SKA Project Series

The FLDO Module for SKA-PSS

Firenze September 2014
C.Baffa, E.Giani

Arcetri Technical Report 3/2014

Abstract

One of the key scientific projects of the SKA radio telescope is a large survey for pulsars both in isolated and binary systems. The data rate of the pulsar search engine is expected to reach 0.6 TeraSamples/sec. For the purposes of extracting hidden pulses from these streams, we need a complex search strategy which allows us to explore a three dimensional parameter space and it requires approximately 10PetaFlops. This problem is well suited for a parallel computing engine, but the dimensions of SKA bring this problem to a new level of complexity.
Current design is based on a large number of GPUs. The PSS team is actively developing a collection of modules which, acting as a pipeline, will perform this search in real time. The Arcetri group is in charge of the FLDO module, and this report documents the development and test of such module.

1 Introduction

The Square Kilometer Array is a giant radio telescope project, and it is the result of a worldwide effort to develop a very large (both in area and extension) multi-wavelength instrument. One of the key scientific projects of the SKA radio telescope is the large survey for pulsars, PSS.

The SKA Pulsar Search Survey, in the baseline design, will get 1750 simultaneous beams, digitized every 50μs, on up to 4096 frequency channels, for a total of 0.5TeraSamples/sec. It is required a complex search strategy to extract the hidden pulses from these streams. To correctly add up the singles pulses, thus overcoming the overwhelming background noise, the pulsar processor needs to explore a three dimensional space: the pulse period, the interstellar medium frequency dispersion and the peculiar acceleration of the source.

In addition, *all the calculation required needs to be performed in real time*. Due to the large data flow involved, it is impossible to retain more than a tiny fraction of the raw data. The SKA Pulsar Search input is approximately 1PetaBytes on each cycle of observation which lasts up to 600s.

This optimization problem is well suited for a parallel computing engine, and there are already example implementations for present-day radio telescopes. However, the dimension of the SKA project brings this problem to a new level of complexity. The PSS is thus designed to be a real-time system, which converts the enormous input data stream reduced to a just a few hundreds of megabytes of pulsar candidates that are passed to the Science Data Processor (SDP).

The PSS team is actively developing a collection of modules which, acting as a pipeline, will perform this search in real time. The Arcetri group is in charge of the last portion of this search, the folding (or *integrating* module, FLDO) module, and we will documents the development and test of such module.

The folding engine would operate on multi-channel time-series raw data and integrate the signal (fold) over time. To get meaningful results, this task needs to know the period, the DM and the acceleration. The previous computing stage would present a set of candidates with tentative parameters.

After having folded raw data on the candidate characteristics the folding engine will then optimize in period, period derivative and dispersion measure to produce the optimal signal parameters and signal-to-noise ratio for each candidate, and send the result to the SDP.

This computing area is tricky to be performed with GPUs, as it has a lower I/O to computation ratio than previous steps. The result is a lower gain transferring his algorithm from a multi-thread multi-core CPU to a GPU, even in spite of the larger computing capability of the latter.

2 Summary of FLDO operations

The present day implementation consists of a four phases process, executed in sequence for each pulsar candidate.

At first data is read from source (now a disk file, but can be a memory area or a network stream) and split into a number of sub-integrations (default 64).

Data is then transposed and converted from 8-bit integers to 32 bit float numbers. If the candidate period is longer than a specific value data is also pre-binned.

The pre-binning operation consists in summing up adjacent time samples. The number of time samples to sum is a power of 2, so subsequent pre-binning can share some of previous computation.

The next phase is the real folding. For each sample is computed the phase relative to the candidate period corrected for DM and acceleration. Each sample is then proportionally split between two successive candidate phases. An array of weights takes care of the different measure numbers added to each candidate phase. For each candidate the algorithm produces a matrix of profiles, one for each sub-integration and group of frequencies (default 64x64).

Fourth phase is the fine tuning of candidates parameters, by means of a grid optimization search.

The optimization result is then forwarded to SDP for further processing.

For the pre-binning procedure, if each sample is *not* proportionally split, there can be an alternative approach. In this case, each measure of original data is added to the nearest phase bin, de-facto performing an a-posteriori pre-binning. This will result in a huge performance hit, as, in this case, re-bin is performed for each candidate.

Final decision to be taken at the final down-select decision.

3 Phase 1 - Read and Split

The first phase of FLDO consists in getting data and splitting the input stream in a small number of parts in order to perform a fine tuning in the fourth phase. Data input is assumed to consist of a continuous stream of 8 bit numbers. Such values are the intensity of I Stokes parameter, and are ordered first in frequency (fastest index) and then in time. The PSS observation default values are assumed as 50µs of time resolution, 4096 frequency channels, 420s integration time. These values will result in a 34GB data chunk. The *origami* implementation is able to process the maximum specified data length of 1800s (145GB).

