sid.inpe.br/mtc-m21c/2020/04.20.16.10-TDI

STUDY AND OPTIMIZATION FOR HIGH PERFORMANCE PROCESSING WITH GALPHAT

Igor Kolesnikov

Master's Dissertation of the Graduate Course in Applied Computing, guided by Drs. Celso Luiz Mendes e Reinaldo Roberto Rosa, approved in April 23, 2020.

URL of the original document:
<http://urlib.net/8JMKD3MGP3W34R/42C2DGH>

INPE
São José dos Campos
2020

PUBLISHED BY:

Instituto Nacional de Pesquisas Espaciais - INPE
Gabinete do Diretor (GBDIR)
Serviço de Informação e Documentação (SESID)
CEP 12.227-010
São José dos Campos - SP - Brasil
Tel.:(012) 3208-6923/7348
E-mail: pubtc@inpe.br

STUDY AND OPTIMIZATION FOR HIGH PERFORMANCE PROCESSING WITH GALPHAT

Igor Kolesnikov

Master's Dissertation of the Graduate Course in Applied Computing, guided by Drs. Celso Luiz Mendes e Reinaldo Roberto Rosa, approved in April 23, 2020.

INPE
São José dos Campos
2020

Cataloging in Publication Data

Kolesnikov, Igor.

K832e Study and optimization for high performance processing with galphat / Igor Kolesnikov. – São José dos Campos : INPE, 2020.

xxii + 65 p. ; (sid.inpe.br/mtc-m21c/2020/04.20.16.10-TDI)

Dissertation (Master in Applied Computing) – Instituto Nacional de Pesquisas Espaciais, São José dos Campos, 2020.

Guiding : Drs. Celso Luiz Mendes e Reinaldo Roberto Rosa.

1. Computational cosmology. 2. Elliptical galaxies. 3. Bayesian statistics. 4. Galaxies structure and environment. 5. High performance computing. I.Title.

CDU 004:523.1

MINISTÉRIO DA CIÊNCIA, TECNOLOGIA E INOVAÇÕES

INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

FOLHA DE APROVAÇÃO

A FOLHA DE APROVAÇÃO SERÁ INCLUIDA APÓS RESTABELECIMENTO DAS ATIVIDADES PRESENCIAIS.

Por conta da Pandemia do COVID-19, as defesas de Teses e Dissertações são realizadas por vídeo conferência, o que vem acarretando um atraso no recebimento nas folhas de aprovação.

Este trabalho foi aprovado pela Banca e possui as declarações dos orientadores (confirmando as inclusões sugeridas pela Banca) e da Biblioteca (confirmando as correções de normalização).

Assim que a Biblioteca receber a Folha de aprovação assinada, esta folha será substituída.

Qualquer dúvida, entrar em contato pelo email: pubtc@inpe.br.

Divisão de Biblioteca (DIBIB).

ACKNOWLEDGEMENTS

Firstly I would like to thank the INPE and its graduate program for the opportunity and all necessary means to be able to learn and grow in the academic area. The teachers' core was extremely helpful both during the curriculum study and research process. Their helpful tips and bits of advice were a valuable addition to the project.

I want to express immense gratitude to Dr. Martin Weinberg from Massachusetts University, the author of the GALPHAT. His unparalleled help and assistance in technical questions concerning the usage and adaptation of GALPHAT were of great help to the whole research.

Besides Martin, I would like to thank Diego Stalder for innumerable conversations and questions answered to help me better understand the pipeline and routines of the processing.

I am grateful to Dr. Reinaldo de Carvalho for the helpful tips and pieces of advice, as well as for the valuable tips on developing of the project. I am thankful to my advisors, Dr. Celso Mendes and Dr. Reinaldo Rosa for the help with preparation of the presentation and the monograph, as well as for the help during the research.

I want to thank CAPES for the financial support that allowed me to conduct this research.

Last, but not least, I would like to express my gratitude to my girlfriend, Amanda Guimarães, who helped a lot in reviewing and providing valuable advice on the monograph, even though she is a professional from the law area and had a hard time understanding the content of the research.

ABSTRACT

The parametric computational modeling of galaxies is a process with a high computational cost. The statistical component of modeling, which may involve model refinements in relation to the source brightness distribution achieves more satisfactory results when the approach is Bayesian. In this research, we are using GALaxy PHotometric ATtributes (GALPHAT) as our main tool for data processing.

The GALPHAT modeling of a galaxy observed by the Sloan Digital Sky Survey (SDSS) can last about 6 hours. In the current scenario of cosmology, this type of modeling, to be scientifically effective, must be performed on a set containing about thousands of objects. The sample analyzed within the scope of the FAPESP thematic project that LABAC participates contains more than 24,309 objects, an amount that demands the use of high-performance computing (HPC) to enable effective modeling of the entire sample.

In this postgraduate project, we have as the main objective to study and optimize HPC solutions that allow GALPHAT processing on a SDSS sample in the fastest possible way. For this, we have two HPC systems that can work in a coordinated way to optimize the modeling strategies. The first system belongs to LABAC and is based on Intel Xeon Phi 7250 platform. The second system belongs to the partition of the multi-core platform of the Santos Dumont supercomputer.

The research, therefore, includes the initial process done to set up and run GALPHAT on both platforms, thus using different types of processors and compilers. Considering the different processing steps, in different modeling strategies we applied refactoring and complete modules rewriting. Our studies have found the optimal combination of software, hardware and optimizations to minimize processing time. This is the first step in implementing and integrating the graphical user interface to make GALPHAT easier to use. This dissertation, therefore, presents all of the activities that were performed to allow, as a final result, to process, in a timely manner, via HPC, the entire selected sample including the description of benchmark among the computational systems used. It includes the development of the auxiliary visualization system as well.

Keywords: Computational Cosmology. Elliptical Galaxies. Bayesian Statistics. Galaxies Structure and Environment. High Performance Computing.

ESTUDO E OTIMIZAÇÃO PARA PROCESSAMENTO DE ALTO DESEMPENHO COM GALPHAT

RESUMO

A modelagem computacional paramétrica de galáxias é um processo com alto custo computacional. O componente estatístico da modelagem, que pode envolver refinamentos do modelo em relação à distribuição do brilho da fonte, obtém resultados mais satisfatórios quando a abordagem é bayesiana. Nesta pesquisa, estamos usando o GALaxy PHotome-tric ATtributes (GALPHAT) como nossa principal ferramenta para processamento de dados.

A modelagem usando o GALPHAT de uma galáxia observada pelo Sloan Digital Sky Survey (SDSS) pode durar cerca de 6 horas. No cenário atual da cosmologia, esse tipo de modelagem, para ser cientificamente eficaz, deve ser realizado em um conjunto contendo milhares de objetos. A amostra analisada dassa pesquisa, que faz parte de projeto temático da FAPESP da qual o LABAC participa contém mais de 24.309 objetos, quantidade que exige o uso do processamento de alto desempenho (PAD) para permitir a modelagem eficaz de toda a amostra.

Neste projeto de pós-graduação, temos como principal objetivo estudar e otimizar soluções de PAD que permitem o processamento com GALPHAT em uma amostra de SDSS da maneira mais rápida possível. Para isso, temos dois sistemas PAD que podem funcionar de maneira coordenada para otimizar as estratégias de modelagem. O primeiro sistema pertence ao LABAC e é baseado na plataforma Intel Xeon Phi 7250. O segundo sistema pertence à partição da plataforma multinúcleo do supercomputador Santos Dumont.

A pesquisa, portanto, inclui o processo inicial feito para configurar e executar o GALPHAT nas duas plataformas, usando diferentes tipos de processadores e compiladores. Considerando as diferentes etapas de processamento, nas diferentes estratégias de modelagem, aplicamos a refatoração e a reescrita completa dos módulos de pipeline. Nossos estudos descobriram a combinação ideal de software, hardware e otimizações para minimizar o tempo de processamento. Este é o primeiro passo na implementação e integração da interface gráfica do usuário para facilitar o uso do GALPHAT. Esta dissertação, portanto, apresenta todas as atividades realizadas para permitir, como resultado final, processar em tempo hábil, via PAD, toda a amostra selecionada, incluindo a descrição de uma referência entre os sistemas computacionais utilizados. Inclui também o desenvolvimento do sistema de visualização auxiliar.

Palavras-chave: Cosmologia Computacional. Galáxias Elípticas. Estatística Bayesiana. Galáxias Estrutura e Ambiente. Computação de Alta Performance.

LIST OF FIGURES

LIST OF TABLES

Page

LIST OF ABBREVIATIONS

ICC – Intel C compiler
GCC – GNU C compiler
GALPHAT – GALaxy PHotometric ATtributes
SDSS – Sloan Digital Sky Survey
HPC – High-performance computing
GUI – Graphical user interface
JSON – JavaScript Object Notation
CPU – Central Processing Unit
RAM – Random Access Memory (main memory)
ETG – Early Type Galaxy
MPI – Message Passing Interface
POSIX – Portable Operating System Interface
BIE – Bayesian Inference Engine
MCMC – Markov chain Monte Carlo
CLI – Command Line Interface
FITS – Flexible Image Transport System
PyPiGALPHAT – Python Pipelining GALPHAT

CONTENTS

Page

1 INTRODUCTION

The parametric computational modeling of galaxies is still a process with a high computational cost and lasting years of CPU time when several thousands of galaxies have to be processed and specially when the Bayesian approach is used. In the current scenario, this type of modeling can be performed only on a limited set of galaxies, hampering more ambitious and complete astrophysical analysis.

Four years ago, our group embarked on a collaboration with Dr. Martin Weinberg from UMASS (University of Massachusetts Amherst), who developed GALPHAT (GALaxy PHotometric ATtributes)[1], a Bayesian program for galaxy photometry. In 2017 a Ph.D. thesis was presented at LAC/INPE showing further developments of this program. Special attention has been given to the modeling of Early-Type Galaxies (ETGs)(STALDER,).

In recent years, since the characterization of galactic components such as bulge, disk, central point sources, and others of smaller mass contribution has been shown an essential diagnosis of physical processes that determine their evolution. In particular, their dependence on the environment where the galaxy resides.

In this research our focus is to extend even further the line of the previous study, processing more than 20 times more of the data than the preceding research to perform the first part of the search on which the model fits better our dataset. To achieve it, we need to process the same data with two models. In our case, these models are «Sérsic» and a much more complex and sophisticated «Sérsic + Exponential». While the part of the pipeline that is responsible for the processing the «Sérsic» model is fully implemented and functional, the part responsible for the «Sérsic+Exponential» is far from being ready, in relation to the code and the testing of the functionality. Therefore, given the context of the massive amount of data and the possibility to use HPC hardware and software, we aim to conduct an in-depth study and optimization of GALPHAT and its components to achieve a viable time while processing on different HPC solutions. At the same time, we started the development of an interface visualise and analyse the results. Our aim, in the near future develop it in a platform where the community will access the products of this processing — having all necessary information without the need of processing data during months/years. We hope to unify and improve the tools from various previous studies and integrate them with visualization capabilities for the

[1]`https://bitbucket.org/mdweinberg/bie.git`

end-user.

1.1 Conceptual context and scientific background

In astronomy, every year, we witness the appearance of new and more sophisticated instruments, telescopes and other observational tools that can gather much more data than previous ones. These data need to be stored and processed for later research. To perform the cycle of «test-result-conclusion» faster, one needs to process and analyze data in a more timely manner. Not only, but the results also should represent the full exploitation of the data. To make the situation of data accumulation worse, the current speed of new galactic data generation is much faster than the growth of the speed of its processing. So, every day, scientists struggle to figure out how to get more meaningful results in less time. Methods based on Bayesian Inference may be part of the solution, given the fact that Bayesian Analysis provides the full posterior distribution of each parameter as an output. However, these tools are very computationally costly in operation. So, to stay competitive in the world of always growing requirements, one needs to look for new and creative solutions with available hardware and software resources, while paying each time more attention to optimization and price/benefit of the tools available.

Establishing nearly 14 billion years of evolution from observing galaxies as they are observed today is a task fraught with danger. In this project, we propose an approach that extends galaxy evolution studies to infrared wavelengths and higher redshifts (further in the past) to consistently investigate the properties of galaxies and their environments over a significant cosmic time interval. A crucial step towards this will be the implementation, optimization and augmentation of the GALPHAT galaxy photometry program pipeline, which will provide a solid statistical basis for the analysis of the structural parameters of the components observed in galaxies (mainly the bulge and the disk).

1.2 Motivation

Galaxies form through the gravitational collapse of density perturbations, produced during the inflation era. The cosmological model of the Big Bang with a cosmological constant, known as the ΛCDM model, explains reasonably well quite important observations as the temperature variations of the cosmic background radiation, the distribution of large-scale galaxies, nucleosynthesis and the acceleration of universal expansion (CONSELICE, 2014). An inventory of mass density in the Universe shows that $\sim 26.8\%$ is in the form of dark matter and dominates the gravitational field on a large scale; the cosmological constant is responsible for $\sim$ 68.3% and determines the accelerated expansion of the Universe; and the remaining $\sim$ 4.9% constitutes the ordinary matter that forms the planets, stars, and galaxies (AGHANIM et al., 2018; ABBOTT et al., 2019).

Much of the astrophysical complexity lies precisely in the component of the ordinary matter, and most observations are based on this baryonic component. How does a galaxy form its stars? What determines the total star content of a galaxy? Astronomers have devoted considerable effort to answer such questions in recent decades. Stellar formation starting from a cold gas cloud is an extremely complex problem and one of the most difficult of modern astrophysics. The problem itself becomes more complicated by the fact that many galaxies reside in larger structures, such as groups and clusters of galaxies, where they interact with neighboring systems. Thus, it becomes imperative to understand separately how internal processes are distinguished from external processes that determine the stellar content of a given galaxy in a given environment and for a given redshift. Which processes are dominant? Barbera et al. (2010) studying a sample of 40000 ETGs defined in SPIDER (Spheroids Panchromatic Investigation in Different Environmental Regions), found a relation between the slope of the IMF (Initial Mass Function) and the velocity dispersion of the ETGs, showing that there is an excess of low mass stars in high mass elliptical galaxies. Independent dynamic studies reinforce this result (CAPPELLARI et al., 2009). In addition to the complexity involved in characterizing the population of these galaxies, environmental effects are present and, in some cases, are dominant (LAUER et al., 2012). Thus, it is of fundamental importance to investigate the properties of ETGs (including stellar population) as a function of the distance to the center of the nearest group/cluster. These properties may vary depending on the redshift and mass of the galaxy.

The environment manifests itself through three main physical processes: 1) ram pressure, where a galaxy that is moving through the hot gas of a cluster of galaxies loses its interstellar medium; 2) «harassment», which is the collective effect of several high-speed encounters of a galaxy with many others in the cluster and its overall potential; and 3) «starvation», which refers to the fact that the lifetime of a galaxy is linked to the availability of gas for a star formation episode. If stars form at a constant rate, with the value currently observed, then all the gas in a spiral galaxy would be consumed in less than a Hubble time. Thus, the lifetime of a galaxy increases if more gas is added to the system more recently. A corollary of this idea is that a spiral galaxy suffers from starvation when it loses its gas reservoir and becomes a lenticular galaxy (CONSELICE, 2014).

These three mechanisms operate in such a way that the transmutation of a galaxy from one morphological type to another may depend on the mass of the galaxy, the mass of the cluster where the galaxy resides (and the gas distribution), and its distance from the center of the galaxy cluster. The efficiency of these regulatory mechanisms depends on the ability of the local environment to remove interstellar gas. Besides, the evolution of the stellar population of a galaxy also depends on the redshift. In order to better understand how galaxies evolve, it is fundamental to define a sample of galactic systems present in the most varied environments, from low galactic density and low-velocity dispersion (field population) to systems with high density and velocity dispersion (groups/clusters).

In summary, the results of this research will improve our understanding of these effects by bringing an analysis done with Bayesian Inference tools, whereas previously only frequentist tools were used. However, to make it happen in a timely fashion and be presented appropriately, we need not only to improve the already existing pipeline for processing but also extend it to process different image modeling. Also, it is of paramount importance in this project the development of a specific database for the organization of all the information generated by GALPHAT. A user-friendly interface is being planned and developed as an integral part of this project so that the community will be able to benefit from our work.

1.2.1 Bayes Factor and model comparison foundation

The key objective and the primary motivation of processing the data with two models is the fact that we distinguish which is the best modeling through the Bayes Factor analysis (KASS; RAFTERY, 1995). Following, we will study the main scale relationships for these systems. Thus, the result, being a compilation and analysis of the massive amount of early-type galaxies that were processed using the Bayesian Inference tool is of considerable impact. This approach brings a good amount of new information and insights that were unavailable to researchers who used frequentist approach on the same data. This shows the potentiality of Bayesian inference and the possibility of extending this strategy to a set of several models. We can combine all observed components in elliptical and spiral galaxies, such as bulge, disc, point source, bar, lens, rings and similar.

Often it is difficult to know which theoretical model explains better a given galaxy light profile. This is crucial when different analytical forms describe physically distinct components such as bulges (Sérsic law), discs (exponential) or unresolved sources associated with active galactic nuclei (point source). The Bayes Factor (BF) provides a mechanism that evaluates the evidence in favor of each considered model rather than only test the goodness of the fit. BFs are the preferred method for model selection, for this reason, the best model will be given by the Bayes Factor (WAKEFIELD, 2013), which measures the relative statistical significance of the various probability distributions, and so the choice of the best physical representation is not made purely based on a Chi-squared statistics test, also known as X^2.

Stalder () shows that BF can be derived from applying the Bayes Theorem to a set of models. The first thing to do is to compute the evidences or marginalizations $P(D|M_i)$ by marginalizing the equation 1.1:

$$P\left(D|M_i\right) = \int d\theta \, P\left(\theta|M_i\right) P\left(D|\theta, M_i\right) \tag{1.1}$$

This quantity is also known as marginal likelihood. Once we have a sample of the posterior distribution obtained by the Markov Chain Monte Carlo (MCMC) algorithm, the computation of the evidence is a numerical challenge. To select which model explains better the surface brightness distribution of a given galaxy image, one can consider individual analytic expressions or combinations of them: a single «Sérsic» fit (M1) and/or «Sérsic+Exponential» (M2). To determine which of these two models is a better representation of an observed brightness distribution, we have to calculate the posterior odds:

$$\frac{P\left(M_1|D\right)}{P\left(M_2|D\right)} = \frac{P\left(D|M_1\right)}{P\left(D|M_2\right)} \frac{P\left(M_1\right)}{P\left(M_2\right)} \tag{1.2}$$

where the ratio $P\left(D|M_1\right)/P\left(D|M_2\right)$ is called Bayes Factor, given by:

$$BF_{12} \equiv \frac{P\left(M_1|D\right)}{P\left(M_2|D\right)} = \left(\frac{P\left(M_1\right)}{P\left(M_2\right)}\right)\left(\frac{P\left(D|M_1\right)}{P\left(D|M_2\right)}\right) \tag{1.3}$$

This ratio assesses the plausibility of the two different models $M1$ and $M2$. If $BF_{12} = 1$, both models are equally supported by the data. However, if $BF_{12} > 1$ or $BF_{12} < 1$, the data is in favor of model $M1$ or $M2$ respectively. Jeffreys (1998) suggest to scale BF_{12} in half-unit steps in log BF_{12} before interpreting the result.

Table 1.1 - Harold Jeffreys interpretation for the BF.

$\log BF_{12}$	BF_{12}	Strength of evidence
< 0	< 1	Negative (supports M_2)
0 to 1/2	1 to 3.2	Barely worth mentioning
1/2 to 1	3.2 to 10	Positive
1 to 2	10 to 10	Strong
> 2	100	Very Strong

SOURCE: Jeffreys (1998).

The prerequisite to conduct this analysis is to have the data processed with two or more models using a tool based on Bayesian Inference (GALPHAT in our case). During this master's dissertation, we processed the data sample with a «Sérsic» model, the main obstacle to go further with processing the data with «Sérsic+Exponential», is the time and consequent computational challenges to process the data in a timely manner. We are describing this process here to point the direction in which the work will be developed after the conclusion of the current research.

