Reproducibility in Science Report

Deliverable 9.3

Authors Stratos Gerakakis
Maria Ieronymaki
Michele Iacobellis

Date September 26th, 2018

Work Package WP09 - Reproducibility in Science

DocId pkh112-06-1.0

Revision History

Version	Authors	Date	Changes
1.0	Stratos Gerakakis Maria Ieronymaki Michele Iacobellis	Sep 26th, 2018	Initial Version

Contents

1. Introduction

This document presents the work that was done researching the current state of art regarding Reproducibility in Science, the results of a survey regarding Reproducibility in Science that we circulated in small group of scientists and our proposed course of action regarding the work we would like to proceed developing in WorkPackage 9.

We also present our findings regarding the assistance of scientific work with the use of GPU processing facilities.

2. Status of Reproducibility in Science

We started our research on Reproducibility in Science by examining the latest developments on the topic. We have identified several tools and services that are available online and aim to provide solutions towards reproducible science. The different tools we have identified cover a variety of provided functionalities that constitute the current state of the art in reproducibility in science.

We have evaluated these tools and workflows and in this section we will briefly present them, together with our impressions and conclusions after using them.

2.1. Reproducibility Theory

The subject of Reproducibility and the ability of scientists to exactly reproduce and confirm a given result, is central to Science in general. At the theoretical level, researchobject (http://www.researchobject.org) have produced plenty of useful theoretical information on their website, regarding the subject of reproducibility, but except for a list of suggested initiatives and resources, the wandering scientist in search for a concrete reproducibility workflow will still be left without explicit direction.

2.2. Workflow Tools

There are plenty of tools specifically made to help scientists define and execute a specific set of tasks, implemented by executing local (or sometimes remote) code, scripts, and other sub-workflows. Each component only being responsible for a small fragment of functionality, therefore many components working together in a pipeline order to obtain the ultimate goal of the workflow, performing a useful task.

We evaluated, two of the most popular workflow engines.

2.2.1. Taverna

Taverna (https://taverna.incubator.apache.org) is an open source and domain-independent Workflow Management System – a suite of tools used to design and execute scientific workflows and aid in silico experimentation.

Taverna also includes the Taverna Workbench that is able to run and monitor a workflow. Nevertheless, the taverna workflows can also be run by a command line execution tool, remote execution server, or a provided online workflow designer.

Taverna is an Apache incubator project since 2014, while it has been an open source project since 2003.

2.2.2. Kepler-Project

The Kepler Project (https://kepler-project.org/) is an open-source, GUI-based, scientific workflow system that aims to to help scientists, analysts, and computer programmers to create, execute, and share models and analyses across a broad range of scientific and engineering disciplines enabling the the project integration between projects with different characteristics. Kepler can operate on data stored locally or remotely and supports a variety of formats.

It is used for integrating disparate software components, such as merging "R" scripts with compiled "C" code, or facilitating remote, distributed execution of models. Using Kepler's graphical user interface, users select and then connect pertinent analytical components and data sources to create a "scientific workflow" an executable representation of the steps required to generate results. The Kepler software helps users share and reuse data, workflows, and components developed by the scientific community to address common needs.

2.3. Online services

This is a list of online services that deal with the concept of Reproducibility in Science that we evaluated.

2.3.1. Open Science Framework

The Open Science Framework (https://osf.io/) OSF, is a free, open source service of the Center for Open Science (https://cos.io/). It's a non-profit organization aiming to align scientific practices with scientific values by improving openness, integrity and reproducibility of research.

They offer a very impressive online presence, allowing users to create multiple online projects. They offer space to upload files and manage their projects. Each project can be individually configured to multiple components offering a high level of project customization. Each uploaded item, gets its own short unique identifier that makes it easy addressable.

It provides an Application Programming Interface (API) that allows simple project maintenance tasks (uploading files, creating folders) to be automated from the command line, or to allow access to information stored in OSF by other applications.

2.3.2. Codeocean

Codeocean (https://codeocean.com/) is a for profit organization that states as its mission the intention to make the world's scientific code more reusable, executable and reproducible. It is a cloud-based computational reproducibility platform that provides researchers and developers an easy way to share, discover and run code published in academic journals and

conferences.

Their platform provides open access to the published software code and data to view and download for everyone for free. The users can execute all published code without installing anything on their personal computer. Everything runs in the cloud on CPUs or GPUs according to the user needs. They make it easy to change parameters, modify the code, upload data, run it again, and see how the results dynamically change.

It looks like an online Integrated Development Environment (IDE) and their website is very nicely done. They offer a free tier, and extended execution times, for accounts that have been opened with educationally based email accounts, but unfortunately for extensive use of their platform, eventually a user will have to start paying a subscription fee that depends on the amount of execution time they spend on Codeocean's servers.

2.3.3. Zenodo

Zenodo (https://zenodo.org/) according to their website is built and developed by researchers, to ensure that everyone can join in Open Science. Funded by the OpenAIRE project (https://www.openaire.eu/), which in turn is another Horizon 2020 supported project funded by the EU, it offers services allowing scientific work to be findable, accessible, interoperable and reusable.

