilities. BBS is named after the blackboard design pattern which denotes its software architecture and the type of calibration it implements, namely self-calibration,

meaning the process uses observation data to perform the calibration. BBS takes as input observation data, a source catalog, a table of model parameters and a configuration file or `parset`. In radio astronomy, observations are typically stored in the measurement set format or `MS`. An MS is actually a directory where data and metadata are stored in tables and subdirectories in that directory. The source catalog or a local sky model (LSM) lists the sources in the part of the sky being observed by the telescope. As output, BBS produces a processed observation and a table of estimated model parameters.

### A. The physics in BBS

As stated in section I, calibration consists in fitting the measured visibilities into the ME model. The latter formalizes several physical phenomena affecting the electromagnetic signal path from the emitting source in the sky to the measuring instrument. Those effects include, among others, ionospheric phase and amplitude delay, Faraday rotation, tropospheric phase delay, station beamshape, the complex gain corresponding to the electronic gain of the antenna receiver, etc. A detailed description of LOFAR calibration can be found in [3]. For a given telescope and a given source distribution, a parameterization of the ME captures all these effects and predicts the visibilities for that setting.

Indeed, using the 2x2 matrix formulation of the ME, a cross-correlated visibility from the interferometer formed by the stations $a$ and $b$ is a 2x2 complex matrix which can be written down as:

$$V_{ab} = G_a \left( \sum_{k=1}^{N} E_{ak} X_k E_{bk}^* \right) G_b^* \tag{1}$$

where $N$ is the number of sources visible in the local sky and $G_a^*$ is the conjugate transpose (or Hermitian) of matrix $G_a$. As the received radiation is decomposed into 2 orthogonal polarization directions, each station has dual matching feeds inducing 2 different voltages leading to 4 measurements per visibility. The contribution (`source coherency`) from the source `k`, $X_k$, is corrupted by instrumental and environmental effects represented by the *Jones matrices*: $E_{ak}$ for the direction-dependent effects associated with station $a$ and the direction towards `k` and representing a product of several other matrices and $G_a$, also a product of matrices, for direction-independent effects associated with station $a$. The Jones matrices are considered in the same order as the effects they represent corrupt the signal path, which is important given that matrix multiplication does not generally commute. However, in practice, knowledge of the structure of those matrices (often scalar, rotation or diagonal) is used to move them around for computational efficiency. For instance, the Jones matrix for ionospheric phase delay is scalar (so, can be located anywhere in Equation (1)), while that of the Faraday rotation is a rotation matrix. Last but not least, Equation (1) assumes that all instrumental effects are station-based; this increases the ratio between the number of equations, given by measured visibilities, and the number of unknowns (over-determination), possibly leading to ill-conditioned nonlinear equation systems.

### B. The model fitting algorithm

Calibration consists in minimizing the difference, in the least squares sense, between measured visibilities and those predicted by a parameterized ME. Given a visibility $M(\nu_i, t_j; p)$ as predicted by Equation (1) and $V_{ij}$ measured by the telescope for the same $(\nu_i, t_j)$), find the value of the parameter vector $p$ which minimizes the norm of the difference. We use subscripts in above definition to show that both predicted and measured values can be, and usually are, functions of other variables as well, such as frequency and time in a typical radio astronomy case. Because of the nonlinearity of the parameters with relation to the model, only nonlinear least squares methods are considered. The one currently implemented in BBS is the Levenberg-Marquardt (LM) algorithm [4], [5], a popular method in addressing nonlinear least squares problems. It is a stable and robust iterative method oscillating between the Gauss-Newton and the steepest descent methods. Nonlinear least-squares (LSQ) methods, as well as linear LSQ, proceed by defining a merit function [6], usually noted $\chi^2$, and by determining best-fit parameter values through its minimization. Considering above notations, $\chi^2$ is defined as follows:

$$\chi^2(p) = \sum_{ij} \omega_{ij}^2 \left( V_{ij} - M(\nu_i, t_j; p) \right)^2 \ , \tag{2}$$

where $\omega_{ij}$ are optional weights corresponding to, *e.g.*, the measurement error or standard deviation associated with each $V_{ij}$ (supposed to be known, otherwise set to 1). Temporarily ignoring weights, let $f$ be the vector field whose components are $f_i(p) = V_i - M(\nu_i; p)$. As LM is an improved Gauss-Newton (GN) method [7], it is based on a linear approximation to the components of $f$ in the neighborhood of $p$. For $\|h\|$ sufficiently small, a Taylor series expansion leads to

