PROCEEDINGS OF SCIENCE

Advantages of GPU-accelerated approach for solving the Parker equation in the heliosphere

S. Della Torre,[a] **G. Cavallotto,**[a,b] **D. Besozzi,**[c] **M. Gervasi,**[a,b] **G. La Vacca,**[a,b] **M. Nobile**[d] **and P.G. Rancoita**[1]

[a] *National Institute for Nuclear Physics (INFN), Milano-Bicocca Unit, Piazza della Scienza, 3 - 20126 Milano (Italy)*

[b] *Department of Physics "Giuseppe Occhialini", University of Milano-Bicocca, Piazza della Scienza, 3 - 20126 Milano (Italy)*

[c] *Department of Informatics, Systems and Communication, University of Milano-Bicocca, Viale Sarca, 336 - 20126, Milano (Italy)*

[d] *Department of Environmental Sciences, Informatics and Statistics (DAIS), Ca' Foscari University Dorsoduro 3246, 30123 Venice (Italy)*

E-mail: stefano.dellatorre@mib.infn.it, giovanni.cavallotto@unimib.it

The increasing of experimental observations' accuracy and model complexity of the heliospheric cosmic rays modulation requires the development of a new class of numerical solvers. In this work, we present a GPU-accelerated code for solving the Parker propagation equation in the heliosphere using a stochastic differential equation (SDE) approach. The presented method uses the CUDA programming language developed for the NVIDIA GPUs. Our approach achieves speedup of the orders of ~10-40, depending on the number of *quasi-particle* simulated, compared to the previous CPU implementation. This allows us to efficiently solve the transport equation for the modulated spectra of charged particles in the heliosphere, opening the field for deeper studies and make the realized simulations available for general purpose studies. We demonstrate the accuracy and efficiency of our method through numerical experiments on a realistic model of the heliosphere.

38th International Cosmic Ray Conference (ICRC2023)
26 July - 3 August, 2023
Nagoya, Japan

PoS(ICRC2023)1290

1. Introduction

Cosmic rays (CR) are ionized atoms that permeate the interplanetary medium, originating from supernova explosions or stellar eruptive events. The proper characterization of the CR flux allows one to the two-fold achievement of exploring the physics of particle interaction in the interstellar medium and asses the space radiation environment that challenges the electronic devices of interplanetary probes. When entering into the heliosphere, the region around the Sun dominated by the solar magnetic field and solar wind, galactic cosmic rays (GCR) experienced the so called *solar modulation*, a reduction of CR flux intensity at ~GeV/n energies. A review of observations and solar modulation models in the heliosphere is reported in Rankin et al. [1] and Engelbrecht et al. [2].

2. The physical model

All relevant physical processes involved in the CR propagation are described by the Parker Transport Equation (PTE), named after Eugene Parker who first proposed it in the 1960s (see, e.g. Parker [3], Boschini et al. [4] and references therein):

$$\frac{\partial U}{\partial t} = \frac{\partial}{\partial x_i}\left(K^S_{ij}\frac{\partial \mathrm{U}}{\partial x_j}\right) + \frac{1}{3}\frac{\partial V_{\mathrm{sw},i}}{\partial x_i}\frac{\partial}{\partial T}\left(\alpha_{\mathrm{rel}} T \mathrm{U}\right) - \frac{\partial}{\partial x_i}\left[\left(V_{\mathrm{sw},i} + v_{d,i}\right)\mathrm{U}\right], \tag{1}$$

where U is the number density of GCR particles per unit of kinetic energy T (GeV/nucleon), t is time, $V_{\mathrm{sw},i}$ is the solar wind (SW) velocity along the axis x_i, K^S_{ij} is the symmetric part of the diffusion tensor, $v_{d,i}$ is the particle magnetic drift velocity (related to the anti-symmetric part of the diffusion tensor), and $\alpha_{\mathrm{rel}} = \frac{T+2m_r c^2}{T+m_r c^2}$, with m_r the particle rest mass per nucleon in units of GeV/nucleon.

