Finding Pulsars in Real-Time

Alessio Sclocco, Henri E. Bal
Faculty of Sciences
Vrije Universiteit Amsterdam
Amsterdam, the Netherlands
a.sclocco@vu.nl, h.e.bal@vu.nl

Rob V. van Nieuwpoort
NLeSC
Netherlands eScience Center
Amsterdam, the Netherlands
r.vannieuwpoort@esciencecenter.nl

***Abstract*—Finding new pulsars has always been a challenging problem, but this challenge is nowadays exacerbated by the increasing data rates of modern radio telescopes. Because of these increased data rates, traditional approaches to searching, based on storing data for off-line processing, are becoming unfeasible. Therefore, we propose a new pulsar searching pipeline that, by exploiting high-performance computing techniques, is able to process observational data in real-time. To achieve the real-time goal we parallelized all the steps of the pipeline to run on many-core accelerators, and used auto-tuning to adapt and optimize the pipeline for different platforms, telescopes, and searching parameters.**

In this paper, we test our pipeline on three different platforms: two Graphics Processing Units from AMD and NVIDIA, and an Intel Xeon Phi. Furthermore, we test it on three different scenarios, based on the operational parameters of three state-of-the-art telescopes. Results show that our pipeline can adapt to all tested platforms and scenarios, and achieves real-time performance and linear scalability. Because power consumption is a main concern for radio telescopes, and will be the main bottleneck for the construction of the Square Kilometer Array, we also measure the power consumed by our pipeline. By comparing the results obtained on many-core accelerators with the results obtained using a traditional multi-core CPU, we conclude that the accelerators can provide up to a factor 8 improvement in execution time, and up to a factor 6 reduction in power consumption.

***Index Terms*—radio astronomy; pulsars; auto-tuning; accelerators**

I. Introduction

Among the most exciting challenges of modern radio astronomy is the discovery of new exotic objects like transients and pulsars. Pulsars are massive and highly magnetized neutron stars rapidly rotating on their axis, whose signal is received on Earth only periodically. Their periodicity is what makes them interesting for scientists, because it can be used to perform experiments on gravitation that are impossible to reproduce in a lab, but it is also what makes them more difficult to detect than stable sources. In fact, pulsars are so difficult to discover that since 1967, the year when the first pulsar was discovered, only around 2,400 have been found, out of a population that astronomers estimate being at least one order of magnitude bigger.

The complexity of finding new pulsars lies in the need to explore a broad three-dimensional search space, looking for a signal with a high enough signal-to-noise ratio (SNR). The three dimensions of the search space are: (1) the position in the sky, (2) the distance of the pulsar from Earth, and (3) the period. The search cannot rely on heuristics to prune the search space, and the more fine grained and extensive it is, the better are the chances of finding new pulsars. Increasing the number of points explored in this search space increases the chances of a discovery, but also boosts the computational costs of the search. Because of the high computational cost, pulsar surveys are traditionally performed off-line, that is the observational data are stored and then processed and analyzed only after the observation took place. However, the growing data rate of modern radio telescopes is pushing this strategy to the limit, making it increasingly more difficult to store the enormous amount of data produced by the instruments. This problem will only be exacerbated in the future, when telescopes like the Square Kilometer Array (SKA) will produce data rates of over 10 Pb/s [1] and require an *exascale* system for processing.

The only foreseeable solution to this problem will be to avoid collecting the data for off-line processing, and perform the search on-line and in real-time. Real-time surveys of transient astronomical sources are already reality both in the optical [2] and radio [3] domain, but the computational costs of non-periodic transient surveys are lower than for periodic sources like pulsars. Moreover, the most common technique to find periodic signals is to use the Fast Fourier Transform (FFT), but this algorithm requires the whole data set to be available for the computation, and this is not possible in a real-time scenario. Furthermore, processing the data in real-time is not the only challenge: the main challenge is to process the data in real-time and within a limited power budget. Therefore, real-time pulsar surveys are not a reality, yet.

We propose to use many-core accelerators and modern high-performance computing techniques to make finding pulsars in real-time a reality. To do so, we designed and developed a parallel pulsar searching pipeline, implemented using MPI and OpenCL. Our pipeline can run on different many-core accelerators, being them Graphics Processing Units (GPUs) or the Intel Xeon Phi, and traditional multi-core CPUs, and can be automatically tuned to adapt to a variety of telescopes and search parameters. The contributions of this paper are: (1) demonstrating that real-time pulsar searching is possible for real instruments, (2) showing the performance, scalability and power consumption of our pipeline, and (3) comparing the use of multi-core CPUs and many-core accelerators for pulsar searching for both performance and power consumption. We

believe that these contributions will remain relevant in the future because pulsar searching is a memory-bound problem, and memory bandwidth is not increasing as fast as the number of floating-point operations per second in modern hardware.

The remainder of this paper is organized as follows. Section II contains an overview of the relevant literature, while Section III introduces the basic concepts of pulsar searching, and provides a description of our real-time pipeline and all its computational kernels. The experiments, platforms and scenarios used to evaluate our pipeline are described in Section IV, while the experimental results are presented and analyzed in Section V. Finally, Section VI provides a further discussion on the results of the performed experiments, and Section VII summarizes our conclusions.

II. Related Work

As mentioned in Section I, new radio telescopes are making real-time transients search a necessity. An example is the Australian SKA Pathfinder (ASKAP) transient survey called CRAFT [3]. The pipeline described in [3] is designed to be implemented in hardware, but the paper mentions other research groups investigating the feasibility of software or FPGA based implementations; the paper also mentions the possible use of GPUs to implement the transient searching pipeline. Another real-time transient pipeline has been prototyped and tested at the BEST-2 radio telescope in Medicina [4]. This software transient searching pipeline is accelerated using GPUs, and uses the dedispersion algorithm described in [5]. Our pipeline differs from these previous works in two aspects: (1) we include a period search to also find pulsars and not only generic transient objects, and (2) we aim to design software that can be *automatically* adapted to different telescopes, instead of being specifically tailored for just one use case.

If real-time transient surveys are a reality, real-time pulsar surveys are much more difficult due to the period search that can no longer be implemented using a FFT. However, searching all the data produced by big pulsars surveys is a challenging task even when performed off-line. In 2010, the Einstein@Home (E@H) volunteer based distributed computing project has been used [6] to process previously collected data, producing the discovery of a new pulsar; the E@H distributed computing project had, at the time of publication, an aggregate computational power of 250 TFLOP/s. Additionally, even a traditional suite like DSPSR [7], a high-performance library that implements various algorithms used in pulsar astronomy, is being enhanced to exploit the computational power of many-core accelerators. Nonetheless, DSPSR does not yet support real-time pulsar searching.

