Initial report on the performance of BBS

J.E. van Zwieten

January 7, 2008

1 Introduction

According to the overall LOFAR project planning, additional stations will become available at Q1 2009. Furthermore, the default integration time will be brought down from 30 seconds to 1 second. As a result, the amount of data coming in from the correlator will increase by a factor of $\sim$140 – 176.

If we want to stand a chance to cope with this amount of data, we have to start identifying performance bottlenecks. Unfortunately, the details of a realistic calibration strategy for LOFAR are currently unknown and part of the functionality we know will be needed has not been implemented yet.

To at least get some idea of where we stand, we derived the time complexity of the key algorithms that are used for the calibration of CS1 observations. Subsequently, we tried to test if the theory correctly predicts how the run time of the current implementation scales with the size of the input. Finally, we estimated the time that would be needed to perform the calibration strategy currently used to calibrate CS1 observations for an observation done with the Q1 2009 configuration.

This report is structured as follows: Section 2 lists some characteristic numbers and sizes at Q1 2009 and Q4 2009. In section 3, we derive the time complexity of the key algorithms used in the current implementation. Section 4 presents the results of a number of performance tests and tries to link these results to the theory derived in section 3. Section 5 concludes this report.

2 Characteristic numbers and sizes

In this section we compute a set of characteristic numbers and sizes for the LOFAR configuration after completion of the next roll-out phase in Q1 2009 and for the proposed configuration at Q4 2009.

At Q1 2009 there will be 13 core stations and 7 remote stations within The Netherlands. This will increase to 18-25 core stations and 18-25 remote stations at Q4 2009. An attempt will be made to get funding for additional stations within The Netherlands, so the final number may be higher than that mentioned here. At the moment, there is one international station up and running (Effelsberg). At Q1 2009, 1-5 international station are foreseen, which may increase to 1-10 by Q4 2009.

Table 1 lists the number of stations and some other characteristics for each configuration. While interpreting the numbers, keep in mind that each core station consists of one LBA and *two* HBA stations.

!!FIXME!! Δf is not accurate, i.e. 32 MHz/No. of channels does not yield the listed value exactly. The question is: should we trust 32 MHz/No. of channels or the value for Δf?

Phase	LBA	HBA	Δt	Δf (KHz)	Channels	Sub-bands
Q1 2009	20 + 1 – 5	33 + 1 – 5	1 s	0.78 / 0.62	41984 / 52480	164 / 205
Q4 2009	36 – 50 + 1 – 10	54 – 75 + 1 – 10	1 s	0.78 / 0.62	41984 / 52480	164 / 205

Table 1: Overview of the number of stations and other characteristics of each configuration. Keep in mind that one core station consists of one LBA and *two* HBA stations.

Table 2 lists the total observation size for several different durations at the maximal temporal resolution. The table is provided to give an impression of the total amount of data that needs to be processed and/or stored.

Phase	Duration	Measurement size (LBA)	Measurement size (HBA)
Q1 2009	1 h	0.92 - 1.32 TB	3.08 - 3.87 TB
	4 h	3.70 - 5.28 TB	12.34 - 15.46 TB
	8 h	7.39 - 10.56 TB	24.68 - 30.92 TB
Q4 2009	1 h	2.93 - 7.79 TB	8.17 - 19.63 TB
	4 h	11.72 - 31.14 TB	32.66 - 78.52 TB
	8 h	23.44 - 62.29 TB	65.32 - 157.04 TB

Table 2: Size of a single measurement for various durations. The sizes listed in this table are based on the raw *cross* correlation data only. Auto correlations, flags, and meta data have *not* been taken into account.

A number of interest is the size in bytes of a single time slot. At the maximal temporal resolution (1 second), it is equivalent to the total amount of data produced by OLAP per second. It is also the amount of data that the calibration software must process per second if calibration is required to run in real time. It can be computed as follows:

$$\text{timeslot size} = \text{baselines} \times \text{channels} \times \text{polarizations} \times 1\,\text{complex float}\,(8\,\text{bytes})$$

Table 3 lists the sizes for each configuration. It also lists the size of a single time slot *per sub-band*, which is equivalent to the amount of data generated per compute node per second if we assume each sub-band is mapped to a separate compute node. This mapping will probably be too coarse for the final configuration.

Phase	Time slot (LBA)	Per sub-band	Time slot (HBA)	Per sub-band
Q1 2009	269.06 – 384.38 MB	1.64 – 2.34 MB	898.48 MB – 1.10 GB	4.38 – 5.49 MB
Q4 2009	853.31 MB – 2.21 GB	5.20 – 13.83 MB	2.32 GB – 5.58 GB	11.60 – 27.89 MB

Table 3: Amount of data contained in a single time slot in total and per sub-band.

Note that the amount of data per compute node per second approaches the speed of a single disk for the final HBA configuration. In other words, it may already become challenging to read and write a single time slot worth of data within a 1 second time limit. Of course, there are ways in which this problem can be alleviated. Using multiple disks per compute node would increase disk I/O speed. Distributing each sub-band over multiple compute nodes would decrease the amount of data that needs to be processed per compute node per second. Alternatively, it may be possible to move the processing that needs to be done on the full resolution data to the BlueGene. In that case, we would only have to deal with *integrated* data on the offline cluster. This would decrease the amount of data by a factor ~30.

Assuming an observation is distributed in frequency, another number of interest is the maximal amount of data in time that can be loaded into main memory per sub-band. This number determines the maximal time scale at which the model can be fitted to the observed visibilities. Of course, larger time scales are possible but would require reading data multiple times for each *iteration* of the fitting procedure. This is currently believed to be impractical. Table 4 lists the maximal domain size in time at the maximal temporal resolution assuming 2 GB of main memory per sub-band.