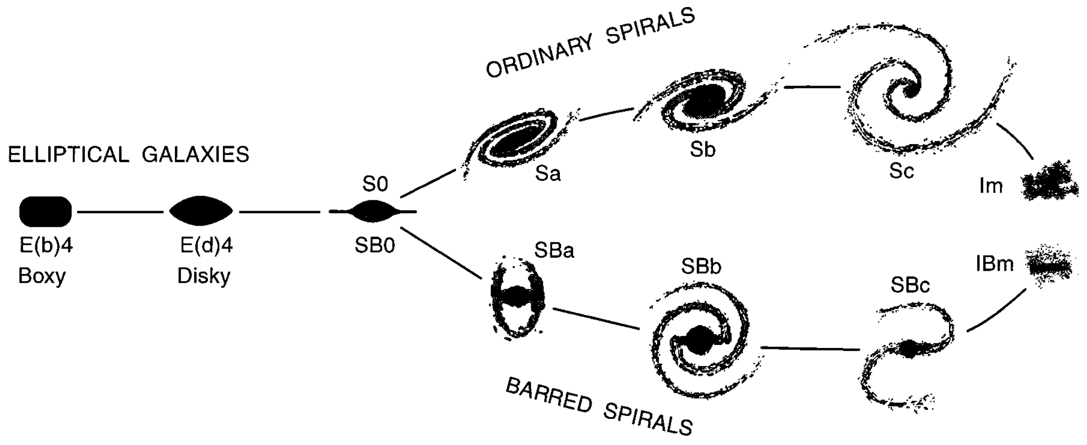
1.3 Data considerations

Galaxy analysis is fundamental to understand the evolution of the Universe. The first step in the galaxy analysis is to study the galaxy scheme. The Hubble Tuning Fork is one of the basic classification schemes, described by Figure 1.1. This image shows steps of the galaxy's formation and classification. We have to notice that there is a gradual change in these features indicating that it reflects physical proprieties (KORMENDY; BENDER, 1996).

Our input data are the elliptical galaxies that were photographed by the Sloan Digital Sky Survey[2] 2.5-meter wide-angle telescope that has been operational since 2000 (GUNN et al., 2006). This survey has photometric observations of more than 500 million objects, of which 3 million objects have spectrum data, making it one of the most detailed maps of Universe ever made.

[2]https://www.sdss.org/

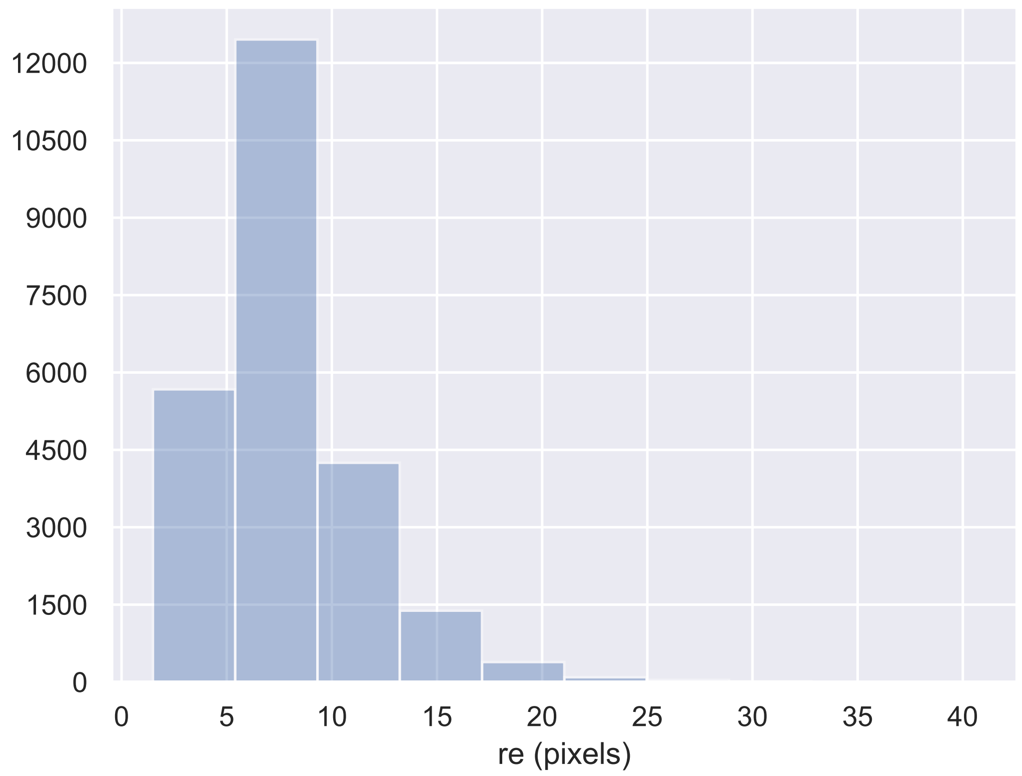
Figure 1.1 - Hubble Tunning Fork is showing the galactic form variation.



SOURCE: Kormendy e Bender (1996).

Our sample consists of 24309 ETG taken from Sloan Digital Sky Survey (SDSS) Data Release 7 (DR7)(ABAZAJIAN et al., 2009). This Legacy Survey is a catalog of the sky from a set of optical and infrared imaging data, comprising 14,000 deg^2 of extragalactic sky visible from the northern hemisphere in three optical bands (g, r, z) and four infrared bands. The total number of galaxies in this catalog is approximately 1.12 million. From this number, the actual SPIDER sample consists of 40000 galaxies. However, for our needs, firstly, tiny galaxies ($re < 2arcsec$) that would be very unlikely to converge during GALPHAT processing were cut off, so the smallest galaxies left in the list were not smaller than $re = 2arcsec$. Secondly, massive galaxies ($kpc > 100$) that naturally could not exist in nature also were cut off. Main reason to remove these galaxies was to minimize debugging and troubleshooting during processing, for this galaxies, being outliers, could introduce a good amount of complexity. Finally, the resulting list was ordered by according to the re parameter (smaller to larger).

Figure 1.2 - Distribution of sizes of images of galaxies in the sample that was used in this research. The x-axis is showing the half-light radius of the galaxies in pixels. The value in pixels is computed by dividing the De Vaucouleurs fit scale radius parameter from the SDSS database by 0.396, which is the scale of the pixel $(devRad_r/0.396)$. The conversion formula to pixels is $(devRad_r/0.396) * 15$.

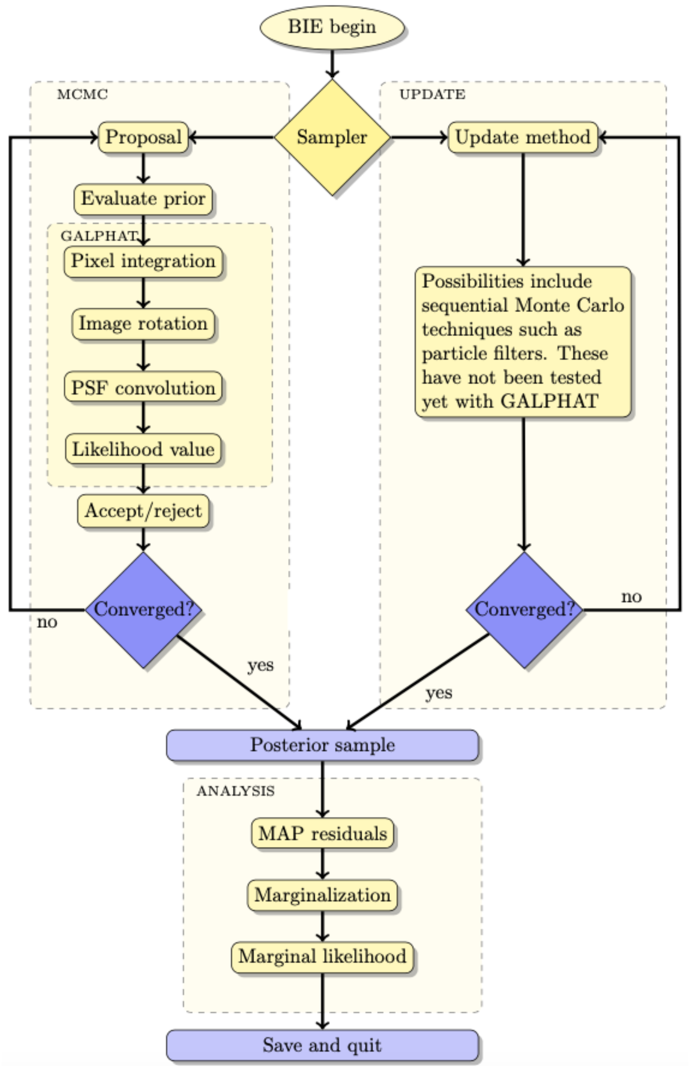


SOURCE: Author.

1.4 Main processing tool - GALPHAT

In this research, data processing is handled by GALPHAT (WEINBERG et al., 2013), which is a front-end application of the Bayesian Inference Engine (BIE), a parallel Markov chain Monte Carlo package. Figure 1.3 shows a graphical depiction of the GALPHAT execution. The BIE uses standard MPI and POSIX threads and, therefore, will run in a broad spectrum of parallel or scalar environments and can be easily ported to high-performance hardware for production analysis.

Figure 1.3 - Graphical description of a posterior inference using BIE with GALPHAT. The BIE is a general platform for estimating high-dimensional posterior distributions using Markov chain Monte Carlo (MCMC) algorithms (left column) and non-Markov chain samplers based on the «update» feature of Bayes Theorem, such as sequential algorithms (right column) flow.



SOURCE: Stalder et al. (2017).

GALPHAT provides full posterior probability distributions and reliable confidence intervals for all model parameters. The BIE relies on GALPHAT to compute the likelihood function (WEINBERG, 2012). GALPHAT generates scale-free cumulative image tables for the desired model family with precise error control. The interpolation of this table yields accurate pixelated images with any center, scale, and inclination angle. GALPHAT then rotates the image by position angle using a Fourier shift theorem, yielding a high speed and accurate likelihood computation.

Fundamentally, the BIE is a library, but the package provides a command line interface (CLI) with access to nearly all the import object classes (YOON et al., 2011).

1.5 Main goals

One of the possible ways to conduct astrophysical research is analyzing photometric parameters of galaxies to obtain insights into its formation, composition, evolution and death. As it could be presumed, when one has more analyzed galaxies – more refined and rich results will be delivered, and thus, more knowledge one will be able to build on it (STALDER et al., 2017). However, this ever-growing «more» has limits. Mainly, the problem consists of ever-growing computational time, so one needs to get more information from less data, or time will be spent just waiting for a sample to process with no concrete results at the end. Given the Big Data context of the large databases like SDSS, used in this work, or LSST[3], potentially to be used in future research, it is necessary to present software and hardware solutions to increase the performance of GALPHAT. So, one of the first goals of this research is to provide tuning of GALPHAT package installation and configuration to improve the performance and process data on the hardware available. By this, we conducted tests with GNU GCC and Intel ICC compilers, on two types of processors that we have in our disposal, Pollux machine based on Xeon Phi 7250 and a dedicated queue on Santos Dumont supercomputer with Intel Xeon E5-2695v2.

The main goal and final result of this project is processing our data sample of 24309 galaxies from the SPIDER sample using the «Sérsic» model (SÉRSIC, 1963). After that, organize the results and information achieved in comprehensive and easy to use form through the web-based visual system, while continuing working on an analysis of obtained data. This system and the back-end parts such as database, visual delivery and architecture are modeled and riped during processing to accommodate all necessary data and deliver it in an optimized manner. Hoping that the results will have great value in this field, we believe that this result will trigger several valuable articles. That being said, we are striving to conduct the analysis of the possibility to process a significant amount of data using Bayesian Inference tools. Given the fact of the upcoming extensive surveys (for example, LSST), we need to answer the

[3]LSST - Large Synoptic Survey Telescope

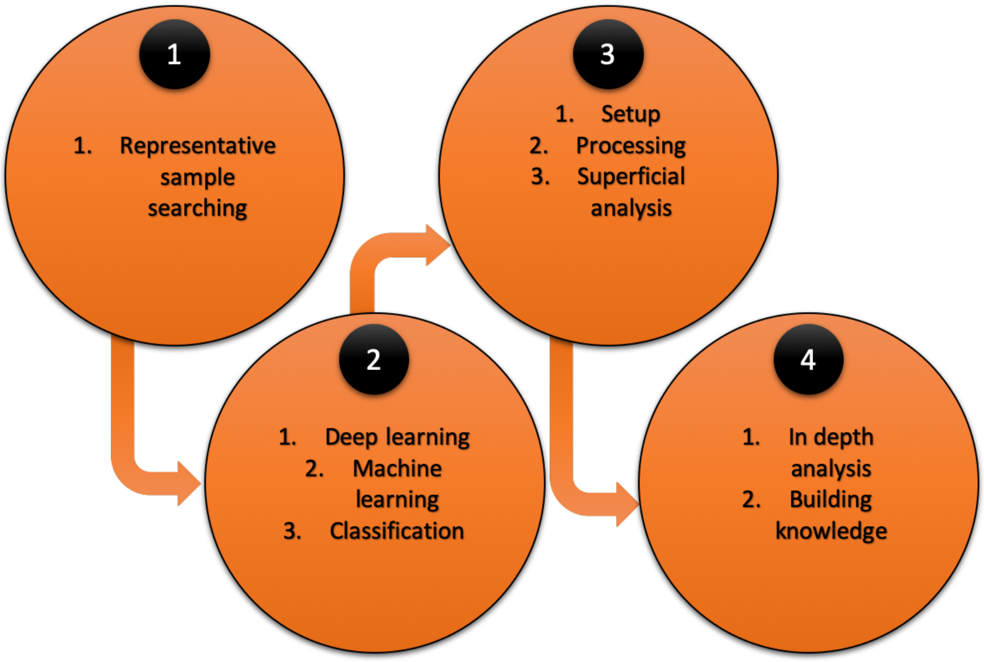
question if it is still possible to conduct in-place processing and analysis of data.

To help ourselves and other researchers who will need to use the same or similar tools to process their data samples, we continue the work on the improvement processing pipeline. Initially, the processing pipeline was fragmented and require different skills and programming language knowledge to be operated. This complexity starts with the proper setup of the environment, having several languages involved in different modules until small tweaks for some minor adjustments. It can get complicated and tedious very rapidly, so one of the simultaneous goals is to improve the routines of the submission and output analysis. By refactoring and parallelizing, we were able to achieve faster, cleaner and more modular code. This will come handy when it is necessary to implement new models. On top of this, we unified several modules into the submission system to facilitate the usage of GALPHAT and optimize the research sequence. We hope this will help other teams, if necessary, to have tools of galaxy processing at their disposal.

1.6 Context of this research

This work is part of a bigger picture of the research with the collaboration of several teams. For now, in the global context of the research, we have achieved results that would serve as a basis for continuing the work. We can divide the overall picture into four steps. The first is the study and analysis of potential data samples (galactic group or cluster, for example), which has the highest potential for further research (in our case, it was the study of the SPIDER sample). At the second stage, these data samples need to be filtered to obtain only the necessary data from the hundreds of thousands of galaxies. An example of this can be the work of (BARCHI et al., 2017) and only after this we come to the work presented in this dissertation. Here we process and store the data for the fourth step, the deep analysis and further research on it (TORTORA et al., 2016).

Figure 1.4 - Illustration of global research approach of the group.



Chapter 2 gives a quick look at our main processing program as well as a detailed description of processing pipeline execution and its modules. Right after, we present the optimizations that were already implemented to improve the processing sequence. The chapter ends with a summary of optimizations.

Chapter 3 describes in detail all the work that we had conducted on testing the processing pipeline in search of better execution speed. We describe three areas where we focused our efforts and include results that were rendered after their implementations.

Chapter 4 presents the troubleshooting of the non-converged galaxies after the first execution of the whole list reached its end, the theoretical background of methods that we used and the results that we have received.

Chapter 5 introduces our visualization interface for data that we received during processing. The chapter describes problems that we faced and our primary motivation for developing the interface. The chapter presents results on already functioning parts and shows the description of implemented functions.

In the conclusion section, we make a summary of all the achievements that we got during the research and present it in condensed form. Our vision of future development of the work can be found at the end of the chapter as well.

Brief specifications of hardware used in this research

Pollux - Main machine for tests and processing of «Sérsic» model.

Specifications

- CPU - Intel Xeon Phi 7250 68 cores (272 threads) at 1.4 GHz (1.6 GHz in TurboBoost)
- RAM – 192 GB (Initially 48 GB) of DDR4@2400GHz
- Operating System - Cent OS7, 3.10.0-693.el7.x86-64 kernel version.

Santos Dumont supercomputer (33 Thin nodes) - main machine for tests and processing of «Sérsic + Exponential» model:

Specifications (each node):

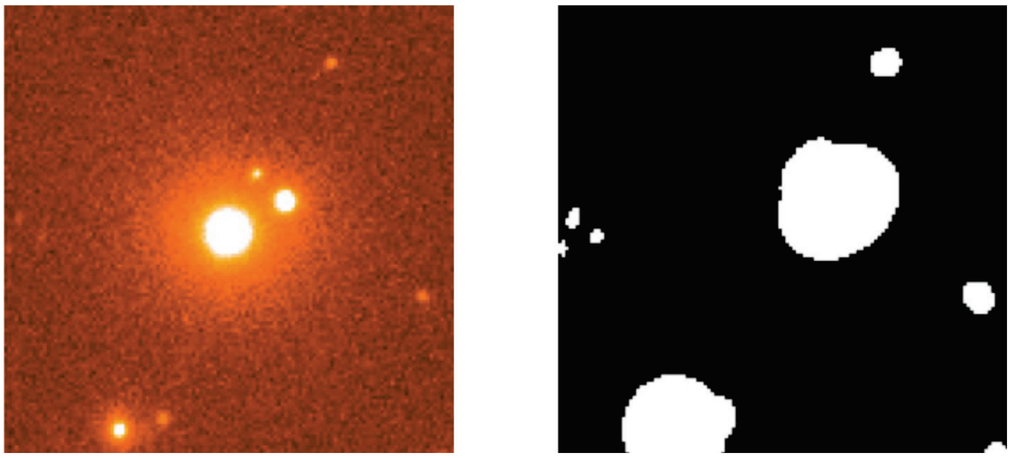
- 2 x CPU Intel Xeon E5-2695v2 Ivy Bridge, 2,4GHZ, 24 cores (12 por CPU), total of 792 cores
- RAM – 64GB DDR3 RAM
- Full specification could be found at `https://sdumont.lncc.br/machine.php?pg=machine`

2 PIPELINE: COMPONENTS, EXECUTION AND OPTIMIZATIONS

As we commented earlier, this research uses GALPHAT as main processing tool. To optimize usage of GALPHAT, especially with long lists of objects, an automated pipeline was developed, called PyPiGALPHAT. Its task is to analyze galaxy structural parameters, retrieve images from a given survey, generate configuration files, detect individual astronomical sources in these images, and finally schedule GALPHAT analyses and manage acquired data from an HPC cluster (STALDER et al., 2017, Hereafter as S17). Initially, this pipeline had support only for «Sérsic» and «Sérsic Source Point» models. Also, it is compatible only with the SDSS database. In the current research, we aimed to optimize, parallelize and rewrite it to permit extension for new models, especially «Sérsic+Exponential», which is one of our main goals in the long run.

When the list of galaxies is to be processed, it needs to pass through several steps. Firstly, data needs to be fetched and prepared in order to be ingested by GALPHAT.

Figure 2.1 - Example of the image and mask of one of the galaxies of our list, that was created by the pre-processing routines. Mask is removing secondary objects from the image before processing.



