AstroAccelerate

GPU accelerated signal processing for next generation radio telescopes

Wes Armour, Karel Adamek, Sofia Dimoudi, Jan Novotny, Nassim Ouannough

Oxford e-Research Centre,
Department of Engineering Science
University of Oxford
27th July 2018

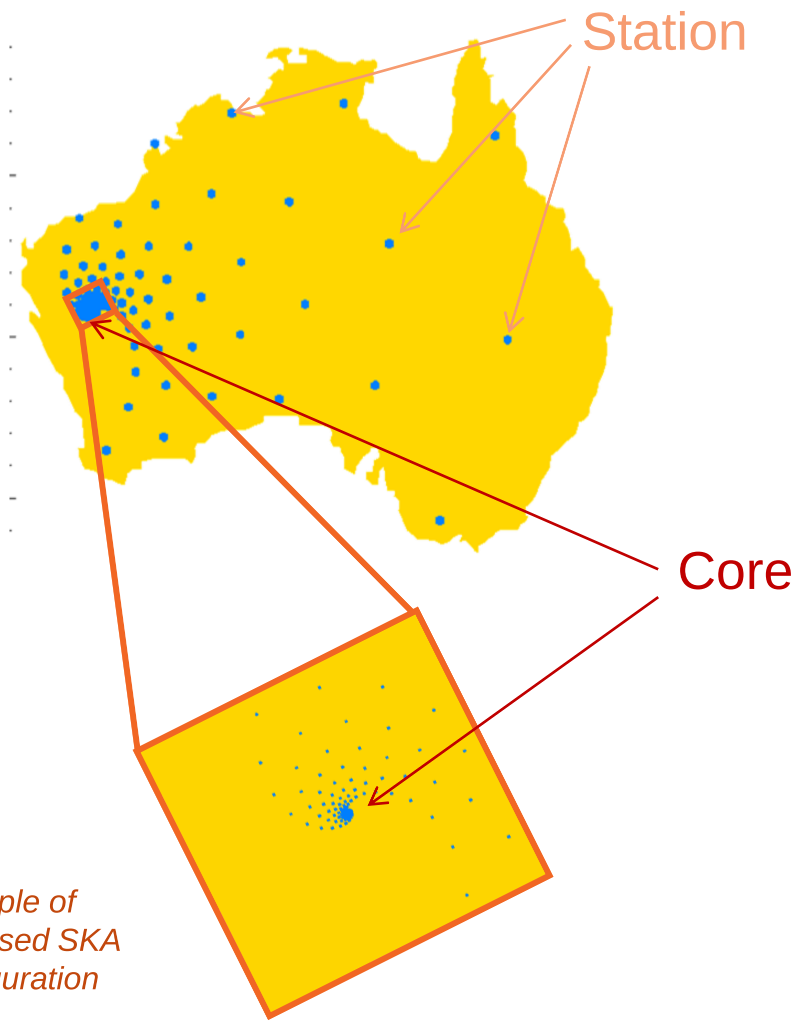
Part One

A brief introduction to

SKA
SQUARE KILOMETRE ARRAY

UNIVERSITY OF OXFORD
Oxford e-Research Centre

What is SKA?



Example of proposed SKA configuration

What is SKA?

SKA is a ground based radio telescope that will span continents.

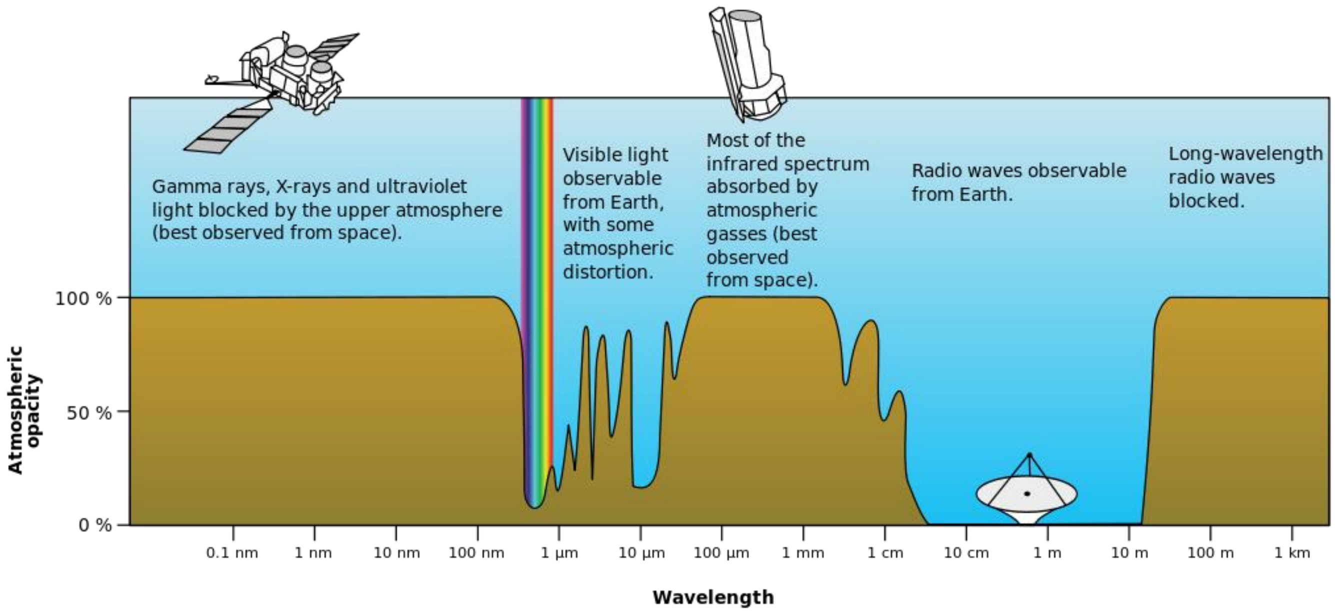
What does SKA stand for?

Square Kilometre Array, so called because it will have an effective collecting area of a square kilometre.

Where will SKA be located?

SKA will be built in South Africa and Australia.

What is SKA?



SKA is a ground based telescope. This means that it is most sensitive to the radio range of frequencies. The radio range of frequencies that can be observed from here on Earth is very wide, specifically SKA will be sensitive to frequencies in the range of 50MHz to 20GHz (wavelengths 15 mm to 6 m). This makes SKA ideal for studying lots of different science cases.

Image source Wikipedia. Authors: NASA (original); SVG by Mysid

What is SKA?

SKA will have the ability to use all of its antennas to produce images of the radio sky in unprecedented accuracy and detail.

It will also be able to use combinations of antennas to perform multiple observations of different regions of the sky at the same time.

In this scenario data from each beam can be computed in parallel.

SKA science

SKA will study a wide range of science cases and aims to answer some of the fundamental questions mankind has about the universe we live in.

- How do galaxies evolve
  - What is dark energy?
- Tests of General Relativity
  - Was Einstein correct?
- Probing the cosmic dawn
  - How did stars form?
- The cradle of life
  - Are we alone in the Universe?

Part Two

Time domain science

SKA time domain - signal processing

The time domain team is an international team led by Oxford and Manchester.

It aims to deliver an end-to-end signal processing pipeline for time domain science performed by SKA (see right).

Our work at OeRC has focussed on vertical prototyping activities. We are interested in using many-core technologies, such as GPUs to perform the processing steps within the signal processing pipeline with the aim of achieving real-time processing for the SKA.

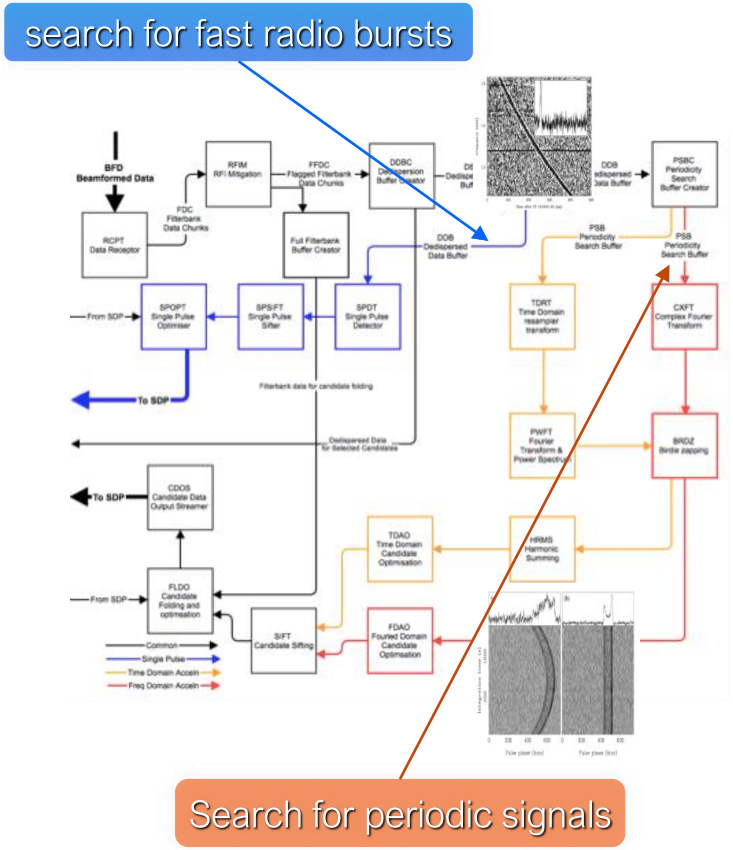


Image courtesy of Aris Karastergiou

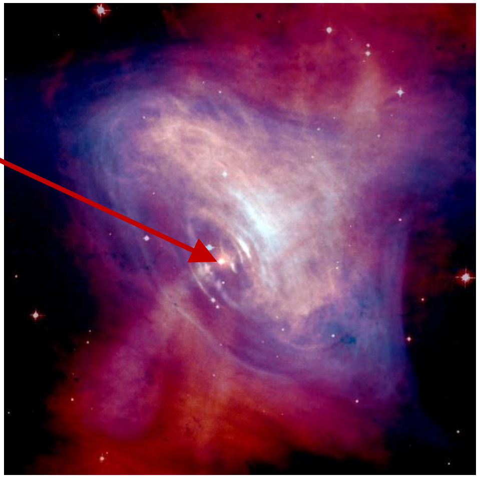
Time Domain Team

SKA time domain science - Pulsars

Pulsars are magnetized, rotating neutron stars. They emit synchrotron radiation from the poles, e.g. Crab Nebula.

Their magnetic field is offset from the axis of rotation as such (as observed from here on Earth, they act as cosmic lighthouses.

They are extremely periodic and so make excellent clocks!



Hester et al.

