Image credit: Graham Allen

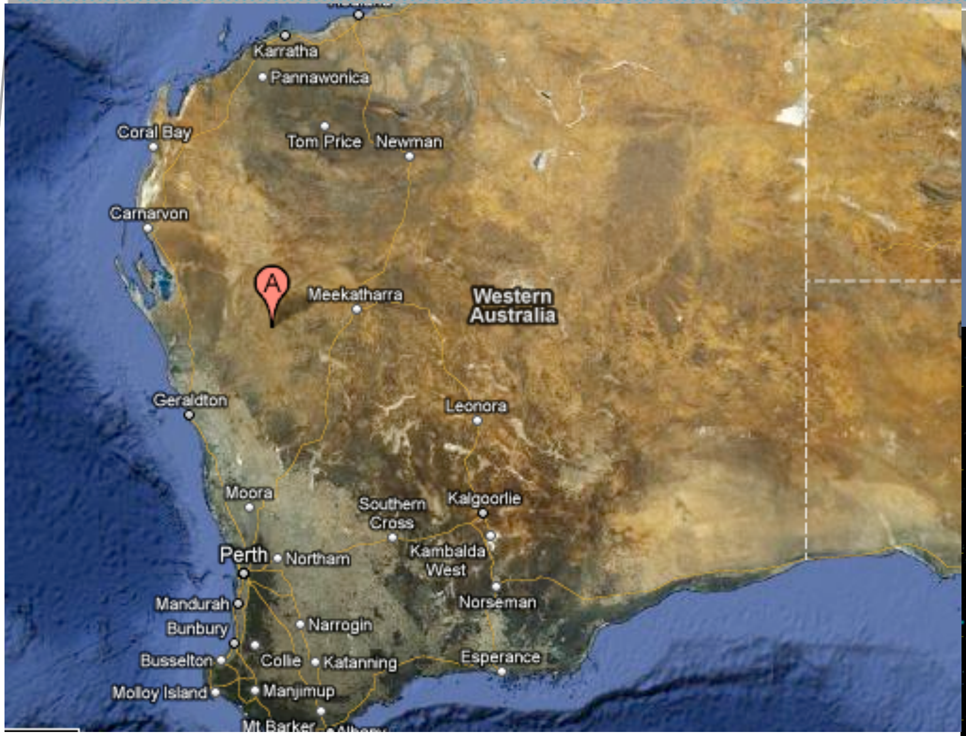
Calibration pipelines for ASKAP

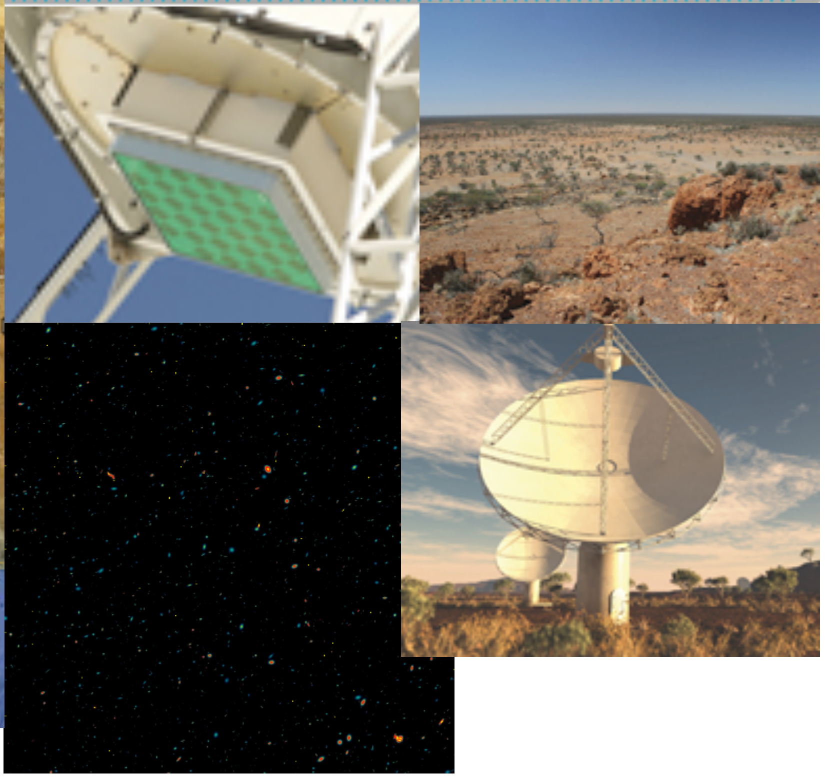
Max Voronkov
ASKAP Software Scientist
In collaboration with Ben Humphreys and Tim Cornwell

25th July 2011

CSIRO

ASKAP overview





http://www.atnf.csiro.au/projects/askap

ASKAP = Australian Square Kilometre Array Pathfinder

- Located at radio-quiet site approx. 300 km inland from Geraldton
- Six antennas are already on site
- Array of 36 12m antennas with phased array feeds (PAF)

