Astronomical Data Analysis Software and Systems IX
ASP Conference Series, Vol. 216, 2000
N. Manset, C. Veillet, and D. Crabtree, eds.

The ACSIS Data Reduction System

J. F. Lightfoot, W. R. F. Dent

U.K. Astronomy Technology Centre, Royal Observatory Edinburgh, Blackford Hill, Edinburgh EH9 3HJ, United Kingdom

A. G. Willis, G. J. Hovey

Dominion Radio Astrophysical Observatory, P.O. Box 248, Penticton, B.C., Canada V2A 6K3

Abstract. ACSIS is a digital auto-correlator being built at the DRAO to handle array heterodyne receivers on the JCMT. This paper describes the online reduction, archiving and display system that will be delivered with the instrument. Spectrum calibration and gridding into cubes will be performed by a system of distributed objects running on an array of Linux PCs, offering high performance at low cost. The system configuration is recipe-driven to allow for changing observing methods and computing hardware. The system is built using AIPS++ classes and objects communicate via the Glish message bus.

1. Introduction

ACSIS is an auto-correlator intended for use with multiple beam receivers on the James Clerk Maxwell Telescope. The instrument is being designed and built at the DRAO[1], Penticton, in collaboration with the UKATC, Edinburgh, and Joint Astronomy Centre, Hawaii.

At the heart of the correlator are 8 correlator modules, each containing 4 correlator boards. A correlator board can be configured to accept a signal with 1 GHz or 250 MHz bandwidth and can measure its auto-correlation function (ACF) with up to 4096 lags. Each correlator module contains a micro computer which reads and coadds ACFs from the correlator chips, eliminating inconsistent lag data as it does so. The shortest time between data dumps from the correlator is 50 ms.

The reduction system delivered with ACSIS will be able to calibrate and grid the measured data into a data cube as they are taken. The reduction will be good enough that the result can be used to assess the quality of the observation. Ideally it will be so good as to require no further reduction off line!

The current plan is for a 16 mixer B-band (330-370 GHz) receiver, HARP, to be built as the front-end for ACSIS.

---

[1] `http://www.drao.nrc.ca/science/jcmt_correlator`

2. Recipe-Driven Data Taking

The coordination of telescope, receiver and ACSIS during an observation will be managed by a Java tool called the TODD (Telescope Observation Designer and Driver) and a small computer called the Real-Time Sequencer (RTS). The TODD executes an observing recipe, sending out commands to the various sub-systems to perform the non real-time coordination required. The RTS, programmed by the TODD, generates the hard real-time signals used to coordinate tightly coupled sub-systems during a sequence of correlator integrations. Each integration result and sub-system state is tagged with a unique 'sequence' number that will be used by the reduction system to knit together a data record containing both the data and a description of the system state at the time.

By developing new TODD scripts it is relatively easy to implement new observing modes.

3. The Aims of the Reduction System

- It must be easy to modify the reduction process to cater for new observing modes.
- The calibration of mm and sub-mm data is difficult because of the poor and variable transmission of the atmosphere. The observer must be able to specify the reduction algorithm best suited to the observing conditions.
- The system must be able to handle data from different front-end receivers which may, for example, differ in their calibration procedures.
- At the correlator's peak operating speed the raw ACFs flow out at the dump rate (20 Hz) * no. lags per ACF (4096) * no. samplers (32) * Bytes per lag (4) = 10.5 MB/s. The reduction must keep up.
- The system must be able to archive the raw data to disk or tape, in case re-reduction is required.
- The observer must be able to track changes in atmospheric conditions by watching, for example, a continuous display of the system temperature.
- The observer must be able to monitor system health by seeing, for example, a continuous display of the receiver temperature. Health data should be stored in a database to track long term trends in system performance.

4. Reduction Hardware

To achieve the computing performance required at an affordable cost we will use a dedicated 8 node Beowulf cluster. Data will enter the system via dedicated 100MB/s Ethernet connections between the micro computer in each correlator module and a cluster partner.

Apart from some ancillary information, such as load temperatures from the front-end receiver, all stages in the reduction process before gridding can be performed using just the data passing through each correlator module. The gridding, however, requires data from all modules to be combined together, the necessary data exchange taking place over the Beowulf network and link switch.

