The LOFAR Correlator: Implementation and Performance Analysis

John W. Romein P. Chris Broekema Jan David Mol Rob V. van Nieuwpoort

ASTRON (Netherlands Institute for Radio Astronomy)
Oude Hoogeveensedijk 4, 7991 PD Dwingeloo, The Netherlands
{romein,broekema,mol,nieuwpoort}@astron.nl

Abstract

LOFAR is the first of a new generation of radio telescopes. Rather than using expensive dishes, it forms a distributed sensor network that combines the signals from many thousands of simple antennas. Its revolutionary design allows observations in a frequency range that has hardly been studied before.

Another novel feature of LOFAR is the elaborate use of *software* to process data, where traditional telescopes use customized hardware. This dramatically increases flexibility and substantially reduces costs, but the high processing and bandwidth requirements compel the use of a supercomputer. The antenna signals are centrally combined, filtered, optionally beam-formed, and correlated by an IBM Blue Gene/P.

This paper describes the implementation of the so-called correlator. To meet the real-time requirements, the application is highly optimized, and reaches exceptionally high computational and I/O efficiencies. Additionally, we study the scalability of the system, and show that it scales well beyond the requirements. The optimizations allows us to use only half the planned amount of resources, *and* process 50% more telescope data, significantly improving the effectiveness of the entire telescope.

Categories and Subject Descriptors J.2 [*Physical Sciences and Engineering*]: Astronomy; C.2.4 [*Computer-Communication Networks*]: Distributed Systems—Distributed Applications; D.1.3 [*Programming Techniques*]: Concurrent Programming; J.7 [*Computers in Other Systems*]: Real time

General Terms Algorithms, Experimentation, Performance

Keywords LOFAR, Correlator, IBM Blue Gene/P

1. Introduction

LOFAR is an acronym for ***LO****w* ***F****requency* ***AR****ray*, an aperture array radio telescope operating in the 10 to 250 MHz frequency range. It is the first of a new generation of radio telescopes, that breaks with the concepts of traditional telescopes in several ways. Rather than using large, expensive dishes, LOFAR uses many thousands of simple antennas that have no movable parts [1, 13] (see Figure 1). Essentially, it is a distributed sensor network that monitors the sky and combines all signals centrally. This concept requires much more signal processing, but the additional costs of silicon are easily offset by cost savings in steel that would be needed for dishes. Moreover, LOFAR can observe the sky in many directions simultaneously and switch directions instantaneously. In several ways, LOFAR will be the largest telescope of the world, and will enable groundbreaking research in several areas of astronomy and particle physics [2]. The different goals and observation types require several different processing pipelines, however.

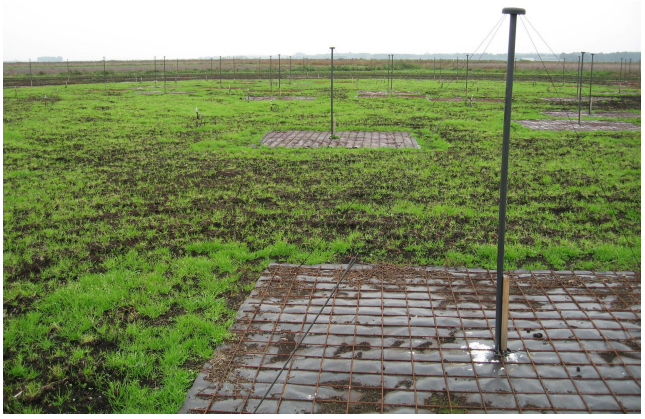


Figure 1. A field with low-band antennas (dipoles).

Another novelty is the elaborate use of *software* to process the telescope data in real time. Previous generations of telescopes depended on custom-made hardware to combine data, because of the high data rates and processing requirements. However, the desire for a flexible and reconfigurable instrument with different processing pipelines for different observation types demands a software solution. The availability of sufficiently powerful supercomputers allows this.

The most common mode of operation for LOFAR is the *standard imaging pipeline*, which is used to generate sky images. This mode filters and correlates the data sent by the stations in the field. Several *pulsar pipelines* are being developed as well, that either search large sky regions to find unknown pulsars or, once found, sensitively observe their characteristics. We also started development of a *transient pipeline*, that observes the sky for transient events. The pipelines share common components, shortening their development time. The software also supports multiple simultaneous observations, even of different types. The first part of each pipeline runs in real time, since the receivers produce too much data to store on disk. Only after substantial reduction of the data volume, intermediate data products are written to disk.

In this paper, we focus on the real-time part of the standard imaging pipeline, commonly called “the correlator”. We first

PPoPP’10, January 9–14, 2010, Bangalore, India.

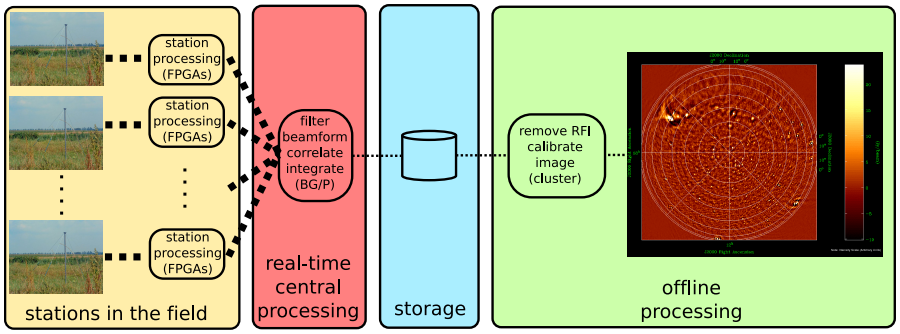


Figure 2. A simplified overview of the LOFAR processing.

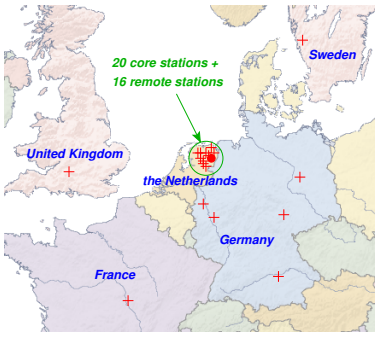


Figure 3. Locations of the stations.

present an integral overview of the correlator and a discussion of the optimizations that we implemented. The main contribution of this paper is an in-depth study of all performance aspects, real-time behavior, and scalability characteristics of the correlator as a whole (previous papers only focused on single aspects and did not include a scalability study). We also enumerate the conditions that are necessary to obtain good, real-time performance, and assert that none of these conditions can be ignored.

The receivers produce hundreds of gigabits per second. To handle the high data rate, we use the BG/P in an unconventional way: we run *application* software on the so-called I/O nodes to preprocess and post-process data that are further handled on the compute nodes. This yields an efficient system and substantially saved us on costs [5]. Additionally, we developed a low-overhead network protocol [10] for communication between I/O nodes and compute nodes, since we were not able to achieve the required internal input and output data rates with the standard network software. The correlator achieves very high computational performance: it sustains 96% of the theoretical floating-point peak performance during the computational phase [11]. The application scales to data rates well beyond the requirements; in fact, the performance is so good that it led to the decision to change the LOFAR specifications: the instrument can observe over 50% more sources or frequencies simultaneously than in its original design specifications, greatly enhancing the efficiency of the entire instrument.