Phase	Max. domain size (LBA)	Max. domain size (HBA)
Q1 2009	20.80 - 14.55 min	7.78 - 6.20 min
Q4 2009	6.55 - 2.47 min	2.93 - 1.22 min

Table 4: Maximal domain size in time assuming 2 GB of main memory per sub-band (scales linearly with the amount of main memory).

3 Time complexity of core operations

In this section we will derive expressions for the (asymptotic) time complexity of each core operation as a function of several external variables, e.g. number of stations, integration time, and model complexity. The notion of *core operation* is used to indicate an algorithm or set of algorithms that perform(s) a specific task withtin BBS, e.g. reading data from disk or iteratively solving for the value of a set of model parameters. They are a reflection of the way a user might divide his/her data reduction into separate tasks. The word *core* should therefore be interpreted from the user's point of view. From a developer's perspective, a core operation can often be subdivided into smaller tasks.

The goal of this section is to clearify how the total amount of work (and therefore ultimately the total run time) changes with certain external variables. For instance, what happens to the total amount of work if we double the number of stations? Or if we decrease the integration time by a factor of 30? Or if we triple the number of sources in the sky model? We focus on the change in the total amount of work as a function of the number of stations and the integration time, because these characteristics will change due to the hardware roll-out in the course of 2008 and 2009 *and* the numbers are more or less known. It is expected that the reduction strategy and the sky/instrument models will become increasingly complex in the near future. This will have a significant impact on the total amount of work as well. However, there are still many unknowns in this area which make it difficult to draw any meaningful conclusions.

In the following we will derive the time complexity for a single threaded machine. We will not consider distributed computation and the communication overhead this will incur.

3.1 Input size

The number of visibilities n_v is proportional to the number of time slots n_t, channels n_f, polarizations n_p, and baselines n_b.

$$n_v = n_b \text{ baselines} \times n_f \text{ channels} \times n_t \text{ time slots} \times n_p \text{ polarizations}$$

The number of baselines expressed in terms of the number of stations n_s is given by:

$$n_b = \begin{cases} \frac{1}{2} \times n_s \times (n_s - 1) & \text{cross correlations} \\ \frac{1}{2} \times n_s \times (n_s + 1) & \text{all correlations} \end{cases}$$

Thus, the number of baselines is proportional to the square of the number of stations.

$$n_b \sim O(n_s^2)$$

3.2 Disk I/O

The majority of data read or written consists of visibility data and flags. Therefore, the time it takes to read or write data from disk is expected to be proportional to the number of visibilities.

$$t_{disk} \sim O(n_v)$$

3.3 PREDICT, SUBTRACT, CORRECT

The PREDICT operation evaluates the model of each interferometer (baseline), which is represented internally as an *expression tree* (actually, a directed acyclic graph). The model expression is some form of the measurement equation. The result of the PREDICT operation is a set of simulated visibilities for each baseline. The SUBTRACT operation is an extension of the PREDICT operation. The simulated visibilities are subtracted from the observed visibilities to yield *residual* visibilities. The CORRECT operation is a variant of the PREDICT operation. It takes a set of visibilities as input and corrects these visibilities for instrumental UV-plane effects. Like the PREDICT operation, it involves the evaluation of an expression tree for each baseline. Generally, the expression tree is less complex (i.e. contains less nodes) than the expression trees used in the PREDICT operation. All three operations have the same asymptotic time complexity. Therefore, we can consider only the PREDICT operation without losing generality.

Given a grid of sample locations in frequency and time, a number of stations, and a number of polarizations, a simulated visibility needs to be computed for every combination of baseline, sample location, and polarization. An important property of the models that are currently used is that each visibility can be computed *independently.* This implies that the number of steps required to compute a single visibility does not depend on the number of channels, time slots, baselines, or polarizations, but is constant. Therefore, the number of steps required to execute the PREDICT operation is proportional to the number of visibilities.

$$t_{predict} \sim O(n_v)$$

Deriving a bound on $t_{predict}$ as a function of model complexity is harder. The number of steps required to evaluate a single node in the expression tree depends on the type of processing it performs. The result of some nodes is constant. The result of other nodes depend on time (e.g. UVW coordinates in meters), or on frequency (e.g. bandpass). The result of still other nodes depend on both time and frequency (e.g. DFT of a point source). In general, the result of a node can depend on any subset of the axes frequency, time, and polarization. Hence, an upper bound on the processing time of a single node is given by:

$$t_{node} \sim O(n_f \times n_t \times n_p)$$

Assuming every node with more than one parent caches its result, each node will only be evaluated once. Let N denote the number of nodes in the expression tree for a single baseline. For simplicity, we will ignore the fact that the result of some nodes can be shared by multiple baselines. An upper bound on the number of steps required to evaluate the expression tree for every baseline is then given by:

$$\begin{aligned} t_{predict} &\sim O(n_b \times N \times t_{node}) \\ &\sim O(n_b \times N \times O(n_f \times n_t \times n_p)) \\ &\sim O(N \times n_v) \end{aligned}$$

With caching, $t_{predict}$ is proportional to the total number of nodes in the model for a given number of visibilities. Obviously, this upper bound need not be very tight.

Suppose the result of a node A is needed by n other nodes. Without caching, A has to be evaluated n times. Thus, without caching, the way the nodes are interconnected in the expression tree becomes important. In the worst case, evaluating a tree with N nodes can take $O(N^2)$ node evaluations. However, in realistic trees, each node is connected to a relatively small number of other nodes. Let c be the maximum number of inputs of any node. Usually, c will be a small constant independent of N. In this case evaluating a tree with N nodes will take at most $O(c \times N) \sim O(N)$ evaluations. Using a cache will most likely speed up the computation, but it will not change the asymptotic time complexity.

