Astropulse: A Search for Microsecond Transient Radio Signals Using Distributed Computing

By

Joshua Solomon Von Korff

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Physics

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Dan Werthimer, Co-chair
Professor Steven E. Boggs, Co-chair
Eric J. Korpela
Professor Geoffrey C. Bower
Professor William L. Holzapfel

Spring 2010

Astropulse: A Search for Microsecond Transient Radio Signals Using Distributed Computing

by Joshua Solomon Von Korff

Abstract

Astropulse: A Search for Microsecond Transient Radio Signals Using Distributed Computing

by

Joshua Solomon Von Korff

Doctor of Philosophy in Physics

University of California, Berkeley

Dan Werthimer, Co-Chair

Steven E. Boggs, Co-Chair

I performed a transient, microsecond timescale radio sky survey, called "Astropulse," using the Arecibo telescope in Puerto Rico. Astropulse searches for brief (0.4 μs to 204.8 μs), wideband (relative to its 2.5 MHz bandwidth) radio pulses centered at 1,420 MHz, a range that includes the hyperfine hydrogen line.

Astropulse is a commensal survey, obtaining its data by sharing telescope time with other surveys, such as PALFA. I scanned the sky visible to Arecibo, between declinations of -1.33 and 38.03 degrees, with varying dwell times depending on the requirements of our partner surveys. I analyzed 1,540 hours of data in each of 7 beams of the ALFA receiver, with 2 polarizations per beam, for a total of 21,600 hours of data. The data were 1-bit complex sampled at the Nyquist limit of 0.4 μs per sample.

Examination of timescales less than 12.8 μs would have been impossible if not for my use of coherent dedispersion, a technique that has frequently been used for targeted observations, but has never before been associated with a radio sky survey. I performed nonlinear coherent dedispersion, reversing the broadening effects on signals caused by their passage through the interstellar medium (ISM). Coherent dedispersion requires intensive computations, and needs far more processing power than the more usual incoherent dedispersion. This processing power was provided by BOINC, the Berkeley Open Infrastructure for Network Computing. BOINC is a distributed computing system, which allowed me to utilize hundreds of thousands of volunteers' computers to perform the necessary calculations for coherent dedispersion. Each volunteer's computer requires about a week to process a single 8 MB "workunit," corresponding to 13 s of data from a single beam and polarization. In all, Astropulse analyzed over 48 TB of data.

I did not aim to detect any particular astrophysical source, intending rather to perform a survey of the transient radio sky. Astrophysical events that might produce brief radio pulses include giant pulses from pulsars, RRATs, or exploding primordial black holes. In discussing

the results of the Astropulse project, I have taken our sensitivity to primordial black holes with a certain size and spatial distribution to indicate our overall sensitivity relative to other surveys.

Radio frequency interference (RFI) and noise contaminated the data; these were mitigated by a number of techniques including multi-polarization correlation, DM repetition detection, and frequency profiling. I also made use of a number of programs that specifically blank RFI from the FAA and aerostat radars near Arecibo.

Ultimately, Astropulse's sensitivity turned out to be similar to that of other very recent surveys, demonstrating that with enough computing power, a radio sky survey can make use of coherent dedispersion. We were unable to prove decisively that any of the signals came from astrophysical sources, but we did notice a surplus of pulses coming from inside the Galactic disk, as opposed to the halo.

In addition to Astropulse, I programmed a "distributed thinking" project called Stardust@home. The two projects are not related by their science content, but they are closely connected by their use of distributed processing methods. The Stardust spacecraft returned pristine interstellar dust samples, and Stardust@home recruited volunteers to locate these dust particles in microscopic-scale images of aerogel. "Distributed thinking" means that volunteers examine our data with their own eyes, judging whether they see the dust particles. In contrast, Astropulse volunteers utilize their computers' processing power. Both methods create opportunities for public outreach, encouraging non-scientists to participate in scientific research. By signing up for Astropulse or Stardust@home, anyone can learn about astronomy and make a contribution to the field.

Dedication

I would like to dedicate my dissertation to my family, and especially my wife, Lucy. My constant conversations with Lucy were the highlight of every day during my thesis work. Although we were separated by thousands of miles for most of that time, I felt her presence always. My parents, Jerry and Connie Von Korff, supported me through adversity, entertained me with their stories, and sometimes even came to visit me in Berkeley. And my brothers, Michael and Benjamin, were always there for me to talk to, understanding me in the way that brothers can.

Acknowledgements

A distributed computing project can be a large undertaking, and my work on Astropulse was by no means a solitary venture. Many people helped me to complete this work, including the programmers and researchers who set up the BOINC infrastructure and the preliminary foundations for Astropulse, as well as those who helped me during the coding of a working Astropulse client and the analysis of our database. In many ways, Astropulse is a branch of the SETI@home program, which preceded Astropulse by several years and utilizes the same data set.

I would like to thank my advisor, Dan Werthimer, who always saw the big picture and encouraged me to keep my expectations high. Dan's many years of experience with science, SETI, and astronomy were an invaluable resource. In addition, Astropulse relies heavily on the electronics created by his CASPER group (the Center for Astronomy Signal Processing and Electronics Research). Eric Korpela's incredibly extensive knowledge of the underlying sofware for Astropulse (and SETI@home) was indispensible. I could not have done without Eric's patience, wisdom, and aptitude for all topics relevant to Astropulse - including astronomy, programming in many languages, statistics, mathematics, computer hardware, and databases. Eric was always willing to share his insights. The programmers in our group include Matt Lebofsky, Jeff Cobb, and Bob Bankay; without their assistance, the servers, databases, and other infrastructure that supported Astropulse would have been impossible to maintain. All of them frequently helped my code to interface with existing SETI software. Dave Anderson, a programmer and co-director of SETI@home, was responsible for the BOINC software that manages the distributed computing aspect of Astropulse. Dave also helped to mentor my work on Stardust@home, my first introduction to distributed processing. The newest member of our group, Andrew Siemion, proofread this dissertation, giving many insightful comments. And some of his work on RFI mitigation in other projects helped to inspire Astropulse's methodology.

I would also like to thank the temporary members of our research group, and those who left before I finished: Courtleigh Cannick, Kevin Douglas, and undergraduates Adam Fries, Arielle Little, and Luke Kelley. Researchers who left our group even before I arrived, but whose work was instrumental in the creation of Astropulse, include Paul Demorest and Eric Heien as well as Nopparat Pantsaena, Ryohi Takahashi, Christopher Day, and Karl Chen.

As programmer for Stardust@home, I worked for Andrew Westphal, who introduced me to public participation science. His boundless enthusiasm was frequently contagious. My co-workers on the Stardust@home team included Anna Butterworth, Matt Paul, Robert Lettieri, Anna Zhang, and Bryan Mendez.

In the UC Berkeley physics department, I am grateful to Professor Steve Boggs, who has served as my departmental advisor, my dissertation committee chair, and a member of my qualifying exam committee. My dissertation committee also includes Professor William Holzapfel and Professor Geoff Bower. Both of these were on my quals committee, as was Professor Jonathan Arons. Anne Takizawa, a staff member in student services in the physics department, helped me with the logistics of scheduling my qualifying exams, organizing my thesis committee, and other important tasks. Without her support, the system would have been nearly impossible to navigate. Donna Sakima's help was crucial as well.

I performed other research in graduate school at UC Berkeley before I came to Astropulse, and I would like to thank the people I worked with. Professor Birgitta Whaley, Julia Kempe, Jiri Vala, Sabrina Leslie, Travis Beals, Neil Shenvi; Professor Robert Littlejohn, Matthew Cargo, and Prashant Subbarao. These experiences shaped my scientific understanding a great deal. Over the years, I have also looked for the good advice and good example of friends in the physics department, including Rob Lillis, Martin Lueker, Emily Riddle, and others.

I would like to thank friends and loved ones who provided moral support and encouragement. Most of all, my wife Lucy, who was genuinely interested in listening to my science stories.

And I am grateful to the National Science Foundation, NASA, and the Friends of SETI@home for funding Astropulse and the SETI@home servers. The Arecibo Observatory is the principal facility of the National Astronomy and Ionosphere Center, which is operated by the Cornell University under a cooperative agreement with the National Science Foundation.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Scientific motivation

This is an exciting time in the field of transient astronomy, both in the radio and in other parts of the spectrum. Improving technology allows astronomers to perform fast followups of transient events, store extensive digital records of observations, and run processor-intensive algorithms on data in real time. These advances make possible instruments that examine optical afterglows of gamma-ray bursts (RAPTOR, Vestrand et al. (2005)) or neutrino sources (Kowalski & Mohr, 2007). High resolution digital images can be recorded and stored quickly using current (Pan-STARRS, Kaiser (2004)) and planned technology (LSST, Ivezic et al. (2008)). In the radio, astronomers search for transients such as orphan GRB afterglows (FIRST-NVSS, Levinson et al. (2002)) or radio bursts of unknown origin (STARE, Katz et al. (2003)).

My thesis research project, called "Astropulse," searches for brief, wideband radio pulses on timescales of microseconds to milliseconds, and surveys the entire sky visible from Arecibo Observatory. The idea of a short-timescale radio observation is not new. Other experiments are well-suited for detecting radio pulses on a microsecond timescale, or even much shorter scales. However, these observations are directed; they examine known phenomena. For instance, such an experiment might record the nanosecond structure of the signals from the Crab pulsar. And of course the idea of a radio survey is not new. Other experiments perform surveys for radio pulses over large regions of the sky. However, these observations examine 50 μs timescales or longer. Astropulse is the first radio survey for transient phenomena with microsecond resolution.

This project is made possible by Astropulse's access to unprecedented processing power, using the distributed computing technique. We send our data to volunteers, who perform coherent dedispersion using their own computers. Then they send the results of this computation back to us, informing us whether they detected a signal, and reporting that signal's dispersion measure, power, and other parameters. Astropulse is processor intensive because we must perform coherent dedispersion, whereas other surveys perform incoherent dedispersion. Coherent dedispersion is necessary to resolve structures below 50 μs or so, depending on the dispersion measure.

We are not committed to detecting any particular astrophysical source; rather, we are motivated by our ability to examine an unexplored region of parameter space. However, we consider that we might detect evaporating primordial black holes, millisecond (or faster) pulsars, or RRATs. I will consider each of these possibilities in turn. We could also detect communications from extraterrestrial civilizations, though we will not discuss this possibility in detail.

1.2 Black holes

1.2.1 What is a black hole?

Astropulse searches for short radio pulses from many possible sources, including evaporating primordial black holes. A black hole is a singular solution to the equations of general relativity. In particular, it's an aggregation of matter concentrated at a single central point. The relativistic description is given by a metric, which specifies the proper time or proper distance between any two points with an infinitesimal separation from each other. For an uncharged, nonrotating black hole, the metric is (Frolov & Novikov, 1998):

$$ds^2 = -(1 - \frac{2GM}{c^2 r})c^2 dt^2 + (1 - \frac{2GM}{c^2 r})^{-1} dr^2 + r^2(d\theta^2 + \sin^2\theta d\phi^2). \quad (1)$$

The surface given by $r = \frac{2GM}{c^2}$ is evidently special, because the radial spatial component of the metric blows up to infinity at that point. This is not a true singularity, as can be shown by a change of coordinates – the true singularity is at the center, $r = 0$. One possible set of non-singular coordinates are the Kruskal-Szekeres coordinates (u, v) satisfying:

$$u = |\frac{c^2 r}{2GM} - 1|^{1/2} e^{c^2 r/4GM} \cosh(\frac{c^3 t}{4GM}) \quad (2)$$

$$v = |\frac{c^2 r}{2GM} - 1|^{1/2} e^{c^2 r/4GM} \sinh(\frac{c^3 t}{4GM}) \quad (3)$$

for $r > 2GM/c^2$, and similar equations for $r < 2GM/c^2$ with sinh and cosh switched (Shapiro & Teukolsky, 1983).

But the surface $r = \frac{2GM}{c^2}$ is often taken to be the "radius" of the black hole, although in truth there is no mass at any point except $r = 0$. The radius is called the Schwarzschild radius, or the event horizon.

According to general relativity, it is not possible for matter to escape from inside the event horizon of the black hole. This is because time and space have taken one anothers' functions – the coefficient of dt^2 is positive, and the coefficient of dr^2 is negative. That is, any trajectory directed out of the black hole would be analogous to a trajectory (in flat Minkowski space) that is either spacelike, or is timelike but directed backward in time.

1.2.2 Hawking radiation

However, it was proposed by Hawking (1974) that a black hole of mass M emits radiation like a black body whose temperature is given by the following relation:

$$T_{BH} = \frac{\hbar c^3}{8\pi k G M} = 10^{-6} \left(\frac{M_\odot}{M} \right) \text{ K} . \tag{4}$$

This is the same temperature studied in the theory of black hole thermodynamics, which also attributes entropy to a black hole proportional to the black hole's surface area (Raine & Thomas, 2005).

The radiation occurs at the event horizon of the hole, which has a radius $r = 2GM/c^2$, and an area $A = 4\pi r^2 = (16\pi G^2/c^4)M^2$. (It turns out that the Euclidean formula for the area of a sphere still holds in this case.) This gives the black hole an intrinsic luminosity of $L = \sigma A T^4 \propto M^{-2}$.

The radiant energy comes directly from the black hole's mass, and as a result, it is losing mass at a rate $\dot{M} \propto -M^{-2}$. Because the black hole radiates more power as it shrinks, we expect a burst of energy in the last moments of the black hole's life. Astropulse hopes to detect this burst of energy, so we would like to know about its duration. One can make different assumptions about the energy distribution of the radiation from a black hole evaporation (Carter et al., 1976). For a "hard" equation of state, with an adiabatic index $\Gamma > \frac{6}{5}$, the radiation does not reach thermal equlibrium. The standard model falls into this category, and would assume that the radiation behaves as a relativistic ideal gas, $\Gamma = \frac{4}{3}$. In this case, the final explosion of the black hole lasts on the order of seconds. However, for a "soft" equation of state, as proposed by Hagedorn (1965), Γ could be much smaller. In this case, the explosion might happen in 10^{-7} seconds or less. Astropulse is ideally suited for detecting such fast explosions.

We can integrate the radiant energy to find the total lifetime of the hole:

$$\tau_{BH} = 10^{10} \text{ year} \left(\frac{M}{10^{12} \text{ kg}} \right)^3 .$$

Clearly, if τ_{BH} is greater than the age of the universe, the black hole cannot be exploding now, no matter when it was created. Therefore, the black hole must have been created at a mass of 10^{12} kg or less. So we should inquire about known black holes, their masses, and their origins.

1.2.3 Classes of black holes

Known (and speculated) types of black holes can be divided into solar mass, supermassive, intermediate mass, and primordial black holes.

Solar mass black holes form when a supernova forces a star's core to implode, resulting in a neutron star or (if the progenitor star is sufficiently massive) a black hole. The Chandrasekhar limit says that a white dwarf cannot be more massive than about 1.4 solar

masses, and similarly a neutron star cannot be more than $2-3$ solar masses depending on the assumed equation of state. The existence of neutron stars was confirmed in 1967 with the discovery of pulsars, but black holes proved harder to pin down. One method (Fre et al., 1999) for detecting a black hole is to search for an x-ray binary, where a star and a black hole orbit one another. The existence of the (invisible) black hole can be established by the redshift and blueshift of its partner, and it's possible to detect x-ray emission as material spirals into the black hole and is absorbed.

Supermassive black holes are much larger, containing millions or billions of solar masses. The first confirmed supermassive black hole is the one at the center of M87 (Macchetto, 1999). Later, such a black hole was discovered in our own Galaxy, detected via the motions of nearby stars (Raine & Thomas, 2005). These stars are moving in extremely fast, tight orbits, implying a huge, dense mass at the center. Even if we assume that the source of the gravity is itself a cluster of neutron stars, it would follow that the cluster should quickly collapse into a black hole. Supermassive black holes also exist in other galaxies, and are crucial in creating the energy output of quasars and similar objects. It is not fully understood how supermassive black holes form, but they probably come from mergers of smaller black holes.

Intermediate mass black holes, for instance between $100-1000M_{\odot}$, are even less well understood. They may be responsible for certain ultraluminous x-ray sources (Raine & Thomas, 2005), or may be located at the centers of certain globular clusters. These black holes must also be formed from mergers of stellar mass black holes.

But from the above calculations, we know that black holes formed from stellar collapse ($M \sim M_{\odot} \sim 10^{30}$ kg) will take about 10^{34} years to evaporate - ridiculously long compared to the age of the universe. And in fact the black hole will grow faster than that just by absorbing the CMB and interstellar medium. At its present size, the black hole would have a temperature of about 10^{-7} K , so it would be completely undetectable. Intermediate mass and supermassive black holes are even less detectable, as the luminosity decreases with mass. Thus there is little hope of detecting Hawking radiation from these "conventional" black holes. However, only one mechanism is known for producing black holes of less than solar mass. Namely, they would have to be created in the big bang (Hawking, 1971).

1.2.4 Primordial black holes

A black hole with an initial mass of 10^{12} kg would be nearing the end of its life now, and may emit a detectable pulse. According to the process outlined above, this black hole would have a temperature of 10^{12} K , mainly visible in gamma rays. Such a small black hole could not have been created via core collapse of a star, nor by mergers of larger black holes. It would be far more dense than a stellar mass black hole, and would compress its Schwarzschild radius into a region the size of a nucleon! Thus, the mini black hole would have to form from "density perturbations in the early universe" (MacGibbon et al., 1990).

Some groups have searched for these primordial black holes (PBHs) in the radio, but many researchers have looked for gamma-ray emission instead (Ukwatta et al., 2010). Radio and gamma-ray surveys make very different assumptions about the PBHs' evaporation time,

so that their results are difficult or impossible to compare in a meaningful way. Ukwatta talks about PBH explosions having timescales of seconds or minutes, whereas Astropulse is looking for microsecond pulses. This means that Ukwatta's black hole explosions are undetectable by Astropulse, since their flux density is comparatively very low. On the other hand, Ukwatta's rate limit (the minimum detectable number of events per volume per time) is 10^{-1} pc^{-3}yr^{-1} , which is many orders of magnitude worse than Astropulse's rate limit of $1.6 \cdot 10^{-12}$ pc^{-3}yr^{-1} .

1.2.5 Electromagnetic pulses

The total amount of energy released in the last second of the black hole's life is about 10^{23} J . Most previous studies have attempted to detect this energy in the cosmic gamma ray background (Raine & Thomas, 2005). But Rees (1977) suggested that some of this energy could be converted into a radio pulse. The idea is that as the black hole shrinks, and becomes hotter, it starts radiating not just photons, but massive particles such as electrons and positrons (due to pair production at the event horizon), and later, heavier particles as well. This forms a plasma fireball expanding around the hole. As this conducting shell expands into the ambient magnetic field, it pushes the field out of the way, creating an electromagnetic pulse. The energy imparted to the field goes like $\gamma^2 \times$ (initial field energy), where γ is the Lorentz factor of the shell. If the fireball appears when the black hole has mass M_{crit}, then we can derive $T \propto (M_{\text{crit}})^{-1}$ (Equation 4) and $kT \sim m_e\gamma$. Then the duration of the pulse is the time between the passage of the initial radiation through the maximum radius of expansion ($r_{\max}/c$) and the passage of the conducting shell through that radius ($r_{\max}/\beta c$, where the shell's velocity is βc). This difference goes like $r_{\max}/c\gamma^2$, so we can take the peak wavelength as $\lambda \sim r_{\max}/\gamma^2$.

We can derive $r_{\max}$ if we know the ambient magnetic field strength B, by assuming that all of shell's kinetic energy goes into the field, so that the field energy, which is initially $\propto B^2V$ for volume V, becomes $B^2V\gamma^2 = M_{\text{crit}}$. This fixes V, hence $r_{\max}$. It turns out that for a magnetic field B around 5×10^{-6} Gauss and a critical mass of $\sim 2 \times 10^{11}$ g, we get $\lambda \sim 10$ cm. So a radio pulse detectable in the 21 cm band is plausible.

An observation of these pulses would be a very significant confirmation of both Hawking radiation and the existence of primordial black holes. At the very least, we can put a limit on the possible maximum density of evaporating black holes in the universe, if we make some assumptions about their distribution, and contingent on the assumption that they produce radio pulses. This information would be relevant to cosmological models describing the big bang.

1.3 Pulsars

1.3.1 Origin

Another radio source we might detect with Astropulse is a pulsar. A pulsar is a rotating neutron star, whose magnetic dipole axis is misaligned with its axis of rotation. When the magnetic axis is directed toward the Earth, we can detect a radio pulse. Since the star rotates with a regular period, the pulses have a regular period as well – although some temporary timing irregularities (glitches) are possible, especially in young pulsars (Lorimer & Kramer, 2005), and the period slowly increases over time.

Neutron stars are created when a massive star runs out of nuclear fuel, so that the fusion process cannot prevent gravitational collapse. However, if the core is not too massive, neutron pressure can halt this collapse. The stellar matter bounces off the core, resulting in a supernova, and the core becomes a neutron star. If the progenitor star has some angular momentum, the neutron star may be spinning, resulting in a pulsar. Furthermore, neutron stars in binaries may accrete matter from their partners, resulting in an increased angular momentum and a "resurrection" of the pulsar.

1.3.2 Radiation mechanism

As a pulsar rotates, its magnetic dipole field rotates with it, carrying the surrounding plasma. Since the plasma cannot move faster than the speed of light, there is a critical surface (called the "light cylinder") at a distance $\frac{Pc}{2\pi}$ from the pulsar's axis of rotation, where P is the period. Some of the magnetic field lines form closed loops from the north to the south pole of the pulsar. But if a magnetic field line does not close inside the light cylinder, it will not close at all. In that case, it's called an open field line.

A simplistic model for pulsar emission is that plasma moves along open field lines, radiating energy in the direction of motion due to the curvature of its trajectory. (Think of synchrotron radiation.) But no known model explains the data very well. Some theories include (Lorimer & Kramer, 2005):

1. Antenna mechanisms: suppose the charged particles move in groups. If N particles with charge q are each moving, the resulting power radiation goes like $(qN)^2$. However, no mechanism is known to make the particles move in groups.
2. Relativistic plasma emission: energy comes from plasma turbulence. But this must be converted into another type of wave so that the energy can escape the pulsar's magnetosphere.
3. Maser mechanisms.

1.3.3 Fastest possible pulsars

We want to know whether pulsars could produce pulses with a width on the order of microseconds. Astropulse is optimized for pulses of 200 μs or less, but its sensitivity relative to other surveys is best at short timescales, around 0.4 to 1.6 μs.

So we should first ask about the minimum period of a pulsar. Most pulsars have periods on the order of 0.2 s to 2 s, but a few have millisecond periods – the period distribution is bimodal. The first millisecond pulsar to be discovered (Backer et al., 1982) was PSR 1937+214, with $P = 1.558$ ms. More recent discoveries include (Hessels, 2006) PSR 1748-2446, with $P = 1.397$ ms, and (Kaaret et al., 2006) XTE J1739-285, which may have evidence of a pulsar with $P = 0.89$ ms.

We could attempt to deduce a minimum period with a classical (Newtonian) calculation. Say the pulsar has radius R, and is spinning such that its centripetal acceleration at the surface is equal to the acceleration of gravity, $R\omega^2 = \frac{(2\pi)^2 R}{P^2} = \frac{GM}{R^2}$. That is, its surface is in a Keplerian orbit. This results in

$$P^2 = \frac{(2\pi)^2 R^3}{GM}, \tag{5}$$

$$\omega = 1.15 \cdot 10^4 (\frac{M}{M_\odot})^{1/2} (\frac{R}{10 \text{ km}})^{-3/2} s^{-1}. \tag{6}$$

So we can scale P smaller by shrinking R and/or increasing M. We could ask about the minimum possible value of R, but let's first consider typical values $M = 1.4 M_\odot$, $R = 10$ km. Then $P = 461$ μs, which is not too much smaller than known pulsar periods. Smaller pulsar radii might lead to much smaller periods due to the 3/2 power of R. Yet the consensus among theorists is that it's very difficult to devise a model that allows a period substantially below 1 ms. Why is this?

First, equation (6) is a classical equation. The Roche model is a simple model that applies general relativity, but assumes that the star's mass is extremely centrally condensed, and distributed as if it were not rotating. This changes the factor in front of equation (6) to

$$\omega = 6.3 \cdot 10^3 (\frac{M}{M_\odot})^{1/2} (\frac{R}{10 \text{ km}})^{-3/2} s^{-1}. \tag{7}$$

If instead we calculate the star's actual mass distribution according to GR (which requires knowing or guessing the equation of state), we find that the maximum mass $M_{\max}$ of the nonrotating configuration differs from (and is $10\% - 20\%$ smaller than) the mass M of the maximally rotating configuration. In this case, it turns out to be simplest to calculate ω in terms of $M_{\max}$ and $R_{\max}$, where $R_{\max}$ is the radius of the nonrotating configuration with the maximum mass $M_{\max}$:

$$\omega = 7.7 \cdot 10^3 (\frac{M_{\max}}{M_\odot})^{1/2} (\frac{R_{\max}}{10 \text{ km}})^{-3/2} s^{-1}. \quad (8)$$

To derive this equation rigorously, one would have to obtain it independently for every plausible equation of state; no general proof is known. But the agreement with equation (8) is better than 4% for all equations of state considered in Lattimer et al. (1990).

We would like a pulsar with small R and large M. In order to prevent the star from collapsing into a black hole, we require that the equation of state be very stiff at high densities, ensuring that the dense center of the star is capable of supporting itself against gravity. On the other hand, the EOS must be soft at lower (nuclear) densities, so that the radius can be compressed to the smallest possible value. Essentially, we want a massive, self-supporting star with a relatively thin outer region of low density.

Several constraints are active. For instance, the speed of sound cannot be larger than the speed of light (causality), which prevents the EOS from becoming too stiff. And the neutron-proton ratio must be in beta equilibrium, in that the ratio with the minimum energy is realized. Mechanisms for softening the EOS at nuclear densities include pion condensation, kaon condensation, and quark stars (including strange stars).

Lattimer et al. (1990) proposes many possible EOS's, but none of them results in ω larger than $1.37 \cdot 10^4 \ s^{-1}$, which yields a period of $P = 459 \ \mu s$; and most models predict significantly larger periods. So in order to imagine a rotating neutron star with a significantly faster period, we would have to invent an as yet unimagined equation of state that allows the maximum mass neutron star to be even more massive and/or smaller.

1.3.4 Shortest possible duty cycles

The duty cycle of a pulsar is the fraction of time in which the pulse is visible. That is, it is determined by the opening angle of the beam, with a correction for the possibly nonzero angle between the beam and the observer's line of sight. According to Lorimer & Kramer (2005), the opening angle scales as $\ell = 2\rho \sim P^{-0.5}$:

$$\rho \approx (3/2)\theta_{\text{em}} \approx \sqrt{\frac{9\pi r_{\text{em}}}{2cP}} \text{ radians} = 1.24^\circ (\frac{r_{\text{em}}}{10\text{km}})^{1/2} (\frac{P}{1s})^{-1/2}, \quad (9)$$

where:

$\rho = \frac{1}{2}\ell$, if $\ell =$ the opening angle of the beam. That is, ρ is the angle between the magnetic axis and a line tangent to the last open field line at the point of emission. This tangent line intersects the magnetic axis at a point with distance $r_{\text{intersect}} > 0$ from the center of the pulsar.

$P =$ the pulsar's period.

θ_{em} = the angle between the magnetic axis, and a line passing through the point of emission and the center of the pulsar. This is smaller than ρ.

r_{em} = the radius of the emission point, i.e. its distance from the center of the pulsar.

Since emission heights don't vary too much, this can be taken as a simple $P^{-0.5}$ law. Unfortunately, this law says that smaller periods have larger duty cycles, which is counter-productive for us. (We want both small periods and short duty cycles.) However, observation shows that some millisecond pulsars have much smaller duty cycles than expected, with a beam opening angle half-width as low as 7° for a 3 ms pulsar, seeming to imply an emission height interior to the neutron star. This means that the beam is visible for just 120 μs. And Kramer et al. (1998) suggest that if sub-millisecond pulsars exist, their emission properties would differ substantially from millisecond pulsars. So there seems to be a great deal of uncertainty about the expected pulse widths of fast pulsars; and the lower bound to the width, if any, is not known. Therefore, if we hope to see shorter pulses from pulsars, the most likely source would be a millisecond or marginally sub-millisecond pulsar with a very narrow pulse width.

1.4 Giant pulses

In addition to regular, periodic pulses, some pulsars (such as the Crab), produce less periodic giant pulses. These pulses can have thousands of times the flux of a normal pulse (Popov & Stappers, 2007) or more. There is no consensus on the origin of giant pulses, though they may come from plasma turbulence.

However, it is clear that giant pulses can have very short timescales suitable for detection by Astropulse. The Crab's giant pulses range from a few nanoseconds at 2 Jy μs (Hankins et al., 2003) to 64 μs or more at 1,000 to 10,000 Jy μs, with a typical duration of a few microseconds. (See Section 3.1 for a discussion of the Jy μs unit.) Popov & Stappers (2007) have measured the energies of giant pulses from the Crab, and found the proportions of pulses with durations as low as 4.1 μs. (In most pulsars, the fraction of giant pulses is too low to measure statistics accurately.)

1.5 RRATs

McLaughlin et al. (2006) describe a new type of pulsed radio source, believed to be a type of rotating neutron star. (Hence the name "RRATs", for "Rotating RAdio Transients.") The RRATs differ from previously known rotating neutron stars (i.e. pulsars) in that:

1. Their periods are very long – 5 out of 10 of them have periods longer than 4 seconds, whereas pulsars almost never have periods that long. (The 11th RRAT's period could not be measured.)

2. The pulses appear sporadically, averaging 4 min to 3 hours between bursts. Fourier analysis and fast folding were unable to detect a period; rather, the authors had to find the greatest common factor of the intervals between pulse arrival times. For most sources, only 3 pulses were detected.

The bursts' durations range from 2 to 30 ms, so Astropulse is not particularly well suited for detecting them. (Astropulse is most sensitive to bursts of duration 200 μs or less.) However, RRATs are a new phenomenon, and few have been discovered. So it's quite possible that some RRATs have much shorter burst durations.

2 Telescope and instrumentation

2.1 Sky coverage

Arecibo Observatory (Figure 1) scans approximately $\frac{1}{3}$ of the sky, between declinations of -1.33 and 38.03 degrees (Figure 2). Because of this, Astropulse cannot see the galactic center (around $-29°$ dec) but can see 452 out of the 1826 pulsars in the ATNF pulsar database [1], including the Crab. Astropulse is a commensal survey; this means that other surveys control the telescope pointing, but allow Astropulse to collect data at all times. Our partner surveys include GALFA (Galactic ALFA) and PALFA (Pulsar ALFA.) Our group also operates SETI@home, another commensal radio survey, and the two projects use the same data: a 1-bit complex sampled 2.5 MHz bandwidth centered at 1420 GHz. To date, we have observed for 1,540 hours with each of the 7 beams (and 2 linear polarizations per beam), for a total of 21,600 hours of observation time. We have been taking data using the ALFA receiver from September 2006 until the present (May 2010), for a total of 3.7 years. This implies that we have had 1/21 of all possible observation time during those years. Since a good deal of Arecibo's time is dedicated to non-astronomical purposes, such as ionospheric science, our fraction of astronomy time is significantly larger than 1/21.

2.2 ALFA receiver

The ALFA receiver (Figure 3) has 7 dual-polarization beams on the sky arranged in a hexagonal pattern, each with a $3.5'$ beamwidth. The central beam has a gain of 11 K / Jy, and the other beams have 8.5 K / Jy [2]. The system temperature is 30 K. The 6 peripheral beam pointings differ from the central beam by a maximum of 384 arcseconds. At present, our database is not set up to differentiate between the pointings of different beams; therefore all pointings have an error of $384'' + 1.75' = 8.15'$ In a future work, I will fix this problem, reducing the error to $1.75'$.

2.3 Downconverter

Multiple experiments use the signal from the ALFA receiver, so we split the signal using an IF splitter. The splitter outputs are labelled 0A, 1A ... 6A, 6B (Figure 4). The letter corresponds to polarization, and the number corresponds to one of the seven ALFA feeds (or beams.) These 14 signals are attenuated by 6 to 13 decibels for purposes of level-matching, and then enter our multibeam quadrature baseband downconverter. Our downconverter has 16 inputs, so at this point two dead (grounded) signals are introduced.

The downconverter translates the signal down by some frequency ν_0, so that our 2.5 MHz bandwidth will be centered at 0 MHz. Naively, we might want a mathematical operation that

[1] `http://www.atnf.csiro.au/research/pulsar/psrcat/`, as of 6/29/2009

[2] `http://www.naic.edu/alfa/gen_info/info_obs.shtml`

Figure 1: Arecibo Observatory's Gregorian dome, from below

sends $\cos \nu t$ to $\cos{(\nu - \nu_0)t}$; but there is no such operation that works for all values of ν and gives exactly the answer we want. To see this, consider that if we translate $f(t) = \cos 2\nu_0 t$ "down" by ν_0, we should get $\cos \nu_0 t$. If we translate $g(t) = \cos -2\nu_0 t$, then according to our rule, we are supposed to get a different result, $\cos -3\nu_0 t$. But the cosine is an even function, so $f(t) = g(t)$, and the two results should have been the same. Note that physical processes which shift frequencies of real functions, such as Doppler shifts, do not modify frequencies by adding or subtracting a constant amount, though they might multiply by a constant.

Instead, we consider our signal to be complex, and multiply by $e^{-\imath\nu_0 t}$. This means that we make a copy of the signal, multiply the first copy by $\cos \nu_0 t$ and call it the "real part", and multiply the second copy by $-\sin \nu_0 t$ and call it the "imaginary part". The multiplication is performed by an NEC UPC2766GR chip. The effect of the downconversion on $\cos \nu t$ is:

$$(\cos \nu t) \cdot e^{-\imath\nu_0 t} \tag{10}$$

$$= \frac{1}{2}(e^{\imath\nu t} + e^{-\imath\nu t}) \cdot e^{-\imath\nu_0 t} \tag{11}$$

$$= \frac{1}{2}(e^{\imath(\nu-\nu_0)t} + e^{-\imath(\nu+\nu_0)t}) \tag{12}$$

$$\tag{13}$$

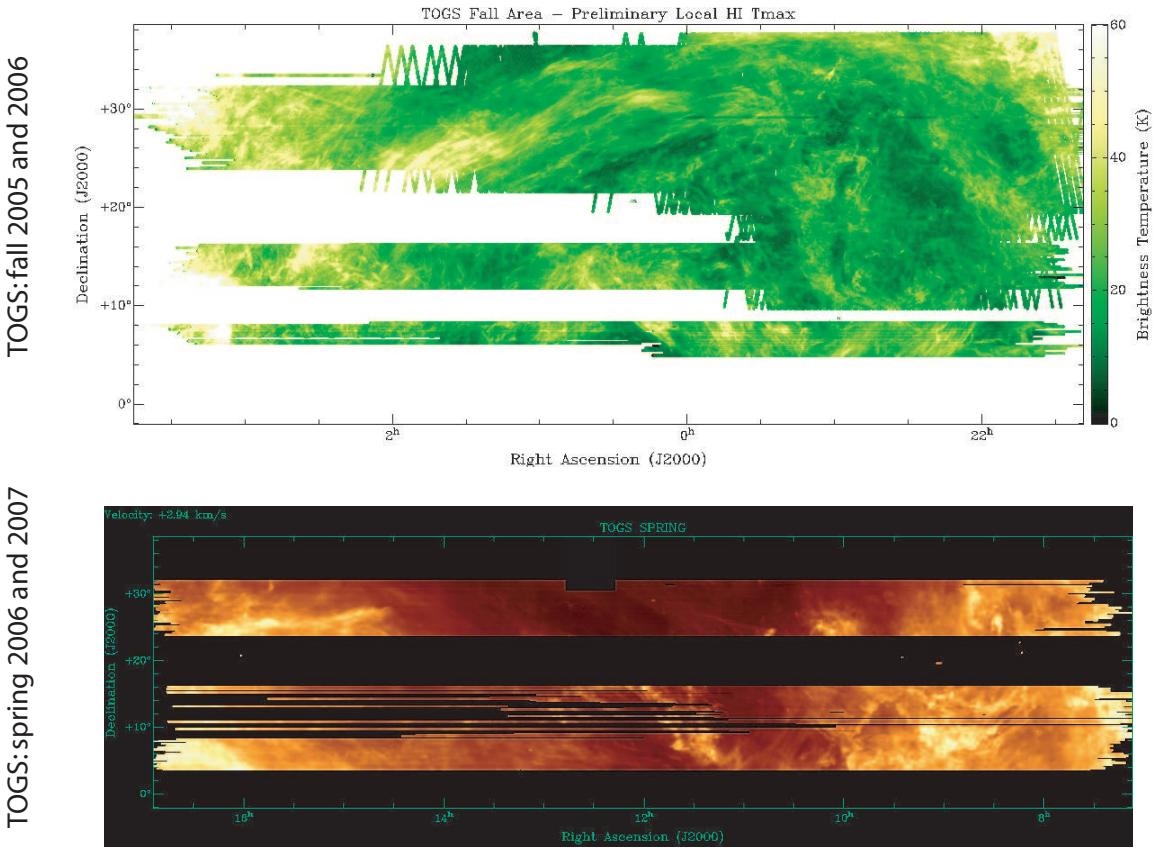


Figure 2: Sky map showing the area covered by the TOGS survey 2005-2007, and concurrently by Astropulse and SETI@home

So, assuming $\nu \approx \nu_0$, the second term can be removed by a low pass filter (linear technology LT C1560-1), and we're left with a single complex Fourier component at the desired frequency. At this point, we have 32 real signals (or 16 complex signals), of which 4 (or 2) correspond to dead RF inputs. The original real-valued signal had two components (sine and cosine) at each frequency from $\nu_0 - B/2$ to $\nu_0 + B/2$, where B is the bandwidth. Our complex signals also have two components (real and imaginary) at each frequency from $-B/2$ to $B/2$.

2.4 DDA cards

These 32 real channels are digitized with 1 bit precision using comparators, and the resulting digital signals (the signs of the 32 voltages) are directed through ribbon cables to Digital Data Acquisition (DDA) cards on a PC. RF inputs 1 through 8 go to DDA card number 1, and RF inputs 9 through 16 go to DDA card number 0. Each DDA card reads 16 bits of data at a time; one real and one imaginary bit for each of 8 complex channels.

Figure 3: The ALFA receiver

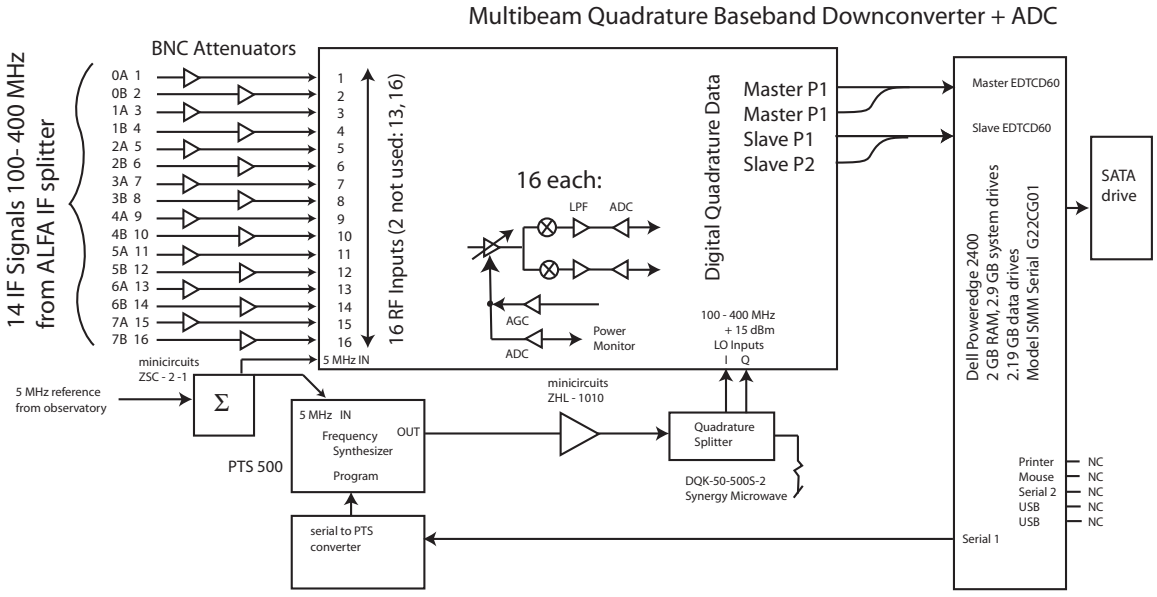


Figure 4: Astropulse / SETI@home backend, including attenuators, downconverter, and Dell PC. The DDA cards are labelled "EDTCD60."

2.5 Data recorder

The 32 channels are directed to the PC, which is running our datarecorder2 program. In addition to reading this data, we read telescope coordinates from the Arecibo telescope's data broadcast network, SCRAMnet. The coordinates consist of the Right Ascension (RA) and Declination (Dec) to which the telescope is currently pointing, as well as the time for which that RA and Dec are valid. As output, the datarecorder2 program creates both quicklook files and ordinary data files. A quicklook file contains only one DDA card, whereas an ordinary data file may interleave data blocks from both DDA cards.

An ordinary data file is composed of "blocks" written to disk, each of which has a 4096-byte header followed by a 2^{20}-byte data segment. The blocks are in groups of 128 blocks, each group coming from a single DDA card. Within each data block, each word of data corresponds to one time sample. A word is composed of the 16 DDA bits, numbered 0-15, for the appropriate DDA card.

The data files are stored on a hot swappable SATA drive, which fills up in 14 to 20 hours of observation time. Since we are taking data 1/21 of the time, we must swap out the SATA drive about once per two weeks.

2.6 Arrival at Berkeley

The staff at the Arecibo Observatory swap the drives out when they are full. When enough drives have been collected, they ship the drives to us at Space Science Lab, UC Berkeley. We copy the files to static hard drives located at Berkeley, wipe the SATA drives, and send them back to Arecibo. We use 20 SATA drives in all, each of which holds 500 or 715 GB. We also send a copy of each file to NERSC, the Nation Energy Research Scientific Computing Center. This ensures that we can retrieve the data at any time.

In all, we have taken over 48 TB of data from ALFA multibeam. We need a large (6 TB) disk array at Berkeley to buffer the data before it is sent to volunteers. The volunteers' PCs then process the data and send the results back to Berkeley. For a discussion of data processing after this point, see Section 4 on BOINC, and Section 3.3 on the algorithm for the volunteers' client program.

3 Pulse detection : thresholds and dedispersion

3.1 Overview

The primary function of the Astropulse program is to dedisperse potential pulsed signals, then determine whether the dedispersed pulse surpasses an appropriate threshold. I will discuss the theory behind dedispersion, the logic of the algorithm, and then methods we use to select the thresholds. Then I will calculate the sensitivity of Astropulse in Jy μs, a unit of "pulse area."