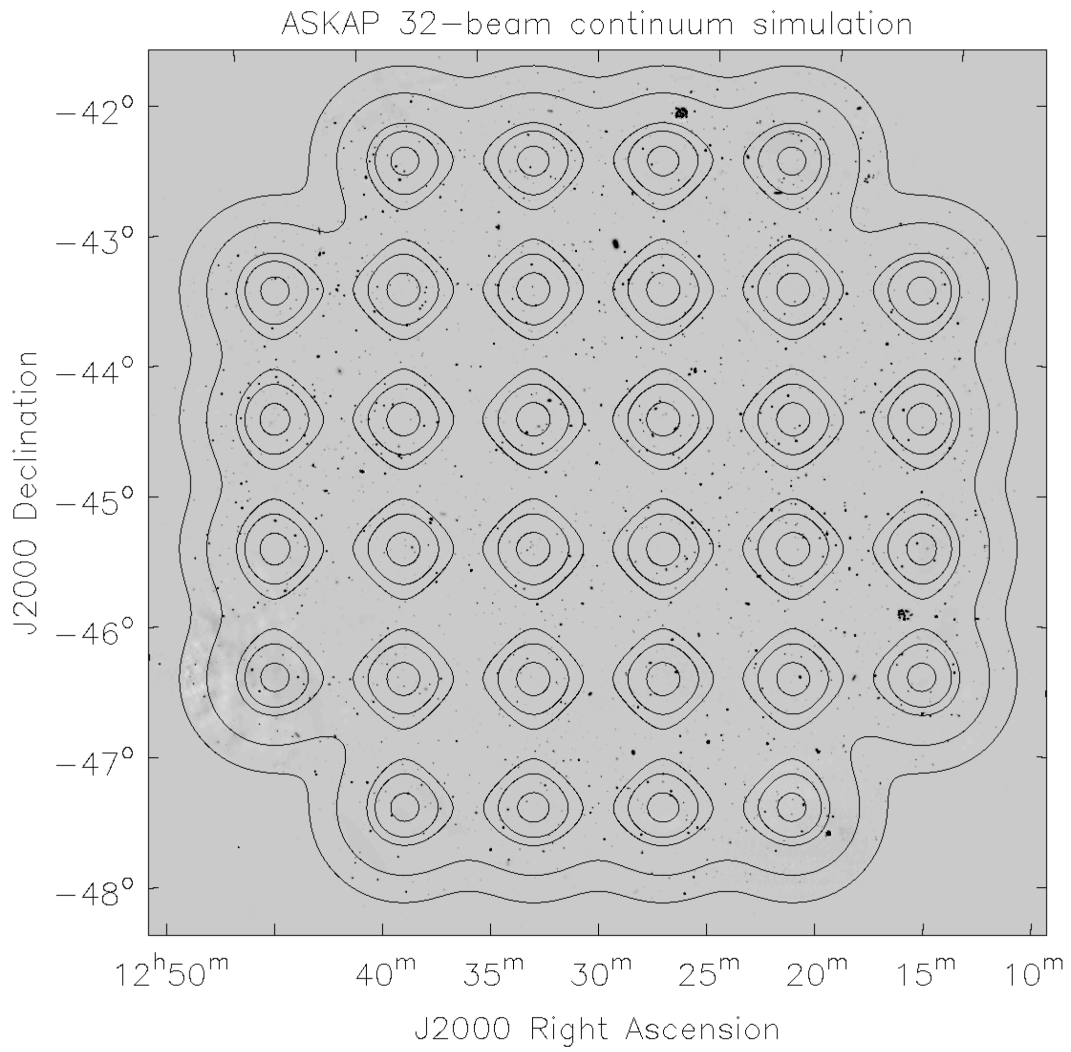
ASKAP is a wide-field of view instrument

Wide field of view
(30 square degrees)

↓

high data rate (3 GB/s)