The PTE is a Fokker-Planck type equation that can be solved using both an approach forward-in-time as well as backward-in-time (in the latter case it is usually named as Kolmogorov equation, see equation 1.7.15 in Klöden and Platen [5]). Both Fokker-Planck and Kolmogorov differential equations are equivalent to a set of stochastic differential equations (SDE), as shown in sections 4.3.2–4.3.5 of Gardiner [6] and Appendix A.13.1 of Kroese et al. [7], following the Ito's formula.

To obtain the SDEs equivalent to Eq. (1), the latter should be rearranged to match the following formulation (see e.g. Equation 13 of Zhang [8], Equation A2 of Strauss et al. [9], Equation 14 of Kopp et al. [10]):

$$\frac{\partial Q}{\partial s} = \sum_i A_{B,i}(s,y)\frac{\partial Q}{\partial y_i} + \frac{1}{2}\sum_{i,j} C_{B,ij}(s,y)\frac{\partial^2 Q}{\partial y_i \partial y_j} - L_B Q + S \tag{2}$$

where $A_{B,i}$ is the advective vector, $C_{B,ij}$ is the diffusion tensor, L_B describes energy loss and S stances for source of particles. In this formulation, $\partial s > 0$ represents the backward time evolution of the propagation. The set of SDEs corresponding to the Eq. (2) can be generally expressed as:

$$dy_i(s) = A_{B,i}ds + B_{B,i,j}dW_j(s), \tag{3}$$

where $\tilde{C} = \tilde{B}\tilde{B}^T$, $d\vec{W}$ represents the increments of a *standard Wiener process*, which can be described as an independent random variable of the form $\sqrt{ds}N(0,1)$, with $N(0,1)$ denoting a

normally distributed random variable with zero mean and unit variance (see e.g., Appendix A of Zhang [8] and section 2 of Higham [11]). To numerically integrate the SDEs the Euler-Maruyana scheme (see, e.g., Kroese et al. [7], section 5.6.1) is the most simple and commonly used, combined with the Ito rule (see discussion in Bobik et al. [12] and reference therein). Propagating backward in time allows one to simulate only the *quasi-particles* actually reaching the restricted subset of phase space points of interest, like the Earth orbit or the spacecraft trajectory (as shown by Strauss and Effenberger [13]).

3. Numerical algorithm and GPU parallelization

Using SDEs, the solution of PTE could be evaluated numerically using a Monte Carlo algorithm. In this approach, *quasi-particle objects* evolves in space position and energy according to SDEs, since the evolution of each object must be independent of the others, the ensemble of the final states of all the computed objects allows to compute the modulated spectra. This method could be easily parallelized on an HPC system, e.g., on GPU architecture. We use the CUDA-C language[1,2,3] (for a complete guide see J. Cheng [14], Sanders and Kandrot [15] handbooks), which provides optimized interactions and low-level code architecture for the NVIDIA-GPUs.

The diagram of the cuda code algorithm can be seen in figure 1. We assign the evolution of each *quasi-particle* to a different GPU thread (i.e. the minimal computing unit of the GPU), following the single instruction, multiple data (SIMD) paradigm [16] (optimal for GPU architecture). The first step of the program is the initialization of the hyper-parameters of the simulation: the number of *quasi-particle* to be simulated ($\mathcal{N}$), the heliospheric model parameters, the particle species characteristics, and the initial positions. The latter are copied into the global memory of the GPU, the other variables, instead are copied into the constant memory, which is a special memory present in the NVIDIA GPUs (see, e.g. Chapter 4,5 of J. Cheng [14]) where the data stored are protected and remain constant over the course of kernel execution (i.e. read-only memory). Moreover, it has short latency, high bandwidth, and, using a broadcasting implemented method, memory reading is no slower than reading from a register. These features make it perfect to store the time-independent and thread-common parameters of the simulation.

The second step in the algorithm is the configuration of the GPU kernel execution and allocation of the hardware resources, to achieve the maximum device usage. The *quasi-particles* objects are divided into subsets each one referring to the different initial positions and heliospheric parameters. To maximize the probability of broadcast data from device memory, we ensure that each warp evolves *quasi-particles* object of the same subset. Furthermore, we adjust $\mathcal{N}$ to be a multiple of the size of a *warp* (i.e. 32 for NVIDIA GPUs with Compute Capability version 8.0), maximizing the GPU occupancy.