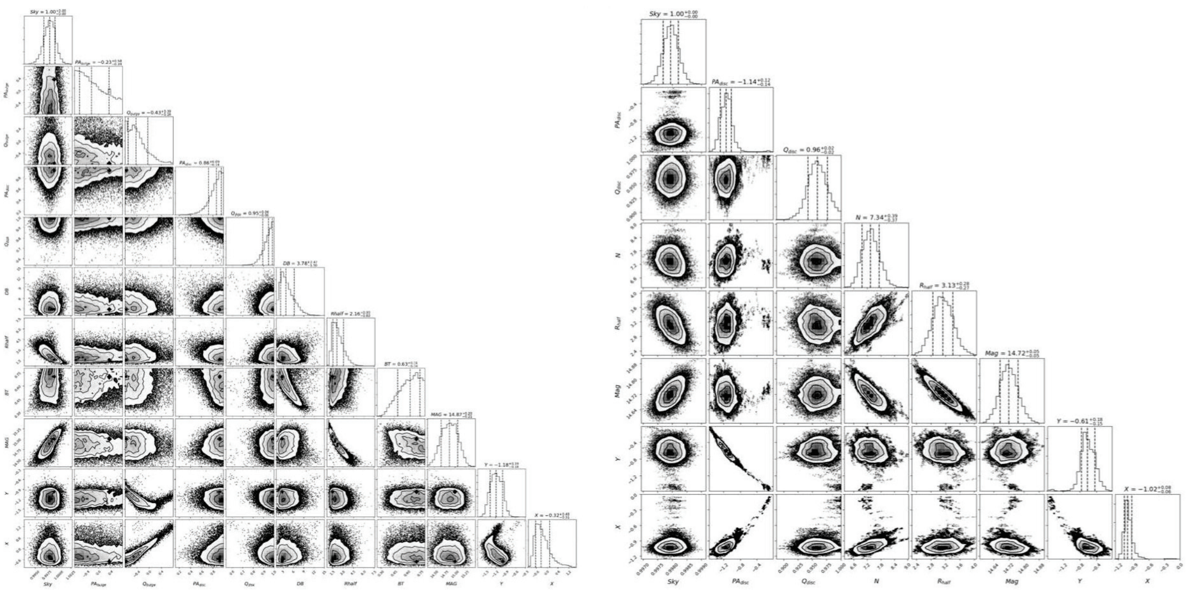
SOURCE: Author.

The program reads two-dimensional galaxy image data from a FITS file, generates a model image, and then computes the likelihood. This can be achieved with a series of auxiliary scripts written in *Python*, which goal is from supplied galaxy's ID, find, download and prepare a list of files that GALPHAT will use for processing. Figure 2.1 is showing an example of already cropped galaxy stamp and corresponding mask to exclude secondary objects. This data is stored in the SDSS telescope's database.

For a few galaxies, this process will not take long, but when the number of galaxies reaches thousands, pre-processing time may start to become very large. So, it is not unwise to think about some way of parallelization of this step.

In the next step, prepared scripts and images from the previous step are submitted for processing to GALPHAT, which depending on the machine, according to our earlier testing, can take from seconds (Supercomputer) to some days (home PC). For this reason, we had to spend some time preparing and optimizing the environment to improve the performance possible on the hardware available. Some insights into these processes are presented in Chapter 3.

Figure 2.2 - Example of the corner plots. They are the covariance matrices of converged values of each parameter after the chains were cleaned and cut. These charts are one of the results of the post-processing step (third step in the pipeline) step on converged galaxies. The left one is for «Sérsic+Exponential» (12 parameters) model and the right one is for the «Sérsic» (7 parameters) model.



SOURCE: Author.

After processing, we arrive at post-processing routines, which are compressing and cleaning the output. Right in the sequence, we produce various kinds of graphics representations from the output and this can tell us some interesting things about data sample that was processed and processing errors if there are some. Figure 2.2 shows the examples for the corner plots (parameter covariance) for the «Sérsic» and «Sérsic+Exponential» models. These charts are obtained by plotting posterior distribution of the parameters. The posterior file is the main output after running GALPHAT on a galaxy (if it has converged). Same as the pre-processing step, these routines are not so time-consuming as actual processing, but if the number of galaxies is high enough, it can get unexpectedly slow. For this purpose, it is also reasonable to implement parallelism to ensure better performance on scalable hardware. A summary of these steps is shown in Figure 2.3.

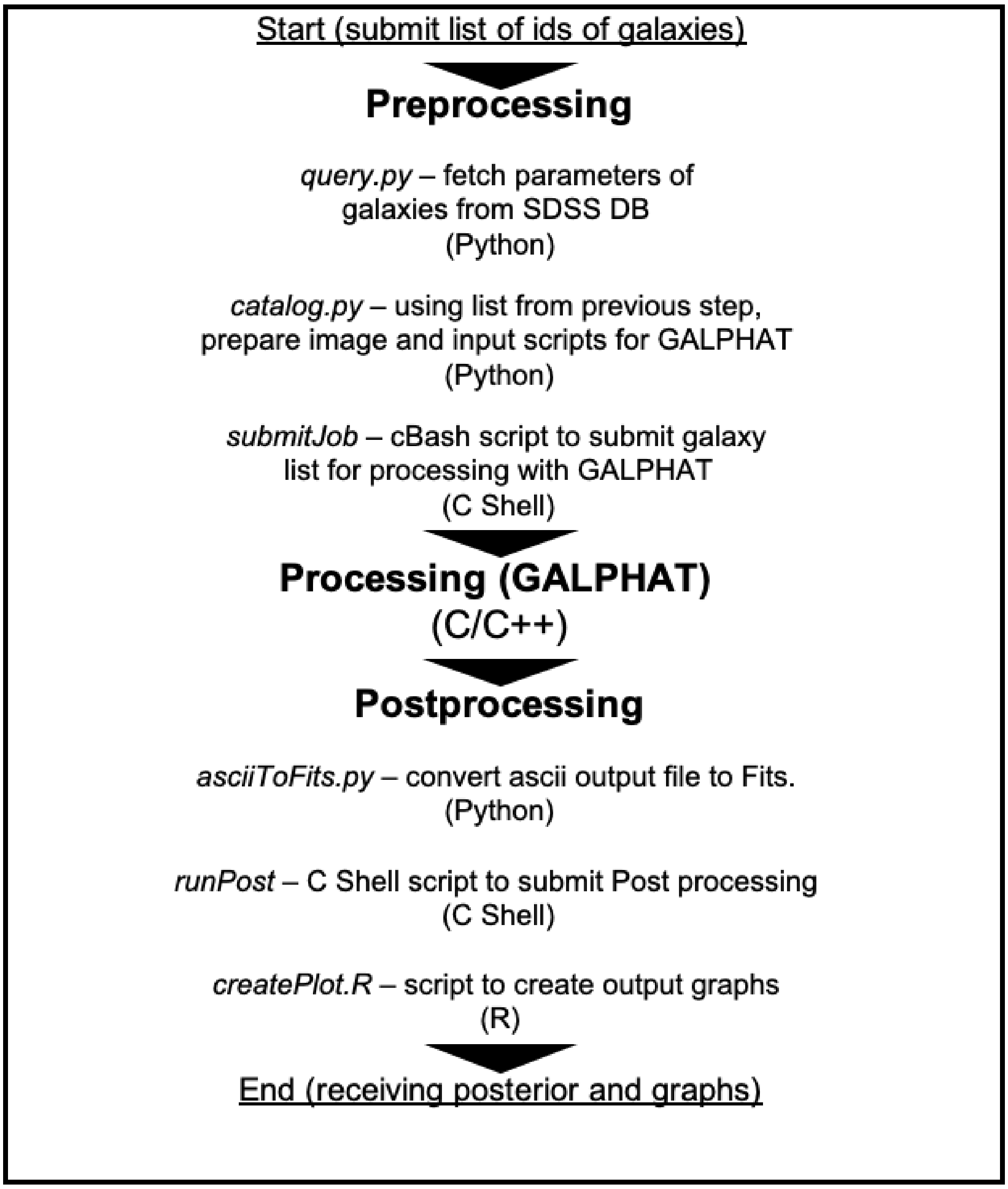
2.1 Composition and execution

As we can see from Figure 2.3, the current pipeline of processing is made up of several steps, where some of them are subdivided into additional steps. The main difficulty of this approach consists of complicated setup and consequent usage.

As we can see from Figure 2.3, currently, we have five different languages (*Python*, C Shell, C/C++ and R) and six separate steps; each step needs to be executed manually in the terminal to process a given list of galaxies. This brings up a problem of optimization because when one executes step-by-step manually and, in our scenario, we are working with thousands of galaxies, it requires a significant amount of unnecessary and repetitive work, which we hope to minimize with focused improvements of the processing pipeline.

Before we can start any of the improvements stated above, some work needs to be done on already existing pre-processing and post-processing code. This code was developed as a fast solution for past problems with not much attention to structure, organization and improvement potential. In the present state, this code is not suitable to be parallelized or modified. As these routines are poorly documented and organized, the first step is to refactor and organize them. We strive to make them clearer and ready to implement unit tests, which will be a handy diagnostics tool for future modification of code.

Figure 2.3 - Detailed overview of pipeline sequence of execution with information of used languages. Roughly, the processing pipeline consists of 3 main stages. Each of the main steps is subdivided into smaller steps. In the end, we are receiving the cleaned and compressed posterior file (the main output of pipeline) and the corner plot generated on it. On this figure, we only present the core steps of the processing, omitting some of the minor routines that are called in the process. Minor routines will be commented in Tables 2.1 and 2.2.


Start (submit list of ids of galaxies)
Preprocessing
query.py – fetch parameters of galaxies from SDSS DB (Python)
catalog.py – using list from previous step, prepare image and input scripts for GALPHAT (Python)
submitJob – cBash script to submit galaxy list for processing with GALPHAT (C Shell)
Processing (GALPHAT) (C/C++)
Postprocessing
asciiToFits.py – convert ascii output file to Fits. (Python)
runPost – C Shell script to submit Post processing (C Shell)
createPlot.R – script to create output graphs (R)
End (receiving posterior and graphs)

SOURCE: Author.

2.2 Improvements of the pipeline

2.2.1 Rewriting the post processing

Having several different languages and steps, it is considered the problem by itself. However, this fact makes things more difficult when one needs to unify the whole pipeline into a single system or even when committing minor changes. Having in mind that the codebase has several issues, it makes sense to rewrite the post-processing step from scratch. In this case, our main motivation of rewriting it was to couple the post-processing with other modules. We opted to start the rewriting using *Python* language. Our main goals were modularity, ability to parallelize if needed and core functions to get started. All the goals were achieved. The parallelization and its advantages are discussed in Chapter 4. Here we will point only to the fact that unifying all the code base to a single programming language makes all the consequent work much more comfortable. In our case, we will be working on the integration of all modules in a single suite.

2.2.2 Optimization and refactoring

As we had commented previously, to be able to commit necessary improvements and extensions, we needed to invest a good deal of time focused on improving the already existing codebase. Next in priority line was the need to implement the parallelism to enable the use of the full potential of the Pollux machine, which has poor single-core performance while having plenty of the available threads for parallel tasks. Most of the existing tasks could be classified as embarrassingly parallel (MATLOFF, 2015). These kinds of tasks are characterized by a great number of independent objects to work on and consequently it is quite easy to distribute them across available cores. Before starting the parallelization process, the first thing to do was optimization and refactoring of our code to prepare it and look for the key spots where we can introduce the parallelization.

In the case of post-processing, we rewrote the main parts of the routines entirely, already in a parallel manner. As for pre-processing, instead of explicitly parallelizing the preparation of input, we opted to leave the distribution of the work across the cores to the automated submission system. When we are submitting a given list of galaxies, the workload of preparing is already being distributed between available hardware. More detailed discussion and results are in Chapter 4. By definition, refactoring is changing the code's structure without changes in its functionality. It was important to firstly understand and then restructure the code to be able to open the way for further improvement. In summary, we achieved satisfactory results in refactoring, which was the first step that opened a possibility to implement the new model and other improvements. Table 3.1 shows the result of the code improvement and the codebase refactory. There we show how the number of functions increased, together with decrease in lines per functions. This made code more readable and maintainable.

Table 2.1 - Refactoring results of main pre-processing scripts. Two main scripts of this step are catalog.py and query.py (they are described in Figure 2.3) rest is auxiliary scripts that are being called during the processing.

	Before		**After**		**More Funct.**	**Less code p/ funct.**
	Functions	**Lines p/ funct.**	**Functions**	**Lines p/ funct.**		
catalog.py	4	27	8	20	**+4.0**	**-7**
cutPsf.py	3	20	3	16	**+.0**	**-4**
genImg.py	3	99	15	23	**+12.0**	**-76**
genScript.py	5	77.2	5	77.2	**+.0**	**0**
query.py	1	116	8	13	**+7.0**	**-103**
sqlcl.py	5	14.2	5	14.2	**+.0**	**0**

SOURCE: Author.

After refactoring, the code became clearer and more understandable, which helped a lot in understanding of its functionality. Consequently, we implemented basic unit tests to be able to control and verify all the consequent changes and improvements. Unit tests are of a great help when one needs to commit some changes to the code, but is unsure if it will continue to work in the same manner. We used Py.test[1] framework to implement tests. Table 2.2 is showing the results of unit test implementation. Some routines, even after unit test implementation, continued to have low test coverage. This is due to the fact that they do not have any output that could be evaluated or compared to test value. Most of the functions are only writing or reading from a file. These routines have very few functions as well, so having two of the total three functions writing to the file, renders a poor percentage of the test coverage.

[1]https://docs.pytest.org/en/latest/

Table 2.2 - Unit test implementation results for main pre-processing scripts.

	Before	**After**
	Test coverage	Test coverage
catalog.py	0%	24%
cutPsf.py	0%	28%
generateImages.py	0%	88%
generateScript.py	0%	22%
query.py	0%	94%
sqlcl.py	0%	31%

SOURCE: Author.

2.2.3 «Sérsic+Exponential» implementation

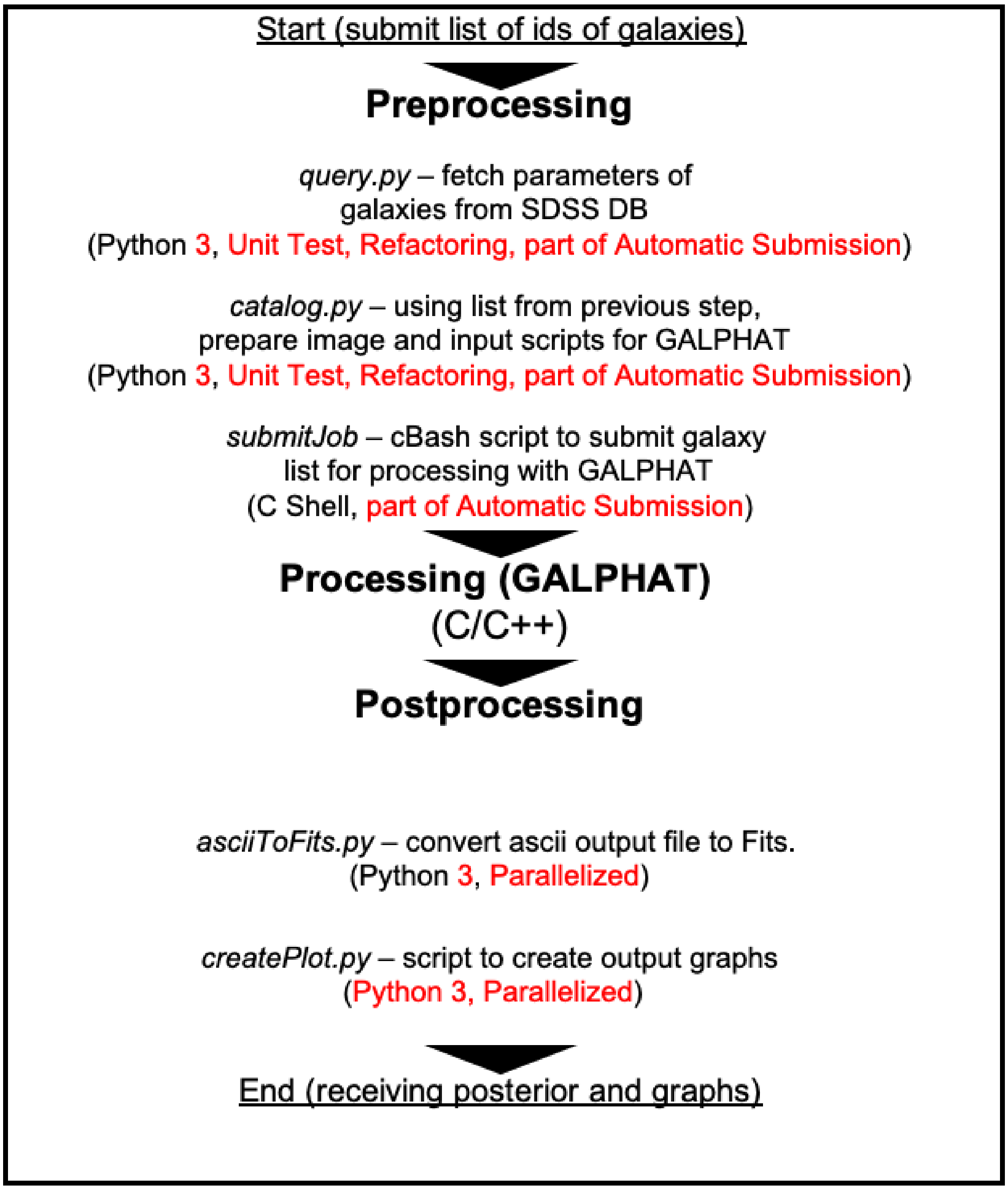
Besides the optional but beneficial improvements like parallelization and optimization, we needed to extend the core functionality of the system with the ability to prepare the input scripts for «Sérsic+Exponential» as well as generate post-processing charts. While «Sérsic» is fully implemented by S17 (Stalder et al. (2017)) and ready to be used in production, «Sérsic+Exponential» is far from being ready, code and testing alike. So, one more reason for refactoring and test implementation is that we need to have clear and comprehensive code to be able to increment new model functionality.

Having this opportunity, the refactoring that was done with Pre-processing and Post-processing facilitated new model implementation. Firstly, we needed to perform all the commented steps above. In the end, we were able to increment the necessary functions and test the required functionality. As for now, it is being used for the processing on Santos Dumont supercomputer.

2.3 Summary of achieved results

Summarizing our achievements with improvements in the processing pipeline, we can conclude that we were able to improve the consistency of the pipeline significantly by rewriting the main routines of the post-processing step. At the same time, pre-processing was refactored and optimized in a way to reduce the complexity of understanding and facilitate further improvement. Figure 2.4 is showing the parts that were modified. Everything that had passed through some change is written in red.

Figure 2.4 - The updated figure of the pipeline sequence with already implemented modifications written in red color. Quick summary: preprocessing was adapted to Python 3, parallelized, and unified into a single automated submission system; main parts of post-processing were rewritten from scratch using Python 3, already with support for SE model and in parallel form.



Start (submit list of ids of galaxies)

Preprocessing

query.py – fetch parameters of galaxies from SDSS DB
(Python 3, Unit Test, Refactoring, part of Automatic Submission)

catalog.py – using list from previous step, prepare image and input scripts for GALPHAT
(Python 3, Unit Test, Refactoring, part of Automatic Submission)

submitJob – cBash script to submit galaxy list for processing with GALPHAT
(C Shell, part of Automatic Submission)

Processing (GALPHAT)
(C/C++)

Postprocessing

asciiToFits.py – convert ascii output file to Fits.
(Python 3, Parallelized)

createPlot.py – script to create output graphs
(Python 3, Parallelized)

End (receiving posterior and graphs)

SOURCE: Author.

The whole sequence received speedup by either parallelizing or using the automated submission system. Due to the fact that Python 2 reached its end of the life, we had refitted the preprocessing steps to run them with Python 3. All the steps, except post-processing, are already integrated into the automated submission system, which greatly decreases the hustle of submission of new processing jobs, leaving the bulk of the work to the automatic scheduler.

3 PERFORMANCE OF PIPELINE AND ITS IMPROVEMENTS

The current chapter seeks to show all the work that was done to improve the performance of the processing pipeline and, by this, bring processing time to its minimum, having in mind that the actual processing time was one of our main problems. Roughly, we can divide our efforts into three separate areas: (1) Compiler and all the tests and optimizations related to it; (2) implicit parallelization of post- and pre-processing routines; (3) scaling and adaptation of pipeline to improve performance on hardware available.

Here we will present in detail the peculiarities of each part and summarize obtained results at the end. As our starting point, we will use the results from (STALDER et al., 2017, Hereafter as S17) as a comparison. We describe the state of the codebase that we received from that research and show the development that was done on top of it.

3.1 Compiler

3.2 Initial general testing

At the beginning of our research, we started with preliminary tests with NAS benchmark[1] to verify two things: (i) influence of compilers on GALPHAT performance and (ii) scaling and performance lose if engaging maximum number of threads on our newly acquired machine. The reason to expect better performance with Intel's C Compiler with GALPHAT on this setup was good results in general benchmark tests such as those seen in Table 3.1.

