Parallelization and benchmarking of a Jupyter based HEP data analysis with Dask

Bachelor Thesis

presented for the degree of polyvalent Bachelor of Science (B.Sc.)
in Physics and Computer Science

Submitted by

Karl Erik Bode

Supervised by

Prof. Dr. Markus Schumacher

Albert-Ludwigs-Universität Freiburg

Faculty for Mathematics and Physics

October 2, 2023

Erklärung

Hiermit versichere ich, die eingereichte Bachelorarbeit selbständig verfasst und keine anderen als die von mir angegebenen Quellen und Hilfsmittel benutzt zu haben. Wörtlich oder inhaltlich verwendete Quellen wurden entsprechend den anerkannten Regeln wissenschaftlichen Arbeitens (lege artis) zitiert. Ich erkläre weiterhin, dass die vorliegende Arbeit noch nicht anderweitig als Bachelorarbeit eingereicht wurde.

Ort, Datum …………………………… Unterschrift ……………………

Abstract

In this thesis, the performance of an interactive analysis in High Energy Physics (HEP) is measured on multiple systems, ranging from a notebook to the local computing clusters. The basis are Jupyter Notebooks, where a PyROOT based analysis is picked as the reference. Its algorithm is changed from an event-loop to a vectorized (column based) one. The modification enables the analysis of larger datasets and decreases the computing time on datasets compared to the previous algorithm. This is made possible by using the scientific Python software stack, with Uproot, Awkward and NumPy being the most important ones. While the initial modifications yield some improvements, the introductions of Dask enables better utilization of the available resources. The single-threaded analysis is parallelized to take advantage of all available CPU cores. With Dask collections, it is possible to work with larger-than-memory datasets with only minor modifications to the algorithm. When compared to the previous algorithm, the Dask-based algorithm supports a dataset size that is eight times larger, containing 400 million events, while reducing the computing time by a factor of 2 on the largest dataset both algorithms can handle. With minimal changes, the Dask-based analysis can be scaled from the notebook to the local computing clusters. Running on the HTC cluster ATLAS-BFG and the HPC cluster NEMO, the upper limit on dataset sizes usable in an interactive analysis can be increased by an additional factor of 2.5. While the described setup enables the analysis of datasets larger than 170 GB, their structure matters. Most datasets in this thesis are monolithic, where one file contains all events. HEP datasets are generally distributed over multiple files, with sizes ranging from tens of kB to approximately 5 GB. The performance of the algorithm presented in the thesis is negatively impacted when running on multiple files, increasing the computing time sixfold on a representative dataset.
The information gained in this thesis can be used for planning future particle physics analyses for bachelor and master students. With the software stack presented in this thesis, the students can analyze large HEP datasets in short time in a familiar environment.

Zusammenfassung

In dieser Arbeit wird die Effizienz einer interaktiven Analyse in der Hochenergiephysik (HEP) auf mehreren Systemen gemessen, die von einem Notebook bis hin zu lokalen Rechenclustern reichen. Sie basiert auf Jupyter-Notebooks, wobei eine PyROOT-basierte Analyse als Referenz verwendet wird. Der Algorithmus wurde von einer ,,event-loop" auf eine vektorisierte (spaltenbasierte) Analyse umgestellt. Dies ermöglicht die Analyse größerer Datensätze und verringert die Rechenzeit für Datensätze, die mit dem vorherigen Algorithmus analysiert werden können. Ermöglicht wird dies durch die Verwendung des wissenschaftlichen Python-Software-Stacks, wobei Uproot, Awkward und NumPy zu den wichtigsten verwendeten Bibliotheken gehören. Während die anfänglichen Änderungen des Algorithmus einige Verbesserungen bringen, ermöglicht die Einführung von Dask eine bessere Nutzung der verfügbaren Ressourcen. Die ,,Single-Thread"-Analyse wird parallelisiert, um alle verfügbaren CPU-Kerne auszunutzen. Mit Dask-Sammlungen ist es möglich, mit nur geringfügigen Änderungen am Algorithmus mit Datensätzen zu arbeiten, die größer als der Arbeitsspeicher sind. Die maximale Datensatzgröße kann um den Faktor 8, auf Datensätze mit 400 Millionen Ereignissen, erhöht werden, während die Rechenzeit im Vergleich zum vorherigen Algorithmus um den Faktor 2 sinkt. Mit minimalen Änderungen kann

die Dask-basierte Analyse vom Notebook auf die lokalen Rechencluster skaliert werden. Auf dem HTC-Cluster ATLAS-BFG und dem HPC-Cluster NEMO kann die Obergrenze der in einer interaktiven Analyse nutzbaren Datensatzgrößen um das 2,5-fache erhöht werden. Die beschriebene Konfiguration ermöglicht zwar die Analyse von Datensätzen, die größer als 170 GB sind, aber ihre Struktur ist entscheidend. Die meisten Datensätze in dieser Arbeit sind monolithisch, das heißt, eine Datei enthält alle Ereignisse. HEP-Datensätze sind im Allgemeinen auf mehrere Dateien verteilt, deren Größe von einigen zehn kB bis zu etwa 5 GB reicht. Die Leistung des in dieser Arbeit vorgestellten Algorithmus wird negativ beeinflusst, wenn er auf mehreren Dateien ausgeführt wird, was dazu führt, dass sich die Rechenzeit für einen repräsentativen Datensatz versechsfacht. Die in dieser Arbeit gewonnenen Informationen können für die Planung zukünftiger Teilchenphysik-Analysen für Bachelor- und Master-Studierende verwendet werden. Mit dem in dieser Arbeit vorgestellten Software-Stack können die Studierende große HEP-Datensätze in kurzer Zeit in einer vertrauten Umgebung analysieren.

Contents

1 Introduction **1**

2 Phenomenological Background and Experimental Setup **3**
2.1 The Standard Model of Particle Physics 3
2.1.1 Particles 3
2.1.2 Interactions 3
2.2 The Higgs-boson 4
2.2.1 Production 4
2.2.2 Decay 5
2.3 Experimental Setup 5
2.3.1 LHC 5
2.3.2 The ATLAS detector 6

3 Benchmark Analysis $H \to \gamma\gamma$ **8**
3.1 Dataset 8
3.2 Hardware and Software used 8
3.2.1 bwNOTEBOOK 8
3.2.2 Software 9
3.3 Reference $H \to \gamma\gamma$ ATLAS Open Data Analysis 9
3.4 Vectorized python analysis 11
3.5 Validation 12
3.6 Performance measurements 14

4 Parallelization of Jupyter based analysis on a bwNOTEBOOK with Dask **18**
4.1 Experimental Setup 18
4.2 Adjust columnar analysis to Dask Syntax 19
4.3 Dask specific constraints on the bwNOTEBOOK 19
4.4 Validation 22
4.5 Performance measurements 23

5 Comparison between the PyROOT, ROOT and Python analyses **28**
5.1 C++ ROOT 28
5.2 PROOF 28

6 Scaling a Jupyter Analysis to the local clusters with Dask **30**
6.1 BFG 30
6.1.1 Setup and architecture of Dask on ATLAS-BFG 31
6.1.2 Analysis Validation 32
6.1.3 Dask specific constraints on the ATLAS-BFG 33
6.1.4 Performance measurements on the ATLAS-BFG 34
6.2 NEMO 39
6.2.1 Setup and architecture of Dask on NEMO 39
6.2.2 Analysis Validation 40
6.2.3 Dask specific constraints on NEMO 41
6.2.4 Performance measurements on NEMO 42
6.3 Comparison between the two clusters 44
6.3.1 BFG Slurm on NEMO 44

6.3.2 Performance measurements 45

7 Runtime Analysis on a representative Dataset **47**
7.1 The $HH \to bbll + E_{\mathrm{T}}^{\mathrm{miss}}$ Analyzis 47
7.2 Preparation of the representative dataset 47
7.3 Performance measurements 49

8 Conclusion and Outlook **51**
8.1 Conclusion 51
8.2 Outlook 53

A Supplementary Figures **54**

B Supplementary Tables **74**

C Details of the Python Packages and Versions used **85**

1 Introduction

Many aspects in modern physics benefit from computers. Some common applications are data gathering, simulations and data analysis. A multitude of tools exist for this purpose. One example is the scientific Python software stack [1]. It consists of Python and a variety of libraries as NumPy [2] for array based operations, matplotlib [3] for data visualization and multiple others. Undergraduate students in physics at the University of Freiburg are introduced to it in combination with Jupyter Notebooks [4]. Jupyter Notebooks combine an interactive shell with rich outputs like plots and text. All of this is contained in one single file with different cells for code, outputs and markdown. Python is generally single threaded with support for multithreading over different libraries. One example for this is Dask [5], a library for parallel computing. Dask offers the possibility to scale python code, both locally and on clusters. It also includes data structures to operate on data sets, which are larger than the memory of the compute resource.

In high-energy physics (HEP), the traditionally used analysis framework is the C++ based ROOT [6] package. It enables statistically correct scientific data analyzes and offers a wide range of mathematical and visualization tools. It also comes with threading and parallelization support as well as an efficient file storage. While interactive tools exist, ROOT analyses are commonly implemented in the form of C++ programs or scripts to perform the given task. Additionally, a binding between Python and C++ is offered with PyROOT [7]. This enables the utilization of ROOT objects in a Python environment. PyROOT is also usable in interactive environments like Jupyter Notebooks.

The focus of this thesis is the combination of the scientific Python software stack and Dask for interactive HEP analyzes. While ROOT is very powerful, the scientific Python software stack is much more accessible to students for aforementioned reasons. For these reasons, exploring this alternative for HEP analyses, especially for undergraduate students, is a worthwhile endeavor. To achieve this, an example analysis is ported to the scientific Python software stack. The ported analysis is then scaled with Dask. First on a notebook, then on the local computing clusters.

The data provided to study the decay of the Higgs-boson into two photons ($H \to \gamma\gamma$) in the 13 TeV ATLAS Open Data release [8] is used as an example analysis for this purpose. It is a PyROOT based analysis, published in a Jupyter notebook. The physical process covered by this analysis is the $H \to \gamma\gamma$ decay mode of the Higgs-boson. The events in the dataset are preselected to include at least 2 photon event candidates. In addition, the example analysis performs a series of selection cuts. They involve the trigger type, geometric-, energy- and momentum constraints and the number of photons. As the subsequent step, the invariant mass of the photon-pairs $m_{\gamma\gamma}$ is computed. The invariant masses are finally presented as a histogram. The algorithm of this analysis is based on an event loop. It simply iterates over all the events in the dataset. The reference analysis is single threaded.

The PyROOT based analysis is first ported to use the scientific Python software stack. The dataset is loaded with Uproot [9]. It then is stored in-memory as an Awkward Arrays [10]. The numerical functions applied to the data are provided by the NumPy library. As a contrast to the PyROOT based analysis, the algorithm is vectorized. Instead of the event loop, all operations are performed on the columns of the array. After verifying the results of this new algorithm, both algorithms are benchmarked for computing time. Both, the verification and the benchmarks are run locally on a bwNOTEBOOK, a laptop computer with 8 logical CPU cores and a RAM of 16 GB.

The NumPy based analysis is then converted to utilize Dask for parallel computations.

Again, the algorithm is verified and benchmarked on the bwNOTEBOOK. Subsequently, the algorithm is scaled out to the local compute clusters. Available for this are the high-throughput computing (HTC) cluster, the ATLAS-Black-Forrest-Grid (ATLAS-BFG)[11], and the high-performance computing (HPC) cluster NEMO [12]. After verifying the algorithm on the clusters, the algorithm is benchmarked on both clusters. In addition to the standard benchmarks, measurements comparing the performance on the two different clusters are included. Finally, a representative dataset is created, modeling the dataset of a typical HEP PhD thesis. After analyzing the file size distribution of the reference, a new dataset is generated with events from the $H \rightarrow \gamma\gamma$ dataset to match this distribution. The performance of the Dask-based algorithm analyzing it is then compared to the analysis of monolithic file containing the same number of events.

The structure of this thesis is as follows: In chapter 2 the physics process of the benchmark analyzes and the ATLAS experiment at the LHC are briefly introduced. Chapter 3 covers the original PyROOT analysis, the vectorized version, their performance and the hardware and dataset used. In chapter 4, an introduction to Dask is given, the conversion process is briefly mentioned and the performance of the converted algorithm is discussed. Chapter 6 first introduces the HTC and HPC clusters, then covers the benchmark results on both individually and compares them. In chapter 7, a typical HEP PhD thesis is introduced, with its dataset used as a reference to build the representative dataset. Then the performance of the Dask-based algorithm analyzing the representative dataset and a monolithic dataset with the same number of events are compared. Finally, in chapter 8, the conclusion and an outlook for further research are given.

2 Phenomenological Background and Experimental Setup

In this section, a brief overview about the theoretical background and the experimental setup for the physics analyses mentioned will be given.

2.1 The Standard Model of Particle Physics

The Standard Model (SM) of particle physics describes the currently known particles and their interactions. Matter consists of fermions, which interact by the exchange of gauge bosons. In the SM, three interactions are described with great accuracy: The electromagnetic interaction, the weak interaction and the strong interaction. In addition to these interactions, the Higgs field exists. It explains how particles acquire mass and predicts the existence of the Higgs-boson.

2.1.1 Particles

In total, 12 fermions and their anti-particles and 12 gauge bosons are known to exist. Some of their properties are listed in tables 2.1 and 2.2.

Table 2.1: Selected properties of the Fermions [13]

	Particle	Symbol	Charge [e]	Mass [GeV]	Interactions
Quarks	Up Quark	u	+2/3	2.16×10^{-3}	Electromagnetic Strong Weak
	Down Quark	d	-1/3	4.67×10^{-3}	
	Charm Quark	c	+2/3	1.27	
	Strange Quark	s	-1/3	93.4×10^{-3}	
	Top Quark	t	+2/3	173.21	
	Bottom Quark	b	-1/3	4.18	
Leptons	Electron	e^-	-1	0.511×10^{-3}	Electromagnetic (only e, μ, τ) Weak (All)
	Electron Neutrino	ν_e	0	$< 1.1 \times 10^{-9}$	
	Muon	μ^-	-1	0.1057	
	Muon Neutrino	ν_μ	0	$< 0.19 \times 10^{-3}$	
	Tau	τ^-	-1	1.777	
	Tau Neutrino	ν_τ	0	$< 18.2 \times 10^{-3}$	

Table 2.2: Selected properties of the gauge-bosons [13]

	Particle	Symbol	Charge [e]	Mass [GeV]	Interactions
Vector bosons	Photon	γ	0	0	Electromagnetic
	Gluon	g	0	0	Strong
	W boson	$W^\pm$	± 1	80.4	Weak
	Z boson	Z^0	0	91.2	Weak

2.1.2 Interactions

As previously mentioned, the SM describes three of the four fundamental interactions. Electromagnetism and weak interactions can be unified in the electroweak theory [14–16]. The

force carrier of electromagnetism is the photon. The force carriers of the weak interaction are the Z and $W^{\pm}$ bosons. The theory of quantum chomodynamics describes the strong interaction. The force carrier of the strong interaction are the gluons.

2.2 The Higgs-boson

The Englert-Brout-Higgs-Guralnik-Hagen-Kibble-Mechanism [17–22] allows to describe massive particles without breaking gauge invariance. It is responsible for the fermion masses as well solving the symmetry breaking problem of the mass of $W^{\pm}$ and Z^0 gauge bosons. This is achived by introducing a complex scalar field, the Higgs field.

From this field, the existence of the Higgs-boson was predicted. The Higgs-boson is predicted to be a charge-neutral spin-0 particle. The Higgs particle was discovered by the LHC in 2012, with an observed mass of approximately 125 GeV [23, 24].

2.2.1 Production

There are multiple production mechanisms for the Higgs-boson at the LHC. In the following, the most common production mechanims with proton-proton collisions will be mentioned: Gluon-gluon fusion and vector boson fusion [25].

ggF The most common production mode is gluon-gluon fusion $gg \to H$. Here the Higgs-boson couples to the gluons via a heavy quark loop. This is illustrated in figure 2.1. The production cross-section for this process is 48.52 pb [25] at 125.09 GeV. In a dataset with the integrated luminosity of $10.06\,\text{fb}^{-1}$, assuming a perfect detector with detection probability of 1, the number of Higgs-bosons produced by ggF and decaying to two photons are expected to be 1108.

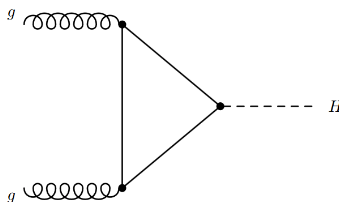


Figure 2.1: Leading order Feynman diagram of gluon fusion [26].

VBF With a cross-section of 3.78 pb at 125.09 GeV. [25], the second most common production mode is vector-boson fusion $qq\prime \to q\prime\prime q\prime\prime\prime H$. A more concrete example would be $ud \to scH$. The production cross-section for this process is 48.52 pb [25].

Here a quark from both protons in the collision is emitting a heavy vector-boson. These then fuse, producing a new Higgs-boson. This is illustrated in figure 2.2. Assuming the same dataset and detector, the number of Higgs-bosons produced by VBF and decaying to two photons are expected to be 86.

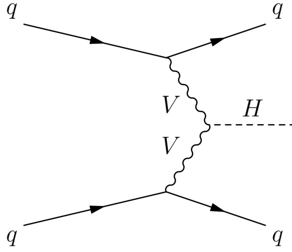
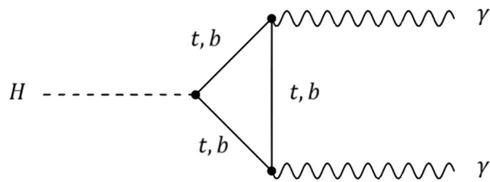


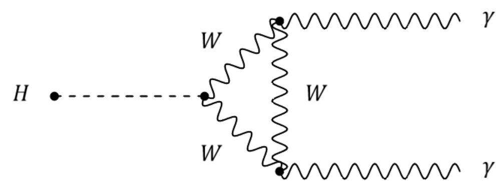
Figure 2.2: Leading order Feynman diagram of vector-boson fusion [27].

2.2.2 Decay

Many decays of a Higgs-boson are possible. Since it can not be measured directly, only the decay products are measured. The most common decay mode is $H \to f\bar{f}$, where the Higgs-boson decays to two fermions. One example for this type would be $H \to b\bar{b}$, which has a Branching Ratio of 5.824×10^{-1} at 125 GeV. [25]. This thesis is concentrating on the $H \to \gamma\gamma$ process, where the Higgs-boson decays to a photon pair. This process has a branching ratio of 2.270×10^{-3} at 125.09 GeV [25]. This is illustrated in figure 2.3.



(a) Leading order Feynman diagram of the $H \to \gamma\gamma$ process, where the Higgs-boson decays over a heavy quark loop.



(b) Leading order Feynman diagram of the $H \to \gamma\gamma$ process, where the Higgs-boson decays over a W boson loop.

Figure 2.3: Leading order Feynman diagrams of the two most common decay modes of the $H \to \gamma\gamma$ process.

2.3 Experimental Setup

The ATLAS detector is one of the four large experiments, located at the Large Hadron Collider (LHC) at CERN (Conseil européen pour la recherche nucléaire) in Geneva, Switzerland.

2.3.1 LHC

The LHC [28] is a circular particle collider, designed for proton-proton and heavy ion collisions. Two particle beams are accelerated in opposite directions through separate tubes under a high vacuum over a length of approximately 27 km. The beams consist of proton "bunches", which are collided at the four interaction points inside the main experiments. These are ALICE [29], ATLAS [30], CMS [31] and LHCb [32]. The center of mass energy of the pp-collisions are planned to be up to $\sqrt{s} = 14$ TeV. To quantify the amount of collected data, the integrated luminosity L_{int} (equation 1) is defined.

$$L_{int} = \int_{t_{\text{start}}}^{t_{\text{stop}}} \mathcal{L} dt \tag{1}$$

$$\mathcal{L} = \frac{N_b^2 n_b f_{\text{rev}} \gamma_r}{4\pi \varepsilon_n \beta_\star} F \tag{2}$$

t_{start} t_{stop} are the start and stop of the considered period. N_b is the number of particles per bunch and n_b is the number of bunches per beam. f_{rev} is the revolution frequency of the beam around the accelerator. γ_r is the relativistic gamma factor, ε_n is the normalized transverse beam emittance, $\beta_\star$ is the beta function at the collision point, and F is the geometric luminosity reduction factor [28].

2.3.2 The ATLAS detector

The ATLAS (A Toroidal LHC ApparatuS) detector is one of the two general purpose detectors at the LHC [30]. The physical dimensions are 25 m in height and 44 m in length with a weight of 7000 t. Its sub detectors are arranged in a layered structure. A schematic overview of them can be found in figure 2.4. Starting from the very center of the detector, the collision point of the protons and ions, it has a layered construction. The inner detector, containing the tracking system, is made up of silicon pixel and strip detectors and the transition radiation tracker. It is surrounded by a superconducting solenoid magnet generating a 2 T magnetic field. The inner magnet is followed by the electromagnetic calorimeter and the hadronic calorimeter. The outer toroidal magnet provides a magnetic field of up to 3.5 T. On the outside of the detector is the muon spectrometer.

For the $H \to \gamma\gamma$ process, the electromagnetic calorimeter and tracking system are the most important subdetectors. The electromagnetic calorimeter is used to measure the energy and direction of the photons. From these quantities, the four-momentum vectors of the individual photons and the invariant diphoton mass is computed. Photons and electrons have a similar signature in the electromagnetic calorimeter. To differentiate them, the tracking system is needed: Unlike an electron, a photon does not leave a track in the inner detector.

The coordinate system To specify points in space inside the detector, a right-handed coordinate system is defined originating in the interaction point of the two beams. The z-axis is defined by the beam direction, the positive x-axis points toward the center of the LHC, and the positive y-axis points upward. The azimuth angle ϕ is around the beam axis, the polar angle θ is defined as the angle from the beam axis. The pseudorapidity is defined as $\eta = -\ln(\tan(\theta/2))$.

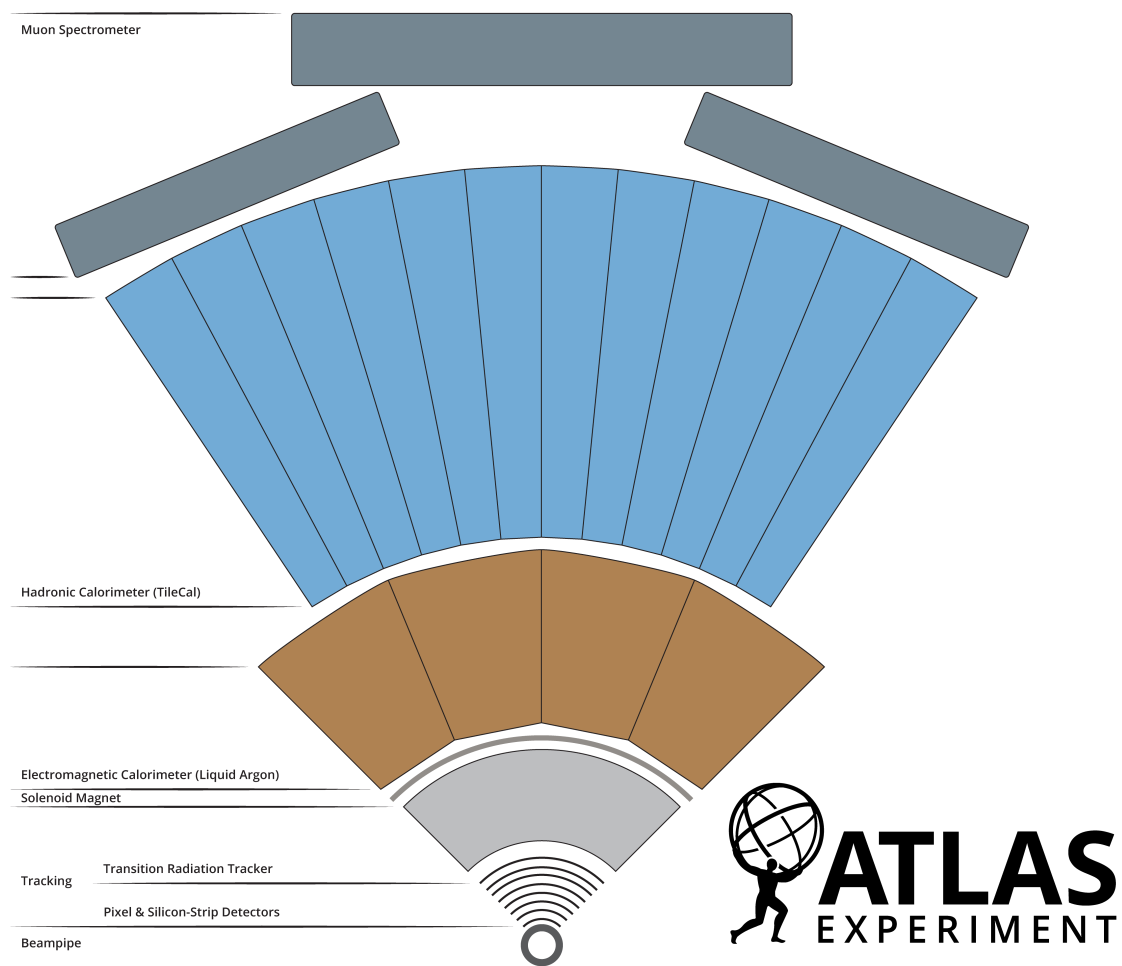


Figure 2.4: Schematic overview of the ATLAS-detector [33].

3 Benchmark Analysis $H \to \gamma\gamma$

This chapter initially introduces the original $H \to \gamma\gamma$ analysis [34], the dataset [35], the vectorized Python analyzes, the validation procedure, and the experimental setup for this part of the thesis.

3.1 Dataset

The 13 TeV ATLAS Open Data set [35] is used in this thesis. It is based on data collected by the ATLAS detector during the first four data-taking periods of proton-proton collisions in run 2 of the LHC in 2016. The data collected during these four periods is separated into the files A, B, C and D. The complete dataset contains approximately 270 million collision events from a total of 61 runs at a center-of-mass energy of $\sqrt{s} = 13\,\text{TeV}$.

The dataset corresponds to an integrated luminosity of $10.06 \pm 0.37\text{fb}^{-1}$ [36] of pp-collisions. The entire dataset is preselected with quality criteria for beam, data and detector. While multiple physical processes are analyzable on the complete dataset, for this thesis the Higgs-Gamma-Gamma subset is used. The Higgs-Gamma-Gamma dataset satisfies two additional criteria: Every event must contain more than two photons, and the leading photon must have an energy of more than 35 GeV. For this thesis, only the measured data of the Higgs-Gamma-Gamma dataset set is utilized. This includes more than 7.7 million events. As seen in sections 3.6, 4 and the subsequent sections, the Higgs-Gamma-Gamma dataset proves too small to fully utilize the available resources. After the verification part of this section, new files based on the Higgs-Gamma-Gamma dataset are generated for benchmarking purposes. While it is possible to repeatedly load parts of the Higgs-Gamma-Gamma dataset or the entire dataset multiple times for analysis, generating new files allows to freely choose the dataset's size and file distribution.
For this purpose, the first 1 million events of the second period of data-taking (stored in file B), are used to generate bigger datasets for the various benchmarks. This file is selected, because it contains over 1 million events, unlike file A of the first data-taking period. Using the files C or D for this purpose would also be possible.
Initially, a base file consisting of the first million events of file B in the Higgs-Gamma-Gamma dataset is generated. The resulting file is iteratively combined until the desired dataset size is reached, using the `hadd` program of the ROOT package. This program allows multiple files, including the repetitions of the same one, to create the larger datasets, without duplicating the source files.
Larger files are built with the largest possible smaller file. For example, a dataset containing 10 million events is constructed by chaining the 5 million events dataset two times, rather than using the dataset with 1 million events directly. This is repeated to a maximum dataset size of one billion events in one single ROOT file.

3.2 Hardware and Software used

This section introduces the notebook used for performance measurements in Chapters 3 and 4, as well as a portion of the software essential for this thesis.

3.2.1 bwNOTEBOOK

The notebook used is a bwNOTEBOOK. Its specific model is 7U14A2, manufactured by Fujitsu in the LIFEBOOK series. The CPU is an Intel i5-1145G7 processor, with four

physical CPU cores and eight logical cores. The notebook is equipped with 16 GB RAM and 512 GB disk capacity. The installed operating system is Ubuntu 22.04 LTS, 64-bit. The installed Python version is 3.10.8, the ROOT version is 6.26. A complete list of installed Python packages can be found in table C.1.

3.2.2 Software

The analysis code is written as a Jupyter Notebook, and the following paragraphs provide a brief introduction to the most important Python libraries used in this analysis.

PyROOT The Python interface of ROOT, PyROOT [7], enables the use of ROOT directly in Python code. In the context of this thesis PyROOT was only used in the reference analysis. It utilizes PyROOT for the following purposes: Loading the dataset from the ROOT files, computing the invariant mass of the selected photons based on their four-vectors, and creating the histogram containing the values of the invariant masses.

NumPy An essential part of the scientific Python software stack, NumPy [2] (Numerical Python) provides a multidimensional array object, the ndarray, and a wide range of mathematical functions ranging from linear algebra to statistics. NumPy relies heavily on precompiled code for high performance. The ndarrays and many of the mathematical functions are written in C and FORTRAN. In this thesis, the mathematical functions and ndarrays are used in every part except the reference analysis.

Awkward The Awkward library [10] contains structures designed to handle irregularly structured data. It offers a NumPy-like language to work on nested, variable-sized data of arbitrary types. The Awkward Array is important for the vectorization. It offers an abstraction layer to ignore e.g. the irregular numbers of photons, ranging from two to four, in each event while making it possible to access information on each of them. This makes them a vital part of the selection process.

Uproot To import ROOT files within pure python and without a complete ROOT installation, Uproot [9] is needed. The contents of the files can be directly loaded into an array. For this thesis, Uproots functionality is combined with Awkward Arrays. Every part of the thesis, except the reference analysis, utilizes Uproot to load the dataset.

Jupyter Notebook The document format Jupyter Notebook [4] combines code, rich outputs and markdown into one file. They are organized into cells of different types, such as code, markdown and outputs. In general, the user interacts with a Jupyter Notebook via a web interface. Many interpretation kernels are available for Jupyter Notebooks, like C++ or Python. In this thesis, the IPython [37] kernel is used.

3.3 Reference $H \to \gamma\gamma$ ATLAS Open Data Analysis

In this section, the reference $H \to \gamma\gamma$ analysis, and the modifications made to it, are discussed.

The reference version of the analysis [38], is a part of the ATLAS outreach repository [34]. The $H \to \gamma\gamma$ channel is one of the decay channels contributing to the discovery of the Higgs-Boson. On a physics level, the analysis first selects photon pairs based on their

trigger, momentum, pseudorapidity, and isolation. The trigger refers to the diphoton trigger, requiring two distinct energy clusters in the electromagnetic calorimeter with energies above $E_{\mathrm{T}} > 35\,\mathrm{GeV}$ and $E_{\mathrm{T}} > 25\,\mathrm{GeV}$ for the leading and sub-leading photons and loose isolation criteria based on track and calorimeter information [36]. In the notebook of the reference analysis, the only criteria constraining the energy of the photons is $p_{\mathrm{T}} > 25\,\mathrm{GeV}$ which for massless photons is equivalent to a selection constraining energy. This is because the diphoton trigger already selects for the energy of the leading photon. The pseudorapidity constraints are necessary to restrict the photons to the fiducial region of $|\eta| < 2.37$ and to avoid the region between the barrel and end caps at $1.37 \geq |\eta| \geq 1.52$. After these selections, only events containing photon pairs are selected. In the isolation selection photons with less than 6.5% of the energy detected outside an area of $\Delta\mathrm{R} = 0.2$ and less than 6.5% of the transverse momentum is outside an area of $\Delta\mathrm{R} = 0.3$. Where $\Delta\mathrm{R}$ is defined as follows: $\Delta\mathrm{R} = \sqrt{\Delta\eta^2 + \Delta\phi^2}$. This is needed to differentiate between photons of the $H \to \gamma\gamma$ decay and other particles. In the detector, many pp-collisions happen simultaneously or in shortly after each other, all producing particles. If a second particle were to hit the detector near one of the photons produced the $H \to \gamma\gamma$ decay, it can influence the measurement by depositing its energy in the detector close to the photon. Only photons clearly separated from other particles are selected to reduce the influence of these background particles. To summarize, the events selected as final diphoton candidates all satisfy the diphoton trigger, have an energy of more than 25 GeV, are within $|\eta| < 2.37$ excluding $1.37 \geq |\eta| \geq 1.52$. Furthermore, each event must contain exactly two photons, both satisfying the aforementioned criteria and are additionally isolated in E_{T} and p_{T}.

The display of one event, which satisfies all these conditions, is illustrated in figure 3.1. For all selected pairs, the invariant mass is computed. The invariant masses are stored in a histogram. As provided, the Jupyter notebook is set up to download a part of the dataset from the internet, run the analysis and display the results in a histogram. The expected shape of the histogram is a small bump at 125 GeV on an exponential, smoothly falling background.

The algorithm of the analysis is based on an event loop, iterating over all events. PyROOT is used for loading the file and computing the invariant mass. The cuts are implemented as normal Python statements.

For this thesis, the original notebook is modified. Instead of downloading the dataset, local storage is accessed. Additionally, the whole dataset is analyzed instead of a small part of it. Time measurements are also added to gather the runtime of each analysis.

