Squeezing a Big Orange into Little Boxes: The AscotDB System for Parallel Processing of Data on a Sphere

Jacob Vanderplas[1,2] Emad Soroush[1] Simon Krughoff[2] Magdalena Balazinska[1]
Andrew Connolly[2]

[1] Department of Computer Science and Engineering, University of Washington, Seattle WA
[2] Department of Astronomy, University of Washington, Seattle WA

Abstract

AscotDB is a new, extensible data analysis system developed at the University of Washington for the interactive analysis of data from astronomical surveys. AscotDB is a layered system: It builds on SciDB to provide a shared-nothing, parallel array processing and data management engine. AscotDB wraps SciDB with a Python middleware that enables efficient storage and manipulation of spherical data, such as images from telescopes or satellites. The goal is to support the efficient storage of raw pixel-level data without any prior preprocessing steps. To enable both exploratory and deep analysis of the data, AscotDB's front-end design integrates a python interface with a graphical interface based on the Astronomy Collaborative Toolkit (ASCOT). AscotDB supports seamless switching between these two modes of interaction and captures a precise trace of a user's operations on the data to ensure repeatability. In this paper, we present an overview of the AscotDB system. While based on astronomy as its key application-domain, AscotDB primitives are general enough to be applicable to other scientific fields concerned with data on a sphere.

1 Introduction

Astronomy, like other scientific fields, is currently moving to a new realm of research driven by large datasets. One example of this new approach is the upcoming Large Synoptic Survey Telescope (LSST) [4], which will begin operations toward the end of this decade. The LSST will repeatedly image the entire southern sky over ten years, resulting in unprecedented volumes of uniformly-calibrated astronomical data – up to a hundred Petabytes by the end of the survey. The scientific results from these data will depend on our ability to perform fundamental operations at scale, including the detection of faint objects through the stacking of multiple exposures and the detection of variable objects through the differencing of exposure pairs, all while taking into account the temporally and spatially varying *point-spread function* – that is, the mathematical model of how light from any given object is detected by the camera after refraction by the atmosphere and telescope optics.

The AscotDB project is a collaboration of an inter-disciplinary team comprising astronomy and database experts with the goal of answering one question: What would be the most transformative tool for processing these next-generation telescope image collections? The AscotDB system has emerged in answer to this question.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

In order to image the visible sky, LSST will undertake repeated exposures over ten years with each image partially overlapping with hundreds of others. To enable efficient stacking and comparison of these images, it is vital to store the data in a way that allows efficient indexing of pixel positions both on the sky and in time. While pipelines are being designed by the LSST team to handle this image processing task and create *catalogs* of detected objects, the truly transformative science will come from providing scientists with the ability to directly query the *raw* data, and to enable *interactive and exploratory* computation and visualization of that data.

AscotDB integrates several pieces of technology: the SciDB engine for data storage and processing, Ascot for graphical data exploration, and Python for easy programmatic access. None of these technologies solves the problem on its own. More importantly, their naïve combination is also insufficient.

SciDB. SciDB [14] is an open-source database system with *inherent* support for multi-dimensional arrays. The inherent support means the entire SciDB stack is designed based on the array data model. In prior work [5, 10, 15], we showed that an array-based database system such as SciDB is well-suited to store and manipulate small patches of sky images pre-projected (or *warped*) onto a regular Cartesian grid. Although this SciDB-based solution is compelling, the solution remains a step removed from the raw data: it requires significant overhead to individually warp images to each local tangential projection before loading them into the database for an analysis, and the local nature of the tangential projection means the entire sky cannot be analyzed at once. Because SciDB is built for efficient computing over Cartesian arrays, it is not straightforward to use it for the inherently spherical data gathered by astronomical surveys.

Ascot. While efficient data storage and manipulation is an important piece of the astronomer's task, another vital component is the ability during an analysis to visually interact with and explore data in order to determine the next analysis step. The AStronomy COllaborative Toolkit (ASCOT) [9][1] developed in the Astronomy community, is a collection of Web-based gadgets that facilitate collaboration between astronomers. These gadgets are assembled into a dashboard and communicate using a node.js server. Through the use of a customizable dashboard interface, users can easily visualize, manipulate, and share large data sets from many different sources. Ascot, however, only enables the display of individual images and the analysis of pre-processed *catalog* data. It does not support the large-scale analysis of pixel data.

Python. For both data analysis and visualization, Python is fast becoming a defacto standard in the astronomy research community. The scientific Python ecosystem, built around the core tools of NumPy & SciPy [12], Matplotlib [7], and IPython [13] (including the web-based interface of the *IPython notebook*), provides a complete environment for the analysis and visualization of data at a small to medium scale. Due to its advantages, a wide selection of current and future astronomical surveys are now building their data analysis pipelines using Python. Of particular relevance for this work, the LSST project plans to use Python as their primary data pipeline interface. In this environment, it is increasingly important for data analysis tools to provide Python-hooks for any computing infrastructure. Python alone, however, provides limited data management capabilities and is non-trivial to use on LSST-sized datasets.

