

PM4Py-GPU: a High-Performance General-Purpose Library for Process Mining

Alessandro Berti^{1,2}, Minh Phan Nghia, and Wil M.P. van der Aalst^{1,2}

¹ Process and Data Science Group @ RWTH Aachen, Aachen, Germany {a.berti, wvdaalst}@pads.rwth-aachen.de, minh.nghia.phan@rwth-aachen.de

² Fraunhofer Institute of Technology (FIT), Sankt Augustin, Germany

Abstract. Open-source process mining provides many algorithms for the analysis of event data which could be used to analyze mainstream processes (e.g., O2C, P2P, CRM). However, compared to commercial tools, they lack the performance and struggle to analyze large amounts of data. This paper presents PM4Py-GPU, a Python process mining library based on the NVIDIA RAPIDS framework. Thanks to the dataframe columnar storage and the high level of parallelism, a significant speed-up is achieved on classic process mining computations and processing activities.

Keywords: Process Mining, GPU Analytics, Columnar Storage

1 Introduction

Process mining is a branch of data science that aims to analyze the execution of business processes starting from the event data contained in the information systems supporting the processes. Several types of process mining are available, including *process discovery* (the automatic discovery of a process model from the event data), *conformance checking* (the comparison between the behavior contained in the event data against the process model, with the purpose to find deviations), *model enhancement* (the annotation of the process model with frequency/performance information) and *predictive analytics* (predicting the next path or the time until the completion of the instance). Process mining is applied worldwide to a huge amount of data using different tools (academic/commercial). Some important tool features to allow process mining in organizational settings are: the *pre-processing and transformation* possibilities, the *possibility to drill-down* (creating smaller views on the dataset, to focus on some aspect of the process), the availability of *visual analytics* (which are understandable to non-business users), the *responsiveness and performance* of the tool, and the possibilities of *machine learning* (producing useful predictive analytics and what-if analyses). Commercial tools tackle these challenges with more focus than academic/open-source tools, which, on the other hand, provide more complex analyses (e.g., process discovery with inductive miner, declarative conformance checking). The PM4Py library <http://www.pm4py.org>, based on

the Python 3 programming language, permits to integrate with the data processing and machine learning packages which are available in the Python world (Pandas, Scikit-Learn). However, most of its algorithms work in single-thread, which is a drawback for performance. In this demo paper, we will present a GPU-based open-source library for process mining, PM4Py-GPU, based on the NVIDIA RAPIDS framework, allowing us to analyze a large amount of event data with high performance offering access to GPU-based machine learning. The speedup over other open-source libraries for general process mining purposes is more than 10x. The rest of the demonstration paper is organized as follows. Section 2 introduces the NVIDIA RAPIDS framework, which is at the base of PM4Py-GPU, and of some data formats/structures for the storage of event logs; Section 3 presents the implementation, the different components of the library and some code examples; Section 4 assess PM4Py-GPU against other products; Section 5 introduces the related work on process mining on big data and process mining on GPU; Finally, Section 6 concludes the demo paper.

2 Preliminaries

This section will first present the NVIDIA RAPIDS framework for GPU-enabled data processing and mining. Then, an overview of the most widely used file formats and data structures for the storage of event logs is provided.

2.1 NVIDIA RAPIDS

The NVIDIA RAPIDS framework <https://developer.nvidia.com/rapids> was launched by NVIDIA in 2018 with the purpose to enable general-purpose data science pipelines directly on the GPU. It is composed of different components: CuDF (GPU-based dataframe library for Python, analogous to Pandas), CuML (GPU-based general-purpose machine learning library for Python, similar to Scikit-learn), and CuGraph (GPU-based graph processing library for Python, similar to NetworkX). The framework is based on CUDA (developed by NVIDIA to allow low-level programming on the GPU) and uses RMM for memory management. NVIDIA RAPIDS exploit all the cores of the GPU in order to maximize the throughput. When a computation such as retrieving the maximum numeric value of a column is operated against a column of the dataframe, the different cores of the GPU act on different parts of the column, a maximum is found on every core. Then the global maximum is a reduction of these maximums. Therefore, the operation is parallelized on all the cores of the GPU. When a group-by operation is performed, the different groups are identified (also here using all the cores of the GPU) as the set of rows indices. Any operation on the group-by operation (such as taking the last value of a column per group; performing the sum of the values of a column per group; or calculating the difference between consecutive values in a group) is also performed exploiting the parallelism on the cores of the GPU.