$$f(p + h) \approx f(p) + J(p)h \tag{3}$$

with $J$ the Jacobian matrix of $f$ and $\|.\|$ denotes either $\|.\|_2$ or $\|.\|_\infty$. Inserting this definition into Equation (2) leads to a linear model $L$ of $\chi^2$

$$\chi^2(p + h) \approx L(h) = \chi^2(p) + h^T J^T f(p) + \frac{1}{2} h^T J^T J h \tag{4}$$

where the superscript $^T$ means transpose. The GN step $h_{gn}$ minimizes $L(h)$ and can be found by solving the so-called *normal equations*

$$(J^T J) h_{gn} = -J^T f \tag{5}$$

and the current iterate is updated following $p = p + h_{gn}$. However, the GN method may fail to find the actual minimizer because the update procedure is not done in a controlled manner. To prevent failure, Levenberg and Marquardt suggested to use a *damped* GN by solving the following equation instead:

$$(J^T J + \lambda I) h_{lm} = -g \tag{6}$$

with $g = J^T f$, $\lambda \geq 0$ is real and $I$ is the identity matrix. The *damping factor* $\lambda$ has several effects:

- $\lambda > 0$ ensures that $h_{lm}$ is a descent direction,
- if $\lambda$ is large, then we have a short step in the steepest direction, which is good if the current iterate is far from the solution,

- if $\lambda$ is very small, then $h_{lm} \simeq h_{gn}$, meaning LM falls back to GN, which is good in the final stages of the iterations when $p$ is close to the solution.

Initiated at the initial parameter guess $p_0$, the LM method produces a series of vectors $p_1$, $p_2$, ... that converge towards a local minimizer $p^*$ for $\chi^2$. So, at each step, it is required to find a $\delta_p$ that minimizes $\chi^2$. The sought $\delta_p$ is solution to a linear least-squares problem (corresponds to $h_{lm}$ in Equation (6)). If the updated parameter vector $\mathrm{p}$ leads to a reduction in $\chi^2$, the update is accepted and the process is repeated with a decreased damping factor. Otherwise, the damping factor is increased, the normal equations are solved again and the process iterates until a value of $\delta_p$ that decreases $\chi^2$ is found.

Both the Jacobian $J$ and the approximate Hessian $J^T J$ of $f$ need to be computed. Usually, the second partial derivatives in the Hessian are dropped. The key operational parameters of the method are $\lambda$ initial value and the stopping criterion determination. The choice of the initial value of $\lambda$ should be related to the size of the elements in $A^0 = J(p_0)^T J(p_0)$, for instance, setting $\lambda_0 = \tau * max_i \left(a_{ii}^0\right)$, with $\tau$ specified by the user. Different authors propose different ways of updating the damping factor and of stopping the iterations. Press *et al.* [6] propose to increase or decrease $\lambda$ by a factor of 10 and to stop iterating when either the fractional decrease in $\chi^2$ is less or equal to 0.001 or the number of iterations exceeds a maximum specified by the user.

### C. Current Implementation

BBS is primarily designed to run on a compute cluster to cope with the LOFAR high data rate (several GBs per second) in a so-called *distributed mode*. Recall that raw visibilities are received from the correlator as a set of several hundreds of sub-bands, or frequency slices. To the aim of efficacy, BBS consists of 3 separate components: `GlobalControl`, `KernelControl` and `SolverControl`. The GlobalControl process monitors and controls a set of KernelControl processes and, possibly one or more SolverControl processes. Each sub-band is processed by a separate KernelControl process which computes a set of equations which it sends to a designated SolverControl process for a new estimate of the model parameters. It turns out that, often, a single sub-band already provides enough signal for parameter estimation. In that case, one can run in so-called *standalone mode* where BBS consists of only a KernelControl and a SolverControl running sequentially on one node. As this is the mode used daily by most radio astronomers, optimizing this mode will give them more time to concentrate on their scientific goals. The optimization will be also beneficial to the distributed mode as the latter uses the same processes spawn on different nodes. The processes' layout is illustrated in Figure 1.

To use BBS for data reduction, the user provides a reduction `strategy` in the parset. A strategy is a sequence of operations (or `steps`) to be performed on the visibility data. Common steps used are `flagging` to remove or ignore bad data (*e.g.* those affected by radio frequency interference), `solve` to fit observed data to a parameterized ME, and `correct` to write back the updated parameter values to the MS.