3.4 SOLVE

During calibration, an often recurring task is fitting a set of model parameters to the observed visibilities. For the calibration of LOFAR an implementation of the Levenberg–Marquardt algorithm is used for this task. This is an iterative algorithm that involves computing the solution of a linear least squares problem each iteration. It requires both an evaluation of the model given a set of model parameters, as well as the first order partial derivatives of the model with respect to the *unknowns* (i.e. model parameters that are to be fitted).

In general, model parameters are bound to a particular station and/or to a particular direction on the sky. The total number of parameters depends on the number of stations, the number of (discrete) directions, and the complexity of the model. The calibration strategy dictates in which order the parameters should be fitted, and which parameters should be fitted *simultaneously.* During the SOLVE operation, the total data domain is partitioned into a specified number of smaller *solve domains.* Each solve domain is treated as an individual fitting problem.

One iteration of the SOLVE operation consists of the following three sub-operations. To derive the time complexity of the SOLVE operation as a whole, we will consider each sub-operation separately.

1. EVALUATE MODEL

   Compute visibilities by evaluating the model given the current values of the model parameters. Also, compute *perturbed visibilities* for each unknown. Perturbed visibilities are the result of evaluating the model after adding a small constant to the value of one of the unknowns. The perturbed visibilities for an unknown are used to approximate the partial derivative of the model with respect to that unknown using forward differences.

2. CONSTRUCT EQUATIONS

   Construct a set of linear equations using the computed visibilities, perturbed visibilities, and the observed visibilities. The set of linear equations represents the linearization of the model around the current values of the unknowns. The set is usually overdetermined, i.e. the number of observed visibilities is larger than the number of unknowns.

3. COMPUTE SOLUTION

   Solve the set of linear equations and update the value of the unknowns.

To simplify the analysis of the EVALUATE MODEL sub-operation, we will first consider only the amount of work involved in processing a *single baseline* for a *single solve domain*. We assume all the data in the solve domain is *locally available* to the compute node. Suppose the total data domain is partitioned into N solve domains of equal size. Then the number of visibilities that need to be computed for a single baseline for a single solve domain satisfies:

$$n'_v = \frac{n_v}{n_b \times N}$$

Also, we need to compute perturbed visibilities for each unknown. What is the maximal number of unknowns per baseline? Parameters can be divided into two categories: station *dependent* and station *independent*. An example of a station dependent parameter is station gain, which may be different for each station. An example of a station independent parameter is the Stokes vector of a source, which should be the same for all stations. Let n_p denote the number of station dependent parameters and n_q the number of station independent parameters. The number of station dependent parameters can be expressed as a constant (the number of parameters per station) times the number of stations, i.e.:

$$n_p = c \times n_s$$

A baseline is the combination of two stations. Only $2 \times n_p / n_s$ station dependent parameters are relevant for any baseline. Hence, the maximal number of unknowns per baseline is $2 \times c + n_q$, which is independent of the number of stations.

Per unknown we need to compute n'_v perturbed visibilities. In section 3.3 we showed that the number of steps required to compute visibilities is proportional to the number of visibilities *for a given model complexity*. Hence, an upper bound on the number of steps required to execute the EVALUATE MODEL sub-operation for a single baseline on a single solve domain is given by:

$$t'_{evaluate\,model} \sim O(n' \times (1 + 2 \times c + n_q))$$

Considering all baselines and all solve domains, this becomes:

$$t_{evaluate\,model} \sim O(n_v \times (1 + 2 \times c + n_q))$$

Although the total number of station dependent parameters is proportional to the number of stations, the number of station dependent parameters *per baseline* is constant. Thus, *for a given model complexity*, the number of steps required to execute the EVALUATE MODEL sub-operation is proportional to the number of visibilities n_v.

The CONSTRUCT EQUATIONS sub-operation consists of adding two condition equations for each visibility to a normal matrix. Adding a condition equation to a normal matrix requires $O(n^2)$ steps, where n denotes the number of unknowns in the condition equation ($2 \times c + n_q$). Considering all baselines and all solve domains, the number of steps required to execute the CONSTRUCT EQUATIONS sub-operation is given by:

$$t_{construct\,equations} \sim O(n_v \times (2 \times c + n_q)^2)$$

The COMPUTE SOLUTION sub-operation uses an algorithm that requires $O(n^3)$ steps, where n denotes the *total* number of unknowns ($n_p + n_q$). The algorithm is invoked for *each* solve domain. Let N denote the number of solve domains. It follows that:

$$t_{compute\,solution} \sim O(N \times (n_p + n_q)^3)$$

After completing the maximal number of iterations or reaching convergence, an estimate of the error on the unknowns can optionally be computed. This algorithm has the same time complexity as the COMPUTE SOLUTION sub-operation.

!!FIXME!! In practice, solveLoop() seems to scale at least as $O(n^4)$? Check with Wim Brouw.

Let n_i denote the maximal number of iterations, then the SOLVE operation as a whole requires a number of steps given by:

$$t_{solve} \sim n_i \times \left[O\left(n_v \times (1 + 2 \times c + n_q)\right) + O\left(n_v \times (2 \times c + n_q)^2\right) + O\left(N \times (n_p + n_q)^3\right) \right]$$

3.5 Summary

We derived the time complexity of the core operations of BBS. Almost all processing scales linearly with the number of visibilities. An exception is the COMPUTE SOLUTION sub-operation, which requires a number of steps proportional to the *cube* of the total number of unknowns.

