The CDS HEALPix library

In Java, Rust and WebAssembly

F.-X. Pineau & the CDS team

francois-xavier.pineau@astro.unistra.fr

CENTRE DE DONNÉES ASTRONOMIQUES DE STRASBOURG

Abstract

The CDS is releasing a new HEALPix library implemented in Java. Before possibly porting it in C, we are experimenting with the Rust programming language. It allows the library to be compiled into WebAssembly or native code, and thus to easily plug it into web browsers, Python codes or PostgreSQL, to name but a few. The library is distributed under the 3-clause BSD licence. It focuses on our own needs, on performances and accuracy.

We are investigating the potential usage of the WebAssembly version into Aladin Lite. The objective is twofold: supporting deeper orders (up to 24, the present limit being 13), and changing the current GPL license to a less restrictive one. Aladin desktop has already started to resort to the Java version.

Unlike the "official" library, coming from the cosmology community, our Java version do not currently supports Spherical Fourier Transformations and do not support the RING scheme (also it is able to transform cells numbers from the nested scheme to the ring scheme and vice-versa). In return, it do support additional features like: distinguishing between cells partially and fully overlapped by a cone; exact cells-overlapped-by-cone solution; very-fast approximate cells-overlapped-by-cone function dedicated to cross-matches; supporting self-intersecting polygons of any size; fast ordered list of small cells surrounding a larger cell; MOC support in cells-overlapped-by-cone queries; BMOC support (MOC with an additional flag for each cell); etc.

Motivations

The motivations bringing the CDS to develop an HEALPix library from scratch are various.

Licence control
Switch from GPL ("official library") to 3-clause BSD ("CDS library") to change the Aladin Lite licence and be compatible with, for example, the Astropy licence.

Internal expertise
Develop an internal expertise in a key component of both CDS services and the HiPS IVOA standard.

Aladin Lite support
Bring the deepest addressable resolution from order 13 to order 24 and add support for polygons.

Easier evolutions
Make evolutions fitting with the CDS needs easier (mainly in Aladin, Aladin Lite and the X-match service).

Languages

Java
Since it is widely used at CDS (Aladin, SIMBAD, Cross-match service, etc), the HEALPix library has been developed first in Java. All features mentioned in this poster were made available in the Java version.

Rust
Rust is a recent open-source language sponsored by Mozilla and *pursuing the trifecta: safety, concurrency, and speed* (Rust weekly newsletter). It aim at offering *high-level ergonomics and low-level control* (online Rust book). Since it is a compiled language, we use Rust to generate both WebAssembly files and static or dynamic libraries that can be called from Python or PostgreSQL. So far, mainly basic HEALPix features meeting with the Aladin Lite needs have been implemented in Rust (cell number from coordinates, cell center, cell vertices, cell neighbours, approximate cells-in-cone, projection/deprojection).

WebAssembly
WebAssembly is a bytecode standardized by the W3C and compatible with all recent Web browsers. It aims at complementing Javascript, with better performances, and can be generated from compiled languages like C, C++ or Rust.

C ?
Rust pre-compiled binaries are similar to C one's and could be distributed in softwares like astropy. However, instaling Rust tools is necessary if a user want to manually compile a module from the source code. It is thus not straitforward to integrate Rust code in large projects like astropy or PostgreSQL modules. It may bring us to write a C version of the library.

Features

Comparing to the "standard library", so far the CDS version

do not supports :

the RING scheme , but the library offers functions converting a NESTED cell number into a RING cell number (and vice-versa). For indexation purposes, the NESTED scheme (based on Z-order curves) has better locality properties than the RING scheme.

spherical harmonics computations and Fast Fourier Transforms which are extensively used in the cosmology community.

supports :

Projection/deprojection : compute Euclidean (X, Y) coordinates from Spherical (α, δ) coordinates, and vice-versa. The library contains an internal projection (see Fig. 1) and a version compatible with the WCS HPX projection. The shifted-rotated internal projection is used to compute the cells values (each of the 12 base cells is sub-divided and indexed following a z-order curve)