III. Pulsar Searching

As introduced in Section I, there are three dimensions in the search space of a standard pulsars survey: (1) position, (2) distance and (3) period. To find new pulsars, all data received by the telescope needs to be processed for all the points in this three-dimensional search space. If there is indeed a signal buried in the data, then this process will enhance its SNR and make it visible. Of these three dimensions, in the rest of this paper we will focus only on distance and period, thus excluding the position in the sky from our discussion. The reason for excluding this dimension is that exploring it means to replicate the process described in this paper for multiple input data streams, called *beams*. While processing multiple beams causes a linear increase in the performance requirements of the search, these beams are completely independent from each other and can thus be processed in parallel. Because of this natural parallelism, and the fact that our pipeline can effortlessly be replicated for multiple independent beams, we decided to focus this paper on how to deal with distance and period.

An overview of our pipeline is presented in Figure 1; while this paper focuses only on our parallel implementation, general information on pulsar searching techniques can be found in [8] and [9]. The first step of the pipeline is *dedispersion*; dedispersion, described in more detail in Section III-A, is an algorithm used to reconstruct a signal assuming that it comes to our planet from a specific distance. Because this distance is one of the unknowns of the search process, the input is processed for a certain number of trial distances. In general, the higher the number of trial distances is, the higher is the chance of not leaving part of the search space unaccounted for. For *each* of these trial distances, dedispersion produces a new time series.

The following step in the pipeline is a matrix transpose that is used to rearrange the dedispersed time series in memory. The reason for this step is that the optimal memory layout for dedispersion is suboptimal for all the successive steps of our pipeline. After a series of performance experiments, we decided to add this step because the improvement in performance caused by the new memory layout by far exceeds the penalty of adding this step to the pipeline. Because matrix transposition is a well studied topic, and because this step is just an accessory part of the pipeline and not a fundamental step of pulsar searching, it will not be discussed further in this paper. We will, however, include this step in all performance measurements.

The transposed time series are then processed by two different kernels: a SNR computation and *folding*. The SNR computation, described in more detail in Section III-C, is used to analyze the dedispersed data to find non periodic sources and pulsars that are bright enough to be detected without a period search. The folding algorithm, described in more detail in Section III-B, is used to perform the period search, fundamental to discover weaker pulsars. In this stage, all the dedispersed time series are folded modulo a period; like for dedispersion, this is a brute-force search and a certain number of trial periods are used. After all these stages, the input signal has been processed for a variety of trial distances and periods; a final SNR computation, described in more detail in Section III-C, is performed to analyze the folded data and identify periodic signals.

All the pipeline's stages, excluding the last SNR computation, are executed for every second of the observation. For the

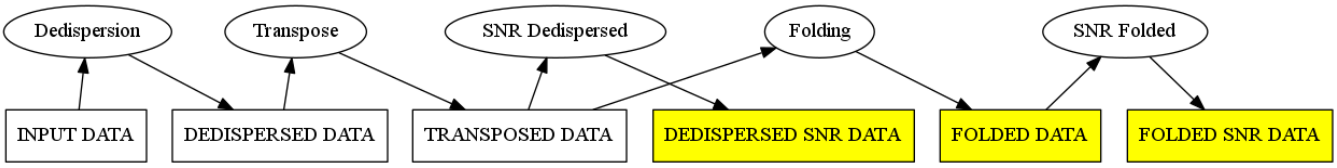

Fig. 1. A schematic representation of the pipeline. Ellipses represent computational kernels and boxes represent data. Yellow boxes are the output.

scope of this paper, the input data are transferred to the many-core accelerator for every second of the computation, while the output data structures are only transferred back to the host at the end of the search; all intermediate data structures are only present on the accelerator. Because our parallel framework of choice for the accelerators is OpenCL, in this paper we use the OpenCL nomenclature; in particular, with the word work-item we refer to a thread and with the word work-group we refer to a block of related threads executed by the same processor.

A. Dedispersion

Traveling from their emitting source all the way to Earth, signals are slowed down by interacting with free electrons in the inter-stellar medium (ISM). This process affects different frequencies of the same signal with an inverse relation: the lower the frequency is, the more it is slowed down by the ISM. Therefore, a delay is introduced between the time at which the different frequencies of a same signal are received by the telescope. The number of seconds of delay, Δ, for a given frequency f_i, and relative to the highest measured frequency f_h, is described by Equation 1; the frequency components of the equation are measured in MHz. In this equation, the *dispersion measure* (DM) represents the number of free electrons between the pulsar and the telescope, and it is linearly dependent with the distance between the two.

$$\Delta \approx 4,150 \times DM \times (\frac{1}{f_i^2} - \frac{1}{f_h^2}) \quad (1)$$

In order to reconstruct the original signal, this delay must be cancelled out by realigning all the different frequencies. This process of reconstructing the original signal is repeated for a number of DMs, each of them representing a different trial distance in the search space; as an example, the number of DMs that will be used in the pulsar surveys for the first phase of the SKA [10] is between 6,000 and 16,000. The time complexity of dedispersion is $O(d \times c \times s)$ where d is the number of trial DMs, c is the number of frequency channels that the input is divided into, and s is the number of samples per second. For additional details on the parallelization of this algorithm and its performance on many-core accelerators refer to [11].

B. Folding

In off-line pulsar searching pipelines, the period search is performed using an FFT, and the input signal is folded only modulo the periods identified by the search. The reason for this is that the complexity of the FFT is lower than the complexity of folding. However, to use the FFT for the period search the whole observation must be available, and in a real-time scenario there is no option but to save at most a few seconds of data, and process them before they are replaced by new incoming data. Because folding does not require us to save the whole observation, and it can be performed for each second of data without knowledge of past or future values, we decided to use it as the algorithm for period search in our real-time pipeline. Therefore, in our pipeline we fold all the dedispersed time series modulo a number of trial periods; the number of trial periods depends on the target of the observation, as pulsar periods may range from a few milliseconds to many seconds. The pseudocode of the folding algorithm is presented in Algorithm 1; the time complexity of the algorithm is $O(d \times p \times s)$ where d is the number of trial DMs, p is the number of trial periods, and s is the number of samples per second.

Algorithm 1 Pseudocode of the folding algorithm.

```
for dm = 0 → d do
  for period = 0 → p do
    for sample = 0 → s do
      bin = computeBin(period, sample)
      output[dm][period][bin] += input[sample][dm]
      counters[dm][period][bin] += 1
    end for
    output[dm][period][bin] /= counters[dm][period][bin]
  end for
end for
```

The algorithm operates on three data structures, one input $s \times d$ matrix, and two output $d \times p \times b$ arrays, where b is the number of *bins* in which a period is divided. This algorithm can be parallelized naturally along three different, and independent, dimensions: DMs, periods and bins. In our parallel implementation, an OpenCL work-group is a three-dimensional structure, in which the first dimension is associated with the DMs, the second with the periods and the third with the bins. Each work-item in a work-group is thus naturally associated with a $(DM, period, bin)$ triple and computes the output item associated with those coordinates. This parallelization scheme maps well to both input and output data structures, so that all memory accesses performed by consecutive work-items can be coalesced [12].

