SIP: a modern, flexible data pipeline

P. Barmby, Z. Wang, M. Marengo, M.L.N. Ashby

Harvard-Smithsonian Center for Astrophyiscs, 60 Garden St., Mailstop 65, Cambridge, MA 02138 USA

ABSTRACT

IRAC, the Infrared Array Camera on the Spitzer Space Telescope, generated well over 150,000 images during the in-orbit checkout and science verification phase of the mission. All of these were processed with SIP, the SAO IRAC Pipeline. SIP was created by and for the members of the IRAC instrument team at the Smithsonian Astrophysical Observatory, to allow short-timescale data processing and rapid-turnaround software testing and algorithm modification. SIP makes use of perl scripting and data mirroring to transfer and manage data, a mySQL database to select calibration data, and Python/numarray to process the image data; it is designed to run with no user interaction. SIP is fast, flexible, and robust. We present some 'lessons learned' from the construction and maintenance of SIP, and discuss prospects for future improvement.

Keywords: data pipelines, data management, image processing

1. INTRODUCTION

The *Spitzer Space Telescope* (formerly SIRTF)[1] was launched on August 25, 2003. About a week later, its Infrared Array Camera (IRAC)[2] was first powered-on and the first science images were collected. Over the next three months, the In-Orbit Checkout (IOC) and Science Verification (SV) phases of the *Spitzer* mission involved detailed testing of IRAC as well as the other two science instruments. IRAC was powered on about 20 times for campaigns lasting from a few hours to a few days. Each of these campaigns involved taking many IRAC images, for testing such things as the telescope focus, instrument calibration, and pointing control system. The results of these initial tests are described elsewhere.[4] All of the IRAC images produced (over 150000 by the end of Science Verification, late November 2003) had to be processed to remove instrumental signatures (array gain and offset, electronic effects) and add contextual information before science and engineering analysis by the instrument team could begin. SIP, the SAO IRAC Pipeline, was created to perform this processing.

The Spitzer Science Center (SSC) controls the official IRAC pipeline, which performs processing steps similar to but not identical to SIP. The decision to create a separate pipeline primarily for the instrument team's use was based on two factors. One was rapid turn-around: because the SSC pipeline produces the official version of *Spitzer* data, it is under strict configuration control and cannot usually be changed on a short timescale (a few days or less). We anticipated that many modifications would be needed in the early days of the mission as we learned about the on-orbit performance of the instrument, and as bugs in the pipeline were discovered and fixed. The second reason was to provide a testbed that could be used to independently verify the SSC pipeline output. Before launch it was unclear how long it might take for the SSC pipeline to be run, and we wanted to have pipeline output available as soon as possible. With completely separate data processing, we expected to be able to compare output with the SSC pipeline, finding problems and prototyping solutions rapidly. SIP was designed and coded by astronomers for the instrument team's internal use, so we made use of existing open source tools and scripting languages wherever possible.

This paper first describes the data flow, from the *Spitzer* spacecraft to finished SIP output. The design of the individual SIP components is then described, followed by an evaluation of how well they worked together. We conclude with a description of 'lessons learned' and prospects for the future of SIP.

Further author information: (Send correspondence to P.B.) E-mail: pbarmby@cfa.harvard.edu

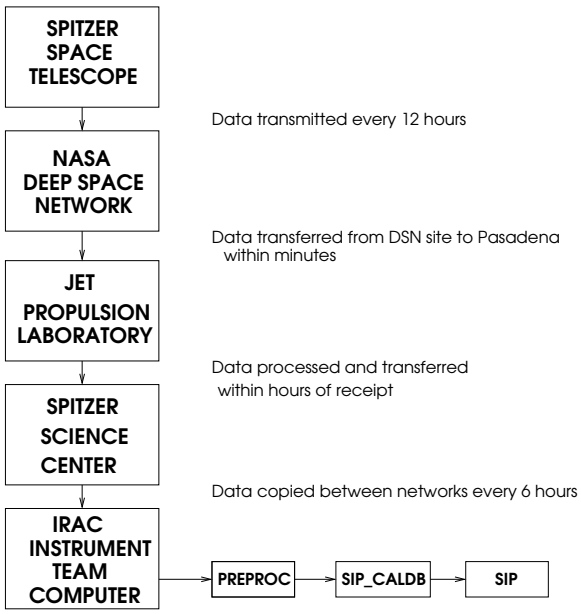

Figure 1. IRAC/SIP data flow, from spacecraft to final pipeline output.

