A GPU algorithm for outliers detection in TESS light curves

S. Fiscale[1[0000−0001−8371−8525]], P. De Luca[2[0000−0001−7031−920X]], L. Inno[1,3[0000−0002−0786−7307]], L. Marcellino[1[0000−0003−2319−8008]], A. Galletti[1[0000−0002−5208−6219]], A. Rotundi[1[0000−0001−5467−157X]], A. Ciaramella[1[0000−0001−5592−7995]], G. Covone[4[0000−0002−2553−096X]], and E. Quintana[5[0000−0003−1309−2904]]

[1] Science and Technology Department, Parthenope University of Naples, Naples, Italy,
stefano.fiscale001@studenti.uniparthenope.it
{laura.inno,livia.marcellino,ardelio.galletti,rotundi,angelo.ciaramella}@uniparthenope.it
[2] Department of Computer Science, University of Salerno, Fisciano, Italy,
deluca@ieee.org
[3] INAF-Osservatorio Astronomico di Capodimonte, Salita Moraliello, Napoli
[4] Department of Physics "Ettore Pancini", University of Naples Federico II, Naples, Italy,
giovanni.covone@unina.it
[5] NASA Goddard Space Flight Center, Greenbelt, MD.
elisa.quintana@nasa.gov

Abstract. In recent years, Machine Learning (ML) algorithms have proved to be very helpful in several research fields, such as engineering, health-science, physics etc. Among these fields, Astrophysics also started to develop a stronger need of ML techniques for the management of big-data collected by ongoing and future all-sky surveys (e.g. Gaia, LAMOST, LSST etc.). NASA's Transiting Exoplanet Survey Satellite (TESS) is a space-based all-sky time-domain survey searching for planets outside of the solar system, by means of transit method. During its first two years of operations, TESS collected hundreds of terabytes of photometric observations at a two minutes cadence. ML approaches allow to perform a fast planet candidates recognition into TESS light curves, but they require assimilated data. Therefore, different pre-processing operations need to be performed on the light curves. In particular, cleaning the data from inconsistent values is a critical initial step, but because of the large amount of TESS light curves, this process requires a long execution time. In this context, High-Performance computing techniques allow to significantly accelerate the procedure, thus dramatically improving the efficiency of the outliers rejection. Here, we demonstrate that the GPU-parallel algorithm that we developed improves the efficiency, accuracy and reliability of the outliers rejection in TESS light curves.

Keywords: ML · light curves · GP-GPU · parallel algorithm · HPC

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77977-1_34

1 Introduction

Machine Learning (ML) can be defined as computational methods exploiting experience to improve performance or to make accurate predictions. Experience refers to the past ground-truth available to the learner. Typically, ML approaches are based on two main steps: the *pre-processing* phase - retrieve and assimilate specific information by starting from an initial raw dataset; *training* phase - where an ad-hoc model is trained by using the above standardized data. In order to compute an adequate forecast, several data assimilation techniques are performed. In order to avoid any inconsistency in the results, anomalous values in the raw data must be detected and removed.

This is achieved by using specific *data-cleaning* techniques, such as removal of inconsistent observations, filtering-out outliers, missing data handling. In exoplanets surveys, pre-processing procedures are very helpful due to random nature of retrieved data [1]. In particular, the NASA's Transiting Exoplanet Survey Satellite, TESS, collected a large dataset of high-precision photometric observations for more than 200,000 stars [2, 3].

Given the low signal to noise ratio (SNR) that characterizes transit signals, dataset are often corrupted by different instrumental *systematic* error, which needs to be removed, in order to perform a correct analysis. In this work, we focus on the rejection of outliers in TESS light curves by adopting the $Z-$Score method, one of the most effectively and widely adopted to this purpose.

Most pre-processing pipelines including outliers rejection are currently implemented by following a sequential approach that, due to the huge data dimension, requires several waiting hours prior to the training phase.

It is well-known that High-Performance Computing (HPC) offers a powerful tool to overcome this issue, thanks to its advanced parallel architectures. In particular, the high computational power of Graphics Processing Units (GPUs) allows to analyse a huge data volume by following the Single Instruction Multiple Thread (SIMT) paradigm. Moreover, thanks to novel GPUs architectures, the numerical stability of each computation is more guaranteed with respect to any CPU computations [4].