The first important detail is that with the usage of a maximum number of threads, speedup obtained with the ICC compiler grew even more prominent, meaning that when we are using near the maximum of the machine for galaxy processing, we can gain even more performance with the Intel compiler.

Another important detail in Table 3.1, we will discuss the same table but slightly different analysis, to show promising capabilities of the Xeon Phi processor in «embarrassingly parallel» tasks. We can see that the Intel compiler gives better results in case of the efficiency of speedup. Efficiency would be 100% (ideal case) if we would get speedup proportional to the increase of active threads. In this case, we had 20 in the first test and 272 in the second. It converts to an increase of 13.6 times. If

[1] https://www.nas.nasa.gov/publications/npb.html#url

Table 3.1 - Results of benchmark using 20 and 272 threads on Xeon Phi (Pollux) to compare the performance and efficiency of GCC and ICC compilers. As load task was used NAS benchmark with EP (Embarrassingly Parallel) Kernel and C class. The results were obtained by running each test three times and getting mean value. The values in the table represent time, in the format mm:ss,ms (mm - minutes, ss - seconds, ms - milliseconds).

NAS benchmark tests with 20 and 272 threads.			
	20 threads	272 threads	**Parallelism Speedup**
GCC	00:58,01	00:07,73	**7,50**
ICC	00:30,36	00:03,64	**8,34**
Compiler Speedup	**1,91**	**2,12**	

SOURCE: Author.

we had obtained the same speedup, we would be able to say that we can use all available threads without losing any performance. Nevertheless, due to the fact that we have only 68 real cores, each capable of running four simultaneous threads, that is not possible. In our case, we start to lose performance while using $\sim$ 55% (for the GCC compiler) and $\sim$ 61% (for the ICC compiler) of the total number of threads. While we do not get the ideal case of efficiency when employing all threads, we are able to use more than half of the machine for our processing. Later in the chapter, we will comment more on this matter in the context of GALPHAT. After receiving this information, we proceed to build the environment to test the compilers on the real target-application data.

3.2.1 Initial testing with target-application data

During the initial setup of the GALPHAT on the Pollux machine at the beginning of the research, we conducted tests on the comparison between the GNU and ICC compilers. Received results were surprisingly good (the difference between the compilers was more then 4 times) in favor of the Intel compiler. Due to the fact that we needed to start as soon as possible the processing of the «Sérsic» model, unfortunately not enough time was dedicated to certifying the correctness of the obtained results. Later on, when we ended the processing of the whole list of the galaxies (24309) and the machine became once again available, we got the opportunity to verify the accuracy of the previous testing. This time we prepared the environment and main packages thoroughly to ensure the correctness. For the test to compare the compilers we installed the following versions of the main packages showed in Table 3.2

Table 3.2 - Information on environment setup of compilers and main modules. BoostLib and OpenMPI for each compiler were built from source with correspondent compiler and linked during building of GALPHAT via *PATH* and *LD_LIBRARY_PATH* environment variables. All other dependencies were the same.

Compiler	**BoostLib/ OpenMPI**	**Environment Variables**
GCC 5.4.0	1.61/4.03	CC="gcc-5", CXX="g++-5", CFLAGS="-O3", CXXFLAGS="-O3", F77="gfortran-5", FC="gfortran-5", F90="gfortran-5"
ICC 19.0.0.117		CC=icc, CXX=icpc, CFLAGS="-O3 -xHost -ip -no-prec-div -static-intel", CXXFLAGS="-O3 -xHost -ip -no-prec-div -static-intel", F77="ifort", FC="ifort", F90="ifort" CPP="icc -E", CXXCPP="icpc -E" FFLAGS="-O3 -xHost -ip -no-prec-div -static-intel"

SOURCE: Author.

Using these settings, we run two kinds of tests, first with prototype image (small galaxy, 48x48 pixels) that comes with GALPHAT installation. Another test was with 15 galaxies from our sample of varied sizes.

Table 3.3 - Results of conducted test with in-built galaxy from GALPHAT installation directory using GCC and ICC compilers. In each case 16 cores were used. The values in the table represent time, in the format hh:mm:ss (hh - hours, mm - minutes, ss - seconds).

Compiler comparison on Pollux with Prototype galaxy	
	Processing time
GCC	00:06:18
ICC	00:04:04
Speedup	**1,55**

SOURCE: Author.

From the comparison Table 3.3, we can see that the Intel compiler still has an edge in comparison to the GNU, but nothing like the previous results. On a small galaxy, we had about 50% of speed up with the usage of the ICC compiler. After tests with prototype galaxy, we proceeded with tests using a list of real varied sized galaxies from our sample.

Table 3.4 - Results of conducted test with 15 real galaxies from SPIDER sample using GCC and ICC compilers. First and second columns show the galaxy's id and size in pixels respectively. The size is calculated using the following formula. Showing mean values for the: size, time and speedup of 15 galaxies. The speedup was calculated using geometric mean formula. The time is given in the format hh:mm:ss (hh - hours, mm - minutes, ss - seconds).

Galaxy ID	**Size**	**Time GCC**	**Time ICC**	**Speedup**
587739706883637600	83	00:32:46	00:25:29	1,28
587739707420836297	97	01:00:18	00:38:14	1,57
587739707420901762	107	00:44:33	00:33:16	1,33
587739706884292961	117	00:35:12	00:27:14	1,29
587739706884030704	127	00:59:45	00:44:23	1,34
587739707420901697	127	01:18:27	01:01:38	1,27
587739706883965152	132	01:11:22	00:50:44	1,40
587739706884489629	137	01:02:00	00:42:17	1,46
587739706884358378	148	01:13:26	00:53:41	1,36
587739707420901732	149	01:09:33	00:54:56	1,26
587739706883899754	158	00:53:42	00:40:22	1,33
587739706883899733	159	01:26:44	01:03:41	1,36
587739706884030676	165	01:04:28	00:49:54	1,29
587739707420901432	177	01:07:26	00:55:04	1,22
587739707420901740	183	01:06:21	00:46:02	1,44
Mean values	**138**	**01:01:44**	**00:45:48**	**1,35**

SOURCE: Author.

The Table 3.4 shows results while comparing GCC and ICC compilers on Pollux, using 20 threads for the processing. Using galaxies from our list as testing input, we had a geometric mean speedup of 1.35, which we can convert in more than two months of total time economy when processing the whole sample. In conclusion, we still got a good deal of performance increase by using ICC to justify the switch and recompilation of the environment.

3.2.2 Comparison between Xeon Phi and Xeon E5 2660 (from S17 research)

After preliminary setup and testing, our goal is to compare the performance between Pollux (Xeon Phi) and setup where S17 obtained his results. This is important to do to have a starting point to make further comparisons taking into account differences between hardware.

Figure 3.1 shows the speed of processing of the entire sample of 1200 galaxies used in that research, which are the subsample of our sample. Basing on this Figure 3.1, and its regression line with the equation, we can calculate the time that it would take when using 20 active threads while processing real galaxy with GALPHAT installed on Helios cluster using GCC. During S17 research, they were using only GCC. According to the equation, we can compute the time that it took to process the same galaxy from Table 3.4 of size 132 would take about 30 minutes to process.

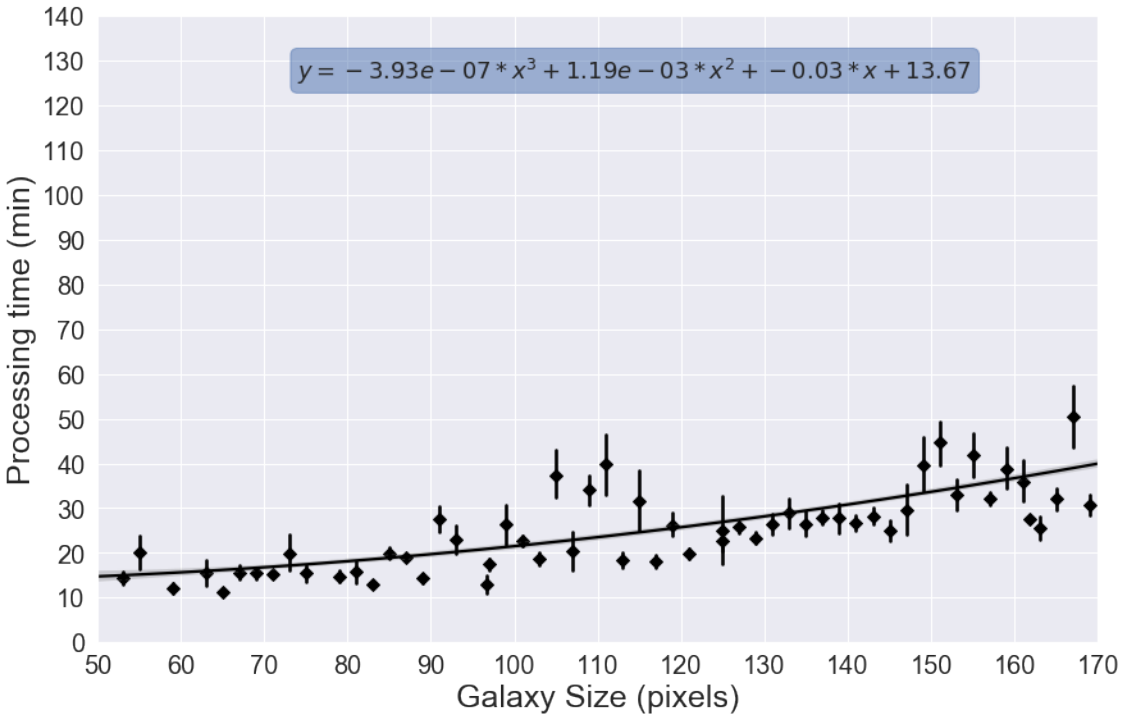
In this case, if we compare the result obtained on Pollux with GCC compiler of the galaxy (587739706883965152), with the estimated time of S17 research, speedup between the two setups would be about ∼2.39 times (71 min vs. 30 min) in favor of Helios cluster, all this while using GCC. On the other hand, using the same setups and input galaxy but changing compiler to ICC on the Pollux, we can see a significant decrease in the processing time and consequent decrease of speedup. This shows one clear argument of the importance of usage of the Intel compiler on the Xeon Phi processor family for GALPHAT.

Table 3.5 - Comparison between Helios using GCC compiler and Pollux using ICC while running same galaxy with Sérsic model and using 20 threads in each case. The values in the table represent time, in the format hh:mm:ss (hh - hours, mm - minutes, ss - seconds).

Compiler comparison on Pollux with ICC and Helios GCC	
	Real Galaxy(132)
Pollux (ICC)	00:50:44
Helios (GCC)	00:30:00
Speedup	**1,69**

SOURCE: Author.

Figure 3.1 - Time per image size obtained in the previous study of S17 with ∼1200 galaxies (a subset of our sample). In that research, GALPHAT was installed using the GCC compiler on Xeon E5 2660 (Helios Cluster). Extrapolating this, we will get bleak predictions on how long it would take to process our data. With obligatory assumption that we would be able to achieve the same processing speed as of S17 research, which we did not achieved at the very beginning. But even being able to achieve the same performance, looking at sizes of the main portion of the galaxies in the Figure 1.2 it turns to be practically unviable.



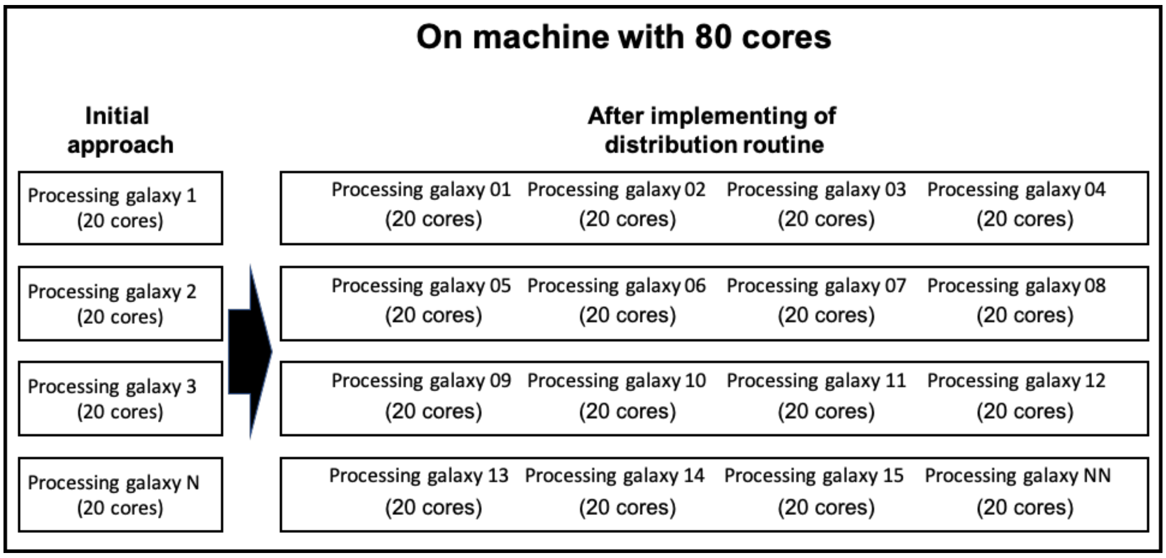
SOURCE: Stalder et al. (2017).

Table 3.5 shows that using ICC improves the performance of the Pollux significantly and shrinks the Helios edge to only 1.69. Just to be clear, while rebuilding GALPHAT with ICC, we had to build the majority of required libraries with ICC as well (OpenMPI, BoostLib). Some compilation succeeded after a lot of testing. BoostLib could be cited as a good example, as we found that only version 1.61 is compatible with ICC.

3.3 Scalability

Besides raw performance issues, we also have scalability problems. GALPHAT, as we have mentioned earlier, is already parallelized with MPI and is scalable inside a single processing node of conventional server CPUs (with core count between 20 and 72). However, it lacks scalability on many-core systems like Xeon Phi or clusters of several nodes containing hundreds or thousands of cores. Currently, one will not benefit from having a machine with more than 72 cores, which can be employed for processing of a single galaxy. This is one of the main issues that was being addressed during this research. In order to solve this, we needed to develop a submission system which will distribute galaxies through available cores. In Figure 3.2 this can be seen as the difference between the current and aimed modes of processing for a list of galaxies with «Sérsic» model (using 20 cores per galaxy).

Figure 3.2 - Demonstration of modification of the processing approach on many-core systems (like Xeon Phi). To be able to use all available core/threads, we prepared the automated submission system that is capable to split and distribute the galaxy list equally into smaller lists and start processing on each of then independently. Processing several galaxies simultaneously is decreasing drastically processing time of a single galaxy.


On machine with 80 cores
Initial approach
After implementing of distribution routine
Processing galaxy 1 (20 cores)
Processing galaxy 2 (20 cores)
Processing galaxy 3 (20 cores)
Processing galaxy N (20 cores)
Processing galaxy 01 (20 cores)
Processing galaxy 02 (20 cores)
Processing galaxy 03 (20 cores)
Processing galaxy 04 (20 cores)
Processing galaxy 05 (20 cores)
Processing galaxy 06 (20 cores)
Processing galaxy 07 (20 cores)
Processing galaxy 08 (20 cores)
Processing galaxy 09 (20 cores)
Processing galaxy 10 (20 cores)
Processing galaxy 11 (20 cores)
Processing galaxy 12 (20 cores)
Processing galaxy 13 (20 cores)
Processing galaxy 14 (20 cores)
Processing galaxy 15 (20 cores)
Processing galaxy NN (20 cores)

SOURCE: Author.

In Figure 3.2, we can see the difference between processing approaches. In the beginning, we had a sequential submission of galaxies when each galaxy processed in a parallel manner. However, a list containing N galaxies will be processed sequentially, one galaxy after another, no matter how many idle cores are available. Our proposal consisted of implementing the algorithm to enable smart distribution between available cores. For example, having 80 cores available on a given machine would enable us to run four galaxies at the same time, in case of the «Sérsic» model[2]. In the initial state, this was not possible, but after the implementation of the distribution system, routines are doing it automatically.

When we started the processing approach, in the middle of this research, we were manually submitting several lists of galaxies to be processed simultaneously as multiple instances of the same program. Firstly this was done on Pollux with 272 threads and then simultaneously testing SE model on Santos Dumont as well. In each case, we needed to calculate the number of lists and prepare them, running each through steps shown in Figure 2.3. However, with time, we have invested a significant effort in developing an automated submission system, which is already functional and being used on Santos Dumont supercomputer.

The main chunk of the work was done and the system is functional both on Santos Dumont and Pollux. The only difference is the final step of submission. On the Pollux, it simply submits the specified number of instances of GALPHAT via *nohup* command and starts the processing. On the Santos Dumont, the final step is the generation of the .srm file, which is used to submit the job to the SLURM[3] (processing manager). On Santos Dumont, we have 33 Thin nodes (each with 24 cores, totaling 792 core in total) to process the «Sérsic+Exponential» model. Given the fact that SE uses 72 cores to process one galaxy, this results in the possibility to process 11 galaxies simultaneously (792/72) with available core count. We have already prepared the environment and the automated submission system is functional on Santos Dumont supercomputer, where we have initiated the processing of the «Sérsic+Exponential» model.[4]

This seemingly simple tweak drastically improved the performance of the Pollux machine, enabling the usage of the more significant portion of resources available on the machine during processing and by this, bringing Xeon Phi machine's performance on par with results obtained during previous research. The only caveat, in this case, is much higher usage of RAM on Pollux, because if processing one galaxy roughly consumes 20 GB of RAM, processing four will require 80 GB and running six galaxies at the same time will need 120 GB of available RAM. In our case, in the beginning we started with 48 GB on Pollux, being able to run two galaxies at most. During

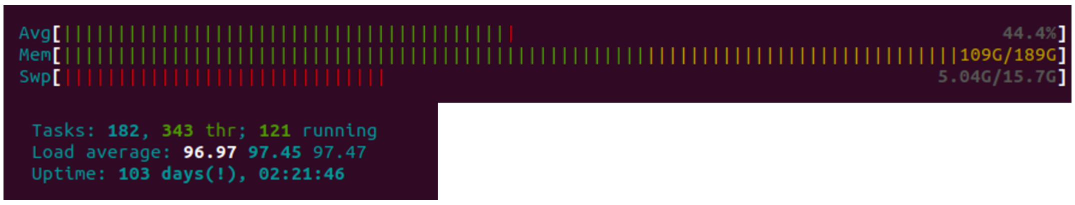
[2]«Sérsic» requires 20 threads and 20 GB of RAM

[3]https://slurm.schedmd.com/documentation.html

[4]The code is available on : https://bitbucket.org/galileo13/sistema-de-submissao-automatica.git

the research, we gradually upgraded to 96 and then to 192 GB, being able to test running ten galaxies at the same time with «Sérsic» model . While it could seem like a good idea to run ten galaxies at the same time, using 200 threads and 200 GB of memory, we cannot forget that Pollux has 68 real cores, each supporting up to four threads, resulting to 272 threads that could run simultaneously. However, it does not mean that these threads have the same performance as the real core. It is merely due to CPU scheduler, by smart planning, enables us to run more than one task on the same core. This technology is called Intel HyperThreding (STOKES, 2002). During the tests, we saw that when one uses more than 120 active threads, performance starts to suffer penalties of over saturation of processors queues and the time of processing starts to grow, detrimental to the gains. This result corroborates our previous findings during testing with NAS benchmark in section 3.2. Through the testing, we found that an optimal number of galaxies to be run simultaneously on this machine is six, using 120 active threads. This could be seen in *htop*, showing that, while processing six galaxies, we are near the limit of the actual capacity of the machine, as one can see in Figure 3.3.

Figure 3.3 - *htop* is an utility that shows the load of the machine. In this figure we are showing the state of the Pollux during processing of 6 galaxies simultaneously. This result corroborates with our previous findings.