In some observing modes the calibration of data must await the taking of a subsequent calibration measurement and the system will have to buffer incoming spectrum data until that occurs. The longest waiting time is 5 minutes, requiring 0.4 GB of storage. In addition, space is needed to store the gridded datacube result of an observation. The largest cube will be 1024 x 1024 (spatial dimension) x 512 (frequency dimension) which, spread over 8 machines, would require 0.3 GB per machine. To hold 2 spectrum buffers and one cube buffer each machine will need 1.1 GB of memory.

5. Recipe-Driven Reduction Software

Reduction flexibility requires that the function and distribution of reduction processes across the Beowulf cluster, and the data flow between them, be controlled by a reduction recipe. A different recipe will be used for each observing mode.

Reduction efficiency requires that the reduction processes be compiled code rather than interpreted scripts. Our tasks are written in C++ and make heavy use of code available from the AIPS++[2] class library and the Parkes MultiBeam[3] Project. Efficiency also demands that the reduction be performed without storing partly reduced data to intermediate files - data must instead flow directly from one reduction process to the next.

The reduction recipe for a given mode is held in an ASCII file and is written in a simple C-like language. As in C you can write comments, define macros, "include" files and vary the recipe using conditional statements; functionality that is provided by the GNU C pre-processor. "Include" files will allow recipes to be developed from modules; for example, one module defining the cluster machines to be used, a second defining standard objects to calculate the system temperature, etc.

The reduction system is constructed from the following objects:

1. **Sync task**. This is a built-in part of the reduction system, not under recipe control, upon which the rest of the system is founded. In each cluster machine there will always be a Sync task running, its job to receive DRAMA data from the diverse parts of the system (the correlator module, telescope system, front-end receiver, etc.), assemble them into a Glish data record containing both the data and the system state, and forward it to the objects that will do the reduction.
2. **Real-Time Display**. The job of the RTD is to display whatever data the observer needs to see to monitor system health and assess the quality of an observation. To this end it can interrogate the reduction system to find out what Reducer objects are present and can display data from any of them as the observer chooses. The RTD is another built-in part of the system, whose creation and destruction will not be under recipe control but whose configuration will be, i.e. its default display for a given observing mode.
3. **Reducer** object. This is the smallest building block in the recipe-controlled part of the reduction system. Each such object will receive data from either the Sync task or from other Reducer objects, perform a recipe-specified series of reduction operations on them, then send them to the

[2] `http://aips2.nrao.edu/docs/aips++.html`

[3] `http://www.atnf.csiro/au/research/multibeam/multibeam.html`

next object in the data path. The reduction performed by each object, its location in the reduction hardware, and its links with other Reducer objects, are entirely controlled by the recipe.

4. **Archiver** object. This is like a Reducer object but its job is to archive incoming data to disk.
5. **ReducerProcess** object. This maps to a real process present on one of the reduction machines. It acts as a "container" for a number of Reducer or Archiver objects. Which objects a ReducerProcess contains, how the ReducerProcess objects are distributed across the cluster, and how data flows between them are all under recipe control. This level of granularity allows systems to be built with reduction and archiving split into separate processes or not (as may be required for convenience or performance reasons) and with the number of archiving processes used for a particular type of data matched to the data volume (e.g. raw data will need more than system health data).

Once configured at the start of an observation the reduction system structure is fixed for that observation. The reduction process itself is data driven, triggered by the arrival of data records at a Sync task. Thereafter the self-describing data record plus the system configuration determine which Reducer and Archiver objects are called and how the data are reduced and stored.

Normal termination of an observation will occur when a data record describing itself as the last arrives from the Sync task. As this passes through the Reducer and Archiver objects they will close data files and shut down.

If an error occurs in any object then it will be reported to the TODD controlling the observation, which will issue commands to terminate data taking.

6. Implementation

Recipes will be "compiled" by a program developed using the GNU tools Bison and Flex. The compiler will check that the recipe is self-consistent, with no undefined or unused objects or circular definitions. It will then generate 2 files; the first a Glish "wiring" procedure to initialize and link together the processes at the start of the observation; the second a simplified representation of the various ReducerProcess types and the names and functions of their constituent Reducer and Archiver objects, to be used by the constructors of the ReducerProcesses.

The RT Display will be implemented using Glish/Tk for GUI construction, the free version of Qt[4] for line graphics and probably the AIPS++ "viewer" tool for image display. Qt was found to be significantly more efficient than the Glish/PGPLOT package available with standard Aips++ and was thus preferred in view of our need for high performance.

[4] `http://www.troll.no/dl`