URSI GASS 2023, Sapporo, Japan, 19 – 26 August 2023

Modernizing The Radio Interferometry Software Ecosystem Towards Distributed, Flexible And Reproducible Workflows

Landman Bester*[(1)(2)], Jonathan Kenyon[(2)(1)], Simon Perkins[(2)] and Oleg Smirnov[(2)(1)]
(1) South African Radio Astronomy Observatory, Cape Town, South Africa
(2) Rhodes University, Grahamstown, South Africa

When combined with the increasing sensitivity requirements of modern scientific objectives, the data rates associated with current and upcoming radio telescopes cause critical tension in radio interferometric software development. Large data rates necessitate highly optimised, end-to-end, automated pipelines capable of reliably and efficiently converting raw interferometric data into science ready data products. Conversely, due to the need to extract scientifically useful information near the noise floor in many observations, it is very difficult to meet dynamic range requirements with generic pipelines. This dichotomy is well-illustrated by numerous projects for which automated science data processing pipelines do not produce satisfactory results, necessitating manual intervention. The ubiquity of projects that require this treatment illustrate why flexibility and ease of development is often more important than performance at all costs. In this talk we present a Python-based framework that leverages tools from the PyData community to create a flexible development environment for radio interferometric data processing with minimal trade-off against computational performance. This framework utilises dask to implement a declarative programming model that is capable of running on both single and multi-node computing infrastructures with minimal changes to the underlying code base. Python's global interpreter lock (GIL) is, in many cases, bypassed by using numba to just-in-time compile performance critical parts of the toolchain with the LLVM compiler framework. This translates Python code into machine code with performance comparable to C/C++.

The talk will focus on four critical pillars, viz. dask-ms, QuartiCal, pfb-imaging and stimela, of the ecosystem which together enable the user to compose flexible and powerful end-to-end data processing pipelines that can be deployed on a variety of computing infrastructures, including cloud services such as Amazon Web Services. One of the major design principles behind dask-ms, which serves as the data access layer, is to allow for seamless interoperability between the measurement set and cloud-native formats such as Zarr and Arrow, both of which allow for parallel reading and writing of data. Data are exposed to the Python layer through xarray datasets, a very versatile and intuitive format which, because of its similarity to Pandas, should be familiar to most data scientists. QuartiCal and pfb-imaging utilise this data format to implement performant, state of the art calibration and imaging algorithms. Importantly, by standardising their output formats, they are able to inter-operate in a way that avoids unnecessarily large intermediary data products (eg. corrected and model visibilities). Both applications implement a form of multi-level parallelism that allow them to scale both horizontally (i.e. simultaneously processing multiple independent chunks of data) and vertically (i.e. multi-threading within a chunk) thus making it possible to balance computational and memory requirements to make optimal use of the available resources. In addition, because of their pure Python interfaces and adherence to software engineering best practices, both applications are able to leverage the plethora of packages that are actively being developed and maintained by the (much larger) PyData community.

The final leg of the above quartet is the stimela pipelining framework. Stimela defines a YaML specification that allows chaining together arbitrary data processing steps by means of *recipes*. Recipes consist of inputs and outputs, defined by the recipe *schema*, followed by a sequence of steps where each step invokes a *cab* (or another recipe) by mapping, after pre-validation, the *params* section of the step onto the cab definition. Cabs, like recipes, have schemas specifying their inputs and outputs but also contain *cargo* i.e. its content. The cargo of a cab can be any executable shell command, a python function, a CASA task or even a snippet of Python code. Depending on the chosen *backend*, cabs can be executed natively on the host system, possibly in dedicated *virtual environments*, or as containers. This makes it possible to combine the heterogeneous radio interferometry software stack, often with conflicting dependencies, into a recipe definition that can be executed with a single command. Stimela ships with cab definitions for most common radio interferometry software but also allows the user to include custom defined cabs of their own. This use-and-include philosophy allows users to structure their cab definitions and recipes into modular, reusable and, most importantly, reproducible libraries. We'll conclude the talk by showcasing recent MeerKAT results obtained from end-to-end pipelines, made available as Stimela recipes, based on the framework described above.