2. DATA FLOW

As IRAC images are acquired, they are transferred from the instrument to the *Spitzer* spacecraft Command and Data Handling (C&DH) computer. There they are stored until the spacecraft communicates with the ground via the NASA Deep Space Network. After downlinking, the data are processed by the Multiple Image Processing Laboratory (MIPL) at the Jet Propulsion Laboratory. MIPL converts the raw spacecraft telemetry into FITS (Flexible Image Transport System)[3] files, which include the IRAC pixel data as the image, and instrument and spacecraft telemetry information as the file header. From MIPL the data are transferred to a secure area at the Spitzer Science Center.

Both the SSC pipeline and SIP begin processing with the 'raw' fits files produced by MIPL; a description of the SSC pipeline can be found elsewhere.[5] Security concerns at the SSC necessitated a several-step procedure for transferring raw data for SIP processing. The data were first transferred from behind the SSC firewall to a computer on the SSC 'Engineering Network', then copied to the machines which ran SIP. The automatic script which controlled the final copying step also ran the three data-processing steps of SIP. Only the first step (transfer across the firewall) required user interaction; all the others were automatic. A flow diagram of the process, from spacecraft to pipeline output, is shown in Figure 1.

3. DATA TRANSFER & PIPELINE CONTROL: ZMIRROR, PIPELINE

During the IOC/SV phase it was crucial to have rapid access to the raw and processed data coming from the spacecraft, at both the SAO and SSC locations in which the instrument team was located. To maximize the speed and reliability of the acquisition, archiving and processing of the incoming data, we implemented a fully automated system to retrieve any new data staged on the SSC Engineering Network computer, and run the SIP pipeline on it. This whole process was designed to run without human intervention, by making use of scripts written in the Bash shell, Perl and Python.

The shell script driving the whole process, ZMIRROR, was run on both our Linux servers at SSC and SAO as a cron job at regular intervals of 6 hours. The data transfer was done with the RSYNC program over a secure SSH (version 2) connection, using a public/private key authentication scheme encrypted with the DSA algorithm. At the end of the data transfer, the script generated a list of newly available raw data (or data otherwise flagged

for re-processing). The list was then passed to the last stage of the script, which was responsible for starting the execution of the data reduction pipeline.

An important part of the ZMIRROR script was the generation of a detailed set of reports and log files, mainly handled with a set of Perl scripts, accessible through a restricted access web interface (running on the open source APACHE HTTP server) for diagnostic purposes. Those pages were used during the IOC/SV phase to monitor the progress in the data transfer and alert the team of possible problems in the authentication and mirroring scripts (through a sendmail server). A page listing the status (number of files expected, transferred and processed by SIP) was produced during the processing, to alert the instrument team of any anomaly in the data transfer, or in the pipeline processing of the data. This report page was crucial for promptly identifying, among the vast amount of incoming data, the missing or corrupted files that needed further attention by SSC and MIPL.

The proper SIP pipeline was run by the last component of the ZMIRROR script, consisting of the Python script PIPELINE, functioning primarily as an 'executive' for the three processing steps. Stepping through the list of directories with new data, the script called each of the three steps, updating the web reports with their progress. PIPELINE made use of Python's error-handling capabilities to ensure that an error encountered while processing one data file or directory merely resulted in a skip to the next file or directory and not a halt of the entire processing job. As described below, the three processing steps are quite different in their scope and architecture, but controlling all three with a Python script was quite straightforward.