This paper is structured as follows. We mention related work in Section 2. In Section 3, we give an overview of the LOFAR telescope. Then, in Section 4, we describe the hardware characteristics of the BG/P. Next, we explain how we process the telescope data on the BG/P, (Section 5), focusing on the processing on the I/O nodes (Section 6) and compute nodes (Section 7). Section 8 extensively discusses performance results. In Section 9, we briefly illustrate the astronomical results of the correlator output. Finally, we discuss, conclude, and describe future work in Section 10.

2. Related Work

The idea to implement a correlator in software has been adopted by others as well. However, the LOFAR correlator is the only system capable of processing a large number of inputs at high data rates in real time. Other systems handle only a few inputs, handle limited data rates, or do not run in real time.

Deller et al. [3] have developed the DiFX distributed software correlator, which is to be deployed on a cluster of PCs. Due to the use of commodity hardware, both their communication and computational capabilities are substantially lower than those available in our Blue Gene/P. The real-time data processing of the Murchison Widefield Array (MWA) telescope is implemented partially in software. However, their correlator, computationally the most demanding part of the processing pipeline, is not implemented in software, but on FPGAs [9]. Finally, the Joint Institute for VLBI in Europe (JIVE) develops a new software correlator for e-VLBI observations, but is not capable of processing telescope data in real time [6], even though the title of their paper suggests otherwise.

On our previous platform, the BG/L, we were dissatisfied with the I/O model and its performance. This led to a joint effort with Argonne National Laboratory to redesign the entire network software infrastructure, and resulted in a new environment called *ZOID* [5]. ZOID yields far better performance and is much more flexible, since it allows application code to be run on the I/O nodes. The standard BG/P software infrastructure is a major improvement over the BG/L, since it now incorporates several of ZOID's key ideas (e.g., that of running application code on the I/O nodes). Therefore, we do not have to use ZOID on the BG/P anymore. Nevertheless, the achieved performance of the collective network that we use to send LOFAR data from the I/O nodes to the compute nodes still is unsatisfactory. We therefore developed our own high-performance low-overhead protocol, called FCNP. We describe FCNP in [10], but summarize the relevant features in Section 6.1.

In another paper, we compare the efficiency of five many-core architectures (GPUs from NVIDIA and ATI, the Cell/B.E., the Blue Gene/P, and the Intel Core i7) for correlation purposes [8].

3. The LOFAR Telescope

LOFAR is driven by the astronomical community, which needs a new instrument to study an extensive amount of new science cases. Five key science projects have been defined. First, we expect to see the *Epoch of Reionization* (EoR), the time when the first star galaxies and quasars were formed. Second, LOFAR offers a unique possibility in particle astrophysics for studying the origin of high-energy *cosmic rays*. Neither the source, nor the physical process that accelerates such particles is known. Third, LOFAR's ability to continuously monitor a large fraction of the sky makes it uniquely suited to find new *pulsars* and to study *transient sources*. Since LOFAR has no moving parts, it can instantaneously switch focus to some galactic event. Fourth, *Deep Extragalactic Surveys* will be carried out to find the most distant radio galaxies and study star-forming galaxies. Fifth, LOFAR will be capable of observing the so far unexplored radio waves emitted by *cosmic magnetic fields*. For a more extensive description of the astronomical aspects of the LOFAR system, see De Bruyn et. al. [2].

A global overview of the LOFAR instrument is given in Figure 2. LOFAR uses two different types of antennas: the Low-Band Antennas (LBA) for the 10–80 MHz frequency range and High-Band Antennas (HBA) for the 110–250 MHz band. FM radio transmissions make the in-between range unsuitable for observations. Figure 1 shows a field with LBAs. Each LBA consists of one dipole per polarization, while each HBA is organized as a tile combining 16 antenna elements. All antennas are dual polarized.

LOFAR's antennas are structured in a hierarchical way to limit the costs of data transport and processing. Tens of thousands of antennas are necessary to obtain sufficient sensitivity. The antennas are distributed over a large area to achieve a high angular resolution. However, combining the data of all individual antennas centrally would require too much network bandwidth and would result in excessive computational requirements. Therefore, multiple antennas are grouped to form a *station*. The signals of the receivers are combined locally, within the station, using FPGAs.

Each station is equipped with 48–96 LBAs and 48–96 HBA tiles. A station also features a cabinet where initial processing is done, like analog-to-digital conversion, filtering, frequency selection, and combination of the signals from the different receivers. One of the distinctive properties of LOFAR is that the receivers are omni-directional, and that multiple, simultaneous observation directions are supported. Since observing the sky in all frequencies and all directions at the same time would result in an unmanageable output data rate, the observer selects a limited number of directions and frequencies, called *subbands*.

Geographically, LOFAR consists of a 2-kilometer wide compact core area of 20 stations, 16 remote stations with a maximum distance of 125 km, and 8 international stations, with a maximum distance of 1300 km (see Figure 3). The heart of LOFAR is installed in the Northern part of the Netherlands. Each HBA field of a compact core station can optionally be split into two fields, so they can appear as 40 core stations to the correlator. The long distance between the European stations allows observations with high angular resolution, but with a limited Field-of-View; therefore the European stations will not be used for all observations. The roll-out of the stations is currently in progress. As of October 2009, six stations are fully functional; another 22 stations are under construction or partially functional.

The station data are transported to the central processing location via a Wide-Area Network (WAN), using dedicated light paths. We use UDP for data transport, since we can easily tolerate some data loss. We do not use a reliable protocol such as TCP, because this significantly complicates the programming of the station FPGAs, due to buffering, flow control, retransmission, and real-time issues.

The UDP packets contain samples, where a sample is a complex number that represents the amplitude and phase of a signal at a particular time. A sample is encoded by a 2×4, 2×8, or 2×16-bit complex integer. Data can be invalid for various reasons, such as lost network packets or *Radio Frequency Interference* (RFI, e.g., caused by TV transmitters). Throughout the entire processing chain, we maintain which data is marked as invalid, so that eventual images are not distorted by bad data.

This paper focuses on the real-time, central processing of LOFAR data on an IBM Blue Gene/P supercomputer, and in particular on the standard-imaging mode. We chose the BG/P as the central processing platform, instead of, for instance, a cluster with a fast local interconnect, for several reasons. First, the BG/P has excellent hardware support for complex numbers, a feature that is of key importance for signal-processing applications, but is lacking in general-purpose architectures. Moreover, the BG/P has a very high memory bandwidth per operation. This leads to superior performance for our data-intensive applications, compared to other platforms [8]. In addition, the BG/P provides a high-speed 3D-torus network that can effectively implement a data-transpose that is crucial for our application, at the bandwidth we require (see Section 7.2). Finally, the BG/P is a power-efficient supercomputer. Electrical power costs form a large part of the operational costs for the LOFAR instrument. At the time of purchase, the BG/P was the highest-ranking system on the green500 list for energy efficient supercomputers (see www.green500.org).

Our pipeline on the BG/P filters the data, and splits the subbands in narrower frequency bands called *channels*, which allow for more accurate RFI removal. In addition, we perform phase shift and bandpass corrections. Finally, the signals from all stations are optionally beam-formed, correlated and forwarded to a storage cluster, where results can be kept for several days. After an observation has finished, further processing is done off-line, on commodity cluster hardware. Despite considerable computational challenges, the scope of this paper does not cover the off-line processing.

Prototypes of other observation modes, used to find and observe pulsars, are functional, but are not optimized for performance yet. Hence, we do not discuss the pulsar modes in this paper. However, the presence of multiple observation modes demonstrates the flexibility of a *software* solution.