AscotDB builds on the combination of these three pieces of technology to provide a compelling and powerful environment for the exploration, analysis, visualization, and sharing of large astronomical datasets. AscotDB further contributes several techniques: It extends SciDB with native support for efficient iterative computations (Section 2.1). It provides more intuitive Python language bindings through a new SciDB-py package (Section 2.2). It further integrates the graphical and programmatic interfaces to support seamless switching between the two modes of interaction (Section 2.3). Finally, it adds a middleware-layer that enables the manipulation of

[1] `http://ascot.github.io/`

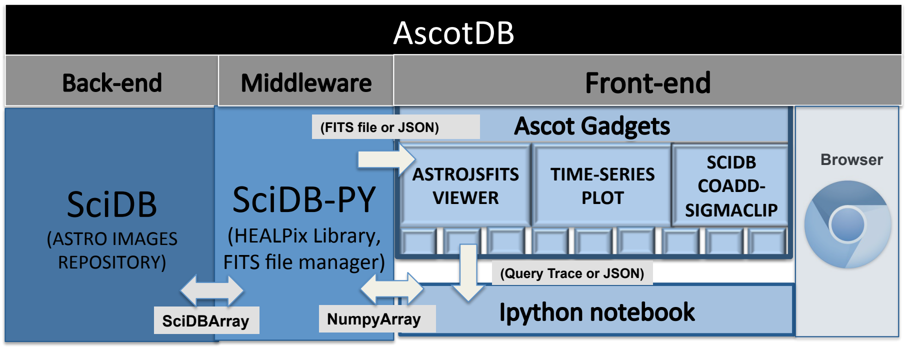


Figure 1: AscotDB architecture: SciDB as back-end, python middleware, Ascot and IPython as front-ends.

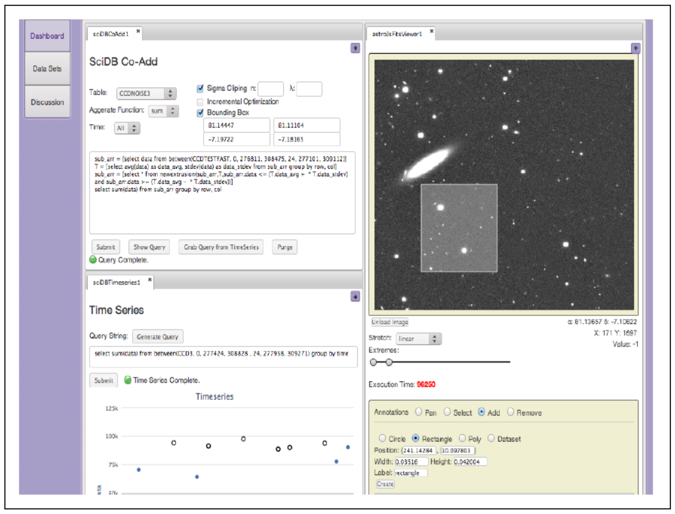
spherical data in SciDB (Section 3). Because it focuses on producing a tool that is transformative and accepted by scientists, AscotDB illustrates how necessary technologies must be integrated and extended.

2 AscotDB

The basic operations astronomers perform on images fall into two categories: detection and measurement. Detection is the process of identifying the location of individual sources in an image, where the sources might be stationary objects such as individual stars and galaxies, or moving/transient objects such as asteroids or supernovae. Measurement involves computing well-calibrated statistics from the light of the individual objects: for example, computing the total optical flux from a star through a detailed model of its response across the CCD pixels. AscotDB provides the user with two modes of interaction with the data: (1) Visual interaction with Ascot gadgets that we describe in Section 2.1 – these visual interactions are an important component of the detection process; and (2) an IPython interface for programmatic interaction for both detection and measurement that we describe in Section 2.2. Importantly, AscotDB integrates the two modes of interaction for a seamless experience (Section 2.3). The overall architecture of AscotDB and the high level interaction between components is diagrammed in Fig. 1.

