Processing Real-Time LOFAR Telescope Data on a Blue Gene/P Supercomputer

John W. Romein
romein@astron.nl

P. Chris Broekema
broekema@astron.nl

Jan David Mol
mol@astron.nl

Rob V. van Nieuwpoort
nieuwpoort@astron.nl

ASTRON (Netherlands Institute for Radio Astronomy)
Oude Hoogeveensedijk 4, 7991 PD Dwingeloo, The Netherlands

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.

This paper focuses on another novel feature: where traditional telescopes use customized hardware, we process the data in *software*. 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.

To meet the real-time requirements, the application is highly optimized, and reaches exceptionally high computational and I/O efficiencies. This 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.

1. INTRODUCTION

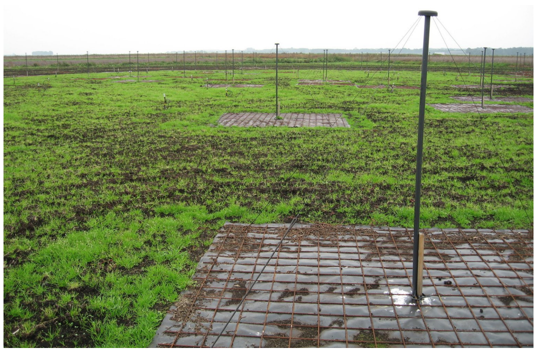


Figure 1: A field with low-band antennas (dipoles).

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, 15], 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 concurrently 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.

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 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. This paper describes the implementation and performance characteristics of the real-time part of this pipeline. We present a highly-optimized implementation that achieves very high computational performance: the correlator sustains 96% of the theoretical floating-point peak performance during the computational phase. 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. The pipelines share common components, shortening their development time. The software also supports multiple concurrent observations, even of different types.

The receivers produce hundreds of gigabits of data per second. To handle the high data rate, we use the Blue Gene/P (BG/P) in an unconventional way: we run *application* software on the so-called I/O nodes to pre-process 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 [12] 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 system software.

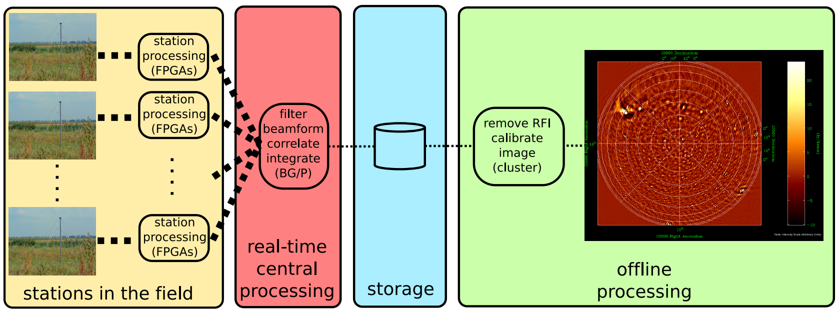

station processing (FPGAs)
station processing (FPGAs)
station processing (FPGAs)
stations in the field
filter beamform correlate integrate (BG/P)
real-time central processing
storage
remove RFI calibrate image (cluster)
offline processing

Figure 2: A simplified overview of the LOFAR processing.

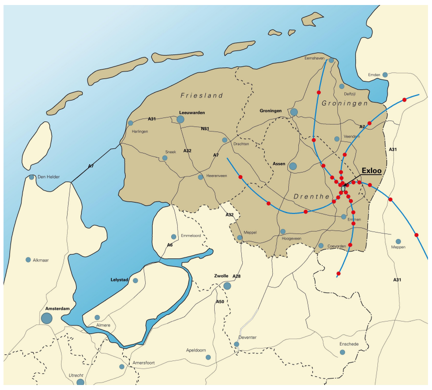


Figure 3: Possible LOFAR layout.

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

2. THE LOFAR TELESCOPE

LOFAR is driven by the astronomical community, that 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 that the first star galaxies and quasars were formed. The 1.42 GHz emission line of hydrogen is expected to be red-shifted into the LOFAR sensitivity range. Second, LOFAR offers a unique possibility in particle astrophysics for studying the origin of high-energy (10^{15}–$10^{20.5}$ eV) *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, wherein 16 antenna elements are combined. 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.