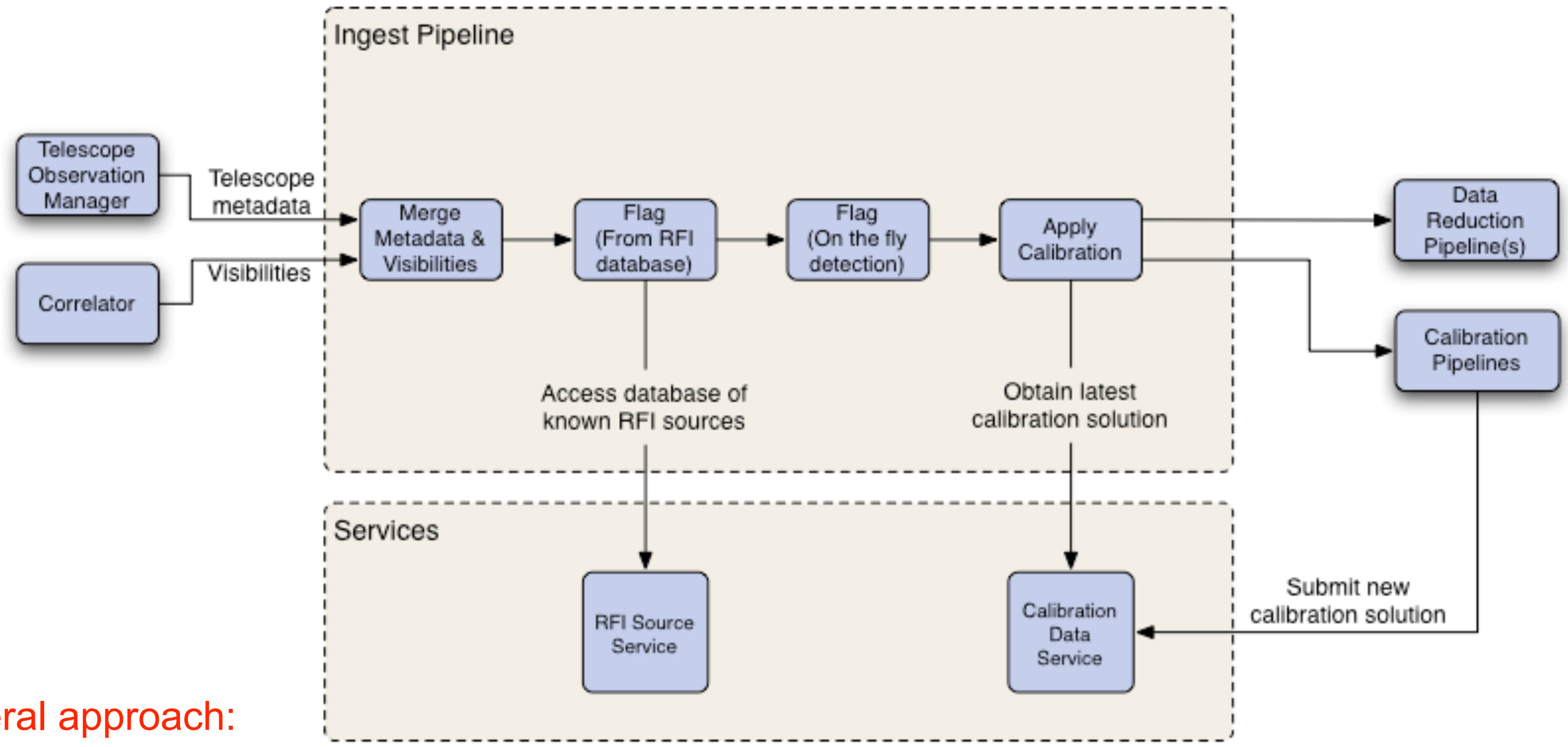
30 beams to fill field of view



What needs calibration?

- Visibility-plane effects
    - Complex gain per beam per antenna
    - Bandpass per beam per antenna, effectively this is a complex gain per fine-resolution frequency channel (18 kHz)
    - Polarisation leakage per beam per antenna (per 1 MHz?)
- Image-plane effects - hopefully not
    - Ionosphere is relatively benign at 1 GHz
    - Primary beam is fixed on the sky (3-axis antenna mount)
    - Pointing can be corrected on-the-fly, but should be fine as it is
    - PAF stability (synthetic beams) is still the biggest unknown. Peeling is expected to help. PAF element-based calibration is taken care of upstream.
- Operations-specific calibration - not something we do in real time
    - Antenna positions on the ground (baseline calibration)
    - Global pointing model (might be done per-antenna as part of commissioning)

Online calibration loop and forward prediction



Ingest Pipeline
Telescope Observation Manager
Telescope metadata
Correlator
Visibilities
Merge Metadata & Visibilities
Flag (From RFI database)
Flag (On the fly detection)
Apply Calibration
Data Reduction Pipeline(s)
Calibration Pipelines
Access database of known RFI sources
Obtain latest calibration solution
Services
RFI Source Service
Calibration Data Service
Submit new calibration solution

General approach:

- We keep the instrument well calibrated at all times
- New calibration solutions are fed back to the ingest pipeline (via calibration data service) to be applied on-the-fly
- A prototype solver was written. It will be the base for the BETA pipeline (and initially the calibration will be offline to keep the ingest pipeline simple).