2.1 Graphical Front-End Support

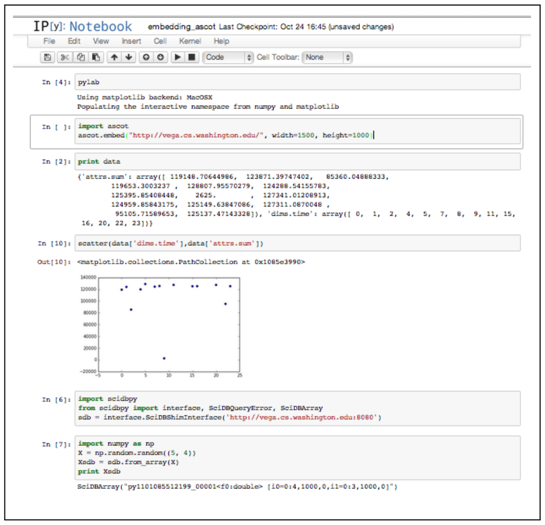
Figure 2(a) shows a screenshot of AscotDB's graphical interface. AscotDB retains important Ascot features including its extensibility through the addition of new gadgets and its sharing capabilities across users. AscotDB, however, radically transforms Ascot's data analysis capabilities. Originally, Ascot enabled users to view a *single* telescope image at a time and overlay on the image catalog data extracted from a back-end relational DBMS. In contrast, AscotDB enables users to manipulate raw pixel data. For example, users can *stack* images of the sky that fall within a region, *clean them* using an iterative process, re-run a source detection algorithm, annotate interesting sources, and generate visual summaries (e.g., light curves) before initiating a measurement process. To support these novel operations, AscotDB stores the telescope image data inside SciDB and translates operations on the interface into queries over SciDB's arrays. Because scientists are technically savvy users and because graphical interfaces can inaccurately capture a user's intent, AscotDB always shows the queries that it generates and enables a user to modify them before executing them. While the general integration of Ascot with SciDB is mostly an engineering project, ensuring high-performance required important extensions to the SciDB engine. In particular, several operations including source detection and data cleaning are iterative in nature. To support these iterative tasks at interactive speeds, we extended SciDB with native support for array iterations, described in a recent system demonstration [10]. The support includes the execution of such iterations in shared-nothing clusters and various optimizations such as incremental processing. Figure 3 illustrates the performance of SciDB



```
sub_arr = [select data from between(CCDTESTFAST, 0, 276811, 308475, 24, 277101, 309112)]
T = [select avg(data) as data_avg, stdev(data) as data_stdev from sub_arr group by row, col]
sub_arr = [select * from newextrusion(sub_arr,T,sub_arr.data <= (T.data_avg +  * T.data_stdev)
and sub_arr.data >= (T.data_avg -  * T.data_stdev))]
select sum(data) from sub_arr group by row, col
```

```
select sum(data) from between(CCD3, 0, 277424, 308828 , 24, 277958, 309271) group by time
```

(a) Graphical front-end of AscotDB.


IP[y]: Notebook
embedding_ascot Last Checkpoint: Oct 24 16:45 (unsaved changes)
File Edit View Insert Cell Kernel Help
Code
Cell Toolbar: None

```
In [4]: pylab
        Using matplotlib backend: MacOSX
        Populating the interactive namespace from numpy and matplotlib

In [ ]: import ascot
        ascot.embed("http://vega.cs.washington.edu/", width=1500, height=1000)

In [2]: print data
        {'attrs.sum': array([ 119148.70644986,  123871.39747402,   85360.04888333,
                119653.3003237 ,  128807.95570279,  124288.54155783,
                125395.85408448,    2625.        ,  127341.01208913,
                124959.85843175,  125149.63847086,  127311.0870048 ,
                 95105.71589653,  125137.47143328]), 'dims.time': array([ 0,  1,  2,  4,  5,  7,  8,  9, 11, 15,
         16, 20, 22, 23])}

In [10]: scatter(data['dims.time'],data['attrs.sum'])

Out[10]: <matplotlib.collections.PathCollection at 0x1085e3990>

In [6]: import scidbpy
        from scidbpy import interface, SciDBQueryError, SciDBArray
        sdb = interface.SciDBShimInterface('http://vega.cs.washington.edu:8080')

In [7]: import numpy as np
        X = np.random.random((5, 4))
        Xsdb = sdb.from_array(X)
        print Xsdb
        SciDBArray("py1101085512199_00001<f0:double> [i0=0:4,1000,0,i1=0:3,1000,0]")
```

(b) Programmatical front-end of AscotDB.

Figure 2: Two modes of interaction with AscotDB: visual interface and IPython interface. The visual interface is embedded in IPython, but here is shown separately for purposes of illustration.

with *incremental iterative processing* extensions on an application called "Sigma clipping"[2]. Incremental iterative processing is based on the idea that the output at each iteration differs only partially from the output at the previous iteration. Performance can thus significantly improve if the system computes, at each iteration, only the part of the output that changes rather than re-computing the entire result.

