# INTERMEDIATE REPORT ON PILOT MIGRATION AND OPTIMIZATION

| | |
|---|---|
| DELIVERABLE NUMBER: | D2.2 |
| DUE DATE: | 30.06.2024 |
| DATE OF SUBMISSION: | 17.07.2024 |
| NATURE: | R |
| DISSEMINATION LEVEL: | PU |
| WORK PACKAGE: | WP2 |
| LEAD BENEFICIARY | KTM |

Grant Agreement No 101092850

Funded by the European Union

## Document Control Sheet

| | |
|---|---|
| Deliverable Title: | Intermediate Report on Pilot Migration and Optimization |
| Authors: | Michael Wurzer (KTM) |
| Contributors: | Daniel Krefl (SED), Claude Chaudet (UNIGE) |
| Reviewers: | Massimiliano Donati (UNIPI), Sergio Saponara (UNIPI), Claude Chaudet (UNIGE) |
| Approved By: | Christos Kotselidis (UNIMAN), Dionisios Pnevmatikatos (ICCS) |

## Document History

| Version | Date | Status | Description/Comments |
|---|---|---|---|
| 0.1 | 18.05.2024 | Draft | ToC |
| 0.2 | 26.06.2024 | Draft | Draft released for internal review |
| 0.3 | 09.07.2024 | Draft | Revised draft based on reviewers' comments |
| 1.0 | 17.07.2024 | Final | Final version submitted to EC |

## DISCLAIMER

AERO has received funding from European Union's Horizon Europe research and innovation programme under Grant Agreement No 101092850. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the granting authority. Neither the European Union nor the granting authority can be held responsible for them.

# TABLE OF CONTENTS

# Executive Summary

This document has the following three objectives:

1. To describe the development and optimisation activities that have taken place until M18 towards the successful migration of the AERO pilots to the AERO envisioned platforms.
2. To present the plans for the development of the AERO pilots during the second part of the project.
3. To report initial performance evaluation results. Unfortunately, as the Rhea platform is delayed, runs have been executed on alternative ARM-based platforms, most notably an NVIDIA GraceHopper platform, as well as x86_64 platforms that serve as baseline.

# List of Abbreviations & Acronyms

| Abbreviation/Acronym | Meaning |
|---|---|
| AI | Artificial Intelligence |
| API | Application Programming Interface |
| AVX | Advanced Vector Extensions |
| CU7 | Gaia Variability Study Coordination Unit |
| CUDA | Computed Unified Device Architecture |
| CUO | Connectivity Unit Offroad |
| DB | Database |
| DBMS | Database Management System |
| DPCG | Gaia Data Processing Center in Geneva |
| DR3, DR4 | Gaia Data Release 3 / Data Release 4 |
| ESA | European Space Agency |
| FDW | Foreign Data Wrapper |
| GPU | Graphical Processing Unit |
| GUC | Grand Unified Configuration |
| HPC | High Performance Computing |
| JVM | Java Virtual Machine |
| KPI | Key Performance Indicator |
| LSTM | Long Short-Term Memory (Machine Learning) |
| MPP | Massive Parallel Processing |
| NVMe | Non-volatile Memory Express |
| PG, TBase | PostgreSQL DBMS or PG MPP fork |
| SPIR/SPIR-V | Standard Portable Intermediate Representation |
| SVE | Scalable Vector Extensions |
| TPC-H | Transaction Processing Performance Council Standard Benchmark |
| UDF | User Defined Functions |

# 1 Introduction

The successful migration of the AERO pilots to the envisioned AERO platform is critical for the success of the AERO project. As these pilots were developed prior to the AERO project to be executed on typical x86_64 platforms, their migration is a complex process that entails significant development and optimisation efforts. This deliverable reports on these efforts that have taken place until M18.

The remainder of this deliverable is organized as follows:

- **Section 2** provides an overview of the current status of the KTM pilot.
- **Section 3** provides an overview of the current status of the UNIGE pilot.
- **Section 4** provides an overview of the current status of the SED pilot.

More specifically, for each pilot the discussion is organised into five subsections that present: (i) the pilot description; (ii) its current status in terms of development within AERO; (iii) an overview of the upbringing and optimisation efforts; (iv) the plans for further development and optimisation in the next period of the project, and (v) current performance evaluation results.

# 2 Automotive "Digital Twins" with IoT-Cloud Interoperability (KTM)

## 2.1 Pilot description

This pilot currently comprises three use-cases: (i) The Vehicle Information Service; (ii) ASP.NET Boilerplate, and (iii) in-house Jump Detection model. The former two have already been described in D2.1. Therefore, we expand below on the latter.

The jump detection use case relies on an AI model designed to detect the airtime of a motorcycle and is built using TensorFlow[1], leveraging the capabilities of a Long Short-Term Memory (LSTM) network. This neural network architecture is particularly adept at handling sequential data, making it ideal for tasks that involve time series and temporal patterns. The model is engineered to accurately distinguish between four specific states associated with motorcycle airtime:

1. **Ramp**: This state indicates that the motorcycle is on a ramp and is about to jump. It captures the moment when the motorcycle is accelerating up the ramp, preparing for liftoff.
2. **Air**: In this state, the motorcycle is completely airborne. The model identifies the period when the motorcycle is not in contact with the ground, capturing the duration of the jump.
3. **Landing**: The landing state is identified when the motorcycle, after being airborne, makes contact with the ground again. This state is crucial for understanding the dynamics of the landing process.
4. **Miscellaneous (Misc)**: This state encompasses all other scenarios, such as when the motorcycle is driving normally on the ground, turning, or performing other manoeuvres that do not involve ramping, being airborne, or landing.

The detection is done solely via inertia sensor data by utilizing accelerometer and gyrometer data. Additionally, the magnitude of the acceleration is calculated, which greatly improves the detection quality. The data is recorded with a Connectivity Unit Offroad (CUO) attached to a KTM off-road bike. The CUO is a device developed by KTM to record, among others, inertial and GPS data while riding on a track. The data can be sent to a smartphone and analysed after the ride. For the jump detection rides were performed on tracks with multiple ramps and therefore multiple jumps. The inertial data was recorded by the CUO and is now used to train and test the model.

The dataset used to train this model is around 5.2 MB in size and consists of hand-labelled examples, ensuring precise annotations. This labelled data is crucial for the model to learn the subtle distinctions between the different states. To facilitate effective training and evaluation, the dataset is split into two parts: 70% of the data is used for training the model, allowing it to learn the patterns and characteristics of each state, while the remaining 30% is reserved for testing and validation.

## 2.2 Current status

For use cases (i) and (ii), there are no updates to report since the revised Deliverable D2.1 (submitted in M16). Use case (iii) was recently added to the KTM pilot. Its code and environment have been

[1] https://www.tensorflow.org/

adjusted to run inside Docker containers in order to be easily deployable for testing and comparing performance.

## 2.3 Upbringing and optimisation efforts

To ensure the highest level of accuracy, all three use cases of the KTM pilot are continuously evaluated for performance and stability/compatibility across all alternatives systems to SIPEARL Rhea. This ongoing evaluation process involves testing and benchmarking to identify any potential bottlenecks or inefficiencies. Tests and benchmarks initially done on Microsoft Azure are now step by step migrated to systems closer to Rhea. The ARM servers used by Microsoft Azure in the KTM cloud are Neoverse N1 cores. These do not support the Scalable Vector Extensions (SVE), but only the Neon instruction set. Therefore, benchmarks are replicated on NVIDIA Grace Hopper servers, which use Neoverse V2. These are more similar to the cores used in Rhea (Neoverse V1) as, for instance, they both support SVE.

## 2.4 Plans for further development

As the project moves forward, the immediate focus is on finalizing an in-depth performance analysis. This phase involves running tests on an ARM platform with GPU acceleration, a configuration that has posed challenges until now.

More specifically, the standard TensorFlow container on Docker Hub is not built for ARM. Therefore, we used the TensorFlow container from the NVIDIA Container Registry[2]. However, when trying to fit the model with this container, no matter on which system, the result is a Graph execution error. A similar issue is described in this GitHub Issue[3], but the cause in our case has not been found yet and is currently under investigation.

Finally, we are gearing up to initiate functional tests and performance benchmarks on the SIPEARL Rhea platform, as soon as it becomes available.

## 2.5 Performance evaluation

For the initial evaluation of use case (iii), we used an x86_64 server (Intel Xeon 8272L with 2 cores/4 threads) and an ARMv64 server (ARM Neoverse N1 with 4 cores/4threads) using a Python container installing TensorFlow for both systems, in order to perform an as fair as possible comparison. This solution was selected, as a TensorFlow container is only available for x86_64.

We performed 10 runs of the fitting process on these two systems and the time needed (in seconds) for each run is presented in Table 1, together with the average execution time across all runs for each system. The x86_64 system requires around 300s to perform the fitting, while the same process takes 586.5s on average on the ARMv64 system. This performance difference is due to the fact that TensorFlow leverages the Advanced Vector Extensions (AVX) on the Intel system. On the other hand, the ARMv64 does not use any vector extensions. Its Neoverse N1 cores do not have SVE but Neon

[2] https://catalog.ngc.nvidia.com/containers
[3] https://github.com/tensorflow/tensorflow/issues/35950

vector extensions, which are not supported by TensorFlow out of the box. Hence, the execution is performed only on the CPU leading to an increased execution time.

**Table 1** Fitting of the jump detection algorithm with ~5.2MB of data

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | Avg. |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Intel Xeon 8272L (2C4T)** | 300.9 | 579.0 | 316.0 | 330.5 | 281.1 | 298.1 | 265.1 | 280.6 | 347.0 | 280.5 | **299.8** |
| **ARM Neoverse N1 (4C4T)** | 579.0 | 609.6 | 588.0 | 566.5 | 561.4 | 592.1 | 614.2 | 579.0 | 582.7 | 592.6 | **586.5** |
| **GraceHopper (4C4T)** | 145.9 | 146.4 | 146.9 | 146.3 | 146.2 | 147 | 146.9 | 147.2 | 146.3 | 146.4 | **146.5** |
| **i9-13900K with NVIDIA RTX4090 GPU** | 75.5 | 75.6 | 76.2 | 76.1 | 76.2 | 76.3 | 76.0 | 76.1 | 76.0 | 75.9 | **76.0** |

To gain a better understanding of the performance that we will anticipate on the Rhea platform, we repeated the tests on an NVIDIA GraceHopper system, which uses ARM Neoverse V2 cores. The time needed for each run together with their average is also reported in Table 1. It is evident that GraceHopper is approximately 2x faster than the x86_64 system.

