Astronomy and Computing 55 (2026) 101043

Contents lists available at ScienceDirect

Astronomy and Computing

journal homepage: www.elsevier.com/locate/ascom

Full length paper

Massive stochastic simulation of cosmic rays propagation in the heliosphere: The COSMICA code☆

Leone Bacciu [a,1], Matteo Grazioso [a,b,1], Giovanni Cavallotto [b,c], Stefano Della Torre [b,c], Massimo Gervasi [b,d], Giuseppe La Vacca [b,d], Sabina Rossi [a], Marco S. Nobile [a,*]

[a] Ca' Foscari University of Venice, Department of Environmental Sciences, Informatics and Statistics, Via Torino 155, Venezia Mestre, 30172, Italy
[b] National Institute for Nuclear Physics (INFN), Division of Milano-Bicocca, Piazza della Scienza, 3, Milano (MI), 20126, Italy
[c] ICSC - Centro Nazionale di Ricerca in HPC, Big Data and Quantum Computing, Via Magnanelli, 2, Casalecchio di Reno (BO), 40033, Italy
[d] University of Milano-Bicocca, Department of Informatics, Systems and Communication (DISCo), Piazza della Scienza, 3, Milano, 20126, Italy

ARTICLE INFO

Keywords:
Galactic cosmic rays
Heliosphere
SDE
GPU
CUDA/C++
Parallel computing
Space radiation environment
Cosmic-ray modulation

ABSTRACT

The accurate modeling of galactic cosmic ray (GCR) propagation in the heliosphere requires solving the Parker Transport Equation (PTE), a multidimensional nonlinear equation that cannot be addressed analytically without strong approximations. In recent decades, stochastic differential equation (SDE)–Monte Carlo methods have emerged as a powerful numerical strategy for this problem, thanks to their numerical stability, relatively low memory requirements, and intrinsic parallelism. The increasing availability of general-purpose Graphics Processing Units (GPUs) has further revolutionized this approach by enabling massive parallelization of particle trajectories at relatively low cost. In this work, we introduce COSMICA (COde for a Speedy Montecarlo Involving Cuda Architecture), a new open-source multi-GPU code written in CUDA/C++ for the three-dimensional solution of the PTE. COSMICA has been specifically designed to optimize GPU resource usage and scalability, with strategies including memory hierarchy exploitation, register-conscious kernel design, warp-aware scheduling, and parameter reordering for multi-GPU execution. Benchmark results demonstrate that COSMICA reduces runtimes from weeks to hours for large-scale simulations. These optimizations make COSMICA a versatile tool for systematic studies of cosmic-ray modulation and parameter exploration, thereby expanding the feasibility of investigations that were previously computationally prohibitive. The present article constitutes the first part of a two-paper series, focusing on code design and computational performance; a companion paper will present its validation against benchmark models.

1. Introduction

The accurate description of Galactic Cosmic Ray (GCR) transport in the heliosphere requires solving the Parker Transport Equation (PTE) (Parker, 1965), a Fokker–Planck-like equation that accounts for diffusion, convection, drift, and adiabatic energy losses of charged particles in the solar environment. Due to its multidimensional and nonlinear nature, the PTE cannot be solved analytically without strong assumptions, and numerical approaches have therefore become essential.

Among the available strategies, stochastic differential equation (SDE) techniques (*e.g.*, Gardiner, 1985) have gained increasing popularity in recent decades for heliospheric transport studies (Florinski and Pogorelov, 2009; Effenberger et al., 2012; Kopp et al., 2012; Zhao et al., 2014; Boschini et al., 2019; Moloto et al., 2019; Vogt et al., 2020).

The SDE–Monte Carlo formulation redefines the transport problem as the integration of ordinary differential equations along stochastic trajectories, rather than directly solving the partial differential equation. This method offers several well-recognized advantages (Strauss and Effenberger, 2017): (i) unconditional numerical stability, (ii) the ability to handle steep spatial or temporal gradients, (iii) relatively low memory requirements, and (iv) intrinsic parallelism, since each stochastic trajectory is independent of the others. The last property is particularly important from a computational perspective, as it makes the approach ideally suited for high-performance parallel computing architectures.

Over the past decade, the widespread availability of graphics processing units (GPUs) has revolutionized computational (astro)physics by enabling highly parallel calculations at relatively low cost. In the

☆ This article is part of a Special issue entitled: 'HPC in Cosmology and Astrophysics' published in Astronomy and Computing.

* Corresponding author.
E-mail addresses: leone.bacciu@unive.it (L. Bacciu), matteo.grazioso@unive.it (M. Grazioso), marco.nobile@unive.it (M.S. Nobile).

[1] These authors contributed equally to this work.

https://doi.org/10.1016/j.ascom.2025.101043
Received 30 September 2025; Received in revised form 24 November 2025; Accepted 4 December 2025
Available online 6 December 2025
2213-1337/© 2025 The Authors. Published by Elsevier B.V. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

field of cosmic-ray transport, GPUs have been successfully exploited to accelerate SDE–Monte Carlo methods, significantly reducing computing times while maintaining accuracy (Dunzlaff et al., 2015; Vogt et al., 2020; Solanik et al., 2022; Solanik et al., 2023).

By mapping each stochastic trajectory onto a separate GPU thread, it becomes possible to integrate a vast number of realizations simultaneously, achieving performance levels previously attainable only with large CPU clusters. This development has opened the door to systematic studies that were once computationally prohibitive, including large-scale parameter scans and time-dependent modeling of solar modulation under realistic heliospheric conditions (see, *e.g.*, Della Torre et al., 2025).

Several codes have implemented this approach with varying degrees of GPU acceleration. The earliest implementations can be found in Strauss et al. (*e.g.*, 2011b), Dunzlaff et al. (*e.g.*, 2015), Moloto et al. (*e.g.*, 2019) where they solved the PTE for some specific cases, paving the way to broader studies, using similar techniques but studying the long term solar modulation for a wide range of GCR observations within the heliosphere. Other frameworks – such as Geliosphere (Solanik et al., 2023), an open-source fork of SOLARPROP (Kappl, 2016), and SDEMMA (Qin and Shen, 2017), a closed-source time-dependent solver – have further demonstrated the feasibility of GPU acceleration. Finally, HelMod-4 (Boschini et al., 2019), in its HelMod-4/CUDA version (Boschini et al., 2024), represents one of the first large-scale applications of GPU computing to the full three-dimensional PTE, solved for all relevant GCR ions, achieving excellent agreement with spacecraft data at Earth and beyond.

Nevertheless, most of these efforts have been primarily motivated by the exploration of physical models, with optimization of raw computational performance often remaining a secondary goal.

In this work, we take a different perspective by focusing on the technological aspects of GPU acceleration for SDE–Monte Carlo solvers. We present the COde for a Speedy Montecarlo Involving Cuda Architecture (COSMICA), a new open-source code written in CUDA/C++ that solves the three-dimensional PTE using multi-GPU acceleration. The aim of COSMICA is not only to reproduce the physics of well-established models such as HelMod-4/CUDA but also to explore the performance limits of modern GPU hardware to enable computationally demanding tasks – such as parameter optimization and long-term modulation studies – that would otherwise require access to national-scale HPC facilities (see, *e.g.*, discussion in Della Torre et al., 2025). The modular design of the code allows the integration of alternative physical models, making the optimization strategies and benchmarks presented here generally applicable to a wide class of transport problems solved through SDE–Monte Carlo techniques.

By shifting the emphasis from purely physical modeling to computational performance, our work highlights how GPU technology can transform the feasibility and accessibility of advanced numerical simulations of cosmic-ray transport. This work is the first part of a series of two contributions (see Cavallotto et al., 2025, for the second part, *i.e.* the validation of COSMICA against a benchmark model) which aims to demonstrate that carefully optimized GPU algorithms are not only capable of reproducing state-of-the-art physics models, but also of providing the computational efficiency necessary for systematic studies of heliospheric modulation and its impact on space radiation environments.

This Paper is organized as follows: Section 2 provides the theoretical background on the Parker Transport Equation and the SDE Monte-Carlo approach; Section 3 details the architecture and implementation of the COSMICA code; Section 4 outlines the advanced GPU optimization strategies; Section 5 describe memory hierarchy exploitation strategies; Section 6 introduces structured reordering of the simulations aimed at improving execution efficiency, memory access patterns, and balanced parallelism across device; and finally, Section 7 presents a performance analysis and benchmarking of the code on various hardware configurations.