In this work, we propose a GPU-parallel algorithm based on the $Z-$Score method for outliers detection in TESS light curves. The parallel implementation exploits the Compute Unified Device Architecture (CUDA) framework [5] for achieving an appreciable gain of performance. Therefore, a consistent workload distribution has been performed by an ad-hoc Domain Decomposition (DD) approach. Moreover, in order to accelerate the reading-writing memory operations, a suitable memory strategy has been designed.

The paper is organized as follows. In section 2 related works are reviewed. Section 3 recalls some preliminaries about TESS light curves and the $Z-$Score method. In Section 4, the underlying domain decomposition strategy and the GPU-CUDA parallel algorithm are provided. The experiments discussed in Section 5 confirm the efficiency of the proposed implementation in terms of performance. Finally, our conclusions are presented in Section 6.

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77977-1_34

2 Detection of transits in TESS light curves

This section briefly summarises the purpose of TESS space mission and the different analysis of its light curves presented in the literature. After the first planets were discovered outside the solar system in 1992 [19], and the subsequent successful observations of additional ones, astronomers started an intense and multi-approached search for exoplanets. With the advance of newest technologies, the search for exoplanets has become one of the most dynamic research field in astronomy.

TESS is the most recent transit survey to join the exoplanets-hunting field. It is a space-borne NASA mission launched in 2018, whose principal objective is to discover transiting Earths and Super-Earths orbiting nearby, bright dwarf stars [6]. In order to achieve its goal, TESS employs a "stare and step" observation strategy: its four on-board cameras observe a 24-by-96-degree sector of the sky for 27 days, then the spacecraft is reoriented to observe the next sector, which has a marginal overlap with the previous one. Over the past two years, TESS used this strategy to tile the entire sky with a total of 26 observation sectors.

The time-series photometric observations collected by TESS have been analysed in several studies with different computational approaches. Here, we focus on those based on ML techniques.

In particular, in [7, 8] ML approaches based on Deep Neural Networks (DNNs) are applied to predict and classify stellar properties from TESS noisy and sparse time series data. Among the family of DNNs, the authors exploit a powerful model (a Convolutional Neural Network) for classifying TESS light curves. Nevertheless, before training on the data, several pre-processing steps are required. The pre-processing pipelines embedded into the ML algorithms mentioned above rely on a sequential approach to the problem, thus requiring long execution times, somehow affecting the performance of the whole process.

Many efforts have been done in the last years for improving performances. In particular, in [9] the framework BATMAN is presented, together with a state of the art study. BATMAN is a Python package for modeling exoplanet transit and eclipse light curves. It supports calculation of light curves for any radially symmetric stellar limb darkening law, using a new integration algorithm for models that cannot be quickly calculated analytically. The code uses C extension modules to speed up model calculation and it is developed for multi-core environment, by means of the OpenMP library.

3 Mathematical background

TESS performs a photometric collections of a large number of stars with a regular time sampling, i.e. a two minutes cadence over a baseline of 27 days. These images are read-out and made available to the community as target pixel files (TPFs) and light curves [10], i.e. tables of the flux emitted by the source over time (Figure 1).

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77977-1_34

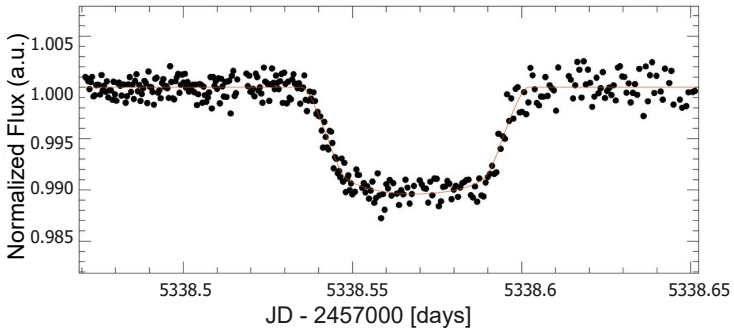

Fig. 1. The plot shows the **the normalized** flux of a star measured by TESS over the course of **approximately one** hour. The decrease in the flux at the center is due to an exoplanet tranist.