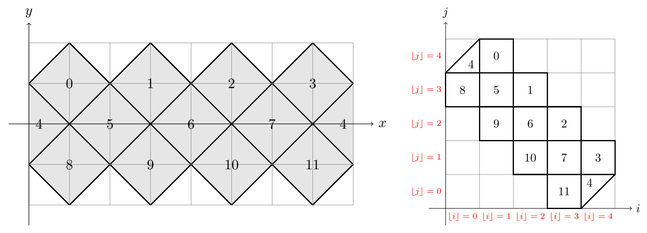

Figure 1: Left: HEALPix internal projection plane, showing the 12 base cells. Right: shifted-rotated projection plane used to compute cells number.

Cell partially/fully-overlapped-by-cone : this binary coverage information allows to avoid useless time-consuming distances computations when, e.g., performing a cone-search query on a table.

Exact cells-overlapped-by-cone solution : no false positive cells in cone-search queries to avoid (again) useless distances computation, and possible disks accesses.

Largest center-to-vertex distance upper limit : depending on both the order and the position of the cell on the sky. Used in fast but approximative cells-overlapped-by-cone queries.

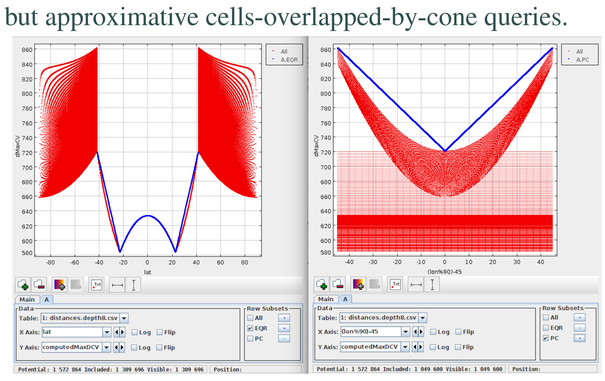

Figure 2: Computed upper limit (in blue) as a function of (α, δ) for the order 8. Red: exact value computed for every order 8 cells. Plot made using TOPCAT.

Very-fast approximate cells-overlapped-by-cone function : dedicated to map/reduce based cross-matches, in which the number of sources in each cone is very small.

Fast ordered list of small cells surrounding a larger cell : used for example to retrieve the list of sources in a large cell taking into account border effects. The ordering is important to maximize the sequential access to data stored on spinning HDDs.

Self-intersecting polygons of any size : (see Fig. 3), also provides the list of cells fully and partially overlapped by the polygon. Like in the "official" library, we so far use an approximation: we do as if a cell border between two vertices was on a great-circle. An exact solution is possible but would be computationally less efficient.

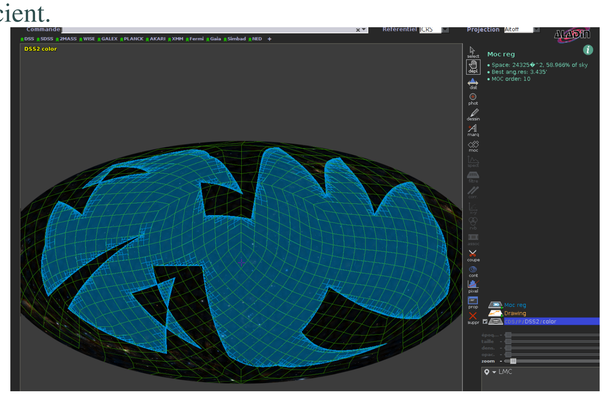

Figure 3: Example of MOC generated by a large scale self-intersecting polygon in Aladin

Read-only BMOCs : a BMOC is an extension of a MOC but storing for each cell an additional status flags telling if the cell is partially or fully covered by the area the MOC represents.

Extra axis-support : as an experiment, we added the possibility to compute a cell number based on a spherical positions plus an extra axis. One can imagine an additional z-axis to Fig. 1: the 12 base cells become 12 base cubes, each hierarchically divided and indexed according to a 3D z-order curve. We still have to implement queries returning 3D BMOCs.