2. Parker transport equation

The theory on the transport of GCR in the heliosphere, and in astrophysical plasma in general, is based on a Fokker–Planck equation named the Parker Equation after Parker (1965). In its fundamental work, Parker demonstrated that charged particles propagating in a magnetized plasma are subject to a random walk that can be described as a Markov process (Zhang, 1999). It is common to express PTE by means of the so-called omnidirectional distribution function f (see, *e.g.*, Strauss and Effenberger, 2017; Engelbrecht et al., 2022):

$$\frac{\partial f}{\partial t} = \nabla \cdot \left(\mathbf{K^S} \cdot \nabla f\right) - \left(\mathbf{v_d} + \mathbf{v_{sw}}\right) \cdot \nabla f + \frac{R}{3} \nabla \cdot \mathbf{v_{sw}} \frac{\partial f}{\partial R} \tag{1}$$

where f is the average on all directions of the number of particles in the infinitesimal space volume element at time t and per unit of momentum; in this formulation it is possible to recognize the principal physical processes that regulate the solar modulation: (a) the diffusion process, related to the symmetric part of the diffusion tensor ($\mathbf{K^S}$), (b) the convection process related to the outward solar wind flow ($\mathbf{v_{sw}}$), (c) the particle drift due to large scale magnetic field gradients and expressed in terms of drift velocity ($\mathbf{v_d}$), and (d) the adiabatic energy loss due to expanding plasma of the solar wind. In Eq. (1), instead of particle momentum we used particle rigidity ($R = \frac{pc}{Ze}$ where p is particle momentum, c is light speed and Ze is the particle charge) since it represents a more natural quantity for exploring the physical process involved and, as can be noted in Boschini et al. (2018), this formulation allows us to reduce the complexity of the stochastic differential equation present in the next section.

2.1. SDE Monte-Carlo approach

PTE is a partial differential equation that cannot easily be solved analytically. Among the different numerical solver techniques, in the last decades it has become popular to use the so called stochastic differential equations (SDE) Monte Carlo technique (Gardiner, 1985; Zhang, 1999; Strauss et al., 2011a; Kopp et al., 2012; Bobik et al., 2016; Strauss and Effenberger, 2017; Boschini et al., 2018). With this approach, PTE should be rewritten in the form of the *backward Kolmogorov equation* (which is the backward in time formulation of the Fokker–Planck equation):

$$\frac{\partial f}{\partial s} = \sum_i A_i(s,y) \frac{\partial f}{\partial y_i} + \frac{1}{2} \sum_{i,j} D_{i,j}(s,y) \frac{\partial^2 f}{\partial y_i \partial y_j} \tag{2}$$

This expression is fully equivalent to a set of stochastic differential equations (SDE) generically expressed in the form (Øksendal, 2010; Freidlin, 1985):

$$dy_i(s) = A_i(y,s)ds + \sum_j B_{i,j}(y,s)dW_j(s) \tag{3}$$

where A_i is the same in both Eq. (2) and (3), and $\mathbf{D} = \mathbf{B}\mathbf{B}^\mathbf{T}$. In the SDE formalism, ds represent the backward in time step; $A_i(y,s)$ is the advective term that contains all deterministic process like convection and particle drift; $\sum_j B_{i,j}(y,s)dW_j(s)$ is the diffusive term that describes all stochastic processes like particle diffusion and shock re-acceleration, in this term dW indicates the increment due to a *standard Wiener process* (see, *e.g.*, Appendix A of Zhang, 1999 and Section 2 of Higham, 2001), a time stationary stochastic Lévy process where the time increments have a Normal distribution with a mean of zero (*i.e.* a Gaussian distribution) and a variance of dt (see introduction of Gardiner, 2009). In case of PTE, the derivation of the terms $\mathbf{A}$ and $\mathbf{B}$ can be found, for example, in Pei et al. (2010) and Appendix A of Boschini et al. (2018).

In the Monte Carlo framework, the determination of modulated cosmic-ray spectra is carried out by numerically propagating a set of representative points in phase space, defined by $(\mathbf{x}, R, s)$, in the reverse-time direction. The backward in time evolution is performed from the so-called evaluation point, – *i.e.*, the coordinates $\mathbf{x}_0, R_0, s_0$ where the flux is to be reconstructed – up to the heliospheric boundary, following

the discretized form of Eq. (3). For this purpose, the Euler–Maruyama scheme is generally adopted as a standard stochastic integrator (see, *e.g.*, Section 5.6.1 in Kroese et al., 2011). The forward-in-time formulation is not accounted for in this work since is was proven to be less efficient from the computational perspective. A comparison of forward-in-time and backward-in-time formulations can be found in Bobik et al. (2016).

The individual entities undergoing this numerical evolution are typically referred to as *quasi-particles*. This terminology emphasizes that, although their propagation resembles the trajectories that real galactic cosmic-ray particles might follow, they do not correspond to physical paths in a strict sense. Instead, each quasi-particle trajectory should be interpreted as a stochastic realization of the underlying transport process. As clarified in Strauss and Effenberger (2017) and further emphasized by Boschini et al. (2024), the integration of Eq. (3) for a single quasi-particle does not carry physical meaning by itself. Rather, the ensemble of trajectories collectively forms a probabilistic description of the transport problem: together, they define an effective quasi-Green's function $G(R_0|R)$, which quantifies the probability that a particle observed at a specific location, with rigidity R_0, originally entered the heliosphere at the boundary with rigidity R, prior to undergoing solar modulation (Zhang, 1999).

The raw output of this solver technique is a histogram G_{R_0} that maps the distribution function of *quasi-particles* rigidity at the end of the stochastic path, which corresponds to the heliosphere boundary. The normalized G_{R_0} allows to reconstruct $G(R_0|R)$ and, in turn, the modulated intensity of GCR.

From an analytical point of view, the modulated spectra is computed by Zhang (1999):

$$J(\mathbf{x}_0, R_0, s_0) = \frac{\beta_0 R_0}{N} \sum_{p=0}^{N} \frac{J_{LIS}(T_p)}{R_p^2} \tag{4}$$

$$\approx \beta_0 R_0 \int_{R_0}^{\infty} \frac{J_{LIS}(R)}{\beta(R)R^2} \cdot G(R_0|R) dR \tag{5}$$

$$\approx \beta_0 R_0 \sum_i \frac{J_{LIS}(R_i)}{\beta(R_i)R_i^2} \cdot G_{R_0}(R_i)\Delta R_i \tag{6}$$

where p is the index identifying each *quasi-particle* object, R_p and T_p are the values of the *quasi-particle* object rigidity and kinetic energy per nucleon at the end of the stochastic trajectory, $J_{LIS}(T_p)$ is the LIS spectra evaluated at T_p, and β_0 is the conversion factor from $J(T_0)$ to $J(R_0)$. A more complete description on how to evaluated the more general PTE solution using SDE approach is described in Cavallotto et al. (2025). Eq. (4) can be computed directly during the numerical solution for a given J_{LIS} modeled within the code. Nevertheless, due to lack of $\sim$ GV rigidities GCR direct measurements at heliosphere boundaries, J_{LIS} can be considered more a model parameter than a real boundary condition for it, thus, it is preferred to allowing to evaluate Eq. (6) offline with different J_{LIS} which must be refined using other technique, like, *e.g.*, the one described in Boschini et al. (2017). Moreover, the choice to save distribution histograms instead of raw events has the advantage of restricting the amount of information that must be stored in memory during the computation, since, instead of recording all particles, it records the histogram of the distribution function.

3. The COSMICA numerical solver

We developed COSMICA to solve PTE in the framework of the Monte Carlo SDE approach having as its main goal to optimize the code to best perform on GPU both in HPC systems and desktop environment. Detail on the implementation of the physical model can be found in Cavallotto et al. (2025). COSMICA is implemented using the CUDA (version ≥12.4) language and C++20, specifically for NVIDIA GPU architectures. Each trajectory is given a unique thread in the GPU allowing for large-scale parallel computation consistent with the *Single Instruction, Multiple Data* (SIMD) paradigm. This flexibility makes the simulation process *embarrassingly parallel* and allows SDE integration to be executed very efficiently on modern High-Performance Computing (HPC) architectures and *Single Instruction, Multiple Thread* (SIMT) devices like modern general-purpose programmable GPUs. Details on this specific architecture, including the GPU programming model, memory hierarchy, and scheduling strategies are described in Section 4.

The simulation process begins with establishing the key parameters: total number of trajectories per rigidity, heliospheric model inputs, particle species attributes, and initial spatial and energy distributions. The particle-specific information and thread-invariant constants are strategically allocated to different types of GPU memory, selected according to their access frequency, mutability, and latency characteristics. This targeted memory allocation – informed by the architectural behavior of each memory type – facilitates efficient resource utilization and contributes to maximizing the usage of the GPU compute capacity. A detailed analysis of these strategies is presented in Section 5.

In the propagation phase, the GPU determines the SDE-based trajectory of every *quasi-particle*'s transport through the heliospheric domain. The local transport coefficients are computed at each time integration step; the calculations are repeated until the particle leaves the simulation area.

Results are stored in block-level histograms and written using atomic operations to prevent critical runs and write conflicts. At the end of kernel's execution, the partial histograms are transferred to the host. When run on multi-GPU systems (as described in Section 6) COSMICA automatically distributes the relevant portion of the computation among the available GPUs. These inter-dependent segments run independently across the devices, without the need for explicit synchronization barriers at runtime. The histograms produced by each GPU are transferred and merged by the host.