The Jy μs unit refers to the pulse's flux density (in Jy) integrated over its duration (in μs.) It is called an "area" because it is calculated using this integral, which is the area under a curve. Although the unit of flux density (Jy) is a more conventional measure of sensitivity, the Jy μs is more meaningful in our case, because we would like to detect unresolved pulses. For example, consider two pulses; one is 500 Jy and lasts 0.2 μs, and the other is 1,000 Jy and lasts 0.1 μs. When these pulses are dispersed, they will be similar in appearance; Astropulse cannot distinguish between them because their dedispersed durations are shorter than Astropulse's time resolution. But Astropulse can determine that both pulses are 100 Jy μs.

Note that when I describe a measured pulse's apparent area in Jy μs, the actual pulse area may be different depending on any contributions to the system temperature. I assume a particular minimal system temperature (10 K) for the ALFA receiver, whereas we might have a different effective system temperature when looking at the Crab nebula.

3.2 Dedispersion

Between a radio pulse's source (i.e. black hole, pulsar, or ET) and our detector, the pulse must travel through the Interstellar Medium (ISM). The ISM is composed of neutral hydrogen and helium atoms which have negligible effect on the wave's propagation, as well as a plasma component consisting of protons, other positively charged ions, and free electrons. As the wave travels through the plasma, it's affected by dispersion, in a manner analogous to the dispersion of light in a prism. In this case, the high frequency component moves slightly faster through the ISM than the low frequency component. The time delay between any two frequencies is given by Wilson et al. (2009):

$$\Delta\tau = \frac{e^2}{2\pi c m_e}\left(\frac{1}{\nu_1^2} - \frac{1}{\nu_2^2}\right)\int N(\ell)d\ell, \tag{14}$$

Where $N(\ell)$ is the electron number density at position ℓ along the pulse's path. Then $\int N(\ell)d\ell$ is the dispersion measure, or DM, and depends only on the distribution of plasma between the source and detector, not on the frequency of the radiation. If $N(\ell)$ is measured in electrons cm^{-3}, and ℓ is measured in parsecs, we say that the dispersion measure DM is

measured in pc cm^{-3}. A useful estimate for the dispersion measure weighted mean electron density in our Galaxy is $N = 0.03$ cm^{-3} (Guélin, 1973).

Consider, for instance, the Crab pulsar, at DM = 56.8. Astropulse sees a bandwidth of 2.5 MHz centered at 1420 GHz. We can deduce an approximate $\Delta\tau$ by saying that $\frac{1}{\nu_1^2} - \frac{1}{\nu_2^2} \approx \frac{2}{\nu^3}\Delta\nu$, setting $\Delta\nu = 2.5$ MHz and $\nu = 1420$ GHz. The resulting $\Delta\tau = 0.00041145s$, which differs from the exact result by 0.000155%. Astropulse has 0.4 μs samples, so this is 1028.6 samples, for a ratio of samples / DM = 18.11. Alternatively,

$$\Delta\tau = (8.3\ \mu s)\frac{\Delta\nu(\text{MHz})}{\nu^3(\text{GHz})}\text{DM}(\text{pc cm}^{-3}). \tag{15}$$

So a hypothetical Crab pulse that initially would have been concentrated in one time sample of our measured time series will be dispersed to $N = 1029$ time samples. This means it will be submerged in N times as much noise (on average), if the noise is measured in Jy μs. (The noise in Jy μs increases when we integrate the noise over a longer duration. If instead we measured the noise in Jy, there would be no time integral.) The probability density function (pdf) of the noise (measured in Jy μs) is an incomplete gamma function, as discussed below, at Equation 57. As the number of samples N increases, this pdf approaches a normal distribution, with standard deviation proportional to $\sqrt{N}$. The signal is not detectable unless it is substantially stronger than this $\sqrt{N}$, because otherwise it would look like a fluctuation in the noise. For a larger dispersion measure, the problem would be even worse.

To increase the SNR, we have to reconstruct the original pulse as well as possible, bringing together the component frequencies and reassembling them so that the signal takes up a single time sample again. This will prevent it from being buried in noise, allowing us to improve our sensitivity by lowering our detection threshold.

3.2.1 Time vs. frequency plots

We can depict the dedispersion process in a time vs. frequency plot, such as Figure 5. Note that there is no strictly correct mathematical way of depicting a function's time vs. frequency, due to the uncertainty principle. That is, we cannot write an arbitrary amplitude $A(t)$ in the form $\nu(t)$. For instance, consider a signal whose time vs. amplitude graph looks like a delta function. It is supposed to contain all frequencies ... so what frequency does it have at time 0? At times other than 0? A time vs. frequency plot of this sort is meaningful only to the extent that the signal looks locally like a monochromatic (sine) wave.

Instead, we can perform Fourier transforms on groups of consecutive samples. Say we FFT 64 samples. They are located at some time t, although the resolution is now 64 sample widths. The FFT gives us an amplitude (hence a power) for each of 64 frequencies, so we now have a plot with z = power, y = frequency, and x = time. Such a plot is shown in Figure 5, in the extreme case that a single frequency is present at each time for the dispersed

signal (i.e. there is no noise.) In this plot, the power is represented by the darkness of each pixel.

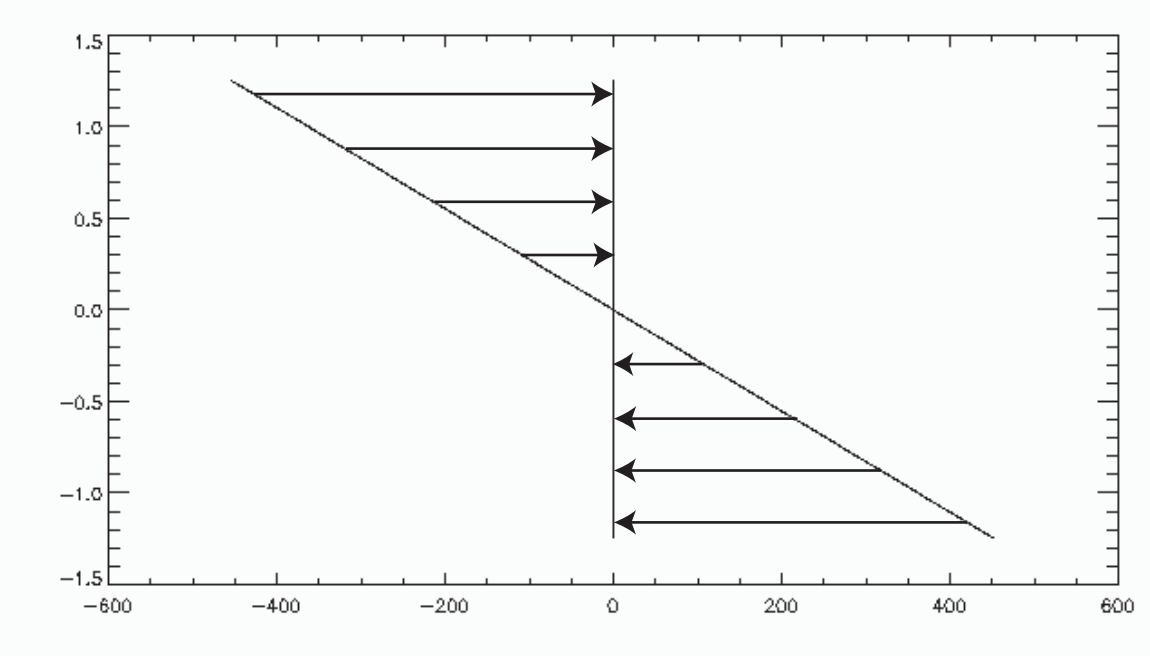


Figure 5: The diagonal line represents the initial, dispersed signal. The vertical line is the dedispersed signal. The x axis is time in Astropulse samples, and the y axis is frequency in MHz. The pulse has a dispersion measure (DM) of +50 pc cm^{-3}. This figure depicts linear dedispersion, as opposed to the nonlinear dedispersion dicussed in Section 3.2.5

3.2.2 Incoherent dedispersion and its limitations

We have two choices for our methodology: coherent dedispersion and incoherent dedispersion. Astropulse uses coherent dedispersion, whereas other radio surveys use incoherent dedispersion. Incoherent dedispersion is much more computationally efficient, and for longer timescales it's almost as good as coherent dedispersion. However, as we will see, Astropulse would be unable to examine the 0.4 μs timescale without coherent dedispersion.

Incoherent dedispersion means that the signal's power spectrum is calculated, and the power vs. time of each sub-band is analyzed. The method is called "incoherent" for this reason – the phase information about individual frequencies is lost; only the total power of each subband is retained. Next, the sub-bands are realigned at all possible dispersion measures, in an effort to find one DM at which the components align to produce a large power in a short period of time. Suppose we use $\Delta\tau$ to denote the difference between the time delay of the highest and lowest frequencies in our bandwidth, as above. If our incoherent dedispersion algorithm makes a linear approximation, then the sub-band with frequency $\nu_0 + \nu$ (where ν_0 is the center frequency) is shifted by a time $\frac{\nu}{\Delta\nu} \cdot \Delta\tau$, where $\Delta\nu$ is our bandwidth.

However, incoherent dedispersion is limited in two ways. First, the goal of recording

power vs. time makes sense only on a timescale greater than $dt_1 = \frac{1}{d\nu}$, where $d\nu$ is the width of each sub-band. This is because of time-frequency uncertainty. Second, in each sub-band the pulse is dispersed by $dt_2 = \Delta\tau \cdot \frac{d\nu}{\Delta\nu}$. So the method cannot localize the pulse better than this. Setting $dt = dt_1 = dt_2$, we find that the minimum timescale for incoherent dedispersion, dt, happens when:

$$\begin{aligned}
\Delta\tau \cdot \frac{d\nu}{\Delta\nu} &= \frac{1}{d\nu} && (16)\\
d\nu^2 &= \frac{\Delta\nu}{\Delta\tau} && (17)\\
d\nu &= \sqrt{\frac{\Delta\nu}{\Delta\tau}} && (18)\\
dt &= \sqrt{\frac{\Delta\tau}{\Delta\nu}} && (19)\\
&= \sqrt{411 \ \mu\text{s} \ \frac{DM}{56.8} \cdot (\frac{1.42 \text{ GHz}}{\nu_0})^3 (\frac{\Delta\nu}{2.5 \text{ MHz}}) \cdot (\Delta\nu)^{-1}} && (20)\\
&= 12.8 \ \mu\text{s} \ (\frac{DM}{56.8})^{0.5} (\frac{\nu_0}{1.42 \text{ GHz}})^{-1.5}. && (21)
\end{aligned}$$

For the Crab pulsar, this is a limit of 12.8 μs, or 32 samples. For a more distant source, the limit might be as much as 50 μs, or 124 samples. (Astropulse considers sources with a DM as high as 830 pc cm^{-3}.)

3.2.3 Coherent dedispersion as deconvolution

Coherent dedispersion is an alternative technique that allows better time resolution, by performing the mathematical inverse of the ISM's dispersion operation. Coherent dedispersion deals with amplitude rather than power, preserving phase information; in the absence of noise or scattering, it would reconstruct the original pulse exactly. We need to analyze the mathematical operation corresponding to dispersion, in order to find its inverse.

If $F(n)$ is the original pulse as a function of sample number n, then suppose $D[F]$ is the dispersed pulse. We can show that D is just a convolution. In particular, $D[F] = F * D[\delta]$, where $*$ is convolution and δ is the discrete δ function.

We know that D is time translation-invariant and linear, because Maxwell's equations in a plasma have these properties. That is, if we combine Maxwell's equations with:

$$\begin{aligned}
m_e \dot{\vec{v}} &= -e\vec{E} && (22)\\
\vec{J} &= -en_e\vec{v} && (23)
\end{aligned}$$

for the velocity of the electrons (assuming the protons do not accelerate substantially), we have a linear set of equations in $\rho, \vec{J}, \vec{E}, \vec{B}$, and $\vec{v}$. (These represent the charge density, current density, electric field, magnetic field, and electron velocity, respectively.) For example, Figure 6 depicts a dispersed (chirped) delta function.

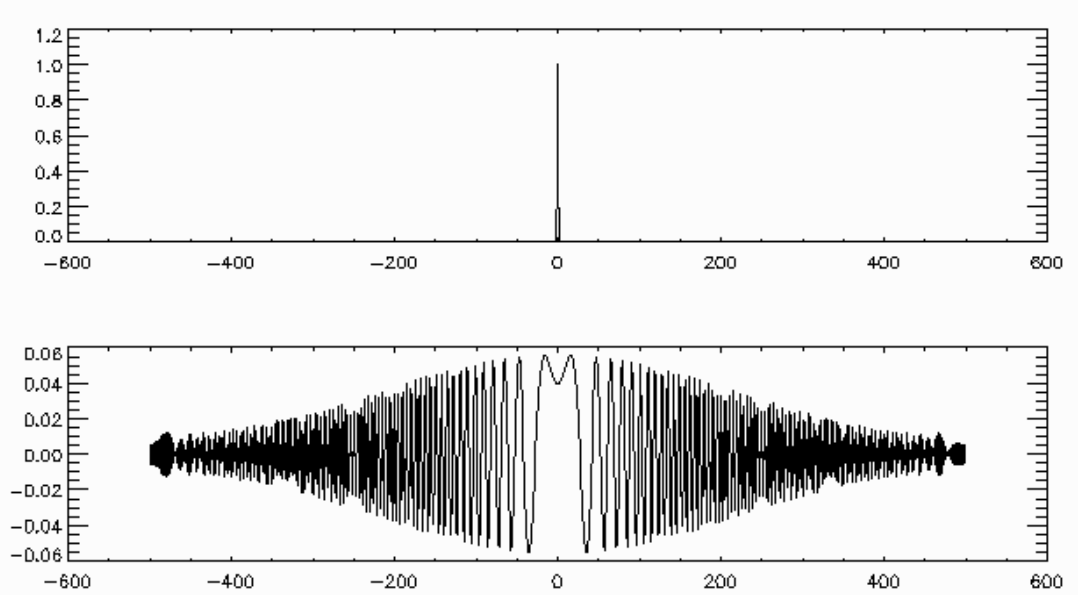


Figure 6: Delta function and chirped delta function. The x axis is time, and the y axis is amplitude. (Units are arbitrary for both axes.) Note that the chirped function tapers off toward 0 amplitude at high and low times. This is because the initial function is not a true delta function; it has nonzero width. For an infinitely narrow delta function, the chirped function's envelope would not taper off at all. Note also that our dedispersion operation assumes that the band's center frequency has a time delay of 0, with the highest frequencies arriving earlier than an undispersed pulse would. Of course this is impossible; and our reconstruction of the pulse at this particular time is purely conventional. Thus, our reconstructed pulse times are correct relative to other pulses of the same DM, but not in an absolute sense.

When the delta function is time translated, so is the dispersed version (Figure 7.) And when two delta functions are summed, so are the dispersed versions (Figure 8.) From figures 7 and 8, we conclude that dispersion is linear and time translation invariant.

Then:

$$F(n) = \sum_{n'} F(n')\delta(n-n') \equiv (F * \delta)(n), \tag{24}$$

$$D[F](n) = \sum_{n'} F(n') \cdot D[\delta](n-n') \equiv (F * D[\delta])(n), \tag{25}$$

where the last step is by linearity and time translation invariance of D.

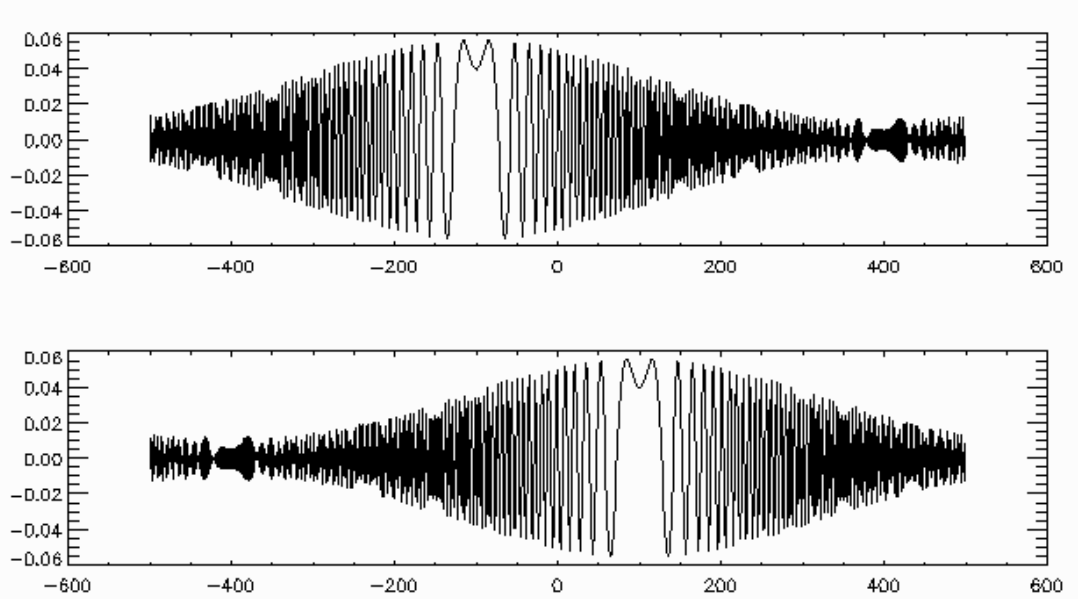


Figure 7: Time translated chirped delta functions. The x axis is time, and the y axis is amplitude. (Units are arbitrary for both axes.)

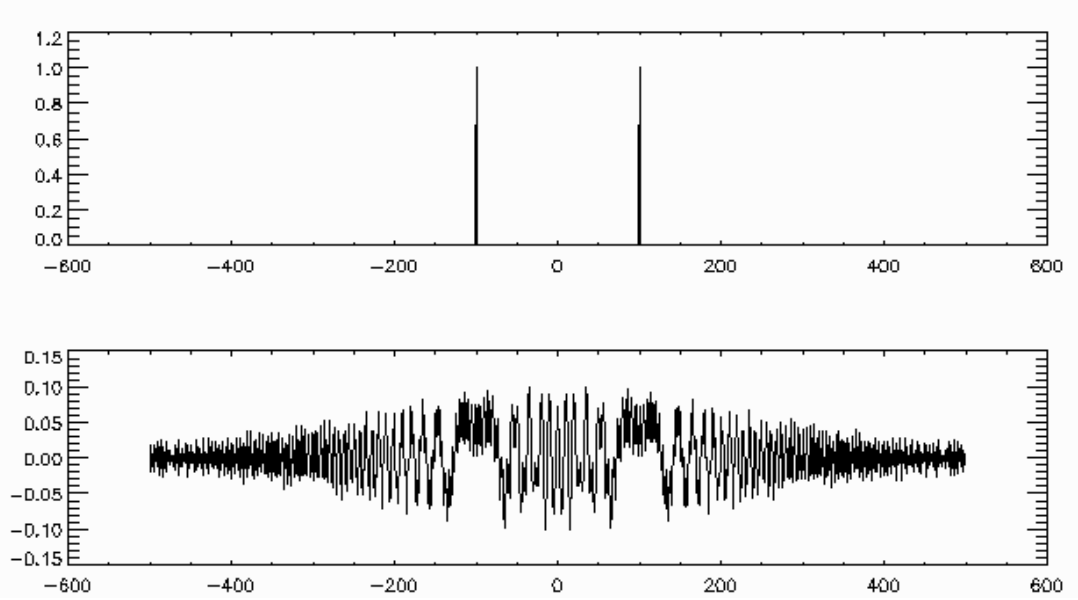


Figure 8: Two chirped delta functions. The x axis is time, and the y axis is amplitude. (Units are arbitrary for both axes.)

3.2.4 Coherent dedispersion using FFTs

Now the pulse that arrives at our detector is the output pulse $F * D[\delta]$, and we want to determine F. To do this, we use the fact that the convolution operator is related to the

Discrete Fourier Transform (DFT). We have $\mathrm{DFT}(f * g) = \mathrm{DFT}(f) \cdot \mathrm{DFT}(g)$

Therefore

$$\begin{aligned} \mathrm{DFT}(D[F]) &= \mathrm{DFT}(F * D[\delta]), & (26) \\ &= \mathrm{DFT}(F) \cdot \mathrm{DFT}(D[\delta]). & (27) \end{aligned}$$

We divide both sides of Equation 27 by $\mathrm{DFT}(D[\delta])$, and apply the inverse DFT, obtaining:

$$F = \mathrm{DFT}^{-1}(\frac{\mathrm{DFT}(D[F])}{\mathrm{DFT}(D[\delta])}). \quad (28)$$

Convolution "the slow way" takes time $O(N^2)$ – that is, convolution by multiplying every value of one function by every value of another function, then adding up the results. But the FFT is faster, taking time $O(N \log N)$, where N is the length of the FFT. ("DFT" refers to a mathematical function, and "FFT" refers to a specific type of algorithm that computes that function quickly.) For every N samples, coherent dedispersion takes time $O(MN \log N)$, where M is the number of dispersion measures to test. If L is the number of samples in the data stream, coherent dedispersion takes time $O(ML \log N)$.

On the other hand, incoherent dedispersion does not test as many dispersion measures as coherent dedispersion does, because its time resolution is imperfect and it cannot always distinguish between different DMs. To distinguish between two dispersion measures, the algorithm would have to notice that the pulses' respective slopes differ in a time vs. frequency plot. But if the pulses' slopes differ by a very small amount, time-frequency uncertainty may prevent us from detecting the difference. (And recall that time-frequency uncertainty is a more significant problem with incoherent dedispersion, because the sub-bands are narrower than the full bandwidth.) Another consideration that limits time resolution, thereby degrading the precision of our DM detection, is dispersion within sub-bands.

Together, these two effects result in some practical time resolution of n samples, allowing us to test not M, but M/n dispersion measures. The value of n can be calculated from Equation 21. For example, with a resolution of 20 μs and a sample duration of 0.4 μs, $n = 50$. For each dispersion measure, we must process L samples, so we require time $O(ML/n)$. (Some additional time is required to FFT the data into a power spectrum, but this process is not dominant.) So the time ratio between coherent and incoherent dedispersion is $O(n \log N)$. In our case, $\log N = 15$, so this goes like $50 \cdot 15 = 750$, a large ratio. Of course we can't calculate the ratio exactly, without knowing the respective constant factors. But this helps explain why coherent dedispersion is not generally used for surveys.

3.2.5 Computing the nonlinear chirp function

To find the chirp function DFT($D[\delta]$), we must first find $D[\delta]$. This is just the chirped delta function. Although it is common to make a linear approximation to the chirp function, we will make no such approximation, resulting in a nonlinear chirp function. We will find $t(\nu)$, the arrival time of frequency ν, and $\nu(t)$, the frequency arriving at time t. Then we will use these functions to compute the frequency domain amplitude of the chirp function in two ways, a less rigorous way and a more rigorous way.

Now, the arrival time of frequency ν is given by Equation 14 as:

$$t(\nu) = \frac{A}{\nu^2} + t_0. \tag{29}$$

Thus, for interstellar dispersion, we expect higher frequencies to arrive first, which agrees with Equation 29 for $A > 0$. We want $\nu = 1420$ MHz to arrive at time 0, which requires that $t_0 < 0$.

Let's determine $\nu(t)$, the frequency arriving at time t. This has:

$$\begin{aligned} t &= \frac{A}{\nu^2} + t_0, & (30)\\ \nu(t) &= A^{1/2}(t - t_0)^{-1/2}. & (31) \end{aligned}$$

Then

$$f(t) = \exp(2\pi i \int_0^t \nu(t')dt') \tag{32}$$

is the pulse amplitude $D[\delta]$ as a function of time, if ∞ frequency arrives at time t_0. We want to find the chirp function DFT($D[\delta]$) $= \tilde{f}(\nu)$, so we can divide by it, as in Equation 28. We will calculate the chirp function in two ways.

For the first way, consider that we have defined the frequency $\nu(t)$ arriving at time t to be the time derivative of the exponent of the time-domain amplitude; see Equation 32. Let's assume (without rigorous justification) that the function's inverse $t(\nu)$ is the frequency derivative of the exponent of the frequency-domain amplitude. (Where the frequency-domain amplitude is defined to be the Fourier transform of the time-domain amplitude.)

In that case, we can find the exponent of the frequency-domain chirp function simply by integrating Equation 30. Define

$$\nu_0 = \nu(0) = A^{1/2}(-t_0)^{-1/2} = 1420 \text{ MHz}. \tag{33}$$

Then

$$t(\nu) = -A(\frac{1}{\nu^2} - \frac{1}{\nu_0^2}). \tag{34}$$

The minus sign on the right hand side has been inserted due to some details of the Fourier transform; the specifics will be given in the second derivation, below. We can find the exponent of the frequency-domain amplitude by integrating this equation; we get:

$$\begin{aligned} \int t(\nu) &= A(\frac{1}{\nu} + \frac{\nu}{\nu_0^2} + C) && (35)\\ &= A(\frac{\nu_0^2}{\nu\nu_0^2} + \frac{\nu^2}{\nu\nu_0^2} + C\frac{\nu_0^2\nu}{\nu\nu_0^2}) && (36)\\ &= A\frac{(\nu-\nu_0)^2}{\nu\nu_0^2}. && (37) \end{aligned}$$

where we have judiciously chosen $C = -2$ to simplify the equation. This results in a frequency domain amplitude:

$$\tilde{f}(\nu) = \exp(2\pi i A\frac{(\nu-\nu_0)^2}{\nu\nu_0^2}). \tag{38}$$

This is an easy way to get the correct answer. However, we have proceeded here from an assumption that $t(\nu)$, defined to be the inverse function of $\nu(t)$, is the derivative of the exponent of $\tilde{f}$. But this is technically speaking not the definition of $\tilde{f}$. To quantitatively justify this result will take a bit longer. We can go back to the definition:

$$\begin{aligned} \tilde{f}(\nu) &= \int_{-\infty}^{\infty} f(t)\exp(-2\pi i t\nu)dt && (39)\\ &= \int_{-\infty}^{\infty} \exp(2\pi i(\int_0^t \nu(t')dt' - \nu t))dt. && (40) \end{aligned}$$

We evaluate this integral using the method of steepest descent, which is used to calculate an integral of the form

$$\int_{-\infty}^{\infty} e^{g(t)} dt. \tag{41}$$

3.2.6 Method of steepest descent: justification

How can we show that we are justified in applying the method of steepest descent? The method is typically considered valid if $\dot{g} = 0$ at some $t = \tilde{t}$, and the path of integration can be arranged so that $\Re(g)$, the real part of g, has a global maximum (along the path) at $\tilde{t}$. Also, the third and subsequent derivatives of g should be small enough that g can be treated as quadratic, at least until $\Re(g)$ becomes much smaller than its maximum. Then $e^{g(t)}$ can be treated approximately as a Gaussian.

In our case g is pure imaginary along the path of integration, leading to an oscillating complex exponential term. So we cannot have $\Re(g)$ smaller than the maximum, although it is conceivable that we could achieve this by changing the path of integration. But instead, let's show that $e^{g(t)}$ can be treated as a Gaussian anyway. A rapidly oscillating term is small, so the right and left "tails" can be ignored; for instance, what is $\int_{t_1}^{\infty} \cos(g(t))dt$ where $g(t)$ and $g'(t)$ are monotonically increasing? We can change variables like $dt = t'(g)dg$ to get $\int_{g_1}^{\infty} \cos(g)t'(g)dg$. Then $t'(g)$ is monotonically decreasing, so that each half-period integral of $\cos(g)$ is smaller than the previous. Since these integrals alternate in sign, the whole integral is on the order of $t'(g_1) = 1/g'(t_1) = 1/\omega_1$, if ω_1 is the initial frequency.

In Equation 49, we will calculate that the magnitude of the integral in Equation 41 is $\sqrt{\frac{2A}{\nu^3}}$. We want to show that the sizes of the left and right tails are much smaller than that, so that we can ignore them. The "tail" is the non-Gaussian part of the integrand of Equation 41. Roughly, we want $1/g'(t) \ll \sqrt{\frac{2A}{\nu^3}}$ by the time any derivatives matter that are higher order than $g''(t)$. First, let's find the region in which the third and higher order derivatives don't matter. Note that ν is a constant throughout this calculation, as we are computing $\tilde{f}(\nu)$ for some particular value of ν. But $\nu(t)$ refers to something entirely different; it is a function of t. Then:

$$\begin{aligned}
g(t) &= 2\pi i(\int_0^t \nu(t')dt' - \nu t), && (42)\\
g'(t) &= 2\pi i(\nu(t) - \nu) = 2\pi i(A^{1/2}(t - t_0)^{-1/2} - \nu), && (43)\\
g''(t) &= -\pi i A^{1/2}(t - t_0)^{-3/2}, && (44)\\
g'''(t) &= \frac{3}{2}\pi i A^{1/2}(t - t_0)^{-5/2}. && (45)
\end{aligned}$$

The $g'''(t)$ term in the Taylor series expansion is insignificant as long as

$$|\frac{1}{4}\pi A^{1/2}(\tilde{t}-t_0)^{-5/2}(t-\tilde{t})^3| \ll 1. \tag{46}$$

Then

$$\nu = \nu(\tilde{t}) = A^{1/2}(\tilde{t}-t_0)^{-1/2}, \tag{47}$$

so we require

$$|\frac{1}{4}\pi\nu^5 A^{-2}(t-\tilde{t})^3| \ll 1,$$
$$|t-\tilde{t}| \ll (\frac{4}{\pi})^{1/3}\nu^{-5/3}A^{2/3}.$$

$A = 4 \cdot 10^{15}$ Hz $\cdot$ DM (pc cm^{-3}), so for a DM of 200 pc cm^{-3} , this last RHS is on the order of $5 \cdot 10^{-4}$ s.

We want to know whether $1/g'(t) \ll \sqrt{\frac{2A}{\nu^3}}$ for all points outside this region. Then $g'(t) = 2\pi(A^{1/2}((t-\tilde{t})+(\tilde{t}-t_0))^{-1/2} - \nu)$, and suppose $t-\tilde{t} = \epsilon$ is small compared with $t_0 = -0.41$ s.

Now, $\nu \approx \nu_0$, so that $\tilde{t} = t(\nu) \approx 0$ and therefore $\tilde{t} - t_0 \approx -t_0$. This means that ϵ is also small compared with $\tilde{t} - t_0$. So

$$\begin{aligned} g'(t) &= 2\pi A^{1/2}(\tilde{t}-t_0)^{-1/2}(1-\frac{1}{2}\frac{\epsilon}{\tilde{t}-t_0}-1) \\ &= -\pi A^{1/2}\frac{\epsilon}{(\tilde{t}-t_0)^{3/2}} \\ &= -\pi\frac{\epsilon\nu^3}{A} \\ 1/|g'(t)| &= 8.9 \cdot 10^{-10}. \end{aligned}$$

In the last step, above, we have set $\epsilon = 5 \cdot 10^{-4}$ s, so that we are considering a point on the boundary of our region, where $g'''(t)$ is becoming important. Furthermore,

$$\sqrt{\frac{2A}{\nu^3}} \approx 1.7 \cdot 10^{-6}. \tag{48}$$

So $1/g'(t) \ll \sqrt{\frac{2A}{\nu^3}}$ as desired. This suggests that for more extreme values of t, where $g'''(t)$ starts to become important, the integral of the oscillating amplitude becomes insignificant. We are probably justified in applying the method of steepest descent, and treating the integrand as a Gaussian.

3.2.7 Method of steepest descent: result

The method of steepest descent says that the value of the integral is:

$$\pm\sqrt{\frac{2\pi}{|g''(\tilde{t})|}}\exp(g(\tilde{t})). \tag{49}$$

The factor in front is just $\pm\sqrt{\frac{2\pi}{|g''(\tilde{t})|}} = \pm\sqrt{\frac{2A}{\nu^3}}$ using Equations 44 and 47. Then we consider the next factor,

$$\begin{aligned}
g(\tilde{t}) &= 2\pi i(\int_0^{\tilde{t}} \nu(t')dt' - \nu\tilde{t}) \\
\nu(t') &= A^{1/2}(t-t_0)^{-1/2} \\
\int_0^{\tilde{t}} \nu(t') &= A^{1/2}2(t-t_0)^{1/2}|_0^{\tilde{t}} \\
\tilde{t} &= \frac{A}{\nu^2} + t_0 \\
\int_0^{\tilde{t}} \nu(t') &= A^{1/2}\cdot 2(\frac{A}{\nu^2})^{1/2} - 2A^{1/2}(-t_0)^{1/2} \\
&= 2A^{1/2}(\frac{A^{1/2}}{\nu} - (-t_0)^{1/2}) \\
g(\tilde{t}) &= 2\pi i(2\sqrt{A}[\frac{\sqrt{A}}{\nu} - (-t_0)^{1/2}] - \nu[\frac{A}{\nu^2} + t_0]) \\
&= 2\pi i(\frac{A}{\nu} - 2\sqrt{A}(-t_0)^{1/2} - \nu t_0).
\end{aligned}$$

Then, using Equation 33:

$$\begin{aligned}
(-t_0)^{-1/2} &= A^{-1/2}\nu_0, \\
t_0 &= -A\nu_0^{-2},
\end{aligned}$$

$$
\begin{aligned}
g(\tilde{t}) &= 2\pi i(\frac{A}{\nu} - 2\sqrt{A}\sqrt{A}\nu_0^{-1} + \nu A\nu_0^{-2}), \\
&= 2\pi i A\frac{(\nu-\nu_0)^2}{\nu\nu_0^2}
\end{aligned}
$$

Since $\nu \approx \nu_0$ in our application, the chirp function looks something like $\exp(2\pi i A\frac{(\nu-\nu_0)^2}{\nu_0^3})$, and the exponent would be a quadratic. But the extra factor ν_0/ν is easily included in our dedispersion algorithm, so we have done so. This extra factor gives rise to a curvature in the time vs. frequency plot of the dispersed pulse, which must be on the order of $(\nu - \nu_0)/\nu_0$, or in our case 1.25 MHz/1.42 GHz = 0.00088, which is about 1 sample out of every 1000. Therefore, this curvature could be quite significant for small coadds, especially for large DM.

3.3 Algorithm logic

Astropulse loops through the data at several nested levels. We consider DMs ranging from -830 pc cm^{-3} to -49.5 pc cm^{-3}, and from 49.5 pc cm^{-3} to 830 pc cm^{-3}.

In the list below, each single iteration of loop N involves several iterations of loop $N+1$. For instance, running on one large DM chunk entails running on 8 small DM chunks, each of which involves running through all 2,048 chunks of the time series data.

1. Large DM chunk: blocks of 128 DMs at a time.

   At the highest level of the nested loop structure, Astropulse considers groups of 128 DMs at a time, analyzing the entire $13s$ workunit at all of these DMs. At the end of each large DM chunk loop, Astropulse runs the modified fast folding algorithm (Section 3.3.1) on the entire data set, with a resolution of 128 samples.

1.a Small DM chunk: blocks of 16 DMs at a time.

   Every time Astropulse completes 16 DMs, we say that it has finished a small DM chunk. At the end of each small DM chunk, Astropulse runs the modified fast folding algorithm on the first 1,048,576 $= 2^{20}$ samples of data, with a resolution of 16 samples.

2. Data chunks of size 32,768 $= 2^{15}$ samples.

   In order to analyze the data using a set of 16 consecutive DMs, Astropulse divides the time series into data chunks of size 2^{15} samples. However, the starting points for these data chunks occur every 2^{14} samples; there is a 50% overlap. So each piece of the time series is analyzed as the first half of one data chunk, and the second half of another. The 50% overlap can catch some pulses that would otherwise extend beyond the edge of a data chunk; part of the dispersed pulse would exist in one data chunk, and part in another. We want to operate on the whole pulse with a single Fourier transform.

   Within this loop, we compute the Fast Fourier Transform (FFT) of the data for use in convolution, producing a frequency spectrum that will later be dechirped and then FFT'ed in reverse.

But the precise DM is not chosen until a lower-level loop. Why are we allowed to compute the FFT before we have decided on a DM? Recall that each deconvolution involves two FFTs, one forward and one backward. The DM determines the chirp function, which is applied after the forward FFT. Therefore, we can don't have to perform the forward FFT once for every single DM.

3. All DMs within a small DM chunk.

   When considering one 2^{15} sample data chunk, Astropulse runs through the current 16 DMs individually.

4. Sign of the DM (positive and negative).

   We consider negative DMs for a few reasons:

   (a) Extraterrestrial civilizations might communicate using signals dispersed with negative DMs.

   (b) Signals detected at both positive and negative DM can be a sign of radar or other radio frequency interference (RFI).

5. Scales (or "co-adds") $0 - 9$.

   Scale (or co-add level) ℓ means that the client combines 2^ℓ adjacent samples and measures the total power. This allows us to search for pulses of any width, ranging from 0.4 μs (1 sample) to 204.8 μs (512 samples.)

6. Samples within the data chunk

   If the power in 2^ℓ samples is above a certain threshold, then Astropulse reports a pulse.

3.3.1 A modified folding algorithm

To search for periodic pulses, we employ a folding algorithm, inspired by the Fast Folding Algorithm (FFA) of Staelin (1969). Staelin's algorithm searches efficiently through a series of nearby periods. (Say, periods of 3, $3\frac{1}{3}$, $3\frac{2}{3}$, and 4.) Our modified folding algorithm, on the other hand, searches efficiently through a series of periods that differ by successive factors of 2. (For instance, periods of 12, 6, 3, and $1\frac{1}{2}$.) The modified algorithm is described in Korpela et al. (2000), and can be summarized as follows.

Before folding, we collapse our data in the time domain by summing either 16 or 128 samples, depending on whether it was created during the large or small DM chunk loop. Below, a collapsed set of samples will be called a "bin." The FFA has several nested loops; each single iteration of loop N generally involves many iterations of loop $N + 1$:

1. Loop over frequencies

   We define the frequency to be the number of periods over the entire time series. The smallest frequency in this loop is always $F_{\min} = 137$, and the largest frequency is twice that. The frequency increments via multiplication by $(1 + \frac{1}{N})$, where N is the number of bins in the sub buffer.

We construct a new, shorter time series from the original time series, by folding it. That is, we divide the time series into F chunks, where F is the frequency, and add them up.

2. Loop over subfrequencies

   This part is what makes the folding "fast." Instead of re-folding at all frequencies greater than $2F_{\text{min}}$, we simply fold frequency F in half to make $2F$, again to make $4F$, and so on.

3. Loop over co-adds

   As in the single pulse algorithm, we search for pulses that take up more than one bin. In this case, of course, a single bin could be 128 samples, two bins would be 256 samples, etc.

4. Loop over bins

   Examine each bin to see if the power exceeds the threshold, see 3.4.

We can compute the dependence of the run time on N, the number of bins, as follows.

1. We must search $O(N)$ frequencies, since the frequencies differ by the ratio $(1 + \frac{1}{N})$.
2. We must search $O(N)$ bins (or phases of the repeating signal) for each frequency.

This makes an overall $O(N^2)$. The run time is also influenced by the number of times we run the modified folding algorithm. (E.g. once every 128 DMs or every 16 DMs.)

3.4 Thresholds

Astropulse searches for pulses whose power exceeds certain thresholds. These thresholds can be calculated either experimentally or theoretically. I'll start by finding the theoretical values, then point out some of the uncontrollable factors that make these values inaccurate, and finally I'll describe the experimental methods for calculating thresholds.

3.4.1 Single pulse thresholds: theory

We want to calculate the distribution (pdf) of the integrated noise power in 2^ℓ samples, after dechirping. (Here, the power refers to the absolute value of the square of the amplitude, where the undispersed time series has amplitudes ± 1.)

First, we assume that the pre-deconvolution time series is pure white noise; that is, each bit of a two bit complex sample is independently distributed with equal probability of a 0

or 1, so each $f(t)$ has equal probability for $\pm 1 \pm i$. Then we deconvolve this data by FFT. In other words,

$$\tilde{f}(k) = \frac{1}{\sqrt{N}} \sum_{t=0}^{N-1} f(t) e^{-2\pi i k t/N}. \tag{50}$$

The distribution of a single $\tilde{f}(k)$ is Gaussian (by the central limit theorem), and the variance of $\Re(\tilde{f}(k))$ comes from the sum of the $2N$ variances of the $\Re(f(t))$ and $\Im(f(t))$, or $2N(\frac{1}{\sqrt{N}})^2 E(\sin^2) = 1$. We will then multiply by a chirp function which looks like $e^{i\text{phase}(k)}$, and doesn't affect the individual variances. Finally, we run the inverse FFT. We assume that the chirp function has scrambled the phases, so that the result is once again a sum of independent and identically distributed (iid) random variables. The previous argument applies, and since the variances of $\Re(\tilde{f}(k))$ and $\Re(f(t))$ are both 1, we get the same result as before: a standard Gaussian, $\sigma = 1$.

The power in each sample after deconvolution will be distributed like $|\Re(\tilde{f}(k))|^2 + |\Im(\tilde{f}(k))|^2$, the sum of the squares of two standard Gaussians. This distribution is easily calculated; the joint distribution is:

$$\begin{aligned}
& \frac{1}{\sqrt{2\pi}} e^{\frac{x^2}{2}} \cdot \frac{1}{\sqrt{2\pi}} e^{\frac{y^2}{2}} dx dy && (51)\\
= \quad & \frac{1}{2\pi} e^{-\frac{r^2}{2}} r dr d\theta && (52)\\
\rightarrow \quad & e^{-\frac{r^2}{2}} r dr && (53)\\
= \quad & e^{-u} du. && (54)
\end{aligned}$$

where $u = \frac{x^2+y^2}{2}$ is half the power in one sample, in the time domain. Therefore, half the power is exponentially distributed with mean 1; or equivalently, the power is exponentially distributed with mean 2.

In future calculations, we will normalize to half of this power, so that the average power per sample is 1.

So for instance, if after dechirping we find that a certain sample has a power of 15.3, we conclude that only one in $e^{15.3}$ samples has a comparable power. To ascertain how unlikely this is, we need to calculate how many such samples we have examined over the entire course of the experiment. This would be:

$$\begin{aligned}
48 \text{ TB } \times 4 \cdot 10^{12} \text{ samples per TB } \times 14208 \text{ DMs} \\
\times 2 \text{ DM signs } \times 2 \text{ from co-adds} \\
= 1.09 \times 10^{19} = e^{43.8}.
\end{aligned} \tag{55}$$

(Co-adds cause a factor of 2 because there are half as many potential pulses at each co-add, compared with the previous co-add. So $1 + \frac{1}{2} + \frac{1}{4} + \ldots + \frac{1}{256} \approx 2$.)

With a threshold of 43.8, we would rule out all but one noise pulse over the course of our entire observation history. However, it seems more prudent to allow one noise pulse per workunit, and sort out false pulses later.