This setup was successfully used for the first 3 months of the mission during IOC/SV, and is still employed to transfer and to process the calibration data at SAO. An updated version of this script is currently used to download and process the scientific GTO data, and work is in progress to implement the modifications necessary to allow its inter-operability with the official Spitzer Archive.

4. PRE-PROCESSING: PREPROC, SIP_CALDB

The raw IRAC images produce by MIPL require some initial 'pre-processing' before they are usable by other programs. This pre-processing includes translating the image header parameters from channel names and data number pairs into human-readable mnemonics and engineering units (e.g., volts, Kelvins).

Unlike the header data, reprocessing of the FITS images included a number of steps: First, raw subarray data (where $N \times 64$ readouts of a 32×32 section of the IRAC arrays are made and packaged in readout order) are reordered (256^2 images thereby become stacks of 64 32^2 images). Second, the raw (integer) data from the two IRAC InSb detectors are multiplied by -1.0 to rectify them (the InSb electronic read out negative data numbers). Third, all negative pixels are then shifted by 65536 (in all channels), because the raw IRAC FITS data are packaged in a nonstandard format whereby the most significant bit is interpreted as a sign bit. This has the effect of shifting pixels from the range $[-32768, -1]$ to $[32768, +65535]$. Fourth, the so-called 'wraparound effect' whereby the preceding processing generates spuriously high-valued pixels (values greater than 57535) are re-mapped to the correct range. Fifth, the pixel data are converted to floating-point representation. Sixth and finally, the electronic barrel shift (only the 16 most significant bits in each IRAC pixel are transferred to the spacecraft and hence the ground) is accounted for and the Fowler normalization simultaneously applied.

This pre-processing was required for analysis of data taken during ground tests of the IRAC instrument prior to delivery for integration with the spacecraft, so a program, PREPROC, to do it existed well before launch. This program was written in SPP, the native Fortran-like IRAF[6] programming environment. SPP is not the simplest language in which to program or maintain code, but the PREPROC code was extensive and well-tested enough that we foresaw little need for changes after launch. Fortunately, the PyRAF[7] environment, which functions as an interface between IRAF and Python, was available. This allowed us to incorporate PREPROC into the Python environment with minimal effort.

An early design decision in SIP was the incorporation of the idea of 'metadata': each IRAC image should contain as much information as possible about how it is to be processed, with no need for reference to parameter files or detailed user instructions. One of the key aspects of this metadata approach was the selection of calibration files. Such files could be images (for example, the flatfield or dark images to be applied to the pixel data) or tables of algorithm parameters, and these could be different for different types of IRAC data. The metadata

information is stored in both the individual image headers and in a FITS-table database. The database entry for each image is created by PREPROC.

When the mission began, there was essentially only one set of calibration files, so their names could be derived from a simple set of rules and inserted in the headers and database by PREPROC. As new calibration files (e.g., 'skyflats' and 'skydarks') were generated, the actual algorithm needed to select appropriate calibration files becomes increasingly complex, as it depends on a multitude of variables of the actual data. Obviously, with the accumulating number of calibration files and the ever-changing calibration rules, it would be impractical to continuously modify the pipeline itself to accommodate this. Instead, a calibration database was implemented in which all information about the available calibration files was saved, and a sql stored procedure encoded as a Python script was utilized to process queries to this database. We used a fully functional mysql database to host the calibration data. In this design, the Python module, SIP_CALDB, is relatively independent of the pipeline processing — it only selects the appropriate calibration files for each image. The rules used to select calibration files are encoded in SIP_CALDB and can include combinations of the image characteristics such as the instrument channel and exposure time, the image timestamp, and data-taking mode. SIP_CALDB runs in sequence after PREPROC; because it essentially processes a standard relational database query and does not involve any pixel data, it runs quite efficiently.

5. SIP

SIP.PY is the Python program which does the bulk of the IRAC data processing. Because it is so central to the processing, we have used its name for the entire pipeline; when we refer to SIP in this section, only the Python modules are intended.