Finally, we report in Table 1 also the execution time for an Intel Core i9-13900K system with an NVIDIA RTX 4090 GPU to assess what a system equipped with a high-end GPU is capable of. It is evident that the GPU-accelerated fitting is 2x faster than the GraceHopper platform, an ARM-based system with SVE. Hence, the combination of an SVE-capable ARM processor with a high-end GPU, as envisioned in the Rhea system, should be able to yield competitive results for this use case.

# 3 High-Performance Algorithms for Space Exploration (UNIGE)

## 3.1 Pilot description

Gaia Data Processing Centre in Geneva (DPCG) and Coordination Unit 7 (CU7) based at Observatory of Geneva - part of University of Geneva (UNIGE) - are responsible for variability studies of the nearly 2.7 billion sources that the ESA Gaia mission observes. The main task is to classify and characterize variable stellar objects/sources based on their time series from five products and three domains: photometry, spectroscopy, and radial velocities. Gaia is the first Petabyte scale astronomical ESA mission and operates on one of the biggest astronomical datasets up to date. The amount and complexity of the data as well as the number of angles from which the data is analysed, poses a plethora of performance challenges in the Big Data and HPC domains.

## 3.2 Current status

The DPCG pipeline is the main data processing software that handles the data coming from the sensors of the Gaia spacecraft to characterise the variability of the different stars in our galaxy, and to classify them among the different standard categories. Catalogues of stars are released regularly, as more and more data are acquired and processed. The mission has just celebrated its 10th birthday and the 4th version of the catalogue is expected to be released in 2026.

This pipeline, when run on the full dataset acquired from billions of stars, represents several days of computation. For the third catalogue data release (DR3), the cumulated computation time approximates 20 days in CPU time, multiplied by the configuration-tuning-preparatory phase. For the fourth release (DR4), the volume of data to be processed will be significantly higher because of the longer observation period and because of additional data sources that scientists may want to use if the computation time allows.

Since the algorithm's execution time is dependent on the number of epochs to process, the algorithm speed is of paramount importance and most of the data processing steps are under constant optimization. A significant speedup is necessary to reach the same level of precision as the previous data release, and any additional gain would allow the scientists to incorporate secondary data sources, which means more and longer time series.

The processing pipeline is composed of several modules implementing different data manipulation algorithms running in sequence or in parallel. Among those algorithms the search for the periodic signal of the different stars is a computationally costly, yet fundamental, building block. The work within the AERO project for this pilot therefore consists of converting one or several of such Period Search algorithms to be run on GPU using TornadoVM and ultimately running on the AERO platform as a demonstrator.

The most common Period Search algorithm belongs to the Least Squares spectral analysis family that attempts to fit data to a sum of sinusoidal signals, similarly to the Fourier methods. Unlike Fourier

analysis, those techniques are designed to work on partial and unevenly sampled data, like the time series captured by the Gaia spacecraft. For each data source and for each acquisition instrument, the algorithm processes one time series of the different observations of the corresponding spatial object at times $t_i$ in the frequency range the instrument is sensitive to, producing a sequence of times and values: ($t_i$, $x_i$). The algorithm tries to express the process underlying the observations as a sum of sinusoidal functions:

$$x(t) = c + c_1 * \cos(2\pi f t) + c_2 * \sin(2\pi f t)$$

The coefficients of that formula (c, c1, c2) are estimated by minimizing the weighted sum of the squared differences between the observations and the model:

$$min \sum_{i=0}^{N} \left(x_i - x(t_i)\right)^2$$

The minimization requires calculating the following sums of sines and/or cosines and combining those sums for all candidate frequency (f):

$$\sum_{i=0}^{N} \sin(2\pi f t_i) \qquad \sum_{i=0}^{N} \cos(2\pi f t_i)$$

$$\sum_{i=0}^{N} \sin^2(2\pi f t_i) \qquad \sum_{i=0}^{N} \cos^2(2\pi f t_i) \qquad \sum_{i=0}^{N} \sin(2\pi f t_i) * \cos(2\pi f t_i)$$

$$\sum_{i=0}^{N} \sin(2\pi f t_i) * x_i \qquad \sum_{i=0}^{N} \cos(2\pi f t_i) * x_i$$

The costliest operations in that minimization process, time-wise, are the evaluation of the trigonometric functions. The complexity of the algorithm therefore is proportional to the product of the number of sources by the number of possible frequencies and the time series length (N). Sources are independent, as well as frequencies, which opens the door to a certain level of parallelization. However, the computed sums for a given source and frequency are interleaved, which limits the speedup that can be achieved as defined by Amdahl's law.

To understand the volume of computation, considering the instruments' acquisition process, the possible periods that may compose the signal span from 48 minutes (30 $day^{-1}$ frequency) to 1000 days (0.001 $day^{-1}$ frequency). For the upcoming data release, this interval needs to be sampled at the Nyquist sampling rate, to prevent both over and under-sampling, which means that our algorithm needs to scan the whole frequency range with a step of $5.10^{-5}$ $day^{-1}$, leading to the exploration of 600000 frequencies for each stellar object.

For variability studies, between one and four time series may be considered by source. For the previous data release, one time series was considered by source with an average length of approximately 37 samples. For the upcoming data release, the time series length shall more than double and, depending on the algorithm performance, additional time series (from different Gaia instruments) may be considered with length up to approximately 1000 samples.

For each of those time series, and for each of the 600000 frequencies, we need to compute and combine a number of sine and cosine proportional to the time series length. Table 2 presents rough estimations of the number of trigonometric operations required between the two data releases. Even if these do not translate directly in execution time, they show that the computational effort could, depending on the strategic choices, be multiplied by up to 1000. This means that the choices to study or not certain perspectives will depend on the performance gain that can be achieved between the two catalogue productions.

**Table 2** Algorithm dimensions for the previous (DR3) and upcoming (DR4) data releases

| Parameter | Data Release 3 (2022) | Data Release 4 (2026) |
|---|---|---|
| Sampling Window | 1729 days | 2704 days |
| Frequencies Range | 57 min - 1428 days | 48 min - 1000 days |
| Frequency Step | $10^{-4}$ | $5.10^{-5}$ |
| Number of frequencies to examine | ~ 300 000 | ~ 600 000 |
| Number of Sources for Period Search | 400 million studied<br>10.5 million variables | Goal: 2.5 billion |
| Number of time series | 400 million | Up to 10 billion (4 bands) |
| Time Series typical length | 37<br><br><br><br>(100k sources sample) | 81.1 (G Band)<br>41.8 (BP band)<br>40.8 (RP band)<br>714.4 (CCD photometry of G band)<br><br>(410k sources sample) |
| Number of sines/cosines to compute | ~ $8.8 * 10^{15}$ | ~ $2.634 * 10^{18}$ |

Over the first 18 months of the AERO project, the teams of UNIGE, SED and UNIMAN have worked closely together to port the existing period search method code to the TornadoVM framework. This required a significant amount of refactoring and adaptation of the existing code, leading to the improvement of both the Gaia processing pipeline and of the TornadoVM framework. TornadoVM provides a framework that facilitates the use of GPUs in Java, acting as a wrapper around technologies such as OpenCL, CUDA and SPIR-V. This framework is particularly suited to the Gaia pipeline, since the pipeline is written in Java and runs over a heterogeneous set of machines that include Intel and NVIDIA GPUs.

Finally, it is worth mentioning that in parallel, we also produced a version of the algorithm to be run directly within the database, without having to pay the price of extracting and transferring the data. This is algorithm is discussed in Section 4.

## 3.3 Upbringing and optimisation efforts

During the first year of the project, TornadoVM shifted to use the Java Panama API for better support and performance when addressing native code, for example written in C or C++. Panama API not only brings in more safety when it comes to memory access, but also better performance and the support for more efficient memory models, such as off heap arrays that improve the communication between the CPU and the GPU memories. However, the Panama API requires Java Development Toolkit version 21, which required the adaptation of the whole Gaia Pipeline that was designed for Java

version 17. Java 21 introduced some incompatible changes that required development and adaptation of the existing code base, including external libraries and frameworks.

This change had a high cost in terms of development but was beneficial in the end, since it first allowed Gaia to benefit from the evolution of TornadoVM and to act as a large-scale and non-standard test scenario for UNIMAN that is actively developing TornadoVM. Memory management has been the subject of several discussions and the collaboration between the two teams has led to significant performance and stability improvements on both sides.