We will describe technical details, including complete background on GPU-accelerated HPC, maximum exploitations of memory hierarchy, precision handling, and particle reordering techniques for multi-GPU workflow, in the following Sections. The code has been released as open source under the GPLv3 license and is accessible through the project repository (COSMICA Repository, 2025) (commit ID: `cab03e3`). Example input configuration files (YAML) and parameter sets used for the benchmarks are included in the repository to facilitate reproducibility.

4. Background on GPU-accelerated high-performance computing

GPUs have become foundational in High-Performance Scientific Computing, which is the substrate for advancements in deep learning, numerical modeling, and large-scale data analytics. While Central Processing Units (CPUs) are architectural designs that support optimized sequential processing and low-latency task performance, GPUs are built for massively parallel computation and leverage thousands of simple cores to execute many threads concurrently (NVIDIA Corporation, 2025b), catalyzing a shift toward General-Purpose (scientific) computing on GPUs.

4.1. NVIDIA GPU architecture and memory hierarchy

In order to leverage the parallel architecture of GPUs, a dedicated programming model is needed. The most popular of these is NVIDIA's Compute Unified Device Architecture (CUDA), is a parallel computing platform and programming model for general-purpose programming on GPUs in languages like C++ as was our case, CUDA abstracts the underlying GPU hardware into a hierarchy of execution and memory models that make explicit fine-grained control to the experts for high-performance computing.

CUDA is based on the SIMT execution model, where groups of threads called *warps* execute instruction in lock-step, with interleaving applied in the presence of thread divergence. Thread divergence occurs when threads in the same *warp* follow different execution paths in a

Table 1
CUDA memory hierarchy: types, usage and performance impact (NVIDIA Corporation, 2025a,b).

Memory type	Characteristics	Performance
Global Memory	The largest and most general memory space in a GPU; accessible by any thread and by the host CPU.	High bandwidth, high latency, hundreds of cycles.
Shared Memory	An on-chip memory space shared by threads within the same block. It can be used as a user-managed cache to avoid accessing global memory. It enables communication between threads within the same block.	Low latency, high bandwidth.
Registers	Threads have access to a small number of fast registers for storage of private variables. When a thread exceeds its register budget (*i.e.*, the register pressure is too high), it spills into "local" memory, which is part of the global memory and therefore incurs the large latency associated with accessing global memory.	Extremely fast unless spilling to local memory.
Constant Memory	Read-only memory space with hardware caching. Values in constant memory can be broadcast to all threads in a warp: if all threads access the same address, the constant memory can serve the request efficiently via a single transaction; else, its throughput is strongly affected. Relatively small (~ 64 KB).	If access is warp uniform, as fast as a register access.
Read-Only Data Cache	By default, global memory accesses are cached in a L2 cache. However, data that remains read-only for the duration of a kernel can be cached in a L1 cache. The compiler may detect read-only accesses automatically, or the programmer can explicitly use `__ldg()` to instruct caching.	Very low latency.

branch. Execution is performed by computational units called Streaming Multiprocessors (SMs). The number of SMs in a GPU determines the extent of parallel workload it can sustain.

SMs contain several CUDA cores, special function units, and a small amount of shared memory. All of these components are carefully designed to maximize throughput by hiding memory and execution latencies, and employing fine-grained multi-threading (Hong and Kim, 2009; Kirk and Wen-Mei, 2016).

The memory hierarchy in NVIDIA GPUs is a primary factor in application performance. In this context, *latency* refers to the time delay between issuing a memory request and receiving the data, typically measured in *clock cycles*. *Bandwidth*, on the other hand, denotes the amount of data that can be transferred per unit of time. While high-bandwidth memory systems can move large amounts of data quickly, high latency can still cause performance bottlenecks, particularly in memory spaces like global or local memory. To optimize performance, we took these factors into consideration and focused our efforts on reducing memory latencies as much as possible, and on maximizing bandwidth efficiency by carefully managing memory access patterns of our simulator. The types of memory available in CUDA, along with their characteristics and performances, are summarized in Table 1. We carefully optimized the performance of our simulator by exploiting the memory hierarchy. Specifically, we maximized shared memory reuse, coalesced global memory accesses, and minimized register spilling.

To classify and characterize the wide variety of features in their GPUs, NVIDIA exploits the concept of Compute Capability (CC). CC is NVIDIA's way of labeling different GPU microarchitectures with certain sets of features and meaning the architectural generation and capabilities of the hardware. For example, NVIDIA's A30 and A100 GPUs have CC 8.0, while the A40 and RTX 3090 have CC 8.6. It is important to note that they both have the same major CC version (8) but have some significant architectural and performance differences.

4.2. CUDA programming model

At the heart of the execution model is the concept of *kernels*, C-like functions that are launched on the device and executed simultaneously by many threads. Threads are grouped into *thread blocks* that are grouped into a *grid* structure. Each thread has a unique index within a block, and each block has a unique index within the grid, allowing the code to map threads to data in a scalable manner. Thread blocks execute independently on the available SMs, achieving massive parallelism and automatic scaling transparently for the programmer. Once a block is scheduled for execution on an SM, it stays resident there until all its threads have completed, freeing space for a new block.

Threads within a block are subdivided into *warps* of 32 threads, which execute instructions in lockstep. Branch divergence within a warp (*e.g.*, due to a conditional statement) is technically implemented by serialization, which strongly affects overall efficiency. Serialization enables interleaving, allowing sub-warp groups to stall on data reads or special instructions while other threads in the warp execute a different branch path (NVIDIA, 2017). This only partially mitigates the impact of serialization; therefore, warp-aware programming is essential for performance, which is a central focus of our work.

Our work took advantage of CUDA's programming model to generate compute kernels that are as parallel, memory efficient, and as close to the GPU architecture as possible.

This meant appropriately setting the thread-block dimensions to best match the problem granularity without exceeding hardware limits, minimizing warp divergence to reduce serialization to maximize SIMT efficiency, and utilizing register level optimizations to minimize latency and maximize occupancy.

CUDA's profiling tools helped detect and reduce serialization bottlenecks, and identify non-coalesced memory accesses. Coupling architecture-aware kernel design together with empirical style performance tuning, we obtained large increases in throughput and run time efficiency, demonstrating how informed optimization can impact GPU-accelerated workloads.

In our implementation, we largely used C++20's features to improve both the expressiveness and performance of the CUDA code: as a matter of fact, it provides better abstractions and compile-time evaluations (Stroustrup, 2025).

For instance, the consistent use of `const` to enforce immutability and guide the compiler toward better optimization opportunities. The use of `auto` enhances expressiveness by enabling concise and type-safe

variable declarations without sacrificing performance. Data structures such as indices, particles, or multidimensional coordinates are modeled as user-defined types (*i.e.*, lightweight objects), which are internally decomposed at compile time into simple, adjacent memory representations. This enables a high level of abstraction in code design without incurring any runtime overhead.

Additionally, the adoption of the `if-init` statement – which introduces variable initialization scoped to a conditional statement – allows for finer-grained control over the lifetime of temporary variables. In the context of GPU programming, this pattern aids in the early release of register resources, reducing register pressure and improving overall occupancy, as discussed in the next Section.

4.3. GPU occupancy and resource saturation

Achieving high performance on CUDA-enabled GPUs requires not only well-parallelized kernels, but also a careful balance between hardware resource utilization and scheduling efficiency. Among the key performance metrics, *occupancy* and *resource saturation* were used to evaluate how effectively a kernel utilizes the computational capabilities of the GPU. The metrics are defined as follows.

- *GPU occupancy* is the ratio of active warps on a Streaming Multiprocessor (SM) to the maximum number of warps the hardware can support. High occupancy improves latency hiding by allowing the scheduler to switch to a ready warp when the other warps are stalled on memory accesses or execution stalls. Hence, occupancy should be maximized and it is (negatively) affected by register pressure, usage of shared memory, and thread-per-block number. The relationship between the number of threads, the threads per block and the maximum occupancy is illustrated in Fig. 1.
- *Resource saturation* happens when all SMs are actively utilized, idle cycles are minimized, warp activity is high, and the device sustains elevated throughput, whether the kernel is compute- or memory-bound. Maintaining a high number of active threads and blocks per SM is particularly beneficial, as it provides the warp scheduler with a sufficient pool of ready warps to select from when others are stalled due to memory latency or execution dependencies. This thread-level parallelism enables effective latency hiding: while some warps wait for memory transactions, others can proceed without interruption. This means that this fine-grained interleaving of warps allows the GPU to maintain high throughput even when some warps are stalled due to memory accesses or other dependencies.