Calibration is a least-square fit

• Understanding of the instrument allows us to relate true (or model) visibilities with the measured ones (non-linear relation on parameters)

For example (considering scalar case for simplicity):

$$V_i = f_i(p_1, p_2, \ldots, p_n, V_i^m)$$

linearise to get design equations:

$$V_i - f_i(p_1, p_2, \ldots, p_n, V_i^m) = \sum_{k=1}^{n} \left. \frac{\partial f_i}{\partial p_k} \right|_{(p_1, p_2, \ldots, p_n, V_i^m)} \delta p_k \qquad \text{or} \qquad V^{res} = A \delta P$$

Solution of normal equations gives an update to parameters:

$$A^H Q^{-1} V^{res} = (A^H Q^{-1} A) \delta P$$

• Need multiple iterations to converge to the correct parameters due to non-linearity

• Master-worker framework allows to distribute normal equations

Calculation of derivatives

- Calibration part of the measurement equation is known analytically
  - In principle, we could calculate all derivatives required for the Least-Square Fit in advance
  - Tedious to do manually, especially if we plan to do any research of the structure of these equations and change them from time to time
  - Numerical differentiation is an option, but has its own drawbacks
- We use automatic analytical differentiation in our code
  - Run-time analytical expansion of equations
  - Overheads are low as the parameter-dependent part of the measurement equation is typically rather simple
  - Same idea of automatic differentiation as in casacore's AutoDiff and SparseDiff classes
  - Our implementation (called ComplexDiff) has full support of complex parameters (and complex conjugation in equations) and works with string parameter names (handy in a parallel environment)

Automatic differentiation

- The main idea is to track derivatives through the equations from the point where their calculation is trivial

For

$$df = \frac{\partial f}{\partial x_1} dx_1 + \frac{\partial f}{\partial x_2} dx_2 + \cdots + \frac{\partial f}{\partial x_n} dx_n$$

and

$$dg = \frac{\partial g}{\partial x_1} dx_1 + \frac{\partial g}{\partial x_2} dx_2 + \cdots + \frac{\partial g}{\partial x_n} dx_n$$

it is possible to compute derivatives for any combination of f and g

for example:

$$d(fg) = gdf + fdg \qquad \text{and} \qquad d(\cos f) = -(\sin f)df$$

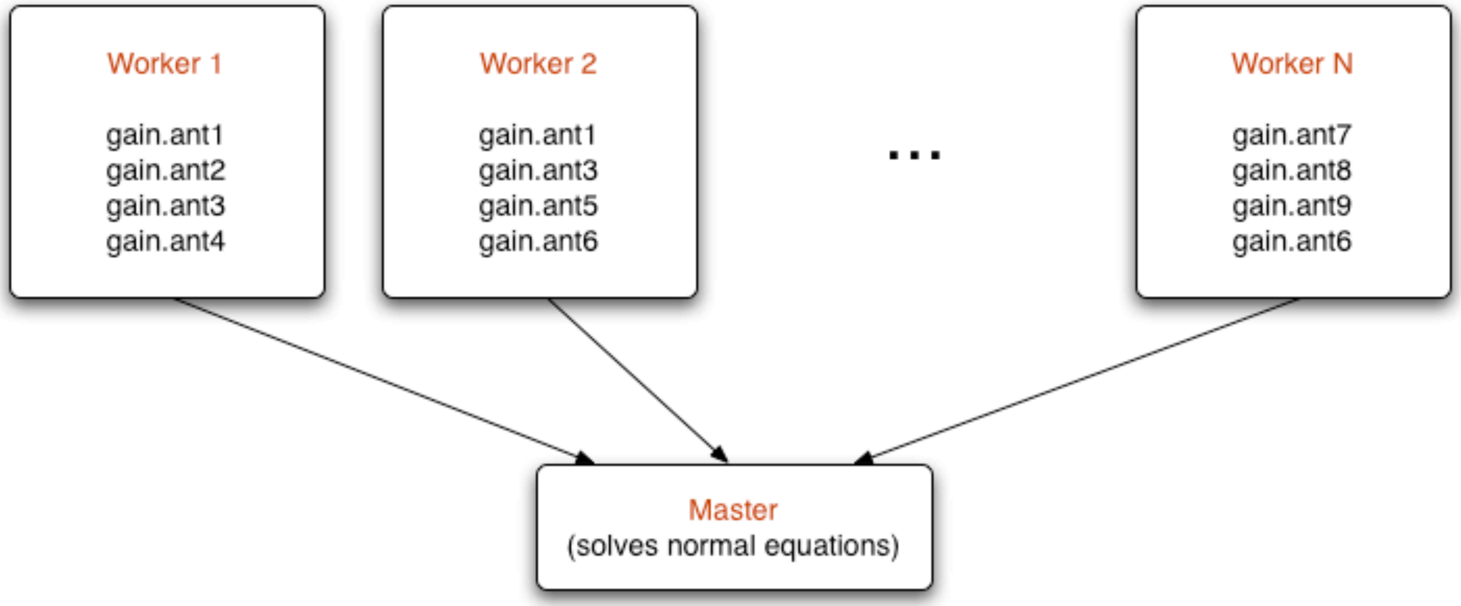
The full complex case requires carrying of:

$$\Re f, \Im f \quad \text{and} \quad \frac{\partial(\Re f)}{\partial(\Re x_i)}, \frac{\partial(\Im f)}{\partial(\Re x_i)}, \frac{\partial(\Re f)}{\partial(\Im x_i)}, \frac{\partial(\Im f)}{\partial(\Im x_i)} \qquad \forall i$$

