# A GPU-based algorithm for environmental data filtering

P. De Luca[1][0000−0001−7031−920X] and A. Galletti[2][0000−0002−5208−6219]
L. Marcellino[2][0000−0003−2319−8008]

[1] International PhD Programme / UNESCO Chair "Environment, Resources and Sustainable Development", Department of Science and Technology, Parthenope University of Naples, Centro Direzionale, Isola C4, (80143) Naples, Italy,
[2] Department of Science and Technology, Parthenope University of Naples, Centro Direzionale, Isola C4, (80143) Naples, Italy
{pasquale.deluca,ardelio.galletti,livia.marcellino}@uniparthenope.it

**Abstract.** Nowadays, the Machine Learning (ML) approach is needful to many research fields. Among these, the Environmental Science (ES) which involves a large amount of data to be processed and collected. On the other hand, in order to provide a reliable output, those data information must be assimilated. Since this process requires a large execution time when the input dataset is very huge, here we propose a parallel GPU algorithm based on a curve fitting method, to filter the starting dataset, by exploiting the computational power of the CUDA tool. The innovative aspect of the proposed procedure can be used in several application fields. Our experiments show the achieved results in terms of performance.

**Keywords:** Machine Learning · Curve fitting · Filtering · GPU parallel algorithm · HPC

## 1 Introduction

In recent years, the Machine Learning approach is becoming very helpful for environmental data analysis, e.g. weather prediction, air pollution quality analysis or earthquakes. The large amount of available data can be successfully used to define, classify, monitor and predict the ecosystem conditions in which we live [1]. However, data are known to have errors of a random nature because, very often the acquisition tools can provide false measurements or with missing data. This implies that mathematical models used to analyze them can provide unacceptable results.
To overcome this problem Data Assimilation (DA) plays a key role in ML methods. DA is a standard practice, which combines mathematical models and detected measures, and it is heavily employed in numerical weather prediction [2]. Lately its application is becoming widespread in many other areas of climate, atmosphere, ocean, and environment modeling; i.e. in all circumstances where one intends to estimate the state of a large dynamical system based on limited information. In this context, here we deal with a smoothing method based

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08760-8_4

on local least-squares polynomial approximation to filter initial data, known as Savitzky-Golay (SG) filter [3]. This filter provides to fit a polynomial to a set of sampled values, in order to smooth noisy data while maintaining the shape and height of peaks. The innovative aspect of the proposed procedure can be used in several applications with satisfactory results: for example for image analysis, signals electrocardiograms processing and for environmental measurements filtering [4–6]. Nevertheless, the large amount of data to be processed requires several waiting hours and this represents a problem that must be solved. High-Performance Computing (HPC) offers a powerful tool to overcome this issue through parallel strategies, suitably designed to be applied in several application fields [7–9].

In this work, we present a novel parallel algorithm, for Graphics Processing Units (GPUs) environment, appropriately designed to efficiently perform the SG filter. Our implementation exploits the computational power of the Compute Unified Device Architecture (CUDA) framework [10], together with the cuBLAS and cuSOLVER libraries, in order to achieve an appreciable gain of performance in dealing advanced mathematical algebraic operations.

Then the rest of the paper is organized as follows. Section 2 recalls the mathematical model related to the SG filter. In Section 3, the GPU-CUDA parallel approach and the related algorithm are described. The experiments discussed in Section 4 confirm the efficiency of the proposed implementation in terms of performance. Finally, conclusions in Section 5 close the paper.

## 2 Numerical model details

In this section we recall some mathematical preliminaries about the model implemented. This allows us to describe a pseudo-algorithm which is the basis of the GPU-parallel implementation, described in next section, we propose. The discussion follows scheme and main notations presented in [11]. In the following, we limit the discussion to the basic information to design the GPU-parallel implementation and recommend the reader to see papers [3, 12, 13] for further details. The model we consider is the Savitky-Golay filter, which consists in fact in applying a least square fitting procedure to the entries values lying in moving windows of the input signal.