Geographically, LOFAR consists of a compact core area and a number of remote stations (See Figure 3). The heart of LOFAR will be installed in the Northern part of the Netherlands. The stations are roughly distributed along five log-spiral arms with a diameter of hundreds of kilometers. The station fields are centrally condensed, following a logarithmic distribution. About 50% of the stations are located in the 2 km-wide central core. Additionally, eight to fifteen European stations will or have been built, extending the maximum distance to roughly 1000 km. The longer baselines (distance between two stations) allows observations with high angular resolution, but limits the Field-of-View; therefore the European stations will not be used for all observations. In the past several years we have deployed a number of prototype antennas, while the roll-out of the final stations is currently in progress.

Each station is equipped with 48–96 LBAs and 48–96 HBA tiles. A station also features a cabinet where initial processing is done, using FPGA technology. Typical operations that are performed here include 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, concurrent 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*.

The station data are transported to the central processing location in Groningen via a Wide-Area Network (WAN), using owned and leased 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 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 filter the data, and split 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. First, the *flagger* uses an RFI detection algorithm to remove bad data. Then, the results are calibrated for instrumental and environmental effects and for sky source parameters (e.g., position, flux) [9]. An imaging algorithm creates an image of the observed source(s). Despite considerable computational challenges, the scope of this paper does not cover the off-line processing.

A prototype of a set of other observation modes, used to find and observe pulsars, is functional, but is 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. More observation modes will be implemented in the future.

3. 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 an equally powerful 2.5-rack Blue Gene/P. Below, we describe the key features of the Blue Gene/P. More information can be found elsewhere [14].

Our system contains 10,880 processor cores that provide 37.0 TFLOPS peak processing power (including I/O nodes). The system is built using SoC (System-on-a-Chip) technology that integrates all processing and networking functionality on a single die. 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. A core can sustain two fused multiply-adds per cycle. The four cores share 2 GiB of main memory. Although a node can run in *SMP mode*, we run the application in *virtual node mode*, where the processor and memory of a node are split into four independent, virtual machines. This simplifies programming, since this allows single-threaded processing on the compute cores. 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. A *global interrupt network* provides

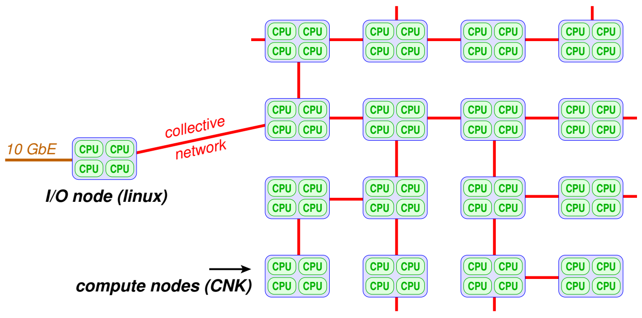


Figure 4: A pset.

support for fast barriers. Additional networks exist for initialization, diagnostics, and debugging.

Each compute node is connected to an I/O node via the collective network (see Figure 4). 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). Since our application demands high bandwidths, our system is configured with the maximum number of 1 I/O node per 16 compute nodes (64 cores). The group of one I/O node and the compute nodes that are connected to it, is called a *pset*. Our system has 160 psets in total, 64 per rack. Normally, the I/O node is used as a black box that provides transparent communication from the compute nodes to external systems: all I/O-related system calls on the compute nodes are forwarded to a daemon that runs on the I/O node and performs the real operation. In Section 5, we show that it is much more efficient to run part of the application software on the I/O node.

4. LOFAR PROCESSING

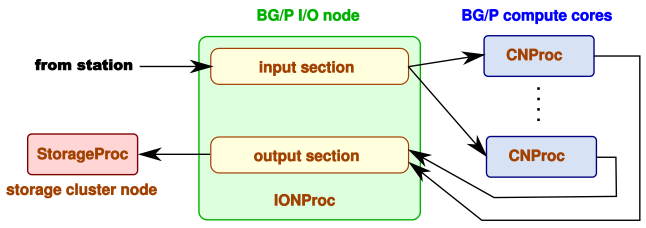


Figure 5: Simplified data flow diagram for the central processing pipeline.

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 5 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 correlate or beam-form 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.

5. 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.

