Astronomy and Computing 14 (2016) 1–7

Contents lists available at ScienceDirect

Astronomy and Computing

journal homepage: www.elsevier.com/locate/ascom

Full length article

Real-time dedispersion for fast radio transient surveys, using auto tuning on many-core accelerators

A. Sclocco [a,b,*], J. van Leeuwen [b,c], H.E. Bal [a], R.V. van Nieuwpoort [d]

[a] Faculty of Sciences, Vrije Universiteit Amsterdam, Amsterdam, The Netherlands
[b] ASTRON, The Netherlands Institute for Radio Astronomy, Dwingeloo, The Netherlands
[c] Anton Pannekoek Institute for Astronomy, University of Amsterdam, Amsterdam, The Netherlands
[d] NLeSC, Netherlands eScience Center, Amsterdam, The Netherlands

ARTICLE INFO

Article history:
Received 2 November 2015
Accepted 1 January 2016
Available online 11 January 2016

Keywords:
Pulsars: general
Astronomical instrumentation, methods and techniques
Techniques: miscellaneous

ABSTRACT

Dedispersion, the removal of deleterious smearing of impulsive signals by the interstellar matter, is one of the most intensive processing steps in any radio survey for pulsars and fast transients. We here present a study of the parallelization of this algorithm on many-core accelerators, including GPUs from AMD and NVIDIA, and the Intel Xeon Phi. We find that dedispersion is inherently memory-bound. Even in a perfect scenario, hardware limitations keep the arithmetic intensity low, thus limiting performance. We next exploit auto-tuning to adapt dedispersion to different accelerators, observations, and even telescopes. We demonstrate that the optimal settings differ between observational setups, and that auto-tuning significantly improves performance. This impacts time-domain surveys from Apertif to SKA.

1. Introduction

Astronomical phenomena such as pulsars (Hewish et al., 1968) and fast radio bursts (FRBs; Lorimer et al., 2007) consist of millisecond-duration impulsive signals over a broad radio-frequency range. As the electromagnetic waves propagate through the interstellar material (ISM) between us and the source, they are dispersed (Lorimer and Kramer, 2005). This causes lower radio frequencies to arrive progressively later, and without correction this results in a loss of signal-to-noise, that often makes the source undetectable when integrating over a wide observing bandwidth. This frequency-dependent delay can be removed in a process called *dedispersion*. Complete removal can be achieved by reverting all phases through a convolution of the signal with the inverse of the modeled ISM (coherent dedispersion; Hankins and Rickett, 1975). Incomplete but much faster removal, especially when many dispersion measure trials are required, can be achieved by appropriately shifting in time the signal frequency channels (incoherent dedispersion; from now on referred to plainly as dedispersion). This dedispersion is a basic algorithm in high-time-resolution radio astronomy, and one of the building blocks of surveys for fast phenomena with modern radio telescopes such as the Low Frequency Array (LOFAR; van Leeuwen and Stappers, 2010; Stappers et al., 2011) and the Square Kilometer Array (SKA; Carilli and Rawlings, 2004). In these surveys, the dispersion measures are a priori unknown, and can only be determined in a brute-force search. This search generally runs on off-site supercomputers. These range from e.g., the CM-200 in the Foster et al. (1995) Arecibo survey, to gSTAR for the Parkes HTRU (Keith et al., 2010), and Cartesius for the LOFAR LOTAAS (Coenen et al., 2014) surveys. In the latter the dedispersion step amounts to over half of all required pulsar-search processing. For the SKA, this processing will amount to many PFLOPS for both SKA-Mid (cf. Magro, 2014) and SKA-Low (Keane et al., 2014).

Above and beyond these pure compute requirements, the similar and often simultaneous search for FRBs demands that this dedispersion is done near real time. Only then can these fleeting events be immediately followed up by telescopes at other energies (Petroff et al., 2015).

We aim to achieve the required performance by parallelizing this algorithm for many-core accelerators. Compared to similar attempts made by Barsdell et al. (2012) and Armour et al. (2012), we present a performance analysis that is more complete, and introduce a novel many-core algorithm that can be tuned for different platforms and observational setups. To our knowledge,

* Corresponding author at: ASTRON, The Netherlands Institute for Radio Astronomy, Dwingeloo, The Netherlands.
E-mail addresses: a.sclocco@vu.nl (A. Sclocco), leeuwen@astron.nl (J. van Leeuwen), h.e.bal@vu.nl (H.E. Bal), r.vannieuwpoort@esciencecenter.nl (R.V. van Nieuwpoort).

http://dx.doi.org/10.1016/j.ascom.2016.01.001
2213-1337/

this is the first attempt at designing a brute-force dedispersion algorithm that is highly portable and not fine-tuned for a specific platform or telescope.

To summarize our contributions, in this paper we: (1) provide an in-depth analysis of the arithmetic intensity (AI) of brute-force dedispersion, finding analytically and empirically that it is memory bound; (2) show that auto-tuning can adapt the algorithm to different platforms, telescopes, and observational setups; (3) demonstrate that many-core accelerators can achieve real-time performance; (4) quantify the impact that auto-tuning has on performance; (5) compare different platforms using a real-world scientific application; and (6) compare the performance of OpenCL and OpenMP for the Intel Xeon devices.