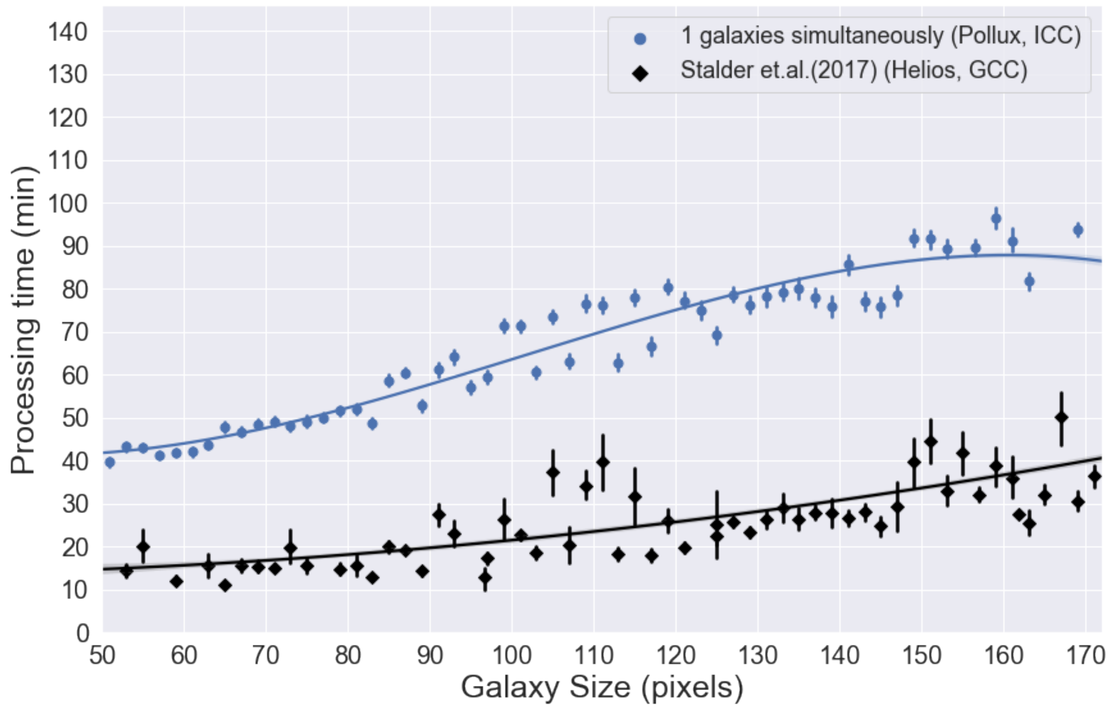
```
Avg[||||||||||||||||||||||||||||||||||||||||||                                 44.4%]
Mem[|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||109G/189G]
Swp[||||||||||||||||||||||||||||||                                          5.04G/15.7G]

  Tasks: 182, 343 thr; 121 running
  Load average: 96.97 97.45 97.47
  Uptime: 103 days(!), 02:21:46
```

SOURCE: Author.

Nevertheless, we are gaining a tremendous amount of performance, while running more than one galaxy on the Pollux. Because when submitting up to 6 galaxies, we are receiving practically linear speedup. As we finished processing the «Sérsic» model, we have 19000 galaxies processed on Pollux using GALPHAT built with ICC. Most of the processing was done running four galaxies simultaneously, and in the end, we gained a RAM upgrade, going from 80 GB to 192 GB. This enabled the possibility to run six galaxies simultaneously. Firstly, we are comparing in the case of running just one galaxy at the same time on Pollux with results obtained from

S17 research. The Figure 3.4 is showing the comparison between the processing of a single galaxy at the same time on Pollux and Helios.

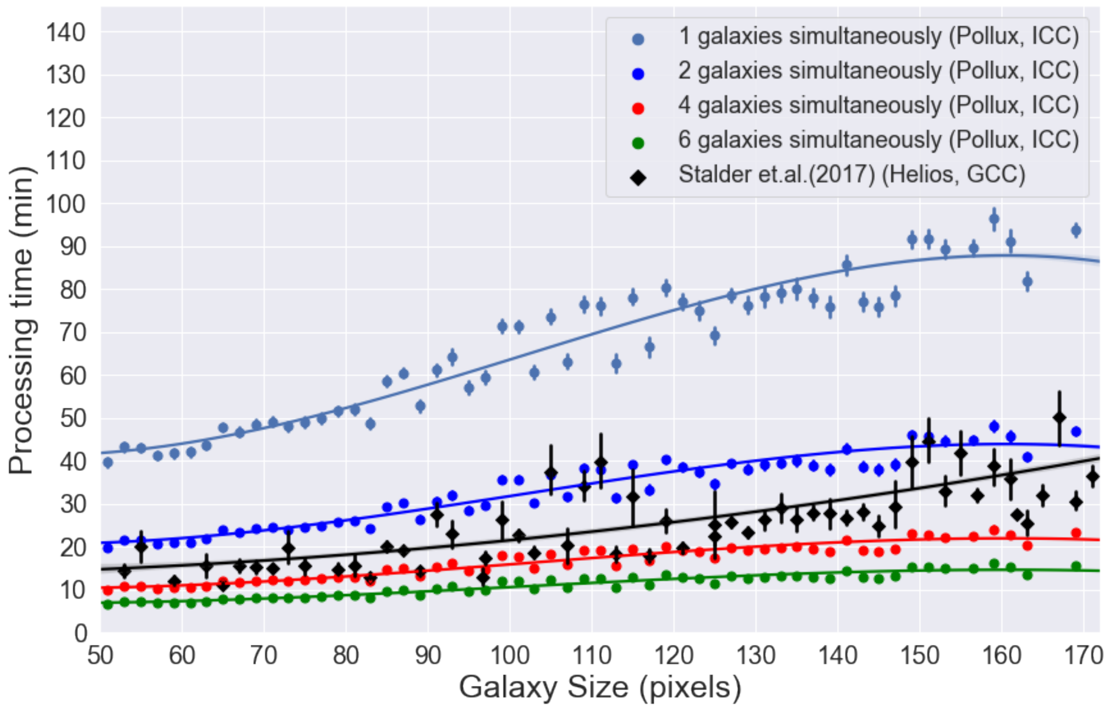
Figure 3.4 - Comparison between Helios (GCC) and Pollux (ICC) running 1 galaxy simultaneously with «Sérsic» model. The data was cleaned from outliers greater than 2 * Standard Deviation(STD) on both X and Y axis.



SOURCE: Author.

From the Figure 3.4, it can be seen that more powerful cores of Xeon E5 2660 cluster is almost twice faster than the Pollux machine. This was the primary problem since the beginning. Intel compiler helped to mitigate it in some way, but still we were far behind of the performance of S17 research. Which in consequence was making the processing time of our dataset unfeasible. Now, if we look at the Figure 3.5, where we are showing the case of several galaxies being processed simultaneously, bringing the time of a single galaxy divided by the number of running galaxies at a given batch.

Figure 3.5 - In this figure, we are showing the results of processing several galaxies at the same time. These are compared to the S17 performance. Each line corresponds to a number of galaxies that are executed simultaneously, using 40, 80 and 120 GB of RAM and 40, 80 and 120 threads, respectively, using «Sérsic» model. The data was cleaned from outliers greater than 2 * STD. This shows that when running four simultaneous galaxies, we were able to surpass results from S17 research. With our final amount of RAM, after all upgrades of the machine, we ended up settling for six galaxies at the same time, resulting almost in 2 fold faster than S17 result.



SOURCE: Author.

Figure 3.5 is our definitive comparison of performance between Pollux and Helios. As it could be seen, while wasting performance when processing just one galaxy, we were able to significantly improve the results if one will process two, four, or six galaxies at the same time. Following the principle described in Figure 3.2, instead of running one galaxy after another, we split the input list in several smaller lists and create one processing instance of GALPHAT for each of the smaller lists. Precisely for these cases, we had developed the automated submission system that helps to distribute and prepare several galaxies into the processing. The automated submission system permits to do it much faster and in automatic manner, without the need to run through each step depicted in Figure 2.3

In conclusion, to get the best performance with GALPHAT it is better to choose a smaller number of high-frequency cores instead of many low-frequency cores (like Xeon Phi) or, in case of Pollux, it is necessary to have higher amounts of RAM to compensate low frequency by running several galaxies at the same time. As for RAM, it is easy to follow a rule of 1 Monte Carlo Chain (mchain) = 1 thread = 1 GB of RAM. It means that if one will process galaxy with «Sérsic» model, which uses 20 *mchains*, for the best performance, he would need to have 20 available threads and for each chain/core 1 GB of free RAM.

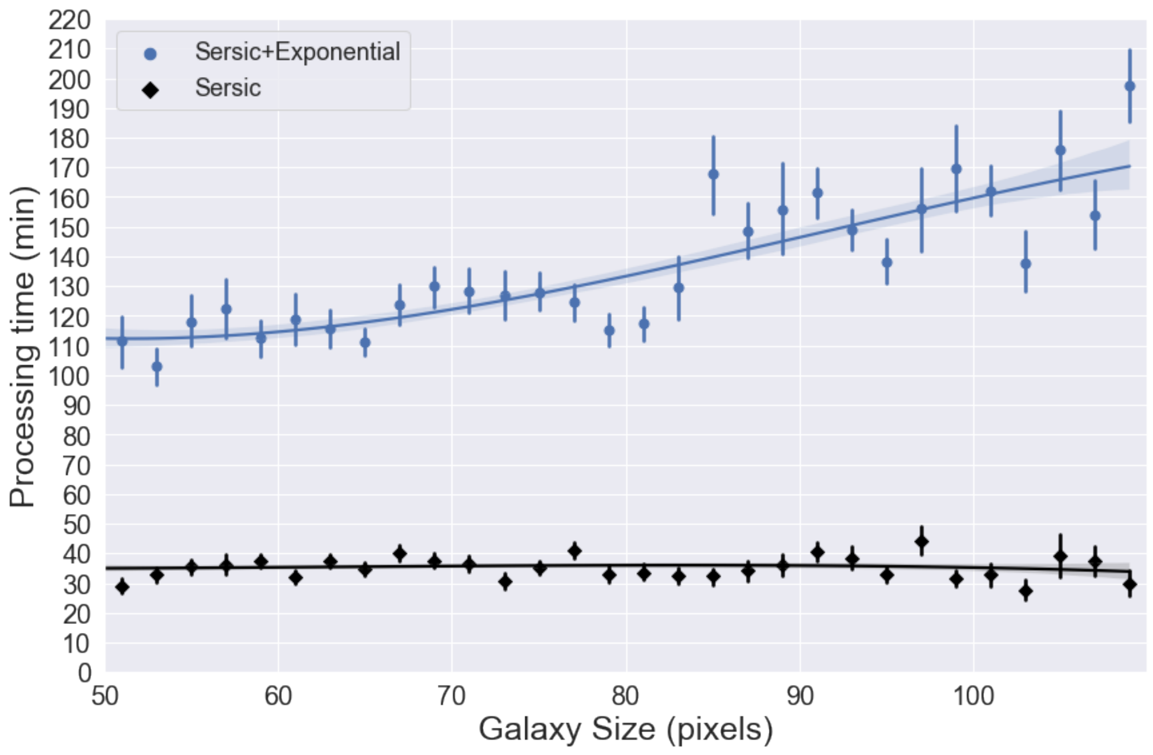
In the case of SE model, we are using 72 threads to run 72 mchains, following the same requirement of 1 GB per mchain. Potentially, the SE model could be run with fewer mchains, but this will result in less convergence chances, as some mchains are dropped during the processing due to the overfitting, which causes individual chains to become blocked and no longer achieve progress in the processing.

3.4 «Sérsic+Exponential» performance remarks

Our initial tests with SE on Pollux had shown that it is entirely unfeasible to use this machine for this model. In some cases, the difference in time was reaching 10 fold or more to process the same galaxy with SE, getting nearly one day of processing time for a single galaxy. Moreover, to make the case worse, we would be able to process only two galaxies simultaneously, because, as we already commented, it is needed 72 threads to process a single galaxy and we are bound by about 120-140 active threads due to the decline of the performance if employing more then this number as it was shown in section 3.2. As a consequence, we decided to proceed with the processing of the SE model on the Santos Dumont, where, at the time, we received access to the dedicated queue. Although we are at fairly early stages of processing of the «Sérsic+Exponential» model, it is possible to use already processed data to analyze the difference in processing requirements for our models. Starting with more then triple[5] of resources required to process a single galaxy, and the comparison given in Figure 3.6, we can see that processing our data sample with SE model will be a much more difficult task than it was with «Sérsic». In future work, to improve this result even further, it could be needed to study the possibility of porting parts of highly CPU intensive code to GPU.

[5]20 CPUs for «Sérsic» against 72 CPUs for SE

Figure 3.6 - Comparison of time per image size obtained while processing 1800 galaxies with «Sérsic» and «SE» models on Santos Dumont. In both tests, we were using Thin nodes. We submitted two different jobs, for each model. Each galaxy was using 72 cores for «SE» and 20 cores for «Sérsic» model. The data was cleaned from outliers greater than 2 * STD.



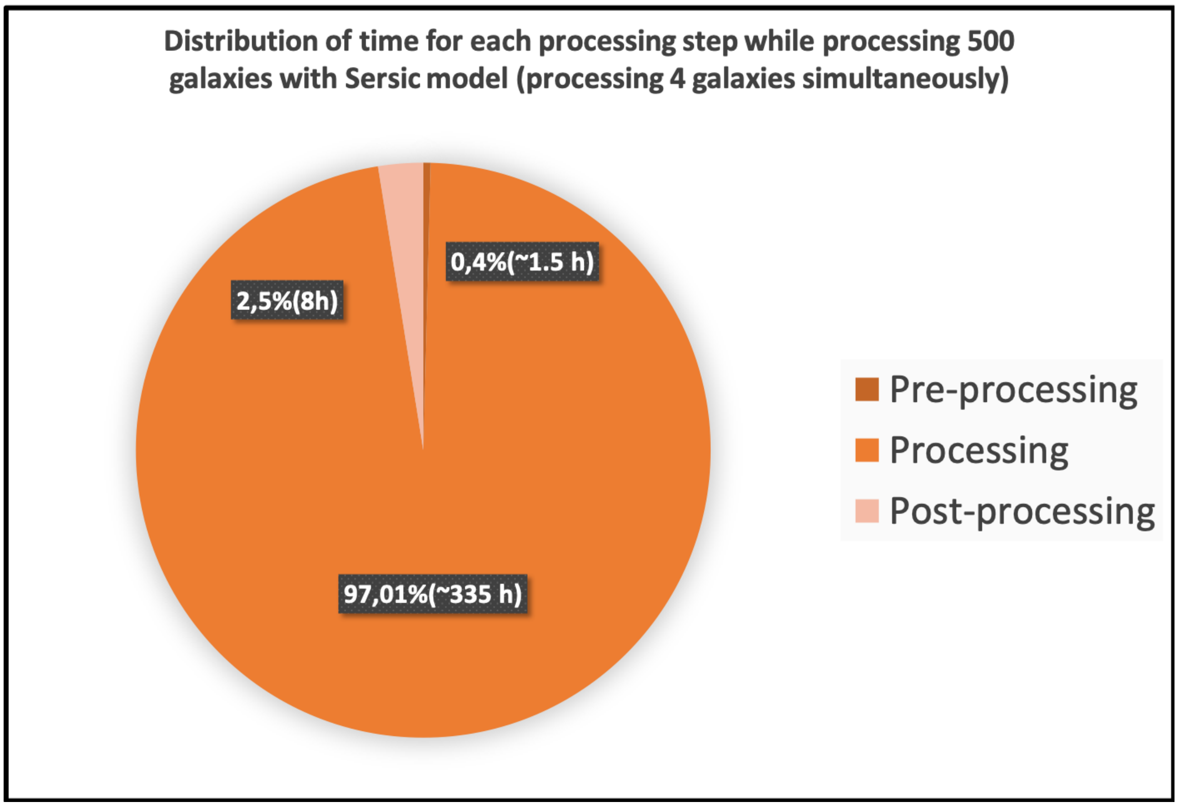
SOURCE: Author.

Figure 3.6 shows the comparison of processing of the list of 1800 galaxies with «Sérsic» and «SE» on Santos Dumont using the ICC compiler. In Figure 3.6 we can see much more promising results: while still consuming more time, the results are inside of the expected margins. Given the fact that we have our dedicated queue with 33 nodes on Santos Dumont, we can run 11 galaxies using SE model simultaneously. Our automated submission system prepares all required setup of the processing and processing could be run almost without interruption.

3.5 Parallelization

Another problem linked with performance is the time consumed for the preparation and analysis of results. As we saw in Figure 2.3, the processing pipeline roughly consists of 3 main steps: Pre-processing, Processing and Post-processing. Figure 3.7 is showing the proportion of time used by each of the steps to process a list of 500 galaxies with «Sérsic» model in the current condition.

Figure 3.7 - Demonstration of time consumption per main processing stage for 500 galaxies on Pollux machine. Even though the GALPHAT processing is responsible for 90% of total time, pre-processing and post-processing could consume a considerable amount of time when the number of galaxies comes to thousands. For this reason, optimization and parallelization needed to be done.



SOURCE: Author.

As could be seen from Figure 3.7, pre-processing and post-processing used an almost insignificant amount of time if compared with actual processing. However, in absolute terms, especially when the number of galaxies in the submitted list gets near to the thousands, even this time needs to be taken into account. Given that data (list of galaxies) is an array of independent objects, we can take advantage of many-core systems (like Xeon Phi) at our disposal to distribute this task between idle cores and gain almost linear speedup.

Before we can start any of the improvements stated above, some work needed to be done on already existing pre-processing and post-processing code. As we had commented earlier, this code is not suitable to be parallelized or improved right away. As these routines are poorly commented and organized, the first step, as it was described earlier, was to refactor and organize them. We strove to make them clearer and more flexible to be able to introduce some level of parallelization. We acquired good progress with refactoring on the main pre-processing scripts while the post-processing routines were rewritten completely in *Python*, to be more coherent with all codebase.

3.5.1 Post-processing

The first place where we started to work was the routine responsible for the compression of the posterior file (right after end of processing with GALPHAT). The problem is that the output of GALPHAT (posterior file containing the probability distribution of the parameters) is in uncompressed form that takes about 40 MB per galaxy. In our case, having a sample of 24309 galaxies that ought to be processed twice, once for «Sérsic» and once for «Sérsic+Exponential», puts us at 2 TB for the total amount. It is not that much, but moving, storing or analyzing that amount of data across the systems is tedious and ineffective. To make things better, a simple and effective way is to convert posterior files to the Flexible Image Transport System (FITS). This format generally is ten times smaller in size than an uncompressed file and does not need to be converted for further work on it because the *astropy* module in Python reads and writes it perfectly well. For the compression task we have developed a standalone script that looks to a given folder in search of uncompressed posterior files and compresses them to FITS format. It is completely parallel and scales on any number of cores. It is only being limited by the storage device speed. It has an option of deleting or leaving the original file after compression as well with automatic skipping in case of the presence of the compressed file in the folder.

In the next step, after converting the posterior distribution, the task consists in the generation of corner plot graph – the primary tool at the beginning of the elemental analysis of output. The script for this task was written from scratch as well and it

Table 3.6 - Table comparing Pollux machine processor with CPU used in S17 research on Helios cluster.

Compare Intel® Products		
	Xeon® E5-2660 v2	**Xeon Phi™ 7250**
# of Cores	10	68
# of Threads	20	272
Processor Base Frequency	2.20 GHz	1.40 GHz
Max Turbo Frequency	3.00 GHz	1.60 GHz
Cache	25 MB Intel® Smart Cache	34 MB L2 Cache
TDP	95 W	215 W
Max Memory Size	768 GB	384 GB
Memory Types	DDR3 1333/1600/1866	DDR4-2400
Max # of Memory Channels	4	6
Max Memory Bandwidth	59.7 GB/s	115.2 GB/s
Advanced Technologies		
Intel® 64 ‡	Yes	Yes
Instruction Set	64-bit	64-bit
Instruction Set Extensions	Intel® AVX	Intel® AVX-512

SOURCE: Intel (2020)[9].