2.2 Dataframes and File Formats for the Storage of Event Logs

In this subsection, we want to analyze the different file formats and data structures that could be used to store event logs, and the advantages/disadvantages of a columnar implementation. As a standard to interchange event logs, the XES standard is proposed <https://xes-standard.org/>, which is text/XML based. Therefore, the event log can be ingested in memory after parsing the XML, and this operation is quite expensive. Every attribute in a XES log is typed, and the attributes for a given case do not need to be replicated among all the events. Event logs can also be stored as CSV(s) or Parquet(s), both resembling the structure of a table. A CSV is a textual file hosting an *header row* (containing the names of the different columns separated by a separator character) and many *data rows* (containing the values of the attributes for the given row separated by a separator character). A problem with the CSV format is the typing of the attributes. A Parquet file is a binary file containing the values for each column/attribute, and applying a column-based compression. Each column/attribute is therefore strictly typed. CuDF permits the ingestion of CSV(s)/Parquet(s) into a dataframe structure. A dataframe is a table-like data structure organized as columnar storage. As many data processing operations work on a few attributes/columns of the data, adopting a columnar storage permits to retrieve specific columns with higher performance and to reduce performance problems such as cache misses. Generally, the ingestion of a Parquet file in a CuDF dataframe is faster because the data is already organized in columns. In contrast, the parsing of the text of a CSV and its transformation to a dataframe is more time expensive. However, NVIDIA CuDF is also impressive in the ingestion of CSV(s) because the different cores of the GPU are used on different parts of the CSV file.

3 Implementation and Tool

In PM4Py-GPU, we assume an event log to be ingested from a Parquet/CSV file into a CuDF dataframe using the methods available in CuDF. On top of such dataframe, different operations are possible, including:

- *Aggregations/Filtering at the Event Level*: we would like to filter in/out a row/event or perform any aggregation based solely on the properties of the row/event. Examples: filtering the events/rows for which the cost is > 1000; associate its number of occurrences to each activity.
- *Aggregations/Filtering at the Directly-Follows Level*: we would like to filter in/out rows/events or perform any sort of aggregation based on the properties of the event and of the previous (or next) event. Examples: filtering the events with activity *Insert Fine Notification* having a previous event with activity *Send Fine*; calculating the frequency/performance directly-follows graph.
- *Aggregations/Filtering at the Case Level*: this can be based on global properties of the case (e.g., the number of events in the case or the throughput

time of the case) or on properties of the single event. In this setting, we need an initial exploration of the dataframe to group the indexes of the rows based on their case and then perform the filtering/aggregation on top of it. Examples: filtering out the cases with more than 10 events; filtering the cases with at least one event with activity *Insert Fine Notification*; finding the throughput time for all the cases of the log.

- *Aggregations/Filtering at the Variant Level*: the aggregation associates each case to its variant. The filtering operation accepts a collection of variants and keeps/remove all the cases whose variant fall inside the collection. This requires a double aggregation: first, the events need to be grouped in cases. Then this grouping is used to aggregate the cases into the variants.

To facilitate these operations, in PM4Py-GPU we operate three steps starting from the original CuDF dataframe:

- The dataframe is ordered based on three criteria (in order, case identifier, the timestamp, and the absolute index of the event in the dataframe), to have the events of the same cases near each other in the dataframe, increasing the efficiency of group-by operations.
- Additional columns are added to the dataframe (including the position of the event inside a case; the timestamp and the activity of the previous event) to allow for aggregations/filtering at the directly-follows graph level.
- A *cases dataframe* is found starting from the original dataframe and having a row for each different case in the log. The columns of this dataframe include the number of events for the case, the throughput time of the case, and some numerical features that uniquely identify the case’s variant. Case-based filtering is based on both the original dataframe and the cases dataframe. Variant-based filtering is applied to the cases dataframe and then reported on the original dataframe (keeping the events of the filtered cases).