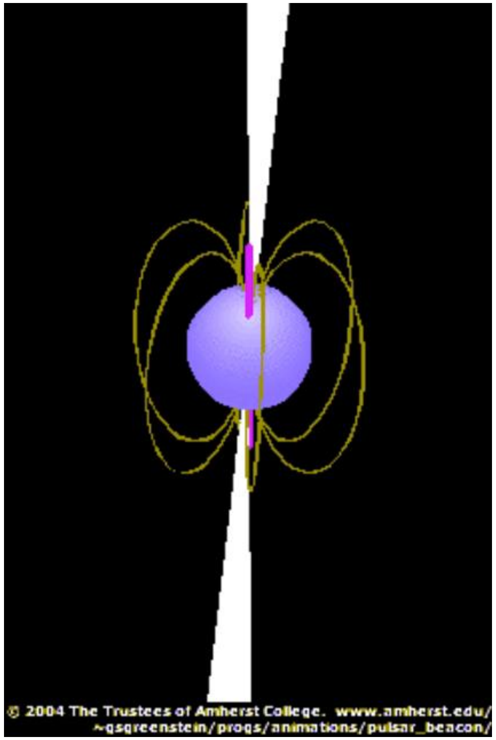
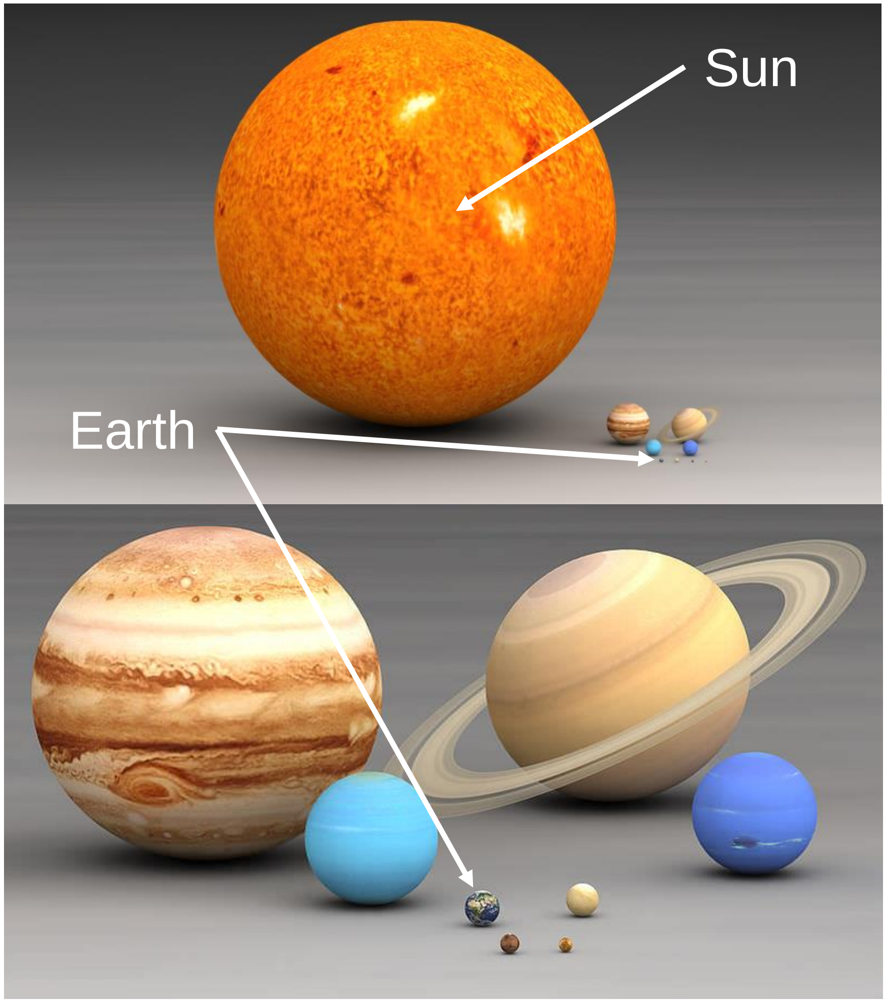


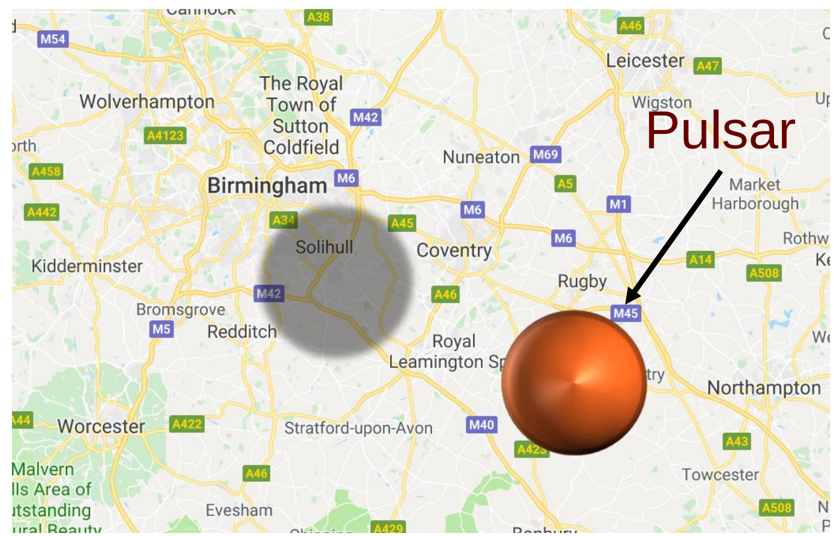
Image: Amherst College

Pulsars – size and scale



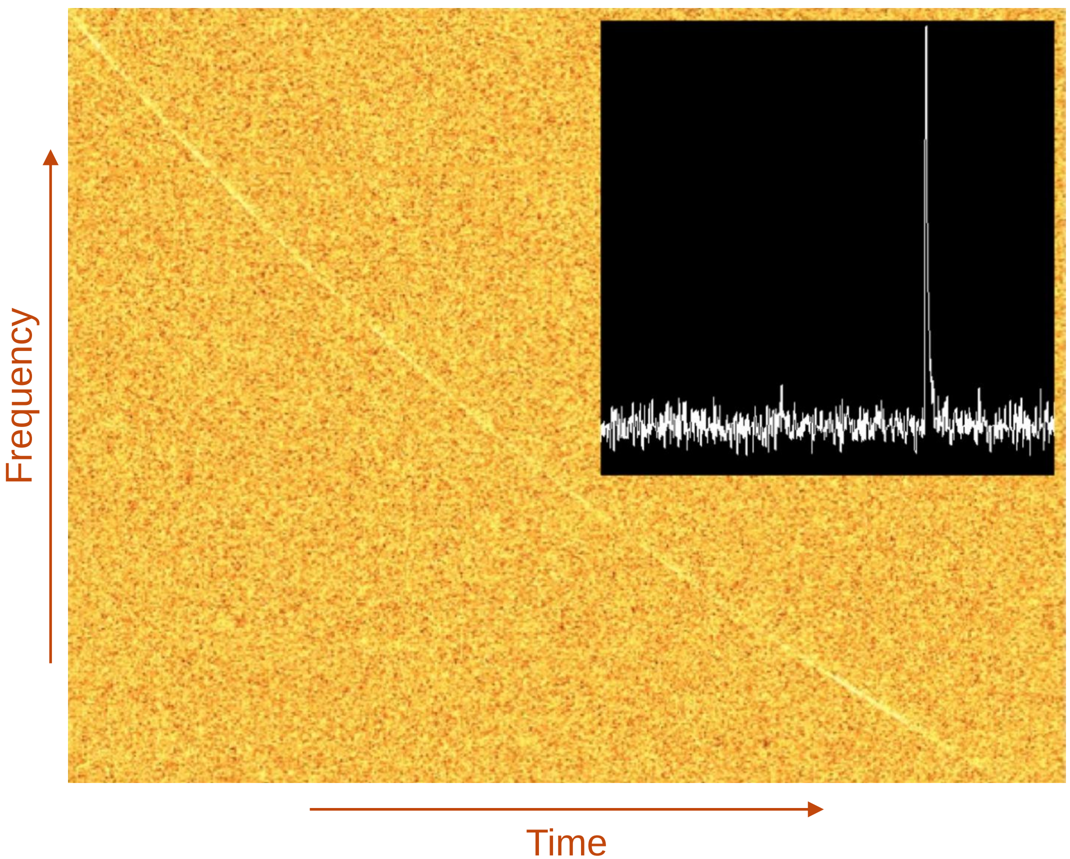
Pulsars are typically 1-3 Solar masses in size, they have a diameter of 10-20 Kilometres and a pulse period ranging from milliseconds to seconds.

Meaning that they are very small, very dense and rotate extremely quickly.



https://commons.wikimedia.org/wiki/File:Planets_and_sun_size_comparison.jpg (Author: Lsmpascal)

SKA time domain science - Fast Radio Bursts



Fast Radio Bursts (FRBs), were first discovered in 2005 by Lorimer et al.

They are observed as extremely bright single pulses that are extremely dispersed (meaning that they are likely to be far away, maybe extra galactic).

So far around 15 have been observed in survey data. They are of unknown origin, but likely to represent some of the most extreme physics in our Universe.

Hence they are extremely interesting objects to study.

Credit: FRB110220 Dan Thornton (Manchester)

Part Three

Data rate challenges

SKA time domain - data rates

The SKA will produce vast amounts of data. In the case of time-domain science we expect the telescope to be able to place ~2000 observing beams on the sky at any one time (there are trivially parallel to compute).

The telescope will take 20,000 samples per second for each of those beams and then it will measure power in 4096 frequency channels for each time sample. Each of those individual samples will comprise of 4x8 bits, although we are only really interested in one of the 8 bits of information.

Doing the math tells us that we will need to process 160GB/s of relevant data. This is approximately equal to analysing 50 hours of HD television data per second.

The most costly computational operations in data processing pipeline are

DDTR ~ $O(n_{dms} * n_{beams} * n_{samps} * n_{chans})$

FDAS ~ $O(n_{dms} * n_{beams} * n_{samps} * n_{acc} * \log(n_{samps}) * 1/t_{obs})$

Requiring ~2 PetaFLOP of Compute!

UNIVERSITY OF OXFORD

Oxford e-Research Centre

SKA time domain – data challenges

Because we would like to monitor interesting and exotic events as they occur we need to process data in real-time (or as near to as possible).

So storing the data and processing later isn't feasible. The data rates mean transporting data offsite would be challenging and costly.

So processing must happen close to the telescope. But how do we put a computer capable of processing big-data streams in real-time in the middle of a desert?

Connectivity, power, operation all pose significant problems.

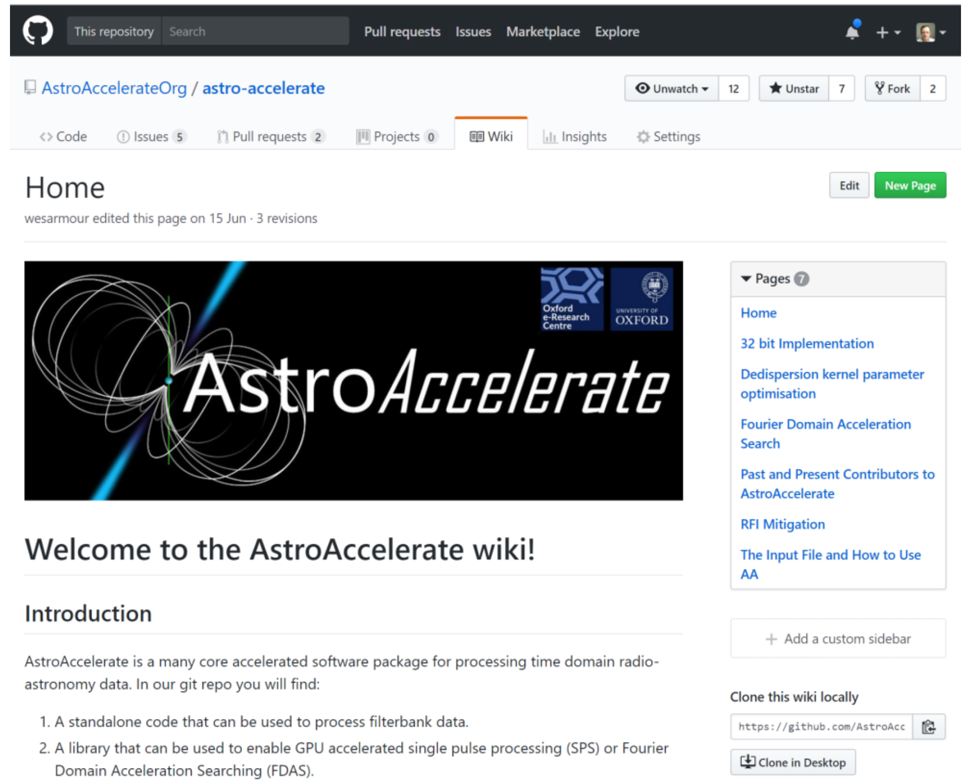
Part Four

Two case studies

AstroAccelerate

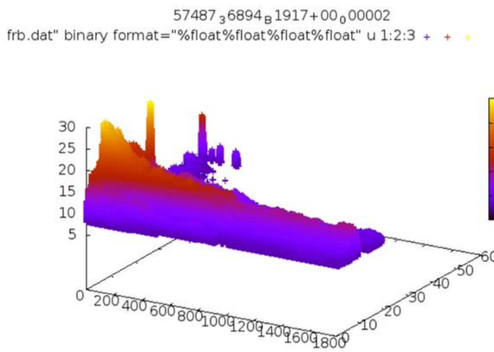
AstroAccelerate is a GPU enabled software package that focuses on achieving real-time processing of time-domain radio-astronomy data. It uses the CUDA programming language for NVIDIA GPUs.

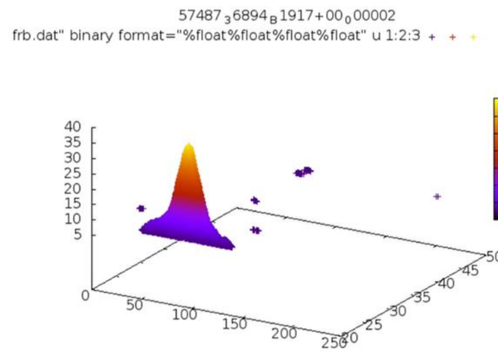
The massive computational power of modern day GPUs allows the code to perform algorithms such as de-dispersion, single pulse searching and Fourier Domain Acceleration Searching in real-time on very large data-sets which are comparable to those which will be produced by next generation radio-telescopes such as the SKA.