A second important design decision in AscotDB's graphical front-end was to strike a balance between a tool specialized for one analysis task and a general-purpose system such as Tableau [17]. For AscotDB, we opted to develop one gadget per high-level activity (*e.g.*, data cleaning, image stacking, and time-series extraction) since the community typically performs a small set of such activities. AscotDB enables users to assemble these gadgets and thus activities in arbitrary combinations.

2.2 Front-End Python Support

Python has seen significant uptake among astronomers and other domain scientists because it is open-source, cross-platform, and easy for non-programmers to learn. Furthermore, third-party packages such as Numpy for array-oriented computing, Scipy for fast scientific algorithms, Matplotlib for publication-quality visualization, and IPython for interactive computation and data exploration provide a foundation for a host of more specialized packages implementing algorithms used in a wide variety of scientific fields.

For this reason, an intuitive Python interface to SciDB is essential to make it as useful as possible for the astronomical data analysis task. While SciDB does provide a low-level Python API for the execution of Array Query Language (AQL) and Array Functional Language (AFL) queries, the interface is too opaque for it to be useful to most astronomers. For this reason, we have created the SciDB-Py[3] package, which provides a high-level Python interface to SciDB, designed with an API familiar to users of the NumPy array computing library. Rather than interacting with data arrays via a query language like AQL or AFL, SciDB-py allows users to express operations in a high-level scripted manner, and thus makes the computing power of SciDB more accessible to data scientists.

[2]http://db.cs.washington.edu/myria/repository/uw-cat.html
[3]http://jakevdp.github.io/SciDB-Py

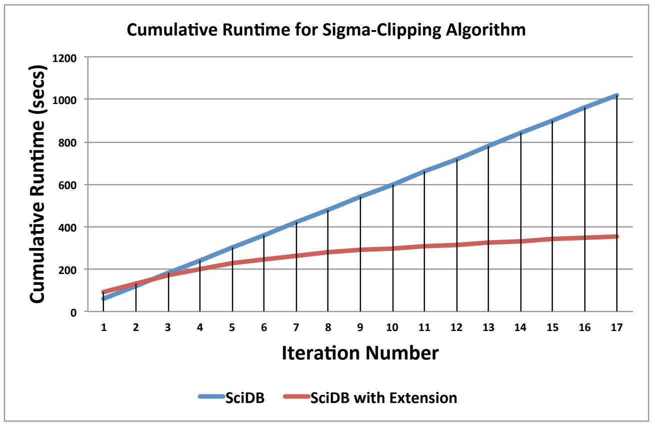


Figure 3: Cumulative runtime of an iterative data cleaning algorithm called *Sigma-Clipping* on SciDB (single machine) with and without *incremental iterative processing* extensions. 3D input array comprises 66 million cells.

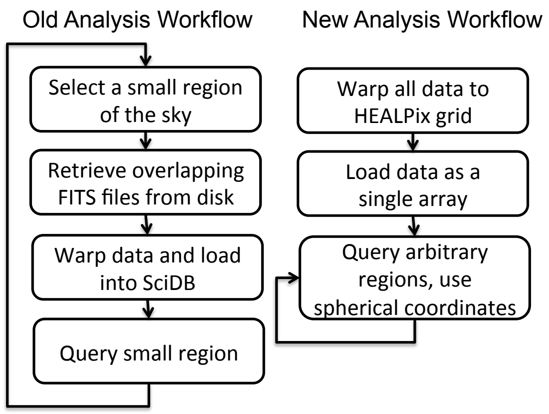

Old Analysis Workflow
New Analysis Workflow
Select a small region of the sky
Retrieve overlapping FITS files from disk
Warp data and load into SciDB
Query small region
Warp all data to HEALPix grid
Load data as a single array
Query arbitrary regions, use spherical coordinates

Figure 4: Data analysis workflow with and without our spherical-coordinates extension. In the old workflow, even if one caches the output of warping small regions, each region must be warped multiple times. Additionally, queries can never span large regions.

The following shows a short SciDB-Py session along with the equivalent SciDB AFL query commands:

SciDB-Py Python Code:

```
# Interface to a scidb cluster on the local machine via Shim
from scidbpy import interface
sdb = interface.SciDBShimInterface('http://localhost:8080')

# Create a random array with 1000 rows & 5 columns:
X = sdb.random((1000, 5))

# Compute the mean and standard deviation of each column
xcolmean = X.mean(0)
xcolstd = X.std(0)

# center and normalize each column
XC = (X - xcolmean) / xcolstd
```