In this case, we just want:

$$2^{25} \text{ samples per workunit } \times 14208 \text{ DMs } \times 2 \text{ signs } \times 2 \text{ from co-adds } = e^{28.3}. \tag{56}$$

When we collapse $n = 2^\ell$ samples to make one bin, we are adding up that many exponential distributions. The result is a gamma distribution, with scale parameter 1 and shape parameter n. The pdf is

$$\frac{1}{\Gamma(n)} x^{n-1} e^{-x}. \tag{57}$$

and the cumulative distribution function is

$$\frac{\gamma(n, x)}{\Gamma(n)}. \tag{58}$$

where γ is the incomplete gamma function. The first few pdfs are shown in Figure 9.

3.4.2 Expected discrepancies with the model

A few differences from the model can be expected:

1. **Hydrogen line and filter shape**. We assumed above that the input data is white noise. In practice, this is not the case, because a portion of our band has higher power due to the hyperfine hydrogen line. The strength of this line can vary depending on our RA and dec. Then $\tilde{f}(k)$ no longer have equal standard deviations. This will cause some correlation between the deconvolved power of adjacent samples, which will modify the pdf of the binned power, increasing the variance.

   To see this, consider the simplest, most extreme case, where the hydrogen line is a strong delta function of amplitude A at frequency k_0, where A is distributed randomly like a Gaussian with standard deviation σ. Then if $\bar{f}$ is the dechirped amplitude, $\bar{f}(t) = Ae^{2\pi i k_0 t/N}$. (The dispersion is not relevant, since the hydrogen line has a single frequency.) So $|\bar{f}(t)|^2$ is exponential with power σ^2. But $|\bar{f}(t_1) + \bar{f}(t_2)|$ is

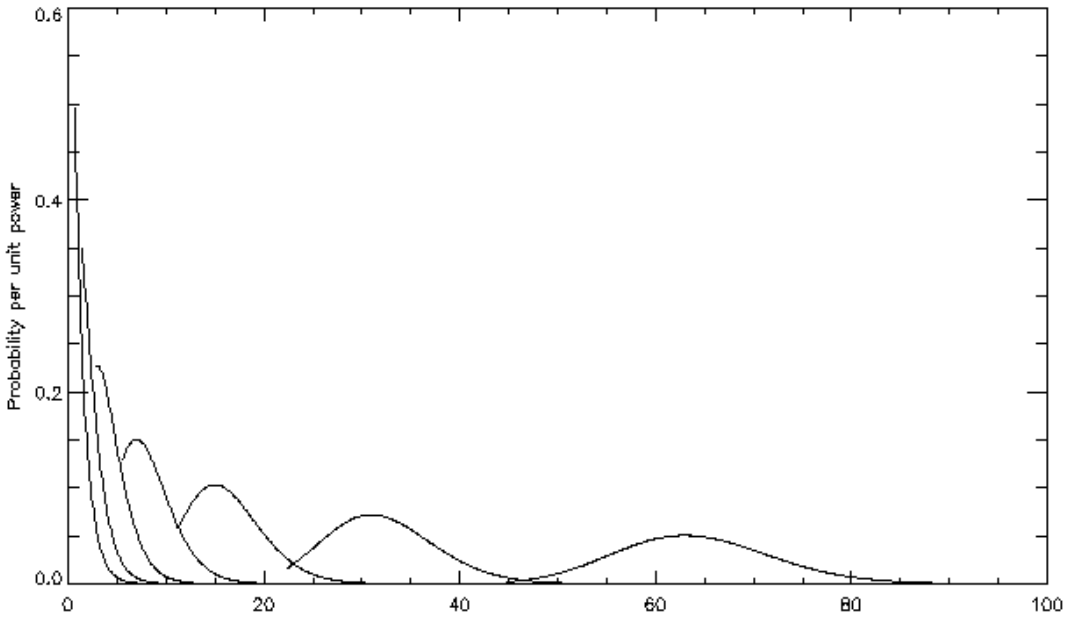


Figure 9: Gamma distributions. The x axis is integrated power (divided by 2, as per our convention), and the y axis is probability per unit power. The leftmost distribution, which is exponential, belongs to $n = 1$. The rest are $n = 2, 4, 8, 16, 32, 64$. Notice that the rightmost distribution is nearly a normal distribution.

$A|e^{2\pi ik_0t_1/N} + e^{2\pi ik_0t_2/N}|$. If $t_1 \sim t_2$, this is roughly $2A$, which is Gaussian with standard deviation 2σ and variance $4\sigma^2$. Whereas if we summed samples without the hydrogen line, adding identical and independently distributed (iid) exponentials, the variances would simply add to give $2\sigma^2$. So this model hydrogen line increases the variance.

In actuality, the effect of the hydrogen line is not so pronounced, but the idea is similar. In the same way, the nonuniform shape of our low pass filters also causes the signal to differ from white noise.

2. **Other disparities**

   Even in the absence of the hydrogen line, tests reveal other differences between the theoretical and actual distributions. For larger bin sizes, the variance is less than expected.

   It's easy to see that power per sample cannot be independently distributed, even in the case of white noise. This is because the total power over all samples must be a constant; in our case, the constant is $32768 = 2^{15}$, the total number of samples in a FFT. This would certainly result in a smaller variance, but we have not established whether this effect suffices to explain the observed disparity.

3.4.3 Repeating pulse thresholds: theory

For repeating pulses, the situation is simpler. We test a repeating pulse by adding up M samples, where M is very large. These samples are not all consecutive. For the minimum

value of M, we add groups of 16 samples together, with a frequency of $F_{\min} = 137$, for a total of $M = 16 \cdot 137 = 2192$. This sum of many iid variables will have a normal distribution with variance M and standard deviation $\sqrt{M}$.

The total number of potential pulses per 13 s workunit is as follows.

1. **Frequencies**. Suppose the number of samples (which is 2^{20} or 2^{25}) is n, and the DM chunk size (16 or 128) is d. Then there are L frequencies, where $(1 + \frac{1}{n})^L = 2$. That is, $e^{\frac{L}{n}} = 2$, so $L = n \cdot \ln(2)$.

2. **Phases**. Each frequency has a period equal to about $n/F_{\min}$, and the phase of the repeating pulse can start at any point in that period.

3. **Subfrequencies**. Each frequency has a number of subfrequencies, each of which has half as many bins as the previous. This doubles the number of potential pulses.

4. **Coadds**. Each subfrequency has a number of co-adds, each of which has half as many bins as the previous. This doubles the number of potential pulses again.

5. **Number of DM chunks**. The FFA runs $\frac{14208}{d}$ times for each of 2 signs, where d is the DM chunk size as above.

Therefore, the total number of potential pulses is

$$n \cdot \ln(2) \cdot (n/F_{\min}) \cdot 4 \cdot (14208/d) \cdot 2 = (n^2 \cdot 8 \cdot \ln(2) \cdot 14208)/(F_{\min} \cdot d^3), \tag{59}$$

which is $3.09 \times 10^{11} = e^{26.5}$ for the large DM chunk, and $1.54 \times 10^{11} = e^{25.8}$ for the small DM chunk.

We want the number of standard deviations such that the probability of exceeding that number is $e^{-26.5}$ or $e^{-25.8}$, respectively. We can perform the integration in idl, finding that we need thresholds of 6.88σ in the large DM chunk case and 6.78σ in the small DM chunk case, where $\sigma = \sqrt{M}$.

3.4.4 Single pulse thresholds: experiment

To choose our thresholds for the single pulse search, we ran the client on 10 workunits, keeping track of the strongest pulses we found at each co-add. The second largest pulse out of 10 is roughly the 90th percentile, so we set our thresholds at that point. This method gives thresholds that are reasonable as long as we don't demand that we detect precisely equal numbers of pulses at each co-add.

3.5 Expected sensitivity

3.5.1 Sensitivity of Astropulse

To calculate Astropulse's expected sensitivity as a pulse area in Jy μs, we will follow the treatments in (Rohlfs & Wilson, 2000) and (Van Vleck & Middleton, 1966). A signal from a telescope receiver consists of a function $f(t)$ that describes the amplitude of the signal at time t. However, we will be discussing a generalization of a signal, called a "random process." A random process is not a particular signal, but a probability distribution over the set of all signals; it is a random variable whose values are signals. It is natural to talk about noise as a random process, since we do not know in advance the amplitude of the noise as a function of time. The random process is associated with a probability density function (pdf) on the space of all signals. Because the signals are functions, this pdf is a function on a domain of functions; so we could call it a functional. Such a functional could be written $f[x(t)]$, where the function $x : \mathbb{R} \to \mathbb{C}$ is a signal with some amplitude at each time t. So for instance, the random process would assign some probability density to the signal $x_1(t) = \sin(t)$, and would assign some other probability densities to the signals $x_2(t) = 2\sin(t)$, $x_3(t) = \sin(2t) - \sin(t)$, and so on.

Another, equivalent way of thinking about a random process is that it can be written as $X(t)$, which is a random variable for each value of t, not necessarily independent. (Two different random variables need not be independent. For instance, the amplitude of the hyperfine hydrogen line at time t is a random variable, where we have applied a filter to isolate a narrow band around the desired frequency. The amplitude at time $t + T$, where $T = \frac{1}{\nu}$ and ν is the frequency of the hydrogen line, is another random variable. But these two random variables are not independent, and they will tend to have the same value if we are considering a narrow enough band.)

We will consider Gaussian random processes; namely, processes X which can be written as a sum like:

$$X(t) = \sum_{\nu} A_\nu (Z_\nu + i\bar{Z}_\nu) e^{2\pi i \nu t}, \tag{60}$$

where A_ν are positive real constants, and $Z_\nu, \bar{Z}_\nu$ are random variables distributed independently as real Gaussians with $\sigma = 1, \mu = 0$. For such processes, the value $X(t)$ for any t is distributed as a complex Gaussian with variance $\sum_\nu A_\nu^2$ in the real and imaginary components. But although Z_ν are independent, $X(t_1), X(t_2)$ may not be independent. In general, there is some correlation coefficient r such that the joint pdf of the real (or imaginary) components of $X(t_1), X(t_2)$ is:

$$p(x_1, x_2) = \frac{1}{2\pi\sqrt{1-r^2}} \exp\left(\frac{1}{1-r^2}(x_1^2 - 2r x_1 x_2 + x_2^2)\right). \tag{61}$$

To calculate Astropulse's sensitivity, we want to relate the following quantities. Subscript

0 refers to the signal emitted by the receiver and entering the one-bit sampler. Subscript 1 refers to the signal emitted by the one-bit sampler.

F = the signal flux entering the telescope (from an astrophysical source).

G = the gain of the telescope, in K Jy^{-1}.

kT_0 = the average noise power emitted by the receiver, per unit bandwidth, and entering the one-bit sampler.

S_0 = the average signal power emitted by the receiver and entering the one-bit sampler.

P_1 = the average noise power emitted the one-bit sampler, in dimensionless units.

S_1 = the average signal power emitted by the one-bit sampler, in dimensionless units.

We will assume, for simplicity, that the power S_0 comes from a signal with amplitude $X_0(t) = A(Z_{\nu_0} + i\bar{Z}_{\nu_0})\exp(2\pi i\nu_0 t)$ for a single ν_0. We require that $E(|X_0|^2) = 2A^2$ is the average power S_0, where E denotes expected value. The Gaussian distribution may seem incorrect – it allows the possibility of zero signal power, for instance. But it turns out that we can calculate the precise effect of the sampler on the signal X_0, so we will make that calculation and then assume that the nonzero expected power is what's important.

Next, we consider autocorrelations, and their relationship with power spectra. The autocorrelation $R_X(\tau)$ is a property of a random process $X(t)$, and has a delay parameter τ. It measures the correlation between the signal at time t and at time $t + \tau$. It's defined by:

$$R_X(\tau) \equiv E(X^*(t)X(t+\tau)). \tag{62}$$

The autocorrelation is useful in several ways. The simplest is that $R_X(0) = E(|X(t)|^2)$, the expected power. Another fact is that the Fourier transform of R is the same as the power spectral density (PSD) $S(\nu)$. The PSD is defined as the expected value of the square of the amplitude in the frequency domain, $S(\nu) \equiv E(|\tilde{X}(\nu)|^2) = 2|A_\nu|^2$.

Now we compute R_{X_0}, the autocorrelation of X_0:

$$\begin{aligned} R_{X_0}(\tau) &\equiv E(X_0^*(t)X_0(t+\tau)) && (63)\\ &= 2A^2\exp(2\pi i\nu_0\tau) && (64)\\ &= S_0\,\exp(2\pi i\nu_0\tau), && (65)\end{aligned}$$

where the last step is consistent with the fact that $R_{X_0}(0)$ must equal the signal power S_0. Now the total power emitted by the receiver is $kT_0B + S_0$, which comes from a signal with amplitude $Y_0(t) + X_0(t)$, where $Y_0(t)$ is white noise. Since Y_0 and X_0 have a random phase

relationship, the autocorrelation function of the sum is just the sum of the autocorrelation functions. (That is, $E(X_0^*(t)Y_0(t+\tau)) = E(Y_0^*(t)X_0(t+\tau)) = 0$). Then the noise is not correlated with itself over long time scales τ, so R_{Y_0}, the autocorrelation of Y_0, is:

$$\begin{aligned} R_{Y_0}(\tau) &= E(Y_0^*(t)Y_0(t+\tau)) && (66)\\ &= kT_0Bg(\tau), && (67)\end{aligned}$$

for some narrow function $g(\tau)$. The total noise power is $E(|Y_0^2|) = R_{Y_0}(0)$, which must be equal to kT_0B, where B is the bandwidth. Therefore $g(0) = 1$. So

$$R_0(\tau) = kT_0Bg(\tau) + S_0\,\exp(2\pi i\nu_0\tau). \tag{68}$$

We need to characterize the output signal that results from clipping R_0. This problem is considered in (Rohlfs & Wilson, 2000) and (Van Vleck & Middleton, 1966). In the case of a Gaussian signal, the joint pdf of amplitudes at any two times (i.e. $X_0(t_1), X_0(t_2)$) takes a closed form as above. From that form, one can derive the correlation between the probability that the clipped noise will be ± 1 at t_1 and t_2. Then one can write down the autocorrelation function of the clipped noise. For the real bits alone, the autocorrelation is:

$$R_1(\tau) = \frac{2}{\pi}\arcsin\left(\frac{R_0(\tau)}{R_0(0)}\right). \tag{69}$$

As expected, the power is $R_1(0) = 1$. For the real and complex bits together, the autocorrelation must be twice as large, $R_1(0) = 2$. For $S_0 \ll kT_0B$, the total power is mostly due to the noise kT_0. However, at values $\tau \neq 0$, the contribution of $g(\tau)$ falls off quickly. Therefore,

$$\begin{aligned} R_1(\tau) &= 2\cdot\frac{2}{\pi}\arcsin\left(\frac{S_0\ \exp(2\pi i\nu_0\tau)}{kT_0B}\right) && (70)\\ &\approx 2\cdot\frac{2}{\pi}\frac{S_0\ \exp(2\pi i\nu_0\tau)}{kT_0B}, && (71)\end{aligned}$$

where we are ignoring $R_1(0)$, because $g(\tau)$ is narrow and we are only going to use $R_1(\tau)$ in an integral. The power at frequency ν_0 is found by integrating

$$S_1(\nu) = \int R_1(\tau)e^{-2\pi i\nu\tau}d\tau = 2\cdot\frac{2}{\pi}\delta(\nu-\nu_0)\frac{S_0}{kT_0B}. \tag{72}$$

which says that the total power near ν_0 is $\int S_1(\nu)d\nu = 2\cdot\frac{2}{\pi}\frac{S_0}{kT_0B}$. Then the ratio $\frac{S_{1,\mathrm{near}}(\nu_0)}{R_1(0)} = \frac{2}{\pi}\frac{S_0}{kT_0B}$. In other words, $S_1/P_1 = \frac{2}{\pi}S_0/P_0$.

As discussed above, in Section 3.4.1, the integrated power has a gamma distribution, both in the frequency domain and (when dechirped) in the time domain. To summarize the results from that section: in the absence of any input signal S_0, the frequency components' amplitudes should behave like Gaussian variables. So after a discrete Fourier transform, $P_1(\nu_0)$ has an expected power of 2, and is distributed as a sum of the squares of two Gaussians (real and imaginary.) This distribution could be seen as a chi squared with 2 degrees of freedom, a gamma distribution, or an exponential distribution. In the time domain, if we dedisperse the noise by performing a Fourier transform, dividing by a chirp function (dechirping), and then performing the inverse Fourier transform, we should still end up with a sum of a real and a complex Gaussian. This should give us an exponential distribution for each time sample.

Now assume that a chirped pulse with energy E_0 emitted by the receiver will have

$$\frac{2}{\pi}(\frac{E_0}{kT_0Bt_{\mathrm{sample}}}) \tag{73}$$

times the expected power in one sample after the one-bit sampler. (This assumption is reasonable, since a chirped pulse looks like a monochromatic signal locally. Therefore we can apply the above reasoning, even though our proof was for monochromatic signals.) The integrated noise will have N times the expected power of one sample, where the duration of the pulse is N samples, so the pulse will be detectable if

$$\frac{(2/\pi)E_0 + kT_0BNt_{\mathrm{sample}}}{kT_0Bt_{\mathrm{sample}}} \tag{74}$$

is above threshold for N samples, where the threshold comes from a chi squared with $2N$ degrees of freedom. Astropulse sets this threshold to $H \sim 30$ for $N = 1$ sample, as discussed in Section 3.4.1. Since the pdf of the noise power is exponential, this excludes all but a fraction e^{-H} of the noise.

Then the minimum detectable energy satisfies the equation:

$$\begin{aligned}
\frac{2}{\pi}E_0 + kT_0BNt_{\mathrm{sample}} &= kT_0Bt_{\mathrm{sample}}H && (75)\\
\frac{2}{\pi}E_0 &= kT_0Bt_{\mathrm{sample}}(H-N) && (76)\\
E_0 &= \frac{\pi}{2}kT_0Bt_{\mathrm{sample}}(H-N) && (77)\\
E_0 &= \frac{\pi}{2}kT_0(H-N) && (78)
\end{aligned}$$

using the fact that $Bt_{\text{sample}} = 1$. If a signal is detected with $\tilde{H}$ times the average power in one sample, the implied energy of the pulse can be determined by inserting $\tilde{H}$ into Equation 78.

We have calculated E_0, the value of the energy emitted by the receiver. This implies that the signal has a power per bandwidth of $E_0/(BNt_{\text{sample}}) = E_0/N$, hence a temperature of $T = \frac{E_0}{Nk}$. This means the astrophysical source has a flux of $T/(2G) = \frac{E_0}{2GNk}$ in this polarization. (By definition of the gain.) If it is actually unpolarized, the total flux in both polarizations is twice that, or T/G. Then its pulse area is:

$$F \cdot (Nt_{\text{sample}}) = \frac{E_0 t_{\text{sample}}}{Gk} = \frac{\pi}{2}\frac{T_0\ t_{\text{sample}}(H-N)}{G}. \tag{79}$$

This value could be measured in Jy μs, for instance. In our case:

1. $T_0 \sim 30\ K$ is the system temperature.[3]
2. $G = 10$ K Jy^{-1} is the telescope gain, roughly equal to $\frac{A}{k}$, where:
3. $A = \frac{\lambda^2}{\Omega}$ is the effective area of the telescope and k is Boltzmann's constant.
4. $\lambda = 21$ cm is the wavelength of the signal.
5. $\Omega = 8.3 \cdot 10^{-7}$ is the beam's solid angle.

So the resulting flux density $\cdot$ duration is 1.9 $(H - N)$ Jy μs.

3.5.2 Sensitivity comparison

While Astropulse detects a signal coherently, other undirected radio surveys use incoherent detection schemes. Typically they use a filter bank, dividing the spectrum into N sub-bands as described in Section 3.2. The method amounts to measuring the deviation ΔT from the receiver temperature T_0 at successive times, but in a different channel at each time. The channel frequency varies at a constant rate that is proportional to some dispersion measure (DM) that one is testing for.

First consider a single-channel device with bandwidth B. Then following Wilson et al. (2009), we assume the spectrometer finds the power in a band by Nyquist sampling at twice the bandwidth, then finding the average of the squares. The square of a Gaussian with $\sigma = 1$ is a chi squared with one degree of freedom, having a variance of 2. In this case, the power P has $\mu = kT_0B$, so $P = X^2$, where X is a Gaussian with $\sigma = \sqrt{kT_0B}$. Then $\frac{P}{kT_0B}$ is a chi squared with variance 2, and P has variance $2(kT_0B)^2$. The variance of the $2Bt$ samples in

[3] `http://www.naic.edu/alfa/gen_info/info_obs.shtml`

time t is $2(kT_0B)^2(2Bt)$. Assuming we can detect a deviation of $m\sigma$, that's $m\sqrt{2}kT_0B\sqrt{2Bt}$. To find the standard deviation of the average power per sample, divide by the $2Bt$ samples to get $mkT_0B/\sqrt{Bt}$. Then find the equivalent temperature of this minimum detectable signal, dividing by kB to get:

$$T_{\text{eff}} = mT_0/\sqrt{Bt}. \tag{80}$$

A multichannel filter spectrometer will have the same sensitivity, assuming it can add up the correct parts of the signal. This might be problematic in the cases of time-frequency uncertainty or dispersion within sub-bands, as discussed in section 3.2. We can understand these effects as widening the pulse width by a factor $\frac{t}{W_i}$ while preserving the pulse area, so that the effective temperature T_{eff} is equal to $\frac{W_i}{t}T_{\text{int}}$ for an intrinsic temperature T_{int}. For instance, Deneva & Cordes (2008) give the sensitivity formula as:

$$T_{\text{int}} = (\frac{t}{W_i})\frac{mT_0}{\sqrt{N_{\text{pol}}Bt}}, \tag{81}$$

$$t = (W_i^2 + \Delta t_{\text{DM,ch}}^2 + \Delta t_{\text{DM,err}}^2 + \Delta t_{\text{sc}}^2)^{1/2}. \tag{82}$$

t is the effective width of the pulse,

W_i is the intrinsic width of the pulse,

$\Delta t_{\text{DM,ch}}$ is the dispersion within one channel, and is given by Equation 15,

$\Delta t_{\text{DM,err}}$ is the error caused by looking at the wrong dispersion measure, and can be calculated by inserting $\frac{1}{2}$ the DM step into Equation 15, using the whole bandwidth as $\Delta\nu$,

$\Delta t_{\text{sc}} \propto f^{-3.86}$ is the error caused by scattering broadening.

This assumes that the channel bandwidth is not so narrow that we are sampling beyond the Nyquist rate. If the channel bandwidth were that narrow, there would be another contribution to the effective width; but all surveys are careful not to sample beyond the Nyquist rate.

In any event, if one cares about the pulse area in Jy μs rather than the instantaneous flux, Equation 80 will suffice. One can calculate the area of a single polarization $A = \frac{mT_0t}{2G\sqrt{Bt}} = m\frac{T_0}{2G}\sqrt{t/B}$, or both polarizations together, $A = m\frac{T_0}{G}\sqrt{t/B}$.

The most difficult variable to quantify above is perhaps t_{sc}. We can estimate it using the empirical formula given in Lorimer & Kramer (2005):

$$\log t_{\text{sc,ms}} = -6.46 + 0.154(\log \text{DM}) + 1.07(\log \text{DM})^2 - 3.86 \log \nu_{\text{GHz}}. \tag{83}$$

However, this formula applies to sources in the Milky Way. Astropulse spends a substantial fraction of the time looking outside our Galaxy, in which case $t_{\rm sc}$ should be much smaller, even for large DMs. But the distribution of the intergalactic medium is not well understood. Lovell et al. (2007) find that extragalactic radio sources at redshifts greater than $z = 2$ do not have microarcsecond structure, suggesting that they are scatter broadened by turbulence in the IGM. A microarcsecond of angular broadening at $z = 2$ corresponds to a pulse width of $\tau = \theta^2 d/c = 8$ μs (using $d = 3.57$ Gpc). According to Ioka (2003), this is a DM of about 2000 pc cm^{-3}. So perhaps we can assume that at the (smaller) DMs of our experiment, pulses will have reasonably small widths. For instance, inside the galaxy, a DM of 830 pc cm^{-3} would have a scattering width 400 times smaller than a DM of 2000 pc cm^{-3}. Even if the scattering width were nearly 8 μs, Astropulse is still good at detecting such pulses. (The threshold is just twice as high as for 1 sample pulses.) Therefore, we will ignore the scattering error in the tables below, and set $t = t_{\rm sample}$. In the tables, the following conventions are observed:

- σ: number of standard deviations for threshold.
- $t_{\rm sample}$: the time resolution.
- t: the effective duration of the pulse after dedispersion.
- beam Ω: the telescope beam with, in steradians.
- beams: number of simultaneous beams.
- $t_{\rm obs}$: observation time per beam, in hours.
- $d_{\rm max}$: the minimum distance from which an exploding $M = 10^8$ kg black hole would be visible, in kpc. It's calculated using

$$U_{\rm min} = \text{energy}/(\text{area} \cdot \text{bandwidth}) = (Mc^2)/(4\pi d_{\rm max}^2 \cdot 1\text{GHz}), \tag{84}$$

  where $U_{\rm min}$ is the minimum detectable signal in Jy μs.
- rate: the minimum rate of black hole explosions under which such a black hole would be detectable, $V^{-1}t_{\rm obs}^{-1}$. Here, $V = (4\pi/3)d^3 n_{\rm beams}\frac{\Omega}{4\pi} = \frac{1}{3}\Omega d^3 n_{\rm beams}$ is the volume of space observed at any one time.

I conclude that Astropulse's minimum detectable rate (in black holes explosions pc^{-3} yr^{-1}) is comparable to that of other surveys, but not superior. Astropulse's rate is similar to Lorimer & Bailes (2007) and Deneva & Cordes (2008). My sensitivity to unresolved pulses, in Jy μs, is superior to all other surveys listed except for Deneva's. This sensitivity comes largely from my microsecond time resolution and high gain. My observation time is also superior. Astropulse does have substantial disadvantages, including a limited bandwidth and narrow ($\Omega = 8.1 \cdot 10^{-7}$) beams. The rate limit and some of the other parameters in these tables are plotted in Figures 27, 28, and 29, in Section 7.

Table 1: Survey parameters

#	author	telescope	year	dedisp	ref	ν_0 (MHz)	σ	T_0 (K)	$t_{\rm sample}(\mu s)$	$t(\mu s)$
1	O'Sullivan et al.	Dwingeloo	1978	incoh	[a]	5000	6	65	2	2700
2	Phinney & Taylor	Arecibo	1979	incoh	[b]	430	6	175	$1.7 \cdot 10^4$	$1.7 \cdot 10^4$
3	Amy et al.	MOST	1989	incoh	[c]	843	6	-	1	$1.7 \cdot 10^4$
4	Katz & Hewitt	STARE	2003	incoh	[d]	611	5	150	125000	125000
5	McLaughlin et al.	Parkes	2006	incoh	[e], [f]	1400	6	21	250	250
6	Lorimer & Bailes	Parkes	2007	incoh	[g]	1400	6	21	1000	1000
7	Deneva et al.	Arecibo	2008	incoh	[h]	1440	5	30	64	64
8	Von Korff et al.	Arecibo	2009	coher	-	1420	30	30	0.4	0.4

#	$\Delta\nu$ (MHz)	G (K Jy^{-1})	beam Ω	beams	$t_{\rm obs}$ (h)	$N_{\rm pol}$	sens (Jy μs)	$d_{\rm max}$ (kpc)	rate (pc^{-3}yr^{-1})
1	100	0.1	$6.6 \cdot 10^{-6}$	1	46	1	$2 \cdot 10^4$	100	$8.4 \cdot 10^{-8}$
2	16	27	$6.6 \cdot 10^{-6}$	1	292	1	1300	250	$8.7 \cdot 10^{-10}$
3	3	-	$3.6 \cdot 10^{-8}$	32	4000	1	$1.6 \cdot 10^5$ [i]	22	$5.4 \cdot 10^{-8}$
4	4	$6.1 \cdot 10^{-5}$	1.4	1	13000	2	$1.8 \cdot 10^9$	0.21	$1.6 \cdot 10^{-7}$
5	288	0.7	$1.3 \cdot 10^{-5}$	13	1600	2	120	810	$1.8 \cdot 10^{-13}$
6	288	0.7	$1.3 \cdot 10^{-5}$	13	480	2	240	580	$1.7 \cdot 10^{-12}$
7	100	10	$8.1 \cdot 10^{-7}$	7	420	2	14	2300	$9.0 \cdot 10^{-13}$
8	2.5	10	$8.1 \cdot 10^{-7}$	7	1460	1	54	1200	$1.6 \cdot 10^{-12}$

[a] O'Sullivan et al. (1978)
[b] Phinney & Taylor (1979)
[c] Amy et al. (1989)
[d] Katz et al. (2003)
[e] McLaughlin et al. (2006)
[f] Manchester et al. (2001)
[g] Lorimer & Bailes (2007)
[h] Deneva & Cordes (2008)
[i] MOST has 1 mJy of noise in each beam after 12 hours, http://www.physics.usyd.edu.au/sifa/Main/MOST

4 Distributed computing : the BOINC platform

Astropulse runs on the BOINC platform (Anderson, 2004), an acronym for "Berkeley Open Infrastructure for Network Computing." BOINC is a set of programs that organizes volunteers' home computers to perform scientific calculations. In a typical BOINC project, a researcher has a computing problem that can run in parallel, that is, on several machines at once. Perhaps the problem involves searching a physical space (for Astropulse, this space is the sky), and performing the same computation on each point in that space (for Astropulse, this computation is dedispersion.) The first BOINC project, SETI@home, searched the sky for narrowband transmissions. The space could also be a parameter space, for instance a space of potential climate models (climateprediction.net) or protein configurations (Rosetta@home). The visible manifestation of a BOINC project is an informative screen saver. Figure 10 shows the Astropulse screen saver, and Figure 11 depicts the climateprediction.net screen saver.

Although the volunteers are providing their computers for free, the bandwidth and storage space required to distribute data to the volunteers is not free. A project is suitable for BOINC only if it is computation-intensive. That is, the monetary cost to perform the computation must be greater than the monetary cost of distributing the data. Coherent dedispersion satisfies this requirement, because we must perform FFTs at many DMs.

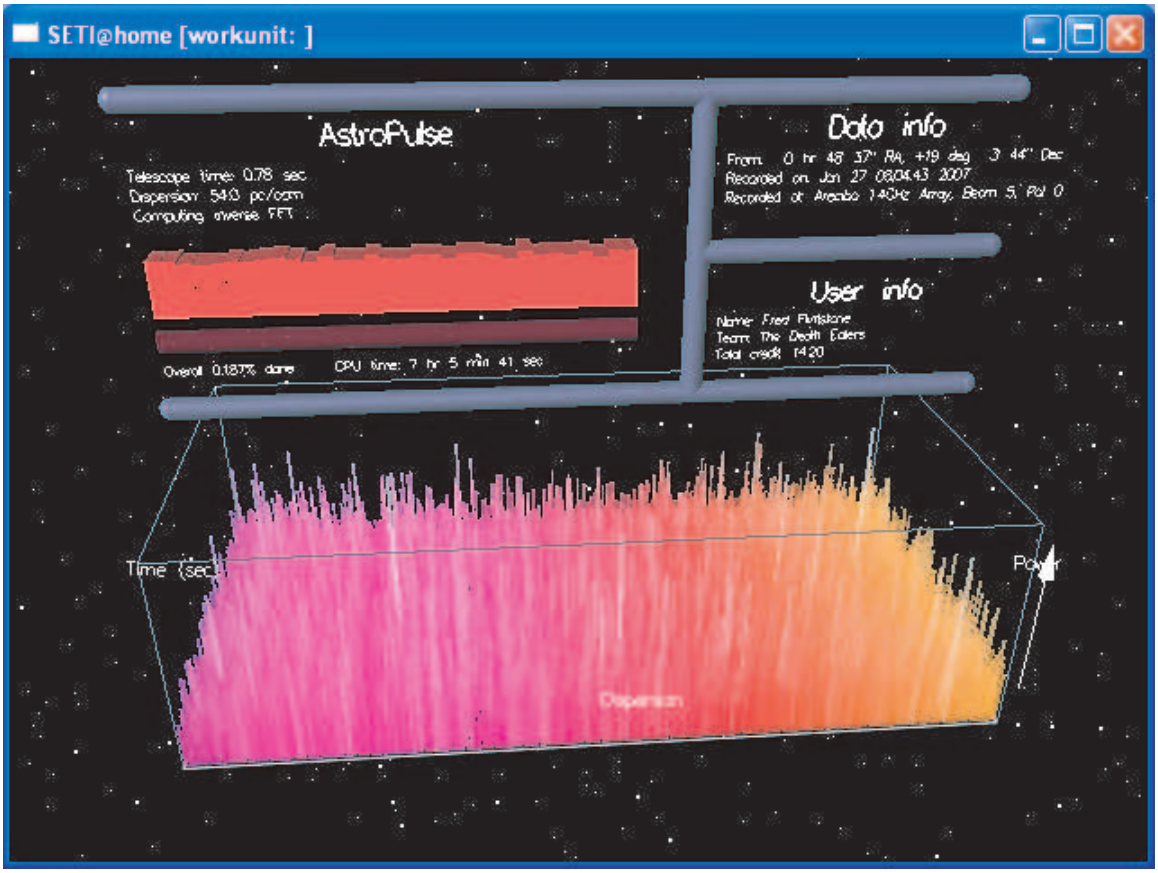


Figure 10: The Astropulse screen saver.

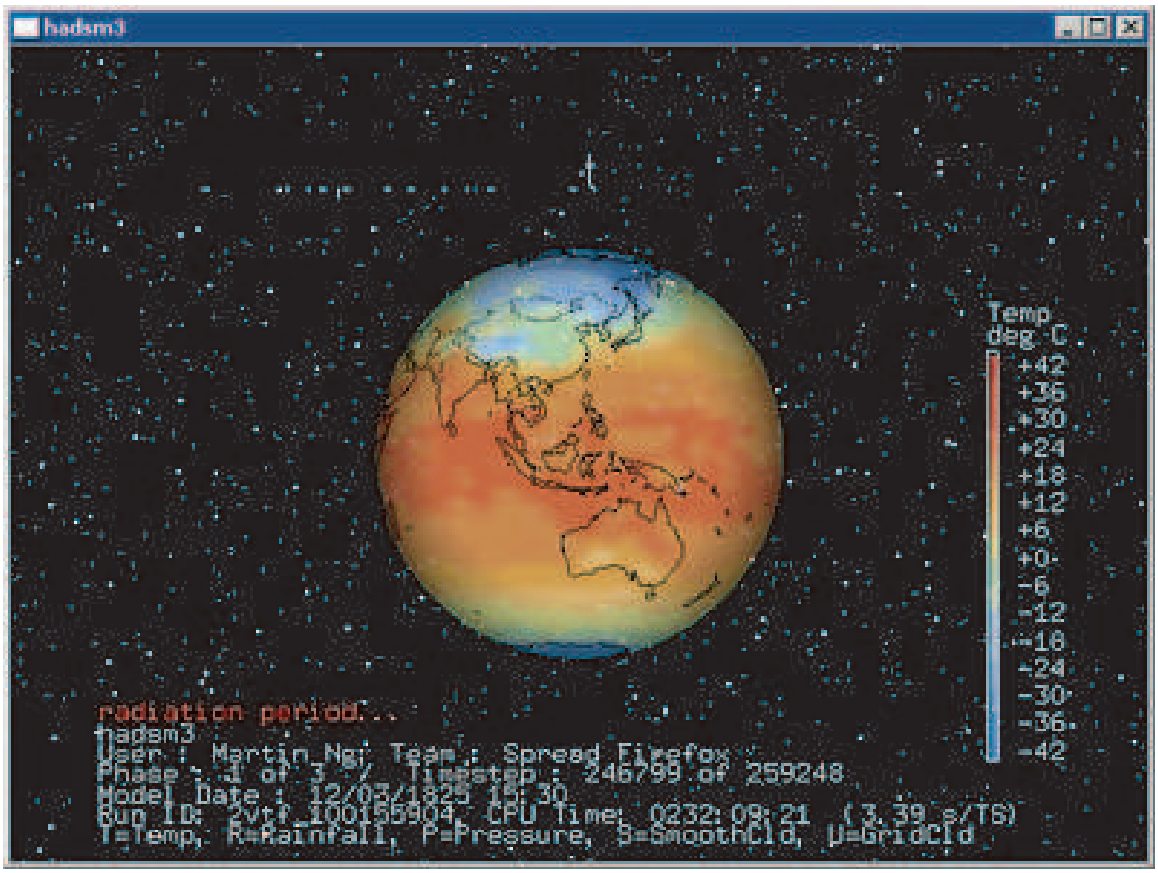


Figure 11: The climateprediction.net screen saver.

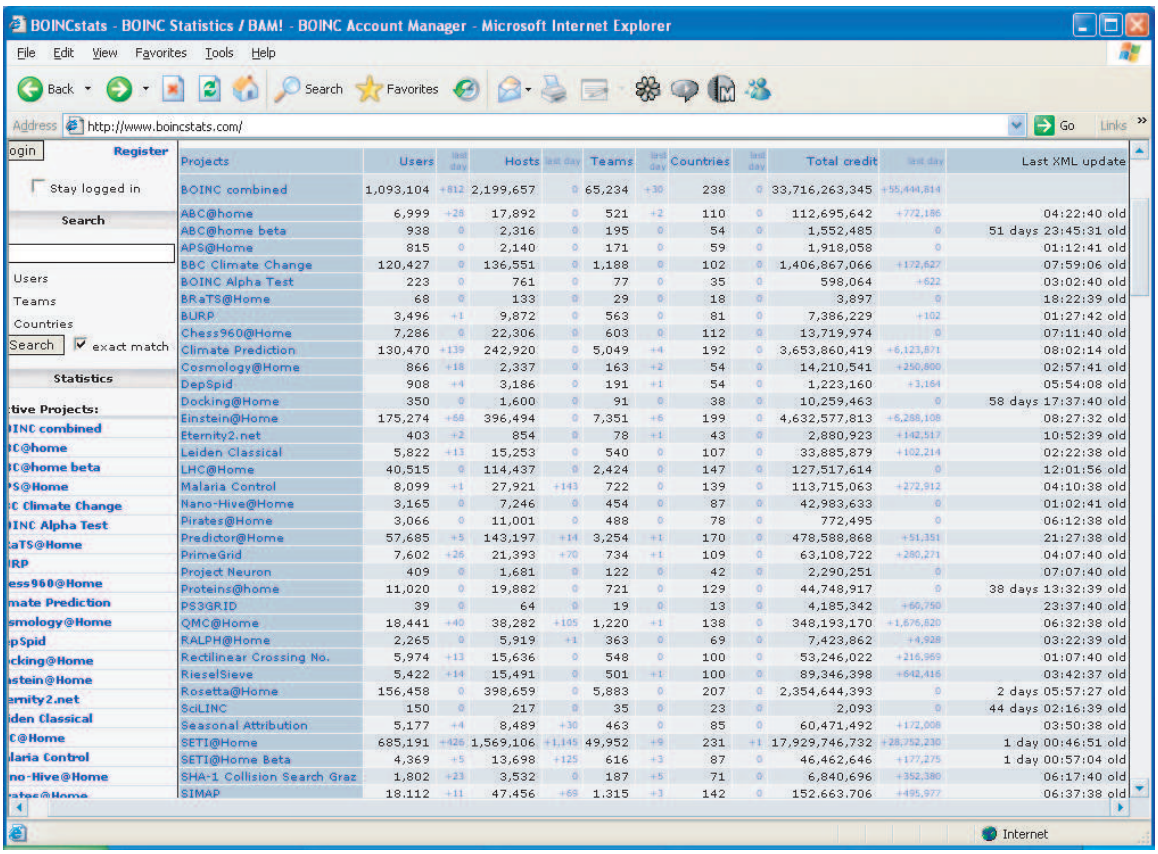
The researcher for a BOINC project need not be affiliated with UC Berkeley, or with the BOINC development team at Berkeley (although we happen to be so affiliated.) BOINC is open source, and can be downloaded, compiled, and operated by anyone with sufficient technical skills; about 50 projects currently exist outside Berkeley.

Likewise, volunteers need not have any particular technical knowledge. They just have to navigate to the BOINC web page with their web browser, and follow the instructions to download the client programs described below. Astropulse has access to around 500,000 volunteers, each of whose machines might have 2 GFLOPs of processing power, and be on 1/3 of the time, for a total of 300 TFLOPs – as much as the world's fastest general purpose supercomputer in 2007, IBM's Blue Gene / L.[5] Since that time, the processing power of the fastest supercomputer has increased to 1800 TFLOPs or more.[6] Volunteer counts for each project, as well as other statistics, are compiled and displayed on web pages as shown in Figure 12 and Figure 13. Both images are from circa 2007.

Figure 14 depicts the parts of a BOINC project and the relationships between them.

1. Data: Astropulse, in particular, processes data in 8 MB workunits, which consist of

[5] http://www.top500.org/list/2007/06/100
[6] http://www.top500.org/list/2009/11/100



Projects	Users	Hosts	Teams	Countries	Total credit	Last XML update
BOINC combined	1,093,104 +812	2,199,657 0	65,234 +30	238 0	33,716,263,345 +55,444,814	
ABC@home	6,999 +28	17,892 0	521 +2	110 0	112,695,642 +772,186	04:22:40 old
ABC@home beta	938 0	2,316 0	195 0	54 0	1,552,485 0	51 days 23:45:31 old
APS@Home	815 0	2,140 0	171 0	59 0	1,918,058 0	01:12:41 old
BBC Climate Change	120,427 0	136,551 0	1,188 0	102 0	1,406,867,066 +172,627	07:59:06 old
BOINC Alpha Test	223 0	761 0	77 0	35 0	598,064 +622	03:02:40 old
BRaTS@Home	68 0	133 0	29 0	18 0	3,897 0	18:22:39 old
BURP	3,496 +1	9,872 0	563 0	81 0	7,386,229 +102	01:27:42 old
Chess960@Home	7,286 0	22,306 0	603 0	112 0	13,719,974 0	07:11:40 old
Climate Prediction	130,470 +139	242,920 0	5,049 +4	192 0	3,653,860,419 +6,123,871	08:02:14 old
Cosmology@Home	866 +18	2,337 0	163 +2	54 0	14,210,541 +250,800	02:57:41 old
DepSpid	908 +4	3,186 0	191 +1	54 0	1,223,160 +3,164	05:54:08 old
Docking@Home	350 0	1,600 0	91 0	38 0	10,259,463 0	58 days 17:37:40 old
Einstein@Home	175,274 +68	396,494 0	7,351 +6	199 0	4,632,577,813 +6,288,108	08:27:32 old
Eternity2.net	403 +2	854 0	78 +1	43 0	2,880,923 +142,517	10:52:39 old
Leiden Classical	5,822 +13	15,253 0	540 0	107 0	33,885,879 +102,214	02:22:38 old
LHC@Home	40,515 0	114,437 0	2,424 0	147 0	127,517,614 0	12:01:56 old
Malaria Control	8,099 +1	27,921 +143	722 0	139 0	113,715,063 +272,912	04:10:38 old
Nano-Hive@Home	3,165 0	7,246 0	454 0	87 0	42,983,633 0	01:02:41 old
Pirates@Home	3,066 0	11,001 0	488 0	78 0	772,495 0	06:12:38 old
Predictor@Home	57,685 +5	143,197 +14	3,254 +1	170 0	478,588,868 +51,351	21:27:38 old
PrimeGrid	7,602 +26	21,393 +70	734 +1	109 0	63,108,722 +280,271	04:07:40 old
Project Neuron	409 0	1,681 0	122 0	42 0	2,290,251 0	07:07:40 old
Proteins@home	11,020 0	19,882 0	721 0	129 0	44,748,917 0	38 days 13:32:39 old
PS3GRID	39 0	64 0	19 0	13 0	4,185,342 +60,750	23:37:40 old
QMC@Home	18,441 +40	38,282 +105	1,220 +1	138 0	348,193,170 +1,676,820	06:32:38 old
RALPH@Home	2,265 0	5,919 +1	363 0	69 0	7,423,862 +4,928	03:22:39 old
Rectilinear Crossing No.	5,974 +13	15,636 0	548 0	100 0	53,246,022 +216,969	01:07:40 old
RieselSieve	5,422 +14	15,491 0	501 +1	100 0	89,346,398 +642,416	03:42:37 old
Rosetta@Home	156,458 0	398,659 0	5,883 0	207 0	2,354,644,393 0	2 days 05:57:27 old
SciLINC	150 0	217 0	35 0	23 0	2,093 0	44 days 02:16:39 old
Seasonal Attribution	5,177 +4	8,489 +30	463 0	85 0	60,471,492 +172,008	03:50:38 old
SETI@Home	685,191 +426	1,569,106 +1,145	49,952 +9	231 +1	17,929,746,732 +28,752,230	1 day 00:46:51 old
SETI@Home Beta	4,369 +5	13,698 +125	616 +3	87 0	46,462,646 +177,275	1 day 00:57:04 old
SHA-1 Collision Search Graz	1,802 +23	3,532 0	187 +5	71 0	6,840,696 +352,380	06:17:40 old
SIMAP	18.112 +11	47.456 +69	1.315 +3	142 0	152.663.706 +495,977	06:37:38 old

Figure 12: Sample screen shot of statistics for all BOINC projects.