When an exoplanet transits in front of its host star (as seen from Earth), a portion of the star light is blocked out and a decrease in the observed flux is measured [11]. This phenomenon is described by a complex model, including quadratic or nonlinear limb darkening effects [20, 21]. However, we can simplify it here for the sake of clarity by considering the ratio of the observed variation in flux, ΔF, to the stellar flux, F, proportional to:

$$\frac{\Delta F}{F} \propto \frac{R_p^2}{R_*^2} \tag{1}$$

where R_p and R_* are the planetary and stellar radii respectively, with the radius of the star related to its luminosity L_* and temperature T_* by the Stefan-Boltzmann law:

$$L_* = 4\pi R_*^2 \sigma_{Boltz} T_*^4.$$

This relation shows that depending on the star luminosity and the planet size, the detection of the transit can be difficult because of the low contrast, making the removal of possible *outliers* critical to the analysis.

Outliers can be defined as extreme entries hovering around the fringes of a normal distribution that are unlikely part of the population of interest, defined as the one within ± 3.0 standard deviations σ from the mean μ [12].

Let us define:

$$\mathcal{D} = \{X_0, X_1, \ldots, X_{d-1}\} \tag{2}$$

where d is the cardinality of $\mathcal{D}$, i.e. the number of light curves we want to pre-process. For each light curve $X_i, i = 0, \ldots, d-1$ of size N, it is: $X_i \sim U[a, b] \quad (a, b \in \mathbb{R})$, where U is a uniform distribution within the boundary values

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77977-1_34

$[a, b]$. Thus, we can set the class:

$$\mathcal{C} = \begin{cases} 1, & \text{outlier} \\ 0, & \text{not-outlier} \end{cases} \tag{3}$$

In our case, $X_i = \{x_0, x_1, \ldots, x_{N-1}\}$ refers to the measured flux F and the range $[a, b]$ represents the boundary values of TESS light curves.

The most widely adopted technique for attributing the class 0 or 1 to any data point $x_j \in X_i$ is the $Z-$Score method, defined as follows:

$$Z_j = \frac{x_j - \mu}{\sigma}. \tag{4}$$

Assuming as threshold the value $\pm 3.0\ \sigma$, we assign the value $\lambda_j = \mathcal{C}$ at each data point x_j according to the following rule:

$$\lambda_j = \begin{cases} 1, & |Z_j| > 3 \\ 0, & |Z_j| \leq 3. \end{cases} \tag{5}$$

The above discussion allows us to introduce the following scheme, Algorithm 1, to solve the numerical problem.

Algorithm 1 Sequential algorithm

Input: $\mathcal{D}, T, N$

1: `matD[i]` $= X_i$
2: **for** each i in `matD` **do**
3: `compute:` μ_i, σ_i
4: `x[j] =` $X_i^{(j)}$ % *Z-Score*
5: **for** each j=1 to N **do**
% `compute` λ_j % *as in* (4)
6: $\lambda_{X^{(i)}} \leftarrow \lambda_j$
7: **end for**
8: $\lambda_{\mathcal{D}} \leftarrow \lambda_{X^{(i)}}$
9: **end for**

Output: $\lambda_D[\,]$

The above algorithm is designed by the following steps:

STEP 1 Loading each light curve X_i as row into matrix `matD`. For each row, both μ and σ are computed.

STEP 2 Starting from previous computed information, a new *loop-for* starts for computing $Z-$Score value for each $x_j \in X_i$. By using (5), a value of $\mathcal{C}$ is assigned to x_j. Hence, it is added to $\lambda_{X^{(i)}}$, which contains the overall classification results.

STEP 3 In order to obtain the ensemble λ_D, a collection operation is performed on each $\lambda_{X^{(i)}}$, where λ_D contains the classification results of every $\lambda_{X^{(i)}}$.

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77977-1_34

Starting from good results achieved, according to aims of $Z-$Score method as in [22], we observed that executing the algorithm sequentially even on the latest-generation CPU requires very long execution times, due to polynomial computational complexity. In particular its upper bound is $\mathcal{O}(dN^2)$.

4 GPU-Parallel algorithm

In order to improve the performance of the Algorithm 1, we decided to take advantage of the computational power offered by the most modern HPC architectures [14–17]. We developed a parallel implementation through GPU architecture, based on a suitable Domain Decomposition (DD) strategy. Moreover, in order to distribute the overall work to a GPU grid of threads and then to design a good DD, the shared-memory is exploited. In fact, with a proper management of this kind of memory, available from the NVIDIA-CUDA environment for GPUs that we used, a considerable improvement, in terms of execution times, has been achieved.

