Improving Smart City Simulation Performance with SimEDaPE and Parallelism

Francisco Wallison Rocha[1], **Emilio Francesquini**[2], **Daniel Cordeiro**[1]

[1] Escola de Artes, Ciências e Humanidades – Universidade de São Paulo (USP)
São Paulo – SP – Brasil

{wallison.rocha, daniel.cordeiro}@usp.br

[2]Centro de Matemática, Computação e Cognição – Universidade Federal do ABC (UFABC)
Santo André – SP – Brasil

e.francesquini@ufabc.edu.br

Abstract. *In the context of smart cities, the use of simulators such as the InterSCSimulator to study urban mobility problems is increasingly frequent. However, InterSCimulator has limitations in its performance to test large scenarios such as the city of São Paulo. SimEDaPE was proposed to improve the performance of this simulator, but there are still some performance bottlenecks, such as the mapping step, that could be tackled. This work proposes two parallel implementations to improve the performance of this step. Experimental results show that the best parallel approach using 8 cores results in a* $3\times$ *speedup when compared to the best sequential implementation.*

1. Introduction

With urban growth, urban mobility has become an aspect of great interest on the part of the population and city government [Martins et al. 2020]. Urban mobility is linked to several problems faced in people's daily lives in large cities such as São Paulo, Tokyo, Paris, etc. The quality of life of people living in these big cities is affected due to these problems [Santana et al. 2017]. Solving such problems is not trivial. They require innovative solutions that are usually expensive and difficult to put into practice. In smart cities research, simulations allow us to evaluate such solutions without the need to put them into practice, decreasing associated costs and improving the number o solutions explored.

An example of such simulators is InterSCSimulator [Santana et al. 2017]. InterSCSimulator is an urban traffic simulator. It simulates the traffic of a map of a city or region with travel by pedestrians, cars, subway, and bus systems. However, the InteSCSimulator has some performance limitations for big scenarios, such as the ones presented by a city as big as São Paulo. Simulation Estimation by Data Patterns Exploration (SimEDaPE) was proposed [Rocha et al. 2021] as a possible solution to these performance bottlenecks. Based on the SimPoint technique [Hamerly et al. 2005], SimEDaPE uses the recurrence of patterns in previous simulations to estimate the result of new similar simulations without the need to run them completely. SimEDaPE identifies patterns throughout the simulation to group them together and select a unique representative for each pattern group. Subsequently, these representatives are used in new partial simulations to estimate the remainder of what would be the full simulation.

However, one important phase of the SimEDaPE algorithm suffers from performance issues. The algorithm must compute a temporal mapping between each simulation point (SP) and the other elements of its cluster. This phase takes time proportional to $O(m^2)$, where m is the size of the time series. In a previous work [Rocha et al. 2022], optimizations on the execution environment (using the Cython Python Library [Dalcin et al. 2011]) and the use of a simple parallel implementation using Python's Joblib library [1] improved the execution time of the algorithm, but not its scalability. Consequently, this work improves on our previous work in the following ways. First, it aims to explore the application of the new implementation of the technique in a simulation much closer to reality using a scenario with more than 4 million trips in the city of São Paulo. Furthermore, this work presents two new parallelization schemes as alternatives to the one proposed in our previous work [Rocha et al. 2022]. Experimental results show that, while we were able to improve the scalability of our solution with speedups of $3\times$ on 8 cores, intrinsic and known issues related to parallelism and Python [Mattson et al. 2021] severely limited the performance improvements of our approach.

2. SimEDaPE: Simulation Exploration by Data Patterns Exploration

In the context of smart cities, simulations are of great value so that we can test our solutions before putting them into practice. When performing these tests, researchers and other professionals are interested in metrics to understand the behavior of the solution throughout the simulation. In urban mobility and urban traffic simulations, some metrics of interest are: average speed, vehicle occupancy rate on the streets, the average number of vehicles on the streets, miles traveled, among other metrics.