SIP is built entirely on object-oriented programming paradigm. It makes extensive use of the PyFITS* and numarray[8] Python modules, which hide the complexities of the FITS data structure and numerical arrays and make them easy to use. Each of these modules is of key importance for SIP: numarray provides the fast array-processing upon which the image manipulation depends, and PyFITS provides the essential input/output interface to the data.

The processing pipeline itself is implemented as a new Python class SCIDATA, with the individual modules as its methods. IRAC images, as represented by FITS files, are exposed as indexed objects of the SCIDATA class. Metadata of the FITS files are implemented as attributes of the objects, i. e., a series of Python dictionaries, with the image filename as key. The attributes are overloaded such that when an attribute matches the name of a keyword in the image headers/database, the timestamps of the header and database entries are compared, and the latter timestamp takes precedence. The World Coordinate System of each image is an instance of yet another class, SIPWCS, which allows the complexities of the WCS to be coded only once, rather than everywhere it is used. The SIPPFLAG attribute of the pipeline is a bitmask which keeps track of which pipeline modules have been run on each FITS file.

A SIP session begins with an instantiation of the SCIDATA class. The relevant image files are read into memory, and the database is opened for access, using PyFITS. Individual processing modules (flat field division, dark subtraction, linearization, etc) are run on each image in the list of filename keys. Global parameters for all images can be set using the module parameters; parameters for individual images (e.g., names of calibration files) are derived on-the-fly from the SCIDATA attributes. At the end of a SIP session, the UPDATE module is run to synchronize the image headers and database, and update the contents of the image data file. A key aspect of SIP is the fact that images are read into memory only once; this applies to calibration files as well as image data to be processed. When a processing step only involves manipulation of the metadata, the image part of the data is neither read or copied (in fact, it is essentially "ignored"). In this respect, SIP benefits greatly from the optimized memory management and automatic garbage collection that are the hallmarks of Python. Keeping all data in memory whenever possible, rather than writing to disk after each processing step (as is required with the series of separate executable modules used in many conventional pipelines), reduces the amount of disk input/output and speeds up the processing tremendously.

*PyFITS, numarray, and PyRAF are products of the Space Telescope Science Institute, which is operated by AURA for NASA.