Equivalent AFL session

```
/* The following are the AFL queries generated and
   executed automatically by the scidbpy module */

store(build(<f0:double> [i0=0:999,1000,0,i1=0:4,1000,0],
            random()/2147483647.0), X);

store(aggregate(X, avg(f0), i1), Xcolmean);
store(aggregate(X, stdev(f0), i1), Xcolstd);

store(project(apply(cross_join(X, Xcolmean, X.i1, Xcolmean.i1),
                    x, X.f0 - Xcolmean.f0_avg), x), tmp);
store(project(apply(cross_join(tmp, Xcolstd, tmp.i1, Xcolstd.i1),
                    x_0, tmp.x/Xcolstd.f0_stdev), x_0), XC);
```

The operations specified in this script are executed entirely within the SciDB architecture through the automatically generated AFL queries; the actual AFL queries are transparent to the user, and the Python interpreter itself never sees the data. By providing such an intuitive, high-level wrapper around the efficient storage and operations available in SciDB, SciDB-py makes the power of SciDB accessible to the average scientific Python user.

The Python interface to AscotDB is provided via the IPython notebook as illustrated in Figure 2(b). IPython is a set of tools designed to facilitate the entire life-cycle of a scientific project, from data exploration to publication. One component is a browser-based *notebook* with the support for editing and running Python code, as well as rich objects such as embedded plots, html objects, and mathematical expressions. Integrating the interactive, visual scientific computing environment of IPython with the power of the SciDB-Py interface within the AscotDB environment enables seamless sharing and processing of large astronomical datasets.

2.3 Graphical Interface and Python Integration

An important feature of AscotDB is the integration of both the graphical and python modes of interaction such that the user can go seamlessly back and forth between them. The main question to be answered is how to keep the working data in both modes synchronized. One possibility is to keep track of all the actions in the Ascot front-end in the form of SciDB queries and, after switching the mode to Python, have the system run the same queries in the same order in the background. This approach makes it easier to track the lineage of data and thus facilitates reproducibility. However, a session where a user interacts with a graphical engine can be long, can contain a large number of queries, and can dwarf the remaining Python script. Another approach is to move minimal amounts of data between the two interfaces to keep the working data synchronized. In AscotDB, we chose to synchronize the two interfaces by capturing user queries because of this approach's lineage tracking and reproducibility properties. To address the problem of large query sessions, however, in the AscotDB system, we are experimenting with a variety of techniques to automatically extract minimal query sets, which produce the results from a visual analysis session. Our key aim is to minimize the number of queries but without re-ordering them nor combining them into more complex and difficult-to-understand queries. It is critical for the user to identify in the summary script all the key steps that he or she took during the visual exploration.

3 Spherical Coordinates

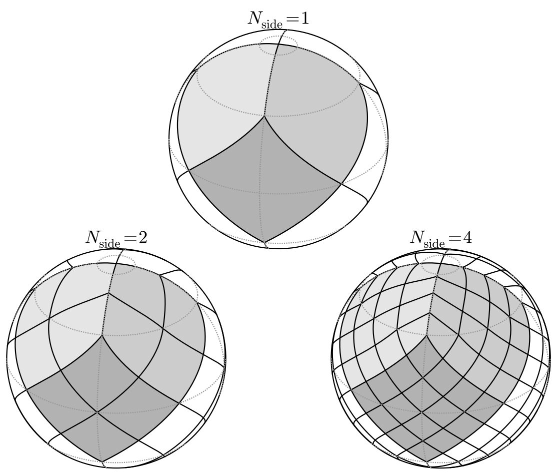
While SciDB provides an efficient environment for storing and manipulating very large array-oriented data, the arrays must be Cartesian: that is, data on a simple N-dimensional rectangular grid. In many scientific fields, especially those relying on astrophysical and geophysical data, the data lie on a sphere and this assumption breaks down.

In the case of a full-sky multiply-imaged optical dataset like LSST, this makes the global analysis of imaging data within SciDB very difficult. The data consists of individual images covering ~ 10 square degrees, which are taken at different times and cover partially-overlapping regions of the sky. Software tools exist that can *warp* overlapping images to a uniform tangential projection for the purposes of image stacking and differencing, but the distortion characteristics of a tangential projection are such that a single warping is sufficient over only a small region of sky. The result is that each individual exposure must be partially warped on the order of four separate times in order to analyze a full sky's worth of data. With images stored as binary files on disk, there is significant overhead in data I/O for these multiple warpings and coadditions. Fig. 4 illustrates this workflow in contrast with the new one that our approach enables.

To avoid this overhead both in performance and data management complexity, we add a middleware layer to SciDB that enables direct operations on spherical data, with the ultimate goal of pushing these middleware extensions into SciDB. This task boils down to the classic problem of full-sphere projections. Cartographers over the centuries have invented schemes for representing the earth's surface as a flat map; the storage of spherical data within SciDB requires an analogous flattening. Many such partitions have been proposed and used in the literature, each with distinct advantages and disadvantages.