InterSCSimulator is one of those simulators that allow us to extract such metrics in its output. InterSCSimulator has, however, some limitations in its performance that make simulating a large city such as São Paulo a challenge. Inspired by SimPoint [Hamerly et al. 2005], the Simulation of Estimation by Exploration of Data Patterns (SimEDaPE) was proposed [Rocha et al. 2021] to improve the performance of InterSCSimulator, allowing researchers and other professionals to obtain metrics from simulations of large cities. SimEDaPE uses pattern recurrence obtained from previous simulations to estimate new similar simulations without running them completely. In SimEDaPE, the behavior patterns of vehicle flow on the roads are identified. These flows are represented by time series. The time series is formed by the events of vehicles entering and leaving the roads in a specific time interval. SimEDaPE allows the execution of partial simulations with a maximum error below 30% for metrics like street occupancy rate with an speedup of 1.5x running only 50% of the full simulation [Rocha et al. 2021].

To compare and group the time series, SimEDaPE uses a clustering algorithm similar to k-Means known as k-Shape [Paparrizos and Gravano 2015]. k-Shape gets its name by using the Shape-Based Distance (SBD) metric [Paparrizos and Gravano 2015]. The SBD is a metric used to calculate the similarity of time series. In addition, k-Shape finds a centroid for each group. This centroid is a time series used as a parameter to group the others according to their similarity. We can select the best series in our cluster as the centroid to represent the others. We call this centroid the *simulation point*.

To perform the mapping step, SimEDaPE uses the Dynamic Time Warping

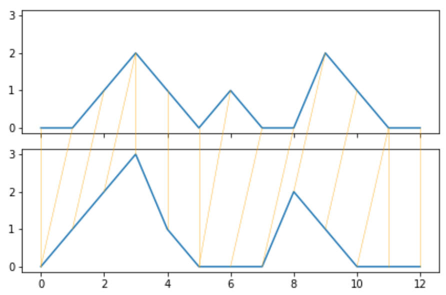
[1] https://joblib.readthedocs.io/en/latest/index.html

(DTW) [Berndt and Clifford 1994] algorithm. DTW provides us with a mapping component, the Warping Path (WP), as an output when applied. The WP is the point-to-point mapping between two-time series as shown in Figure 1a. It is used to perform the mapping between the SP and the other series in its cluster. The current implementation of the DTAIDistance[2] library used by SimEDaPE is written entirely in Python. This mapping is later used to estimate metrics for new simulations. However, the WP calculation for the mapping and warping of simulation points in the time series has the complexity of $O(m^2)$, where m is the number of points in the time series, making it one of the most computationally expensive steps of the whole process.

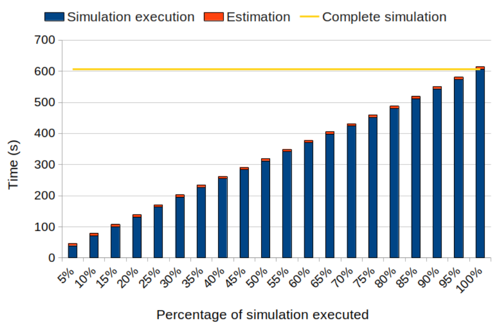
SimEDaPE execution can be divided into four steps: Clustering, Warping Path calculation, Simulation Execution and Estimation. The Clustering and Warping Path calculation steps need to be performed only once, as all future similar simulations may reuse the pre-processed data. For an execution and estimation of a partial simulation 50% of a complete scenario, SimEDaPE took about $9,989s$ of execution time. Figure 1b shows the execution time of two of the SimEDaPE steps, Simulation Execution and Estimation. In blue is the time for partial execution of the scenario (according to the x axis), in orange is the time to estimate the rest of the simulation, in yellow (as a reference) is the time it would take to execute the complete simulation. In addition to the steps shown in Figure 1b, two additional pre-processing steps taking $159.31s$ for the Clustering step and $9511s$ for the Warping Path Calculation step. Using a 50% sample as an example, the Warping Path Calculation time corresponds to 95.21% of the total time spent on the application of SimEDaPE.This can make it a bottleneck for SimEDaPE.

A preliminary work [Rocha et al. 2022] investigated how to accelerate the preprocessing phase. A new implementation of the Warping Path calculation component of the DTAIDistance library using Cython was proposed. Also, a parallel implementation was proposed using Python's Joblib. These modifications allowed a three-fold performance gain compared to the original version when using 8 processes in 8 cores. Thus, this work proposes the implementation of two new data parallelization schemes to compare with the scheme presented in [Rocha et al. 2022]. The scenario used to test this case was the city of São Paulo, with more than 4 million trips.

[2]https://pypi.org/project/dtaidistance/



(a) Example of association of points (WP) from two time series using DTW.



(b) Time spent on each step of the SimEDaPE application.

