Pulsar Searching at the NAOC, Guizhou Normal University, and FAST Radio Telescope

Shana Li

Summer 2017

Abstract

During this summer from May 22, 2017 to July 28, 2017, I, along with Zach Komassa (University of Wisconsin-Milwaukee), undertook pulsar searching research at the National Astronomical Observatories, Chinese Academy of Sciences (henceforth NAOC), Guizhou Normal University (henceforth GZNU), and the Five Hundred Meter Aperture Spherical Telescope (henceforth FAST). We were responsible for accessing the NAOC and GZNU servers through Secure Shell (ssh) and VNC Viewer, installing **PRESTO** (the **PulsaR Exploration and Search TOolkit** developed by Scott Ransom) [12] on machine nodes on said servers, and writing shell scripts using the Bash language to run **PRESTO** commands on a Unix shell. We worked with Di Li, a professor, and Zhichen Pan, a postdoc, at NAOC, both of whom walked us through the **PRESTO** data processing pipeline, along with Maura McLaughlin of West Virginia University, who provided us with recent data from the Arecibo radio telescope. After vigorously developing, testing, and debugging our scripts by searching for existing pulsars in Parkes Multibeam Pulsar Survey data, we ultimately processed Maura's data using our scripts to look for signals of the known pulsar **J0509+08** [8] among other plausible new signals.

Contents

1 Introduction to Pulsar Astronomy

Pulsars, which are strongly magnetized neutron stars, are remnants of main sequence stars (exceeding eight solar masses) that exhaust their fuel and explode in supernovae. After their cores collapse, they rotate at high speeds due to the conservation of energy, and emit periodic pulses of electromagnetic waves that can be detected by radio telescopes such as FAST in Guizhou, Arecibo in Puerto Rico, and the Parkes Observatory telescope in New South Wales.

Although all neutron stars rotate and emit periodic signals, pulsars is the name given to those with signals detectable from Earth and allow us to explore certain galactic phenomena. For instance, the stringent periodicity of pulsar photon emission, which occasionally rivals even the precision of atomic clocks, can be used to measure and compare timing positions [6]; international efforts have combined to form pulsar timing arrays, such as NANOGrav, which aims to detect gravitational waves by discerning minute differences between pulsar times [10]. Furthermore, analyses of pulsar signals' propagation (including dispersion, scintillation, and scattering) through space can reveal key characteristics of the interstellar medium (ISM) that they travel through, and pulsars' unique physical conditions provide vital information about the galaxy's gravitational potential, magnetic field, and evolution as well [6].

After Burnell and Hewish's discovery of the first pulsar in 1967 (which eventually led to a 1974 Nobel Prize), numerous breakthroughs in pulsar astronomy have propelled our scientific understanding of the universe. The discovery of the first binary pulsar system by Hulse and Taylor first experimentally indicated the existence of gravitational waves; the discovery of a pulsar within a globular cluster by Lyne opened doors to further exploration of globular clusters, including the revelation of ionized gas in 47 Tucanae; the detection of pulsar planetary systems, double pulsar systems, and triple systems involving pulsars are all noteworthy advancements that elucidate the diverse scenarios available for us to explore [6].

To date, the ATNF Pulsar Catalog, which indexes all published rotation-based pulsars, documents upwards of 2600 distinct objects, and as major pulsar surveys and individual pulsar searches are conducted every day, the number will continue to grow as more data is processed and publicized. The expansion and development of the known pulsar database are imperative to the advancement of our galactic knowledge, since as more pulsars become recognized, the possibility of illuminating previously unknown phenomena is amplified. For example, fast radio bursts (FRBs), which have been detected in pulsar signals and are of currently unidentified origin, are being intensively researched in the pulsar community with the potential to uncover vital information [1]. Efforts to streamline the pulsar documenting procedure can also increase the efficiency of data processing; Hui Zhang et al. at GZNU are currently creating an online database using SQL that effectively organizes large amounts of international pulsar data, compares search results to existing statistics, and presents information in a user-friendly graphical interface.

Without a doubt, our project in China has not only the potential to uncover new scientific knowledge, but also provides us with indispensable experience in operating scientific machines, understanding the vast areas of pulsar astronomy (in particular the intricate pulsar searching process), and working alongside each other and experienced personnel while efficiently managing our limited time. Although pulsar searching is a rather specific branch of astrophysics, it is not unlikely that we will conduct it again in the future while moving on in the field (for pulsar science continues to be a highly relevant subject), and our skills in developing and debugging code, operating a Unix shell, visually processing substantial amounts of data, and finally producing a professional written report can be exceedingly applicable in countless other academic areas as well.

2 Technical Background

At the beginning of our project, Maura sent us a total of 29 files taken by the Arecibo telescope, dated back to May of 2010, each of them covering 129 seconds of a single drift scan. While our final goal was to process those files with scripts that ran different PRESTO processes on them, our code developing experience also involved processing a number of older files originating from PMPS. Due to the fact that drift scan surveys involve continuous recording of data across a certain area in the sky, resulting data files, which also often overlap each other to ensure comprehensive analysis, tend to be very large, and hence very time- and resource-consuming to analyze. Maura's raw data took a week or so just to transfer completely, taking up upwards of 40 gigabytes on the disk. Meanwhile, on the GZNU server where we ran our scripts, each 30 second section that we cut from Maura's data took about four and a half days to completely process while the PMPS data took about a day less.

Needless to say, the sheer volume of data that results from a single data collection operation requires more than just regular computing equipment. By running VNC servers on our local machines, we were able to gain remote access to the industrial research computing nodes located at NAOC and the GZNU campus, which enable high processing speeds with higher quality and quantities of computing cores. In Beijing, we were able to ssh into node number four (n04) on the Orion server, which has significantly more memory and CPU power than the others, allowing us to cut processing times. In GZNU, we gained access to an online job submitting system called JobGrid, and although we were unable to use it due to unexpected errors, it was able to run processes faster with access to more remote nodes. Even with industrial machinery, the process remains tedious; to debug our code after making changes, we would have to wait for the script to run from start to finish (or terminate at a runtime error) before going back and checking the logs and output. Thus, fixing even a couple of lines of code could take days, or even weeks.

Hence, developing elegant ways to optimize code and computing capabilities is of vital importance in the field of pulsar searching and astrophysics. Individual personal comput-

ers are out of the question for even a simple project of this magnitude; while my simple Macbook Pro has four CPU cores and 16GB of memory, the nodes on the GZNU server has 32 cores and 120GB of memory. A simple machine like the former would be unable to handle even the installation of PRESTO programs, thus rendering large research institutions with a wider range of resources (such as NAOC) or a distributed computing system (like Einstein@Home [7]) as the only options for processing telescope data. On these platforms, the ability to optimize operations and run them simultaneously is just as important as the hardware required to do so; we have experimented with parallel processing by writing separate code to run our scripts on several files simultaneously and speed up the process, but to truly optimize computing power, we may incorporate more complex procedures such as completing the same steps for each process in parallel instead of the entire script at once, splitting the script up to send to different nodes, and letting the machines calculate the optimal amount of threads to run given the environment.

As scientific exploration in astrophysics has greatly accelerated in the past 20 years, the necessity of more elaborate and powerful computing equipment has been growing just as fast. As the amount, rate, and ease of data collection increases, the effort needed to analyze it grows manifold. As such, there is a burgeoning demand for more sophisticated computing clusters, stronger servers, and exceptional computational skills to synthesize the components.

3 Pulsar Searching Using PRESTO

3.1 PRESTO Installation

PRESTO, which stands for PulsaR Exploration and Search TOolkit, is a collection of computational tools used to look for pulsars developed by Scott Ransom [12]. In our project, we installed PRESTO on individual NAOC and GZNU nodes, wrote scripts in Bash to run PRESTO processes on raw telescope data, and looked through the resulting candidate plots for possible pulsars. The first week of our time at NAOC was dedicated to the installation of all the packages required for PRESTO to run correctly, which was an imperative procedure to ensure that the rest of the processes run smoothly. Below is a documentation of the steps we underwent to install PRESTO on the remote machines:

First, create a folder in the home directory for all the presto packages:

```
> mkdir presto
```

3.1.1 .bashrc Configuration

Before installing anything, make sure that the hidden file `.bashrc` in the home directory contains the correct definitions and functions to ensure that PRESTO can run smoothly. Change the file as so:

```
# .bashrc

# Source global definitions
if [ -f /etc/bashrc ]; then
. /etc/bashrc
fi

# Uncomment the following line if you don't like systemctl's auto-paging feature:
# export SYSTEMD_PAGER=

# User specific aliases and functions

export PRESTO=~/presto/presto
export TEMPO=$PRESTO/../tempo
export PGPLOT_DIR=$PRESTO/../pgplot-exe
export PATH=$PATH:$PRESTO/bin:$PRESTO/../tempo-exe/bin
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$PRESTO/lib:$PRESTO/lib64:$PRESTO/../fftw-exe/lib:$PRESTO/../
   pgplot-exe:$PRESTO/../cfitsio-exe/lib
export PKG_CONFIG_PATH=$PKG_CONFIG_PATH:$PRESTO/../cfitsio-exe/lib/pkgconfig:$PRESTO/../fftw-exe/lib/
   pkgconfig
export PYTHONPATH=$PYTHONPATH:$PRESTO/lib/python:$PRESTO/lib64/python
ld -shared -o $PGPLOT_DIR/libcpgplot.so --whole-archive $PGPLOT_DIR/libcpgplot.a

export RPPPS_DIR=/home/pzc/RPPPS_3.1
```

For the following installation processes for PRESTO packages, note that the configuration directory varies by the server; we installed to the *nfshome* directory on the NAOC server, and the *home* directory for the GZNU server:

3.1.2 Tempo

The Tempo package provides the tools necessary to prepare data through interference detection, interference removal, and barycentering [12].