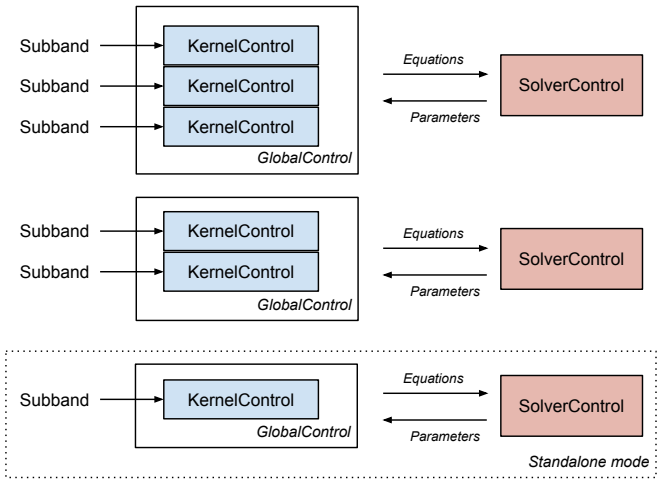


Figure 1: BBS processes' layout. The schematic shows 3 calibration groups monitored by 3 different GlobalControl processes containing 3, 2 and 1 KernelControl, respectively, each with a designed SolverControl. The last one in the dotted box corresponds to a run in standalone mode.

For the solve step, BBS currently uses an implementation [8] of the LM algorithm from the AIPS++/CASA library[1]. Visibilities are first partitioned into chunks themselves subdivided into a number of smaller solve domains (called `cells`), each with its own solver. The chunks are then iterated on. For each chunk, normal equations are first generated for all the cells, then the equations are solved for on a per-cell basis. The whole process of generating and solving the normal equations for the cells of a chunk is repeated until a stopping criterion is met (either all cells have converged or a maximum number of iterations is reached). Both operations make call to routines of the AIPS++/CASA solver.

The main tasks involved in the fitting process are as follows:

**Model evaluation:** Simulated visibilities ($M(\nu_i, t_j; p)$) are constructed by evaluation of the model using the current values of the parameters. The elements of $J$ which are partial derivatives of $M$ w.r.t. $p$ are evaluated here. They are approximated using a perturbed value $\delta_k$ for each parameter $p_k$ following the equation below:

$$\frac{\delta M}{\delta p_k}(\nu_i, t_j; p) = \frac{M(\nu_i, t_j; p_k + \delta_k) - M(\nu_i, t_j; p_k)}{\delta_k} \qquad (7)$$

where $M(\nu_i, t_j; p_k + \delta_k)$ is evaluated with the value of the $k$-th parameter incremented with $\delta_k$.

**Equation generation:** A set of linear equations is constructed using the *residuals* obtained by subtracting measured ($V_{ij}$) from simulated visibilities ($M(\nu_i, t_j; p)$, multiplied by some weighting factors, giving the values of elements of the normal equation matrix. The diagonal elements of this matrix will later be multiplied by $(1 + \lambda)$ leading to the modified normal equations in Equation (6).

[1] http://casa.nrao.edu/index.shtml

**Solution Computation:** Solve the linear system for the residuals and update the values of the parameters, *i.e.* $p_k$ becoming $p_k + \delta_{p_k}$.

Therefore, from the fitting problem perspective, the following sketch summarizes the calibration process:

Listing 1: Fitting algorithm in BBS

```
1  subdivide visibilities into chunks
2  for (chunk=0, chunk < nb_of_chunks; chunk++) {
3    ...
4    while (!(all cells converged || num_iter >=
         MAX_ITER)) {
5      evaluate model for each (baseline,
           polarization) pair
6      generate normal equations for all cells in
           chunk
7      solve equations for all cells in chunk
8    }
9    ...
10 }
```

### D. Performance of the current code

Profiling the application, we discovered two main hot-spots consuming up to 40% (line 6 of Listing 1) and 60% (line 7 of Listing 1), respectively, of the CPU time spent by the whole process. Note that these numbers correspond to different runs with different MS, LSM and parset; they are highly variable for different sets. Those hot-spots correspond to the normal equations generation and to their solution steps. While the performance of the solving step is proportional to the number of parameters, the equation generation portion time depends on the convergence threshold, the maximum number of iterations set by the user and the LSM. So, according to Amdhal's law [9] a maximum speedup of only 1.66x and 2.5x, respectively, can be achieved. In the remainder of this section, we first detail implementation of the equation generation step and then, that of the solving step.