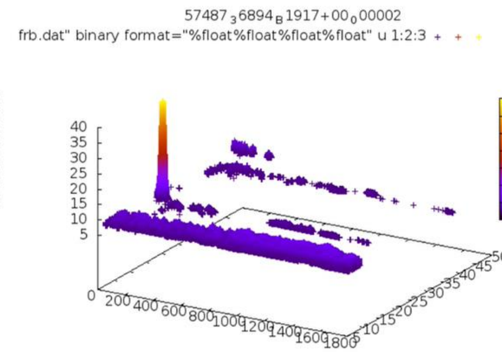


https://github.com/AstroAccelerateOrg/astro-accelerate

AstroAccelerate - Features

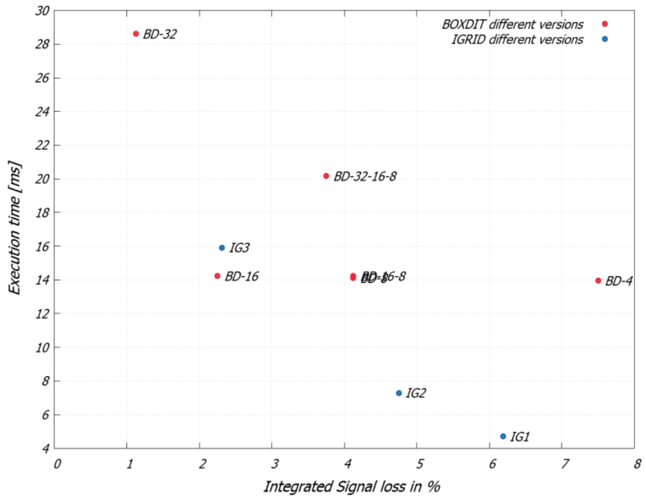






AstroAccelerate has the following features…

- Zero DM and basic RFI Mitigation
- DDTR
- Single Pulse Search
- Fourier Domain Acceleration Search
- Periodicity search with harmonic sum

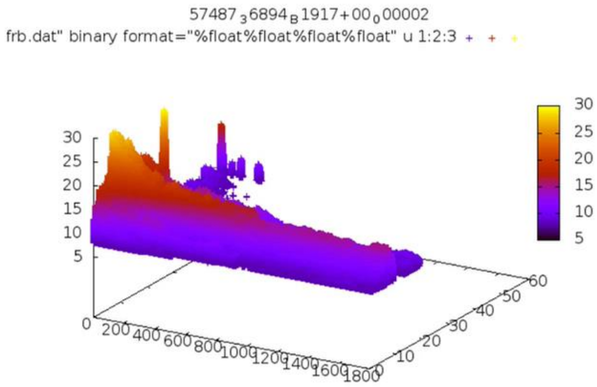


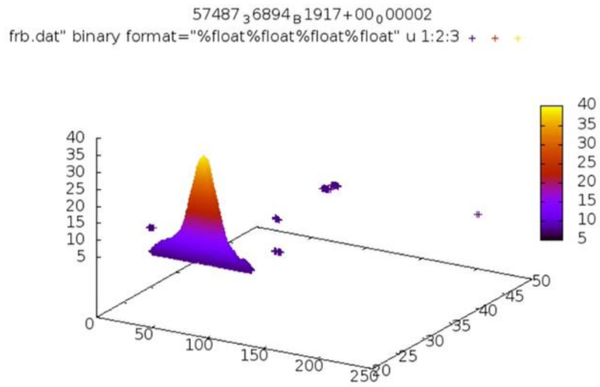
RFI Mitigation

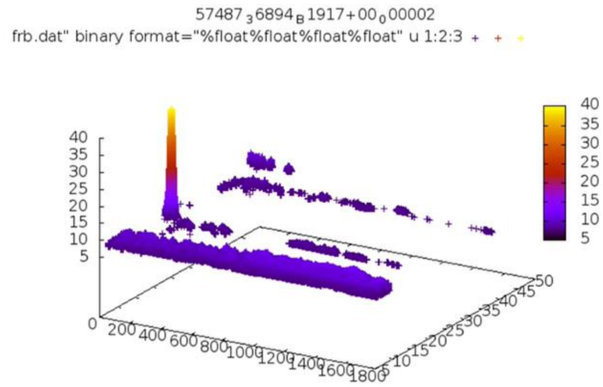
Image Left: No RFI mitigation.

Image Center: Old RFI AstroAccelerate Algorithms.

Image Right: New algorithms using a moving average (enabled with both "zero_dm_with_outliers" and "rfi" keywords).







With thanks to Mitch Mickaliger, Jayanta Roy and Ben Stappers for supplying test data and help with testing

UNIVERSITY OF OXFORD

Oxford e-Research Centre

Single Pulse Search

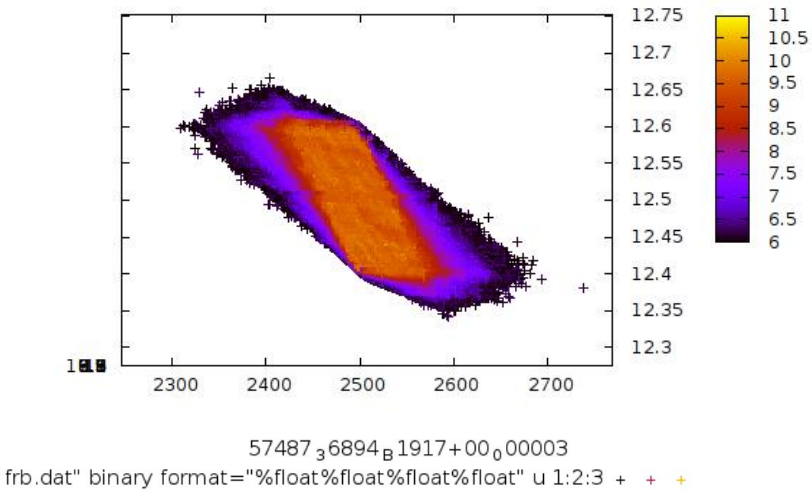
Our single pulse search uses DDTR, SPDT and peakfind.

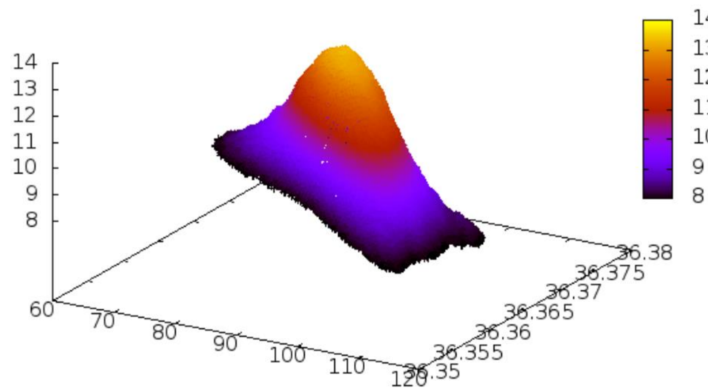
We have two codes to perform single pulse detection, BOXDIT and IGRID.

Both of these codes use boxcars in the time domain to recover signals.

TOP: A single pulse recovered from a fake file at DM = 2500

BOTTOM: A single pulse from B1917+00





Thanks to Mitch Mickaliger and Ben Stappers for data from the Lovell telescope

Single Pulse Search: BOXDIT

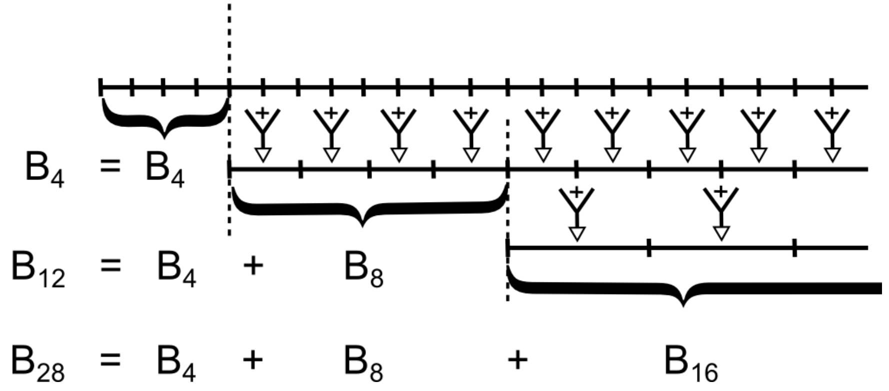
This algorithm works by using reusing previously (time) decimated data to build longer width boxcars.

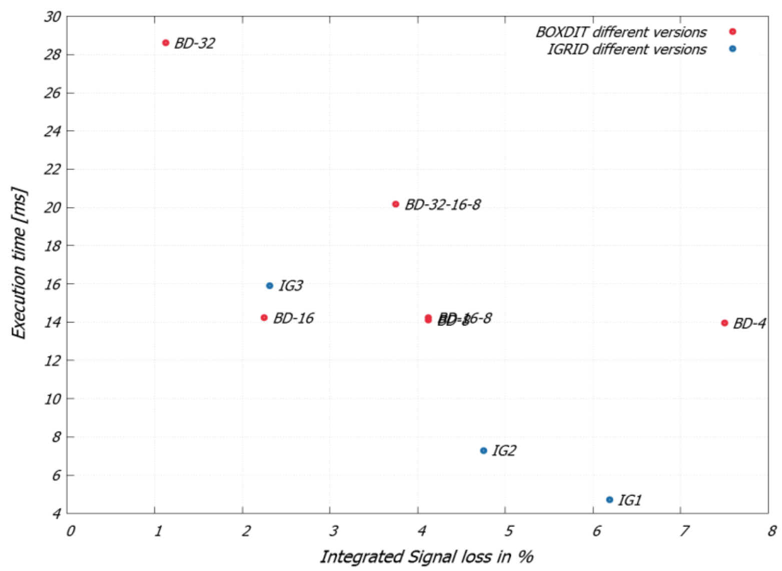
As we recursively decimate in time (adding nearest neighbour samples at each decimation) we save previously decimated data.

TOP: Using combinations of data at different decimation levels allows us to construct different width boxcars.

BOTTOM: The number of decimation levels saved has an impact on the speed and sensitivity of the code.

For a SPS with boxcars up to a width of 8192 time samples BOXDIT is ***444x*** *faster than the naïve boxcar approach.*





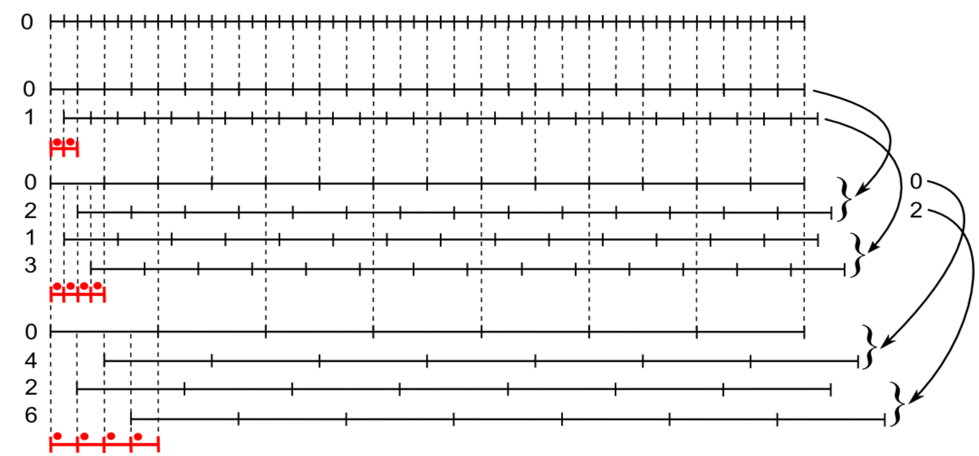
Single Pulse Search: IGRID

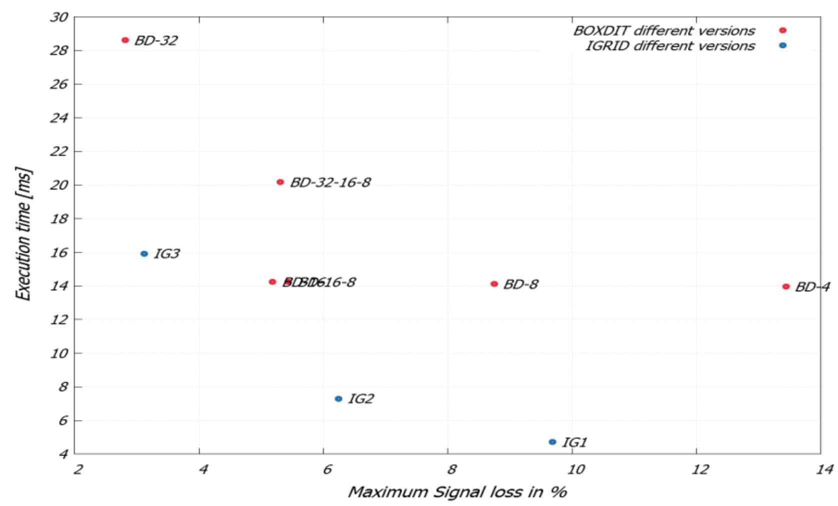
IGRID: This algorithm works by using shifted (in time) nearest neighbour decimated data to build boxcars of differing widths at different positions in time.