1. Create a directory in the base *presto* folder (the one in the home directory) for the **Tempo** package:
   ```
   > mkdir tempo-exe
   > cd tempo-exe
   ```

2. In the command line, type:
   `> git clone git://git.code.sf.net/p/tempo/tempo`
   to clone over the GitHub **Tempo** package compressed in the .tar.gz format.

3. Unpack the package using the command:
   `> tar xvzf [filename]`

4. Use the following commands to configure and install the package:
   ```
   > ./prepare
   > ./configure
   --prefix=/home/[username]/presto/tempo-exe
   > make
   > make install
   ```

3.1.3 CFITSIO

CFITSIO allows the manipulation of FITS (Flexible Image Transport System) files in the C and Fortran languages [9].

1. Download the required packages for the system on the CFITSIO website:
   `https://heasarc.gsfc.nasa.gov/fitsio/fitsio.html`

2. Extract metadata using the command:
   `> pkg-config --libs cfitsio`

3. In the base *presto* folder, create a folder for the program:
   ```
   > mkdir cfitsio-exe
   > cd cfitsio-exe
   ```

4. Configure and install the program:
   ```
   > ./configure --prefix=/home/[username]/presto/cfitsio-exe
   > make
   > make install
   ```

3.1.4 LaTeX2HTML

LaTeX2HTML allows the conversion of LaTeX documents to HTML web pages [5].

1. Retrieve the most recent version of the LaTeX2HTML package on CTAN or GitHub on its homepage:
   `http://www.latex2html.org/`

2. Create a directory in the base *presto* folder for the program:
   `> mkdir latex2html-exe`

3. Go into the folder for the downloaded package. The folder name may vary depending on the most recent version's release date:
   ```
   > cd latex2html-2017.2/
   ```

4. Configure and install the package:
   ```
   > ./configure --prefix=/home/[username]/presto/latex2html-exe
   > make
   > make install
   ```

3.1.5 FFTW

FFTW is a subroutine library that computes the discrete Fourier transforms of data.

1. Download the most recent version of the package on the homepage:
   `http://www.fftw.org/`

2. In the base *presto* directory, create a new folder for the program:
   ```
   > mkdir fftw-exe
   ```

3. Go into the folder for the downloaded package. The folder name may vary depending on the most recent version's release date:
   ```
   > cd fftw-3.3.6-pl2/
   ```

4. Configure and install the package:
   ```
   > ./configure --prefix=/home/[username]/presto/fftw-exe --enable-
   shared --enable-single --enable-sse2 --enable-avx
   > make
   > make install
   ```

3.1.6 PGPLOT

PGPLOT is a graphics subroutine library that allows the creation of scientific graphs.

1. Download the most recent version of the package on the homepage:
   `http://www.astro.caltech.edu/ tjp/pgplot/`

2. Unpack the package using the command:
   ```
   > tar xvzf [filename]
   ```

3. In the base *presto* directory, create a folder for the package:
   ```
   > mkdir pgplot-exe
   > cd pgplot-exe
   ```

4. Copy over the *drivers.list* file containing driver information:
   ```
   > cp ../pgplot/drivers.list ./
   ```

5. Edit the *drivers.list* using Gedit or any other text manipulation application. `> gedit drivers.list`

   Remove the "!" in front of the following lines:
   ```
   PSDRIV 1 /PS PostScript printers, monochrome, landscape Std F77
   PSDRIV 2 /VPS Postscript printers, monochrome, portrait Std F77
   PSDRIV 3 /CPS PostScript printers, color, landscape Std F77
   PSDRIV 4 /VCPS PostScript printers, color, portrait Std F77
   TTDRIV 5 /XTERM XTERM Tektronix terminal emulator Std F77
   XWDRIV 1 /XWINDOW Workstations running on X Window System
   ```

6. Edit *makefile* to make it compatible with Fortran compiling on the machine:
   ```
   > /home/[username]/presto/pgplot/makemake /home/[username]/presto/pgplot
   linux g77_gcc
   > gedit makefile
   ```
   Change the line:
   `FCOMPL=g77`
   to:
   `FCOMPL=gfortran`

7. Clean and make the package to finish installation:
   ```
   > make
   > make clean
   > make cpg
   > ld -shared -o libcpgplot.so --whole-archive libcpgplot.a
   ```

3.1.7 PRESTO

This is for Scott Ransom's PRESTO package that has all the necessary commands to search for pulsars.

1. In the base *presto* folder, make a folder for the package that is also named *presto*:
   ```
   > cd presto
   > mkdir presto
   ```

2. In the command line, type:
   ```
   > cd presto
   > git clone git://github.com/scottransom/presto.git
   ```
   to clone over the GitHub PRESTO package compressed in the .tar.gz format.

3. Unpack the package:
   `> tar xvzf [filename]`

4. Configure FFTW packages:
   ```
   > pkg-config --libs fftw3f
   ```

5. Go into the *presto* subfolder for this package to configure and make:
   ```
   > cd presto
   > cd src
   > make makewisdom
   > make
   ```

6. Edit *setup.py* for python compilation support:
   ```
   > cd ../python
   gedit setup.py
   ```

   Add the bolded parts in the following lines of the file:
   ```
   ppgplot_libraries = ["gfortran", "cpgplot", "pgplot", "X11", "png",
   "m"]
   include_dirs = ["/home/[username]/presto/fftw-exe/include"]
   presto_library_dirs = ["/home/[username]/presto/fftw-exe/lib"]
   ```

7. Edit the *Makefile* file to add Fortran support:
   ```
   > gedit Makefile
   ```
   Add the bolded section to the following line in the file:
   ```
   cd fftfit_src ; f2py -c fftfit.pyf *.f --f77exec=gfortran
   ```

8. Finally, run the make command again to complete installation:
   ```
   > make
   ```

3.2 PRESTO Pipeline

While the entire PRESTO pulsar searching pipeline consists of numerous intricate sequences, we applied the most relevant processes to our research, which involves the following steps in this section. The specific code I wrote is attached in the Appendix; Appendix A1 includes the script I ran on Einstein@Home data, and Appendix A2 contains the alternate version that I ran on Arecibo data.

3.2.1 Preparing Files

Since the raw data files are very large and are stored in a shared server space, it is impractical to copy over all the files to a local directory to proceed with the data processing. Hence, we make links to the remote files on the nodes, so that we can access the data and perform processes on it locally.

To do so, we first require the directories and file names of the original data files as parameters in our script. We then add a layer of error checking: if the directory or file does not exist, we exit the program; if the given file has already been processed in the same local directory, we remove the folder containing the old results, effectively overwriting it with data from the new procedure. The latter is particularly useful when repeatedly processing a single file for testing purposes while debugging.

After ensuring that we can safely proceed, we create a folder to store the raw data, intermediate files, and results, and then create a link to the raw data file given by the program's parameters within the folder using the `ln -s` command.

3.2.2 Search for RFI

The first step in processing the file is identifying RFI (radio frequency interference) that may disguise itself as pulsar signals in the data. Although a lot of RFI cannot be successfully filtered and goes through into the results, the `rfifind` command does a satisfactory job of locating strong RFI and generating a "mask" for it to filter it out, given a specific integration time [13].

3.2.3 Make De-dispersion Plan

Dispersion of time-dependent data during its collection needs to be corrected in order to gain accurate interpretations of astral events. The phenomenon occurs due to signals' propagation in the ISM, which delays their interaction with the observer. The problem becomes complex, as the delay is not constant throughout the entire DM spectrum, since higher DMs indicate further distance - signals with higher DMs travel through more ISM and are delayed further than those that precede them, causing "smearing" across the spectrum, which can significantly impact sensitivity. Running `DDplan.py` on the raw data generates a "plan" to de-disperse the signals based on their varying DM bands, neutralizing the effects of "smearing". Figure 1 below shows an example of how `DDplan.py` constructs a de-dispersion plan based on the lowest DM, highest DM, center frequency of the telescope, number of channels of the telescope, the telescope's bandwidth, and the sample time:

```
[shanali@orion PM0061]$  DDplan.py -l 0 -d 4000 -f 1374 -n 96 -b 288 -t .00025

Minimum total smearing     : 0.354 ms
------------------------------------------
Minimum channel smearing   : 0 ms
Minimum smearing across BW : 0.00461 ms
Minimum sample time        : 0.25 ms

Setting the new 'best' resolution to : 0.25 ms
Best guess for optimal initial dDM is 0.543

  Low DM    High DM     dDM  DownSamp   #DMs  WorkFract
   0.000     88.000    0.50       1    176   0.7325
  88.000    151.000    1.00       2     63   0.1311
 151.000    289.000    2.00       4     69   0.0718
 289.000    639.000    5.00       8     70   0.03642
 639.000   1279.000   10.00      16     64   0.01665
1279.000   2559.000   20.00      32     64   0.008324
2559.000   4029.000   30.00      64     49   0.003187
```

Figure 1: DDplan generated using parameters of the Parkes telescope.