Their services mainly allow online storage of data, by offering users a very reasonable 50GB per uploaded file (with the ability to extend that size limitation, on a case by case basis), and allowing a tremendous amount of metadata to be associated with each one. It caters to the general scientific workflow, allowing the imposing of embargoes, where the repository restricts access to scientific results until the end of the embargo, at which point the content automatically becomes publicly available. It also makes it easy to add EC funding information and reports via OpenAIRE.

It offers persistent identifiers (DOIs) for all uploads to their service, free from cost. They aldo include metrics and statistics on the use of the uploaded content. Their API access allows access to content stored on site, even though it seems to require a username and password, even for publicly available info.
Overall, we were very impressed by the functionality and services offered by Zenodo, and we plan to fully support their services in our proposed plan for our Reproducibility in Science efforts.

2.3.4. Gitlab/Github/Bitbucket

All three services in this section, Github (https://github.com), Gitlab (https://gitlab.com) and Bitbucket (https://bitbucket.org) offer similar tools that mainly cater for the hosting and online development of source code.

Gitlab and Bitbucket, in addition to publicly available repositories, are also offering private repositories. This option might make them a better candidate for users that want to start their project as a private repository, but later on, closer to publication time, switch to a fully public repository.

Being implemented by well established organizations, they offer a full suite of online development tools, with bug/issues management, wiki pages, and file hosting capabilities. All of them offer API access to the files hosted there and between the three of them, they have captured the majority of online code development. We are very familiar with the list of tools offered by these companies, and we plan to integrate with their services in our Reproducibility in Science efforts.

2.4. Analysis and Usability

From the list of the existing Reproducibility services that we chose to evaluate, we came to the following conclusions.

Most of the services offer a very streamlined and user friendly interface that specializes in the features that each service has decided are the most important to their users. In our evaluation, we did not find any service that provides a complete, unobtrusive, workflow that will explain and fit the needs for a complete reproducible workflow. The reproducibility surveys (further analyzed in a later section), also validates this as the users we questioned were not able to find something that it would be convenient for them to use.

Most of the services were free to use, although some of them were business endeavors that offered a free trial or a free tier. Unfortunately for a more serious and heavy usage the users would require to pay fees in order to continue using the service, which is something that we do not believe our audience is willing to do, simply to have their work in a reproducible form.

Furthermore, the online services we evaluated did not offer a clear way to fully automate the complete process of reproducing the authors work. Some offered online storage space only, while others offered computational resources so the produced code could execute online. But none offered an easy way to reproduce the full cycle.

Some services, offered a very important functionality, of offering free DOI (Digital Object Identifiers) numbers for the final published datasets, but they were only acting as a file storage and not as a computational environment.

Overall from what we evaluated, we saw plenty of functionality offered in the existing services, with interesting features spread between the various services, but we were not happy enough to use or suggest a single one of the online providers as being able to offer a complete and user friendly reproducible environment. On the other hand, it would be really beneficial if we could utilize some of the functionalities offered from the existing services

(especially the free to use ones) and combine the best of each one, in a new workflow tool (as we propose later on section **4. Proposed Course of Action**) This way we provide added value to the existing services, while at the same time avoid duplicating the work that has already been produced.

3. Reproducibility Survey

3.1. Introduction

We have decided to conduct a survey in order to gather valuable information for the views of scientists on reproducibility.

For this purpose, we have created on online questionnaire, with an introduction page, publicly available at https://beyondplanck.page.link/reproducability_survey, that would help us gather information about the development habits of scientists, along with their experience with the subject of Reproducibility in Science. The intention was to scope out, how familiar scientists are with Reproducibility is general, if they do follow any reproducibility workflows, and how they generally structure their work, so we might be able to tailor a potential Reproducibility workflow to their existing workflows.

The topics that were covered in our survey, included usage of Version Control Software, backup strategies, coding experience, use of operating systems, familiarity with virtual machines or containers and Reproducibility workflow specifics.

An invitation to the online survey was distributed to our audience by email. Our audience included the consortium members, scientists we have previously collaborated with in the past, connections in other scientific fields including research fellows, scientific personnel, professors, etc.

3.2. Results

So far, 39 responses have been collected. The answers included scientists from multiple scientific fields that were covering a wide demographic spectrum and variety on ages and years of research. The total number of survey responses is not exceptionally large, but we intent to host a version of the survey on the BeyondPLANCK website, when this goes online shortly, and solicit more responses from our extended audience there. Once we have more responses we plan to provide an update report on these results.

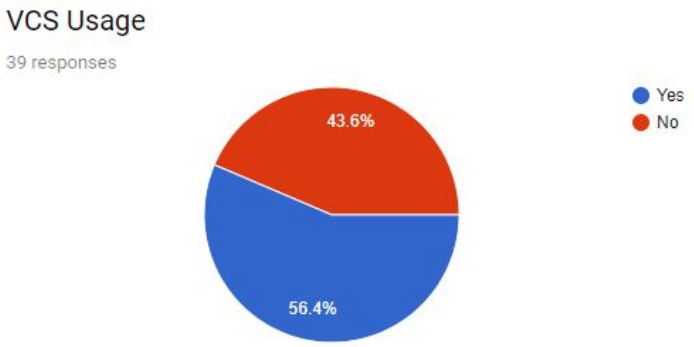
3.3. Findings

In this section we will analyse the findings of the survey that illustrate what are the views of the scientific community on the topic of reproducibility.