Different decimations and different time shifts are used to achieve a similar level of sensitivity to BOXDIT whilst using less boxcars.

TOP: Using combinations of different decimation levels and different time shifts allows us to construct different width boxcars.

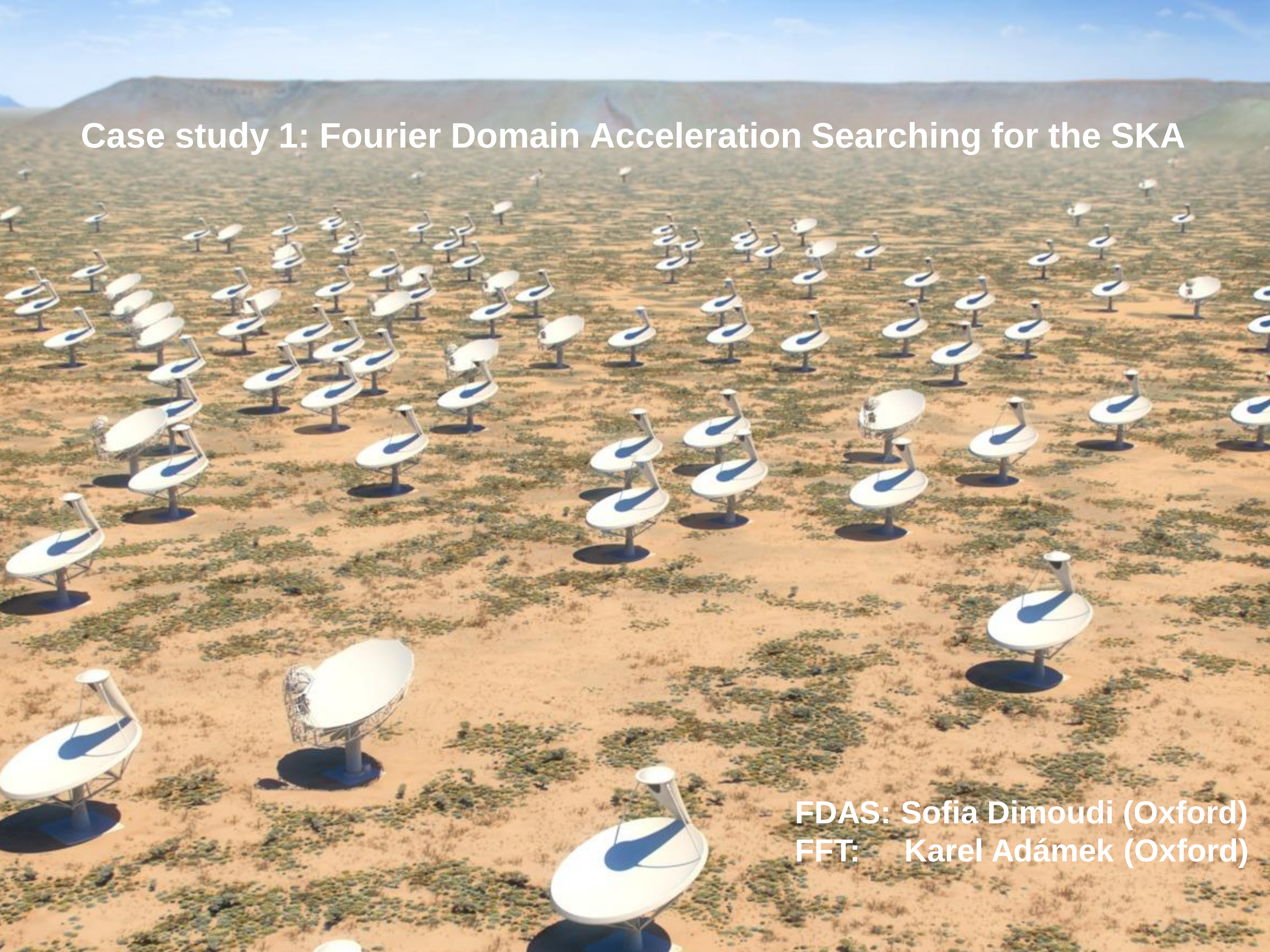
BOTTOM: Execution time as a function of maximum signal loss.





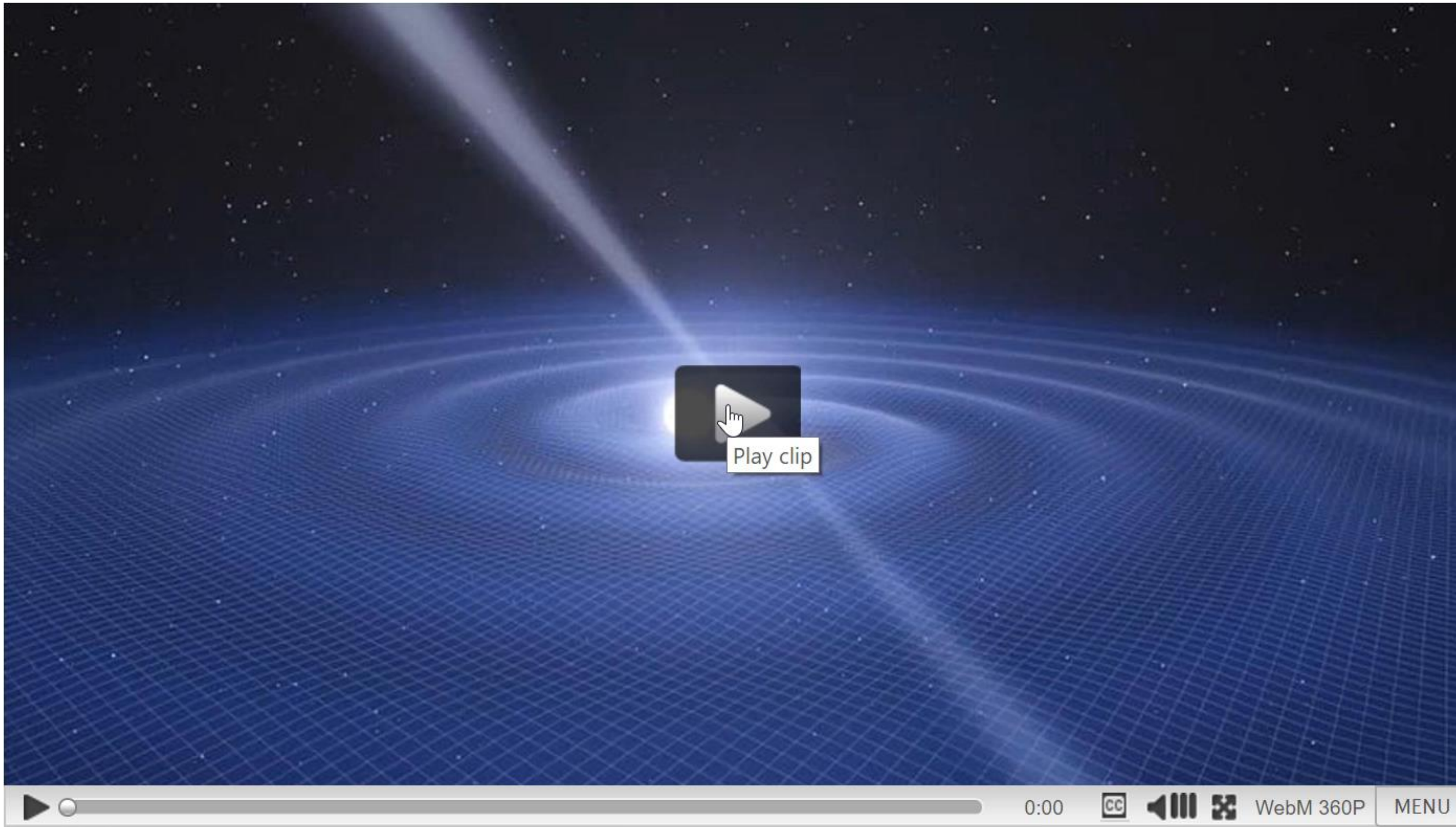
K. Adamek et.al Publication in prep.

Case study 1: Fourier Domain Acceleration Searching for the SKA



FDAS: Sofia Dimoudi (Oxford)
FFT: Karel Adámek (Oxford)

Binary pulsars and gravitational waves



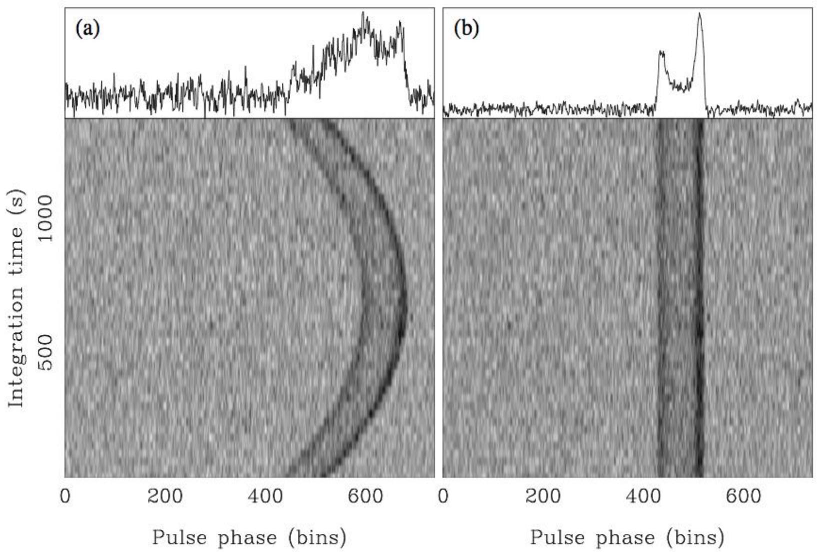
http://www.eso.org/public/videos/eso1319a/ Author: ESO/L. Calçada

Fourier Domain Acceleration Search - FDAS

Signals from binary systems can undergo a Doppler shift due to accelerated motion experienced over the orbital period.

Much like the sound of a siren approaching you and then speeding away.

This can be corrected by using a matched filter approach.