3.2.4 Subband De-dispersion

After determining the de-dispersion plan, the signals are de-dispersed by using the `prepsubband` command. Based on the parameters given by `DDplan.py`, `prepsubband` shifts the arrival times of each distinct channel for each particular DM a calculated amount. Figure 2 below indicates graphically how this is done [13]:

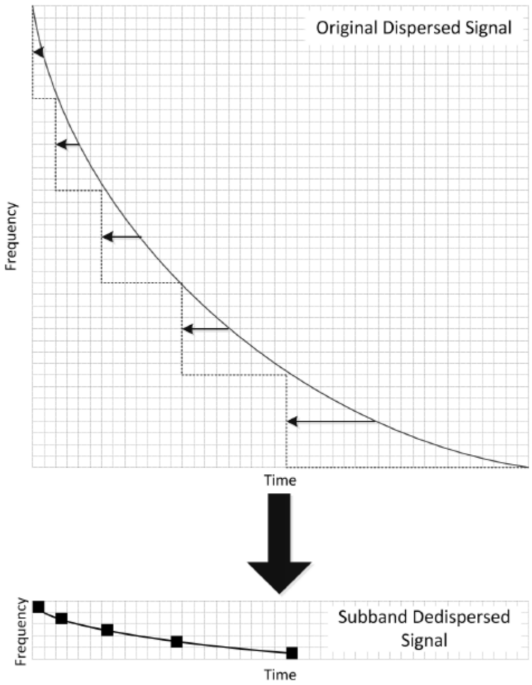


Figure 2: How *prepsubband* de-disperses data using the de-dispersion plan. [13]

3.2.5 FFT Time Series

To prepare to search the data, fast Fourier transform must be completed on the time series to convert it to its frequency domain. Doing so allows graphs to be made of periodic pulsar signals so that we can look through them for possible pulsars. We use Bash's `xargs` command to easily run PRESTO's `realfft` process on all of the `.dat` files in the folder.

3.2.6 Search for Periodic Signals

After conducting FFT on the data, the time series can be searched for identifiable pulsar attributes, using "a Fourier domain acceleration search with harmonic summing" [11]. To do so, we run `accelsearch` while specifying `zmax`, the maximum number of Fourier bins of the highest harmonic.

3.2.7 Sift Through Candidates

After searching, we use `Accelsift.py` to sift through the periodic candidates, rejecting unlikely candidates based on how well they adhere to pulsar attributes. The results are exported onto a `.txt` file, which is human- and machine-readable.

3.2.8 Fold Best Candidates

When the best pulsar candidates are highlighted, we can fold the candidates' data using `prepfold` to generate plots of the periodic pulse power, phase, and period, so that we may identify the ones that show the strongest pulsar attributes to check for existing pulsars or discover new ones.

3.2.9 Examine Plots

The pipeline can result in hundreds and thousands of plots, but only a small percentage of them may be actual pulsar candidates, with the rest being just RFI. After the data is processed, we can convert the plots from their original PostScript format to the more portable PNG format, and manually look through them. To identify strong pulsar candidates, we look for large pulses in the pulse profiles that indicate high pulse power, clear, dark lines straight down the phase diagrams that suggest unwavering pulses, and a bell-shaped DM graph.

4 Einstein@Home Data Processing

After developing a completely functional Bash scripts that run through the entire pipeline, we tested their capabilities on existing data to ensure that we proceeded with accurate and streamline code. We referenced Knispel et al.'s paper documenting their discovery of 24 new pulsars by re-processing PMPS data using the distributed computing project Einstein@Home, which listed the pulsar names, their R.A.s, declinations, periods, DMs, and other relevant information, along with the names of the PMPS files they appeared in [4]. We processed those files with our scripts, and attempted to re-discover those pulsars.

4.1 Results

Table 1 below lists the attributes of the pulsars discovered by Knispel et al. using Einstein@Home, as well as those of the same pulsars I re-discovered by processing the same PMPS data using my script. I was able to successfully find a majority of the pulsars, with most of them having attributes within a probable enough range of the Einstein@Home values to determine that they are the same objects, although some of them have numbers that are further off and require explanation. However, the overall success and accuracy of my results were sufficient for exploration with the Arecibo data. Besides the re-discovery of those existing pulsars, I picked up other pulsar signals, which I matched to information on the ATNF Pulsar Database on existing pulsars, documented in Table 2 [3]. The values labeled with *PRESTO* are the values that my program output, the ones labeled *E@H* are those recorded by Knispel et al., and those labeled *psrcat* are those documented on the ATNF database.

File Name	Jname	DM (PRESTO)	Period (s) (PRESTO)	RA (PRESTO)	Decl. (PRESTO)	DM	Period (s) (E@H)	Possible reasons for discrepancies
0026_00511	J0811-38	330.70	0.4825952	08:11:40	-38:57:28	336.20	0.4852594	
0058_036D1	J1227-62081	364.00	0.0345196	12:27:34	-62:10:25	363.20	0.034529685	
0109_00331	J1305-66	319.00	0.1972759	13:05:38	-66:39:25	316.10	0.1972763	
0001_00161	J1322-62	719.00	1.0448551	13:22:53	-63:20:37	733.60	1.044851	
0038_01821	J1455-59	499.00	0.1761933	14:55:05	-59:22:60	498.00	0.1761912	
0042_00391	J1601-50	68.83	0.8607696	16:01:25	-50:23:28	59.00	0.860777	
0137_041B1	J1619-42	299.00	1.0231580	16:19:05	-42:02:22	172.00	1.023152	Possible issue with de-dispersion/ACCEL_sift
0125_077C1	J1626-44	N/A	N/A	N/A	N/A	269.20	0.3083536	
0039_00551	J1637-46	636.28	0.4930971	16:37:36	-46:13:03	660.40	0.493091	
0056_020B1	J1644-44	559.00	0.1739100	16:44:35	-44:09:34	535.10	0.1739106	
0035_02931	J1644-46	472.89	0.4550680	16:44:05	-46:25:43	405.80	0.2509406	
0085_02541	J1652-48	N/A	N/A	N/A	N/A	187.80	0.003785124	Binary MSP, difficult to find with these parameters
0054_015A1	J1726-31	275.00	0.1234700	17:26:33	-31:56:48	264.40	0.12347018	
0102_00591	J1748-3009	419.00	0.0096843	17:48:34	-30:10:54	420.20	0.009684273	
0002_00891	J1750-2536	179.00	0.0347492	17:50:27	-25:35:48	178.40	0.034749053	
0141_00971	J1755-33	241.00	0.9594445	17:55:07	-33:25:25	266.50	0.959466	
0137_039B1	J1804-28	225.00	1.2730080	18:04:45	-28:06:38	203.50	1.273011	
0149_01081	J1811-1049	299.00	2.6238450	18:11:23	-10:47:21	253.30	2.623858562	
0011_03231	J1817-1938	N/A	N/A	N/A	N/A	519.60	2.046837629	Period too long, so can't be found with FFT
0148_01971	J1821-0331	N/A	N/A	N/A	N/A	171.50	0.902315629	File not on server
0132_06271	J1838-01	N/A	N/A	N/A	N/A	320.40	0.1832948	
0140_00641	J1838-1849	161.00	0.4882397	18:38:21	-18:48:11	169.90	0.488242009	
0060_02061	J1840-0643	494.00	0.0355779	18:40:09	-06:50:34	500.00	0.035577876	
0143_00511	J1858-0736	193.00	0.5510591	18:58:37	-07:36:60	194.00	0.551058591	

Table 1: Table of attributes of pulsars discovered by Knispel et al. that I found myself in the same PMPS data.

File Name	Jname (PRESTO)	DM (PRESTO)	Period (s) (PRESTO)	RA (PRESTO)	Decl.	DM (psrcat)	Period (s)
0039_00551	J1640-4648	478.08	0.2275340	16:37:36	-46:13:03	474.00	0.178352043
0042_00391	B1557-50	264.02	0.1926016	16:01:25	-50:23:28	260.56	0.192601233
0058_036D1	J1244-6359	308.24	0.1498982	12:27:34	-62:10:25	286.00	0.14727431
0102_00591	B1746-30	509.00	0.6098765	17:48:34	-30:10:54	509.40	0.609873649

Table 2: Table of attributes of earlier existing pulsars that I found in the PMPS data.

4.2 Rationale for Undiscovered Pulsars

The reprocessing of existing telescope data that has already been worked on by others is a common practice in the astrophysics field, and is in no regard an unnecessary procedure. Similar in outcome to the work done by Knispel et al., new discoveries often come out of reworking old data. An obvious reason for this is the rapid development of technology, since as computing equipment become more complex and capable, it becomes more efficient to process large amounts of data, and more possible to use more inclusive but time consuming parameters on computer functions that can illuminate a lot more in the same files than before.

Another prominent reason for the fruitfulness of reworking data is the fact that pulsar candidate identification is often a highly subjective process. After PRESTO is run on the raw files, humans are required to manually sift through heaps of images in search for a small proportion of strong pulsar candidates. Varying levels of experience and fatigue can impact human judgment, and the ranking of candidates based on how well they display pulsar attributes can be significantly different based on the individual in front of the screen. Ultimately, it is always the strongest pulsar signals that are submitted for further review, and the lower ranking ones are dismissed. However, the reprocessing of the same data by different people at a later time can bring the less visually apparent pulsar candidates to light, and many profiles that were not ranked highly before can be submitted and confirmed.

Pulsar science is still a field that requires a notable amount of human labor. As it develops, machines will be involved to a greater degree, decreasing the margin of human error and increasing efficiency. However, the fact that tens of terabytes of data are taken by telescopes per day makes that very difficult at this stage; in the future, though, it is probable that technology in this field can be advanced enough to involve artificial intelligence in the ranking of pulsar candidates, so that it becomes a more exact and objective process.

5 Arecibo Data Processing

5.1 Introduction to Drift Scanning

Drift scanning is a telescope data collection technique that allows the observation of a substantial area of the sky without moving the telescope at all. The telescope is turned on and left to observe its focal plane continuously for a period of time, hence imaging continuous portions of the sky as the Earth rotates. Drift scans are greatly useful in that they require relatively little effort to set up, allow the observation of strips of the sky that are kilometers long within a short time period, and allow us to effectively survey for objects, such as pulsars, across a sizable area. [2]

Both the PMPS and Arecibo data are results of telescope drift scans, but a more intricate process was applied to our evaluation of the Arecibo files, described in the subsections below.