The *origami* implementation assumes data is stored in a local disk file, but it can easily modified to accept data from a network stream or a memory buffer.

4 Phase 2 – Transpose, conversion and Pre-bin

Data is then transposed and converted from 8-bit integers to 32 bit float numbers.

The pre-binning procedure consists in summing up adjacent time samples. The number of time samples to sum is a power of 2: for the moment the maximum value is 16 but in future we think to increase this number.

The pre-binning helps us to reduce the size of data to be processed and also let us handle pulsar with longer spin period.

The pulsar candidates input list is sorted by ascending periods. The candidates are divided in groups. In the next table (where we assume the default sampling rate of 50μs) we devise two possible strategies. The *Fastest method* strategy gives the fastest execution times, at the expense of final time resolution, while the *Conservative* method gives the maximum resolution compatible with the hardware limits. Maximum attainable S/N ratio seems to favor the conservative approach.

Candidate Period - Fastest method	*Candidate Period Conservative*	*Re-bin Factor*
P < 0.0008 sec	P < 0.0032 sec	**1**
0.0008 < P < 0.0016 sec	0.0032 < P < 0.0064 sec	**2**
0.0016 < P < 0.0032 sec	0.0064 < P < 0.0128 sec	**4**
0.0032 < P < 0.0064 sec	0.0128 < P < 0.0256 sec	**8**
0.0064 < P < 0.049 sec	0.0256 < P < 0.049 sec	**16**

The folding process gives a representation of the distribution of the data as function of the phase relative to the pulse period. The folding generation is a operation where each sample votes in a reduced set of phases (bins). The number of natural phases is calculated using the next formula (the factor 2 comes from the Nyquist rule):

$$n_{phases} = 2 * P/t_{sampling}$$

The current implementation in GPU of the FLDO module works with a maximum of 126 bins[1]: higher values limit the GPU resources with an impact on the number of threads running in parallel. With this limit on the number of the phases and a $t_{sampling} = 50$ µsec, the FLDO module could only handle pulsars with periods up to 3.2 ms if we do not implement re-bin.
The *origami* program uses a maximum re-binning factor of 16, so the FLDO can detect pulsars with period up to 49 ms. Higher pre-binning values can be easily implemented. There is not an hard maximum folding factor, we only put on our code a limit of a 16 re-bin folding, resulting in a maximum candidate period of 51ms, only as a temporary provision, to limit the complexity of set-up code during the developing phase. Adding higher pre-bin, we can ramp up to seconds without problems

5 Phase 3 – Coherent Folding

The folding algorithm consists in a synchronous summation of input data in order to improve the Signal/Noise.

1 We use 126 and not 128 as a maximum to avoid the necessity to check for 'end of buffer' condition. This saves the use of a costly modulo function. Phase bins 127 and 128 are respectively added to phase 0 and 1 at the end of computation.

Input data is processed in blocks of 64 sub-integrations. Each sub-integration is then divided in 64 bands of frequencies. These are the default values, but the folding program can change them through the argument options (you can display them by the '-h' options)

The original data can be seen as a matrix of:

Quantity	**Value**
nrow	nchan
ncol	nsampl

Where nsampl is equel to (tobs/t_sampling)/nsubint.
The execution of the corner-turning of each block of input data facilitates coalesced reads and it results in (leads to) better execution performances. The data of each sub-integration is transposed before the folding.

So the folding algorithm works on an array of data where:

Quantity	**Value**
nrow	nsampl
ncol	nchan

For each block the folding produces two array on n_{phases} elements: the first with the coherent sum of the data (intensity profile) and the other with the weights of each phase (weights profile).
We devised two different coherent summing approaches. The first one, which we analysed more thoroughly, will consist of a synchronous summation with phase split. This summation is illustrated in Figure 1.
The second summation approach doesn't split input data, but it sums up each sample in the bin which correspond to its central phase (Figure 2). This approach is 20-30% faster, but it implies a small resolution loss.
At the end of the whole process we get 64 * 64 intensities profiles and 64 * 64 weights profiles.

