IP2

ISAAC Parallel Image Processing

Getting Started

v0.006

1 Introduction

1.1 What is IP2?

The shortest answer is ISAAC Parallel Image Processing (IP2), which hints at a few things, but is otherwise not terribly informative. IP2 is an astronomy image processing application designed to support a very effective, but very computationally intensive, image differencing method. The image processing in IP2 uses commodity parallel-processing methods for significant acceleration. While the original application is primarily image differencing (a.k.a. subtraction), this differencing function is a first utility in what is intended to be a more general pipeline application for high-speed processing of large astronomical images.

To address the scalability and performance required by large data volumes, the current data acquisition, processing and analyses algorithms require review, and in some cases, rewrite. Several efforts are underway to attain the needed high-performance computing by exploiting the emerging hardware availability, and development software support, of massively parallel many-core and accelerator architectures. In collaboration with one such effort spearheaded at UC Berkeley and Lawrence Berkeley National Laboratory (LBNL), titled Infrastructure for Astrophysics Applications Computing (ISAAC), IP2 was initially created to explore acceleration of a non-traditional and high-impact spatially-varying convolution algorithm as a part of astronomical image subtraction.

The initial focus of this application has been accelerating a computationally intensive method for image differencing in astronomy. The technique is known as Optimal Image Subtraction (OIS) [1] which uses a convolution technique to match the point spread function (PSF) between images. In many situations, especially with larger images, the PSF can vary across the image, requiring a spatially-varying convolution [2] in order to achieve high quality subtractions.

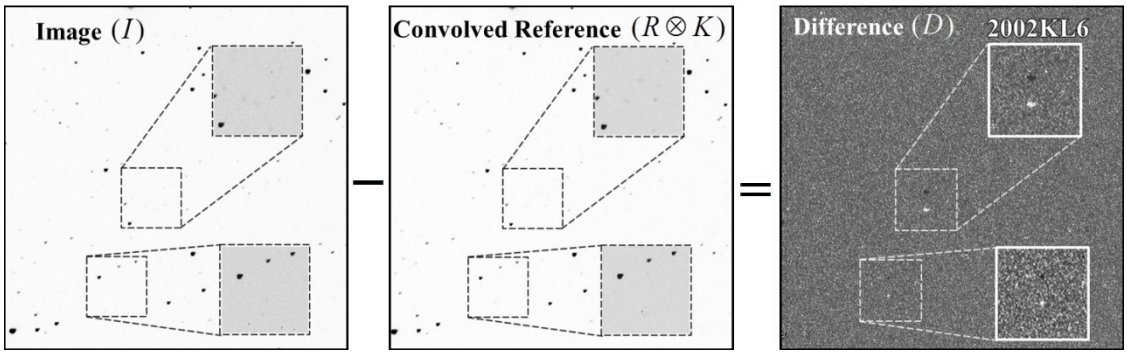

Figure 1.1 Example OIS subtraction. Two images taken at different times are matched and subtracted, yielding a difference image showing what has changed photometrically. The difference image on the right reveals two faint asteroids that have moved in the time between exposures. Image credit for the original exposures on the left: NEAT, courtesy NASA/JPL-Caltech. The difference image on the right has been generated with IP2. Image source [3].

The convolution in OIS relies on fitting a superposition of basis functions to describe the convolution kernel. Traditionally this has been done using a superposition of Gaussian function bases (GFB). However, the GFB assumes a highly symmetric PSF of a Gaussian nature. The PSF from an optical telescope should tend toward a superposition of Gaussian functions in the ideal situation. In reality, many images have asymmetric and non-Gaussian PSFs. This can be caused by any number of effects, from

atmospherics and the telescope itself to side effects from adaptive optics or other upstream image processing. Alternatively is the Dirac delta function basis (DFB) [4] which is seeing increasing use due to its adaptability to complex PSF structure, but the DFB adds dramatically to the computation (typically an order of magnitude, but up to two orders). Using OpenMP or GPUs can dramatically accelerate the DFB OIS [3], making it practical for applying to large images with high acquisition cadence or very large data sets. The acceleration can be over three orders of magnitude from the IDL implementation of the 2nd-order fit DFB OIS for large images, as shown in Figure 1.2.