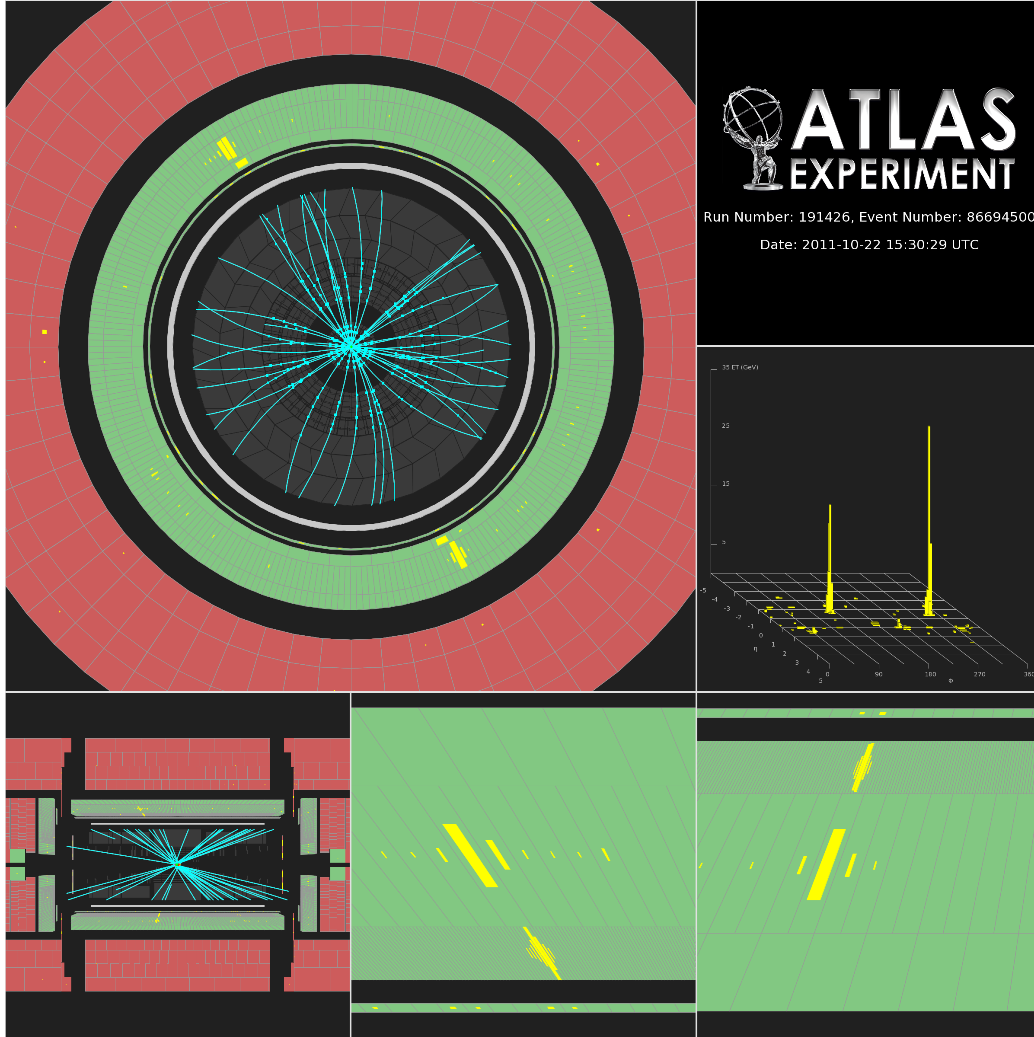


Figure 3.1: This figure contains an event display from the ATLAS detector of a diphoton candidate [39]. The electromagnetic calorimeter is colored green in all parts of the picture. The upper left picture shows a radial cross-section of the detector. While many particles are detected in the inner part of the detector, the two photons are clearly separated in the electromagnetic calorimeter. A side view of the detector is in the lower left corner of the figure. Next to it are the sections of the calorimeter where the photons are detected. Yellow bars represent the detected energy in the sub parts of the detector. Above these is a diagram displaying the detected energy in the calorimeter in the $\eta - \phi -$ plame, showing the clear separation of both photons.

3.4 Vectorized python analysis

For the initial experiment, the PyROOT based reference analysis is ported to pure Python. The new analysis has two major differences to the reference analysis. It is vectorized, the main loop is replaced by a column based algorithm. Since PyROOT is not used, Uproot is needed to load the dataset. Due to the irregular shape of the dataset, extra considerations must be given to the data structure used for storage. The irregular shape of the imported data is caused by the variable number of photons detected in each event, ranging from

two to four. From a data perspective, this causes fluctuating sizes in some columns of the dataset. While it is possible to store the events as-is in a rectangular NumPy array, the variables would be converted to a different type to ensure the same dimensions of all columns, increasing the processing difficulty. On an array like this, many of the functions the vectorized analysis relies on are not directly usable. Alternatively, zero padding could be used to insure the same dimensions over all columns of the dataset. This method has two significant drawbacks, necessitating computationally expensive preprocessing and an increase in the memory needed. Awkward Arrays allow analyzing the dataset as if it were rectangular, without the need for preprocessing. This dedicated loading step in the algorithm ensures that the entire dataset is loaded into system memory at the beginning of the calculations, rather than being loaded gradually. Additionally, NumPy based functions are implemented for the computation of the invariant diphoton mass and to build and fill the histogram. From a physics perspective, both analyzes are virtually identical. The results of both are compared in section 3.5. To work on a dataset consisting of multiple files two strategies are explored.

Classic loading The simplest method is to work on the files sequentially. With this algorithm, each file is processed on its own, saving the results for each file in memory by appending them to the same array. This is needed for the validation section, since the reference dataset consists of 4 files. In section 3.6, the appending step is not necessary. Instead, larger files are generated for the use with this algorithm.

In-Memory appending An alternative approach is to append the datasets into one array before analyzing them. For the monolithic datasets in section 3.6, the algorithm is modified. Instead of loading the large monolithic files, the 1 million events file is loaded repeatedly to gather the required number of events for a given benchmark measurement.

3.5 Validation

To validate the computation of the new algorithm, their respective results are compared in two ways. First, the cut flows of both algorithms are compared with each other. For the PyROOT and NumPy analyzes, it can be found in figure 3.2. The individual cuts are numbered from zero to four. Cut number zero is the number of events in the entire dataset. The first cut requires that the photon trigger has fired. Cut number two requires two photons with certain kinematic properties. The third cut requires that the leading photon fulfills the isolation requirement. The fourth cut requires that the sub-leading photon also fulfills the isolation requirement. These are the events used for the evaluation of the invariant diphoton mass. Comparing the cut flows of both algorithms, the number of events matches for all but one cut. In the NumPy based analysis, cut three and four are combined into one. This is due to the vectorized algorithm. The events after cut three, which contain photon pairs, are analyzed in separate steps in the PyROOT based algorithm. In the NumPy based one, both are analyzed in the same step.

The second verification is a direct comparison between the results of both algorithms. The histograms of the invariant diphoton masses are compared directly with each other, and their per-bin differences are computed. For the PyROOT and NumPy algorithms, this is visualized in figure 3.3. The total number of events is identical, but two events shift between bins. In conclusion: The results of the two algorithms are in good agreement. The differences between both algorithms are below 0.1 % for every bin. These fluctuations can

be explained by differences in floating point precision, shifting single events from one bin to the next.

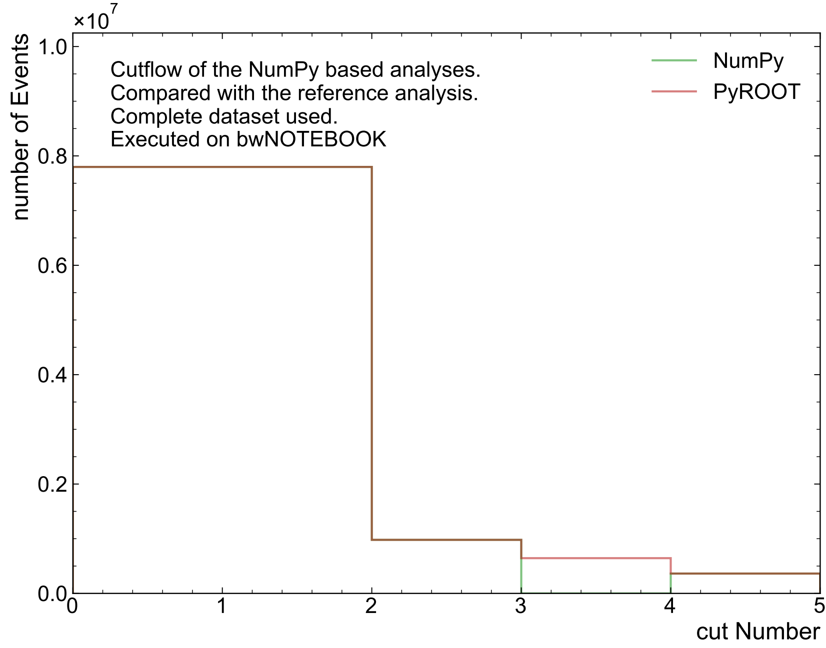


Figure 3.2: Cut flow comparing the PyROOT and NumPy based analysis. The exact numbers can be found in table 3.1 or table B.1

Table 3.1: Table listing the sequential number of events retained after each cut.

		PyROOT	NumPy
Cut 0	All events in the dataset	7798424	7798424
Cut 1	Photon Trigger	7798424	7798424
Cut 2	($\|p_{\mathrm{T}}\| > 25$GeV and $\|\eta\| < 2.37$) excluding ($\|\eta\| < 1.37$ or $1.52 < \|\eta\|$) and $N_{\gamma} = 2$	979404	979404
Cut 3	Leading photon: Isolation requirements for E_{T} and p_{T} met	644574	–
Cut 4	Sub-leading photon: Isolation requirements for E_{T} and p_{T} met	362972	362972

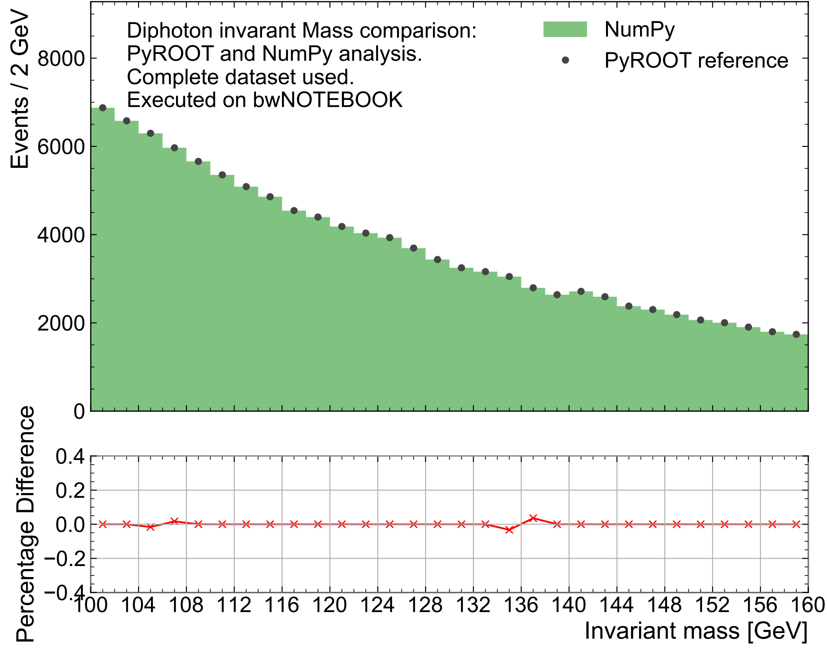


Figure 3.3: Validation of the vectorized Python analysis against the PyROOT Analysis on the original dataset. Shown is the spectrum of the invariant diphoton mass. The lower panel shows the percentage differences between the two measurements.

3.6 Performance measurements

In this section, the performance of the classic (sequentially) loading and the in-memory appending NumPy-based algorithms and the PyROOT based algorithm is determined and compared.

The measurements here compare the computing and loading times on the bwNOTEBOOK for the three algorithms on different sized datasets. Starting from 1 million events, the datasets were scaled until the Jupyter notebook running the algorithm crashed.

To ensure comparability, the same setup is used for each measurement. This setup includes the bwNOTEBOOK, which is only connected to the charger with no other peripherals. Additionally, one terminal is used in which the Jupyter Notebook server is running, along with one Firefox window that has two tabs open: the running notebook and the Jupyter tab to open the notebook.

Throughout the thesis ten measurements are performed for each set-up. As the result the mean value of these ten measurements and the estimated uncertainty of the mean value are quantified. To keep the setup as consistent as possible, a new Jupyter kernel is used for each measurement. This was automated with the library Papermill [40].

In figure 3.4, the loading and computing times of the classical loading algorithm are shown. The ratio of compute and loading time is similar for increasing dataset sizes. The runtimes of this algorithm roughly follow a shallow parabola for increased dataset sizes. For increases in dataset size, the required computing power increases faster than linearly, but not significantly. As seen in figure 3.5, this changes for the algorithm with in-memory appending of the input files. Here, the ratio between loading and computing time is changing with growing datasets. While the computing times stay comparable to the classical loading

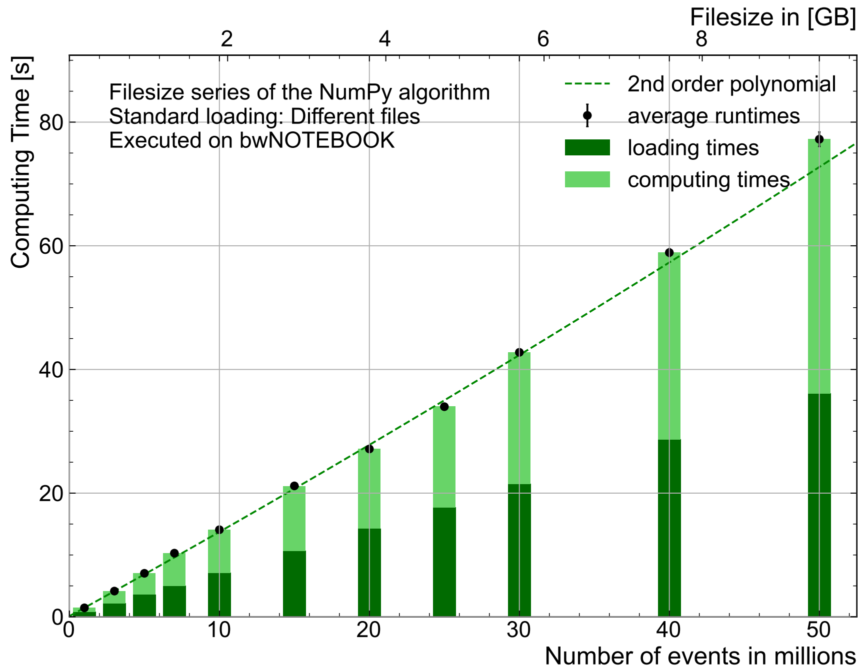


Figure 3.4: Loading and compute times of the NumPy-based algorithm with classical loading. The loading and compute times are comparable in length for each dataset. The numerical values are shown in table B.3

algorithm, the loading times increase greatly. The cause for this increase is the concatenation of the repeatedly loaded file. The computing times of the in-memory appending algorithm also follow a second order polynomial. The PyROOT algorithm behaves differently, as shown in figure 3.6. In the figure, the loading time is not visible. Only in table B.5 a short and consistent loading time can be found. This is because the PyROOT based algorithm does not split the execution of the algorithm in loading and computation, but mixes them. The three major differences to the NumPy-based algorithms are the huge increase in computing time, the nonexistent loading time and the decreased maximal dataset size.

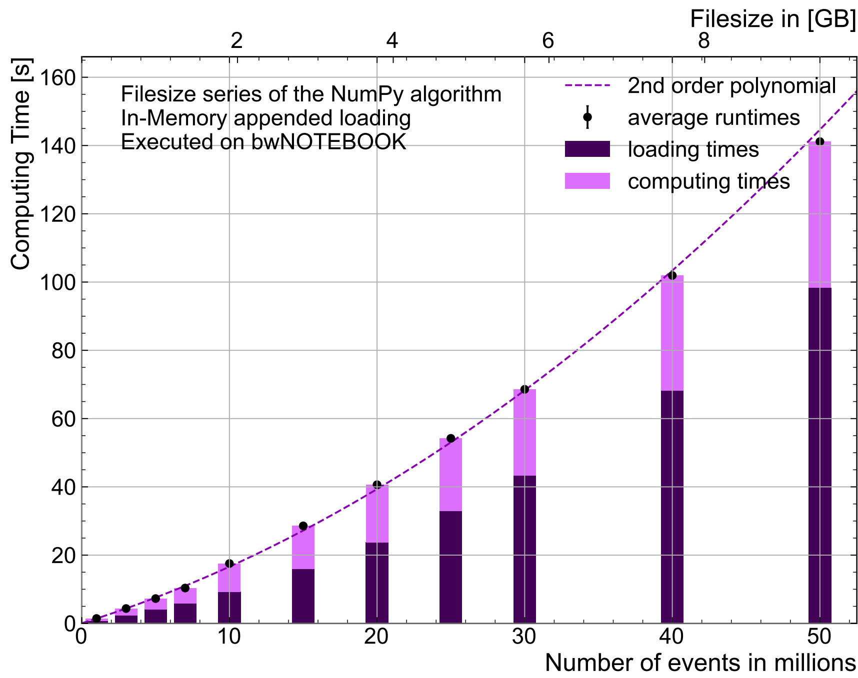


Figure 3.5: Loading and compute times of the NumPy-based algorithm with in-memory-appending based loading. The numerical values are shown in table B.4

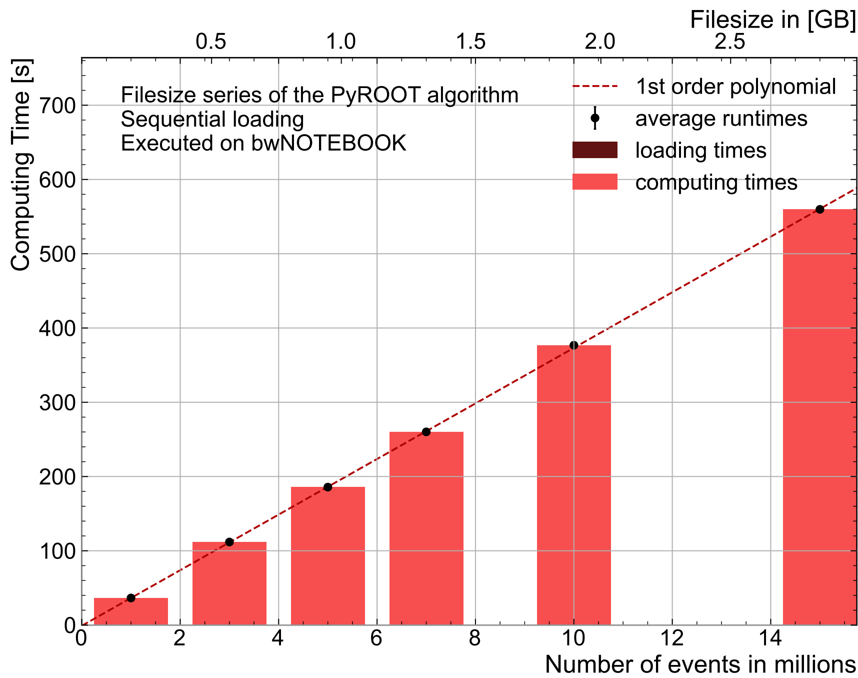


Figure 3.6: Loading and compute times of the PyROOT based algorithm. This algorithm is not split into separate loading and compute phases. The numerical values are shown in table B.5

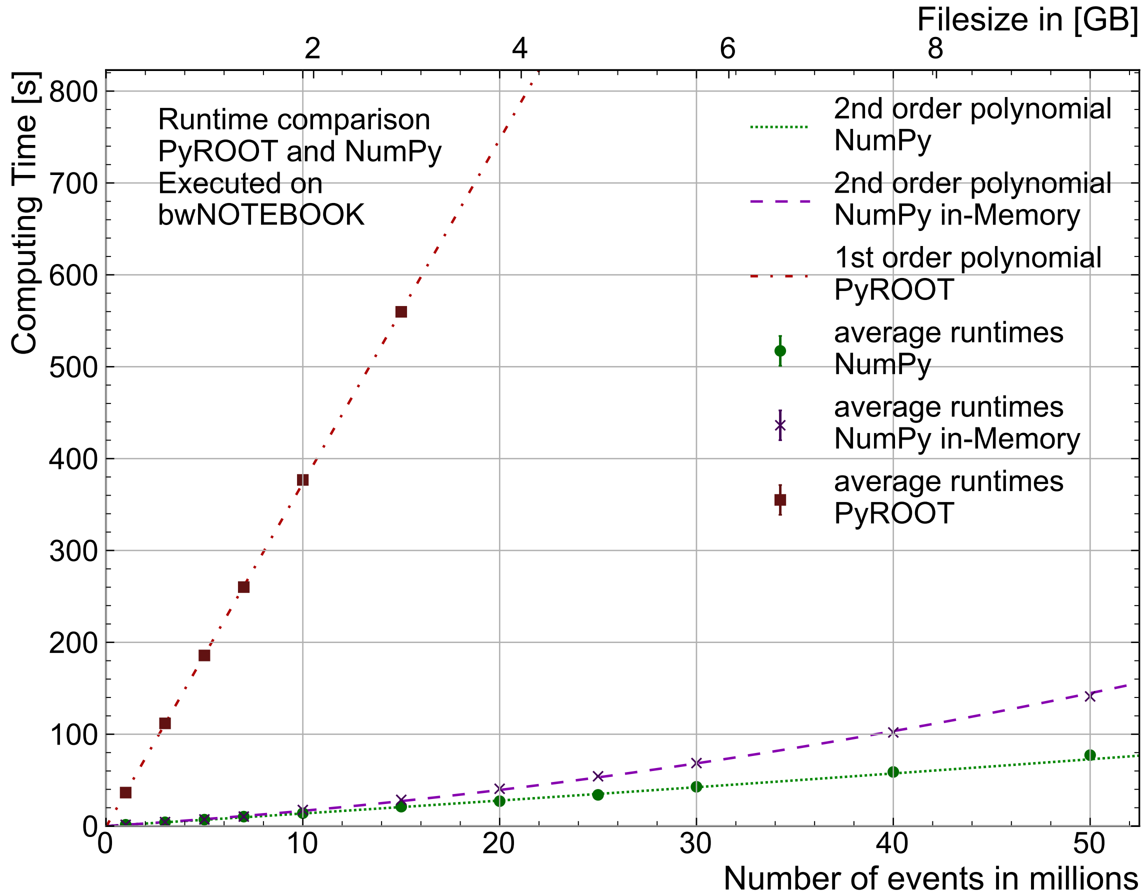


Figure 3.7: Comparison of the computing time of the PyROOT and the two NumPy-based algorithms. The numerical values are shown in table B.2

All three algorithms are compared in figure 3.7, which omits the loading times. Here the differences between the reference PyROOT analysis and the two NumPy based ones are clearly visible. The PyROOT based algorithm is a lot slower compared to the other two. It also can not work on dataset with more than 15 million data points. One possible explanation for this could be a memory leak in the reference analysis. The memory usage of the Jupyter notebook running this analysis is steadily increasing over the runtime of the analysis. Both NumPy-based algorithms perform similar, but the in-memory appending of the dataset has an increasing impact on performance. The difference in computing time between the NumPy and PyROOT based algorithms can have multiple reasons. NumPy supports SMID, vectorized instructions, which enable modern CPUs to compute multiple simple calculations at the same time. A bigger influence on performance could be the memory structure used by NumPy. The entire dataset is stored in one contiguous part of system memory.

4 Parallelization of Jupyter based analysis on a bwNOTEBOOK with Dask

This section covers the parallelization of the NumPy-based algorithm with Dask. Initially, a short introduction to Dask is given and the conversion process from pure NumPy and Awkward Arrays to Dask equivalents is discussed. This is followed by the verification of the Dask-based algorithm on the notebook and the performance measurements.

4.1 Experimental Setup

In this section, changes to the software running on the bwNOTEBOOK are made. By introducing Dask [5], multiple aspects of the algorithm, as defined in the previous section, are updated. Dask is a parallel computing library designed for interactive computing. An overview of Dask is given in figure 4.1. It offers a wide compatibility to other Python libraries and offers out of the box diagnostics in form of a dashboard. Dask has three major components: Collections, the task graph and scheduler.

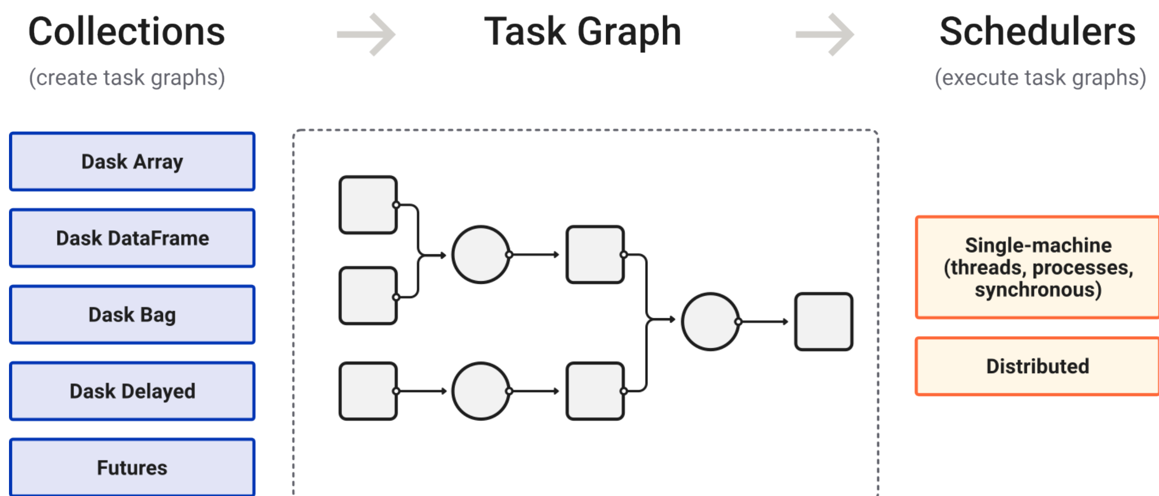


Figure 4.1: This figure shows an overview of the main components of Dask[41]. Dask collections are used to create the task graph. The scheduler executes the task graph. Multiple collections and schedulers are possible.

Dask collections are designed to be similar to other Python data collections, such as NumPy array and Awkward Arrays. They are lazily executed. Operations on them do not result in immediate calculations, but additions to the task graph. Dask collections make it possible, to work with larger than memory datasets. This is achieved by partitioning the dataset into chunks. In case of the Awkward Arrays used in thesis, this means grouping a certain amount of events together, forming a sub-array. Awkward Arrays in Dask are enabled by a separate library, dask-awkward [42]. Usually, the chunks are hidden by abstraction, resulting in less overhead for the user. The chunk size can be adjusted by the user, either to rows or columns of the dataset or to memory sizes. This chunk size is a variable influencing the performance of the analysis. More details can be found in section 4.5.

Dask Scheduling The basis of Dask's scheduling is the task graph. All operations with either Dask collections or APIs result in nodes on the task graph. A task graph is a directed acyclic graph, where the nodes represent computation steps, which are connected via their dependencies. The task graph is lazily evaluated. This means, that instructions are not directly evaluated as they normally would be, but instead added to the graph. When the execution is started, the scheduler of Dask begins to assign the computation steps to workers, parallelizing whenever possible.

Workers The computations initiated with Dask are executed by the workers. Workers can be processes on the same machine, using the single-machine scheduler, or entirely different machines in a cluster with the distributed scheduler. This section covers local workers, running on the same machine as the scheduler. While easier to use, they have a significant drawback: Sharing the system resources with the scheduler. Workers introduce many parameters, this thesis measures the influence of the number of them in figure 4.7 in section 4.5. Some insight into the effects of allocated memory in combination with increasing chunk sizes can be found in figure 6.6 of section 6.1.4.

Dashboard For feedback and diagnostics, Dask offers a dashboard. This can be accessed with the browser or from within Jupyter lab as separate tabs. The displayed information contains, among others, the progress of the computation as well as the worker memory. In section 4.3, the information from the dashboard is used to aid scaling the analysis.

4.2 Adjust columnar analysis to Dask Syntax

Dask is built to be mostly compatible with other libraries, such as NumPy and Uproot. It is possible, to directly load the ROOT files to a dask-awkward array. This simplifies the conversion greatly. For this thesis, the conversion is mostly a matter of replacing the library for function calls, as well as triggering the computation. This is mostly, because the NumPy algorithm is designed for such modifications. For example, the calculations are all performed by discrete function calls, not by $+$ or $-$. While converting the code is straightforward, finding initial values for Dask specific parameters is more difficult. More details on this are given in section 4.3.

4.3 Dask specific constraints on the bwNOTEBOOK

In the scope of this thesis, Dask specific parameters are mostly chunk size, the number of workers and the memory allocated to them. Both chunk size and worker memory need to be set to reasonable values for a given dataset size, that the algorithm can work stable. Working combinations of both are interdependent, one chunk size will work for one memory size, but not necessarily for other sizes or larger datasets.

A starting point are the best practice recommendations by the developers [43]. They state, that chunk sizes between 100 MB and 1 GB are safe in most cases. Workers also need to be able to keep multiple chunks in-memory at the same time, at least two or three according to the best practice guidelines. From this, the standard value for the chunk size is set to approximately 200 MB, or one million events. According to the aforementioned recommendations, the RAM should be 400 to 600 MB. This results in instable performance, require more memory to be allocated. With the limitations of the device in mind, a maximum of 2 GB RAM is allocated to the 6 workers. This saves resources for the scheduler,

monitoring and operating system to keep the machine responsive. The chunk size also influences the task graph, since most operations scale directly with the total number of chunks. The number of chunks should stay above the number of workers available, to fully utilize the CPU resources. While smaller chunks mean less burden for the workers, they also increase the size of the task graph by splitting the dataset in more parts. This increases the load in the scheduler, resulting in delays before the workers begin after the computation is triggered and high RAM usage by the Dask scheduler. An upper bound for the number of chunks is suggested between 10 000 and 100 000. During this thesis, even chunk numbers of more than 1 000 performed poorly, with more than 30 GB RAM usage by the Dask scheduler on the clusters (section 6).

While these are a lot of parameters to consider, worker utilization and computational progress can be directly viewed in the Dask dashboard. The dashboard of an analysis running normally can be found in figure 4.2. An additional screenshot, after the analysis is finished, is shown in figure A.1. The worker memory display helps to determine the upper limit for the chunk size. It is a bar graph, representing the memory usage of each worker. The colors of the bars give a quick indicator, how the workers are performing. Blue means, the memory is within save limits. If the bar is orange, a worker is spilling from RAM to disk, it's allocated memory is filling up. This can be seen in figure A.2. Spilling data from ram to disk hurts performance and can indicate bad parameters, if occurring shortly after the computation is started. The gray portion of the memory graph indicates, how much a worker has spilled to disk. If the bar is red, a worker is halted because not enough memory is available. This is shown in figure A.3. While it may recover from this state, this is generally a sign of an instable state. E.g. a wrong combination of chunk size and allocated RAM per worker. The task stream display also helps to visualize, what the workers are doing. Large gaps between the colored blocks would mean, that the workers have idle time. This sometimes can be explained by wrong chunk sizes for the dataset, or even too light of a workload for the number of workers. If new rows appear in it, it means new workers have joined the computation. This usually happens around the start of the computation, or if workers terminated. The progress diagram can also be an indicator, if everything is configured well. The progress of the tasks should be consistent over time. If the execution stops at a certain point, it can be used to get an indication where the problem lies. E.g. the selection could work properly, but the workers could be running into memory issues during the computation part.

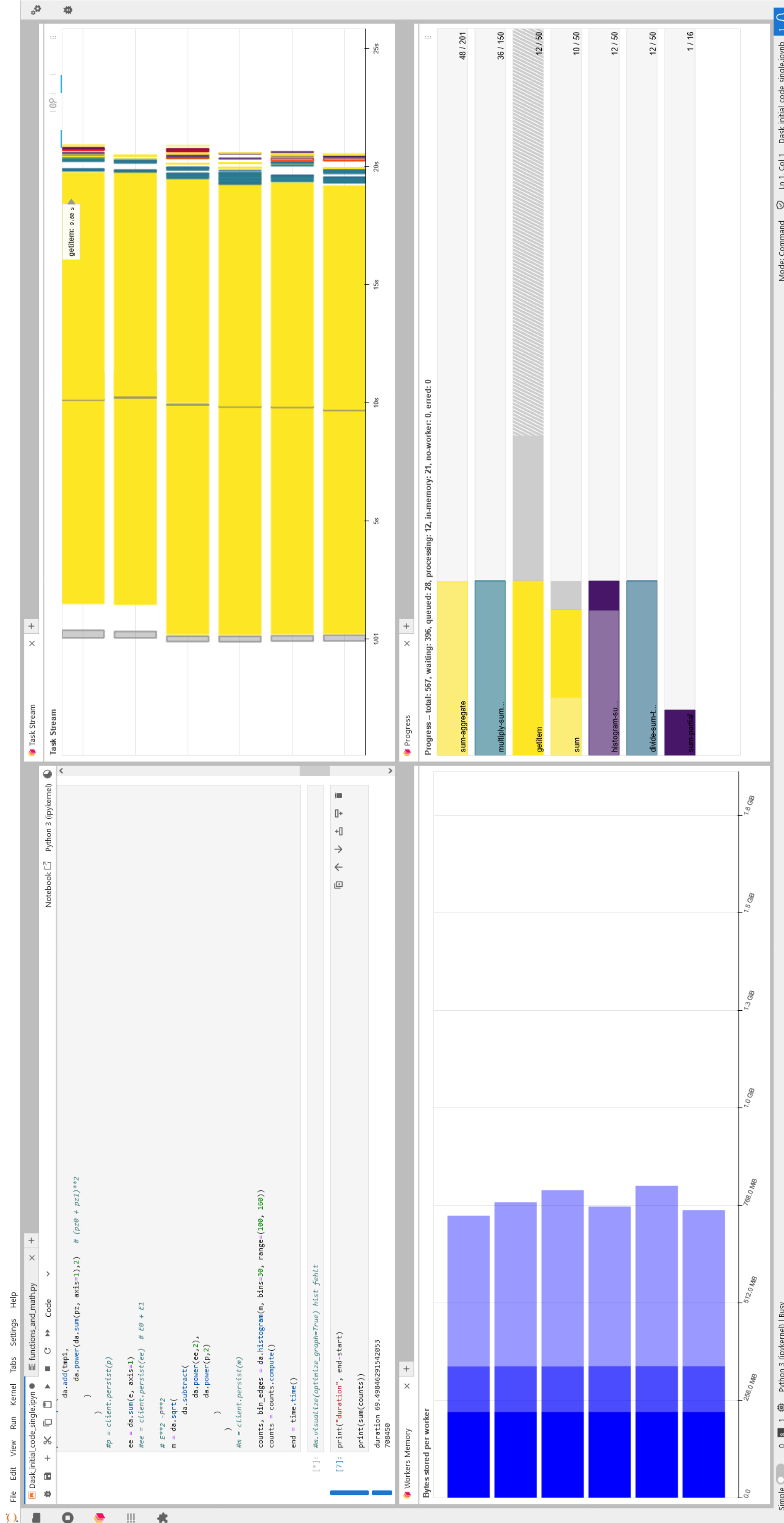


Figure 4.2: Screenshot of the Jupyter Lab website while an analysis is running. In the upper left corner, the Jupyter Notebook is shown. The lower left corner contains the bar graph of the worker memory. The Task Stream is in the plot of the upper right corner. The lower right corner contains the progress display.