As can be seen from line 4 of Listing 1, the while loop is iterated on until either all cells have converged or a maximum number of iterations is reached. The loop body mainly consists of the equation generation and their solving. For a simple sky model (*e.g.* composed of a few point sources), the number of iterations is usually low and the solve operation dominates the execution time. But as the sky model becomes complex, the number of iterations gets higher and the equation generation becomes dominant. Indeed, there is a fixed number of cells per chunk; consequently, the solve routine gets called a fixed number of times per iteration. As of the equation generation routine, it gets called $N_b \times N_p \times N_t \times N_f$ times, where $N_b$ is the number of baselines, $N_p$ the number of polarizations, $N_t$ the number of time steps and $N_f$ the number of frequencies as specified by the user in the parset. The computations in the equation generation routine are mostly memory accesses consisting in updating elements in the solvers' coefficient matrices.

For each cell, the solve step is actually a loop (LM iteration) consisting of: *a*) factorizing the normal equation matrix into LU form with application of singular value decomposition (SVD) to handle ill-conditioned cases, *i.e.* where there is not enough information to solve for all unknowns (see Listing 2) and *b*) computing a solution for $\delta_p$ using backward and forward substitution. The actual hot-spot is the matrix factorization which, besides the SVD, corresponds to an in-place LU decomposition with partial pivoting. A pseudo-code of the algorithm is shown in Listing 2 below with some code factorization for the sake of readability where $m$ refers to a hypothetical global variable containing the normal equation matrix, $mHeight$ and $mWidth$ being its dimensions.

Listing 2: LU decomposition code in BBS

```
1  for (row=0; row<mHeight; row++) {
2    computeElement(row, row);
3    if (singular)
4      doSVD();
5    for (col=row+1; col < mWidth; col++)
6      computeElement(row, col);
7  }
8  void computeElement(int row, int col) {
9    for (i=0; i<row; i++)
10     m[row][col] -= m[i][row]*m[i][col] /m[i][i];
11 }
```

## III. Optimization approach

### A. Equation generation

From empirical knowledge of the calibration process in LOFAR and the data dependence graph from the equation generation code, it appears that per-cell processing can be done in parallel. So, our first attempt was to try OpenMP by inserting a "parallel for" directive at the appropriate place in the code. Unfortunately, this straightforward approach could not be implemented given that the code uses templates and we need to use the clause `firstprivate` for one of them, which is not allowed in current OpenMP implementations. Eventually, we implemented the step using the C++-11 threading library. The approach consists in implementing a pool of worker threads concurrently extracting and executing tasks from a shared task queue. For parallel execution, the length of the latter can be set up to the number of hardware threads on the node but, in practice, we noticed that using less threads leads to better performance on the used hardware. A task corresponds to generating equations for a pair $\{baseline, polarization\}$ spanning the whole time and frequency domains, *i.e.* for all cells in the current chunk. For each such pair, cells are iterated on, creating and enqueuing tasks for that pair. Because the next pair will be accessing the same cells, a synchronization barrier is required here.

The hot-spot is a small function that gradually constructs the normal equation matrix for each cell solver given a set of parameter indexes and the equivalent data values. The construction is done by looping over the indexes and accumulating in the corresponding matrix element the product of the corresponding data value with all data values corresponding to the smaller indexes in the same row. When the indexes are sorted, the updated matrix elements form an triangle. It happens that this is the case in LOFAR environment. As described in the previous section, this function is called very frequently: *e.g.* it is called 38649600 times for one of the datasets in our experiments. This makes it a good candidate for vectorization which we implemented using SSE2 intrinsics. The idea is to exploit the sorted nature of the indexes to invert and unroll the looping so that value accumulation is

done only once for every element in the triangle, to load and compute matrix elements by pairs using compiler intrinsics and to jump out of the triangle when we meet an index smaller than the first in the input set. Unfortunately, our implementation under-performs the non-SSE version on the used compiler and hardware. Indeed, to jump out of the triangle, we needed to add conditionals, which probably, are negatively impacting the use of the intrinsics.