Automatic differentiation with ComplexDiff

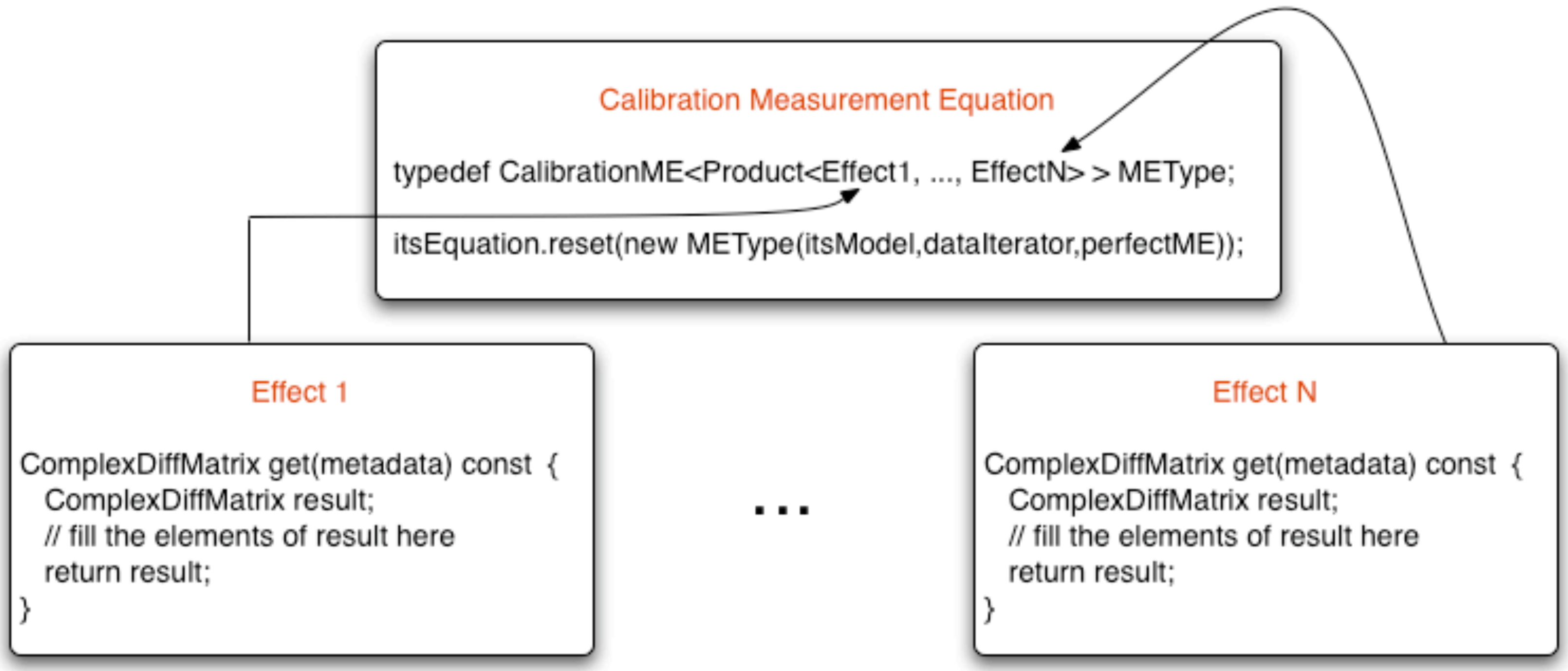
```
ComplexDiff g("par1", Complex(35., -15.)); // complex parameter
ComplexDiff f("par2", 0.5);  // real parameter
// some equation
ComplexDiff result = g * f + Complex(0., -2.1) * f + 2 * conj(g) + 1.;
// access to value
cout << result.value() << endl;
// access to derivatives
cout << result.derivRe("par1") << " " << result.derivIm("par1") <<
        result.derivRe("par2") << endl;
```

String-based indices are handy if equation calculation is distributed:



Implementation details of the calibration ME

$$\begin{pmatrix} I \\ Q \\ U \\ V \end{pmatrix} = \begin{pmatrix} m^1_{11} & m^1_{12} & m^1_{13} & m^1_{14} \\ m^1_{21} & m^1_{22} & m^1_{23} & m^1_{24} \\ m^1_{31} & m^1_{32} & m^1_{33} & m^1_{34} \\ m^1_{41} & m^1_{42} & m^1_{43} & m^1_{44} \end{pmatrix} \cdots \begin{pmatrix} m^N_{11} & m^N_{12} & m^N_{13} & m^N_{14} \\ m^N_{21} & m^N_{22} & m^N_{23} & m^N_{24} \\ m^N_{31} & m^N_{32} & m^N_{33} & m^N_{34} \\ m^N_{41} & m^N_{42} & m^N_{43} & m^N_{44} \end{pmatrix} \begin{pmatrix} I \\ Q \\ U \\ V \end{pmatrix}_{\mathrm{model}}$$



```
typedef CalibrationME<Product<Effect1, ..., EffectN> > METype;

itsEquation.reset(new METype(itsModel,dataIterator,perfectME));
```

```
ComplexDiffMatrix get(metadata) const {
  ComplexDiffMatrix result;
  // fill the elements of result here
  return result;
}
```