These two metrics provide insight into the degree to which a GPU's SMs are kept busy and how resource constraints, such as registers and thread slots, affect kernel execution and scalability. The metrics are strongly influenced by both hardware architecture and the software environment. Achieving saturation requires scheduling the maximum possible workload in each execution. Notably, GPUs equipped with a greater number of streaming multiprocessors (SMs) demand proportionally more work to reach saturation.

In contrast, occupancy is determined by the specific constraints of the GPU's CC, including limits on the number of registers per block, the maximum threads per block, and other parameters.

In this work, we strived for very high occupancy while effectively using per-thread resources on GPUs with CC 8.x. Specifically, by scheduling the particles propagation in such a way that both optimized warp cache local accesses and maximized the overall number of ready warps.

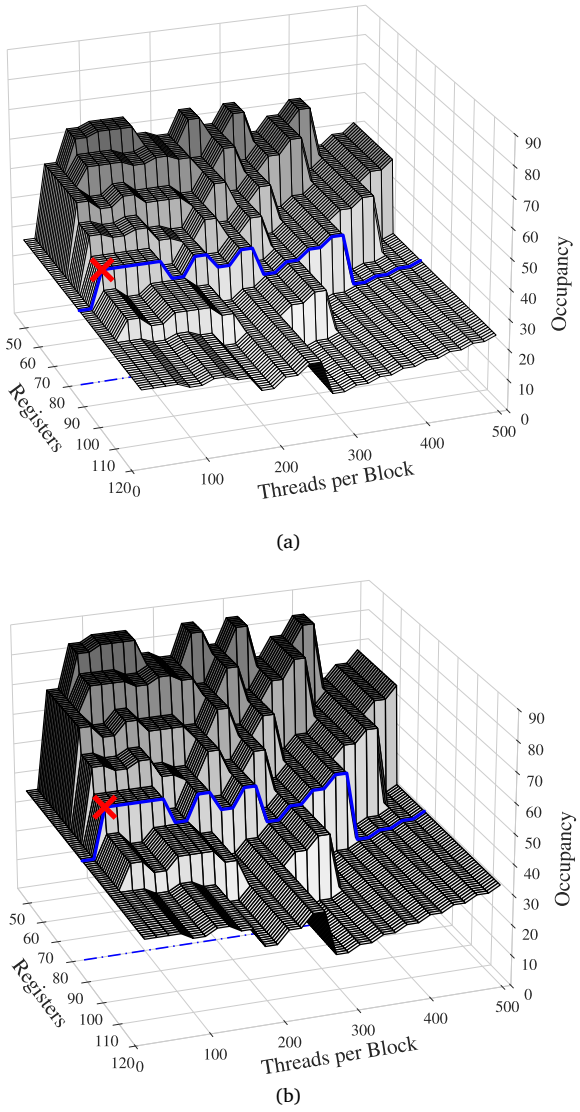


Fig. 1. Visualization of the relationship between the number of registers, threads per block, and the corresponding maximum achievable occupancy on a CUDA GPU. The blue curve highlights the Section relevant to COSMICA's configuration, where each thread utilizes 80 registers. The red cross marks our chosen configuration of 48 threads per block, which is the smallest thread count that achieves the maximum occupancy for this register usage. Results for CC 8.0 are shown in Fig. (a), while results for CC 8.6 are shown in Fig. (b).

5. Memory hierarchy optimization and precision control in GPU accelerated simulation

In this Section we will discuss the implementation choices and optimizations of our CUDA-powered simulator. The strategies we will describe are related to the specific architecture that we targeted (*i.e.*, CC $\geq$ 8) and are essential for sustaining high throughput in register-bound kernels, effectively managing limited on-chip resources such as constant memory.

5.1. Precision strategy: preferential use of single-precision arithmetic

An important architectural and algorithmic choice in our implementation was the consistent choice of single precision (`float`) versus

double precision (`double`) in all computational kernels. This choice is based on three interconnected factors affecting GPU and resource performance and efficiency.

Single-precision operations can achieve much higher throughput on NVIDIA GPUs due to existing hardware fast paths in the execution units. Both NVIDIA and software vendors expect that workloads on the modern architecture such as CC 8.x are optimized for single-precision workloads.

HelMod-4/CUDA already used single-precision for all floating-point operations, but many mathematical computations were executed using mixed-precision functions. This does not affect the numerical result, as both the inputs and outputs have the same precision, but when single-precision variables are used into mixed-precision expressions, the compiler automatically performs implicit type casting. Moreover, the validated results from HelMod-4/CUDA strongly indicate that additional floating point precision is not required for these simulations. When single-precision variables are used into mixed-precision expressions, the compiler automatically performs implicit type casting. These implicit conversions introduce additional instructions and increase register usage, as temporary variables are generated during type promotion. This elevated register demand can constrain scalability and degrade performance, particularly in highly parallel workloads. For this reason, we employed CUDA's Single Precision Intrinsics, special functions that compute special mathematical operations such as square root and sine. Intrinsic functions are restricted to device code, as they leverage dedicated machine instructions and execution units to deliver significantly faster computations compared to standard instructions (NVIDIA, 2025).

The last and most current constraint imposed on our procedures is register pressure. Profiling our data, we recognize that our application is not primarily bottlenecked by global memory access, but by the high number of registers that are required by each thread.

5.2. Register-conscious design

Given the physical constraints of CC 8.x GPUs, we can calculate the maximum theoretical occupancy achievable by a kernel. The main kernel our simulator was strongly optimized in order to reduce the number of registers to 80. Further reductions in register usage, either by simplifying expressions or modifying the kernel logic, have proven ineffective: they either compromise numerical accuracy or result in register spilling.

Registers are allocated at the warp level (with each warp consisting of 32 threads) which translates to $80 \cdot 32 = 2560$ registers per warp. Each SM offers 65 536 registers, which ultimately limits the number of simultaneously active warps to roughly $65\,536 / 2560 \approx 25$, but due to warp allocation granularity (in units of 4 warps), the actual maximum is 24 warps per SM. For CC 8.0, the hardware would, in principle, allows up to 64 resident warps per SM, yielding a theoretical occupancy of $24 / 64 = 37.5\,\%$. For CC 8.6, such limit is 48 warps, yielding a theoretical maximum occupancy of 50 %.

Another factor influencing theoretical occupancy is the number of threads per block. As shown in Fig. 1, the minimum number of threads per block achieving maximum occupancy is 48 for both CC versions. We selected this minimal size to allow blocks to be evicted from SMs as soon as possible; larger blocks would remain resident even if only a single thread were still active, *e.g.*, when processing a slower particle. With smaller blocks, a single long-running thread monopolizes fewer resources, thus reducing overall resource contention. The absolute best configuration depends on many factor like the hardware architecture and the nature of the specific problem to be solved during the specific run, such generalization goes beyond the scope of this work.

We also tested the use of shared memory to reduce latencies, but the strategy made the implementation more complex without a real improvement of performance, at the cost of increased register pressure. Because of that, user shared memory is not used in this implementation.

It is worth noting that these occupancy figures are theoretical and are only achieved when the GPU is sufficiently saturated with scheduled threads.

5.3. Warp-aware hybrid memory strategy

Constant memory on NVIDIA GPUs is a specialized, read-only memory space with a limited size that is highly optimized for broadcast-style access patterns, where all the threads in a warp read the same address, as already described in Section 4.1. Access to constant memory is serviced from a dedicated on-chip cache, so the memory is read with low latency under warp-uniform conditions. This makes constant memory well-suited for scalar values that are frequently reused, as well as configuration parameters. However, the limited size of constant memory prohibits usage for simulations that must work with many different configurations.

In our case, the total volume of simulation parameters and especially dynamic coefficients such as K_0 that may vary at each parameterization can easily exceed the hardware (64 KB) limit imposed on constant memory. To bypass the limit to work within constant memory, we employ a hybrid memory strategy that distinguishes between static and dynamic parameter types.

COSMICA can simulate multiple parametrizations within a single run. These parametrizations constitute the *dynamic parameters*, whereas parameters shared to all parametrizations, *e.g.* those describing the heliosphere, are referred to as *static parameters*. Static parameters have to be allocated in advance, which limits their flexibility, but their access is extremely fast (see Table 1).

On the other hand, *dynamic parameters* are stored in the global memory, but accessed using the `__ldg()` intrinsic function, which enables caching them in the Read-Only Data Cache (see Table 1).

While these *dynamic parameters* are located in global memory, their access patterns are very orderly. We arranged our execution model so that thread blocks are able to utilize few parameterizations at a time. We developed this scheduling mechanic based on the temporal proximity of the parameterizations in the simulation domain. In fact, by ensuring sure that threads that operate on the same parameter set or an adjacent parameter set are launched at the same time, we are able to maintain a high level of warp-uniformity in memory access. This memory access uniformity allows the hardware to automatically cache the global memory loads in the constant cache, effectively extending the utility of constant memory beyond what is available based on its nominal capacity.