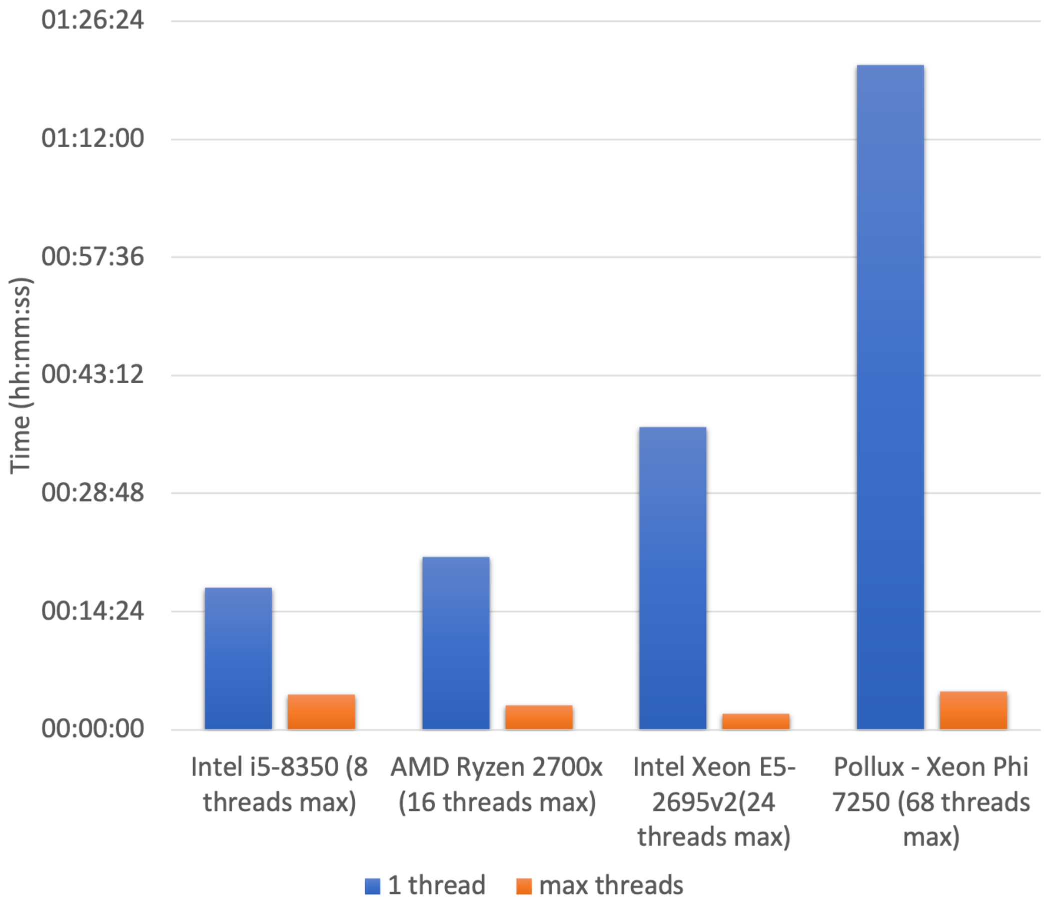
is also fully parallelized and scalable.

Again, nothing of this would be a problem if not the size of the sample. The Figure 3.8 is showing time consumption for converting and then generating corner plots for 500 galaxies at Pollux in a sequential and parallel manner.

Our choice of parallelization tool for both implementations was the *joblib*[6] library. By default *joblib.Parallel* uses the *«loky»* back-end module to start separate *Python* worker processes to execute tasks concurrently on separate CPUs. This is a reasonable default for generic *Python* programs but can induce a significant overhead as the input and output data need to be serialized in a queue for communication with the worker processes (VAROQUAUX; GRISEL,). This library enables simple and logical parallelization of embarrassingly parallel tasks. Rewriting our sequential implementations of the *toASCII.py* converter and corner plot generator rendered tremendous speedup on the Xeon Phi machine. As we were saying earlier, the «price» that Intel had to pay for such high core count was the strength of the individual cores. Brief comparison of the CPUs is shown in the Table 3.6

[6]https://joblib.readthedocs.io/en/latest/

Figure 3.8 - The figure is showing the benefits of compression routine parallelization in case of the comparison is between usage of 1 thread and maximum optimal amount on the machine. Even though it has been considerably faster across the board, the speedup on the Xeon Phi machine is the most prominent, showing the importance of usage of as many threads as possible on this processor family. Although the node with the E5-2695v2 processor showed the fastest time.

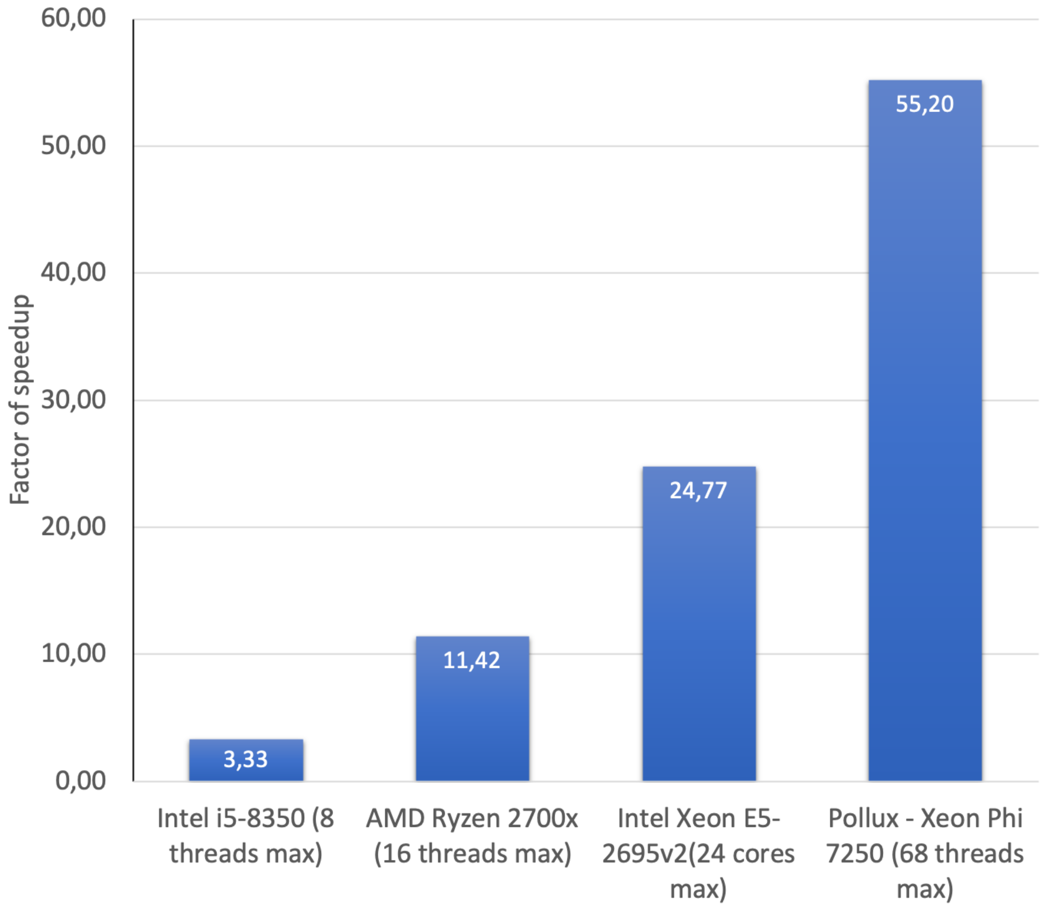


SOURCE: Author.

So, having 1.4 GHz (1.6 GHz in TurboBoost) is quite apparent the weakness of single-core performance of Pollux even in comparison with 8th generation of energy-efficient U line for notebooks (Intel i5-8350u (4 cores/8 threads@1.8GHz; 3.6GHz in Turbo-Boost)) and second generation or AMD Ryzen 2700X (8 cores/16 threads@3,7GHz; 4.3GHz in TurboBoost). However, when we switch to parallel versions of the script and run it with high enough core count, the Pollux comes out as a clear winner in relation to the speedup. The Figure 3.8 is showing a comparison of the compression of the posterior files by the mentioned machines. One more time, we need to point out that during this operation, Pollux was consuming considerably more (proportional to increase of threads used) of RAM, which is significantly more than its counterparts.

The parallel version of the corner plot generation showed similar speedup for Pollux, while also being higher even on RAM consumption.

Figure 3.9 - The figure is showing the comparison between a factor of improvement that each of the systems had while comparing sequential and parallel versions of the code for generation of the corner plot. We can see from the chart that once again, the Pollux machine had more significant improvement across the board. This data was generated by following equation: $factor = sequentilTime/parallelTime$.



SOURCE: Author.

In summary, parallelizing these two phases decreased the time of post-processing steps significantly. As a pleasant bonus, the new codebase of post-processing is highly scalable, so if it is necessary to integrate new functionality (a new type of chart), it could be done effortlessly and already in a parallel manner.

3.5.2 Pre-processing

Post-processing routines were written from scratch. It enabled more natural implementation of parallelization and preparation for the SE model integration into the processing system. In the case of pre-processing, we only have done refactoring and extensions, which did not open a clear way to parallelize them. Having this in mind, we decided that the integration of pre-processing routines into the automated submission system would do the trick. It is impossible to run a consistent benchmark, for the reason that this step is hugely dependent on the network speed and SDSS database server load, so, in this case, we will use a logical assumption that if the job is divided between several processors, it will be completed much faster.

3.6 Summary of the chapter

From the discussion above, we can see that we were able to improve the performance and usability of GALPHAT's pipeline significantly; making it more automated, faster on Xeon Phi processors family and multi-core platforms. We were able to extend its functionality with the SE model and modularize it to facilitate future extensions. A significant part of the pipeline was unified in the single system and comes with *requirements.txt* from *pip freeze* to a fast setup of a virtual python environment with all necessary packages of corresponding versions. We have gathered as well the list fo dependencies and simple instructions for building GALPHAT with GCC and ICC compilers.

4 DIAGNOSTICS AND TROUBLESHOOTING

The main goal of whole research was to process our data sample of 24309 galaxies with Sersic model. Having reached the end of the list of the galaxies to be processed with «Sérsic» model, from the total 24309 galaxies, we had about 21000 galaxies that converged and about 3309 that ended up with some problems during the processing, i. e. non-converged. To complete the processing up to the maximum amount of processed galaxies, it was necessary to investigate these galaxies to find out what went wrong and what could be done to minimize, or ideally, completely correct the galaxies that did not converge.

4.1 Steps of analysis

To remedy this situation of non-convergence, there are several actions that we can take on. We started from the least troublesome and kept going to most complex if there were still galaxies that did not converge.

4.1.1 Increasing the processing time limit

The first and the easiest thing to implement was to increase the time limit for processing that every galaxy has. The reason why we had imposed the limit is that some galaxies will not converge, no matter how long one will give them to process. This could negatively impact our processing time overall during the whole processing of the list with the «Sérsic» model. So, to optimize this point, we conducted several tests before starting the main project, to find the optimal time limitation for a given galaxy to process. We ended up with 2 hours time limit for the processing on the Pollux. It means that if the galaxy does not converge in this time amount, its job will be canceled, and the list moves on to the next one. As we reached the end of the list and had the idle machine, the first thing to do was to submit to reprocessing these galaxies with a much longer time limit. This measure resulted in dropping the non-converged galaxies number down to 800.

4.1.2 Increasing the Markov Chain number

The next step was to increase the number of Markov Chain Monte Carlo (MCMC). Typically, for «Sérsic» model, we settled upon 20 *mchains* per galaxy. This considered an optimal number, given the fact that each chain needs 1 CPU thread and about 1 GB of free RAM on our machine, so we would be able to run the optimal number of galaxies simultaneously. The general rule of thumb is that more *mchains* would provide better results while processing and potentially faster processing time for each galaxy, but not by a drastic amount. During processing, overfitting tends to cause some individual chains to get halted, and they are eventually dropped from the final analysis. This effect becomes even more pronounced with increased dimensionality, for example, «Sérsic+Exponential» model, which has 12 parameters, against 7 of «Sérsic». However, using more chains will require more CPU threads, which we do not have available. So, the main chunk of processing was done with 20 *mchains* and galaxies that did not converge were rerun with 48 mchains. By doing this, we were able to decrease to almost one half the non-converged galaxies number.

4.1.3 Tweaking processing parameters

The next option that we tried was adjustments to the internal setup of the processing algorithm. There are two parameters that can be adjusted which can provide more convergence chances, depending on the galaxy.

Our main factor in determining the convergence is the *Rhat* parameter. We generate an analysis file that contains this information after execution of the GRAnalyze routine based on the Gelman & Rubin (GELMAN; RUBIN, 1992) convergence diagnostic on the output posterior file. The result is, in case of convergence, trimmed posterior file with faulty chains and values being discarded and analysis file, which contains *Rhat* values for each of the parameters of the model. For «Sérsic», there are seven such parameters. Table 4.1 is describing them in more detail.

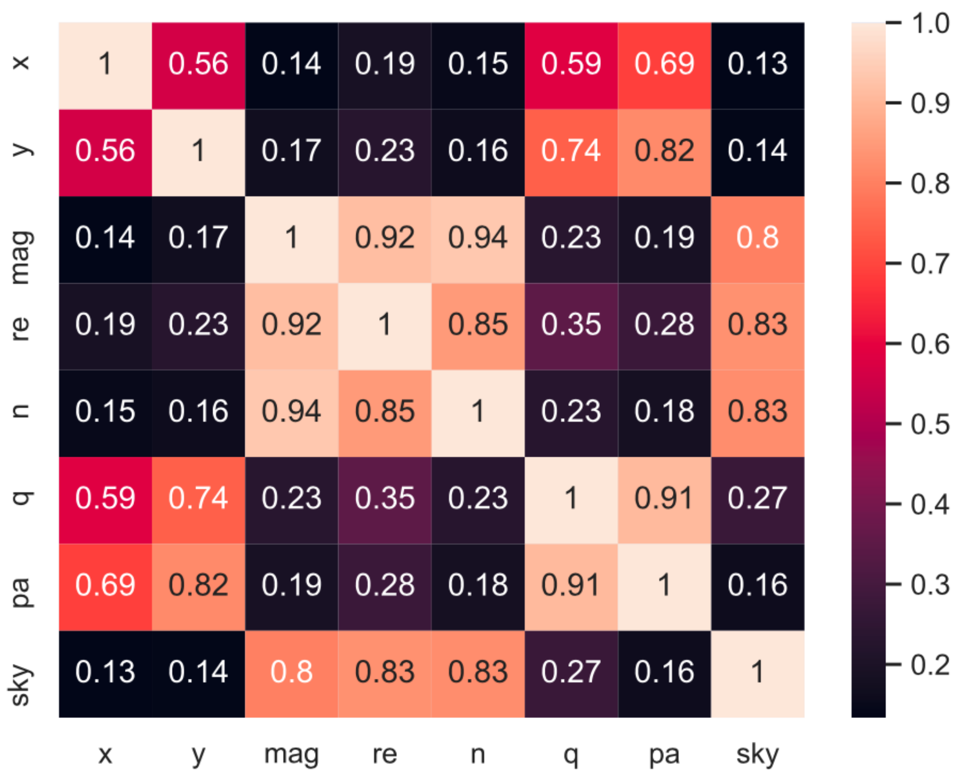
Table 4.1 - Table showing the 7 (X and Y count as one, being the center of the galaxy) parameters of the «Sérsic» model, its prior values and units. Each of these parameters has its *Rhat* value that we can look after running the posterior distribution through Gelman&Ruben algorithm.

Parameters	**Min**	**Max**	**Distribution**	**Units**
X	-3.0	+3.0	Normal ($\mu = 0.0$, $\sigma = 1.5$)	pixels
Y				
Mag	-1	+1	Normal ($\mu = 0.0$, $\sigma = 0.2$)	
Re	0.1	10	Weibul ($\kappa = 1.21$, $\Lambda = 2.5$)	pixels
n	0.5	14	Normal ($\mu = 6.0$, $\sigma = 6.0$)	
q	0.09	0.99	Uniform	
PA	-1.57	+1.57	Normal ($\mu = 0.0$, $\sigma = 0.69$)	radians
Sky	0.97	1.03	Normal ($\mu = 1.0$, $\sigma = 0.01$)	counts

SOURCE: Author.

This indicator is (up to an order-unity factor) the square root of the between-chains and within-chain variance. *Rhat=1* implies that it is of little difference in the posterior distribution between any of the chains, a necessary condition for convergence. Meanwhile, values that grow further from 1 are indicating a non-converging parameter. Gelman & Rubin recommend that max (*Rhat*)<1.2 be used to indicate convergence (where max (*Rhat*) is the maximum value of *Rhat* for any parameter). That said, using this information, we can find out which values are dominating max (*Rhat*). For this, we develop a set of routines that collect the data from the analysis files and compose a covariance matrix to help identify the culprit parameter. Below we can see the example of this matrix for the set of 400 galaxies that did not converge.

Figure 4.1 - The figure is showing the correlation matrix of the sample of 400 non-converged galaxies. From the matrix, we can perceive what variables have higher correlations between them. In our case, we can see that X, Y, q and PA are having one of the highest values of the of correlation.

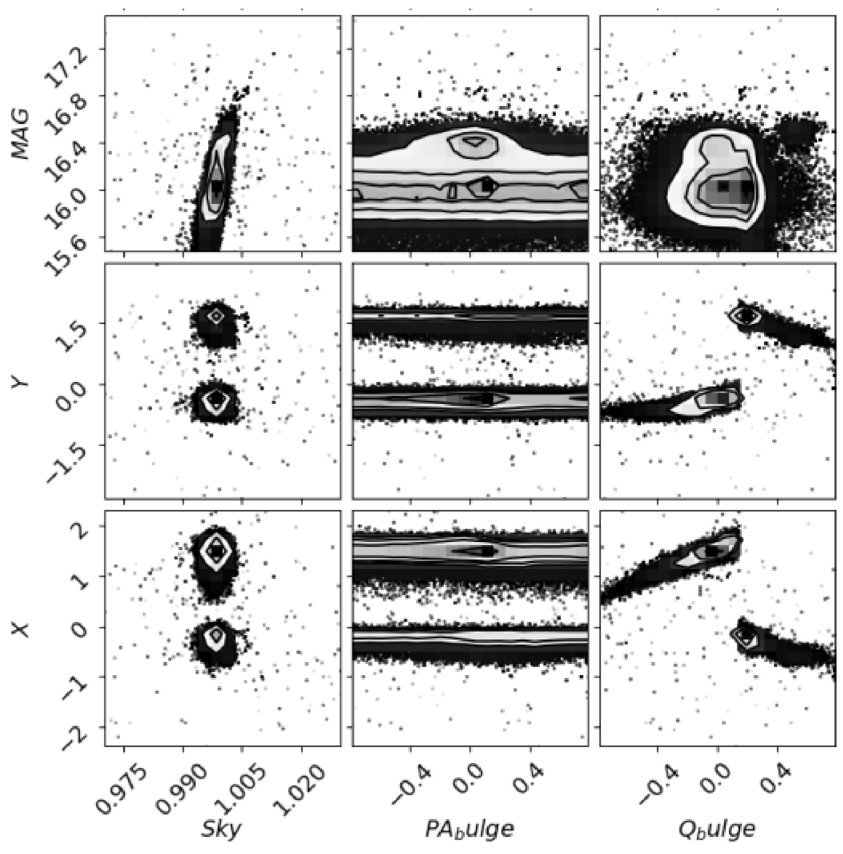


	x	y	mag	re	n	q	pa	sky
x	1	0.56	0.14	0.19	0.15	0.59	0.69	0.13
y	0.56	1	0.17	0.23	0.16	0.74	0.82	0.14
mag	0.14	0.17	1	0.92	0.94	0.23	0.19	0.8
re	0.19	0.23	0.92	1	0.85	0.35	0.28	0.83
n	0.15	0.16	0.94	0.85	1	0.23	0.18	0.83
q	0.59	0.74	0.23	0.35	0.23	1	0.91	0.27
pa	0.69	0.82	0.19	0.28	0.18	0.91	1	0.16
sky	0.13	0.14	0.8	0.83	0.83	0.27	0.16	1

SOURCE: Author.

From the matrix, we concluded that it could be a good idea to look at the variance of X and Y and perhaps the posterior marginals (corner plots) and even parameter traces for a few of these galaxies. It is possible that X and Y are not mixing well for each chain. In this case, one might decrease the gamma value (the tunable parameter for the transition probability kernel). However, making gamma too small will also slow convergence. As a result, setting *NewGamma* parameter to half of its past value rendered in convergence of about 20% of the galaxies.

Figure 4.2 - Example of the corner plot (cropped) of the galaxy with bimodality in several parameters. The effect could be seen in two-point distributions in every plotted parameter.