4.4 Validation

In this section, the Dask-based analysis is validated against the PyROOT based analysis. Both are executed on the bwNOTEBOOK. The cut locations are the same as described in section 3.5. The cut flow can be found in figure 4.3. The number of events after each cut matches perfectly, except for cut three, which the Dask-based algorithm performs simultaneously with cut four. A direct comparison of both histograms is made in figure 4.4. Both analyses deviate below 0.1 % per bin, meaning they are again in good agreement.

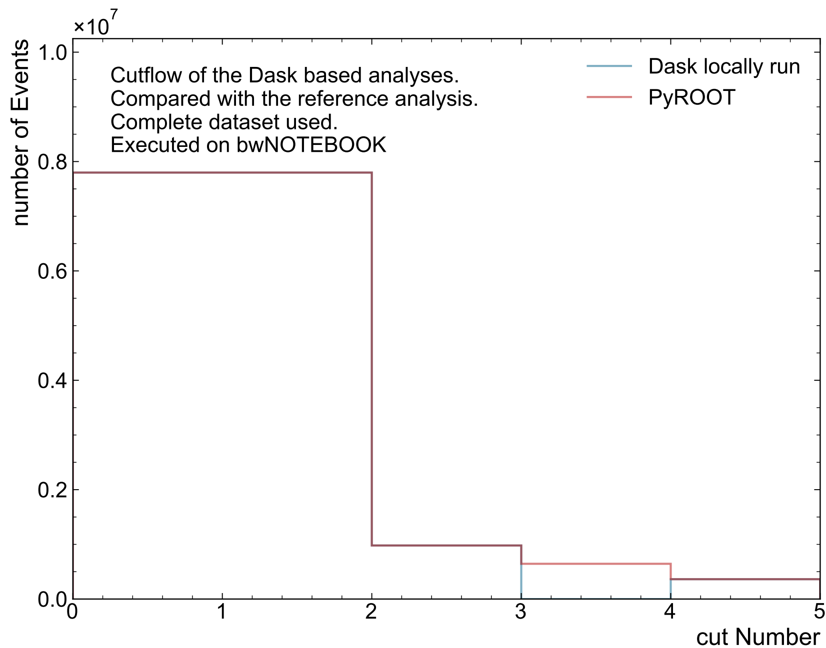


Figure 4.3: Cut flow of the Dask analysis on the bwNOTEBOOK and the reference analysis. The numerical values are shown in table B.1

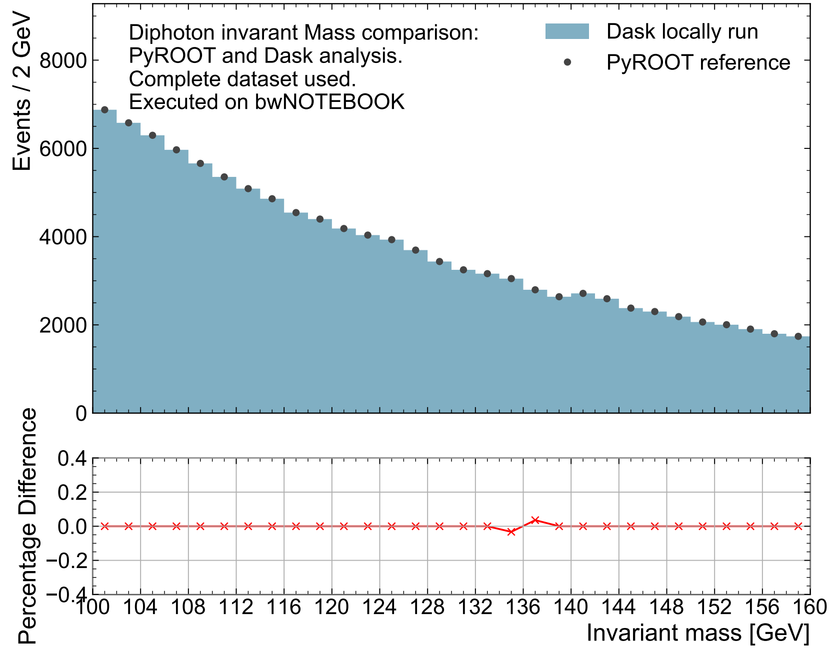


Figure 4.4: Validation of the locally run Dask analysis against the reference analysis on the original dataset. Shown is the spectrum of the invariant diphoton mass. The lower panel shows the percentage differences between the two measurements.

4.5 Performance measurements

The Dask-based analysis is run repeatedly to measure the influence of different parameters on the run time of the algorithm. During all measurements in this section, if a parameter is not explicitly mentioned, it is of standard value. The standard values for Dask workers are the following: Six workers in total, with each using one logical CPU core and 2 GB RAM. The chunk size is usually set to 1 million events. The normally used dataset consists of 50 million events. To insure comparability, the Notebook is prepared similar as described in section 3.6. After measuring one data point, the local cluster is shut down and the notebook kernel is reset.

Starting with the series in figure 4.5, the influence of the chunk size is measured. From a

size of 500 000 up to 3 million, the computing time is consistent. With the smaller chunks, the execution slows down due to increased overhead on the scheduler side. Workers need to communicate with it more often and spend less time computing. With larger chunk sizes, the workers become instable due to memory overflowing. Because of this, they get halted and even restarted, slowing down the computation. Following the recommendation [43], a chunk size of approximately 666 MB should be safe. This is roughly equivalent to a 3.6 million events file. While this chunk size would work, it is already firmly in the instable region.

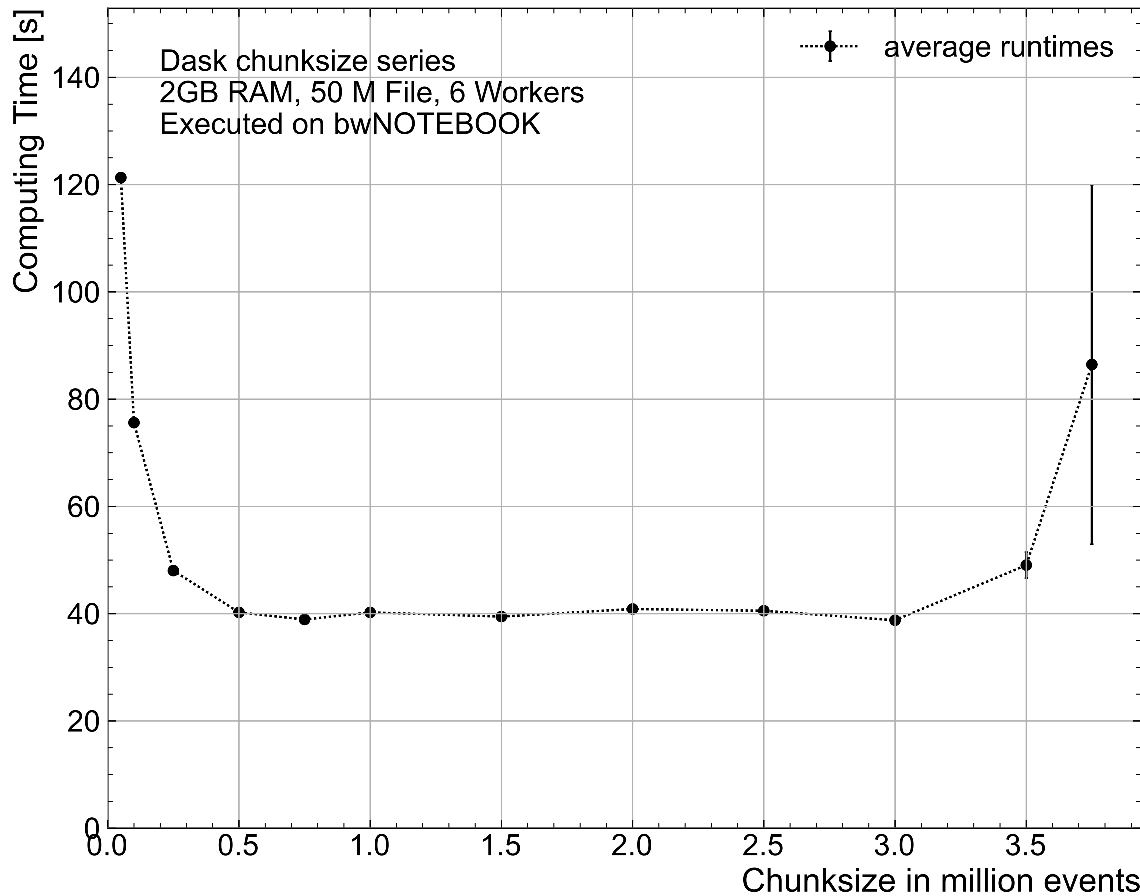


Figure 4.5: Measured computing time for different chunk sizes. The numerical values are shown in table B.6

The series in figure 4.6 measures the influence of the dataset size on the computing time. It also shows which dataset is the largest that can be analyzed. The computing time has an approximately linear relationship to the size of the analyzed dataset. The main limiting factor for the upper limit for the file size is the system memory. While the analysis starts with the 500 million events dataset, the memory fills up crashing the notebook kernel. Compared to the file size series of the NumPy algorithms in figure 3.7, it is clear that Dask enables the analysis of larger than memory datasets. The largest dataset measured has a size of more than four times the total system memory with approximately 80 GB.

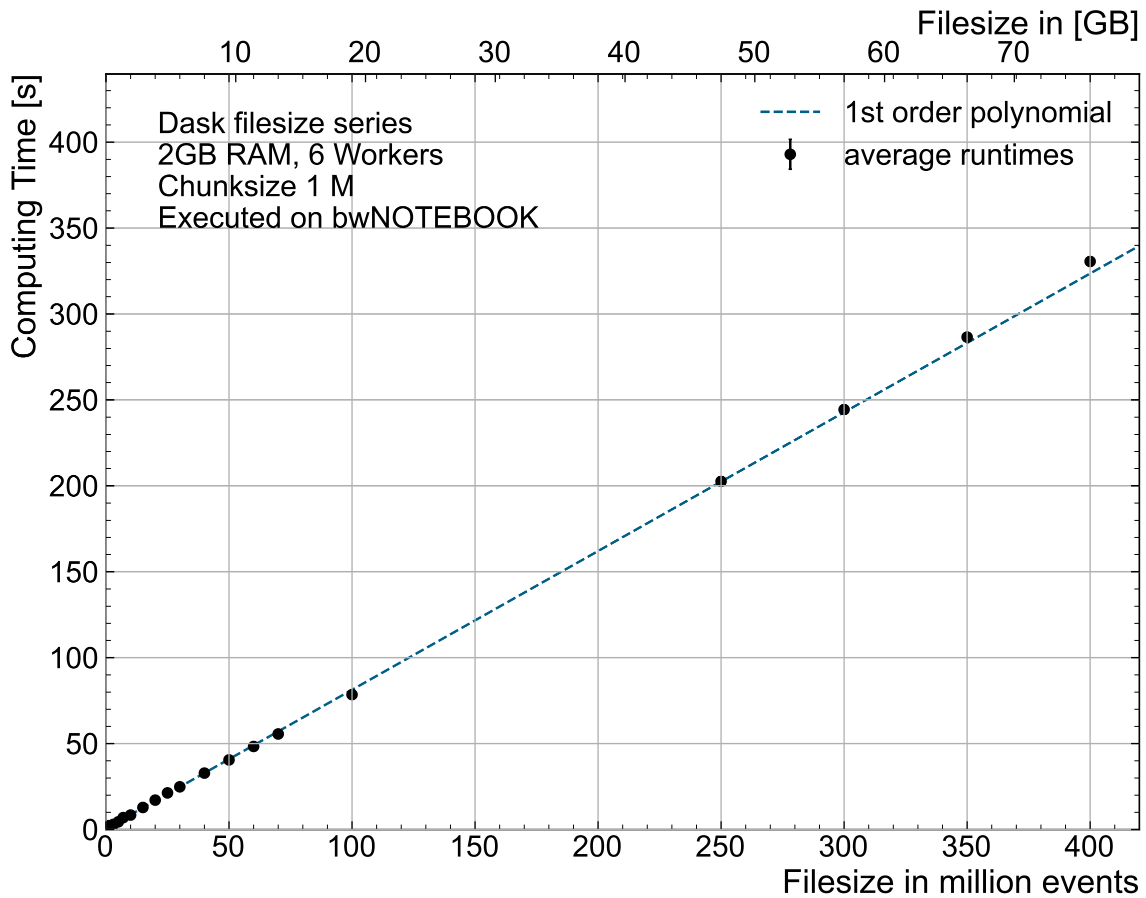


Figure 4.6: Measured computing time for a variety of dataset sizes. The numerical values are shown in table B.7

For the series in figure 4.7, the number of workers is varied. Each worker is allocated the standard amount of memory, 2 GB. Doubling the workers from one to two almost halves the computing time. The decrease of computing time is less pronounced over the next data points, saturating at four workers. This also coincides with the number of physical CPU cores of the notebook.

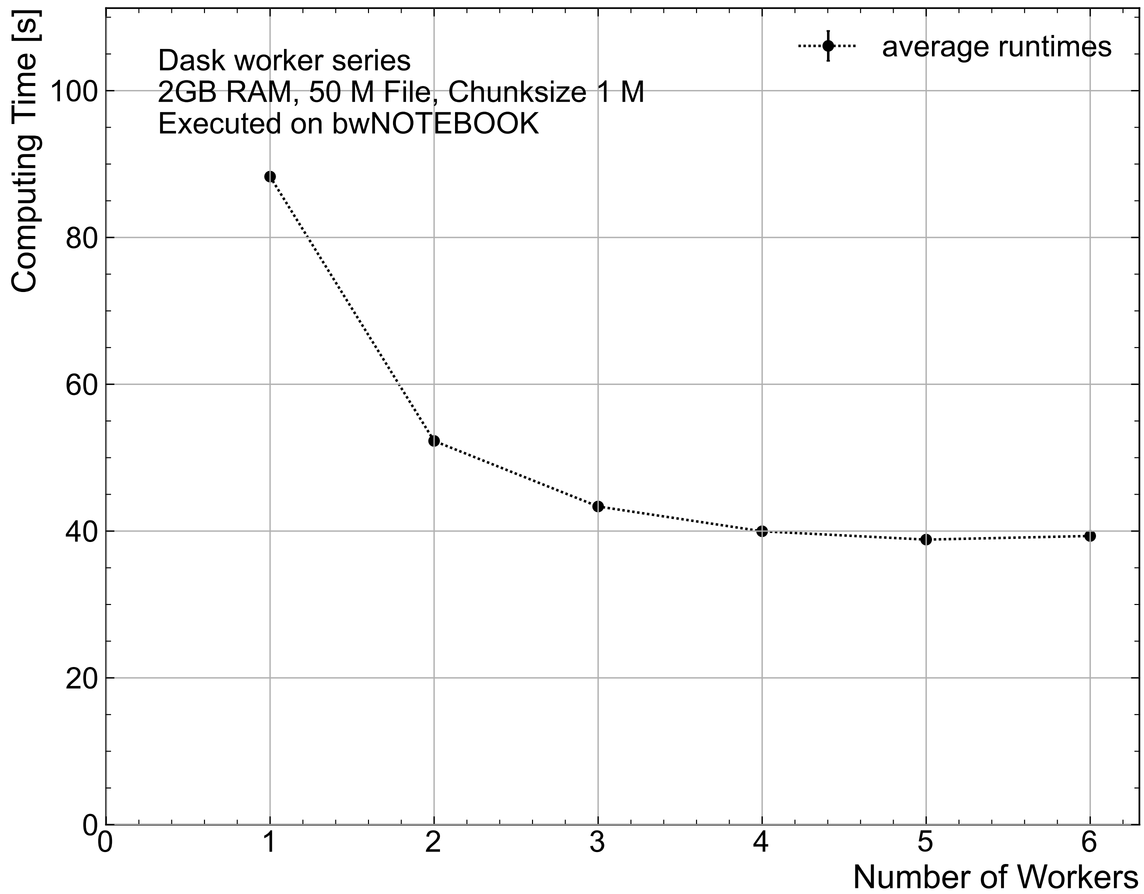


Figure 4.7: Measured computing time for varying amounts of workers. The numerical values are shown in table B.8

The influence of loading multiple files instead of one larger file is measured in two subseries, with different total combined file sizes. Both are visualized in figure 4.8. In the first subseries, files equivalent to the 50 million events dataset are loaded. In the second subseries, this was repeated with the 100 million events dataset. Both are shown individually in the figures A.4 and A.5. The size of the individual files is indicated on the x-axis. E.g. for the 10 million events file, it is loaded 5 times for the 50 million series and 10 times for the 100 million series. If the files are loaded 10 times or more often, the impact on performance is becoming more significant. In section 7, this effect is also observed.

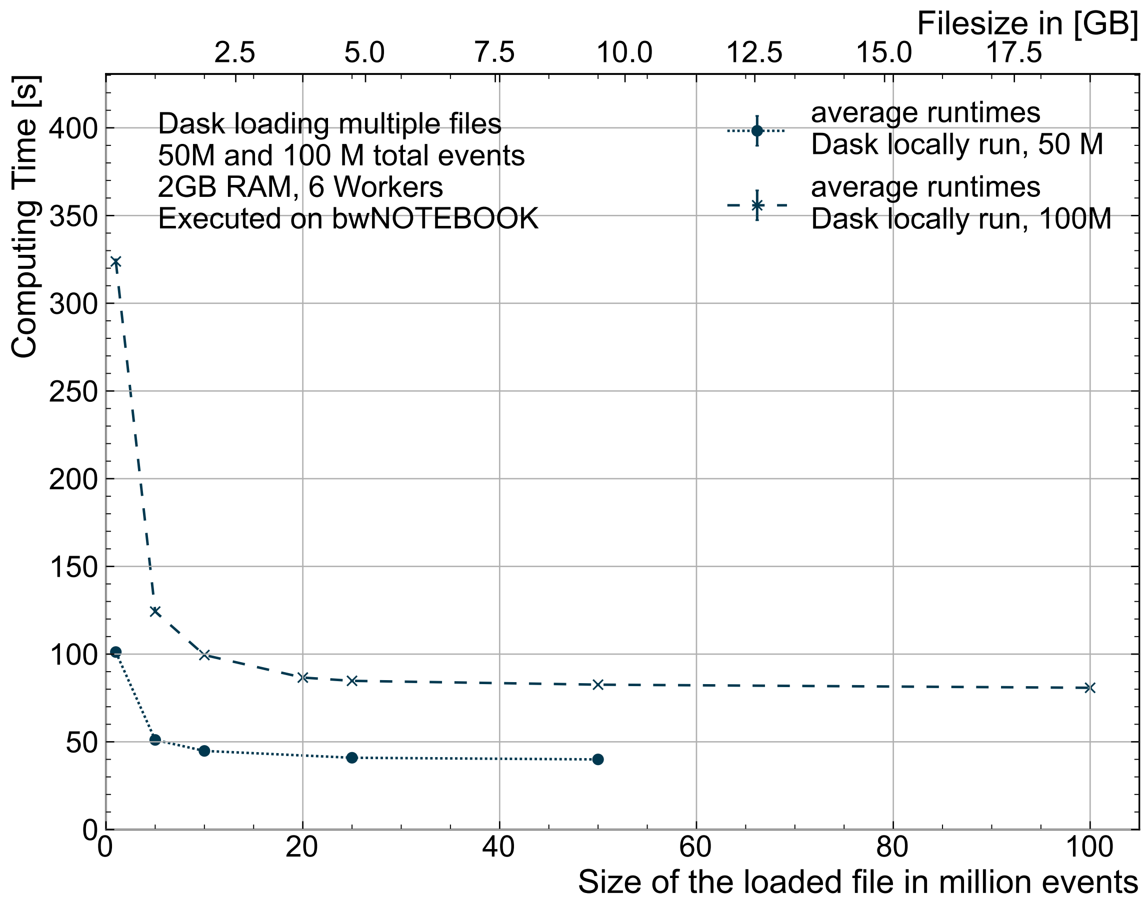


Figure 4.8: Comparison of the computing time when loading multiple smaller files with a combined number of Data points of 50M and 100M. The numerical values are shown in table B.9

5 Comparison between the PyROOT, ROOT and Python analyses

As seen in section 3.6, re-implementing the reference analysis in a vectorized Python algorithm enhances the performance compared to the PyROOT reference. The ATLAS outreach repository [44] also provides a different implementation of the same analysis, written in C++. In this section, the performance of the C++ version is compared to the PyROOT, the NumPy and Dask algorithms, since the C++ version can also be setup to run parallelized. The dataset used for these measurements is the unmodified Higgs-Gamma-Gamma dataset as described in section 3.1 and not the inflated dataset utilized in the other benchmarks in the previous sections.

5.1 C++ ROOT

The ATLAS collaboration also published a classic ROOT $H \to \gamma\gamma$ analysis. As provided on the GitHub repository [44], the analysis is started by a shell script. The user is prompted to input if the analysis should run on the data only or also on simulated events and if PROOF, parallelized ROOT, should be used. The algorithm of this analysis is event loop based. To ensure repeatability of the measurements, the shell script has been adjusted. The modified version runs the ROOT analysis directly, skipping the selecting steps. Because C++ is not interpreted, extra steps are necessary to avoid measuring compilation time during the first run. The C files are adjusted to avoid recompiling. Before measuring the runtime, the analysis is run at least once without logging the time to compile it. The computing times of the four single threaded algorithms can be found in table 5.1 and visualized in figure 5.1. The PyROOT algorithm is significantly slower compared to the other algorithms. While the C++ ROOT algorithm is faster than PyROOT, it is still slower than the NumPy-based algorithms. This performance difference can be explained by the different algorithms. The vectorization of the NumPy based implementation enables it to outperform the loop based C++ algorithm. It is possible to vectorize the C++ based analysis by using RDataFrame, but implementing this algorithm is out of the scope of this thesis.

Table 5.1: Computing times in seconds of the single threaded algorithms on the bwNOTEBOOK, using the Higgs-Gamma-Gamma dataset.

C++ ROOT computing time	NumPy Classic computing time	NumPy inMem computing time	PyROOT computing time
32.01 ± 0.09	10.61 ± 0.07	12.32 ± 0.07	294.9 ± 1.4

5.2 PROOF

The Parallel ROOT facility, PROOF [45] operates similar to Dask. Multiple worker processes compute the workload. In this case, they are running locally, sharing the resources of the system with the scheduler. Before the computing times are measured, some modifications to the C++ analysis are made. Similar described in section 5.1, a shell script starts this analysis. In the ROOT reference analysis, the number of workers is automatically set to the number of logical CPU cores available. To modify this, the C++ analysis itself is modified, that the number of PROOF workers can be specified from the shell script.

In figure 5.1, the computing time for varying numbers of PROOF and Dask workers is illustrated. The single threaded algorithms are also included, as if they use one worker. PROOF and Dask behave similarly, but the Dask-based algorithm needs approximately half the computing time of the PROOF-based one. Doubling the number of workers from one to two does not reduce the computing time to half on either of them. The fastest are three workers, which leaves one physical CPU core for rest of the system. When comparing one worker with the single-threaded version, PROOF has and ROOT have a smaller deviation compared to Dask and the NumPy algorithms. While the computing time of the ROOT-based algorithm is less than one second shorter than the PROOF-based one, Dask deviates approximately 3.3 s and 5 s from the NumPy based ones. A possible explanation are the additional steps needed by the Dask-based algorithm, delaying the start of computing: Dividing the dataset into chunks and building the task graph.

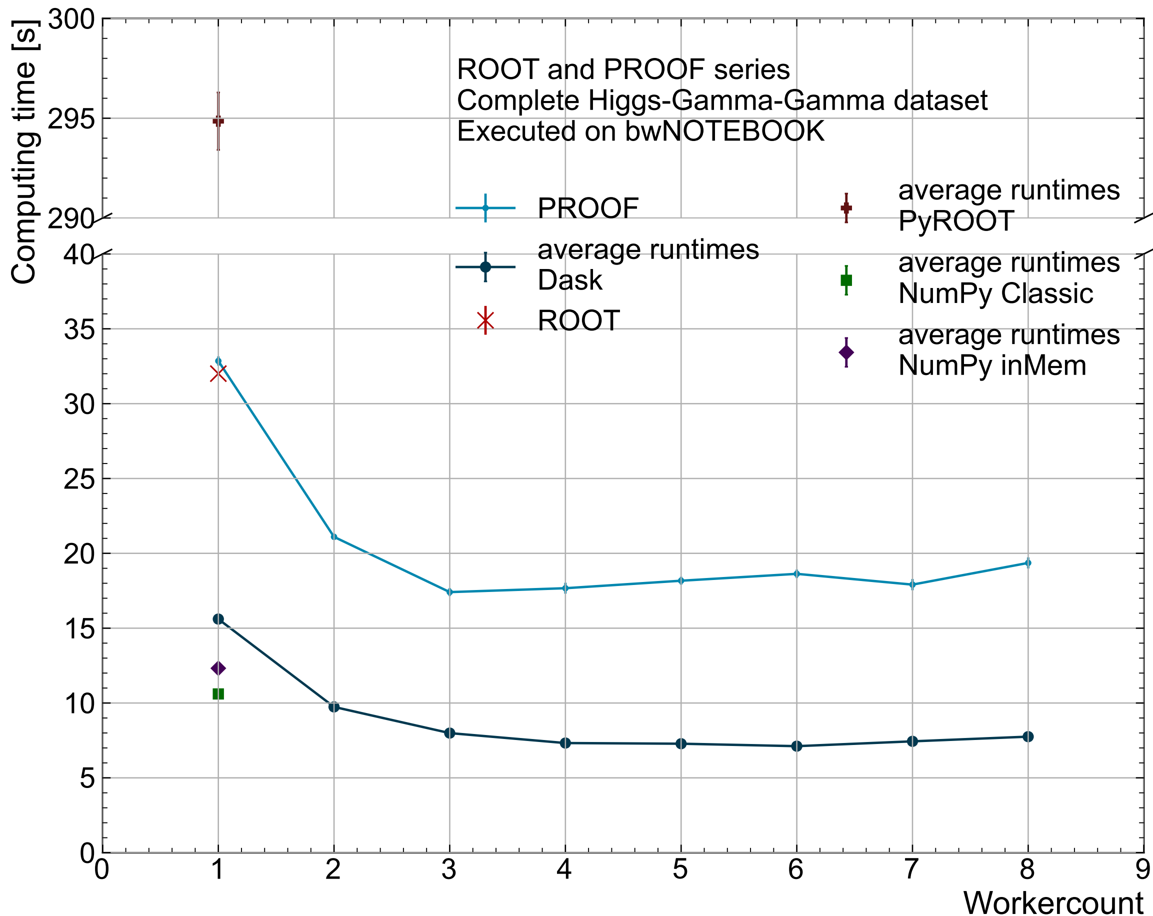


Figure 5.1: Comparison between ROOT and PROOF. The single threaded algorithms are plotted as if they have one worker. The scale of the PyROOT data point differs from the rest of the plot. The numerical values are shown in table B.10

6 Scaling a Jupyter Analysis to the local clusters with Dask

This section introduces Dask on clusters and how it differs from locally running Dask. Then, the high-throughput computing (HTC) cluster ATLAS-BFG and the high-performance computing (HPC) cluster NEMO are introduced and an overview on their architectures is given. The performance of both clusters is first evaluated separately, then they are compared with each other.

Computing clusters provide a large amount of computing power to the user. This is usually significantly more than one machine can provide. Accessing these resources is not as straightforward as local computing. Shared resources need management, adding complexity to their use. When switching from locally running software to a cluster, a minor change is the split between work and home directories. The user directory is limited in disk space per user, but generally does not expire. This makes it good for storing programs and results in this case. The work directory has a lot more capacity available, but after a certain amount of time it will be deleted. This makes it suited for temporarily storing large datasets. A bigger change is the shared nature of the computing resources of clusters. Depending on the system, it is not easily possible to access the allocated computing resources directly. Usually, a batch job is send by the user to the scheduling system of the cluster. When compute resources, workers, are allocated to the job, it gets processed. After finishing, the computing resources are released, making them available to other users. This creates multiple issues for interactive analyzes. After requesting them, the user needs to wait for the resources. The user can only begin with her/his work, after the resources are allocated. Depending on the remaining wall time of the workers, they could be released during the runtime of the analysis. When requesting more than one worker, there can be a delay between the allocation of the first and subsequent ones. Dask has build in tools to mitigate these issues during normal use. New workers can be requested automatically, when the end of the allocation time is reached. Dask also supports later joining workers. These functionalities cannot be utilized in the scope of this thesis. For repeatability, it is crucial to keep the number of workers constant during a measurement. This introduces a new aspect to this part of the thesis: Actively managing workers.

6.1 BFG

The Black Forrest Grid (BFG) [11] is an HTC cluster, located at the University of Freiburg. The cluster is exclusively used by users related to the ATLAS experiment. All nodes of the ATLAS-BFG run on the same hardware: DALCO model S2600K servers with Intel Xeon E5-2630v4 CPUs, which have 20 cores, operating in hyperthreading mode, providing 40 logical cores. Each node has 128 GB RAM. Three user-interface (UI) nodes and 84 worker nodes are available. A total of 3200 logical CPU cores is made available through the Slurm (Simple Linux Utility for Resource Management) batch system [46]. Multiple Slurm queues are available to the user, such as the standard queue, the express queue and the `nemo_vm_<group-id>` queues. Every group of the ATLAS experiment at the University of Freiburg has their own group-id. In this section, only the express queue is used. The `nemo_vm_atlsch` queue is utilized for two series in section 6.3. A schematic overview can be found in figure 6.1.

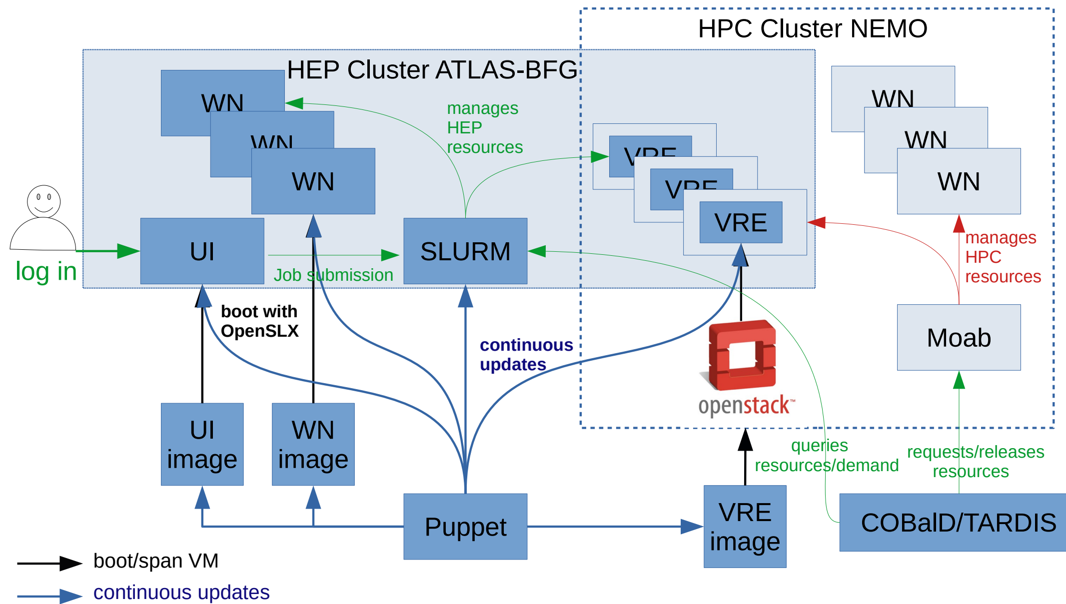


Figure 6.1: Schematic overview of the ATLAS-BFG and NEMO [47]. The user logs into the UI node to submit their work to the Slurm scheduler. Depending on the selected queue, the scheduler sends the job to workers of ATLAS-BFG or to NEMO workers, running a virtual machine.

6.1.1 Setup and architecture of Dask on ATLAS-BFG

Running the Dask-based analysis on the ATLAS-BFG requires some setup. The Dask-Jobqueue library [48] enables Dask to communicate with Slurm and other schedulers. The standard configuration for the workers is defined in the jobqueue.yaml file as follows: One CPU core and one thread with 2 GB RAM. Unless specified otherwise, all Dask workers used in this thesis on the ATLAS-BFG use this configuration. The default network interface with 1 Gb/s is used.

The UI nodes of the ATLAS-BFG are the entry point for users to access the cluster. They are intended for setting up the jobs before sending them to the scheduler for the actual computation. The Jupyter Notebook and the Dask scheduler are running on an UI node, when the ATLAS-BFG is in use for a measurement. The Dask workers should not run on the UI node, but on the worker nodes of the cluster. This is possible with Dask-Jobqueue. With it, the user can directly request resources from Slurm for the workers. For better comparability and to reduce waiting times, the workers are requested from the `express` queue. This distributes the workers over a set of 24 nodes, with a maximum of two logical cores available on each of them. Each worker has a maximum wall time of 30 minutes.

6.1.2 Analysis Validation

The Dask-based analysis on the ATLAS-BFG was again validated against the PyROOT reference analysis. The Reference analysis is executed on the bwNOTEBOOK. After verifying the cut flow, as seen in figure 6.2, the histograms for both series are compared. The number of events after each cut matches perfectly, except for cut three, which the Dask-based algorithm performs simultaneously with cut four. Figure 6.3 visualizes them and their deviations. No events have shifted their bins. The Dask-based analysis running on the ATLAS-BFG shows very good agreement with the reference analysis.