C. SNR Computation

In our pipeline there are two distinct kernels computing different SNR values: the first one works on dedispersed time series, while the second one works on folded time series. The function of both steps is to separate the points in the search

space where only noise is present from the points where a statistically significant signal may be present. Even if these two kernels work on different input and output data structures, they use the same formula for computing the SNR, so we can discuss them together. This formula is $(m-a)/r$, where m is the maximum value of the time series, a the mean and r the root mean square.

The first SNR computation works on the dedispersed time series, just after the transposition takes place; the input to this kernel is a $s \times d$ matrix, and the output is an array of d elements. The time complexity of this algorithm is $O(d \times s)$. The second SNR computation works on the folded time series, and differently from every other kernel of the pipeline is only executed *once* at the end of the computation. The input to this kernel is a $d \times p \times b$ three-dimensional array, and the output is a $d \times p$ matrix; the time complexity of this algorithm is $O(d \times p \times b)$. Their parallelization schemes are also similar, with the first kernel parallelized in the DM dimension, thus having each work-item associated with a particular DM, and the second kernel parallelized in both DM and period dimensions, so that each work-item is associated with a $(DM, period)$ pair.

D. Auto-Tuning

All the computational kernels described in this section expose configurable parameters to the user. By configuring these parameters the user can control various characteristics of the kernels, such as (1) the number of work-groups and work-items in a work-group, (2) the amount of work a single work-item is responsible for, (3) the amount of resources needed per work-item, (4) how programmable caches are used, (5) vectorization and (6) loop unrolling. These parameters do not only influence the parallelization of each kernel, but can also affect algorithmic properties like the *arithmetic intensity* (AI), i.e. the ratio between floating-point operations and bytes accessed in global memory, of a kernel. This is important because a higher AI is associated with higher attainable performance [13], especially for platforms like many-core accelerators where the gap between memory bandwidth and peak computational performance is so wide, and even more so for a memory-bound application like our pulsar searching pipeline.

However, we have no a priori knowledge about how to configure these parameters. Moreover, our goal is to execute our pulsar searching pipeline on different platforms and in different scenarios, without having to manually modify the algorithms. Therefore, we rely on auto-tuning to find the optimal configuration of each kernel's parameters, for each platform and on every scenario. To tune the kernels we execute them on each platform and for all the scenarios described in Section IV, measuring the achieved performance while varying all their parameters. We therefore explore, for each kernel, an optimization space that can consist of hundreds of configurations, even for a single platform-scenario combination. In the performance experiments presented in this paper we use, for each kernel of the pipeline, the optimal configuration found through the auto-tuning process.

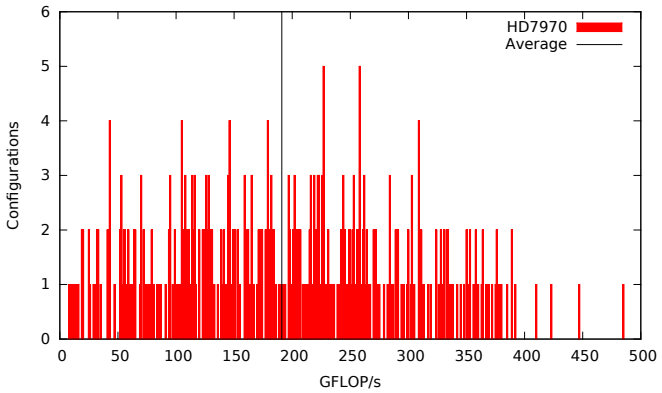

Fig. 2. Example of auto-tuning for the dedispersion kernel.

Platform	CEs	GFLOP/s	GB/s	Watt
AMD HD7970	64×32	3,788	264	250
NVIDIA K20	192×13	3,519	208	225
Intel Xeon Phi 5110P	2×60	2,022	320	225
Intel Xeon E5-2620	6×1	192	42	95

TABLE I
CHARACTERISTICS OF THE TESTED PLATFORMS.

To exemplify how much auto-tuning can affect performance, Figure 2 shows the performance histogram of dedispersion for one specific platform-scenario combination. What can be seen from this example is that finding the optimal configuration of a kernel without tuning is not trivial, because the optimum may lie far from the majority of configurations. Moreover, the performance obtained by using the optimal configuration can be much higher than the performance associated with any other configuration. An in-depth analysis of the tuning of the dedispersion kernel used in this paper can be found in [11]; the same process described in [11] has been applied in this paper to all the other kernels.

IV. EXPERIMENTAL SETUP

In this section we describe all the experiments that we performed in this paper, providing all the necessary information that can be used to replicate them. We first describe the factors common to all experiments, and then provide experiment-specific information in dedicated subsections. The many-core accelerators that we used to test our pipeline are described in Table I; the table provides, for each platform, the number of OpenCL *compute elements* (CEs), the maximum dissipated power, and the theoretical peak performance and memory bandwidth. Together with the three many-core accelerators, the two graphics processing units (GPUs) and the Xeon Phi, we also included the characteristics of a multi-core Intel CPU, the Xeon E5-2620, that we will use to provide a comparison between the accelerators and a more traditional platform.

The source code[1], is the same for all tested platforms, and is implemented in C++, using MPI and OpenCL for

[1]https://github.com/isazi/PulsarSearch

	Apertif	LOFAR	SKA1
Samples per Second	20,000	762	20,000
Center Frequency	1,425 MHz	142 MHz	800 MHz
Total Bandwidth	300 MHz	6.24 MHz	300 MHz
Channels	1,024	8,192	15,000
Channel Bandwidth	292 kHz	762 Hz	20 kHz
First DM	0 pc/cm^3	0 pc/cm^3	0 pc/cm^3
DM Step	0.03 pc/cm^3	0.145 pc/cm^3	0.009 pc/cm^3
First Period	1.6 ms	41 ms	1.6 ms
Period Step	1.3 ms	1.3 ms	1.3 ms
Bins	32	32	32

TABLE II
CHARACTERISTICS OF SCENARIOS AND SEARCH PARAMETERS.

the parallelization. The OpenCL runtime used for the AMD HD7970 GPU is the AMD APP SDK 2.9, the runtime used for the NVIDIA K20 GPU is CUDA 5.5, and the runtime used for the Intel Xeon Phi and CPU is the Intel OpenCL SDK 3.2.1; the C++ compiler is version 4.4.6 of the GCC. The accelerators are installed in computing nodes of the Distributed ASCI Supercomputer 4 (DAS-4 [2]); the DAS-4 uses CentOS version 6 as operating system, and version 2.6.32 of the Linux kernel. All the computing nodes are connected to *power distribution units* (PDUs) that we used to monitor the power consumption of the pipeline.