Concerning the algorithmic adaptations, the Least Squares Period Search method can be decomposed in six steps, as shown below for one time series, i.e. one stellar object as seen by one acquisition instrument:

1. The time series are prepared (normalisation, filtering) upstream to produce clean series. The frequency range is examined to identify each frequency that is to be scanned. Particular sub-ranges (called bins) can also be specified as regions of the frequency space that need particular attention for specific studies. This initialisation phase defines the input of the algorithm as triplets acquisition time, observed values and error estimate and as a set of frequencies to examine. All values are double precision decimal numbers.
2. For each frequency, a synthetic value, the amplitude, is calculated from different combinations of the sines and cosines of the frequency multiplied by each observation time. This step, that is the most computationally intensive, produces as many couples of frequencies and amplitudes as the number of sampled frequencies. This set is called the periodogram (or frequencygram). A graphical example of such periodogram is depicted in Figure 1.
3. On the full periodogram, the N frequencies with the highest amplitudes are searched. The number of frequencies (N) is a number specified by the user. Those frequencies are the ones that contribute the most to the final aggregated signal. This step, which is a simple selection algorithm, is repeated on each of the bins if multiple bins have been specified by the user, leading to the identification of up to N * B frequencies across the whole frequencygram, B being the number of bins. In Figure 1 the 6 highest frequencies are circled in blue.
4. Once these main frequencies have been identified, a local search refinement is performed around those frequencies at a finer frequency resolution. Typically, if the initial frequency step is $5.10^{-5}$, the final resolution is down to $10^{-8}$. This means repeating the period search main algorithm around the identified peaks; therefore, this step has a similar complexity to step 1.
5. When the frequencies have been refined, the algorithm computes global performance metrics for the identified main frequencies, namely the false alarm probability and the signal detection efficiency. Both metrics require reduction operations (average, variance, etc.) over the whole frequencygram.
6. Finally, the output of the algorithm is composed of the list of peaks with their characteristics (frequency, amplitude, false alarm probability, signal detection efficiency), alongside with the full frequencygram if the user requires it. In the general pipeline, such results are stored in a database, which is outside of the scope of that task

Considering the computational cost and the data volume, porting that algorithm to GPU required to balance communication and computation. The initial approach consisted in keeping the whole

computation on the GPU. The algorithm sends the time series to the GPU and receives at least the peaks identification and the associated metrics. This approach rationale was to be light in terms of communication, since the larger part of the data, i.e. the frequencygram, remains on the GPU.

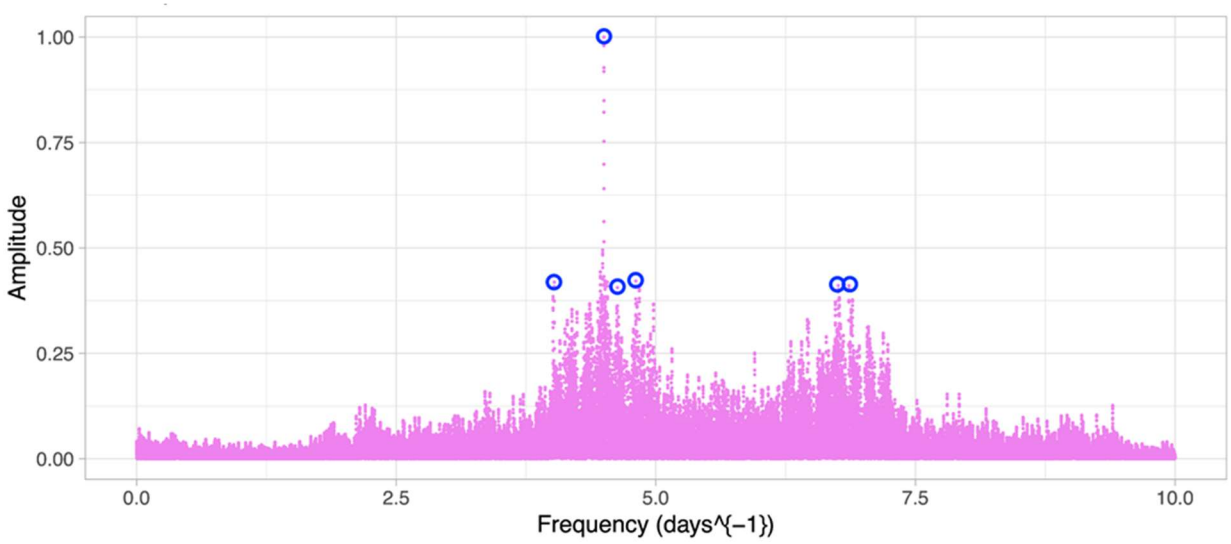



**Figure 1** Example of periodogram with 6 highest peaks outlined (blue)

However, after initial benchmarks, the algorithm's successive steps exhibit different levels of parallelism. The initial frequencygram can involve a number of GPU threads up to the number of frequencies (300 000 to 600 000), while reduction operations, such as average, standard deviation or selection of the maxima, require multiple steps in which less and less threads are active. In our scenario, considering the relative times for all the steps, the approach that offloaded everything on the GPU turned out to induce a very low occupancy of the GPU, and therefore a waste of resources. The code analysis showed that during the reduction operation, few threads were active, blocking nevertheless the whole array.

We therefore adapted the algorithm so that only the initial frequencygram is computed on the GPU, and the subsequent steps are computed by the CPU, freeing the GPU for the next frequencygram to be computed. The cost of communication is naturally higher than in the first scenario, but it is compensated by a more efficient occupancy of the GPU.

Most GPUs are designed for processing single precision numbers, which means that the number of computation units dedicated to double precision numbers is low. We therefore evaluated the numerical accuracy that could be achieved by computing the initial frequencygram in single precision. This would not only provide a better usage of the GPU, but also halve the communication time for retrieving the frequencygram on the CPU. Numerical accuracy can be recuperated at the refinement step, the only risk being in the selection of the peaks. Rounding at a lower numerical precision may change the order of the amplitudes, and some of those order inversions may lead to misidentification of the highest amplitudes. We evaluated this effect and found out that if it occurs, it is not prevalent and can be mitigated by selecting, initially, more frequencies than necessary for refinement.

To improve the communication vs. computation overlap, we also made sure our algorithms were ready for multithreaded execution. Some GPUs can communicate in both ways and compute in parallel. While one thread is blocked waiting for the GPU to finish the frequencygram computation, another thread may be sending the next time series to the GPU and a third one, depending on the

time ratios and buffers, may be retrieving the previous frequencygram. To be safe, some operations need to be carefully scheduled; for instance, the computation thread shall not write the frequencygram in a memory space that is being read by the GPU to CPU communication thread.

To improve the occupancy and the ratio between computation and communication, we also confirmed with experiments that it was more efficient to process time series in batches, i.e. try to send to the GPU as many data sources as possible in one communication step and retrieve all frequencygrams in one step too. Since memory allocation must happen on the GPU, this approach is naturally limited by the GPU available memory and by the different drivers and APIs that impose limitations on the maximum amount of memory that can be allocated in a single step. This is particularly important on our setup since we plan on running the computation on a heterogeneous platform, using possibly all the TornadoVM backends (OpenCL, PTX and SPIR-V) and we need the code to be as adaptable as possible once the Rhea platform becomes available.

Those approaches lead to a better utilization of the GPU in the general case when 1 to 10 frequencies need to be identified. However, the number of frequencies to search for can increase drastically when binning occurs. Binning consists in specifying additional intervals of interest within the general interval, where the algorithm searches for the same number of peaks. It can be understood as a more detailed focus, a zoom, on some specific areas where certain types of stellar objects typically show periodicity.

When such binning happens, the number of frequencies selected and then refined greatly increases and changes the balance between GPU computation, communication and CPU computation, leading again to a low GPU occupancy. Hence, we have started to adapt again the algorithm to offload to the GPU also the refinement step, which has become the bottleneck. This induces more back and forth communication, but that delay should be compensated by the gain in computation time. Besides, on the contrary to the first approach, this strategy splits the computation in two separate steps, which means that the system scheduler may be able to produce a better overlap.

## 3.4 Plans for further development

As of M18, significant progress has been made to produce a usable and reliable version of the Least Squares Period Search algorithms. Several iterations were necessary to correctly split the tasks between the CPU and GPU. The code is still far from having reached its optimal performance, but most of the necessary changes in the data processing pipeline and in the code architecture have been realised and tested. TornadoVM has also been updated to better suit our scenario.

Still, as discussed in Section 3.3, some blocks of the algorithm could also be computed on the GPU and we are currently working on that adaptation. Once we are satisfied and confident with the algorithmic aspects, we will start the optimization and adaptation. We indeed have several machines and clusters that will need to participate in the main calculation by the end of 2024. These systems span different generations, have different amounts of memory, different architectures, use different APIs and using each of them at its best potential will need proper profiling and optimization.

Finally, the Least Squares algorithm is only one of a set of algorithms that target similar goals. Once the base is stable enough, we plan to port other algorithms to work on GPU. In particular, algorithms

working on spectral data (in which individual values are replaced by histograms) could be among the interesting tracks for optimization.

## 3.5 Performance evaluation

Evaluation has been an integral part of the porting process. The first goal was to assess that the new algorithm produced results that were scientifically sound. The correctness has been evaluated based on a comparison with the previous sequential implementation on CPU. For a set of representative time series, we compared the frequencies that were identified and the corresponding amplitude.

Figure 2 represents the number of peaks that had been identified by the original algorithm and that are different from the GPU algorithm when asked to select the 5 highest amplitudes. Over the 100000 sources that compose this dataset, the two algorithms fully agree in most cases. For about 3% of the sources, one single peak is wrongly identified, generally the fifth one, i.e. the last one in the window. In a small subset, most or all of the frequencies are mis-identified. After further investigation, it turned out that these situations happen for very small time series (less than 10 samples), for which a very large number of amplitudes are close to 1 and are not easy to distinguish.