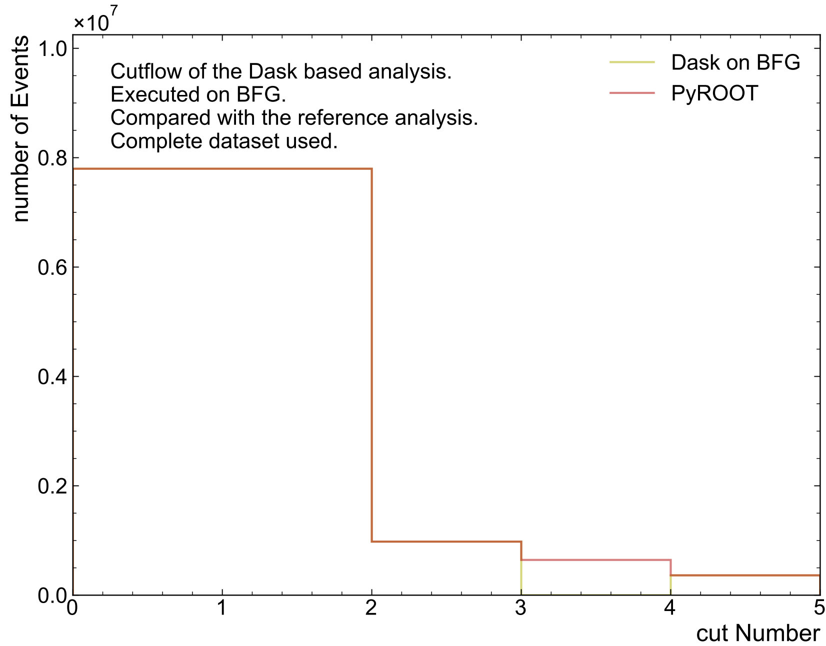


Figure 6.2: Cutflow comparing Dask Analysis on ATLAS-BFG to the reference analysis. The numerical values are shown in table B.1

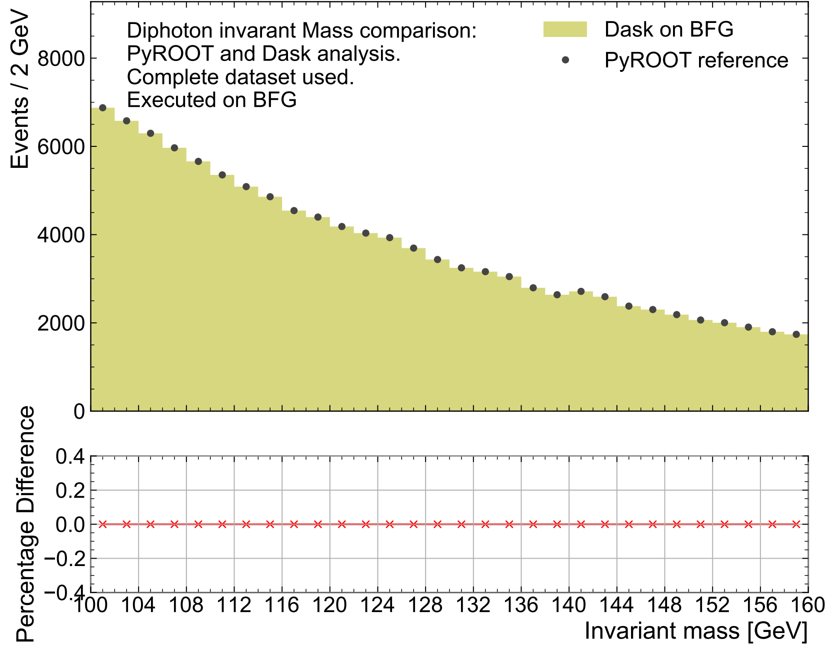


Figure 6.3: Validation of the Dask Analysis on ATLAS-BFG against the reference analysis on the original dataset. Shown is the spectrum of the invariant diphoton mass. The lower panel shows the percentage differences between the two measurements.

6.1.3 Dask specific constraints on the ATLAS-BFG

The combination of a Jupyter Notebook with Dask and Dask-Jobqueue enables the user to run an interactive analysis on a cluster. As mentioned in section 6, Dask can handle common issues of this architecture. If the parameter of interest is not the number of workers, it must be assured that their number is constant during the entire measurement. This makes actively monitoring the workers necessary. The main goal here is to ensure repeatability of the measurements, but efficient resource usage is also important. Many of the measurements are short compared to the wall time of the workers. Releasing the current workers and requesting new ones generates a lot of overhead. To avoid this, the Dask cluster is not shut down after measuring a single data point, but reset. This refreshes the Dask workers, but keeps the Slurm workers active. While this reduces overhead, releasing the Slurm workers is inevitable when longer series are measured. Slurm automatically reallocates them after 30 minutes. If this were to happen during a running measurement, the data point would become unusable. Two mitigation strategies are possible: Firstly, Recording such events and marking or deleting the affected data point. Alternatively, the workers can be restarted if the remaining wall time is too short to measure the next data point. In this thesis, the second option is used. For each configuration a runtime estimate is saved or updated after the data point is measured. This estimate is used to determine if the Slurm workers should be restarted.

6.1.4 Performance measurements on the ATLAS-BFG

For the benchmarks on the ATLAS-BFG, the standard setup consists of 48 workers with a chunk size of one million events. While larger files were generated, ROOT limits the size of single files to approximately 100 GB. For this reason, the largest generated dataset has one billion events.

For the first measured series on the ATLAS-BFG, the number of workers is varied. This is done for the 50 million and 100 million events datasets, individually visualized in figures A.6 and A.7. Both series are combined in figure 6.4. Doubling the number of workers from one to two halves the needed computing time on both datasets. For the 50 million events file, significant performance gains stop with more than 6 workers. This saturation effect starts at 10 to 16 workers for the 100 million events file. This can be explained by the number of chunks, or in this case available jobs for the workers. The datasets are split into 5 and 10 pieces. For this reason, an upper limit to the parallelization exists. Once a certain number of workers is exceeded, some are underutilized.

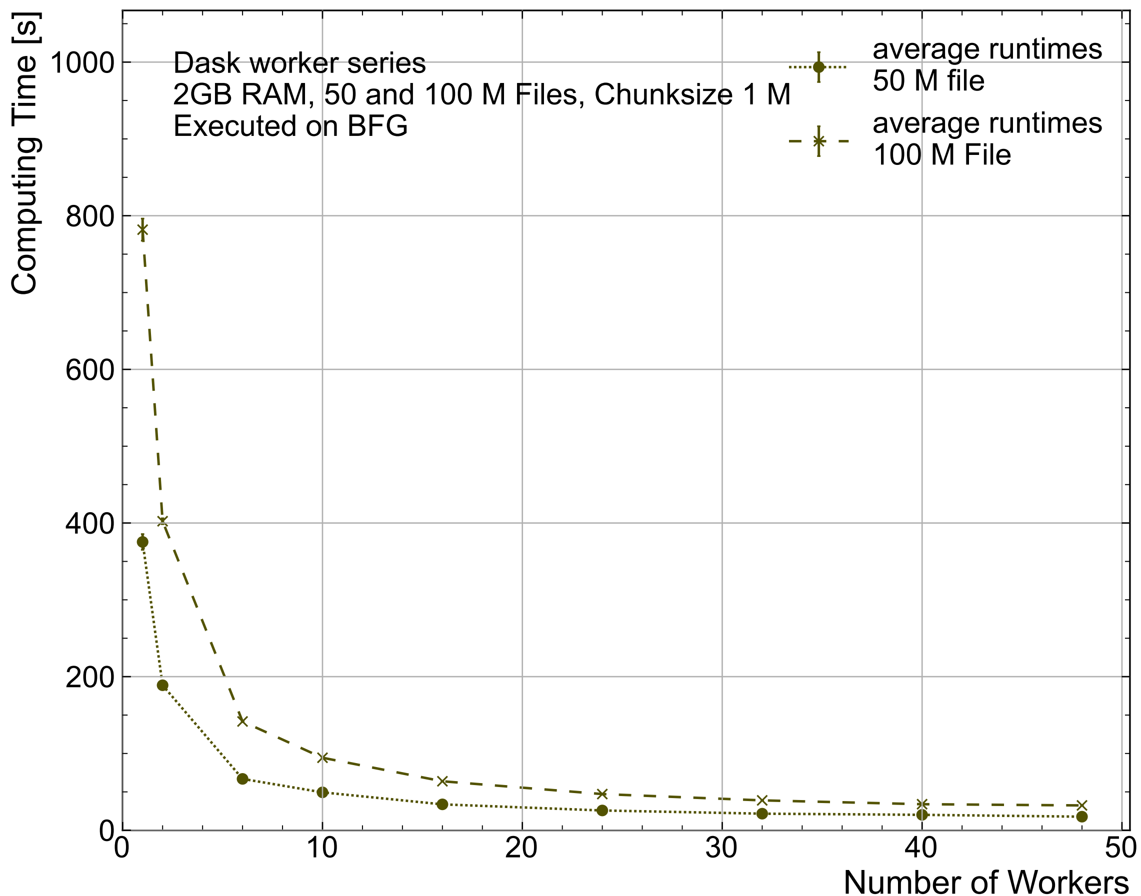


Figure 6.4: Comparison of the computing time for varying amounts of workers on the 50M and 100 M events files. The numerical values are shown in table B.11

The next series measures the performance impact of increased dataset sizes for chunk sizes of one and three million events. Both subseries individually can be found in figures A.8 and A.9. They are visualized together in figure 6.5. The computing time roughly follows a second order polynomial. When comparing both subseries, the one with a smaller chunk size outperforms the one with the larger chunk size. The increased chunk size reduces the overall number of chunks needed for the dataset, reducing the size of the task graph. This decreases the overhead in the Dask scheduler running on the UI node, decreasing RAM usage significantly. A larger influence on the overall performance are the modified jobs themselves. The larger chunks increase the computing time per job for each worker and the amount of RAM needed by the workers. Since the dataset size is constant, larger jobs also mean fewer jobs, reducing parallelization of the workload. This contributes to a larger overall compute time.

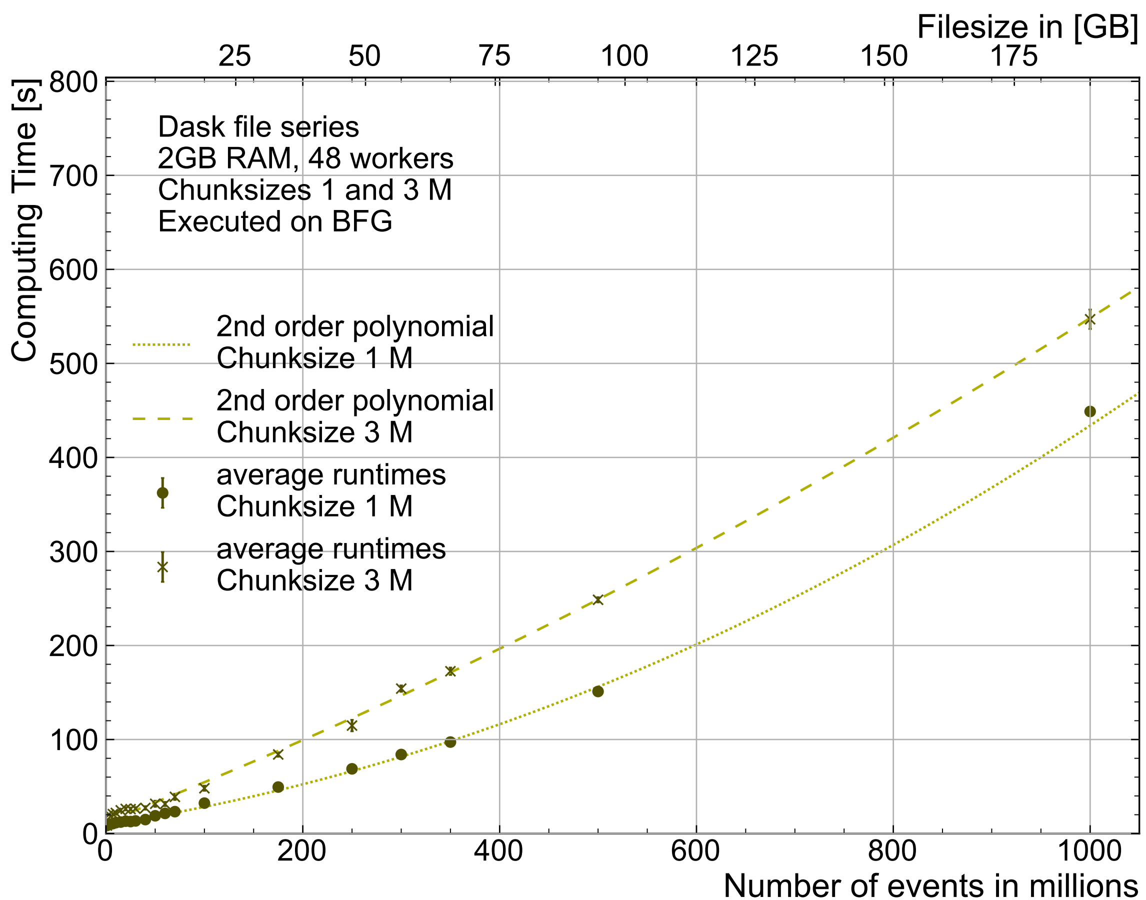


Figure 6.5: Comparison of the computing time over multiple sizes of datasets. 48 workers with each using one thread with a chunk size of 1 M and 3 M events. The numerical values are shown in table B.12

For the last series in this section, both chunk size and the RAM allocated to the workers are varied. The ratio of chunks per GB RAM is constant over the three measured series. The first one, 500 thousand events per GB (500k/GB) are the standard ratio, resulting in a chunk size of one million events for 2 GB of allocated memory. For the second (1M/GB) and third (2M/GB) ratios, the available memory per event is halved and quartered. All data points are measured with the 500 million events dataset and 48 workers. Because this series has two variables, the allocated RAM per worker and chunk size, two visualizations are made. In figure 6.6 the allocated memory is the independent variable, in figure 6.7 it is the chunk size. Because the storage system was under heavy load caused by outside factors during the measurement of this series, it was remeasured. When comparing figures 6.6 and A.10 as well as figures 6.7 and A.11, it is clear that the influence is not significant. Table 6.1 lists the total number of chunks for each measurement, as an estimate for the number of jobs.

Table 6.1: Number of chunks per data point of the chunk size and RAM size series.

Subseries	0.5M per GB	1M per GB	2M per GB
Ram size [GB]	Number of chunks		
2	500	250	N/A
4	250	125	63
6	167	84	42
8	125	63	32
10	100	50	25
12	84	42	21
14	72	36	18
16	63	32	16

Beginning with figure 6.6, some clear trends are visible. After an initial decrease in computing time, all three subseries seem to lose performance with increased RAM allocation. The initial increase in performance can be explained by the decreasing overhead in the Dask scheduler with increasing chunk sizes. The decrease in performance for larger memory size can be explained by a decrease in parallelization. Since the file size is constant, larger chunks result in fewer total chunks.

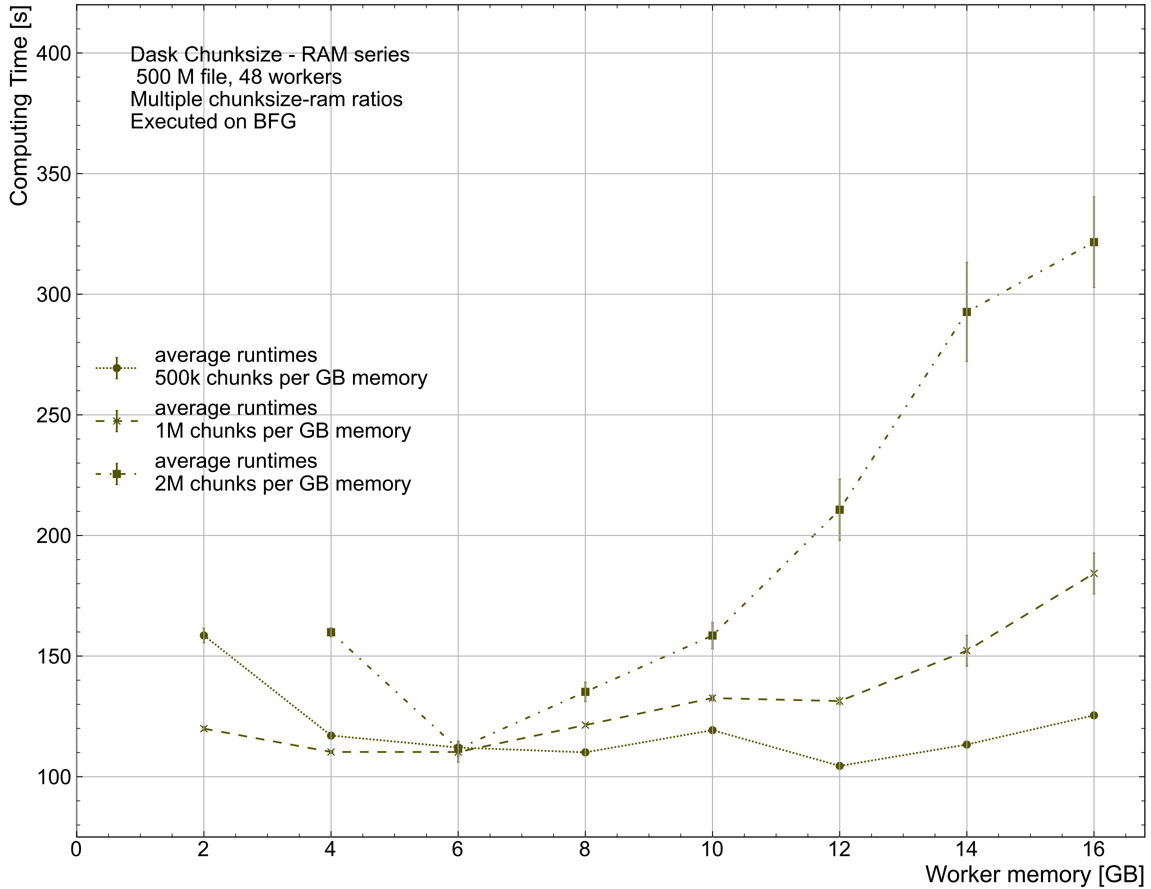


Figure 6.6: Comparison of the computing time for different chunk sizes and RAM allocated to workers, using the 500M events file. For 2 GB RAM, only chunk sizes of 1 and 2 million events could be measured. In this series, two parameters are varied: The chunk size and allocated RAM of the workers. This figure displays the relationship of the RAM size and computing time. The connection between the absolute chunk size and the computing time is visualized in fig. 6.7. The numerical values are shown in table B.13. Remeasured series in fig. A.10

Reviewing figure 6.7 a similar trend is observed. After a certain point, an increase in chunk size decreases the overall performance. Comparing the three subseries, it is expected that doubling the allocated memory for a constant chunks size increases performance. At a chunk size of 8 million events, the workers in the 2M/GB series have 4 GB RAM, the 1M/GB have 8 GB and the 500k/GB workers have 16 GB RAM allocated. As expected, the 2M/GB series performs worst. The 1M/GB series performs better than the 500k/GB series. While not expected when viewing the series in isolation, all Dask workers are running on ATLAS-

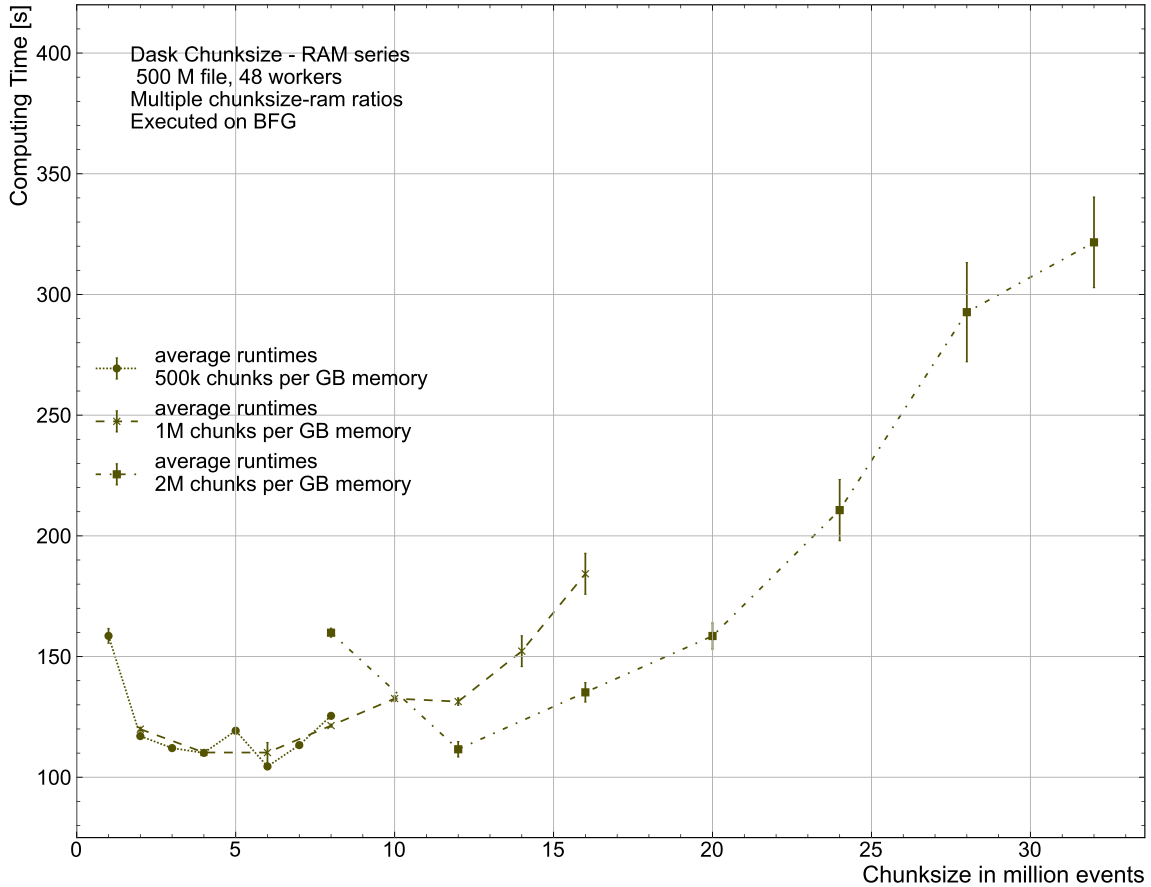


Figure 6.7: Comparison of the computing time for different chunk sizes and RAM allocated to workers, using the 500M events file. For 2 GB RAM, only chunk sizes of 1 and 2 million events could be measured. In this series, two parameters are varied: The chunk size and allocated RAM of the workers. This figure displays the relationship of the absolute chunk size and computing time. The connection between the allocated RAM and the computing time is visualized in fig. 6.6. The numerical values are shown in table B.14. Remeasured series in fig. A.11

BFG worker nodes, which are shared between different users. It is possible, that the other jobs on the same machine had different performance impacts for both series. Viewing the same data point in the remeasured series, the 500k/GB subseries has a much higher spread. In general, the differences between the 500k/GB and 1M/GB series are small compared to the 2M/GB series. This matches with the results from section 4.5, where both points would be in the high performing region of the chunk size series. Two other points match the expectation: For the chunk sizes of 2 and 6 million events, the 500k/GB series outperforms the 1M/GB series by a small margin. For the chunk sizes above 8 million events, increases in allocated memory no longer seem to be the driving factor for improvements in performance. 12 and 16 million events, the 2M/GB subseries performs better than the 1M/GB series.

In conclusion, workers with more than 6 GB RAM become inefficient. Within this memory range, chunk sizes behave proportionally to the computing time. For best performance, it should be as small as possible within the constraints of the overall system.

6.2 NEMO

The Research Cluster for Neuroscience, Elementary Particle Physics, Microsystems Engineering and Materials Science (NEMO) [12] is located at the University of Freiburg. Unlike the ATLAS-BFG, it is not limited to members of the ATLAS experiment. It is available to users researching the aforementioned domains in Baden-Württemberg. NEMO is equipped with the same hardware as the ATLAS-BFG, but with disabled hyperthreading. It has 900 compute nodes resulting in 18 000 available CPU cores. In figure 6.8, an overview of NEMO is shown. NEMO has two login nodes and two directories: home and work. While the home directory is different for NEMO and the ATLAS-BFG, the work partition is the same. The main differences between both clusters are related to the workers. In NEMO, each user gets an entire worker node allocated. This single-user-policy avoids that the workflow of a given user might interfere with the workflow of a different user. Nevertheless, when requesting more than one Dask worker with one CPU core each, they get assigned to a single compute node. Only when requesting more resources as one node can provide, a second node is allocated. The wall time of the workers on NEMO can be up to four days. Requesting resources on NEMO is done via the Moab scheduler [49].

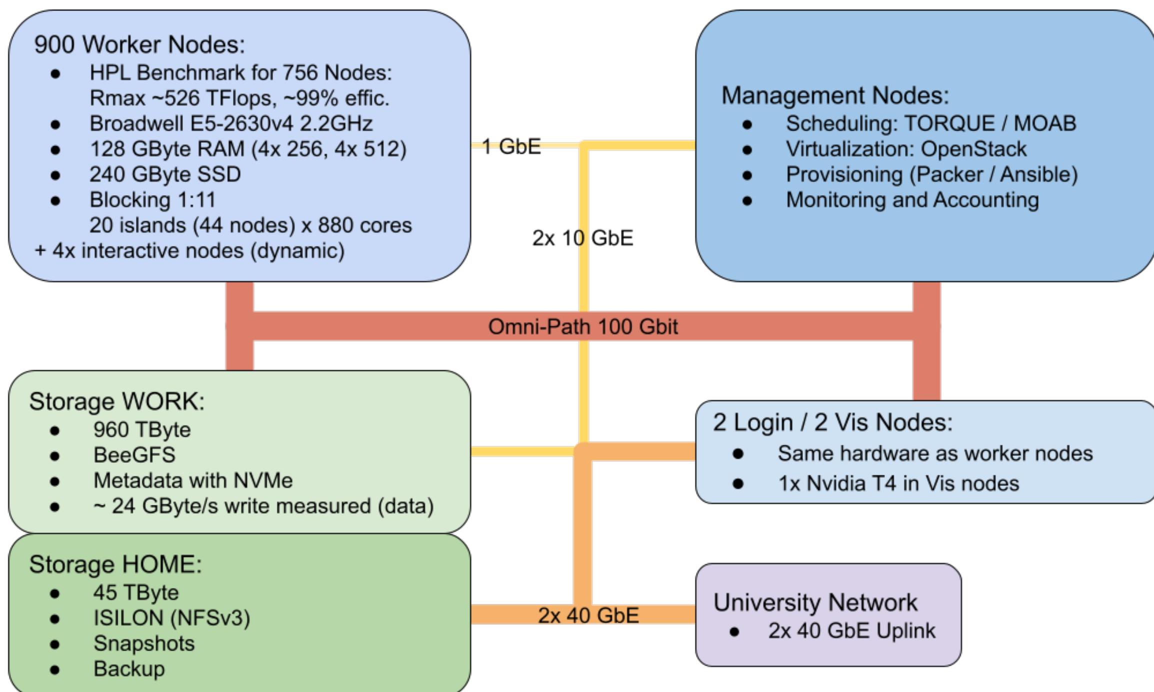


Figure 6.8: Overview of the NEMO cluster [50].

6.2.1 Setup and architecture of Dask on NEMO

Since Dask-Jobqueue is compatible with Moab, the setup on NEMO is similar to the ATLAS-BFG setup. After installing the required Python libraries, a configuration file defining the standard parameters for the Dask workers is created. These are one CPU core and one thread, 2 GB RAM and a wall time of 15 hours. The standard number of workers is set to 20, filling one worker node. Both wall time and worker count are chosen to minimize the waiting time, because NEMO has a high utilization with no reservation for interactive jobs.

6.2.2 Analysis Validation

The Dask analysis on NEMO is also validated against the reference PyROOT analysis on the bwNOTEBOOK. Both cut flows are visualized in figure 6.9. The number of events after each cut matches perfectly, except for cut three, which the Dask-based algorithm performs simultaneously with cut four. The histograms of the invariant diphoton mass of both analyses do not deviate from each other, as shown in figure 6.10. This implies a good comparability of both algorithms.

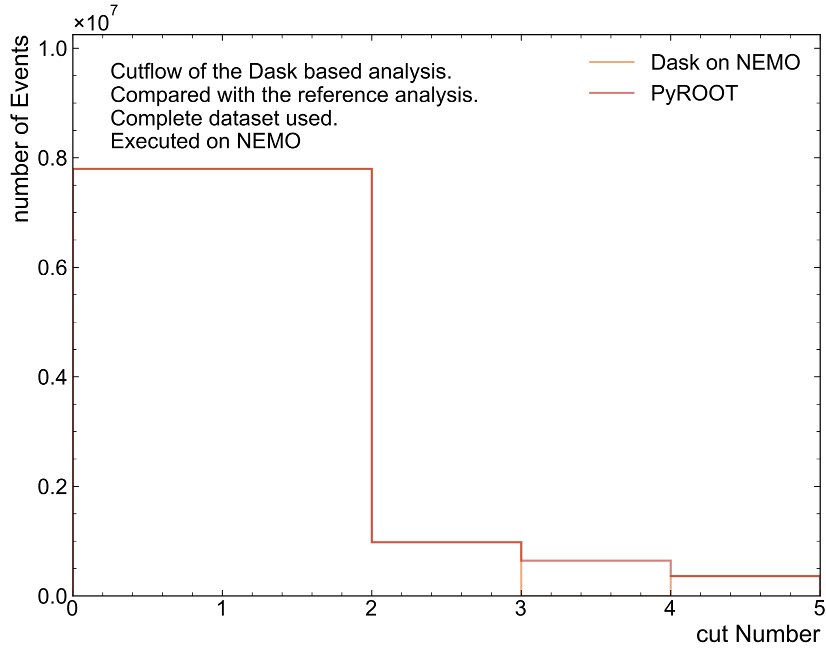


Figure 6.9: Validation of the Dask Analysis on NEMO. The numerical values are shown in table B.1

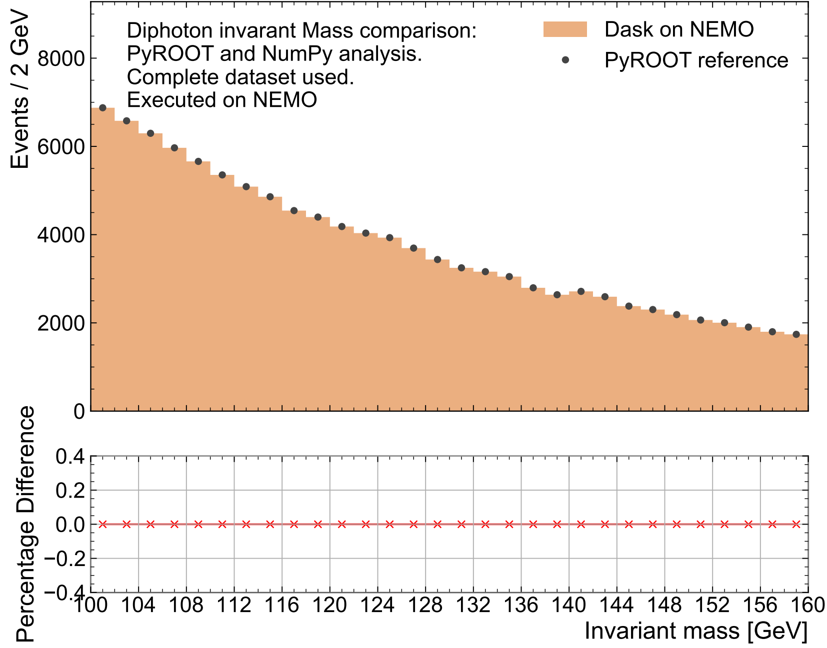


Figure 6.10: Validation of the Dask Analysis on NEMO against the reference analysis on the original dataset. Shown is the spectrum of the invariant diphoton mass. The lower panel shows the percentage differences between the two measurements.

6.2.3 Dask specific constraints on NEMO

Similar to the setup on ATLAS-BFG, the Jupyter Notebook and Dask scheduler are run on the login node. The maximum possible wall time of the NEMO workers is longer than the wall time of the express queue on the ATLAS-BFG. With an upper limit of four days, the workers can be requested for a longer timespan than the total time necessary for any single series measured on the ATLAS-BFG for this thesis. This eliminates the need for actively managing workers. As previously mentioned, a wall time of 15 hours is used as the default value. This enables to measure an entire series without requesting new resources. Resetting the Dask workers on NEMO is more problematic as on the ATLAS-BFG. Often, not all workers respond after repeated resets within the default times set in Dask. This necessitates a more robust restart function, combining longer wait times and repeated restarts. While needed for the benchmarks in this thesis, during normal use this should not be required. Requesting and releasing resources on NEMO is slower compared to the ATLAS-BFG. For this reason, workers are requested before the benchmarks are started. Occasionally, the demand for resources is high enough, that wait times are multiple hours. This makes interactive use difficult while working with the standard queue. While an express queue exists, it is not used in this thesis. Its wall time of 15 minutes would need even more active job management to ensure repeatability. With the longer waiting times, this would significantly increase the overhead on any benchmark measurement. For this reason, the standard queue with a wall time of 15 hours is used. Once the resources are allocated, the complete series can usually be measured without further delays.

6.2.4 Performance measurements on NEMO

The first benchmark run on NEMO is the dataset size series. For this, two subseries with chunk sizes of 1 and 3 million events are measured.. The results are visualized together in figure 6.11, individually in figures A.12 and A.13. The computing time for increased dataset sizes does not follow a liner trend. For the subseries with the smaller chunk size, the computing times loosely follow a second order polynomial. The highest dataset size in this series is 750 million events with a chunk size of one million events. When three million events are used, only the 250 million data point file can be processed. A possible explanation for the smaller maximal amount of processable number of events compared to the measurement on the ATLAS-BFG is the reduced amount of total RAM available. On the ATLAS-BFG, 48 workers are used with 2 GB RAM each, for a total of 96 GB RAM. In this measurement, 20 workers with 2 GB RAM each are used, with a total of 40 GB RAM. While not the complete dataset needs to be stored in RAM, the workers need to keep the results they previously worked on. This fills up their RAM to the point where they eventually cannot continue working on larger datasets. The increased chunk size complicates this by necessitating a larger amount of RAM for the active computation, leaving less space for previous results. This could explain, why tripling the chunk size limited the maximal dataset size to a third.