5.2 Cutting and Combining Raw Data

Since the survey strips were quite long, we measured and cut them into smaller portions with a precise length of overlap to make our analyses as comprehensive as possible. The following are the steps we undertook to do so:

5.2.1 Calculating Transition Time

The transition time of a telescope is the time it takes for a point entering the telescope frame from one side to travel through and exit the frame. Using the `readfile` command, we obtained the central frequency of the telescope, which we used to calculate the wavelength using the ratio $c = f\lambda$ for wave phenomena. Then, we calculated the beam size of the telescope by dividing the wavelength by the telescope collection diameter and converting it from radians to degrees. Finally, we took a ratio with respect to 360 degrees in 24 hours to get the time it takes for a point in space to pass through that beam length given the speed of rotation of the Earth. We came up with about 72 seconds for the transition time, which we truncated to just 72. The transition determines how long each newly cut strip will be.

5.2.2 Calculate Times for Shorter Strips

We decided to overlap a third of each strip, in other words 24 seconds of each 72-second cut file. So, viewing all the original 129-second files as sequential parts of a very long survey, we would cut the subints of the survey starting from 0-71, 25-96, 49-120, etc. (note that the cutting and combining program takes in time values inclusively, so each raw file has seconds 0-128 and is 129 subints, or seconds, long).

5.2.3 Determine Strips to Cut and Combine

Since each file is only 129 subints long, when the subint number required for our strips exceed 129, we would have to combine sections from the next strip. So, we would cut the first few strips in the following way:

Strip 1:
1-72
25-96
49-120
73-144 (Since 144 is greater 129, combine 73-129 from strip 1 and 1-15 from strip 2)
97-168 (Combine 97-129 from strip 1 and 1-39 from strip 2).

Appendix A3 contains my code that calculates the times, cuts, and combines the raw Arecibo files.

5.3 Known Pulsars

After successfully processing all the strips we cut from the raw Arecibo files, and in the process encountering a couple of errors and speed bumps along the way, we were finally able to identify the existing pulsar J0509+08, which Maura confirmed was in the data. Figure 3 below is the plot of the pulsar that my program output:

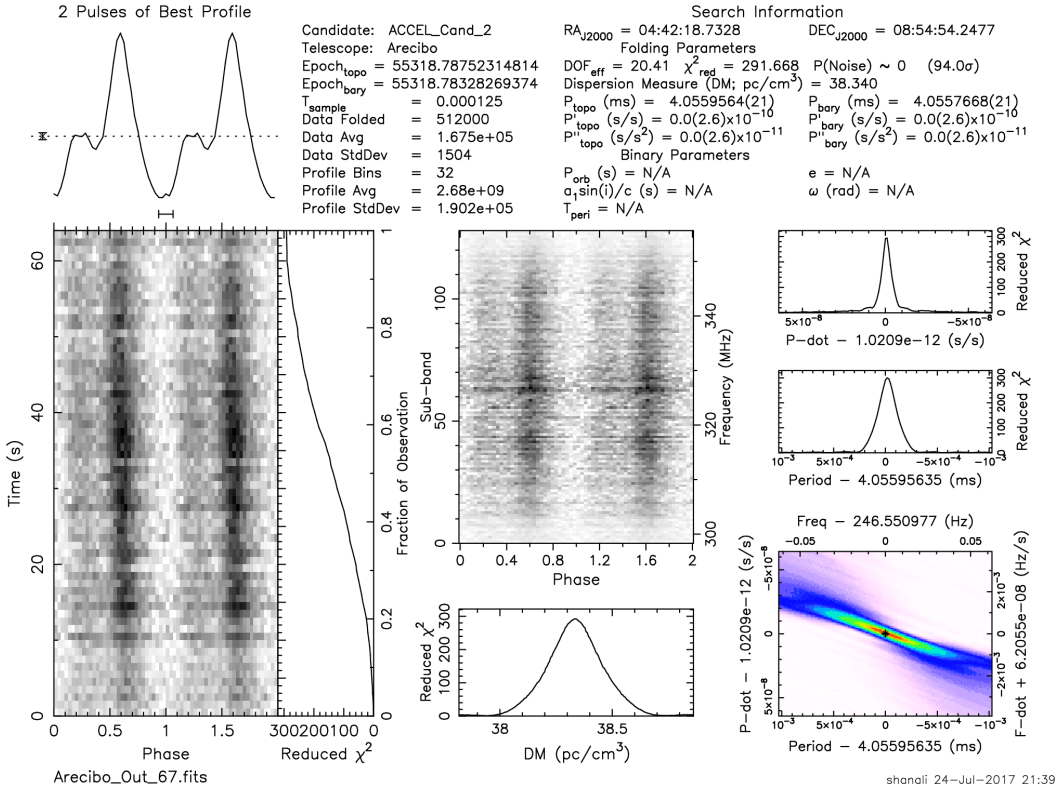


Figure 3: The pulse profile plot of J0509+08 that my script output in the end.

According to the Arecibo Observatory documentation of new drift search discoveries, J0509+08, which was initially discovered in 2013, is a binary millisecond pulsar with a 4.06ms and a DM of 38.3. My profile showed a period of about 4.05577ms and a DM of 38.34, which adheres to the original drift scan's results (but with more precision). [8]

5.4 New Pulsar Candidates

Apart from rediscovering an existing pulsar, I also came across a few profiles that somewhat display pulsar characteristics, although they are quite weak and visually nowhere as apparent as the existing pulsar's profile, but may be worth further consideration. They are displayed in Figures 4 to 6 below:

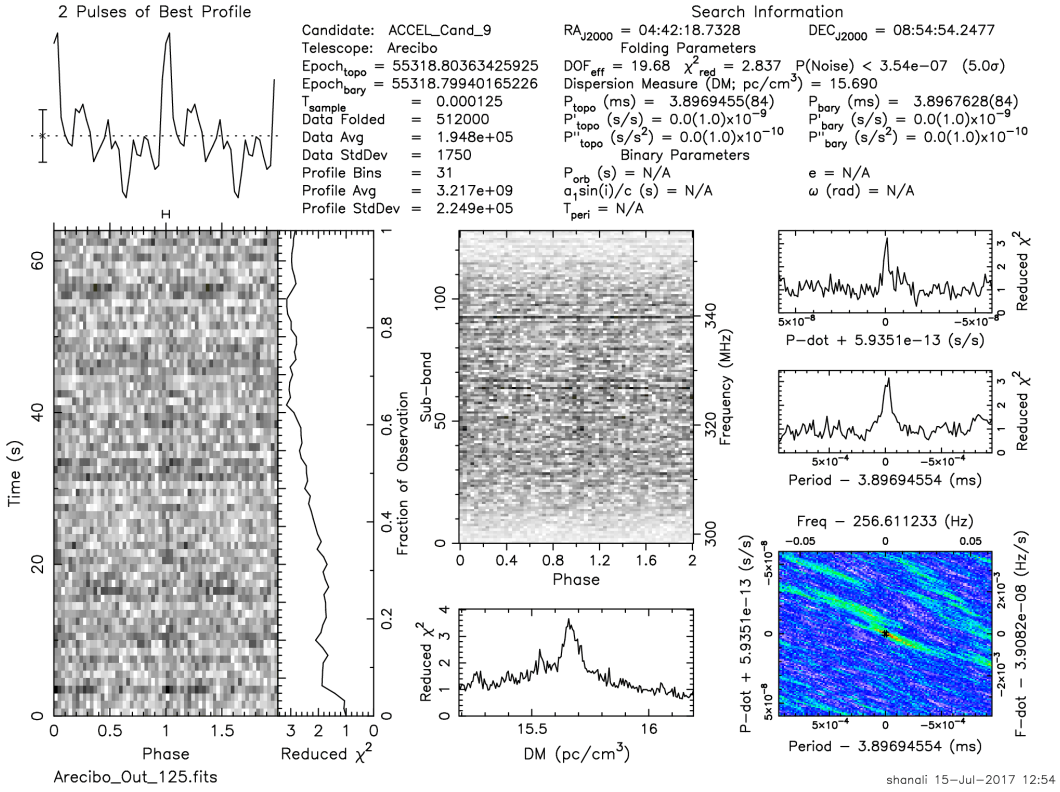


Figure 4: A pulse profile that resembles a pulsar signal. This is the strongest one of the candidates I found, and is most worthy of further exploration.