We started by the following consideration: light curves of the TESS dataset $\mathcal{D}$ are related to different stars, which means that there is no correlation between data of two different light curves. Thanks to this, data distribution can be made by assigning at each CUDA-block the data structure corresponding to a single light curve, using d CUDA-block to process all data in a parallel-embarrassingly way. Each block will exploit its own shared memory to store and computing data of each single light curve.

More specifically, in each block th threads work, following the SIMT paradigm, on N elements of each array $B_i, i = 0, ..., d-1$, corresponding to a light curve, indicated with X_i in the previous section. The workload distribution is organized by assigning to each thread a sub-set of data of size L, where:

$$L = \begin{cases} \frac{N}{th} & \text{if mod(N,th)} = 0 \\ \frac{N}{th} + 1 & \text{if mod(N,th)} \neq 0 \end{cases} \tag{6}$$

The described DD schema is illustrated in Figure 2.

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77977-1_34

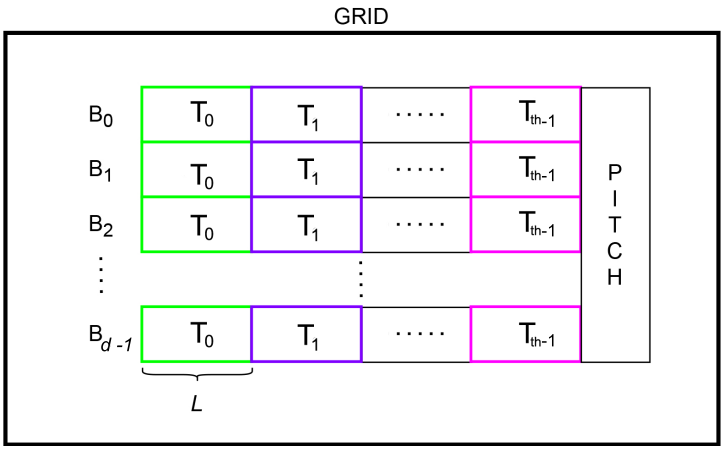

Fig. 2. Domain decomposition strategy with pitch allocation method.

The d light curves are distributed respectively on d blocks, in each of them th threads are activated to process each light curve. In particular, for each threads a chunk of L is processed. Moreover, for each row, the columns number is computed by using the pitch method for an efficient memory allocation and storage. In fact, the pitch method provides to compute the length each row should have so that the number of memory accesses to manage (fetch/insert) data is minimized, [18] and this can be done, exploiting the CUDA routine `cudaMallocPitch`. After the data distribution, in each block the values μ and σ have to be computed, in parallel by all threads activated, and stored efficiently using the fast read/write access of the shared memory. Therefore, the global values of μ and σ will be used to compute the Z-score of each light curve element, in each CUDA block. The parallel procedure, which includes the implemented CUDA kernel, is reported in Algorithm 2.

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77977-1_34

Algorithm 2 GPU algorithm

Input: $\mathcal{D}, th$

STEP 1: % *Domain Decomposition.*

L = (6) % *chunk size*

`i = threadIdx.x+(blockDim.x` $\times$ `blockIdx.x)` % *thread id*

$X_k \leftarrow \mathcal{D}[k]$% *load light curve k*

STEP 2:% *Mean value computation*

for each thread i **do**

x_i %*local sums*

$count = count + 1$

end for

`master thread 0:`

$\mu_k \leftarrow \frac{1}{count_i} \sum_{i=0}^{th} x_i$

STEP 3: *Threads synchronization and Z-score computation*

for each thread i **do**

$sq_i = \sum_{j \in L} (x_j - \mu)^2$

end for

`master thread 0:`

$\sigma_k \leftarrow \sqrt{\frac{\sum_{i=0}^{th}(sq_i)}{count_i}}$

for each thread i **do**

for $j_i \in L_i$ **do**

%*compute Z−Score as in Eq.* (4)

$\lambda_{k^{(i)}} \leftarrow \lambda_j$

end for

end for

STEP 4:% *Data collection*

for each thread i **do**

$\lambda_{\mathcal{D}} \leftarrow \lambda_{k^{(i)}}$

end for

Output: $\lambda_D[\,]$

Main operations of Algorithm 2 are summarized in the following steps:

STEP 1 - Data distribution: the Domain Decomposition approach describe previously is here used by computing for each thread its id-number and its portion of the data set $\mathcal{D}$. Therefore, each block $k = 0, ..., d-1$ of threads loads locally a row of $\mathcal{D}$ by extracting the related light curve stored into X_k, which is the data-structure representing B_k, the $k-th$ light curve.