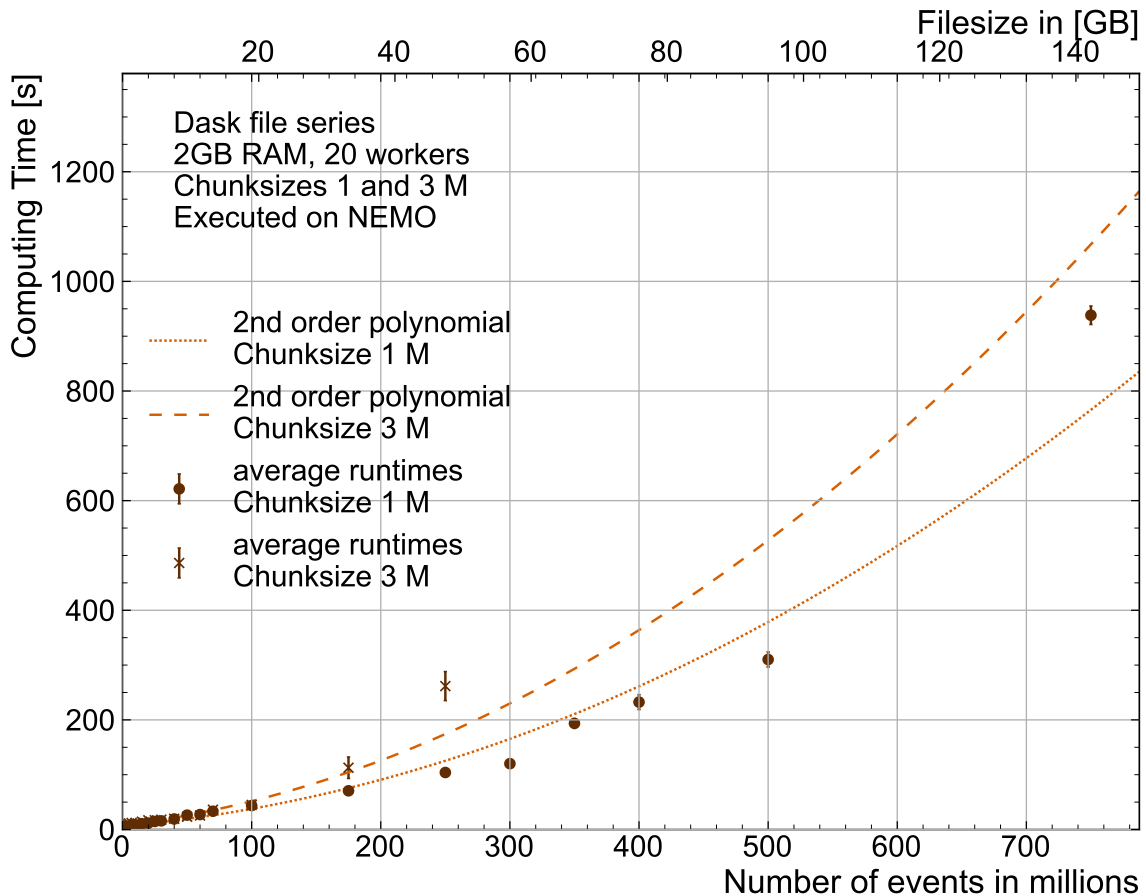


Figure 6.11: Comparison of the computing time over multiple sizes of datasets with chunk sizes of 1 and 3 million events. The numerical values are shown in table B.17

The second measured series on NEMO investigates the influence of the number of workers on computing time. Two subseries are run for this, one starting with 40 workers and one with 20 workers. In each subseries, the 50 million and the 100 million event datasets are analyzed on varying numbers of workers. In figure 6.12 the complete measurement is visualized. The individual measurements can be found in figures A.14 - A.19. Similar to the worker series measured on ATLAS-BFG in figure 6.4, doubling the number of workers from one to two halves the required computing time. In the subseries of the 50 million event data set, significant performance gains stop after 6 to 10 workers. For the 100 million events subseries, the effect starts at 10 to 15 workers. This matches with the ATLAS-BFG series, where also approximately 10 chunks per worker seems the most efficient utilization of the workers.

The two subseries have some obvious differences: One series has one fewer data point and the subseries starting with 40 workers outperforms the other subseries for some numbers of workers.

The 100 million data point file could not run on one single Dask worker in the series starting with 20 workers. Here, the computation stopped after the memory allocated to the worker is filled. This contradicts the expectation of previous measurements, one single worker should have sufficient memory to analyze this file. Restarting the Dask workers also empties their allocated memory, so leftover information of previous analyzes occupying it should be excluded as a possible reason. The two series starting with more workers (40 on NEMO, 48 on ATLAS-BFG) can both analyze this file, indicating the influence of previous measurements is dependent on the maximum worker size and probably the total memory capacity of them.

The series starting with 40 workers are faster compared to the series starting with 20 workers. To explain this, the resource allocation needs to be considered. When starting with 40 Dask workers, two worker nodes are allocated. On the 20 workers data point, the workers are spread 13:7 between both worker nodes. The subsequent data points are distributed between the worker nodes as follows: 10:6, 8:2, 5:1, 2:0, and 1:0. This means, for all data points with 6 or more Dask workers, two worker nodes of NEMO are running the calculations. More even distributions of the Dask workers on both worker nodes result in better performance of the analysis. The random distribution of Dask workers over the two NEMO worker nodes is caused by the Dask scheduler. It is deciding which resources to release while unaware, how they are distributed over the machines. For this reason, the possible utilization optimizations of Moab, running the fewest worker nodes per use do not apply. With the single-user-policy of Moab, other users are blocked from running on the same node, resulting in underutilization of resources. If the Dask scheduler was aware of the distribution of workers per worker node, a different optimization would be possible. For best performance, the Dask workers should be distributed evenly between all allocated worker nodes.

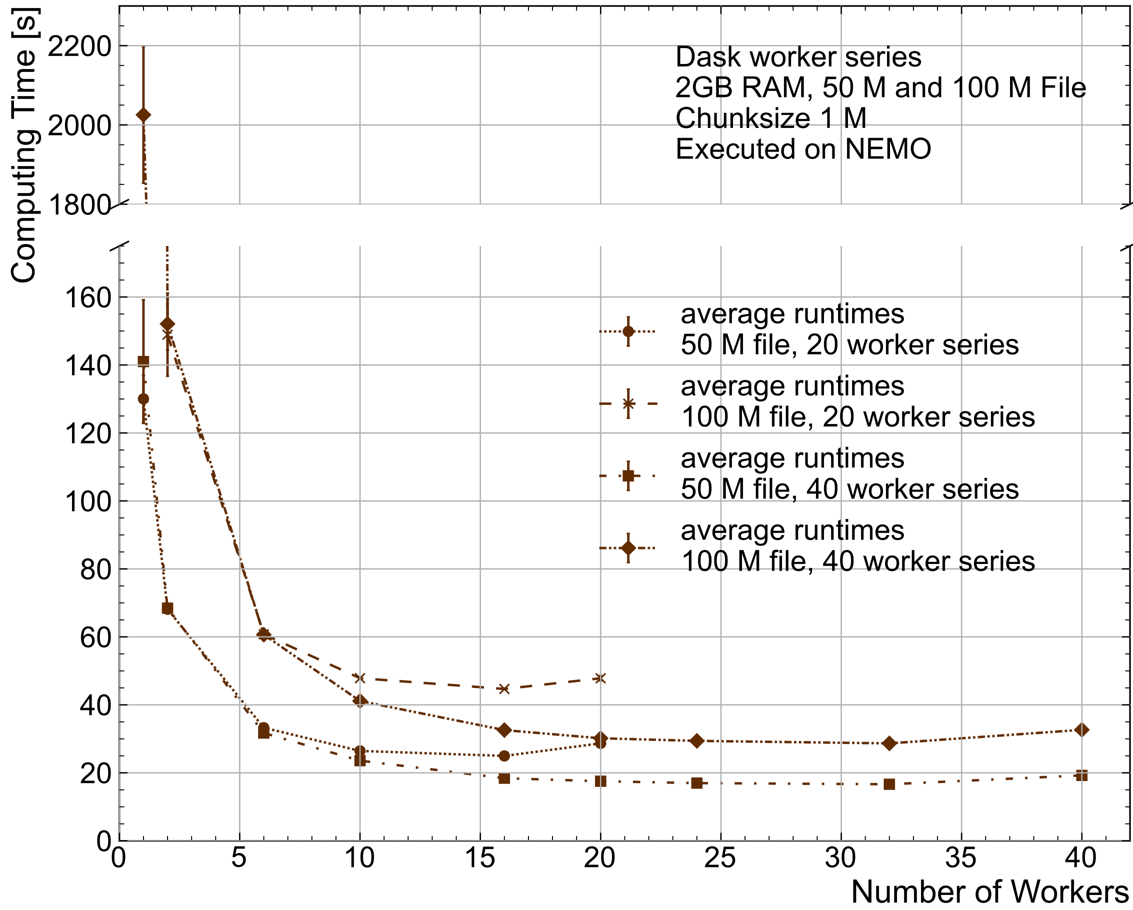


Figure 6.12: Comparison of the computing time with the 50 M and 100 M events files, starting from 20 and 40 workers. The scale differs between the upper and lower part of the figure. The numerical values are shown in table B.18

6.3 Comparison between the two clusters

In this section, the performance of both clusters is compared to each other. Five benchmark series are measured, with one utilizing the workers of ATLAS-BFG for computations. The four remaining series are running on worker nodes of NEMO, accessed in two different ways. In the first section, these different ways of accessing the resources in NEMO are discussed. The second section discusses the performance differences between the subseries.

6.3.1 BFG Slurm on NEMO

While most subseries in this measurement are running as described in sections 6.1 and 6.2, one subseries is utilizing worker nodes of NEMO as if they are a part of ATLAS-BFG. This can be done by submitting jobs to the vNEMO queue of Slurm on the ATLAS-BFG. As illustrated in figure 6.1, this sends the workload to NEMO worker nodes. While the user experience is straightforward, many systems work interconnected to enable this functionality. Summarized, based on the demand in the Slurm scheduler, worker nodes on NEMO are requested. They start a virtual machine (VM), which connects to the ATLAS-BFG scheduler, Slurm, to receive its workload. Since the actual computing workload is not

send by Moab, the singe-user-policy is bypassed. Jobs requested by multiple users can run on each physical worker node, enabling better resource utilization while sacrificing isolation of workflows.

6.3.2 Performance measurements

In this section, the performance of ATLAS-BFG, NEMO and vNEMO is compared. While using the same hardware, some differences in software and configuration exist between them. To measure the influence of the clusters' configuration on computing time, five different subseries are measured. Measuring the computing time for a range of datasets, the performance of one worker node of the clusters is compared. This is done by keeping the datasets used, chunk size and, for the most part, the amount of memory is constant between subseries.

Keeping the number of physical CPU cores utilized constant for the subseries is not as simple. On the worker nodes used by ATLAS-BFG and vNEMO, the CPUs have hyperthreading enabled. This results in 40 logical cores available per worker node, so 40 Dask workers are used in the first two subseries running on ATLAS-BFG and vNEMO. The worker nodes on NEMO do not enable hyperthreading, resulting in 20 logical cores available per worker node to completely saturate the CPU. The third subseries, the first running on NEMO is using 20 Dask workers of the standard configuration. Using half the number of workers is severely limiting the usable parallelization and total available memory for the analysis. For this reason, two extra subseries are measured on NEMO. In the first one, the Dask workers are allocated 4 GB RAM, reducing the differences between this and subseries and the first two. The last subseries is running with 40 standard Dask workers on NEMO. While increasing comparability in the Dask workers, this subseries needs two worker nodes on NEMO, contrasting the other four.

The results of this series are visualized in figure 6.13. On the datasets containing up to 250 million events, the 40 worker NEMO subseries outperforms the other subseries. The vNEMO and ATLAS-BFG subseries perform comparable.

When reviewing the numerical values in table B.19 vNEMO outperforms the ATLAS-BFG by 17% on the 50 million events dataset and by 15 % on the 100 million events dataset.

The 20 worker series on NEMO perform consistently worse compared to the others. For datasets larger than 100 million events, the 20 worker series with 2 GB RAM performs significantly worse than the others.

The difference between each of the 20 worker subseries is most likely due to the limit in total memory, shown by the increasing computing time for larger datasets. Running on 20 workers also significantly reduces the parallelization of the workload, increasing the load on each worker and reducing performance compared to the other three series. The better performance of the 40 worker subseries on NEMO is within expectation, because doubling the number of physical CPU cores should increase performance on parallel workloads. For larger datasets, the ATLAS-BFG subseries outperforms it, due to a decrease in overall load per used worker. With the single-user-policy, the NEMO worker nodes running the analysis are completely utilized. The ATLAS-BFG worker nodes are utilized less than 100% and only host two Dask workers each. This influence is only relevant for the larger datasets. For the smaller datasets, the decreased communication overhead by running on fewer machines and better performance of non-hyperthreaded CPU cores gives NEMO the advantage. The

increased overhead of the VM contribute to the worse performance of vNEMO on the larger datasets.

Running the analysis on hyperthreaded CPU cores enables this to happen on fewer physical machines. The performance gained by increasing the number of workers has a higher impact on performance than running with hyperthreading disabled. The total amount of memory also influences performance, but an increases in parallelization outweigh increases in allocated memory. Running on fewer worker nodes is advantageous for smaller datasets, but the increased load per machine hurts performance on larger datasets.

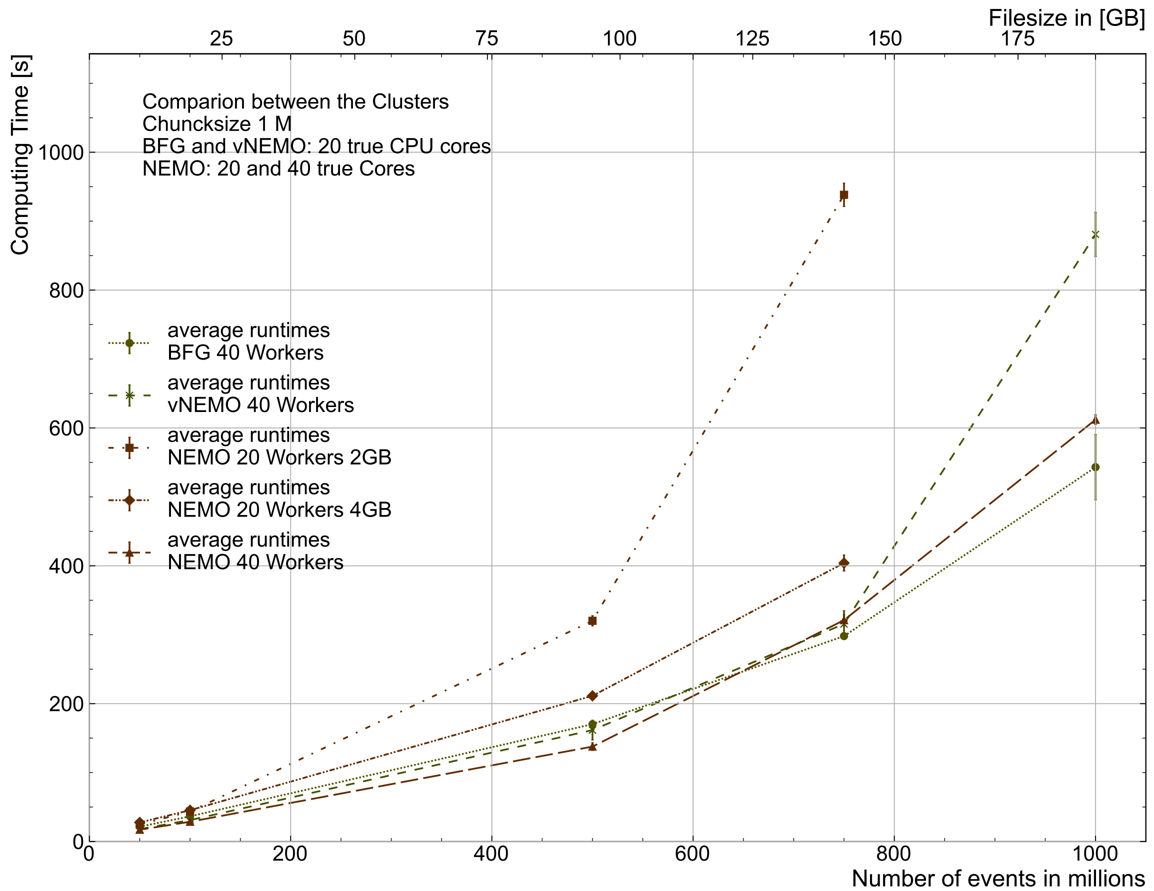


Figure 6.13: Comparison of the computing time between the three different ways of accessing the two clusters. The numerical values are shown in table B.19

7 Runtime Analysis on a representative Dataset

The measurements in the previous sections analyze the performance of the interactive analysis on monolithic datasets. In typical physics analyses of PhD theses, the datasets differ on file level. They are not monolithic, but distributed over multiple files of different sizes. For this reason, the file distribution of the $HH \to bb\ell\ell + E_{\mathrm{T}}^{\mathrm{miss}}$ analysis by Benjamin Rottler [51] is analyzed and a model data set with a similar distribution of number of files and file sizes representing 1 % of the total analysis is created. The model dataset is based on the Higgs-Gamma-Gamma dataset and includes approximately 540 million events distributed over 58 files. More information on it can be found in section 7.2.

7.1 The $HH \to bbll + E_{\mathbf{T}}^{\mathbf{miss}}$ Analyzis

The thesis "Search for Higgs-Boson Pair-Production in the $bb\ell\ell + E_T^{miss}$ final state with the ATLAS detector at $\sqrt{s} = 13$ TeV" [51] measures the signal strength and cross-section of Higgs-boson pair-production of the ggF and VBF production modes. Additionally, a constraint on the self-coupling modifier $\kappa_\lambda = \lambda/\lambda^{\mathrm{SM}}$ is set. λ denotes the measured trilinear Higgs-boson self-coupling strength, λ^{SM} the value predicted by the SM. To achieve this, the full dataset taken with the ATLAS detector during Run-2 of the LHC was analyzed. The integrated luminosity of the dataset is $139\,\mathrm{fb}^{-1}$. While the complete data collected by the ATLAS detector in this timeframe is orders of magnitude larger, after preprocessing and preselection specific to the PhD thesis, the dataset size is reduced to 8 TB.

7.2 Preparation of the representative dataset

Before the representative dataset can be generated, the reference dataset needs to be analyzed. This is done by extracting the file size distribution of the reference dataset. Then, the representative dataset is generated based on the Higgs-Gamma-Gamma dataset. More details on both procedures are given in this section.

Analysis of the reference dataset The file list of the reference analysis is exported from rucio [52] in form of a CSV file. This CSV file contains fields for, among others, the scope and name of the individual files as well as their file sizes. Figure 7.1 shows a frequency distribution of the different file sizes. To keep the number of files per bin roughly equal, a modified logarithmic scale is used.

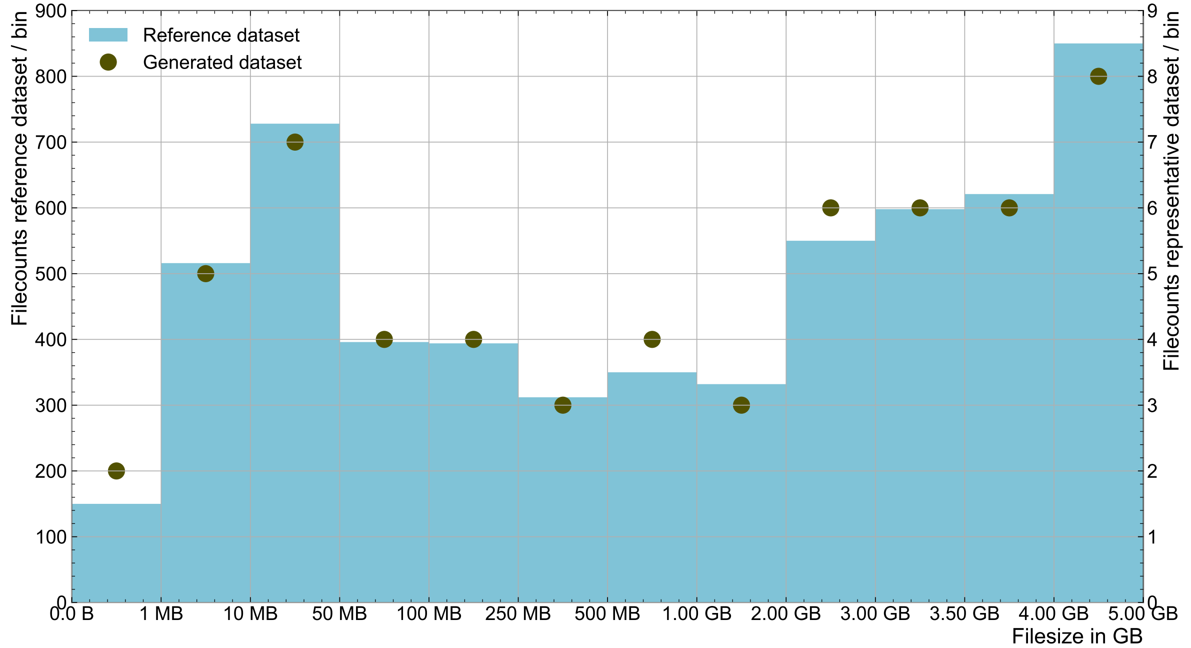


Figure 7.1: Comparison of the file distributions of the generated and reference dataset. The file numbers per bin of the reference dataset are marked on the left y-axis. On the right x-axis, the same is shown for the generated dataset.

Generating the representative dataset According to the file distribution in the reference dataset, new files based Higgs-Gamma-Gamma dataset are generated. The targeted sizes are the bin centers in the histogram, e.g. 500 kB, 5.5 MB, and so forth. For the total size of the representative dataset, 1% of the HH-analysis is targeted. Information on the frequency, how often each file type occurs in the final dataset, can be found in table 7.1. Similar to the dataset used in the other sections, ROOT is used to generate the new files for this dataset. More details on the previously generated files can be found in section 3.1.

The specific process used to generate the files uses a larger file and splits a specified number of events into a new one. To establish a relationship between file size and the number of events inside a ROOT file of the dataset used in most parts of this thesis, a linear model is fitted to the previously generated files up to the 30 million events file. The others are not included, because they are significantly larger than the files required for the representative dataset. More details to the linear model can be found in figure A.20. With this linear model, the number of events for each required file size is determined: For the 500 kB file, 2551 events are needed, for the 5.5 MB file 28921 events and so forth. The events needed for the other sizes can be found in table 7.1.

Table 7.1: List of the files for the representative dataset. The number of events used for the monolithic file is also included.

File size [MB]	Number of Events	Replications
0.5	2551	2
5.5	28921	5
30	158138	7
75	395476	4
175	922893	4
375	1977726	3
750	3955538	4
1500	7911162	3
2500	13185328	6
3250	17140953	6
3750	19778036	6
4500	23733660	8
Σ	542514147	1

Using the file containing 30 million events, all files for the representative dataset are created. While the actual file sizes deviate from the targeted sizes, they are well within their respective bins. This can be verified in table B.20. In order to get a good representation of the HH-analysis, the newly created files are replicated as depicted in figure 7.1 and listed in table 7.1: The 500 kB file two times, the 5 MB file five times and so on. With this, every file is loaded once and any influence of multiple workers accessing the same file is eliminated. Additionally, a monolithic version is generated with the same number of events as the entire dataset has. Due to file size constraints in ROOT, this results in two separate files. The source file for this dataset is the one billion events dataset.

7.3 Performance measurements

The analyses of both, the representative dataset and monolithic equivalent to it, are executed on the ATLAS-BFG using 48 workers where each worker is allocated 2 GB RAM. The chunk size is set to one million events. The analysis is run 10 times on each dataset to compensate for differences in the load of the worker nodes. The mean computing times and their errors are listed in table 7.2.

Table 7.2: Compute times on representative and monolithic datasets executed with 48 workers and 2 GB RAM per worker on ATLAS-BFG. The chunk size is 1 M.

Dataset	Total time [s]
Representative dataset	1284 ± 5
Monolithic dataset	201 ± 4

By loading 58 separate files, a large overhead is generated. The dataset in one single file is more than 6 times faster. This can be compared to the series measured on the bwNOTEBOOK, visualized in figure 4.8 with the numerical values in table B.9. Two of the data points in this series load individual files a similar number of times as the representative

dataset. The first loads the 1 million events dataset 50 times, resulting in a computing time of 101.2 ± 0.4 s. Compared to the 50 million events dataset with a computing time of 39.97 ± 0.15 s, the analysis of the monolithic file is more than two times better. The second data point loads the 1 million events dataset 100 times, enabling a comparison to the 100 million events dataset. Here the repeatedly loaded dataset needs 323.8 ± 1.2 s, while the monolithic dataset needs a computing time of 80.80 ± 0.25 s. This makes the monolithic file on this data point approximately 4 times faster.

These measurements indicate, that the use of larger files increases the performance of the algorithm. The influence of the individual file size can not be determined with the data gathered so far. While the representative datasets files are larger on average, the performance increase when analyzing the monolithic dataset is also larger. That being said, both measurements differ greatly. Running on the bwNOTEBOOK, 6 workers with a total of 12 GB RAM analyze a dataset of 100 million events from local storage. The representative dataset has more than 540 million events and is analyzed by 48 workers with a total 96 GB RAM from a networked storage.

To conclude, the structure of the dataset influences the performance of the algorithm. Monolithic datasets perform much better than datasets distributed over a number of files of different sizes.

8 Conclusion and Outlook

In this thesis, the performance of a parallelized interactive HEP analysis running in Jupyter Notebooks is benchmarked The standard for HEP analyzes is ROOT, a C++ based analysis framework. ROOT based analyzes are often implemented as scripts. With PyROOT, a binding between Python and ROOT exists, enabling easy use of ROOT in an interactive Python environment, like Jupyter Notebook. Interactive analyzes have a lower barrier to entry compared to script based ones. It is possible to further decrease it by switching from PyROOT to the scientific Python software stack, as demonstrated in section 3, while maintaining comparable or better performance and accurate results. When designed with mostly explicit function calls, an algorithm which relies on Uproot, Awkward Arrays and NumPy can be modified for parallelized computation with Dask. In section 4, this is first demonstrated on a single notebook, then scaled to multiple clusters in section 6.

8.1 Conclusion

Vectorizing the reference PyROOT analysis results in great performance gains, with computing times decreasing by an order of magnitude. This allows the utilization of larger datasets than possible with the reference algorithm while significantly decreasing the needed computing times, compared to the reference, when analyzing the same dataset. As seen in figure 3.7, the largest dataset analyzable with the PyROOT algorithm contains 15 million events with a computing time of more than 500 seconds. Both variations of the NumPy algorithm could analyze a dataset containing 50 million events in under 100 and under 200 seconds, depending on the loading mechanism. While comparatively poor performance of the PyROOT analysis could be caused by a memory leak, optimizing the reference analysis is out of the scope of this thesis. The C++ script based version of the reference analysis outperformed the PyROOT version by a wide margin, but the NumPy based analysis is still faster than the C++ reference. The difference is smaller, with the faster NumPy algorithm outperforming it by a factor of approximately 2.
To utilize all resources available on the notebook, the algorithm is parallelized with Dask. This is possible by breaking down the dataset into smaller parts and parallel running computations on these parts. With Dask workers, it is easily possible to scale the analysis to exploit all locally available resources. The six used Dask workers are configured with one thread and 2 GB RAM for each. In practice, the performance increases plateaued when all physical CPU cores are fully utilized, as illustrated in figure 4.7. Working with Dask collections, datasets of almost an order of magnitude larger can be analyzed. Comparing the computing time of the Dask-based algorithm to the fasted NumPy based one on, running on the largest file analyzable by it, the parallelization halves the required computing time. This can be seen in figure 4.6. In this figure, the largest dataset analyzed on the notebook during this thesis is also listed, containing 400 million events with a file size of more than 70 GB and a computing time of less 350 seconds. Loading a file this large is possible, because Dask collections split the original file into smaller parts, determined by the chunk size. A wide range of chunk sizes is determined to be usable, but optimal performance is influenced by multiple variables.

With the structure of Dask and the information gained by running the Dask-based analysis on the notebook, scaling to the clusters is straight forward. Two clusters, the ATLAS-BFG and NEMO, are benchmarked. Running on the clusters, more resources are available. Beginning with an increase of the number of workers, (fig. 6.4, 6.12) saturation

effects start before all physical CPU cores are utilized by Dask workers. This is because the datasets analyzed in this series contain too few events to saturate the CPUs. More workers also make the analysis of larger dataset sizes possible, as seen in e.g. figure 6.13. With 40 workers, it is possible to analyze files containing one billion events in around 10 minutes. Datasets in this scale increase the overhead in the login nodes of the cluster significantly. The Dask scheduler running on the login node needs to create the jobs sent to the Dask workers beforehand. Since the dataset is split into usually smaller pieces which get analyzed in parallel, the Dask scheduler is determining which parts are to be analyzed by each worker and which worker collects intermediate results. With fewer divisions of the dataset, the workload of the scheduler can be decreased. This is done by enlarging the chunk size. While reducing the overhead on the login node, it also results in worse overall performance. The larger blocks in the workers' memory result in an increase of computing time on both clusters and the maximum file size analyzable on NEMO (fig. 6.5, 6.11).

Most of the previously mentioned benchmarks utilize datasets contained in one single ROOT file. Typical HEP analyzes of PhD thesis use datasets consisting of multiple thousands files with a combined size of approximately 10 TB. In section 7, a more representative dataset is generated, based on the dataset of a HEP PhD thesis. After analyzing the file size distribution, a new dataset is generated with events from the Higgs-Gamma-Gamma dataset to match this distribution. This more representative dataset has approximately 1% of the total size of the reference dataset. Additionally, a monolithic dataset containing the same amount of events as the representative dataset is generated. The performance of the Dask algorithm analyzing both datasets is then compared. Utilizing 48 workers on the ATLAS-BFG, the representative dataset is analyzed in more than 21 minutes, while the monolithic dataset needs less than 4 minutes.
For general performance optimizations of an interactive HEP analysis with Dask, the following key parameters and constraining factors are identified. The first parameters influence the Dask workers. Simplest, the number of them, has a large influence. The total available RAM for the analysis scales proportionally to the number of workers, as well as the potential for parallelization of the workload. Second, the amount of RAM allocated to each Dask worker. While performance can be improved by allocating up to 6 GB per worker, this is not feasible in every environment. Increasing these two parameters enables analysis of larger files by either distributing the dataset and intermediate results better or simply having more physical space for similar effects. When too many workers are used, it is possible to underutilize the requested resources. For larger datasets, the overhead in the login node of a cluster increases. To reduce it, the size of the task graph needs to be reduced, e.g. by increasing the chunk size, decreasing the number of blocks, the dataset is separated into. This can also necessitate increasing the amount of memory allocated to the workers. While this is an effective method to decrease the burden on the Dask scheduler, larger chunk sizes also increased the computing time for the same file size. When comparing the ATLAS-BFG and NEMO, the infrastructure on the former is more optimized for interactive analyses. The express queue and reservation make it possible to access the resources of the cluster with shorter waiting times.
To summarize, Dask is an excellent module to scale an interactive HEP analysis. Switching from the standard event loop algorithm based analysis to vectorized or column based algorithms, local parallelization and the utilization of compute clusters is trivial when using Dask. The datasets usable with this setup range from tens of GB locally to hundreds of GB on clusters. This setup combines easy to use Jupyter Notebook environment and the Python software stack, both well known to students, with the functionality of Dask

to easily scale from single machines to computing clusters. In particular for a HEP B.Sc. Thesis analysis, a setup like this should become the standard. It allows students to analyze large datasets in the environment they are familiar with since the first semesters while also serving as an easy introduction to distributed computing.

8.2 Outlook

Although providing an easy starting point for HEP analyses, interactive analyses with algorithm similar to the one in this thesis have limits in terms of dataset size. Further optimizations on the algorithm could change this. While the algorithm as described in this thesis loads the entire dataset into one Dask collection, it is easy to imagine a different one inspired by the normal use of the batch system on the computing cluster. Single files or smaller groups could be analyzed individually by the Dask workers. This could be implemented with a different Dask collection: Dask Delayed. The task-graph size should also decrease with this algorithm, enabling the analysis of larger datasets distributed over multiple files.

When using the algorithm described in this thesis, the login node of the clusters could get overwhelmed if multiple users are running their analyses at the same time. On large datasets, approximately 30 GB RAM is needed per user, degrading the performance for other users of the cluster. A dedicated notebook server, Jupyter hub, could avoid this, with the benefit of having a standardized environment for all users. It is also possible to run a Jupyter server on an interactive NEMO worker node. With this dedicated machine, the aforementioned resource usage on login nodes can be avoided. It can be used to run an interactive analysis with more resources available than a typical notebook, while running the same code.