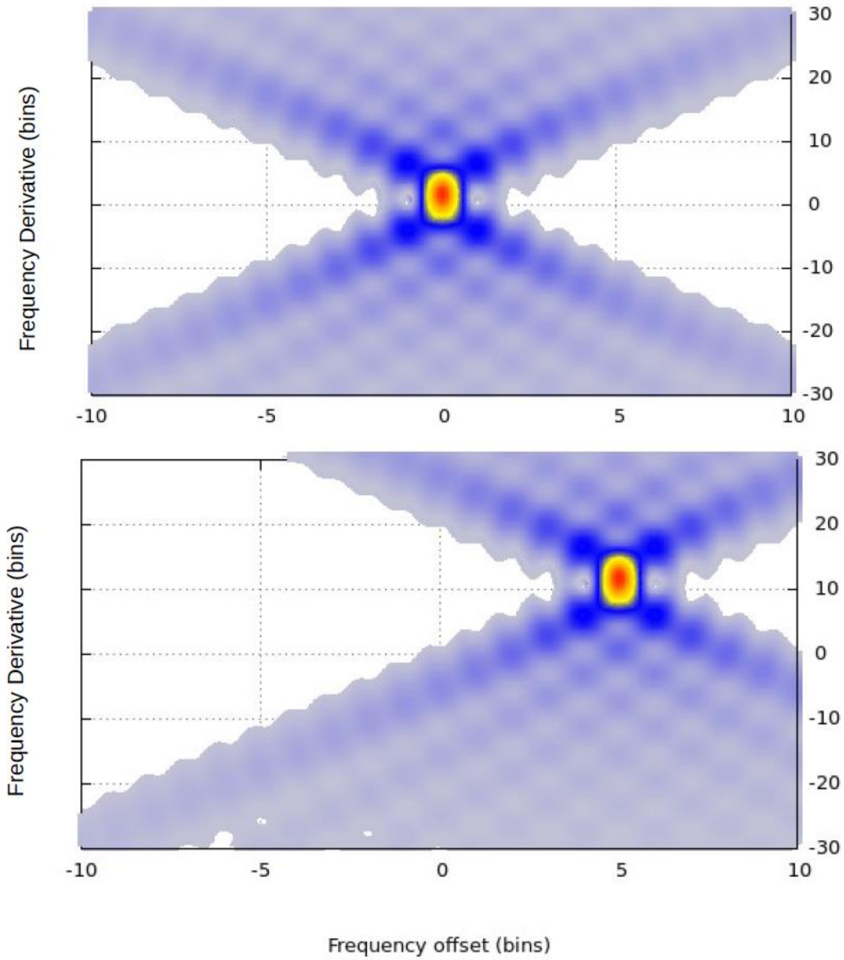


Ransom, Eikenberry, Middleditch: AJ, Vol 24, Issue 3, pp. 1788-1809

UNIVERSITY OF OXFORD

Oxford e-Research Centre

FDAS Example



The two plots illustrate the effect of orbital acceleration.

The first plot shows a signal without acceleration, the signal is centred on its frequency and lies on the f-dot template corresponding to zero acceleration.

The second plot shows a signal with a frequency derivative, and has drifted from the original frequency by a number of bins.

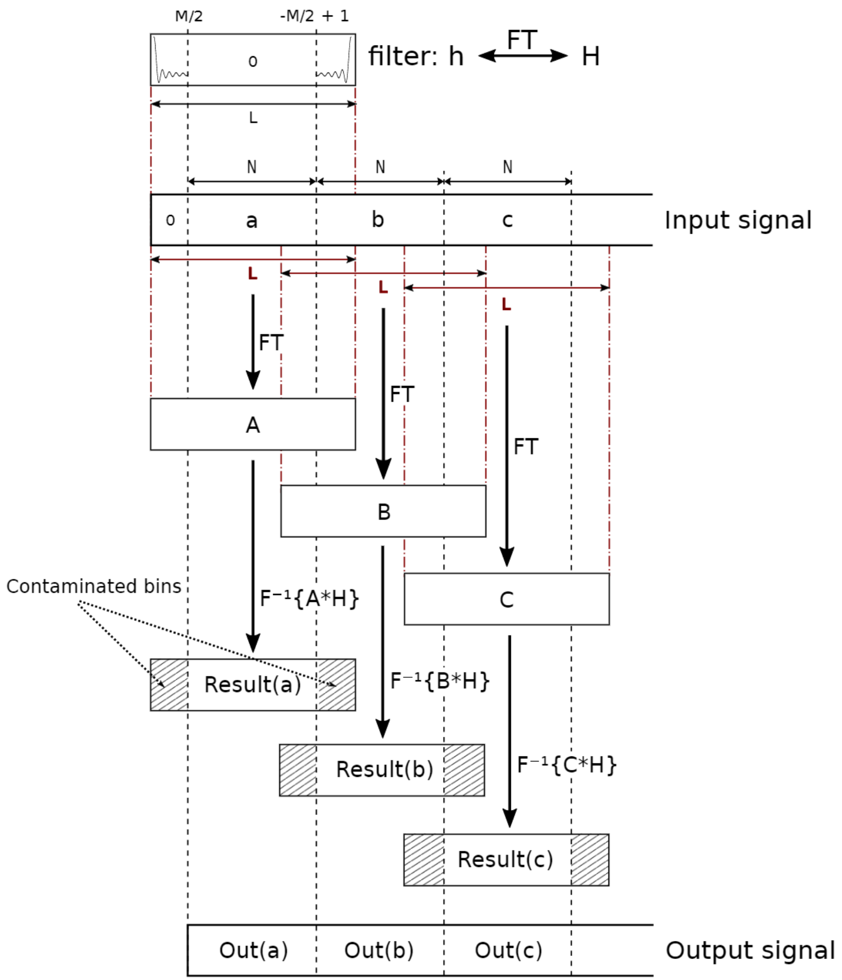
Fourier Domain Acceleration Search

Use overlap-save algorithm to compute cyclic N-point convolution of template with signal segment.

Avoids the need for synchronisation because contaminated ends of convolved data are discarded (as opposed to overlap-add).

Code calculates the convolution, powers and extracts peaks.

Currently has no harmonic sum.



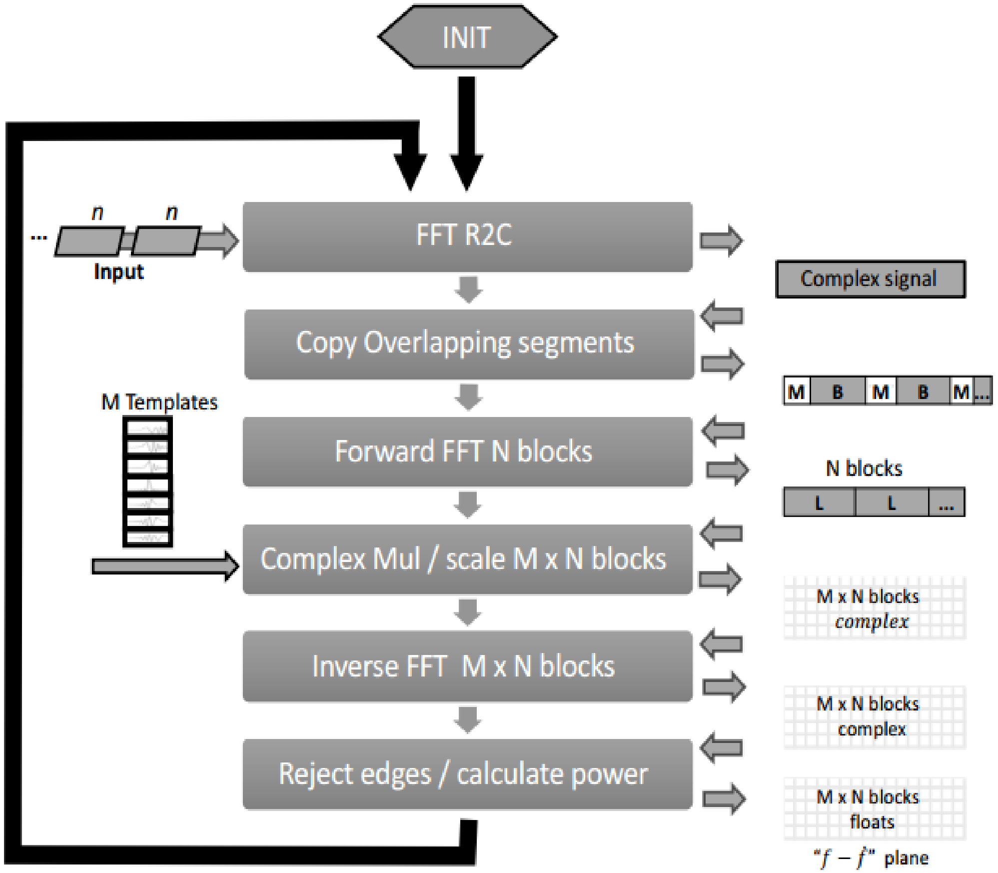
S. Dimoudi et.al. Accepted *ApJS.*

Fourier Domain Acceleration Search

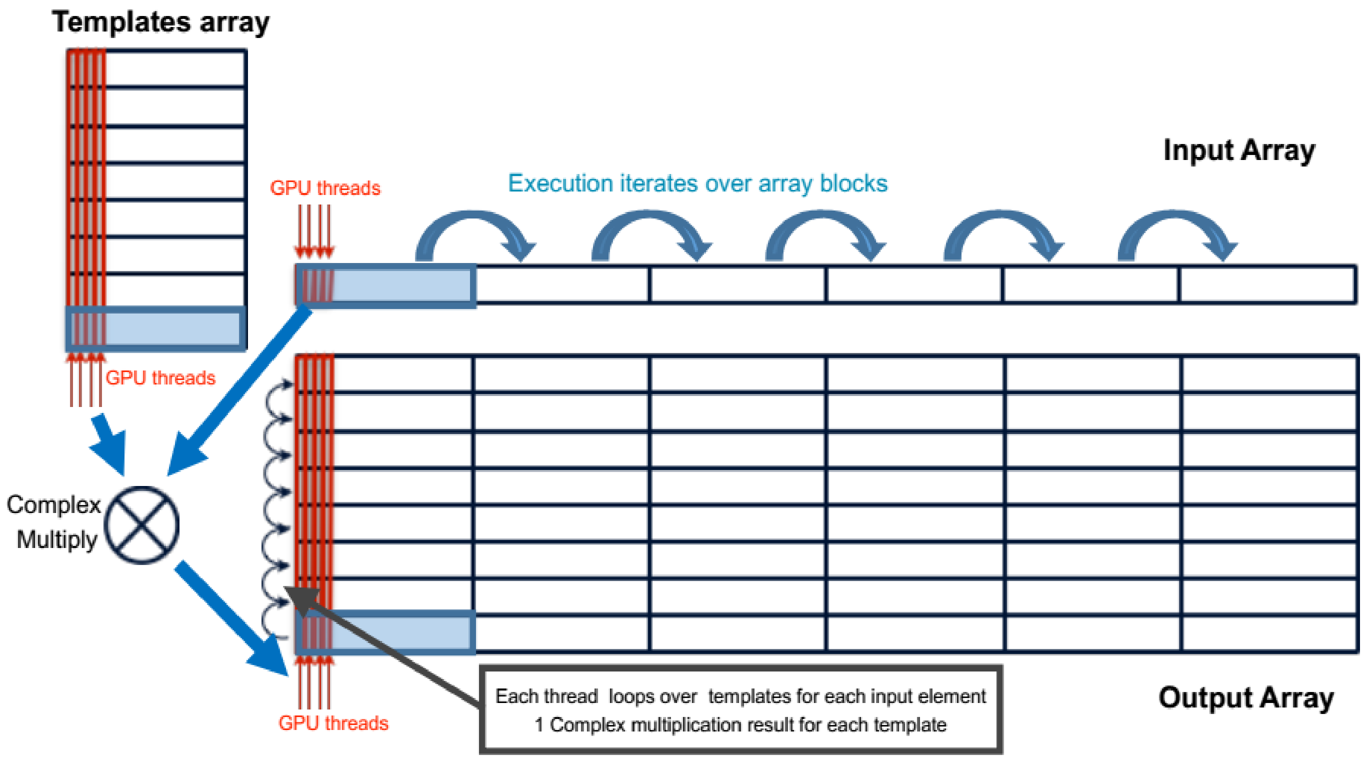
Using cuFFT means many transactions to device memory on the GPU (represented by grey arrows on the right of the diagram).

This causes the computation to be limited by global memory bandwidth (the lowest common denominator on a GPU).

This means that a cuFFT based implementation is very slow.