STEP 2 - Mean value computation: each thread computes a local sum related to its chunk. Hence, starting from each chunk's sum, the master thread computes the global sum and the mean value, which is just stored into the shared memory.

STEP 3 - Threads synchronization and Z-score computation: after the parallel computation and in order to preserve the memory consistent, all threads of each

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77977-1_34

block have to be synchronized by using the CUDA routine `__syncthreads()`. Therefore, in a similar way the standard deviation is computed. Finally, the $Z-$Score value is computed for each element within the chunk (line 14), leveraging on shared-memory data as in Eq. (4).

STEP 4 - Data collection: the final operation is to collect the local λ values, computed by each thread. In particular, in order to perform a correct memory access and avoid any memory contention, each thread i copies into the global $\lambda_{\mathcal{D}}$, in the position L_i, the local results.

5 Results

In this section we discuss different experimental tests in order to highlighting the gain of performance achieved by the GPU-parallel algorithm with respect to the sequential CPU-based one. Two different metrics are used: we first show a comparison between the two implementations in terms of execution times; then we present a comparison in terms of Gflops.

The parallel code runs on a machine with the following technical specifications:

- 1 x CPU Intel i7 860 with 4 cores, 2 threads per core, 2.80 Ghz, 8 GB of RAM
- 1 x GPU NVIDIA Quadro K5000 with 1536 CUDA Cores, 706 MHz Core GPU Clock, and 4 GB 256-bit, GDDR5 memory configuration.

In order to exploit the overall parallel powerful nature of GPUs, the CUDA framework has been adopted. The high programming flexibility of this framework allow us to model specific memory strategies for improving the performance. In fact, different experimental tests with different ad-hoc memory-based strategies will be considered. The main aim of these tests is to find the best strategy so that we can confirm the reliability of our proposed algorithm. Two different GPU memory strategy will be discussed, the first one with basic global memory data allocation, the second one making use of the *pitch* technique.

The following tests have been performed by repeating the executions 10 times.

Test 1: CPU vs GPU execution times comparison.

The first experimental test shows the gain in execution time of the GPU version with respect to CPU-version.

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77977-1_34

Table 1. Execution times: CPU vs GPU for different input sizes. $N = 16741$, `block` $\times$ `thread` $= d \times 1024$.

d	**Sequential Time (s)**	**GPU Time (s)**
5 000	1.591	0.142
10 000	2.970	0.284
15 000	4.392	0.426
20 000	6.523	0.521
25 000	8.114	0.724

Table 1 highlights the appreciable gain of performance of the GPU-based version with respect to the CPU version. As d increases, the execution time of the GPU-based algorithm decreases of 90% with respect to that of the sequential algorithm on the CPU Here, the memory strategy is based on global memory storing of $\mathcal{D}$ without pitch. In particular the reliability of parallel algorithm is confirmed by varying the input data problem. The CUDA configuration `block` $\times$ `threads` is fixed to $d \times 1024$, according to Domain Decomposition strategy adopted. Next text deals with find the best CUDA configuration.

Test 2: GPU execution times comparison with different memory strategies. Despite the appreciable performance obtained in previous experimental test, we observed that the execution times can slow down by choosing a different memory strategy. In particular the next test adopt the global memory allocation with *pitch* technique.

Table 2. Execution times: GPU with different memory strategies by varying d, `block` $\times$ `thread` $= d \times 1024$.

d	**GPU no pitch (s)**	**GPU Pitch (s)**
5 000	0.142	0.125
10 000	0.284	0.260
15 000	0.426	0.384
20 000	0.521	0.503
25 000	0.724	0.672

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77977-1_34

Table 2 shows the execution times for two different memory strategies. In particular, the first column includes the results obtained with a GPU version with global memory storing strategy without the pitch method, while in the second column the ones with the pitch method. Indeed, an improving lower than 10% has been achieved. More in details, the pitch method is able to determine the length that each row should have. Therefore, the number of memory accesses to manage (fetch/insert) data is minimized.