The PM4Py-GPU library is available at the address <https://github.com/Javert899/pm4pygpu>. It does not require any further dependency than the NVIDIA RAPIDS library, which by itself depends on the availability of a GPU, the installation of the correct set of drivers, and of NVIDIA CUDA. The different modules of the library are:

- *Formatting module (format.py)*: performs the operations mentioned above on the dataframe ingested by CuDF. This enables the rest of the operations described below.
- *DFG retrieval / Paths filtering (dfg.py)*: discovers the frequency/performance directly-follows graph on the dataframe. This enables paths filtering on the dataframe.
- *EFG retrieval / Temporal Profile (efg.py)*: discovers the eventually-follows graphs or the temporal profile from the dataframe.
- *Sampling (sampling.py)*: samples the dataframe based on the specified amount of cases/events.
- *Cases dataframe (cases_df.py)*: retrieves the cases dataframe. This permits the filtering on the number of events and on the throughput time.

Table 1: Event logs used in the assessment, along with their number of events, cases, variants and activities.

Log	Events	Cases	Variants	Activities
roadtraffic_2	1,122,940	300,740	231	11
roadtraffic_5	2,807,350	751,850	231	11
roadtraffic_10	5,614,700	1,503,700	231	11
roadtraffic_20	11,229,400	3,007,400	231	11
bpic2019_2	3,191,846	503,468	11,973	42
bpic2019_5	7,979,617	1,258,670	11,973	42
bpic2019_10	15,959,230	2,517,340	11,973	42
bpic2018_2	5,028,532	87,618	28,457	41
bpic2018_5	12,571,330	219,045	28,457	41
bpic2018_10	25,142,660	438,090	28,457	51

- *Variants* (*variants.py*): enables the retrieval of variants from the dataframe. This permits variant filtering.
- *Timestamp* (*timestamp.py*): retrieves the timestamp values from a column of the dataframe. This permits three different types of timestamp filtering (events, cases contained, cases intersecting).
- *Endpoints* (*start_end_activities.py*): retrieves the start/end activities from the dataframe. This permits filtering on the start and end activities.
- *Attributes* (*attributes.py*): retrieves the values of a string/numeric attribute. This permits filtering on the values of a string/numeric attribute.
- *Feature selection* (*feature_selection.py*): basilar feature extraction, keeping for every provided numerical attribute the last value per case, and for each provided string attribute its one-hot-encoding.

An example of usage of the PM4Py-GPU library, in which a Parquet log is ingested, and the directly-follows graph is computed, is reported in the following listing.

```
import cudf
from pm4pygpu import format, dfg
df = cudf.read_parquet('receipt.parquet')
df = format.apply(df)
frequency_dfg = dfg.get_frequency_dfg(df)
```

Listing 1.1: Example code of PM4Py-GPU.

4 Assessment

In this section, we want to compare PM4Py-GPU against other libraries/-solutions for process mining to evaluate mainstream operations' execution