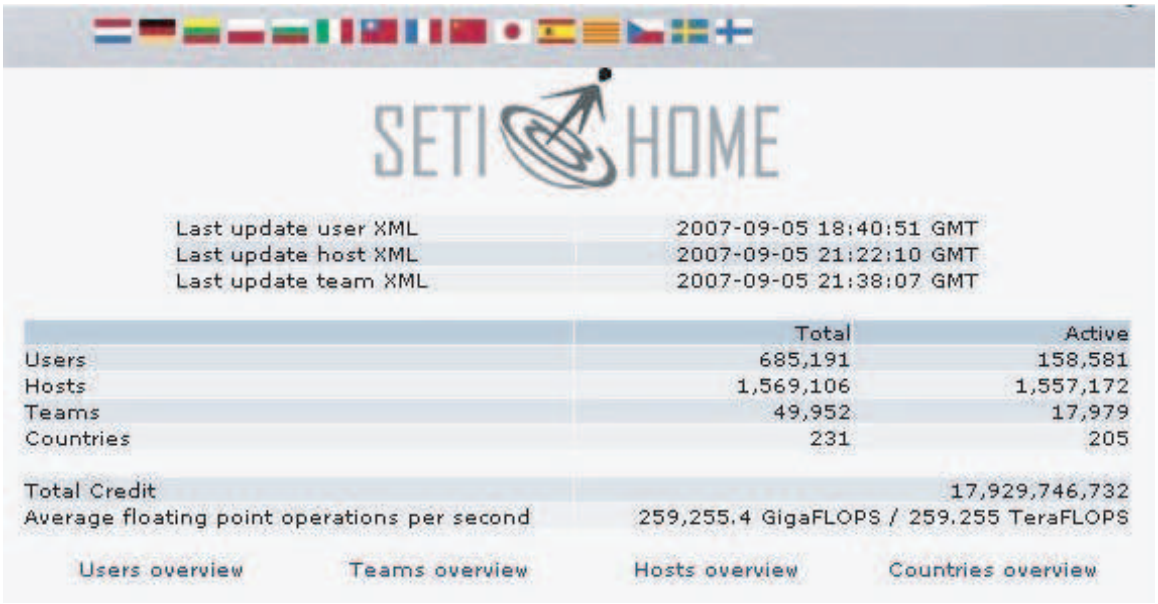


Figure 13: Sample screen shot of statistics for SETI@home.

1-bit complex sampled time series recorded at Arecibo.

2. Client code: runs on the volunteer's machine.

   (a) The application client code, written by the researcher who owns that particular project. In the case of Astropulse, this is the code that performs the dedispersion. Because our code is open source, a few ($< 1\%$) highly computer literate volunteers make minor adjustments to this code in order to make it run faster on their machines. The validator (see below) ensures that the volunteers' modifications do not alter the scientific results.

   (b) The BOINC client code, written by the BOINC team at Berkeley. It relays the data to the application client and performs monitor and control functions, such as handling application client crashes.

3. Server code: runs on the researcher's machines.

   (a) The validator, written by the researcher, checks whether a given workunit produces the same results when sent to different volunteers. This means that volunteers can construct their own application clients without corrupting our data if their results are erroneous. This precaution also handles the hypothetical possibility that a volunteer would modify our client code for malicious reasons.

   Care must be taken to understand what the validator does and does not permit. If volunteers alter their pulses' DMs, pulse areas, detection times, or other scientific parameters, the validator will notice and complain. However, the validator does not check bookkeeping information such as the client's version id number, which is not relevant to our scientific results.

   (b) The assimilator, written by the researcher, records scientific results in a science database.

   (c) BOINC backend programs schedule workunits to be sent to different volunteers, run the validator and the assimilator, and record bookkeeping information in a separate BOINC database.

In the case of Astropulse, preliminary versions of the application client, validator, and assimilator existed prior to my joining the project. This code was contributed by previous members of the SETI@home research group, including Nopparat Pantsaena, Ryohi Takahashi, Christopher Day, Karl Chen, Paul Demorest, and Eric Heien, all of whom had left the group by the time I joined. I was responsible for causing these programs to run together for the first time by completing the client, and then augmenting all programs as necessary. A copy of the final version of the application client can be found in Appendix B.

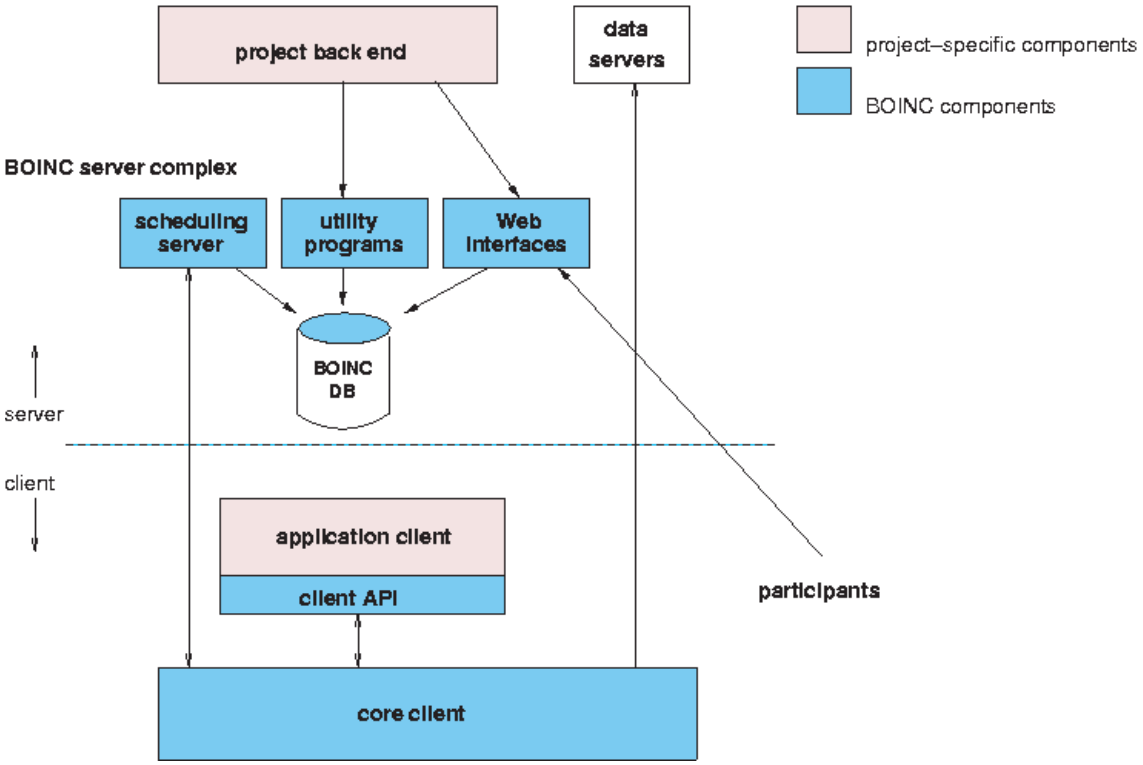


Figure 14: BOINC infrastructure. "Project back end" refers to the validator and assimilator. The scheduling server, utility programs, and web interfaces are the BOINC backend.

5 RFI mitigation

Radio frequency interference (RFI) from terrestrial sources must be eliminated from the data in order to select for signals from astrophysical sources. RFI detections overwhelm detections from pulsars, black holes, or other astrophysical sources. Noise is another source of non-astrophysical pulses; the statistics of the noise are described in Section 3.4.1. The current section describes the methods by which I have classified pulses in the database, deciding whether they might correspond to RFI or noise. Pulses are not deleted from the database, they are merely marked as RFI and can be ignored during any later analysis.

The dominant RFI sources at Arecibo Observatory are nearby radars, which emit one of (at least) 6 repeating patterns. Several of these radars produce chirped pulses that are detectable by Astropulse: (as usual, a sample is 0.4 μs)

1. The Federal Aviation Administration (FAA) radar: a repeating signal, consisting of 5 consecutive interpulse intervals of different lengths: 6,581, 7,052, 6,864, 6,487, and 8,274 samples, in that order. The signal is outside our band, at 1,330 MHz and 1,350 MHz. However, we can still detect the FAA radar when it saturates the receiver or IF electronics. According to Phil Perillat, "the transmitter is located east of San Juan. The radar is used for traffic control around Puerto Rico (it is not the radar used for landing the planes.)" [7] This radar is very strong every 12 seconds (between 11.88 s and 12.01 s, with an average of 11.95 s.) This period corresponds to the rate at which the radar spins around in azimuth; it points toward Arecibo every 12 seconds. The signature of the radar would be found throughout almost all parts of our data if we did not mitigate it. The FAA and aerostat radars (described next) result from the saturation of the receiver or IF electronics, not necessarily from the detection of a pulse. So they can take on the appearance of a DC signal, in which all of our data bits are set to a single value (1 or 0) for some period of time. They can also look like a wideband dispersed signal.

2. The aerostat radar: a repeating signal, consisting of 7 consecutive interpulse intervals of different lengths: 8,759, 7,688, 7,021, 7,260, 8,224, 9,189, and 9,428 samples, in that order. This is "a tethered balloon radar that flies above Lajas Puerto, and is used for drug interdiction." [8] This radar transmits 10% to 50% of the time.

3. Four single period signals: 6998, 6900, 6782, and 7044 samples. These radars are usually not detectable in our data.

5.1 RFI mitigation methods

I rejected RFI using several methods:

[7] http://www.naic.edu/p̃hil/rfi/rdr/faa/faardr.html
[8] http://www.naic.edu/p̃hil/rfi/rdr/aerostat/aerostat.html

5.1.1 Arecibo's high pass filter

Arecibo can turn on a high pass filter in the receiver that will reject the FAA radar's band from the data. The filter removes signals at the FAA radar's frequencies and below, but permits signals in our 1420 MHz band. However, SETI@home / Astropulse is a commensal (piggyback) survey, and not all users of the ALFA receiver want the high pass filter to be turned on. (Some of them wish to observe sources at the radar's frequencies.) Currently, the filter is usually turned off. So I must rely on other methods.

5.1.2 Hardware blanker

Arecibo Observatory provides us with a blanking signal (which we will call the "hardware" blanking signal), a single bit which is turned on when the FAA radar is transmitting a pulse. Since ALFA has 7 beams and 2 polarizations, each with a real and complex bit, we store our data in 4 bytes per 0.4 μs sample. That's 28 bits for the data, and 4 are left over. This means we have some extra space to record the hardware blanking signal. When the Astropulse splitter turns a tape file into a workunit, it detects this signal and blanks the surrounding data as it creates the workunit. So the "hardware" blanker has two components:

1. The hardware component at Arecibo, which adds a blanking bit to our tape files.
2. The software component in our splitter, which detects the bit and blanks the appropriate data.

It is critical that I blank the data with noise that has the same frequency profile as the clean data. If I instead blank the data with white noise (bits set randomly to 1 and 0), I'll get a plot like Figure 15.

This is a "waterfall plot" of time versus frequency, obtained by performing 64-point Fourier transform across a block of data from a workunit, dated 1/8/09.

You can see two effects here. First, there's a series of pulses that weren't caught by the hardware blanker. (The black horizontal bars, centered vertically.) This first problem is unavoidable at this stage, if the hardware blanker fails to notice the radar. But the second problem could have been avoided: notice that the blanker has altered the frequency envelope. The highest and lowest frequency has much lower power – this is the natural envelope of our filter. But at regular intervals, the envelope becomes flat. This part of the data has been blanked, and the inserted noise had a flat spectrum. If I were to use this method, the blanked regions would be detected by the client as chirped pulses, because the client would see a high (and low) frequency, followed by a region with no high or low frequency, followed by a low (and high) frequency. The sequence of high - middle - low is a chirped pulse.

Instead, I need to blank the data using noise whose envelope matches our filter. To do this, I sample the unblanked data, take its Fourier transform, and construct an appropriate stream of shaped noise.

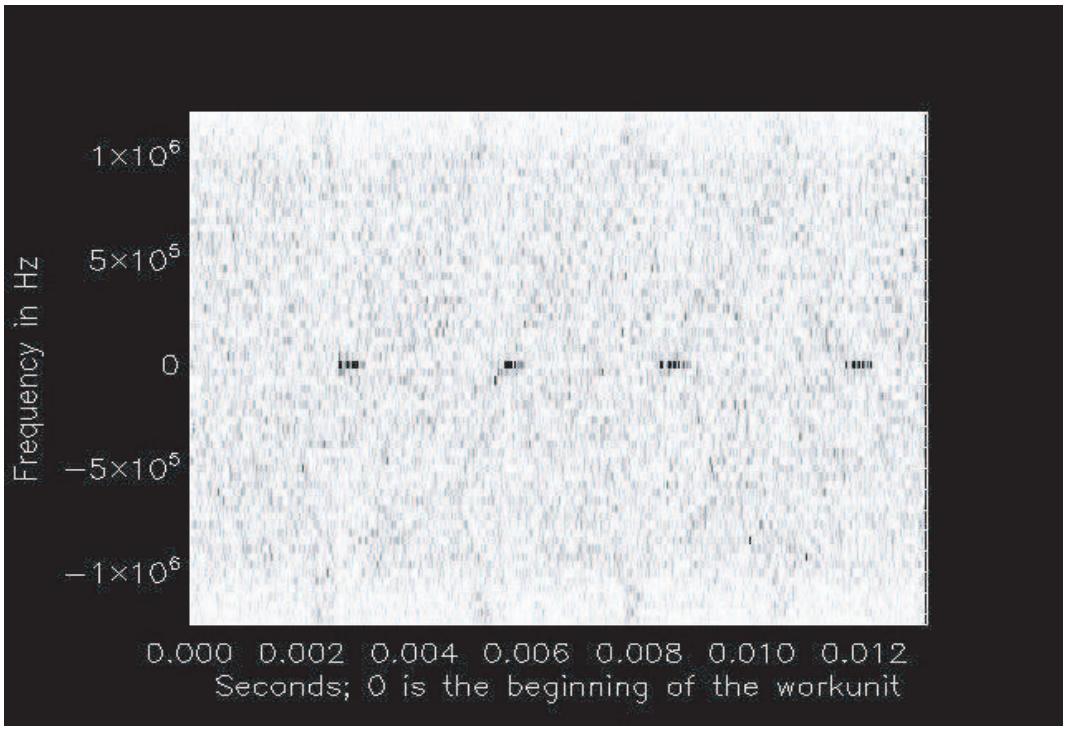


Figure 15: Blanking attempt. The image depicts a time vs. frequency "waterfall plot." Frequency is offset so that the 0 Hz point on the y-axis corrsponds to 1,420 GHz. The DC signal comes from radar, and the lighter periodicity (along the top and bottom edge of the plot) is an artifact of the hardware blanking.

Unfortunately, the hardware blanker is imperfect. First, we believe it doesn't mark every FAA radar pulse. Sometimes the radar's phase changes, and it takes some time for the hardware blanker to catch up. At other times, a single radar pulse may arrive that is out of sync with the other pulses. Second, the hardware blanker only searches for the FAA radar, not for other radar. So we have written our own software blanker, which processes the data downstream from the hardware blanker. The software blanker handles both the FAA radar and the aerostat radar.

5.1.3 Software blanker

The software blanker runs as part of our splitter program, examining the data for the repeating patterns that signify radar. It looks specifically for the FAA and aerostat radar. The software blanker was programmed by Luke Kelley and Matt Lebofsky.

Like the hardware blanker, the software blanker has two components:

1. A software component at our lab in Berkeley, which adds a blanking bit to our tape files. This function is performed by a suite of programs. One program identifies the radar and generates a "blanking instruction file." Another program takes the raw data and the instruction file, and sets the new software blanking bit accordingly.
2. The software component in our splitter, which detects the bit and blanks the appropriate data.

To find the aerostat radar, component 1 looks at samples in groups of 10. We would like to know whether the bits are predominantly 1 or 0. (That's 280 bits, counting all 28 bits for each sample – 2 polarizations, 7 beams, and both real and imaginary bits.) At maximum strength, the radar will produce long strings of bits that are all set to 1, regardless of whether they represent real or imaginary data. At lesser strengths, the radar produces less skewed sets of samples, with a "ring down" oscillation between 1 and 0 bits. Nevertheless, these samples at lesser strengths can provide an important indication of radar. We fold the data over 25 seconds at the known radar period, and threshold the resulting amplitudes at 25% above the mean. Actually, we fold at around 200 trial periods, each varying slightly from the average radar period. This is necessary because the radar's period can drift slightly.

Once a radar signal has been detected, we blank the areas where we detect the radar signal, and some number of samples before and after these areas.

5.1.4 Software blanker: previous attempts

We made several attempts at creating a software blanker before settling on our current program. Our first software blanker looked for all types of radar, but assumed that only one radar was hitting us at any time. This turned out to be an erroneous assumption. We

looked for regions of samples set to 1's, and determined the number of samples between such regions. We then attempted to find the implied sequence of interpulse period(s), and deduce the radar pattern. However, this computation can be somewhat complicated, and becomes unmanageable when several radar signatures overlap in a given data set.

Our next attempt was to cross correlate the data stream with a simulated FAA and aerostat radar sequence. (But not both simultaneously.) This involves using some FFT's to determine how well the data is matched by a sequence of pulses that simulate the radar. Effectively, we were taking the dot product of our data with the radar-like sequence, viewing the sample times as elements of a vector. By displacing the sequences by a varying amount, one finds that the dot product is largest when the phases align. We tried radar sequences in the shape of square waves, gaussians, and sawtooths, and found little difference between them.

However, it turned out that the folding method was even faster than the cross correlation method. The cross correlation takes time $O(N^2)$ when it's done the "obvious" way, and time $O(N \log N)$ when done the "fast" way, where N is the length of the data stream. But the folding algorithm takes time $O(N)$.

5.1.5 Client blanker

As discussed above, one characteristic of the radar noise is that it's strong enough to saturate our electronics, producing a long string of identical samples. While this signal is not always detected by the Astropulse client (as it is not dispersed), it tells us that the radar is active at that time. Therefore, I blank all data within $400,000$ samples (0.16 s) of the detected event. This method differs from the software blanker in that I consider individual RFI events, rather than folding several events together. This enables detection of RFI with unknown periods. The client blanker is located in the Astropulse client, and proceeds by performing a Fourier transform of the data, and examining the power in the central bin (the DC component.)

5.1.6 Fraction blanked restriction

For each workunit, I record the fraction of the data that I blanked using the client blanker. I remove workunits entirely if too much RFI was present, since the presence of too much RFI in one region of the workunit may indicate a smaller amount of RFI in other regions.

5.1.7 DM repetition

If I see a signal at the same DM repeatedly at different parts of the sky, I conclude that it came from a terrestrial source and reject these signals as RFI. There's no reason that the same DM should have been observed from several different directions in quick succession. This test, like the multi-beam and multi-polarization tests described below, is implemented

by a program that examines our database of detected pulses, putting them in time order and searching for the appropriate pattern.

5.1.8 Multiple simultaneous beams

Since our beams are separated by several arcminutes, an astrophysical source (or any relatively weak source) should not appear in multiple beams simultaneously. However, a very strong source, for instance a terrestrial source, might appear in the beams' sidelobes. The source's radio waves might arrive at the telescope by scattering from nearby terrain, or by bouncing off the telescope support structure. In this case, the waves might appear in multiple beams. Since the telescope never points at or below the horizon, such sources would appear only in the sidelobes and never in the main lobe.

Therefore, we could rule out some RFI by ignoring pulses that appear in multiple beams simultaneously. Unfortunately, experiments show that a few real, astrophysical signals appear in multiple beams simultaneously. This may happen because the main lobes intersect slightly, albeit at greatly diminished sensitivity, or because strong astrophysical signals could be detected in sidelobes. So such a method is imperfect at best.

5.1.9 Two simultaneous polarizations

A signal from an unpolarized astrophysical source will appear in both polarizations simultaneously (unless it is only marginally detectable.) RFI might also behave this way, but noise will not. Therefore, we can reject a great deal of noise by requiring detections in two simultaneous polarizations. Unfortunately, highly polarized astrophysical signals may also be rejected, especially if the signal's axis of polarization lines up with the telescope's axis of polarization. This drawback is balanced by the extraordinary efficacy of the polarization test as a noise rejection technique.

5.1.10 Frequency profile

We are looking for broadband pulses with a short intrinsic timescale. Thus, the pulse should have roughly the same mean power at all frequencies. We perform a chi square test to determine whether the mean power is the same everywhere. However, the chi square distribution is not a perfect description of the power vs. frequency distribution unless the power has a Gaussian distribution at each frequency. In fact, it has an exponential distribution.

The frequency profile test calculates a "log_prob" statistic, which is the natural log of the estimated probability that this frequency profile would occur by chance. However, if the chi square is inaccurate, so is the log_prob. Nevertheless, the log_prob should decrease (and is negative) as the power becomes concentrated at particular frequencies. Although the log_prob has uncertain meaning in the absolute sense, its relative value is meaningful.

5.2 Figure of merit

We can assign a figure of merit to each RFI rejection algorithm, or to all algorithms together. The figure of merit is defined as:

(% of astrophysical pulses passing) / (% of all pulses passing).

If the figure of merit is equal to 1, the algorithm does not change the % of pulses that are astrophysical. In other words, we could have achieved the same result by throwing out a random collection of our pulses. Therefore, an algorithm cannot be useful unless its figure or merit is greater than 1.

This definition of the figure of merit is not the only one imaginable. For example, suppose we have 1,000 pulses, of which 100 are astrophysical, and two algorithms. The first algorithm cuts the list down to 10 pulses, of which 9 are astrophysical. It has a figure of merit equal to 9. The second algorithm instead cuts the list down to 100 pulses, of which 80 are astrophysical. It has a figure of merit equal to 8. Then one might prefer the latter algorithm, on the grounds that it yields more data to work with. (Even though a smaller % of that data is good.)

Nevertheless, this figure of merit is reasonable, and we calculate its value for each of our RFI rejection algorithms in the following sections.

5.2.1 Fraction blanked restriction: figure of merit

It turns out that we can obtain the best figure of merit by passing only those workunits for which the fraction blanked (by the client blanker) is $< 20\%$. Note that the client blanker has already removed a portion of each workunit. Here, we do not consider the figure of merit resulting from the operation of the client blanker itself. Rather, we are throwing out workunits for which a large portion has already been removed. As of December 2009, the figure of merit statistics are as given in Table 2. Instead of simulating astrophysical pulses, I have counted the space available for such pulses in all workunits. Only unblanked space may contain astrophysical pulses (or any pulses that originate outside the telescope), and I am assuming that the likelihood for an astrophysical pulse to appear in a workunit is proportional to the amount of unblanked space in that workunit.

5.2.2 DM repetition: figure of merit

To simulate the fraction of astrophysical pulses that would be accepted by the DM repetition algorithm, I performed a Monte Carlo study, generating a list of 37,572 pulses at random times. The times were determined by considering the start times of actual workunits, then determining a random time within that workunit. The random-time test pulses were also given a random dispersion measure, beam, polarization and scale (co-add). These values were distributed uniformly over the range of allowed values. I compared the random-time test pulses with the list of all detected pulses stored in our database. Using the random

dispersion measures, I counted the number of detected pulses with the same dispersion measure preceding and following the test pulses. The test pulses were accepted or rejected using the same criteria as the DM repetition RFI rejection method.

Monte Carlo statistics for simulated astrophysical pulses, after 3 passes through 12,524 workunits, generating 37,572 test pulses, are listed in Table 2.

5.2.3 Multiple simultaneous beams: figure of merit

To simulate the fraction of astrophysical pulses that would be accepted by the "simultaneous beams" algorithm, I used the same Monte Carlo study that I performed for DM repetition. The test pulses were accepted or rejected using the criteria from the "simultaneous beams" RFI rejection method.

Monte Carlo statistics, for the same pulses as in Section 5.2.2, are listed in Table 2.

5.2.4 Two simultaneous polarizations: figure of merit

If all detected astrophysical pulses were completely unpolarized, or were above threshold in both polarizations, then all of them would pass the "simultaneous polarizations" test. However, even if the astrophysical component of the pulse is unpolarized, the noise component may not be. Thus, pulses near threshold may be detectable in only one polarization.

To simulate the fraction of unpolarized astrophysical pulses that would be accepted by this test, I generated pulse area values with a cumulative distribution $c(s) \propto s^{-3/2}$, or a probability density function $h(s) \propto s^{-5/2}$, where s, drawn from the random variable S, is the pulse area. To generate this distribution, we take the $-2/3$ power of a uniform distribution. That is, if X is uniform with distribution $f(x)$, and $S = s(X)$ is the pulse area, then:

$$\begin{aligned} h(s) &\propto s^{-5/2} && (85)\\ h(s(x))ds/dx &= f(x) = \text{constant} && (86)\\ ds/dx &\propto s^{5/2} && (87)\\ s^{-5/2}ds &\propto dx && (88)\\ s^{-3/2} &\propto x && (89)\\ s &\propto x^{-2/3} && (90) \end{aligned}$$

The reason for the $s^{-3/2}$ cumulative distribution is that if we assume a standard candle source (same luminosity vs. time for all sources), then the sources at distance r have flux at Earth proportional to $\frac{1}{r^2}$. The number of sources within distance r (hence with flux greater than $S \propto \frac{1}{r^2}$), goes like $r^3 \propto S^{-3/2}$.

After determining the test pulse's area, I generate two mini workunit files that contain the pulse. Each file combines the pulse with noise randomly, so that different noise is generated in the two mini workunits. Then, I dedisperse the two files and find the noise-modified pulse areas. The pulse passes the "simultaneous polarization" test if it is above the detection threshold in both polarizations. If it is only above threshold in one polarization, it fails the test. And if it is below threshold in both polarizations, it would not be detected at all, so it does not pass or fail.

Note that for a given power threshold at a particular pulse scale (co-add), there is a unique probability density function (pdf) with $h(s) \propto s^{-5/2}$, so there is no ambiguity about normalization. If more astrophysical sources are present, the total number of sources detected will increase, but the pdf will not change.

After 1,000 pairs of test pulses, the figure of merit statistics are given in Table 2. The x statistic is not the number of pulses generated (2,000), but the number detected; some pulses were below threshold. In the table, the x statistic counts pairs of corresponding pulses as two, whereas the y statistic counts each pair as a single pulse. The z_2 statistic was arrived at in a similar manner.

5.2.5 Frequency profile: figure of merit

Using the same mini workunits generated for the polarization test, I determine whether the pulse would pass the frequency profile test. The pulse passes the frequency profile test if its spectrum is flat. Again, the pdf of the pulse power is unique, given the thresholds for each scale, therefore there is no ambiguity as to the pulse powers we should use.

After a Monte Carlo using threshold log_prob > -1, and after 1,000 pairs of test pulses, the figure of merit statistics are given in Table 2. The x statistic is not the number of pulses generated (2,000), but the number detected; some pulses were below threshold.

5.2.6 Overall: figure of merit

There seems to be no reason that the astrophysical pulses' passing fractions, as described above, should be correlated. (Especially if we exclude the multi-beams test, which is probably unreliable.) An astrophysical pulse that passes the DM repetition test is no likelier than any other to pass the multi-pols test, the fraction blanked test, or the frequency profile test.

To see this, one has to consider the tests in pairs, and think about the nature of the tests. In each case, the property measured by one test is entirely unrelated to the property measured by the other. A pulse passes the multi-pols test if it is strong and/or unpolarized, and it fails the DM repetition test if nearby (noise or RFI) pulses have the same DM as the signal. It passes the fraction blanked test if its workunit has a lot of RFI that overwhelms the receiver or IF electronics, and it passes the frequency profile test if it spectrum is flat.

Table 2: Figures of merit for RFI mitigation algorithms. In the table, x is the number of pulses analyzed in a Monte Carlo test or other simulation, y is the number of those pulses that pass this test, and z is the fraction passing for simulated pulses. x_2 is the number of database pulses analyzed so far, y_2 is the number of database pulses passing, and z_2 is the fraction passing for database pulses. The figure of merit (FoM) is defined as z/z_2. Note that in the row for "fraction blanked", x and y refer to un-blanked space in all workunits (before and after the test is applied), in units of full workunit lengths.

algorithm	x	y	z	x_2	y_2	z_2	FoM
fraction blanked	2,457,187	1,368,632	0.557	256,085	122,627	0.479	1.16
DM repetition	37,572	35,994	0.958	204,994	114,795	0.560	1.71
multi-beams	37,572	37,420	0.996	256,085	220,573	0.861	1.16
multi-pols	1,086	522	0.481			0.0386	12.5
frequency profile	1,075	857	0.797	246,870	149,277	0.605	1.32

So we expect the fraction of astrophysical pulses passing all tests to be: $0.557 \cdot 0.958 \cdot 0.481 \cdot 0.797 = 0.205$, where we have just multiplied the fraction passing from each test above.

On the other hand, the fraction of database pulses passing all tests is: $47/412001 = 0.000114$, as of February 2001. This makes for a figure of merit equal to 1797, substantially larger than the product of the individual figures of merit. This makes sense, because the multi-pols test is designed to catch noise, whereas the other tests are designed to catch RFI. So we might expect each test to be less effective on its own, but more effective in combination with other tests. (For instance, imagine a fictitious data set in which 49% of all signals are noise, 49% are RFI, and 2% are real. If algorithm A removes all noise, and algorithm B removes all RFI, then the two together have a figure of merit of $1/0.02 = 50$, whereas separately they have $1/0.51 \approx 2$.)

6 Testing and verification

I tested Astropulse in several ways. First, I observed the Crab pulsar (J0534+2200) over the course of 2 hours. Astropulse detected giant pulses at the expected dispersion measure, and we found the same pulses using Arecibo's "Mock" spectrometer bank (created by Jeff Mock.) This procedure tested Astropulse's end-to-end data stream, using all components of Astropulse including the ALFA receiver, downconverter, data storage to disk, splitter, BOINC software (client, validator, and assimilator), and retrieval from our informix database.

Second, I determined that the hyperfine hydrogen line (1420.41 MHz) is present in the data. In addition to verifying the integrity of our data stream, this shows that my interpretation of the data bits is correct. For instance, if I had confused the real and imaginary bits, frequencies above 1420 MHz would have been swapped with frequencies below. Since the hydrogen line is in the right place, the data bits must have been interpreted correctly.

6.1 Verification using known pulsars

To examine the Crab pulsar, we used both Astropulse and the "Mock" spectrometer bank[9] to examine the same data stream. The Mock spectrometer was configured as a fast readout spectrometer with 512 channels, 1 ms dump time, 1300 MHz center frequency, and 172 MHz bandwidth. It contains 14 component spectrometers, each of which handles a single (beam, polarization) pair. It belongs to Arecibo Observatory, and can be used by any researcher. It is currently used as the spectrometer for PALFA, a pulsar survey.

We observed intermittently from 11:39 AST to 13:27 AST on June 7, 2009. To examine the Crab pulsar, I dedispersed the signal at ± 56.76 pc cm^{-3}, as well as ± 165.7 pc cm^{-3}.

In a typical segment of data, I saw one giant pulse per 13 seconds, which appeared only at $+57$ pc cm^{-3} as expected. We detected these pulses independently, using the Mock spectrometer, and verified that the two methods found the pulses at precisely the same times. For instance, the 1420 MHz component of the pulse in Figure 16 was detected by the spectrometer on 6/7/2009, at 16:19:42.07 UTC. Astropulse detected a pulse with the same DM at julian date 2454990.180348, which is the same time. (Astropulse's measurement of the DM, which is somewhat less accurate, gives a value ranging from 50.9 pc cm^{-3} to 58.2 pc cm^{-3} with an average of 55.7 pc cm^{-3}. The canonical value of the Crab's DM is 56.8 pc cm^{-3}.)

We will consider the largest signal reported by Astropulse at this DM, over a period of 50 seconds. The signal's apparent strengths at various time scales are given in Table 3.

In the table, a value of $\gamma < -30$ is unlikely to appear by chance, even when a workunit is searched at all DMs. γ is the log of the incomplete γ function, evaluated at the listed pulse area and scale (samples). The incomplete γ function is a cumulative distribution function

[9] `http://www.naic.edu/science/userguide_set.htm`

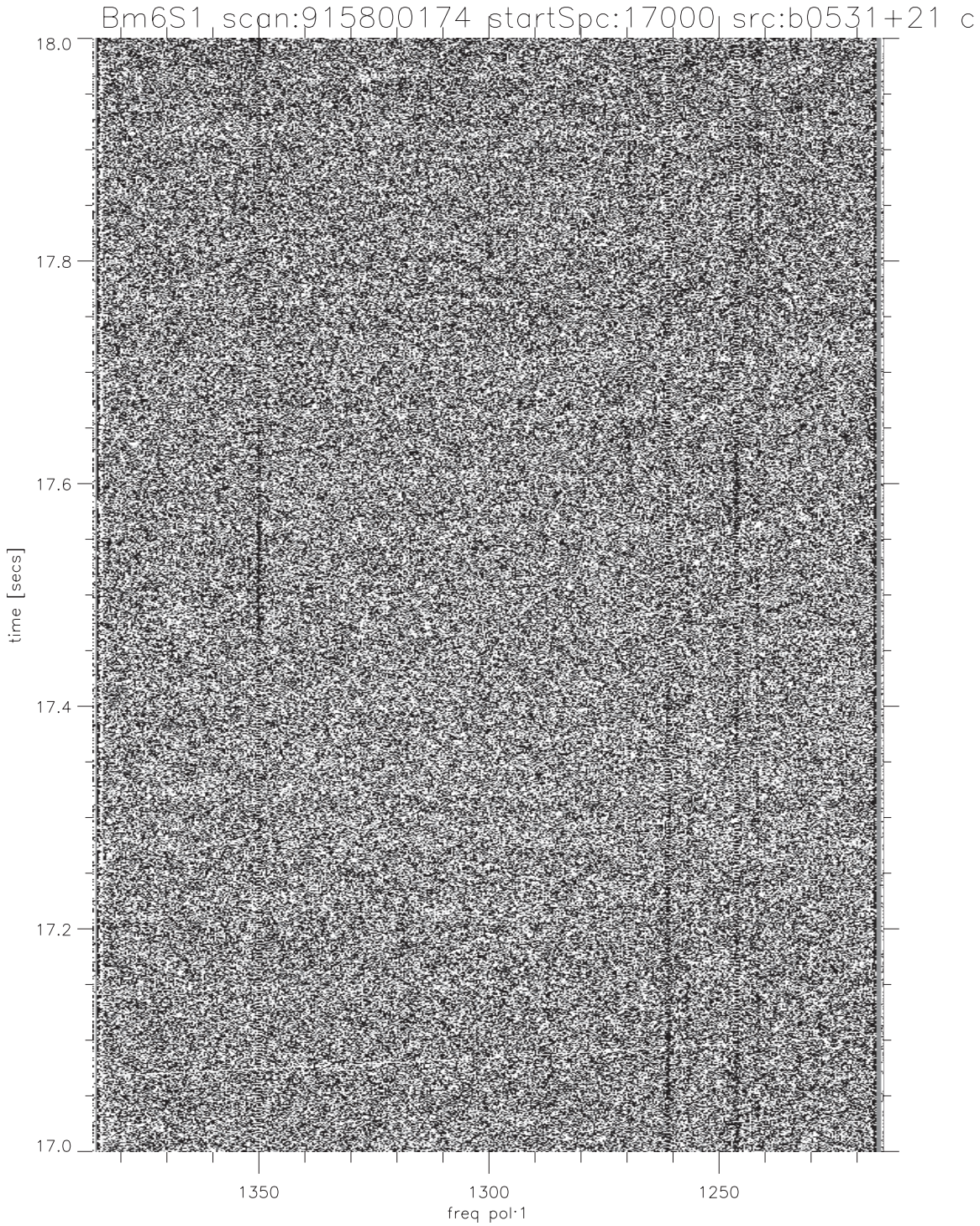


Figure 16: A giant pulse from the Crab, at 17.07s in the figure, recorded by the Mock spectrometer. The x-axis is frequency in MHz, and the y-axis is time in seconds. Notice that the high frequency component of the pulse arrives earlier, as expected.

for noise area (Section 3.4.1), so a very negative value of γ implies a small probability for the pulse to appear by chance.

Table 3: Apparent Crab signal strengths. "samples" refers to the time scale of the pulse in 0.4 μs samples. The intrinsic area is estimated from the detected area by subtracting the average noise present in that many samples. (Recall that our average noise area is 1.9 Jy μs per sample.) γ is the exponent of the incomplete gamma distribution for the noise area.

samples	detected area (Jy μs)	γ	intrinsic area (Jy μs)
1	62.00	-32.6	60.10
2	66.12	-31.2	62.32
4	107.0	-46.9	99.35
8	150.5	-57.1	135.3
16	236.6	-80.0	206.2
32	273.8	-67.9	213.0
64	318.4	-45.5	196.8

The γ function is at its most extreme for a scale of 4, which suggests that the pulse has an intrinsic scale of 4, meaning $2^4 = 16$ bins or 6.4 μs. The pulse appears to have a total area around 209 Jy μs, or 13.1 per sample. However, the Crab nebula increases the observed temperature above its usual value. It contributes to the system temperature with the power law $F_\nu = 955\nu^{-0.27}$ Jy, ν in GHz (Cordes et al., 2004). The received flux density is reduced if the telescope beam is narrower than the nebula. The characteristic diameter of the Crab nebula is 5.5′, whereas the telescope beam is 3.5′, so only 0.40 of the nebula is visible. Therefore $F_{1.4} = 872$ Jy $\cdot$ 0.40 = 348 Jy. This is substantially larger than the dark sky noise, T_0/G; the data are strongly dominated by noise from the nebula. So we should replace T_0/G by this flux. Then the signal has a total of

$$209 \text{ Jy } \mu\text{s} \cdot (348 \cdot \frac{\pi}{2} \text{Jy} \cdot 0.4 \ \mu\text{s}) / (1.9 \text{ Jy } \mu\text{s}) = 24{,}000 \text{ Jy } \mu\text{s} \tag{91}$$

This seems consistent with Popov & Stappers (2007), who found perhaps 50 pulses of size 10 kJy μs or more at scales from 4 μs to 16 μs, over the course of 3.5 hours. Thus, Popov & Stappers detected one such pulse every 250 seconds. Since the Crab pulsar scintillates and changes in brightness over time, we were unable to perform a rigorous quantitative analysis to compare our pulse powers with previous observations.

6.2 Verification using the hydrogen line

A time vs. frequency plot of ALFA data shows the hydrogen line faintly but clearly. The hydrogen line is the dark horizontal (constant frequency) line in the upper half of the Figure 17.

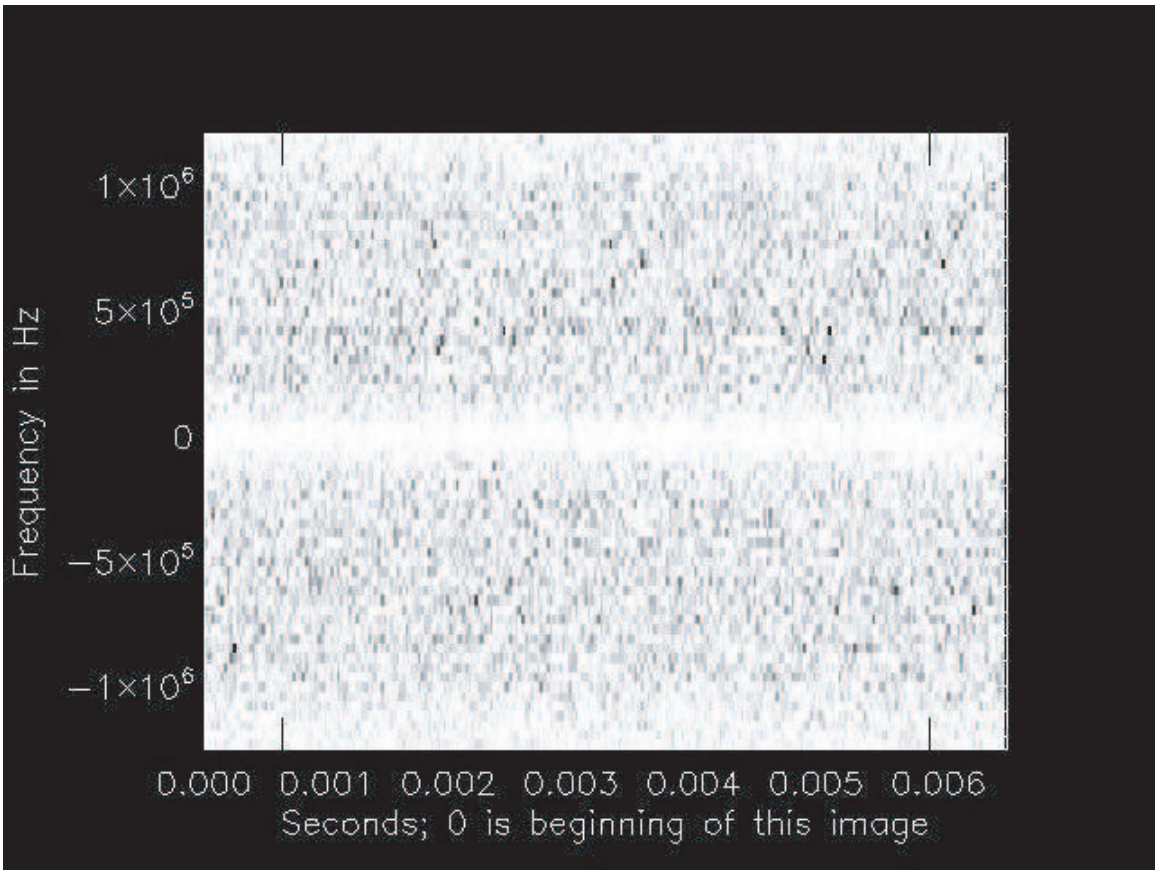


Figure 17: Hyperfine hydrogen line, as seen by ALFA, in a time vs. frequency plot. On the y-axis, the frequency's offset is 1,420 MHz. The hydrogen line can be see as a horizontal line where the pixels in the plot are slightly darker, implying a greater flux density in a narrow bandwidth. The hydrogen line should be at 1420.41 MHz, and the horizontal line in the plot appears in the expected location.