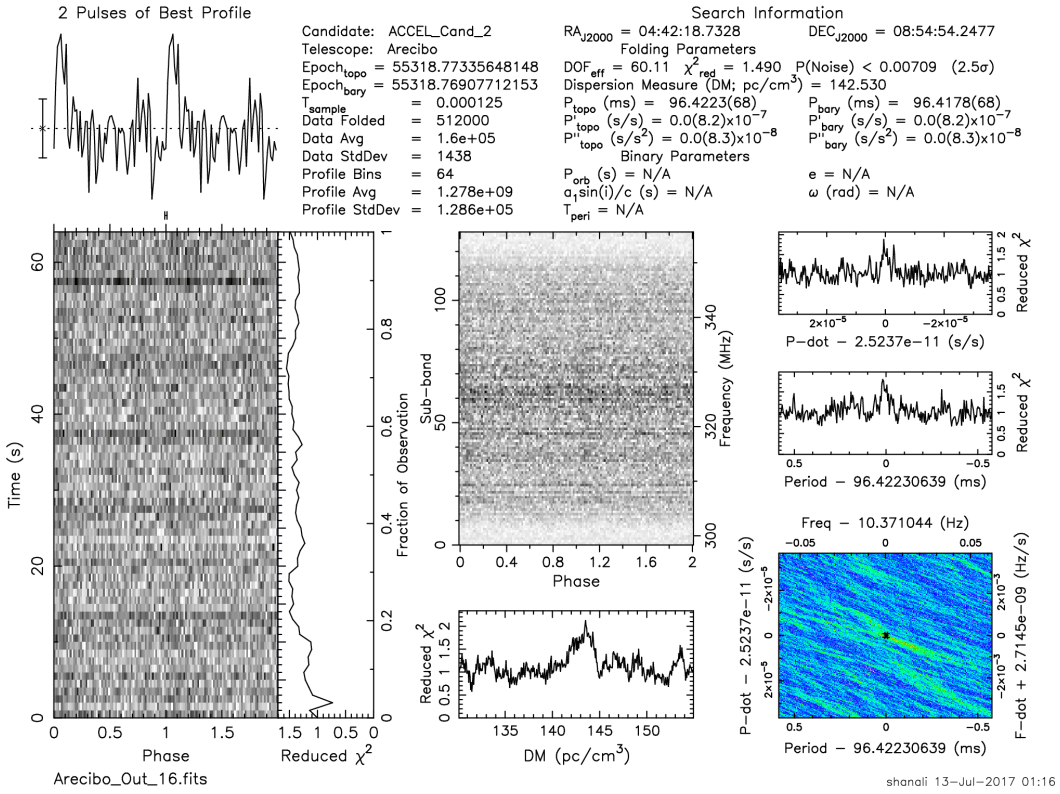


Figure 5: A weaker pulse profile that somewhat displays pulsar characteristics, but is most likely just RFI.

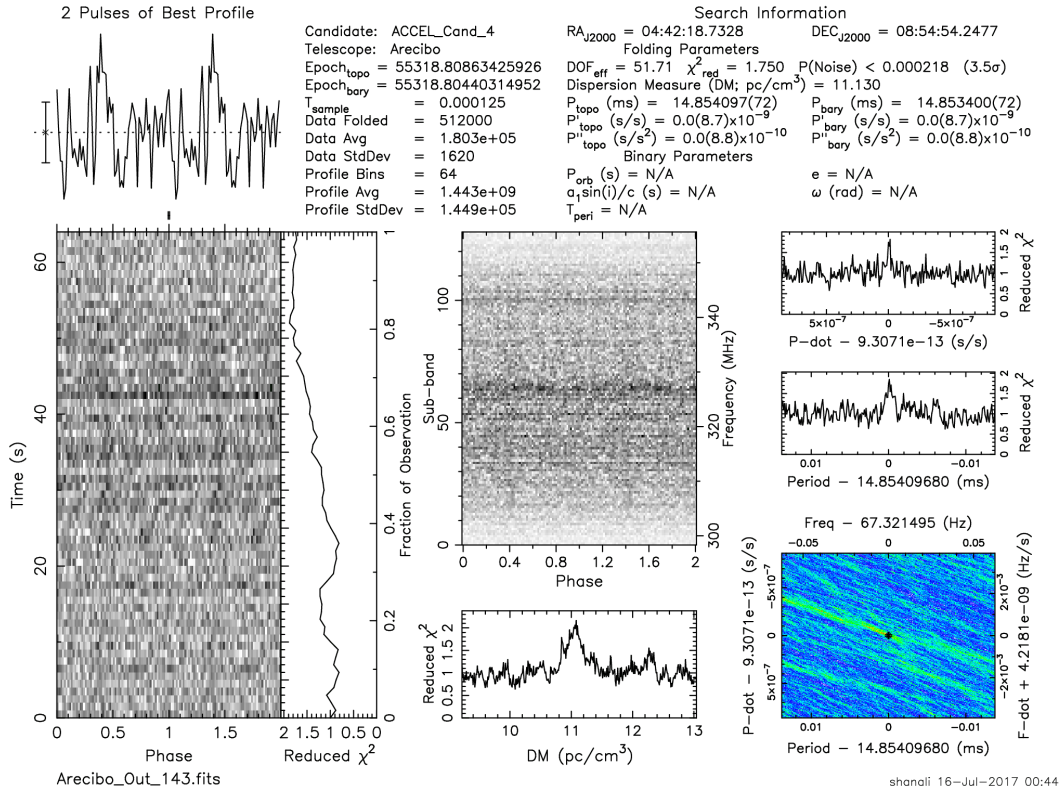


Figure 6: Another weak pulse profile that somewhat displays pulsar characteristics, but is most likely just RFI.

6 Conclusion and Further Exploration

Overall, this trip was an incredible and unforgettable experience for me, both in terms of visiting various places in China and being immersed in local cultures, and also gaining invaluable knowledge about scientific research. Apart from attaining practical workplace experience from working alongside coworkers and professionals in the field, I also obtained a breadth of experience working with scripting code, complex servers, and high-end computing machines. Both branches of knowledge can add to my work in physics and other subjects, and has heightened my understanding and interest of the pulsar astrophysics field.

Although our time in China has come to an end, this project can be extended into our time in our respective institutions, since there is an abundance of it that can be explored

further. Firstly and most importantly, we should try to optimize the way our code is run and minimize the running time, since CPU power on the server is shared and there is an enormous amount of data that will need to be processed at once. We have already tried to run processes in parallel (my attempt at it is attached in Appendix A4), but we only ran the script multiple times on multiple files simultaneously. A smarter and more efficient way to parallelize the processes would be to run each PRESTO command simultaneously and allowing the machine to determine the amount of threads to run at a time, but we are yet to grasp how to do so.

Next, it would be a good idea to re-process the entire set of cut Arecibo data, since due to a glitch in our code that halted the process partly through each of the files, we only had enough time remaining to fully process the strip that we knew the known pulsar would be in. Re-processing all the data would take an extremely long time and cost quite a lot of computing power, so the code should be optimized before we start to do so.

Finally, we could look into the new pulsar candidates that we find, with the guidance of our supervisors at our institutions. After determining the profiles that are worthy of being revisited, we should compare their values with ones on ATNF's pulsar database, and if there are no matches, it would be possible to reserve telescope time in hopes of new discoveries coming to light.

7 References

[1] Adam Mann: Core Concept: Unraveling the enigma of fast radio bursts, *https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5380068/*

[2] ATNF: ATNF Pulsar Catalogue, *http://www.astrosurf.com/audine/English/result/scan.htm*

[3] Astrosurf: THE DRIFT-SCAN TECHNIQUE, *http://www.naic.edu/ deneva/drift-search/*

[4] B. Knispel et al. *EINSTEIN@HOME DISCOVERY OF 24 PULSARS IN THE PARKES MULTI-BEAM PULSAR SURVEY.* The Astrophysical Journal, 774:93, 2013.

[5] Dan Gildea: LaTeX2HTML, *http://www.latex2html.org/*

[6] D.R. Lorimer, M. Kramer. *Handbook of Pulsar Astronomy.* Cambridge University Press, 2005.

[7] Einstein@Home: Einstein@Home, *https://einsteinathome.org/*

[8] National Astronomy and Ionosphere Center (NAIC): Discoveries from the AO 327 MHz Drift Survey, *http://www.naic.edu/ deneva/drift-search/*

[9] NASA HEASARC: CFITSIO - A FITS File Subroutine Library, *https://heasarc.gsfc.nasa.gov/fitsio/fitsio.html*

[10] National Science Foundation: North American Nanohertz Observatory for Gravitational Waves, *http://nanograv.org/*

[11] Scott Ransom: PRESTO accelsearch on GitHub, *https://github.com/scottransom/presto/blob/master/docs/accelsearch.1*

[12] Scott Ransom: PRESTO Home, *http://www.cv.nrao.edu/ sransom/presto/*.

[13] Scott Ransom: Searching for Pulsars with PRESTO, *http://www.cv.nrao.edu/ sransom/PRESTO_search_tutorial.pdf*