The experiments are performed in three different scenarios, two based on telescopes of the *Netherlands Institute for Radio Astronomy* (ASTRON), LOFAR [14] and the Apertif [15] system on Westerbork, and one based on the baseline design for the first phase of the internationally designed Square Kilometer Array [10], SKA1. The constant parameters of these three scenarios are listed in Table II. The first half of the table contains the parameters that define each scenario, while the second half contains the parameters associated with an hypothetical pulsar survey. Using different scenarios in the experiments is not only important to show the adaptability of our pipeline to different telescopes and surveys, but also because each different scenario stresses a different part of the pipeline. Moreover, by using scenarios based on the operational parameters of telescopes that are still under development, like Apertif or the SKA1, we aim at providing astronomers with insights into how to build the systems that will have to process the data of these telescopes. The variable component of each experiment will be the number of DMs and periods used for the search. Both DMs and periods will vary between the seven powers of 2 that range from 32 to 2,048; thus, the total number of tested instances per experiment will be 49 for each platform-scenario combination.

A. Pipeline Scalability

The first experiment consists in measuring the performance of our pulsar searching pipeline. The goal of this experiment is to test the feasibility of real-time pulsar searching, and show the scalability of our pipeline in terms of the number of DMs and periods. The pipeline is executed on every platform, for each scenario and every combination of DMs and periods; each of the pipeline's kernels use the optimal configuration, found through auto-tuning, for the specific test case. The metric used to measure performance is the execution time, measured in seconds and using software clocks, to complete the processing of a single beam of 60 seconds.

[2]http://www.cs.vu.nl/das4/

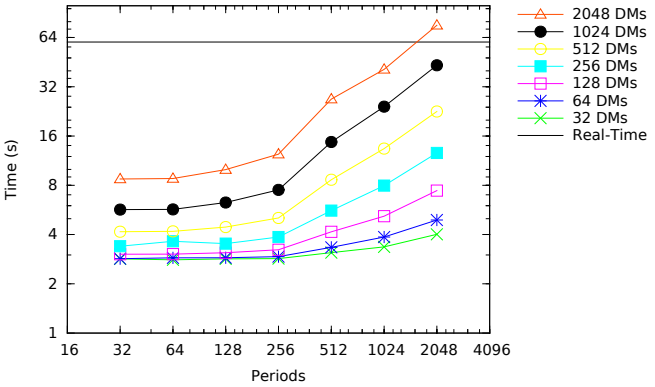

Fig. 3. Pipeline performance for the AMD HD7970, Apertif scenario.

B. Power Consumption

The second experiment consists in measuring the power consumed by our pulsar pipeline during its execution. The goal of this experiment is to identify the more power efficient platform for real-time pulsar searching. The pipeline is executed on every platform, for each scenario and every combination of DMs and periods; the same optimal configurations used for the experiment described in Section IV-A are used. Because the granularity of the PDUs is a whole compute node, the measured values are normalized using the power consumption of the same node in an idle state; in this way we can take into account just the power consumed by the pipeline. The metric used in this experiment is the total consumed power, measured in kW using the DAS-4 PDUs, used by the pipeline to process a single beam of 60 seconds.

V. RESULTS

In this section, we present the results of the experiments described in Section IV. For each experiment we first introduce the results, and then provide an analysis of them. We conclude each experiment with a summary of the findings.

A. Pipeline Scalability

The first experiment, described in Section IV-A, aims at measuring the performance of the pipeline to understand the feasibility of real-time pulsar searching and the scalability of the pipeline in terms of DMs and periods. All figures in this section are log-log plots and each line represents a different number of DMs; a black line, labeled "Real-Time", represents the threshold over which the execution time of the pipeline exceeds the observation time (i.e. 60 seconds) thus breaking the real-time constraint. We start presenting the results obtained with the first platform, the AMD HD7970 GPU.

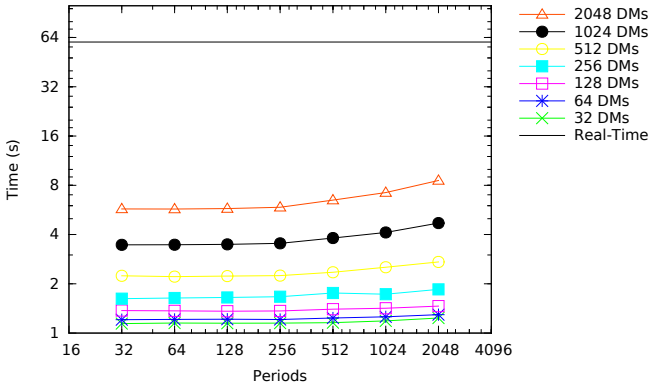

Fig. 4. Pipeline performance for the AMD HD7970, LOFAR scenario.

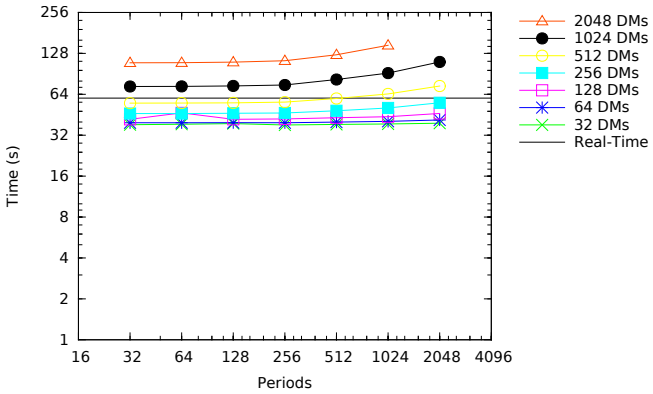

Fig. 5. Pipeline performance for the AMD HD7970, SKA1 scenario.

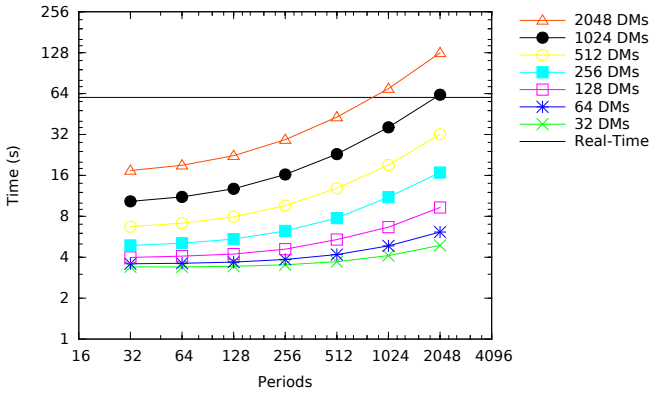

Fig. 6. Pipeline performance for the NVIDIA K20, Apertif scenario.