7 Results and interpretation

7.1 Results of RFI mitigation

As discussed in Section 2, we observed for 1,540 hours with each of 7 beams (and 2 linear polarizations), for a total of 21,600 hours of observation time. This resulted in 5,785,125 "workunits," each of which contains 13 seconds of data. This data was analyzed by volunteers using the Astropulse client program, which was programmed to find up to 30 nonrepeating pulses per workunit before halting. (If the client program halted, it would still deliver the 30 pulses to our database, but it wouldn't look for more pulses in that workunit.) We obtained 100,023,171 pulses above threshold, or an average of 17 per workunit. Each pulse that surpassed the threshold pulse area (in Jy μs) was stored in the Astropulse database.

I applied all types of RFI mitigation described in Section 5.1, except for the multibeam correlation method. After I applied these mitigation methods, 330 candidate pulses were left, representing 114 potential sources. (Pulses within each 0.001 day segment, or 86 seconds, were deemed to originate from the same source. Many of these sources produced several "pulses" within a few microseconds of each other, which actually correspond to features of a single pulse.) All RFI mitigation algorithms set flags in the database, marking certain pulses as RFI or noise. Using these flags, it is a simple matter to apply any subset of the algorithms and count how many pulses have passed.

Of the 12,987,887 pulses that were analyzed as of 3/30/2010, 49,112 passed the multipolarization correlation test and 9,768,672 passed the DM repetition test. 10,966 pulses passed both tests.

The frequency profile test was too slow for me to run it on all pulses in our database. In the tests described in Section 5.1, I ran it on as many pulses as possible. However, in order to best examine the whole database, I finally ran it only on the pulses that passed both the multipolarization correlation test and the DM repetition test.

When I did this, of the 10,966 pulses analyzed, 5,946 passed the frequency profile test, 5,443 passed the fraction blanked test (blanking < 0.2 of the workunit), and 6,995 appeared in the recorded data. 1,293 pulses passed all of the tests.

In addition, some pulses were thrown out because they had negative DMs. 419 pulses remained after this test. Note that this number did not result from an initial overall surplus of negative DM pulses; of the 12,987,887 pulses that were analyzed, close to half (6,367,033) had positive DM. Finally, some pulses were removed because they came from our examination of the Crab pulsar. After doing this, I was left with 330 pulses.

The "recorded data test" (that a pulse must appear in the recorded data) requires some further explanation. It was not anticipated in the planning stages of the project, and it is not described in the RFI mitigation section, but its importance became clear as I began to obtain results.

When I store a pulse in the database, I attach a 26.2 ms time series centered at the pulse.

This time series requires 16 kb of raw data from the workunit (65,536 samples.) The raw data is used to perform the frequency profile test. When the frequency profile test is applied, some pulses cannot be found at all, whether by visual inspection of a time vs. frequency plot, or by dedispersion and thresholding. When the RFI mitigation code tries to dedisperse the data at the appropriate DM, no pulse surpasses the threshold. This can occur for a number of known reasons.

1. DFT sensitivity to position: some pulses may or may not be visible, depending on the precise positioning of the DFT that is used to detect them. (Recall that pulse detection involves convolution, which uses DFTs. Each DFT has a finite width, and it has some offset with respect to the pulse. The pulse may be at the center of the DFT, or at the edge.) These sensitive pulses appear only when the DFT's center is chosen at just the right spot. This spot is generally not the same as the bin where the client claims to detect the pulse. In fact, these pulses normally appear only at the edge of a DFT, not at the center. Since the recorded data test attempts to detect a pulse by performing a DFT centered on the pulse, these pulses are invisible to the test. These pulses may be so sensitive to DFT positioning that a displacement of 100 to 200 samples causes them to disappear almost entirely.

   This problem may have a variety of sources, but it is likely that all of them are manifestations of RFI. I have analyzed the cause in one case. In the workunit I examined, a pulse appearing at the left edge of the client's DFT window was an illusory artifact of the dedispersion algorithm; it was caused by a narrowband signal located at the right edge of the DFT window.

   In this DFT window, the time series contained a strong narrowband pulse with a frequency above 1,420 MHz. The pulse started about $\frac{2}{3}$ of the way into the time series of the client's DFT window, and extended to the end of the window. (It may have extended past that, but I was unable to discern this by examining the recorded time series.) Since the time series always has a uniform power per sample (that is, 2), this means that other frequencies in that region had below average power, to compensate for the narrowband pulse's above average power. Now, consider the effect of a dedispersion attempt with a positive DM. This will shift high frequencies to the right (to a later time), and low frequencies to the left. The narrowband pulse shifts to the right, because its frequency is above 1,420 MHz. Now, recall that the narrowband pulse extended to the right edge of the DFT window. Since a Fourier transform is cyclical, the convolution for the dedispersion considers the DFT window to wrap around; anything that goes to the right of the right edge will wrap back around and appear at the left edge. So after dedispersion, a part of the narrowband pulse wraps around and appears at the left edge of the window. But the lower frequencies at the left edge (below 1,420 MHz) have average power. (As opposed to below or above average power.)

   So the total power at the left edge of the DFT window is above average after dedispersion. This can create the illusion of a broadband pulse at the left edge of the FFT window, when in reality the pulse is narrowband, and is located at the right edge of the FFT window. This particular pulse also failed the frequency profile test, because it originated from a narrowband signal.

2. Blanking noise artifact: the pulse appears in a region that was blanked by the client software blanker. Because the pulse appears in a region of artificially generated noise, it represents a fluctuation in that artificial noise. Since the recorded data is taken from

the (unblanked) workunit, the pulse is not recorded. These pulses typically fail the polarization correlation test, since different random noise is (usually) generated for two workunits corresponding to the two polarizations.

3. Bad DM magnitude: the DM was stored incorrectly, as ±30,000, in the database. This is a bug from an older version of the assimilator, affects a small fraction of the data from that era (less than 3%), and these pulses are best ignored. The old code assumed that DMs with $|\mathrm{DM}| > 828.3$ pc cm^{-3} were in error, whereas in fact a few DMs legitimately have slightly higher values.

4. Bad DM sign: the DM was stored incorrectly, due to a sign error. An older version of the code made this mistake. However, this has been fixed, and the fix was retroactively applied to all old pulses. Therefore, this problem should not affect any current data analysis.

I am not always able to distinguish between the above four causes of failure, and my RFI mitigation routine does not attempt to do so. So in all of these cases, the offending pulses are deemed to fail the same test, namely, the recorded data test. I detect the problematic pulses by comparing the pulse area detected from the recorded data (using an IDL routine) to the threshold required by the volunteer's Astropulse client. If the pulse's area would not have been accepted by the client, then the pulse fails the test. The pulse area detected by the IDL routine is usually within 2 Jy μs of the power detected by the Astropulse client.

7.2 Telescope pointings and potential sources

Figure 18 depicts the distribution of telescope observations over the course of our survey, as well as the distribution of the 114 potential sources. Because Astropulse is a commensal survey, we are unable to control the telescope pointing. Figure 19 also plots our potential sources, but it overlays the Leiden / Argentine / Bonn (LAB) Galactic HI survey, described in Kalberla et al. (2005), Hartmann & Burton (1997), Bajaja et al. (2005), and Arnal et al. (2000). The contours of this survey data can be taken to outline the Galactic disk. According to Figures 18 and 19, a large portion of our time was spent pointing at the Galactic disk. If all of our data are RFI or noise, then the potential sources will be distributed in the same way as our telescope pointing, concentrating in places where we observed often. If, on the other hand, we have a significant number of astrophysical pulses from the Galactic disk, then more pulses will come from the disk. If instead we have astrophysical pulses from the halo, from intergalactic space, or from other galaxies, then more pulses will come from outside the disk.

1,024,135 of all telescope pointings in our survey were from inside the disk (within 10° latitude of 0), and 4,497,151 from outside. Of the 330 candidate pulses, 143 were from inside the disk, and 187 from outside. Suppose we arrange these pulses in groups of 86 seconds (0.001 days), and consider that pulses detected at nearby times must have originated from the same source. Under this interpretation, we really have only 114 independent events, of which 61 came from inside the disk, and 53 from outside.

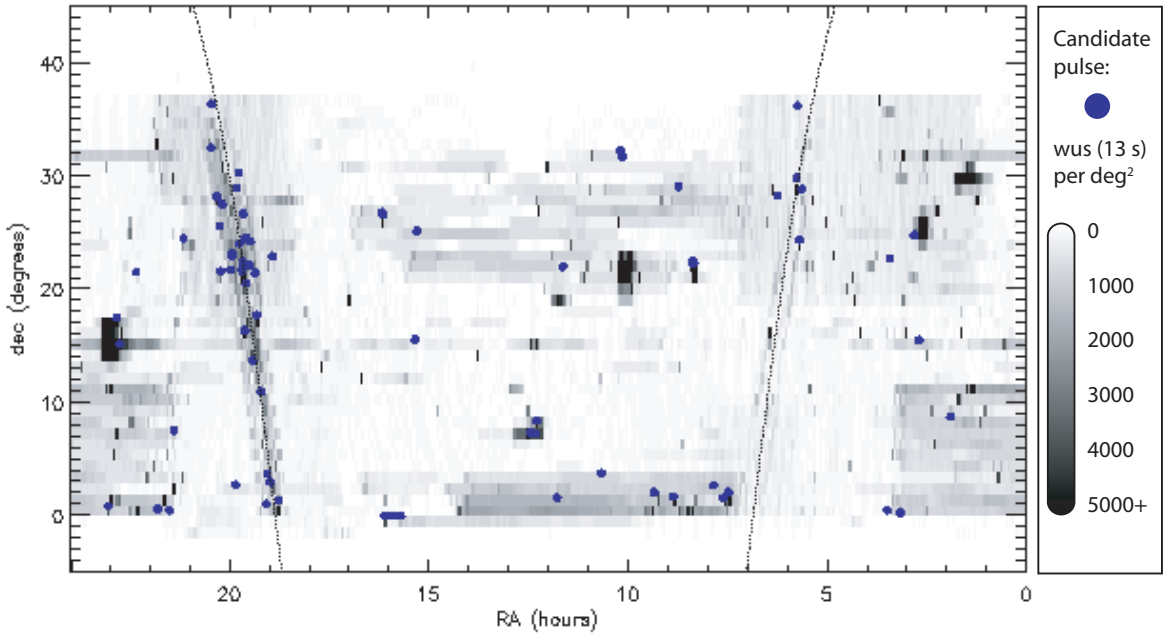


Figure 18: Telescope pointing and potential sources. The greyscale region shows the distribution of telescope pointing, where the black areas were visited most frequently and the white areas least frequently. Each 13 s of observation time (the typical beam transit time, and the duration of a workunit, or "wu") is considered a single visit. The most frequently viewed locations were actually visited 174,000 times per square degree; anything higher than 5,000 visits is still black. The blue dots are the potential sources. The two thin, curved lines represent the 0 latitude lines of the Galactic disk.

It is clear that this sample is extremely unlikely to come from the 1 : 4 ratio parent distribution implied by the telescope pointing statistics. Therefore, it is possible that we have detected some astrophysical signal from the Galactic disk. However, we can do a numerical analysis just to make sure. Our estimate for the fraction of pointings that result in detections is $L = 114/5{,}521{,}286 = 2.06 \cdot 10^{-5}$. Let $a = 61$ be the number of events inside the disk, and $b = 53$ the number of events outside. Let $c = 1{,}024{,}135$ be the number of pointings inside, and $d = 4{,}497{,}151$ the number of pointings outside. Then if a came from a Poisson distribution with mean $c \cdot L = 21.15$, and b came from a Poisson with mean $d \cdot L = 92.85$, these distributions have standard deviations $\sigma = \sqrt{\mu}$, so $\sigma_c = 4.60$ and $\sigma_d = 9.64$. So the samples deviate from their means by $8.67\sigma_c$ and $4.13\sigma_d$. Even taken individually, each of these is much larger than the 1.96σ required for statistical significance.

7.3 Coincidences with catalogs

I have listed the galactic longitude and latitude, RA and dec, dispersion measure, julian date, power, and width for all 114 of the potential sources in Appendix A. What can we say

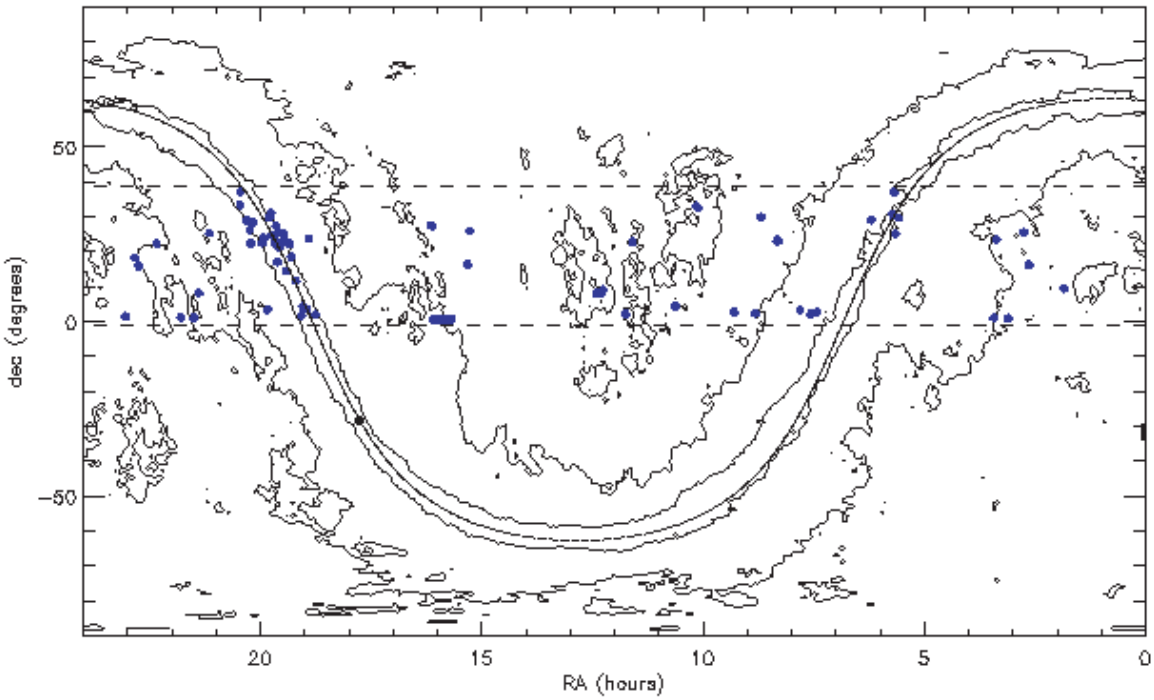


Figure 19: LAB HI contour map with 4 levels, representing the log of the antenna temperature. At each pixel, the data are integrated over velocity. The blue dots are the potential sources, as in Figure 18. The dashed horizontal lines represent the the limits of Arecibo's viewing range.

about the origins of these pulses? The first thing we could try is to compare their RA and dec to lists of known pulsars and other objects. If we do this, we find that 13 of the pulses correspond to known pulsars; 8 are associated with B1933+16 at galactic longitude $\ell = 57.51$ and latitude $b = -0.29$, and 5 are associated with B1937+21 at $\ell = 52.55$, $b = -2.09$. The latter certainly produces strong giant pulses (Sallmen & Backer, 1995).

None of the pulses correspond to 11 known RRATs (McLaughlin et al., 2006) or to 1,451 sources from the Fermi LAT 1-Year Point Source Catalog[10].

Comparison with the 18,810 objects in the ROSAT catalog (Voges et al., 1999) results in 3 coincidences, however this is to be expected given the large number of objects and our $8.15'$ error. The coincidences are J151551.9-41825, J151607.5-385208, and J154912.0-492944.

7.4 Pulses with high dispersion measure

What about the rest of the pulses? One fact to notice is that the pointings inside the Galactic disk often have very high DMs; the average DM inside the disk is 370.4 pc cm^{-3} , and outside

[10]http://fermi.gsfc.nasa.gov/ssc/data/access/lat/1yr_catalog/

the disk is 179.5 pc cm^{-3}. Furthermore, the variance of the DMs is quite high, and some are well above 370.4 pc cm^{-3}. Figure 20 is a scatter plot of the pulses' DM vs. galactic latitude. These figures seem contrary to my hypothesis that the pulses have astrophysical sources, since pulses with DMs above 230.5 pc cm^{-3} from inside the Galaxy would have widths above our limit of 204.8 μs, according to Equation 83.

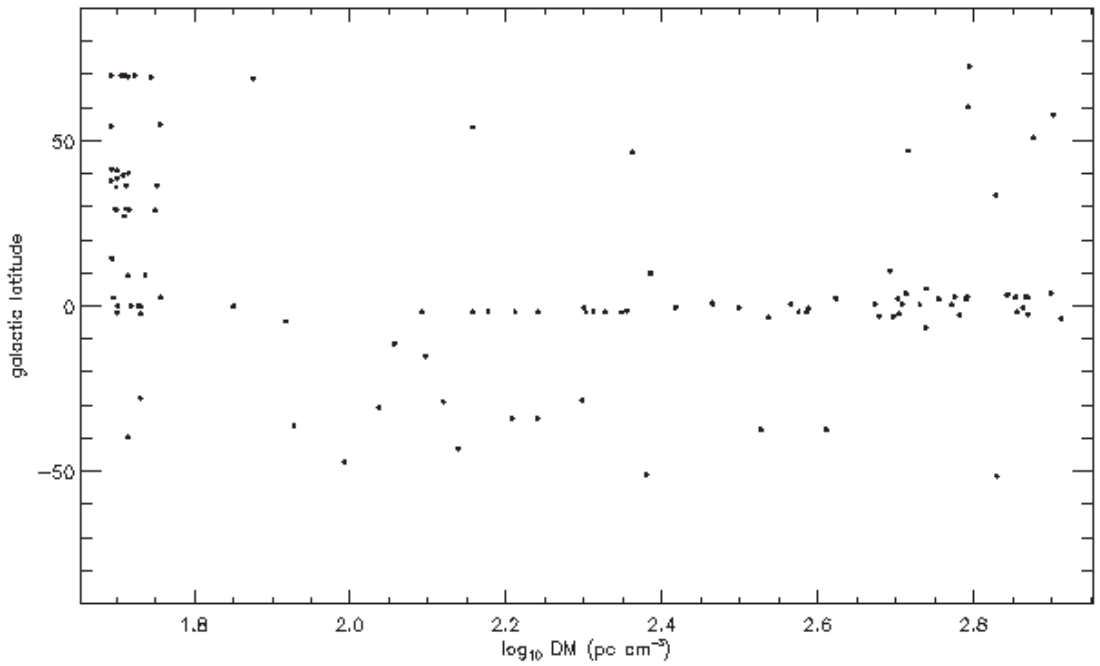


Figure 20: $\log_{10}$ DM (pc cm^{-3}) vs. galactic latitude (b)

It would be possible to detect pulses with higher DMs if they were extremely strong, but we have little chance of detecting pulses with widths substantially longer than our DFT length of 13.1 ms, since power is conserved during a DFT; we re-construct pulses essentially by moving power from one part of the DFT to another. If a pulse had the same RMS amplitude for a 13.1 ms duration, it would be undetectable by our algorithm. According to Equation 83, DMs above 478 pc cm^{-3} would have widths above this limit. Even at this width, a pulse might be detectable since its power is not constant as a function of time in the DFT window. However, such a detection seems extremely unlikely.

Because of the scattering described by Equation 83, the high DM pulses from the Galactic disk are difficult to explain. Nevertheless, I have listed all potential sources in the Appendix, including these. I have only a few theories as to the source of these pulses. One must explain their high DM as well as their correlation with pointings into the Galactic disk. The pulses may be a form of RFI that is correlated with the telescope pointing due to a seasonal variability in radar. Or, perhaps they come from an unusual source such that Equation 83 does not apply. Finally, they might be related to the staring vs. scanning behaviour of the telescope; our partner surveys tend to stare longer at positions in the Galactic disk.

7.5 Estimated energy

We would like to estimate the radio frequency energy, in ergs, released by the sources of our pulses. If we histogram this estimated energy, we might hope to find that the sources have a narrow range of energies. To the same end, we can look for correlations between the pulses' DMs (which is related to a source's distance) and their pulse area in Jy μs. If distant sources have the same intrinsic energy as nearby sources, we would expect the distant sources to appear weaker.

In the most extreme case, the pulses might correspond to a monoenergetic astrophysical source. An exploding primordial black hole might fit this description. Presumably, all black holes evaporate at the same rate, in which case the total energy released by any two explosions is identical. (On the other hand, the flux density at 1,420 MHz might still vary depending on the peak frequency and intrinsic bandwidth of the explosion, which in turn would depend on the magnetic field in the black hole's environment.)

To estimate the source's energy, we start with the DM and peak power (i.e the pulse area or flux density $\cdot$ duration, in Jy μs) of each pulse, which are stored in our database. From the DM, we can estimate the distance of a pulse's source, assuming DM = distance $\cdot$ 0.03 cm^{-3} (Guélin, 1973). From this distance, and an assumption about the pulse's bandwidth, we can guess the energy (in ergs or Joules) of the event that caused the pulse, using Equation 84.

Figure 21 histograms number of pulses vs. log estimated energy (ergs). Figure 22 shows number of pulses vs. log DM (pc cm^{-3}), and Figure 23 shows number of pulses vs. log peak power (in units of Jy μs, see Section 3.5.1.)

In Figures 22 and 23, the actual data from Astropulse is compared to a set of theoretical curves, representing uniformly distributed monoenergetic sources. The green curve would be expected if the sources were distributed in a disk (such as the Galactic disk), and the blue curve would be expected if the sources were distributed in the volume of a sphere.

The peak on the left-hand side of Figure 21, as well as the similar peak on the left-hand side of Figure 22, are probably artifacts of the pulse detection algorithm. The Astropulse client searches through low DMs first, and stops processing any given 13 s workunit after detecting 30 pulses. This ensures that our database is not filled up with thousands of pulses from an RFI-contaminated workunit. So workunits with very large numbers of pulses will not be fully explored, and higher DMs in these workunits will not be examined. Therefore, some workunits will provide only low DM pulses, even though they contain pulses at all DMs. Thus, these low-DM peaks are probably misleading.

Even if we were convinced that one of the peaks in Figure 21 represents a population of pulses, it would be premature to conclude that we have detected a source with constant energy at many DMs. It could happen that the DM and peak power are both restricted to narrow regions, so of course any combination of them (such as the estimated energy) would also be restricted to a narrow region. So instead of relying on Figure 21, let's plot DM vs. peak power and see if we can detect a correlation; see Figure 24. We are hoping that DM and peak power will be negatively correlated, meaning that objects with larger DMs would be farther away, and would therefore have smaller peak powers.

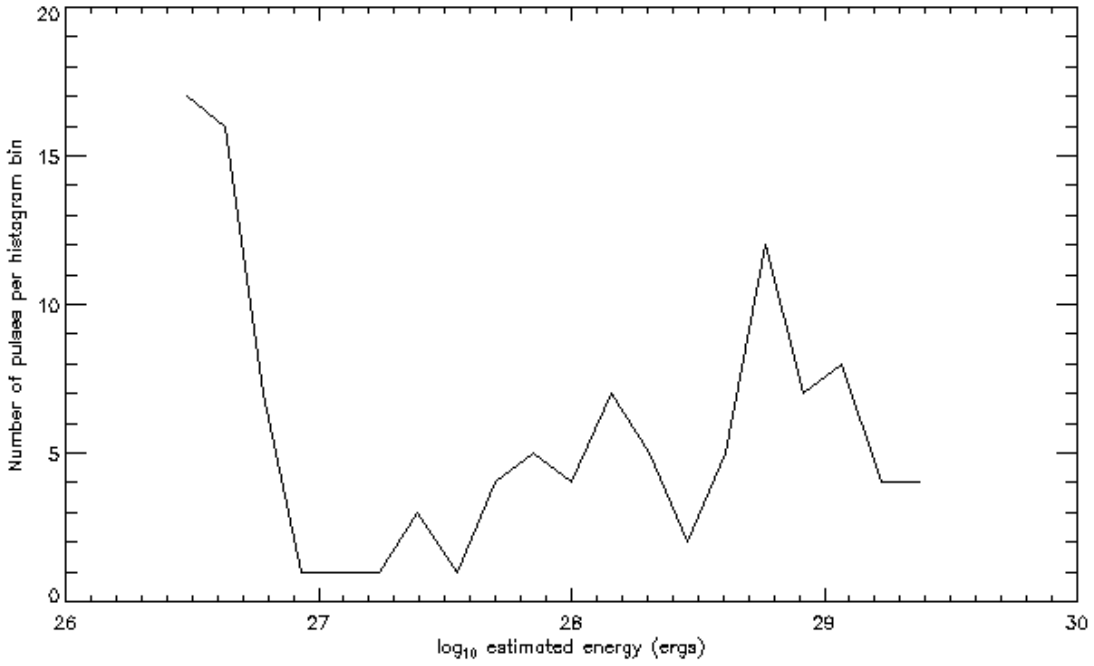


Figure 21: Histogram: number of potential sources vs. $\log_{10}$ estimated energy (ergs)

Using this data, we could ask whether a correlation exists between the two variables (DM and peak power.) To do this, let's perform a Pearson correlation test on the data. When we do this with the data in Figure 24, we obtain a correlation coefficient of 0.447, a fairly high value. Unfortunately, this correlation coefficient has an unexpected sign; the plot shows objects with larger DMs having larger peak powers.

To quantify the strength of the correlation, we perform a permutation test with 10,000 iterations, as follows. At each iteration of the permutation test, we permute the DM values randomly, assigning them to new peak powers. We then perform the Pearson correlation test on the new, randomly selected pairs. If N out of 10,000 of the pairs are more correlated than the original data, we say that the "p-value" is $N/10{,}000$.

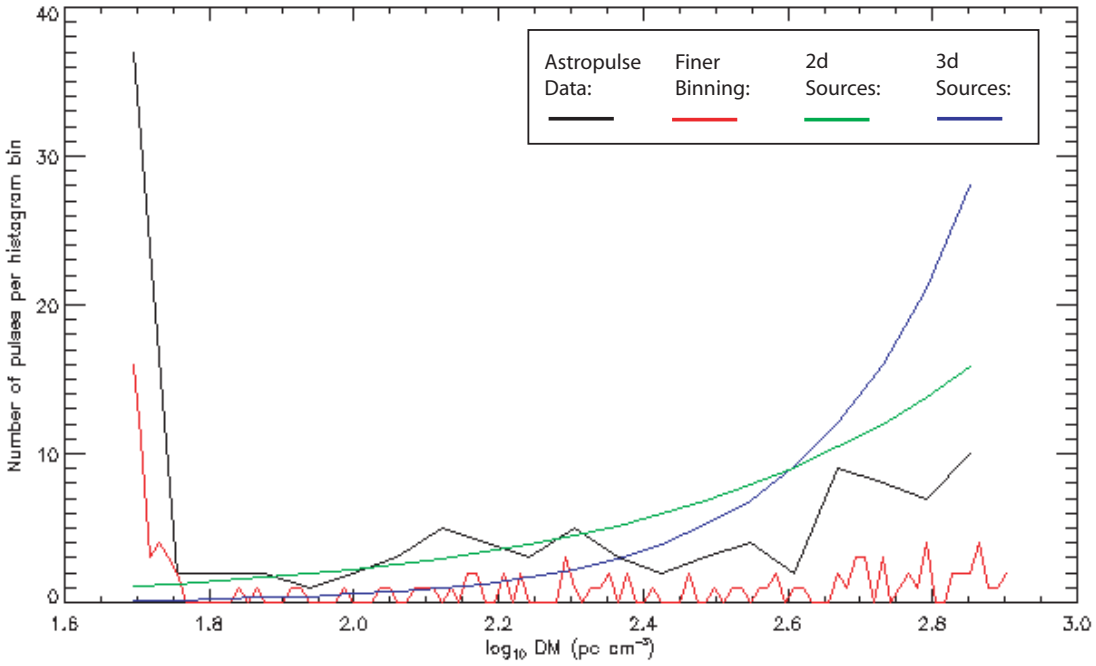


Figure 22: Histogram: number of potential sources vs. $\log_{10}$ DM (pc cm^{-3}). The green curve would be expected for a monoenergetic source distributed uniformly by volume in a disk-shaped (2d) region; and the blue curve corresponds to a sphere-shaped (3d) region. The red curve histograms the data with a finer binning; 100 bins in all instead of 20.

When we perform this test on the data in Figure 24, we find that the p-value for the permutation test is 0 after 10,000 iterations, indicating that no permutations of the data were more correlated than this. This indicates that the correlation is extremely strong.

Perhaps the sign problem is due to the low-DM data ($DM \approx 60$ pc cm^{-3}) that's visible at the lower edge of the plotted region in Figure 24. Suppose we assume that this part of the data is a meaningless artifact of the client algorithm, as discussed above, and we remove it.

For the data in Figure 25, the Pearson correlation coefficient is insignificant at -0.0528, and the p-value is 0.659 after 10,000 permutations.

We might ask how many data points we would have to add, with the desired linear relationship, in order to create a correlation with the desired sign. (And a p-value of 0.05 or smaller.) We would like to add data points with flux $\propto$ distance^{-2}, or peak power $\propto$ DM^{-2}, which says that $\log(\text{DM}) = -\frac{1}{2}\log(\text{peak power}) + \text{constant}$. Experiment shows that 33 points are enough, resulting in a correlation coefficient of -0.19 and a p-value of 0.0486, see Figure 26.

So it is conceivable that we could have detected 33 sources in our data set (or $33/0.2 = 165$ sources prior to RFI mitigation), but this calculation assumes that the additional sources are perfectly monoenergetic.

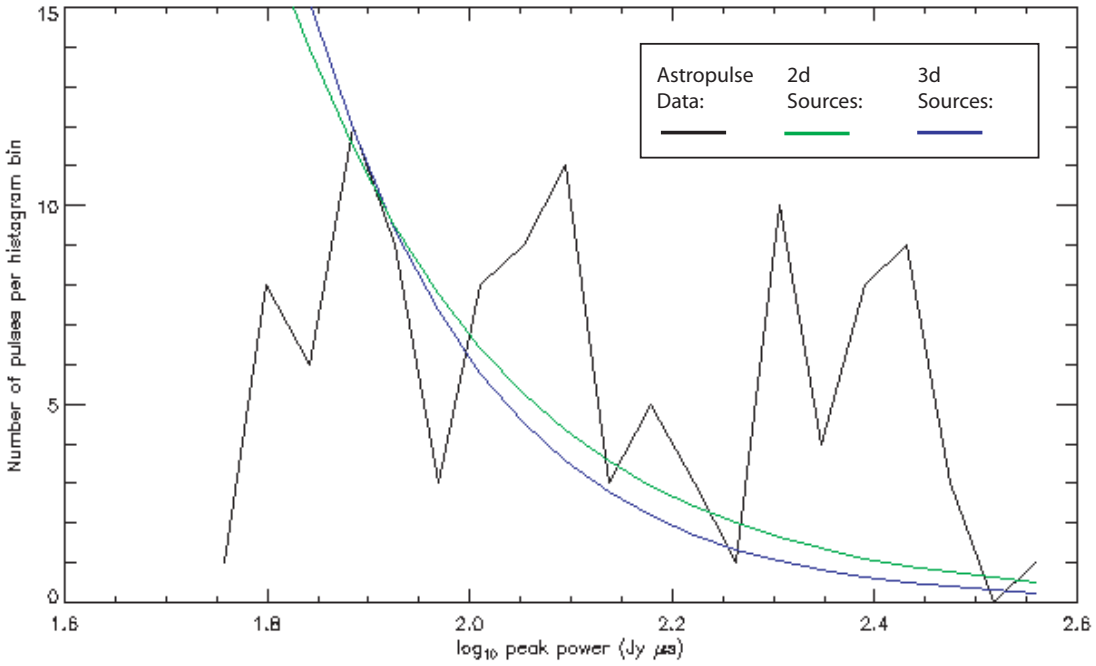


Figure 23: Histogram: number of potential sources vs. $\log_{10}$ peak power (Jy μs). For a discussion of the green and blue curves, see Figure 22.

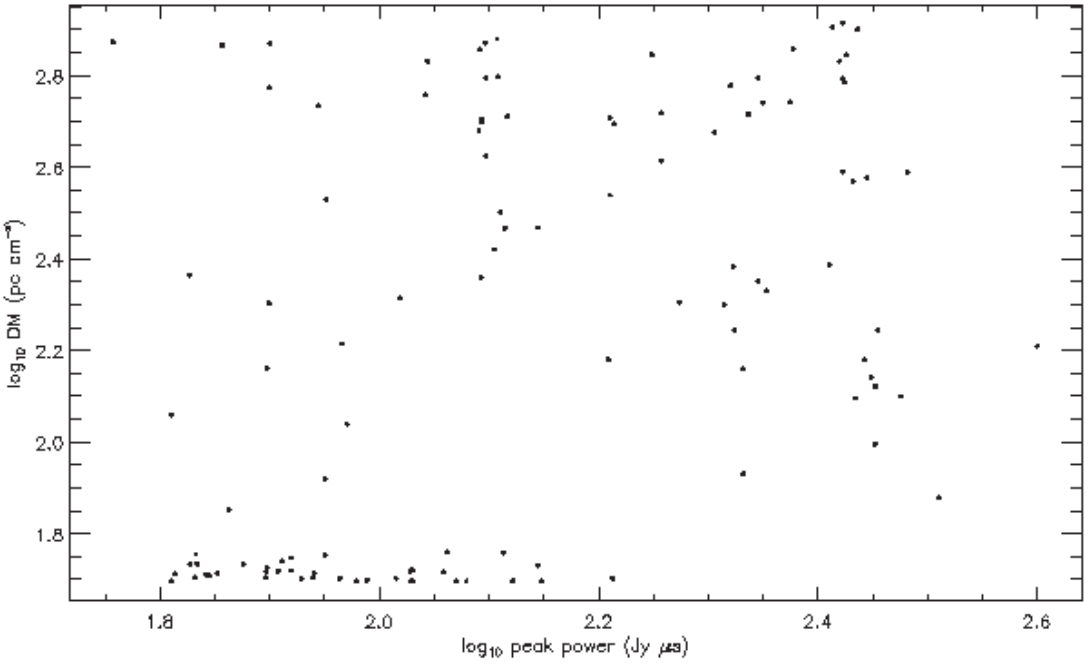


Figure 24: $\log_{10}$ DM (pc cm^{-3}) vs. $\log_{10}$ peak power (Jy μs)

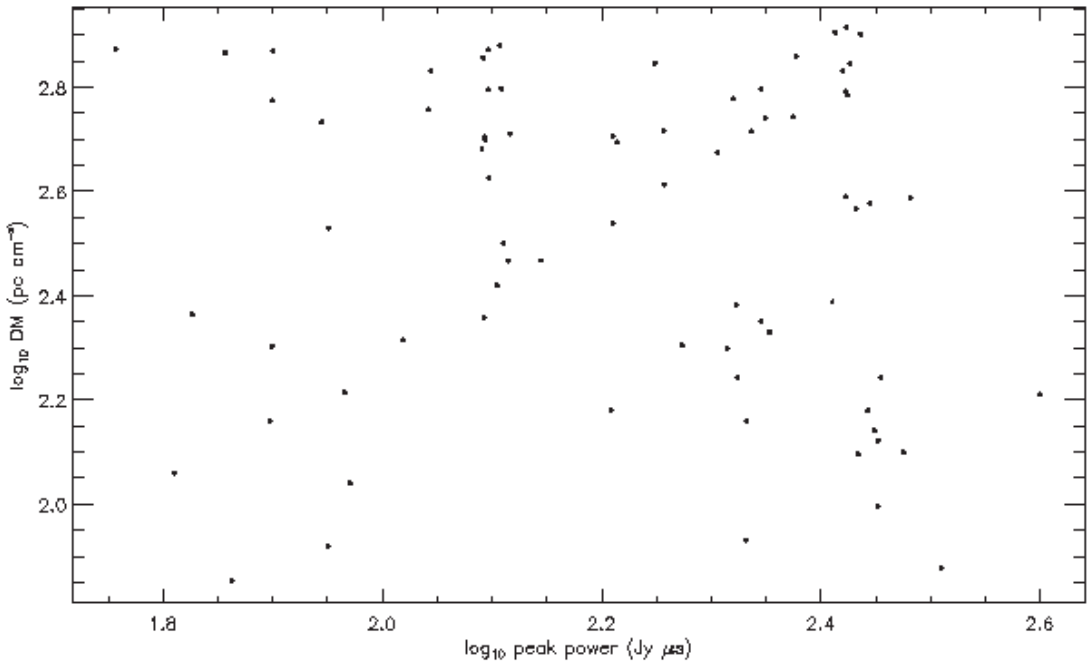


Figure 25: $\log_{10}$ DM (pc cm^{-3}) vs. $\log_{10}$ peak power without low DM potential sources

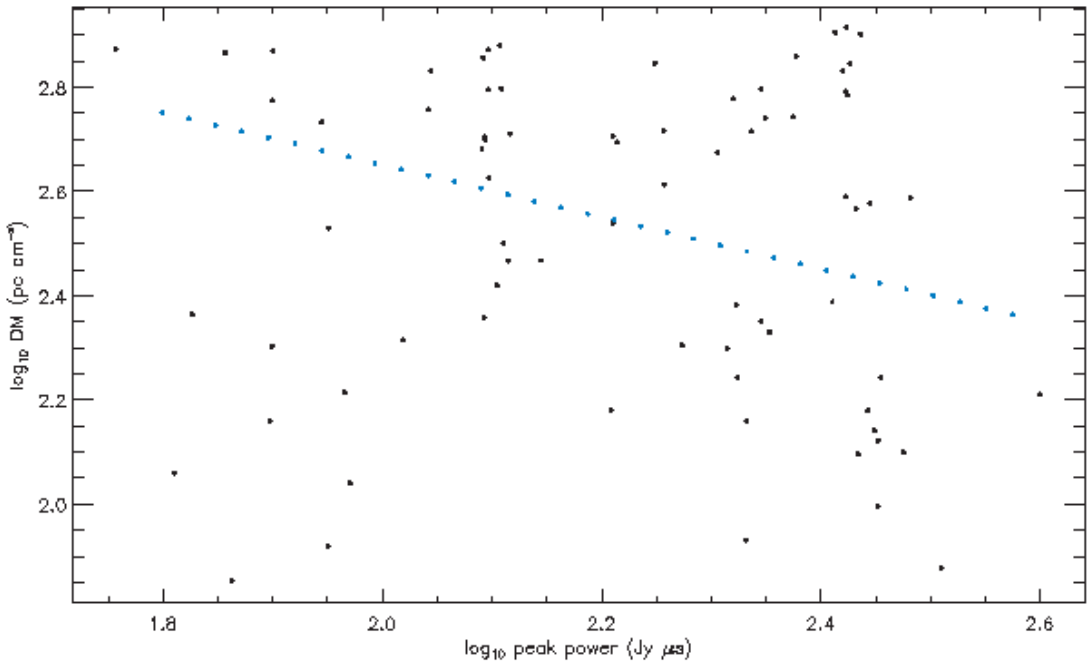


Figure 26: $\log_{10}$ DM (pc cm^{-3}) vs. $\log_{10}$ peak power without low DM potential sources. A set of linear data has been artificially inserted (blue), which is just enough to make the a negative correlation detectable.

7.6 Results: evaporating primordial black holes

Astropulse has detected 114 potential sources so far, but it is unclear whether they represent astrophysical events. How many of them might come from exploding primordial black holes, for instance? We can remove all pulses that repeat, since a black hole can explode only once. Furthermore, I will rule out all pulses with DMs higher than our scattering equation allows. After both of these steps, 78 pulses remain.

As described in Table 1, if Astropulse detected a single event, this would imply a rate of $1.6 \cdot 10^{-12}$ evaporating black hole events $\text{pc}^{-3}\ \text{yr}^{-1}$, under the assumption that the black holes are evenly distributed throughout the space visible to Astropulse, and explode with an energy equivalent to 10^8 kg in a 1 GHz radio band. However, we may have detected as many as 78 events over the course of our observation (if all of them represent astrophysical signals.) Assuming that only 20% of all astrophysical pulses get detected, as discussed in Section 5.2.6, this implies that we may have detected up to 390 events. Therefore, an upper limit of $6.2 \cdot 10^{-10}$ events $\text{pc}^{-3}\ \text{yr}^{-1}$ is more realistic. Of course, the actual event rate might be much lower than this, if many of our detections come from noise or RFI.

These figures also vary dramatically, depending on one's assumptions about the mass of the black holes, their locations (in the Galactic disk, the halo, or intergalactic space), and their radio bandwidth.

Notwithstanding the data in Table 1, it is difficult to know for certain the corresponding figures from other groups' research, since they may not have been handling their data in precisely the same way as Astropulse, or using the same RFI mitigation techniques. Therefore, in one figure below (Figure 29) I have included Astropulse's event rate both with the factor of 390 (assuming all of our detections might be astrophysical) and without it (assuming none of our detections are astrophysical.)

Figure 27 plots sensitivities (in Jy μs) of current and historical surveys against their observation time in hours. A sensitivity represents the maximum possible event rate in light of the survey data. As in Table 1, all sensitivities assume that the pulse is unresolved, i.e. it is narrower than 0.4 μs in the case of Astropulse. Observing time, not coverage area, is the relevant parameter; when searching for one-time events such as a primordial black hole explosion, a survey that stares in a single direction for a long time may be just as good as a survey that scans the whole sky. This is because for non-repeating events, looking in the event's direction at the wrong time is just as bad as looking in the wrong direction entirely.