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 6). 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,

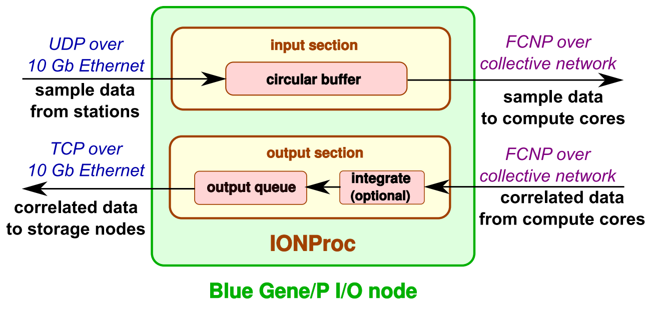


Figure 6: Simplified data flow diagram for the I/O nodes.

only one of them, or none at all, depending on the configuration. Both tasks are described in detail below.

5.1 The input section

The FPGAs at a LOFAR station 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 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 remote 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 6.3. 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 [12]. 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 ruthless method 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.

5.2 The output section

The bulk of the signal processing is done on the compute nodes, on which we elaborate in Section 6. 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 (e.g., due to a disk or network failure), the queue fills up and subsequent data are simply discarded until space is available. 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. In practice, under normal circumstances, no data are lost here.

5.3 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 unknown 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 [16]. It allows a process to map 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

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

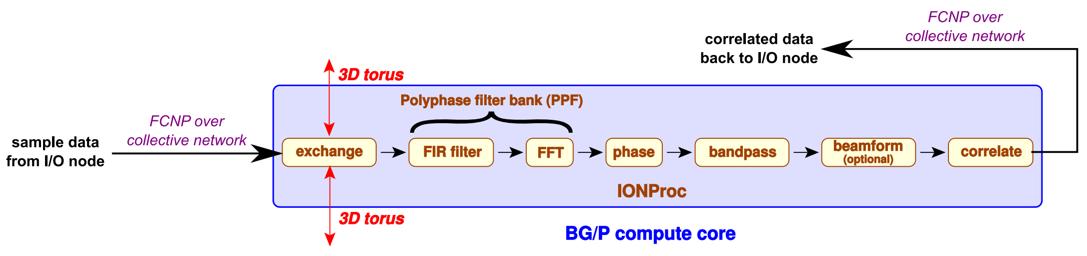

FCNP over collective network
correlated data back to I/O node
3D torus
Polyphase filter bank (PPF)
sample data from I/O node
FCNP over collective network
exchange
FIR filter
FFT
phase
bandpass
beamform (optional)
correlate
IONProc
3D torus
BG/P compute core

Figure 7: Simplified data flow diagram for the compute nodes.

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 equally 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 also important, but 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 7.1).

6. 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 5. 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 7.

6.1 Scheduling

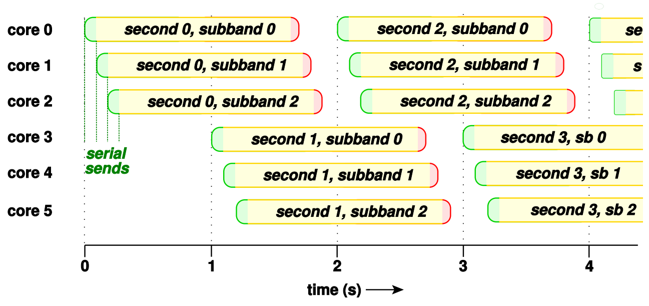


Figure 8: Round-robin processing of work over compute-node cores.

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 round-robin distributed over a group of processor cores, as illustrated by Figure 8. Subsequent chunks are processed by different processor cores. A core must finish its work before it is time to process the next chunk. A core first receives data from the I/O node (green in the figure), processes them (yellow), sends back the results (red), and idles until the I/O node sends new data.

For simplicity, Figure 8 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 round-robin scheduled over the 64 cores *within the pset*. For example, if a pset processes six subbands, 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.

6.2 All-to-All data exchange

The compute nodes perform several operations on the data, as shown in Figure 7. 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 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.

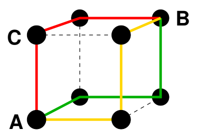


Figure 9: The bandwidth between colinear nodes is lower than between non-colinear nodes.