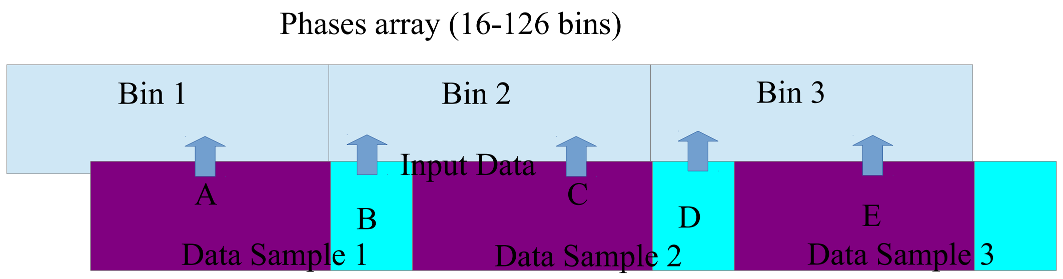

Figure 1. *Synchronous summation with phase split: input data is split according to its phase relative to the pulsar candidate phase array. In the above figure, Data Sample portion A will be summed up to Bin1, Data Sample portions B and C will be summed up to Bin2, Data Sample portions D and E will be summed up to Bin3, and so on.*

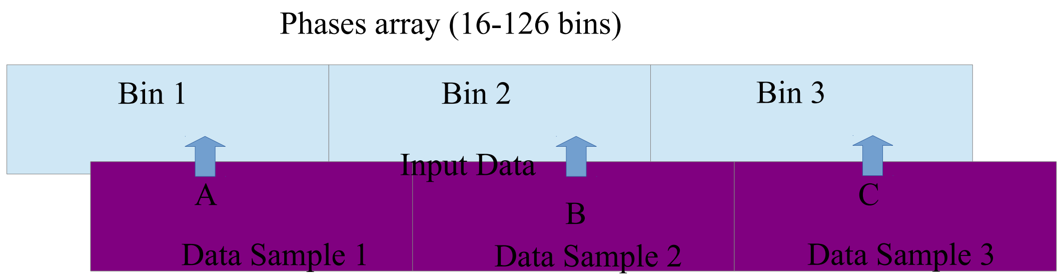

Figure 2. *Synchronous summation without phase split: input data is summed to the phase bin corresponding to its central phase. In the above figure, Data Sample A will be summed up to Bin1, Data Sample B will be summed up to Bin2, Data Sample C will be summed up to Bin3, and so on.*

6 Phase 4 – Normalization and Optimization

Last steps are the normalization and the optimization phases.

6.1 Normalization

Each intensity profile is divided for the corresponding weight profile, correcting for a non uniform number of samples added to each phase bin. This portion has a negligible impact in the total execution time.
Then the value of the mean of the current measure is computed[2] and subtracted from the normalized profile. Also this portion has a negligible impact in the total execution time (0.5 %).

6.2 Optimization

As last step the folding data are "Optimized". This portion consists to calculate on the data the effect of small perturbation of candidate parameter on the final signal to noise ratio. This optimization is performed by a simple grid search, well suited for a parallel engine.
Some effort have been devoted to the possible use of an "amoeba style" optimization approach. But we did not get good results, probably for the 'clumpiness' of the numerical function.

6.3 Reduction

The normalized profiles are summed up along frequencies and sub-integrations to produce the reduced profiles, which are the result of our computation.

7 Performances

We have collected in a table the GPU execution times and the relative S/N, for 128 candidates with different folding algorithms. Candidates periods are distributed uniformly up to 49ms. All measurements are relative to a Nvidia K40 GPU.

2 We use a customized version of Cuda::Thrust library

Measure total time	**Split coherent folding fastest (R1)**		**Split coherent folding conservative (R2)**		**Non-Split folding conservative (R2 N)**	
	Gpu time	**S/N**	**Gpu time**	**S/N**	**Gpu time**	**S/N**
60	11.3	64	19.3	115	14.0	116
120	19.1	99	34.8	164	24.4	164
240	35.0	137	65.9	228	45.7	235
420	59.9	187	113.4	307	80.5	306
600	90.0	---	188.2	---	164.5	---

The 600s observation time line do not gives meaningful S/N as data are wrapped non coherently.

8 Conclusions

The performances shown in the previous chapter drive us to consider the non-split folding with conservative pre-binning as the best compromise.

Some more work should still to be planned in the optimization part.

9 References

[1] Dewdney, P., “Ska1 system baseline design,” in [SKA1 Key documents], Diamond, P., ed., SKA-TEL–SKO–DD–001, SKA Organization, Manchester, UK (2013). https://www.skatelescope.org/home/technicaldatainfo/key-documents/.

[2] Keith, M., “Ska csp ska1-mid array non-imaging processing pulsar search sub-element architecture design document,” in [SKA1 CSP Thecnical Documents], Carlson, B., ed., SKA–TEL.CSP.NIP.PSS–TDT–ADD– 001, SKA Organization, Manchester, UK (2014).

[3] K., C., R., C., and et al., “Toward early-warning detection of gravitational waves from compact binary coalescence,” APJ 748,2, id136–14 (2012).

[4] D., L. and M., K., [Handbook of Pulsar Astronomy], Cambridge University Press, Cambridge (2005).