4. The Blue Gene/P

Initially, LOFAR used a 6-rack IBM Blue Gene/L supercomputer for real-time processing of the station data. We recently replaced the system by a more powerful 3-rack Blue Gene/P. Below, we describe the key features of the Blue Gene/P. More information can be found elsewhere [12].

Our system contains 12,480 processor cores that provide 42.4 TFLOPS peak processing power. One chip contains four PowerPC 450 cores, running at a modest 850 MHz clock speed to reduce power consumption and increase package density. Each core has two Floating-Point Units (FPU) that provide support for operations on complex numbers. The compute nodes run a fast, simple, single-process kernel *(Compute Node Kernel, CNK)*,

The BG/P contains several networks. A fast *3-dimensional torus* connects all compute nodes and is used for point-to-point and all-to-all communications. Unlike the BG/L, the torus uses DMA to offload the CPUs and allows asynchronous communication. The *collective network* is used for MPI collective operations, but also for external communication. Additional networks exist for fast barriers, initialization, diagnostics, and debugging.

Each group of (in our case) 16 compute nodes is connected to an I/O node via the collective network. Normally, the I/O node is used as a black box that provides transparent communication from the compute nodes to external systems. In Section 6, we show that it is much more efficient to run part of the application software on the I/O node. An I/O node uses the same hardware as a compute node, but has its 10 Gb/s Ethernet interface connected and runs another operating system (a modified Linux kernel). The group of one I/O node and its associated compute nodes is called a *pset*. Our system has 192 psets in total, 64 per rack.

5. LOFAR Processing

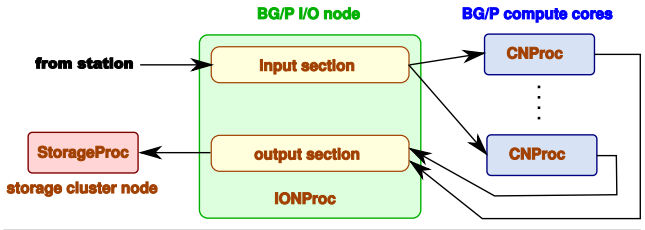


Figure 4. Data flow diagram for the central processing pipeline.

The LOFAR station data are centrally processed in real time by a collection of three distributed applications. These applications run on different platforms: the Blue Gene/P *I/O nodes*, the Blue Gene/P *compute nodes*, and on external (PC-like) *storage nodes*. Figure 4 shows how the data flows through the entire processing

chain. The first application, *IONProc*, runs on the Blue Gene/P I/O nodes. Its main tasks are to receive the station UDP data, to buffer the data for up to 2.5 seconds, and to forward it to the compute nodes in the pset. The second application, called *CNProc*, runs on the Blue Gene/P compute nodes, where the compute-intensive processing takes place. The main tasks are to reorder the data across the compute nodes over the internal torus network, to filter the data, and to beam-form and/or correlate the filtered data. The resulting data are then sent back to the I/O-node application, that collects the data from the compute nodes and sends the data to the storage nodes. This is where the third application (StorageProc) runs. The storage nodes are PC-like systems with large disks. The storage application collects the data from the I/O nodes and writes the data to disk.

6. I/O-node Processing

We use the Blue Gene in an innovative way, by running application software on the I/O nodes. On the Blue Gene/L, this required rewriting major parts of the system software [5], but this idea is much better supported on the Blue Gene/P.

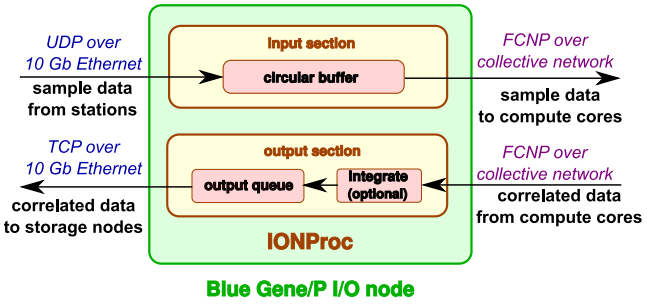


Figure 5. Data flow diagram for the I/O nodes.

We run one multi-threaded process on each I/O node that takes care of two tasks: the handling of input and the handling of output (see Figure 5). The *input section* deals with receipt of UDP station data, buffering, and forwarding to the compute nodes. The *output section* collects outgoing result data from the compute nodes, optionally integrates the data over multiple seconds in time, and forwards the data to storage. An I/O node may run both sections, only one of them, or none at all, depending on the configuration. Both tasks are described in detail below.

6.1 The Input Section

The LOFAR stations send UDP packets with sampled data over a dedicated Wide-Area Network to a BG/P I/O node. The data are received by the *input section*. To simplify the implementation of the correlator, there is a one-to-one mapping between stations and I/O nodes, so that one I/O node receives all data from a single station. However, handling the full 3.1 Gb/s data rate of a station on a relatively slow CPU is quite a challenge, since sufficient processing time must be left for handling output as well. Note that an I/O node does not run the input section if it is not connected to a station.

The input section receives the UDP packets, taking care of out-of-order, duplicated, and lost packets. At each station, four of the FPGAs send data to their associated I/O node, each FPGA to a different UDP port. The I/O node runs four "input" threads, one thread per socket. Multiple threads are necessary, since we have to utilize multiple cores; a single core is too slow to receive all data. Together, the threads receive a total of 48,828 packets per second.

The samples from the received UDP packets are copied into a circular buffer that holds the most recent 2.5 seconds of data. The buffer serves three purposes. First, it is used to synchronize the stations, since the travel times over the WAN are higher for the international stations than for the central stations. Second, the buffer prevents data loss due to small variations in processing times of the remainder of the pipeline. Third, the buffer is used to artificially delay the stream of samples, as we will explain in Section 7.3. The buffer is limited by the small memory size, but due to good real-time behavior of the application, 2.5 seconds is sufficient.

Another thread reads data from the circular buffer and sends the data to the compute nodes for further processing. It sends data in large bursts that contain approximately one second worth of samples. Unfortunately, existing network software did not provide sufficient bandwidth and consumed too much CPU time. We therefore developed *FCNP (Fast Collective-Network Protocol)*, a network library for high-bandwidth communication between the I/O nodes and the compute nodes [10]. FCNP achieves link-speed bandwidths for large messages, due to its low overhead. The data are sent directly from the circular buffer without additional copying. In contrast to the UDP receive, one thread is sufficient to obtain the required throughput, thanks to the low processing overhead of FCNP.

The correlator typically processes in real time, but can also correlate pre-recorded data off-line, frequently used for experimental observations. When processing in real time, the NTP-synchronized wall-clock time is used to trigger the sending of a new block of data. A block of data containing samples from time t_1 to t_2 are sent some hundreds of milliseconds (the WAN delay plus a safe margin) after t_2, whether or not all data were actually received from the station. This assures real-time continuation of the correlator and provides fault-tolerance against a failing station or WAN link. In practice, this method causes hardly any data loss. When processing off-line, the input is read from file or TCP socket rather than a UDP socket. In off-line mode we do not use the wall-clock time as trigger, but we synchronize the threads that read and write the circular buffer differently to prevent them from overtaking each other.

6.2 The Output Section

The bulk of the signal processing is done on the compute nodes, on which we elaborate in Section 7. The resulting output data are sent back to the I/O node. The second major task of the I/O-node application is the *output section*, that handles output data. This task consists of four operations.