Profiling results obtained using *NVIDIA Nsight Compute* have verified that these dynamic parameters are indeed serviced via the constant cache during execution. Although in global memory, the cache behavior is very similar to true constant memory, which benefits from frequent cache hits and extremely low latencies. This cache mechanism alleviates traffic between registers and global memory and alleviates some pressure on local memory.

This hybrid use of constant and global memory, combined with `__ldg()` and a warp-aware scheduling strategy, allows our simulation to handle parameter spaces larger than the physical size of the constant space meaning that it has kept multiple frequently-accessed values available in low-latency cached memory again without consuming register resources and while maintaining reasonable kernel throughput.

5.4. Architectural implications

The architectural considerations stated in this Section inform every stage of our design. By aligning kernel structure and memory layout with the hardware's register and memory hierarchy, we reduced latency, avoid register spilling, and sustained high throughput. This hardware-aware approach is necessary to optimize performance and promote numeric reliability in our GPU-based simulations. These optimizations are most effective when the GPU is fully saturated. To that end, we developed an indexing scheme that enables the scheduling of the maximum number of particles within a single kernel launch, even when particles differ in parameterization, rigidity, or other physical properties. This strategy ensures that all available GPU resources are utilized efficiently, maximizing concurrency and improving overall throughput.

6. Particle reordering and parallel execution in multi-GPU architecture

In the multi-GPU implementation of COSMICA, we introduce a structured reordering of the simulations aimed at improving execution efficiency, memory access patterns, and balanced parallelism across devices. Each simulated particle is characterized by a rigidity parameter, a parameterization and an isotope – defining a unique simulation instance –, a temporal period and a repetition index – identifying statistically independent realizations of the same physical scenario through variations in the random seed.

To ensure that similar computational tasks are executed consecutively and to exploit hardware coherence, particles are first sorted by increasing *rigidity*. Rigidity directly influences the propagation time and numerical stability, and therefore grouping particles by rigidity allows uniform scheduling of kernel workloads. Within each rigidity class, particles are then ordered by simulation *instance*, which includes a combination of configuration parameters and isotope. This grouping ensures that particles with identical physical settings are processed together, enabling better use of instruction and data caches. Each instance is further subdivided by *simulation period* allowing adjacent time periods to be simulated at the same time.

Finally, within a fixed period, particles are organized by their *repetition index*. Each repetition corresponds to a unique stochastic realization, namely random seed, keeping all other physical properties unchanged. Specifically, we exploit the repetition index as the primary axis for distributing the computational workload across available GPUs. Assuming the hardware consists of homogeneous GPUs – for example, multiple NVIDIA A100 units – the total number of repetitions is divided evenly among the devices. Since each GPU receives an equal number of input items, with identical characteristics, the execution time per device becomes effectively synchronized. This yields highly efficient device-level parallelism: each GPU executes its assigned kernel set, and the overall simulation completes when the last GPU finishes its share of repetitions — ideally with no delay imbalance across devices. Automatic balancing of repetitions across heterogeneous hardware in an efficient manner is not yet supported, and addressing this challenge represents an interesting direction for Future Work.

To support this execution model with minimal overhead, we introduce an index array used to quickly resolve the data location for each particle. It encodes, for each thread, the index of the correct particle data and configuration parameters in their respective arrays. By accessing simulation input data through this indirection table, the kernel execution achieves a high degree of memory coalescence and benefits from constant memory caching, reducing access latency and improving throughput in the memory subsystem.

To visually summarize the particle reordering and parallelization strategy adopted in the multi-GPU implementation of COSMICA, Fig. 2 provides a schematic overview of the hierarchical organization and index-driven memory access.

7. Computational performance assessment

To comprehensively evaluate the computational performance of the simulator, we performed an extensive series of benchmark tests across a diverse range of configurations.

These benchmark tests are grouped into two sets: in *Test A* we vary only the number of *quasi-particles* within the set $\{2^n \mid 9 \leq n \leq 16\}$, while in *Test B* we select one of the model parameter (*i.e.*, the diffusion parameter K_0 which set the absolute scale of $\mathbf{K}^{\mathbf{S}}$ in Eq. (1)) and we vary the number of its values simulated in parallel within the set $\{2^n \mid 0 \leq n \leq 7\}$ using a fixed number of *quasi-particles*. Notably, in *Test B* each individual k_0 value is tested multiple times to evaluate the variability and stability of the runtime. For the evaluation of the runtime we consider the total computing time to simulate the two isotopes of hydrogen nuclei (*i.e.*, proton and deuteron), for a single carrington rotation, spanning 23 bins from 1 to 11 GV mapping the ones used for the latest proton measurements provided by AMS–02 (Aguilar et al., 2021). Since performance increases with increasing the particle rigidity, higher bins were not considered in this work.

Table 2
Simulated time intervals along with the corresponding solar activity and HMF polarity.

CR number	Start–End date	Solar activity	HMF polarity
2117	Nov. 16 – Dec. 13, 2011	Solar minimum	Negative
2155	Sep. 17 – Oct. 15, 2014	Solar maximum	Undefined
2168	Sep. 7 – Oct. 4, 2015	Medium descendant	Positive
2181	Aug. 27 – Sep. 23, 2016	Medium descendant	Positive
2192	Jun. 23 – Jul. 20, 2017	Medium descendant	Positive
2209	Sep. 29 – Oct. 26, 2018	Solar minimum	Positive

Table 3
Key specifications of the NVIDIA GPUs used for the tests.

	A30	A100	RTX 3090	A40	RTX 3060	A2000
CC	8.0	8.0	8.6	8.6	8.6	8.6
Chipset	GA100	GA100	GA102	GA102	GA106	GA107
Boost Clock (MHz)	1440	1410	1900	1740	1425	1687
Memory Clock (MHz)	1215	1215	1219	1812	1750	1500
Bandwidth (GB/s)	933.1	1560	936.2	695.8	336	192
SMs Count	56	108	82	84	30	20
Core Count	3584	6912	10 496	10 752	3840	2560
L1 Cache (KB per SM)	192	192	128	128	128	128
L2 Cache (MB)	24	40	6	6	3	2
VRAM (GB)	24	64	24	48	6	4
TDP (W)	165	400	350	300	80	95
FP32 (TFLOPS)	10.32	19.49	35.58	37.42	10.94	8.64
Interface	PCIe	SXM	PCIe	PCIe	PCIe	PCIe

For each set, simulations were conducted across 6 distinct periods, as detailed in Table 2, each representing a different solar activity scenario during the AMS–02 data taking period (see Cavallotto et al., 2025, for the consideration of performance and validation regarding the solar activity). To assess the reliability of the COSMICA framework, the same periods were subsequently used in the second part of this Study (Cavallotto et al., 2025) for code validation against the HelMod-4/CUDA model. Validation was performed by comparing the simulated energy range from 1 to 11 GV, evaluating the relative differences between the two simulations. Due to the intrinsic differences in the underlying algorithms, it was not possible to use identical random seeds; instead, K independent realizations were generated. The analysis shows that, on average, the differences between the outputs of the two models are consistent with the expected numerical uncertainties for the given statistics, confirming the reliability of COSMICA framework.

To ensure statistical robustness, all simulations were repeated 6 times with independent random seeds and executed on various GPU configurations.

Specifically, benchmarking was conducted on nine distinct GPU setups: 6 different GPU architectures (see Table 3) as well as multi-GPU systems configured with 2, 3, and 4 NVIDIA A100 GPUs. Additionally, in Cavallotto et al. (2025), we benchmarked the legacy HelMod-4/CUDA simulator on NVIDIA A100 systems with 1, 2, 3, and 4 GPUs, following the same protocol as described above.

7.1. Runtime analysis

A primary objective of COSMICA is to enable the rapid execution of large-scale simulations. The majority of development decisions described in Sections 5 and 6 were driven by this goal.