A Supplementary Figures

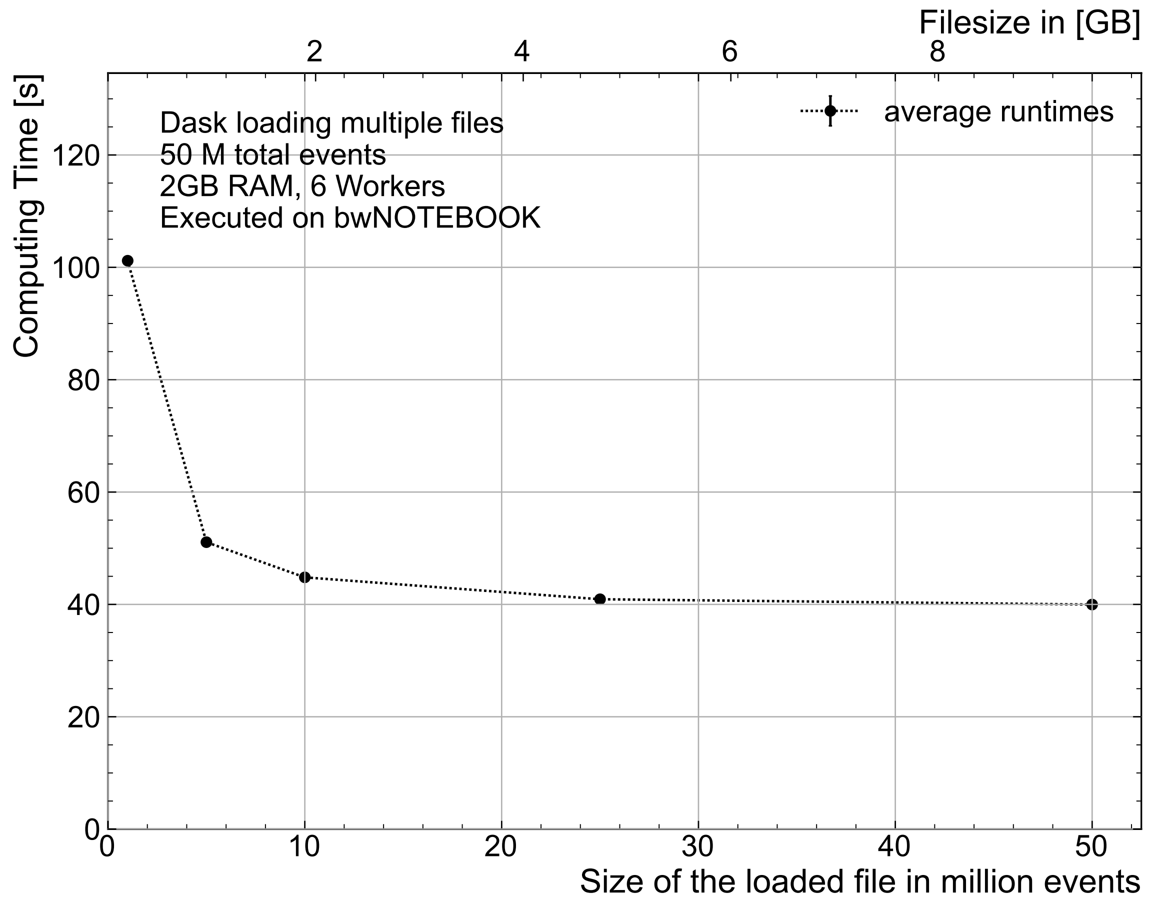


Figure A.4: Comparison of the computing time when loading multiple smaller files with a combined number of events of 50M

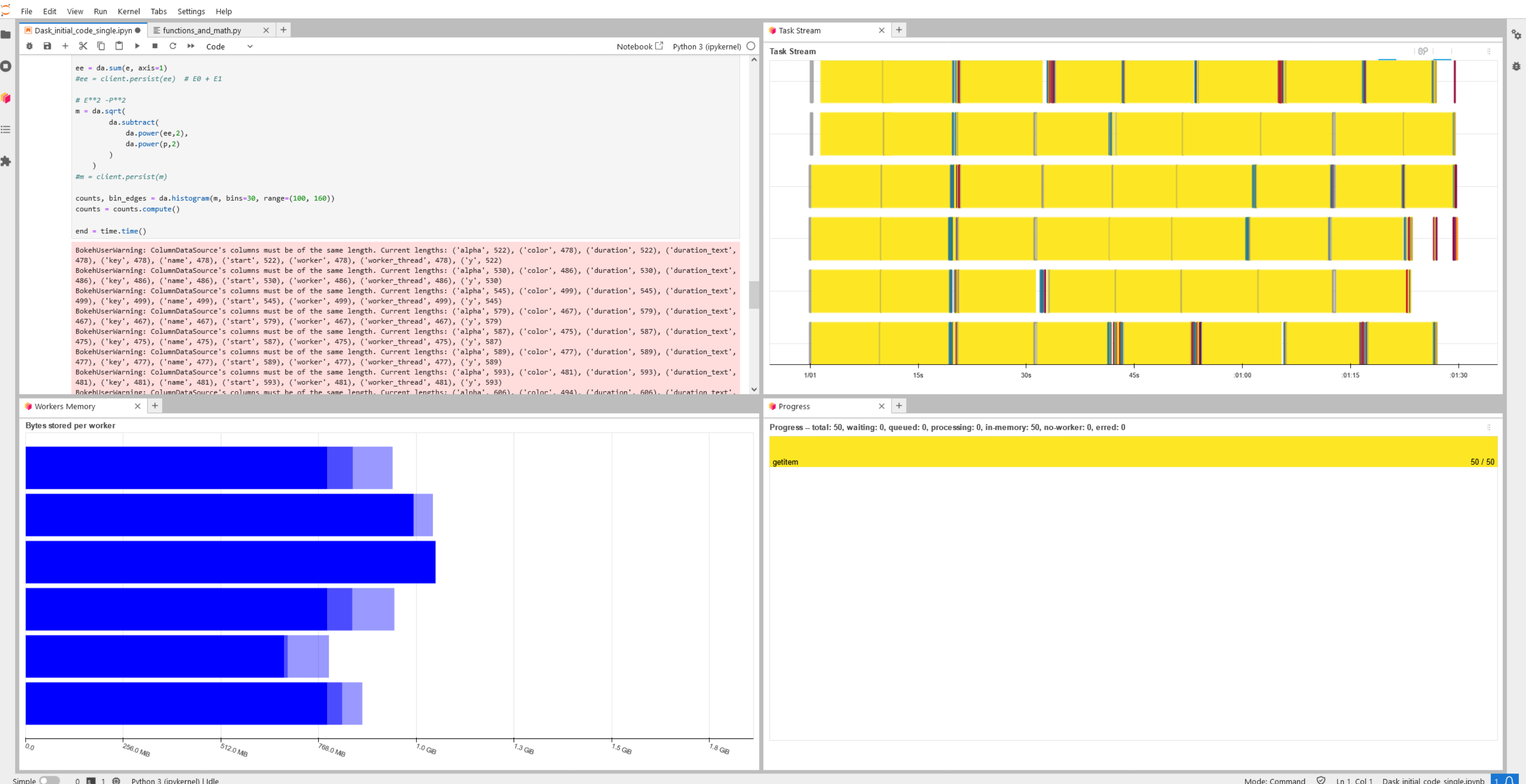


Figure A.1: Screenshot of the Jupyter Lab website after an analysis is finished. In the upper left corner, the Jupyter Notebook is shown. The lower left corner contains the bar graph of the worker memory. The Task Stream is in the plot of the upper right corner. The lower right corner contains the progress display.

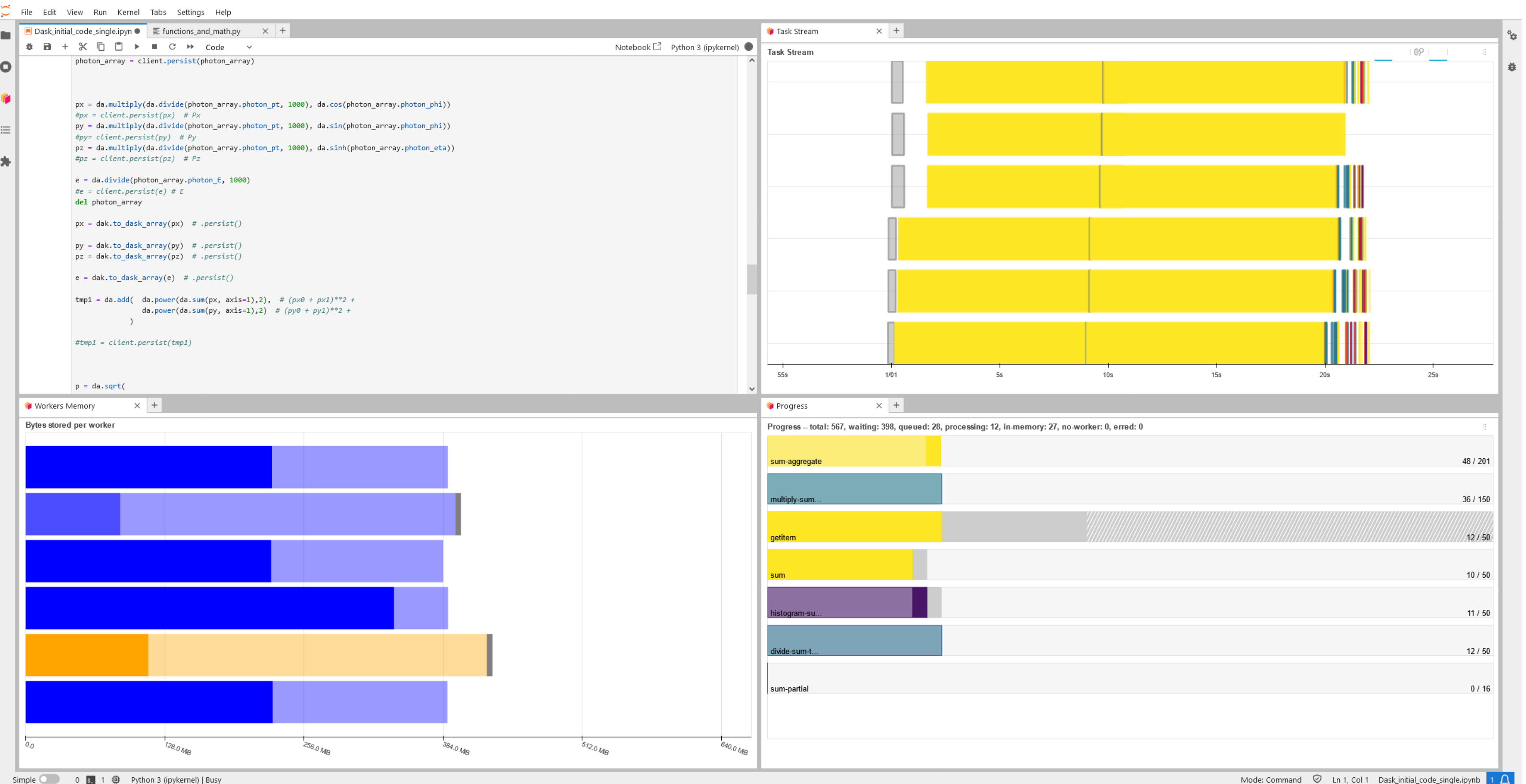


Figure A.2: Screenshot of the Jupyter Lab website with a poorly running analysis. In the upper left corner, the Jupyter Notebook is shown. The lower left corner contains the bar graph of the worker memory. The orange memory indicator is a warning, that not RAM is allocated to the workers, especially when it appears this early into the run. The Task Stream is in the plot of the upper right corner. The lower right corner contains the progress display.

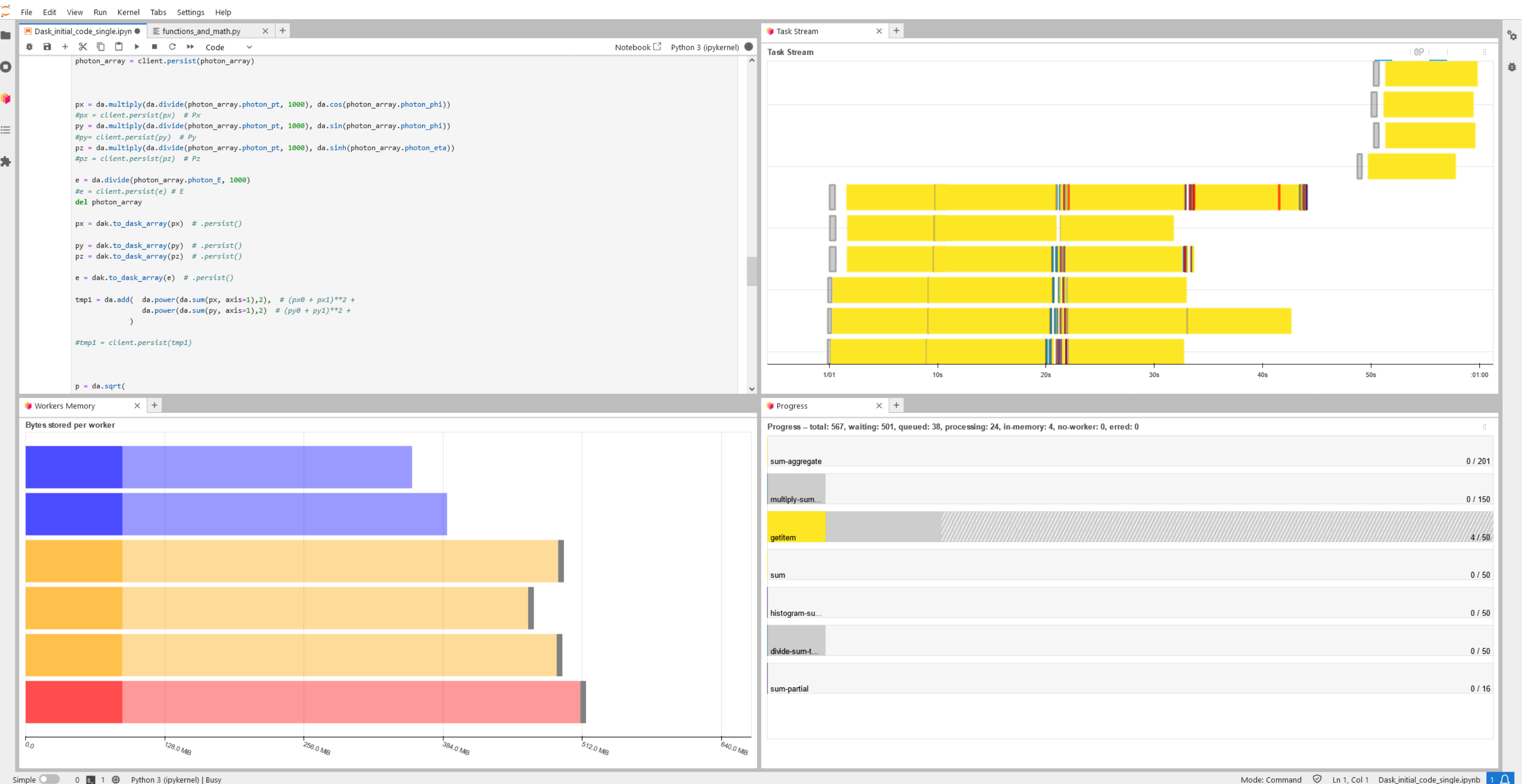


Figure A.3: Screenshot of the Jupyter Lab website with a running analysis, where the workers have not enough memory. In the upper left corner, the Jupyter Notebook is shown. The lower left corner contains the bar graph of the worker memory. The Task Stream is in the plot of the upper right corner. It is visible, that the workers are restarting by the new lines appearing in it. The lower right corner contains the progress display.

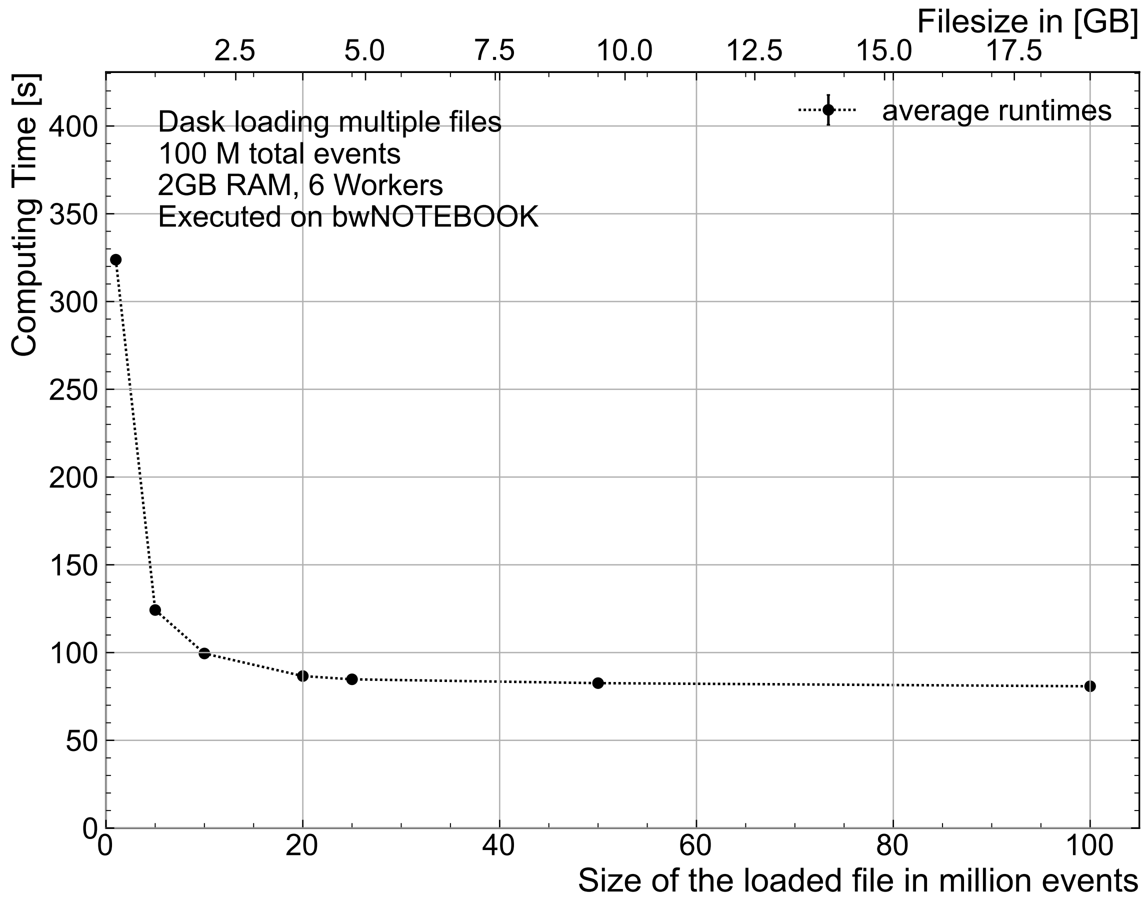


Figure A.5: Comparison of the computing time when loading multiple smaller files with a combined number of events of 100M

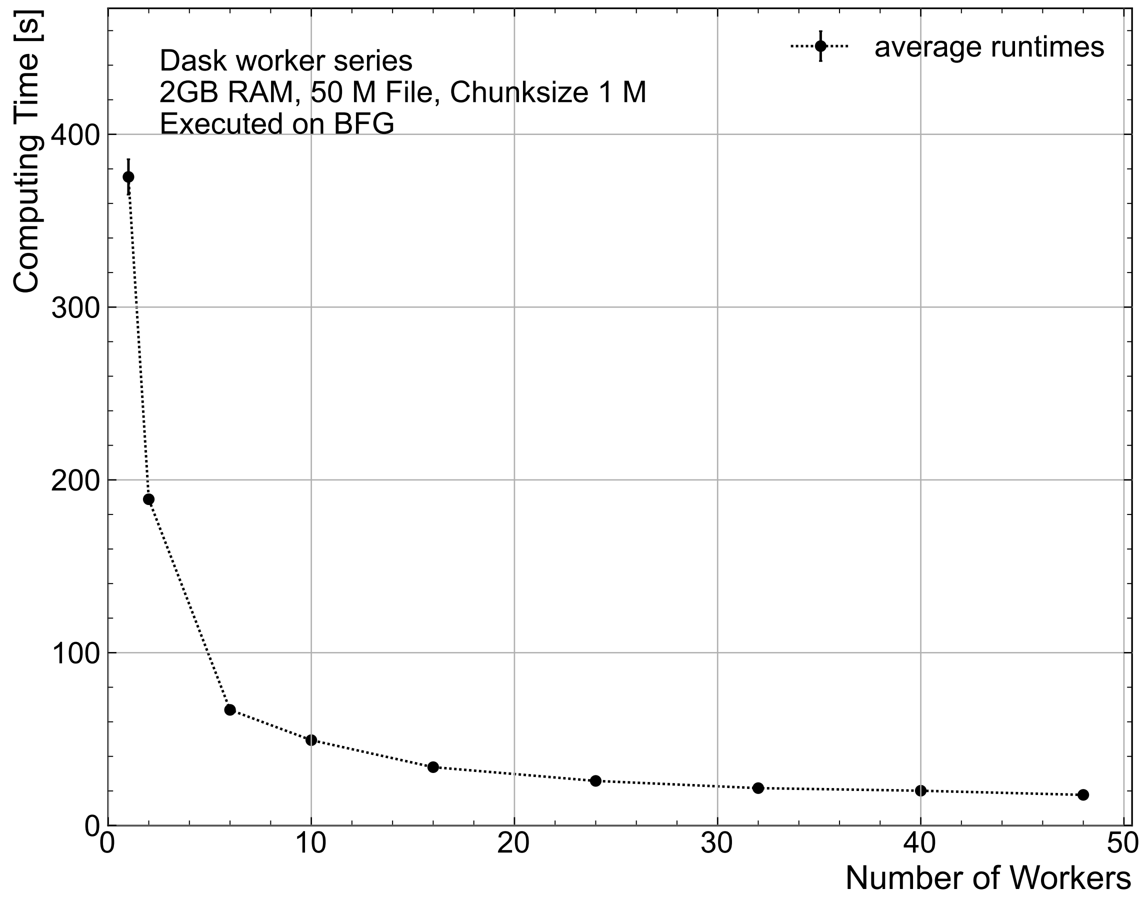


Figure A.6: Comparison of the computing time for varying amounts of workers on the 50M events file

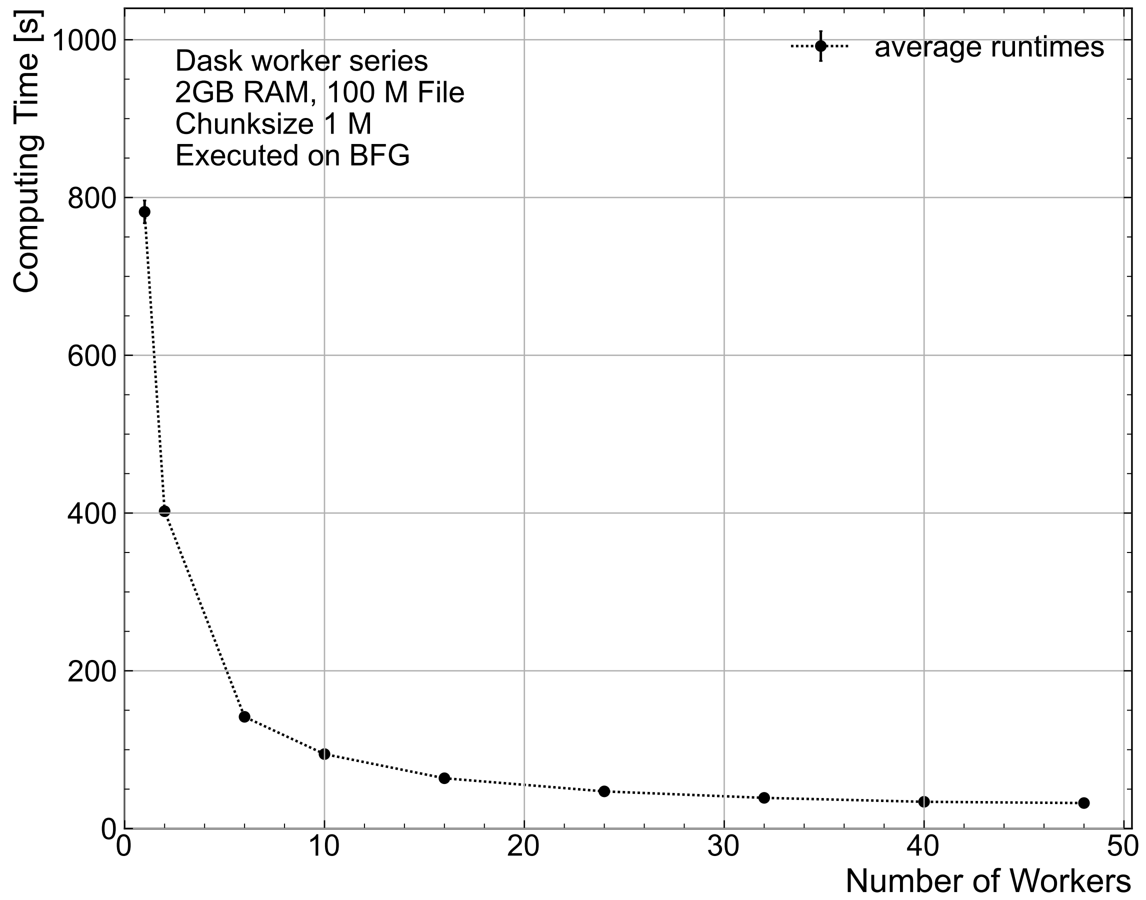


Figure A.7: Comparison of the computing time for varying amounts of workers on the 100M events file

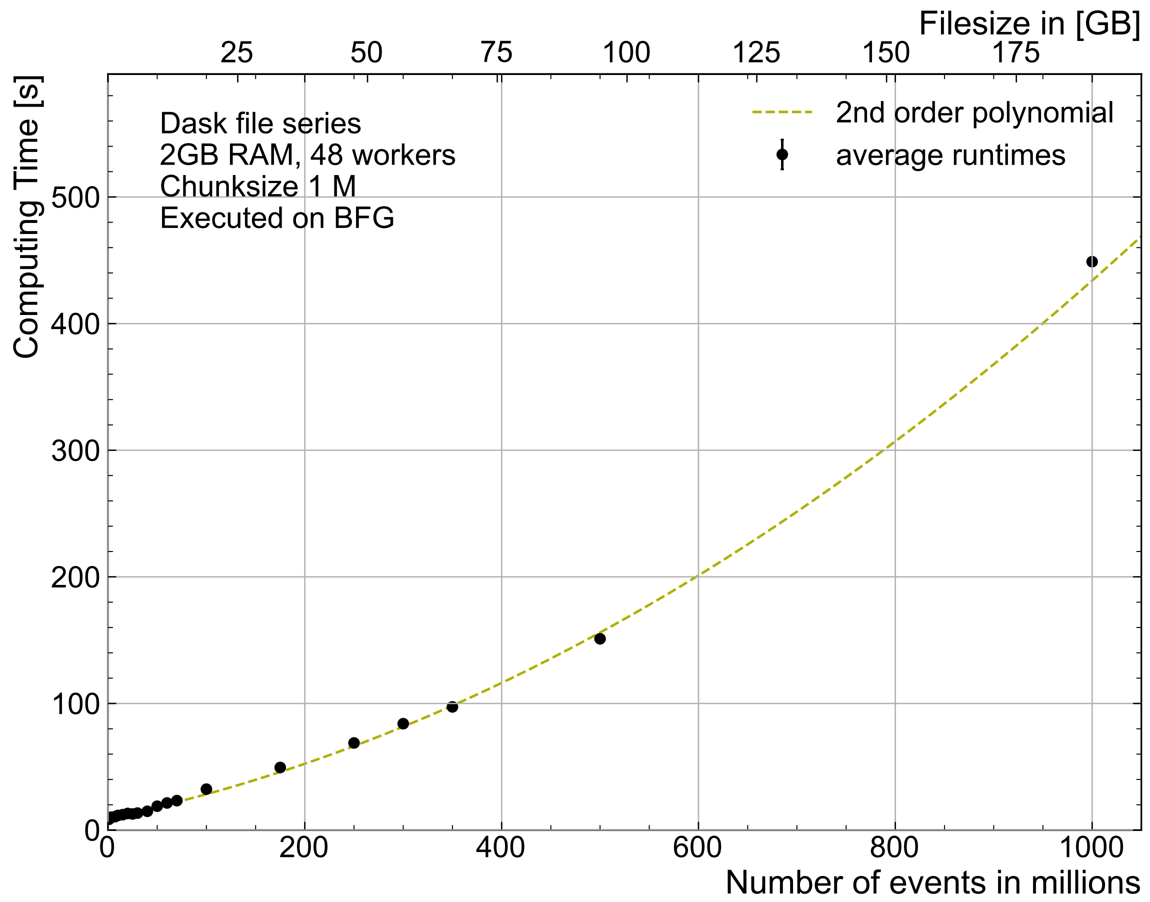


Figure A.8: Comparison of the computing time over multiple sizes of Datasets. 48 Workers with each using one thread with a chunk size of 1 M events.

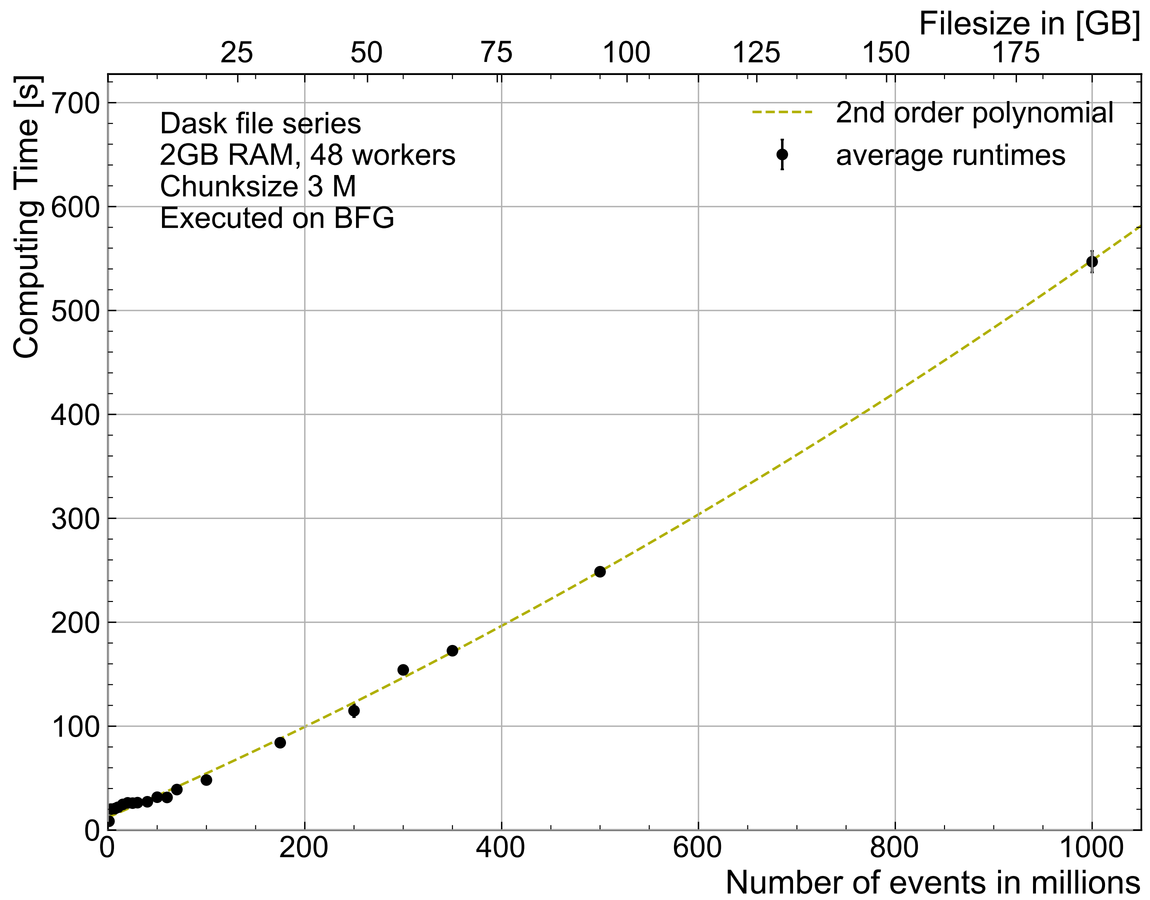


Figure A.9: Comparison of the computing time over multiple sizes of Datasets. 48 Workers with each using one thread with a chunk size of 3 M events.