SOURCE: Author.

As we commented above, to further diagnose these galaxies, we can plot the corner plot of the posterior file, even if there was no convergence. Doing this, we found another problem, which was bimodality. Moreover, it is probably responsible for the large *Rhat* values. To attempt to improve the convergence rate, we can experiment with a parameter for adjusting the frequency of attempts to jump between modes, *SetJumpFreq*. The default value is set to 10. That is, every ten steps, it proposes a transition, which is the full difference between two chains, rather than gamma times the difference between two chains (currently our gamma was set to 0.06). One could try making this smaller (e.g., 5). More generally, multimodality (in this case, very well separated modes) does increase the convergence time. It is hard to get around that. There is a chance that a Hamiltonian MCMC algorithm would work better for this, but it is not implemented yet.

These tweaks helped to converge about 40% of the remaining and brought the number of non-converged down to about 200.

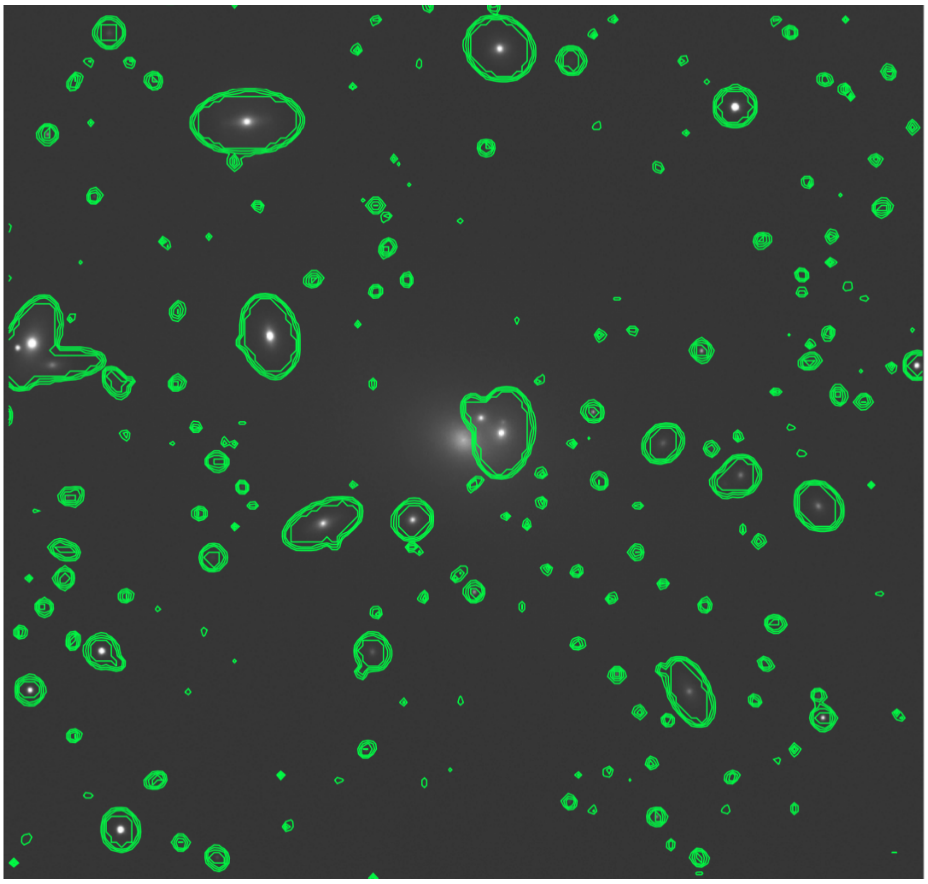
4.1.4 Processing without Math Kernel Library (MKL)

As an auxiliary option, we had our processing queue on the Santos Dumont supercomputer. The main difference is that GALPHAT on Pollux was built with MKL flags, which potentially could influence some internal operations while processing. Santos Dumont's installation was done without them. It helped a lot to process the rest of the galaxies there. This step had the most significant impact and decreased the number of non-converged galaxies to just 2.

4.1.5 Masking images manually

As a last resort, for the galaxies that ended up not converging even after all the cited steps, we can look at the actual galaxy image and try to understand what could be wrong. Sometimes, a galaxy is located in a really «crowded» area, an image of which presents several of the other objects. The problem of the «crowded» image is that our pre-processing routines could have trouble to create a precise mask for a given image. SExtractor (BERTIN; ARNOUTS, 1996) is the program that we are using to detect the areas of the image that are not the target image and generate preliminary photometric parameters. Figure 4.3 is showing a particularly tricky case.

Figure 4.3 - The figure shows the example of a complicated case, where in the center we have our target galaxy surrounded by other objects which could interfere during the processing. Green contour lines show the mask that was created by the pipeline. Pixels inside of the contours are marked as bad pixels and ignored during processing. It is clear that masked areas are far bigger than it is necessary and overlap the target galaxy in the center.



SOURCE: Author.

As we can see from Figure 4.3, the pre-processing routines created the mask that is covering much more area then it ought to. This is putting the GALPHAT through a lot of difficulties during the processing.

One of the possible solutions, in this case, is to create a manual mask for the image. For this, we are using the Ds9[1] program to create contours around all of the polluting objects and then pass this file to the little routine[2] that will create the mask. An

[1]http://ds9.si.edu/site/Home.html

[2]https://www.public.asu.edu/~rjansen/linux/mkmask_hlp.html

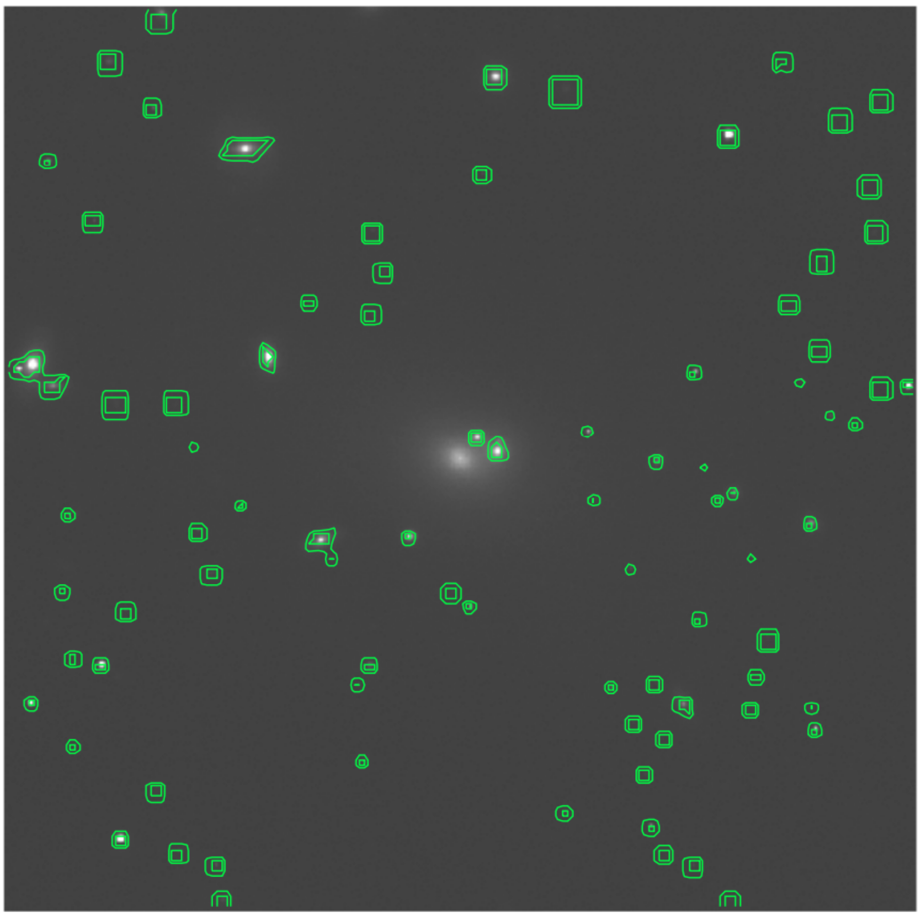
example of the calling command is below:

```
mkmask outputMaskFile.fits ncols=NUMBER_X nrows=NUMBER_Y regionsfile.reg
```

This is a short example of the command to call *mkmask* routine. It takes four arguments in our case: *outputMaskFile.fits* is the output file name, *ncols and nrows* are the sizes of the output file (must be the same as the original image) and *regionsfile.reg* is the input contours file.

As a result, we can create an extremely precise masking image that will make processing of the galaxy converge. The Figure 4.4 is showing the difference between the mask in case of the manual and automatic manner of generation.

Figure 4.4 - The figure shows the result of manual masking. If we will compare it with the automatic masking, it is clear that we lose a lot less of space around the objects.

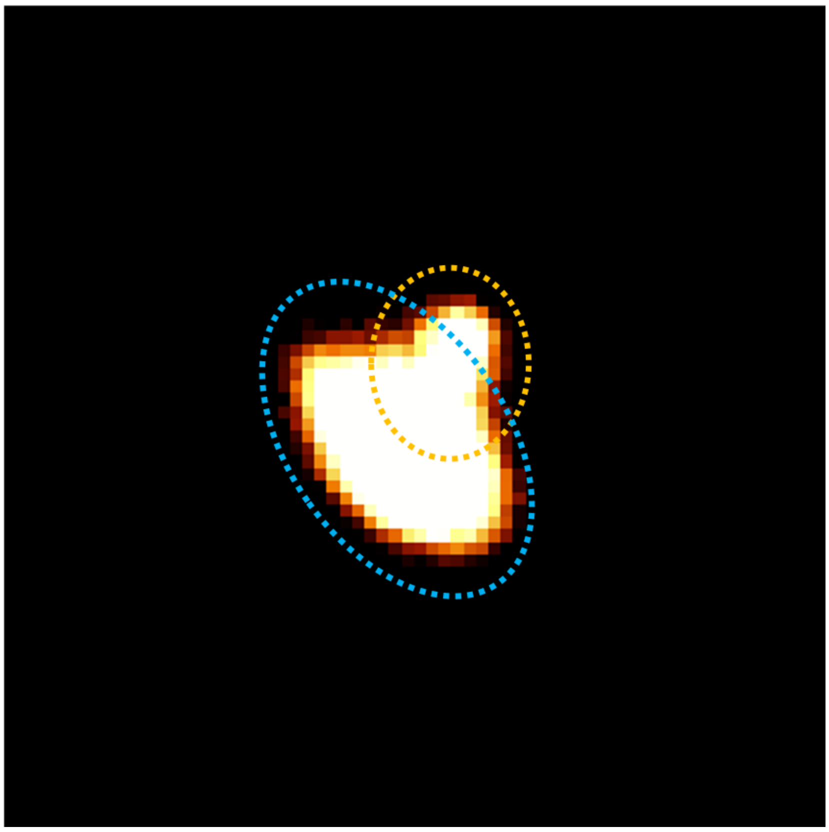


SOURCE: Author.

4.2 Results of the chapter

The conclusion of all these tweaking steps is that we ended up dropping the total number of the non-converged galaxies to just 2, having practically our entire sample being processed. One of the two galaxies that have been left is a pathological case, where the much bigger and brighter object is directly above the target galaxy. The Figure 4.5 is showing the case.

Figure 4.5 - One of two non-converged galaxies. The figure is showing a practically impossible case where a much bigger and brighter object is directly on top of the target galaxy. This creates an impossible condition to decouple it with the mask without interfering with the target galaxy. Blue contour - secondary object; Green contour - target galaxy.



SOURCE: Author.

Thus, it is not possible to mask it without interfering with the galaxy. The diagnostics on another galaxy are in progress with further attempts to make it converge.

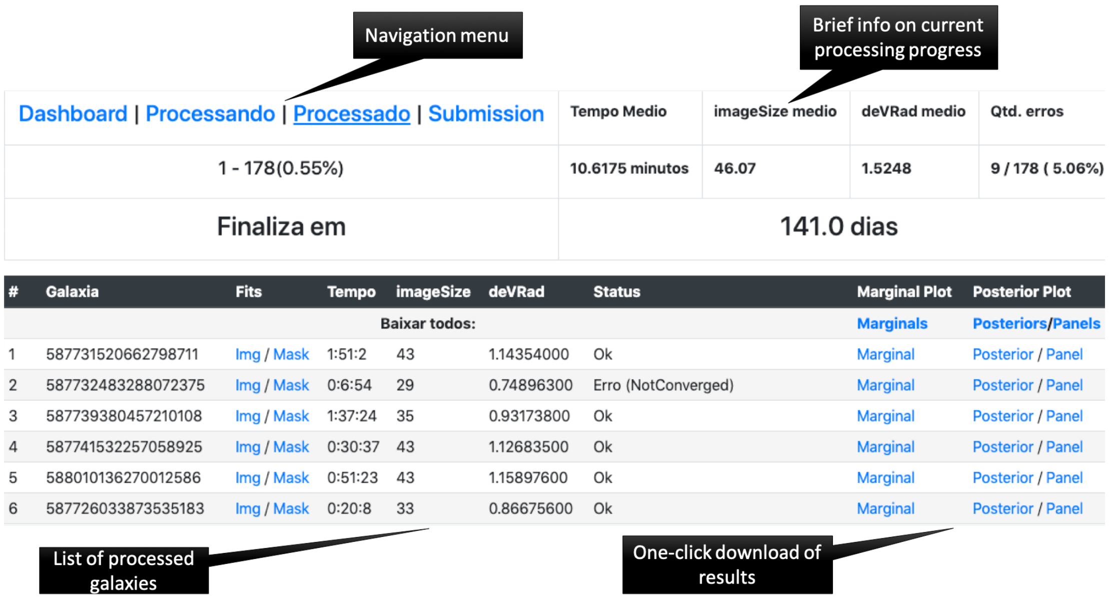
5 STORAGE AND VISUALIZATION OF PROCESSED DATA

Having a high volume of data to be processed implies not only the problem of processing time but the output data organization, storage and delivery to the end-user. It could be possible to limit oneself with some rudimentary tools by the command line when one has dozens or even hundreds of entries, but this becomes rapidly overwhelming when the count goes to dozens of thousands, separated in a multitude of folders and having relevant information all over the place. So, one could lose valuable time just for finding all essential bits, even before starting the actual research. To solve this issue, a solution was implemented in the form of a database populated with main processing results and a graphical visualization tool connected to it for easier consumption, which includes visualization and analysis

5.1 Database and storage

As stated above, initially, all our data was stored in processing folders for each galaxy, text files, logs, .fits images and posterior distributions. All this was indexed in a rudimentary way by the monitoring system that we had implemented at the very beginning of the research as a basic diagnostic tool.

Figure 5.1 - Example of the diagnostic system in its early development state with brief comments of its functionality.


Navigation menu
Brief info on current processing progress
Dashboard | Processando | Processado | Submission
List of processed galaxies
One-click download of results

Dashboard | Processando | Processado | Submission	Tempo Medio	imageSize medio	deVRad medio	Qtd. erros
1 - 178(0.55%)	10.6175 minutos	46.07	1.5248	9 / 178 (5.06%)
Finaliza em	141.0 dias			

#	Galaxia	Fits	Tempo	imageSize	deVRad	Status	Marginal Plot	Posterior Plot
			Baixar todos:				Marginals	Posteriors/Panels
1	587731520662798711	Img / Mask	1:51:2	43	1.14354000	Ok	Marginal	Posterior / Panel
2	587732483288072375	Img / Mask	0:6:54	29	0.74896300	Erro (NotConverged)	Marginal	Posterior / Panel
3	587739380457210108	Img / Mask	1:37:24	35	0.93173800	Ok	Marginal	Posterior / Panel
4	587741532257058925	Img / Mask	0:30:37	43	1.12683500	Ok	Marginal	Posterior / Panel
5	588010136270012586	Img / Mask	0:51:23	43	1.15897600	Ok	Marginal	Posterior / Panel
6	587726033873535183	Img / Mask	0:20:8	33	0.86675600	Ok	Marginal	Posterior / Panel

SOURCE: Author.

With its help, it was possible to rapidly access and consult non-converged galaxies, input and output files. Figure 5.1 shows the screenshot of the diagnostic system in its early development state. Undoubtedly it was of great help during testing and early usage of GALPHAT on Pollux. The problem was that each time a query was made, it would run through all the folders of the processing galaxies, trimming, searching and indexing them for the visual interface. As the number of galaxies was growing up, the time of access and page loading was growing as well, becoming too long to render page unuseful. However, the time of access was not our only problem. We have a high volume of data and to be able to present only necessary bits of it we need to have an organized database where it is all stored. It greatly facilitates the necessary modifications to the visualization tool. Instead of writing several new functions to search for a given file, trim it, format the information and present it, it is easier to add one more column to the database query. Having in mind these points, we already started to model the database to store all of our data, which we plan to present as a solution and continue to work on its improvement.

5.2 User interface and interaction

We cannot underestimate the value of data that we will receive at the end of processing our sample. So, to make this achievement available to the scientific community, we need to implement an graphics user interface (GUI) where we can present results in a user-friendly and comprehensive manner. To address this matter, we are developing a web-based interface that will be connected with a database containing data from processing results.

An essential aspect of this project consists in the fact that we are aiming to publish processing results to the interested community of scientists who do not have time or resources to process this amount of raw data. In our case, as stated above, we used and are still using different tools, hardware and software techniques to be able to process the SPIDER sample as fast as possible. By releasing the processing results, we want to permit that everyone have access and be able to conduct its study. It could be considered as the most significant achievement of this research at the very end.

As the starting point of development of the platform, we took the diagnostics tool that was developed earlier during this project. By incrementing the visual aspect of the interface, we were able to achieve a good deal of progress in implementing useful functions to the visualization tool and to build easy-to-use and user-friendly system to deliver necessary data to the community. As for the next step, we will improve our interface with functionality similar to that found on SkyServer Object Explorer[1], where each galaxy has its page with relevant information. In our case, we will enhance the page with our results and preliminary analysis tools, integrated into the user interface.

5.3 Composition

This user interface consists mainly of three parts: 1. Routine to collect data and populate the database. 2. Routine to make a bridge between the database and the visual interface. 3. The visual interface itself with all the necessary features.

5.3.1 Collection of data and population of data base

Having a high volume of data makes every operation on it become time-consuming. In this case, we need to swoop through the folders of processed galaxies, cut necessary bits of files and move post-processing data to destination folders. To perform these tasks, we developed routines in *Python* 3, which gives excellent compatibility with the rest of our system and has a number of ready-to-go functions to begin with.

To understand its functionality, firstly, we need to show how exactly we were processing our data. As it is already noted, our main machine for processing the «Sérsic» model is Pollux. On it we started to test the processing of several galaxies simultaneously. To keep things reasonably organized, we were submitting lists of 300-500 galaxies per instance. However, even in this case, we faced several blackouts and lagging, which interrupted the processing, forcing us to recreate whole pre-processing of the interrupted lists all over again. In the end, this resulted in more than 100 directories that contained output files of each galaxy in its due directory. So, having this in mind, collection routing needed to be able to traverse through all the directories and get the files. The routines are not parallel, and the task is limited by the speed of hard-drive operation, which was discovered during one of the diagnostics.

At the end of the traverse, the script creates *pandas DataFrame* with all required information from the directory that was scanned and moves necessary files to the target location. It is connected to the SQLite[2] database using a SQLAlchemy wrapper (MYERS; COPELAND, 2015). That is being updated with the new entries. If, in case of reprocessing, some galaxies converged, we can scan the directory again and edit the entries of galaxies that contained unconverged data.

[1]http://skyserver.sdss.org/dr7/en/tools/explore/obj.asp?id=588848899897753865
[2]https://www.sqlite.org/index.html

The choice of such simple database implementation is explained by the early stage of development, with several tests and experiments. As the system becomes more mature, we will be able to switch to something more robust, given the ease of SQLAlchemy.