The torus 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 9 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 data exchange using asynchronous point-to-point communication, that overlaps the communication over the torus network with the data 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.

6.3 Signal processing

After the data exchange, a compute core possesses the samples of one subband, from all stations. The data are processed through a number of filters, as briefly described below. More details on the filters (except bandpass correction and beam forming) can be found elsewhere [13].

The subband data are first filtered by a *Poly-Phase Filter bank* (PPF) that splits a frequency subband into narrow frequency channels, trading time resolution for frequency resolution. The high frequency resolution allows for removal of narrow-band RFI later in the pipeline. The PPF itself consists of a number of 16-tap Finite Impulse Response (FIR) filters, the outputs of which are Fourier Transformed. 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 FIR filters and a 256-point FFT are implemented in assembly, for optimal performance. For other FFT sizes, we use the Blue Gene "Vienna" version of FFTW [7]. This demonstrates the flexibility of a software solution: our software automatically designs a filter bank with the desired properties and number of channels at run time, generating the FIR filter constants on the fly.

As a side effect, the PPF implicitly converts 4-bit, 8-bit, or 16-bit little-endian integer samples to 32-bit big-endian floating point numbers, since the Blue Gene is much better at floating-point processing than integer processing. Unfortunately, the FPU has no hardware support for integer-to-floating-point conversions (unlike floating-point-to-integer conversions). We use a look-up table to convert 4-bit and 8-bit numbers, and a reasonably efficient series of integer and floating-point instructions to convert 16-bit numbers. Since the conversion increases the data size, the PPF runs *after* the data exchange.

The next step is *delay compensation*. Due to the finite speed of electromagnetic waves, the wavefront from a celestial source hits stations at different times (see Figure 10). 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. Before the signals can be correlated, all station streams are aligned, by artificially delaying the streams of station samples. For example, the bulk of a delay of 22 μs is achieved by shifting four 5.12 μs samples. This shift was already done on the I/O node, by moving the read pointer of the circular buffer (see Section 5.1). Here, on the compute nodes, the remaining error (1.52 μs) is corrected by rotating the phase of the signal. The phase rotation itself costs a complex multiplication per sample. Since the rotation depends on the frequency, the correction is done after the PPF: the correction is more accurate on narrow frequency channels. The delays are computed exactly for the begin time and end time of a chunk, and interpolated in frequency and time for each individual sample, with another complex multiplication.

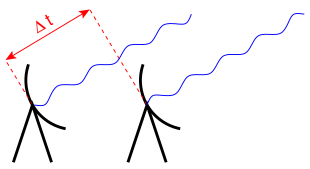


Figure 10: The left antenna receives the wave later.

The *bandpass correction* step compensates for an artefact introduced by a station filter bank (a PPF on the FPGAs that created the subbands). Without correction, some channels contain more power than others. 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 artefact itself, since it is only visible in channels, not in subbands. Another document describes how the correction factors are computed [11].

Up to this point, processing the chunks from different stations can be done independently, but from here on, the data from all stations are required. This means that the asynchronous data exchange ends here.

The next step, called *beam forming*, is optional, and adds the samples from a group of co-located stations, so that the group forms a virtual "superstation" with more sensitivity. This step will typically be used in European observations. From the perspective of an international station, the baselines to the core stations are nearly identical. Treating the baselines separately makes no sense and increases the correlator output unnecessarily. Therefore, the core stations will be grouped. Other uses of grouped, beam-formed stations are foreseen.

This *coherent* way of beam forming is a special case of the beam forming algorithms that are also used for pulsar and transient observations. With coherent beam forming, the (phase-corrected) complex samples from different stations are added. The applied phase correction determines the observation direction. *Incoherent* beam forming is performed by adding the powers (amplitudes) of the samples. Since the phase information is lost, this mode sees a much larger part of the sky, and is used to search for unknown pulsars. Both modes are implemented but not optimized yet, therefore we do not include them in the performance measurements of Section 7.

In the standard imaging mode, the samples from individual or grouped stations are *correlated*. Computationally, this is the most time-consuming operation. 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. Correlation is done for each pair of stations, and for each channel separately. Since the correlation of station A and B is the complex conjugate of the correlation of station B and A, only one pair is computed. Stations are also autocorrelated, i.e., with themselves. Both polarizations of station A are correlated with both polarizations of station B, yielding correlations in XX, XY, YX, and YY directions. The result, a correlation, contains the combined contribution of all visible sky sources. These are disambiguated during the imaging step in the off-line processing pipeline.