time against significant amounts of data. The compared solutions include PM4Py-GPU (described in this paper), PM4Py (CPU single-thread library for process mining in Python; <https://pm4py.fit.fraunhofer.de/>), the PM4Py Distributed Engine (described in the assessment). All the solutions have been run on the same machine (Threadripper 1920X, 128 GB of DDR4 RAM, NVIDIA RTX 2080). The event logs of the assessment include the Road Traffic Fine Management https://data.4tu.nl/articles/dataset/Road_Traffic_Fine_Management_Process/12683249, the BPI Challenge 2019 https://data.4tu.nl/articles/dataset/BPI_Challenge_2019/12715853 and the BPI Challenge 2018 https://data.4tu.nl/articles/dataset/BPI_Challenge_2018/12688355 event logs. The cases of every one of these logs have been replicated 2, 5, and 10 times for the assessment (the variants and activities are unchanged). Moreover, the smallest of these logs (Road Traffic Fine Management log) has also been replicated 20 times. The information about the considered event logs is reported in Table 1. In particular, the suffix (_2, _5, _10) indicates the number of replications of the cases of the log. The results of the different experiments is reported in Table 2. The first experiment is on the importing time (PM4Py vs. PM4Py-GPU; the other two software cannot be directly compared because of more aggressive pre-processing). We can see that PM4Py-GPU is slower than PM4Py in this setting (data in the GPU is stored in a way that facilitates parallelism). The second experiment is on the computation of the directly-follows graph in the four different platforms. Here, PM4Py-GPU is incredibly responsive. The third experiment is on the computation of the variants in the different platforms. Here, PM4Py-GPU and the PM4Py Distributed Engine perform both well (PM4Py-GPU is faster to retrieve the variants in logs with a smaller amount of variants).

Table 2: Comparison between the execution times of different tasks. The configurations analyzed are: P4 (single-core PM4Py), P4G (PM4Py-GPU), P4D (PM4Py Distributed Engine). The tasks analyzed are: importing the event log from a Parquet file, the computation of the DFG and the computation of the variants. For the PM4Py-GPU (computing the DFG and variants), the speedup in comparison to PM4Py is also reported.

Log	Importing		DFG			Variants		
	P4	P4G	P4	P4G	P4D	P4	P4G	P4D
roadtraffic_2	0.166s	1.488s	0.335s	0.094s (3.6x)	0.252s	1.506s	0.029s (51.9x)	0.385s
roadtraffic_5	0.375s	1.691s	0.842s	0.098s (8.6x)	0.329s	3.463s	0.040s (86.6x)	0.903s
roadtraffic_10	0.788s	1.962s	1.564s	0.105s (14.9x)	0.583s	7.908s	0.055s (144x)	1.819s
roadtraffic_20	1.478s	2.495s	3.200s	0.113s (28.3x)	1.048s	17.896s	0.092s (195x)	3.380s
bpic2019_2	0.375s	1.759s	0.980s	0.115s (8.5x)	0.330s	3.444s	0.958s (3.6x)	0.794s
bpic2019_5	0.976s	2.312s	2.423s	0.156s (15.5x)	0.613s	8.821s	0.998s (8.9x)	1.407s
bpic2019_10	1.761s	3.156s	4.570s	0.213s (21.5x)	1.679s	19.958s	1.071s (18.6x)	4.314s
bpic2018_2	0.353s	1.846s	1.562s	0.162s (9.6x)	0.420s	6.066s	5.136s (1.2x)	0.488s
bpic2018_5	0.848s	2.463s	3.681s	0.214s (17.2x)	0.874s	14.286s	5.167s (2.8x)	0.973s
bpic2018_10	1.737s	3.470s	7.536s	0.306s (24.6x)	1.363s	29.728s	5.199s (5.7x)	1.457s

5 Related Work

Process Mining on Big Data Architectures: an integration between process mining techniques and Apache Hadoop has been proposed in [3]. Apache Hadoop does not work in-memory and requires the serialization of every step. Therefore, technologies such as Apache Spark could be used for in-memory process mining³. The drawback of Spark is the additional overhead due to the log distribution step, which limits the performance benefits of the platform. Other platform such as Apache Kafka have been used for processing of streams [5]. Application-tailored engines have also been proposed. The “PM4Py Distributed engine”⁴ has been proposed as a multi-core and multi-node engine tailored for general-purpose process mining with resource awareness. However, in contrast to other distributed engines, it misses any failure-recovery option and therefore is not good for very long lasting computations. The Process Query Language (PQL) is integrated in the Celonis commercial process mining software <https://www.celonis.com/> and provides high throughput for mainstream process mining computations in the cloud.