5.3.2 Connection between database and visual interface

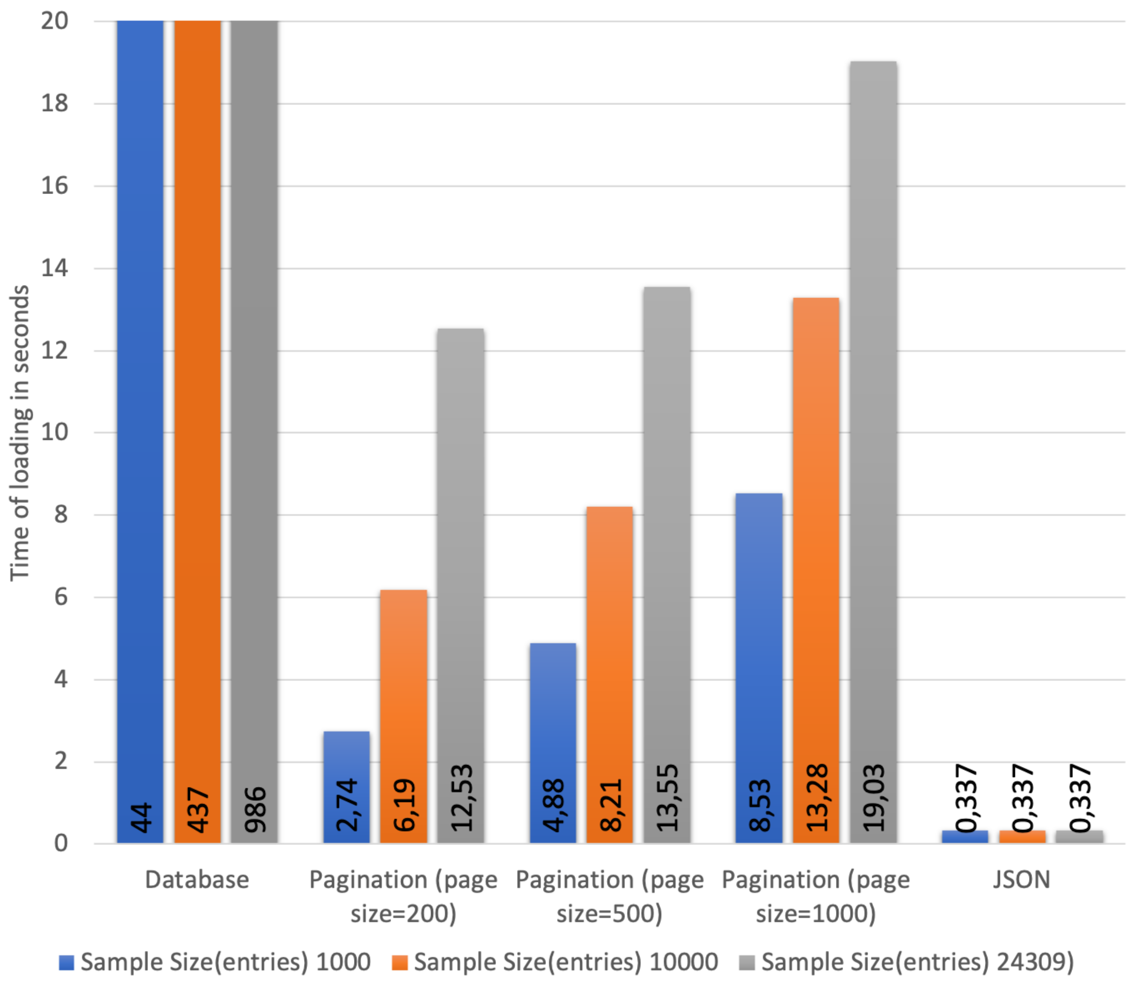
We cannot stress enough how influential in our case is the volume of the data and the problems that it is generating. In the case of visualization, it is highly troublesome to present to a remote user all data in a manner that it will not take 20-30 minutes to load in a browser. Moreover, the presentation page must have some useful features like sorting, search and fast access to crucial bits of processed data. All this imposes a list of requirements to be implemented by a system. This section treats about the speed of access to data.

The central dilemma consists in the following: we need to present to the user all of the information without overwhelming him, but at the same time giving the possibility to see all if he so wishes. In the case of an organization, pagination is the most logical choice; doing so, we offer the option to access all data by navigating pages.

Having pagination is good, but when we connect the database directly to visualization, it will load all of the entries and then split them into pages, giving no real speedup. To remedy this problem, we used the DataTables[3] framework, which permits to use *JSON* files as input and then paginates the entries in a way that next page is loaded only when the user accesses it. This gave a tremendous speedup and ease of use. The trade-off was that we needed to transfer our logic of page loading and apparel to the JavaScript and deal with some compatibility issues, which would not occur if pure *Python* were used. Below we can see the comparison between these implementation methods.

[3] `https://datatables.net`

Figure 5.2 - The figure shows the comparison between ways of supplying data to the end-user, starting with direct connection of database to the Flask logic. This is the most straightforward way, but it stops working when one has more than 200 entries. We compare it with native *Python* and SQLAlchemy pagination implementation with variable data sets and page size(number of presented entries for the user). And the third method is to leave the logic of supplying the information to the DataTables framework and JavaScript supplementary file. The last method delivers the best results overall.



SOURCE: Author.

To make this function, we needed to implement a routine that creates JSON file each time when the database is updated or when a user wishes so. This routine is written in *Python 3* as well. Its sole purpose is to convert the content of the database to the JSON dump file and auxiliary statistics file (to save the time and just read the parameters such as mean time of processing, or mean size of the galaxies, from the file and not to calculate them each time the user updates the page). Then it stores the newly created file in the directory where the main page will look for the database entry. It could be said that we could settle on the pagination method with smaller page size and it would work. We had two reasons why not to do it. Firstly, when a user would like to use functionality such as sorting or searching for the desired galaxy, the operation would be limited to the page presented; the next page is still in the database and not in the browser memory. This hugely hampers the utility of such functions and the system itself. The second reason is the fact that, while it has not the worst time of loading, it still losses greatly to the snappiness of the JSON method. With this, we can conclude that the current state of connection between the database and web logic will be able to handle the addition of the «Sérsic+Exponential» model, which is on its way and will double the number of entries.

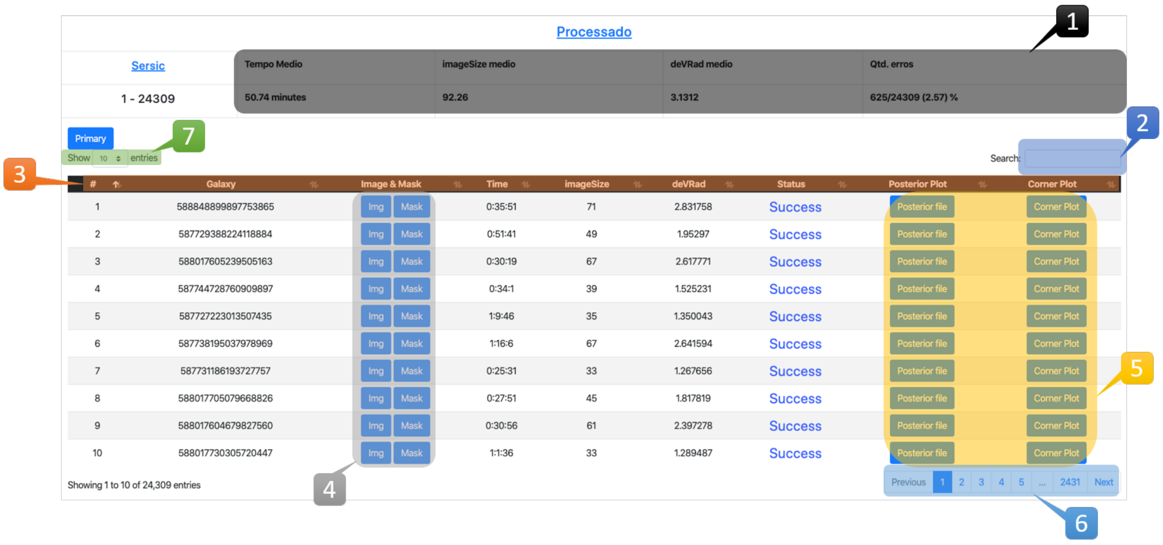
5.3.3 Visualization interface

Last but not least, we have the visual interface itself. As the tool is still in development, our main focuses for the initial versions were simplicity and core functions. In Figure 5.3, we can see the current state of the interface.

This interface was developed using Flask (*Python 3*) which is a lightweight WSGI (Web Server Gateway Interface) web application framework. It is designed to make getting started quick and easy, with the ability to scale up to complex applications. It began as a simple wrapper around Werkzeug[4] and Jinja[5] and has become one of the most popular *Python* web application frameworks (GRINBERG, 2018). DataTables framework was used for the back-end and Bootstrap framework for front-end. Using JavaScript and DataTables gives almost painless possibility to implement our priority features like sorting, searching and pagination. All of the above are already working. As Figure 5.3 is showing, we have direct links for download of posterior distribution file or corner plot, giving the results with a simple click. Currently, we are developing new functionality for our future usage scenarios. The next step is to have the possibility to select several entries for further processing.

[4]https://www.palletsprojects.com/p/werkzeug/
[5]https://jinja.palletsprojects.com/en/2.11.x/

Figure 5.3 - Example of the visualization page in the current state of the development. The numbers from the figure correspond to: 1. Brief statistics of the data based on the current JSON dump file. The data is calculated and stored in the auxiliary file for subsequent usage; 2. Search field that allows searching the whole list (not just current page) based on any column criteria (id, size or status); 3. Sorting ribbon. The whole list is sortable in any order based on any column; 4. Direct links for download of image and mask for a given galaxy. This facilitates the debugging of the galaxies that ended up with an error. Usually, a look at the image would suffice for a basic diagnostic; 5. In case the galaxy is processed successfully, the system will create buttons for download of compressed posterior and visualization of a corner plot directly in the browser. If needed, a corner plot is available for download in the preview window; 6. Pages for navigation in huge lists of data. Switching is happening almost instantly when one compares with SQLAlchemy's native implementation of pagination; 7. Number of galaxies per page.



Sersic	Tempo Medio	imageSize medio	deVRad medio	Qtd. erros
1 - 24309	50.74 minutes	92.26	3.1312	625/24309 (2.57) %

#	Galaxy	Image & Mask	Time	imageSize	deVRad	Status	Posterior Plot	Corner Plot
1	588848899897753865	Img Mask	0:35:51	71	2.831758	Success	Posterior file	Corner Plot
2	587729388224118884	Img Mask	0:51:41	49	1.95297	Success	Posterior file	Corner Plot
3	588017605239505163	Img Mask	0:30:19	67	2.617771	Success	Posterior file	Corner Plot
4	587744728760909897	Img Mask	0:34:1	39	1.525231	Success	Posterior file	Corner Plot
5	587727223013507435	Img Mask	1:9:46	35	1.350043	Success	Posterior file	Corner Plot
6	587738195037978969	Img Mask	1:16:6	67	2.641594	Success	Posterior file	Corner Plot
7	587731186193727757	Img Mask	0:25:31	33	1.267656	Success	Posterior file	Corner Plot
8	588017705079668826	Img Mask	0:27:51	45	1.817819	Success	Posterior file	Corner Plot
9	588017604679827560	Img Mask	0:30:56	61	2.397278	Success	Posterior file	Corner Plot
10	588017730305720447	Img Mask	1:1:36	33	1.289487	Success	Posterior file	Corner Plot

Showing 1 to 10 of 24,309 entries

SOURCE: Author.

6 CONCLUSIONS

We started the development of the two auxiliary systems, automation of submission process and organization via implementation and consequent visualization of generated data. Both of these systems have potential to facilitate and improve the work that will be done in the future on the processed data. For this, we need to finish the processing of the data with the «Sérsic» model (only non-converged galaxies left, they are currently in diagnostic or reprocessing) and do the same for the «Sérsic»+Exponential». The second model, as we could see from our comments in this document, is much more complex and sophisticated, which leads to more complicated and much more time-consuming processing. That processing might possibly require usage of GPU or field-programmable gate array (FPGA) to improve the processing time of this model.

We started where the previous work was done. Stalder et al. (2017, Hereafter as S17), during the Ph.D. program, performed the initial study and performance analysis of GALPHAT, starting the implementation of the automatic submission pipeline. We continued and built on that progress by improving and extending the pipeline and its modules.

Answering one of the questions of this research, is it still possible/viable to process a large amount of data locally with Bayesian Inference tools, we can give a partial and not very positive answer. Surely, having several thousands of galaxies will not make much of trouble, but as soon as a number of objects start to climb to dozens of thousands, the problem of time will take a more stark appearance. The problem will deteriorate even further if complex models (such as Sersic + Exponential) will be used. In summary, if given research does not have strict time constraints, it is possible to engage in processing large amounts of data. However, if it is necessary to show results urgently, it is advisable to look for alternatives, smaller datasets, or use a large partition of a supercomputer.

To summarize, the work conducted in this research puts us on a path to develop the automated submission system to facilitate the usage of the processing pipeline. This system is built upon the refactored, optimized, and extended code base that we inherited from S17 research. At the same time, the volume of the resulting data requires organization and a tool for its visualization, so we started to implement the first simple diagnostic and monitoring systems that evolved into the visualization and preliminary analysis tool for our data. All this together gives a complete image of the direction of our efforts.

6.1 Future work

For the future work, we have the goal of finishing the processing of the «Sérsic+Exponential» model, gather and organize all the data and start the analysis of the results, focusing on the comparison of Bayes Factor of the two models. However, before that, one ought to continue the implementation of the improvements and polishment of the processing pipeline up until the end of the processing. This will conclude the first phase of the research. Next phase will consist in detailed analysis of data and building the astronomical knowledge on it. We hope that with data acquired during the first phase we will be able to get some insights on galaxy formation and evolution inside the clusters. The tools that were developed during this research should facilitate greatly the process of analysis. As parallel task, we will invest time into improvig the GUI tools. The main chunk of these efforts must be directed to the implementation of the new functionality to our visualization system, with the hope to implement a suite of essential posterior analysis tools before delivering it to the community.

REFERENCES

ABAZAJIAN, K. N.; ADELMAN-MCCARTHY, J. K.; AGÜEROS, M. A.; ALLAM, S. S.; PRIETO, C. A.; AN, D.; ANDERSON, K. S.; ANDERSON, S. F.; ANNIS, J.; BAHCALL, N. A. The seventh data release of the sloan digital sky survey. **The Astrophysical Journal Supplement Series**, v. 182, n. 2, p. 543, 2009. 7

ABBOTT, T.; ALLAM, S.; ANDERSEN, P.; ANGUS, C.; ASOREY, J.; AVELINO, A.; AVILA, S.; BASSETT, B.; BECHTOL, K.; BERNSTEIN, G. First cosmology results using type ia supernovae from the dark energy survey: constraints on cosmological parameters. **The Astrophysical Journal Letters**, v. 872, n. 2, p. L30, 2019. 3

AGHANIM, N.; AKRAMI, Y.; ASHDOWN, M.; AUMONT, J.; BACCIGALUPI, C.; BALLARDINI, M.; BANDAY, A.; BARREIRO, R.; BARTOLO, N.; BASAK, S. Planck 2018 results. vi. cosmological parameters. **arXiv preprint arXiv:1807.06209**, 2018. 3

BARBERA, F. L.; CARVALHO, R. D.; ROSA, I. de L.; LOPES, P.; KOHL-MOREIRA, J.; CAPELATO, H. Spider–i. sample and galaxy parameters in the grizyjhk wavebands. **Monthly Notices of the Royal Astronomical Society**, v. 408, n. 3, p. 1313–1334, 2010. 3

BARCHI, P.; COSTA, F. da; SAUTTER, R.; MOURA, T.; STALDER, D.; ROSA, R.; CARVALHO, R. de. Improving galaxy morphology with machine learning. **arXiv preprint arXiv:1705.06818**, 2017. 11

BERTIN, E.; ARNOUTS, S. Sextractor: software for source extraction. **Astronomy and Astrophysics Supplement Series**, v. 117, n. 2, p. 393–404, 1996. 49

CAPPELLARI, M.; ALIGHIERI, S. di S.; CIMATTI, A.; DADDI, E.; RENZINI, A.; KURK, J.; CASSATA, P.; DICKINSON, M.; FRANCESCHINI, A.; MIGNOLI, M. Dynamical masses of early-type galaxies at z 2: are they truly superdense? **The Astrophysical Journal Letters**, v. 704, n. 1, p. L34, 2009. 3

CONSELICE, C. J. The evolution of galaxy structure over cosmic time. **Annual Review of Astronomy and Astrophysics**, v. 52, p. 291–337, 2014. 2, 4

GELMAN, A.; RUBIN, D. B. Inference from iterative simulation using multiple sequences. **Statistical Science**, v. 7, n. 4, p. 457–472, 1992. 46

GRINBERG, M. **Flask web development: developing web applications with python**. [S.l.: s.n.], 2018. 58

GUNN, J. E.; SIEGMUND, W. A.; MANNERY, E. J.; OWEN, R. E.; HULL, C. L.; LEGER, R. F.; CAREY, L. N.; KNAPP, G. R.; YORK, D. G.; BOROSKI, W. N. The 2.5 m telescope of the sloan digital sky survey. **The Astronomical Journal**, v. 131, n. 4, p. 2332, 2006. 6

JEFFREYS, H. **The theory of probability**. [S.l.: s.n.], 1998. 5, 6

KASS, R. E.; RAFTERY, A. E. Bayes factors. **Journal of the American Statistical Association**, v. 90, n. 430, p. 773–795, 1995. 4

KORMENDY, J.; BENDER, R. A proposed revision of the hubble sequence for elliptical galaxies. **The Astrophysical Journal Letters**, v. 464, n. 2, p. L119, 1996. 6, 7

LAUER, T. R.; BENDER, R.; KORMENDY, J.; ROSENFIELD, P.; GREEN, R. F. The cluster of blue stars surrounding the m31 nuclear black hole. **The Astrophysical Journal**, v. 745, n. 2, p. 121, 2012. 3

MATLOFF, N. **Parallel computing for data science: with examples in R, C++ and CUDA**. [S.l.: s.n.], 2015. 19

MYERS, J.; COPELAND, R. **Essential SQLAlchemy: mapping Python to databases**. [S.l.: s.n.], 2015. 55

SÉRSIC, J. Influence of the atmospheric and instrumental dispersion on the brightness distribution in a galaxy. **Boletin de la Asociacion Argentina de Astronomia La Plata Argentina**, v. 6, p. 41, 1963. 10

STALDER, D. **Applied computing to study structural and environmental properties of SDSS's galaxies**. Tese (Doutorado em Computação Aplicada) - Instituto Nacional de Pesquisas Espaciais, São José dos Campos, 2017. Available from: `http://urlib.net/rep/8JMKD3MGP3W34P/3NQHRKL`. 1, 5

STALDER, D.; CARVALHO, R. R. de; WEINBERG, M. D.; REMBOLD, S. B.; MOURA, T. C.; ROSA, R. R.; KATZ, N. Bayesian surface photometry analysis for early-type galaxies. **arXiv preprint arXiv:1711.02188**, 2017. 9, 10, 15, 21, 25, 30, 61

STOKES, J. Introduction to multithreading, superthreading and hyperthreading. **ArsTechnica. com**, 2002. 33

TORTORA, C.; BARBERA, F. L.; NAPOLITANO, N.; ROMANOWSKY, A.; FERRERAS, I.; CARVALHO, R. de. Systematic variation of central mass density slope in early-type galaxies. In: NAPOLITANO N. R.; LONGO, G. M. M. P. M. I. E. E. (Ed.). **The universe of digital sky surveys**. [S.l.: s.n.], 2016. p. 215–218. 11

VAROQUAUX, G.; GRISEL, O. Joblib: running python function as pipeline jobs. 2009. Available from `https://pypi.python.org/pypi/joblib#downloads`. 40

WAKEFIELD, J. **Bayesian and frequentist regression methods**. [S.l.: s.n.], 2013. 5

WEINBERG, M. D. Computing the bayes factor from a Markov Chain Monte Carlo simulation of the posterior distribution. **Bayesian Analysis**, v. 7, n. 3, p. 737–770, 2012. 9

WEINBERG, M. D.; YOON, I.; KATZ, N. A remarkably simple and accurate method for computing the bayes factor from a Markov Chain Monte Carlo simulation of the posterior distribution in high dimension. **arXiv preprint arXiv:1301.3156**, 2013. 8

YOON, I.; WEINBERG, M. D.; KATZ, N. New insights into galaxy structure from galphat–i. motivation, methodology and benchmarks for sérsic models. **Monthly Notices of the Royal Astronomical Society**, v. 414, n. 2, p. 1625–1655, 2011. 10