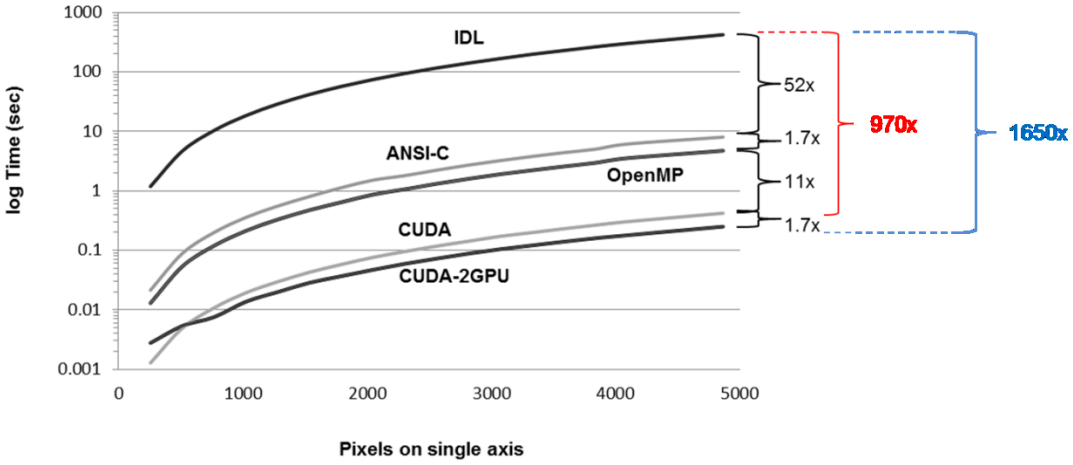

Figure 1.2 Computation times for the DFB spatially-varying convolution using the original IDL starting point compared to the IP2 accelerated CPU and GPU implementations.

For smaller images, IP2 offers little to no advantage over traditional single threaded code, but images continue to grow in size, and even the tiles of mosaic cameras are now usually larger than 2k x 2k. As CCD technology continues to advance, raw images and tiles on the scale of 10k x 10k are on the immediate horizon.

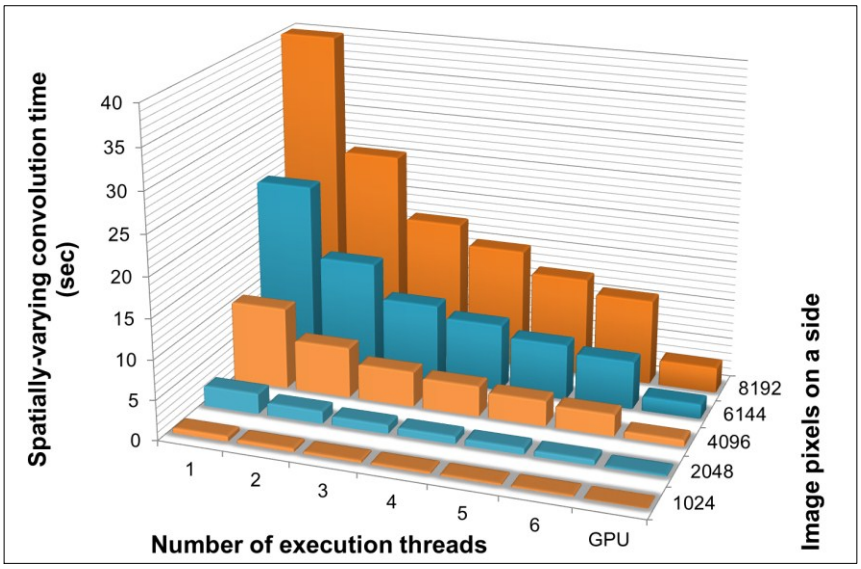

Figure 1.3 Dirac delta function spatially-varying convolution speeds using multi-core CPUs or a GPU with IP2.

IP2 also introduces techniques for ameliorating a tendency of the DFB OIS to over-fit the kernel to noise in the image, such as threshold limiting small signals in the samples used in fitting.

1.2 Installation

This is a Beta release of IP2. It operates on a single node computer. A cluster-scalable version has been demonstrated and is available for collaborative deployments. Initial deployments of the cluster code have indicated a frequent need for detailed integration, which is best served by engaging directly with the ISAAC team.

1.2.1 Hardware and Software Requirements

This release of IP2 is only supported under 64-bit Linux. The application utilizes the g++ compiler by default, though it has also been built with Intel, HP, and PGI compilers. Additionally, the GOMP library for the GNU version of OpenMP multi-core CPU support should be installed. Other OpenMP libraries will work in many cases, but have not been extensively tested. Any deviation from g++ and GOMP may well require the user to edit the IP2 makefiles.

The IP2 application may be built with or without GPU support. For the GPU supported version, CUDA 4.1 or higher must be installed. Only NVIDIA GPU cards with a compute capability 2.0 or higher will be used for GPU computation in IP2.

1.2.2 Installing IP2

Copy the downloaded tar file to the desired location and extract the contents, e.g.:

```
tar –xzf ip2sn_x.xx.xxxxb.tgz
```

Then build the code:

```
cd ip2sn_x.xx.xxxxb
./buildip2sn.sh
```

Or

```
./buildip2sn.sh nogpu
```

To force a build without CUDA and GPU support.

If your GPU libraries are in a non-standard location (/usr/local/cuda has been assumed), you may need to edit the script `buildip2sn.sh` and supply the correct paths (clearly marked near line 48). For non-GPU builds this is not an issue.

The build script will attempt to automatically detect the presence of CUDA and build the appropriate version. If successful, the build process places the executable `ip2sn` into the build directory.

For OpenMP, the environment variable `OMP_NUM_THREADS` should be set to the number of available CPU cores (or possibly the total number of CPU cores minus one on some systems) for the best performance.