Let us begin by denoting by: $x[n], (n = \ldots, -2, -1, 0, -1, 2, \ldots)$ the entries of the input, and set two nonnegative integer values $ML$ and $MR$. Then, for all value $x[i]$ to be filtered, let consider the window:

$$\mathbf{x}_i = \big(x[i - ML], \ldots, x[i], \ldots, x[i + MR]\big) \tag{1}$$

centered at $x[i]$ and including $ML$ values "to the left" and $MR$ values "to the right". $\mathbf{x}_i$, that is: In the SG filter, to get the filtered value $y[i]$, firstly we find the polynomial:

$$p_N(n) = \sum_{k=0}^{N} a_k n^k \quad -ML \leq n \leq MR \tag{2}$$

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08760-8_4

of degree $N$ that minimizes, in the least square sense, its distance from values in $\mathbf{x}_i$, i.e. the quantity: $\varepsilon_N = \sum_{k=-ML}^{MR} \big(p_N(k) - x[i+k]\big)^2$. Then we set $y[i] = p_N(0) = a_0$., as the value the polynomial takes at $n = 0$. We observe that $p_N$ is the least square polynomial approximating points $(i, x[i])$, (for $i = -ML, \ldots, MR$) and it can be proved that its coefficients $a_k$ solve the normal equations linear system:

$$\mathcal{A}^T \mathcal{A} a = \mathcal{A}^T \mathbf{x}_i^T, \quad \text{where} \quad \mathcal{A} = \big( \ i^j \ \big)_{i=-ML,\ldots,MR}^{j=0,\ldots,N}. \tag{3}$$

It follows that $a_0$ and all other coefficients of $p_N$ depend on (are linear combination of) $\mathbf{x}_i$. However, by rearranging (3) we get: $a = (\mathcal{A}^T\mathcal{A})^{-1}\mathcal{A}^T\mathbf{x}_i^T$ the 0-th row of the pseudo-inverse matrix:

$$H = (\mathcal{A}^T\mathcal{A})^{-1}\mathcal{A}^T, \tag{4}$$

whose entries do not depend on $\mathbf{x}_i$. In other words each filtered value $y[i]$ is a linear combination of values in $\mathbf{x}_i$ with coefficients in the 0-th row of $H$, that can be pre-computed once, independently from the filtered value we need, i.e.:

$$y[i] = \sum_{k=-ML}^{MR} h_{0,k} \cdot x[i+k] \tag{5}$$

Previous discussion allows us to introduce the pseudo-algorithm 1, which summarizes the main steps needed to solve the numerical problem.

**Algorithm 1** Sequential pseudo-algorithm

```
Input: x, ML, MR, N.   Output: y
build A                % as in (3)
build H                % as in (4)
extract H_0            % the 0-th row of H
for i ∈ ℤ do
    extract x_i        % from the input x
    compute y[i]       % as in (5)
end for
```

## 3 Parallel approach and GPU algorithm

Observing the significant results achieved in [14–16], due to large amount of produced data from scientific community, we have chosen to develop a GPU-based parallel implementation, equipped by a suitable Domain Decomposition (DD) with overlapping, already proposed in [17], for most modern GPU architecture. From now on, in this section, to describe the parallel algorithm, we set: $ML = MR = M$ and assume the input signal $x$ to have finite size $s$, i.e., there is no likelihood of confusion by setting: $x = \big(x[0], x[1], \ldots, x[s-1]\big)$. We observe that to apply the algorithm to all moving windows, included the edge ones, we also

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08760-8_4

need to increase the size of the input with a *zero-padding* procedure, which consists of extending the original input signal with artificial $M$ zeros at the left and $M$ zeros at the right boundaries, as follows:

$$x^M = (0, \ldots, 0, x[0], \ldots, x[s-1], 0, \ldots, 0). \tag{6}$$

The overall parallel scheme is shown in the pseudo-algorithm 2.