4 In practice

Several measurements of the performance of BBS have been carried out. This section describes the test setup, and dicusses the results.

4.1 Single core

The following two machines were used to test the single core performance of BBS:

- **lioff008**
  Intel Xeon, 2.80 GHz, 512 Kb cache
  g++ (GCC) 4.1.0 (SUSE Linux)
- **Barcelona**
  Quad-Core AMD Opteron(tm) Processor 2350, 2.0 GHz, 512 Kb cache
  g++ (GCC) 4.1.3 20070929 (prerelease) (Ubuntu 4.1.2-16ubuntu2)

4.1.1 PREDICT

The performance of the PREDICT operation was measured using various settings for the number of baselines, the number of time slots, and the number of channels. The sky model contained two sources at the location of CasA and CygA. Sarod's analytical LBA dipole beam model was included in the instrument model, as well as a separate J-jones (identity) matrix for each source – station combination.

The PREDICT operation was repeated 10 times for each combination of number of baselines, number of time slots, number of channels. Only the evaluation of the expression tree was measured (using a high precision timer around the `process()` function call inside the `Prediffer`). The time of every first run was discarded because it includes computation of UVW-coordinates that are reused during subsequent runs.

Figure 1 shows a subset of the results on lioff008. Figure 2 shows the associated results on Barcelona. To first order the results support the time complexity derived in section 3.3: The figures shows a linear relation between processing time and the number of baselines, the number of time slots, and the number of visibilities.

Figure 1 shows a remarkable peak for the run with 160 time slots and 256 channels. Similar peaks can be seen in figure 2, for example the run with 160 time slots and 256 *or* 512 channels. The cause of these peaks is currently unknown. However, most likely is has something to do with the product of the number of channels and the number of time slots, which determines the buffer size used during evaluation of the expression tree.

Furthermore, there seems to be a gap between the time it takes to process N time slots compared to N times the time it takes to process a single time slot. For example, according to figure and 1(d), it takes 18.62 seconds to process 120 baselines $\times$ 180 time slots $\times$ 256 channels $\times$ 4 polarizations. But 1(c) suggests it should be possible in approximately $0.02592 \times 180 = 4.666$ seconds. It would be worth investigating the cause of this descrepancy.

Finally, comparing figures 1(d) and 2(d), the PREDICT operation seems to run roughly a factor 3.6 faster on Barcelona for the typical case of 256 channels.

4.1.2 SOLVE

The performance of the SOLVE operation was measured using various settings for the number of baselines, the number of time slots, and the number of channels. The sky model contained two sources at the location of CasA and CygA. Sarod's analytical LBA dipole beam model was included in the instrument model, as well as a separate J-jones (identity) matrix for each source – station combination. A *single* solve domain was used, which contained the entire data domain. The set of unknowns comprised all four (complex) elements of each J-jones matrix. All unknowns were modelled as 0-order polynomials (constants).

For each combination of number of baselines, number of time slots, and number of channels, 11 iterations of the fitting procedure were performed. Both the evaluation of the expression tree and the construction of condition equations was measured (using a high precision timer around the `process()` function call inside the `Prediffer`). The time of every first iteration was discarded because it includes computation of UVW-coordinates that are reused during subsequent runs. The time required to solve the normal equations was *not* measured.

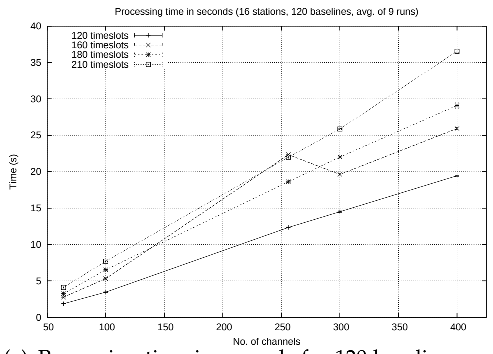
Figure 3 shows a subset of the results on lioff008. Figure 4 shows the associated results on Barcelona. Note that the time required to solve the normal equations was not measured. Therefore, we should ignore the third term in the time complexity derived in 3.4, which results in a time complexity that is linear in the number of visibilities. Indeed, to first order the results show a linear relation between processing time and the number of baselines, the number of time slots, and the number of visibilities.

The figures show several odd peaks, similar to those described in section 4.1.1. Contrary to the PREDICT operation, there does not seem to be a significant gap between the time required to process N time slots and N times the time it takes to process a single time slot. Comparing figures 3(d) and 4(d), the SOLVE operation seems to run roughly a factor 2 faster on Barcelona.

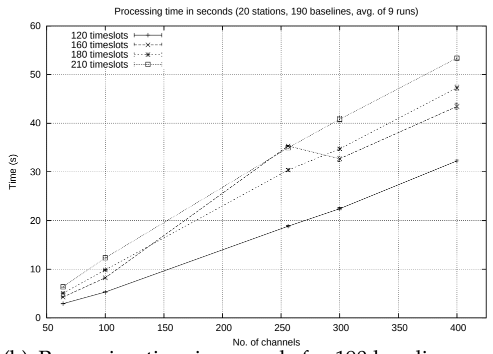
4.2 Multi-core

Barcelona (see section 4.1) was used to measure the multi-core performance of BBS. This is a dual quad core AMD machine. Each quad core has its own memory controller. We used g++-4.2 (GCC)

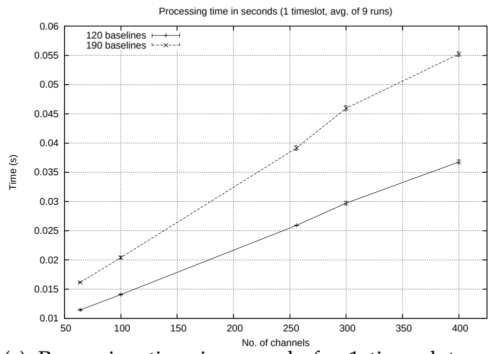
Figure 1: Timings of the PREDICT operation on lioff008, excluding the time required to compute UVW-coordinates.