3.3.1. VCS Usage

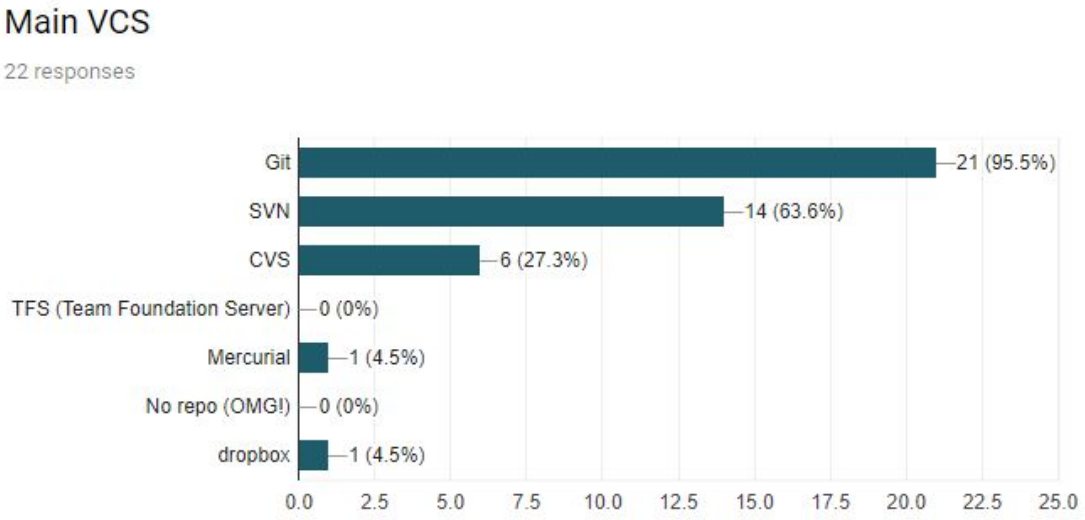
Version Control Software is an important aspect of the reproducibility workflow. For this

reason, in the beginning of the questionnaires we included a series of questions aiming to get insights on the VCS usage among the scientific community. We found out that almost 45% of our participants does **not** use VCS.



This is quite a disconcerting fact, since without having the ability to pull previously stored data from a repository, it will be almost impossible to extract information from a centralized location and attempt to make it reproducible. We look forward on checking these statistics again, once we have collected more responses from our questionnaire.

From the users that do use a VCS, almost everyone uses Git [95,2%] but there is good familiarity with SVN as well.



3.3.2. Coding experience

As far as coding experience is concerned, we found out that Python is very popular among the scientific community, with Fortran and IDL following.

3.3.3. Operating systems

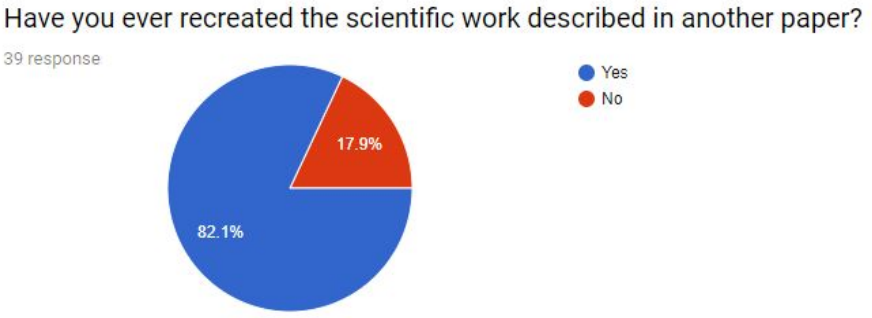
Most of the users are Linux & MacOS users, and the majority of them [81.6%] have root access on their machines, allowing them to easily install any tools they might require, without having to go through the hassle or the paperwork required to have it installed by an IT department.

A large number of them [68.4%] where familiar with the use of virtualization machines (VM) software, mainly VirtualBox and VMWare. A smaller percentage of them [36.8%] are familiar with containers technology, mainly Docker and Singularity.

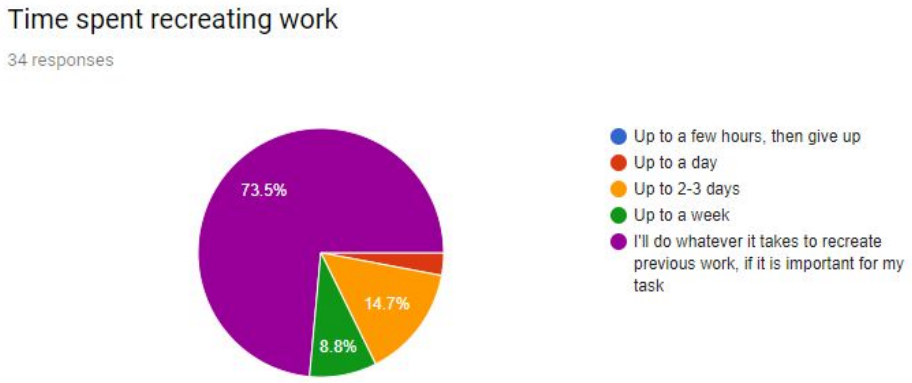
3.3.4. Recreated Science

Studying the answers we received regarding Reproducibility in Science and the need to recreate the scientific work that is described in other papers, we found out that 81% of the scientists answering our questionnaire have indeed recreated scientific work done from others in the past. The main reasons for doing so, were to:

- extend the work performed,
- because the methods used looked interesting, and
- because some of the published results seemed suspicious.



