Benchmarking of GPU-based pulsar processing pipeline of 40-m Thai national radio telescope

Tinn Thongmeearkom1,*, Phrudth Jaroenjittichai1, Thomas Scrage2, Charlie Walker2, Rene Breton2 and Ben Stappers2

1 National Astronomical Research Institute of Thailand, Maerim, Chiangmai, 50180, Thailand
2 The University of Manchester, Oxford Road, Manchester, M13 9PL, United Kingdom

*E-mail: 01tinn.th@gmail.com

Abstract. In recent years, Graphics Processing Unit (GPU) have been widely used in several astronomical applications. The 40-m Thai National Radio telescope (TNRT) is under construction in Chiang Mai, Thailand. We conducted benchmarking of the pulsar processing software to evaluate the capabilities of a computer with Xeon E5-2630 and GPU GTX1080Ti. The pulsar software DSPSR was used to simulate raw baseband data, coherently de-disperse the data and generate a folded time-frequency-domain pulse profile. We experimented with combinations of bandwidth, the size of the sub-bands, a range of dispersion measure (DM) values, and parallel instances of DSPSR jobs. The result shows the processing time increases with higher values of bandwidth DM as expected. However, the processing time appears to decrease with the size of the sub-bands, that at the same total bandwidth the processing with 1.5625 MHz/channel is faster than those with 3.125, 6.25 and 12.5 MHz/channel by approximately 10, 25 and 50 percent, respectively. This indicates that the processing in DSPSR is best when the channel resolution is high, however, further investigation is needed to determine the highest optimal value. We also consider parallel processing and to this, one, two and four identical scripts were simultaneously executed in parallel, where we found that single job is six times faster than four simultaneous jobs. In principle, parallel computing is expected to be more efficient, however, this can be explored further to find the actual bottleneck in the pipeline and hardware.

1. Introduction

A backend is a group of instruments and programs. Normally, we consider everything after the detection instruments as backend. Different types of observing modes require different types of software and hardware. A telescope can have several observing modes depends on the design and demand, the most well-known modes are timing mode, searching mode and spectral line mode. Astronomers use a pulsar timing mode to detect pulsar’s signal from known pulsars, this mode requires pulsar related programs, for instance, PSRCHIVE to check pulsar’s parameters in a catalogue and DSPSR in order to coherently de-disperse the signal [1]. The backend designing is a crucial part of telescope development: dealing with data stream, managing storage, processing real-time signal and preventing the bottleneck. Therefore, this is an arduous task. Nonetheless, there are developments for backend around the world. To illustrate, HIPSR which is the design exclusively for Parkes 21-cm multibeams receiver, contributes potential for mapping experiment and large bandwidth [2]. In addition, there is another study which investigates the capability of GPU for Hebe which is a pulsar timing system designed of a telescope at Kuntunse [3]. To find a suitable specifications for backend, the first step that we can do is Benchmarking. We did the benchmarking by changing the combination of initial conditions such as bandwidth, number of channels, and DM. We investigate the relationship between those parameters and processing time.
2. Pulsar signal processing

The radio telescope’s receiver detects pulsar’s radio signal from the sky in the form of voltages. Then, the digitiser digitises these voltages into raw digitised data. After that, they are separated and sent to several work stations. These data are called baseband data. There are variety of techniques for pulsar signal processing: filterbanking and de-dispersion.

2.1. Filterbanking

Filterbanking or Channelisation is a step where the program uses Fourier transform or Fast Fourier Transform (FFT) to turn raw baseband data into a filterbank file which has intensity, time, and frequency component. Fourier transform is a tool for changing the data from one to another domain. Due to the nature of the Fourier transform, the time resolution is inversely proportional to frequency resolution \[4\]. Restriction of Fourier transform is sampling rate must be twice the full band by the virtue of Nyquist frequency.

2.2. De-dispersion

Interstellar Medium (ISM) between two places influences an electromagnetic wave that propagates through those places. The wave experiences phased delay, lower frequency experiences more delay than higher frequency this effect known as Dispersion Measure (DM). Nevertheless, the DM can be calculated by the equation (1), \[ t_2 - t_1 (ms) = 4.15 \times DM \times \left(\frac{1}{f_{\text{low}}} - \frac{1}{f_{\text{high}}} \right) \] (1)

Incoherent de-dispersion is a post-filterbank process. The algorithm separates the signal into chunks of data. By doing this it can calculate a delay for each individual chunk. Furthermore, it regulates those delays. Incoherent de-dispersion doesn’t require an immensely computational machine and it can be done offline. Nonetheless, it comes with a downside of smearing effect which can happen because of imperfect reconstruction technique of incoherent de-dispersion. The smearing happens when the delay is longer than those chucks \[5\]. On contrary, coherent de-dispersion is a pre-filterbank process. It can be done at the exact same time with filterbanking, coherent de-dispersion can only be done online because we can’t store all of the numerous raw data. It also requires more intensive, more powerful machine than incoherent de-dispersion. To use this technique, the algorithm uses a specific function to adjust the phase of a signal and does FFT \[1\]. Consequently, this technique prevents the smearing effect. Because of the computationally intensive and requirements of real-time processing, we use coherent de-dispersion as a standard set up for benchmarking to test the capability of the machine.

