Astronomical Data Analysis Software and Systems XIII
ASP Conference Series, Vol. 314, 2004
F. Ochsenbein, M. Allen, and D. Egret, eds.

Adapting the BIMA Image Pipeline for Miriad Using Python

David M. Mehringer and Raymond L. Plante

National Center for Supercomputing Applications, University of Illinois Urbana-Champaign, Urbana, IL 61801

Abstract. Through our experience using AIPS++ in the BIMA Image Pipeline, we found that a sophisticated scripting environment is crucial for supporting an automated pipeline. Miriad V4, now in development, introduces support for calling Miriad programs from a Python environment (referred to as Pyramid). We are creating processing recipes using Miriad through Python that can be used with the BIMA Image Pipeline. As part of this work, we are prototyping tools that could be integrated into Pyramid. These include two Python classes, *UVDataset* and *Image* for examining the contents of Miriad datasets. These simple tools have allowed us to recast our Pipeline using Miriad in only a couple of months. Python recipes are used for such things as determining line-free channels for continuum subtraction and determining if data will benefit from self-calibration. We are currently using the Pipeline to do massive processing of hundreds of tracks of archival data using NCSA's Teraflop IA-32 Linux cluster.

1. Introduction

The BIMA[1] Image Pipeline is part of the BIMA Data Archive[2] and is a system for automated processing of BIMA data after they have been transferred from the Array located near Hat Creek, CA to NCSA, ingested, and archived. This processing includes calibration, self-calibration, continuum subtraction, and imaging of target datasets and calibrators. The products of the processing are also ingested and archived in the BIMA Data Archive where they can be retrieved by astronomers. At this point, the processed data products are meant to give astronomers a "first look" at their data. However, as new processing recipes are developed, we foresee that the data products will approach publication quality and will therefore reduce the amount of processing that the end user will have to do on his or her desktop.

[1] `http://bima.astro.umd.edu/`

[2] `http://bimaarch.ncsa.uiuc.edu`

We have recently re-implemented our Pipeline in Python[3], using Miriad[4] as the underlying astronomical data processing engine. This paper discusses this development and its results.

2. Importance of Using a Scripting Language

Our initial implementation of the BIMA Image Pipeline was using AIPS++[5] as the astronomical data processing engine. We found the powerful scripting language used by this package, called Glish[6], to be immensely useful for quickly and efficiently constructing data processing scripts. Thus, it was clear that we would need a powerful scripting language for our new implementation.

Use of a scripting language provides several benefits. It provides a means of rapid development. Because there are no compilation steps, the write → test → debug cycle can proceed quickly. In our case, we were able to write a complex, fully functional pipeline in only a couple of months by writing a Python layer which calls Miriad tasks for the processing of astronomical data. In addition, a scripting language provides a relatively easy way for end users to develop their own recipes. Because the learning curves for scripting languages tend to be significantly shallower than for compiled languages, the cost of code development for end users is relatively small. Our experience with AIPS++ shows that many users have been able to quickly implement complex algorithms using Glish. Because our goal is to have users write and submit processing recipes for the BIMA Image Pipeline so they may be used by the larger community, it is important that we provide a scripting language interface to allow this.

3. Why We Chose Python, the One-Stop Scripting Language Solution for All Our Data Processing Needs

We decided to use Python as the scripting language for the BIMA Image Pipeline for several reasons. Python supports both the procedural and object oriented code paradigms, and therefore it is easy for users familiar with one or both of these to implement algorithms. Python has a rich, yet simple to use, set of data types such as various types of sequences (lists, tuples, etc.) and dictionaries (often called hashes in other languages). Furthermore, Python allows these types to be nested *ad infinitum*, so, for example, one could have a dictionary which contains lists of dictionaries, integers, strings, or any combination of these. We have found this unlimited flexibility to be quite important in our development of recipes. Because Python is open source and has a large user community, it benefits from having a mature collection of standard libraries. These include libraries for regular expression manipulation, system command execution, mathematical function evaluation, XML parsing, etc. Thus, we do not have to