**Algorithm 2** Parallel pseudo-algorithm

`Input:` $x$, $M$ $N$ . `Output:` $y$

1: build $x^M$ as in (6) % *zero-padding of* $\mathbf{x}$
% `step 1: Pseudo-inverse building`
2: build $\mathcal{A}$, as in (3) by using the cuSOLVER routine
3: build $H$, as in (4) by using the cuBLAS routine
4: extract $H_0$ % *the* $0-th$ *row of* $H$
% `step 2: Domain Decomposition with overlapping`
5: **for each** $t_i$ **do**
6: $\mathcal{S}_i \leftarrow \mathbf{x}_i$
7: **end for**
% `step 3: Computation of final output`
8: compute $y$ as in (7), by using the cuBLAS routine

The main steps of the pseudo-algorithm 2 are described below:

STEP 1 - *Pseudo-inverse building.*
This step is based on the $QR$ factorization of a matrix $A$, which is suitably efficient for computing the pseudo-inverse $H$ when $A$ is symmetric. To this aim, the algorithm exploits the potential of the cuSOLVER and cuBLAS libraries [18, 19] for numerical linear algebra operations on GPU. In particular, we firstly build matrix $\mathcal{A}$ by using the `cusolverDnDgeqrf` cuSOLVER routine, then to build of $H$, we observe that it holds: $H = R^{-1}Q^T$, where $Q$ and $R$ come from the $QR$ factorization of $\mathcal{A}$. Then, we use the `cublasDgetriBatched` cuBLAS routine to compute $R^{-1}$ and the `cublasDgemm` CUBLAS routine, to get the matrix-matrix multiplication $H$.

STEP 2 - *Domain Decomposition with overlapping.*
Here, we decompose the problem by building a matrix: $\mathcal{S} = [\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{s-1}]^T$ whose rows are the $\mathbf{x}_i$ defined in (1). According to SIMT paradigm, which is used by the CUDA configuration, we use: $s$ threads, $t_0, t_1, \ldots, t_{s-1}$, and each of them copies the $i-th$ moving window from $x^M$ to $i-th$ row of $\mathcal{S}$, in a fully parallel way. To this aim, we implemented a CUDA kernel which takes advantage of the large number of processing units of the GPU environment.

STEP 3 - *Computation of final output.*
The final output $y$ can be regarded as the matrix-vector product

$$y = S \cdot H_0^T. \tag{7}$$

This operation is performed by using the `cublasDgemv` CUBLAS routine.

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08760-8_4

## 4 Performance analysis

In this section, some experimental results show and confirm the efficiency of proposed software. Following, the hardware specifications where the GPU algorithm has been implemented and tested, are listed: 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. Thanks to CUDA framework, Algorithm 2 exploits the overall GPU's parallel computational power. The evaluation of a parallel algorithm is covered by different metrics, and here: we preliminarily show a execution time comparison in order to highlight the gain of performance obtained, then we focus on the spent time for each computational expensive operation listed in pseudo-algorithm 2. Finally, last subsection shows a example output of the parallel algorithm applied to a set of environmental data.

**Execution time comparison.** Table 1 exhibits an execution time comparison among the GPU implementation and the native function *sgolayfilt* in MATLAB, by varying the input data sample dimension. We underline the *sgolayfilt* function runs in a parallel way by using 4 cores. Input data come from the environmental dataset related to the last London Atmospheric Emissions Inventory (LAEI) for year 2019. The area covered by the LAEI includes Greater London, as well as areas outside Greater London up to the M25 motorway (see [20]). These emissions have been used to estimate ground level concentrations of key pollutants $NO_x, NO_2, PM_{10}$ and $PM_{2.5}$ across Greater London for year 2019, using an atmospheric dispersion model. In this test we used $M = 2$ and $N = 3$. The CUDA configuration is static and set to $512 \times 1024$ block per threads. We highlight a significant time reduction of GPU algorithm with respect to multi-core execution. The achieved speed-up is closely linked to accelerated operations computed in parallel way by giving a strong impact of time reduction. Despite the accomplished good performance, we must stop to increase the input data sample due to small memory size adopted by the CPU. Conversely, the large dimension and accurate management of GPU memory together, allows us to introduce a great increment of the input data sample in order to process big data sample.