1.3 Running the Basic Test Sample

The distribution includes some simple prebuilt examples that can be used to verify the operation, and serve as samples for creating your own processing operations. IP2 uses a plain text input file called a recipe to describe what is to be done. The use of a recipe avoids the need for an overly complex command line.

To verify the installation run:

```
./testrunip2sh
```

The `testrunip2.sh` script contains a very standard example of an IP2 instantiation. The general command line parameters are as follows:

```
./ip2sn –v <verbosity> -c <configfile> -f  <recipefile> -l <logfile>
```

- -v ranges from 0 to 3, a value of 1 or 2 is usually best for useful information. -v is optional.
- -c defines a configuration file for setting global values, an example is provided in the ip2etc directory provided in the distribution. A configuration file is optional but recommended.
- -f defines the recipe file. A recipe file is required. The recipe defines what actions are to be performed on what data files, and where the out is to be written.
- -l defines a log file. A log file definition is optional, but if you do not provide one, IP2 will create a unique log file name every time it runs. The unique name will consist of a concatenated date and time that application was launched. If a log file definition is provided, the any existing log file by the same name will be over written. It is recommended that you provide a log file name.

2 Recipe Files

2.1 Command Structure

Commands are defined in a recipe file and structured within a task or a series of tasks. The concept of the task is to encapsulate operations that require multiple commands, but that are operating on the same set of image files. An example of such a scenario for task encapsulation is a pair of images that must be calibrated and then co-added.

2.1.1 Recipe Files

Tasks and commands are defined in a plain text recipe file. The performance motivation of the recipe file is to eliminate the overhead and complexity of writing complex shell scripts in order to describe a large number of image processing steps. Simple helper applications or shell scripts can be used to automate the generation of the recipe files, or they can be written by hand in any text editor. The basic structure of the recipe is that of a keyword-parameter pair on a single line:

```
[keyword] <parameter string> [end of line].
```

Note that keywords and parameters are case sensitive.

The purpose of the recipe file is to provide a human readable and writable interface to IP2. Once the application is launched, the recipe file is immediately parsed and converted into linked-lists of data structures for processing in the compute nodes. The parsing is accomplished by high-speed matching of keywords in the recipe file to a lookup table. Once matched, the remainder of the line is processed as a parameter, where the parameter type is also specified in a lookup table. Any necessary conversions from a text string variable to integers or floating point numbers take place at this stage. Keywords and parameters are converted into data structures that are linked in the order of discovery, which, for command sequences, becomes the order of execution.

The syntax is that of nested `Task` and `Cmd` (command) operations. All command parameters must appear between the `Cmd` keyword and an associated `CmdEnd` keyword. All commands must be fully contained within a task between the `Task` and `TaskEnd` keywords. The basic structure of the task/command relationship is illustrated in Figure 2.1 and demonstrated in the example recipe shown in Figure 2.6.

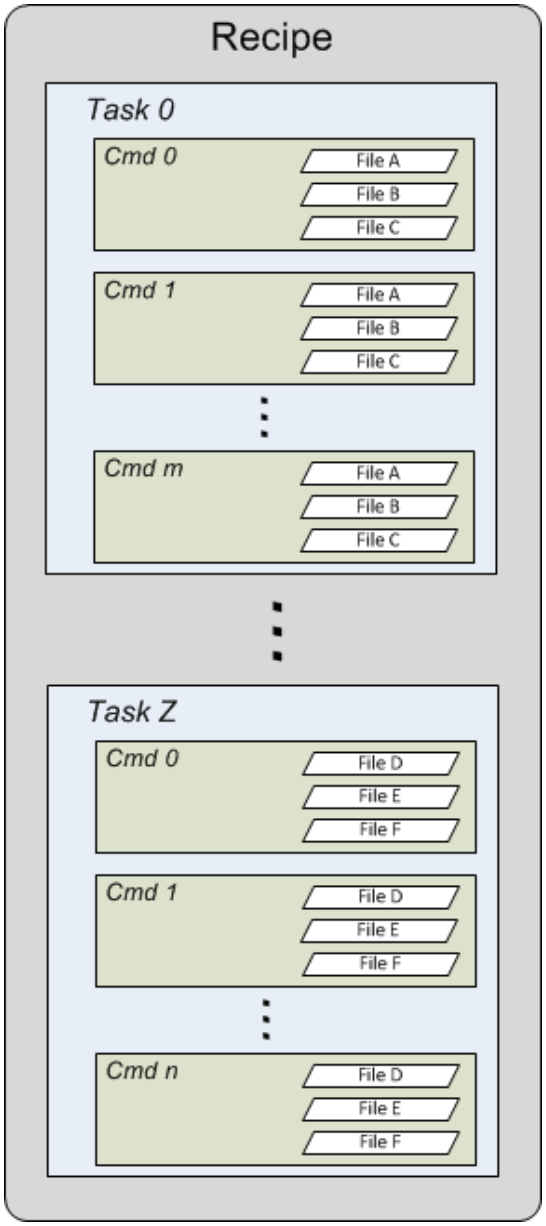

Figure 2.1 Diagram of task and command hierarchical structure

In the example recipe that follows in Figure 2.6, there are two independent tasks, each operating on separate image files. The tasks can be processed on the same node, each in turn, or they can be sent to separate nodes for simultaneous processing. The first task in the example performs a calibration step on a set of images, followed by a co-addition of the calibrated results. The second task performs an OIS subtraction of two other images.