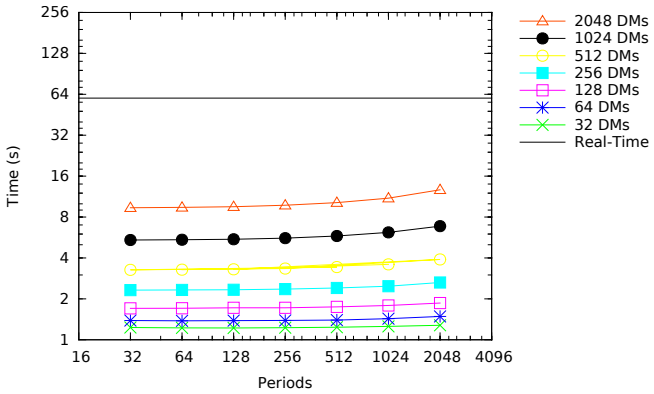

Fig. 7. Pipeline performance for the NVIDIA K20, LOFAR scenario.

Figure 3 presents the results of the Apertif setup. In this scenario, the pipeline achieves real-time performance in all but one of the test cases. The pipeline scales better than linearly for all DMs until the threshold of 256 periods, because there are still resources available for the computation; after this point, however, the resources available to the GPU are saturated and the pipeline scales linearly in the number of periods. While increasing the number of periods affects the pipeline's scalability, increasing the number of DMs has a lower impact on performance. As a matter of fact, the pipeline scales better than linearly in the number of DMs for most of the test cases, approaching linear scalability only for the bigger ones.

In the LOFAR scenario, presented in Figure 4, performance requirements are lower, and the pipeline satisfies the real-time constraint in all test cases. In fact, even the biggest of test cases, the $2,048 \times 2,048$ case, is 7 times faster than real-time. Achieving faster than real-time performance means that the pipeline can scale to even bigger search spaces, or process the data streams associated with multiple beams. Therefore, it would be possible to process 7 beams per GPU and still satisfy the real-time constraint even for the biggest of test cases. Furthermore, the pipeline scales better than linearly in both the number of DMs and periods.

Figure 5 introduces the results for the most computationally challenging scenario, SKA1; one of the data points, the one associated with the $2,048 \times 2,048$ case, is missing due to the platform's memory limitations. In this scenario, most of the test cases satisfy the real-time constraint, but only up to 512 DMs. After this point, all results lie over the real-time threshold; in correspondence with the bigger computed test case, the pipeline is 2.4 times slower than real-time. If in the LOFAR scenario better than real-time results indicated that the pipeline could scale to even bigger search spaces, pulsar surveys in the SKA1 scenario would have to be distributed: this can be done by replicating the data streams to multiple accelerators, and executing the pipeline on subsets of the global search space. However, better than linear scalability is achieved in both the number of DMs and periods also for the SKA1 scenario.

The second platform we tested is the NVIDIA K20 GPU; Figure 6 presents the results of the Apertif scenario. Like for the previous platform, we notice that almost all test cases satisfy the real-time constraint. The pipeline's execution time is higher on the K20 than on the HD7970, but the scalability is better. In fact, while scalability in both dimensions is better or close to linear, the figure shows that this platform scales more smoothly than the previous one in the number of periods.

While in the Apertif scenario we can identify differences in

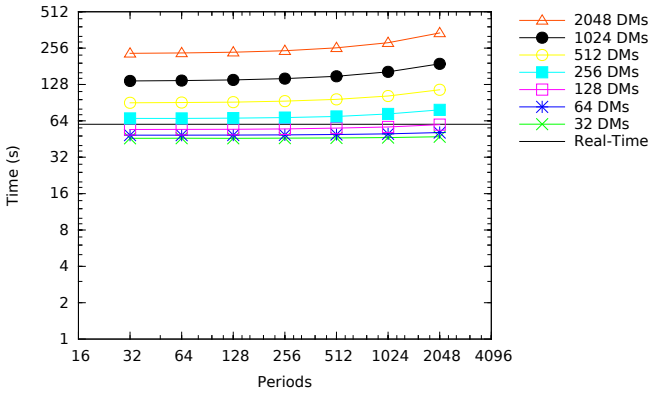

Fig. 8. Pipeline performance for the NVIDIA K20, SKA1 scenario.

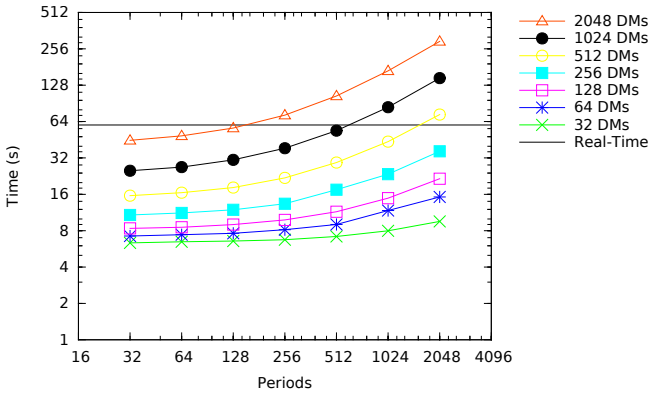

Fig. 9. Pipeline performance for the Intel Xeon Phi, Apertif scenario.

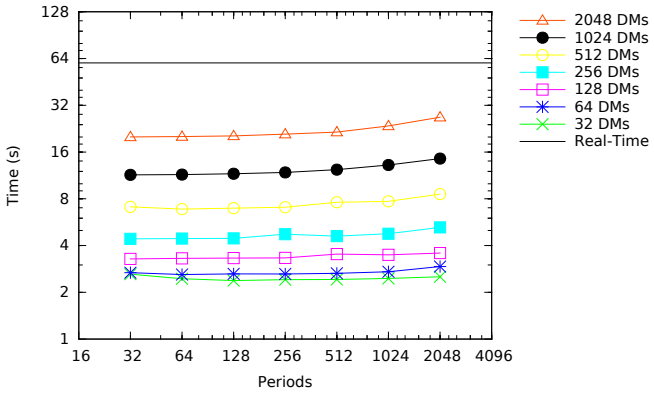

Fig. 10. Pipeline performance for the Intel Xeon Phi, LOFAR scenario.

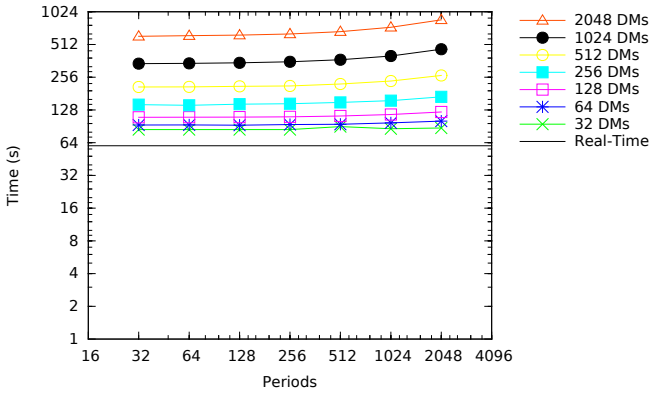

Fig. 11. Pipeline performance for the Intel Xeon Phi, SKA1 scenario.

how the two GPUs scale, Figure 7 shows that results in the LOFAR scenario are almost identical. The real-time constraint is satisfied for all test cases, and in the biggest one the pipeline is 4.7 times faster than real-time. The pipeline scales better than linearly in the number of periods, and linearly or better in the number of DMs.