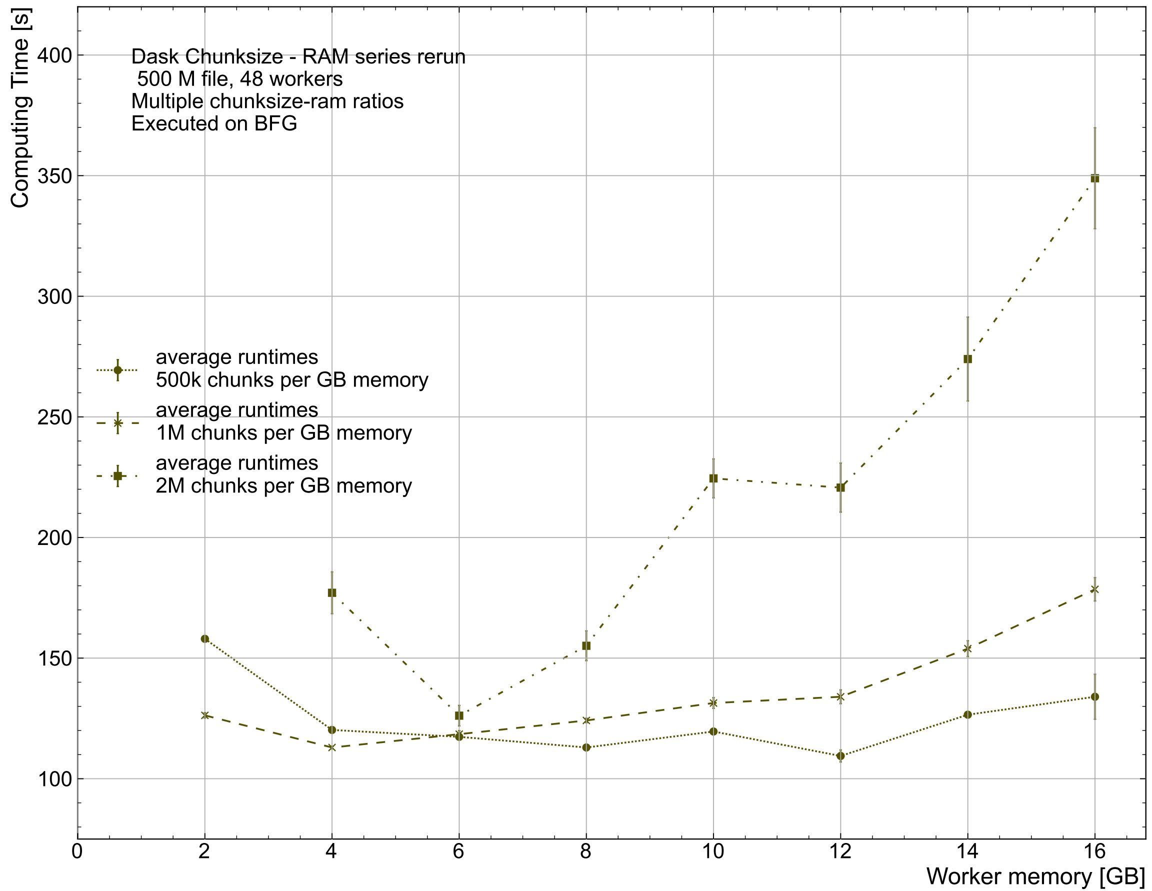


Figure A.10: Remeasured version of the comparison of the computing time for different chunk sizes and RAM allocated to workers, using the 500M events file. For 2 GB RAM, only chunk sizes of 1 and 2 million events could be measured. In this series, two parameters are varied: The chunk size and allocated RAM of the workers. This figure displays the relationship of the RAM size and computing time. The connection between the absolute chunk size and the computing time is visualized in fig. A.11. The numerical values are shown in table B.15

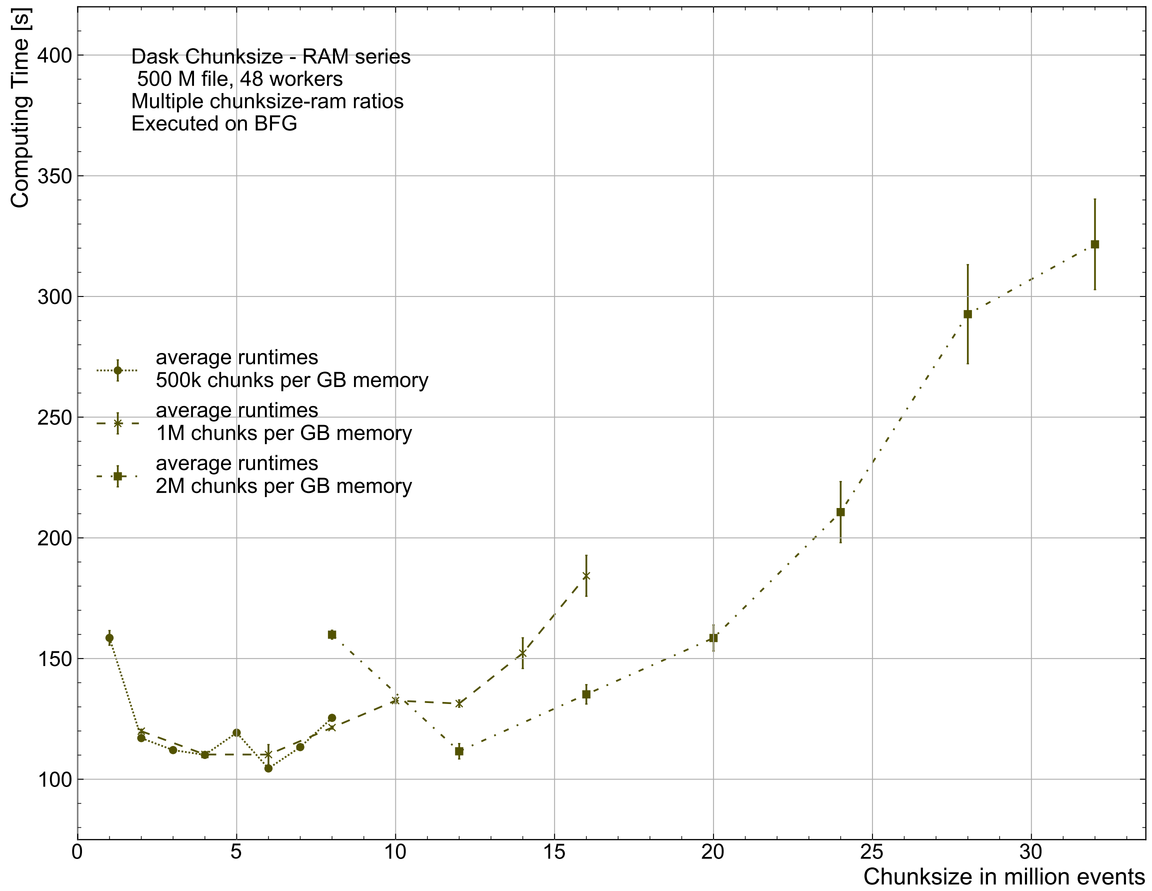


Figure A.11: Comparison of the computing time for different chunk sizes and RAM allocated to workers, using the 500M events file. For 2 GB RAM, only chunk sizes of 1 and 2 million events could be measured. In this series, two parameters are varied: The chunk size and allocated RAM of the workers. This figure displays the relationship of the absolute chunk size and computing time. The connection between the allocated RAM and the computing time is visualized in fig. A.10. The numerical values are shown in table B.16

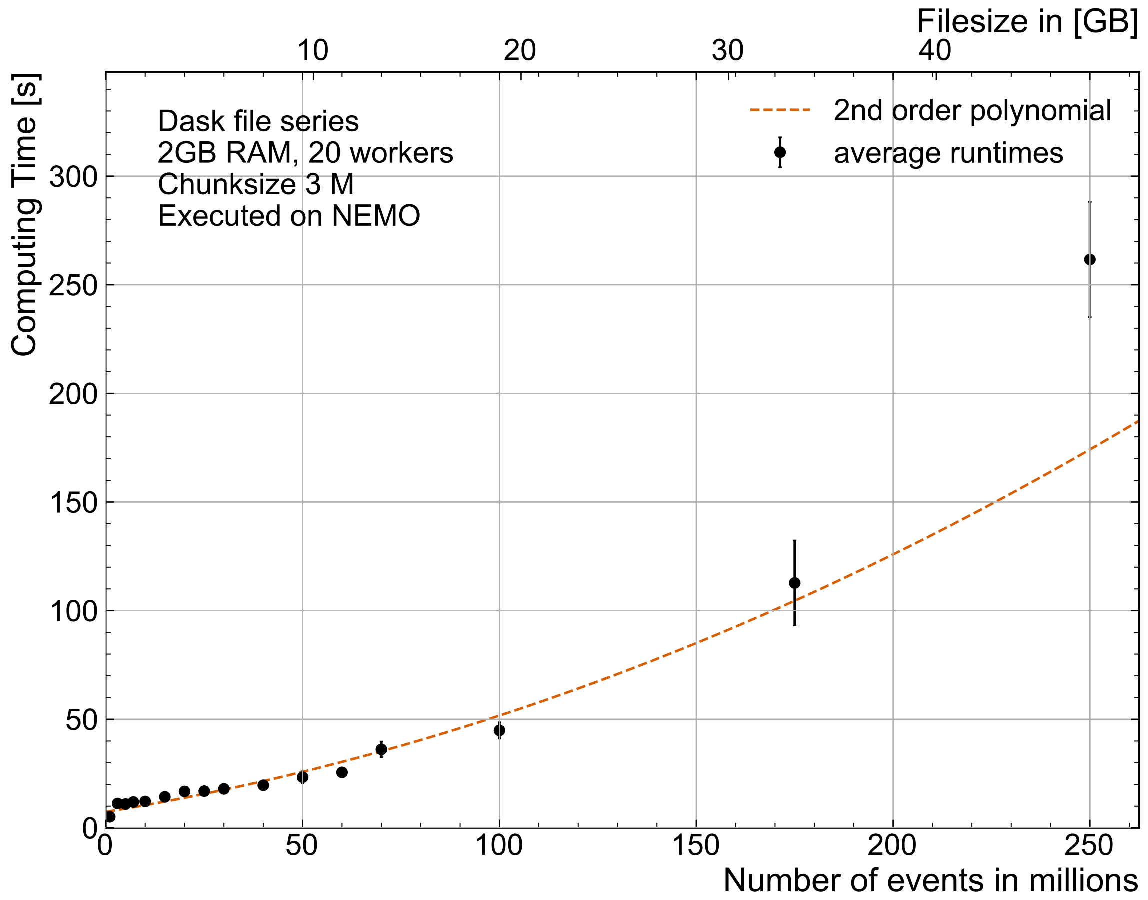


Figure A.12: Comparison of the computing time over multiple sizes of Datasets with a chunk size of 1 million events.

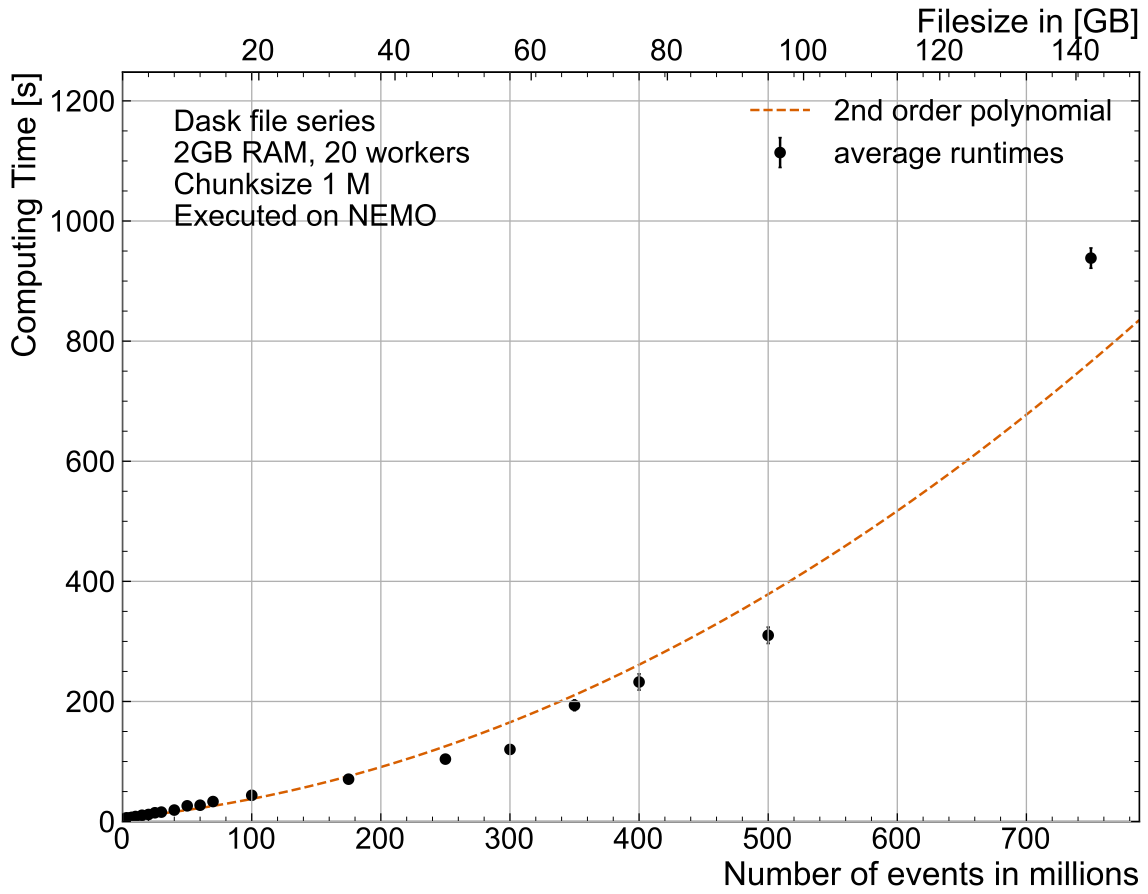


Figure A.13: Comparison of the computing time over multiple sizes of Datasets with a chunk size of 3 million events.

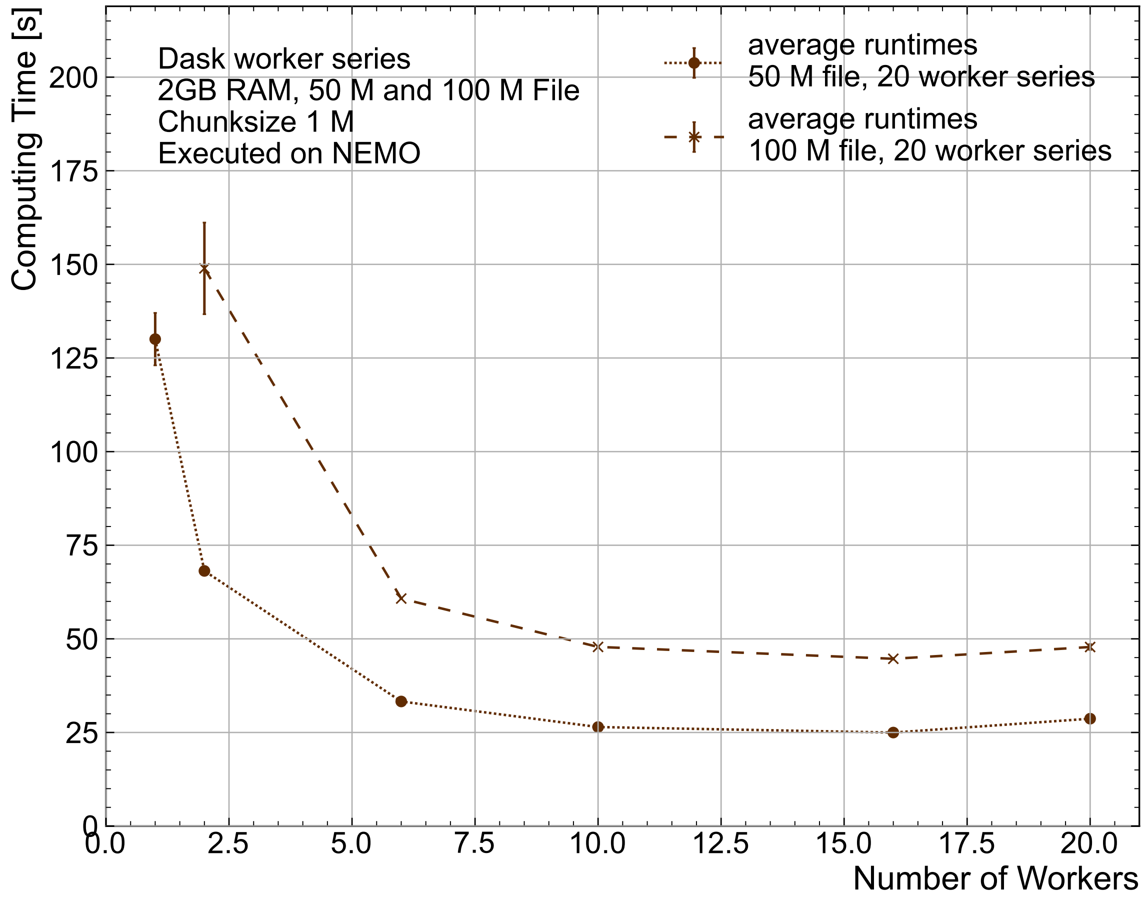


Figure A.14: Comparison of the computing time with the 50 M and 100 M events files, starting from 20 workers. The sale of the upper part of the figure differs from the lower part.

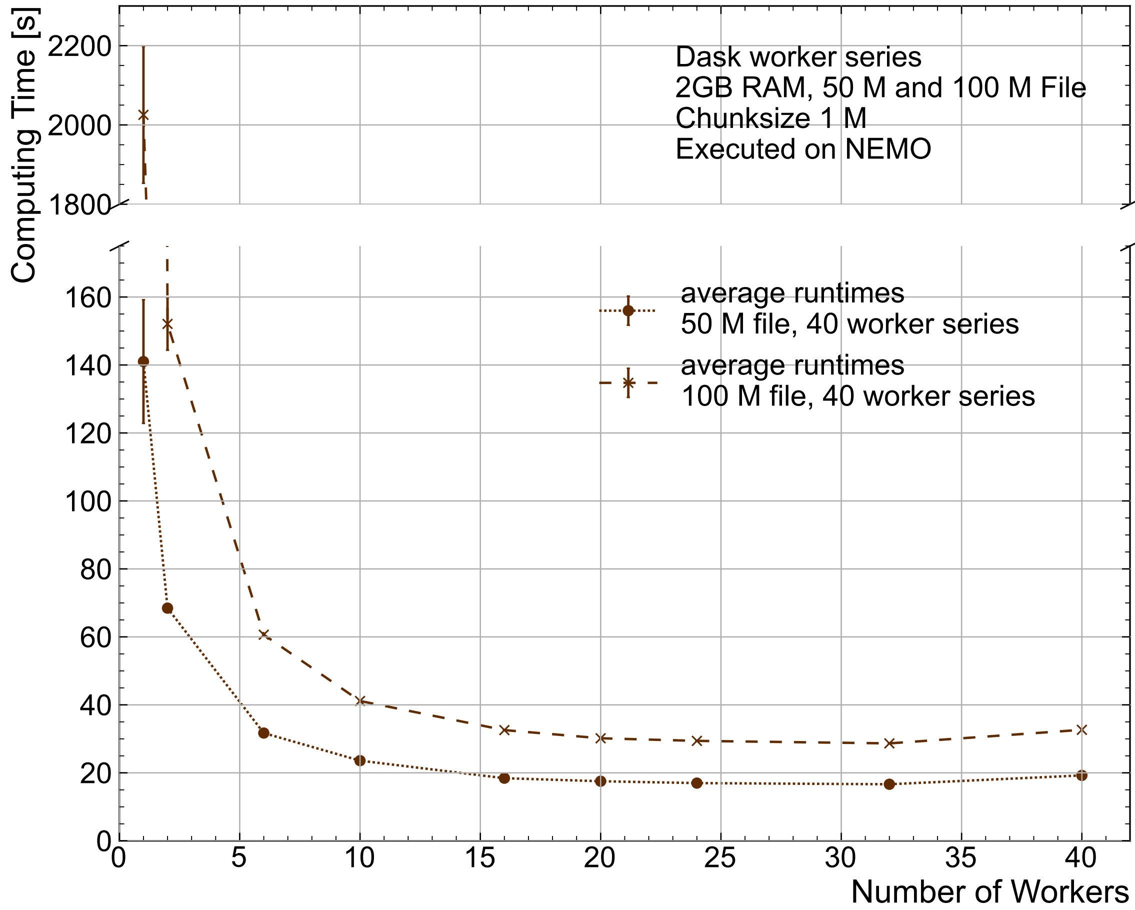


Figure A.15: Comparison of the computing time with the 50 M and 100 M events files, starting from 40 workers. The sale of the upper part of the figure differs from the lower part.

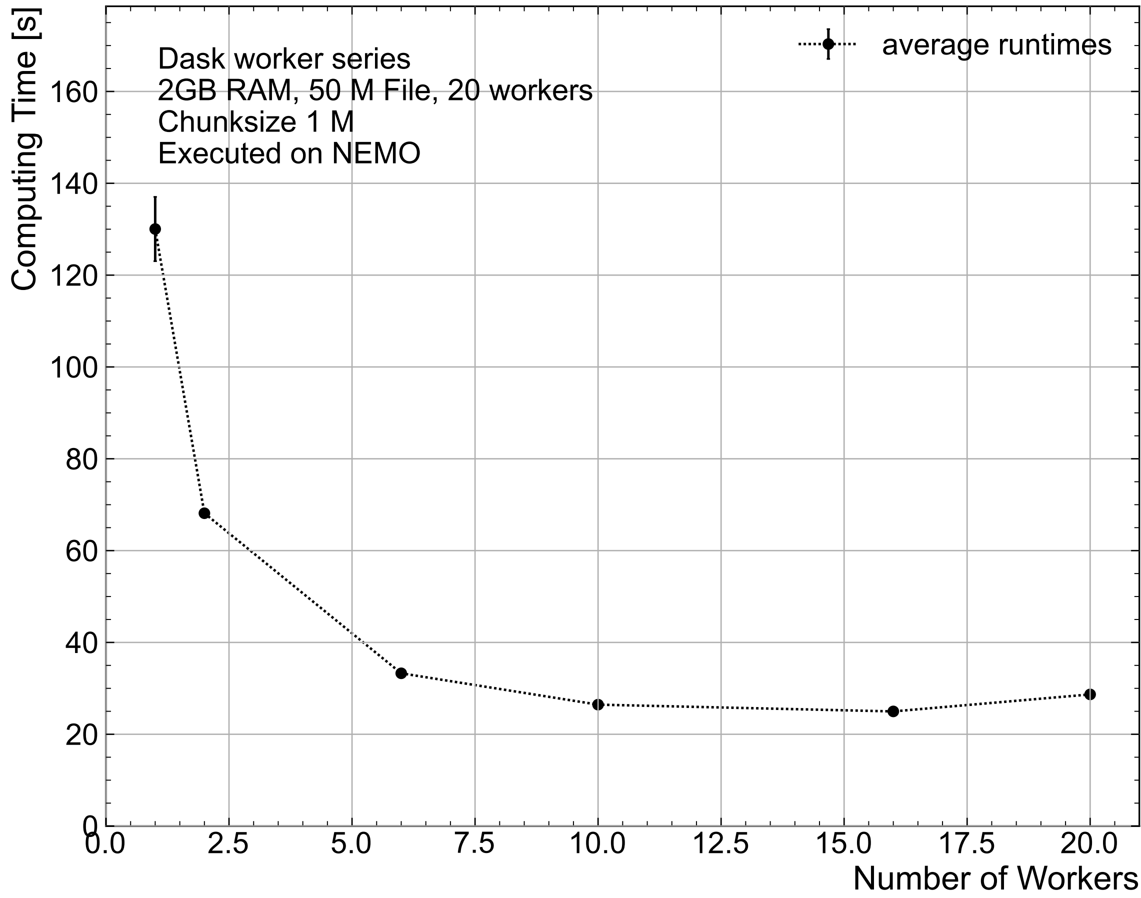


Figure A.16: Comparison of the computing time with the 50 M events file, starting from 20 workers. The sale of the upper part of the figure differs from the lower part.

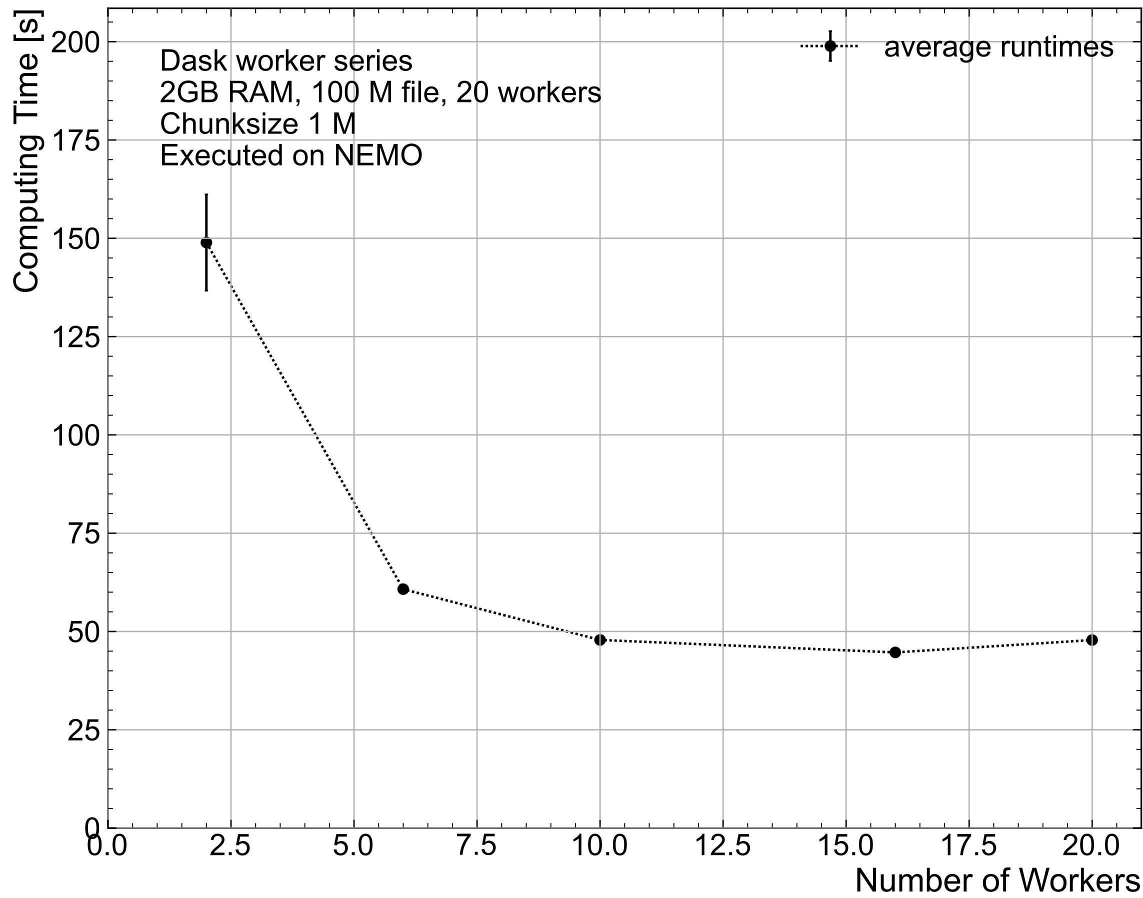


Figure A.17: Comparison of the computing time with the 100 M events file, starting from 20 workers. The sale of the upper part of the figure differs from the lower part.

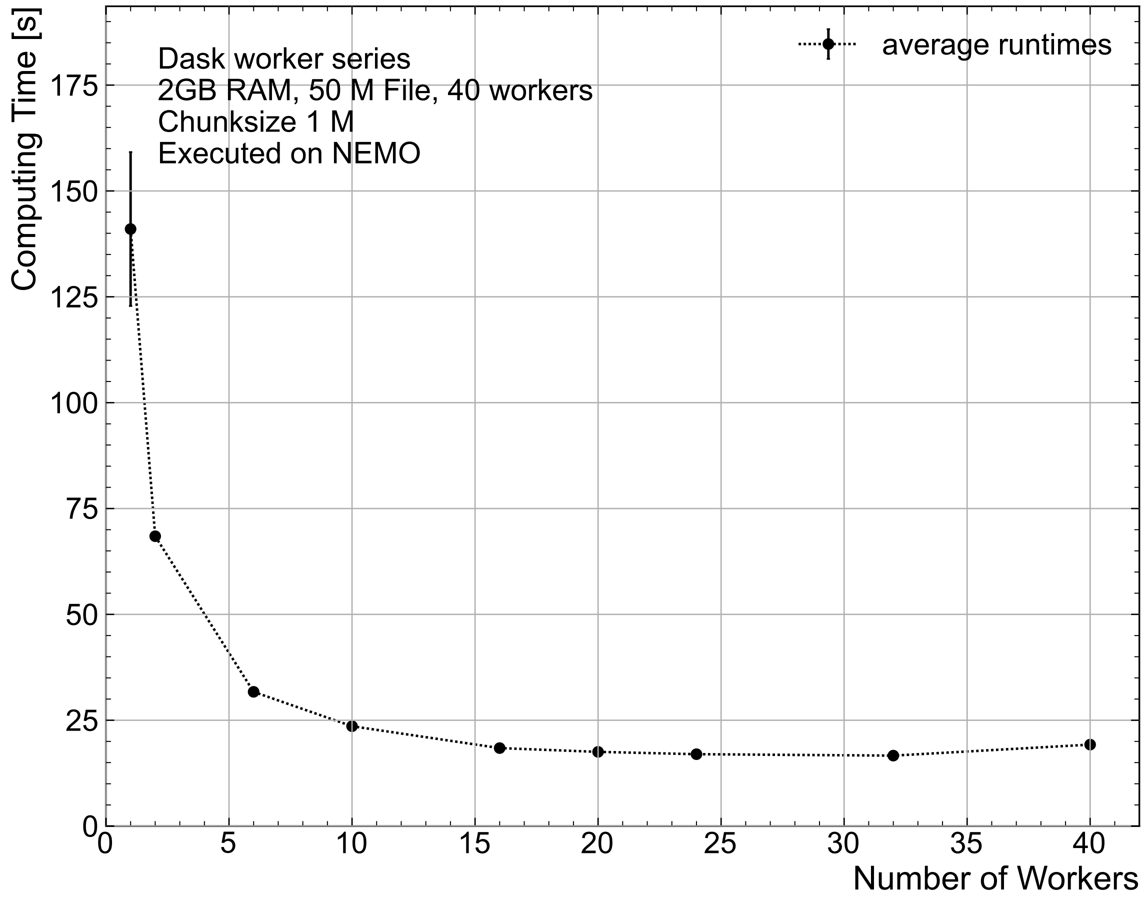


Figure A.18: Comparison of the computing time with the 50 M events file, starting from 40 workers. The sale of the upper part of the figure differs from the lower part.

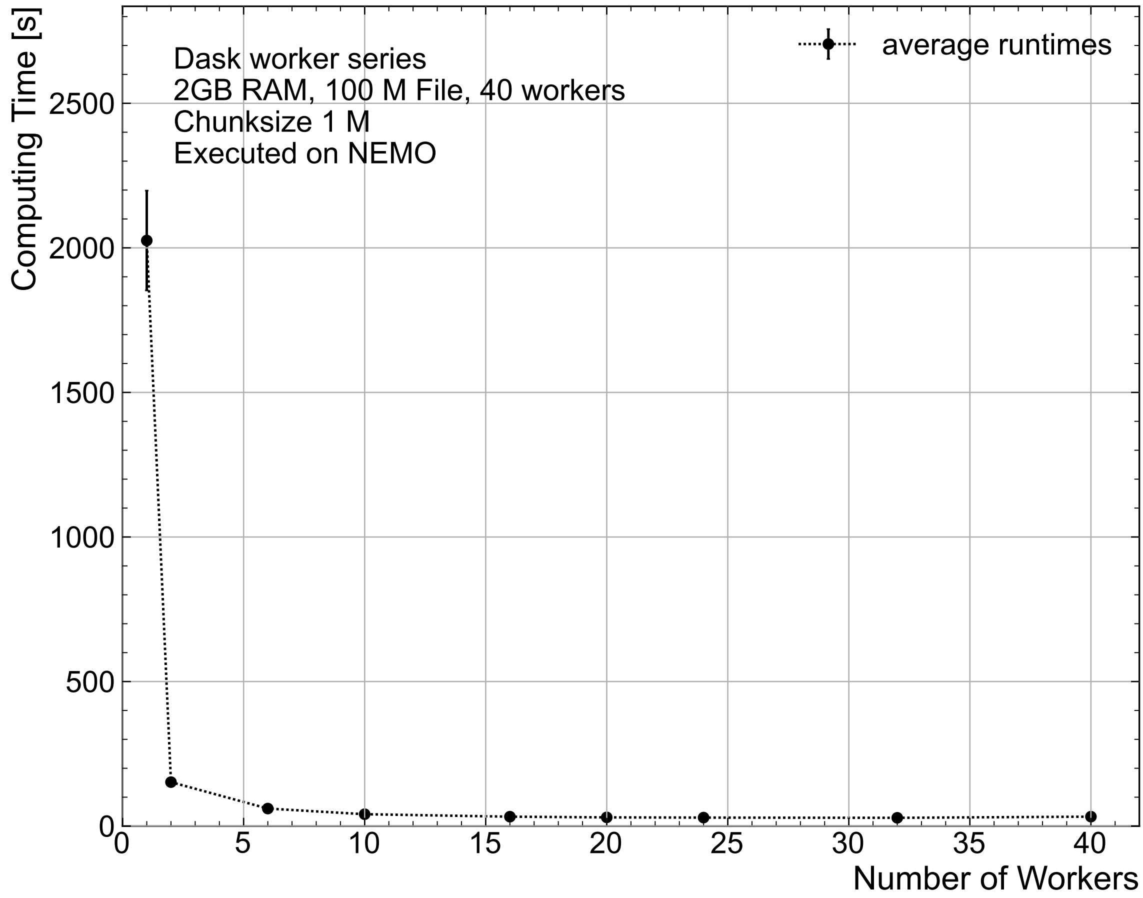


Figure A.19: Comparison of the computing time with the 100 M events file, starting from 40 workers. The sale of the upper part of the figure differs from the lower part.

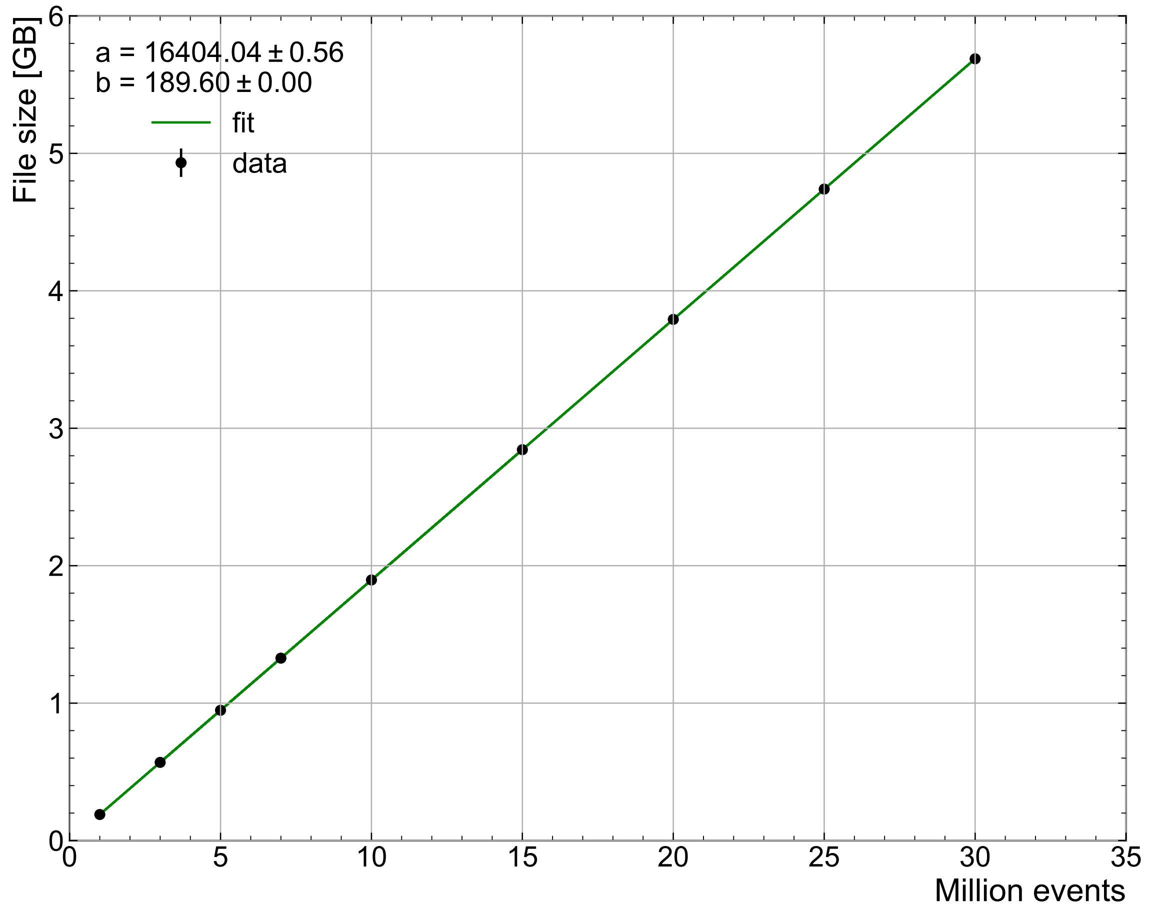


Figure A.20: This figure contains a plot of the file sizes of the datasets with less than 30 million events and linear model fitted to them. The parameters contribute according to this formula: (File size) $= a + b *$ (Number of events).

B Supplementary Tables

Table B.1: Table listing the sequential number of events retained after each cut for all measured validation series.

		PyROOT	NumPy	Dask locally run	Dask on BFG	Dask on NEMO
Cut 0	All events in the dataset	7798424	7798424	7798424	7798424	7798424
Cut 1	Photon Trigger	7798424	7798424	7798424	7798424	7798424
Cut 2	($\|p_{\mathrm{T}}\| > 25$GeV and $\|\eta\| < 2.37$) excluding ($\|\eta\| < 1.37$ or $1.52 < \|\eta\|$) and $N_{\gamma} = 2$	979404	979404	979404	979404	979404
Cut 3	Leading photon: Isolation requirements for E_{T} and p_{T} met	644574	–	–	–	–
Cut 4	Sub-leading photon: Isolation requirements for E_{T} and p_{T} met	362972	362972	362972	362972	362972

Table B.2: Runtime comparison of the PyROOT algorithm and both NumPy algorithms. Executed on the bwNOTEBOOK.