Test 3: GPU times with shared-memory strategy Here, we show a comparison with another memory strategy applied to our parallel algorithm.

Table 3. Execution times: GPU with shared-memory strategies by varying d, `block` $\times$ `thread` $= d \times 1024$.

d	**GPU** GB_t **(s)**	**GPU** SM_t **(s)**
5 000	0.125	0.121
10 000	0.260	0.255
15 000	0.384	0.375
20 000	0.503	0.495
25 000	0.672	0.617

where GB_t and SM_t are, respectively, the GPU times with the global memory strategy and the shared-memory strategy, both with the pitch method. Table 3 confirms that our domain decomposition approach combined with ad-hoc memory-based strategy is more performing with respect to previous memory configurations. In particular, the shared-memory is involved in our algorithm due to its hardware position close to the threads. In other words, the access time is clearly reduced. In fact, according to the CUDA architecture, the shared-memory is placed close to each blocks threads of grids. Moreover, by considering the advantages achieved in the previous test, the pitch method was adopted in this memory storing strategy as well.

Test 4: GPU times with different CUDA configurations. Here we show the execution times achieved by varying the CUDA configurations and input size problem. In particular, thanks to the appreciable results of last memory strategy just shown, the next test is based on this configuration in order to retrieve the best parallelism available. According to pitch-based memory strategy applied, the parameter N is set to 16768.

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77977-1_34

Table 4. GPU execution times comparison by varying input size and CUDA threads configuration, `block` $\times$ `threads` = $d \times t$.

	# Threads				
d	$t = 128$	$t = 256$	$t = 512$	$t = 1024$	$t = 2048$
5 000	0.193	0.180	0.146	0.121	0.302
10 000	0.405	0.378	0.306	0.255	0.785
15 000	0.598	0.558	0.452	0.375	0.815
20 000	0.785	0.732	0.592	0.495	0.845
25 000	0.984	0.918	0.744	0.617	0.875

Table 4 is used to find the ideal CUDA configuration in order to confirm the best exploit of overall GPU parallelism offered. We observe that the best configuration relies on 1024 threads with the shared-memory strategy. In particular, according to coalescing rules of CUDA, a large threads number slow down the performance due to hardware resource limits. In other words, the number of Streaming Multiprocessor of our GPU is saturated, and an implicit overhead is introduced. In fact, the remaining number of blocks will be scheduled in pipeline mode when the run-time blocks complete the related computation.

Test 5: FLOPS comparison. In this test, in order to confirm the gain of performance with respect to the sequential implementation, we analyse an addiction theoretical metric, i.e. the performance analysis in Giga floating point operations per second (Gflops). The results obtained are referred to the previous tests and to the best observed CUDA configuration.

Table 5. Performance in terms of Gflops.

Input size	Gflops CPU	Gflops GPU
5 000	2.634	146.446
10 000	5.645	257.568
15 000	8.590	276.080
20 000	10.823	264.989
25 000	12.915	328.541

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77977-1_34

Table 5 shows the gain in terms of Gflops, comparing the operation number per seconds (CPU vs GPU) in several execution by varying the input size. We observe an appreciable enhancement of performance obtained by exploiting the GPU architecture. Indeed, the table shows an increasing of performance of about 37×, in terms of Gflops, for all the executions.

The results presented above confirm the reliability and growing of the performance related to the parallel implementation.

6 Conclusion

In this work, we presented a GPU-parallel algorithm based on $Z-$Score for detecting outliers in TESS light curve. The method we presented here allows to significantly improve the efficiency of the outliers detection and cleaning on TESS data. The same approach can be employed to all the remaining pre-processing steps (such as e.g. noise reduction, flattening and folding of the light curves) required to produce assimilated data, which are necessary for ML applications. Therefore, we anticipate that a complete pre-processing pipeline based on GPU-parallel computation will significantly contribute to accelerate the following processes required during Machine Learning flow operations.