The *Hierarchical Equal Area isoLatitude Pixelization* (HEALPix) [6] spherical pixelization is commonly used in astronomy, especially for datasets which cover a large portion of the sky. This approach has three key scientific benefits. First, it produces pixels of equal area, which significantly facilitates essential operations based on integration, such as kernel-smoothing, convolution, and regridding. Second, it can be represented in a number of useful ways, including a hierarchical encoding that facilitates spatial queries such as neighbor-searches, a ring encoding which facilitates efficient computations of the fast spherical harmonic transform (SpHT), and a *double-pixellization*, outlined below, which allows us to take advantage of the efficient array processing primitives already implemented within SciDB.

HEALPix is based on a tesselation of twelve equal-area partitions of the sphere, each of which are hierarchically sub-divided in a quad-tree structure to attain arbitrarily precise angular resolution. The twelve fundamental partitions are chosen so as to minimize the distortion within any single sub-pixel, and oriented so as to align sub-pixels along lines of constant latitude (vital for SpHT computations). The resolution of a HEALPix map is governed by the number of levels in the pixel hierarchy, specified via the parameter $N_{\text{side}} = 2^{\ell}$, with $\ell = 0, 1, 2....$ The number of pixels is $N_{\text{pix}} = 12N_{\text{side}}^2$, and it follows that the angular size of each pixel is



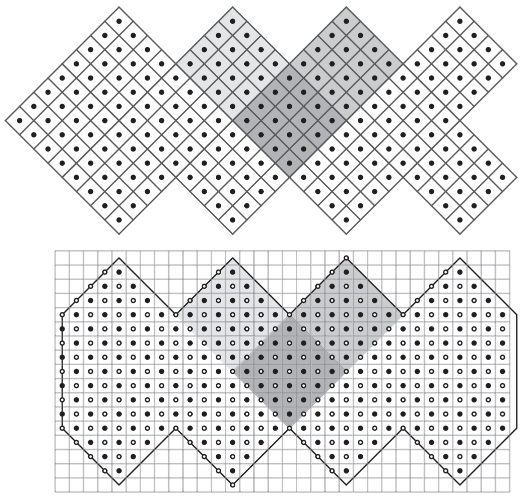


Figure 5: (Left) First three levels of the HEALPix hierarchical pixelization, shown in Orthographic projection. (Right) Schematic showing the construction of the HEALPix double-pixelization scheme for $N_{\rm side} = 4$

$\Omega_{\rm pix} = (\pi/3)N_{\rm side}^{-2}$. A visualization of the first three levels of HEALPix partitioning is shown in Fig. 5(left).

3.1 Representation and Transformations

While the equal-area hierarchical layout discussed above has advantages, one disadvantage is that it cannot be represented easily as a Cartesian grid of pixels. While the subdivisions within each of the 12 fundamental partitions do form a warped 2-dimensional grid, the junction of the three shaded regions in Fig. 5(left), for example, shows that the boundaries between the twelve fundamental partitions do not fit this Cartesian model.

We choose to address this using the *double pixellization* scheme outlined in [3]. As shown in the upper panel of Fig. 5(right), if the twelve fundamental partitions can be unwrapped onto the plane in a modified cylindrical equal-area projection (as specified in [6]), the pixels are arranged in a partially-filled regular grid, rotated 45° with respect to the primary x, y axes. Additionally, in this orientation, each valid (x, y) pair has an invertible mapping to a (latitude, longitude) pair (see details in [3, 6]), easing the ability to convert between the HEALPix representation and the standard world coordinate system.

The key insight for the double pixelization is that by interposing a new pixel between each of the standard HEALPix pixels, we can recover a grid with a standard orientation which preserves nearly all of the desirable properties of the original HEALPix pixelization, as shown in Fig. 5(right). Here, the solid points show the pixel centers from the standard pixelization, while the open points are the pixel centers added to construct the oriented grid. One disadvantage of this re-orientation is the fact that eight of the pixels (those located at the concave edges) now represent only $3/4$ the area of the other pixels: the missing area is accounted for by the addition of a new pixel at each pole. Thus the number of pixels in the double-pixelization scheme is $24N_{\rm side}^2 + 2$, compared to the $12N_{\rm side}^2$ pixels of the standard pixelization. One other disadvantage of the double pixelization is that regridding between HEALPix representations of different resolutions (i.e. changing $N_{\rm side}$) becomes much less straightforward. Note that on this grid, the pixels have a filling-factor of $\sim 75\%$, and at any desired resolution this can be efficiently stored within SciDB as a sparse Cartesian array.