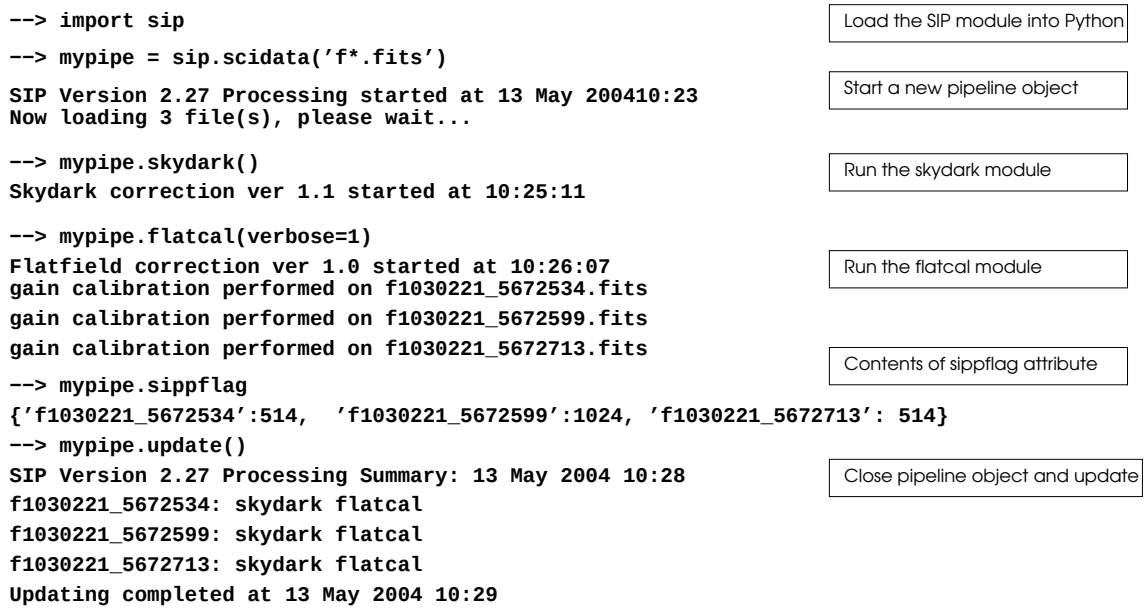

Figure 2. SIP.PY sample interactive session. Lines beginning with $-->$ indicate the PyRAF prompt. Boxed text is commentary.

SIP can be run interactively under the Python or PyRAF environments; a sample session and the accompanying data flow chart are shown in Figure 2. When used in 'automatic' mode, the PIPELINE driver script takes care of starting the processing, running the individual modules, and updating the files. The list of modules to be used is given in a file whose name is passed as a parameter to PIPELINE.

6. EXPERIENCE

We began using SIP on in-flight data starting with the very first use of IRAC on September 1, 2003. Its critical role continued until the last IRAC Science Verification data were processed on November 29. In total, 155492 images (26 GB of raw data) were processed, some of them several times as we developed new versions of the software or generated new calibration files. In general, the IRAC instrument team was pleased with the performance of SIP and the quality of its output.

SIP was run on two identical machines, Dell Precision 530 (2×2.26 GHz Pentium) PCs running RedHat Linux 8. One machine was in Pasadena at the SSC while the second was in Cambridge at the Center for Astrophysics. Data transfer between machines in Pasadena was fast, but transfer time to Cambridge was sometimes a problem: average speeds were 1.7 MB/s, but sometimes as slow as 0.2 MB/s. The time to process a given datafile depended on the type of file and the modules run. Average processing time for the standard modules on a large dataset was about 2 seconds per image. This is considerably faster than the SSC pipeline's speed on a single machine; however in practice the SSC pipeline uses many machines running in parallel.

The various processing codes underwent different amounts of revision during the IOC/SV period. ZMIRROR was not dependent on the quirks of in-flight IRAC data, and had been well-tested before launch, so it was revised only once during IOC/SV. PREPROC was stable before launch and was also revised only once. SIP.PY, which did most of the processing, was revised about 25 times. Many of these were relatively minor bug fixes, for errors such as zero division which occurred only on a tiny fraction of data. Much of the revision effort went toward improving the SIP.PY use of IMWCS.[9] This publicly-available C program uses stars in an image to set the World Coordinate System; because it is such a specialized and detailed task, we chose to call it from SIP rather than incorporate similar algorithms into our code. Getting the details of the interface between the two programs correct consumed more time than we expected. It also proved to be more important than we expected, since

much analysis depended on knowing where the telescope was pointed, and the SSC pipeline pointing module also had problems during the early mission.

Some instrumental artifacts were only discovered, or became fully characterized, after flight operations began. As a consequence we devoted considerable effort during the IOC/SV phase of the mission to developing new SIP modules to deal with a few new 'features' of the data. One such module corrected for 'column pulldown', a behavior whereby detector array columns on which the numerous photons from exceedingly bright (i.e., saturating, or nearly so) stars fell caused one or more columns to exhibit depressed count levels. The cosmetic correction algorithm was derived by L. Moustakas and D. Stern, and originally implemented in the IDL language. Porting it to Python/numarray was straightforward and took only a few hours of effort. Similarly, a Python module to remove the 'jailbar' pattern (another artifact arising from the fact that each detector column is read out by one of four amplifiers, the gains of which are not identical, leading to a so-called 'jailbar' pattern of vertical lines especially evident in IRAC channel 3) was developed and used with success. The now ubiquitous usage of these modules demonstrated the great usefulness of maintaining a flexible processing setup that could respond quickly to new and better information. These algorithms are becoming implemented as part of the SSC IRAC post-pipeline software.