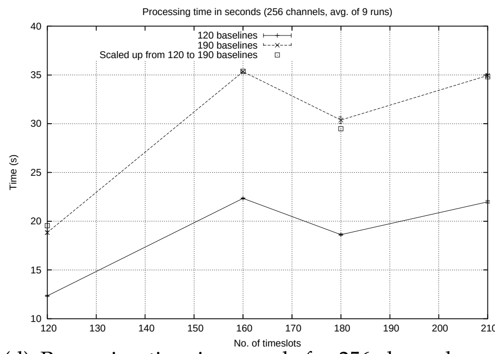
(a) Processing time in seconds for 120 baselines as a function of the number of channels and the number of time slots.



(b) Processing time in seconds for 190 baselines as a function of the number of channels and the number of time slots.

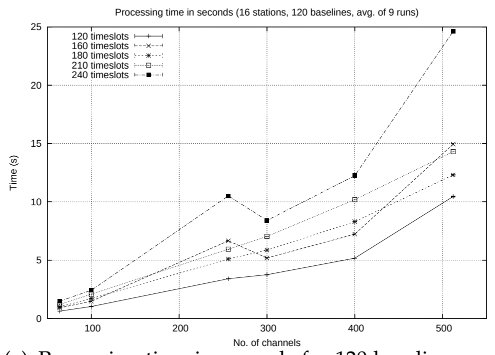


(c) Processing time in seconds for 1 time slot as a function of the number of channels and the number of baselines.

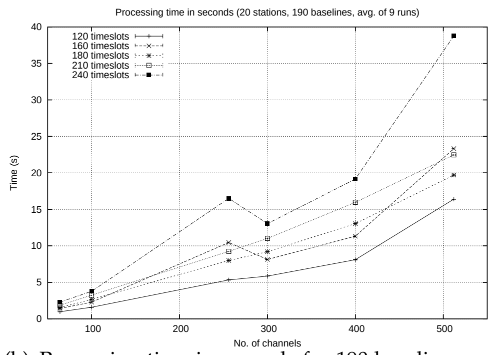


(d) Processing time in seconds for 256 channels as a function of the number of time slots and the number of baselines.

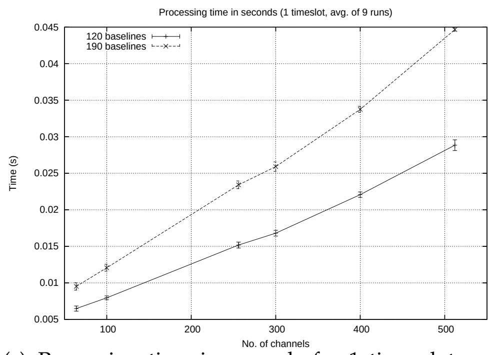
Figure 2: Timings of the PREDICT operation on Barcelona, excluding the time required to compute UVW-coordinates.



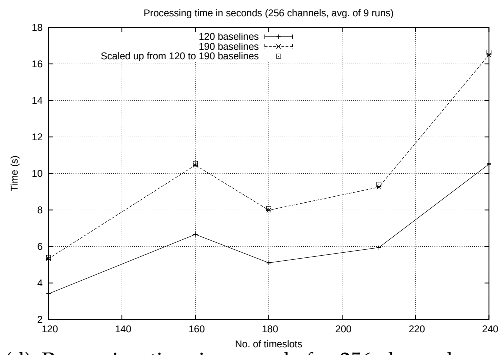
(a) Processing time in seconds for 120 baselines as a function of the number of channels and the number of time slots.



(b) Processing time in seconds for 190 baselines as a function of the number of channels and the number of time slots.

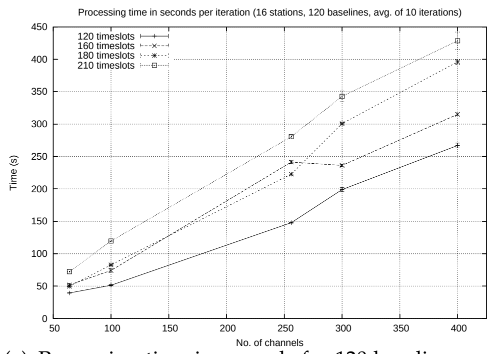


(c) Processing time in seconds for 1 time slot as a function of the number of channels and the number of baselines.

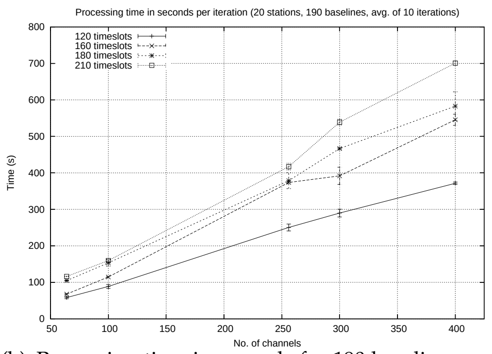


(d) Processing time in seconds for 256 channels as a function of the number of time slots and the number of baselines.