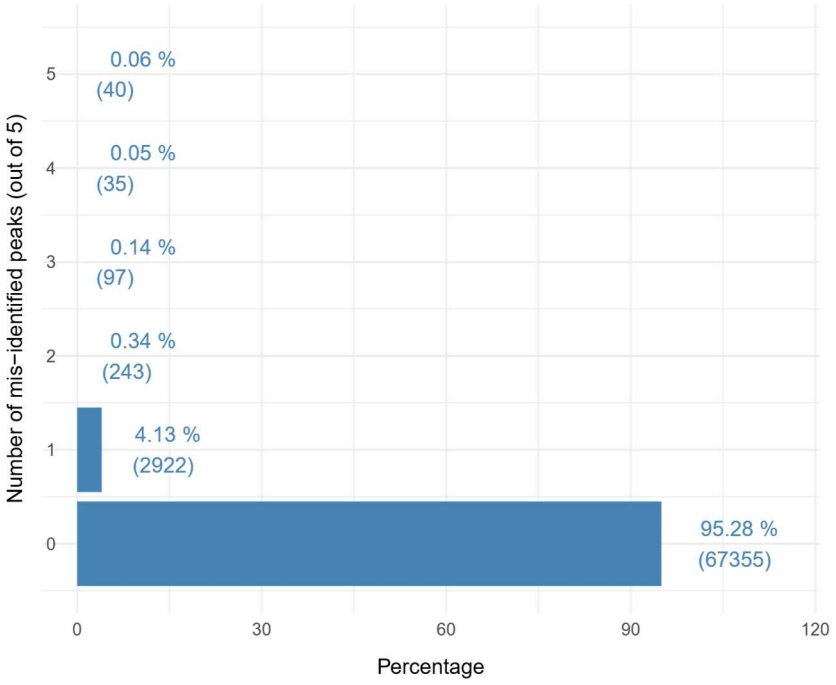



**Figure 2** Number of peak frequencies that differ between CPU and GPU implementations

The reader should notice, at that stage, that such small time series are very unlikely to be present in the upcoming datasets since the observation period has increased significantly. Besides, in those situations, all algorithms are in a borderline situation when it comes to identifying periodicity, which simply means that those sources should be analysed with different tools.

If numerical differences are expected, they should be as small as possible to bear the same meaning and to preserve the ordering. Figure 3 and Figure 4 depict the differences over a small/medium size run (100k sources) over the identified frequencies and over the corresponding amplitudes. The relative difference over the frequencies (Figure 3) is on average approximately $10^{-6}$, while the difference on the amplitudes (Figure 4) is close to $1.5*10^{-3}$. If this value is larger than the single

precision rounding error, it is precise enough to order and rank the frequencies, as the results on the peaks identification and ordering demonstrate.

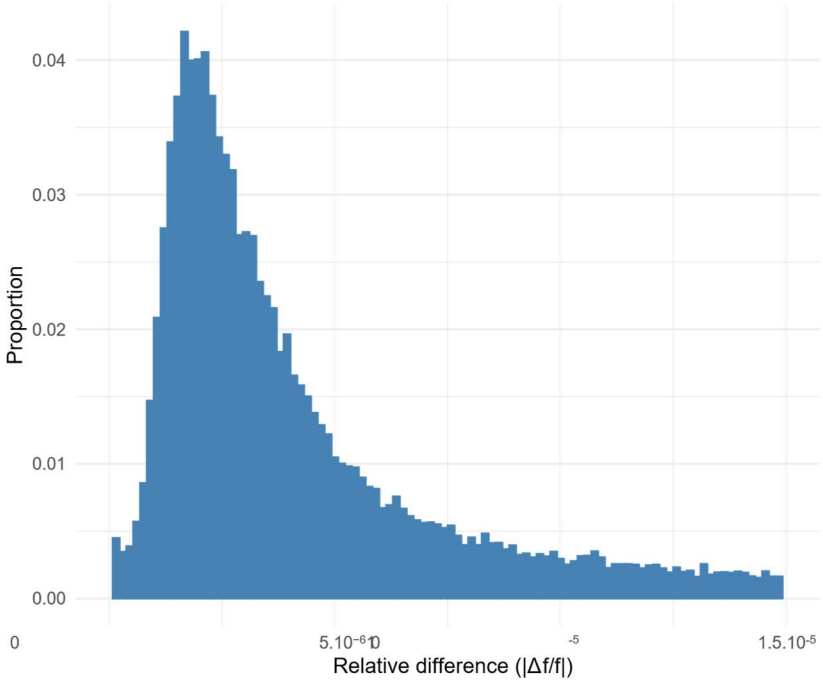



**Figure 3** Relative difference between the frequencies identified by the CPU and GPU implementations

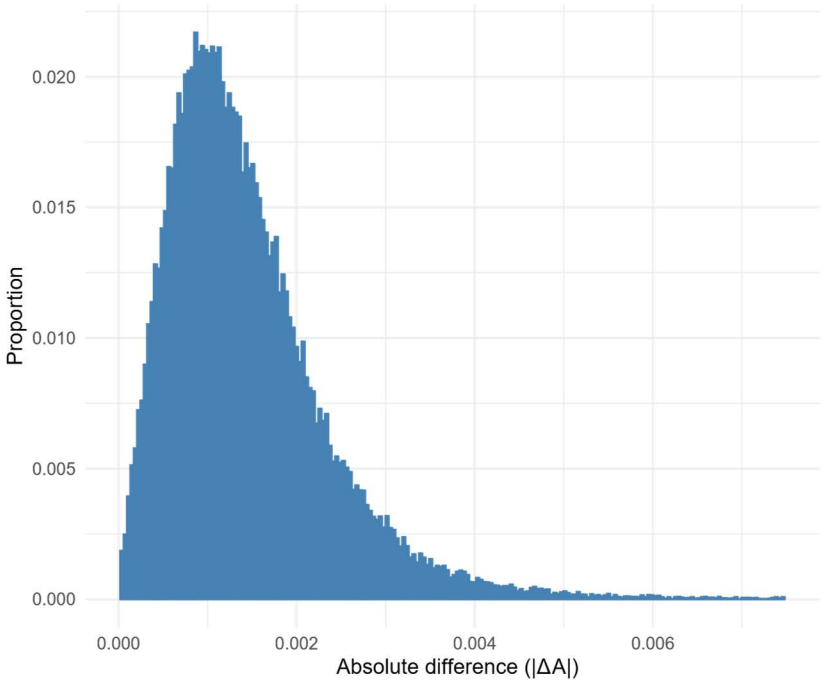



**Figure 4** Absolute difference between the amplitudes of the matching frequencies between the CPU and GPU implementations

The purpose of the evaluation presented above is to validate that the results produced by the adapted algorithm are sound. Indeed, as the algorithm and the execution platform change, approximations are introduced that could change the conclusions. In parallel with correctness, we also need to evaluate the speed and performance gain resulting from the algorithm adaptations. To that extent, we ran benchmarks by varying the length of the time series (DR3, DR4-G and DR4-CCD), the number of frequencies to identify (from 1 to 10) and the binning (70 bins or 1). We ran our benchmark suite over

the most recent machines among the ones we envision to use for the final calculation. The characteristics of the different machines used are described in Table 3.

**Table 3** Platforms used for running the benchmark suite

| Platform | CPU | RAM | GPU | GPU RAM | API |
|---|---|---|---|---|---|
| GaiadbGPU | AMD EPYC 9354 | 1 TB | NVIDIA A100 | 80 GB | PTX |
| Gaiainteractive | AMD EPYC 9354 | 500 GB | NVIDIA L4 | 40 GB | PTX |
| GraceHopper | Grace CPU | 600 GB | NVIDIA H100 | shared with CPU | PTX |

The results of the benchmarks are averaged over 5 runs that follow 2 warmup iterations, which are meant to allow the just-in-time compiler optimise the code. Figure 5 and Figure 6 present the time required by the algorithm to process one source in different scenarios on the GPU and CPU, respectively.

The first dimension is the time series used, which essentially influences the completion speed because of its length. DR4 CCD time series can be as long as 992 samples with an average of 621 observations on the dataset used for this benchmark. DR3 time series do not go over 105 samples and their average is 46 observations. Finally, the DR4-G time series are on average 70 samples long and can go up to 111 and even to more than 300 in a few cases. The remaining factors, i.e., the number of frequencies to identify and the number of bins, have an impact on the refinement steps that are, in that version, running on the machine's CPU.

