A Real-Time Radio Transient Pipeline for ARTS

Alessio Sclocco, Joeri van Leeuwen
ASTRON
Netherlands Institute for Radio Astronomy
Dwingeloo, the Netherlands
sclocco@astron.nl, leeuwen@astron.nl

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

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

***Abstract*—ARTS is a new instrument designed to discover transient astronomical sources in the radio spectrum. To operate, it will require a high-performance radio transient pipeline, capable of processing hundreds of data streams in real-time, at a data rate of 36 GB/s. What makes the processing of these data streams challenging is the fact that radio signals received from space are dispersed by the interaction with the inter-stellar medium, and dispersion is a function of the distance between source and receiver. Because the distance of yet to discover objects is not known in advance, every data stream needs to be processed for thousands of trial distances, an extremely time consuming process.**

In this paper we introduce and describe a prototype for the ARTS radio transient pipeline. Our proposed pipeline is highly parallel and uses Graphics Processing Units as accelerators for its computational kernels. We test the pipeline on two different and widely used GPUs, the HD7970 from AMD and the K20X from NVIDIA, and show linear scalability and real-time performance on both. Using these performance results, we provide an estimate on the size of the system necessary to implement ARTS, and a lower bound on its power consumption. The results of this paper are also relevant in the context of designing the Square Kilometer Array.

***Index Terms*—radio astronomy; graphics processing units; dedispersion**

I. Introduction

ARTS, the Apertif Radio Transient System, is a new instrument designed to survey the sky for transient radio sources, such as pulsars [1] or fast radio bursts [2]. To find new transients ARTS will process, in real-time, 444 different streams of input data received from the Apertif instrument on Westerbork [3]. Real-time processing is not just technically necessary to cope with an input data rate of more than 36 GB/s, but it is also important to allow astronomers to save a copy of the recorded data at the highest possible resolution, and to trigger follow-up observations in other frequency bandwidths or with other instruments. However, detecting radio transients in real-time, and at such high data rates, is not trivial.

In principle, detecting a transient object is a straightforward process that consists in simply monitoring a stream of data looking for a peak with high enough signal-to-noise ratio (SNR). Even considering the need to perform this process in real-time, the computational requirements associated with it would still be low. Unfortunately, electromagnetic signals generated in space interact with external factors, and what is captured on Earth looks different from the original signal. In particular, the main challenge in a radio transient pipeline is to revert the effects of dispersion. Dispersion is caused by the interaction between the emitted signal and the free electrons in the inter-stellar medium, and results in the different frequencies composing a signal traveling at different relative velocities. Therefore, what was originally a detectable peak with high SNR, becomes a scattered series of small peaks, easily drawn in the background noise. To detect the signal it is thus necessary to process the received data and reconstruct the original signal, in a process called *dedispersion*.

Because the effects of dispersion are bigger the farther the emitting source is from the receiver, and because in a search for new objects the distance is not known a priori, each input stream must be processed and reconstructed for a large number of possible trial distances. This brute-force search is what makes real-time transient search a difficult and interesting problem. A way to speedup this search is to process the different trial distances, and the different input streams, in parallel. Given that we already used Graphics Processing Units (GPUs) to accelerate dedispersion [4], and that these accelerators have been used for similar, though less computationally demanding, pipelines [5] [6], in this paper we introduce a prototype for a GPU accelerated, real-time radio transient pipeline for ARTS.

To summarize our contributions, in this paper we: (1) introduce a new parallel and GPU accelerated radio transient pipeline, (2) show that our proposed pipeline achieves real-time performance and linear scalability, and (3) estimate the size of the ARTS system and a lower bound on its power consumption. Although our focus is on ARTS, the Square Kilometer Array (SKA) will also need a high-performance transient pipeline, one able to cope with an estimated data rate of more than 10 Pb/s [7]. Our pipeline can also be seen as a first step into addressing the challenges of the SKA.

II. The Radio Transient Pipeline

As highlighted in Section I, to find new transient sources in the data captured by Apertif, ARTS will process, in real-time, the data of 444 input streams, or *beams*. Figure 1 contains an overview of our proposed pipeline; the whole pipeline is executed for each second of data in each of the 444 beams. Although we focus on describing our specific solution to the problem of finding radio transients in real-time, the main computational kernels of this pipeline are generic and can be used in other pipelines as well.