### B. Equation solving

Just as for equation generation, the equation solving can also be done in parallel on a per-cell basis. This is simply implemented by an OpenMP directive immediately before line 6 in Listing 1. To further optimize the computation, we believe we could use calls to some highly optimized linear algebra library routines for some of the functions, notably, for the LU decomposition routine. We consider various implementations of BLAS/LAPACK [10] such as ATLAS[2] [11], PLASMA[3] and MAGMA[4] [12]. Examples of tried routines are *PLASMA_dgetrf* (*magma_dgetrf_gpu* for MAGMA) and *PLASMA_dpotrf* (*magma_dpotrf_gpu* for MAGMA) which are general and Cholesky LU decomposition routines, respectively. While those routines are sensibly faster than the custom code, they fail to handle the ill-conditioned cases. Furthermore, those libraries use different matrix layouts, and not the packed format used in the CASA solver, which requires converting between layouts. Using blocked versions of the routines somewhat improves the performance but the ill-condition cases still remain an issue. The only routine from the tested linear algebra libraries to produce the same results for all the runs is LAPACKE_dpptrf, but appears slower despite the convenience of the matrix layout.

We have also prospected ways of harnessing the available GPUs' power to speedup the matrix (pseudo-)inversion using both CUDA [13] and OpenCL[5] [14]. However, there are two constraints hindering the use of the GPUs. Firstly, the matrix decomposition routine is called from the body of a loop (see Listing 1)) where the normal equation matrix content gets modified every iteration consecutively to the equation generation. This would require to move them back and forth between the host memory and the device memory along the slow PCI-e channel. Furthermore, the looping combined with the dynamic compiling (even if it can be worked around with some tricks) almost disqualifies OpenCL as a viable programming model for the case. Secondly, as solving computations are already multi-threaded at a higher level, all the threads would be sending data through the same PCI-e channel which would become then a bottleneck. For instance, when using exclusively the CUDA implementation of the inversion computation, the latter is on average 30% slower than the original code for the 118-dataset runs.

However, further analyzing the code, we found the way the symmetric matrix elements are accessed during the factorization also exhibits some additional parallelism. Indeed, apart from SVD, the decomposition operation proceeds by updating the matrix elements column by column using already updated elements from previous steps. The data access pattern is illustrated in Figure 2. From this schematic, we see that the

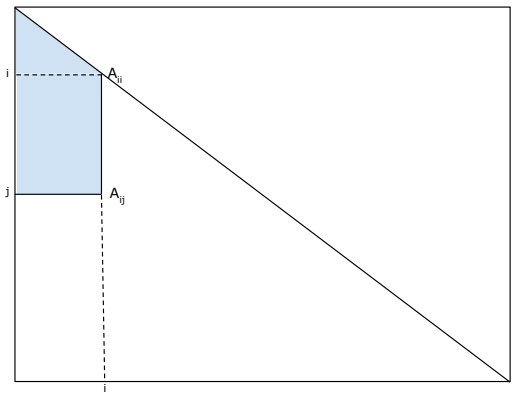


Figure 2: Data access during the coefficient matrix factorization. Only the lower half of the matrix is stored and is accessed column by column. Update of the $i^{th}$ column is illustrated: for $A_{ii}$, elements in upper left triangle are accessed and for $A_{ij}$ those in the upper left trapeze.

update of an element does not require access to any element belonging to the same column. Therefore, all the elements on a column can be updated in parallel. However, the update accesses the elements in the upper left triangle (respectively trapeze) immediately next to the diagonal (respectively off-diagonal) element. Following our minimal modification approach, we have first considered OpenMP. We have inserted a OpenMP "parallel for" directive just before the loop iterating over column elements. We use nested parallelism feature of OpenMP to harness the parallelism at the two levels. Detailed results are given in Section IV for all implementations.

## IV. Implementation and Evaluation

We ran all our experiments on a regular compute node of the DAS-4 infrastructure[6] from The Netherlands radio astronomy center (ASTRON) site. The node has a 24GB of RAM and 2 quad-core Intel E5620 CPUs. The code was compiled with GCC 4.8.1 using "-O2" optimization flag. The execution time breakdown is obtained using the Gperftools CPU profiler[7]. We used 4 different datasets corresponding to different MS, parset and LSM files. We identify each one with the number of parameters to fit, the solution accuracy setting and the number of sources in the LSM. The results of running the original sequential code on above datasets are presented in TABLE I. The acronyms used in the latter are defined in TABLE II. What we notice from examining TABLE I is that `EGC` and `ESC` are highly variable and dependent upon `PA` and `SN` as well as on the number of iterations specified by the user. Next, we examine the impact of individually parallelizing the equation generation and the solving, respectively, and the effect of parallelizing both steps.