Figure 28 depicts the same data, but includes the survey's time resolution on the z-axis. One might argue that time resolution is not relevant except insofar as it affects sensitivity (which goes like the square root of the time resolution.) However, improved time resolution may enable superior RFI mitigation techniques, increasing the effectiveness of Astropulse's frequency profile test.

And Figure 29 shows the minimum detectable event rate for 10^8 kg black holes, plotted against the year in which the survey was performed. As mentioned above, this value is highly dependent on many assumptions. The data point in blue shows Astropulse's event rate, taking into account our RFI excision.

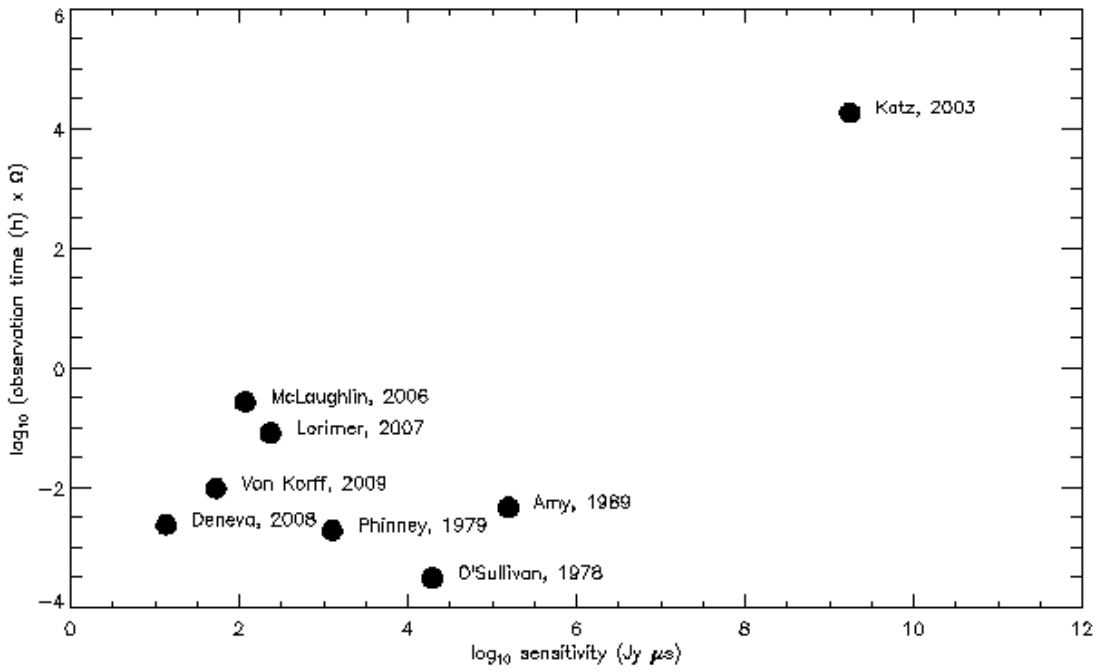


Figure 27: $\log_{10}$ sensitivity (Jy μs) vs. $\log_{10}$ observation time (h). Only the first author's name is given. The year of the survey is also listed.

In conclusion, Astropulse's maximal event rate is comparable to, but not better than, recent radio surveys. (Especially Deneva's, which also uses the Arecibo telescope.) Nevertheless, Astropulse has proven the effectiveness of a novel search technique using coherent dedispersion. In the final chapter, Section 9, I will address a few possibilities for further research using coherent dedispersion radio surveys.

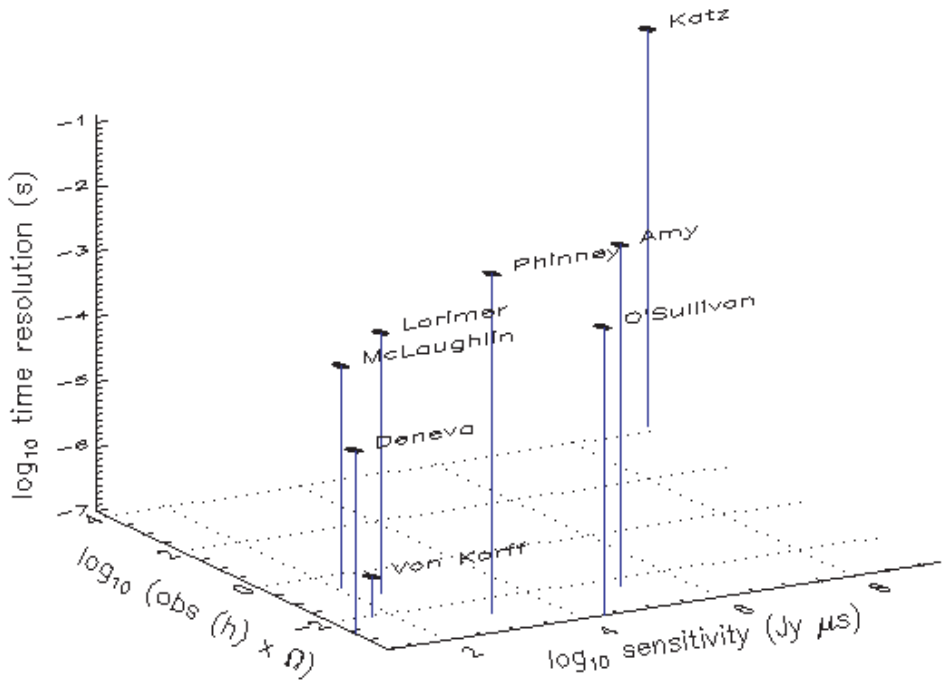


Figure 28: $\log_{10}$ sensitivity (Jy μs) vs. $\log_{10}$ observation time (h), with time resolution (s) on the z-axis.

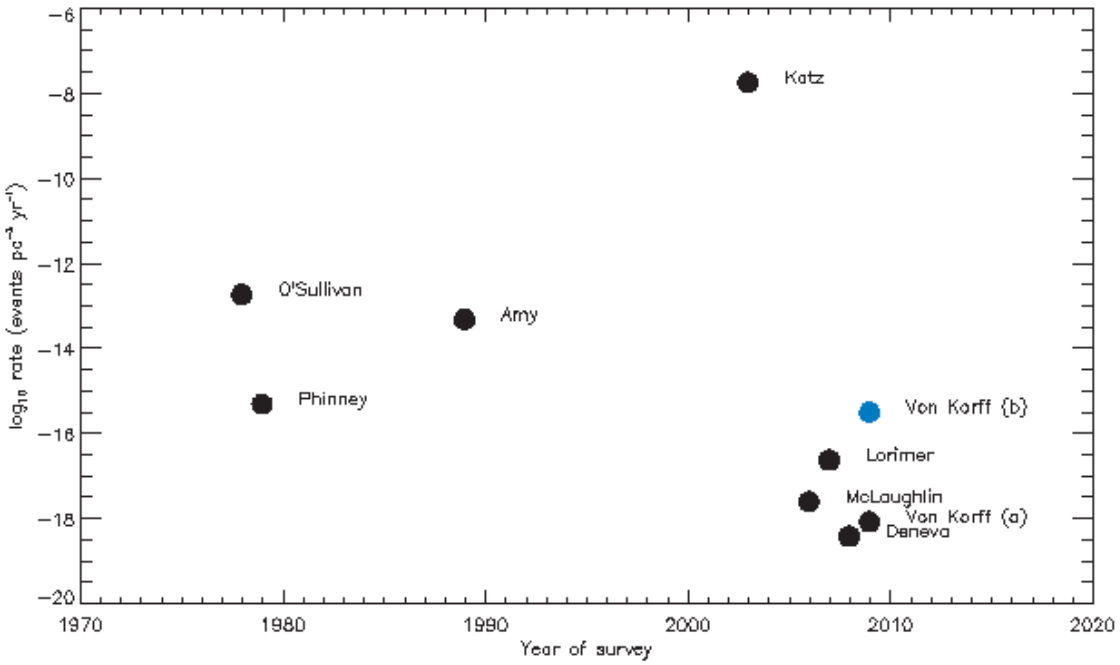


Figure 29: Year of survey vs. $\log_{10}$ event rate (pc^{-3} yr^{-1}). The blue data point counts all of our pulse detections after RFI mitigation, multiplying by a factor of 390.

8 Stardust@home

Stardust@home is a separate project from Astropulse, and is completely unrelated in many ways. However, the two projects are similar in their use of distributed processing methods. Whereas Astropulse is a distributed computing project (employing hundreds of thousands of volunteers' computers), Stardust@home is a distributed thinking project, employing nearly 30,,000 volunteers' brainpower and eyes. I was the main programmer for the web component of Stardust@home, including programs that trained volunteers, collected their input, and stored their observations in a database.

Launched in 1999, the Stardust spacecraft travelled through the tail of the comet Wild 2 in 2004, collecting sample particles.[11] At two points during its journey, it also raised a collector plate toward interstellar space, with the goal of gathering interstellar dust. In 2006, it landed in the Utah desert. The interstellar dust collector, a tray of aerogel blocks, was then scanned automatically by microscope at the Curatorial Facility in Houston's Johnson Space Center. Aerogel is a transparent, ultra light weight solid that can capture the dust grains without destroying them. When a dust grain strikes the aerogel, it burrows in, leaving a cylindrical crater, or "track."

Although interstellar dust had been detected previously by the Ulysses spacecraft in 1993, the Stardust mission is the first that brought the dust back to Earth. No other mission has brought back any matter from outside the orbit of the moon (although of course meteorites and cosmic rays hit the Earth of their own accord.)

The microscope's scan of the aerogel blocks resulted in 500,000 "focus movies", each consisting of 42 images. 1/5 of these images are low resolution, and the rest are high resolution. Each image in a focus movie represent a picture seen by the microscope as it focuses at different depths in the aerogel. The images are compressed to 12 kilobytes each, and stored on Amazon's S3 servers. Amazon then delivers the images to our volunteers' browsers as necessary.

By comparing the images within a focus movie, one can detect the telltale sign of a dust impact: a small round shape, representing the cross-section of a hole, that is present in images at all depths. Alternatively, the hole might shrink to nothing as one scans deeper into the aerogel, implying that the microscope can see the end of the track. Or, if the particle entered the aerogel at a shallower angle, the cross-section might look much longer and thinner, as in Figure 30. The volunteer's role is to examine the focus movies, decide whether a track is present, and if so, mark the location of the track. The volunteer's decision is then stored in the Stardust@home database.

We could have attempted to design an artificial intelligence that would automatically scan the focus movies and detect tracks. However, this method would likely have failed, since we didn't know exactly what we were looking for. In addition, many features are present in the data that superficially resemble tracks. Without a precise rule to differentiate cracks and flaws from dust tracks, we would not have been able to design an AI. Instead, I train volunteers on a sequence of images of cracks and flaws, and show them a few pictures

[11] `http://stardustathome.ssl.berkeley.edu/about.php`

of dust tracks from terrestrial experiments. It's up to them to extrapolate from this training data. Figures 30, 31, 32, 33, 34, and 35 depict sample training images.

In addition to giving Stardust aerogel images to the volunteers, I also mix some "calibration images" into the data stream. These test images are like the training data, in that I know the correct answer to the test. The calibration images supplement the training data to help us decide which volunteers can reliably detect the presence (or absence) of tracks in the aerogel. By combining many judgements of volunteers and applying our knowledge of volunteers' reliability, the Stardust@home team can automatically rule out many focus movies, leaving a handful to be examined by our own researchers. An additional advantage of this quantitative approach is that we were able to generate a "score" for each user, rewarding them for their volunteer activity by posting a list of top scorers on our website. The volunteer's score is simply the number of right answers on calibration images, minus the number of wrong answers. This rewards volunteers who view as many focus movies as possible, without rewarding those who answer randomly. In Stardust@home (and in Astropulse and SETI@home) we found that volunteer scores are the most successful technique for motivating volunteers.

Stardust@home has detected 28 tracks to date. Typically, the initial threshold has been approval by 5% to 10% of users viewing the focus movie; if a focus movie passes this threshold, then the Stardust@home team will view and judge the movie. In most cases, the tracks don't look much like our calibration images, justifying our human-driven approach as opposed to reliance on AI. Most of them have turned out to be secondary tracks from the spacecraft; that is, micrometeoroids hit the spacecraft and produced secondary ejecta from the impact, which fly into the collector.

The Stardust@home team recently discovered an interesting feature, named Orion by the first user to discover it (Figure 31). This subtle feature was uncovered by our "red team", a group of 30 users who are selected on the basis of their high scores with the calibration movies. The red team was given the ability to promote particles to an "alpha list" for further study. In order to increase our chances of finding more movies like it, we have added the Orion focus movie to the calibration movies and training data. Orion was clicked on by just two users apart from the red team's judgement, so it probably would not have been detected by the general population of volunteers. (A typical track has been viewed 400 times to date.)

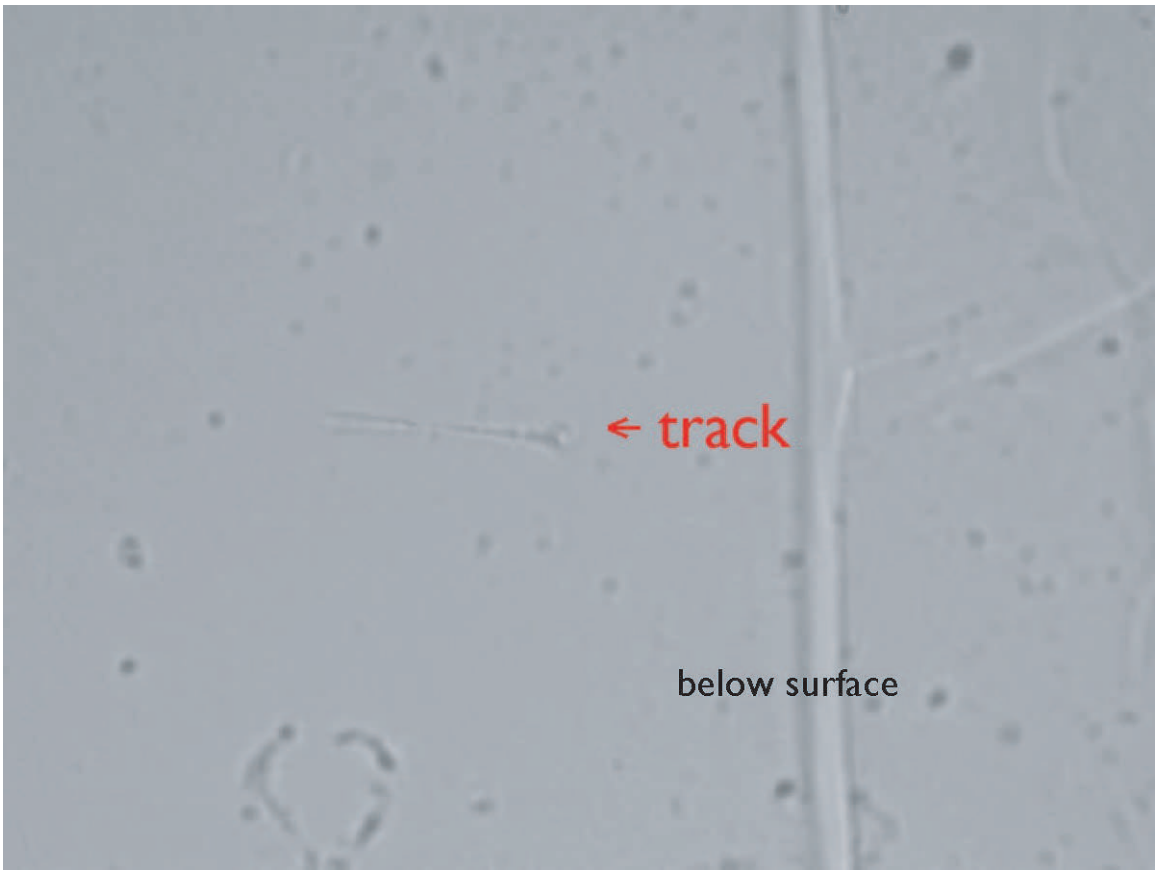


Figure 30: A track.

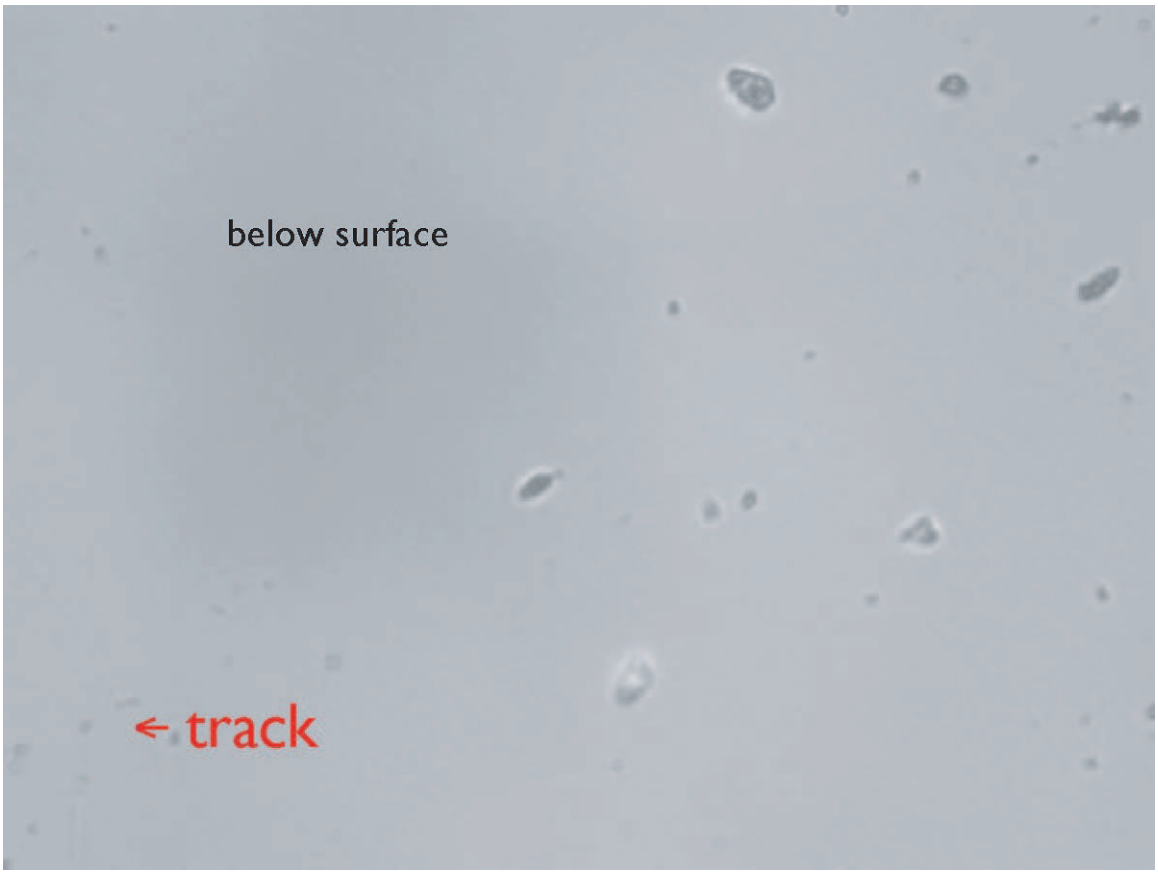


Figure 31: The Orion track.

Figure 32: A piece of terrestrial dust is stuck to the surface of the aerogel.

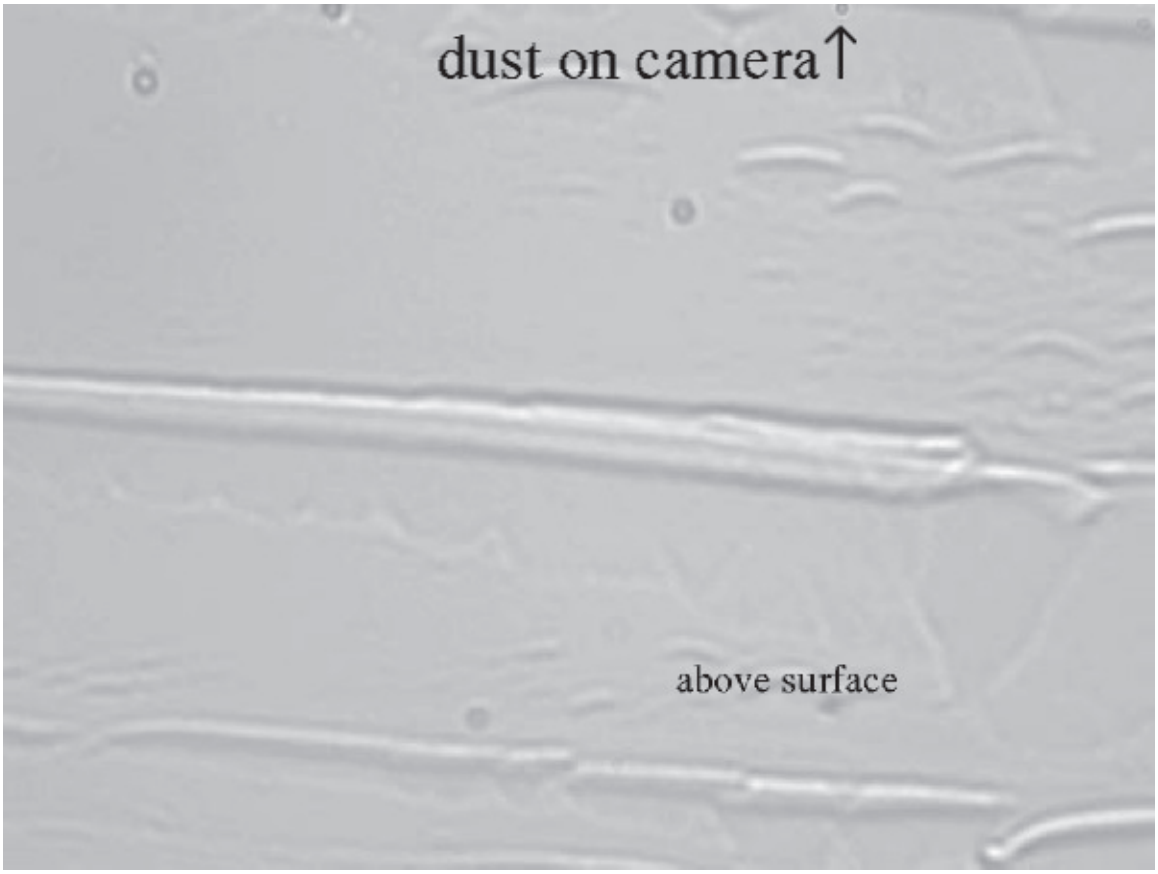


Figure 33: A piece of terrestrial dust is stuck to the camera. The volunteer can easily verify this because the dust looks the same at all depths in the movie.

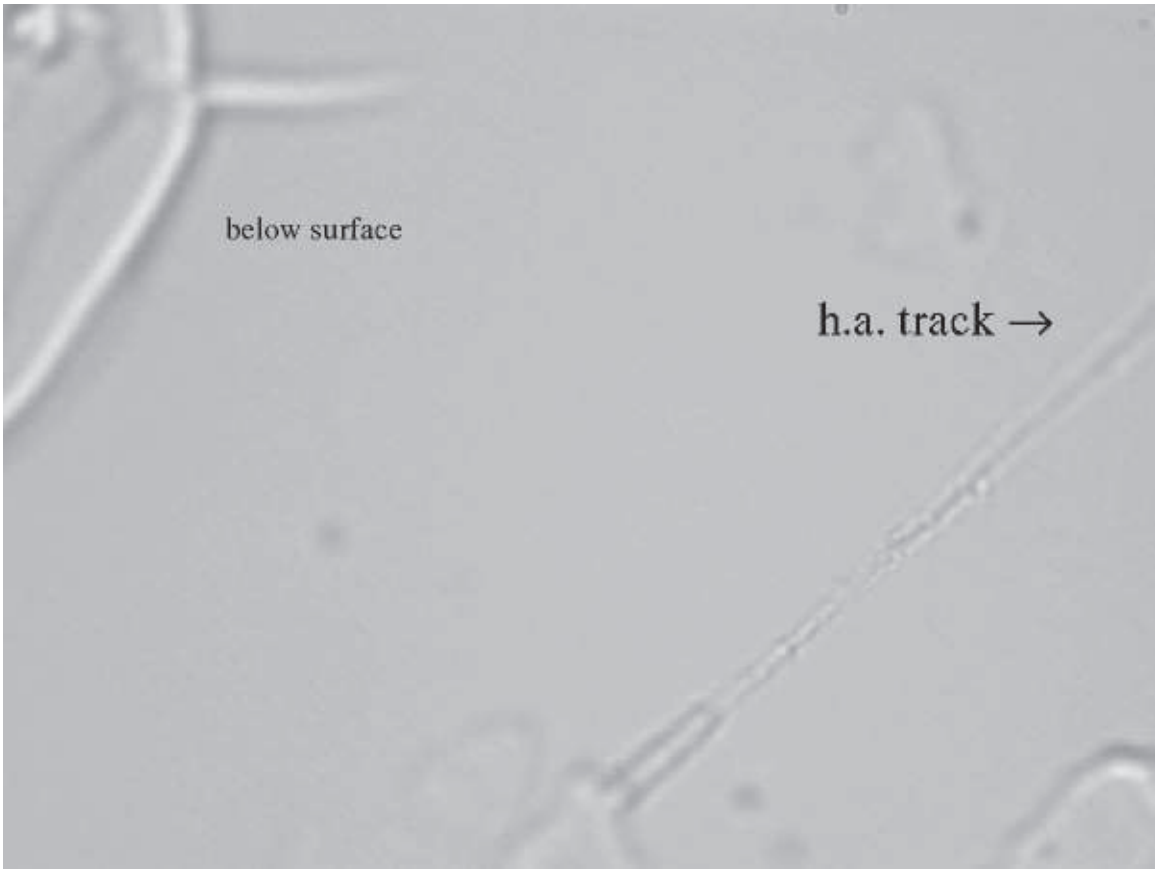


Figure 34: A particularly high angle track (having a shallow trajectory.) The volunteer can verify this because the track's location seems to move left and right as the microscope focuses up or down.

Figure 35: The surface of the aerogel is tilted with respect to the microscope.

9 Suggestions for further research

Future researchers could modify or improve on Astropulse in several ways. Possibilities include observing at different (probably higher) frequencies to reduce pulse broadening due to scattering, observing with multiple telescopes for RFI mitigation, performing a directed search, or improving time resolution by increasing bandwidth. An improvement to time resolution would also require a dramatic increase in processing power, as discussed in Section 9.5.

9.1 Directed search

An undirected search, like Astropulse, examines large sections of the sky. Furthermore, Astropulse is a commensal survey, so it has no control over its pointing. Depending on the target of the search, researchers might have prior knowledge about the likely locations of the radio sources. Are primordial black holes most likely to be near the galactic center? In other galaxies? In the space between galaxies? If we could answer these questions, we would be better equipped to perform a search for black holes. A search for unknown objects might be most productively directed toward the galactic center, in the absence of other information. On the other hand, in Astropulse's case, the comparison between sources in the Galactic disk and in the halo proved useful.

9.2 Multiple telescopes

Astropulse uses a single telescope. Observing with two telescopes simultaneously is difficult, but would permit the use of an extremely powerful RFI mitigation technique – namely, correlating signals between multiple telescopes that simultaneously observe a single point on the sky. Since the telescopes may be far from each other, it is necessary to take into account the time delay caused by the path length difference between the source and the telescopes. But this technique ensures that even if some RFI source were visible to both telescopes, if it appeared in the sidelobes, then its time delay would differ from the expected delay.

9.3 Multipolarization workunits

In order to make the most effective use of the Astropulse client, workunits should have contained information from both polarizations. This would have allowed me to set thresholds lower, for instance. (In the current setup, any decrease in our thresholds, even if justified by our multi-polarization RFI mitigation technique, would have flooded our database with pulses. Our database was barely large enough to handle all of the 26.2 ms time series that I stored in it.)

In a survey utilizing multiple telescopes, workunits could likewise contain information from all of the telescopes.

9.4 Higher frequencies

Radio scattering is substantially less significant at higher frequencies. For instance, frequencies of 6 GHz or higher allow observers to resolve the Crab pulsar at sub-nanosecond time scales. Our ability to see deep into the Galactic disk is severely limited by scattering at high DMs, especially since we want to examine a 0.4 μs time scale. Observation at higher frequencies would allow us to make use of our 0.4 μs time resolution at all DMs. One problem with higher frequency observations is that many sources have steep spectra, with much lower flux densities at higher frequencies.

Intergalactic electron densities are much smaller, so scattering is far less important when looking outside the Galaxy. However, these electron densities are hard to predict, so the effect is difficult to quantify.

9.5 Time resolution and processing power

Astropulse has access to a large amount of computing power, so it can employ coherent dedispersion in order to detect short-timescale events. One might hope that one could improve sensitivity by handling shorter time resolutions, which might require more processing power. However, it turns out that computation time increases like the fourth power of the sensitivity.

The client's computation takes time $O((M/N)N^2 \log N) = O(MN \log N)$, where N is the number of samples in a FFT length, and M is the number of samples in the entire datastream. (N dispersion measures must be tested, each FFT takes time $N \log N$, and there are M/N FFTs in all.) Now, M and N increase linearly with time resolution, so $MN \log N$ increases roughly quadratically. However, our sensitivity improves like the square root of the time resolution. Therefore, the computation time increases like the fourth power of the sensitivity.

This is a hard wall which will tend to prevent future surveys from improving on Astropulse's results by improving the time resolution, even if they have access to vast computational resources.

9.6 Parameter space of potential searches

Parameters relevant to a transient radio survey for detecting single, nonrepeating pulses include:

1. Sensitivity, which could be measured in Jy $\cdot$ μs . Astropulse has a different sensitivity at each pulse width, and is optimized for a certain range of pulse widths. (0.4 μs to 204.8 μs .)

2. Depth surveyed, in pc (the maximum distance at which an explosion with some standard energy could be detected.) This value can be derived from the sensitivity, and in fact knowing one is equivalent to knowing the other.

3. Solid angle surveyed.

4. Time scale surveyed. (Total observation time on the telescope, multiplied by the number of simultaneous beams.)

5. Minimum detectable rate, in events per pc^3 per year for some standard energy. This can be derived from the above values, although the derivation may contain many uncertain factors such as the astrophysical event's energy. For a given pulse width, the minimum detectable rate is really the only thing that matters, assuming the survey is looking in the right direction. If events are equally likely to occur in all telescope pointings over a certain solid angle, then staring at one point for a year is just as good as sweeping the whole solid angle over the course of a year.

6. Dispersion measure range. A survey which sees a larger range of DMs has a greater chance to detect something interesting.

So ultimately, the three relevant questions are: Are we pointing the telescope in the right area of the sky to detect an event? For a standard pulse energy, what is the minimum detectable rate of events at each pulse width? And what is our range of dispersion measures? Many of these parameters are plotted in Figures 27, 28, and 29 assuming a pulse width of 0.4 μs .

Astropulse can also search for repeating pulses. In this case, relevant parameters include:

1. Solid angle surveyed.

2. Time scale surveyed.

3. Dispersion measure range.

4. Sensitivity

5. Time resolution

However, Astropulse's repeating pulse search has little or no advantage over previous work, due to its relatively short integration time (13 s), even for pulses of the optimal width (0.4 μs or less.) Therefore, I have put less effort into that project. Nevertheless, it is possible that future researchers may obtain useful results from Astropulse's repeating pulse data.

References

Amy, S. W., Large, M. I., & Vaughan, A. E. 1989, Proc. Astron. Soc. Aust., 8, 2

Anderson, D. P. 2004, in Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing

Arnal, E. M., Bajaja, E., Larrarte, J. J., Morras, R., & Pöppel, W. G. L. 2000, A&AS, 142, 35

Backer, D. C., Kulkarni, S. R., Heiles, C., Davis, M. M., & Goss, W. M. 1982, Nature, 300, 615

Bajaja, E., Arnal, E. M., Larrarte, J. J., Morras, R., Pöppel, W. G. L., & Kalberla, P. M. W. 2005, A&A, 440, 767

Carter, B., Gibbons, G. W., Lin, D. N. C., & Perry, M. J. 1976, A&A, 52, 427

Cordes, J. M., Bhat, N. D. R., Hankins, T. H., McLaughlin, M. A., & Kern, J. 2004, ApJ, 612, 375

Deneva, J. S., & Cordes, J. M. 2008, in preparation, http://arxiv.org/abs/0811.2532

Fre, P., Gorini, V., & Magli, G. 1999, Classical and Quantum Black Holes (Institute of Physics Publishing, Bristol and Philadelphia)

Frolov, V. P., & Novikov, I. D. 1998, Black Holes Physics: Basic Concepts and New Developments (Springer Science & Business)

Guélin, M. 1973, Proc. IEEE, 61, 1298

Hagedorn, R. 1965, Nuovo Cimento Suppl., 3, 147

Hankins, T. H., Kern, J. S., Weatherall, J. C., & Eilek, J. A. 2003, Nature, 422, 142

Hartmann, D., & Burton, W. B. 1997, Atlas of Galactic Neutral Hydrogen (Cambridge University Press, Cambridge)

Hawking, S. 1971, MNRAS, 152, 75

—. 1974, Nature, 248, 30

Hessels, J. W. T. 2006, Sci, 311, 1901

Ioka, K. 2003, ApJ, 598, L79

Ivezic, Z., Tyson, J. A., Allsman, R., Andrew, J., & Angel, R. 2008, http://arxiv.org/abs/0805.2366

Kaaret, P., et al. 2006, ApJ, 657, L97

Kaiser, N. 2004, Proc. SPIE, 5489, 11

Kalberla, P. M. W., Burton, W. B., Hartmann, D., Arnal, E. M., Bajaja, E., Morras, R., & Pöppel, W. G. L. 2005, A&A, 440, 775

Katz, C. A., Hewitt, J. N., Corey, B. E., & Moore, C. B. 2003, PASP, 115, 675

Korpela, E. J., Heien, E. M., & Werthimer, D. 2000, in Bulletin of the American Astronomical Society, Vol. 32, 1492–+, http://setiathome.berkeley.edu/ korpela/papers/pulse_poster/

Kowalski, M., & Mohr, A. 2007, Astropart. Phys., 27, 533

Kramer, M., Xilouris, K. M., Lorimer, D., Doroshenko, O., Jessner, A., Wielebinski, R., Wolszczan, A., & Camilo, F. 1998, ApJ, 501, 270

Lattimer, J. M., Prakash, M., Masak, D., & Yahil, A. 1990, ApJ, 355, 241

Levinson, A., Ofek, E. O., Waxman, E., & Gal-Yam, A. 2002, ApJ, 576, 923

Lorimer, D., & Bailes, M. 2007, Sci, 318, 777

Lorimer, D., & Kramer, M. 2005, Handbook of Pulsar Astronomy (University Press, Cambridge)

Lovell, J. E. J., et al. 2007, in preparation, http://arxiv.org/abs/astro-ph/0701601

Macchetto, F. D. 1999, The Supermassive Black Hole of M 87 and the Kinematics of its Associated Gaseous Disk (Springer, Berlin / Heidelberg), 291–300

MacGibbon, J. H., Bailes, M., & Weber, B. R. 1990, Phys. Rev. D, 41, 3052

Manchester, R. N., et al. 2001, MNRAS, 328, 17

McLaughlin, M. A., Lyne, A. G., Lorimer, D. R., Kramer, M., Faulkner, A. J., Manchester, R. N., Cordes, J. M., & Camilo, F. 2006, Nature, 439, 817

O'Sullivan, J. D., Ekers, R. D., & Shaver, P. A. 1978, Nature, 276, 590

Phinney, S., & Taylor, J. H. 1979, Nature, 277, 117

Popov, M. V., & Stappers, B. 2007, A&A, 470, 1003

Raine, D., & Thomas, E. 2005, Black Holes: An Introduction (Imperial College Press, London)

Rees, M. J. 1977, Nature, 266, 333

Rohlfs, K., & Wilson, T. L. 2000, Tools of Radio Astronomy (Springer-Verlag, Berlin)

Sallmen, S., & Backer, D. C. 1995, ASP Conf. Ser., 72, 340

Shapiro, S. L., & Teukolsky, S. A. 1983, Black Holes, White Dwarfs, and Neutron Stars (Weiley-VCH, Weinheim)

Staelin, D. H. 1969, Proc. IEEE, 57, 724

Ukwatta, T. N., et al. 2010, in preparation, http://arxiv.org/abs/1003.4515

Van Vleck, J., & Middleton, D. 1966, Proc. IEEE, 54, 2

Vestrand, W. T., et al. 2005, Nature, 435, 178

Voges, B., et al. 1999, A&A, 349, 389

Wilson, T. L., Rohlfs, K., & Hüttemeister, S. 2009, Tools of Radio Astronomy (Springer)

A Candidate sources

Table 4: List of candidates with distinct times. Pulses are ordered first by galactic latitude, then (for latitudes within 1°) by galactic longitude. "#" is the count of the number of pulses detected from this source. DM is given in pc cm^{-3}. DM_1 is the lowest DM detected from this source, and DM_2 is the highest. "width" is the width of one of the pulses associated with this source, in μs.

	#	ℓ (°)	b	RA (h)	dec	julian date	DM_1	DM_2	Jy μs	width
1	2	75	-52	23.0	1	2455087.686	676.3	699.1	206.9	102.4
2	1	148	-51	1.9	9	2454840.463	240.6	240.6	210.6	102.4
3	1	179	-47	3.1	0	2455079.862	98.6	98.6	283.7	204.8
4	2	183	-43	3.5	0	2455077.880	103.7	137.8	281.5	204.8
5	5	158	-40	2.7	15	2454634.033	49.9	52.0	106.9	6.4
6	3	57	-38	21.8	0	2455153.434	332.3	338.0	116.6	12.8
7	1	83	-38	22.7	15	2454793.415	409.3	409.3	181.0	51.2
8	5	86	-36	22.8	17	2455043.734	84.9	145.2	299.5	204.8
9	2	54	-34	21.5	0	2455077.630	161.6	210.7	399.1	204.8
10	1	54	-34	21.5	0	2455077.631	174.4	174.4	285.5	204.8
11	1	154	-31	2.8	25	2454806.593	109.2	109.2	93.5	6.4
12	1	82	-29	22.3	21	2455026.781	131.9	131.9	283.9	204.8
13	2	59	-29	21.4	7	2454869.201	176.8	198.8	206.7	102.4
14	3	163	-28	3.4	23	2454853.467	51.8	53.9	67.3	1.6
15	1	71	-15	21.1	24	2455068.640	125.3	125.3	299.2	204.8
16	2	42	-12	19.8	3	2454525.058	114.3	114.7	64.7	0.8
17	2	61	-7	20.2	22	2455044.689	532.5	548.6	265.3	204.8
18	1	65	-5	20.2	25	2455056.635	82.8	82.8	89.4	6.4
19	1	68	-4	20.3	28	2454884.115	818.8	818.8	265.3	204.8
20	1	60	-4	19.9	22	2454892.059	344.6	344.6	162.5	51.2
21	1	66	-3	20.2	27	2454903.040	606.8	606.8	266.4	204.8
22	2	72	-3	20.4	32	2454901.071	475.8	478.2	123.5	25.6
23	2	183	-3	5.7	24	2454823.667	499.0	511.8	124.2	25.6
24	2	35	-2	19.1	1	2454926.924	50.1	50.3	103.7	6.4
25	1	52	-2	19.6	16	2454728.479	174.5	174.5	211.2	102.4
26	1	52	-2	19.6	16	2454728.480	213.0	213.0	225.8	102.4
27	8	52	-2	19.6	16	2454728.481	143.8	282.4	215.2	102.4
28	6	52	-2	19.6	16	2454805.249	133.4	173.2	277.7	204.8
29	1	52	-2	19.6	16	2454805.250	376.9	376.9	279.0	204.8
30	2	52	-2	19.6	16	2454815.320	150.6	170.7	162.0	51.2
31	1	52	-2	19.6	16	2454827.197	201.2	201.2	188.0	51.2
32	1	52	-2	19.6	16	2454827.198	124.2	124.2	272.0	204.8
33	2	52	-2	19.6	16	2454828.200	153.1	163.3	180.4	51.2
34	2	53	-2	19.6	16	2454828.203	223.9	231.0	300.8	204.8
35	3	53	-2	19.6	16	2454828.204	706.7	759.1	238.9	102.4
36	1	52	-2	19.6	16	2455063.547	385.7	385.7	303.7	204.8

Continued on next page...

	#	ℓ (°)	b	RA (h)	dec	julian date	DM_1	DM_2	Jy μs	width
37	1	61	-3	19.9	23	2454913.019	507.0	507.0	162.5	51.2
38	6	61	-3	19.9	23	2454913.020	50.8	54.3	119.7	12.8
39	2	60	-3	19.9	23	2455084.546	743.0	764.8	57.2	0.4
40	1	179	-2	5.6	29	2454822.672	227.3	227.3	124.0	25.6
41	2	179	-2	5.6	29	2454822.676	201.1	205.6	88.5	6.4
42	2	37	-1	19.0	4	2454869.073	732.1	732.2	72.0	1.6
43	2	49	-1	19.4	14	2454549.010	262.2	265.7	127.5	25.6
44	1	49	-1	19.4	14	2454549.012	200.2	200.2	79.4	3.2
45	1	49	-1	19.4	14	2454552.995	315.9	315.9	129.2	25.6
46	5	58	-0	19.7	22	2454807.269	49.5	50.4	68.0	1.6
47	5	57	-0	19.7	22	2454807.271	50.9	325.6	107.4	3.2
48	2	58	-0	19.7	22	2454870.073	70.8	71.0	73.0	1.6
49	1	75	-1	20.4	36	2454705.604	388.0	388.0	265.0	204.8
50	2	36	0	18.9	3	2454574.911	539.4	572.9	88.2	3.2
51	1	45	1	19.2	11	2454552.983	292.2	292.2	130.4	25.6
52	1	56	0	19.6	20	2455084.531	511.8	511.8	131.1	25.6
53	1	58	0	19.6	22	2455063.616	292.5	292.5	139.6	25.6
54	1	60	0	19.7	24	2455063.622	592.0	592.0	79.5	3.2
55	1	179	0	5.7	30	2454824.687	368.4	368.4	270.9	204.8
56	2	179	0	5.7	30	2454824.690	459.2	472.1	268.6	204.8
57	1	59	2	19.6	24	2455056.608	618.5	618.5	265.1	204.8
58	1	65	2	19.8	29	2455084.540	570.0	570.0	110.3	12.8
59	1	33	2	18.7	1	2455153.292	420.8	420.8	125.3	25.6
60	1	33	2	18.7	1	2455153.298	505.5	505.5	124.2	25.6
61	1	52	2	19.3	18	2454815.273	741.1	741.1	125.1	25.6
62	3	52	2	19.3	18	2454815.283	733.9	737.4	79.6	3.2
63	1	52	2	19.3	18	2454827.207	715.7	715.7	123.7	25.6
64	1	57	2	19.5	22	2454807.322	49.8	49.8	97.6	6.4
65	5	59	2	19.6	24	2454807.336	54.9	57.3	115.6	12.8
66	2	62	2	19.6	27	2454815.305	555.5	598.0	209.4	102.4
67	2	62	2	19.6	27	2454815.307	621.1	624.6	125.2	25.6
68	1	66	3	19.7	30	2454675.658	697.7	697.7	267.4	204.8
69	1	55	4	19.3	21	2455056.598	794.1	794.1	273.4	204.8
70	3	58	4	19.4	24	2455055.605	517.6	555.2	217.3	102.4
71	1	174	3	5.7	36	2454887.494	698.8	698.8	177.7	51.2
72	2	184	5	6.2	28	2454822.692	550.9	550.9	237.4	102.4
73	2	215	9	7.5	2	2454850.667	51.2	52.1	81.0	1.6
74	5	215	9	7.5	2	2454850.669	50.2	54.7	81.7	3.2
75	3	54	10	18.9	23	2455087.496	118.5	243.4	211.6	102.4
76	1	217	10	7.6	1	2454852.691	493.4	493.4	164.1	51.2
77	1	217	14	7.8	3	2454834.726	49.6	49.6	107.0	3.2
78	1	226	27	8.8	2	2454871.694	51.4	51.4	65.2	1.6
79	2	201	29	8.3	22	2454806.832	53.3	56.4	89.3	3.2
80	6	201	29	8.3	22	2454874.612	50.2	50.7	163.6	51.2
81	1	201	29	8.3	22	2454874.677	51.6	51.6	87.4	6.4
82	3	201	29	8.3	22	2454911.502	50.2	51.5	134.2	25.6

Continued on next page...