Then we generate the random number sequences needed for the stochastic Wiener process term in Eq. (3), using the *Philox4_32_10* generator [17] aPseudo-Random Number Generator (PRNG) provided by the device API of the cuRAND library to generate per-thread random numbers within *quasi-particle* propagation kernel. The PRNG is initialized by using the same seed for each kernel

[1] https://developer.nvidia.com/cuda-toolkit
[2] https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html
[3] https://docs.nvidia.com/cuda/cuda-c-best-practices-guide/index.html

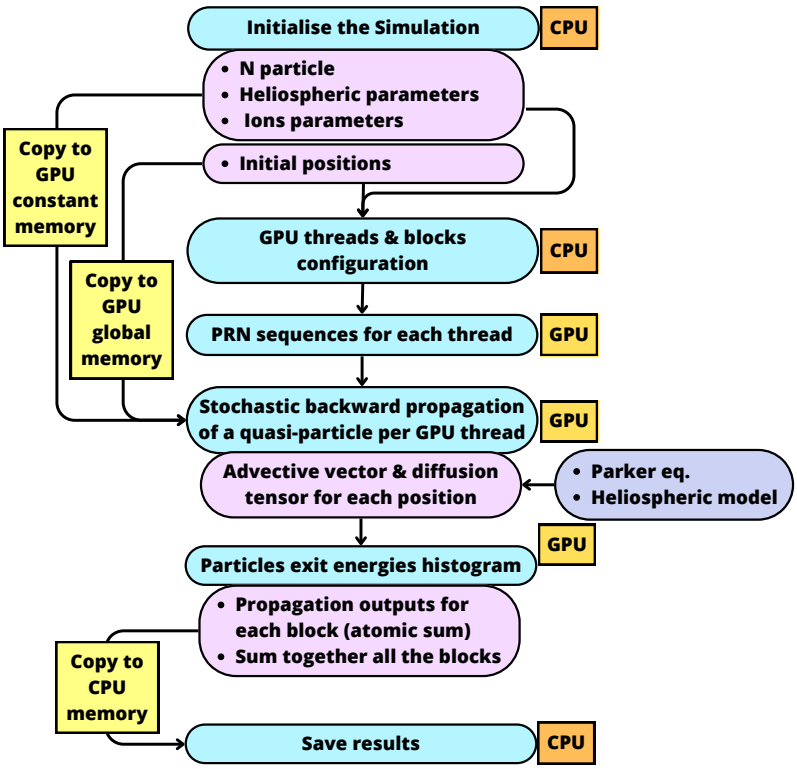

Figure 1: Scheme of the GPU algorithm architecture implemented for the HelMod-cuda code. In sky blue are represented the macroscopic steps of the code, in purple the output of the respective step, in yellow the memory interaction between the host CPU and device GPU and in orange which processor executes the respective computation. The arrows here are indicated the links between various inputs and outputs of the numerical algorithm.

call on the same GPU but specifying a different sequence identifier related to the id of the thread (see, e.g., PRNG implementation in Romero et al. [18] and Askar et al. [19] for the PRNGs choice).

At this point, all is set to execute the stochastic *quasi-particle* propagation for all the treads. The device computes the particle heliospheric location and the corresponding $A_{B,i}$, $B_{B,i,j}$ in the Eq. (3), at each integration step until a heliospheric boundary is reached. Eventually, the exit modulated *quasi-particle* objects are collected in a partial histogram for each block, with an atomic function to avoid memory conflicts. Then, the partial histograms are merged and the final results are copied to the host memory.

In the case of multiple GPUs, the algorithm assigns to each GPU a subset of energies to be simulated. With this approach, each GPU-thread can proceed independently and it is not needed to share memory across devices even in the histogram building (only the final histograms are transferred to the host without the need for further merging).

