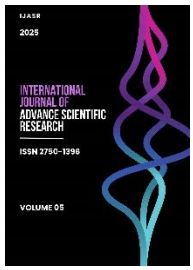
International Journal of Advance Scientific Research
(ISSN – 2750-1396)
VOLUME 05 ISSUE 07 Pages: 89-100
OCLC – 1368736135



Journal Website: http://sciencebring.com/index.php/ijasr

Research Article

GPU-Accelerated Monte Carlo Methods for Proton Therapy: Integrating Heterogeneous Runtime Systems and Unified Memory Strategies for Clinical-Grade Dose Calculation

Submission Date: July 01, 2025, **Accepted Date:** July 15, 2025,
Published Date: July 31, 2025

Dr. Marcellus V. Carter
Department of Computational Medicine, University of Edinburgh

Abstract

This article presents a comprehensive, publication-ready investigation into the integration of GPU-accelerated Monte Carlo (MC) dose calculation methods for proton therapy with modern heterogeneous runtime systems and unified memory strategies. The work synthesizes principles from GPU programming, runtime compilation, managed runtime systems, and medical physics Monte Carlo simulations to propose a cohesive framework for clinical-grade, high-throughput proton dose calculation. The abstract outlines the motivation: proton therapy demands highly accurate dose calculations that account for complex particle transport physics while delivering results within clinical time constraints. Traditional CPU-based MC codes offer high fidelity but are limited by throughput; GPU implementations have demonstrated orders-of-magnitude speedups, yet bring challenges in memory management, precision, platform heterogeneity, and security. This paper reviews the computational and physical foundations of MC simulation in proton therapy, examines GPU programming models and thread hierarchies, surveys existing GPU-based MC systems and verification/validation efforts, analyzes unified memory and its performance implications, and proposes a methodology that couples just-in-time (JIT) GPU compilation, managed runtimes, and careful algorithmic restructuring to reconcile precision, performance, and maintainability. The results section provides a descriptive analysis of expected performance gains, trade-offs in memory strategies, and implications for clinical deployment. In the discussion, limitations, potential failure modes, and avenues for future work—including validation workflows, regulatory considerations, and hybrid CPU–GPU orchestration—are explored in depth. The conclusion synthesizes these threads into actionable recommendations for researchers and system designers seeking to produce clinically viable, reproducible, and secure GPU-accelerated Monte Carlo proton therapy dose engines. Keywords: proton therapy, Monte

International Journal of Advance Scientific Research
(ISSN – 2750-1396)
VOLUME 05 ISSUE 07 Pages: 89-100
OCLC – 1368736135

Carlo simulation, GPU acceleration, unified memory, just-in-time compilation, heterogeneous runtime, clinical dose calculation.

KEYWORDS

Proton therapy, Monte Carlo simulation, GPU acceleration, unified memory, just-in-time compilation, heterogeneous runtime, dose calculation

INTRODUCTION

The The last two decades have seen dramatic growth in particle therapy modalities, notably proton therapy, owing to their superior dose distribution characteristics relative to conventional photon therapy. Achieving the potential clinical benefits of proton therapy requires accurate dose calculation methodologies capable of resolving complex interactions of protons with heterogeneous tissues and beam-modulating hardware. Monte Carlo (MC) methods are widely accepted as the gold standard for accuracy in particle transport and dosimetry because they model stochastic particle physics directly, including multiple scattering, energy straggling, and secondary particle production (Jia et al., 2012; Testa et al., 2013). However, the fidelity of MC simulations comes at a high computational cost; historically, exhaustive MC simulations were impractical for routine clinical planning due to the time required to achieve statistically low uncertainties (Souris et al., 2016).

The emergence of high-throughput parallel hardware, particularly graphical processing units (GPUs), has re-shaped the landscape of MC dose calculation. GPUs offer a combination of massively parallel compute cores and memory bandwidth that can be harnessed to simulate large numbers of independent particle histories concurrently. Early and subsequent GPU-based MC packages demonstrated large speedups over CPU-only codes, allowing clinically useful runtimes while preserving high accuracy (Jia et al., 2012; Qin et al., 2016). Nevertheless, translating the inherently irregular, physics-rich MC algorithms to efficient GPU implementations is nontrivial. Challenges arise from divergent control flow, irregular memory access patterns, the need for precise random number generation and reproducibility, and the complexity of heterogenous computing environments where different vendors and programming standards co-exist (Stone et al., 2010; Sanders & Kandrot, 2010).