8 Appendices

8.1 Appendix A - Scripts

8.1.1 Appendix A1 - Data Processing Script for E@H Data

```
#!/bin/bash
#A script to process a single file, involving making a link to said file and running
#rfifind, DDplan.py, prepdata, realfft, accelsearch, accelsift, and prepfold on it.
#don't forget to do this on n04 -X! (BJ server)
#modified for use on the GZNU server.

#command line arguments are in the format: ddscript.bash directory filename
#boolean to check whether or not everything is in the right format and if the
#directory and filename are valid

STARTTIME="$(date -u +%s)"

echo "************************************************************"
echo "\textbf{PRESTO} data processing"
echo -e "by Shana Li\n"
echo "Arguments should be in the format: directory filename."
echo "Filename should be sans file extension (.sf)."
echo "************************************************************"

#set variables for directory and filename
directory=$1
filename=$2

echo -e "\n************************************************************"
echo "Directory: $directory"
echo "Filename: $filename"

#if directory is invalid, exit
if [ ! -d $directory ]; then
        echo "Invalid directory."
        echo -e "************************************************************\n"
        exit 0
fi

#if file doesn't exist, exit
if [ ! -e "$directory/$filename.sf" ]; then
```

```
37          echo "Invalid filename."
38          echo -e "************************************************************\n"
39          exit 0
40  fi
41
42  #if folder for this particular file already exists, delete it
43  if [ -d $filename ]; then
44          echo -e "\nExisting files found, overwriting them now."
45          rm -r $filename
46          echo -e "Done.\n"
47  fi
48
49  #create new folder for data files
50  mkdir $filename
51  cd $filename
52
53  #create link to file in current directory in the folder
54  echo "Making link to file:"
55  ln -s $directory/$filename.sf ./ >> /dev/null
56  echo "Done."
57  echo -e "************************************************************\n"
58
59
60  echo "************************************************************"
61  #run rfifind
62  echo "Running rfifind:"
63  START="$(date -u +%s)"
64  echo -e "Start time: $START\n"
65  echo "Filename: $filename.sf"
66  echo -e "Time: 2 \n"
67
68  rfifind $filename.sf -time 2 -o $filename >> /dev/null
69
70  echo "Done."
71  END="$(date -u +%s)"
72  echo "End time: $END"
73  echo "Time for rfifind: $(($END - $START)) seconds."
74  echo -e "************************************************************\n"
75
76
77  echo "************************************************************"
78  echo "Running DDplan.py:"
```

```
START="$(date -u +%s)"
echo -e "Start time: $START\n"
#run DDplan.py:
#get variables needed:
#low dm
ldm=0
#high dm
hdm=4000
#time resolution
tres=0.5
#central frequency
cfreq=`readfile $filename.sf | grep "Central freq" | awk '{print $5}'`
#number of channels
numchan=`readfile $filename.sf | grep "Number of channels" | awk '{print $5}'`
#Parkes multibeam bandwidth
bandw=`readfile $filename.sf | grep "Total Bandwidth" | awk '{print $5}'`
#sample time
sampletime=`readfile $filename.sf | grep "Sample time" | awk '{print $5}'`
sampletime=0.000$sampletime

#run DDplan.py
echo "Central Frequency: $cfreq"
echo "Number of Channels: $numchan"
echo "Total Bandwidth: $bandw"
echo -e "Sample Time: $sampletime \n"

DDplan.py -l $ldm -d $hdm -f $cfreq -b $bandw -n $numchan -t $sampletime -r $tres
    -o $filename | tee ${filename}_ddplaninfo.txt >> /dev/null
echo "Done."
END="$(date -u +%s)"
echo "End time: $END"
echo "Time for DDplan: $(($END - $START)) seconds."
echo -e "************************************************************\n"

echo "************************************************************"
#subband de-dispersion: call prepsubband on each call in DDplan
#count the lines that contain the information for each call
#ignore the last line in DDplan
echo "Running subband de-dispersion:"
START="$(date -u +%s)"
echo -e "Start time: $START\n"
lastline=14
```

```
line=`head -$lastline ${filename}_ddplaninfo.txt | tail -1`
while [ "$line" != "" ]; do
        ((lastline++))
        line=`head -$lastline ${filename}_ddplaninfo.txt | tail -1`
done
echo -e "Looping prepsubband for $(( $lastline-14-1 )) calls.\n"

#loop prepsubband for all calls in ddplaninfo
for (( i=14; i<lastline-1; i++ ))
do
        #get the call from the current line in the file
        call=`head -$i ${filename}_ddplaninfo.txt | tail -1`

        #put call into an array
        IFS=' ' read -a arr <<<"$call"

        #get all the variables:
        #nsub
        nsub=`readfile $filename.fits | grep "samples per spectra" | awk '{print $5}'`

        #low dm
        ldm=${arr[0]}

        #dm step
        dms=${arr[2]}

        #number of dms
        ndm=${arr[4]}

        #downsamp
        ds=${arr[3]}

        #numout (numout from DDplan / downsamp)
        numo=$(( $numout / $ds ))

        #run prepsubband command
        echo "nsub: $nsub; Low DM: $ldm; DM step: $dms; Number of DMs: $ndm; Numout: $numo;
            Downsample: $ds"
        prepsubband -nsub $nsub -lodm $ldm -dmstep $dms -numdms $ndm -numout $numo -downsamp $ds
            -mask ${filename}_rfifind.mask -o $filename $filename.fits >> /dev/null
        echo -e "Done.\n"
done
```

```
END="$(date -u +%s)"
echo "End time: $END"
echo "Time for prepsubband: $(($END - $START)) seconds."
echo -e "************************************************************\n"


#run realfft:
echo "************************************************************"
echo "Running realfft:"
START="$(date -u +%s)"
echo -e "Start time: $START\n"
ls *.dat | xargs -n 1 --replace realfft {} >> /dev/null
echo "Done."
END="$(date -u +%s)"
echo "End time: $END"
echo "Time for realfft: $(($END - $START)) seconds."
echo -e "************************************************************\n"

#accelsearch for periodic signals
echo "Running accelsearch:"
START="$(date -u +%s)"
echo -e "Start time: $START\n"
ls *.fft | xargs -n 1 accelsearch -zmax 50 >> /dev/null
echo "Done."
END="$(date -u +%s)"
echo "End time: $END"
echo "Time for accelsearch: $(($END - $START)) seconds."
echo -e "************************************************************\n"


#sift through periodic candidates
echo "************************************************************"
echo -e "Running ACCEL_sift.py:"
START="$(date -u +%s)"
echo -e "Start time: $START\n"
python "$\textbf{PRESTO}/python/ACCEL_sift.py" | tee cands.txt >> /dev/null
echo "Done. Candidate info saved in cands.txt."
END="$(date -u +%s)"
echo "End time: $END"
echo "Time for ACCEL_sift: $(($END - $START)) seconds."
echo -e "************************************************************\n"

```

```
205
206 #fold best candidates: (referenced from Maura's siftandfold.bash)
207 echo "************************************************************"
208 echo "Folding candidates:"
209 START="$(date -u +%s)"
210 echo -e "Start time: $START\n"
211 #make list of candidate names
212 candlist=`grep "ACCEL" cands.txt | awk '{print $1}'`
213 #make into array
214 candarr=($candlist)
215
216 #make list of respective DMs of best candidates from cands.txt
217 dmslist=`grep "ACCEL" cands.txt | awk '{print $2}'`
218 #make into array
219 dmsarr=($dmslist)
220
221 #make list of respective periods of best candidates
222 periodslist=`grep "ACCEL" cands.txt | awk '{print $8/1000.}'`
223 #make into array
224 periodsarr=($periodslist)
225
226 #get total number of best candidates to loop through them
227 tot=`grep "ACCEL" cands.txt | wc | awk '{print $1}'`
228
229 nsub=`readfile $filename.sf | grep "samples per spectra" | awk '{print $5}'`
230
231 #loop prepfold through all viable candidates
232 for (( i=0; i<tot; i++ ))
233 do
234         echo "Running prepfold on candidate # $(( $i+1 )) out of $tot:"
235
236         #get filename of ACCEL_0 cands file and candnum
237         line=${candarr[$i]}
238         accelfilename="$(cut -d':' -f1 <<< $line)"
239         candnum="${line: -1}"
240
241         #get dat filename
242         length=$(( ${#accelfilename}-8 ))
243         datfilename="${accelfilename: 0:$length}"
244
245         #run prepfold command
246         echo -e "File: $datfilename; Candidate number: $candnum; DM: ${dmsarr[$i]};
```

```
247             nsub: $nsub \n"
248
249         #fold raw data
250         prepfold -mask ${filename}_rfifind.mask -dm ${dmsarr[$i]} $filename.sf
251             -accelfile $accelfilename.cand -accelcand $candnum -noxwin -nosearch
252             -o ${filename}_${dmsarr[$i]} >> /dev/null
253
254 done
255
256 echo -e "\nDone. Moving png files to folder:"
257 ls *.png
258 cp *.png ../results_png
259 END="$(date -u +%s)"
260 echo "End time: $END"
261 echo "Time for folding candidates: $(($END - $START)) seconds."
262 echo -e "**********************************************************\n"
263 #open all the plots
264 #eog *.png
265
266 ENDTIME="$(date -u +%s)"
267 echo -e "Total time elapsed for processing $filename: $(($ENDTIME - $STARTTIME))
268     seconds.\n"
269
270 exit 0
```