2.2 Image Manipulation Commands

Images inputs for OIS must be in the standard FITS file format [5], used widely throughout astronomy. FITS file images must not be in compressed format.

The primary motivation for the creation of IP2 is to provide an accelerated platform for OIS. But a single operation tool, even one as powerful as OIS, is of limited appeal. A full-featured pipeline has many image

manipulation operations to perform. The most basic operations are those used in image calibration. In this context calibration is the process taken to clean up the image data from the raw form that comes directly off of the detector, a CCD in most cases, by removing instrument induced artifacts. Several basic image math operations have been implemented in IP2 to support standard calibration functions. The standard calibration operations in Table 2-1 are all implemented in OpenMP parallelism. Because most of the commands only perform one or two mathematical operations per pixel on the image data, the computation time is not sufficient to cover the data movement overhead required for GPU implementations, thus GPU implementations are actually slower than OpenMP for these operations.

Table 2-1 IP2 Basic Commands for Image Calibration

Command	Description
Add	Basic pixel by pixel addition of two or more images
Median Add	Adding two or more images while maintaining scale, this is useful for creating co-added or "stacked" images that are scaled to the level of the initial input images
Subtract	Basic pixel by pixel subtraction (not OIS), useful for subtracting dark and bias images to remove amplifier and thermal noise
Scale	Multiply all pixels in an image by a given floating point value
Multiply	Multiply two images pixel by pixel
FlatField	A scaled divide of an image by a flat field image, see below for details

Table 2-2 lists the PSF matching commands currently implemented in IP2. The type of PSF matching is defined through the basis functions and the order of the bivariate fitting polynomial. `Set` functions in the configuration file or the recipe file will define the nature of the PSF matching.

Images must be co-aligned in an external tool prior to OIS subtraction or OIA addition in this release of IP2. Images up to 8k x 8k pixels have been tested successfully on the GPU version, and larger images can be processed by GPU if there is sufficient memory available. Much larger images can also be processes by disabling the GPU in the build process or in the configuration file and relying on OpenMP CPU processing where there is generally more memory available.

Table 2-2 IP2 OIS Commands

Command	Description
OIS	OIS between two images. The inputs are the image and the reference image to be subtracted.
OIA	OIA is optimal image addition. This is an experimental technique for co-adding images using the same PSF matching methods of OIS. The concept is that by matching the PSF, the photometric linearity of the combined image is preserved.

2.2.1 Basic Calibration Commands

Raw data as read directly from a CCD has several sources of detector induced and optically induced artifacts. Systemic thermal photons and offsets induced by digital to analog converters and amplifiers from the instrument and the environment can be reduced through the use of dark frames and bias frames [6]. Dark frames are images made from exposures with a cover over the telescope or the shutter on the camera closed, an example is shown in Figure 2.2. The dark frame is taken for the same exposure duration and at the same detector temperature as the science images. As a result, the dark frame is a "picture" of the thermal noise photon characteristics of the detector. A second type of dark frame is the bias frame, an example of which appears in Figure 2.3. The bias frame has zero exposure time, and thus it characterizes an image of the offsets induced by the electronics in the camera readout stage. Once captured, the bias and the dark can both be subtracted from a raw image through a simple pixel-by-pixel subtraction. It is standard practice to co-add several samples of dark and bias frames into a master dark and master bias in order to reduce random noise effects. The very simple operation for dark frame subtraction is shown in equation(0.1).

$$Corrected_{x,y} = Original_{x,y} - \left(\frac{\sum_{N} Dark}{N} \right) - \left(\frac{\sum_{M} Bias}{M} \right) \qquad (0.1)$$

While the dark frames primarily correct for detector issues, the flat frames correct for optical defects in the telescope system [6]. The flat field image can capture issues with even small amounts of dust and debris on any of the optical surfaces, including mirrors, filters, or lenses. The flat field image is also able to characterize uneven illumination effects across the FOV such as those induced by vinetting. The flat field image is acquired by taking an exposure of an even "flat-light" illuminated field, hence the name. It is also standard practice to co-add several samples of the flat images into a master flat in order to reduce random noise effects. An example flat field image is shown in Figure 2.4.

To remove the flat field artifacts from the science image, it is necessary to divide the image by a normalized version of the flat field image. The normalization factor is acquired by finding an average pixel value from a sample region near the center of the flat field, were darkening effects due to vinetting are minimal. The science image is then divided, pixel by pixel, by the flat field.