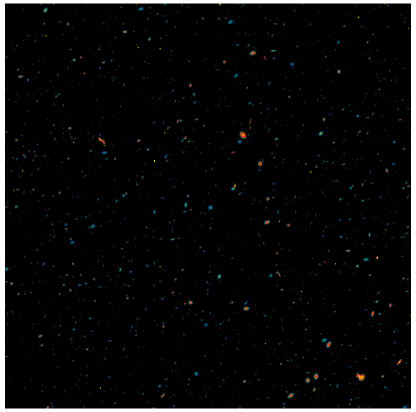
```
ComplexDiffMatrix get(metadata) const {
  ComplexDiffMatrix result;
  // fill the elements of result here
  return result;
}
```

Individual effects return Mueller matrices based on the given metadata

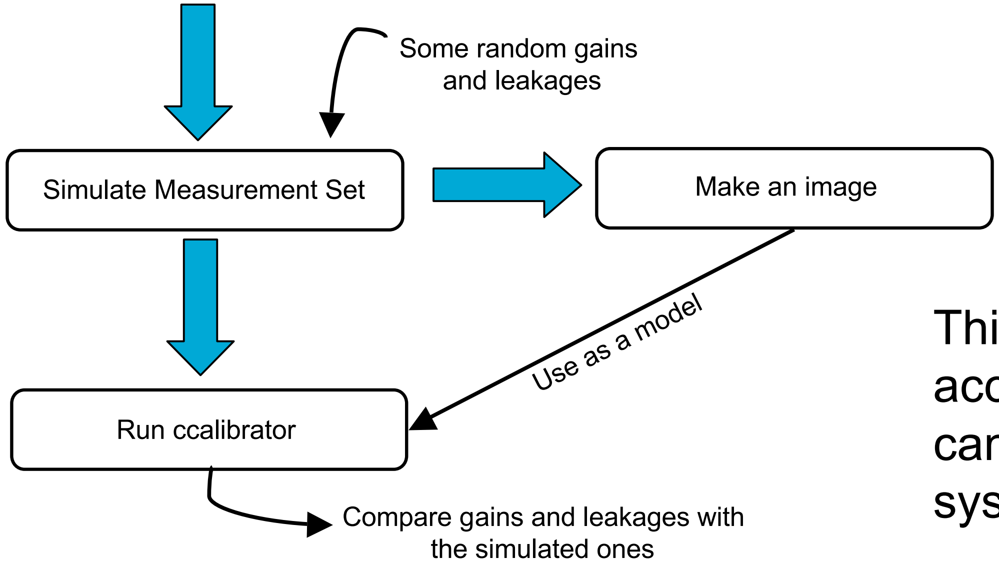
Performance tests

Calibration has to keep up with observations

- deliver the solution faster then the required integration time



- Simulated full ASKAP with 36 antennas
- Full Stokes observations
- 11 5-minute scans at different hour angles
- But a single 1-MHz spectral channel and 1 beam
- Similar data volume to the amount of data a single worker will see with the actual telescope in 5 min.


Some random gains and leakages
Simulate Measurement Set
Make an image
Use as a model
Run ccalibrator
Compare gains and leakages with the simulated ones

On our Dell R710 it took 666 seconds to run ccalibrator!

This is too long even taking into account the initial setup which can be factored out in the final system.

Pre-averaging calibration

- Aim to achieve calibration with just one iteration over data

Use the fact that the equation is linear on model visibilities. Considering the scalar case again for simplicity:

$$V_i = f_i(p_1, p_2, \ldots, p_n, V_i^m) = \tilde{f}_i(p_1, p_2, \ldots, p_n) V_i^m$$

Now divide both sides by the model visibilities

$$\frac{V_i}{V_i^m} = \tilde{f}_i(p_1, p_2, \ldots, p_n)$$

This division stops fast variations in both time and frequency (makes the model equivalent to a point source in the phase centre). We can now average in time and frequency

$$< \frac{V_i}{V_i^m} >=< \tilde{f}_i(p_1, p_2, \ldots, p_n) >= \tilde{f}_i(p_1, p_2, \ldots, p_n)$$

This is not a new approach, e.g. casa uses something similar

- It becomes less trivial in the non-scalar case (full polarisation)!
- We also want to retain our general calibration framework

CSIRO

Different approach to pre-averaging

$$V_i = f_i(p_1,p_2,\ldots,p_n,V_i^m) = \tilde{f}_i(p_1,p_2,\ldots,p_n)V_i^m$$

The only assumption is the structure the of the measurement equation

$\xrightarrow{\text{Linearise}}$

$$V^{res} = A\delta P = \begin{pmatrix} V_1^m & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & V_n^m \end{pmatrix} \tilde{A}\delta P$$