In Section 2 we describe the brute-force dedispersion algorithm, our parallel implementation and its optimizations; and the theoretical analysis of the dedispersion AI. We next present our experiments (Section 3), results (Section 4) and further discussion (Section 5). Finally, relevant literature is discussed in Section 6, and Section 7 summarizes our conclusions.

2. The brute-force dedispersion algorithm

In dispersion (Lorimer and Kramer, 2005), the highest frequency in a certain band f_h is received at time t_x, while lower simultaneously emitted frequency components f_i arrive at $t_x + k$. For frequencies expressed in MHz this delay in seconds is:

$$k \approx 4150 \times DM \times \left(\frac{1}{f_i^2} - \frac{1}{f_h^2}\right). \quad (1)$$

Here the Dispersion Measure *DM* represents the projected number of free electrons between the source and the receiver. In incoherent dedispersion, the lower frequencies are shifted in time and realigned with the higher ones, thus approximating the original signal.

In a survey, the incoming signal must be brute-force dedispersed for thousands of possible DM values. As every telescope pointing direction or *beam* can be processed independently, performance of the dedispersion algorithm can be improved by means of large-scale parallelization.

2.1. Sequential algorithm

The input of this algorithm is a frequency-channelized time series, represented as a $c \times t$ matrix, with c frequency channels and t time samples needed to dedisperse one second of data. The output is a set of d dedispersed trial-DM time-series, each of length s samples per second, represented as a $d \times s$ matrix. All data are single precision floating point numbers; their real-life rates are e.g. 36 GB/s input and 72 GB/s output for the pulsar search with Apertif on the Westerbork telescope (van Leeuwen, 2014).

Dedispersion (sequential pseudocode shown in Algorithm 1) then consists of three nested loops, and every output element is the sum of c samples: one for each frequency channel. Which samples are part of each sum depends on the applied delay (i.e. Δ) that, as we know from Eq. (1), is a non-linear function of frequency and DM. These delays can be computed in advance, so they do not contribute to the algorithm's complexity. Therefore, the complexity of this algorithm is $O(d \times s \times c)$.

In the context of many-core accelerators, there is another, extremely important algorithmic characteristic: the *Arithmetic Intensity* (AI), i.e. the ratio between the performed floating-point operations and the number of bytes accessed in global memory. In many-core architectures the gap between computational capabilities and memory bandwidth is wide, and a high AI is thus a prerequisite for high performance (Williams et al., 2009).

Algorithm 1 Pseudocode of the brute-force dedispersion algorithm.

```
for dm = 0 → d do
  for sample = 0 → s do
    dSample = 0
    for chan = 0 → c do
      dSample += input[chan][sample + Δ(chan, dm)]
    end for
    output[dm][sample] = dSample
  end for
end for
```

Unfortunately, the AI for Algorithm 1 is inherently low, with only one floating point operation per four bytes loaded from global memory. For dedispersion,

$$\text{AI} = \frac{1}{4+\epsilon} < \frac{1}{4} \quad (2)$$

where ϵ represents the effect of accessing the delay table and writing the output. This low AI shows that brute-force dedispersion is memory bound on most architectures. Its performance is thus limited not by computational capabilities, but by memory bandwidth. One way to increase AI and thus improve performance, is to reduce the number of reads from global memory, by implementing some form of data reuse. In Algorithm 1 some data reuse appears possible. For some frequencies, the delay is the same for two close DMs, dm_i and dm_j, so that $\Delta(c, dm_i) = \Delta(c, dm_j)$. Then, one input element provides two different sums. With data reuse,

$$\text{AI} < \frac{d \times s \times c}{4 \times ((s \times c) + (d \times s) + (d \times c))} = \frac{1}{4 \times \left(\frac{1}{d} + \frac{1}{s} + \frac{1}{c}\right)}. \quad (3)$$

The bound from Eq. (3) theoretically goes toward infinity, but in practice the non-linear delay function diverges rapidly at lower frequencies. There is never enough data reuse to approach the upper bound of Eq. (3); for a more in-depth discussion see Sclocco et al. (2014). We thus categorize the algorithm as memory-bound. In this conclusion we differ from previous literature such as Barsdell et al. (2010) and Barsdell et al. (2012). The importance of the above mentioned data reuse in dedispersion was identified early on and implemented in e.g. the tree dedispersion algorithm (Taylor, 1974). That fast implementation has the drawback of assuming the dispersion sweep is linear. Several modern pulsar and FRB surveys with large fractional bandwidths have used modified tree algorithms that sum over the *quadratic* nature of the sweep (e.g. Manchester et al., 2001; Masui et al., 2015).

2.2. Parallelization

We first determine how to divide and organize the work of different threads, and describe these in OpenCL terminology. We identify three main algorithm dimensions: DM, time and frequency. Time and DM are independent, and ideal for parallelization, avoiding any inter- and intra-thread dependency. In our implementation, each OpenCL work-item (i.e. thread) is associated with a different (DM, time) pair and it executes the innermost loop of Algorithm 1. An OpenCL work-group (i.e. group of threads) combines work-items that are associated with the same DM, but with different time samples.

This proposed organization also simplifies memory access, using coalesced reads and writes. Different small requests can then be combined in one bigger operation. This well-known optimization is a performance requisite for many-core architectures, especially for memory-bound algorithms like dedispersion. Our output elements are written to adjacent, and aligned, memory locations. The *reads* from global memory are also coalesced but, due to the shape of the delay function, are not always aligned. The worst-case memory overhead is at most a factor two (Sclocco et al., 2014).