4. Performances

In this section, we compare the simulation execution time of CPU-only code and CPU+GPU one. As a study case, we refer to the HelMod-4 modelBobik et al. [12], Boschini et al. [20] that was designed to solve PTE in heliosphere tuning its parameters on the state-of-art of GCR measurements. The CPU-only code was executed on a server with two CPU Intel(R) Xeon(R) 2.10GHz. The GPU-cuda code, instead, was tested on the same server with both two GPU NVIDIA A30[4] and NVIDIA A100[5] GPUs boards.Technical information on used GPU boards A30 and A100 GPUs is shown in table 1.

	A30	**A100**
Architecture	Ampere	Ampere
Boost clock speed	1440 MHz	1410 MHz
Core clock speed	930 MHz	1095 MHz
Peak FP32 performance	10.32 TFLOPS	19.49 TFLOPS
Maximum RAM	24 GB	80 GB
Memory bandwidth	933.1 GB/s	2039 GB/s

Table 1: NVIDIA's GPU boards comparison on the main hardware features of interest for the computations illustrated in this paper. All the quantities reported correspond to the maximum of the hardware available resources. Each code uses a fraction between $[0, 1]$ of them.

Moreover, the CPU+GPU code was compiled using `fast_math` library, which converts all the mathematical functions into device intrinsic functions, resulting in a reduction of the numerical accuracy (negligible for our purposes) and in a huge run time speed-up.

We simulated from 10^2 to 10^5 *quasi-particle* objects per energy bin and, for each value of $\mathcal{N}$ and energy bin, we computed the execution time as the mean value for ~ 200 code instances. An example of the execution times of the different code versions and hardware is shown in Fig. 2 as a function of $\mathcal{N}$. Due to the linearity of the algorithm, the execution time of CPU-only code scales as a linear function of $\mathcal{N}$ in logarithmic scale. On the other hand, the execution time of CPU+GPU code shows two different regimes: up to $\mathcal{N} \sim 10^4$ it scales as a power law with spectral index ~ 0.1, then the spectral index becomes steeper. This behavior comes from the fact that at $\mathcal{N} \sim 10^4$ the program saturated the GPU resources, so no more parallelization can be exploited and part of the operations are executed sequentially as it is for CPU executions. At $\mathcal{N} \sim 10^5$ we also register sporadic *illegal memory errors*. One can note also that the A100 curve is the same as A30, except for the shift of 1 $\mathcal{N}$ bin. This can be explained by the fact that the two GPUs have nearly the same processor clock. Therefore, the code is executed at the same speed until the GPU's memory or parallelization available is saturated, which occurs at higher $\mathcal{N}$ for the A100 boards.

The observed performances represent a huge improvement with respect to the initial code, especially with $\mathcal{N} \sim 10^4$ when it reaches a factor of ~ 40. Furthermore, the processor power consumption are: $165W$ for the NVIDIA A30, $300W$ for the NVIDIA A100 and $80W$ for the Intel Xeon E5520. Considering the execution time of the two codes, for $\mathcal{N} \sim 10^4$, the total energy

[4] https://www.nvidia.com/en-us/data-center/products/a30-gpu/
[5] https://www.nvidia.com/en-us/data-center/a100/