An example of basic image processing is illustrated in Figure 2.5. Actual processing results will vary greatly depending on the calibration images. At this time, IP2 does not represent any innovation in basic image calibration commands, they are merely provided so that the pipeline can keep data internal to the system while preparing images for subtraction or future IP2 features. Future work may examine refinements to these basic image manipulation and calibration operations.

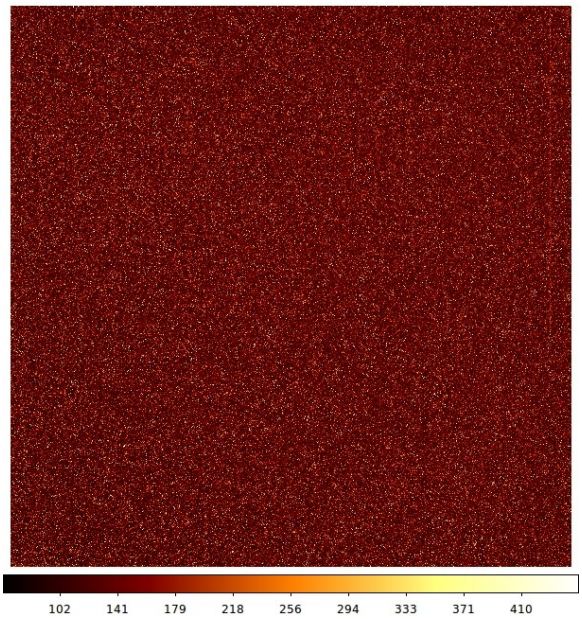

Figure 2.2 Example stacked master dark frame. A stacked image of 8 individual dark combined using IP2. Image credit: W. Green, F. Mezzalira and S. Hartung/SBO.

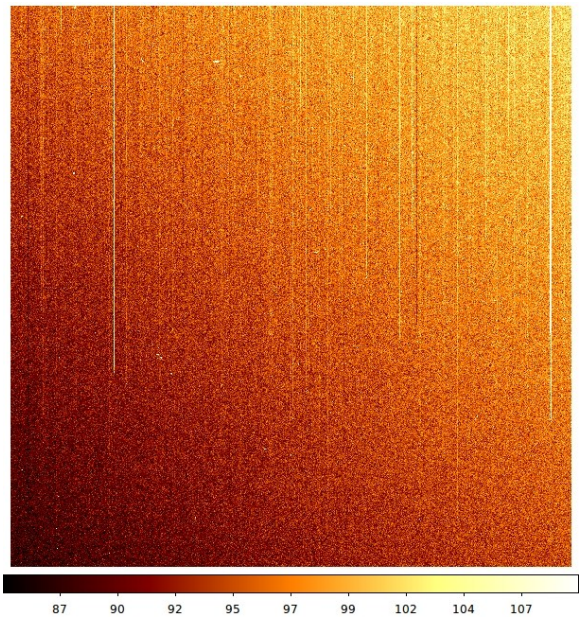

Figure 2.3 Example stacked master bias frame. A stacked image of 10 individual bias frames combined using IP2. Image credit: W. Green, F. Mezzalira and S. Hartung/SBO.

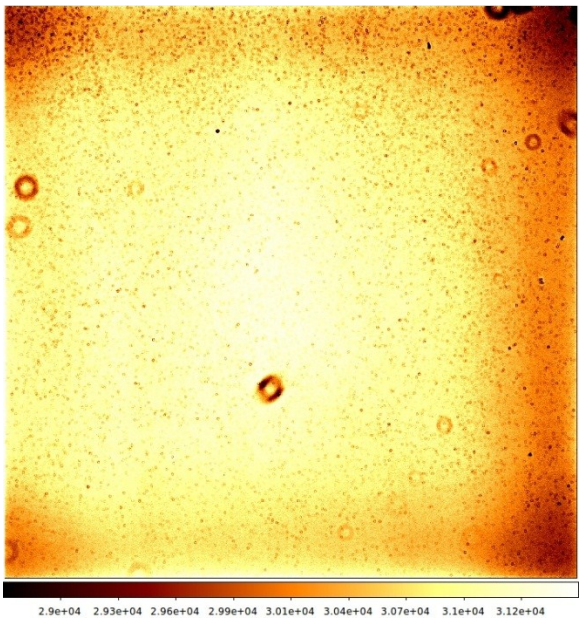

Figure 2.4 Example stacked master flat frame. Artifacts are due to dust and debris on the optics, which are out of focus and generate circular Airy disk patterns, and vignetting seen as a darkening in the corners. Image credit: W. Green, F. Mezzalira and S. Hartung/SBO.

$$MasterFlat_{x,y} = \frac{\sum_{N} Flat_{x,y}}{N} \tag{0.2}$$

$$Corrected_{x,y} = \frac{Original_{x,y}}{MasterFlat_{x,y}} MasterFlat_{SampleAverage}$$