First, the data are received from the compute nodes, also using FCNP. Second, the data are optionally added to previously received data from other compute nodes in the pset, if integration over multiple seconds is desired. Third, the (possibly integrated) output is queued in a buffer. Fourth, another thread asynchronously dequeues the data and sends them to a storage node, using TCP.

The queue improves real-time behavior and increases fault tolerance, since it handles data on a best-effort basis. If, for any reason, the data are not sent quickly enough to the storage node, the queue fills up and subsequent data are simply discarded until space is available. This way, we can tolerate disk and network failures. This mechanism is important to keep the correlator running in real time: it is much better to lose a small part of the data than to stall the entire correlator and lose *all* data. Under normal circumstances, no data are lost here.

6.3 Optimizations

Processing power on the I/O nodes is a scarce resource, and most observation modes are I/O bound. We performed many optimizations to improve processing speed. An important improvement was to implement the function that copies data from a received UDP packet to the circular buffer in assembly. This way, we can exploit the efficient 16-byte load and store instructions, which are un-

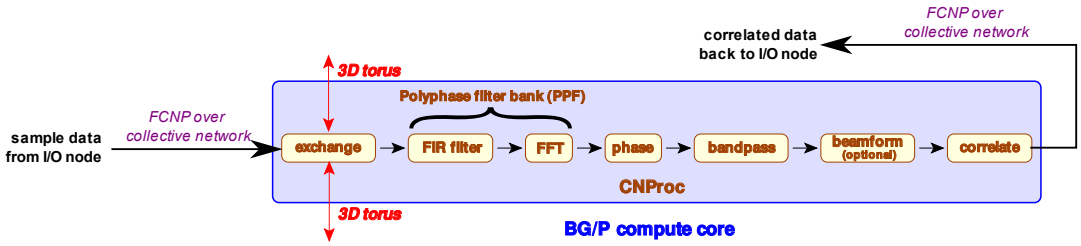


Figure 6. Data flow diagram for the compute nodes.

known to the C++ compiler. Unfortunately, the copy itself cannot be avoided, since an UDP packet contains data of many frequency subbands that must be stored to different memory locations.

Despite this optimization, we initially found that copying was very slow. This was caused by the fact that the PowerPC 450 cannot handle TLB[1] misses in hardware, but generates an interrupt and handles the fault in software. This is not a problem on the compute nodes, where the compute-node kernels map all memory using a few large pages, so that TLB misses do not occur. However, the I/O nodes run a Linux kernel that typically uses a page size of 4 KiB, generating a huge number of TLB-miss interrupts.

To avoid the interrupts, we use a modified ZeptoOS (Linux-based) kernel[14]. It allows a process to pin 1.5 GiB (out of 2 GiB) of physical memory in its virtual memory map, using six fixed mappings of 256 MiB that are never evicted from the TLB. Hence, this memory does not generate TLB misses. The remainder of the memory is used for normal, paged operation. The application uses the fast memory for the circular buffer and for the output queues. Copying data from received UDP packets to the input buffer is up to five times faster than when using paged memory.

To achieve good real-time behavior, we found that it is of utmost importance to carefully manage thread priorities using the Linux real-time scheduler. Since the compute nodes must always be able to proceed, they must be fed with data without delays. Therefore, the thread that sends data from the circular buffer to the compute nodes runs at the highest priority, and is scheduled as soon as the wall-clock time triggers. The thread that reads results from the compute nodes is almost as important, since compute nodes will not accept new work before the previous results were read by the I/O node. Other threads, such as the threads that read UDP data, and the threads that send data from the output queues are less important: if they would ever fail to meet a real-time deadline, only a small amount of data is lost. In practice, under normal circumstances, this rarely happens (see Section 8.1).

7. Compute-node Processing

The bulk of the signal-processing computations take place on the compute nodes. In this section, we continue describing the processing pipeline depicted in Figure 4. We explain how the work is scheduled over the compute nodes, how the data are received from the I/O nodes, how the data are exchanged between other compute nodes, what signal processing takes place, and which optimizations were implemented. The compute node pipeline is shown in more detail in Figure 6.

7.1 Scheduling

The I/O node chops the data stream that comes from the station into chunks of one frequency subband and approximately one second of time. Such a chunk is the unit of data that is sent to the compute node for further processing. Since processing a chunk typically takes much longer than one second, the chunks are distributed round robin over a group of processor cores, as illustrated by Figure 7. Subsequent chunks are processed by different processor cores. A core first receives data from the I/O node, processes them, sends back the results, and idles until the I/O node sends new data. A core must finish its work before it is time to process the next chunk.

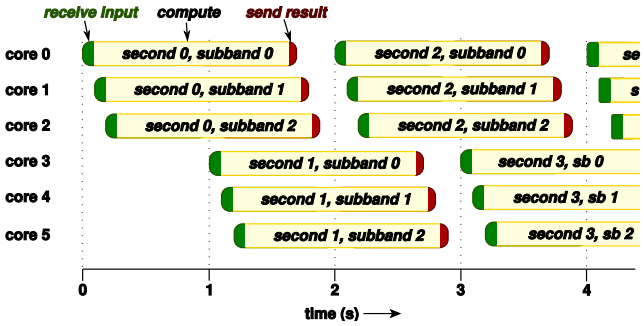


Figure 7. Round-robin work distribution.

For simplicity, Figure 7 shows the processing of three subbands on six cores. In reality, scheduling is more complex. The subbands that must be processed are first (more or less) evenly divided over the psets. Typically, a pset is responsible for a fixed set of four to sixteen subbands. Then, the subbands are scheduled round robin over the 64 cores *within the pset*. For example, if a pset processes six subbands, then every second, the next six cores are scheduled and each of the cores will process one subband. In this example, the available time to process one subband is ten seconds ($\lfloor \frac{64}{6} \rfloor$). Since consecutive chunks of a particular subband are always processed by cores within the same pset, the output for the subband is always sent via the same I/O node. This greatly simplifies communication to the storage nodes and avoids all-to-all communication over the 10 GbE switches. If we would have scheduled all subbands over one large pool of compute cores rather than psets, additional communication over the torus to reroute the output would have been necessary. On the BG/L, this could not be implemented efficiently due to the inability to asynchronously communicate data using a DMA engine; on the BG/P, it unnecessarily increases torus communication.

7.2 All-to-All Data Exchange

The compute nodes perform several operations on the data, as shown in Figure 6. The very first step is to exchange data with another group of processor cores. This is necessary, because an I/O node receives all frequency subbands from one station, but the correlator requires one frequency subband from all stations (we explain this in more detail below). The data exchange is challenging, since it involves hundreds of gigabits per second. Unfortunately, an

[1] Translation Look-aside Buffer: a cache that caches virtual-to-physical address mappings — indispensable for efficient virtual memory.

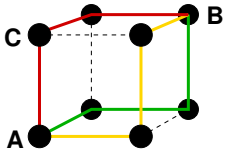


Figure 8. The bandwidth between colinear nodes is lower than between non-colinear nodes.

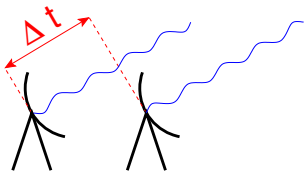


Figure 9. The left antenna receives the wave later.