[2]http://math-atlas.sourceforge.net/faq.html
[3]http://icl.eecs.utk.edu/plasma
[4]http://icl.eecs.utk.edu/magma
[5]https://www.khronos.org/opencl/
[6]http://www.cs.vu.nl/das4/
[7]http://code.google.com/p/gperftools

TABLE I: Performance of optimized equation generation

| PN | PV | SN | EGC | ESC | SEG | SES | ST |
|---|---|---|---|---|---|---|---|
| 236 | $10^{-9}$ | 1 | 6.3 | 59.9 | 20.11 | 32.91 | 53.06 |
| 240 | $10^{-9}$ | 1 | 13 | 38.1 | 125.79 | 81.67 | 207.53 |
| 118 | $10^{-6}$ | 250 | 41.1 | 0.8 | 1361.61 | 10.61 | 1372.36 |
| 244 | $10^{-9}$ | 270 | 7.7 | 4.9 | 1314.78 | 69.86 | 1384.96 |

TABLE II: Definition of the acronyms in TABLE I.

| Acronym | Meaning | Range |
|---|---|---|
| PN | Number of parameters to estimate | integer |
| PV | Convergence threshold for solution accuracy | small real |
| SN | Number of sources in the used LSM | integer |
| EGC | Contribution of the equation generation to ST | [0, 100] |
| EGC | Contribution of the equation solving to ST | [0, 100] |
| SEG | Clocktime of the equation generation computation | real |
| SES | Clocktime of the equation solving computation | real |
| ST | Total sequential parameter estimation time | real |

### A. Equation generation

Figure 3 zooms into the equation generation optimization performance. This figure is actually a simplification as it only shows the dependence of the execution time upon the number of sources in the sky model whereas that time also depends on the convergence criteria as well as the number of baselines used during the observation in the sens that finer solution will increase the number of iterations, while a large number of baselines will increase the number of normal equations (see section II-D). We observe from this figure and the actual measurements that for very simple models, *e.g.* only 1 source in the sky model, the parallel version performs approximately or below the original code because of the parallelization overhead, notably contention because of some shared data structures. As the model becomes complex up to a few hundreds of sources, there is a clear win in parallelizing the equation generation. This gain, however, is drastically eroded again as the model becomes even more complex. An explanation can be found from the profiling of the 244-dataset, with the highest number of sources, which shows only 7.7% as contribution of the equation generation phase to the global execution time of the fitting.

### B. Equation solving

The performance of the equation solving computation is shown in Figure 4 where we observe that the parallel version clearly outperforms the original code in all scenarios. However, the percent of global execution time accounted for by the stage is too modest to effectively impact the global performance for most realistic cases(cf. TABLE I).

### C. Equation generation and solving

The impact of the parallelization in both stages is shown in Figure 5 where, again, we only retain, for either stage, most sensible variables among the many that influence the execution time. We can see that for simple sky models and irrespective of the number of parameters, the original and the optimized codes perform approximately the same. In fact, the equation solving optimization reduces the run time but that gain is lost in the

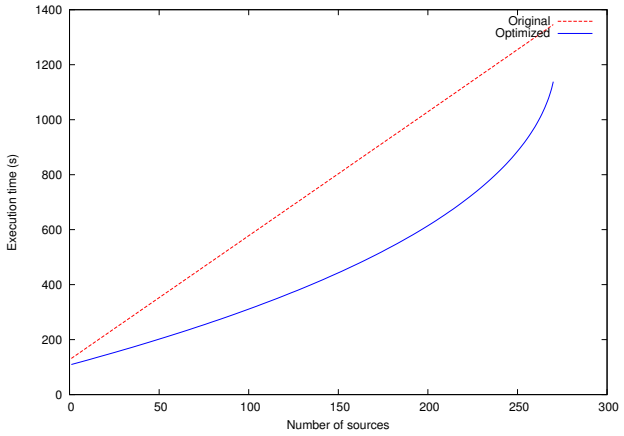


Figure 3: Performance of optimized equation generation.