**Table 1.** Execution times: MATLAB vs. GPU.

| | Time (ms) | |
|---|---|---|
| $s$ | Multi-core | GPU |
| $3.4 \times 10^1$ | 5.92 | 2.96 |
| $4.0 \times 10^3$ | 6.59 | 3.54 |
| $4.6 \times 10^4$ | 7.01 | 3.45 |
| $7.3 \times 10^4$ | 7.50 | 3.50 |
| $3.9 \times 10^5$ | 7.90 | 3.51 |

**CUDA kernels analysis.** An additional performance analysis acts as a support for previous experimental result. More precisely, for each single kernel by varying the input sample dimension, we show the time consumption for each CUDA kernel which computes the operations of most expensive procedure of pseudo-

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08760-8_4

algorithm 2. Table 2 shows time values for each CUDA kernel by positively confirming the efficiency of the proposed software. Specifically, a good work-load balancing among CUDA threads is performed. Transfer and copy times are avoided to underline the gain of performance for each kernel.

**Table 2.** Time execution analysis for each CUDA kernel in milliseconds.

| | Execution time (ms) | | | | |
|---|---|---|---|---|---|
| operation | $3.4 \times 10^1$ | $4.0 \times 10^3$ | $4.6 \times 10^4$ | $7.3 \times 10^4$ | $3.9 \times 10^5$ |
| QR | 0.11 | 0.11 | 0.11 | 0.11 | 0.11 |
| Pseudo-inverse | 0.30 | 0.32 | 0.31 | 0.30 | 0.31 |
| S computation | 0.12 | 0.12 | 0.13 | 0.07 | 0.28 |
| Polynomial evaluation | 0.11 | 0.12 | 0.20 | 0.26 | 0.92 |
| Overall time | **0.68** | **0.69** | **0.78** | **0.84** | **1.73** |

**Environmental data testing.** Last plot, Figure 1, aims to show how the algorithm performs when applied to data of environmental nature. In particular, this qualitative test is referred to the Greater London mean $PM_{10}$ particulate matter arising from the dataset in [20]. Input data is plotted in blue while the SG filtered output is the red line. We recall that results sharply overlap with the ones obtained by means of the MATLAB routine `sgolayfilt`. We remark that, unlike the MATLAB procedure, our implementation is able to manage input data, of very large size, in a reasonable time.

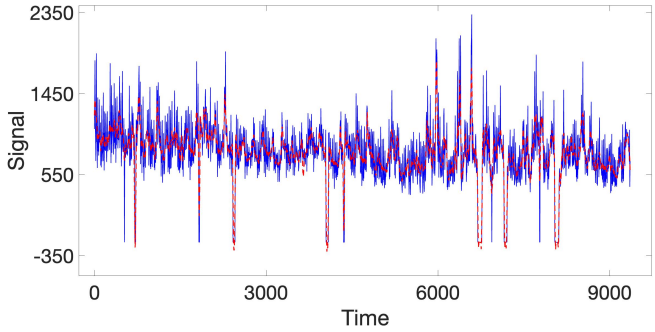


**Fig. 1.** Application of the SG filter to $PM_{10}$. Input signal is reported in blue, filtered output of the Savitzky-Golay filter in red.

## 5 Conclusion

In this work, we presented a novel GPU-parallel algorithm, based on the SG filter, to approximate by a polynomial sampled data values. This procedure can be seen as a pre-processing step to correctly assimilate data within the Machine Learning approach in order to provide a reliable output and without affecting execution times. Our experiments showed very good performance even with real environmental data. Future work could consider a further application of our implementation to the ECG denoising field.