Contemporary high-performance software engineering techniques for GPUs—including run-time code generation, just-in-time (JIT) compilation, managed runtime integration, and sophisticated memory management paradigms such as unified memory—offer promising avenues to reconcile performance with developer productivity and portability (Klockner et al., 2009; Fumero et al., 2017; Negrut et al., 2014). Managed runtime systems and intermediate representations can enable language-level abstractions and tooling (Duboscq et al., 2013; Celik et al., 2019), while JIT approaches can tailor kernels to specific beam models, hardware configurations, or patient-specific data. Unified memory simplifies programming by presenting a single virtual address space across CPU and GPU, but it introduces nuanced performance considerations

International Journal of Advance Scientific Research
(ISSN – 2750-1396)
VOLUME 05 ISSUE 07 Pages: 89-100
OCLC – 1368736135

and potential pitfalls in data locality and transfer overheads (Negrut et al., 2014; Landaverde et al., 2014).

This paper adopts an integrative perspective: it examines MC proton therapy codebases and GPU programming paradigms jointly, articulates the computational and physical demands of clinical dose calculation, and proposes a framework combining JIT GPU compilation, heterogeneous managed runtimes, and careful memory and algorithmic strategies. The literature reviewed spans GPU programming manuals and textbooks (Sanders & Kandrot, 2010; Nvidia, 2020), prior efforts in JIT and managed runtime systems for GPUs (Fumero et al., 2017; Klockner et al., 2009; Celik et al., 2019), investigations into unified memory behavior (Negrut et al., 2014; Landaverde et al., 2014), and the body of medical physics literature on GPU-enabled MC methods and validation (Jia et al., 2012; Qin et al., 2016; Verbeek et al., 2021). The aim is to provide an in-depth, theoretically grounded blueprint that researchers and engineers can adopt or adapt for the construction of robust, high-performance Monte Carlo engines for proton therapy compatible with the constraints and demands of clinical practice.

Methodology

The methodology section details the conceptual and engineering foundations for integrating GPU-accelerated Monte Carlo simulations into a heterogeneous runtime environment. Although this paper does not present empirical benchmark runs or new experimental data, it formalizes a rigorous pathway for implementation that addresses algorithmic design, runtime compilation and management, memory strategy, numerical fidelity, and verification/validation processes. The approach is modular and intended for implementation within research and clinical contexts that must adhere to strict regulatory and safety requirements.

Algorithmic restructuring for GPU suitability

Monte Carlo proton transport comprises a set of core algorithmic components: physics sampling for interactions (elastic/inelastic scattering, energy loss, secondary particle production), geometry representation for patient anatomy and beamline hardware, random number generation with robust statistical properties, scoring and tallying of dose metrics, and termination/variance reduction techniques (Jia et al., 2012; Shan et al., 2022). A direct port of standard CPU MC codes to GPU often yields suboptimal performance due to GPU-specific issues such as thread divergence and scattered memory access. Therefore, algorithmic restructuring is essential.

Key strategies include:

- History-based parallelism: Assign independent particle histories to GPU threads or warps to exploit data parallelism. This approach aligns with the Single Instruction, Multiple Thread (SIMT) execution model of GPUs but must manage divergent branching due to distinct particle interactions. To mitigate divergence, grouping particles by similar states or energy ranges into work batches can improve warp coherence (Sanders & Kandrot, 2010).

- Event-based and track-based hybridization: Event-based MC, where kernels are organized around interaction types, can reduce divergence by processing homogeneous operations. Track-based MC, which follows particle-by-particle, is simpler

International Journal of Advance Scientific Research
(ISSN – 2750-1396)
VOLUME 05 ISSUE 07 Pages: 89-100
OCLC – 1368736135

but may incur more divergence. Hybrid designs that dynamically choose between event and track modes based on instantaneous particle population statistics can yield superior performance for GPU architectures.