$\xrightarrow{\text{Form Normal Equations}}$

$$\tilde{A}^H \begin{pmatrix} (V_1^m)^* & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & (V_n^m)^* \end{pmatrix} Q^{-1} V^{res} = \tilde{A}^H \begin{pmatrix} (V_1^m)^* & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & (V_n^m)^* \end{pmatrix} Q^{-1} \begin{pmatrix} V_1^m & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & V_n^m \end{pmatrix} \tilde{A}\delta P$$

Normal matrix element:

$$\alpha_{lm} = \sum_i \omega_i \left(\frac{\partial \tilde{f}_i}{\partial p_l}\right)^* \left(\frac{\partial \tilde{f}_i}{\partial p_m}\right) (V_i^m)^* V_i^m = \sum_g \left(\frac{\partial \tilde{f}_g}{\partial p_l}\right)^* \left(\frac{\partial \tilde{f}_g}{\partial p_m}\right) \underbrace{\sum_{i \ni g} \omega_i (V_i^m)^* V_i^m}_{\text{buffered}}$$

Data vector element:

Accumulated on the 1st iteration

$$\beta_l = \sum_i \omega_i \left(\frac{\partial \tilde{f}_i}{\partial p_l}\right)^* (V_i^m)^* (V_i^{obs} - \tilde{f}_i V_i^m) = \sum_g \left(\frac{\partial \tilde{f}_g}{\partial p_l}\right)^* \left(\overbrace{\sum_{i \ni g} \omega_i (V_i^m)^* V_i^{obs}}^{\textit{buffered}} - \tilde{f}_g \overbrace{\sum_{i \ni g} \omega_i (V_i^m)^* V_i^m}^{\textit{buffered}} \right)$$

Vector case (full Stokes)

In the full Stokes case $V_i = \tilde{f}_i(p_1, p_2, \ldots, p_n) V_i^m$ is replaced by:

$$\begin{pmatrix} V_{1i} \\ V_{2i} \\ V_{3i} \\ V_{4i} \end{pmatrix}_i = \begin{pmatrix} \tilde{f}_i^{11}(p_1,p_2,\ldots,p_n) & \tilde{f}_i^{12}(p_1,p_2,\ldots,p_n) & \tilde{f}_i^{13}(p_1,p_2,\ldots,p_n) & \tilde{f}_i^{14}(p_1,p_2,\ldots,p_n) \\ \tilde{f}_i^{21}(p_1,p_2,\ldots,p_n) & \tilde{f}_i^{22}(p_1,p_2,\ldots,p_n) & \tilde{f}_i^{23}(p_1,p_2,\ldots,p_n) & \tilde{f}_i^{24}(p_1,p_2,\ldots,p_n) \\ \tilde{f}_i^{31}(p_1,p_2,\ldots,p_n) & \tilde{f}_i^{32}(p_1,p_2,\ldots,p_n) & \tilde{f}_i^{33}(p_1,p_2,\ldots,p_n) & \tilde{f}_i^{34}(p_1,p_2,\ldots,p_n) \\ \tilde{f}_i^{41}(p_1,p_2,\ldots,p_n) & \tilde{f}_i^{42}(p_1,p_2,\ldots,p_n) & \tilde{f}_i^{43}(p_1,p_2,\ldots,p_n) & \tilde{f}_i^{44}(p_1,p_2,\ldots,p_n) \end{pmatrix} \begin{pmatrix} V_{1i}^m \\ V_{2i}^m \\ V_{3i}^m \\ V_{4i}^m \end{pmatrix}$$

Oleg's tensor-based measurement equation formalism could probably help to deal with these extra dimensions in a neat way, but it is clear that the main implication is that one needs to buffer all cross-polarisation products now:

$$\sum_{k \ni g} \omega_{ijk} (V_{ik}^m)^* V_{jk}^m$$

4 real and 6 complex numbers per group with the same parameter dependence (i.e. per baseline)