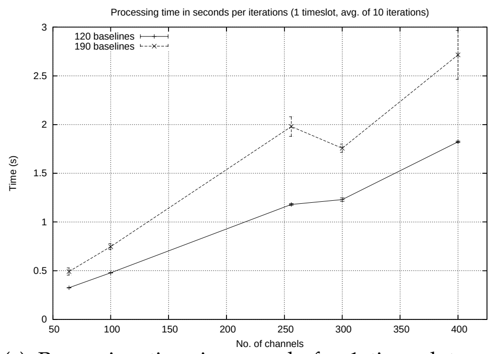
Figure 3: Timings of the SOLVE operation on lioff008, excluding both the time required to compute UVW-coordinates and the time required to communicate and solve the normal equations.



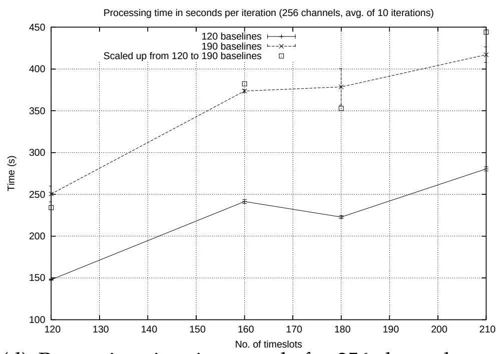
(a) Processing time in seconds for 120 baselines as a function of the number of channels and the number of time slots.



(b) Processing time in seconds for 190 baselines as a function of the number of channels and the number of time slots.

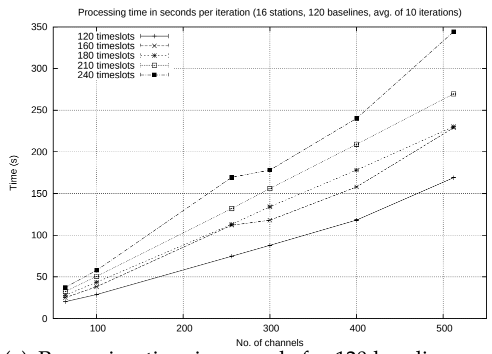


(c) Processing time in seconds for 1 time slot as a function of the number of channels and the number of baselines.

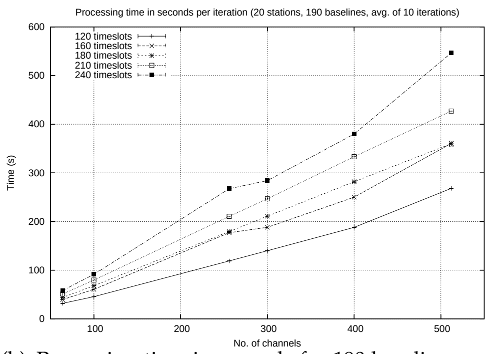


(d) Processing time in seconds for 256 channels as a function of the number of time slots and the number of baselines.

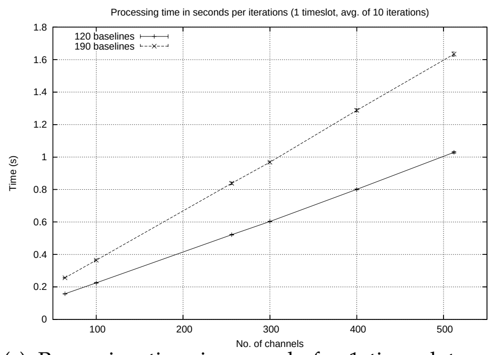
Figure 4: Timings of the SOLVE operation on Barcelona, excluding both the time required to compute UVW-coordinates and the time required to communicate and solve the normal equations.



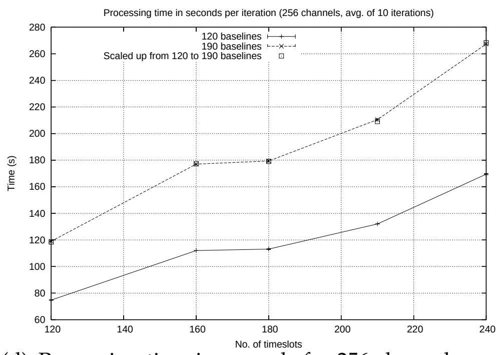
(a) Processing time in seconds for 120 baselines as a function of the number of channels and the number of time slots.



(b) Processing time in seconds for 190 baselines as a function of the number of channels and the number of time slots.

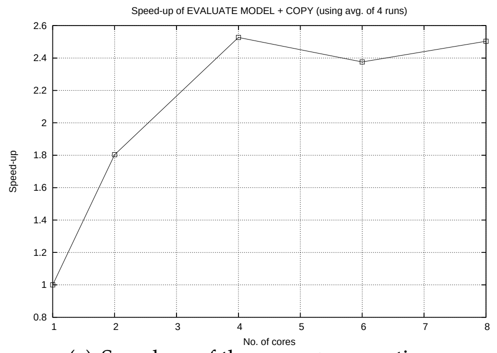


(c) Processing time in seconds for 1 time slot as a function of the number of channels and the number of baselines.

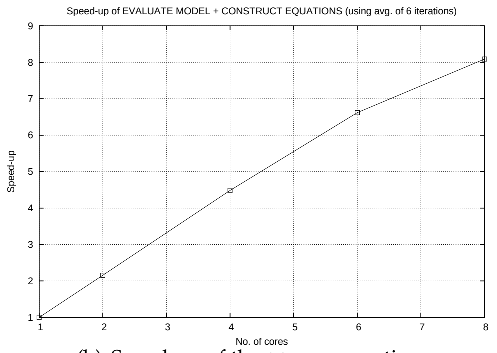


(d) Processing time in seconds for 256 channels as a function of the number of time slots and the number of baselines.