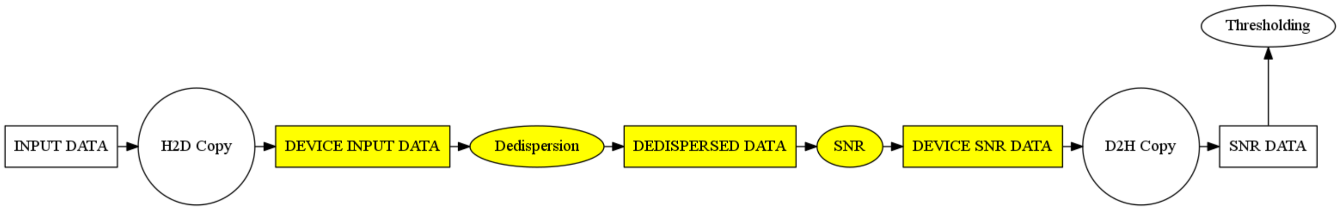

Fig. 1. Overview of the radio transient pipeline. In the figure, ellipses represent computational kernels, circles represent memory transfers and boxes data structures. Yellow objects are stored or executed on the GPU, white objects on the host.

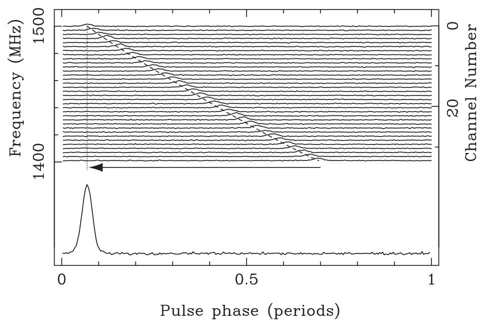

Fig. 2. The effect of dispersion on a pulsating signal, courtesy of Lorimer and Kramer [1].

The main computational kernels of this pipeline are called dedispersion, SNR and thresholding. Of these three kernels, dedispersion and SNR are executed on the GPU and thresholding is executed on the host. Data are copied between host and device in two distinct memory operations; the first operation transfers the input time series on the GPU, while the second operation transfers the computed SNR values on the host for further processing. Intermediate data produced and consumed on the GPU are kept in device memory and not transferred back and forth.

The first, and more computationally intensive, kernel of our pipeline is dedispersion. As already mentioned in Section I, dedispersion is the algorithm used to reverse the effects of dispersion. Because of dispersion, the different frequencies of the signal emitted by a radio transient source are scattered in time, causing a significant reduction in the signal's SNR; the effect of dispersion on a pulsating signal is exemplified by Figure 2. To reverse this effect, the different frequencies of a time series are shifted in time, relatively to each other, and realigned, before being summed together to recreate the original signal. Assuming that this shift is known, the number of operations necessary to *dedisperse* one second of data is $O(c \times s)$, where c is the number of different frequencies, called *channels*, the input data is divided into, and s is the number of samples per second. However, in the search for new transients the shift is not known in advance, thus a large number of different shifts are tried in what is known as a brute-force search; the different trial values are called *Dispersion Measures* (DMs). Therefore, in a survey with d DMs, the computational cost of dedispersing one second of data is $O(c \times s \times d)$ floating point operations per beam. If for ARTS this means a required throughput of 40 GFLOP/s per beam, and 18 TFLOP/s in total, what it means for the SKA is a required throughput of 4 TFLOP/s per beam, and 10 PFLOP/s in total. Moreover, dedispersion is a memory-bound algorithm and this makes almost impossible to achieve peak performance on any platform. The design, parallelization and implementation details of the used dedispersion algorithm are available in [4], thus we are not going to discuss them further in this paper.

The next computational kernel in our pipeline is called SNR. In this step, each of the d dedispersed time series generated by the previous kernel is processed, independently, to compute the SNR of the sample with highest intensity. The SNR of $\max$, i.e. the sample with highest intensity in the time series, is defined as $(\max - \mu)/\sigma$, where μ is the mean value of the time series and σ the standard deviation. These statistical properties of each time series are computed, in parallel and on the GPU, using Chan's algorithm [8]. In our kernel, each dedispersed time series is assigned to a different block of threads. Inside a block, each thread computes maximum, mean and standard deviation of a subset of samples, and then combines these intermediate results with the one computed by the other threads to produce the final SNR value associated with a given DM.