$$\sum_{k \ni g} \omega_{ijk} (V_{ik}^m)^* V_{jk}^{obs}$$

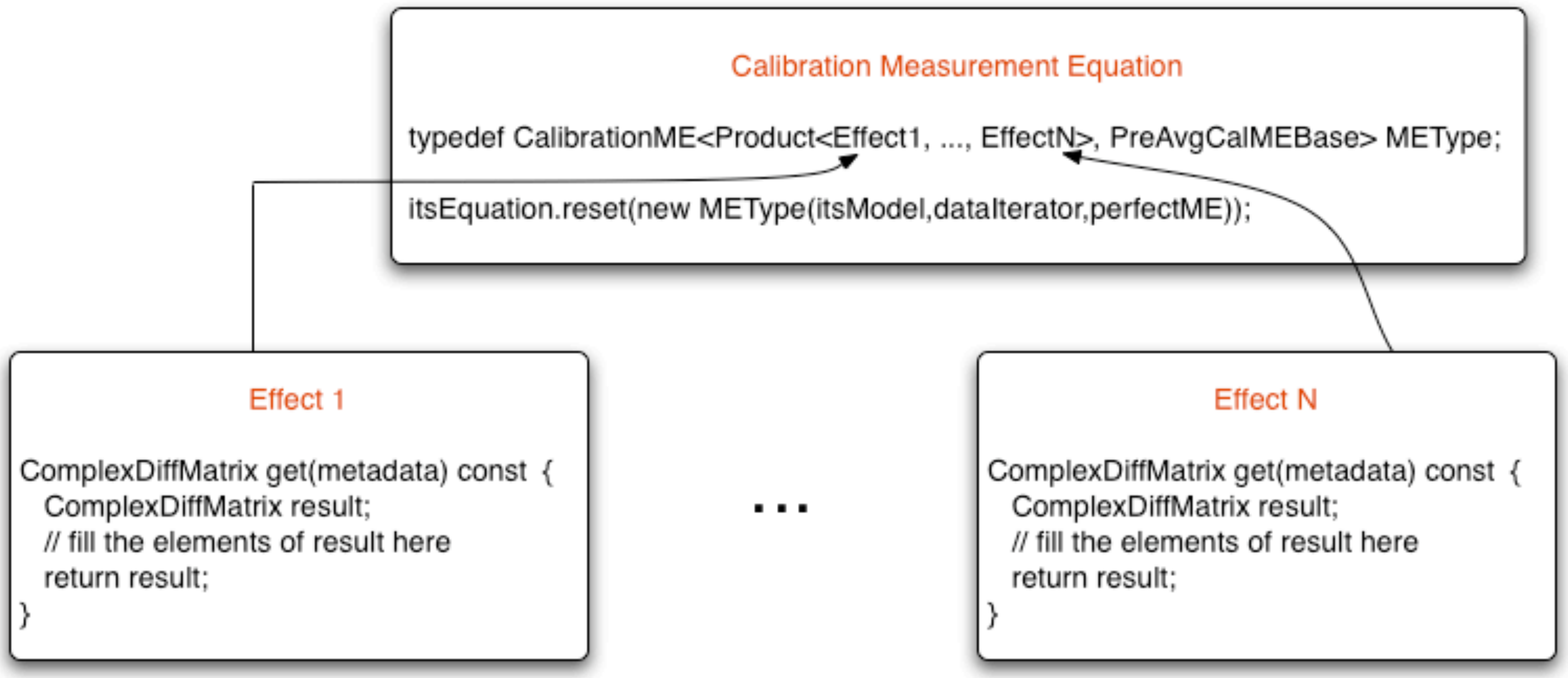
16 complex numbers per group

In total, about 0.4 Mb per worker

Physical interpretation: multiplication by the conjugate of the model visibilities stops fast variations.

What we've got at the end

Same performance test as before was done in 23 seconds as opposed to 666 seconds for the brute force least-square fit (and only 11 seconds if polarisation leakages are not solved for)



```
typedef CalibrationME<Product<Effect1, ..., EffectN>, PreAvgCalMEBase> METype;

itsEquation.reset(new METype(itsModel,dataIterator,perfectME));
```

```
ComplexDiffMatrix get(metadata) const {
  ComplexDiffMatrix result;
  // fill the elements of result here
  return result;
}
```

```
ComplexDiffMatrix get(metadata) const {
  ComplexDiffMatrix result;
  // fill the elements of result here
  return result;
}
```

Buffering happens behind the scene, move to pre-summing is simple

Additional issues

- The suggested pre-summing approach is quite general
    - Works for any effect which can be represented by Mueller matrix as long as the equations can be grouped as expected
- Polarisation calibration of a classical Alt-Az telescope is one of the cases where the grouping per baseline is not enough
    - Parallactic angle rotation couples parameters in a different way at different hour angles
    - The solution is to buffer polarisation products separately for each such scan
    - We have this functionality in our code because we may end up using the sky rotation control for the ASKAP antennas to assist polarisation calibration
- The computation of data vector often involves subtraction of two large numbers (two sums)
    - Numerical precision issues have to be watched
    - No problems found so far

CSIRO

Summary

- Pre-summing approach to build normal equations is very effective
  - Factor of 20 increase in performance on top of brute force least-square fit approach
  - No approximations made
  - It is the structure of equations which allows us to do it this way
- ASKAP calibration code includes
  - Autodifferentiation supporting full complex case and distributed calculations of equations
  - Reuse of the master-worker parallel framework designed for imaging
  - Neat way to specify measurement equation

Australia Telescope National Facility
Max Voronkov
Software Scientist (ASKAP)

Phone: 02 9372 4427
Email: maxim.voronkov@csiro.au
Web: http://www.atnf.csiro.au/projects/askap/

www.csiro.au

Thank you