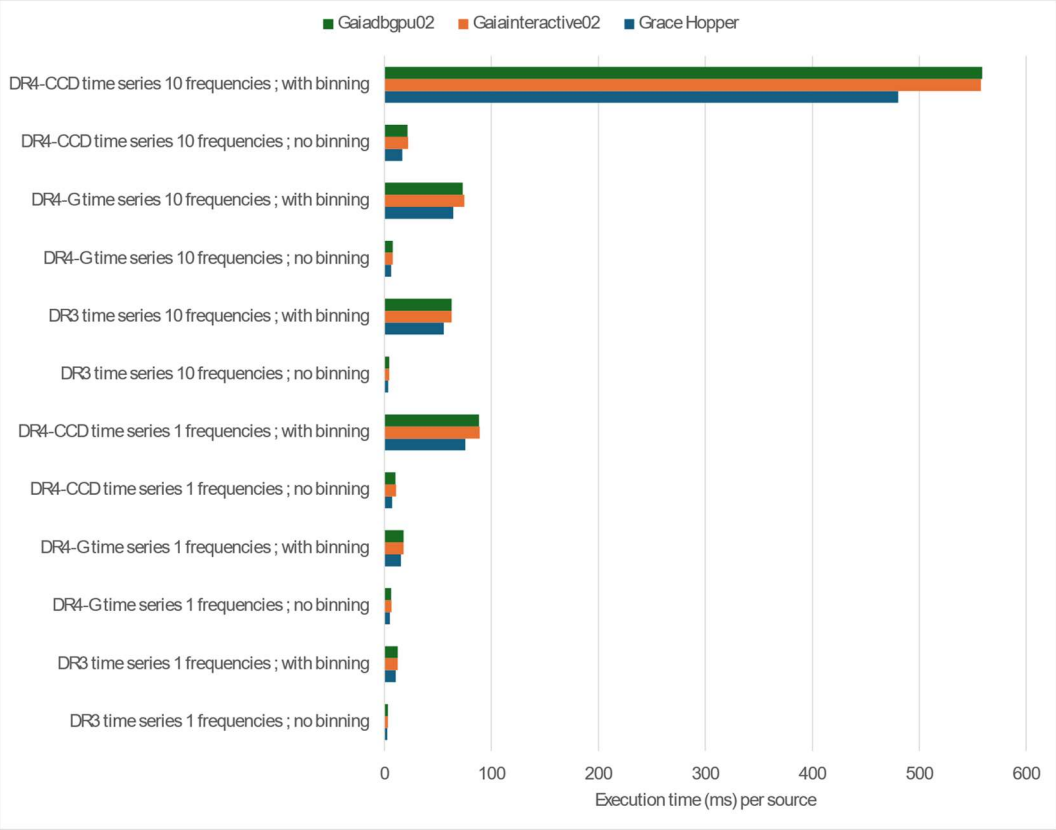



**Figure 5** Period Search execution time (ms) per source on GPU

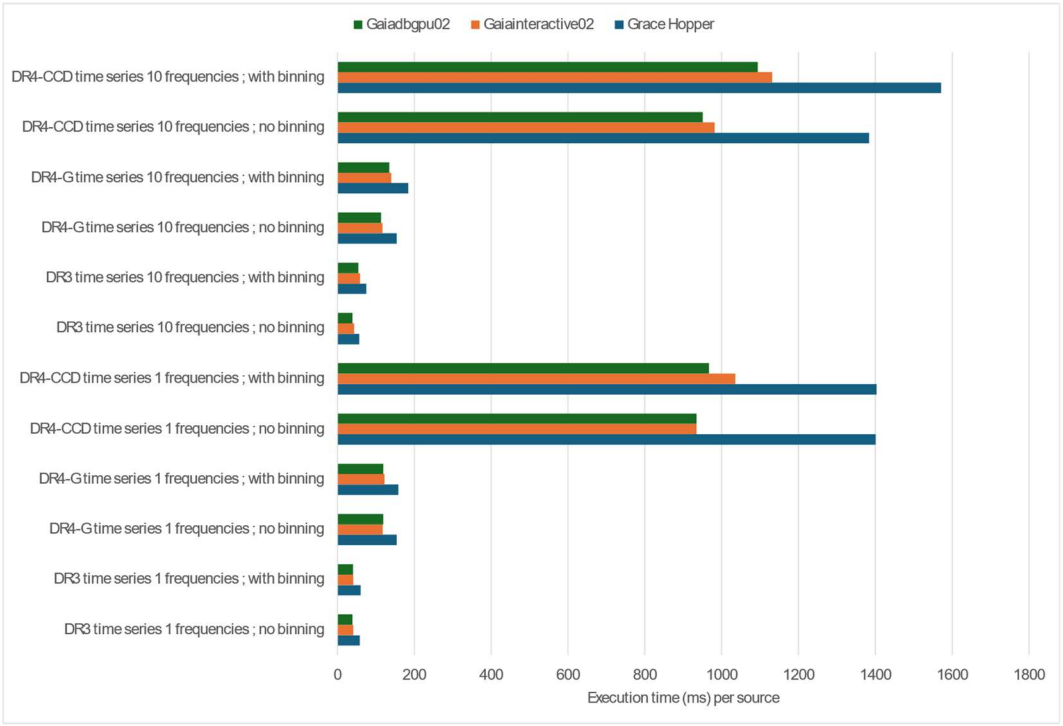



**Figure 6** Period Search execution time (ms) per source on CPU

Figure 5 and Figure 6 hint that performance can differ depending on the scenario. Both the time series length and the necessity for post-treatment seem to have an impact on performance. One may notice that between the three machines, the GraceHopper Superchip architecture exhibits better performance running the algorithm on GPU than its two competitors, while the relative performance is reversed when it comes to CPU execution. Since the Grace superchip is designed for data centres and High Performance Computing, it provides many cores whose individual performance is slightly below the other machines in our testbed. Since the CPU version of the algorithm is purely sequential, it misuses the capacity of that chip.

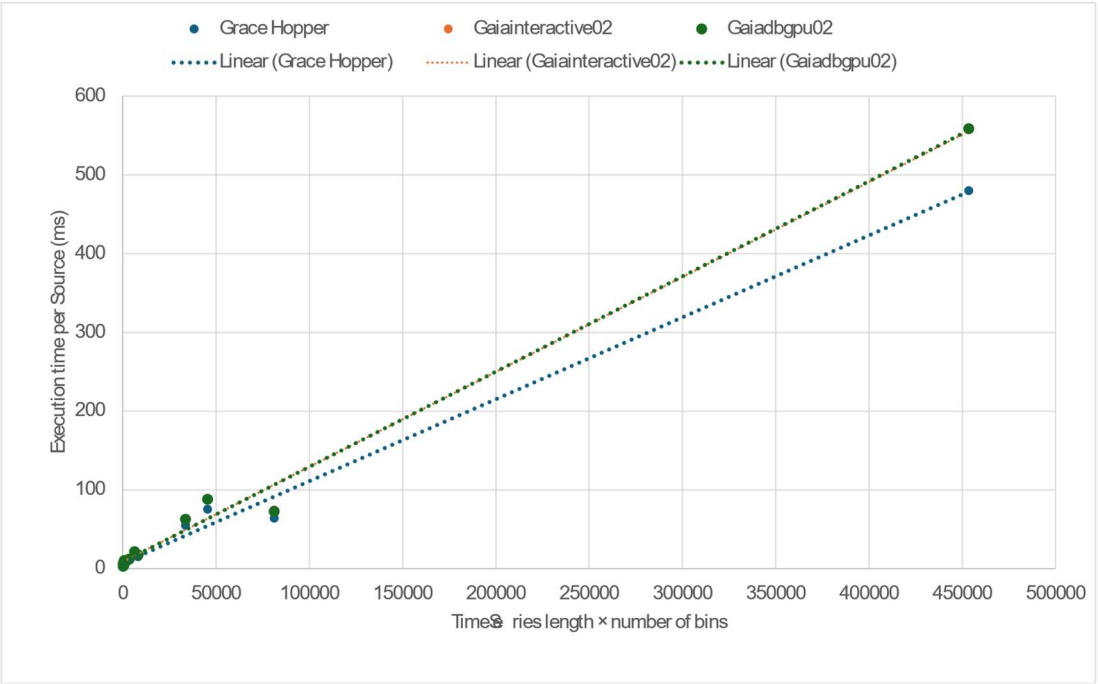



**Figure 7** Correlation between the execution time (GPU) and the product Time Series length x number of bins

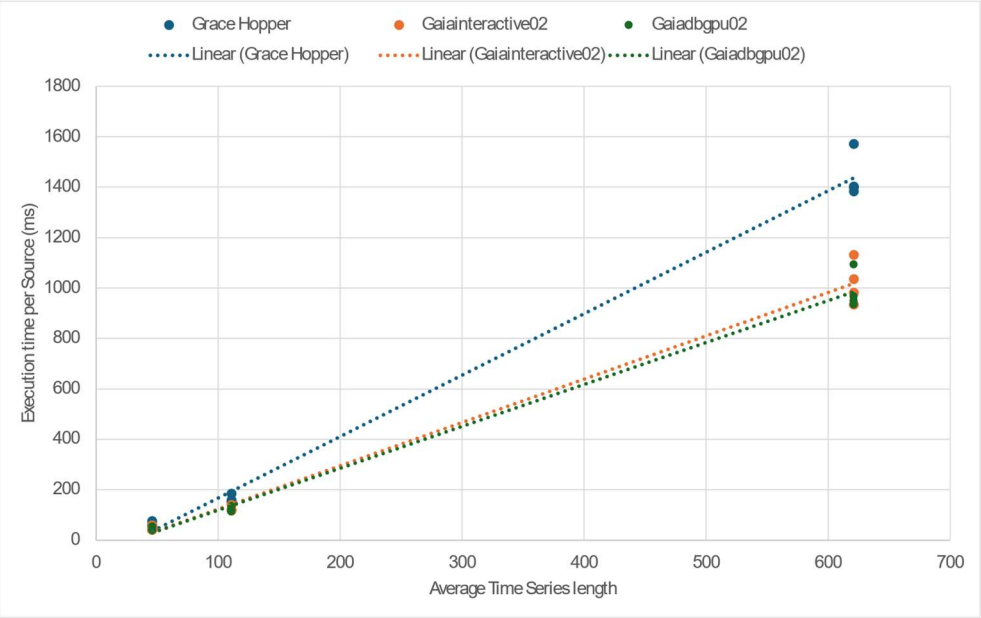



**Figure 8** Correlation between the execution time (CPU) and Time Series length

Figure 7 shows the correlation between the total execution time of the GPU algorithm and the product between the time series length and the number of bins. That product gives an indication of the number of frequencies that need to be post-processed by the CPU. The figure, on which linear fits are represented as well, reveals that there is a strong correlation between those two values. If we look at the execution times, we can see that the GPU algorithm can achieve significant speedups (between 12 and 91 on regular machines) when the post-processing phase is small (i.e. no bins and a single frequency to select), as shown in Table 4. We can therefore conclude that in the GPU scenario, the execution time is strongly dominated by the post-processing phase, which corroborates our intuition to optimize that part, as discussed in Section 3.4.

Replication of the same analysis on the CPU revealed that the correlation is poor between the aforementioned product and execution time. Examining though the relationship between time series length and total execution time, we find a much stronger correlation, even if not perfect, as illustrated in Figure 8. The number of bins plays a role, but less prominent than in the GPU implementation.

**Table 4** Speedups on different platforms and setups

| | GraceHopper | Gaiainteractive | Gaiadbgpu |
|---|---|---|---|
| DR3 time series 1 frequencies; no binning | 23,96 | 12,67 | 12,38 |
| DR3 time series 1 frequencies; with binning | 5,64 | 3,34 | 3,32 |
| DR4-G time series 1 frequencies; no binning | 31,19 | 17,65 | 18,76 |
| DR4-G time series 1 frequencies; with binning | 10,26 | 6,83 | 6,71 |
| DR4-CCD time series 1 frequencies; no binning | 196,88 | 85,39 | 91,47 |
| DR4-CCD time series 1 frequencies; with binning | 18,60 | 11,64 | 10,95 |
| DR3 time series 10 frequencies; no binning | 15,73 | 9,65 | 8,65 |
| DR3 time series 10 frequencies; with binning | 1,365 | 0,94 | 0,86 |
| DR4-G time series 10 frequencies; no binning | 24,64 | 14,84 | 14,59 |
| DR4-G time series 10 frequencies; with binning | 2,88 | 1,873 | 1,85 |
| DR4-CCD time series 10 frequencies; no binning | 82,98 | 44,22 | 44,36 |
| DR4-CCD time series 10 frequencies; with binning | 3,27 | 2,029 | 1,96 |

In a nutshell, the evaluation shows that we are progressing in the right direction, even if there still exists some optimisation margin.