For pulsar signal processing, the program does coherent de-dispersion in real-time and channelisation. Lastly, the output is a filterbank file. To get the data for analysis, users use DSPSR to fold the filterbank file using folding parameters from PSRCHIVE and get a result as an archive file. This archive file is a final product which users use several programs to plot graph and clean radio frequency interferences.

DSPSR is the main software for processing pulsar data, for example, folding and de-dispersion. Moreover, DSPSR has a command for simulating data. By this generated data and ability to process data, we decided to use DSPSR to investigate the ability of a machine. We used DSPSR which installed with CUDA 8.0 for benchmarking. Distinct CUDA version can influence processing time.

3. Benchmarking

Benchmarking is a technique for investigating the potential of machines or potential of programs. In this study, we focused on coherent de-dispersion algorithm and parameters. We used DSPSR to simulate baseband data. We adjusted various combinations of initial conditions to investigate the relationship between those parameters and results. The initial conditions are size of bandwidth, size of sub-bands (table 1) and value of DM (1, 3, 10, 30, 100, 300, 1000 and 3000). There are other parameters such as data length, centre frequency and ram usage that we must input them however we didn’t vary those parameters. We used centre frequency of 1.4 GHz and minimum ram usage of 512 MB running on a computer with Xeon E5-2630 and GPU GTX1080Ti.
Table 1. The combination of intial conditions for single task.

| Bandwidth(MHz) | Sub-bands(MHz) | Number of channels |
|----------------|----------------|-------------------|
| 100            | 1.5625         | 64                |
| 100            | 3.125          | 32                |
| 100            | 6.25           | 16                |
| 100            | 12.5           | 8                 |

We did this by using bash script provided by Straten [1]. We adjusted the script to print out more information about each individual processing time. Hence, we could use standard deviation as error. The script uses for-loops to repeat the process ten times. By changing each parameter, the program prints out processing time for each combination. We used these results to calculate the error for each point and plotted graphs using an in-house python script. In this report, we present the result of various numbers of channels and the result of parallel processing. To do parallel processing, we used the same script running at the exact same time on the identical GPU with a combination of initial conditions. For one task, we used bandwidth of 400 MHz with 64 channels. Conversely, we used bandwidth of 200 and 100 MHz with 32 and 16 channels for two and four tasks respectively.

4. Results and conclusions

For single task (figure 1), the result of 100 MHz total bandwidth with 1.5625, 3.125, 6.25, 12.5 MHz/channel (64, 32, 16, 8 channels) exhibits that 1.5625 MHz/channel is the fastest. At DM of 1-1000, it is faster than 3.125, 6.25, 12.5 MHz/channel approximately 10, 25 and 50 percent respectively. At DM of 3000, 12.5 MHz/channel spends more time than 1.5625 MHz/channel about 2.4 times.

Figure 1. The graph between DM and normalised time with bandwidth of 100 MHz, data length of 4 seconds, number of channels of 8, 16, 32 and 64 represented by colour blue, orange, green and red respectively.
For multiple tasks, the result shows that the processing time of one task is approximately 2.6 seconds. Two tasks, on the other hand, spend approximately 5 seconds. Lastly, four tasks have two groups: longer time about 17.5 and shorter time about 1 seconds. This indicates that the hardware can’t handle this combination of parameters that results in spending more time on multiple tasks.

In conclusion, the processing time is decreased by the size of the sub-bands. The same total bandwidth of 100 MHz with 1.5625 MHz/channel is faster than those with 3.125, 6.25 and 12.5 MHz/channel respectively by approximately 10, 25 and 50 percent. Therefore, this shows that DSPSR is foremost with high frequency resolution. In contrast, for parallel processing, the result indicates that a single task is six times faster than four simultaneous tasks. In theory, parallel computing should be more effective than a single job. Nevertheless, we should investigate further about the pipeline and hardware in order to find the bottleneck.

Acknowledgements

We would like to thanks Newton-Fund project for funding. We also would like to thank Kaustubh Rajwade, Mateusz Malenta, Attasit Phakam, Nattaporn Thoonsaengngam and Chaiyaphorn Nilarat for the technical part of software and hardware.

References

[1] van Straten W and Bailes M 2011 Publ. Astron. Soc. Aust. 28 1–14
[2] Price D C, Staveley-Smith L, Bailes M, Carretti E, Jameson A, Jones M E, van Straten W and Schediwy S W 2016 J. Astron. Instrum. 5 1641007
[3] Scragg T W, Stappers B W, Breton R P, Smith J N, Adomako D, Asabere B D, Chibueze J O and Cloete K Pulsar Observations at the Ghana Radio Astronomy Observatory (Pulsar Astrophysics the Next Fifty Years (IAU Symposium vol 337)) (Cambridge : Cambridge University Press) ed P Weltevrede, B B P Perera, L L Preston and S Sanidas pp 410-1
[4] Smith S W The Scientist and Engineer’s Guide to Digital Signal Processing (San Diego: California Technical Publishing) Online: http://www.dspguide.com.
[5] De K and Gupta Y 2016 Exp. Astron. 41 67–93