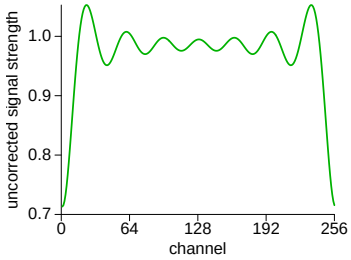


Figure 10. Channels have different signal powers.

I/O node cannot send the data directly from the circular buffer to the compute core that will process the data, since the I/O node is only connected to the compute nodes in its own pset. The data are thus first sent over the collective network from the I/O node to a compute node and then over the 3-D torus network. The torus provides high bandwidth and switches packets efficiently.

Unfortunately, it is not possible to optimize for short torus paths; due to the physical location of I/O nodes and compute nodes, the data are necessarily exchanged between distant nodes. The bandwidth between colinear and coplanar nodes is lower than between non-coplanar nodes, since non-coplanar nodes communicate over more links (in three dimensions) simultaneously. Figure 8 illustrates this; the bandwidth between nodes A and B is (in theory) three times as high as the bandwidth between nodes A and C (in practice, it is somewhat less). Therefore, we schedule work that needs exchange of data on non-coplanar cores as much as possible. We also schedule the work so that multiple cores of the same processor do not need to access the torus or collective network simultaneously, since these resources are shared and simultaneous access decreases performance. The program parts that implement the data exchange and scheduling are, in the presence of many stations, many subbands, time slicing, round-robin core allocation, and avoidance of resource conflicts, *extremely* complicated, but highly efficient.

On the BG/L, the data exchange was implemented synchronously, using `MPI_Alltoallv()`. The BG/P, in contrast, uses DMA for the torus, allowing asynchronous communication. We re-implemented the exchange using asynchronous point-to-point communication, that overlaps the communication over the torus network with the transfer from the I/O nodes to the compute nodes, and with the next four processing steps. As soon as a chunk of data from one station has arrived, the core starts processing them, up to the point that the data from *all* stations are required. As we will explain in the next section, this is before the beam-forming step.

7.3 Signal Processing

After the data exchange, a compute core has the samples of one subband from all stations. The data are processed in a number of steps, as shown in Figure 6. We briefly describe the steps below; more details can be found elsewhere [11]. We finish the section with some remarks on flexibility and optimizations.

7.3.1 Data Conversion

First, we convert the 4-bit, 8-bit, or 16-bit little-endian integer samples to 32-bit big-endian floating point numbers. We do this because the Blue Gene is much better at floating-point processing than integer processing. Unfortunately, there is no hardware support for integer to floating-point conversions. We therefore use a lookup table to convert 4-bit and 8-bit numbers, and an efficient assembly implementation to convert 16-bit numbers. Since the conversion increases the data size, we perform it *after* the data exchange. Since the samples are at most 16-bit wide, single-precision floating point is enough for our purposes. Other telescopes use typically 2–4 bits per sample.

7.3.2 The Poly-Phase Filter Bank

Next, the subband data are processed by a Poly-Phase Filter bank (PPF) that splits a frequency subband into a number of narrower frequency channels. In this step, we trade time resolution for frequency resolution: we split a subband into N separate channels, but with an N-times lower sampling rate per channel. With the higher frequency resolution, we can remove RFI artifacts with a higher accuracy later in the pipeline. Typically, a 195 KHz subband is split into 256 channels of 763 Hz, but the filter supports any reasonable power-of-two number of channels for different observation modes.

The PPF consists of two parts. First, the data are filtered using Finite Impulse Response (FIR) filters. A FIR filter simply multiplies a sample with a real weight factor, and also adds a number of weighted samples from the past. Since we have to support different numbers of channels, our software automatically designs a filter bank with the desired properties and number of channels at run time, generating the FIR filter weights on the fly. This again demonstrates the flexibility of a software solution. For performance reasons, the implementation of the filter is done in assembly. Next, the filtered data are Fourier Transformed. We use the Blue Gene "Vienna" version of FFTW [7] to do this. Since the most common observation mode uses 256 channels, we optimized this case a bit further, and manually wrote a more efficient assembly implementation for the 256-point FFT.

7.3.3 Phase Shift Correction

Due to the finite speed of electromagnetic waves, the wavefront from a celestial source hits stations at different times (see Figure 9). The time difference depends on the direction of the observed source and on the station positions, and is continuously altered by the rotation of the earth. Therefore, all station streams have to be aligned before the signals can be correlated.

Since delays can be larger than the sample period, we perform delay compensation in two steps. First, we correct for integer multiples of the sample period by simply delaying the streams of station samples. This shift is performed on the I/O node, by moving the read pointer of the circular buffer (see Section 6.1).

Second, the remaining error is corrected by rotating the phase of the signal. The phase rotation itself requires a complex multiplication per sample. The exact delays are computed for the begin time and end time of a chunk, and interpolated in frequency and time for each individual sample, with another complex multiplication.

7.3.4 Bandpass Correction

The bandpass correction step compensates for an artifact introduced by a filter bank that runs on the FPGAs in the stations. This filter bank performed the initial division of the antenna signals into subbands. Without correction, some channels have a stronger signal

than others (see Figure 10). The correction is performed by multiplying each complex sample by a real, channel-dependent value that is computed in advance. A station cannot correct for this artifact itself, since it is only visible in channels, not in subbands.

7.3.5 Finalizing the Asynchronous Transpose

Up to this point in the pipeline, processing chunks from different stations can be done independently, but from here on, the data from all stations are required. Therefore, the asynchronous exchange ends here, before the beam forming.

7.3.6 Beam Forming

The beam forming step is optional, and adds the samples from a group of stations that are close together, so that the group forms a virtual "superstation" with more sensitivity. By applying an additional phase rotation (a complex multiplication), beam forming can also be used to select observation directions, or to observe a large parts of the sky simultaneously. The first is used for known pulsar and transient observations, while the latter can be used when searching for unknown pulsars, for instance. The different beam forming modes are implemented, but not yet fully optimized. Therefore we only mention them here to show the flexibility of a software solution, but do not include them in the performance measurements of Section 8.

7.3.7 Correlation

Finally, the samples from individual or grouped stations are correlated. The received signals from sky sources are so weak, that the antennas mainly receive noise. To see if there is statistical coherence in the noise, simultaneous samples of each *pair* of stations are correlated, by multiplying the sample of one station with the complex conjugate of the sample of the other station. To reduce the output size, the products are integrated, by accumulating all products. We accumulate 768 correlations at 763 Hz, so that the integration time is approximately one second, the size of a chunk. The correlator is the most time-consuming operation in the signal processing path, because its cost grows quadratically with the number of stations. All other steps have a lower time complexity.

7.3.8 Flexibility

We support simultaneous pulsar and imaging observations, even on the same data. This is more efficient since the computations in the shared components of the pipelines are done only once. Moreover, more astronomical science can be done with a single observation. Additionally, in the future these pipelines can benefit from each other. For example, the results from the standard imaging pipeline can be used to calibrate the data in the pulsar pipeline in real time.

7.3.9 Optimizations

For optimal performance, time-critical code is written in assembly, because the performance from compiled C++ code was completely inadequate. We maintain equivalent C++ reference code for testing and portability. The assembly version hides load and instruction latencies, issues concurrent floating point, integer, and load/store instructions, and uses the L2 prefetch buffers in the most optimal way. Most instructions are parallel fused multiply-adds, that sustain four operations per cycle.