# 4 HPC/Cloud Database Acceleration for Scientific Computing (SED)

## 4.1 Pilot description

SED is the provider of a petabyte-scale Massively Parallel Processing (MPP) database based on PostgreSQL for the Gaia Variability studies of University of Geneva. Our MPP database is based on TBase[4], a fork of Postgres-XL by the Chinese company Tencent[5] with improved stability and scalability. While Tencent kept the BSD license, both the source code and license have diverged significantly from the original open-source driven non-MPP PostgreSQL. The idea behind this pilot is two-fold:

1. To preserve and expand the European-based know-how and open-source ecosystem of MPP databases, in particular for scientific data-intensive applications. This will be achieved through turning the stand-alone TBase fork into a dynamically loadable extension of the original PostgreSQL. We will refer to this new extension as PG-XZ.
2. To exploit the accelerators' capabilities in the open source parallel and distributed PostgreSQL leveraging the AERO stack for (i) User Defined Functions (UDF) using GPU acceleration, and (ii) DB planner/executor using GPU acceleration.

## 4.2 Current status

### 4.2.1 PG-XZ fork of TBase

SED's main focus has been to introduce fixes and enhancements to its own XZ fork[6] of TBase to allow leveraging the AERO stack for acceleration.

### 4.2.2 Leveraging the AERO stack

The necessary software developments for supporting GPU acceleration of UDF via the AERO stack, and in particular via the TornadoVM is complete. Basic functionality and initial performance tests have been performed on x86_64 (AMD) systems as well on a NVIDIA GraceHopper superchip using ARM Neoverse V2 cores at hand of two independent usage examples. On the x86_64 systems, we have successfully achieved the KPI of a GPU induced performance gain of at least 2x for both examples.

Regarding the GPU acceleration of DB planner/executor, SED has adapted a version of pg-strom to run inside its TBase based MPP database in order to accelerate specific SQL queries via GPU. However, incompatibilities between MPP and the main branch of pg-strom[7] restrict us to the usage of an older version of pg-strom, which still allows for PostgreSQL v10 compatibility. In particular, this means that back porting of bug fixes and enhancements of newer pg-strom versions is increasingly difficult and may require significant effort. Distribution of GPU accelerated queries is functional, and

[4] https://github.com/Tencent/TBase
[5] https://www.tencent.com/
[6] https://github.com/AERO-Project-EU/TBase/tree/mods
[7] https://heterodb.github.io/pg-strom/

an initial assessment of GPU performance has been performed at hand of a small test query set on a x86 and an NVIDIA GraceHopper, fulfilling on both platforms the KPI of a GPU induced performance gain of at least 2x.

We also performed an initial assessment of performance on the NVIDIA GraceHopper superchip at hand of an unofficial TPC-H benchmark. However, for this benchmark we did not observe a GPU induced performance gain. Possible explanations may either be that the system is bottlenecked by the network attached storage, or that the queries in the benchmark (or the execution plans selected) are not suitable for massive parallelization.

## 4.3 Upbringing and optimization efforts

### 4.3.1 PG-XZ fork of TBase

To support the AERO software stack and its applications on the MPP database, SED added the following functionalities to the database code itself:

- Foreign Data Wrapper (FDW) support to allow for columnar storage engines like Apache Arrow.
- Execute Direct functionality for several datanodes in one query (necessary for the k-means application).

### 4.3.2 Leveraging the AERO stack

As discussed before, for the GPU acceleration of UDF in the PostgreSQL-based database, TornadoVM is leveraged. This requires an interface layer between PostgreSQL and the Java Virtual Machine (JVM). The open source pUava framework provides such an interface. Due to certain limitations of pUava, SED developed also its own custom interface, which we will refer to as MS.

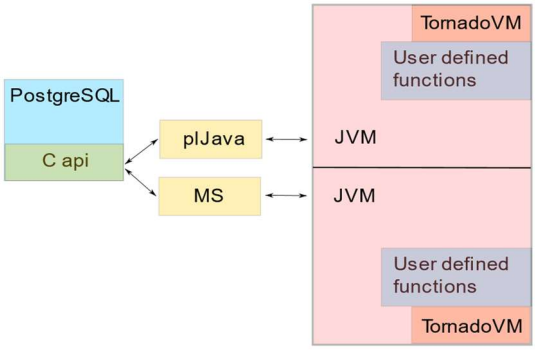



**Figure 9** Interfacing User Defined Functions (UDF) and TornadoVM (TVM) with PostgreSQL

The interface setup between PostgreSQL and UDF is depicted in Figure 9. The upbringing of TornadoVM for UDF in PostgreSQL required the following modifications to pUava[8] for functionality and/or increased performance:

- Support for specific '@' supplied JVM arguments for easy integration of TornadoVM.
- Modifications and introductions of GUC variable to enable internal arrays with native (non-object) data types for performance.
- Enabling retrieval of two-dimensional arrays from the database.

[8] https://github.com/AERO-Project-EU/pljava/tree/2d_array_return

Note that pUava creates and runs the JVM on a per session/connection basis, which has been an obstacle for its wider use. For the integration of TornadoVM into PostgreSQL, this is particularly problematic, as it hinders management of the limited available GPU resources. As the modification of pUava to run in a background worker process would have been too complicated, we rather opted to develop a new simplified interface (MS), which allows to run the JVM in a PostgreSQL background worker process (BG), and makes use of the novel Java 21 Panama API functionalities for increased performance. In particular, MS offers the following new features:

- JVM either in a user session (FG), as a user specific background worker (BG), or as a shared global background worker (GBC). Figure 10 depicts MS as a background worker process.
- Non-JDBC data interface utilising Java native memory access functionalities for increased data transfer performance.

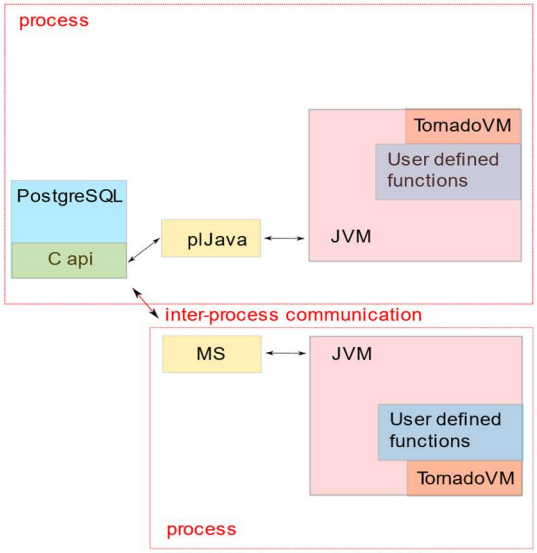



**Figure 10** Using MS as a background worker process to interface UDF and TornadoVM with PostgreSQL

The developed stack has been utilised to execute directly on a database the Period Search algorithm developed by UNIGE (Section 3.2), as well as a distributed version of the k-means unsupervised clustering algorithm. This distributed version[9] was developed in Java and makes use of the AERO stack for easy GPU acceleration thereof. We aim for the application of this distributed and GPU accelerated machine learning algorithm to very large-scale scientific Gaia data. Another application we have in mind is to perform vector similarity search in our distributed TBase database. For this purpose, we modified the Postgres extension pg_vector[10] to be able to use externally computed centroids for the IVFFlat algorithm. Note that for similar reasons as for pg_strom, we had to use an older version of pg_vector as base (v0.3.2).

In summary, for this second application example, the following have been implemented:

- K-Means via gradient descent, running directly on a distributed database.
- GPU acceleration of gradient calculation leveraging TornadoVM.
- Modification of pg_vector to be able to run in TBase and use externally supplied centroids.

[9] https://github.com/AERO-Project-EU/xz-unsupervised
[10] https://github.com/AERO-Project-EU/pgvector/tree/XZ_new

In addition to these efforts, in order to accelerate the DB execution stage of distributed PostgreSQL (TBase fork) via GPU, SED adapted the existing software package pg_strom, which offers such functionality to vanilla PostgreSQL, to the distributed setting. Note that pg_strom is actively developed only for new PostgreSQL versions, and support for some older PostgreSQL versions has been dropped over time. Unfortunately, our distributed database is based on PostgreSQL v10, which is not supported anymore in pg_strom. Therefore, we had to take an older version of pg_strom (commit a14a8539cd plus some cherry picked later commits) as base for modification to the distributed setting. In detail, besides PostgreSQL v10 compatibility fixes, the following modifications[11] of pg_strom were necessary to enable functionality in distributed PostgreSQL (TBase):

- Adaptation of heap header structures.
- Adaptation of plan cost estimation to the distributed setting.
- Remote distribution of GPU scan functionality.
- Remote distribution of GPU aggregation functionality.

## 4.4 Plans for further development

### 4.4.1 PG-XZ fork of TBase

SED plans to iron out a remaining issue with the FDW implementation so that it can be used to process large scale data. This will be required for the possible performance improvements of the GPU acceleration of DB planner/executor described in Section 4.4.2.

The PostgreSQL v10 base of XL/TBase is an increasingly serious obstacle for developments in the AERO stack, as well as other projects. Unfortunately, we have deduced that the vast and aged code base, combined with lack of sufficient documentation of core design concepts of XL and TBase, make a modernization in terms of upgrading the base PostgreSQL version to a newer version, or conversion into an extension, a formidable problem. Nevertheless, we plan to increase our efforts towards this direction, as far as available project time permits. One potential avenue could be to first bring the original XC, which is well documented, forward in time, and then add selectively desired features of XL and TBase.