The SKA1 scenario is presented in Figure 8, and the first noticeable result is that most of the test cases do not satisfy the real-time constraint. In fact, starting from 256 DMs, no line lies below the real-time threshold. The presence of some lines below the threshold suggests that SKA1 pulsar surveys will have to be distributed among multiple K20 GPUs in the DM dimension. Although it is not always possible for the K20 to satisfy the real-time constraint in this scenario, the achieved scalability is better than linear in both the number of DMs and periods.

The last accelerator we tested is the Intel Xeon Phi. Figure 9 shows the performance results for this accelerator in the Apertif scenario. Most of the test cases still lie below the real-time threshold, but the number of instances that do not satisfy this requirement is higher than for the GPUs, with the biggest test cases executing 4.9 times slower than real-time. The execution time, compared with the GPUs, is also higher for all test cases. The pipeline's scalability is better than or equal to linear, and the observed trend is similar to the one of the NVIDIA GPU.

The results for the LOFAR scenario are presented in Figure 10. In this scenario, as seen for the other platforms, all test cases satisfy the real-time constraint, even if performance is lower on the Phi than on the GPUs; the biggest test case is 2.2 times faster than real-time. Scalability in both the number of DMs and periods is better than linear.

Results for the last scenario, SKA1, are presented in Figure 11. Performance obtained in this scenario by the Xeon Phi is too low to satisfy the real-time constraint in any of the test cases, with the biggest one resulting 14 times slower than real-time, and the smallest one still 1.4 times slower. Achieving real-time performance in this scenario, even if theoretically possible by computing different DMs on different Xeon Phis, would require too many devices to be feasible in practice. Like for the other platforms, scalability in both the number of DMs and periods is better than linear in this scenario.

To summarize the results of this experiment, all platforms achieve linear or better than linear scalability in all scenarios in both the number of DMs and periods. The performance achieved by each platform is, however, different, with the HD7970 GPU resulting the fastest among the platforms we tested; both GPUs performed better than the Xeon Phi in all

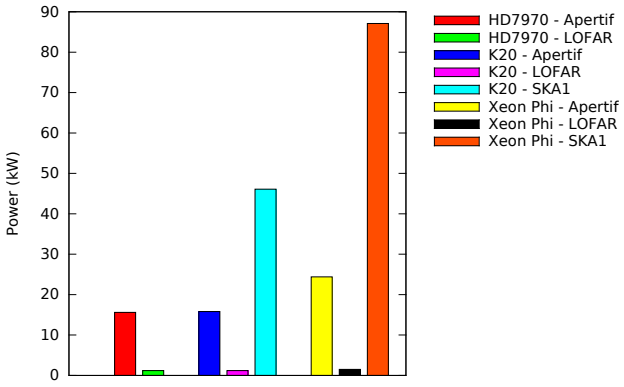

Fig. 12. Total consumed power, $2,048 \times 2,048$ case.

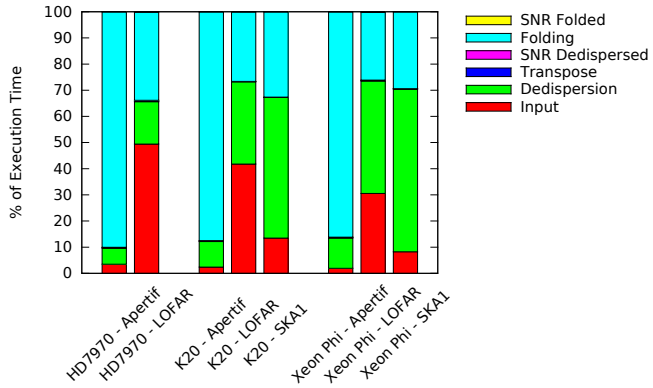

Fig. 13. Performance breakdown, $2,048 \times 2,048$ case.

scenarios. Although the scenarios are different in their performance requirements, the results of this experiment show that our pipeline would be able to satisfy the real-time requirement for all of them by distributing the pipeline over multiple many-core accelerators where performance on a single accelerator would not be enough. Therefore, we can conclude that real-time pulsar searching could already be a reality for current radio telescopes, and it will enable future telescopes to perform bigger and more accurate surveys.

B. Power Consumption

In this section we present the results of the experiment described in Section IV-B. Due to space limitations, we will not present the results for all the test cases like we did in Section V-A, but use a specific test case to highlight the results. The test case we selected is the biggest one, with 2,048 DMs and periods. We believe that it makes sense to use this particular test case because we know, from the results previously presented, that it is the test case with the higher execution time in each of the scenarios. Therefore, even if not representative of every other test case, it still provides an upper bound on the power consumed by the pipeline running on our three many-core platforms.

The results are presented in Figure 12; because the AMD HD7970 was not able to execute this test case in the SKA1 scenario due to memory limitations, the value is missing from the figure. However, the almost identical results obtained by the two GPUs in the Apertif and LOFAR scenarios let us believe that the total power consumption would be similar also for the SKA1 scenario. The two GPUs require 15 and 1.2 kW to execute the pipeline for the Apertif and LOFAR scenario, respectively. Because we know from Section V-A that the execution time of the two platforms is different, we can conclude that the energy per second consumed by the K20 is lower than the energy per second consumed by the HD7970. It is also clear, from these results, that the more computationally expensive the scenario is, the more power is required for the computation, which is what we expect.

The Xeon Phi consumes more total power executing the pipeline, but this is caused only by the higher execution times and not by a higher amount of energy consumed per second. In fact, the Xeon Phi is the accelerator that, per second, consumes less energy to execute the pulsar searching pipeline. Comparing the power consumption between the GPUs and the Phi, the Phi uses only between 1.2 and 1.8 more power to complete the execution of the pipeline, while being on average 2.5 times slower than the GPUs.

To summarize the results of this experiment, the power required during the execution of the pipeline is, in all cases, sensibly lower than the thermal design power of the accelerators. Among the GPUs, the K20 is less power hungry than the HD7970, but because of the higher execution time requires the same total power as the AMD GPU to execute the pipeline. The accelerator that consumes less energy per second is the Xeon Phi, however the performance gap between it and the GPUs is too wide and this platform ends up consuming more power, in total, than the other two accelerators.