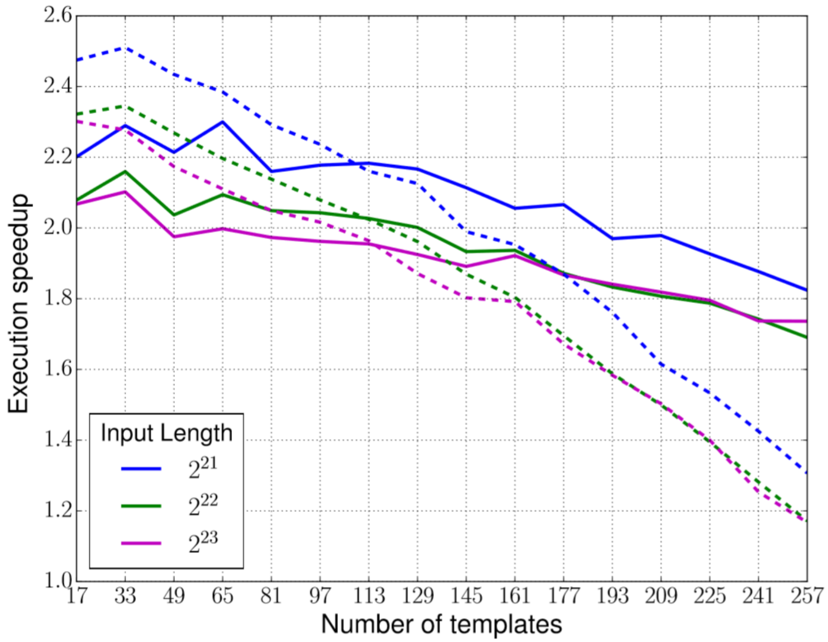


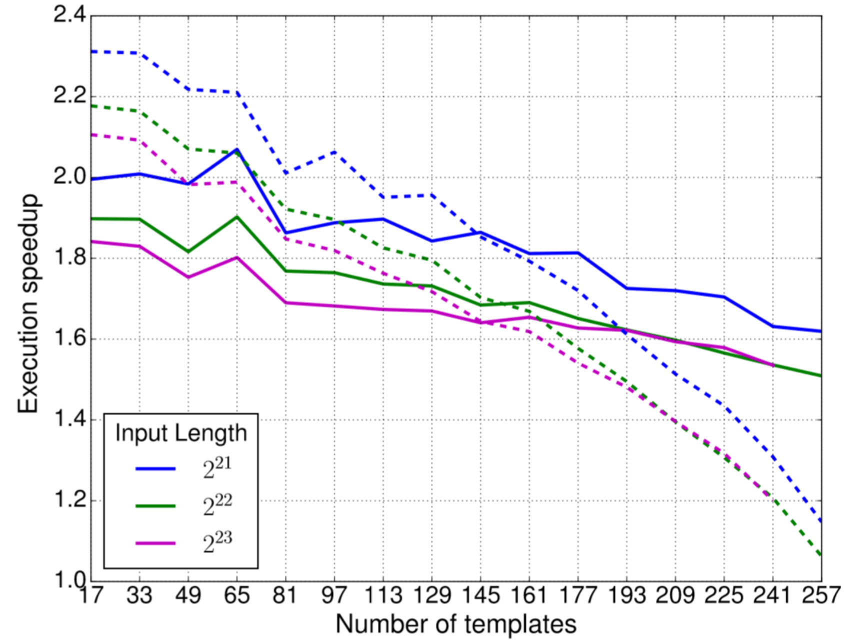
Fourier Domain Acceleration Search



By writing our own custom I/FFT codes to work on shared memory we can perform the FFT, pointwise multiply and scale, IFFT and edge rejection all in one kernel.

Fourier Domain Acceleration Search





Results from our tests on a Tesla P100. In the SKA region of interest – signal length 2^{23}, template size of 512 (solid line) and no interbinning (left graph)

For our latest FFT codes we achieve approximately a 3.5x speed increase on a P100

S. Dimoudi et.al. *Submitted to ApJS.*

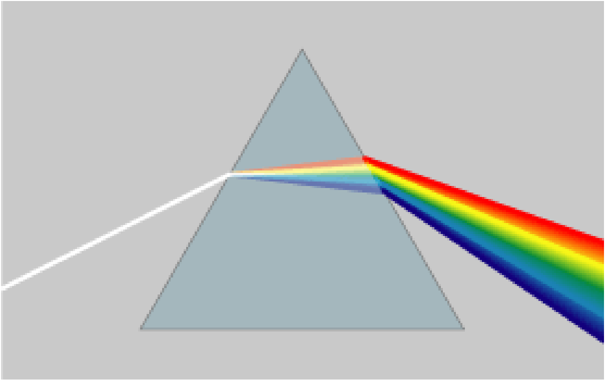
UNIVERSITY OF OXFORD

Oxford e-Research Centre

Case study 2: Real-time de-dispersion for the SKA

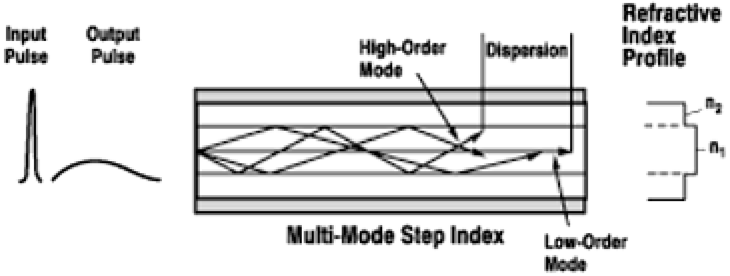
Mike Giles (Oxford)
Karel Adámek (Oxford)
Jan Novotný (Oxford)
Kate Clark (NVIDIA)
Tom Bradley (NVIDIA)
Tim Lanfear (NVIDIA)

What is dispersion?



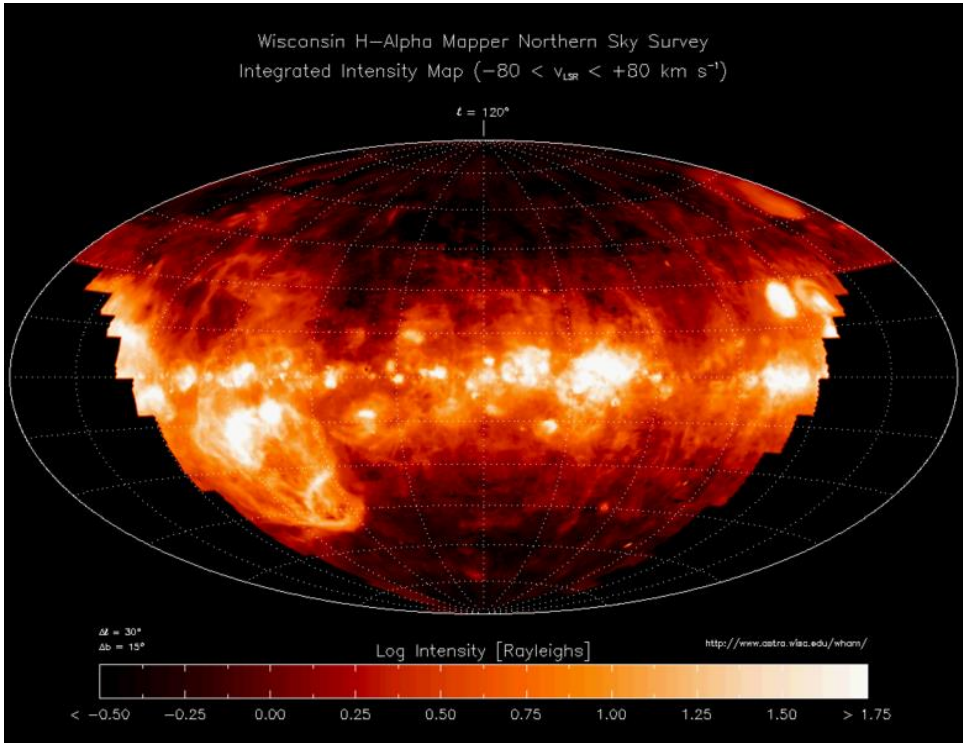
Chromatic dispersion is something we are all familiar with. A good example of this is when white light passes through a prism.

Group velocity dispersion occurs when pulse of light is spread in time due to its different frequency components travelling at different velocities. An example of this is when a pulse of light travels along an optical fibre.



Dispersion by the ISM

The interstellar medium (ISM) is the matter that exists between stars in a galaxy.



Haffner et al. 2003

In warm regions of the ISM (~8000K) electrons are free and so can interact with and affect radio waves that pass through it.

The dispersion measure - DM

The time delay, $\Delta\tau$, between the detection of frequency f_{high} and f_{low} is given by:

$$\Delta\tau = C_{DM} \times \mathrm{DM} \times \left(\frac{1}{f_{low}^2} - \frac{1}{f_{high}^2}\right)$$

Where C_{DM} is the dispersion constant. DM is the dispersion measure:

$$DM = \int_0^d n_e dl$$

This is the free electron column density between the radio source and observer.

We can measure $\Delta\tau$ and f and so can study DM

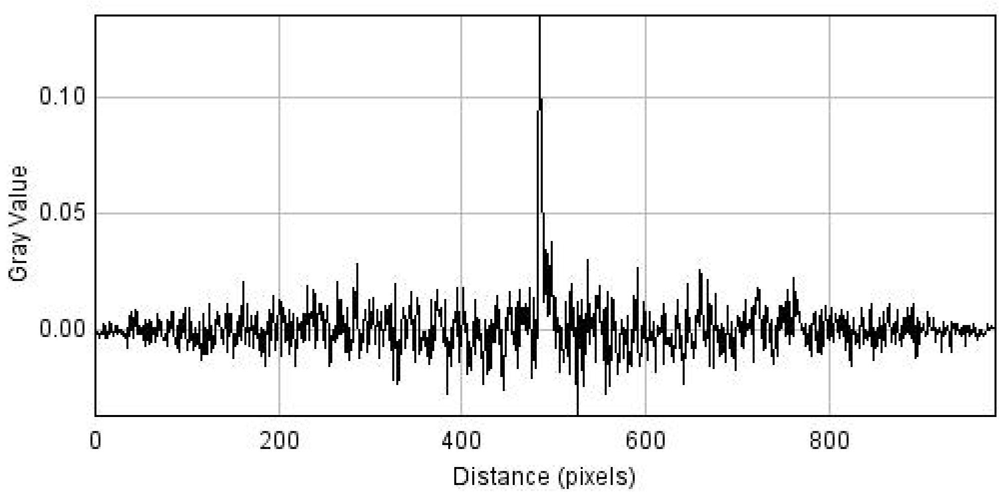
Experimental Data

Most of the measured signals live in the noise of the apparatus.

Experimental Data

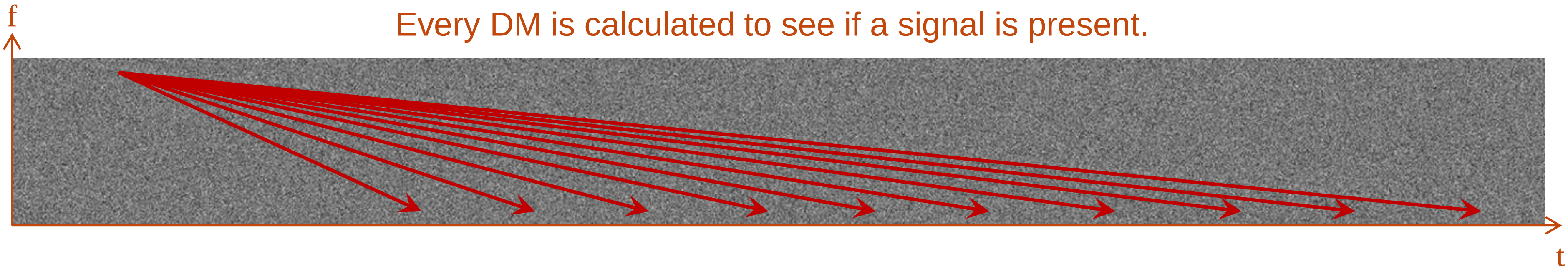
Most of the measured signals live in the noise of the apparatus.

Hence frequency channels have to be “folded”



De-dispersion

Every DM is calculated to see if a signal is present.



In a blind search for a signal many different dispersion measures are calculated.

This results in many data points in the (f,t) domain being used multiple times for different dispersion searches.

This allows for data reuse in a GPU algorithm.

All of this must happen in real-time i.e. the time taken to process all of our data must not exceed the time taken to collect it

De-dispersion Transform

Our DDTR is an implementation of incoherent brute force de-dispersion.

1. We brute force optimise the tuneable parameters of the code, such as the thread block size and number of registers used.