	#	ℓ (°)	b	RA (h)	dec	julian date	DM_1	DM_2	Jy μs	width
83	1	230	33	9.3	2	2454850.744	675.4	675.4	110.9	12.8
84	1	195	36	8.7	29	2454530.595	50.2	50.2	85.0	3.2
85	2	11	36	16.1	-0	2454616.691	50.4	51.8	114.8	12.8
86	4	9	38	16.0	-0	2454616.661	49.5	53.1	95.6	3.2
87	5	10	38	16.0	-0	2454616.662	49.5	51.4	115.4	12.8
88	4	9	38	15.9	-0	2454616.660	49.8	51.0	87.0	3.2
89	9	7	40	15.8	-0	2454616.654	49.5	52.4	124.1	3.2
90	1	8	39	15.8	-0	2454616.656	51.2	51.2	69.6	1.6
91	8	7	41	15.7	-0	2454616.652	49.5	52.5	86.6	6.4
92	3	6	41	15.7	-0	2454616.650	49.5	49.6	132.6	25.6
93	1	44	47	16.1	26	2454509.906	520.0	520.0	180.9	51.2
94	2	244	51	10.6	4	2454843.818	755.1	756.4	128.2	25.6
95	1	195	55	10.2	32	2454556.585	57.2	57.2	130.0	25.6
96	1	196	54	10.1	32	2454554.588	49.5	49.5	120.2	12.8
97	1	38	58	15.3	25	2454548.819	801.0	801.0	259.4	204.8
98	2	269	60	11.8	1	2454852.840	622.3	706.4	222.0	102.4
99	2	280	69	12.3	7	2454571.649	75.3	101.9	324.3	204.8
100	1	277	69	12.3	8	2454911.679	53.0	53.0	79.1	3.2
101	6	277	69	12.3	8	2454911.684	49.5	53.8	107.0	3.2
102	1	277	69	12.3	8	2454911.685	51.6	51.6	71.3	1.6
103	5	277	69	12.3	8	2454911.688	49.5	52.5	158.9	25.6
104	1	277	69	12.3	8	2454911.690	49.5	49.5	140.9	25.6
105	6	277	69	12.3	8	2454911.709	49.5	52.5	83.6	3.2
106	2	284	69	12.4	7	2454571.605	49.5	52.0	81.0	3.2
107	1	284	69	12.4	7	2454571.637	55.7	55.7	83.3	3.2
108	2	226	72	11.6	22	2454905.689	618.7	624.5	167.3	51.2

B Source code for coherent dedispersion, in C++

B.1 ap_client.cpp

The file ap_client.cpp contains the main program for the Astropulse client, which runs through the nested loops described in Section 3.3.

```
// Copyright 2003 Regents of the University of California

// Astropulse is free software; you can redistribute it and/or
   modify it under
// the terms of the GNU General Public License as published by the
    Free
// Software Foundation; either version 2, or (at your option) any
   later
// version.

// Astropulse is distributed in the hope that it will be useful,
   but WITHOUT
// ANY WARRANTY; without even the implied warranty of
   MERCHANTABILITY or
// FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public
   License for
// more details.

// You should have received a copy of the GNU General Public
   License along
// with Astropulse; see the file COPYING.  If not, write to the
   Free Software
// Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA
   02111-1307, USA.

/* ap_client_main.C */
/* Main program for AstroPulse client */
#include "ap_config.h"
#ifdef _WIN32
#include "boinc_win.h"
#endif

#ifdef HAVE_UNISTD_H
#include <unistd.h>
#endif

#ifdef _WIN32
#include <windows.h>
#define BOINC_APP_GRAPHICS 1
#endif
```

```
#include <cassert>
#include <cstdio>
#include <cstdlib>
#include <iostream>
#include "boinc_api.h"
#include "util.h"
#include "str_util.h"

#include "astropulse.h"
#include "diagnostics.h"

#include "mtrand.h"

#ifdef BOINC_APP_GRAPHICS
// #include "ap_graphics.h"    Not part of the main program any
    more
#include "ap_gfx_main.h"
#include "graphics2.h"
#endif

#include <time.h>

long starttime = time(0);
int lastcputicks = clock();
float cputime = 0;
int maxcputicksinterval = 0;

// #define DEBUGGING_0 1 // If set to 1, print program status
    regularly (i.e. run in a verbose manner)

#ifdef _WIN32
#include <windows.h>
#include "stackwalker_win.h"

extern int main(int argc, char** argv);

int WINAPI WinMain(HINSTANCE hInst, HINSTANCE hPrevInst, LPSTR
    Args, int WinMode) {
  LPSTR command_line;
  char* argv[100];
  int argc;

  //InitAllocCheck();

  command_line = GetCommandLine();
  argc = parse_command_line(command_line,argv);

  int ret = main(argc, argv);
```

```
  //DeInitAllocCheck();

  return ret;
}
#endif

/* Default thresholds */
static float thresh_def[] = {
                              (float)21.50,
                              (float)24.748650,
                              (float)30.015163,
                              (float)38.773587,
                              (float)53.662645,
                              (float)79.565603,
                              (float)125.71365,
                              (float)209.85380,
                              (float)366.44671,
                              (float)662.93874,
                              (float)1232.2705,
                             0, 0, 0, 0
                            };

#ifdef HAVE_GL_LIB
#ifdef UNIX

#include <GL/glut.h>

void displayCB(void)  /* function called whenever redisplay needed
    */
{
  glClear(GL_COLOR_BUFFER_BIT);  /* clear the display */
  glColor3f(1.0, 1.0, 1.0);  /* set current color to white */
  glBegin(GL_POLYGON);    /* draw filled triangle */
  glVertex2i(200,125);    /* specify each vertex of triangle */
  glVertex2i(100,375);
  glVertex2i(300,375);
  glEnd();     /* OpenGL draws the filled triangle */
  glFlush();     /* Complete any pending operations */
}

void keyCB(unsigned char key, int x, int y) /* called on key press
    */
{
  if ( key == 'q' ) exit(0);
}

#endif
#endif
```

```
/*
enum OnOff {on, off};

class timer {
  private:
  clock_t ticks;
  clock_t last_clock;
  clock_t start_clock;

  long seconds;
  long last_time;
  long start_seconds;

  OnOff state;

  public:
  timer() {
    ticks = 0;
    seconds = 0;
    start_clock = clock();
    start_seconds = time(NULL);
    state = off;
  }
  void start() {
    last_clock = clock();
    last_time = time(NULL);
    assert(state == off);
    state = on;
  }
  void stop() {
    clock_t ticks_this_time = clock() - last_clock;
    ticks += ticks_this_time;
    long seconds_this_time = time(NULL) - last_time;
    seconds += seconds_this_time;
    assert(state == on);
    state = off;
  }
  void reset() {
    ticks = 0;
    seconds = 0;
    start_clock = clock();
    start_seconds = time(NULL);
    state = off;
  }
  clock_t get_ticks() {
    return ticks;
  }
  clock_t get_total_ticks() {
    return clock() - start_clock;
```

```
  }
  long get_seconds() {
    return seconds;
  }
  long get_total_seconds() {
    return time(NULL) - start_seconds;
  }
};

void print_time() {
  long seconds = time(NULL) - starttime;
  long minutes = seconds / 60;
  int hours = minutes / 60;
  int days = hours / 24;
  printf("Time since start: %d days, %d hours, %ld minutes, %ld
     seconds\n",
   days, hours % 24, minutes % 60, seconds % 60);
  fprintf(stderr, "Time since start: %d days, %d hours, %ld
     minutes, %ld seconds\n",
   days, hours % 24, minutes % 60, seconds % 60);
}
*/


// This was a function used temporarily for debugging.
// It looks at a short time series, and determines whether
// the power stays above a certain threshhold for a certain length
// of time in consecutive bins.
// Not sure why I wanted to know this.
#if 0
int detect_hit(float *arr, float& total) {

  int stage = 0;
  total = 0;
  int count = 0;
  for (int i = 0; i < 10; i++) {
    if ( arr[i] < 1 && stage == 0) {
      return 0;  // not a hit; goes below 1 too soon.
    }
    if ( arr[i] > 5 && stage == 0) { // first time above 5
      stage = 1;
      count++;
    }
    if ( arr[i] > 5 && stage == 1) { // Above 5, but not first
       time
      count++;
    }
    if ( arr[i] < 5 && stage == 1) { // Going below 5
      if (count < 5) return 0; // goes below 5 too soon
```

```
      else stage = 2; // goes below 5 at the right time
    }
    total += arr[i];
  }

  if (count < 5) return 0;
  return 1;
}
#endif

/* updatecputime
 *
 * Purpose: determine the runtime of the program.
 * Sets the global variable cputime to equal the number of seconds
     the program has been running so far.
 * Sets the global variable maxcputicksinterval to the maximum
    interval in clock ticks so far between
 * calls to this function.
 */
void updatecputime() {
  int cputicksinterval = abs(clock() - lastcputicks);
  lastcputicks = clock();
  maxcputicksinterval = (maxcputicksinterval > cputicksinterval)?
     maxcputicksinterval:cputicksinterval;
  cputime += ((float)cputicksinterval / (float)CLOCKS_PER_SEC);
}

/* worker
 *
 * Function is not called unless graphics are turned on.
 */
void worker() {
  Astropulse::client.init();
  Astropulse::client.science.mainloop();
  Astropulse::client.finish();
  boinc_finish(0);
}

// Command-line arguments must be declared globally, because
   worker() takes no arguments
bool skipffa_long = false;
bool skipffa_short = false;
bool debug_msg; // set to true if debugging
bool debug_loop_msg; // set to true if debugging
bool write_lcgf_short = false;
bool write_lcgf_long = false;
bool remove_radar = false;
bool print_best = false;
bool pulsegraphs = false;
```

```
int single_dm_chunk_large = -1;
int single_dm_chunk_small = -1;
int data_byte_limit = -1;
int single_dm = -1;
int full_dm = -1;
int pulse_limit_single = 30;
int pulse_limit_rep = 30;
int fold_dm = 0;
int fold_dm_chunk_small = 0;
double ffa_thresh_mult = 1.0;

APP_INIT_DATA app_init_data;

/* Main Program */
int main(int argc, char *argv[]) {

  Astropulse::main_timer.start();

  //char* foo = (char*) (void*) 0xbadf00d;
  //*foo = 'x';

  //fprintf(stderr, "START TIME: %ld\n", time(NULL));

  int i, retval;
  bool standalone = false;

  /* Initialize Diagnostics
   *
   * The diagnostics flags are defined in boinc/lib/diagnostics.h
   * There are many flags, and we are setting some of them.  E.g.
   * BOINC_DIAG_REDIRECTSTDERR redirects stderr to stderr.txt
   */
  unsigned long dwDiagnosticsFlags = 0;

  dwDiagnosticsFlags =
    BOINC_DIAG_DUMPCALLSTACKENABLED |
    BOINC_DIAG_HEAPCHECKENABLED |
    BOINC_DIAG_TRACETOSTDERR |
    BOINC_DIAG_REDIRECTSTDERR;

  /* boinc_init_diagnostics is a function in the boinc API.
   */
  retval = boinc_init_diagnostics(dwDiagnosticsFlags);
  if (retval) {
    fprintf(stderr, "boinc_init_diagnostics failed: %d\n", retval)
        ;
    exit(retval);
  }
```

```
boinc_parse_init_data_file();
boinc_get_init_data(app_init_data);
// We've moved the state variable into the graphics shmem
   segment,
// so we always need to initialize graphics.
ap_graphics_init(app_init_data);


/* Possible arguments:
 * -standalone
 * -reset: reset ap_state.dat and pulse.out
 * -thresh0 %f: Set the power threshhold value for zero co-adds
 * -skipffa_long, -skipffa_short: Don't run the fast folding
    algorithm
 * -print_best: Print out the best pulses on a regular basis
 * -debug_msg: Print out debugging messages
 * -debug_loop_msg: Print out debugging messages for all
    statements in the main loop
 */
for (i=1; i<argc; i++) {
  if (!strcmp(argv[i], "-standalone")) {
    standalone = true;
    fprintf(stderr, "## debug standalone mode\n");
  }
  if (!strcmp(argv[i], "-reset")) {
    fprintf(stderr, "## resetting ap_state.dat and pulse.out\n")
       ;
    unlink("ap_state.dat");
    unlink("pulse.out");
    continue;
  }
  if (standalone && !strcmp(argv[i], "-thresh0") && ++i < argc)
     {
    fprintf(stderr, "## thresh[0] = %f\n",
            (Astropulse::client.ap_shmem->ap_gdata.state.thresh
               [0] =(float)atoi(argv[i])) );
    continue;
  }
  if (!strcmp(argv[i], "-skipffa_long")) {
    skipffa_long = true;
    fprintf(stderr, "Skipping the long ffa\n");
  }
  if (!strcmp(argv[i], "-skipffa_short")) {
    skipffa_short = true;
    fprintf(stderr, "Skipping the short ffa\n");
  }
  if (!strcmp(argv[i], "-remove_radar")) {
    remove_radar = true;
    fprintf(stderr, "Removing radar\n");
```

```
}
if (!strcmp(argv[i], "-debug_msg")) {
  debug_msg = true;
  fprintf(stderr, "Debugging messages on\n");
}
if (!strcmp(argv[i], "-debug_loop_msg")) {
  debug_loop_msg = true;
  fprintf(stderr, "Debugging loop messages on\n");
}
if (!strcmp(argv[i], "-write_lcgf_long")) {
  write_lcgf_long = true;
}
if (!strcmp(argv[i], "-write_lcgf_short")) {
  write_lcgf_short = true;
  if(write_lcgf_long) {
    fprintf(stderr, "Error: can't write both lcgf_short and
       lcgf_long.\n");
    exit(1);
  }
}
if (!strcmp(argv[i], "-print_best")) {
  print_best = true;
}
/* We can print data files that will let us graph
 * number of pulses vs. dm, or # vs. power
 */
if (!strcmp(argv[i], "-pulsegraphs")) {
  pulsegraphs = true;
  fprintf(stderr, "Printing pulsegraphs.\n");
}
/* -dm_chunk (the command line argument) should be set to
 * dm_low + multiple of dm_chunk (the values defined in
 * the header of the workunit.)
 */
if (!strncmp(argv[i], "-dm_chunk_large", 15)) {
  single_dm_chunk_large = atoi(argv[i] + 15);
  fprintf(stderr, "Running on a single large dm chunk: %d\n",
     single_dm_chunk_large);
  /* -dm_chunk_small (the command line argument) should be set
      to a value
   * divisible by dm_chunk_small, and between 0 and
      dm_chunk_large
   * (defined in the header of the workunit)
   */
} else if (!strncmp(argv[i], "-dm_chunk_small", 15)) {
  single_dm_chunk_small = atoi(argv[i] + 15);
  fold_dm_chunk_small = single_dm_chunk_small;
  fprintf(stderr, "Running on a single small dm chunk: %d\n",
     single_dm_chunk_small);
```

```
      /* -dm (the command line argument) should be set to a value
         between 0 and
       * dm_chunk_small (defined in the header of the workunit)
       */
    } else if (!strncmp(argv[i], "-dm", 3)) {
      single_dm = atoi(argv[i] + 3);
      fold_dm = single_dm;
      fprintf(stderr, "Running on a single dm: %d\n", single_dm);
    } else if(!strncmp(argv[i], "-full_dm", 8)) {
      full_dm = atoi(argv[i] + 8);
    } else if (!strncmp(argv[i], "-pulse_limit_single", 19)) {
      // To get no pulse limit, set -pulse_limit_single0
      pulse_limit_single = atoi(argv[i] + 19);
      fprintf(stderr, "Running on pulse limit single %d\n",
         pulse_limit_single);
    } else if (!strncmp(argv[i], "-pulse_limit_rep", 16)) {
      // To get no pulse limit, set -pulse_limit_rep0
      pulse_limit_rep = atoi(argv[i] + 16);
      fprintf(stderr, "Running on pulse limit rep %d\n",
         pulse_limit_rep);
    }
    /* -data_byte_limit should be set to a value between 0 and
     * datasize (which is 1/4 of nsamples, defined in the header
        of
     * the workunit.)  Actually the upper limit is datasize - (
        something small)
     * The smallest meaningful amount of data bytes is 4096, which
        is
     * (fft_len / 4) / 2, see below for discussion.
     */
    if (!strncmp(argv[i], "-data_byte_limit", 16)) {
      data_byte_limit = atoi(argv[i] + 16);
      fprintf(stderr, "Running on data chunks starting up to: %d\n
         ", data_byte_limit);
    }
        if (!strncmp(argv[i], "-ffa_thresh_mult", 16)) {
          ffa_thresh_mult = atof(argv[i] + 16);
          fprintf(stderr, "Multiplying ffa threshold by: %f\n",
             ffa_thresh_mult);
        }
  }

// #ifndef BOINC_APP_GRAPHICS
  /* The function below is defined in boinc/api/boinc_api.C */
  retval = boinc_init();
  if (retval) {
    fprintf(stderr, "boinc_init failed: %d (try -standalone for
       testing)\n", retval);
    exit(retval);
```

```
  }
// #else
  //app_graphics_init();
  // ap_graphics.data_struct_init();
  /* Boinc API: initialize boinc with graphics.  worker() is a
     function
  * that performs the role of main(), whereas main() should run
  * the graphics.  It's not clear to me that this is in fact what
     happens:
  * it looks to me like both main and worker are playing the same
     role here.
  */
  // retval = boinc_init_graphics(worker);
  // if (retval) {
  //   fprintf(stderr, "boinc_init_graphics failed: %d\n", retval)
     ;
  //   exit(retval);
  // }
// #endif

  /* The 3 functions below are all defined in this file */
  if (debug_msg) {
    printf("In ap_client_main.cpp: at Astropulse::client.init()\n"
       );
    fprintf(stderr, "In ap_client_main.cpp: at Astropulse::client.
       init()\n");
  }
  // checking for blank statefile
  // printf("Just before client.init()\n");
  // fprintf(stderr, "Just before client.init()\n");
  Astropulse::client.init();
  if (debug_msg) {
    printf("In ap_client_main.cpp: at Astropulse::client.science.
       mainloop()\n");
    fprintf(stderr, "In ap_client_main.cpp: at Astropulse::client.
       science.mainloop()\n");
    fflush(stdout);
  }
  Astropulse::client.science.mainloop();
  if (debug_msg) {
    printf("In ap_client_main.cpp: at Astropulse::client.finish()\
       n");
    fprintf(stderr, "In ap_client_main.cpp: at Astropulse::client.
       finish()\n");
    fflush(stdout);
  }
  Astropulse::client.finish();

  //fprintf(stderr, "END TIME: %ld\n", time(NULL));
```

```
  Astropulse::main_timer.stop();

  /* The function below is defined in boinc/api/boinc_api.C
   * The argument 0 means we've successfully finished the result.
   */
  boinc_finish(0);
  return 0;
}

double interpolate_dec(double jd0, double jd1, double pulse_jd,
   double dec0, double dec1);
double interpolate_ra(double jd0, double jd1, double pulse_jd,
   double ra0, double ra1);

void segfault_atexit() {
  assert(0);
}

/* variables and functions defined here may be called via
 * Astropulse::<variable> or Astropulse::<function>
 */
namespace Astropulse {
/* The class below is defined in astropulse.h
 * Each Client contains an Outfile, Wufile, Foldfile, Statefile,
    and Science.
 */
Client client;
timer main_timer;

/* calculate_fold_level
 *
 * Argument: int dm_chunk
 * Return value: The highest power of 2 that is less than or equal
     to dm_chunk.
 * So calculate_fold_level(32) == 5 and calculate_fold_level(63)
    == 5.
 */
inline int calculate_fold_level(int dm_chunk) {
  /* Figure out sizes */
  int level = 0;
  for (;; ++level) {
    int temp = dm_chunk >> level;
    if (temp == 0) {
      fprintf(stderr, "Error calculating fold_level.\n");
      state_t &state=client.ap_shmem->ap_gdata.state;
      state.print();
      exit(PARSE_ERR);
    }
```

```
    if (temp == 1) {
      return level;
    }
  }
}

int pow_int(int a, int b) {
  int retval = 1;
  for(int i = 0; i < b; i++) {
    retval *= a;
  }
  return retval;
}

/* Calculates the highest power of two that is a factor
 * of the input.  For instance,
 * input 4 => output 2
 * input 8 => output 3
 * input 12 => output 2
 * UNUSED FUNCTION
inline int calculate_power_of_two_factor(int dm) {
  int retval = 0;
  assert(dm != 0);
  while(dm % 2 == 0) {
    retval++;
    dm /= 2;
  }
  return retval;
}
*/

state_t::state_t() {
  /* Default parameters for a state_t.
   * This constructor function is mostly useless.
   * In our program, the only state_t is the global variable
      client.state.
   * In Client::init, some of the members of client.state are
      reset by
   * Wufile::parse_header_to_state, which alters the
   * global variable by reading
   * from in.dat.  Other members are set in the body of Client::
      init
   *
   * Variables that do not appear to get reset include:
   * best[].peak_power, best[].period
   * data_chunk_now, frac_done, and result_count
   */
  datasize  = 8*1024*1024;
```

```
  dm_low     = 1;                                          // ~ 0 pc/cm
      ^3
  dm_hi      = 15000;                                      // ~ 100 pc/
     cm^3
  // NOTE: I think the correct conversion is
  // dm = DM * 18.11,
  // so dm_hi = 15000 corresponds to DM = 828 pc / cm^2
  dm_chunk_large  = 128;                                   // DM
     resolution for folding
  dm_chunk_small =  8;
  fft_len    = 32768;                    // Needs to be at least twice
      DM_HI
  max_coadd = 10;
  for (int l = 0; l < max_coadd; ++l) {
    thresh[l] = thresh_def[l];

    // NOTE: none of these initial values except peak_power
    // should ever be used
    best[l].peak_power = -INFINITY;
    best[l].period     = -1.0;
    best[l].ffa_scale   = 0;
    best[l].num_std_devs = -100.0;
  }

  /* Starting place within computation */
  data_chunk_now = 0;
  dm_chunk_large_now   = dm_low;
  dm_chunk_small_now = 0;
  dm_now           = 0;
  dm_sign          = 1;
  sub_buffer     = 0;
  freq             = 0;
  min_freq                = 137;
  fold_buf_loc_long_pos = 0;
  fold_buf_loc_long_neg = 0;
  fold_buf_loc_short_pos   = 0;
  fold_buf_loc_short_neg   = 0;
  frac_done       = 0.0;
  result_count_single   = 0;
  result_count_rep   = 0;
}

/* Client::init
 *
 * This function initializes the Client by performing a large
    number of
 * unrelated actions.  See below for descriptions of these actions
    .
 */
```

```
void Client::init() {
  /* Read the statefile, to see if we have been working on this
     computation before.
   * Initialize the wufile (in.dat)
   * Initialize the outfile (pulse.out)
   */
  state_t &state=ap_shmem->ap_gdata.state;

  if (debug_msg) {
    printf("In ap_client_main.cpp, in Client::init(), at statefile
       .read()\n");
    fprintf(stderr, "In ap_client_main.cpp, in Client::init(), at
       statefile.read()\n");
  }
  statefile.Read();
  if (debug_msg) {
    printf("In ap_client_main.cpp, in Client::init(), at wufile.
       init()\n");
    fprintf(stderr, "In ap_client_main.cpp, in Client::init(), at
       wufile.init()\n");
  }
  wufile.init();
  // We have to parse the header from in.dat
  // to client.wuheader each time, because
  // it is not part of the state, hence it does not get saved.
  wufile.parse_header_for_writeout();
  if (!statefile.resumed) {
    /* if no state file exists, then we have to start from scratch
       .
     * Parse the data header from client.wuheader into client.
       state,
     * then set the starting dm chunk
     */
    wufile.parse_header_to_state();
    state.mtr.srandom(-1);
    state.mtr_new.srandom(-1);
    state.numSamplesBlanked = 0;
    state.numSamplesBlankedNew = 0;
    state.lastDataChunkBlanked = 0;
    state.lastDataChunkBlankedNew = 0;
    // state.idum = -1; // for tracking pseudo-random number
       generator
    // state.idum_new = -1;
    state.dm_chunk_large_now = state.dm_low;
  } else {  // state file exists, so there might be a result file
    // FILE *randsampfile;
    // randsampfile = fopen("randsamp.txt", "a");
    // fprintf(randsampfile, "In Client::init(), recovering state
       file.\n");
```

```
  // fclose(randsampfile);
  if (debug_msg) {
    printf("In ap_client_main.cpp, in Client::init(), reading
       signal_vector\n");
    fprintf(stderr, "In ap_client_main.cpp, in Client::init(),
       reading signal_vector\n");
  }
  read_signal_vector(signal_vector);
}

// if (debug_msg) printf("in ap_client_main.cpp, in Client::init
   (), at outfile.init()\n");
// outfile.init();  Not initing this here anymore

/* Figure out the fold level, which is the exponent in the power
    of 2 that
 * describes the number of consecutive samples to sum, for the
    purpose of
 * the folding algorithm.  (Which detects repeating pulses.)
    The folding
 * algorithm works on groups of samples, not individual samples,
     because
 * there are too many samples to do otherwise.
 * The fold level is the same as the exponent in the power of 2
    that
 * describes the dm_chunk size.  Why?
 */
state.fold_level_large = calculate_fold_level(state.
   dm_chunk_large);
state.fold_level_small = calculate_fold_level(state.
   dm_chunk_small);

/* If the user specified a full_dm on the command line, deduce
 * the various dm components from it.
 */
if(full_dm != -1) {
  single_dm_chunk_large = full_dm - full_dm % state.
     dm_chunk_large;
  single_dm_chunk_small = full_dm % state.dm_chunk_large -
     full_dm % state.dm_chunk_small;
  single_dm = full_dm % state.dm_chunk_small;
}


/* Figure out the fold_buf_size_long, which is the number of
   complex samples in the
 * entire wufile, divided by the fold level's power of 2.  This
    is enough to
```

```
 * describe the entire workunit, because the folding algorithm
    combines samples
 * into groups of 2^state.fold_level.
 * fold_buf_size_short is based on fold_buf_bytes_short, instead
    of on
 * state.datasize, because it would take too long to run the
    full time series
 * on a smaller dm chunk.
 */
state.fold_buf_size_long = (state.datasize*4)>>state.
   fold_level_large;
state.fold_buf_size_short = (state.fold_buf_bytes_short*4)>>
   state.fold_level_small;

/* In Client::init.  Initialize variables, fft plans, etc. */

if (debug_msg) {
  printf("In ap_client_main.cpp, in Client::init(), at science.
     init()\n");
  fprintf(stderr, "In ap_client_main.cpp, in Client::init(), at
     science.init()\n");
}
science.init();

/* Power array needs to hold half of the fft_len time bins that
   will result from the
 * dechirping process.  I'm not sure why only half.  ?????
 */
power.resize(state.fft_len/2);

/* See above for the definition of state.fold_buf_size
 * and allow for folding which can access one element past
    actual data
 */
fold_buf_long_pos.resize(state.fold_buf_size_long + 4);
fold_buf_short_pos.resize(state.fold_buf_size_short + 4);

/* Negative dms */
fold_buf_long_neg.resize(state.fold_buf_size_long + 4);
fold_buf_short_neg.resize(state.fold_buf_size_short + 4);

/* In Client::init.  Read/write files */

if (debug_msg) {
  printf("In ap_client_main.cpp, in Client::init(), at foldfile.
     read()\n");
  fprintf(stderr, "In ap_client_main.cpp, in Client::init(), at
     foldfile.read()\n");
}
```

```
  if (foldfile.Read()) {
    /* Couldn't read fold file => reset to beginning of dm chunk
       */
    state.fold_buf_loc_long_pos = 0;
    state.fold_buf_loc_long_neg = 0;
    state.fold_buf_loc_short_pos = 0;
    state.fold_buf_loc_short_neg = 0;
    state.dm_now        = 0;
    state.dm_sign       = 1;
  }

  if (debug_msg) {
    printf("In ap_client_main.cpp, in Client::init(), at wufile.
       read_raw_data()\n");
    fprintf(stderr, "In ap_client_main.cpp, in Client::init(), at
       wufile.read_raw_data()\n");
  }
  wufile.read_raw_data();
  if (debug_msg) {
    printf("In ap_client_main.cpp, in Client::init(), at wufile.
       finish()\n");
    fprintf(stderr, "In ap_client_main.cpp, in Client::init(), at
       wufile.finish()\n");
  }
  wufile.finish();
  if (debug_msg) {
    printf("In ap_client_main.cpp, in Client::init(), at statefile
       .write()\n");
    fprintf(stderr, "In ap_client_main.cpp, in Client::init(), at
       statefile.write()\n");
  }
  statefile.write();
}

/* time_series_b_t::build
 *
 * Arguments:
 * array: The power array, containing a power level for each of
    fft_len/2 time samples
 *        If we are dealing with the FFA, we instead have a power
    level for each folded & coadded sample.
 * index: m, the index of a power bin which is above threshhold.
 * range: fft_len/2, or smaller (fft_len/(2 >> ell)) if there is a
     coadd ell > 0.
 * n_bins: the number of bins that we add up to get each element
    in the power array.
 *   Since the average noise level is 1 per bin, n_bins serves as
    a baseline or typical value
 *   for the total power.
```

```
 *
 * This function builds a time series of length at most MAXLENGTH,
     which is defined in astropulse.h.
 * A time series consists of a series of power values from the
    array, but generally
 * not the whole array.  The power values are normalized so that
    array[index] has
 * value 255, and all power values can be stored as unsigned chars
    .
 *
 * struct time_series_b_t is defined in astropulse.h
 */
void time_series_b_t::build(float array[], int index, int range,
   float n_bins) {
  int temp_data_val;
  /* Set peak to equal the power that was above threshhold */
  float peak = array[index];
  /* Remove all elements from the data vector, a member of
     time_series_b_t */
  data.clear();
  if (range >= MAXLENGTH) {
    /* Create a time series of length MAXLENGTH, centered at index
       .
     * In some cases, this may be impossible, i.e. if the index is
        very
     * close to the beginning or end of the array.  In this case,
       we
     * do the best we can.
     */
    data.resize(MAXLENGTH);
    index -= MAXLENGTH/2;
    if (index < 0) {
      index = 0;
    } else if (index + MAXLENGTH > range) {
      index = range - MAXLENGTH;
    }
    for (int j = 0; j < MAXLENGTH; ++index, ++j) {
      /* Normalize so the peak has height 255, and n_bins has
         height 63. */
      /* The value of index increments each time, so array[index]
         == peak only when
       * index is set to its original value. */
      temp_data_val = (int) (63 + 192 * (array[index] - n_bins)/(
         peak - n_bins));
      if (temp_data_val < 0) temp_data_val = 0;
      if (temp_data_val > 255) temp_data_val = 255;
      data[j] = (unsigned char) temp_data_val;
    }
    length = MAXLENGTH;
```

```
  } else {
    /* Create a time series out of all the data */
    data.resize(range);
    for (index = 0; index < range; ++index) {
      /* normalize so that the peak has height 255. */
      temp_data_val = (int) (63 + 192 * (array[index] - n_bins)/(
          peak - n_bins));
      if (temp_data_val < 0) temp_data_val = 0;
      if (temp_data_val > 255) temp_data_val = 255;
      data[index] = (unsigned char) temp_data_val;
    }
    length = range;
  }
}

/* time_series_f_t::build
 *
 * This function serves the same purpose as time_series_b_t::build
    , but
 * it creates an array of floats instead of an array of chars.
    Therefore,
 * the power values do not need to be normalized.
 */
void time_series_f_t::build(float array[], int index, int range) {
  /* Remove all elements from the data vector.  No need to store a
      float called
   * <peak> in this function, because the <array> already consists
      of floats.
   */
  for(int i = 0; i < MAXLENGTH; i++) {data[i] = 0;}  // clear data
  if (range >= MAXLENGTH) {
    /* Create a time series out of part of the data */
    index -= MAXLENGTH/2;
    if (index < 0) {
      index = 0;
    } else if (index + MAXLENGTH > range) {
      index = range - MAXLENGTH;
    }
    for(int i = 0; i < MAXLENGTH; i++) { data[i] = array[index + i
        ]; }
    length = MAXLENGTH;
  } else {
    /* Create a time series out of all the data */
    for(int i = 0; i < range; i++) { data[i] = array[index + i]; }
    length = range;
  }
}

/* I need to rewrite this completely.
```

```
 * Currently it doesn't know about the small and large
 * dm chunks.
 */
void state_t::compute_fraction_done(double ell) {
    int dm_chunk_log = int_log2(dm_chunk_large);

    if(code_segment == main_ffa_long) {
        frac_done = log(freq / min_freq) / log((double)2);
        frac_done += sub_buffer;
        frac_done /= num_sub_buffers;
        frac_done *= frac_in_main_ffa[dm_chunk_log] / 2;
        if(dm_sign == 1)
                frac_done += (frac_in_main_ffa[dm_chunk_log] +
                   frac_in_btt[dm_chunk_log])/ 2;
        frac_done += (1 - frac_in_main_ffa[dm_chunk_log] -
           frac_in_btt[dm_chunk_log]/2);
        frac_done += (dm_chunk_large_now - dm_low) /
           dm_chunk_large;
        frac_done /= ((dm_hi - dm_low + dm_chunk_large - 1)/
           dm_chunk_large);
    } else if(code_segment == main_ffa_short ||
              code_segment == build_threshold_table_short) {
        frac_done = log(freq / min_freq) / log((double)2);
        frac_done += sub_buffer;
        frac_done /= num_sub_buffers;
        frac_done *= frac_in_main_ffa_short / 2;
        if(dm_sign == 1) frac_done += frac_in_main_ffa_short / 2;
        frac_done += (1 - frac_in_main_ffa_short);
        frac_done += (dm_chunk_small_now / dm_chunk_small);
        frac_done /= (dm_chunk_large / dm_chunk_small); // % of a
           large dm chunk
        frac_done *= (1 - frac_in_btt[dm_chunk_log] -
           frac_in_main_ffa[dm_chunk_log]);  // we haven't run btt
            or ffa yet, so our current % is too large.
        frac_done += (dm_chunk_large_now-dm_low)/dm_chunk_large;
        frac_done /= ((dm_hi - dm_low + dm_chunk_large - 1)/
           dm_chunk_large);
    } else if(code_segment == build_threshold_table_long) {
        frac_done = (float)nfb / (float)fold_buf_size_long;
        frac_done *= frac_in_btt[dm_chunk_log] / 2;
        if(dm_sign == 1)
                frac_done += (frac_in_main_ffa[dm_chunk_log] +
                   frac_in_btt[dm_chunk_log]) / 2;
        frac_done += (1 - frac_in_btt[dm_chunk_log] -
           frac_in_main_ffa[dm_chunk_log]); // we have already run
            the single pulse finder
        frac_done += (dm_chunk_large_now - dm_low) /
           dm_chunk_large;
```

```
        frac_done /= ((dm_hi - dm_low + dm_chunk_large -1)/
            dm_chunk_large);
    } else {   // In single pulse
        frac_done = ell / (double)max_coadd;  // % coadds complete
        frac_done += (1-dm_sign) / 2;
        frac_done /= 2;   // % of the 2 dm signs
        frac_done += dm_now;
        frac_done /= dm_chunk_small;  // % of a small dm chunk
        frac_done += data_chunk_now/((fft_len/2)/4);
        frac_done /= (datasize/((fft_len/2)/4))-1; // % of data
        frac_done *= (1 - frac_in_main_ffa_short);
        frac_done += (dm_chunk_small_now / dm_chunk_small);
        frac_done /= (dm_chunk_large / dm_chunk_small); // % of a
            large dm chunk
        frac_done *= (1 - frac_in_btt[dm_chunk_log] -
            frac_in_main_ffa[dm_chunk_log]);  // we haven't run btt
             or ffa yet, so our current % is too large.
        frac_done += (dm_chunk_large_now-dm_low)/dm_chunk_large;
        frac_done /= ((dm_hi - dm_low + dm_chunk_large - 1)/
            dm_chunk_large);
    }
}


void Science::mainloop() {
  // Uncomment the following line (atexit) if the program is
     exiting
  // without saying why.  It will segfault when the program exits.
  // atexit(segfault_atexit);

  // FILE* hit_file;
  // hit_file = fopen("hit_file.txt", "w");
  // FILE* power_file = fopen("power_file.txt", "w");  // temp for
      debugging
  float total; // temp for debugging
  int last_hit; // temp for debugging
  int count = 0;  // temp for debugging
  state_t& state = client.ap_shmem->ap_gdata.state;
  time_series_f_t temp_time_series; // Holds time series before it
      gets
                                    // stored in shmem
  if(remove_radar) state.remove_radar = 1;  // override, turn
     remove_radar on

  /* It's not clear why the line below is necessary.  It's
     basically setting a pointer
   * called power, to equal the address of the first element in a
      std::vector<float>
```

```
 * called client.power.  Why can't we just use the vector
    instead of the pointer?
 */
float*    power = &(client.power[0]);
if (debug_msg) {
  printf("state.dm_low: %d\n", state.dm_low);
  fprintf(stderr, "state.dm_low: %d\n", state.dm_low);
}

/* Stuff for making pulse graphs */
long int count_pulses_per_fold_power[MAX_COADD][POWER_MAX] =
   {{0}};
int count_pulses_per_dm[DM_MAX] = {0};
FILE *pulses_per_fold_power_file;
FILE *pulses_per_dm_file;

FILE *best_file;
if (print_best) {
  best_file = boinc_fopen("best_file.txt", "w");
}

timer checkpoint_timer;
timer ffa_timer;

state.code_segment = single_pulse;
// state.print();

int should_print_DC_strengths = 0;

long maxSamplesBlanked = 0;

// This line tests a randomized workunit, to see how strong the
   DC component gets in any N-sample FFT
if(should_print_DC_strengths) print_DC_strengths();

// These lines randomize parts of a workunit
std::vector<long> indices;
int num_indices;
if(state.remove_radar) {
  num_indices = get_indices_to_randomize(indices, 0);
// Now that we know which segments to randomize, we generate
// an average noise envelope out of the RFI-free segments.
// The results are stored in members of the Science class.
  generate_envelope(indices, num_indices);
  generate_pre_envelope();
}

/* mainloop level 1:
```

```
 * This loop iterates over dm chunks.  Each dm_chunk consists of
    , say, 32 dms.
 * The number 32 (or whatever it is) comes from state.dm_chunk,
    which
 * is defined in in.dat
 */
for (;state.dm_chunk_large_now < state.dm_hi;
     state.dm_chunk_large_now += state.dm_chunk_large) {

  client.checkpoint();
  debug_loop(0, 0, debug_loop_msg);

  printf("In ap_client_main.cpp: in mainloop(): at
     dm_chunk_large %d\n", state.dm_chunk_large_now);
  fprintf(stderr, "In ap_client_main.cpp: in mainloop(): at
     dm_chunk_large %d\n", state.dm_chunk_large_now);

  if (single_dm_chunk_large != -1) {
    if (state.dm_chunk_large_now != single_dm_chunk_large)
      continue;
  }
  debug_loop(0, 1, debug_loop_msg);

  debug_loop(0, 2, debug_loop_msg);

  /* Print best pulses so far. [Was this intended to be
     temporary?  What purposes does it serve?] */
  if (print_best) {
    for (int scale = 0; scale < state.max_coadd; scale++)
      fprintf(best_file, "scale %d, peak_power %f\n", scale,
         state.best[scale].peak_power);
    fprintf(best_file, "\n");
    fflush(best_file);
  }
  debug_loop(0, 3, debug_loop_msg);

  /* Create an array of chirps, containing state.dm_chunk
     different chirps,
  * with dms ranging from state.dm_chunk_large_now to
  * state.dm_chunk_large_now + state.dm_chunk - 1
  */
  build_chirp_table(state.dm_chunk_large_now);
  debug_loop(0, 4, debug_loop_msg);

  /* mainloop level 1.a
   * This loop iterates over smaller dm_chunks.
   */
  for (;state.dm_chunk_small_now < state.dm_chunk_large;
       state.dm_chunk_small_now += state.dm_chunk_small) {
```

```
  debug_loop(1, 0, debug_loop_msg);

  if (single_dm_chunk_small != -1) {
    if(state.dm_chunk_small_now != single_dm_chunk_small)
      continue;
  }
  debug_loop(1, 1, debug_loop_msg);

  /* Start over at fractionBlanked = 0. */
  state.numSamplesBlanked = 0;
  state.numSamplesBlankedNew = 0;
  state.lastDataChunkBlanked = 0;
  state.lastDataChunkBlankedNew = 0;

/* THIS SHOULD MOVE BACK UP ONE LEVEL! */
if(pulsegraphs) {
  pulses_per_fold_power_file = boinc_fopen("
     pulses_per_fold_power_file.txt", "w");
  pulses_per_dm_file = boinc_fopen("pulses_per_dm_file.txt", "
     w");

  /* file pointer to beginning of the file.  (Not necessary
     anymore, we open the file just above.)
  fseek(pulses_per_fold_power_file, 0, SEEK_SET);
  fseek(pulses_per_dm_file, 0, SEEK_SET); */
  for(int fold_level = 0; fold_level < MAX_COADD; fold_level
     ++)
    for(int power_level = 0; power_level < POWER_MAX;
       power_level++)
      fprintf(pulses_per_fold_power_file, "%d %d %ld\n",
         fold_level, power_level, count_pulses_per_fold_power[
         fold_level][power_level]);

  for(int dm = 0; dm < DM_MAX; dm++)
    fprintf(pulses_per_dm_file, "%d %d\n", dm,
       count_pulses_per_dm[dm]);

  fflush(pulses_per_fold_power_file);  fclose(
     pulses_per_fold_power_file);
  fflush(pulses_per_dm_file);          fclose(
     pulses_per_dm_file);
}

  /* mainloop level 2
        * This loop iterates over all data chunks.  Each
           data chunk consists of fft_len
        * complex samples, or fft_len / 4 bytes.  The number
           state.data_chunk_now follows the
   * pattern: 0, 4096, 4096 * 2, 4096 * 3, ...
```

```
 *
 * However, there is an overlap of fft_len / 2 complex
    samples, or fft_len / 8
 * bytes.  That is, each time we iterate to the next data
    chunk, we only move
 * ahead by fft_len / 8 bytes of data.
 *
 * This means that the loop runs state.datasize / (state.
    fft_len / 8)
 * times per data chunk.
 *
 * I suspect that the criterion to continue ought to be
 * state._nowdata_chunk_now <= (state.datasize - state.
    fft_len/4),
 * but that the two criteria are equivalent in this case.
 */
for (; state.data_chunk_now<(state.datasize-state.fft_len
   /2/4); state.data_chunk_now+=(state.fft_len/2)/4) {

  client.checkpoint();

  debug_loop(2, 0, debug_loop_msg);

  /*  Should we just skip the data?
  bool good_data_chunk = true;

  if(state.remove_radar) {
    for(int n=0; n<num_indices; n++) {
      if( indices[n] < data_chunk_now + 100000 && indices[n]
          > data_chunk_now - 100000) {
        good_data_chunk = false;
        break;
      }
    }
  }

  if(good_data_chunk == false) continue;
  */

  if (data_byte_limit != -1) {
    if (state.data_chunk_now >= data_byte_limit) {
      // state.mtr = state.mtr_new;  // I don't think this
         is actually necessary here.
      continue;
    }
    // Only consider the first few data chunks, up to the
       one
    // that starts with data_byte_limit bytes.
  }
```

```
debug_loop(2, 1, debug_loop_msg);

/* Print debugging information about the current dm_chunk
   and data_chunk
 * We do this on data chunks 4096 * {0, 1, 2, 4, 8, ...}
    for the first dm of each dm_chunk
 */
if (debug_msg) {
  int i = state.data_chunk_now / ((state.fft_len/2)/4);
  if (i==0) i=1; // Handle the edge case that i==0
  while (i % 2 == 0 && i > 1) {
    i /= 2;
  }
  if (i == 1) {
    printf("In ap_client_main.cpp, Science::mainloop\n");
    fprintf(stderr, "In ap_client_main.cpp, Science::
       mainloop\n");
    printf("state.dm_chunk_large_now: %d, state.
       dm_chunk_small_now: %d, state.data_chunk_now: %d\n"
       , state.dm_chunk_large_now,
            state.dm_chunk_small_now, state.data_chunk_now)
               ;
    fprintf(stderr, "state.dm_chunk_large_now: %d, state.
       dm_chunk_small_now: %d, state.data_chunk_now: %d\n"
       ,
            state.dm_chunk_large_now, state.
               dm_chunk_small_now, state.data_chunk_now);
    main_timer.print_total_time();
    fflush(stdout);
  }
}
debug_loop(2, 2, debug_loop_msg);

debug_loop(2, 3, debug_loop_msg);

/* convert_bits_to_float calls splitter_bits_to_float,
* which is defined in sbtf.cpp.  This function converts
   each bit into
 * a float, resulting in a large array of floats.  The
    number of complex
* samples converted is equal to the length of one FFT.
* If one of the "random indices" is within 100,000 bytes
   of our data_chunk_now, we randomize all of the data.
*/
convert_bits_to_float();

if(state.remove_radar) {
  state.mtr_new = state.mtr; // mtr_new will contain the
     new mtr _after_ this function runs.
```

```
        state.numSamplesBlankedNew = state.numSamplesBlanked;
        state.lastDataChunkBlankedNew = state.
            lastDataChunkBlanked;
        randomize_indices(indices, num_indices, state.
            data_chunk_now, state.fft_len, state);
        maxSamplesBlanked = (maxSamplesBlanked > state.
            numSamplesBlanked)?maxSamplesBlanked:state.
            numSamplesBlanked;
      }

      debug_loop(2, 4, debug_loop_msg);

      debug_loop(2, 5, debug_loop_msg);

      /* FFT the time-domain data, stored in the float array
         client.science.data,
      * into frequency-domain data
      */
      compute_forward_fft();
      debug_loop(2, 6, debug_loop_msg);

      /* Filter the frequency-domain data to get rid of low
         frequency "noise".
       * This line should be removed if at a later time we
         figure out how to get rid of the noise.
       */
      high_pass_filter();
      debug_loop(2, 7, debug_loop_msg);

      /* Graphics stuff */
#ifdef BOINC_APP_GRAPHICS
      double max = 0.0;
      if(!nographics()) {
        rarray.init_data(state.fft_len/2, state.dm_chunk_small);
            // const
      }
#endif

      /* mainloop level 3
        * This loop iterates over dms within a dm chunk.
       */
      for (; state.dm_now<state.dm_chunk_small; state.dm_now++)
         {
        client.checkpoint();
        debug_loop(3, 0, debug_loop_msg);

        if (single_dm != -1) {
          if (state.dm_now != single_dm) continue;
        }
```

```
        debug_loop(3, 1, debug_loop_msg);

        /* Checks if app client is in standalone mode, for
           testing
         * purposes.  (As opposed to being run by the boinc
         * client.)  If so, force checkpoint if over 60 seconds.
         */
        if (boinc_is_standalone()) {
          if(checkpoint_timer.get_total_seconds() > 60) {
            checkpoint_timer.reset();
            client.do_checkpoint();
          }
        }
        debug_loop(3, 2, debug_loop_msg);

        /* mainloop level 4
         * This loop iterates over the two dm signs, positive
           and negative
         */
        for (; state.dm_sign>=-1; state.dm_sign-=2) {
          client.checkpoint();
          debug_loop(4, 0, debug_loop_msg);

          debug_loop(4, 1, debug_loop_msg);

          /* Dechirp the frequency domain data via
             multiplication by a dechirping
           * function.  Then perform the inverse fourier
             transform
           */
          dechirp(state.dm_chunk_small_now + state.dm_now, power
             , state.dm_sign);
          debug_loop(4, 2, debug_loop_msg);

          /* Graphics stuff */
#ifdef BOINC_APP_GRAPHICS
          if (!nographics() && state.dm_sign > 0) { // add
             positive dispersions to graph
            rarray.add_source_row(power);
            memcpy(&client.ap_shmem->rarray_data, &rarray,
               sizeof(REDUCED_ARRAY_DATA));
          }
#endif

          /* measuring the time that has passed since the
             program started running */
          updatecputime();
          debug_loop(4, 3, debug_loop_msg);
```

```
/* mainloop level 5 * This loop iterates over coadd
   levels.
 */
for (int l=0; l<state.max_coadd; l++) {
  debug_loop(5, 0, debug_loop_msg);

  /* Uses ell, along with some members of client.state
   * Stores the result in state.frac_done.
   * The sole purpose of this computation is to print
     the fraction
   * done on the screen.
   */
  state.compute_fraction_done(l);
  debug_loop(5, 1, debug_loop_msg);

  /* boinc API boinc_fraction_done tells BOINC how
     much
   * of the current workunit has been completed.  This
     is
   * used to inform the core client GUI of the %
     complete.
   */
  boinc_fraction_done(state.frac_done);
  boinc_ops_cumulative(state.frac_done*FLOPS_PER_DM*(
     state.dm_hi-state.dm_low)*log((float)state.
     fft_len)/log(32768.0),0);
  debug_loop(5, 2, debug_loop_msg);

  /* This is the size of the power array.
   * It starts at state.fft_len / 2, for ell = 0.  As
     ell gets larger,
   * we perform coadds in place, and this reduces the
     effective
   * size of the power array.
   * Note that from the very beginning, we ignore the
     second
   * half of the power bins.  Due to the overlap in
     ffts,
   * this second half is not necessary.
   */
  int dechirped_range_data_length = state.fft_len >> (
     l+1);
  debug_loop(5, 3, debug_loop_msg);

  /* mainloop level 6
  * This loop iterates over all of the power bins,
     searching for
  * a bin that exceeds the threshhold power.
  */
```

```
                last_hit = -10;
                double cur_cpu_time;
                boinc_wu_cpu_time(cur_cpu_time);
                client.ap_shmem->ap_gdata.cpu_time = cur_cpu_time;
                for (int m=0; m<dechirped_range_data_length; m++) {

                  // fprintf(power_file, "m = %d, power[m] = %f,
                     state.data_chunk_now = %d\n", m, power[m],
                     state.data_chunk_now);  // temp for debugging
                  if (state.dm_sign == 1 && l == 0 && m <
                     dechirped_range_data_length - 10) {
                    /* This was for debugging.  Not sure for what
                       exactly.
                    if(m > last_hit + 10 && detect_hit(&power[m],
                       total)) {
                      count++;
                      fprintf(hit_file, "m = %d, sample = %d, total
                         = %f, num = %d\n", m, m + state.
                         data_chunk_now * 4, total, count);
                      last_hit = m;
                    } */
                  }
                  /* Graphics stuff */
#if BOINC_APP_GRAPHICS
                  if (power[m] > max && !nographics()) {
                    temp_time_series.build(power, m,
                       dechirped_range_data_length);
                    memcpy(&client.ap_shmem->time_series_shmem, &
                       temp_time_series, sizeof(time_series_f_t));
                    max = power[m];
                  }
#endif

                  if(pulsegraphs) {
                  /* For plotting pulse graph of # hits vs. power */
                    int cur_power = (int)power[m];
                    if(cur_power >= POWER_MAX || l >= MAX_COADD) {
                      fprintf(stderr, "Error: power %d, coadd %d is
                         overflow\n", cur_power, l);
                      exit(0);
                    }
                    count_pulses_per_fold_power[l][cur_power]++;

                  /* For plotting a graph of # hits vs. dm */
                    if (l == 0 && state.dm_sign == 1 && power[m] >
                       10.0) {
                      int cur_dm = state.dm_chunk_large_now + state.
                         dm_chunk_small_now + state.dm_now;
                      if(cur_dm >= DM_MAX) {
```

```
        fprintf(stderr, "Error: dm %d is overflow\n"
            , cur_dm);
        exit(0);
      }
      count_pulses_per_dm[cur_dm]++;
    }
  }
  if (power[m] > state.thresh[l]) {

  /* for debugging: check that the client randomizes
      these bits
     if(state.data_chunk_now == 1359872 - 4096 && l
        == 3) {
       for(int i = 16470; i < 16500; i++)
         printf("data[%d][0] = %f, data[%d][1] = %f
             \n",
             i, data[i][0], i, data[i][1]);
     }
   */

  /* If power exceeds threshhold, record a pulse.
   * This involves first creating an ap_signal,
      which is defined in
   * astropulse/server/db/ap_schema.h.  Then, we
      write the signal
   * to pulse.out using Outfile::output_result.
   */

    if(pulse_limit_single && state.
       result_count_single >= pulse_limit_single)
       // We've already found enough single pulses.
        (Must be that we haven't found enough
       repeating pulses.)  Don't record this pulse.
        continue;
    ap_signal s;
    time_series_b_t time_series;
    time_series.build(power, m,
       dechirped_range_data_length, pow_int(2, l));
    state.pulse_jd = state.jd0 + ((float)state.
       data_chunk_now)/((float)state.datasize)*(
       state.jd1-state.jd0);

    s.dm= state.dm_sign*(state.dm_chunk_large_now+
       state.dm_chunk_small_now + state.dm_now); //
       dm
    s.scale=l;                          // scale
    s.peak_power=power[m];              // peak_power
    s.fft_num=state.data_chunk_now*4;
```

```
    s.peak_bin=state.data_chunk_now * 4 + (m << l);
       // index
    s.time=state.pulse_jd;          // time
    s.period=0.0;                   // period
    s.ffa_scale=0;                   // ffa bin size
    s.num_std_devs=-100.0;                  // single
       pulse has no num_std_devs
    // Interpolate betw. ra0 & ra1 using state.
       pulse_jd
    s.ra = interpolate_ra(state.jd0, state.jd1,
       state.pulse_jd, state.ra0, state.ra1);
    // Interpolate betw. dec0 and dec1
    s.decl = interpolate_dec(state.jd0, state.jd1,
       state.pulse_jd, state.dec0, state.dec1);
    s.time_series_len=time_series.length;
    s.time_series=time_series.data; // WARNING why
       can we use an equal sign here?
    s.time_series.encoding=_x_hex;
    if(add_signal(client.signal_vector, s)) state.
       result_count_single++;
    // client.outfile.output_result(s);

    if(pulse_limit_single && state.
       result_count_single >= pulse_limit_single &&
       pulse_limit_rep && state.result_count_rep >=
       pulse_limit_rep) {
      // Quit early
      fprintf(stderr, "Found %d single pulses and %d
          repeating pulses, exiting.\n", state.
         result_count_single, state.result_count_rep
         );
      printf("Found %d single pulses and %d
         repeating pulses, exiting.\n", state.
         result_count_single, state.result_count_rep
         );
      fflush(stdout);
      client.outfile.write_main(client.signal_vector
         );
      client.outfile.finish_results();
      client.finish();
      main_timer.stop();
      boinc_finish(0);
      exit(0);
    }
}

/* We record the best peak at each coadd level.
   That is, there is
```

```
     * one "best" value for each coadd level, for the
        entire run of
     * the client.
     */
    if (power[m] > state.best[l].peak_power) {
      time_series_b_t time_series;
      time_series.build(power, m,
         dechirped_range_data_length, pow_int(2, l));
      state.pulse_jd = state.jd0 + (state.
         data_chunk_now)/(state.datasize)*(state.jd1-
         state.jd0);
      state.best[l].fft_num=state.data_chunk_now*4;
      state.best[l].peak_bin = state.data_chunk_now *
         4 + (m << l);
      state.best[l].peak_power = power[m];
      state.best[l].scale = l;
      state.best[l].dm = state.dm_sign*(state.
         dm_chunk_large_now+state.dm_chunk_small_now+
         state.dm_now);
      state.best[l].period = 0.0;
      state.best[l].ffa_scale = 0;
      state.best[l].num_std_devs = -100.0;
      state.best[l].ra = interpolate_ra(state.jd0,
         state.jd1, state.pulse_jd, state.ra0, state.
         ra1);
      state.best[l].decl = interpolate_dec(state.jd0,
         state.jd1, state.pulse_jd, state.dec0, state.
         dec1);
      state.best[l].time = state.pulse_jd;
      state.best[l].time_series_len = time_series.
         length;
      assert(time_series.length <= MAXLENGTH_OF_TS);
         // Don't want an overflow
      // This memcpy works because state_signal.
         time_series is an array, and so is
      // time_series.data.  (ap_signal.time_series
         would be an sqlblob.)
      memcpy(state.best[l].time_series, &(time_series.
         data[0]), time_series.length);
    }
  } /* End mainloop level 6, bin number (m) */
  debug_loop(5, 10, debug_loop_msg);

  /* When l is at two specific values (called state.
     fold_level_large and
   * state.fold_level small), we are ready to
      contribute data to the fast
   * folding algorithm.
   *
```

```
 * Large scale version: This happens only once per (
    dm_chunk_large, data_chunk) pair.
 * So we only hit one out of every 2^(state.
    fold_level_large)
 * dms in this way (henceforth 2^(s.fll)). This is
    why we do it at a fold_level
 * such that 2^(s.fll) is equal to state.
    dm_chunk_large: it's not meaningful to look
 * at data at a better time resolution than that
    provided by the dm resolution.
 *
 * We also look at a finer resolution (below),
    namely 2^(state.fold_level_small).
 * In this case, we contribute data more often, and
    there is more of it. To compensate for this,
 * we must cut off the power array at some number of
     bytes, state.fold_buf_bytes_short.
 */
// Temporary fix
if ((l==state.fold_level_large) && (state.dm_now ==
   fold_dm) && (state.dm_chunk_small_now ==
   fold_dm_chunk_small)) {
  /* Save for folding */

  // positive dms
  if(state.dm_sign == 1) {
    memcpy(&client.fold_buf_long_pos[state.
       fold_buf_loc_long_pos], power,
          sizeof(float)*(state.fft_len>>(l+1)));
    state.fold_buf_loc_long_pos += (state.fft_len>>(
       l+1));
    // On the last data chunk, do one extra memcpy.
    // This is because the number of iterations in
       the data chunk loop is one less than enough
    // to make the long fold buffer fill up, due to
       the overlap between data chunks.
    // Note that datasize is measured in bytes, and
       increments by fft_len/8.
    if(state.data_chunk_now == (state.datasize -
       state.fft_len/4)) {
      memcpy(&client.fold_buf_long_pos[state.
         fold_buf_loc_long_pos], power,
            sizeof(float)*(state.fft_len>>(l+1)));
      state.fold_buf_loc_long_pos += (state.fft_len
         >>(l+1));
    }

  // identical code, but for negative dms
  } else if(state.dm_sign == -1) {
```

```
      memcpy(&client.fold_buf_long_neg[state.
          fold_buf_loc_long_neg], power,
              sizeof(float)*(state.fft_len>>(l+1)));
      state.fold_buf_loc_long_neg += (state.fft_len>>(
          l+1));
      if(state.data_chunk_now == (state.datasize -
          state.fft_len/4)) {
        memcpy(&client.fold_buf_long_neg[state.
            fold_buf_loc_long_neg], power,
              sizeof(float)*(state.fft_len>>(l+1)));
        state.fold_buf_loc_long_neg += (state.fft_len
            >>(l+1));
      }
    } else {
      fprintf(stderr, "Error in ap_client_main.cpp:
          bad dm_sign value");
      exit(-1);
    }
  }
  debug_loop(5, 11, debug_loop_msg);

  /* More frequently, we copy the power array into the
      short fold buffer.
   * Note that the condition  state.data_chunk_now <
      state.fold_buf_bytes_short,
   * even though each data chunk extends beyond the
      start of the next one.
   */
  if ((l==state.fold_level_small) && (state.dm_now ==
     fold_dm)
                              && (state.data_chunk_now <
                                  state.
                                 fold_buf_bytes_short))
                                 {
        /* Save for folding in short time series, for
            finding short period repeating pulses. */
    if(state.dm_sign == 1) {
      memcpy(&client.fold_buf_short_pos[state.
          fold_buf_loc_short_pos], power,
            sizeof(float)*(state.fft_len>>(l+1)));
      state.fold_buf_loc_short_pos += (state.fft_len
          >>(l+1));
    } else if(state.dm_sign == -1) {
      memcpy(&client.fold_buf_short_neg[state.
          fold_buf_loc_short_neg], power,
              sizeof(float)*(state.fft_len>>(l+1)));
      state.fold_buf_loc_short_neg += (state.fft_len
          >>(l+1));
    }
```

```
        }
        debug_loop(5, 12, debug_loop_msg);

        /* coadd_in_place is defined in ap_science.cpp
         * Its purpose is to halve the size of the power
            array by adding
         * array elements in consecutive pairs.
         * The second argument is the initial array size.
            Note that
         *
         */
        coadd_in_place(power, state.fft_len>>(l+1));
        debug_loop(5, 13, debug_loop_msg);

      } /* End mainloop level 5, coadd level (l) */
      debug_loop(4, 10, debug_loop_msg);

      // client.checkpoint(); Can only checkpoint at
         beginning of for loop
      debug_loop(4, 11, debug_loop_msg);

    } /* End mainloop level 4, dm sign (s) */
    debug_loop(3, 10, debug_loop_msg);

    state.dm_sign = 1;
    debug_loop(3, 11, debug_loop_msg);

  } /* End mainloop level 3, inner dm (k) */
  debug_loop(2, 10, debug_loop_msg);

  state.dm_now = 0;
  debug_loop(2, 11, debug_loop_msg);

  // Now that we've finished that data_chunk,
  // we change over the mtr.
  // * If we crash after a later checkpoint takes effect,
  //    state.dm_now will equal 0 and our
  //    next run of randomize_indices will (correctly) happen
  //    with the newmtr
  // * If we crash after this, but before any checkpoint
     takes effect,
  //    then the state.dm_now = 0 won't have taken effect,
  //    so we'll (correctly) start again with the old mtr.
  if(state.remove_radar) {
    state.mtr = state.mtr_new;
    state.numSamplesBlanked = state.numSamplesBlankedNew;
    state.lastDataChunkBlanked = state.
       lastDataChunkBlankedNew;
  }
```

```
  } /* End mainloop level 2, data (j) */

  debug_loop(1, 10, debug_loop_msg);

  if (state.fold_buf_loc_short_pos != state.
     fold_buf_size_short) {
      fprintf(stderr, "Short positive fold buffer didn't fill
         up (lol=%d, size=%d).\n",
              state.fold_buf_loc_short_pos, state.
                 fold_buf_size_short);
  }
  if (state.fold_buf_loc_short_neg != state.
     fold_buf_size_short) {
      fprintf(stderr, "Short negative fold buffer didn't fill
         up (lol=%d, size=%d).\n",
              state.fold_buf_loc_short_neg, state.
                 fold_buf_size_short);
  }

  ffa_timer.start();
  debug_loop(1, 11, debug_loop_msg);

  /* Arguments are: (pointer to beginning of array, ?, are we
     on large dm
   * chunk, ?, ?, max # of pulses before we exit the ffa
   */
  if(!skipffa_short) {
state.dm_sign = -1;
    state.code_segment = build_threshold_table_short;
    ffa(&client.fold_buf_short_neg[0], 1, false,
       write_lcgf_short, debug_msg, pulse_limit_single,
       pulse_limit_rep, ffa_thresh_mult);
state.dm_sign = 1;
    state.code_segment = build_threshold_table_short;
    ffa(&client.fold_buf_short_pos[0], 1, false,
       write_lcgf_short, debug_msg, pulse_limit_single,
       pulse_limit_rep, ffa_thresh_mult);
    state.code_segment = single_pulse;
  }
  debug_loop(1, 12, debug_loop_msg);

  ffa_timer.stop();
  debug_loop(1, 13, debug_loop_msg);

  state.data_chunk_now = 0; // I moved this line down in case
     we crash during ffa

  debug_loop(1, 14, debug_loop_msg);
```

```
  state.fold_buf_loc_short_pos = 0;
  state.fold_buf_loc_short_neg = 0;
  debug_loop(1, 15, debug_loop_msg);


} /* End mainloop level 1.a, dm_chunk_small */
debug_loop(0, 10, debug_loop_msg);

/* When we've finished the data loop, we will have hopefully
 * filled up the fold buffer, by successively copying the
    power array
 * into it.  We are then ready to reset the fold buffer to the
     beginning
 * of the buffer, and reset the data chunk to the first data
    chunk.
 */
if (state.fold_buf_loc_long_pos != state.fold_buf_size_long) {
  fprintf(stderr, "Long pos fold buffer didn't fill up (lol=%d
     , size=%d).\n",
       state.fold_buf_loc_long_pos, state.fold_buf_size_long);
}
if (state.fold_buf_loc_long_neg != state.fold_buf_size_long) {
  fprintf(stderr, "Long neg fold buffer didn't fill up (lol=%d
     , size=%d).\n",
       state.fold_buf_loc_long_neg, state.fold_buf_size_long);
}
debug_loop(0, 11, debug_loop_msg);

/* do fold - results output within ffa */
// fclose(hit_file);  // temp for debugging
ffa_timer.start();
debug_loop(0, 12, debug_loop_msg);

/* Arguments are: (pointer to beginning of array, ?, are we on
    large dm
 * chunk, ?, ?, max # of pulses before we exit the ffa
 */
if(!skipffa_long) {
  state.code_segment = build_threshold_table_long;
  state.dm_sign = -1;
  ffa(&client.fold_buf_long_neg[0], 1, true, write_lcgf_long,
     debug_msg, pulse_limit_single, pulse_limit_rep,
     ffa_thresh_mult);
  state.dm_sign = 1;
  // state.code_segment = single_pulse;
  state.code_segment = build_threshold_table_long;
  ffa(&client.fold_buf_long_pos[0], 1, true, write_lcgf_long,
     debug_msg, pulse_limit_single, pulse_limit_rep,
     ffa_thresh_mult);
```

```
      state.code_segment = single_pulse;
    }
    debug_loop(0, 13, debug_loop_msg);

    ffa_timer.stop();
    debug_loop(0, 14, debug_loop_msg);

    state.dm_chunk_small_now = 0;  // I moved this line down in
       case we crash during ffa
    debug_loop(0, 15, debug_loop_msg);

    state.fold_buf_loc_long_pos = 0;
    state.fold_buf_loc_long_neg = 0;
    debug_loop(0, 16, debug_loop_msg);


  } /* End mainloop level 1, DM chunk (i) */

  /* output the top result no matter how many we have >= threshold
     , so
   * that if none >= threshold we can still validate redundant
     results
   * against each other.
   */
  /* Was used for debugging before the ap_timer class was invented
  FILE* progress_file = fopen("progress_file.txt", "a");
  fprintf(progress_file, "Total time: %d\n", time(0) - starttime);
  fprintf(progress_file, "Total cpu time: %f\n", cputime);
  fprintf(progress_file, "Max cpu ticks interval: %d\n",
     maxcputicksinterval);
  fclose(progress_file);
  */

  /* Write the best pulses, and various other xml tags. */
  client.outfile.write_main(client.signal_vector);
  client.outfile.finish_results();
  if (debug_msg) {
    printf("FFA timer reads: %ld seconds, %ld ticks\n", ffa_timer.
       get_seconds(), ffa_timer.get_ticks());
    fprintf(stderr, "FFA timer reads: %ld seconds, %ld ticks\n",
       ffa_timer.get_seconds(), ffa_timer.get_ticks());
  }
} /* End mainloop */

/* Client::finish
 *
 * Calls all finish functions.
 */
void Client::finish() {
```

```
  // Clean up stuff and exit
  // unloadGl();
  science.finish();
  // outfile.finish();
  statefile.finish();
  foldfile.finish();
}

/* debug_loop
 * int level: the level of the loop hierarchy that we are
    currently in.
 * bool debug_loop_msg: true if we should print out these messages
 *
 * debug_loop
 */
void debug_loop(int level, int number, bool debug_loop_msg) {
  if(debug_loop_msg) {

    int max = 2;
    const int num_levels = 10;
    const int after_inner_loop = 10;
    static int loopnum[num_levels] = {0, 0, 0, 0, 0, 0, 0, 0, 0,
       0};
    bool print_msg = true;

    // If we've just finished an inner loop,
    // reset all levels deeper than this one
    if(number == after_inner_loop) {
      for(int i = level + 1; i < 10; i++) {
        loopnum[i] = 0;
      }
    }

    // If we're at the beginning of the loop,
    // increment current level's loop count
    if(number == 0) {
      loopnum[level]++;
    }

    // If all loop counts are less than or equal to max, then
       print the message
    print_msg = true;
    for(int i = 0; i < num_levels; i++) {
      if(loopnum[i] > max) {print_msg = false;}
    }

    if(print_msg) {
      for(int i = 0; i < level; i++)
        fprintf(stderr, "* ");
```

```
      fprintf(stderr, "In ap_client_main.cpp, Science::mainloop,
         loop level %d, number %d\n", level, number);
    }
  }
}
/*
void state_t::print() {
  fprintf(stderr, "\n****************state file
     ********************\n");
  fprintf(stderr, "char recvname[64]: %s\n", recvname);
  fprintf(stderr, "char wuname[64]: %s\n", wuname);
  fprintf(stderr, "int datasize: %d\n", datasize);
  fprintf(stderr, "int dm_low: %d\n", dm_low);
  fprintf(stderr, "int dm_hi: %d\n", dm_hi);
  fprintf(stderr, "int dm_chunk_large: %d\n", dm_chunk_large);
  fprintf(stderr, "int dm_chunk_small: %d\n", dm_chunk_small);
  fprintf(stderr, "int fft_len: %d\n", fft_len);
  fprintf(stderr, "float high_pass: %f\n", high_pass);
  fprintf(stderr, "float ffa_threshold: %f\n", ffa_threshold);
  fprintf(stderr, "int max_coadd: %d\n", max_coadd);
  fprintf(stderr, "int fold_buf_size_short: %d\n",
     fold_buf_size_short);
  fprintf(stderr, "int fold_buf_size_long: %d\n",
     fold_buf_size_long);
  fprintf(stderr, "int fold_buf_bytes_short: %d\n",
     fold_buf_bytes_short);
  fprintf(stderr, "int fold_level_large: %d\n", fold_level_large);
  fprintf(stderr, "int fold_level_small: %d\n", fold_level_small);
  for(int i = 0; i < MAX_COADD; i++)
    fprintf(stderr, "double thresh[%d] = %f\n", i, thresh[i]);
  fprintf(stderr, "double ra0: %f\n", ra0);
  fprintf(stderr, "double ra1: %f\n", ra1);
  fprintf(stderr, "double dec0: %f\n", dec0);
  fprintf(stderr, "double dec1: %f\n", dec1);
  fprintf(stderr, "double jd0: %f\n", jd0);
  fprintf(stderr, "double jd1: %f\n", jd1);
  fprintf(stderr, "int data_chunk_now: %d\n", data_chunk_now);
  fprintf(stderr, "int dm_chunk_large_now: %d\n",
     dm_chunk_large_now);
  fprintf(stderr, "int dm_chunk_small_now: %d\n",
     dm_chunk_small_now);
  fprintf(stderr, "int dm_now: %d\n", dm_now);
  fprintf(stderr, "int dm_sign: %d\n", dm_sign);
  fprintf(stderr, "int num_sub_buffers: %d\n", num_sub_buffers);
  fprintf(stderr, "int sub_buffer: %d\n", sub_buffer);
  fprintf(stderr, "float freq: %f\n", freq);
  fprintf(stderr, "float min_freq: %f\n", min_freq);
  fprintf(stderr, "int nfb: %d\n", nfb);
```

```
    fprintf(stderr, "int fold_buf_loc_long_pos: %d\n",
        fold_buf_loc_long_pos);
    fprintf(stderr, "int fold_buf_loc_long_neg: %d\n",
        fold_buf_loc_long_neg);
    fprintf(stderr, "int fold_buf_loc_short_pos: %d\n",
        fold_buf_loc_short_pos);
    fprintf(stderr, "int fold_buf_loc_short_neg: %d\n",
        fold_buf_loc_short_neg);
    fprintf(stderr, "double frac_done: %f\n", frac_done);
    fprintf(stderr, "double pulse_jd: %f\n", pulse_jd);
    fprintf(stderr, "int result_count_single: %d\n",
        result_count_single);
    fprintf(stderr, "int result_count_rep: %d\n", result_count_rep);
    fprintf(stderr, "enum code_segment: %d\n", (int) code_segment);
    for(int j = 0; j < MAX_COADD; j++) {

    }
    fprintf(stderr, "\n*****************************************\
        n");
}
*/

} // namespace Astropulse

double interpolate_ra(double jd0, double jd1, double pulse_jd,
    double ra0, double ra1) {
  assert(jd1 > jd0);
  double time_fraction = (pulse_jd - jd0) / (jd1 - jd0);
  double retval;
  if (fabs(ra0 - ra1) < 12) {
    return ra0 + (ra1 - ra0) * time_fraction;
  } else {  // We've crossed RA 0 -- RA 24.
    if (ra0 > ra1) // crossed from high RA to low RA
      retval = (ra0 - 24) +
               (ra1 - (ra0 - 24)) * time_fraction;
    else // crossed from low RA to high RA
      retval = ra0 +
               ((ra1 - 24) - ra0) * time_fraction;
    if (retval > 0) return retval;
    else return retval + 24;
  }
}

double interpolate_dec(double jd0, double jd1, double pulse_jd,
    double dec0, double dec1) {
  assert(jd1 > jd0);
  double time_fraction = (pulse_jd - jd0) / (jd1 - jd0);
  return (dec0 + (dec1 - dec0) * time_fraction);
}
```