Fig. 3 describes the actual runtimes. A single simulation involving 4096 particles – which yields an error of approximately 1% – requires only about 10 s with COSMICA, in contrast to nearly 100 s with HelMod-4/CUDA. As an exhaustive example, a simulation involving a

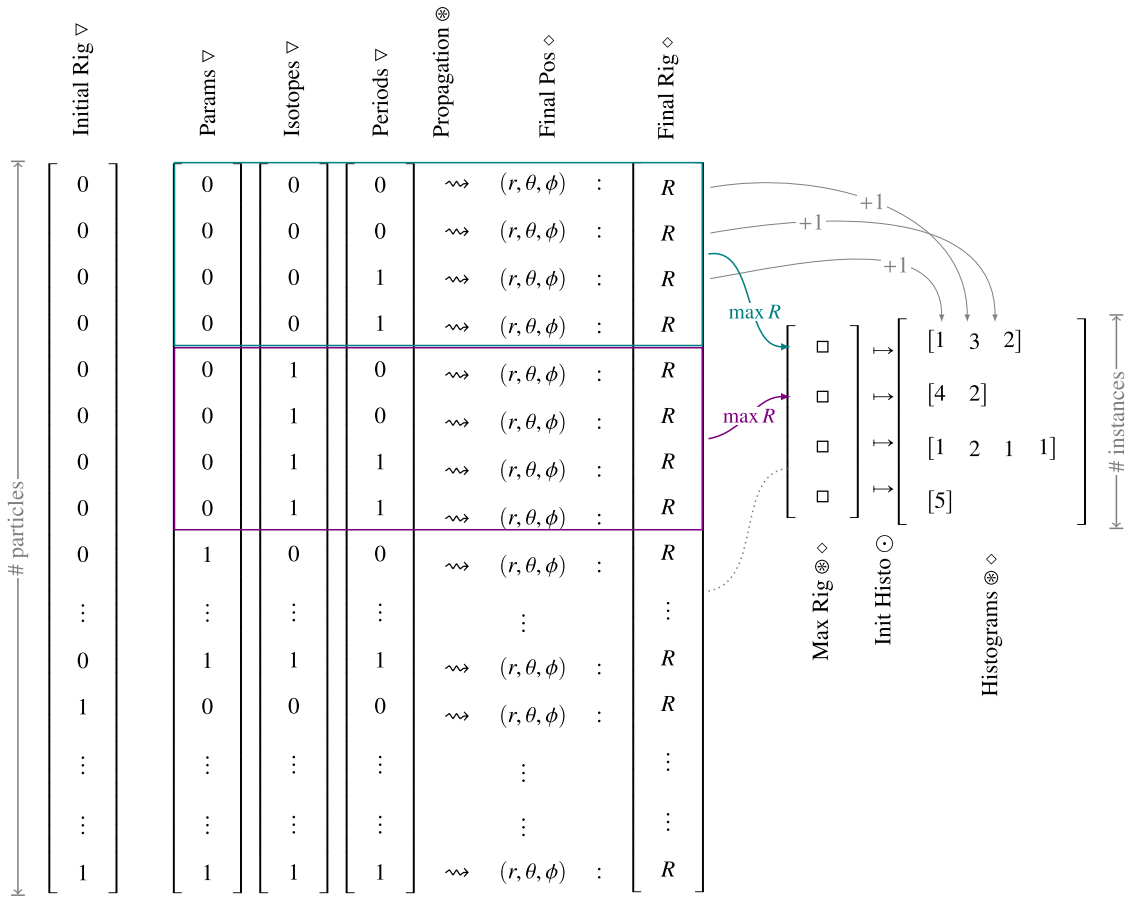


Fig. 2. Schematic overview of the hierarchical particle reordering and workload distribution strategy used in the multi-GPU version of COSMICA. Legend: ▽: index array ⋄: state array ⊛: GPU operation ⊙: CPU operation.

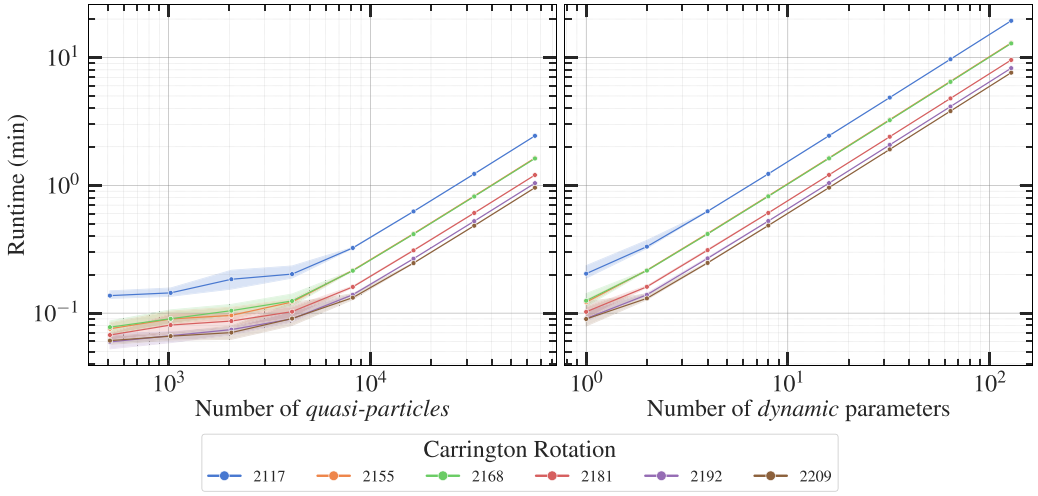


Fig. 3. Scaling analysis of the COSMICA multi-GPU implementation for proton + deuteron simulations (23 bins, 1–11 GV) across different GPU architectures and simulation periods. **(Left)** Runtime as a function of *quasi-particle* count for different periods. **(Right)** Runtime as a function of parameter count (K_0) for different periods. All results are shown for different simulation periods, with shaded areas indicating the 95% percentile interval.

large number of parametrizations that would previously have required approximately one month to complete with HelMod-4/CUDA can now be executed in just over 7 hours using COSMICA (see Cavallotto et al., 2025, for the full comparison).

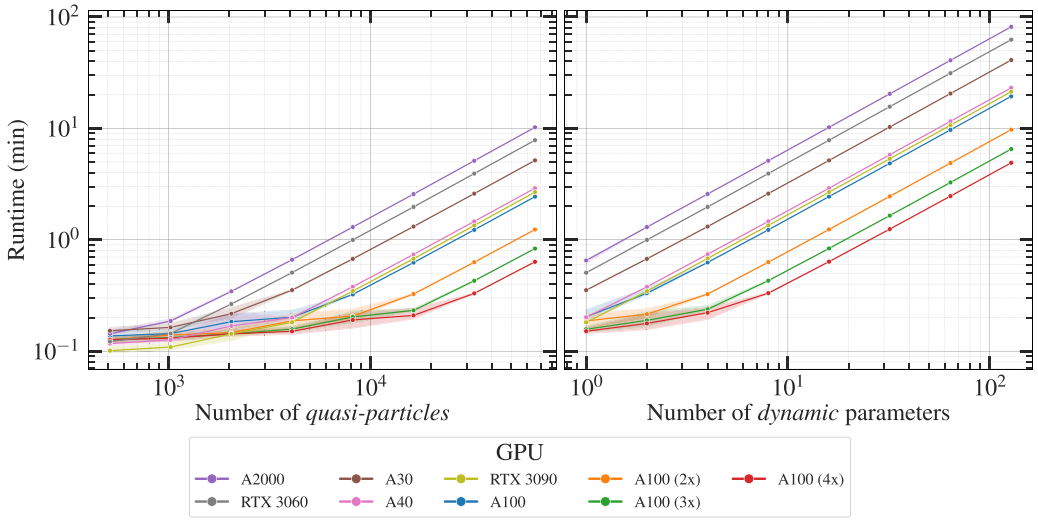


Fig. 4. Runtime scaling for proton + deuteron simulations for CR-2117 (23 bins, 1–10 GV) on different GPU architectures. *Left*: Runtime as a function of *quasi-particle* count. *Right*: Runtime as a function of parameter count (K_0). Results are shown for single and multi-GPU configurations (A100, A2000, A30, A40, RTX 3060, RTX 3090). The shaded area indicates the 95% percentile interval.

7.2. Comparison between GPUs

Fig. 4 presents a comparative analysis of the computational performance on different hardware configurations for the heliospheric parameters corresponding to CR number 2117. Analogue plots for the other CR periods are available in the Supplementary Materials. The left panel explores the scaling regarding the number of *quasi-particles* in each instance for a selected set of model parameters, while the right panel investigates the scaling regarding the number of *dynamic parameters* tested, with a fixed number of particles per instance. For each, the runtime is reported for various GPUs, including single and multi-GPU setups.

We employed both enterprise-grade GPUs (A30, A100, A40, and A2000) and consumer GPUs (RTX 3060 and RTX 3090) in our experiments. The consumer GPUs and the A2000 are installed in desktop computers, whereas the A30 and A40 are located on university servers. The A100 GPUs, instead, are part of the CINECA Leonardo supercomputing cluster (Turisini et al., 2024). For multi-GPU configurations, all GPUs are housed within the same node to maximize inter-device communication efficiency. Notably, Leonardo's A100 GPUs are custom-built and utilize the SXM interface, which offers superior performance compared to standard PCIe connections.

The scaling we see is almost linear in log–log space, indicating a power-law relationship between both independent variables – number of *quasi-particles* or parameters – and running time. The runtime is monotonically increasing with problem size; the slope is a product of computational complexity and the physical constraints of the device. The NVIDIA A100 series has consistently the highest performance, with the multi-GPU configurations (2x, 3x, 4x A100s) achieving a proportional reduction in running time without significant overhead as a result of favorable parallelization. The A100 (4x) configuration has the lowest running time across all problem sizes, indicating extensive scalability for intraday (high throughput) scenarios.