The last computational kernel of our pipeline is thresholding. In this kernel, the d elements array containing the SNR values is scanned to look for values that exceed a certain user specified threshold. If a value exceeds this threshold, an entry is added to the pipeline output containing the id of the DM associated with this value, and a time stamp. Before thresholding takes place, the array containing the SNR values computed by the previous kernel is copied back to the host. This is because thresholding, not benefiting from massive parallelization, is the only computational kernel of our pipeline that is not executed by an accelerator but by the host.

Our discussion so far covers the steps necessary to process a single beam, but our pipeline natively supports the processing of multiple beams. Because different beams are completely independent from each other, and can thus be independently processed, we decided to execute one instance of the pipeline for each beam. The pipeline can either be executed sequentially, one beam after the other, or in parallel, with different instances of the pipeline running at the same time, each instance associated with a different beam. Although computing different beams in parallel makes it possible to overlap the

Platform	Cores	GFLOP/s	GB/s	Watt
AMD HD7970	64×32	3,788	264	250
NVIDIA K20X	192×14	3,935	250	235

TABLE I
CHARACTERISTICS OF THE TESTED PLATFORMS.

Samples per Second	20,000
Center Frequency	1,425 MHz
Total Bandwidth	300 MHz
Channels	1,024
Channel Bandwidth	292 kHz
First DM	0 pc/cm^3
DM Step	0.03 pc/cm^3

TABLE II
OBSERVATIONAL PARAMETERS USED FOR THE EXPERIMENT.

memory transfers with the execution of the computational kernels, experiments showed that this strategy does not lead to better performance on all platforms, due to driver issues. Therefore, we decided to support both modes in the pipeline, leaving the final choice to the user.

III. EXPERIMENTAL SETUP

To test both performance and scalability of our pipeline, we measure its execution time on two different GPUs: an AMD HD7970 and a NVIDIA K20X. The main characteristics of these two platforms are described in Table I; in particular, the table shows the number of cores that each GPU has, the theoretical peaks for single precision floating point operations per second and memory bandwidth, and the thermal design power (TDP).

The source code of our pipeline, the same for the two GPUs, is implemented in C++ and OpenCL, with OpenCL used to parallelize the computational kernels executed on the accelerators. The OpenCL runtime used for the AMD HD7970 is the AMD APP SDK 2.9, and the runtime used for the NVIDIA K20X is CUDA 5.5; the C++ compiler is version 4.9.0 of the GCC. The two GPUs are installed in different computing nodes of the Distributed ASCI Supercomputer 4 (DAS-4); DAS-4 uses CentOS version 6 as operating system, and version 2.6.32 of the Linux kernel.

The observational parameters used in this experiment are described in Table II; they have been chosen to realistically represent one of the configurations in which ARTS will operate. The OpenCL kernels were automatically tuned for both GPUs and for this specific scenario. This tuning is necessary to achieve better performance, and to provide a more fair comparison between the two devices; an in depth description of the auto-tuning process for dedispersion can be found in [4].

The pipeline is executed on each platform varying the number of trial DMs and the number of beams; for simplicity, the used values are the powers of two between 2^5 and 2^{11} for the DMs, and between 2^0 and 2^3 for the beams. Each run of the experiment involves the processing of 60 seconds of synthetically generated data; the total execution time and the execution time of each of the pipeline's steps are measured using software timers.

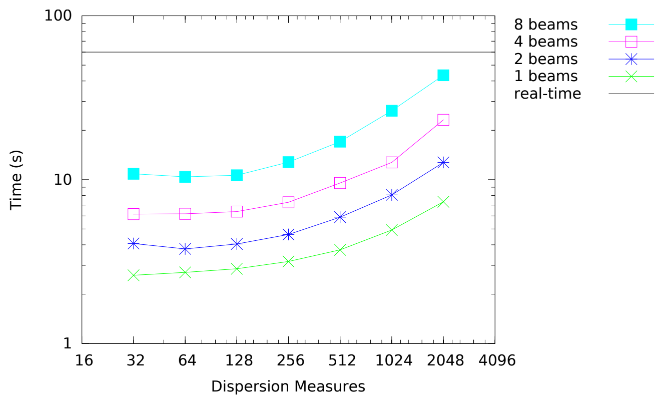

Fig. 3. Pipeline performance on the HD7970.