VI. Discussion

In this section we provide further analysis of the results presented in Section V, and introduce additional data to complement the experiments already presented. We start by analyzing the differences between the three scenarios that we used for our experiments. Figure 13 provides a breakdown of the percentage of time spent on each of the pipeline stages during the execution; for simplicity, and due to space limitations, we only show the biggest test case. The first result is that, of all of the pipeline stages, only three determine the total execution time: input handling, dedispersion and folding. All the other kernels (i.e. the transpose and the two SNR computations) combined, account only for few percentage points of the total execution time. Therefore, any future performance improvement of this pipeline will have to focus on the three main stages.

Another interesting result is that the relative impact of the three main components of the pipeline on the total execution time differs by scenario, but not so much by platform. In fact, we can see the same pattern for each platform: in the Apertif scenario folding dominates performance, while handling and transferring the input to the accelerator determines

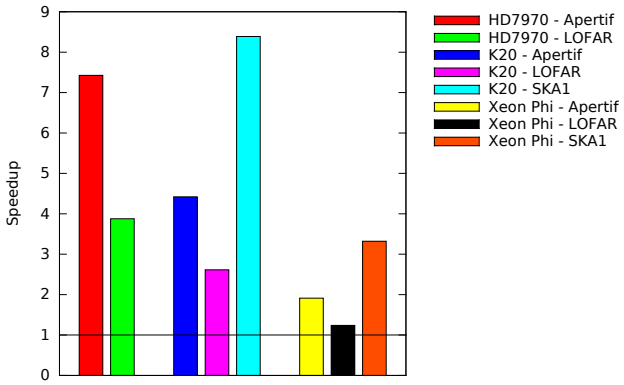

Fig. 14. Speedup over CPU, $2,048 \times 2,048$ case.

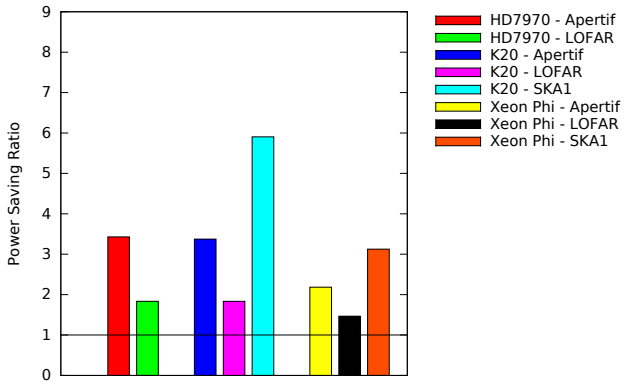

Fig. 15. Power saving ratio over CPU, $2,048 \times 2,048$ case.

performance for the LOFAR scenario, and dedispersion is the main contributor to the SKA1 execution time. These different scenarios were chosen because, for their characteristics, they would stress different aspects of the pipeline, however they are modeled on the operational parameters of real telescopes, showing that different instruments require focus on different stages of the same pipeline. In this context, we believe that having a pipeline where each kernel can be tuned and optimized *automatically* for each particular scenario is extremely important to perform real-time pulsar surveys.

Figure 13 provides also an explanation for the scalability trends observed in Section V-A. In the Apertif scenario, performance results for all three platforms showed that the pipeline scaled better than or close to linearly in the number of DMs, but only linearly in the number of periods, and this is because folding dominates performance in this scenario. Similarly, the pipeline scales better than linearly in the number of periods for both LOFAR and SKA1 because folding is not the main performance driver in these scenarios. Because input handling dominates the LOFAR scenario, the pipeline scales better than linearly in both dimensions, but will scale linearly with an even bigger DM space. As for the SKA1 scenario, it will also approach linear scalability with more DMs to search, as dedispersion is the main factor affecting the pipeline's performance in this scenario.

Figure 14 provides the speedup achieved by our pipeline on many-core accelerators, compared with the same pipeline tuned and running on a multi-core CPU, the Intel Xeon E5-2620; the characteristic of this platform are also presented in Table I. Also for this experiment, only the biggest test case is presented. The achieved speedup ranges between 1.2 and 8.3, with the highest speedup achieved in the SKA1 scenario, and the lowest one in the LOFAR scenario. This result is useful to highlight how many-core accelerators contribute to achieve real-time performance in pulsar searching, especially in the context of future telescopes like the SKA. Achieving high-performance means also that less hardware is necessary to process the same amount of data, lowering the costs associated with maintaining and operating the systems used for this task. Even if higher speedups are possible for individual kernels (e.g. for dedispersion [11]), obtaining higher speedups is more difficult for the pipeline as a whole as performance is always determined by the slowest kernel.

By analyzing Figure 15 it can be seen that many-core accelerators also require less power than using CPU. Comparing Figure 14 and 15 we see that the GPUs reduce their advantage over the multi-core CPU, as their energy consumption is much higher, although they are still 1.8–5.9 more power efficient. While the GPUs reduce their gain over the CPU when power consumption is taken into account, the Xeon Phi maintains its advantage, and achieves the same power efficiency and speedup when compared to the tested CPU. This is because the measured energy consumed per second while executing the platform is almost the same for the two Intel devices, but the execution time is lower for the Phi. Overall, many-core accelerators are a viable platform for the execution of a real-time pulsar searching pipeline, both from performance and energy efficiency perspectives.

These results can also be used to estimate the number of accelerators that we would need today to build a pulsar searching system for the first phase of the SKA. From the SKA1 baseline design [10] we know that, in the operational mode that we used for our experiments, a pulsar survey will explore a search space of 2,222 beams and 16,113 DMs; using the best performing accelerator, the AMD HD7970, we would need 63 GPUs per beam to dedisperse all the DMs and search to up to 2,048 periods. Therefore, the total number of GPUs to process all beams would be nearly 140,000, and this would require more than 30 MW of power just for pulsar searching. However, the number of GPUs necessary could be reduced down to only 20 per beam by having the input already available in the accelerator memory, without having to transfer it just for this pipeline. In this case, the total number of GPUs would be reduced to less than 44,500 and the power required for the search to 9.5 MW. Future improvements in the power efficiency of many-core accelerators will help reduce even more the power budget for the SKA.

VII. Conclusions

In this paper we introduced a new pulsar searching pipeline. All the stages of this pipeline are parallel, implemented using OpenCL, and can run on a variety of many-core accelerators. We use OpenCL to achieve code portability between different platforms, but we also provide adaptability to different telescopes and search parameters. To adapt the code, we exploit auto-tuning, finding the optimal configuration for each of the pipeline kernels in every specific scenario. The pipeline can also be distributed over multiple nodes using MPI, thus further increasing its scalability.