Technical details

Projection: simplified equations

HEALPix is quite extensively described in Calabretta (2004), Górski et al. (2005), Calabretta & Roukema (2007) and Reinecke & Hivon (2015). It is first of all an equal-area projection composed from two other projections. Internally we have chosen a projection scale such that all coordinates in the projection plane are $\in [0, 8[$ on the X-axis and $\in [-2, 2]$ on the Y-axis. The internal simplified equations are:

Collignon (pseudo-cylindrical equal-area) projection in the polar caps, for $\alpha \in [0, \pi/2]$ and $\sin\delta > 3/2$:

$$\left\{\begin{array}{l} t = \sqrt{3(1-\sin\delta)} \\ X = (\alpha\frac{4}{\pi} - 1)t + 1 \\ Y = 2 - t \end{array}\right. \Rightarrow \left\{\begin{array}{ll} \alpha \in [0, \frac{\pi}{2}] & t \in [0, 1[\\ \sin\delta \in]\frac{2}{3}, 1] & \rightsquigarrow\ X \in]0, 2[\\ & Y \in]1, 2] \end{array}\right.$$

Cylindrical equal-area projection in the equatorial region:

$$\left\{\begin{array}{l} X = \alpha \times \frac{4}{\pi} \\ Y = \sin(\delta) \times \frac{3}{2} \end{array}\right. \Rightarrow \left\{\begin{array}{ll} \alpha \in [0, 2\pi] & \rightsquigarrow\ X \in [0, 8] \\ \sin\delta \in [-\frac{2}{3}, \frac{2}{3}] & \rightsquigarrow\ Y \in [-1, 1] \end{array}\right.$$

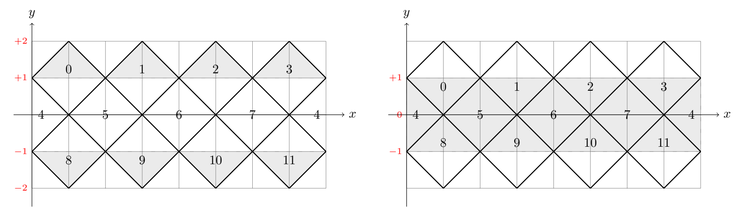

Figure 4: Left: Collignon projection. Right: CEA projection.

Precision at poles

The formula $t = \sqrt{3(1-\sin\delta)}$ causes non-negligible numerical inaccuracies near the poles due to the $1 - \sin\delta$ expression it contains:

- $\arcsin(1 - 1.0 \times 10^{-15}) \approx 89.99999919\,\text{deg}$;
- $\frac{\pi}{2} - \arcsin(1 - 1.0 \times 10^{-15}) \approx 2.917\,\text{mas}$.

We thus replaced the previous equation by the equivalent but numerically stable form: $\boxed{t = \sqrt{6}\cos(\frac{\delta}{2} + \frac{\pi}{4})}$.

Demo: $1 - \sin\delta = 1 + \cos(\delta + \frac{\pi}{2}) = 2\frac{1+\cos(2(\frac{\delta}{2}+\frac{\pi}{4}))}{2} = 2\cos^2(\frac{\delta}{2} + \frac{\pi}{4})$. This form is also computationally less expensive since we spare a time-consuming square-root operation (the square-root applying here on a constant instead of a variable).

The exact cells-in-cone solution

Let's note Ω_c, Ω_p the surface covered by the cone and the cell (or pixel) respectively. The basic cell selection algorithm is:

- 4 vertices in the cone $\Rightarrow \Omega_c \cap \Omega_p = \Omega_p$
- 1-3 vertices in the cone $\Rightarrow \Omega_c \cap \Omega_p \neq \varnothing$
- cell contains a special point $\Rightarrow \Omega_c \cap \Omega_p \neq \varnothing$

There is 4 + 1 special points:

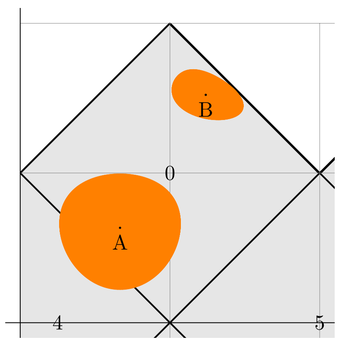

Figure 5: Tissot's indicatrices.