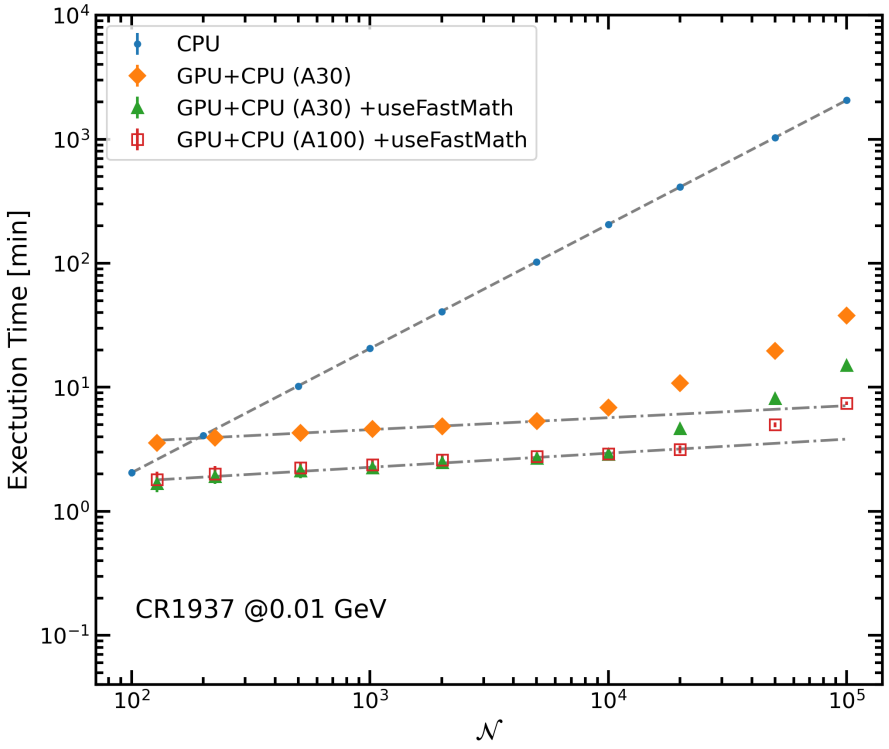

Figure 2: CPU-only (blue points) and CPU+GPU code on A30 without fast math (orange points), with fats math (green points) and on A100 with fast math (red points) execution time in minutes with respect to the number of simulated events $\mathcal{N}$. Here 0.01 GeV GCR protons at Earth Orbit on CR-1937 are simulated. The grey lines represent the linear fit of the point in the $\mathcal{N}$-range $10^2 \sim 10^4$. CPU+GPU code execution times are evaluated by using 1 GPU board.

consumption of the CPU+GPU code is $\sim$ 19 and $\sim$ 11 times smaller than the CPU-only one, when executed with the A30 and A100 respectively.

5. Conclusion

From the illustrated tests, it was evident that, for a large number of injected particles (i.e. less numerical uncertainties), CPU+GPU code is game-changing in the execution time of solving Parker's equation with the SDE approach compared to the CPU-only one. Furthermore, we tested the performances of the code on different GPUs to find the best hardware features to execute it. These results, joined with the availability of high-performance GPUs at affordable cost, as well the possibility to install several GPUs on a relatively small cluster and with sustainable power consumption allow the application of parameters scanning algorithms to improve the knowledge of model parameters space.

Acknowledgments

This work was carried out using the HelMod tool which currently is supported within the framework of space radiation environment activities of ASIF –ASI (Agenzia Spaziale Italiana) Supported Irradiation Facilities–, e.g., ASIF implementation agreements 2017-22-HD.0 ASI-ENEA, 2017-15-HD.0 ASI-INFN, 2021-39-HH.0 ASI-ENEA and ASIF implementation agreement 2021-36-HH.0 involving ASI and Milano-Bicocca University. This work is also supported by ESA (European Space Agency) contract 4000116146/16/NL/HK.

References

[1] J. Rankin, V. Bindi, A.M. Bykov, A.C. Cummings, S. Della Torre, V. Florinski, B Heber, M.S. Potgieter, E.C. Stone, and M. Zhang. Galactic cosmic rays throughout the heliosphere and in the very local interstellar medium. *Space Science Reviews*, 218(5), jul 2022. doi: 10.1007/s11214-022-00912-4.

[2] N. Eugene Engelbrecht, F. Effenberger, V. Florinski, M. S. Potgieter, D. Ruffolo, R. Chhiber, A. V. Usmanov, J. S. Rankin, and P. L. Els. Theory of Cosmic Ray Transport in the Heliosphere. *Space Sci. Rev.*, 218(4):33, 2022. doi: 10.1007/s11214-022-00896-1.

[3] E. N. Parker. The passage of energetic charged particles through interplanetary space. *Plan. Space Sci.*, 13:9–49, January 1965. doi: 10.1016/0032-0633(65)90131-5.

[4] M. J. Boschini, S. Della Torre, M. Gervasi, G. La Vacca, and P. G. Rancoita. The helmod model in the works for inner and outer heliosphere: From ams to voyager probes observations. *Adv. Space Res.*, 64 (12):2459 – 2476, 2019. ISSN 0273-1177. doi: 10.1016/j.asr.2019.04.007.