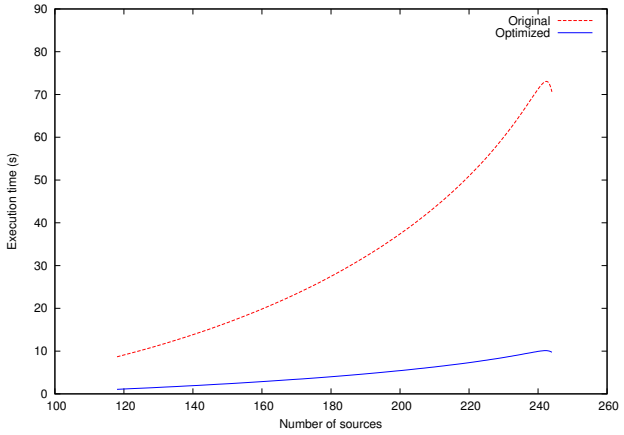


Figure 4: Performance of optimized equation solving.

generation stage parallelization overhead. As the sky model gets complex, the behavior of the execution time becomes more complex depending on the number of both sources and parameters.

## V. Related work and discussion

BBS is not the only software packagein use within the LOFAR code base. The Space Alternating Generalized Expectation Maximization Calibration (SAGECal) [15] is another calibration tool that uses a maximum likelihood estimation approach. Sky and instrument parameters are estimated using an improved Expectation Maximization algorithm [16] with faster convergence at less computational cost. Even though both SAGECal and BBS address the calibration issue, they are 2 totally different pieces of software and this paper is concerned only with improving the performance of BBS. A GPU accelerated hybrid approach that combines LM and the limited memory Broyden-Fletcher-Goldfarb-Shanno algorithm [17] and use of optimized linear algebra libraries is recently presented in [18]. The hybrid method exhibits a speedup of up to 3x compared to a custom, CPU-only implementation; it is based on SAGECal and thus corresponds to a different goal than the one pursued in this work.

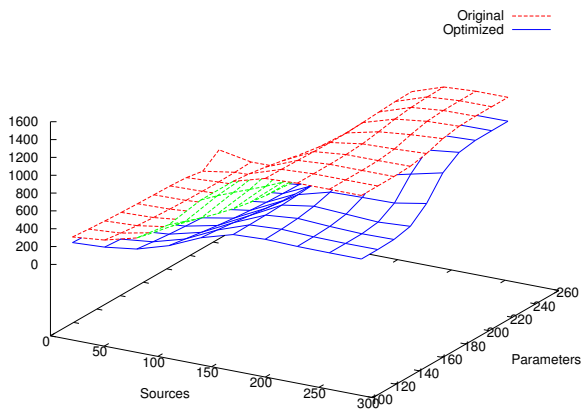


Figure 5: Performance of the optimized equation generation and solving.

BBS and those approaches mainly differ in the way they implement the solver of the model fitting problem. There are several performance shortcomings within the CASA solver used in BBS. Firstly, it does not explicitly use the hardware parallelism despite implementing other optimizations such as packed matrix layout and equation index sorting. Secondly, it does not harness current highly optimized linear algebra libraries despite the occurrence of frequent matrix computations. Furthermore, other performance issues lie in the way the higher level code in BBS uses the solver. Consequently, for a maximum performance on current and future architectures, both the higher level code in BBS that uses the solver and the solver itself should be redesigned with parallelism in mind. Just as in [18], heterogeneous computing and optimized linear algebra routines should be used. From section IV, it appeared that equation generation is the actual performance bottleneck in realistic cases. So, perhaps, nonlinear optimization algorithms that do not use normal equations should be prospected.

## VI. Conclusion

We have presented in this paper the optimization of a sizable interferometric calibration software using "bound-and-bottleneck" analysis. Because of its size and complexity, only localized modifications were allowed. Despite this restriction, we have achieved up to 2.3x speedup compared to the original code by using available hardware parallelism via multi-threading and optimized linear algebra libraries. However, we believe that for a maximum performance, both part of BBS and the solver should be redesigned in a parallelism-aware manner. Also, heterogeneous computing and use of optimized linear libraries should be considered. As of the implementation of the solver, alternative or improved fitting algorithms to LM should also be studied. For instance, nowadays, nearly-linear algorithms, such as in [19] , exist for solving linear systems of equations which can be used to solve the linear least squares problem repeatedly arising during the fitting. And this makes for a natural extension to the work presented here.

## Acknowledgment

This work is in part funded by the research programme of the Netherlands eScience Center (www.nlesc.nl).