Figure 1

3. Fast SimEDaPE

Here we show two parallel approaches using Cython and Joblib to improve the mapping step implemented internally in Python using the DTAIDistance library. The two implementations differ in the way of partitioning the data for parallel processing: **Division by Time Series**, and **Balanced Partitioning of Clustering Data** (BPCD). The implementation of **Division by Time Series** consists of a data partition at time series level, where all processes are responsible for calculating in parallel the mappings of the all-time series from a cluster until moving to the next one. The last implementation, **BPCD**, partitions data in a balanced way, where each process receives a similar amount to the others. However, each process must keep track of which cluster each time series belongs, because the warping path needs to be calculated between the SP and all the series of its cluster. Therefore, a task distribution algorithm was implemented to determine the clusters or cluster segments that would be processed by each process.

In Algorithm 1, the process of partitioning the series between the processes is shown. The algorithm takes as inputs the total number of time series T, the number of processes NP, and a vector C with the number of series of each cluster. Each position of C corresponds to the number of time series in the cluster (*e.g.*, s_0 is the number of series in cluster 0). In the algorithm, S corresponds to the optimal partition size. CI corresponds to the cluster start or end index when it doesn't fit entirely inside a process. CS corresponds to the start cluster the process will consider adding to its list. Algorithm 1 also has some helper functions not shown here. The len function returns the size of a list. The $comp$ function determines if the process has already reached the optimal partition size. The $fullAlloc$ function checks if the cluster has been completely allocated between processes or if there are still parts to be allocated. The acc function returns the cumulative of the process p, that is, how many series of clusters were allocated to p. The append function adds an element to a list. At the end of the whole process, Algorithm 1 returns a vector X with NP vectors of n triples in the format (c, i, e). Where c is the cluster number, i is the segment's start index, and e is the end of the segment that process p will be responsible for processing.

4. Experimental Results

To determine the performance of each of the parallel approaches compared to the best sequential implementation, we performed four experiments to calculate the warping path (mapping). This experiment was done using a dataset with $445,501$ time series of size $3,403$ data points, distributed into 64 clusters. This dataset was obtained from a simulation with more than 4 million trips. The machine used to run the experiment has an Intel Core i7-9700K CPU 3.60 GHz octa-core processor with 64 GB RAM, and for parallel implementations 8 cores were used.

Table 1 shows the execution time for the sequential implementation in Python using DTAIDistance with Cython. The table also shows the time for the 3 (two new) parallel implementations using Python's Joblib library. The first is the [Rocha et al. 2022] approach. The second is at the time series level, and each cluster is responsible for calculating a single time series and then proceeds to the next available one. The last one is an implementation where we balance the time series between the processes without losing the reference of the cluster of each one. In addition to the time, the speedup of the parallel implementations compared to the sequential implementation is also shown.

Algorithm 1 Algorithm for partitioning data

```
Input: $T, NP, C \leftarrow [s_0, s_1, ..., s_n]$
Output: $X$                                          ▷ Partition result with segmented cluster list by process
  $S \leftarrow T/NP$
  $CI \leftarrow 0$
  $CS \leftarrow 0$
  for $p = 0$ to $NP$ do
      for $c = CS$ to $len(C)$ do
          if $comp(p)$ then
              $CS \leftarrow \begin{cases} c & \text{if } fullAlloc(C, c-1) \\ c-1 & \text{otherwise} \end{cases}$
              break
          end if
          $CZ \leftarrow C(c) - CI$
          if $CI \neq 0$ then
              $AUX \leftarrow CI$
              $CI \leftarrow \begin{cases} 0 & \text{if } acc(p) + CZ \leq S \\ CZ - ((acc(p) + CZ) - S) & \text{otherwise} \end{cases})$
              $X[p] \leftarrow append(\begin{cases} (c, CI, C(c)) & \text{if } acc(p) + CZ \leq S \\ (c, AUX, AUX + CI) & \text{otherwise} \end{cases})$
          else
              $CI \leftarrow \begin{cases} 0 & \text{if } acc(p) + C(c) \leq S \\ CZ - ((acc(p) + CZ) - S) & \text{otherwise} \end{cases})$
              $X[p] \leftarrow append(\begin{cases} (c, 0, C(c)) & \text{if } acc(p) + C(c) \leq S \\ (c, 0, CI) & \text{otherwise} \end{cases})$
          end if
      end for
  end for
```