### 4.4.2 Leveraging the AERO stack

As necessary development and integration for GPU acceleration of UDF are complete, final benchmarks will be executed on the Rhea platform once it becomes available. In the meantime, SED has identified an opportunity for further performance optimisation by integrating the new Postgres to Java interface closer to TornadoVM; for instance, via handing over native memory segments directly to TornadoVM. Further, the Java interface could be developed to become a more fully fledged replacement for pUava. In particular, this would require developing a Postgres language handler so that function signatures would not need to be hardcoded anymore. Such a more general applicable version of our new interface could be of wider industry interest.

[11] https://github.com/AERO-Project-EU/pg-strom/tree/XZ_NEW_step2

Regarding the GPU acceleration of DB planner/executor, further optimisations of pg_strom are required to achieve a more significant performance gain. SED has identified two possible routes. Firstly, SED observed that in the TPC-H benchmark query execution plans for join operations are often not optimally generated in the distributed setting. This could be potentially improved by bringing over some of the more advanced TBase planner functionality to the distributed pg_strom planner.

Secondly, utilizing a columnar data storage model instead of a row-based model, may significantly improve performance. In the PG-XZ fork of TBase part of this project, SED has already added FDW functionality to our distributed database. This functionality can be used to utilize Apache Arrow as a columnar data store. SED will therefore explore the integration of this alternative datastore into its distributed database, and re-design the benchmark accordingly. Final benchmarking on the Rhea platform will be then performed with the re-designed benchmark. Finally, SED will consider porting of pg_strom parts from CUDA to Level Zero API only after a sufficiently high GPU induced speedup is achieved for the main use case of processing UNIGE Gaia data.

## 4.5 Performance evaluation

In general, SED has performed the evaluation reported below using an NVIDIA GraceHopper system as an alternative to the delayed Rhea platform. Besides this ARM-based system, SED has also used various x86_64 systems as baseline.

### 4.5.1 PG-XZ fork of TBase

An initial performance assessment of the database stack on an NVIDIA GraceHopper platform was performed via an unofficial version of the TPC-H benchmark. We ran 3 coordinator and 3 data nodes inside a Docker image of Ubuntu 22.04 on a single superchip. The utilized test data had been stored on network attached storage due to space limitations of the provided attached NVMe drive. Therefore, it is not possible to compare with the previous benchmarking reported in Deliverable D2.1 that was performed on x86_64 systems, which had the test data stored on per node attached RAIDs of NVMe drives. Nevertheless, the setup can be used to assess the performance gain due to GPU acceleration of the DB.

### 4.5.2 Leveraging the AERO stack

The evaluation of the GPU acceleration of UDF was initially performed using UNIGE's Period Search algorithm directly on a database. The average runtime per sample for different batch sizes at hand of a 10000-sample dataset originating from Gaia DR3 was measured single threaded (data was fetched from 1 datanode only) for different Java interfaces. For all platforms and runs, TornadoVM v1.0.5 with OpenCL backend was used.

Table 5 reports the average runtime per sample for each platform and Java interface. On a 6-node AMD cluster, equipped with dual AMD EPYC 9354 and NVIDIA A100 GPUs, we consistently observe a GPU to CPU speedup factor of around 2.5x for Period Search. In contrast, for the Intel-based 3 node setup, we observe a significant dependency on the batch size. A similar dependency on the batch size is also observed for the GraceHopper-based 3 node cluster, as well.

A closer investigation revealed that this difference between the AMD-based and the Intel- and GraceHoppper-based clusters is due to differences in the database execution plans generated by the planner. While on the AMD setup an index scan is performed, this is not the case on the GraceHopper and Intel platforms. Unfortunately, we suspect that is caused by the fact that on these two platforms the full database cluster is run on a single machine. Therefore, the automatically generated table statistics may be sub-optimal. In particular, for the sub-optimal plan a significant time is spent on data I/O, dwarfing the actual compute time.

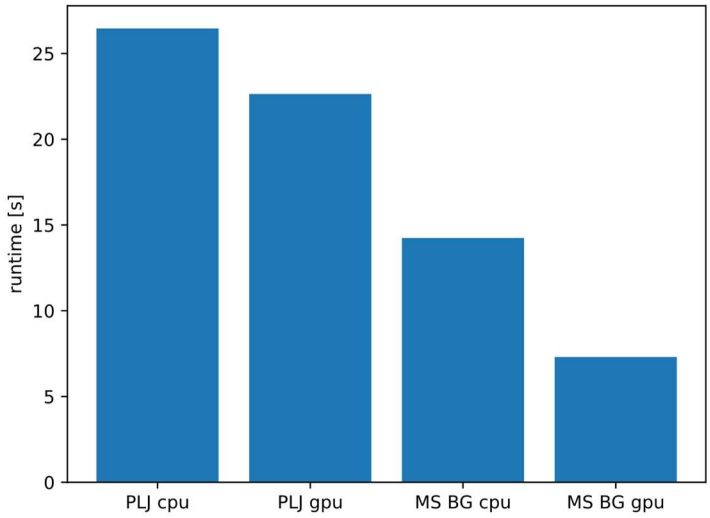
**Table 5** Average runtime (ms) per sample. Measured time includes data fetching. FG refers to JVM interfaced via a user session foreground worker, GBC via a global background worker, and PLJ to pUava.

| Platform | Batch | MS FG CPU | MS GBC CPU | PLJ CPU | MS FG GPU | MS GBC GPU | PLJ GPU |
|---|---|---|---|---|---|---|---|
| AMD EPYC with 1x NVIDIA A100 GPU | 10 | 20.6 | 20.5 | 20.8 | 8.5 | 8.6 | 8.6 |
| | 100 | 20.1 | 20.1 | 20.4 | 7.5 | 7.7 | 7.7 |
| | 1000 | 20.1 | 20.0 | 20.1 | 7.6 | 7.7 | 7.7 |
| Intel i9-13900H with RTX 4070 GPU | 10 | 354.3 | 353.0 | 358.5 | 340.7 | 339.5 | 343.6 |
| | 100 | 57.7 | 57.3 | 58.8 | 42.1 | 42.0 | 43.3 |
| | 1000 | 28.1 | 27.6 | 28.7 | 12.8 | 12.8 | 13.3 |
| NVIDIA GraceHopper | 10 | 1997.8 | 1975.9 | 2178.6 | 2003.8 | 1975.9 | 2191.5 |
| | 100 | 224.3 | 220.7 | 241.8 | 207.3 | 205.8 | 228.7 |
| | 1000 | 48.4 | 48.1 | 47.9 | 29.3 | 29.2 | 31.6 |

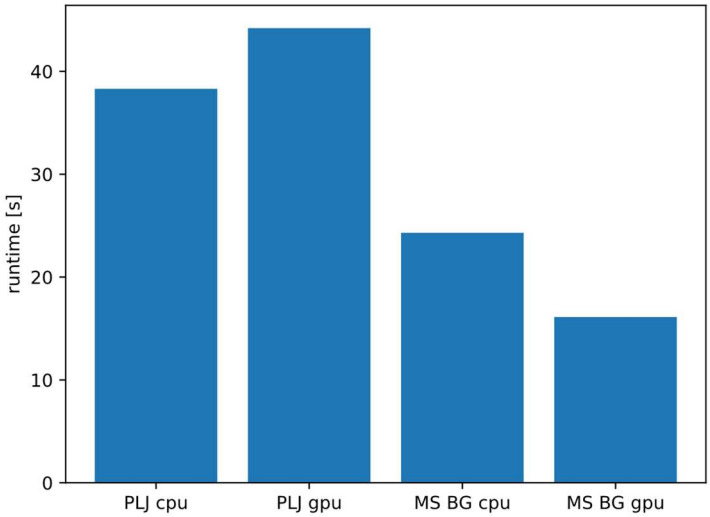
In addition to the Period Search algorithm, SED compared the performance between pLJava and MS background worker-based Java interface with and without GPU acceleration, using K-Means and a ~1.8M sample dataset with 79 features from the Gaia mission on a distributed database. To achieve GPU acceleration, TornadoVM v1.0.5 with the OpenCL backend was used. K-means was run with 20 centroids for 50 gradient step iterations with 50% of data sampled at each step and 10000 sample GPU batch size.

The benchmark was executed 13 times and the first three were discarded as warm-up. Figure 11 presents the average execution time in seconds averaged over the remaining 10 runs for all systems and interfaces. For the AMD-based system we observe a significant performance gain of around 2x for MS on GPU compared to the CPU version. Similarly, compared to pUava on CPU, MS GPU yields a performance improvement of around 3.6x. The observed performance difference of the Java interfaces can be attributed to the fact that MS provides directly the Postgres native arrays to Java via the new Panama native interface, while pUava has a more convoluted procedure for internal data copying. For the GraceHopper platform, we observe a speedup for MS CPU of ~2.2x compared to pUava CPU and a speedup of ~1.4x for MS GPU compared to MS CPU.

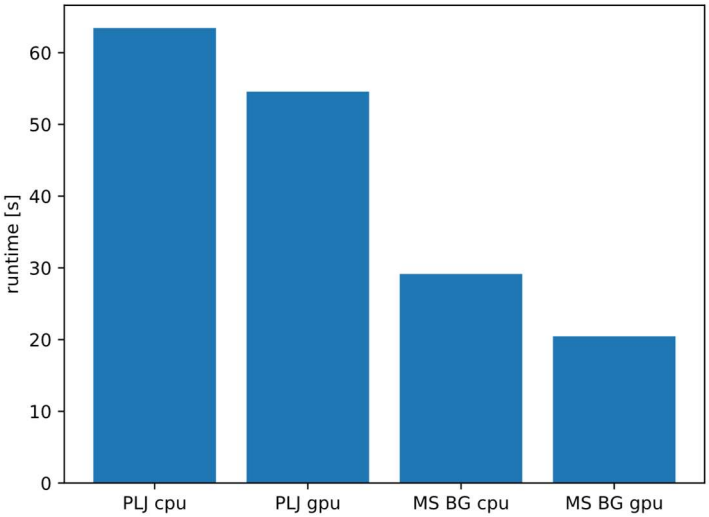
On the other hand, for the Intel-based platform, GPU acceleration improves the performance only of MS, providing a ~1.5x speedup compared to the MS CPU version and a ~2.4x speedup compared against the pUava CPU version. Interestingly, the GPU version of pUava appears to be slower than the CPU version, which indicates that pUava cannot deliver data fast enough to the GPU to compensate for the GPU overhead.