On the other hand, GPUs such as the A2000, A30, and RTX 3060 have a steeper scaling and longer runtimes, meaning the devices have lower memory bandwidth, reduced core count, or architectural limitations. The A40 and RTX 3090 approach the top single-GPU performance levels, defined by the A100, with the RTX 3090 slightly outperforming the A40 and even the A100 at low *quasi-particle* counts.

The plot shown corresponds to a representative period of low solar activity; further results for other periods of activity can be found in the Supplementary Material.

8. Further development and perspective

The main goal of the COSMICA project was to revise the Monte-Carlo SDE numerical implementation, applied to GCR solar modulation propagation problem, applying the most advanced and efficient programming techniques on the GPU. The physical model analyzed has the peculiarity to be compute-bound rather than memory-bound on the GPU. This was achieved by the choice, at design level, to support in actual code analytical or semi analytical description for the PTE terms. For the sake of completeness, it should be noted that a different implementation, allowing for a MHD description of the heliosphere, would invalidate our solution and push toward a more memory-bound execution. That would mandate a different and more tailored algorithm able to limit the amount of memory accesses to the global memory, for instance by leveraging CUDA's shared memory.

Nevertheless, with the current code, it is already possible to design interesting tests for GCR propagation model. Works from Corti et al. (2023) and Della Torre et al. (2025) represent examples of problems that can be issued by COSMICA: in the first case, the correlation among diffusion parameters were explored for a specific Bartel rotation proton spectra measured by AMS-02; the second work explored a single parameter of the PTE but fitting separately each energy bind during five selected Forbush decrease events. Thanks to the COSMICA speedup, we would be able in future works to extend these studies including helium nuclei and other GCR nuclei in the global fitting, as well as increasing the number of parameters to be studied for time dependent correlation effects decreasing, at the same time, the degree of freedom by including more data from different detectors.

9. Conclusions

In this work, we have introduced COSMICA, a novel GPU-accelerated SDE Monte Carlo solver specifically designed for the numerical modeling of GCR transport in the heliosphere. While previous implementations of GPU-enabled transport solvers have successfully demonstrated the feasibility of accelerating SDE techniques, they have often prioritized the exploration of physical scenarios over a systematic optimization of computational performance. Our contribution deliberately shifts this focus by addressing the architectural, algorithmic, and software design aspects that maximize the efficiency of large-scale simulations on modern GPU platforms.

The development of COSMICA relied on several key strategies: (i) adopting a register-conscious kernel design that balances occupancy and throughput while avoiding register spilling; (ii) implementing a hybrid memory management scheme that exploits both constant memory caching and read-only data caches to reduce latencies in parameter-intensive workloads; (iii) utilizing single-precision intrinsics to enhance execution speed while maintaining sufficient numerical accuracy for modulation studies; and (iv) introducing a particle reordering and scheduling mechanism that enables efficient scaling across multiple GPUs. Together, these choices allow COSMICA to sustain high performance on both enterprise-grade and consumer-grade hardware, thereby democratizing access to computationally demanding heliospheric simulations. An important characteristic of COSMICA is that its performance is primarily compute-bound, rather than memory-bound. Consequently, it does not necessitate large amounts of GPU memory, which is often the primary factor influencing the cost of enterprise-grade GPUs. Instead, optimal performances are achieved through a high number of SMs and clock frequencies, making these hardware attributes significantly more relevant for COSMICA.

The benchmarking campaign has demonstrated that COSMICA sensibly reduces the runtime. Simulations that previously required weeks of execution on high-end clusters can now be completed within hours, opening the possibility for systematic parameter scans, sensitivity analyses, and time-dependent modeling across extended solar cycles. Importantly, the scalability of COSMICA across multiple GPUs confirms its potential to support intraday analyses and large-scale ensemble studies that are of increasing relevance for space weather forecasting and the assessment of radiation environments for space missions.

Beyond its immediate computational advantages, COSMICA has been designed with modularity and extensibility in mind. Its architecture allows for the seamless integration of alternative heliospheric models, transport coefficients, and numerical schemes, ensuring that the code will remain adaptable to future scientific challenges. This design philosophy also facilitates its application to a broader class of astrophysical and plasma transport problems beyond cosmic rays, wherever SDE–Monte Carlo methods are appropriate.

In conclusion, COSMICA provides a high-performance, flexible, and open-source framework that bridges the gap between state-of-the-art physics modeling and the computational efficiency required for systematic studies. The work presented here constitutes the first part of a two-paper series: this first installment has outlined the code design, optimization strategies, and performance benchmarks, while the forthcoming companion paper will focus on the validation of COSMICA against benchmark physical models and observational data. Taken together, these contributions establish COSMICA as both a scientifically reliable and computationally powerful tool for advancing our understanding of cosmic-ray modulation in the heliosphere.

CRediT authorship contribution statement

Leone Bacciu: Writing – review & editing, Writing – original draft, Visualization, Software, Formal analysis. **Matteo Grazioso:** Writing – review & editing, Writing – original draft, Visualization, Software. **Giovanni Cavallotto:** Writing – review & editing, Writing – original draft, Visualization, Software. **Stefano Della Torre:** Writing – review & editing, Validation, Supervision, Funding acquisition, Conceptualization. **Massimo Gervasi:** Writing – review & editing, Resources. **Giuseppe La Vacca:** Writing – review & editing, Formal analysis. **Sabina Rossi:** Writing – review & editing, Supervision. **Marco S. Nobile:** Writing – review & editing, Supervision, Software, Project administration, Funding acquisition.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Marco S. Nobile reports financial support was provided by Ca' Foscari University of Venice. If there are other authors, they declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This activity is supported by Fondazione ICSC, Spoke 3 Astrophysics and Cosmos Observations, National Recovery and Resilience Plan (Piano Nazionale di Ripresa e Resilienza, PNRR) Project ID *CN_00000013*. MG, SDT and GLV are supported by INFN and ASI under ASI-INFN Agreement No. 2019-19-HH.0 and its amendments and by ASIF implementation agreement No. 2021–36-HH.0 involving ASI and Milano-Bicocca University.

The project *"SDEGnO – Optimization and performances testing of CUDA-(multi) GPU-accelerated codes for the automatic parameterization of physical models based on SDE Monte Carlo algorithms"* is funded through the *"Bando a Cascata"* issued by Spoke 3 Astrophysics and Cosmos Observations, under the National Recovery and Resilience Plan (*Piano Nazionale di Ripresa e Resilienza*, PNRR, Project ID CN_00000013, Spoke Leader: Istituto Nazionale di Astrofisica – INAF, CUP C53C220003 50006), and is carried out under the Agreement Rep. 531/2024 Prot. 283696/2024 between INAF and Ca' Foscari University of Venice.

This work was also supported by the high-performance computing infrastructure developed under the project "CONVECS", funded by the PR Veneto FESR 2021–2027 program, Priority 1 – Specific Objective 1.1 – Action 1.1.2.

The authors thank Alessandro Miotti for generously providing access to a desktop workstation equipped with an NVIDIA RTX 3090 GPU.

We acknowledge the ICSC for awarding this project access to the EuroHPC supercomputer LEONARDO, hosted by CINECA (Italy).

Appendix A. Supplementary data

Supplementary material related to this article can be found online at doi:10.1016/j.ascom.2025.101043.

Data availability

Data will be made available on request.

References

Aguilar, M., Ali Cavasonza, L., Alpat, B., et al., 2021. Periodicities in the daily proton fluxes from 2011 to 2019 measured by the alpha magnetic spectrometer on the international space station from 1 to 100 GV. Phys. Rev. Lett. 127 (27), 271102. doi:10.1103/PhysRevLett.127.271102.

Bobik, P., Boschini, M.J., Consolandi, C., Torre, S.D., Gervasi, M., Grandi, D., Kudela, K., Vacca, G.L., Mandolesi, N., Rancoita, P.G., Rozza, D., Tacconi, M., 2016. On the forward-backward-in-time approach for Monte Carlo solution of Parker's transport equation: One-dimensional case. J. Geophys. Res.: Space Phys. 121 (5), 3920–3930. doi:10.1002/2015JA022237.

Boschini, M.J., Cavallotto, G., Della Torre, S., Gervasi, M., La Vacca, G., Rancoita, P.G., Tacconi, M., 2024. Fast and accurate evaluation of deep-space galactic cosmic ray fluxes with HelMod-4/CUDA. Adv. Space Res. 74 (9), 4302–4320. doi:10.1016/j.asr.2024.04.021.

Boschini, M.J., Della Torre, S., Gervasi, M., La Vacca, G., Rancoita, P.G., 2018. Propagation of cosmic rays in heliosphere: The HelMod model. Adv. Space Res. 62 (10), 2859–2879. doi:10.1016/j.asr.2017.04.017.