8.1.2 Appendix A2 - Data Processing Script for Arecibo Data

```
#!/bin/bash
#A script to process a single file, involving making a link to said file and running
#rfifind, DDplan.py, prepdata, realfft, accelsearch, accelsift, and prepfold on it.
#don't forget to do this on n04 -X! (BJ server)
#modified for use on the GZNU server.

#command line arguments are in the format: ddscript.bash directory filename
#boolean to check whether or not everything is in the right format and if the
#directory and filename are valid

STARTTIME="$(date -u +%s)"

echo "************************************************************"
echo "PRESTO data processing"
echo -e "by Shana Li\n"
echo "Arguments should be in the format: directory filename."
echo "Filename should be sans file extension (.fits)."
echo "************************************************************"

#set variables for directory and filename
directory=$1
filename=$2

echo -e "\n************************************************************"
echo "Directory: $directory"
echo "Filename: $filename"

#if directory is invalid, exit
if [ ! -d $directory ]; then
        echo "Invalid directory."
        echo -e "************************************************************\n"
        exit 0
fi

#if file doesn't exist, exit
if [ ! -e "$directory/$filename.fits" ]; then
        echo "Invalid filename."
        echo -e "************************************************************\n"
        exit 0
```

```
40  fi
41
42  #if folder for this particular file already exists, delete it
43  if [ -d $filename ]; then
44          echo -e "\nExisting files found, overwriting them now."
45          rm -r $filename
46          echo -e "Done.\n"
47  fi
48
49  #create new folder for data files
50  mkdir $filename
51  cd $filename
52
53  #create link to file in current directory in the folder
54  echo "Making link to file:"
55  ln -s $directory/$filename.fits ./ >> /dev/null
56  echo "Done."
57  echo -e "************************************************************\n"
58
59
60  echo "************************************************************"
61  #run rfifind
62  echo "Running rfifind:"
63  START="$(date -u +%s)"
64  echo -e "Start time: $START\n"
65  echo "Filename: $filename.fits"
66  echo -e "Time: 2 \n"
67
68  rfifind $filename.fits -time 2 -o $filename >> /dev/null
69
70  echo "Done."
71  END="$(date -u +%s)"
72  echo "End time: $END"
73  echo "Time for rfifind: $(($END - $START)) seconds."
74  echo -e "************************************************************\n"
75
76
77  echo "************************************************************"
78  echo "Running DDplan.py:"
79  START="$(date -u +%s)"
80  echo -e "Start time: $START\n"
81  #run DDplan.py:
```

```
#get variables needed:
#low dm
ldm=0
#high dm - changed from 4000 to 2000 for Arecibo data
hdm=2000
#time resolution
tres=0.5
#central frequency
cfreq=`readfile $filename.fits | grep "Central freq" | awk '{print $5}'`
#number of channels
numchan=`readfile $filename.fits | grep "Number of channels" | awk '{print $5}'`
#bandwidth
bandw=`readfile $filename.fits | grep "Total Bandwidth" | awk '{print $5}'`
#sample time
sampletime=`readfile $filename.fits | grep "Sample time" | awk '{print $5}'`
sampletime=0.000$sampletime
#spectra per file
numout=`readfile $filename.fits | grep "Spectra per file" | awk '{print $5}'`

#run DDplan.py
echo "Central Frequency: $cfreq"
echo "Number of Channels: $numchan"
echo "Total Bandwidth: $bandw"
echo -e "Sample Time: $sampletime \n"

DDplan.py -l $ldm -d $hdm -f $cfreq -b $bandw -n $numchan -t $sampletime -r $tres -o $filename >
    ${filename}_ddplaninfo.txt
echo "Done."
END="$(date -u +%s)"
echo "End time: $END"
echo "Time for DDplan: $(($END - $START)) seconds."
echo -e "**********************************************************\n"


echo "**********************************************************"
#subband de-dispersion: call prepsubband on each call in DDplan
#count the lines that contain the information for each call
echo "Running subband de-dispersion:"
START="$(date -u +%s)"
echo -e "Start time: $START\n"
lastline=14
line=`head -$lastline ${filename}_ddplaninfo.txt | tail -1`
```

```
124 while [ "$line" != "" ]; do
125         ((lastline++))
126         line=`head -$lastline ${filename}_ddplaninfo.txt | tail -1`
127 done
128 echo -e "Looping prepsubband for $(( $lastline-14 )) calls.\n"
129
130 #loop prepsubband for all calls in ddplaninfo
131 for (( i=14; i<lastline; i++ ))
132 do
133         #get the call from the current line in the file
134         call=`head -$i ${filename}_ddplaninfo.txt | tail -1`
135
136         #put call into an array
137         IFS=' ' read -a arr <<<"$call"
138
139         #get all the variables:
140         #nsub
141         nsub=`readfile $filename.fits | grep "samples per spectra" | awk '{print $5}'`
142
143         #low dm
144         ldm=${arr[0]}
145
146         #dm step
147         dms=${arr[2]}
148
149         #number of dms
150         ndm=${arr[4]}
151
152         #downsamp
153         ds=${arr[3]}
154
155         #numout (numout from DDplan / downsamp)
156         numo=$(( $numout / $ds ))
157
158         #run prepsubband command
159         echo "nsub: $nsub; Low DM: $ldm; DM step: $dms; Number of DMs: $ndm; Numout: $numo;
160             Downsample: $ds"
161         prepsubband -nsub $nsub -lodm $ldm -dmstep $dms -numdms $ndm -numout $numo -downsamp $ds
162             -mask ${filename}_rfifind.mask -o $filename $filename.fits >> /dev/null
163         echo -e "Done.\n"
164 done
165 END="$(date -u +%s)"
```

```
166 echo "End time: $END"
167 echo "Time for prepsubband: $(($END - $START)) seconds."
168 echo -e "************************************************************\n"
169
170
171 #run realfft:
172 echo "************************************************************"
173 echo "Running realfft:"
174 START="$(date -u +%s)"
175 echo -e "Start time: $START\n"
176 ls *.dat | xargs -n 1 --replace realfft {} >> /dev/null
177 echo "Done."
178 END="$(date -u +%s)"
179 echo "End time: $END"
180 echo "Time for realfft: $(($END - $START)) seconds."
181 echo -e "************************************************************\n"
182
183 #accelsearch for periodic signals
184 echo "Running accelsearch:"
185 START="$(date -u +%s)"
186 echo -e "Start time: $START\n"
187 ls *.fft | xargs -n 1 accelsearch -zmax 0 >> /dev/null
188 echo "Done."
189 END="$(date -u +%s)"
190 echo "End time: $END"
191 echo "Time for accelsearch: $(($END - $START)) seconds."
192 echo -e "************************************************************\n"
193
194 #sift through periodic candidates
195 echo "************************************************************"
196 echo -e "Running ACCEL_sift.py:"
197 START="$(date -u +%s)"
198 echo -e "Start time: $START\n"
199
200 #in GZNU server, re-direct X11 interface to admin1
201 tmp=`echo $DISPLAY`
202 export DISPLAY=10.10.10.24:38.0
203 python $PRESTO/python/ACCEL_sift.py > cands.txt
204 export DISPLAY=$tmp
205 #cd £tmp
206 echo "Done. Candidate info saved in cands.txt."
207 END="$(date -u +%s)"
```

```
208 echo "End time: $END"
209 echo "Time for ACCEL_sift: $(($END - $START)) seconds."
210 echo -e "************************************************************\n"
211
212
213 #fold best candidates: (referenced from Maura's siftandfold.bash)
214 echo "************************************************************"
215 echo "Folding candidates:"
216 START="$(date -u +%s)"
217 echo -e "Start time: $START\n"
218 #make list of candidate names
219 candlist=`grep "ACCEL" cands.txt | awk '{print $1}'`
220 #make into array
221 candarr=($candlist)
222
223 #make list of respective DMs of best candidates from cands.txt
224 dmslist=`grep "ACCEL" cands.txt | awk '{print $2}'`
225 #make into array
226 dmsarr=($dmslist)
227
228 #make list of respective periods of best candidates
229 periodslist=`grep "ACCEL" cands.txt | awk '{print $8/1000.}'`
230 #make into array
231 periodsarr=($periodslist)
232
233 #get total number of best candidates to loop through them
234 tot=`grep "ACCEL" cands.txt | wc | awk '{print $1}'`
235
236 nsub=`readfile $filename.fits | grep "samples per spectra" | awk '{print $5}'`
237
238 #loop prepfold through all viable candidates
239 for (( i=0; i<tot; i++ ))
240 do
241         echo "Running prepfold on candidate # $(( $i+1 )) out of $tot:"
242
243         #get filename of ACCEL_0 cands file and candnum
244         line=${candarr[$i]}
245         accelfilename="$(cut -d':' -f1 <<< $line)"
246         candnum="${line: -1}"
247
248         #get dat filename
249         length=$(( ${#accelfilename}-8 ))
```

```
250         datfilename="${accelfilename: 0:$length}"
251
252         #run prepfold command
253         echo -e "File: $datfilename; Candidate number: $candnum; DM: ${dmsarr[$i]}; nsub: $nsub \n"
254
255         #fold raw data
256         prepfold -mask ${filename}_rfifind.mask -dm ${dmsarr[$i]} $filename.fits -accelfile
257             $accelfilename.cand -accelcand $candnum -noxwin -nosearch -o ${filename}_${dmsarr[$i]}
258             >> /dev/null
259
260 done
261
262 echo -e "\nDone. Moving png files to folder:"
263 ls *.png
264 cp *.png ../results_png
265 END="$(date -u +%s)"
266 echo "End time: $END"
267 echo "Time for folding candidates: $(($END - $START)) seconds."
268 echo -e "************************************************************\n"
269 #open all the plots
270 #eog *.png
271
272 ENDTIME="$(date -u +%s)"
273 echo -e "Total time elapsed for processing $filename: $(($ENDTIME - $STARTTIME))
274     seconds.\n"
275
276 exit 0
```

8.1.3 Appendix A3 - Script for Cutting and Combining Arecibo Data

```
#!/bin/bash
#A script to run pzc's cutting and combining scripts on Arecibo data.

#The file prefix for the files
filepre="p1693.20100502.Strip46.b0s1g0.00"

#The starting file number
startfnum=500

#The ending file number (doesn't change)
endfnum=527

#Length between starting subint and ending subint (length of cut -1)
length=71

#Starting subint for first file
startn=0

#Overlap increment
overlap=24

#Strip number
stripnum=0



while [ $startfnum -le $endfnum ]; do

        #increment strip number for naming
        stripnum=$(($stripnum + 1))

        if (( $startn == 128 )); then
                startn=0
        fi

        #check if the starting subint number is greater than 129; if so, get the
        #starting n of the next file
        if (( $startn > 128 )); then
                startn=$(($startn % 128 - 1))
```

```
40                 startfnum=$(($startfnum + 1))
41         fi
42
43         #The starting filename
44         startfile="${filepre}${startfnum}.fits"
45
46         #Ending subint
47         endn=$(($startn + $length))
48
49         #check if the ending subint number is greater than 129; if so, get the
50         #ending n of the next file and combine; else, just cut
51         if (( $endn > 128 )); then
52                 #for combining:
53
54                 endn=$(($endn % 128 - 1))
55
56                 #The next file number
57                 nextfnum=$(($startfnum + 1))
58
59                 #The ending filename; the one immediately after the starting filename
60                 nextfile="${filepre}${nextfnum}.fits"
61
62                 #combine
63                 echo "Combine: $startfile: from $startn to 128; $nextfile: from 0
64                     to $endn."
65                 python /home/pzc/pulsar_search/AO_201707/combine_Arecibopsrfits_freq_-
66                     time_splitpol.py 0 1023 $startn $endn $startfile $nextfile
67                     >> /dev/null
68
69         else
70                 #for cutting:
71
72                 #cut
73                 echo "Cutting: $startfile: from $startn to $endn."
74                 python /home/pzc/pulsar_search/AO_201707/cut_Arecibopsrfits_freq_-
75                     time_splitpol.py $startfile 0 1023 $startn $endn >> /dev/null
76
77         fi
78
79         #set next startn
80         startn=$(($startn + $overlap))
81
```

```
82          #rename file
83          mv Arecibo_Out.fits Arecibo_Out_$stripnum.fits
84
85          echo "Saved as Arecibo_Out_$stripnum.fits."
86
87  done
```