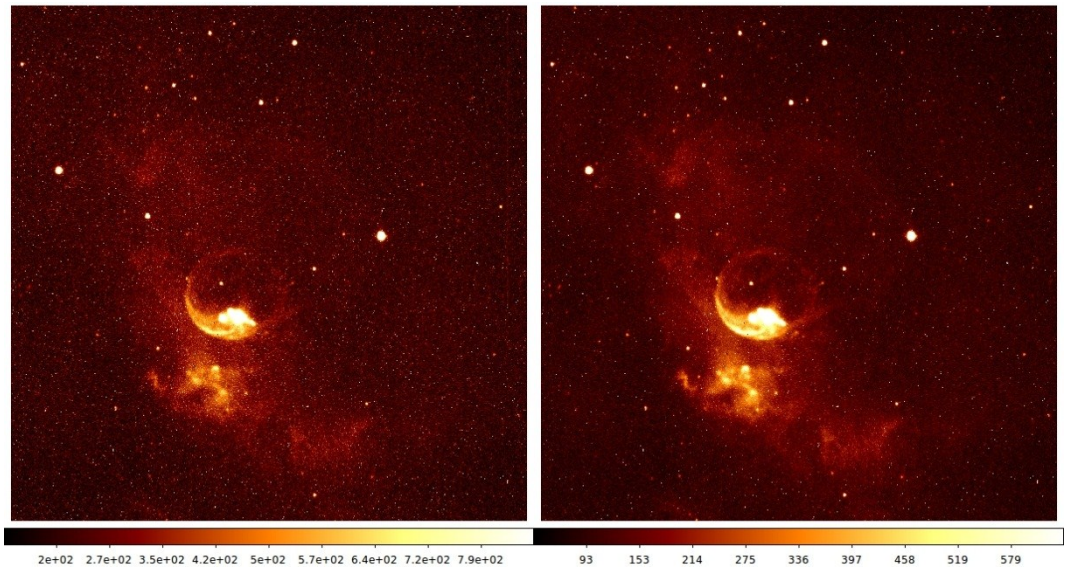

Figure 2.5 Example raw image on the left as read from the CCD, image dark subtracted and flat fielded in IP2 on the right. While some random noise remains, a great deal of the systemic artifacts has been removed. False color image of the Bubble nebula in hydrogen-alpha. Image credit: W. Green, F. Mezzalira and S. Hartung/SBO.

2.3 Diagnostic and Utility Commands

It is convenient to use the recipe command interface to provide other functions in addition to the built-in image manipulation routines. The extended diagnostic and utility commands are listed in Table 2-3.

The shell command, in particular, allows for a great deal of flexibility by allowing an arbitrary string to be passed to the operating system command line. Simple uses of the shell allowing recipe files to make directories or move files as needed. The shell command also allows the use of externally compiled tools (e.g. the Astromatic application suite) for accessing functionality that is not yet available inside of IP2.

Table 2-3 IP2 Diagnostic and Utility Commands

Command	Description
ImgStats	Performs a basic analysis on a specified image to determine max, min, and standard deviation characteristics of the image as a whole and of the sky background
Shell	Allows passing a command string to the operating system command shell for execution from a recipe file

Figure 2.6 shows an example IP2 recipe file.

```
# Input file for the IP2 pipeline defining the processing recipe
# This is a comment
JobName [some job name]
User [some user name for logging]
# the common base directory for all tasks
JobBaseDir /home/worker/

Task NameForTask1
# dark subtract images
   # basic subtraction works for darks and bias types
   Cmd Subtract
          # relative sub-directories and file names
          InputPath swtest/sbo/bubble/
          OutputPath swtest/sbo/bubble/
          OutputFitsFile CAL_00000200.Bubble_Nebula.FIT
          InputFitsFile SCI_00000200.Bubble_Nebula.FIT
          InputFitsFile DARK_-20_Ha.FIT
   CmdEnd
   Cmd Subtract
          InputPath swtest/sbo/bubble/
          OutputPath swtest/sbo/bubble/coadd/
          OutputFitsFile CAL_00000207.Bubble_Nebula.FIT
          InputFitsFile SCI_00000207.Bubble_Nebula.FIT
          InputFitsFile DARK_-20_Ha.FIT
   CmdEnd
# median combine the calibrated images
   Cmd MedianAdd
          InputFileCount 2
          InputPath swtest/sbo/bubble/
          OutputPath swtest/
          OutputFitsFile medadd_img.fits
          InputFitsFile CAL_00000200.Bubble_Nebula.FIT
          InputFitsFile CAL_00000207.Bubble_Nebula.FIT
   CmdEnd
TaskEnd

# define another task that operates on a different set of files
Task NameForTask2
      # perform Optimal Image Subtraction (OIS)
      Cmd OIS
          InputPath tmt/
          OutputPath tmt/diff/
          OutputFitsFile A-B_diff.fits
          InputFitsFile A_img.fits
          InputFitsFile B_img.fits
      CmdEnd
TaskEnd
```

Figure 2.6 Example IP2 recipe file with two independent tasks

2.4 Configuration File and Set Parameters

Configuration **Set** parameters are global values that affect the operations of IP2. A **Set** parameter can be defined in either the main configuration file or in a recipe file. The last instance of the **Set** parameter that is encountered is the one that will be used, thus it is possible to create a default set in a configuration file, but override them whenever needed in a specific recipe file without needing to modify the configuration file. An example configuration file with all of the currently available **Set** commands is provided in the file "ip2etc/ip2.conf" in the source code release package.