The transformations needed to map a variety of image-plane coordinate representations onto HEALPix coordinates are defined via the WCS standard [2], which is implemented in standard Python tools such as *astropy*[4]. Astronomical image data typically comes in the form of *Flexible Image Transfer System* (FITS) files, whose

[4]"A Community Python Library for Astronomy" `http://astropy.org`

headers contain image projection parameters within the FITS/WCS standard. Because of this, the process for loading images onto the HEALPix grid is generic enough to be used for any modern astronomical data. Note that any individual image (or set of images at a given time) will fill only a small subset of the entire field: in SciDB, this can be represented as a sparse two-dimensional array.

There are two potential approaches warping observed data to the HEALPix grid: it can be performed as a preprocessing step prior to loading the data into the database, or as an operation within the database itself.

The first approach is to pre-warp images in an analog to the typical approach. Images, however, are now pre-projected onto a super-resolution grid of HEALPix pixels rather than a local tangential projection. A suitable super-resolution warp allows the precise pixel overlap to be treated as a second-order effect. While the typical approach warps each image to one of a number of local tangential representations, the new spherical approach requires each image to be warped only once onto a global HEALPix representation, cutting computation and data access costs up to about 75% for the warping calculation. The final dataset is represented in SciDB as a sparse 3-dimensional array, containing HEALPix-gridded two-dimensional images indexed by the time at which they were taken. Fig. 4(right) illustrates the workflow when using this method.

The second approach pushes the warping step into SciDB itself, obviating the need for preprocessing outside SciDB. By choosing an extremely fine resolution for the raw data within SciDB, the flux within each input image pixel can be stored at the location of the pixel center with nearly arbitrary resolution[5], leaving all other pixels empty. The result is an extremely sparse data structure which efficiently indexes the actual pixel-level data, without need for any preprocessing step. The translation of this sparsely-indexed data into the warped dense HEALPix representation is a well-defined operation rooted in 2D spatial convolution. Because SciDB does not currently implement efficient convolutions, we chose the first approach, leaving the second option for future work.

3.2 Operations

Once the data is warped and stored on a HEALPix grid within SciDB, the basic class of desired operations come virtually for free.

- **Selection:** SciDB implements efficient selection along the axes of the stored array. This allows efficient selection of data at a given time, as well as angular selection on the sky. Because there is a straightforward algebraic relationship between typical local sky queries and the HEALPix double-pixel coordinates, desired data selection can be implemented as a native SciDB query.
- **Aggregation:** In the same way, built-in SciDB aggregates can be used to compute statistics of interest, most notably the *co-addition* of sub-images across time.
- **Iterations:** Many essential image processing operations, from trimmed co-addition to PSF matching, require an iterative approach to the data. As discussed in Section 2.1, we extended SciDB to enable efficient iterative operations on large datasets. This work can be used with the spherical projections to allow powerful iterative analysis of full-sky datasets.
- **Convolution:** Many image processing tasks require efficient convolutions, from warping to object detection and tracking to PSF matching for image coaddition. Though SciDB currently implements a rudimentary windowing convolution, it does not contain the full convolution operator required for these tasks. Implementation of efficient convolutions is an important area of future work, as it will allow a wide variety of image processing tasks to be implemented in a scalable way within SciDB.
- **Regridding:** With the HEALPix double-pixelization scheme, regridding can be implemented using a convolution operation, with some extra care at the boundaries.

[5]Technically, due to the int64 index labels, the resolution is limited to $\gtrsim 0.1$ pico-arcsec, which is more than sufficient for any current astronomical observation.

Because the translation between spherical coordinates and SciDB array coordinates is negligible in time, the performance of the above operations is equal to the performance of the underlying SciDB queries.

4 Related Work

Because the AscotDB project is so broad in its aims and the array of technologies it utilizes, there is related work in many areas.

EXTASCID [4] is an extensible parallel system that provides native support both for arrays and key-value relations. The execution strategy in EXTASCID is through the UDA interface with easy reasoning for parallelism, while AscotDB provides parallelism through SciDB with native support for arrays only. On the other hand, AscotDB provides support for iterations and direct query on spherical data with two modes of interaction: visual front-end and Python programmatic front-end.

AstroShelf [11], similar to AscotDB, is a collaborative system that enables astrophysicists to investigate celestial objects using catalog data hosted at different sites. Astroshelf is a stream processing system that matches events (either celestial events or user annotations) to users who are likely to be interested in them. In contrast, AstroDB focuses on interactive, collaborative processing of archived data.

Blaze [1], often billed as the "Next generation NumPy", aims to implement a very general array framework within Python. It will support a wide variety of table and array-like structures capable of handling arbitrary local and distributed memory layouts, type heterogeneity, axis labels, missing or masked values, and other commonly-requested features. SciDB is just one of a wide variety of potential backends for large-scale distributed array storage and computing through Blaze: in this light, SciDB-Py can be considered a precursor to the much more ambitious framework Blaze promises to implement.