8.1.4 Appendix A4 - Script to Parallelize Processing for Arecibo Strips

```
bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_1 &
bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_2 &
bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_3 &
bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_4 &
bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_5 &
bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_6 &
bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_7 &
bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_8 &
wait

bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_9 &
bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_10 &
bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_11 &
bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_12 &
bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_13 &
bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_14 &
bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_15 &
wait

bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_16 &
bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_17 &
bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_18 &
bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_19 &
bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_20 &
bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_21 &
bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_22 &
wait

bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_23 &
bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_24 &
bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_25 &
bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_26 &
bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_27 &
bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_28 &
bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_29 &
wait

bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_30 &
bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_31 &
```

```
bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_32 &
bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_33 &
bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_34 &
bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_35 &
bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_36 &
wait

bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_37 &
bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_38 &
bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_39 &
bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_40 &
bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_41 &
bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_42 &
bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_43 &
wait

bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_44 &
bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_45 &
bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_46 &
bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_47 &
bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_48 &
bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_49 &
bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_50 &
wait

bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_51 &
bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_52 &
bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_53 &
bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_54 &
bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_55 &
bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_56 &
bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_57 &
wait

bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_58 &
bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_59 &
bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_60 &
bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_61 &
bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_62 &
bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_63 &
bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_64 &
wait
```

```
82
83  bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_65 &
84  bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_66 &
85  bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_67 &
86  bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_68 &
87  bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_69 &
88  bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_70 &
89  bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_71 &
90  wait
91
92  bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_72 &
93  bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_73 &
94  bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_74 &
95  bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_75 &
96  bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_76 &
97  bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_77 &
98  bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_78 &
99  wait
100
101 bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_79 &
102 bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_80 &
103 bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_81 &
104 bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_82 &
105 bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_83 &
106 bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_84 &
107 bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_85 &
108 wait
109
110 bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_86 &
111 bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_87 &
112 bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_88 &
113 bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_89 &
114 bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_90 &
115 bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_91 &
116 bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_92 &
117 wait
118
119 bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_93 &
120 bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_94 &
121 bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_95 &
122 bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_96 &
123 bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_97 &
```

```
bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_98 &
bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_99 &
wait

bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_100 &
bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_101 &
bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_102 &
bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_103 &
bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_104 &
bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_105 &
bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_106 &
wait

bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_107 &
bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_108 &
bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_109 &
bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_110 &
bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_111 &
bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_112 &
bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_113 &
wait

bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_114 &
bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_115 &
bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_116 &
bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_117 &
bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_118 &
bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_119 &
bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_120 &
wait

bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_121 &
bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_122 &
bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_123 &
bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_124 &
bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_125 &
bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_126 &
bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_127 &
wait

bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_128 &
bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_129 &
```

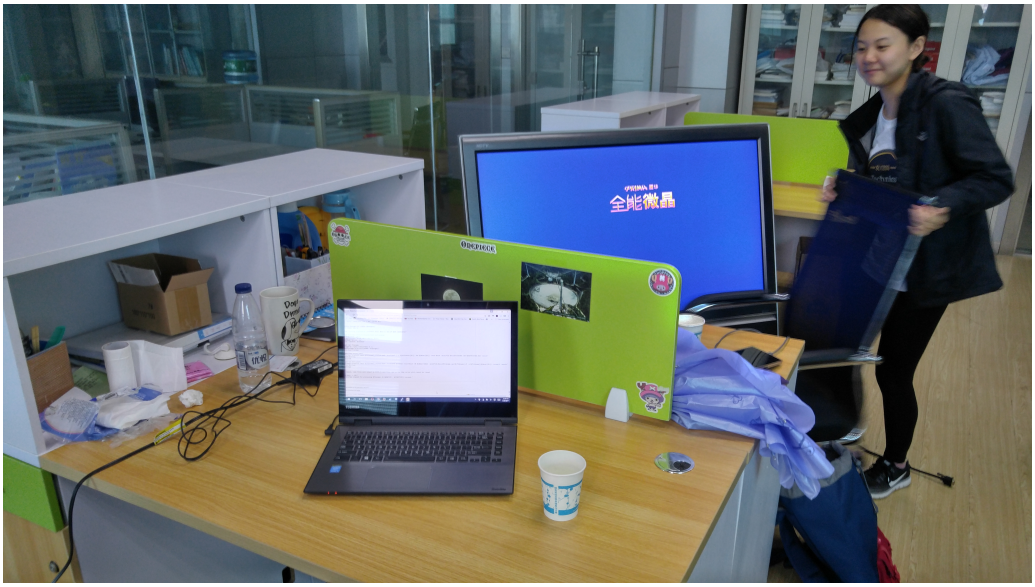
```
bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_130 &
bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_131 &
bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_132 &
bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_133 &
bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_134 &
wait

bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_135 &
bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_136 &
bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_137 &
bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_138 &
bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_139 &
bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_140 &
bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_141 &
wait

bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_142 &
bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_143 &
bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_144 &
bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_145 &
bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_146 &
bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_147 &
bash ~/ddscript_arecibo.bash ~/arecibo_cut Arecibo_Out_148 &
wait
```

8.2 Appendix B - Photos

Appendix B1 - My workspace setup in the NAOC office in Beijing.



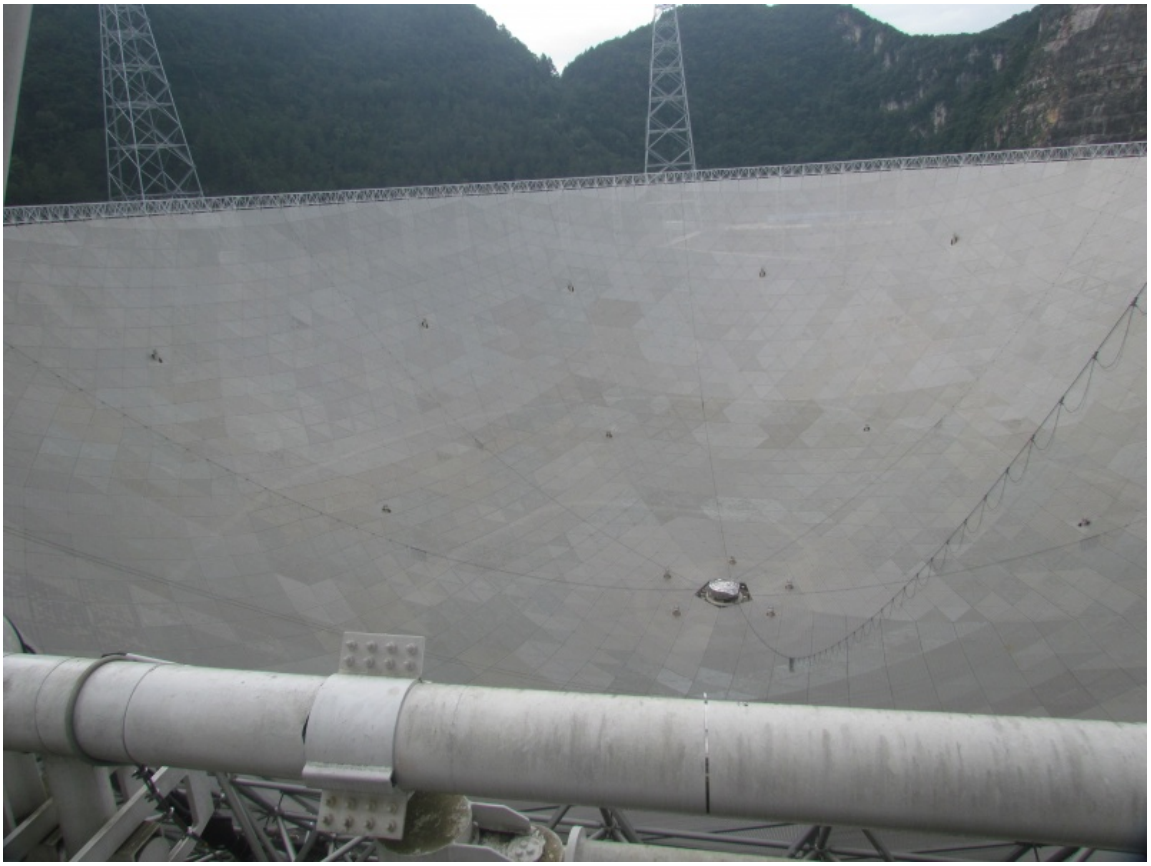
Appendix B2 - My workspace setup in the GZNU office in Guiyang.

Appendix B3 - The GZNU server for all FAST processes.

Appendix B4 - The entrance to the FAST site in Guizhou.

Appendix B5 - Zach and I in front of the FAST site entrance.

Appendix B6 - A panorama shot that captures the entire diameter of the FAST telescope dish.



Appendix B7 - A closeup shot of the FAST dish.