We attempted to make SIP simple and well-documented enough that team members other than ourselves could use it with minimal help. Our hope was that team members who wanted to use different calibration files or run different sets of modules would be able to do so. We were only partially successful; while ZMIRROR was run manually several times (for time-critical data transfer), few of our colleagues attempted to run PIPELINE.PY for batch processing, preferring to ask us to do this. Several team members did learn how to use SIP.PY interactively, however, and managed to custom-process small sets of files without assistance. The reluctance to use the full SIP machinery may have been based on the learning curve (Python was new to most team members) during several very busy months.

7. LESSONS LEARNED; THE FUTURE

While mostly experimental, SIP has nevertheless incorporated several new and useful concepts in rapidly developing software tools for small research groups. These include distributed computing (data mirroring, modular development), legacy code reuse (incorporating many different software segments), and database integration, as well as the several key features within the object-oriented programming paradigm. From the onset, our main goals were flexibility and efficiency. With the help of Python and its many resources developed elsewhere, we have achieved these goals with far less effort than would have been needed otherwise. Most importantly, this has allowed us to focus our attention more on the actual effects of data calibration algorithms rather than the often-tedious inner workings of number crunching.

SIP proved to be a useful tool for rapid data processing during the early phases of the *Spitzer* mission, as well as providing a learning experience for its developers. If we were to repeat the process, we would probably have tried to ensure that more people understood and were able to contribute to the software, spreading the burden of responsibility.

SIP is still being used on the science data received by the IRAC instrument team, although we consider it more of a verification tool and a backup against catastrophic problems with the SSC pipeline (which so far have not occurred). We are also using it to test new algorithms to be applied in the SSC pipeline in the future.

ACKNOWLEDGMENTS

The *Spitzer Space Telescope*, is operated by the Jet Propulsion Laboratory, California Institute of Technology under NASA contract 1407. Support for this work was provided by NASA through Contract Number 1256790 issued by JPL/Caltech. Support for the IRAC instrument was provided by NASA through Contract Number 960541 issued by JPL.

REFERENCES

1. M. Werner *et al.*, "The Spitzer Space Telescope," *Astrophys. J. Suppl.* **in press**, 2004.
2. G. Fazio *et al.*, "The Infrared Array Camera (IRAC) for the Spitzer Space Telescope," *Astrophys. J. Suppl.* **in press**, 2004.
3. R. Hanisch, A. Farris, E. Greisen, W. Pence, B. Schlesinger, P. Teuben, R. Thompson, and A. Warnock, "Definition of the Flexible Image Transport System (FITS)," *A&A* **376**, 2001.
4. J. Hora *et al.*, "In-flight Performance and Calibration of the Infrared Array Camera (IRAC) for SIRTF," this volume.
5. J. Surace *et al.*, "SSC IRAC pipeline," *PASP* **in preparation**, 2004.
6. D. Tody, "IRAF in the nineties," in *Astronomical Data Analysis Software & Systems*, R. Hanisch, R. Brissinden, and J. Barnes, eds., *ASP Conf. Ser.* **52**, p. 173, 1993.
7. P. Greenfield and R. White, "A new CL for IRAF based on Python," in *Astronomical Data Analysis Software & Systems IX*, N. Manset, C. Veillet, and D. Crabtree, eds., *ASP Conf. Ser.* **216**, p. 59, 2000.
8. P. Greenfield, T. Miller, J.-C. Hsu, and R. White, "An array module for Python," in *Astronomical Data Analysis Software & Systems XI*, D. Bohlender, D. Durand, and T. Handley, eds., *ASP Conf. Ser.* **281**, p. 140, 2002.
9. D. Mink, "WCSTools: more tools for image astrometry and catalog searching," in *Astronomical Data Analysis Software & Systems XI*, D. Bohlender, D. Durand, and T. Handley, eds., *ASP Conf. Ser.* **281**, p. 169, 2002.