Boschini, M.J., Della Torre, S., Gervasi, M., La Vacca, G., Rancoita, P.G., 2019. The HelMod model in the works for inner and outer heliosphere: From AMS to Voyager probes observations. Adv. Space Res. 64 (12), 2459–2476. doi:10.1016/j.asr.2019.04.007.

Boschini, M.J., Della Torre, S., Gervasi, M., et al., 2017. Solution of heliospheric propagation: Unveiling the local interstellar spectra of cosmic-ray species. Astrophys. J. 840, 115. doi:10.3847/1538-4357/aa6e4f.

Cavallotto, G., Della Torre, S., Bacciu, L., Grazioso, M., Gervasi, M., La Vacca, G., Rossi, S., Nobile, M.S., 2025. Validation of COSMICA code for massive stochastic simulation of cosmic rays propagation in the heliosphere. (submitted to Astronomy and Computing, Submitted to this issue).

Corti, C., Sadowski, P., Nikonov, N., Potgieter, M., Bindi, V., 2023. Constraining the global heliospheric transport of galactic cosmic rays in solar cycles 23 and 24. PoSICRC2023, In: Proceedings of 38th International Cosmic Ray Conference —, vol. 444, p. 1283. doi:10.22323/1.444.1283.

COSMICA Repository, 2025. COSMICA repository. URL https://github.com/ICSC-Spoke3/Cosmica-dev.

Della Torre, S., Cavallotto, G., La Vacca, G., Gervasi, M., 2025. General properties of charged particle diffusion in heliosphere inferred from forbush decreases. Adv. Space Res. doi:10.1016/j.asr.2025.04.053.

Dunzlaff, P., Strauss, R.D., Potgieter, M.S., 2015. Solving Parker's transport equation with stochastic differential equations on GPUs. Comput. Phys. Comm. 192, 156–165. doi:10.1016/j.cpc.2015.03.008.

Dunzlaff, P., Strauss, R., Potgieter, M., 2015. Solving Parker's transport equation with stochastic differential equations on GPUs. Comput. Phys. Comm. 192, 156–165. doi:10.1016/j.cpc.2015.03.008.

Effenberger, F., Fichtner, H., Scherer, K., Büsching, I., 2012. Anisotropic diffusion of Galactic cosmic ray protons and their steady-state Azimuthal distribution. Astron. Astrophys. 547, A120. doi:10.1051/0004-6361/201220203, arXiv:1210.1423.

Engelbrecht, N.E., Effenberger, F., Florinski, V., Potgieter, M.S., Ruffolo, D., Chhiber, R., Usmanov, A.V., Rankin, J.S., Els, P.L., 2022. Theory of cosmic ray transport in the heliosphere. Spac. Sc. Rev. 218 (4), 33. doi:10.1007/s11214-022-00896-1.

Florinski, V., Pogorelov, N.V., 2009. Four-dimensional transport of galactic cosmic rays in the outer heliosphere and heliosheath. Astrophys. J. 701, 642–651. doi:10.1088/0004-637X/701/1/642.

Freidlin, M., 1985. Functional integration and partial differential equations. In: Annals of Mathematics Studies, (No. 109), Princeton University Press.

Gardiner, C.W., 1985. Handbook of Stochastic Methods: For Physics, Chemistry and the Natural Sciences. Springer, doi:10.1007/978-3-662-02452-2.

Gardiner, C., 2009. Stochastic Methods : A Handbook for the Natural and Social Sciences. Springer Berlin, Heidelberg.

Higham, D.J., 2001. An algorithmic introduction to numerical simulation of stochastic differential equations. SIAM Rev. 43 (3), 525–546. doi:10.1137/S0036144500378302.

Hong, S., Kim, H., 2009. An analytical model for a GPU architecture with memory-level and thread-level parallelism awareness. In: Proceedings of the 36th Annual International Symposium on Computer Architecture. pp. 152–163.

Kappl, R., 2016. Solarprop: Charge-sign dependent solar modulation for everyone. Comput. Phys. Comm. 207, 386–399. doi:10.1016/j.cpc.2016.05.025.

Kirk, D.B., Wen-Mei, W.H., 2016. Programming Massively Parallel Processors: A Hands-On Approach. Morgan kaufmann.

Kopp, A., Büsching, I., Strauss, R.D., Potgieter, M.S., 2012. A stochastic differential equation code for multidimensional Fokker-Planck type problems. Comput. Phys. Comm. 183 (3), 530–542. doi:10.1016/j.cpc.2011.11.014.

Kroese, D.P., Taimre, T., Botev, Z.I., 2011. Handbook of Monte Carlo Methods. Wiley, doi:10.1002/9781118014967.

Moloto, K., Engelbrecht, N., Strauss, R., Moeketsi, D., Van den Berg, J., 2019. Numerical integration of stochastic differential equations: A parallel cosmic ray modulation implementation on Africa's fastest computer. Adv. Space Res. 63 (2), 626–639. doi:10.1016/j.asr.2018.08.048.

Moloto, K.D., Engelbrecht, N.E., Strauss, R.D., Moeketsi, D.M., van den Berg, J.P., 2019. Numerical integration of stochastic differential equations: A parallel cosmic ray modulation implementation on Africa's fastest computer. Adv. Space Res. 63 (1), 626–639. doi:10.1016/j.asr.2018.08.048.

NVIDIA, 2017. NVIDIA Tesla V100 GPU architecture. White Paper, Volta Architecture White Paper, WP-08608-001 v1.1.

NVIDIA, 2025. CUDA Math API Reference Manual, Release 13.0. NVIDIA Corporation.

NVIDIA Corporation, 2025a. CUDA C++ Best Practices Guide, Release 12.9 ed. NVIDIA Corporation.

NVIDIA Corporation, 2025b. CUDA C++ Programming Guide, Release 12.9 ed. NVIDIA Corporation.

Øksendal, B., 2010. Stochastic differential equations: An introduction with applications. In: Universitext, Springer Berlin Heidelberg.

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

Pei, C., Bieber, J.W., Burger, R.A., Clem, J., 2010. A general time-dependent stochastic method for solving Parker's transport equation in spherical coordinates. J. Geophys. Res.: Space Phys. 115 (A12), doi:10.1029/2010JA015721.

Qin, G., Shen, Z., 2017. Modulation of galactic cosmic rays in the inner heliosphere, comparing with PAMELA measurements. Astrophys. J. 846 (1), 56. doi:10.3847/1538-4357/aa83ad.

Solanik, M., Bobik, P., Genci, J., 2022. Cosmic rays modulation in heliosphere models on GPU. In: 37th International Cosmic Ray Conference. 12-23 July 2021. Berlin. p. 1320.

Solanik, M., Bobík, P., Genči, J., 2023. Geliosphere - parallel CPU and GPU based models of cosmic ray modulation in the heliosphere. Comput. Phys. Comm. 291, 108847. doi:10.1016/j.cpc.2023.108847.

Strauss, R.D.T., Effenberger, F., 2017. A Hitch-Hiker's guide to stochastic differential equations. Space Sci. Rev. 212 (1–2), 151–192. doi:10.1007/s11214-017-0351-y.

Strauss, R.D.T., Effenberger, F., 2017. A Hitch-Hiker's guide to stochastic differential equations. Solution methods for energetic particle transport in space physics and astrophysics. Spac. Sc. Rev. 212 (1–2), 151–192. doi:10.1007/s11214-017-0351-y, arXiv:1703.06192.

Strauss, R.D., Potgieter, M.S., Büsching, I., Kopp, A., 2011a. Modeling the modulation of galactic and Jovian electrons by stochastic processes. Astrophys. J. 735 (2), 83. doi:10.1088/0004-637X/735/2/83.

Strauss, R.D., Potgieter, M.S., Kopp, A., Büsching, I., 2011b. On the propagation times and energy losses of cosmic rays in the heliosphere. J. Geophys. Res.: Space Phys. 116 (A12), A12105. doi:10.1029/2011JA016831.

Stroustrup, B., 2025. C++20 - cppreference.com. http://en.cppreference.com/w/cpp/20.html. (Accessed: 04 July 2025).

Turisini, M., Cestari, M., Amati, G., 2024. Leonardo. J. Large-Scale Res. Facil. JLSRF 9 (1).

Vogt, A., Engelbrecht, N.E., Strauss, R.D., Heber, B., Kopp, A., Herbst, K., 2020. The residence-time of Jovian electrons in the inner heliosphere. Astron. Astrophys. 642, A170. doi:10.1051/0004-6361/201936897, arXiv:2006.16768.

Zhang, M., 1999. A Markov stochastic process theory of cosmic-ray modulation. Astrophys. J. 513 (1), 409–420. doi:10.1086/306857.

Zhao, L.L., Qin, G., Zhang, M., Heber, B., 2014. Modulation of galactic cosmic rays during the unusual solar minimum between cycles 23 and 24. J. Geophys. Res. (Space Phys.) 119 (3), 1493–1506. doi:10.1002/2013JA019550, arXiv:1310.7076.