## References

[1] I. journal of Research and D. staff, "Overview of the ibm blue gene/p project," *IBM J. Res. Dev.*, vol. 52, no. 1/2, pp. 199–220, Jan. 2008. [Online]. Available: http://dl.acm.org/citation.cfm?id=1375990.1376008

[2] J. P. Hamaker, J. D. Bregman, and R. J. Sault, "Understanding radio polarimetry. I. Mathematical foundations." *Astronomy & Astrophysics Supplement Series*, vol. 117, pp. 137–147, May 1996.

[3] S. van Der Tol, B. Jeffs, and A.-J. van der Veen, "Self-calibration for the lofar radio astronomical array," *Signal Processing, IEEE Transactions on*, vol. 55, no. 9, pp. 4497–4510, Sept 2007.

[4] K. Levenberg, "A method for the solution of certain non-linear problems in least squares," *Quarterly Journal of Applied Mathmatics*, vol. II, no. 2, pp. 164–168, 1944.

[5] D. W. Marquardt, "An algorithm for least-squares estimation of nonlinear parameters," *SIAM Journal on Applied Mathematics*, vol. 11, no. 2, pp. 431–441, 1963. [Online]. Available: http://dx.doi.org/10.1137/0111030

[6] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical recipes in C (2nd ed.): the art of scientific computing*. New York, NY, USA: Cambridge University Press, 1992.

[7] K. Madsen, H. B. Nielsen, and O. Tingleff, "Methods for non-linear least squares problems (2nd ed.)," Richard Petersens Plads, Building 321, DK-2800 Kgs. Lyngby, p. 60, 2004. [Online]. Available: http://f

[8] W. N. Brouw, "Aips++ note 224: Aips++ least squares background," ASTRON, Dwingeloo, The Netherlands, Tech. Rep., 1996.

[9] G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," in *Proceedings of the April 18-20, 1967, spring joint computer conference*, ser. AFIPS '67 (Spring). New York, NY, USA: ACM, 1967, pp. 483–485. [Online]. Available: http://doi.acm.org/10.1145/1465482.1465560

[10] 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*, 3rd ed. Philadelphia, PA: Society for Industrial and Applied Mathematics, 1999.

[11] R. C. Whaley and A. Petitet, "Minimizing development and maintenance costs in supporting persistently optimized BLAS," *Software: Practice and Experience*, vol. 35, no. 2, pp. 101–121, February 2005, `http://www.cs.utsa.edu/~whaley/papers/spercw04.ps`.

[12] E. Agullo, J. Demmel, J. Dongarra, B. Hadri, J. Kurzak, J. Langou, H. Ltaief, P. Luszczek, and S. Tomov, "Numerical linear algebra on emerging architectures: The PLASMA and MAGMA projects," *Journal of Physics: Conference Series*, vol. 180, 2009.

[13] J. Nickolls, I. Buck, M. Garland, and K. Skadron, "Scalable parallel programming with cuda," *Queue*, vol. 6, no. 2, pp. 40–53, Mar. 2008. [Online]. Available: http://doi.acm.org/10.1145/1365490.1365500

[14] J. E. Stone, D. Gohara, and G. Shi, "Opencl: A parallel programming standard for heterogeneous computing systems," *IEEE Des. Test*, vol. 12, no. 3, pp. 66–73, May 2010. [Online]. Available: http://dx.doi.org/10.1109/MCSE.2010.69

[15] S. Yatawatta, S. Zaroubi, G. D. Bruyn, L. Koopmans, and J. Noordam, "Radio interferometric calibration using the sage algorithm," in *Proceedings of 13th IEEE DSP workshop*, January 2009, p. 150155.

[16] J. Fessler and A. Hero, "Space-alternating generalized expectation-maximization algorithm," *Trans. Sig. Proc.*, vol. 42, no. 10, pp. 2664–2677, Oct. 1994. [Online]. Available: http://dx.doi.org/10.1109/78.324732

[17] J. Nocedal and S. Wright, *Numerical optimization*. Springer (2nd edition), 2006.

[18] S. Yatawatta, S. Kazemi, and S. Zaroubi, "Gpu accelerated nonlinear optimization in radio interferometric calibration," in *Innovative Parallel Computing (InPar), 2012*, May 2012, pp. 1–6.

[19] I. Koutis, G. L. Miller, and R. Peng, "Approaching optimality for solving sdd linear systems," in *Proceedings of the 2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, ser. FOCS '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 235–244. [Online]. Available: http://dx.doi.org/10.1109/FOCS.2010.29