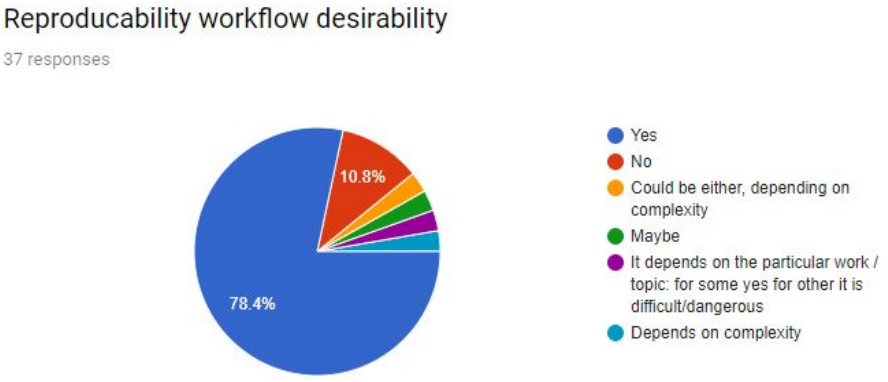
Continuing on the analysis of our findings, [72.7%] of the scientists answered that they will do whatever it takes to recreate previous work, if it is important for their task.



[77.8%] would like to have access to an automated reproducibility workflow but only [47.2%] has actively seeked for one and only [45.5%] have actually found some way to make their work reproducible.

Only 1 person [3,7%] is consistently using a reproducibility workflow tool.

Concluding, more the three quarters of our participants [78.4%], stated that they would desire to have a reproducibility workflow in place.



4. Proposed Course of Action

4.1. Introduction

Based in the overview of the existing Reproducibility services and tools, as we evaluated them, we concluded that the existing services are not able on their own to offer a convenient and unobtrusive way for scientists to make their work easily reproducible.

Our proposal is to introduce a tool that allows the users to selectively choose an existing service of their liking and have the tool perform the manual and repetitive tasks of bringing everything together.

4.2. Proposed Solution

We propose to implement a command line tool, that will be available on all major Operating Systems (Linux, MacOs and Windows). It will be accompanied by a companion website that will act as an online guide, a frequent asked questions section, a way for users to provide feedback, and as a hub where all development on it will take place.

4.2.1. Basic Functionality

The command line tool will offer three fully automated major functionalities:

1. Allow the automated collection of required input data files from various online sources
2. Allow the execution of any required computational tasks that will operate in the input data files
3. Allow the dissemination of the output results.

The tool should be used by the original author during the process of their scientific work, but it could also be used by any other users that would like to recreate the same scientific work.

The tool will work from a text based configuration file that will define all requirements necessary to make the task of reproducing the majority of scientific workflows as automated as possible. The original author will define three major pieces of information:

1. List of input files to be collected
2. Define an executable environment where the produced code would execute in
3. List of output files to be published

It will be developed in an extensible way, so that at release time, only a few selected existing third party services would be supported, but, upon demand, more can be later on added. This will also future proof the core functionalities offered by the tool, by being able to be

extended to support other services that might not be available today, but can be really popular or useful in the future

It should offer clear, concise and user friendly error messages for the cases where the automation process cannot continue unassisted. In the majority of the cases, a solution or additional help should be already available in the companion website. For the missing cases, the companion website should be frequently edited to include updated information for new edge cases.

4.2.2. Reproducibility File

The driving force behind the functionality offered by the Reproducibility tool, would be the `reproducible.yml` file. This will be a plain text file that will encapsulate all required information for making a scientific paper reproducible.

The final format of the file structure has not been finalized yet, and this is something will be done in further detail during the implementation phase. A sample of the current proposed implementation (and something that will be definitely augmented in the future) is presented below, along with some annotations on the most important parts:

reproducible.yml:

```yaml
version: '0.1'  ❶

input:  ❷
 service: zenodo  ❸
 - id: ris-12345  ❹
 - id: ris-99999:/lfi/some_file.fits

code:  ❺
 engine: docker  ❻
 scm: git  ❼
 repo: https://gitlab.com/BeyondPlanck/repo.git
 branch: master
 directory: WP9/reproducibility/usecase

output:  ❽
 service: zenodo  ❾
 user: my_zenodo_username
 files:  ❿
   - output/planets_volume.csv
```

```
    - output/
```

Code 1: Sample `reproducible.yml` file defining all required info for reproducing a paper

Some notes on the contents of the configuration file:

❶: The file format version for the reproducible file. This will allow us to know the available options supported by the configuration file
❷: Input files, the first phase in the reproducibility phase, is a section that defines all the required files that must be locally present, before the execution process can begin executing.
❸: A subsection that defines where the required input files are to be found. This is part of the extensibility of the Reproducible Tool, allowing for more services to be added
❹: IDs are used to identify input files that need to be download. The format can be as complex as required to download a single file, or as in the second ID in the example, point to a specific file inside an archive full of various files.
❺: The code section, the second phase of the the reproducibility flow, describes specific information regarding the process of executing the provided code
❻: Specifies the execution environment to be used. Another extensible part of the tool. Initially Docker supported, but other candidates can also include Singularity, etc.
❼: Source code management specifies what kind of repository (git, svn, etc) the required code is stored at. Further details, specific to each scm, follows, regarding branches, directories etc
❽: The output section, the third phase in the reproducibility flow, tracks info regarding where the produced files should be uploaded, as their final resting place.
❾: Once again, a list of uploading services will be available, with the potential to add more, as more services become available/popular
❿: Detail section where specifics about which of the output files are supposed to be the official ones that have to be uploaded/archived.