Figure 5: Speed-up of the PREDICT and SOLVE operations on Barcelona.



(a) Speed-up of the PREDICT operation.



(b) Speed-up of the SOLVE operation.

4.2.1 (Ubuntu 4.2.1-5ubuntu4) because the multi-threaded implementation of BBS uses OpenMP, which is not supported by g++-4.1.

4.2.1 PREDICT and SOLVE

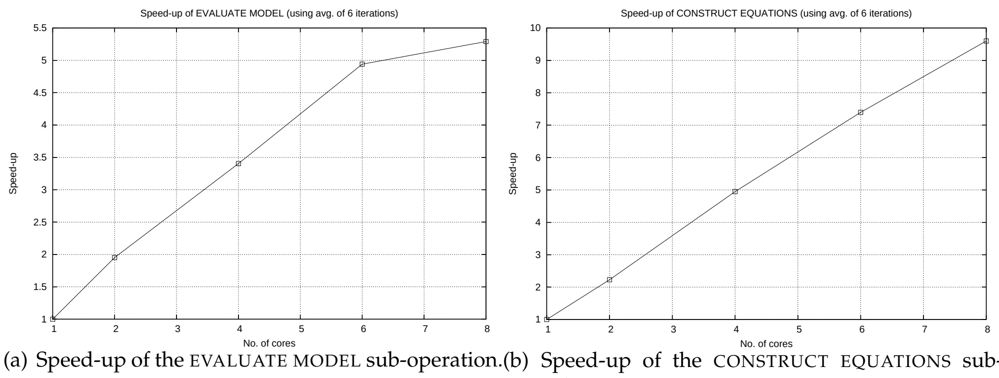
The results presented in this section were obtained just before Barcelona had to be shipped back to the supplier. Therefore, we were unable to perform a representative number of runs. Yet, the results were sufficiently interesting to include them in this report.

To test the performance of the PREDICT operation, we measured the time required to compute visibilities for 120 baselines $\times$ 240 time slots $\times$ 256 channels $\times$ 4 polarizations. The sky model contained two sources at the location of CasA and CygA. Sarod's analytical LBA dipole beam model was included in the instrument model, as well as a separate J-jones (identity) matrix for each source – station combination. The PREDICT operation was repeated 5 times. Only the evaluation of the expression tree was measured (using a high precision timer around the `process()` function call inside the `Prediffer`). The time of every first run was discarded because it includes computation of UVW-coordinates that are reused during subsequent runs.

To test the performance of the SOLVE operation, we measured the time required to process the visibilities for 120 baselines $\times$ 240 time slots $\times$ 256 channels $\times$ 4 polarizations. The sky model contained two sources at the location of CasA and CygA. Sarod's analytical LBA dipole beam model was included in the instrument model, as well as a separate J-jones matrix for each source – station combination. A *single* solve domain was used, which contained the entire data domain. The set of unknowns comprised all four (complex) elements of each J-jones matrix. All unknowns were modelled as 0-order polynomials (constants). The fitting procedure was iterated 6 times. The time of the first iteration was discarded because it includes computation of UVW-coordinates that are re-used during subsequent runs. The time required to solve the normal equations was *not* measured.

Figure 5 shows the speed-up as a function of the number of threads for both the PREDICT and the SOLVE operation. The figure shows that the speed-up achieved for the SOLVE operation is much better than that achieved for the PREDICT operation. This seems strange because the SOLVE operation involves computing visibilities as well (in the EVALUATE MODEL sub-operation). Figure 6 shows a separate speed-up curve for the EVALUATE MODEL sub-operation and the CONSTRUCT EQUATIONS sub-operation. It shows that the speed-up of the EVALUATE MODEL sub-operation flattens off although it is still better than the speed-up of the PREDICT operation. However, the CONSTRUCT EQUATIONS sub-operation achieves super-linear speed-up, which compensates for the loss of performance in the EVALUATE MODEL sub-operation.

Figure 6: Speed-up of the EVALUATE MODEL and CONSTRUCT EQUATIONS sub-operations of the SOLVE operation on Barcelona.



(a) Speed-up of the EVALUATE MODEL sub-operation. (b) Speed-up of the CONSTRUCT EQUATIONS sub-operation.

4.2.2 Total run time of a CS1 calibration strategy

A goal of the performance tests described in this section was to come up with an estimate of the time it will take to reduce an observation done with the Q1 2009 configuration. The details of a realistic calibration strategy for LOFAR are still unknown and some functionality that will be required has not been implemented yet. Therefore, we measured the performance of the calibration strategy we currently use for CS1 data, knowing that this can only provide a lower bound on the time that will eventually be needed to calibrate LOFAR data to a sufficient level.

Part of a real CS1 measurement was used for the performance tests. This partial measurement contained 120 baselines, 240 time slots, 256 channels, and 4 polarizations. The sky model contained two sources at the location of CasA and CygA. Sarod's analytical LBA dipole beam model was included in the instrument model, as well as a separate J-jones (identity) matrix for each source – station combination. Each solve domain contained 256 channels and 1 time slot, yielding 240 solve domains in total. The set of unknowns comprised all four (complex) elements of each J-jones matrix. All unknowns were modelled as 0-order polynomials (constants). The calibration strategy consisted of solving for the unknowns using 5 iterations, subtracting CasA and CygA from the data, and correcting the residuals for the J-jones matrix in the direction of CasA.