---

[3] `http://www.python.org/`

[4] `http://bima.astro.umd.edu/miriad/intro.html`

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

[6] `http://aips2.nrao.edu/docs/glish/glish.html`

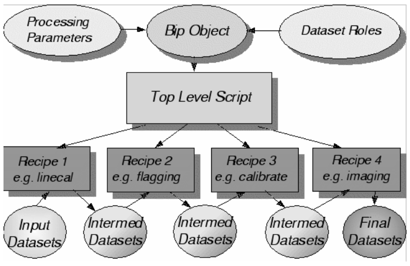

Figure 1. Architecture of the BIMA Image Pipeline

re-invent the wheel and can concentrate on developing astronomical processing recipes. Python provides simple but powerful mechanisms for manipulating lists (*i.e.,* arrays) via slicing and function mapping with its built-in *map()* function. This is especially important for producing code to handle astronomical images. In addition, Python provides a command line interface which allows for interactive (and thus rapid) development, testing, and execution of code. Finally, this language provides a means of interfacing to compiled code (*e.g.,* C) libraries. We plan on developing such a Python interface to Miriad libraries for improved performance.

4. Python Classes for Data Access

One of the most important aspects of developing any data processing pipeline is that it must be easy to access the metadata for the various datasets. In radio interferometry, metadata are used to determine processing parameters such as image extents and image pixel sizes, the number of spectral windows for which images must be created, etc. To make accessing metadata simple, we wrote two Python classes, called *UVDataset* and *Image*, which provide APIs for access of metadata from these types of datasets. Methods allow retrieval of such information as the number of spectral windows, system temperatures, and antenna positions for *uv*-datasets and image dimensions, pixel dimensions, and statistics for images.

5. BIMA Image Pipeline Architecture

The architecture of the BIMA Image Pipeline is depicted in Figure 1. The *bip* (BIMA Image Pipeline) object holds relevant fields which are used throughout the run by the top-level script and the processing recipes it calls. The processing parameters are contained in a text file as *name=value* pairs. These parameters control how various recipes are executed. Roles information about the various input datasets are contained in another *name=value* pair text file. The roles describe how the datasets are to be used during processing (e.g., target source, phase calibrator, flux calibrator, etc.). The top-level script calls various processing recipes in order. Most recipes take input datasets (usually the output from a previous recipe) and create output datasets. Each recipe is essentially a Python function which is passed a dictionary describing what processing parameters it should use and returns a dictionary describing its results (such as on which input datasets it was successful, the names of the output datasets which were generated, etc.). The top-level script makes decisions based on this information such as whether or not to run the next recipe, what intermediate datasets should be used when the next recipe is run, etc.

6. Parallel Processing of Tracks

Using NCSA's IA-32 cluster[7] which has a peak performance of 1 Teraflop, we have processed several hundred tracks of BIMA data. This is typically done by processing about 100 tracks at a time using 32 processors. This processing is controlled via a master *csh* script. A CPU is dedicated to processing a single track at a time. Unprocessed tracks are held in a queue. When a track has finished being processed, the CPU which has been freed is sent the next track in the unprocessed track queue. In the future we plan to implement at the Miriad level applications which have been written to take full advantage of cluster technology. One of the first such applications we will implement will be a parallelized version of CLEAN, which is an algorithm for deconvolving images. Each channel of a multi-channel dataset can be deconvolved independently of the other channels. This problem is considered to be *embarrassingly parallel*, and so is a good first step to taking full advantage of modern clusters.

7. User Access of Processed Data

Users (astronomers) may access the products of processing runs in the same way they access raw data in the BIMA Data Archive. The user simply searches our database by keying her project id, investigator name, and/or numerous other search parameters into a web form. The page which is returned contains all datasets matching the query parameter(s), and from this page, the user may download as many datasets as she wishes using our DaRT[8] download client (Mehringer & Plante 2000), or she may proceed to pages with more detailed information and download datasets one at a time. Many types of processing products are archived, including deconvolved wide band images and spectral line cubes of all target sources in FITS format, calibrated target datasets *uv* data which has had the continuum subtracted, calibration solutions used to calibrate the target datasets, various plots (in Postscript format) of images and calibration solutions.

References

Mehringer, D. M. & Plante, R. L. 2000, in ASP Conf. Ser., Vol. 216, ADASS IX, ed. N. Manset, C. Veillet, & D. Crabtree (San Francisco: ASP), 703

---

[7] `http://www.ncsa.uiuc.edu/UserInfo/Resources/Hardware/IA32LinuxCluster/`

[8] `http://monet.ncsa.uiuc.edu/ADC/DaRT/`