Although the FIR filters, FFTs, delay compensation, and bandpass correction are conceptually separate, consecutive blocks, their implementations are highly interleaved to achieve better performance. This increases the efficiency of the L1 cache. Also, the data are laid out in memory in such a way that they are read consecutively as much as possible, allowing burst transfers through the cache hierarchy.

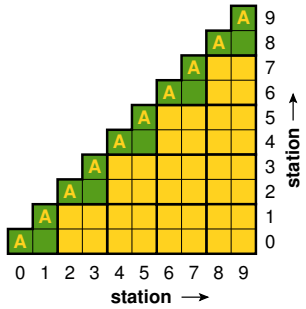


Figure 11. The correlation triangle is divided into 2×2 tiles.

An example of an optimization that we implemented is the reduction of memory references by the correlator [11]. This is achieved by keeping correlations that are being integrated in registers, and by reusing samples that are loaded from memory as often as possible. A sample can be used multiple times by correlating it with the samples from multiple other stations in the same step. For example, a sample from station A in the X polarization that is loaded into a register pair can be correlated with the X and Y polarizations of stations B and C, using it 4 times. Figure 11 shows how we correlate multiple stations at the same time. Each square represents the XX, XY, YX, and YY correlations of the stations as indicated by row and column number. The figure is triangular, because we only compute the correlation of each pair of stations. The squares labeled "A" are autocorrelations, that are treated specially since they require fewer computations. The triangle is divided into as many 2×2 tiles as possible. With this size, the best performance is obtained. For example, the lower right-hand-side rectangle correlates stations 8 and 9 with stations 0 and 1. The X and Y samples of each of these four stations are read, requiring eight memory load instructions (one load instruction reads a complex sample). Computing the correlations requires 128 real operations, i.e., 32 instructions. Hence, four floating-point instructions per load instruction are performed. An unoptimized implementation would perform four times more memory accesses, making the memory subsystem a severe bottleneck. The interleaved correlation computations also help to hide the 5-cycle instruction latencies of the fused multiply-add instructions, since the correlations are independently computed.

8. Performance Analysis

Since only a small number of LOFAR stations have been constructed (the majority will become operational later this year), we will provide performance measurements with externally generated artificial data. We use one Blue Gene/P rack to generate UDP data, another rack for the correlator, and half a rack to receive and dump the correlated data. The signal processing pipeline performance does not depend on the input data: the exact same operations are always performed, regardless of the data values. The experiments thus are *completely realistic*, since the correlator runs exactly the way it would run with real station data.

The storage section, however, does not write the data to disk, since we do not have enough storage nodes available yet, but this does not influence the performance measurements of the correlator. With one rack, we can process up to 64 stations, one per I/O node.

We show the performance results of the application by means of three challenging observation modes which are likely to be commonly used. Table 1 lists the characteristics of these modes. Mode A is the standard mode, where the stations send 16-bit samples. In this mode, the FPGAs can send at most 248 subbands. The 248 subbands are evenly divided over 62 psets, so that each pset

Observation mode	A	B	C
nr. bits per sample	16	8	4
max. nr. of subbands	248	496	992
nr. channels per subband	256	256	256
max. nr. of stations	64	64	48
input bandwidth (nr. I/O nodes * Gb/s)	64 * 3.1 = **198**	64 * 3.1 = **198**	48 * 3.1 = **149**
output bandwidth (nr. I/O nodes * Gb/s)	62 * 0.58 = **36**	62 * 1.2 = **72**	62 * 1.3 = **81**
available compute time per subband (s)	16.1	8.05	4.03

Table 1. Characteristics of three challenging observation modes.

processes 4 subbands (the remaining two psets handle input data, but do not correlate). Since there are 64 cores in one pset and an integration time equals 1.007 second (768 samples), the available time to process one chunk of data (1 subband) is 16.1 second.

Mode B trades accuracy for observation bandwidth, by reducing the sample size to 8 bits and doubling the number of subbands. This doubles the number of frequencies or beams that are observed simultaneously. It implies that the total input data rate remains the same, but that the processing requirements and output data rate double. The 62 psets that are used to correlate have to process 8 subbands each, reducing the available time per subband to 8.05 second.

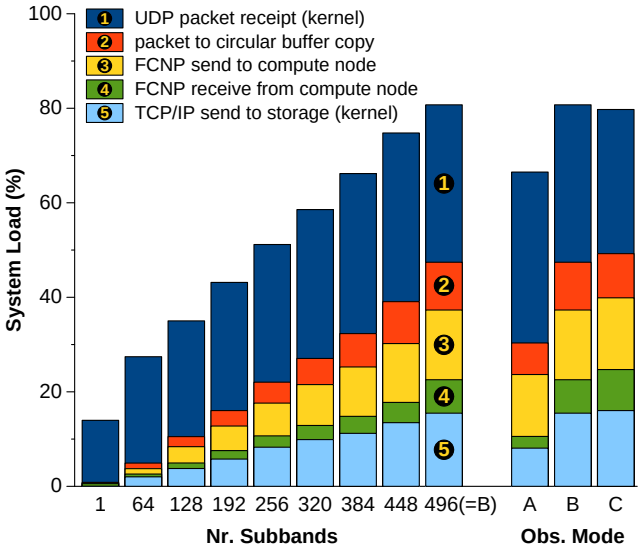
Mode C uses 4-bit samples, and is only suitable for frequency subbands that are mostly free of RFI (otherwise, the bits are used to encode the RFI, not the signal of interest). This mode is planned for *Epoch-of-Reionization* (EoR) observations, where the high number of subbands is used to observe the sky at 32 MHz bandwidth in six directions simultaneously. If the same amount of stations were used, the processing requirements and output data rate would double again, but EoR observations will only use the stations near the center, not the remote ones. The exact number of stations that will be correlated is not yet known, but is likely between 36 and 46. For the performance measurements, we assume the most challenging case, and use 48 stations.

8.1 Performance on the I/O Nodes

The I/O requirements are challenging, and the processing power on the I/O nodes is limited. Figure 12 shows where the cores of the I/O nodes spend their time in various situations. The five major tasks are each represented by a different color in the bar graph; the size of each bar is proportional to the contribution to the total work load. A load of 100% means that all four cores are fully occupied. A load above 85% must be avoided to prevent major data loss.

We first show how the performance scales with the number of subbands. We use a setting resembling observation mode B, for up to 496 subbands, see Figure 12(a). The I/O nodes receive and forward the samples of one station (up to 3.1 Gb/s) and send the correlations of up to 8 subbands to storage (up to 1.2 Gb/s). The figure shows that most time is spent in the receipt of UDP packets. This amount is partially independent of the number of subbands, since a lower number of subbands decreases the packet size (down from 7,998 bytes), but not the amount of packets. The I/O nodes have to handle 48,828 packets per second. All other work scales linearly with the number of subbands.

Figure 12(b) shows the performance breakdown for the three challenging observation modes. In the standard 16-bit sample mode, the stations can produce at most 248 subbands (observation mode A). Hence, the output data rate (the lower two bars) is half as high as in the 8-bit mode of scenario B. Also, copying 16-bit samples into the circular buffer is somewhat more efficient, due to L3-cache effects. In the 4-bit mode, only 48 stations are used.



(a) Performance as function of number of subbands. The input samples are 8 bit, and in total, 64 stations are used.

(b) The three observation modes.

Figure 12. I/O node performance breakdown.