We support concurrent pulsar and imaging observations, even on the same data. This is not only computationally more efficient (the computations in the shared components of the pipelines are done only once), more astronomical science can be done with a single observation. Additionally, in the future these pipelines can mutually benefit from each other. For example, the correlations from the standard imaging pipeline can be used to calibrate the station samples in real time in a feedback loop, so that the pulsar pipeline can beam-form calibrated samples. This would not be possible without correlating the data, and would be particularly useful to compensate for station clock drifts, for example.

6.4 Optimizations

For optimal performance, time-critical code is written in assembly, because the performance from compiled C++ code was by far not sufficient. 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.

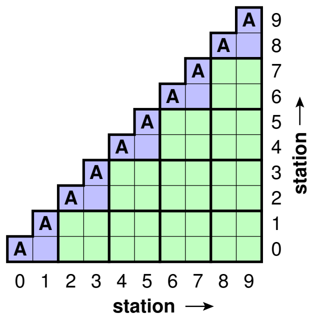


Figure 11: The correlation triangle is divided into as many 2×2 tiles as possible.

An example of an optimization that we implemented is the reduction of memory references by the correlator [13]. This is achieved by keeping correlations that are being integrated in registers, and by reusing samples that are loaded from memory as many times as possible. A sample can be used multiple times by correlating it with the samples from multiple other stations in the same loop iteration. 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. In fact, it is used 8 times, since a complex multiply/accumulate requires two instructions. 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.

7. PERFORMANCE EVALUATION

Since only a small number of LOFAR stations has already 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 the remaining half rack to receive and dump the correlated data. The simulation is 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.

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	64 * 3.1	64 * 3.1	48 * 3.1
(nr. I/O nodes * Gb/s)	= **198**	= **198**	= **149**
output bandwidth	62 * 0.58	62 * 1.2	62 * 1.3
(nr. I/O nodes * Gb/s)	= **36**	= **72**	= **81**
available compute time per subband (s)	16.1	8.05	4.03

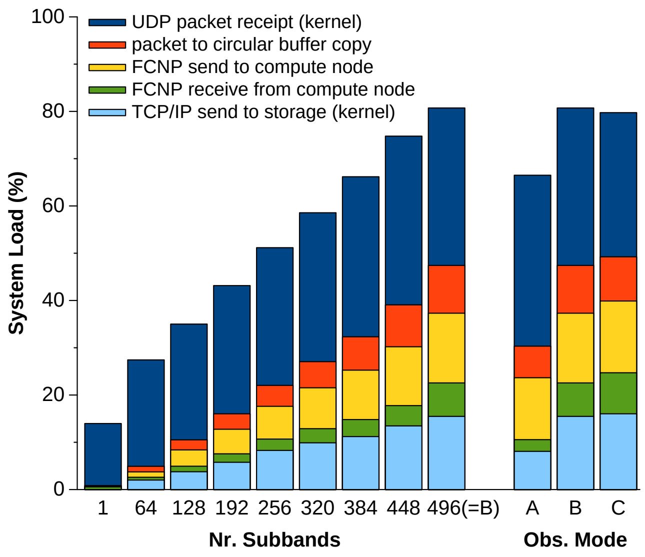
Table 1: Characteristics of three realistic, challenging observation modes.

We show the performance results of the application by means of three challenging observation modes which are likely to be commonly used. The characteristics of these modes are listed in Table 1. 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 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.

7.1 Performance on the I/O Nodes



(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.

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.