For spherical partitioning of datasets, one well-studied method is the Hierarchical Triangular Mesh (HTM) [16], used in the SDSS SkyServer. HTM is efficient, but suffers two disadvantages compared to the current work: First, its triangular representation does not lend itself to the Cartesian representation required to take advantage of operations within SciDB. Second, its non-equal pixel areas add overhead to integration-based tasks common in image manipulation and processing.

5 Conclusion

Efficient data exploration, visualization, and analysis in the context of future large astronomical surveys will require a combination of advances in the areas of large-scale data storage, processing, and visualization, as well as specialized operations suitable for data on the sphere of the sky. We have presented one set of approaches to this problem: AscotDB. We utilize a HEALPix-based spherical projection to unravel spherical data and store it within a three-dimensional SciDB array, indexed both by time and by location on the sphere. We choose the representation carefully so as to utilize the full functionality of the native array-processing power of SciDB. With the current capabilities of SciDB, this approach already decreases the image pre-processing load by around a factor of four; with future work on a native convolution operator within SciDB, the need for an out-of-database preprocessing will be eliminated. The work here, along with future directions of development in SciDB, points to a system where a full-sky worth of time-domain astronomy imaging data can be directly stored and indexed in a way that will enable efficient image analysis tasks to be performed on-demand and at-scale.

Acknowledgments: This work is partially supported by grants from the LSST corporation, NASA grant NNX09AK84G, DOE award DESC0002607, NSF grants ACI-1226371 and IIS-1110370, and the Intel Science and Technology Center for Big Data.

References

[1] Blaze: A python compiler for big data. `http://continuum.io/blog/blaze`.

[2] M. R. Calabretta and E. W. Greisen. Representations of celestial coordinates in FITS. *A&A*, 395:1077–1122, December 2002.

[3] M. R. Calabretta and B. F. Roukema. Mapping on the HEALPix grid. *MNRAS*, 381:865–872, October 2007.

[4] Yu Cheng and F. Rusu. Astronomical data processing in extascid. In *Proceedings of the 25th International Conference on Scientific and Statistical Database Management*, SSDBM, pages 47:1–47:4, 2013.

[5] Cudre-Mauroux et. al. A demonstration of scidb: a science-oriented dbms. In *Proc. of the 35th VLDB Conf.*, pages 1534–1537, 2009.

[6] K. M. Górski, E. Hivon, A. J. Banday, B. D. Wandelt, F. K. Hansen, M. Reinecke, and M. Bartelmann. HEALPix: A Framework for High-Resolution Discretization and Fast Analysis of Data Distributed on the Sphere. *ApJ*, 622:759–771, April 2005.

[7] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing In Science & Engineering*, 9(3):90–95, 2007.

[8] Large Synoptic Survey Telescope. `http://www.lsst.org/`.

[9] D. Marcos, A. J. Connolly, K. S. Krughoff, I. Smith, and S. C. Wallace. ASCOT: A Collaborative Platform for the Virtual Observatory. In *Astronomical Data Analysis Software and Systems XXI*, volume 461 of *Astronomical Society of the Pacific Conference Series*, page 901, 2012.

[10] M. Moyers, E. Soroush, S.C. Wallace, S. Krughoff, J. Vanderplas, M. Balazinska, and A Connolly. A demonstration of iterative parallel array processing in support of telescope image analysis. In *Proc. of the 39th Int. Conf. on Very Large DataBases (VLDB)*, 2013.

[11] P. Neophytou, R. Gheorghiu, R. Hachey, T. Luciani, D. Bao, A. Labrinidis, E. G. Marai, and P. K. Chrysanthis. Astroshelf: understanding the universe through scalable navigation of a galaxy of annotations. In *SIGMOD'12: Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, pages 713–716, 2012.

[12] Travis E. Oliphant. Python for scientific computing. *Computing in Science & Engineering*, 9(3):10–20, 2007.

[13] Fernando Pérez and Brian E. Granger. IPython: a System for Interactive Scientific Computing. *Comput. Sci. Eng.*, 9(3):21–29, May 2007.

[14] J. Rogers et al. Overview of SciDB: Large scale array storage, processing and analysis. In *Proc. of the SIGMOD Conf.*, 2010.

[15] E. Soroush, M. Balazinska, and D. Wang. ArrayStore: A storage manager for complex parallel array processing. In *Proc. of the SIGMOD Conf.*, pages 253–264, June 2011.

[16] A. S. Szalay, J. Gray, G. Fekete, P. Z. Kunszt, P. Kukol, and A. Thakar. Indexing the Sphere with the Hierarchical Triangular Mesh. *eprint arXiv:cs/0701164*, January 2007.

[17] Tableau. http://tableausoftware.com.