We compared the speed-up using multiple threads within a single process to that using multiple processes. In the first case, the total run time was measured as a function of the number of threads used. Each thread processed a different part of the same data set. Figure 7 shows the results. In the second case, a variable number of processes were started simultaneously. The total run time of each individual process was measured. Each process operated on its own data set. Figure 8 shows the results.

The results suggest that with the current implementation it is better to use multiple processes. However, this may also indicate that the serial part of the code is inefficient. For instance, suppose writing solutions to disk is implemented inefficiently such that it is only able to make use of one tenth of the disk's bandwidth. In such cases, it is likely that performance can be gained by having multiple processes write solutions simultaneously. However, if the implementation of the writing routine could be improved such that it makes efficient use of the disk's bandwidth, using multiple processes will not gain much compared to using multiple threads.

Figure 7: Total run time and speed-up running BBS using multi-threading.

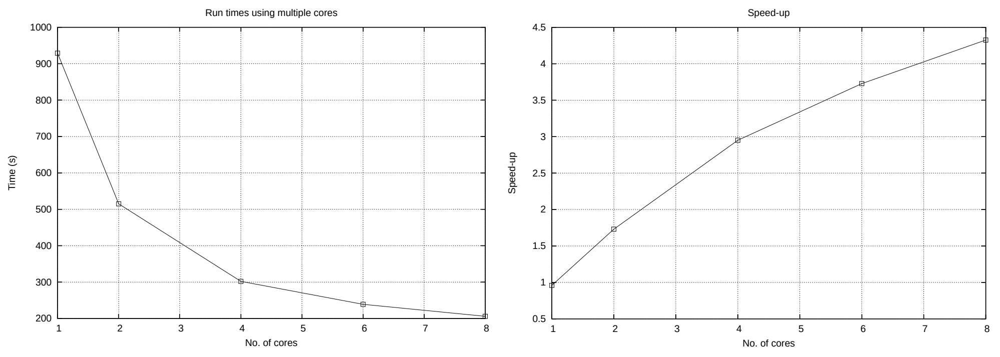
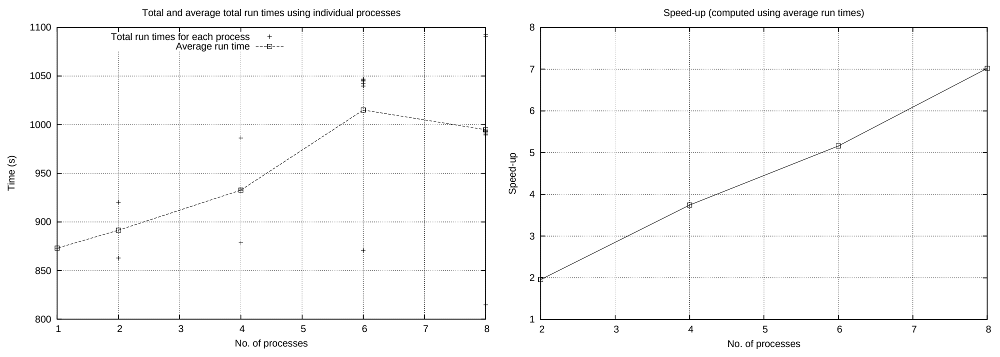


Figure 8: Total run time and speed-up running BBS using multiple processes.



5 Conclusion

We derived time complexities for the core BBS operations and partially verified this theory against measured run times of the current implementation. The derived time complexities can be used to scale measured run times to future telescope configurations. (Of course, provided that the derived time complexities are correct). Also, they can be used to check our understanding of the software. If we measure run times that contradict the theory, than there is an error either in the software or in the theory (or both). Improving either is progress.

The run times documented in section 4 of this report can be used as a benchmark for future versions of BBS and for judging the performance on different hardware. However, one should be careful to use exactly the same test setup. This highlights on of the difficulties we encountered while testing the performance of BBS: There are so many possible settings that is becomes a challenge to test the performance in a structured way.

Now, let us return to the goal of providing an estimate of the run time of the calibration software on Q1 2009 observations. Comparing the situation at Q1 2009 with the situation at Q1 2008, the number of stations[1] will increase from 16 to 34 – 38. The number of baselines will increase from 120 to 561 – 703, which corresponds to a factor $\sim$4.68 – 5.86. The integration time will decrease from 30 s to 1 s. Therefore, in total, the number of visibilities will increase by a factor $\sim$140 – 176. The number of station dependent parameters will increase by a factor $\sim$2.13 – 2.38.

The run time of BBS is dominated by code that operates on visibilities. Therefore, we may expect that if we would run the current implementation on an observation performed with the Q1 2009 configuration, the run time will at least increase with the same factor as the number of visibilities. Using the Barcelona as an example, it takes approximately 875 seconds to process 240 time slots of a single sub-band (256 channels) using the current implementation (see figure 8(a)). Scaling this up to the Q1 2009 configuration it will take around 21 hours and 22 minutes to process 1 hour of data *for a single sub-band*. Assuming we can use multiple cores effectively, we would need around 22 cores per sub-band to process the data in real time. As there are 205 sub-bands, this amounts to 4510 cores in total.

At Q4 2009 the number of stations will increase again, to 55 – 85. As a result, the number of baselines will increase by a factor $\sim$2.65 – 5.08 and the number of station dependent parameters will increase by a factor $\sim$1.62 – 2.24.

Of course, the above is a 'back-of-an-envelope' estimate. The run time of 875 seconds mentioned above was measured for a specific calibration strategy (described in section 4.2.2). However, it can be viewed as a lower bound in the sense that the eventual calibration strategy will likely be much more complex.

[1]We quote the number of HBA stations at Q1 2009 because there will be more HBA stations than LBA stations and we want to give the worst case scale factor.