**(a)** AMD EPYC with 1x NVIDIA A100 GPU




**(b)** Intel i9-13900H with RX 4070 GPU




**(c)** NVIDIA GraceHopper

**Figure 11** Average runtime (s) of the K-Means benchmark

Finally, to evaluate the GPU acceleration of DB planner/executor we ran two benchmarks. A set of simple queries on two test tables with 100 million rows of uniformly random floats (column d1) in the range [0,1000] and a unique integer (column id), and TPC-H.

The executed queries, described in Table 6, test the functionality of GPU acceleration of basic query building blocks, i.e., scan, aggregate, group, and join. In detail, the first query tests pure aggregation; the second and third queries test conditional aggregation; the fourth query tests aggregation under conditional and grouping, and the fifth query aggregation under table join and conditionals.

**Table 6** Test queries definition.

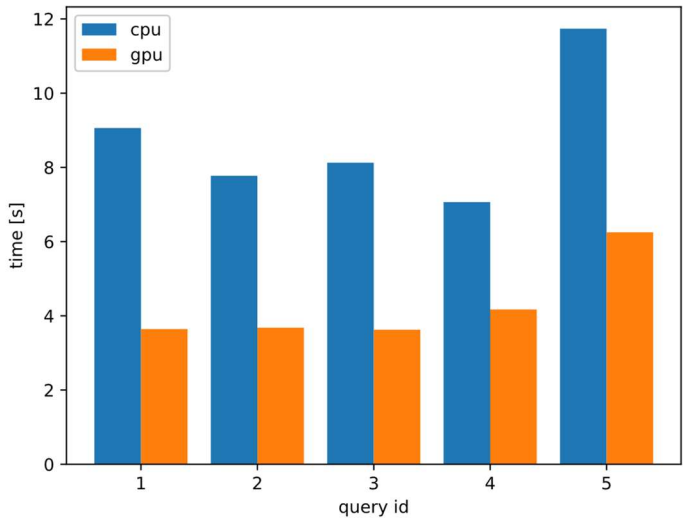
| id | Query |
|---|---|
| 1 | explain analyze select avg(d1), min(d1), max(d1) from test; |
| 2 | explain analyze select count(*) from test where d1 > 0.1; |
| 3 | explain analyze select max(d1) from test where d1 > 0.1; |
| 4 | explain analyze select avg(d1) from test where d1 < 10 group by id; |
| 5 | explain analyze select avg(test.id) from test join test2 on test.id = test2.id where test.d1 < 1 and test2.d1 < 1 |

The benchmark was executed 13 times and the first three were discarded as warm-up. Figure 12 presents the average execution time in seconds with and without GPU acceleration averaged over the remaining 10 runs for all queries on two different systems; an AMD EPYC equipped with an NVIDIA L40 GPU and a ZFS raid of 4 SSDs, and an NVIDIA GraceHopper platform with a single NVMe drive. For both systems, the benchmark is executed on a 3-node distributed database instance with all nodes located on the same single physical node.

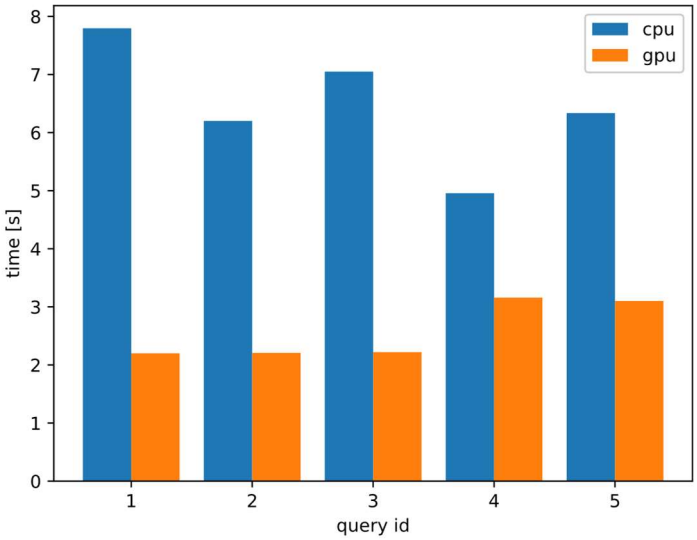
For the x86_64 system we observe on average a GPU performance gain of around 2.1x. Similarly, on the GraceHopper platform, we observe a performance gain on the GPU of around 2.6x. Even though the two platforms utilised different storage devices, it is safe to compare them as the relatively small test dataset will have been cached in memory at the end of the first three discarded executions. Under this assumption, we observe that GraceHopper to perform better around 1.4x on CPU and around 1.7x on GPU compared to the x84_64 platform.

Whereas the higher GPU performance of GraceHopper is expected, the higher CPU performance is a bit surprising. It can be probably attributed to the LPDDR5X memory used by GraceHopper, which provides higher bandwidth.

The TPC-H benchmark was executed for a single client on CPU with pg_strom based GPU acceleration on the NVIDIA GraceHopper platform for 15 times. While the CPU version completed all runs without any errors, the GPU version encountered CUDA errors due to lack of GPU memory. These were probably caused by the sharing of the machine between multiple users. The failed runs have been removed from the analysis that follows.




**(a)** AMD EPYC with 1x NVIDIA L40 GPU




**(b)** NVIDIA GraceHopper

**Figure 12** Average runtime (s) of the simple test queries

Figure 13 reports the average runtime of TPC-H with and without GPU acceleration for different data size scaling factors. It is evident that the GPU version performs consistently worse than the CPU version for all scale factors. To better understand this behaviour, Figure 14 depicts the average runtime per query for scale factor equal to 100 with and without GPU acceleration. It seems that in particular the GPU versions of queries 4, 13, and 15 are significantly slower than the respective CPU versions. The comparison of the query plans has revealed that GpuHasJoin and Bitmap index scan are not selected for the distributed GPU version, which points towards sub-optimally generated query plans that can be improved.

On the other hand, as the test data could not fit in the locally attached NVMe drive of the platform, they were stored on a network attached storage instead. There is a chance that this could impact the performance of the benchmark. Therefore, SED plans to run the exact same benchmark on the AMD EPYC platform to clarify whether the chosen TPC-H test is not suitable in general for GPU acceleration or whether the observed performance could be attributed to I/O bottlenecks.

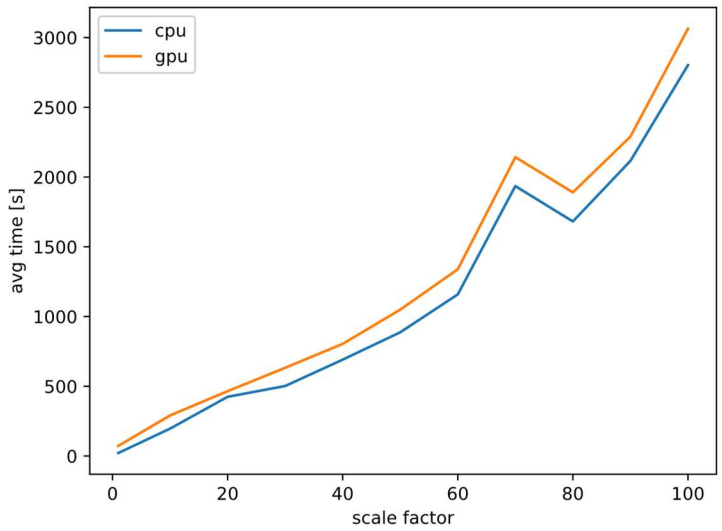



**Figure 13** Average runtime of TPC-H on NVIDIA GraceHopper with and without GPU acceleration

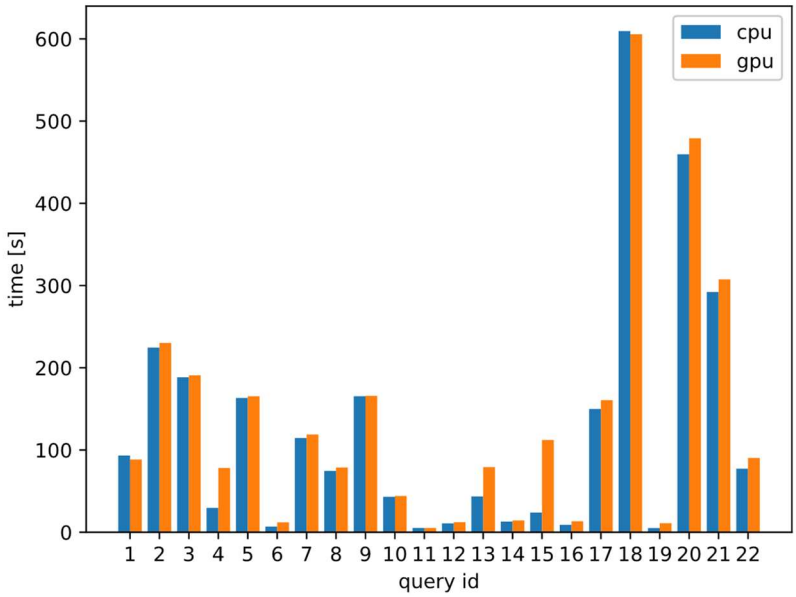



**Figure 14** Per query average runtime of TPC-H for scale factor 100 on NVIDIA GraceHopper with and without GPU acceleration

# 5 Summary

This deliverable presented the progress of the AERO pilots since the (revision of) Deliverable D2.1. More specifically, it introduced a new use case in the KTM pilot, reported on the upbringing and optimisation activities that have taken place already and presented their plans for further development. Even though the Rhea platform is not yet available, the pilots have been evaluating the performance of their code and the developed optimisations on various alternative platforms, most notably an NVIDIA GraceHopper platform.