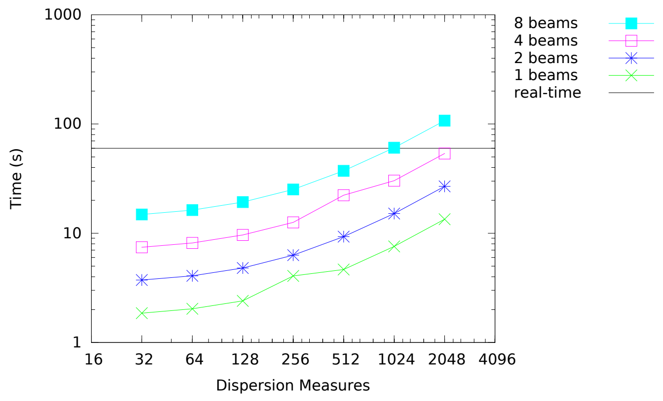

Fig. 4. Pipeline performance on the K20X.

IV. PERFORMANCE RESULTS

In this section we describe the results of the experiment described in Section III. Figures 3 and 4 present the execution time of the pipeline, running on the HD7970 and K20X respectively; the two figures are log-log plots. The first result is that, for both GPUs, the pipeline scales linearly in both the number of beams and trial DMs. The performance, however, are different for the two devices, with the AMD GPU being faster than the NVIDIA one in almost all test cases. The reason for the difference in performance between the two devices is simple enough: the most time consuming step of the whole pipeline is dedispersion, and the HD7970 is faster at dedispersion than the K20X. While a complete analysis of dedispersion performance can be found in [4], we can say that the higher achievable memory bandwidth of the AMD GPU, coupled with its better cache system, makes it a better platform for a memory-bound kernel like dedispersion. The black line labeled "real-time" in the figures represents the threshold over which the pipeline is too slow to process the 60 seconds of input data in less than 60 seconds of execution time. As can be seen from the results, the HD7970 always satisfy the real-time requirement, while the K20X is unable to do so for a number of DMs higher than 1,024 and 8 beams.

Figure 5 provides a performance breakdown of the

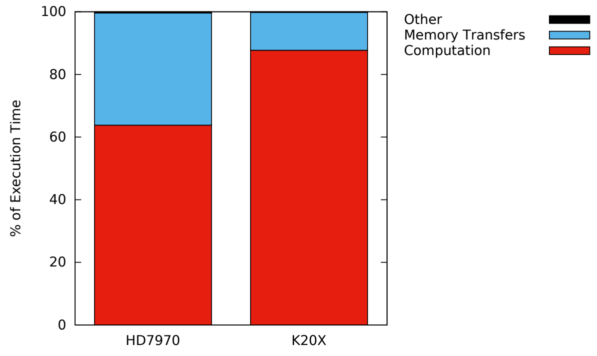

Fig. 5. Performance breakdown of the pipeline, 1 beam and 2,048 DMs.

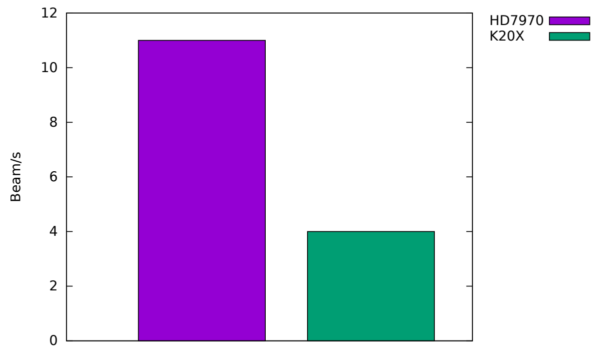

Fig. 6. Throughput in beams per second, 2,048 DMs.

pipeline's execution time processing 2,048 DMs and a single beam. This breakdown is useful to analyze where the pipeline's bottlenecks are, and check if they are the same for each platform. From the figure we see that, for both platforms, 63 to 87 percent of the execution time is spent executing the kernels on the GPU, 35 to 12 percent on memory transfers between host and device, and only less than 1 percent on thresholding and other accessory operations. Therefore, to increase the performance of this pipeline all future efforts should be dedicated to optimizing and further improving the two computational kernels, and especially dedispersion.