N	NumPy Total time [s]	NumPy in-Memory Total time [s]	PyROOT Total time [s]
1	1.448 ± 0.004	1.440 ± 0.005	36.45 ± 0.12
3	4.156 ± 0.013	4.35 ± 0.05	111.8 ± 0.4
5	7.03 ± 0.12	7.274 ± 0.021	185.7 ± 0.5
7	10.29 ± 0.11	10.36 ± 0.04	260.1 ± 0.7
10	14.08 ± 0.19	17.54 ± 0.20	376.8 ± 1.5
15	21.17 ± 0.28	28.57 ± 0.09	559.7 ± 1.6
20	27.14 ± 0.13	40.63 ± 0.13	N/A
25	33.98 ± 0.21	54.22 ± 0.09	N/A
30	42.77 ± 0.20	68.57 ± 0.25	N/A
40	58.9 ± 0.6	101.92 ± 0.30	N/A
50	77.2 ± 1.2	141.15 ± 0.28	N/A

Table B.3: Loading, compute and total times of the file size series of the NumPy algorithm with standard loading. Executed on the bwNOTEBOOK

N	Loading time [s]	Compute time [s]	Total time [s]
1	0.7711 ± 0.0028	0.6770 ± 0.0028	1.448 ± 0.004
3	2.1810 ± 0.0031	1.9752 ± 0.0031	4.156 ± 0.013
5	3.5737 ± 0.0025	3.4581 ± 0.0025	7.03 ± 0.12
7	4.993 ± 0.007	5.298 ± 0.007	10.29 ± 0.11
10	7.090 ± 0.004	6.991 ± 0.004	14.08 ± 0.19
15	10.67 ± 0.08	10.50 ± 0.08	21.17 ± 0.28
20	14.23 ± 0.12	12.91 ± 0.12	27.14 ± 0.13
25	17.657 ± 0.025	16.323 ± 0.025	33.98 ± 0.21
30	21.475 ± 0.033	21.296 ± 0.033	42.77 ± 0.20
40	28.662 ± 0.032	30.244 ± 0.032	58.9 ± 0.6
50	36.09 ± 0.10	41.15 ± 0.10	77.2 ± 1.2

Table B.4: Loading, compute and total times of the file size series of the NumPy algorithm with in-memory appended loading. Executed on the bwNOTEBOOK.

N	Loading time [s]	Compute time [s]	Total time [s]
1	0.7497 ± 0.0014	0.6906 ± 0.0014	1.440 ± 0.005
3	2.395 ± 0.028	1.959 ± 0.028	4.35 ± 0.05
5	4.062 ± 0.005	3.212 ± 0.005	7.274 ± 0.021
7	5.928 ± 0.007	4.434 ± 0.007	10.36 ± 0.04
10	9.23 ± 0.04	8.31 ± 0.04	17.54 ± 0.20
15	15.91 ± 0.04	12.67 ± 0.04	28.57 ± 0.09
20	23.67 ± 0.05	16.95 ± 0.05	40.63 ± 0.13
25	32.96 ± 0.06	21.26 ± 0.06	54.22 ± 0.09
30	43.27 ± 0.15	25.30 ± 0.15	68.57 ± 0.25
40	68.24 ± 0.18	33.68 ± 0.18	101.92 ± 0.30
50	98.30 ± 0.20	42.84 ± 0.20	141.15 ± 0.28

Table B.5: Loading, compute and total times of the file size series of the PyROOT algorithm. Executed on the bwNOTEBOOK.

N	Loading time [s]	Compute time [s]	Total time [s]
1	0.1171 ± 0.0006	36.3339 ± 0.0006	36.45 ± 0.12
3	0.1134 ± 0.0006	111.7025 ± 0.0006	111.8 ± 0.4
5	0.1138 ± 0.0007	185.6264 ± 0.0007	185.7 ± 0.5
7	0.1135 ± 0.0007	260.0173 ± 0.0007	260.1 ± 0.7
10	0.1139 ± 0.0008	376.6550 ± 0.0008	376.8 ± 1.5
15	0.1140 ± 0.0007	559.6345 ± 0.0007	559.7 ± 1.6

Table B.6: Total times of the Dask chunk size series. 6 workers with 2 GB RAM allocated to each one. 50 M events File. Executed on the bwNOTEBOOK.

Chunk size million events	Total time [s]
0.05	121.3 ± 0.5
0.10	75.62 ± 0.12
0.25	48.03 ± 0.08
0.50	40.25 ± 0.08
0.75	38.92 ± 0.20
1.00	40.24 ± 0.19
1.50	39.46 ± 0.15
2.00	40.88 ± 0.22
2.50	40.55 ± 0.12
3.00	38.78 ± 0.06
3.50	49.1 ± 2.4
3.75	86 ± 34

Table B.7: Total times of the Dask file size series. The file sizes are given in million events. 6 workers with 2 GB RAM allocated to each one. Chunk size of 1 M events. Executed on the bwNOTEBOOK.

N	Total time [s]
1	2.231 ± 0.028
3	2.997 ± 0.007
5	4.484 ± 0.013
7	6.93 ± 0.07
10	8.433 ± 0.017
15	12.83 ± 0.04
20	17.12 ± 0.05
25	21.32 ± 0.19
30	24.86 ± 0.22
40	32.84 ± 0.30
50	40.52 ± 0.26
60	48.3 ± 0.4
70	55.6 ± 0.4
100	78.5 ± 0.7
250	202.6 ± 0.7
300	244.4 ± 0.6
350	286.6 ± 0.8
400	330.6 ± 1.1

Table B.8: Dask worker series. Each worker is allocated 2 GB RAM. The 50 M events file is used. The chunk size is 1 M events. Executed on the bwNOTEBOOK.

Workers	Total time [s]
1	88.29 ± 0.16
2	52.28 ± 0.08
3	43.35 ± 0.08
4	39.97 ± 0.11
5	38.84 ± 0.19
6	39.33 ± 0.35

Table B.9: Loading multiple files with equivalent event numbers to 50M and 100 M total events. The sizes of the files loaded are given in million events. Computed with the Dask algorithm. 6 workers with 2 GB RAM allocated to each one. Executed on bwNOTEBOOK

N	50 M combined size Total time [s]	100M combined size Total time [s]
1	101.2 ± 0.4	323.8 ± 1.2
5	51.08 ± 0.16	124.24 ± 0.13
10	44.82 ± 0.25	99.5 ± 0.5
20	N/A	86.6 ± 0.4
25	40.92 ± 0.27	84.76 ± 0.33
50	39.97 ± 0.15	82.6 ± 0.4
100	N/A	80.80 ± 0.25

Table B.10: Runtime of all algorithms executed on the bwNOTEBOOK, using the Higgs-Gamma-Gamma dataset.

Number of Workers	C++ PROOF computing time [s]	C++ ROOT computing time [s]	Dask computing time [s]	NumPy Classic computing time [s]	NumPy inMem computing time [s]	PyROOT computing time [s]
1	32.85 ± 0.31	32.01 ± 0.09	15.61 ± 0.04	10.61 ± 0.07	12.32 ± 0.07	294.9 ± 1.4
2	21.10 ± 0.21	N/A	9.74 ± 0.10	N/A	N/A	N/A
3	17.41 ± 0.27	N/A	7.990 ± 0.024	N/A	N/A	N/A
4	17.7 ± 0.4	N/A	7.325 ± 0.023	N/A	N/A	N/A
5	18.17 ± 0.26	N/A	7.281 ± 0.028	N/A	N/A	N/A
6	18.63 ± 0.28	N/A	7.117 ± 0.022	N/A	N/A	N/A
7	17.9 ± 0.4	N/A	7.440 ± 0.023	N/A	N/A	N/A
8	19.4 ± 0.4	N/A	7.752 ± 0.019	N/A	N/A	N/A

Table B.11: Dask worker series. Each worker is allocated 2 GB RAM. The datasets with 50 and 100 million events are used. The chunk size is 1 million events. Executed on the ATLAS-BFG

Number of Workers	50 M events Total time [s]	100M events Total time [s]
1	375 ± 10	782 ± 14
2	188.8 ± 0.7	402 ± 4
6	66.9 ± 0.5	141.7 ± 0.7
10	49.4 ± 0.4	94.4 ± 1.1
16	33.76 ± 0.25	63.8 ± 0.6
24	25.80 ± 0.33	47.15 ± 0.20
32	21.62 ± 0.20	38.95 ± 0.26
40	20.12 ± 0.10	33.96 ± 0.19
48	17.71 ± 0.20	32.4 ± 0.4

Table B.12: Dask file size series. The file sizes are given in million events. 48 workers are used, each one is alloceted 2 GB RAM. The chunk sizes of the series are 1 and 3 million events. Executed on the ATLAS-BFG.

N	Chunk size 1 M Total time [s]	Chunksize 3 M Total time [s]
1	8.5 ± 0.5	8.5 ± 0.5
3	10.2 ± 0.4	20.2 ± 1.0
7	10.58 ± 0.35	20.7 ± 1.1
10	11.55 ± 0.22	21.9 ± 1.6
15	12.18 ± 0.30	24.7 ± 1.6
20	13.14 ± 0.34	26.2 ± 1.6
25	12.74 ± 0.32	25.9 ± 1.7
30	13.4 ± 0.5	26.4 ± 1.7
40	14.8 ± 0.5	27.3 ± 1.9
50	18.9 ± 0.8	31.6 ± 3.3
60	21.4 ± 0.6	31.4 ± 2.2
70	23.3 ± 1.0	39.0 ± 3.3
100	32.3 ± 0.7	48.2 ± 2.8
175	49.4 ± 0.9	84.1 ± 2.4
250	68.8 ± 0.6	115 ± 6
300	84.0 ± 1.7	154.1 ± 3.4
350	97.3 ± 0.6	173 ± 4
500	151.1 ± 0.8	248.6 ± 2.8
1000	448.9 ± 3.1	547 ± 10

Table B.13: Dask Chunk size - RAM series. The 500 million events dataset and 48 workers are used. Multiple chunk size to ram ratios are measured: 500k events per GB, 1M events per GB and 2M events per GB. This table displays the series by the amount of memory allocated to the workers. The same data, but displayed by absolute chunk size can be found in table B.14 Executed on the ATLAS-BFG.

Allocated Memory [GB]	500k/GB Total time [s]	1M/GB Total time [s]	2M/GB Total time [s]
2	158.6 ± 3.1	119.9 ± 0.8	N/A
4	117.1 ± 0.5	110.3 ± 0.6	159.9 ± 1.9
6	112.1 ± 0.7	110 ± 4	111.6 ± 3.2
8	110.1 ± 0.4	121.4 ± 0.6	135 ± 4
10	119.3 ± 0.8	132.6 ± 1.2	159 ± 5
12	104.5 ± 0.4	131.3 ± 1.5	211 ± 13
14	113.3 ± 0.5	152 ± 6	293 ± 21
16	125.4 ± 1.1	184 ± 8	322 ± 19

Table B.14: Dask Chunk size - RAM series. The 500 million events dataset and 48 workers are used. Multiple chunk size to ram ratios are measured: 500k events per GB, 1M events per GB and 2M events per GB. This table displays the series by the absolute chunk sizes. The same data, but displayed by the amount of allocated memory can be found in table B.13 Executed on the ATLAS-BFG

Chunk size million events	500k/GB Total time [s]	1M/GB Total time [s]	2M/GB Total time [s]
1	158.6 ± 3.1	N/A	N/A
2	117.1 ± 0.5	119.9 ± 0.8	N/A
3	112.1 ± 0.7	N/A	N/A
4	110.1 ± 0.4	110.3 ± 0.6	N/A
5	119.3 ± 0.8	N/A	N/A
6	104.5 ± 0.4	110 ± 4	N/A
7	113.3 ± 0.5	N/A	N/A
8	125.4 ± 1.1	121.4 ± 0.6	159.9 ± 1.9
10	N/A	132.6 ± 1.2	N/A
12	N/A	131.3 ± 1.5	111.6 ± 3.2
14	N/A	152 ± 6	N/A
16	N/A	184 ± 8	135 ± 4
20	N/A	N/A	159 ± 5
24	N/A	N/A	211 ± 13
28	N/A	N/A	293 ± 21
32	N/A	N/A	322 ± 19

Table B.15: Rerun of the Dask Chunk size - RAM series. The 500 million events dataset and 48 workers are used. Multiple chunk size to ram ratios are measured: 500k events per GB, 1M events per GB and 2M events per GB. This table displays the series by the amount of memory allocated to the workers. The same data, but displayed by absolute chunk size can be found in table B.16 Executed on the ATLAS-BFG.

Allocated Memory [GB]	500k/GB Total time [s]	1M/GB Total time [s]	2M/GB Total time [s]
2	158.0 ± 0.8	126.3 ± 1.1	N/A
4	120.2 ± 0.6	112.9 ± 0.6	177 ± 9
6	117.4 ± 0.8	118.5 ± 1.5	126 ± 4
8	113.0 ± 1.0	124.2 ± 1.0	155 ± 6
10	119.6 ± 0.8	131.4 ± 2.3	225 ± 8
12	109.4 ± 2.5	134.0 ± 2.9	221 ± 10
14	126.5 ± 1.3	153.9 ± 3.3	274 ± 17
16	134 ± 9	179 ± 5	349 ± 21

Table B.16: Rerun of the Dask Chunk size - RAM series. The 500 million events dataset and 48 workers are used. Multiple chunk size to ram ratios are measured: 500k events per GB, 1M events per GB and 2M events per GB. This table displays the series by the absolute chunk sizes. The same data, but displayed by the amount of allocated memory can be found in table B.15 Executed on the ATLAS-BFG

Chunk size million events	500k/GB Total time [s]	1M/GB Total time [s]	2M/GB Total time [s]
1	158.0 ± 0.8	N/A	N/A
2	120.2 ± 0.6	126.3 ± 1.1	N/A
3	117.4 ± 0.8	N/A	N/A
4	113.0 ± 1.0	112.9 ± 0.6	N/A
5	119.6 ± 0.8	N/A	N/A
6	109.4 ± 2.5	118.5 ± 1.5	N/A
7	126.5 ± 1.3	N/A	N/A
8	134 ± 9	124.2 ± 1.0	177 ± 9
10	N/A	131.4 ± 2.3	N/A
12	N/A	134.0 ± 2.9	126 ± 4
14	N/A	153.9 ± 3.3	N/A
16	N/A	179 ± 5	155 ± 6
20	N/A	N/A	225 ± 8
24	N/A	N/A	221 ± 10
28	N/A	N/A	274 ± 17
32	N/A	N/A	349 ± 21

Table B.17: Dask file size series. Dataset sizes given in million events. 20 workers are used, each one is allocated 2 GB RAM. Two subseries with chunk sizes of 1 and 3 million events are used. Executed on NEMO.

N	Chunk size 1 M Total time [s]	Chunk size 3 M Total time [s]
1	5.0 ± 0.4	5.1 ± 0.5
3	6.3 ± 0.7	11.2 ± 1.5
5	6.1 ± 0.5	11.0 ± 0.8
7	7.1 ± 0.5	11.9 ± 1.1
10	8.5 ± 0.4	12.2 ± 0.9
15	10.6 ± 0.5	14.3 ± 1.5
20	11.8 ± 0.7	16.8 ± 2.0
25	14.7 ± 1.0	16.9 ± 0.9
30	15.8 ± 1.2	18.0 ± 1.6
40	19.2 ± 1.3	19.6 ± 1.7
50	26.3 ± 1.9	23.3 ± 2.7
60	27.3 ± 1.8	25.6 ± 2.1
70	33.3 ± 1.9	36 ± 4
100	43.8 ± 2.0	45 ± 4
175	70.6 ± 3.5	113 ± 20
250	104 ± 6	262 ± 26
300	120 ± 5	N/A
350	194 ± 9	N/A
400	232 ± 14	N/A
500	310 ± 14	N/A
750	938 ± 17	N/A

Table B.18: Dask worker series with two different maximum numbers of workers. Each worker is allocated 2 GB RAM. For each maximum worker count, the datasets with 50 and 100 million data points are measured. The chunk size is 1 million events. Executed on NEMO.

Number of Workers	maximum of 20 workers 50 M dataset Total time [s]	 100 M dataset Total time [s]	maximum of 40 workers 50 M dataset Total time [s]	 100 M dataset Total time [s]
1	130 ± 7	N/A	141 ± 18	2030 ± 170
2	68.14 ± 0.33	149 ± 12	68.5 ± 0.4	152 ± 8
6	33.28 ± 0.11	60.77 ± 0.16	31.71 ± 0.11	60.64 ± 0.11
10	26.45 ± 0.19	47.85 ± 0.21	23.59 ± 0.14	41.14 ± 0.07
16	24.98 ± 0.27	44.68 ± 0.23	18.42 ± 0.14	32.58 ± 0.14
20	28.7 ± 0.5	47.8 ± 0.8	17.53 ± 0.15	30.17 ± 0.11
24	N/A	N/A	16.98 ± 0.07	29.40 ± 0.11
32	N/A	N/A	16.64 ± 0.08	28.67 ± 0.11
40	N/A	N/A	19.2 ± 0.8	32.66 ± 0.22

Table B.19: Comparison between the Clusters. Most workers are allocated 2 GB RAM, except for one series on NEMO where 4 GB are allocated. The dataset size is given in million events. A chunk size of 1 million events is used. Both series on ATLAS-BFG and vNEMO are using 40 workers, resulting in a utilization of 20 physical CPU cores. The workers on NEMO are using one physical CPU core each, resulting in 20 and 40 utilized cores for 20 and 40 workers.

Cluster	BFG	vNEMO		NEMO	
# Workers	40	40	20	20	40
Allocated RAM	2 GB	2 GB	2 GB	4 GB	2 GB
N	Total time [s]	Total time [s]	Total time [s]	Total time [s]	Total time [s]
50	21.5 ± 0.4	17.9 ± 1.8	24.7 ± 0.5	27.7 ± 1.2	17.1 ± 0.9
100	36.5 ± 0.8	31.1 ± 2.9	43.50 ± 0.28	45.39 ± 0.22	28.6 ± 1.0
500	170.2 ± 1.0	162 ± 14	320 ± 7	211.3 ± 0.9	138 ± 5
750	298.0 ± 2.8	316 ± 19	938 ± 17	404 ± 12	321 ± 8
1000	540 ± 50	881 ± 32	N/A	N/A	612 ± 7

Table B.20: Real file sizes of the files created for representative dataset. The size of the file used to create the sub-files and the monolithic equivalent file are also included

File name	File Size
data_B.GamGam_2551.root	506 kB
data_B.GamGam_28921.root	5.3 MB
data_B.GamGam_158138.root	29 MB
data_B.GamGam_395476.root	72 MB
data_B.GamGam_922893.root	167 MB
data_B.GamGam_1977726.root	358 MB
data_B.GamGam_3955538.root	716 MB
data_B.GamGam_7911162.root	1.4 GB
data_B.GamGam_13185328.root	2.4 GB
data_B.GamGam_17140953.root	3.1 GB
data_B.GamGam_19778036.root	3.5 GB
data_B.GamGam_23733660.root	4.2 GB
data_B.GamGam_30M.root	5.3 GB
data_B.GamGam_542514147_complete.root	94 GB
data_B.GamGam_542514147_complete_1.root	296 B

C Details of the Python Packages and Versions used

Table C.1: Most of the Python packages needed to run the Notebooks for this thesis can be installed via pip. For the every setup the folllowing packages are needed: uproot, awkward, numpy, matplotlib, dask[complete], dask-awkward, jupyter dask-labextension and jupyter-lab or notebook. Specifically on the notebook, a ROOT installation is necessiary. For the cluster, dask-jobqueue is needed to interface with the schedulers. For completeness, the table below contains a complete list of all Python packages installed on the Notebook and both clusters.

Package	Version	Package	Version
aiofiles	22.1.0	dask-labextension	6.1.0
aiohttp	3.8.4	datashader	0.14.4
aiosignal	1.3.1	datashape	0.5.4
aiosqlite	0.19.0	debugpy	1.5.1
alabaster	0.7.12	decorator	5.1.1
anaconda-client	1.11.2	defusedxml	0.7.1
anaconda-navigator	2.4.0	diff-match-patch	20200713
anaconda-project	0.11.1	dill	0.3.6
ansiwrap	0.8.4	distributed	2023.7.0
anyio	3.5.0	docstring-to-markdown	0.11
appdirs	1.4.4	docutils	0.18.1
argon2-cffi	21.3.0	entrypoints	0.4
argon2-cffi-bindings	21.2.0	et-xmlfile	1.1.0
arrow	1.2.3	execnb	0.1.5
astroid	2.14.2	executing	0.8.3
astropy	5.1	fastcore	1.5.29
asttokens	2.0.5	fastjsonschema	2.16.2
astunparse	1.6.3	filelock	3.9.0
async-timeout	4.0.2	flake8	6.0.0
atlas-mpl-style	0.22.1	Flask	2.2.2
atomicwrites	1.4.0	flit_core	3.6.0
attrs	22.1.0	fonttools	4.25.0
Automat	20.2.0	fqdn	1.5.1
autopep8	1.6.0	frozenlist	1.4.0
awkward	2.3.1	fsspec	2022.11.0
awkward-cpp	21	future	0.18.3
Babel	2.11.0	gensim	4.3.0
backcall	0.2.0	ghapi	1.0.4
backports.functools-lru-cache	1.6.4	glob2	0.7
backports.tempfile	1.0	gmpy2	2.1.2
backports.weakref	1.0.post1	graphviz	0.20.1
bcrypt	3.2.0	greenlet	2.0.1
beautifulsoup4	4.11.1	h5py	3.7.0
binaryornot	0.4.4	HeapDict	1.0.1
black	22.6.0	holoviews	1.15.4
bleach	4.1.0	huggingface-hub	0.10.1

Continued on next page

Package	Version	Package	Version
bokeh	2.4.3	hvplot	0.8.2
boltons	23.0.0	hyperlink	21.0.0
Bottleneck	1.3.5	idna	3.4
brotlipy	0.7.0	imagecodecs-lite	2019.12.3
certifi	2023.5.7	imageio	2.26.0
cffi	1.15.1	imagesize	1.4.1
chardet	4.0.0	imbalanced-learn	0.10.1
charset-normalizer	2.0.4	iminuit	2.22.0
click	8.0.4	importlib-metadata	6.8.0
cloudpickle	2.0.0	importlib-resources	6.0.0
clyent	1.2.2	incremental	21.3.0
colorama	0.4.6	inflection	0.5.1
colorcet	3.0.1	iniconfig	1.1.1
comm	0.1.2	intake	0.6.7
conda	23.5.2	intervaltree	3.1.0
conda-build	3.24.0	ipykernel	6.19.2
conda-content-trust	0.1.3	ipyparallel	8.6.1
conda-pack	0.6.0	ipython	8.10.0
conda-package-handling	2.0.2	ipython-genutils	0.2.0
conda_package_streaming	0.7.0	ipywidgets	7.6.5
conda-repo-cli	1.0.41	isoduration	20.11.0
conda-token	0.4.0	isort	5.9.3
conda-verify	3.4.2	itemadapter	0.3.0
constantly	15.1.0	itemloaders	1.0.4
contourpy	1.0.5	itsdangerous	2.0.1
cookiecutter	1.7.3	jedi	0.18.1
cryptography	39.0.1	jeepney	0.7.1
cssselect	1.1.0	jellyfish	0.9.0
cycler	0.11.0	Jinja2	3.1.2
cytoolz	0.12.0	jinja2-time	0.2.0
daal4py	2023.0.2	jmespath	0.10.0
dask	2023.7.0		

References

[1] T. E. Oliphant. “Python for Scientific Computing”. In: *Computing in Science & Engineering* 9.3 (2007), pp. 10–20. DOI: 10.1109/MCSE.2007.58.

[2] C. R. Harris et al. “Array programming with NumPy”. In: *Nature* 585.7825 (Sept. 2020), pp. 357–362. DOI: 10.1038/s41586-020-2649-2. URL: https://doi.org/10.1038/s41586-020-2649-2.

[3] J. D. Hunter. “Matplotlib: A 2D graphics environment”. In: *Computing in Science & Engineering* 9.3 (2007), pp. 90–95. DOI: 10.1109/MCSE.2007.55.

[4] T. Kluyver et al. “Jupyter Notebooks – a publishing format for reproducible computational workflows”. In: *Positioning and Power in Academic Publishing: Players, Agents and Agendas*. Ed. by F. Loizides and B. Schmidt. IOS Press. 2016, pp. 87–90. DOI: 10.3233/978-1-61499-649-1-87.

[5] M. Rocklin. “Dask: Parallel computation with blocked algorithms and task scheduling”. In: *Proceedings of the 14th python in science conference*. 130-136. Citeseer. 2015.

[6] R. Brun and F. Rademakers. “ROOT: An object oriented data analysis framework”. In: *Nucl. Instrum. Meth. A* 389 (1997). Ed. by M. Werlen and D. Perret-Gallix, pp. 81–86. DOI: 10.1016/S0168-9002(97)00048-X.

[7] M. Galli, E. Tejedor, and S. Wunsch. “A New PyROOT: Modern, Interoperable and More Pythonic”. In: *EPJ Web Conf.* 245 (2020). Ed. by C. Doglioni et al., p. 06004. DOI: 10.1051/epjconf/202024506004.

[8] ATLAS Collaboration. *ATLAS 13 TeV samples collection Gamma-Gamma, for 2020 Open Data release. CERN Open Data Portal.* 2020. DOI: 10.7483/OPENDATA.ATLAS.B5BJ.3SGS.

[9] J. Pivarski et al. *scikit-hep/uproot3: 3.14.4*. Version 3.14.4. Feb. 2021. DOI: 10.5281/zenodo.4537826. URL: https://doi.org/10.5281/zenodo.4537826.

[10] J. Pivarski et al. *scikit-hep/awkward-0.x: 0.15.5*. Version 0.15.5. Feb. 2021. DOI: 10.5281/zenodo.4520562. URL: https://doi.org/10.5281/zenodo.4520562.

[11] *The Homepage of ATLAS-BFG*. URL: https://www.hpc.uni-freiburg.de/atlas-bfg (visited on 09/04/2023).

[12] *The Homepage of bwForCluster NEMO*. URL: https://www.nemo.uni-freiburg.de/ (visited on 09/04/2023).

[13] P. D. Group et al. “Review of Particle Physics”. In: *Progress of Theoretical and Experimental Physics* 2022.8 (Aug. 2022), p. 083C01. ISSN: 2050-3911. DOI: 10.1093/ptep/ptac097. eprint: https://academic.oup.com/ptep/article-pdf/2022/8/083C01/49175539/ptac097.pdf. URL: https://doi.org/10.1093/ptep/ptac097.

[14] S. L. Glashow. “Partial Symmetries of Weak Interactions”. In: *Nucl. Phys.* 22 (1961), pp. 579–588. DOI: 10.1016/0029-5582(61)90469-2.

[15] A. Salam and J. C. Ward. “Weak and electromagnetic interactions”. In: *Nuovo Cim.* 11 (1959), pp. 568–577. DOI: 10.1007/BF02726525.

[16] S. Weinberg. “A Model of Leptons”. In: *Phys. Rev. Lett.* 19 (1967), pp. 1264–1266. DOI: 10.1103/PhysRevLett.19.1264.

[17] P. W. Higgs. "Broken symmetries, massless particles and gauge fields". In: *Phys. Lett.* 12 (1964), pp. 132–133. DOI: 10.1016/0031-9163(64)91136-9.

[18] P. W. Higgs. "Spontaneous Symmetry Breakdown without Massless Bosons". In: *Phys. Rev.* 145 (1966), pp. 1156–1163. DOI: 10.1103/PhysRev.145.1156.

[19] G. S. Guralnik, C. R. Hagen, and T. W. B. Kibble. "Global Conservation Laws and Massless Particles". In: *Phys. Rev. Lett.* 13 (1964). Ed. by J. C. Taylor, pp. 585–587. DOI: 10.1103/PhysRevLett.13.585.

[20] F. Englert and R. Brout. "Broken Symmetry and the Mass of Gauge Vector Mesons". In: *Phys. Rev. Lett.* 13 (1964). Ed. by J. C. Taylor, pp. 321–323. DOI: 10.1103/PhysRevLett.13.321.

[21] P. W. Higgs. "Broken Symmetries and the Masses of Gauge Bosons". In: *Phys. Rev. Lett.* 13 (1964). Ed. by J. C. Taylor, pp. 508–509. DOI: 10.1103/PhysRevLett.13.508.

[22] T. W. B. Kibble. "Symmetry breaking in nonAbelian gauge theories". In: *Phys. Rev.* 155 (1967). Ed. by J. C. Taylor, pp. 1554–1561. DOI: 10.1103/PhysRev.155.1554.

[23] ATLAS Collaboration. "Observation of a new particle in the search for the Standard Model Higgs boson with the ATLAS detector at the LHC". In: *Physics Letters B* 716.1 (2012), pp. 1–29. ISSN: 0370-2693. DOI: https://doi.org/10.1016/j.physletb.2012.08.020. URL: https://www.sciencedirect.com/science/article/pii/S037026931200857X.

[24] CMS Collaboration. "Observation of a new boson at a mass of 125 GeV with the CMS experiment at the LHC". In: *Physics Letters B* 716.1 (2012), pp. 30–61. ISSN: 0370-2693. DOI: https://doi.org/10.1016/j.physletb.2012.08.021. URL: https://www.sciencedirect.com/science/article/pii/S0370269312008581.

[25] D. de Florian et al. "Handbook of LHC Higgs Cross Sections: 4. Deciphering the Nature of the Higgs Sector". In: 2/2017 (Oct. 2016). DOI: 10.23731/CYRM-2017-002. arXiv: 1610.07922 [hep-ph].

[26] ATLAS Collaboration. *ATLAS glossary: Gluon fusion.* URL: https://atlas.cern/glossary/gluon-fusion (visited on 09/30/2023).

[27] ATLAS Collaboration. *ATLAS glossary: Vector boson fusion.* URL: https://atlas.cern/glossary/vector-boson-fusion (visited on 09/30/2023).

[28] "LHC Machine". In: *JINST* 3 (2008). Ed. by L. Evans and P. Bryant, S08001. DOI: 10.1088/1748-0221/3/08/S08001.

[29] K. Aamodt et al. "The ALICE experiment at the CERN LHC". In: *JINST* 3 (2008), S08002. DOI: 10.1088/1748-0221/3/08/S08002.

[30] G. Aad et al. "The ATLAS Experiment at the CERN Large Hadron Collider". In: *JINST* 3 (2008), S08003. DOI: 10.1088/1748-0221/3/08/S08003.

[31] S. Chatrchyan et al. "The CMS Experiment at the CERN LHC". In: *JINST* 3 (2008), S08004. DOI: 10.1088/1748-0221/3/08/S08004.

[32] A. A. Alves Jr. et al. "The LHCb Detector at the LHC". In: *JINST* 3 (2008), S08005. DOI: 10.1088/1748-0221/3/08/S08005.

[33] S. Mehlhase. "ATLAS detector slice (and particle visualisations)". In: (2021). URL: https://cds.cern.ch/record/2770815.

[34] ATLAS Collaboration. *The ATLAS outreach data tools GitHub repository*. http://precog.iiitd.edu.in/people/anupama. 2021.

[35] ATLAS Collaboration. *ATLAS 13 TeV samples collection Gamma-Gamma, for 2020 Open Data release*. http://opendata.cern.ch/record/15006. 2020. DOI: 10.7483/OPENDATA.ATLAS.B5BJ.3SGS.

[36] *Review of the 13 TeV ATLAS Open Data release*. Tech. rep. Geneva: CERN, 2020. URL: https://cds.cern.ch/record/2707171.

[37] F. Pérez and B. E. Granger. "IPython: a System for Interactive Scientific Computing". In: *Computing in Science and Engineering* 9.3 (May 2007), pp. 21–29. ISSN: 1521-9615. DOI: 10.1109/MCSE.2007.53. URL: https://ipython.org.

[38] ATLAS Collaboration. *The ATLAS OpenData 13 TeV Hyy example jupyter notebook*. https://github.com/atlas-outreach-data-tools/notebooks-collection-opendata/blob/master/13-TeV-examples/python/ATLAS_OpenData__13-TeV_analysis_example-python_Hyy_channel.ipynb. 2021.

[39] ATLAS Collaboration. *Event display of the Atlas detector, a diphoton candidate*. URL: https://atlas.web.cern.ch/Atlas/GROUPS/PHYSICS/CONFNOTES/ATLAS-CONF-2011-161/fig_13.png (visited on 09/11/2023).

[40] *Papermill Homepage*. URL: https://papermill.readthedocs.io/en/latest/ (visited on 09/10/2023).

[41] *Dask homepage*. URL: https://docs.dask.org/en/stable/ (visited on 09/11/2023).

[42] *Dask awkward GitHub repository*. URL: https://github.com/dask-contrib/dask-awkward (visited on 09/03/2023).

[43] *Dask blog: Choosing Chunk sizes*. URL: https://blog.dask.org/2021/11/02/choosing-dask-chunk-sizes#rough-rules-of-thumb (visited on 09/11/2023).

[44] ATLAS Collaboration. *The ATLAS OpenData example C++ analysis*. https://github.com/atlas-outreach-data-tools/atlas-outreach-cpp-framework-13tev/tree/master. 2021.

[45] D. Feichtinger et al. "PROOF - The Parallel ROOT Facility". In: *2006 15th IEEE International Conference on High Performance Distributed Computing*. 2006, pp. 379–380. DOI: 10.1109/HPDC.2006.1652193.

[46] A. B. Yoo, M. A. Jette, and M. Grondona. "SLURM: Simple Linux Utility for Resource Management". In: *Job Scheduling Strategies for Parallel Processing*. Ed. by D. Feitelson, L. Rudolph, and U. Schwiegelshohn. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 44–60. ISBN: 978-3-540-39727-4.

[47] D. Sammel et al. "Lightweight Integration of a Data Cache for Opportunistic Usage of HPC Resources in HEP Workflows". In: *Computing and Software for Big Science* 7.1 (July 2023), p. 7. ISSN: 2510-2044. DOI: 10.1007/s41781-023-00100-1.

[48] *Dask-Jobqueue Homepage*. URL: https://jobqueue.dask.org/en/latest/ (visited on 09/03/2023).

[49] *The landing page for MOAB*. URL: https://adaptivecomputing.com/moab-hpc-suite/ (visited on 09/04/2023).

[50] *The NEMO cluster schema*. URL: https://wiki.bwhpc.de/e/File:NEMO-Cluster-Schema.png (visited on 09/04/2023).

[51] B. Rottler. “Search for Higgs-Boson Pair-Production in the $bb\ell\ell + E_T^{miss}$ final state with the ATLAS detector at $\sqrt{s} = 13\,\text{TeV}$”. PhD thesis. Fakultät für Mathematik und Physik, Albert-Ludwigs-Universität Freiburg, 2023. DOI: 10.6094/UNIFR/238604.

[52] M. Barisits et al. “Rucio: Scientific Data Management”. In: *Computing and Software for Big Science* 3.1 (Aug. 2019), p. 11. ISSN: 2510-2044. DOI: 10.1007/s41781-019-0026-3. URL: https://doi.org/10.1007/s41781-019-0026-3.

Acknowledgement

Acknowledgement: The author acknowledges support by the state of Baden-Württemberg through bwHPC and the German Research Foundation (DFG) through grant no INST 39/963-1 FUGG (bwForCluster NEMO).