4.3. Example Use Case Scenario

Let us assume that a user is writing a hypothetical paper where they use the Planck component maps (as they are published in the official PLA archives) to investigate the statistical properties of the residual signal around the galactic center. For this, they start with the Planck full-frequency sky maps at various frequencies. Then, they extrapolate the various diffuse component signals based on the official Planck component maps, using the official Planck bandpasses, and subtracted the resulting signal from the corresponding sky

maps. Finally, they present the statistical properties of the residual signal in the form of plots and data tables.

For this paper the author wants to make the whole process completely reproducible and they decide to use the Reproducibility tool we intend to develop. This section describes the process that they will have to follow to achieve this.

4.3.1. Define the Required Input files

Firstly, the author will have to clearly define the required input files needed for their paper. In this example let us assume that the files are the Planck component maps A and B from the following locations (the links are hypothetical).

The input section in the `reproducible.yml` will then be filled, thus:

reproducible.yml (snippet):

```
input:
 service: http
 - id: http://link.to.planck.archives/components/map-A.fits
 - id: http://link.to.planck.archives/components/map-B.fits
```

Code 2: Snippet of the input section of the usecase `reproducible.yml` file

4.3.2. Define the Computational Tasks

As far as the computational phase of the reproducible workflow, the author will have to define an execution workflow.

In our use case scenario, the author chooses to host all required code on Gitlab, in a publicly available repository. The repository will have to be publicly accessible, only once the author decides to make the work publicly accessible. It might remain a private repo, while the work is being performed, and accessible only to users with the right credentials, since it might be awkward sometimes to release work in progress or different versions of experimental work. But for the final work, to be really reproducible by everyone, by publication time all work will have to be publicly accessible by everyone.

Assuming that the work will be done with Python, the author defines a Docker executable environment, in a `Dockerfile` text file, that uses an official Docker Python base image, and they also install a list of python packages that are required for their computational needs.

Dockerfile:

```
FROM python:3

# Install more dependencies here (as needed)
RUN pip install numpy pandas healpix astropy
```

Code 3: Sample Dockerfile defining an executable environment

Now they define a runtime environment that defines where the input files, their code base, and the output files would be (defined as `/data/input`, `/data/output` and `/code` in our example) and what is the execution startup script

docker-compose.yml:

```
version: '3.6'
services:

 app:
   build: .
   image: author_name/my-project-123
   volumes:
     - ./input:/data/input
     - ./output:/data/output
     - ./code:/code
   working_dir: /code
   command: ["python", "start.py"]
```

Code 4: Sample `docker-compose.yml` defining a runtime environment

With these configuration files, a simple command line command of:

```
$ docker-compose up
```

will be enough for compiling their execution environment and bring up the Docker container, where their Python scripts will automatically start executing. Once their code successfully completes, their output files will be conveniently placed in the /output folder (by convention, the executing code has the responsibility of storing all output code in the `/output` folder). At this point they are ready to proceed to the final phase of the reproducibility workflow, publishing their results.

4.3.3. Define the Publishing of Results

Once all the output files have been produced they can then be uploaded to a hosting service for their final release.

They could be reuploaded in the same hosting service where the execution code was residing, but a better solution would be to utilize another service that provides unique DOI IDs, like Zenodo.

This can be easily provided by a section in the `reproducibility.yml` file as in:

reproducible.yml (snippet):

```
output:
 service: zenodo
 user: author_name

 files:
   - output/diagram1.png
   - output/diagram2.png
   - output/tableA.csv
```

Code 5: Snippet of the output section of the use case `reproducible.yml` file

The files then will be picked up by the reproducibility tool, uploaded to the service and assigned a unique DOI ID, making them publicly available for any interested parties.

4.4. Synopsis

We understand that it will be impossible to satisfy the tastes and/or needs of all scientists, but we strongly believe that by introducing a tool that combines multiple services in a self contained and automated way, will make it more adaptable from our intended audience. Being a universal command line tool that will be able to utilize and work with multiple existing services, will allow our audience to use the existing online services that makes most sense to them, without inconveniencing and forcing on them yet another one.

Also, we understand that the provided reproducibility workflow might not be applicable in certain edge case scenarios, due to their complexity. Special cases with either exceptional executable phases (requiring really big amounts of execution resources, more applicable to a computing grid) or in their usage of input or output files (for example, big datasets that are really prohibitive in downloading and storing locally) might not be candidates for this specific reproducibility workflow. It is not impossible to devise such an extension to the proposed Reproducibility workflow, capable of handling these complicated edge cases, but this would be something that could be expanded on, in a further iteration of the tool.