Table 1. Speedup compared to sequential implementation

Approach	Sequential	[Rocha et al. 2022]	Division by Time Series	BPCD
Time (s)	*23,151s*	*7,665s*	*22,697s*	*7,576s*
Speedup	-	*3x*	*1x*	*3x*

In Table 1, we can see that for a large scenario like the city of São Paulo with more than 4 million trips, the implementation per cluster (cluster-level) [Rocha et al. 2022] achieved a speedup of $3\times$. The approach based on the time series level was not a good alternative, with the speedup remaining at $1\times$, that is, without any improvement. No further improvement on the performance was perceived when per-process load balancing was used. However, it remained at $3\times$. The cause of this limitation in speedup can be explained by intrinsic issues faced when doing parallelism with Python. To avoid access and concurrency issues, Python uses a Global Interpreter Lock (GIL) to ensure that only one thread progresses at any point in time. This blocks any attempt to take advantage of the performance benefits of running multiple threads in parallel on a multi-core CPU [Mattson et al. 2021]. To workaround this issue, we use Python's Joblib to split the work between processes (not threads) on a multi-core CPU. However, to communicate data between the processes Joblib serializes and copies the data to each new process, hindering performance. This is the same limitation that made the time series level approach to perform poorly. Moreover, it also explains why we did not obtain better speedups the case of division by processes.

5. Conclusions

This work presented two new parallel implementations using the Joblib library together with Cython to improve the performance of the mapping step of the SimEDaPE technique. They were designed to try to improve the speedup presented by the parallel implementation of our previous work [Rocha et al. 2022]. The calculation step of the WP still is the major bottleneck for processing large simulations, and further research is required.

Experimental results for these new implementations have shown that Python's limitations on multithreading and multiprocessing need to be overcome to improve the performance of our approach. In particular, the implementation that split the tasks at the time series level, has shown worse results than the alternatives, without any gain when compared to the sequential implementation. In the implementation with load balancing, the speedup remained at $3\times$. We believe that this may have been due to the serialization and data copying process done by Joblib. Therefore, as future work, we intend to use alternative implementations in languages with robust multithreading support.

Acknowledgments:
This research is part of the INCT of the Future Internet for Smart Cities funded by CNPq proc. 465446/2014-0, Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Finance Code 001, FAPESP proc. 14/50937-1, and FAPESP proc. 15/24485-9.

References

Berndt, D. J. and Clifford, J. (1994). Using dynamic time warping to find patterns in time series. In *Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining*, AAAIWS'94, page 359–370, Seattle, WA. AAAI Press.

Dalcin, L., Bradshaw, R., Smith, K., Citro, C., Behnel, S., and Seljebotn, D. (2011). Cython: The best of both worlds. *Computing in Science & Engineering*, 13(02):31–39.

Hamerly, G., Perelman, E., Lau, J., and Calder, B. (2005). Simpoint 3.0: Faster and more flexible program phase analysis. *Journal of Instruction Level Parallelism*, 7(4):1–28.

Martins, T. G., Lago, N., de Souza, H. A., Santana, E. F. Z., Telea, A., and Kon, F. (2020). Visualizing the structure of urban mobility with bundling: A case study of the city of São Paulo. In *Anais do IV Workshop de Computação Urbana*, pages 178–191. SBC.

Mattson, T. G., Anderson, T. A., and Georgakoudis, G. (2021). Pyomp: Multithreaded parallel programming in python. *Computing in Science Engineering*, 23(6):77–80.

Paparrizos, J. and Gravano, L. (2015). K-shape: Efficient and accurate clustering of time series. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, SIGMOD '15, page 1855–1870, New York, NY, USA. Association for Computing Machinery.

Rocha, F. W., Francesquini, E., and Cordeiro, D. (2022). Fast SimEDaPE: Simulation estimation by data patterns exploration. *13ª Escola Regional de Alto Desempenho de São Paulo*. To publish.

Rocha, F. W., Fukuda, J. C., Francesquini, E., and Cordeiro, D. (2021). Accelerating smart city simulations. *Latin America High Performance Computing Conference*. To publish.

Santana, E. F. Z., Lago, N., Kon, F., and Milojicic, D. S. (2017). InterSCSimulator: Large-scale traffic simulation in smart cities using erlang. In *International Workshop on Multi-Agent Systems and Agent-Based Simulation*, pages 211–227. Springer.