2. It utilises GPU shared memory and typically achieves 60-80% of peak throughput.

3. It uses SIMD in work to process multiple time samples per machine word for data less than or equal to 16 bits.

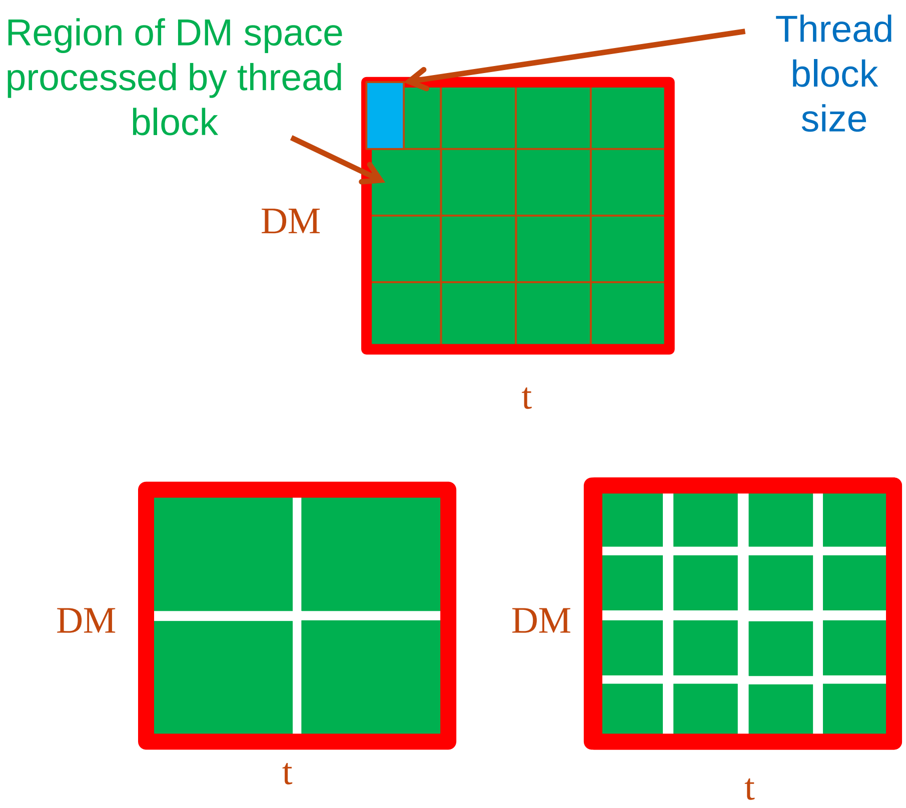
```
for (i = 0; i < SNUMREG; i++)
{
        local = 0;
        unroll = ( i * 2 * SDIVINT );
        for (j = 0; j < UNROLLS; j++)
        {
                stage = *(int*) &f_line[j][( shift[j] + unroll )];
                local += stage;
        }
        local_kernel_one[i] += (local & 0x0000FFFF);
        local_kernel_two[i] += (local & 0xFFFF0000) >> 16;
}
```

De-dispersion Transform – 1. tuning

Each thread processes a tunable number of time samples, each de-dispersion trial associated with one time sample is stored in a GPU register.

Along with this the number of time samples per thread block and the number of de-dispersion trials (which is where data reuse comes from) are tuned.

Finally the code performs a tunable number of SIMD in word operations which are periodically unloaded to a floating point accumulator.

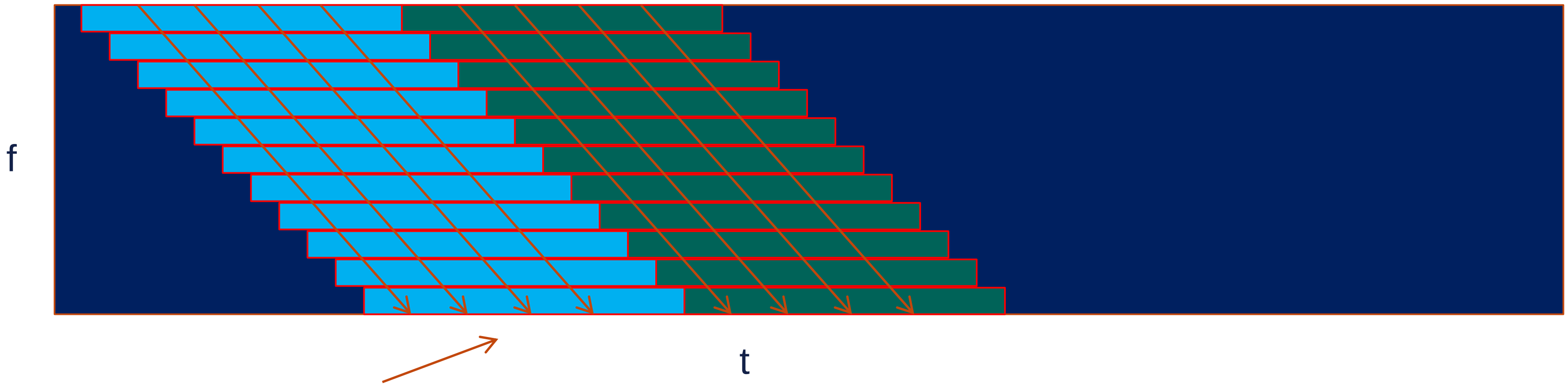


Optimising the parameterisation

De-dispersion Transform – 2. shared memory

Exploiting registers and fast shared memory…

Each dispersion measure for a given frequency channel needs a shifted time value.



Constant DM's with varying time.

In practice a thread will process multiple time samples and a thread block will also process neighboring DM trials to increase data reuse.

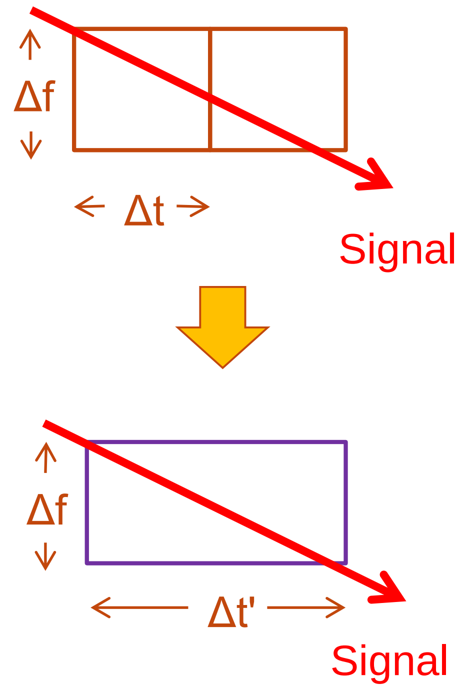
Incrementing all of the registers at every frequency step ensures a high data reuse of the stored frequency time data in the L1 cache or shared memory.

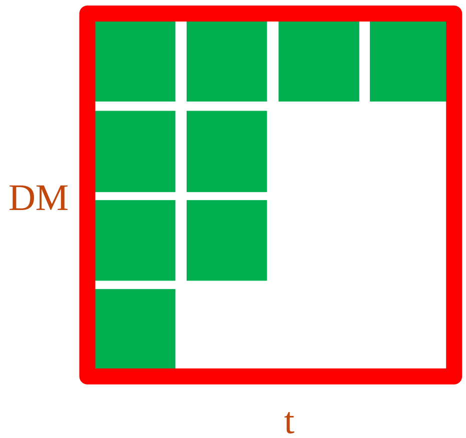
De-dispersion Transform – 2. time binning

One issue with using a shared memory based algorithm is that for high DM trials (those that represent distant objects, forming long broad curves in our input frequency-time data) we need to store increasing lengths of constant frequency varying time data in shared memory.

This ultimately limits the highest DM trial that can be searched at full time resolution.

To overcome this we've added a time binning (scrunching) kernel that decimates data in time. This has the effect of decreasing time resolution and allows us to search to arbitrary high DM trials.





Has the added advantage of reducing the amount threads that are needed to process a region of (DM,t) space, speeding up the code.

De-dispersion Transform – 3. SIMD in word

We exploit the fact that one frequency-time sample of SKA data will be 8 bits.

We pack the data in such a way so that we can perform two de-dispersion trials per integer operation.

We convert the unsigned char to an unsigned short and pack as ushort2, we mask this as an int and add ints.

Once a single trial nears the maximum allowable value for a ushort we store the value in a floating point accumulator. This has the effect of increasing the speed of the code and also it's precision.

Recorded telescope data (t_n = 8 bits) is stored in global as a uchar array

char[] = $[t_0,t_1,t_2,t_3,t_4,t_5,t_6 \ldots]$

This is converted to ushort when loaded though the texture pipe (doubling the size of the array stored because it is now interleaved with 8 bits of zeros

ushort[] = $[0\ t_0,\ 0\ t_1,\ 0\ t_2,\ 0\ t_3,\ 0\ t_4,\ 0\ t_5,\ 0\ t_6,\ \ldots]$

Masking this with an int allows us to add two samples per one instruction issued.

De-dispersion Transform – SIMD in word

In reality we have to odd/even interleave the data to ensure correct byte alignment within shared memory banks (4 bytes wide).

ushort2[] = [0 t_0,0 t_1][0 t_1,0 t_2][0 t_2,0 t_3][0 t_3,0 t_4]…

For thread with an even shift (lets say 2)…

(t_0,t_1) (t_1,t_2) (t_2,t_3) (t_3,t_4) (t_4,t_5) (t_5,t_6)

$t_i = t_2$

$t_{i+1} = t_3$

For thread with an odd shift (lets say 3)…

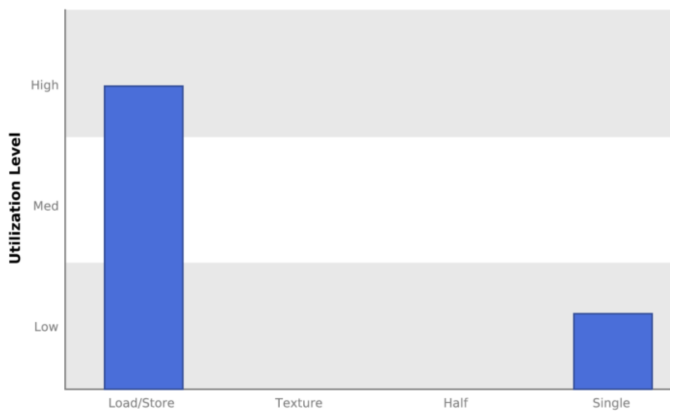
(t_0,t_1) (t_1,t_2) (t_2,t_3) (t_3,t_4) (t_4,t_5) (t_5,t_6)

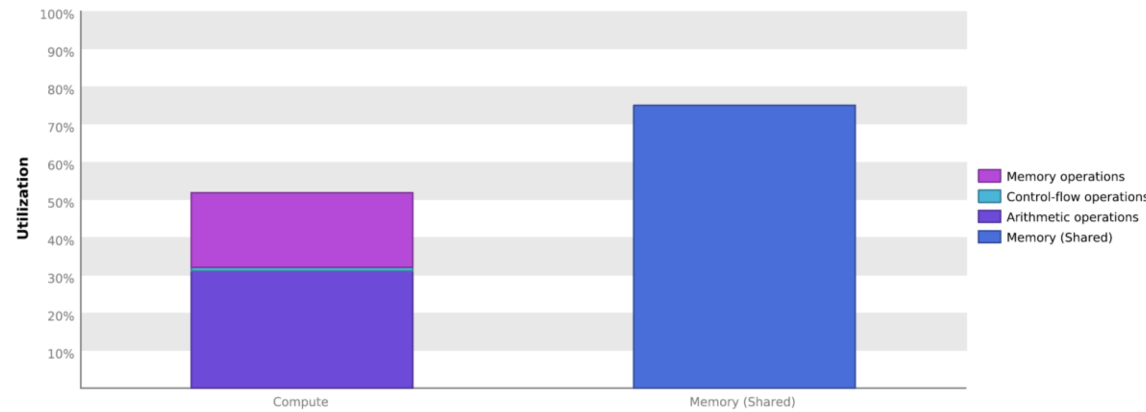
$t_i = t_3$

$t_{i+1} = t_4$

Now each thread computes the correct two time values and at double data rate

De-dispersion Transform - results





Shared Memory

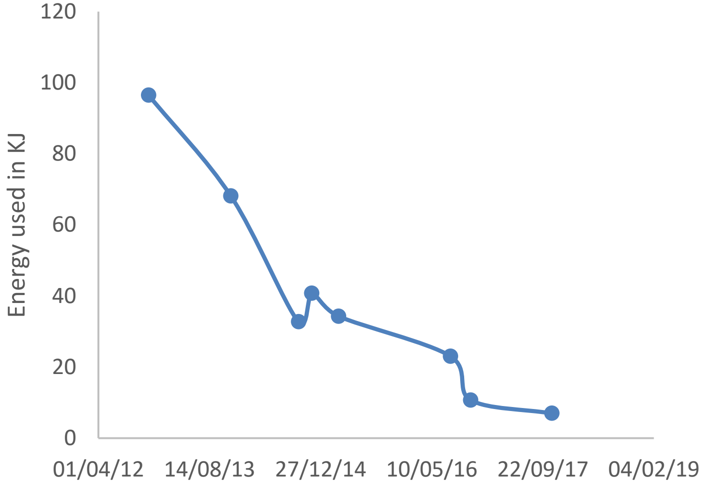
Shared Loads	43769001212	7,976.627 GB/s	
Shared Stores	5655113738	1,030.609 GB/s	
Shared Total	49424114950	9,007.236 GB/s	Idle Low Medium High Max

Results showing the shared memory utilisation, which is this codes limiting factor.
We achieve 75% of peak throughput, limited by load/store.