The format for a Set parameter is:

```
Set <ParameterKeyWord> <value>
```

Table 2-4 IP2 Set Parameters

Parameter	Description	Type
NoGPU	0 = default, GPU will be used if found, 1 = suppress GPU usage (CPU only) even if a suitable GPU is found	integer
ImageSat	Pixel value for saturation (should actually be set for the level above which pixel response may be non-linear). Default value is 50000.0	float
SrcFindWindowSz	Size in pixels of the sliding window used to identify sources (usually stars) in images for OIS samples. Numbers between 10 and 20 are often appropriate.	integer
SrcFindSigmaThresh	Threshold above background for detecting OIS sample sources in multipliers of 1σ of the background standard deviation in general sky noise.	float
OISEnableAutoScan	Default is 0. If set to 1, OIS will attempt DFB and GFB kernels at each of the polynomial orders of 0, 1, and 2, and it will keep the solution with the lowest σ of residuals in the resulting subtraction	integer (0 or 1)
OISKernelSize	Size in pixels of the convolution kernel to use. The maximum value is 21 pixels. Generally values between 7 and 15 are appropriate. If set to an even number, it will be rounded up to the next odd number. If set to 0 (the default) OIS will attempt to devise its own kernel width based on the statistics of samples taken from the image.	integer
OISPolynomialDeg	The spatially-varying convolution bivariate polynomial order 0, 1 or 2	integer (0, 1, or 2)
SaveConvRef	Will cause the convolved reference image from the OIS process to be written out to a fits file with _cref appended to its name if set to 1.	integer (0 or 1)
SaveStamps SaveKernels SaveConvStamps	These diagnostic parameters will save sample stamp fits file images of the unmodified stamps, the kernel derived at each stamp, and the convolved reference at each stamp. WARNING: this can generate a very large number of small image files. These are all set to 0 by default	integer (0 or 1)

BrightSuppressionEnable	Enable bright object residual suppression in OIS by setting to 1. Disable if 0. This is not a traditional masking system, but feedback mechanism that nulls out excursions above a set threshold in the vicinity of bright objects.	integer (0 or 1)
BrightSuppressionDetect	Detection threshold for what is considered a bright object in multipliers of 1σ above background in the original OIS input image. Values between 20.0 and 100.0 are often useful, but this can be very camera and telescope dependent.	float
BrightSuppressionThresh	Threshold for what residuals in the OIS subtracted image output will be suppressed, in multipliers of 1σ above background.	float
UseSrcExtApp	OIS requires a system of finding sources, preferably isolated stars of moderate brightness, in order to select sample regions for fitting the convolution kernel solution. Default behavior uses an internal source detector, however, the Astromatic S[ource]Extractor program (http://www.astromatic.net/software/sextractor) may also be used if it is available by setting the parameter to 1 (default is 0).	integer (0 or 1)
SrcExtAppName	If the S[ource]Extractor application name has been changed from the default name, then this parameter may be used to inform IP2 of the application's current name on the system.	string
The parameters that follow define the Gaussian forms used in the GFB OIS		
UseGauss	0 = default, OIS will use a 2nd-order Miller Dirac delta function basis (DFB) for the spatially-varying convolution kernel. 1 = use a more traditional Gaussian function basis (GFB) for the spatially-varying convolution kernel.	integer (0 or 1)
UseAstierGFB	0 = default, use internally or user defined Gaussian parameters. 1 = use the Gaussian basis parameters defined by Astier as described by Miller [4] (see eqs. 12 and 14 in Miller) . The Astier defaults are widths 0.7, 1.5, and 2.0, and degrees 6, 4, and 2 respectively.	integer (0 or 1)
GaussWidth1 GaussWidth2 GaussWidth3	There are three Gaussian basis functions supported. Each of these parameters sets the width of one of them. This is the standard deviation of the Gaussian as a multiple of the kernel width in pixels. The default values are 0.7, 1.5, and 2.0 respectively.	float
GaussDeg1 GaussDeg2 GaussDeg3	There are three Gaussian basis functions supported. Each of these parameters sets the degree of the bivariate polynomial that expands them. 0 is the non-Astier default consisting of only the Gaussian with no polynomial expansion terms. It is not recommended to set this above 6 for any one value as that creates a very set of basis functions and will be very slow with little potential value in the resulting subtraction. In practice values above 0 are often better solved by the use of the DFB.	integer (0 - 6)

2.5 Command Parameters

This section describes each of the IP2 image processing commands by examples of the recipe file description.