References

1. Shallue, C. J., & Vanderburg, A. (2018). Identifying exoplanets with deep learning: A five-planet resonant chain around kepler-80 and an eighth planet around kepler-90. The Astronomical Journal, 155(2), 94.
2. Ricker, George R., et al. "Transiting exoplanet survey satellite (tess)." American Astronomical Society Meeting Abstracts 215. Vol. 215. 2010.
3. Wright, Jason T., et al. "The exoplanet orbit database." Publications of the Astronomical Society of the Pacific 123.902 (2011): 412.
4. Krüger, J., & Westermann, R. (2005). Linear algebra operators for GPU implementation of numerical algorithms. In ACM SIGGRAPH 2005 Courses (pp. 234-es). ISO 690
5. https://developer.nvidia.com/cuda-zone
6. Space, J. J. P. N. G. (2018). Transiting Exoplanet Survey Satellite (TESS) Flight Dynamics Commissioning Results and Experiences.
7. Osborn, H. P., Ansdell, M., Ioannou, Y., Sasdelli, M., Angerhausen, D., Caldwell, D., & Smith, J. C. (2020). Rapid classification of TESS planet candidates with convolutional neural networks. Astronomy & Astrophysics, 633, A53.
8. Yu, Liang, et al. "Identifying exoplanets with deep learning. III. Automated triage and vetting of TESS candidates." The Astronomical Journal 158.1 (2019): 25.
9. Kreidberg, L. (2015). batman: BAsic transit model cAlculatioN in python. Publications of the Astronomical Society of the Pacific, 127(957), 1161.
10. https://tasoc.dk/docs/EXP-TESS-ARC-ICD-TM-0014-Rev-F.pdf
11. Abukhaled, M., Guessoum, N., & Alsaeed, N. (2019). Mathematical modeling of light curves of RHESSI and AGILE terrestrial gamma-ray flashes. Astrophysics and Space Science, 364(8), 1-16.

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77977-1_34

12. Osborne, J. W. (2013). Best practices in data cleaning: A complete guide to everything you need to do before and after collecting your data. Sage.
13. Bansal, D., Cody, D., Herrera, C., Russell, D., & Campbell, M. R. (2015). Light Curve Analysis for Transit of Exoplanet Qatar-1b. Baylor University Department of Astrophysics.
14. De Luca, P., Galletti, A., Ghehsareh, H.R., Marcellino, L., & Raei, M. A gpu-cuda framework for solving a two-dimensional inverse anomalous diffusion problem. In: Foster, I., Joubert, G.R., Kučera, L., Nagel, W.E., Peters, F. (eds) Parallel Computing: Technology Trends, Advances in Parallel Computing. Vol 36. pp 311 - 320. IOS Press, 2020.
15. Amich, M., De Luca, P., & Fiscale, S. (2020, July). Accelerated implementation of FQSqueezer novel genomic compression method. In 2020 19th International Symposium on Parallel and Distributed Computing (ISPDC) (pp. 158-163). IEEE.
16. Marcellino, L., Montella, R., Kosta, S., Galletti, A., Di Luccio, D., Santopietro, V., ... & Laccetti, G. (2017, September). Using GPGPU accelerated interpolation algorithms for marine bathymetry processing with on-premises and cloud based computational resources. In International Conference on Parallel Processing and Applied Mathematics (pp. 14-24). Springer, Cham.
17. Montella, R., Di Luccio, D., Troiano, P., Riccio, A., Brizius, A., & Foster, I. (2016, November). WaComM: A parallel Water quality Community Model for pollutant transport and dispersion operational predictions. In 2016 12th International Conference on Signal-Image Technology & Internet-Based Systems (SITIS) (pp. 717-724). IEEE.
18. https://docs.nvidia.com/cuda/cuda-runtime-api/group__CUDART__MEMORY.html
19. Wolszczan, A. & Frail, D. A. 1992, Nature, 355, 145. doi:10.1038/355145a0
20. Torres, G., Winn, J. N., & Holman, M. J. 2008, The Astrophysical Journal, 677, 1324. doi:10.1086/529429
21. Mandel, K. & Agol, E. 2002, The Astrophysical Journal Letters, 580, L171. doi:10.1086/345520
22. Zucker, S., & Giryes, R. (2018). Shallow Transits—Deep Learning. I. Feasibility Study of Deep Learning to Detect Periodic Transits of Exoplanets. The Astronomical Journal, 155(4), 147.

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77977-1_34