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08760-8_4

## References


1. Kanevski, M. (2009). Machine learning for spatial environmental data: theory, applications, and software. EPFL press.
2. Cuomo, S., Galletti, A., Giunta, G., & Marcellino, L. (2017). Numerical effects of the gaussian recursive filters in solving linear systems in the 3dvar case study. Numerical Mathematics: Theory, Methods and Applications, 10(3), 520-540.
3. Savitzky, A., & Golay, M. J. (1964). Smoothing and differentiation of data by simplified least squares procedures. Analytical chemistry, 36(8), 1627-1639.
4. Liu, Yang & Dang, Bo & Li, Yue & Lin, Hongbo & Ma, Haitao. (2015). Applications of Savitzky-Golay Filter for Seismic Random Noise Reduction. Acta Geophysica. 64. 10.1515/acgeo-2015-0062.
5. Chen, J., Jönsson, P., Tamura, M., Gu, Z., Matsushita, B., & Eklundh, L. (2004). A simple method for reconstructing a high-quality NDVI time-series data set based on the Savitzky–Golay filter. Remote sensing of Environment, 91(3-4), 332-344.
6. Rodrigues, J., Barros, S., & Santos, N. (2021, July). FULMAR: Follow-Up Lightcurves Multitool Assisting Radial velocities. In Posters from the TESS Science Conference II (TSC2 (p. 45).
7. D'Amore, L., Casaburi, D., Galletti, A., Marcellino, L., & Murli, A. (2011). Integration of emerging computer technologies for an efficient image sequences analysis. Integrated Computer-Aided Engineering, 18(4), 365-378.
8. Luca, P. D., Galletti, A., Giunta, G., & Marcellino, L. (2020, June). Accelerated Gaussian convolution in a data assimilation scenario. In International Conference on Computational Science (pp. 199-211). Springer, Cham.
9. De Luca, P., Galletti, A., & Marcellino, L. (2020, July). Parallel solvers comparison for an inverse problem in fractional calculus. In 2020 Proceeding of 9th International Conference on Theory and Practice in Modern Computing (TPMC 2020).
10. https://developer.nvidia.com/cuda-zone
11. Schafer, R. W. (2011). What is a Savitzky-Golay filter?[lecture notes]. IEEE Signal processing magazine, 28(4), 111-117.
12. Schafer, R. W. (2011). What is a Savitzky-Golay filter?[lecture notes]. IEEE Signal processing magazine, 28(4), 111-117.
13. Luo, J., Ying, K., & Bai, J. (2005). Savitzky–Golay smoothing and differentiation filter for even number data. Signal processing, 85(7), 1429-1434.
14. Cuomo, S., De Michele, P., Galletti, A., & Marcellino, L. (2016, June). A GPU parallel implementation of the local principal component analysis overcomplete method for DW image denoising. In 2016 IEEE Symposium on Computers and Communication (ISCC) (pp. 26-31). IEEE.
15. Cuomo, S., De Michele, P., Galletti, A., & Marcellino, L. (2016, March). A GPU-parallel algorithm for ECG signal denoising based on the NLM method. In 2016 30th International Conference on Advanced Information Networking and Applications Workshops (WAINA) (pp. 35-39). IEEE.
16. De Luca, P., Galletti, A., Giunta, G., & Marcellino, L. (2021). Recursive filter based GPU algorithms in a Data Assimilation scenario. Journal of Computational Science, 53, 101339.
17. De Luca, P., Galletti, A., & Marcellino, L. (2019, November). A Gaussian recursive filter parallel implementation with overlapping. In 2019 15th international conference on signal-image technology & internet-based systems (SITIS) (pp. 641-648).
18. https://docs.nvidia.com/cuda/cusolver/index.html
19. https://docs.nvidia.com/cuda/cublas/index.html
20. https://data.london.gov.uk/


ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08760-8_4