Due to the reduced number of stations, the output data rate is only 13% higher than in the 64-station/8-bit mode, rather than twice the bandwidth of observation mode B.

Both FCNP and the fixed TLB mappings significantly contribute to the low resource usage. Without either of them, the application cannot handle these data rates in real time.

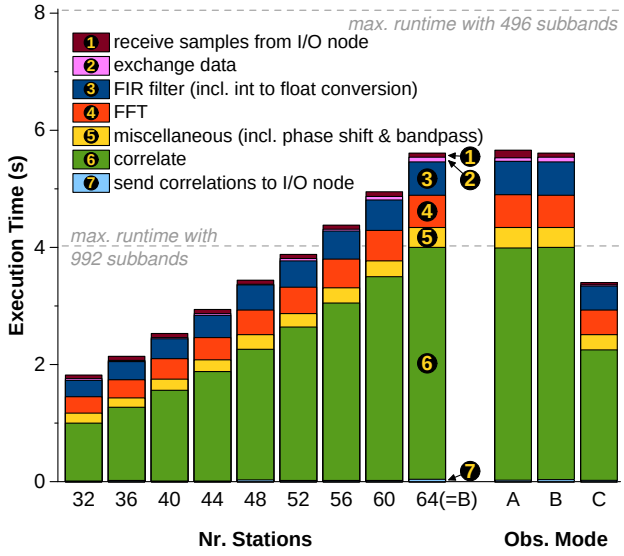
The data loss due to missed UDP packets is low: only between 1 per 10^6 and 1 per 10^4 packets are dropped under full load. These numbers include the data loss caused by the (software) generators and by the 10 GbE network switches. The data loss is negligible to other places where data can be lost, (e.g., we sometimes have to reject tens of percents of the data due to RFI), and does not hurt the astronomical signal quality.

With the I/O-related optimizations, we obtain sufficient bandwidth to support all currently foreseen observation modes on a single rack. If the requirements would change and the need would arise to achieve even higher bandwidths, UDP packet receipt could be optimized by not using the `read()` system call interface, but by using another interface that reads the data directly from kernel buffers and does not enforce a (370 MiB/s!) kernel-to-user-space copy. Right now, we feel no need to implement the required kernel changes. Alternatively, the second rack could be used.

8.2 Performance on the Compute Nodes

Figure 13 shows how the compute nodes spend their time. The vertical axis shows the execution time to process one subband with 1.007 second of station samples.

Before presenting the performance of the three observation modes described above, we show how the performance scales with the number of stations. Figure 13(a) shows execution times for up to 64 stations in a setting that is similar to observation mode B. The $O(n^2)$ complexity of the correlator is clearly visible (the correlations between all *pairs* of stations are computed), while other components scale linearly with the number of stations. Despite the high data rates, I/O requires hardly any time *on the compute nodes*. It is important to realize that the time for input or output cannot exceed $1/64^{\text{th}}$ of the total time, since the associated I/O node also needs time to communicate with the other 63 cores in the pset.



(a) Performance as a function of the number of stations. The input samples are 8 bit, and in total, 496 subbands are processed.

(b) The three observation modes.

Figure 13. Compute node performance breakdown.

The performance results hardly differ for the 16-bit and 8-bit modes, since only the performance of the data receipt from the I/O node and data exchange phase are affected by the sample size, both of which hardly contribute to the total run time. This is clearly illustrated by Figure 13(b), where the execution times for observation modes A and B are nearly the same. The run time for observation mode C is lower, since this mode processes 48 rather than 64 stations. All modes run within their real-time constraints of 16.1, 8.05, and 4.03 seconds respectively. The load on the compute nodes is 35%, 70%, and 84% respectively.

The asynchronous transpose is much more efficient than the original synchronous version. It successfully overlaps communication with computations, reducing the data exchange overhead by roughly a factor of four.

The correlator is extremely efficient: it achieves 96% of the FPU peak performance, due to the highly-optimized assembly code. The FIR filter runs at 86% of the peak performance, and the hand-crafted 256-point FFT runs at 44%. Compared to "Vienna" FFTW, which is already efficient, our hand-written FFT is about 34% faster. Compared to equivalent C++ code that is written for clarity and not specifically tuned for optimal performance, the hand-written assembly code is typically an order of magnitude faster.

Due to all optimizations, the correlator can process 50% more data than the specifications require, on only half the amount of planned resources. Only if the need to correlate more than 64 stations would arise, or if significant additional real-time signal processing would be needed, the second rack must be employed. We can exploit the compute power we saved to run other observation types simultaneously, or to do additional signal processing that improves the signal quality, such as real-time flagging and real-time calibration.

9. Astronomical Results

The system we described is used on a daily basis for observations, using the currently available stations. The images we show in this section are created with *real* data. A graphical representation of the correlator output is depicted in Figure 14. The figure shows the cross-correlations from two of the stations used during a 9-hour observation. The horizontal axis represents time; the vertical axis represents the 256 channels of one frequency subband. Each pixel corresponds to a (complex) correlation, where the color represents the phase of the signal; the intensity matches the amplitude (power). The phase changes over time, due to the earth rotation that alters the relative position of the observed sources and thus the time difference between the two stations. The white spots are caused by RFI; these bad data are detected and ignored in the remainder of the processing pipeline.

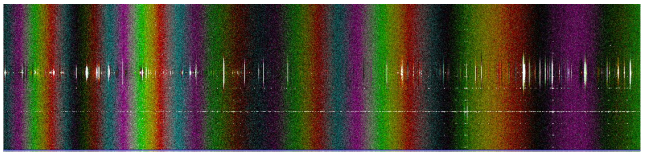


Figure 14. Correlations from a 9-hour observation.

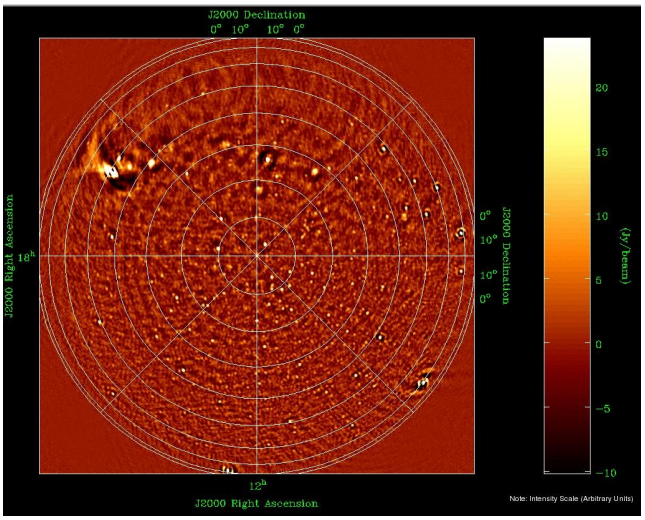


Figure 15. An all-sky image created with LOFAR antennas.

The correlations are used to create images. Even with the limited amount of stations that have been employed, impressive (all-sky) images were made (see Figure 15). Also, the prototype pulsar pipeline software successfully detected several known pulsars [4].