Due to the inherent complexity of supporting any and all kinds of possible scientific workflows, we purposefully chose to cater to a specific subset of scientific workflows. We chose to provide a tool that provides an automated way of reproducing the scientific work produced in a paper, while the scope of its capabilities have been decided so that the development of the tool would fit in the allocated time of this project.

The criteria by which a project would benefit from this proposed workflow are:

- Have an exact definition of initial input files. The author should be able to define what input files are required and where these can be found.
- Have a process that programmatically manipulates these input files (along with any other files that are generated for the purposes of this paper)
- Does not have excessive computational requirements for the execution of the above programmatic code, preferably able to be executed in the modern laptop or even a high end desktop machine
- Similarly it should have modest file system requirements, able to be accommodated, once again, from a laptop or high end desktop machine.
- Produces some results (files, diagrams, tables etc) that is the output of the computational phase.

Although these might appear as very limiting criteria, we believe that they will be able to cover a large number of scientific publications. In addition, to the best of our ability we believe that a fully automated reproducibility workflow such as this, has never been attempted before.

The tool will also be accompanied with a companion site where a detailed description of the workflow and usage hints will also be published. The companion site will also include a section for collecting user feedback, offering an additional layer of assistance in addition to the existing help topics.

In conclusion, our goal is to develop a reproducibility workflow and accompanying helper tool, that will be able to provide:

- A frictionless way for original authors to document and set up an automated reproducible workflow,
- An even more easier way, for any other interested parties to locally recreate the same project and expect to get the same results as the original author.

5. GPU for PLANCK

In this section, we describe our hypothesis and investigation results about whether and where to introduce a GP-GPU acceleration stage in the processing pipeline of Planck, based on our recently acquired knowledge of the project and the related objectives, computations and issues. Moreover, we firstly provide a quick overview of GPU technologies and capabilities and then we focus on the pipeline and its components.

5.1. General Purpose GPU

GPU devices are on the market since the nineties, but NVIDIA revealed the first programmable device in 2001. Since then, GPUs (by NVIDIA, AMD, Intel and many other manufacturers) acquired many new capabilities and processing speed. Designers and developers discovered their potential for not only producing graphics on a display, but also for performing general computations by leveraging the internal parallelism of these chips. In fact, a GPU contains hundreds or thousands of cores with SIMD (Single Instruction Multiple Data) capabilities for each core.

Modern HPC GPUs such as the NVIDIA Tesla V100 are ready for datacenter integration and already used by scientists and engineers. For example, 1GPU node replaces up to 54 CPU nodes, as reported on https://www.nvidia.com/it-it/data-center/tesla-v100/.

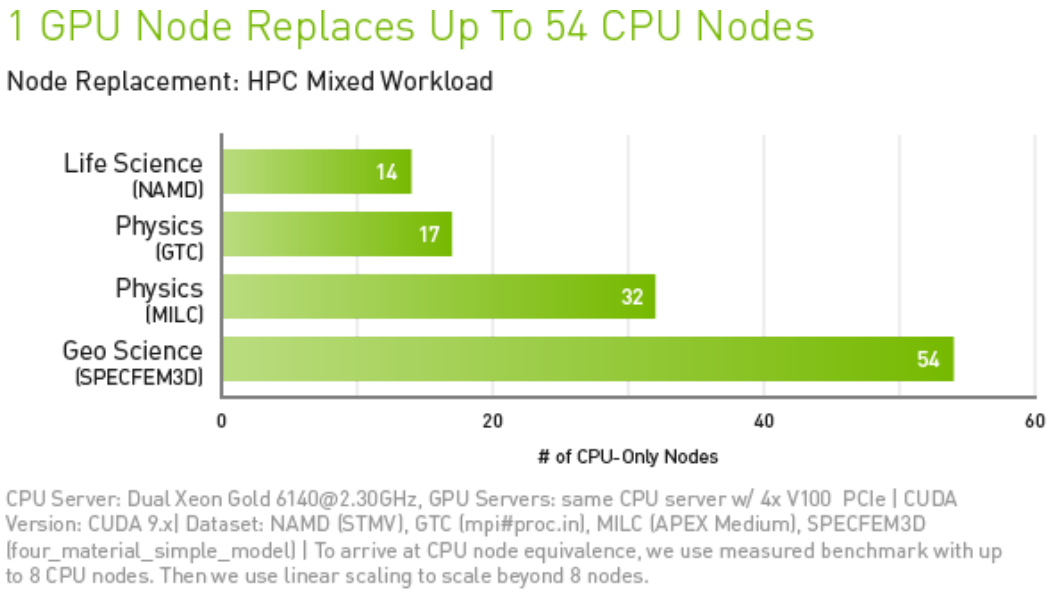


Figure 1 - Source: NVIDIA (www.nvidia.com)

But, how can we program GPUs ? Many APIs exist today for developing on heterogeneous devices. For example, Khronos Group standardized an API for generic computing, named OpenCL, which is pervasive and supported by all GPU vendors in the market, including Intel and AMD (for details, see https://www.khronos.org/opencl/). NVIDIA provides its own

proprietary and very powerful CUDA library that runs only on NVIDIA GPUs, enriched by many 3rd-party libraries (for details, https://developer.nvidia.com/cuda-zone). Microsoft provides its DIRECTCOMPUTE (together with its well known DirectX) infrastructure for Windows OS only.