B.2 ap_science.cpp

The file ap_science.cpp contains functions related to the coherent dedispersion algorithm itself, including the calculation of the chirp.

```
// Copyright 2003 Regents of the University of California

// Astropulse is free software; you can redistribute it and/or
   modify it under
// the terms of the GNU General Public License as published by the
    Free
// Software Foundation; either version 2, or (at your option) any
   later
// version.

// Astropulse is distributed in the hope that it will be useful,
   but WITHOUT
// ANY WARRANTY; without even the implied warranty of
   MERCHANTABILITY or
// FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public
   License for
// more details.

// You should have received a copy of the GNU General Public
   License along
// with Astropulse; see the file COPYING.  If not, write to the
   Free Software
// Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA
   02111-1307, USA.

/* ap_science.C */
/* Functions for AstroPulse scientific computations */

#ifndef _WIN32
#include "ap_config.h"
#else
#include "boinc_win.h"
#endif

#include <cmath>
#include <cassert>
#include "astropulse.h"
#include "boinc_api.h"
#include "sbtf.h"
#include "ap_debug.h"
#include "fftw3.h"

/* Windows is stupid */
#ifndef M_PI
```

```
#define M_PI 3.14159265358979323846
#endif

namespace Astropulse {
const char* WISDOM_FNAME =  "wisdom.dat";

inline void Science::Wisdom::load() {
  FILE *wisdom;
  if (wisdom=boinc_fopen(WISDOM_FNAME,"r")) {
    char *wiz=(char *)calloc(1024,64);
    int n=0;
    while (wiz && n<64*1024 && !feof(wisdom)) {
      n+=fread(wiz+n,1,80,wisdom);
    }
    fftwf_import_wisdom_from_string(wiz);
    free(wiz);
    fclose(wisdom);
  }
}

inline void Science::Wisdom::save() {
  // Save new wisdom
  FILE *wisdom=boinc_fopen(WISDOM_FNAME,"w");
  if (wisdom) {
    char *wiz=fftwf_export_wisdom_to_string();
    if (wiz) {
      fwrite(wiz,strlen(wiz),1,wisdom);
    }
    fclose(wisdom);
  }
}

// Allocate memory, and create fft plans
void Science::init() {
  const size_t& fft_len = client.ap_shmem->ap_gdata.state.fft_len;
  int fft_len2 = 1024;
  fftlen_envelope = 128;
  fftlen_patch = 512;

  data  = (fftwf_complex *)fftwf_malloc(sizeof(fftwf_complex)*
     fft_len);
  temp  = (fftwf_complex *)fftwf_malloc(sizeof(fftwf_complex)*
     fft_len);
  chirp  = (fftwf_complex *)fftwf_malloc(sizeof(fftwf_complex)*
     fft_len*client.ap_shmem->ap_gdata.state.dm_chunk_large);
  patch = (fftwf_complex *)fftwf_malloc(sizeof(fftwf_complex)*
     fftlen_patch);
  envelope_part = (fftwf_complex *) fftwf_malloc(sizeof(
     fftwf_complex)*fftlen_envelope);
```

```
  envelope = new double[fftlen_envelope];
  pre_envelope = new double[fftlen_envelope];

  client.status = "Creating FFT plans.";

  wisdom.load();

  client.status = "Creating forward FFT plan.";

  fp = fftwf_plan_dft_1d(fft_len, data, data, FFTW_FORWARD,
    FFTW_MEASURE);
  fp2 = fftwf_plan_dft_1d(fft_len2, data, data, FFTW_FORWARD,
    FFTW_MEASURE);
  fp_patch_inv = fftwf_plan_dft_1d(fftlen_patch, patch, patch,
    FFTW_BACKWARD, FFTW_MEASURE);
  fp_envelope = fftwf_plan_dft_1d(fftlen_envelope, envelope_part,
    envelope_part, FFTW_FORWARD, FFTW_MEASURE);


  client.status = "Creating inverse FFT plan.";
  ip = fftwf_plan_dft_1d(fft_len, temp, temp, FFTW_BACKWARD,
    FFTW_MEASURE);

  wisdom.save();

}

void Science::finish() {
  fftwf_destroy_plan(fp);
  fftwf_destroy_plan(ip);
  fftwf_destroy_plan(fp_patch_inv);

  fftwf_free(chirp);
  fftwf_free(temp);
  fftwf_free(data);
  fftwf_free(patch);
}

/* Science::build_chirp_table
 *
 * arguments:
 * int dm_start: the first dm to use in the chirp table.
 *  The chirp table will use all dms from dm_start to
 * dm_start + client.ap_shmem->ap_gdata.state.dm_chunk - 1.
 *
 * This function creates an array of complex numbers
 * (of type fftw_complex) which
```

```
 * represent a chirp.  The array indices represent frequencies, so
 * the chirp is described in frequency space.  We will multiply
    the
 * chirp by the frequency-space representation of the incoming
    signal
 * in order to dechirp the signal, so it is really a "dechirp"
    table.
 * That is, we use the reciprocal of the chirp, instead of the
    chirp
 * itself.  Since the chirp's amplitude is 1 everywhere, taking
    its
 * reciprocal is a simple matter of taking its complex conjugate.
 *
 * The first <state.fft_len> complex numbers correspond to the
 * dm indicated by dm_start.  The next set of numbers correspond
    to
 * dm_start+1, and so on until dm_start + client.ap_shmem->
    ap_gdata.state.dm_chunk - 1.
 */
void Science::build_chirp_table(int dm_start) {
  int i,j;
  double freq;
  double phase;

  client.status = "Building chirp table.";

  state_t& state = client.ap_shmem->ap_gdata.state;

  /* For debugging, see below
  static int idum = -1;
  */

  // phase function determined by method of steepest descent.
  // 0.00176 = 2.5 MHz / 1.42 GHz.  When multiplied by the maximum
  // value freq = 0.5, it will give 1.25 MHz / 1.42 GHz.
  for (i=0; i<state.dm_chunk_large; i++) {
    for (j=0; j<state.fft_len; j++) {
      // To get exact chirp function, we've multiplied the phase
      // by 1 / (1 + 0.00176 * freq) from the linear chirp
         function
      // This is equal to nu0 / nu, where nu0 = 1.42 GHz and
      // nu = 1.42 GHz + freq in Hz = 1.42 GHz + (freq / (4 *
         10^-7)) Hz,
      // resulting in the value 0.00176 = 1 / (1.42 * 10^9 * 4 *
         10^-7)
      freq = (double)j/(double)state.fft_len;
      freq = (j<state.fft_len/2)?freq:freq-1.0;
      phase = M_PI*(double)(dm_start+i)*freq*freq*(1 / (1 +
         0.00176 * freq));
```

```
      /* For debugging: use a random phase instead of a chirp
      phase = 2 * M_PI * ran1(idum);
      */

      chirp[i*state.fft_len + j][0] = (float)cos(phase);
      chirp[i*state.fft_len + j][1] = (float)sin(phase);
    }
  }

}

void Science::convert_bits_to_float() {
  client.status = "Converting raw data.";
  if (splitter_bits_to_float(&client.rawdata[client.ap_shmem->
     ap_gdata.state.data_chunk_now],
                             (float *)data, client.ap_shmem->
                               ap_gdata.state.fft_len)) {
    exit(SBTF_ERR);
  }
}

void Science::compute_forward_fft() {
  client.status = "Computing forward FFT.";
  fftwf_execute(fp);
}

void Science::high_pass_filter() {
  /* One frequency bin is 2.5 MHz / fft_len worth of frequency.
   * So the width of the Gaussian filter in bins is:
   * (Note the sample rate is stored in the recorder_config, which
       is not part of the
   *  client.ap_shmem->ap_gdata.state.)
   */
  const size_t& fft_len = client.ap_shmem->ap_gdata.state.fft_len;
  if (client.ap_shmem->ap_gdata.state.high_pass > 0) {
    float sigma = client.ap_shmem->ap_gdata.state.high_pass * ((
       float) fft_len / 2500000);
    int i;
    int nf;  // For negative frequencies
    float Gaussian, multiplier;
    static float normalize = 1 / sqrt(1 - sqrt(2 * 3.1415926) *
       sigma / fft_len);

    for (i = 0; i < (int) fft_len; i++) {
      if (i < fft_len/2) {
        Gaussian = exp(-i * i / (2 * sigma * sigma));
      } else {
        nf = fft_len - i;
```

```
        Gaussian = exp(-nf * nf / (2 * sigma * sigma));
      }
      /* Multiply the power by 1 - Gaussian. */
      multiplier = sqrt(1 - Gaussian);
      /* Conserve total noise power. */
      multiplier *= normalize;
      data[i][0] *= multiplier;
      data[i][1] *= multiplier;
    }
  }
}

/* Science::dechirp
 * Now DIVIDES by the chirp function, instead of multiplying.
 *
 * Arguments:
 * dm: dispersion measure to dechirp by, in units of 0.4
     microseconds
 * power: array of length fft_len/2, to hold the power in each
     time bin
 *        I'm not sure why we only record the first fft_len/2 bins
     .
 * sign: +1 for a positive dm, -1 for a negative dm
 */
void Science::dechirp(int dm, float *power, int sign) {
  const size_t& fft_len = client.ap_shmem->ap_gdata.state.fft_len;
  int n = dm*fft_len;
  size_t i;
  static float avg_pow = 2*fft_len*float(fft_len);
  // avg_pow contains N^2 for unnormalized fft, and 2 for 1-bit
  // convention

  client.status = "Dechirping.";
  /* For debugging.  Actually I don't think this runs fast enough
  static int idum = -1;
  for(i=0; i< fft_len; i++) {
   data[i][0] = random_discrete_sum(fft_len, idum);
   data[i][1] = random_discrete_sum(fft_len, idum);
  } */

  // Multiply by the complex conjugate of chirp, so we are
  // DIVIDING by the chirp function.
  for (i=0; i<fft_len; i++) {
    /* positive DM */
    if (sign >= 0) {
      temp[i][0] = data[i][0]*chirp[n+i][0] + data[i][1]*chirp[n+i
          ][1];
      temp[i][1] = data[i][1]*chirp[n+i][0] - data[i][0]*chirp[n+i
          ][1];
```

```
    }
    /* negative DM */
    else {
      temp[i][0] = data[i][0]*chirp[n+i][0] - data[i][1]*chirp[n+i
          ][1];
      temp[i][1] = data[i][1]*chirp[n+i][0] + data[i][0]*chirp[n+i
          ][1];
    }
  }

  client.status = "Computing inverse FFT.";
  fftwf_execute(ip);

  /* only save 1/2 of the fft ... need to check this */
  for (i=0; i<fft_len/2; i++) {
    client.power[i] = (float)((temp[i][0]*temp[i][0] + temp[i][1]*
        temp[i][1])/avg_pow);
  }
}

/* coadd_in_place
 *
 * Arguments:
 * power: an array of power values, obtained after dechirping
 * size: the number of floats in the initial power array.
 * (After the coadd, the number of floats will be halved.)
 *
 * This function halves the size of the power array by adding
 * array elements in consecutive pairs.
 */
void Science::coadd_in_place(float *power, int size) {
  int i;

  for (i=0; i<size/2; i++) {
    power[i] = power[2*i] + power[2*i+1];
  }
}
}
```