Figures 6 and 7 shows the throughput of our pipeline, in

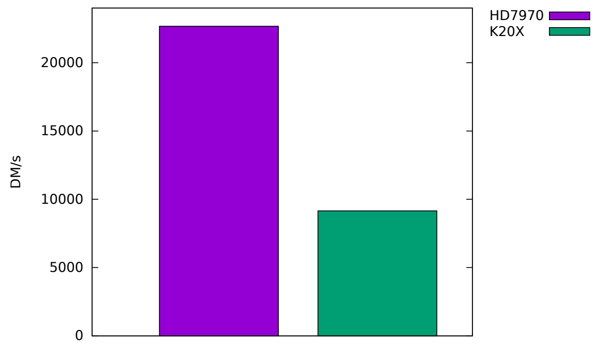

Fig. 7. Throughput in DMs per second, 1 beam.

terms of beams and DMs that can be processed per second. The results show that the HD7970 could compute 11 beams per second, in real-time, or more than 22,000 DMs in a single beam scenario; as expected from previous results, this is more than twice the throughput achieved by the K20X. Although the results presented so far can be used to provide a comparison between these two GPUs in the context of radio transient surveys, the same results can also be used to estimate the number of GPUs that would be necessary to build the transient pipeline of ARTS as of today.

ARTS will have to survey 444 beams in real-time, and process each of them for 2,000 different trial DMs. This means that we could implement ARTS today, using our pipeline, with 41 AMD HD7970, or 111 NVIDIA K20X GPUs. As expected, the higher throughput of the AMD GPU makes it possible to build the same instrument with less hardware. Although it may seem exaggerated to build a system capable of hundreds of TFLOP/s for a problem that only requires tens of them, we need to stress that the compute kernels of this pipeline are all memory-bound and this makes impossible to achieve peak performance. Still, GPUs outperform CPUs by a factor of 30 [4] because of their higher memory bandwidth.

A more compact system is not only easier to manage and less expensive to build, but it is also less power hungry, thus cheaper to operate. Just taking into account the TDP of the GPUs, provided in Table I, we can provide here a lower bound on the power necessary to operate ARTS: 10.2 kW using the AMD solution, or 26 kW for the NVIDIA one. We believe that the introduction of new technologies, such as three-dimensional stacked memory, would help us reduce this lower bound even further in the coming months.

V. Conclusions

In this paper we introduced a prototype for the radio transient pipeline of ARTS. This pipeline will have to process, in real-time, 444 different input beams at a data rate of 36 GB/s, and the processing will require a throughput of more than 18 TFLOP/s. Therefore, we designed and developed a pipeline that leverages the massive amount of parallelism provided by modern GPUs. Our pipeline, implemented using C++ and OpenCL, is both high-performance and portable, and can be automatically tuned for different hardware platforms and observational scenarios.

The performance results presented in this paper show that our pipeline scales linearly, on both tested GPUs, in the number of beams and trial DMs. Moreover, we are able to show that this pipeline can process in real-time thousands of DMs, even for more than one beam. In particular, using an AMD HD7970 GPU we show that this pipeline can process 11 beams per second at 2,048 DMs, or more than 22,000 DMs in a single beam scenario. We can then conclude that, using currently available hardware, ARTS could be built today using 41 GPUs, and require 10.2 kW of power for running this pipeline.

References

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

[2] D. R. Lorimer, M. Bailes, M. A. McLaughlin, D. J. Narkevic, and F. Crawford, "A bright millisecond radio burst of extragalactic origin," *Science*, vol. 318, no. 5851, pp. 777–780, 2007.

[3] 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.

[4] 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.

[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, vol. 461, Sep. 2012, p. 33.

[6] A. Magro, J. Hickish, and K. Z. Adami, "Multibeam GPU transient pipeline for the medicina BEST-2 array," *Journal of Astronomical Instrumentation*, 2013.

[7] 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.

[8] T. F. Chan, G. H. Golub, and R. J. Leveque, "Algorithms for computing the sample variance: Analysis and recommendations," *The American Statistician*, vol. 37, no. 3, pp. 242–247, 1983.