[5] P. E. Klöden and E. Platen. *Numerical Solution of Stochastic Differential Equations*. Springer Edition, 1999.

[6] C.W. Gardiner. *Handbook of stochastic methods: for physics, chemistry and natural sciences*. Springer Edition, 1985.

[7] D. P. Kroese, T. Taimre, and Z. I. Botev. *Handbook of Monte Carlo methods*. Wiley, 2011.

[8] M. Zhang. A Markov Stochastic Process Theory of Cosmic-Ray Modulation. *Astrophys. J.*, 513: 409–420, March 1999. doi: 10.1086/306857.

[9] R. D. Strauss, M. S. Potgieter, I. Büsching, et al. Modeling the Modulation of Galactic and Jovian Electrons by Stochastic Processes. *Astrophys. J.*, 735:83, July 2011. doi: 10.1088/0004-637X/735/2/83.

[10] A. Kopp, I. Büsching, R. D. Strauss, et al. A stochastic differential equation code for multidimensional Fokker-Planck type problems. *Comput. Phys. Commun.*, 183:530–542, March 2012. doi: 10.1016/j.cpc.2011.11.014.

[11] D. J. Higham. An Algorithmic Introduction to Numerical Simulation of Stochastic Differential Equations. *SIAM Review*, 43:525–546, January 2001. doi: 10.1137/S0036144500378302.

[12] P. Bobik, M. J. Boschini, S. Della Torre, et al. On the forward-backward-in-time approach for monte carlo solution of parker's transport equation: One-dimensional case. *Journal of Geophysical Research: Space Physics*, 121(5):3920–3930, 2016. ISSN 2169-9402. doi: 10.1002/2015JA022237.

[13] R. Du Toit Strauss and Frederic Effenberger. A hitch-hiker's guide to stochastic differential equations. *Space Science Reviews*, 212(1-2):151–192, mar 2017. doi: 10.1007/s11214-017-0351-y. URL https://doi.org/10.1007%2Fs11214-017-0351-y.

[14] T. McKercher J. Cheng, M. Grossman. *Professional CUDA C Programming*. Wrox Press Ltd., GBR, 1st edition, 2014. ISBN 1118739329.

[15] Jason Sanders and Edward Kandrot. *CUDA by Example: An Introduction to General-Purpose GPU Programming*. Addison-Wesley Professional, 1st edition, 2010. ISBN 0131387685.

[16] J Cheng, M. Grossman, and T. McKercher. *Professional CUDA C Programming*. Wrox Press Ltd., GBR, 1st edition, 2014. ISBN 1118739329.

[17] John K. Salmon, Mark A. Moraes, Ron O. Dror, and David E. Shaw. Parallel random numbers: As easy as 1, 2, 3. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, number 16 in SC '11, page 12, New York, NY, USA, 2011. Association for Computing Machinery. ISBN 9781450307710. doi: 10.1145/2063384.2063405. URL https://doi.org/10.1145/2063384.2063405.

[18] Joshua Romero, Mauro Bisson, Massimiliano Fatica, and Massimo Bernaschi. High performance implementations of the 2d ising model on gpus. *Computer Physics Communications*, 256:107473, 2020. ISSN 0010-4655. doi: https://doi.org/10.1016/j.cpc.2020.107473. URL https://www.sciencedirect.com/science/article/pii/S0010465520302228.

[19] Tair Askar, Bekdaulet Shukirgaliyev, Martin Lukac, and Ernazar Abdikamalov. Evaluation of pseudo-random number generation on gpu cards. *Computation*, 9(12), 2021. ISSN 2079-3197. URL https://www.mdpi.com/2079-3197/9/12/142.

[20] M. J. Boschini, S. Della Torre, M. Gervasi, G. La Vacca, and P.G. Rancoita. Propagation of cosmic rays in heliosphere: the helmod model. *Adv. Space Res.*, 62(10):2859 – 2879, 2018. ISSN 0273-1177. doi: 10.1016/j.asr.2017.04.017.