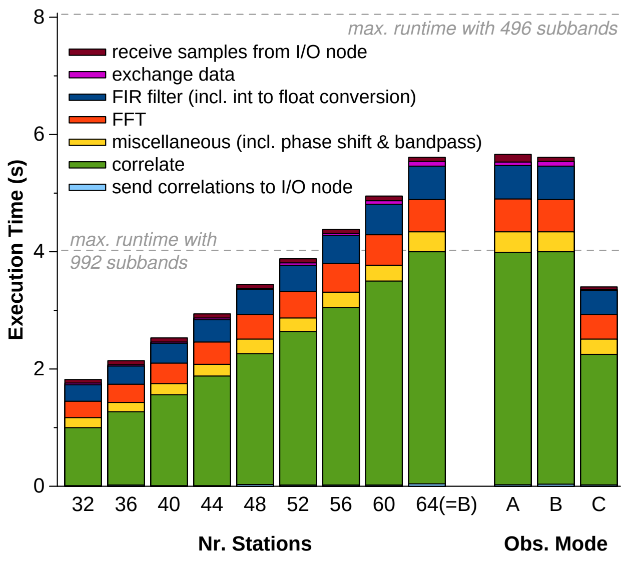
Figure 12(b) shows the performance breakdown for the three challenging observation modes. In the standard 16-bit sample mode, the stations can produce at most 248 subbands (observation mode A). Hence, the output data rate (the lower two bars) is twice as low as in the 8-bit mode of scenario B. Also, 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. 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., the flagger sometimes rejects 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 felt no need to implement the required kernel changes. Alternatively, the second rack could be used.

7.2 Performance on the Compute Nodes



(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.

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 1/64th of the total time, since the associated I/O node also needs time to communicate with the other 63 cores in the pset.

The performance results hardly differ for the 16-bit and 4-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, thanks 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 ever 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.

8. ASTRONOMICAL RESULTS

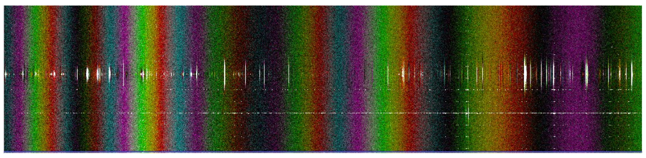


Figure 14: Correlations from a 9-hour observation.

A graphical representation of the correlator output is depicted in Figure 14. The figure shows the cross-correlations from two stations during a 9-hour observation. The horizontal axis varies in 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.

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

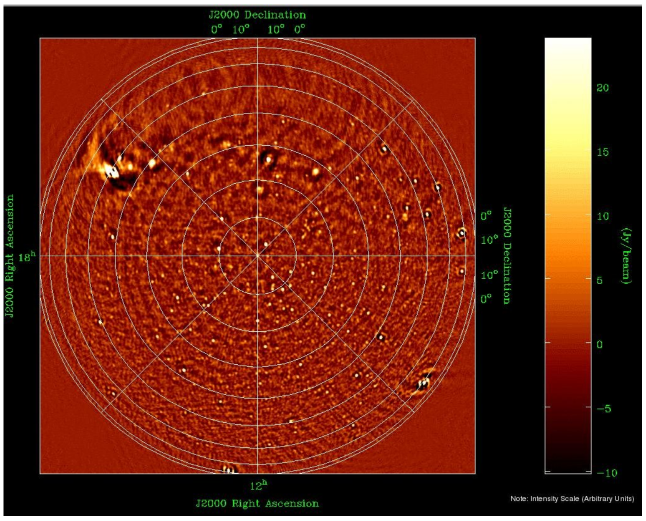


Figure 15: An all-sky image created with prototype LOFAR antennas.

9. 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 [10]. 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.

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

10. DISCUSSION, 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. 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.

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.

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 Blue Gene system software. Bruce Elmegreen, Todd Inglet, Tom Liebsch, and Andrew Taufener from IBM provided the support to optimally use the BG/P.

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.

11. 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. To appear. 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)*, New York, NY, June 2009. To appear.

[9] R. J. Nijboer and J. E. Noordam. LOFAR Calibration. In R. A. Shaw, F. Hill, and D. J. Bell, editors, *Astronomical Data Analysis Software and Systems (ADASS XVII)*, number 376 in ASP Conference Series, pages 237–240, Kensington, UK, September 2007.

[10] 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.

[11] J.W. Romein. Bandpass Correction in LOFAR. Technical report, ASTRON, August 2008. http://www.astron.nl/~romein/papers/BandPass/bandpass.pdf.

[12] 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.

[13] 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.

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

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

[16] K. Yoshii, K. Iskra, P.C. Broekema, H. Naik, and P. Beckman. Characterizing the Performance of Big Memory on Blue Gene Linux. 2009. Under review.