Data/Process Mining on GPU: many popular data science algorithms have been implemented on top of a GPU [1]. In particular, the training of machine learning models, which involve tensor operations, can have huge speed-ups using the GPU rather than the CPU. In [7] (LSTM neural networks) and [6] (convolutional neural networks), deep learning approaches are used for predictive purposes. Some of the process mining algorithms have been implemented on top of a GPU. In [4], the popular alpha miner algorithm is implemented on top of GPU and compared against the CPU counterpart, showing significant gains. In [2], the discovery of the paths in the log is performed on top of a GPU with a big speedup in the experimental setting.

6 Conclusion

In this paper, we presented PM4Py-GPU, a high-performance library for process mining in Python, which is based on the NVIDIA RAPIDS framework for GPU computations. The experimental results against distributed open-source software (PM4Py Distributed Engine) are very good, and the library seems suited for process mining on a significant amount of data. However, an expensive GPU is needed to make the library work, which could be a drawback for widespread usage. We should also say that the number of process mining functionalities supported by the GPU-based library is limited, hence comparisons against open-source/commercial software supporting a more comprehensive number of features might be unfair.

³ <https://www.pads.rwth-aachen.de/go/id/ezupn/lidx/1>

⁴ <https://www.pads.rwth-aachen.de/go/id/khbht>

Acknowledgements

We thank the Alexander von Humboldt (AvH) Stiftung for supporting our research.

References

1. Cano, A.: A survey on graphic processing unit computing for large-scale data mining. *Wiley Interdiscip. Rev. Data Min. Knowl. Discov.* **8**(1) (2018), <https://doi.org/10.1002/widm.1232>
2. Ferreira, D.R., Santos, R.M.: Parallelization of transition counting for process mining on multi-core cpus and gpus. In: Dumas, M., Fantinato, M. (eds.) *Business Process Management Workshops - BPM 2016 International Workshops*, Rio de Janeiro, Brazil, September 19, 2016, Revised Papers. *Lecture Notes in Business Information Processing*, vol. 281, pp. 36–48 (2016), https://doi.org/10.1007/978-3-319-58457-7_3
3. Hernández, S., van Zelst, S.J., Ezpeleta, J., van der Aalst, W.M.P.: Handling big(ger) logs: Connecting prom 6 to apache hadoop. In: Daniel, F., Zugal, S. (eds.) *Proceedings of the BPM Demo Session 2015 Co-located with the 13th International Conference on Business Process Management (BPM 2015)*, Innsbruck, Austria, September 2, 2015. *CEUR Workshop Proceedings*, vol. 1418, pp. 80–84. CEUR-WS.org (2015), <http://ceur-ws.org/Vol-1418/paper17.pdf>
4. Kundra, D., Juneja, P., Sureka, A.: Vidushi: Parallel implementation of alpha miner algorithm and performance analysis on CPU and GPU architecture. In: Reichert, M., Reijers, H.A. (eds.) *Business Process Management Workshops - BPM 2015, 13th International Workshops*, Innsbruck, Austria, August 31 - September 3, 2015, Revised Papers. *Lecture Notes in Business Information Processing*, vol. 256, pp. 230–241. Springer (2015), https://doi.org/10.1007/978-3-319-42887-1_19
5. Nogueira, A.F., Rela, M.Z.: Monitoring a CI/CD workflow using process mining. *SN Comput. Sci.* **2**(6), 448 (2021), <https://doi.org/10.1007/s42979-021-00830-2>
6. Pasquadibisceglie, V., Appice, A., Castellano, G., Malerba, D.: Using convolutional neural networks for predictive process analytics. In: *International Conference on Process Mining, ICPM 2019, Aachen, Germany, June 24-26, 2019*. pp. 129–136. IEEE (2019), <https://doi.org/10.1109/ICPM.2019.00028>
7. Tax, N., Verenich, I., Rosa, M.L., Dumas, M.: Predictive business process monitoring with LSTM neural networks. In: Dubois, E., Pohl, K. (eds.) *Advanced Information Systems Engineering - 29th International Conference, CAiSE 2017, Essen, Germany, June 12-16, 2017, Proceedings. Lecture Notes in Computer Science*, vol. 10253, pp. 477–492. Springer (2017), https://doi.org/10.1007/978-3-319-59536-8_30