10. Conclusions and Future Work

In general, we are rather satisfied with the capabilities of the Blue Gene/P as a platform for a real-time correlator. The double FPU is highly efficient and provides excellent support for complex numbers, which is indispensable for signal processing. The relatively high memory bandwidth helps to keep the FPUs busy. The 3-D torus easily handles the all-to-all exchange, thanks to the high bandwidth, its switching capabilities, and a DMA controller. We also think that the programming environment of the Blue Gene/P is a considerable improvement over its predecessor, and are pleased with the open programming interfaces. The power efficiency of the Blue Gene/P is good, thanks relatively low clock frequency of the processors. The Cell BE is more energy efficient [8], but does not incorporate a high-speed interconnect.

There are some disadvantages as well. Most notably, the separation of compute nodes and I/O nodes, along with their limited connectivity (within a pset only), and the use of *two* networks types complicates and impedes efficient streaming of data into the machine. For example, data cannot be sent directly from an I/O node

to compute nodes outside its pset. Also, the absence of a hardware TLB-miss handler causes significant performance degradation with paged memory. Furthermore, double-precision floating-point arithmetic is overkill for our application. While many other architectures (e.g., the IBM PowerXCell 8i, SSE) provide twice the number of FLOPS for single-precision arithmetic, this is not the case for the Blue Gene. A minor disadvantage is the omission of an integer-to-floating-point conversion instruction. Finally, the need to use assembly to obtain sufficient performance complicates programming; the gap with compiled C++ code is large.

To handle the high LOFAR station data rates, we use the I/O nodes in an unorthodox way: they run the part of the application software that takes care of external communication. A custom network protocol (FCNP) provides link-speed bandwidths between the I/O nodes and compute nodes, and reduces the CPU utilization. On the I/O nodes, the use of a large, pinned memory area avoids excessive amounts of TLB-miss interrupts. Managing thread priorities using the Linux real-time scheduler is important to achieve good real-time behavior. Special provisions were made to obtain fault tolerance against station, WAN link, and disk failures.

Furthermore, we demonstrated that the correlator achieves exceptionally high performance, both computationally and with respect to I/O, due to the applied optimizations. The correlations are computed at 96% of the FPU peak performance; other signal-processing functions perform impressively as well. The work distribution scheme is efficient but complex, due to the real-time requirements, the need to exchange data, and the avoidance of resource contention.

We showed performance measurements for the most challenging observation modes that are currently foreseen. Due to the optimizations, we need only *half the amount of planned resources* to process *50% more station data* than the LOFAR specifications require. The latter ability led to the decision to adjust the specifications, resulting in a major improvement in the effectiveness of the *entire* telescope.

Generalizing the lessons learned, we conclude that to achieve high performance, high bandwidths, and real-time behavior, it is necessary to consider *all* performance-related aspects of the application integrally, without ignoring any of them:

- close integration with the hardware, e.g., by writing kernels in assembly;
- using real-time thread scheduling;
- using a work distribution scheme that avoids all forms of resource contention;
- using optimized network protocols and asynchrounous I/O;
- computing on I/O nodes;
- operating system modifications to circumvent inefficient hardware (TLBs).

Traditionally, real-time telescope data are processed using customized hardware. However, LOFAR's innovative, dishless design, with many thousands of omni-directional antennas, allows new types of observations that need different processing pipelines. The required flexibility is obtained by using the *software* presented in this paper. For example, we have other functional pipelines for pulsar observations, that we are currently optimizing. Future work includes the integration of other processing pipelines, real-time calibration, and possibly real-time RFI removal.

11. Acknowledgments

We thank Ger van Diepen, Martin Gels, Marcel Loose, and Ruud Overeem for their contributions to the LOFAR software, and many other colleagues for their work on the LOFAR telescope. We also thank Kamil Iskra and Kazutomo Yoshii from Argonne National Laboratory for their work on the BG/P system software. Bruce Elmegreen, Todd Inglett, Tom Liebsch, and Andrew Taufener from IBM provided the support to optimally use the BG/P. Figure 3 was edited from a map from Wikimedia.

LOFAR is funded by the Dutch government in the BSIK program for interdisciplinary research for improvements of the knowledge infrastructure. Additional funding is provided by the European Union, European Regional Development Fund (EFRO), and by the "Samenwerkingsverband Noord-Nederland," EZ/KOMPAS. Part of this work was performed in the context of the NWO STARE AstroStream project.

References

[1] H.R. Butcher. LOFAR: First of a New Generation of Radio Telescopes. *Proceedings of the SPIE*, 5489:537–544, October 2004.

[2] A.G. de Bruyn et al. Exploring the Universe with the Low Frequency Array, A Scientific Case, September 2002. http://www.lofar.org/PDF/NL-CASE-1.0.pdf.

[3] A. Deller, S. Tingay, M. Bailes, and C. West. DiFX: A Software Correlator for Very Long Baseline Interferometry Using Multiprocessor Computing Environments. *Astronomical Society of the Pacific*, 119:318–336, 2007.

[4] J. Hessels, B. Stappers, and J. van Leeuwen. The Radio Sky on Short Timescales with LOFAR: Pulsars and Fast Transients. In *The Low-Frequency Radio Universe*, ASP Conference Series. To appear. http://arxiv.org/pdf/0903.1447.

[5] K. Iskra, J.W. Romein, K. Yoshii, and P. Beckman. ZOID: I/O-Forwarding Infrastructure for Petascale Architectures. In *ACM SIGPLAN Symposium on Principles and Practice on Parallel Programming (PPoPP'08)*, pages 153–162, Salt Lake City, UT, February 2008.

[6] N. Kruithof and D. Marchal. Real-time Software Correlation. In *International Workshop on Distributed Cooperative Laboratories (INGRID'08)*, April 2008. http://www.jive.nl/dokuwiki/doku.php/scarie:scarie.

[7] J. Lorenz, S. Kral, F. Franchetti, and C.W. Ueberhuber. Vectorization Techniques for the Blue Gene/L Double FPU. *IBM Journal of Research and Development*, 49(2/3):437–446, March 2005.

[8] R.V. van Nieuwpoort and J.W. Romein. Using Many-Core Hardware to Correlate Radio Astronomy Signals. In *ACM International Conference on Supercomputing (ICS'09)*, pages 440–449, New York, NY, June 2009.

[9] S. Ord, L. Greenhill, R. Wayth, D. Mitchell, K. Dale, H. Pfister, and G. Edgar. GPUs for data processing in the MWA. In *Astronomical Data Analysis Software and Systems (ADASS XVIII)*, November 2008. To appear. http://arxiv.org/abs/0902.0915.

[10] J.W. Romein. FCNP: Fast I/O on the Blue Gene/P. In *Parallel and Distributed Processing Techniques and Applications (PDPTA'09)*, Las Vegas, NV, July 2009.

[11] J.W. Romein, P.C. Broekema, E. van Meijeren, K. van der Schaaf, and W.H. Zwart. Astronomical Real-Time Streaming Signal Processing on a Blue Gene/L Supercomputer. In *ACM Symposium on Parallel Algorithms and Architectures (SPAA'06)*, pages 59–66, Cambridge, MA, July 2006.

[12] IBM Blue Gene team. Overview of the IBM Blue Gene/P Project. *IBM Journal of Research and Development*, 52(1/2), January/March 2008.

[13] M. de Vos, A.W. Gunst, and R. Nijboer. The LOFAR Telescope: System Architecture and Signal Processing. *Proceedings of the IEEE*. To appear.

[14] K. Yoshii, K. Iskra, H. Naik, and P.C. Broekema P. Beckman. Performance and Scalability Evaluation of "Big Memory" on Blue Gene Linux. *International Journal of High Performance Computing*. To appear.