2.5.1 Cmd OIS

Optimal Image Subtraction, by either GFB or DFB means of convolution PSF matching, depending on Set parameter values.

```
Cmd OIS
    InputPath images/
    OutputPath results/
    OutputFitsFile subtraction.fits
    InputFitsFile image.fits
    InputFitsFile reference.fits
CmdEnd
```

Alternatively, if "`Set UseSrcExtApp 1`" is enabled, then the external application S[ource]Extractor is used to find sample sites in the image. If S[ource]Extractor is to be called in real-time from by IP2, then the command is no different from the above. However, if the user has a particular source list file generated by S[ource]Extractor that they want to use, then they may specify it explicitly.

```
Cmd OIS
    InputPath images/
    OutputPath results/
    OutputFitsFile subtraction.fits
    InputTextFile listfile.stars
    InputFitsFile image.fits
    InputFitsFile reference.fits
CmdEnd
```

If this later method is used, the star list file must conform to a minimum of the S[ource]Extractor parameters as defined in the parameter file provided in the file `ip2etc/ip2srcext.param`. Running S[ource]Extractor with the provided `param` file will ensure proper operation. Additional parameters may also be present in the star list file, but none of the parameters enabled by `ip2etc/ip2srcext.param` should be excluded.

2.5.2 Cmd OIA

Optimal Image Addition, by either GFB or DFB means, depending on Set parameter values. OIA performs the same PSF matching methods to mean addition as OIS does to subtraction. This is an experimental technique for allowing the preservation of linear photometric response in the PSF during co-addition.

```
Cmd OIA
    InputPath images/
    OutputPath results/
    OutputFitsFile co-added.fits
    InputFitsFile image.fits
    InputFitsFile reference.fits
CmdEnd
```

2.5.3 Cmd Subtraction

Simple pixel by pixel subtraction, useful in removal of dark frame or bias frame features.

```
Cmd Subtract
    InputPath images/
    OutputPath results/
    OutputFitsFile correctedimage.fits
    InputFitsFile image.fits
    InputFitsFile dark.fits
CmdEnd
```

2.5.4 Cmd Add

Simple pixel by pixel addition of two or more images.

```
Cmd Add
    InputPath images/
    OutputPath results/
    InputFileCount 3
    OutputFitsFile image123.fits
    InputFitsFile image1.fits
    InputFitsFile image2.fits
    InputFitsFile image3.fits
CmdEnd
```

2.5.5 Cmd MedianAdd

Simple pixel by pixel addition and averaging.

```
Cmd Add
    InputPath images/
    OutputPath results/
    InputFileCount 3
    OutputFitsFile image123div3.fits
    InputFitsFile image1.fits
    InputFitsFile image2.fits
    InputFitsFile image3.fits
CmdEnd
```

2.5.6 Cmd Scale

Simple pixel by multiplication of a fixed value.

```
Cmd Scale
    InputPath images/
    OutputPath results/
    Float 3.0
    OutputFitsFile image12.fits
    InputFitsFile image1.fits
    InputFitsFile image2.fits
CmdEnd
```

2.5.7 Cmd Multiply

Simple pixel by multiplication of a two images.

```
Cmd Multiply
    InputPath images/
    OutputPath results/
    Float 3.0
    OutputFitsFile image1mult3.fits
    InputFitsFile image1.fits
CmdEnd
```

2.5.8 Cmd FlatField

Scaled division by flat field image.

```
Cmd FlatField
     InputPath images/
     FlatPath flats/
     OutputPath results/
     OutputFitsFile correctedimage.fits
     InputFitsFile image.fits
     InputFitsFlat flat.fits
CmdEnd
```

2.5.9 Cmd ImgStats

Calculate some basic image statistics and output them to the console.

```
Cmd ImgStats
     InputPath images/
     InputFitsFile image.fits
CmdEnd
```

2.5.10 Cmd Shell

Send the specified string to the unix shell and wait for the result. Useful for running external operations as part of a recipe. Can be used to create directories, copy or move files, remove intermediate files that are no longer needed, or run an application that is complimentary to IP2 as part of the processing, just as a few examples.

```
Cmd Shell
     String <any valid unix shell command line string>
CmdEnd
```

[1] C. Alard and R. H. Lupton, "A Method for Optimal Image Subtraction," *The Astrophysical Journal,* p. 325, 1998.

[2] C. Alard, "Image subtraction using a space-varying kernel," *Astronomy & Astrophysics Supplement Series,* vol. 144, pp. 363-370, 2000.

[3] S. Hartung*, et al.*, "GPU Acceleration of Image Convolution using Spatially-varying Kernel," presented at the IEEE International Conference on Image Processing (ICIP) 2012, Orlando, Florida, 2012.

[4] J. P. Miller*, et al.*, "Optimal Image Subtraction Method: Summary Derivations, Applications, and Publicly Shared Application Using IDL," *Publications of the Astronomical Society of the Pacific,* vol. 120, pp. 449-464, 2008.

[5] R. J. Hanisch*, et al.*, "Definition of the Flexible Image Transport System (FITS)," *Astronomy & Astrophysics,* vol. 376, pp. 359-380, 2001.

[6] S. B. Howell, *Handbook of CCD Astronomy*, 2nd ed. vol. 5. New York, NY: Cambridge University Press, 2006.