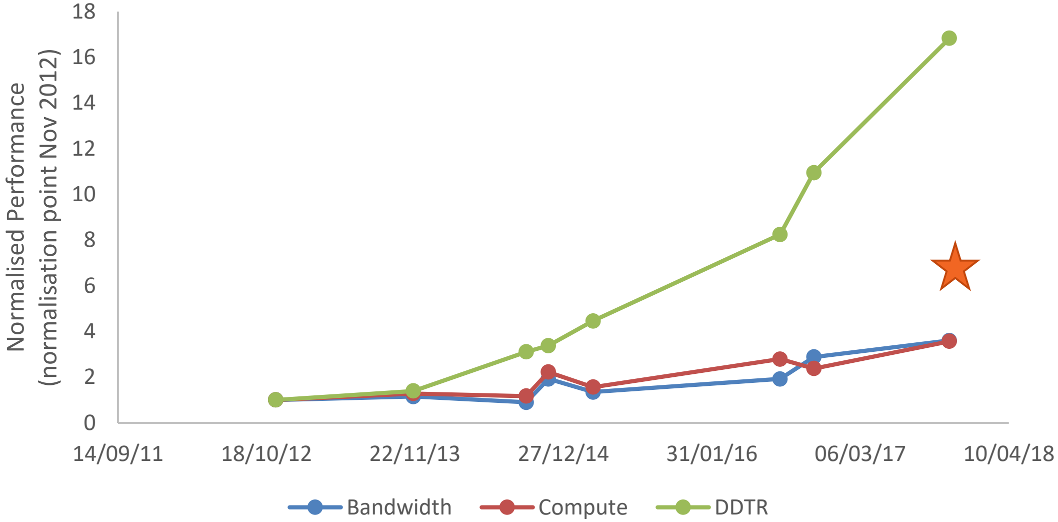
The total shared memory bandwidth throughput achieved on a TITAN V is 9 TB/s.

De-dispersion Transform - results

Energy used (KJ) by a GPU when performing the DDTR algorithm for a single SKA beam



Summary of the performance increases in our DDTR GPU algorithm over a 6 year period starting November 2012



These two plots demonstrate how we have reduced power consumption and increased performance for the DDTR algorithm over a six year period.

The red star indicates the performance of our initial (optimised) code running on current hardware. Demonstrating how invested effort algorithm optimisation over a long period can deliver significant gains.

UNIVERSITY OF OXFORD

Oxford e-Research Centre

De-dispersion Transform – cost / benefit analysis

But is it worth the effort?

Estimated runtime for DDTR in the PSS pipeline (conservative 25%)
Estimate of speed increase compared to initial code ~17x

⇨ Total PSS pipeline acceleration ~ 4x

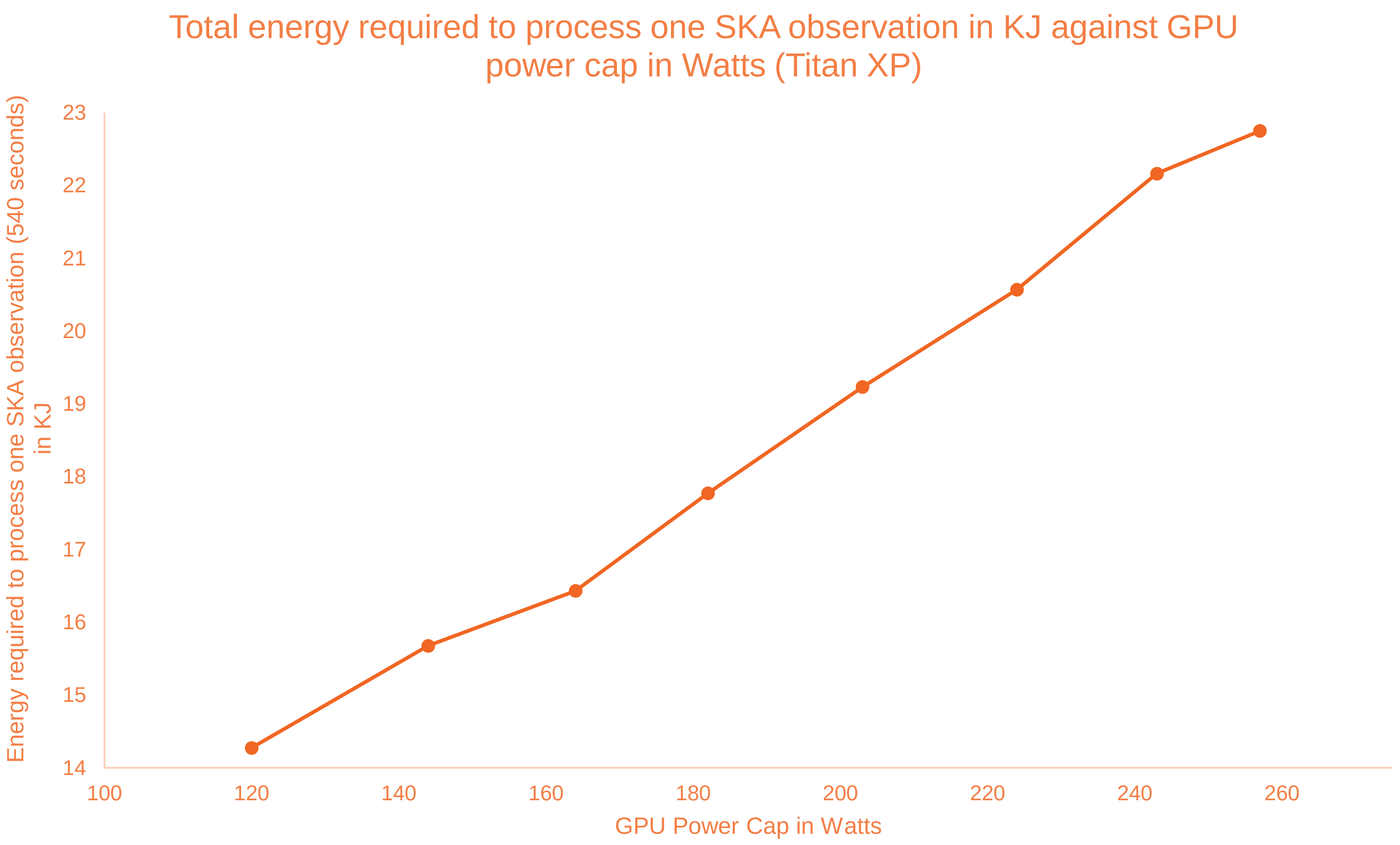
So to deliver the science in the same wall clock time you'd need 4x the GPU capacity.

Even if you're prepared to wait 4x longer… Energy efficiency has increased by 14x
Very rough estimate of PSS OpEx saving ~ £1M
Estimate of total effort ~ 1.0FTE for four years ~ £250K (FEC)

Hence a £750K saving in OpEx costs alone (this is a conservative estimate).

(You can't just go out and buy this at a later date. Domain expertise in both radio astronomy data processing and many-core acceleration are needed)

Energy needed to process one observation



Conclusions – Comparisons of GPUs

Technology	Kepler (K40)	Kepler (K80)	Kepler (780Ti)	Maxwell (980)	Maxwell (Titan X)	Pascal (Titan XP)	Pascal P100	Volta V100	Volta Titan V
Fraction of real-time	1.035	2.5	2.88	2.3	3.3	6.1	8.1	12.5	10.9
Watts per beam (Average)	127W	76 W	~70W	~61W	~64W	~43W	~24W	13W	10W
Cost per beam (capital, accelerator only)	£3K?	£4K?	£250	£200	£240	~£200	~£420	~£530	~£270
Cost per beam (2 year survey, GPU only, based on 1KWh costing £0.2)	~£430	~£265	~£245	~£213	~£224	~£151	~£84	~£45	~£35

Improvement between generations comes from a combination of advances in both the hardware and algorithm

UNIVERSITY OF OXFORD

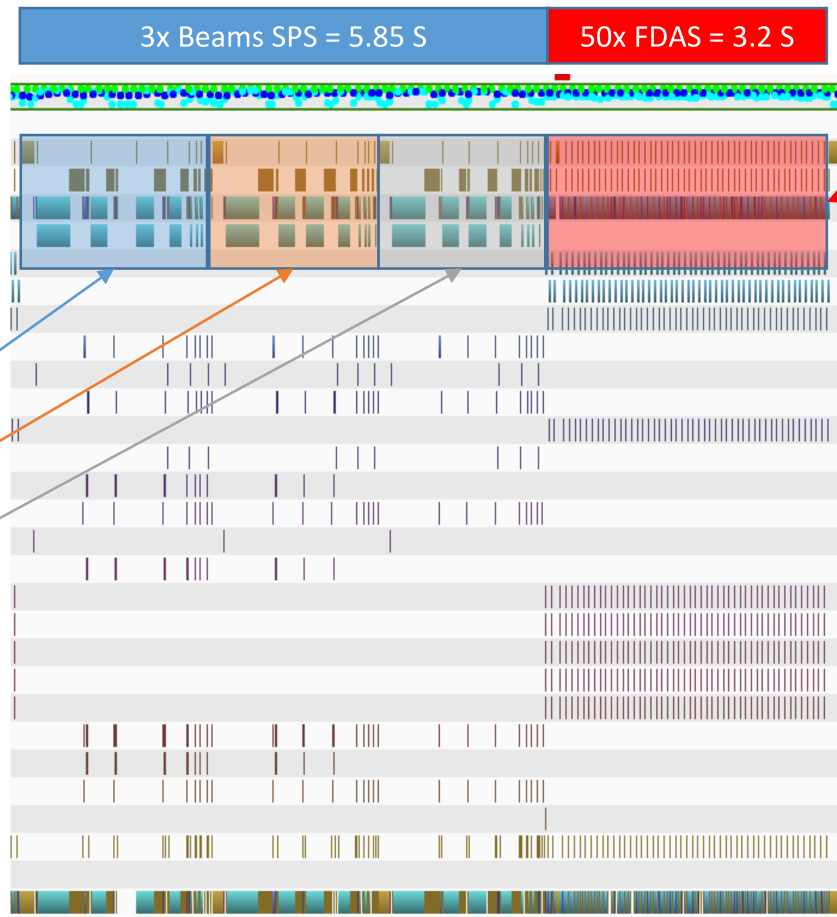
Oxford e-Research Centre

Conclusions - Prospects for SKA

3x Beams of SPS using 9.63S input chucks and 50 FDAS trials.
Total Processing taking about 9 seconds -> Faster than real-time.

The input data for Beams A, B and C is 9.63 Seconds of data collected by the telescope.

9.63(S)/0.000064(S)
= 150500 time samples
(each having 4096 channels)



Beam A

Beam B

Beam C

NVIDIA Profiler output from a run on a Tesla P100 GPU

This shows 50 full resolution FDAS trials being performed from a previous pointing: 2^23 samples using 96 templates.
NO HARMONIC SUM

Given that each observation is about 536 seconds long this means that it is possible to perform 536/9.7 x 50 = 2750 full resolution FDAS trials while performing SPS on 3x beams. Current work indicates that the harmonic sum will (at most) half this -> ***450 FDAS trials per beam***

Acknowledgments and Collaborators

University of Oxford
Mike Giles (Maths)
Aris Karastergiou (Physics)
Chris Williams (Physics)
Steve Roberts (Engineering)
Sofia Dimoudi (OeRC)
Nassim Ouannoughi (OeRC)
Karel Adamek (OeRC)
Jayanth Chennamangalam (Physics)
Griffin Foster (Physics)

University of Manchester
Ben Stappers
Mike Keith
Prabu Thiagaraj
Jayanta Roy
Mitch Mickaliger

NRAO/UVA
Scott Ransom

University of Bristol
Dan Curran (Electrical Engineering)
Simon McIntosh Smith (Electrical Engineering)

ASTRON
Cees Bassa
Jason Hessels

University of Opava
Jan Novotny

NVIDIA
Kate Clark
Tim Lanfear
Tom Bradley

Max Plank
Ewan Barr

AstroAccelerate

http://www.oerc.ox.ac.uk/projects/astroaccelerate