Each API has its own programming and usage paradigm, but the concept behind GPU programming is always the same: the CPU hosts a program responsible of communicating with the GPU to exchange inputs, outputs and kernels. Kernels are programs that run on the GPU. Their source code consists mainly in a C dialect and they always execute in parallel as if they were independent processes on a multicore CPU.

One of the main advantage in using a GPU for offloading the CPU is that both CPU and GPU execute in parallel, of course being the GPU typically the fastest device of the two. Moreover, most of the math functions in a GPU exist as hardware implementations and operate on vector data, in a SIMD fashion.

5.2. The Pipeline, OWL and OpenMPI

Figure 2 shows the Planck Pipeline as reported in the document “Planck 2018 results. II. Low Frequency Instrument data processing”, September 12, 2018. It is quite complex and component rich.

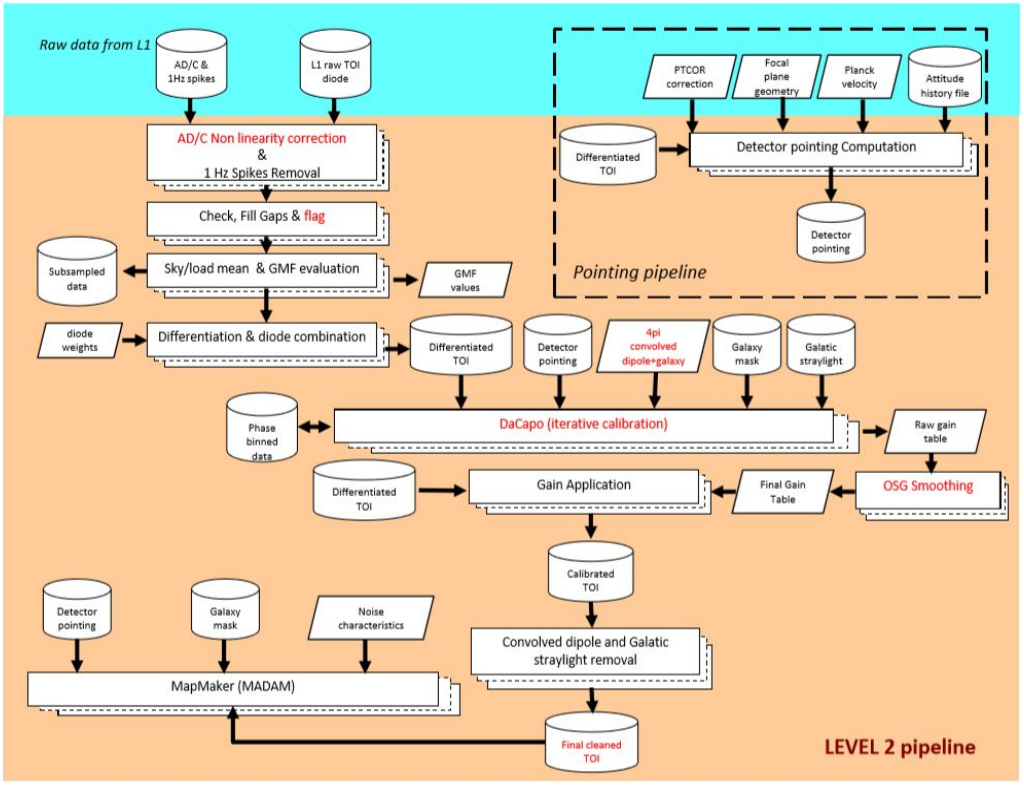


Figure 2 - The PLANCK Pipeline.

As emerged by examining the actual code base in GitLab and by analyzing related public

scientific articles, most (if not all) of the components in the pipeline requires an HPC infrastructure to run on. We noticed that the preferred solution has been to write code compliant with the MPI (Message Passing Interface) specification (in particular, for running on the OWL computing cluster, designers and developers used the OpenMPI programming library and the Intel MKL math acceleration library for BLAS functionalities).

The *ompi_info* command, executed on e.g. node owl26 of the cluster, shows many useful information about the actual OpenMPI implementation running on that node. We noticed that the implementation supports the following MPI extensions: *affinity* and *cuda*. As briefly explained above, CUDA is a technology for implementing algorithms that run on GPUs. Anyway, no GPGPU device is installed on that node, but the actual MPI API seems to support for it.

By inspecting the remaining OWL's nodes, we finally determined that the cluster does not host any GPU device, but interesting boards were available, such as the Mellanox ConnectX-3 Pro and Connect-IB. In particular, the latter board is capable of further accelerating inter-host GPU communications, to boost MPI performances, when involving GPU computing.

In this context, it is worth noting that OpenMPI supports integration with GPU through the NVIDIA CUDA infrastructure, since v1.7.0. OpenMPI 1.8.x provides improved support and performances for this type of technology. Best performances and stability are available in version 1.10.1, as reported in https://www.open-mpi.org/faq/?category=runcuda.