The main contribution of this paper is to show the feasibility of a real-time pulsar searching pipeline, as new radio telescopes will force a shift from the traditional off-line processing to a more challenging real-time scenario, and we showed in this paper that this is possible even using currently available hardware. We showed in Section V-A that, in different scenarios and using different platforms for its execution, our pipeline is able to process data in real-time, i.e. to process one second of input data in less than one second. In cases where we cannot show real-time performance using a single accelerator, we have the possibility to distribute the search over multiple nodes, thus satisfying the real-time constraint.

The pulsar pipeline introduced in this paper does not only achieve real-time performance, but shows linear or better than linear scalability in all search dimensions: number of DMs and periods. This is another important result, because it allows astronomers to easily predict the number of accelerators needed for a specific search, and to minimize the number of necessary accelerators. This allows for smaller and less expensive systems, and it helps keeping under control the power required to operate them.

In Section VI we showed that using many-core accelerators provides good speedup over multi-core CPUs, with GPUs being 2–8 times faster than an Intel Xeon CPU. Not only are many-core accelerators faster than a traditional CPU in executing our pulsar searching pipeline, but they also consume 1.8–5.9 less power in our tested scenarios, another characteristic that makes these platforms the most suitable for pulsar searching in the SKA era.

In the future, we plan to test our proposed pipeline on newer many-core accelerators. In particular, we are interested in measuring the impact that the introduction of 3D stacked memory will have on this pipeline, considering that all its most time consuming algorithms are memory-bound. Furthermore, we plan to add the possibility to perform acceleration search, thus being able to more easily detect pulsars in binary systems.

References

[1] P. C. Broekema, R. V. van Nieuwpoort, and H. E. Bal, "Exascale high performance computing in the square kilometer array," in *Proceedings of the 2012 Workshop on High-Performance Computing for Astronomy*. New York, NY, USA: ACM, 2012, pp. 9–16.

[2] A. J. Drake, S. G. Djorgovski, A. Mahabal, E. Beshore, S. Larson, M. J. Graham, R. Williams, E. Christensen, M. Catelan, A. Boattini, A. Gibbs, R. Hill, and R. Kowalski, "First results from the catalina real-time transient survey," *The Astrophysical Journal*, vol. 696, no. 1, p. 870, 2009.

[3] J.-P. Macquart, M. Bailes, N. D. R. Bhat, G. C. Bower, J. D. Bunton, S. Chatterjee, T. Colegate, J. M. Cordes, L. D'Addario, A. Deller, R. Dodson, R. Fender, K. Haines, P. Halll, C. Harris, A. Hotan, S. Jonston, D. L. Jones, M. Keith, J. Y. Koay, T. J. W. Lazio, W. Majid, T. Murphy, R. Navarro, C. Phillips, P. Quinn, R. A. Preston, B. Stansby, I. Stairs, B. Stappers, L. Staveley-Smith, S. Tingay, D. Thompson, W. van Straten, K. Wagstaff, M. Warren, R. Wayth, and L. Wen (the CRAFT Collaboration), "The Commensal Real-Time ASKAP Fast-Transients (CRAFT) Survey," *Pub. Astr. Soc. Australia*, vol. 27, no. 3, pp. 272–282, 2010.

[4] A. Magro, J. Hickish, and K. Z. Adami, "Multibeam Gpu Transient Pipeline for the Medicina BEST-2 Array," *Journal of Astronomical Instrumentation*, 2013.

[5] W. Armour, A. Karastergiou, M. Giles, C. Williams, A. Magro, K. Zagkouris, S. Roberts, S. Salvini, F. Dulwich, and B. Mort, "A GPU-based Survey for Millisecond Radio Transients Using ARTEMIS," in *Astronomical Data Analysis Software and Systems XXI*, ser. Astronomical Society of the Pacific Conference Series, P. Ballester, D. Egret, and N. P. F. Lorente, Eds., vol. 461, Sep. 2012, p. 33.

[6] B. Knispel, B. Allen, J. M. Cordes, J. S. Deneva, D. Anderson, C. Aulbert, N. D. R. Bhat, O. Bock, S. Bogdanov, A. Brazier, F. Camilo, D. J. Champion, S. Chatterjee, F. Crawford, P. B. Demorest, H. Fehrmann, P. C. C. Freire, M. E. Gonzalez, D. Hammer, J. W. T. Hessels, F. A. Jenet, L. Kasian, V. M. Kaspi, M. Kramer, P. Lazarus, J. van Leeuwen, D. R. Lorimer, A. G. Lyne, B. Machenschalk, M. A. McLaughlin, C. Messenger, D. J. Nice, M. A. Papa, H. J. Pletsch, R. Prix, S. M. Ransom, X. Siemens, I. H. Stairs, B. W. Stappers, K. Stovall, and A. Venkataraman, "Pulsar discovery by global volunteer computing," *Science*, vol. 329, no. 5997, p. 1305, 2010.

[7] W. van Straten and M. Bailes, "DSPSR: Digital Signal Processing Software for Pulsar Astronomy," *PASA - Publications of the Astronomical Society of Australia*, vol. 28, pp. 1–14, 2011.

[8] J. M. Cordes, "Pulsar Observations I. – Propagation Effects, Searching Distance Estimates, Scintillations and VLBI," in *Single-Dish Radio Astronomy: Techniques and Applications*, ser. Astronomical Society of the Pacific Conference Series, S. Stanimirovic, D. Altschuler, P. Goldsmith, and C. Salter, Eds., vol. 278, Dec. 2002, pp. 227–250.

[9] D. Lorimer and M. Kramer, *Handbook of pulsar astronomy*. Cambridge Univ Pr, 2005, vol. 4.

[10] P. Dewdney, W. Turner, R. Millenaar, R. McCool, J. Lazio, and T. Cornwell, "SKA1 system baseline design," Tech. Rep., 2013.

[11] A. Sclocco, H. E. Bal, J. Hessels, J. van Leeuwen, and R. V. van Nieuwpoort, "Auto-tuning dedispersion for many-core accelerators," *International Parallel and Distributed Processing Symposium (IPDPS)*, 2014.

[12] J. W. Davidson and S. Jinturkar, "Memory access coalescing: a technique for eliminating redundant memory accesses," *SIGPLAN Not.*, vol. 29, no. 6, pp. 186–195, Jun. 1994.

[13] S. Williams, A. Waterman, and D. Patterson, "Roofline: an insightful visual performance model for multicore architectures," *Commun. ACM*, vol. 52, pp. 65–76, April 2009.

[14] M. de Vos, A. Gunst, and R. Nijboer, "The LOFAR telescope: System architecture and signal processing," *Proceedings of the IEEE*, vol. 97, no. 8, pp. 1431–1437, 2009.

[15] M. A. W. Verheijen, T. A. Oosterloo, W. A. van Cappellen, L. Bakker, M. V. Ivashina, and J. M. van der Hulst, "Apertif, a focal plane array for the wsrt," *AIP Conference Proceedings*, vol. 1035, no. 1, pp. 265–271, 2008.