- 4 points such that the slope of the tangent line to the projected cone on that point $= \pm 1$
- for large cells the center of the cone is also a special point

Computing the "special points" coordinates

We use the Haversine formula to get an accurate cone expression at small radii:

$$\Delta\alpha = 2\arcsin\left(\sqrt{\frac{\sin^2\frac{\theta}{2} - \sin^2\frac{\delta-\delta_0}{2}}{\cos\delta_0\cos\delta}}\right)$$

In the equatorial region, the equation of tangent lines is:

$$\frac{\mathrm{d}\Delta X}{\mathrm{d}Y} = \frac{\mathrm{d}\Delta X}{\mathrm{d}\Delta\alpha}\frac{\mathrm{d}\Delta\alpha}{\mathrm{d}\delta}\frac{\mathrm{d}\delta}{\mathrm{d}z}\frac{\mathrm{d}z}{\mathrm{d}Y} = \pm 1$$

The projection formulae: $z = \sin\delta$, $X = 4/\pi\alpha$, $Y = 3/2z$ lead to $\frac{\mathrm{d}\Delta X}{\mathrm{d}\Delta\alpha} = 4/\pi, \frac{\mathrm{d}\delta}{\mathrm{d}z} = \frac{1}{\cos\delta}, \frac{\mathrm{d}z}{\mathrm{d}Y} = 2/3$ and, finally, we find the special points latitudes by solving numerically (Newton's method):

$$\boxed{\frac{1}{\cos\delta}\frac{\mathrm{d}\Delta\alpha(\delta)}{\mathrm{d}\delta} \mp \frac{3\pi}{8} = 0}.$$

In the polar caps, the equation of tangent lines is:

$$\frac{\mathrm{d}\Delta X}{\mathrm{d}Y} = \frac{\mathrm{d}\Delta X}{\mathrm{d}\delta}\frac{\mathrm{d}\delta}{\mathrm{d}z}\frac{\mathrm{d}z}{\mathrm{d}Y} = \pm 1$$

with $z = \sin\delta$, $t = \sqrt{3(1-z)} = \sqrt{6}\cos(\frac{\delta}{2} + \frac{\pi}{4})$, $X = (\frac{4}{\pi}\alpha - 1)t + 1$, $Y = 2 - t$, leading to $\frac{\mathrm{d}\delta}{\mathrm{d}z} = \frac{1}{\cos\delta}$, $\frac{\mathrm{d}z}{\mathrm{d}Y} = \frac{2}{3}t$ and, finally, we find the special points latitudes by solving numerically (Newton's method):

$$\boxed{\frac{t(\delta)}{\cos\delta}\frac{\mathrm{d}}{\mathrm{d}\delta}\left[(\frac{4}{\pi}\alpha(\delta) - 1)t(\delta)\right] \mp \frac{3}{2} = 0}.$$

References

Calabretta, M. R. 2004, ArXiv Astrophysics e-prints. astro-ph/0412607

Calabretta, M. R., & Roukema, B. F. 2007, MNRAS, 381, 865

Górski, K. M., Hivon, E., Banday, A. J., Wandelt, B. D., Hansen, F. K., Reinecke, M., & Bartelmann, M. 2005, ApJ, 622, 759. astro-ph/0409513

Reinecke, M., & Hivon, E. 2015, A&A, 580, A132. 1505.04632

François-Xavier Pineau (francois-xavier.pineau@astro.unistra.fr)
Université de Strasbourg, CNRS, Observatoire astronomique de Strasbourg, UMR 7550, F–67000 Strasbourg, France

ADASS XXVIII, Astronomical Data Analysis Software & Systems
11 – 15 Nobvember 2018, College Park, Maryland, USA