5.3. Storage Requirements vs GPU Workflow

We noticed that the project has quite an impressive amount of data to manipulate. This may be a real issue when dealing with CPU/GPU data transfer; in fact, the controlling applications that run on CPUs shall deal with continuous transfer of data to/from GPU.

As reported in "*3D and 4D map data objects*" by Elina Keihänen, August 21, 2018, "*The typical size of a 4D map file [is] 8 GB (Nside=1024, Npsi=4096), for a file containing the full 4-year mission data. The full LFI data set takes 90 GB of disk space, and can easily be stored on a modern laptop.*"

Moreover, in the same document: "*[...]The full 4-year pointing takes 217 GB (70 GHz), 127 GB (44 GHz), or 92 GB (30 GHz) per horn. The full detector pointing takes 1870 GB of disk space. The signal TOI is stored as 4-byte floats, and adds 620 GB to the count[...].*"

Since GPUs have a limited amount of main memory available, algorithms need to take into account the minimum amount of data they need for running correctly. In general, a stream-oriented approach is required when porting algorithms from general purpose CPUs to discrete GPUs. One shall structure the data so that a process as the one shown in Figure 3 is possible.

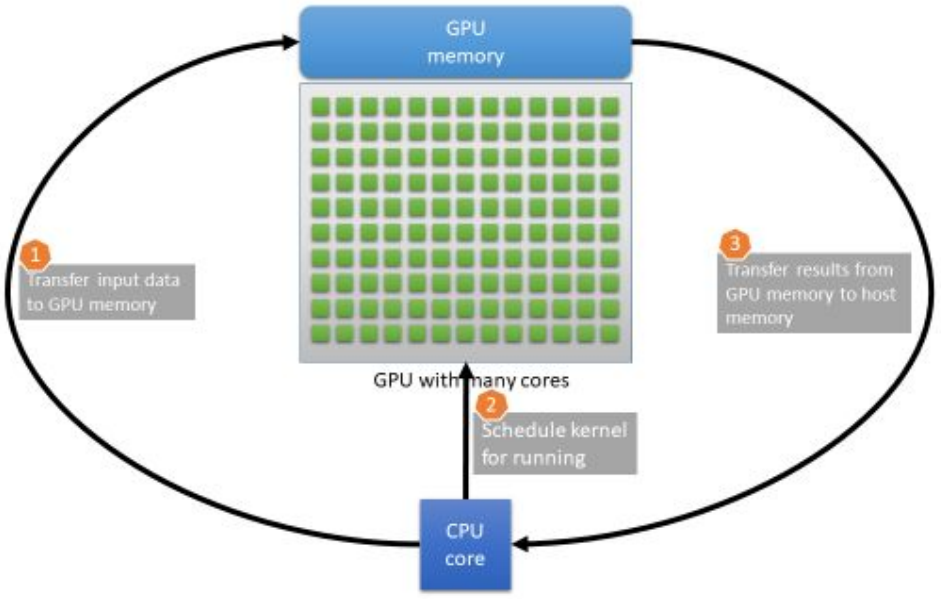


Figure 3 - Typical CPU/GPU transfer workflow.

The black arrows highlight data flow direction. One of the CPU cores (*CPU core*) hosts a program that interact with the GPU (it is worth noting that a system can host more than one GPU, on different slots, to further boost computational power). The program works in a loop and executes mainly three steps:

1. It first sends any input data to the GPU main memory;

2. Then, it schedules a specific kernel so that it can run on the GPU;

3. Lastly, it asks for transferring output data from the GPU memory to the host memory.

Any application that deals with heterogeneous computing follows this program structure. This means that input and output data shall be in some way segmented to be compatible with the overall memory size in the GPU. Depending upon the minimum required memory footprint of the algorithm implementation, it is possible to choose the right device for processing: e.g. on board memories goes from 2GiB up to 32GiB of memory in the most advanced HPC ready GPUs.

5.4. GPU Usage Scenarios for PLANCK

The project, its data and its processing pipeline are all extremely complex and rich of interacting people and components. The fact that most of the software applications uses OpenMPI on a very powerful cluster (OWL) make us consider any overall GPU optimization useless/unworthy as a direct pipeline component.

However, by analyzing each single component in the pipeline, we perceived some interest by the involved people and space for further parallelizing/improving the overall single

component execution time. For example:

1. The application of satellite attitude data (represented by a triplet of angles) to the input data stream, in one of the preliminary steps of the pipeline, seems quite ideal for GPU intervention, thanks to device intrinsic capability of applying trigonometric function and per-sample processing in real-time.
2. In the map-making process, as reported in "*Making CMB temperature and polarization maps with Madam*", by E. Keihänen et al., Feb.21 2013, the overall memory footprint of the algorithm makes direct GPU usage impossible, unless a "*Split-mode*" with "*Baseline 61*" (memory usage of 5.7GiB) or "*Baseline 8*" (memory usage of 16.7GiB) is used. In this case efforts to port the algorithm to run on high end GPUs could be evaluated in strong coordination with MADAM owners and designers. (In both cases, we shall take into consideration GPUs with 8Gib, 16GiB or 32GiB, to match the aforementioned memory requirements).
3. As a general rule, components that execute per-pixel operations could benefit from execution of programs on GPU since they run in parallel on all pixels of the image.