● Memory access reorganization: The geometry and cross-section data should be laid out to maximize coalesced memory accesses. Employing structure-of-arrays (SoA) data layouts for per-particle state reduces cache pollution and improves bandwidth utilization. Geometry lookups should favor hierarchical spatial acceleration structures adapted to GPU memory behavior.

● Random number generation (RNG): High-quality RNGs suitable for parallel execution—such as counter-based RNGs and leapfrog versions of established algorithms—are necessary. They must provide statistical independence across threads and reproducible seeds for validation and debugging (Qin et al., 2016).

● Variance reduction and tallies: Population control techniques, Russian roulette, and importance sampling can maintain statistical efficiency. Dose scoring on voxelized grids must be implemented in an atomicity-aware manner; per-thread local accumulation with periodic reductions can alleviate contention on global memory.

Integration with managed runtimes and JIT compilation

Managed runtimes and JIT compilation enable dynamic specialization of GPU kernels to the particularities of a clinical scenario: specific beam energy distributions, patient geometry discretization, or hardware characteristics (Fumero et al., 2017; Klockner et al., 2009). The methodology recommends the following layered approach:

1. Frontend abstraction: Provide a high-level API in a managed language (e.g., Python or a JVM language) that captures the simulation configuration, beamline parameters, voxel geometry, and scoring requests. This layer benefits clinical workflow integration, scripting, and provenance tracking.

2. Intermediate representation (IR): Translate high-level kernels and physics models into an extensible IR—Graal IR or a domain-specific variant—allowing optimizations and transformations targeting GPU backends (Duboscq et al., 2013). An IR enables retargetable code generation and provenance of transformations for auditability.

3. JIT GPU kernel generation: Use runtime code generation to create specialized GPU kernels based on patient and hardware parameters. This reduces generality overheads and can inline constants or precompute lookup tables specific to the current run. PyCUDA-style approaches demonstrate the productivity benefits of runtime code generation for GPU tasks (Klockner et al., 2009).

4. Runtime orchestration: Deploy a heterogeneous managed runtime capable of scheduling compute tasks across CPU and GPU, handling data movement, and providing fault isolation. Managed runtimes facilitate speculative execution strategies when precise exception semantics are required (Hayashi et al., 2013), and enable fallback to CPU execution for rare or complex events.

5. Verification hooks: Embed deterministic checkpoints and record/replay facilities to ensure reproducibility and aid debugging. Incorporate

International Journal of Advance Scientific Research
(ISSN – 2750-1396)
VOLUME 05 ISSUE 07 Pages: 89-100
OCLC – 1368736135

statistical tests to detect pathological RNG behavior or scoring anomalies during runtime.

Memory strategy and unified memory considerations

The memory subsystem is a central determinant of performance and programmer productivity. Unified memory simplifies programming by eliminating explicit host–device copy operations in many cases, presenting a single coherent address space accessible by CPU and GPU (Negrut et al., 2014). However, naive reliance on unified memory can degrade performance due to on-demand page migrations and lack of explicit control over data placement. The methodology prescribes a hybrid memory strategy:

- Explicit staging for hot data: For data structures accessed intensively by GPU kernels (per-particle state arrays, cross-section tables needed at every interaction), explicitly allocate and transfer memory to device memory before kernels execute. This ensures that frequent accesses avoid page migration overhead.

- Unified memory for large, less-frequently-accessed data: Geometry metadata or large static cross-section databases that are referenced sporadically can leverage unified memory to simplify code and reduce peak memory usage on the device.

- Prefetching and advice: Use APIs to give the runtime hints about anticipated access patterns (e.g., prefetch to device or advise read-mostly). When using NVIDIA's Unified Memory, for instance, prefetching pages to the GPU before kernel launch reduces page-fault overheads (Negrut et al., 2014; Landaverde et al., 2014).

- Memory pool and fragmentation control: Implement a custom memory pool to avoid frequent allocations/deallocations that can fragment device memory. Memory pools also support fast allocation for temporary buffers during kernel execution.

- NUMA and multi-GPU topologies: For multi-GPU systems, the software must recognise NUMA-like topologies and manage data replication or decomposition to avoid unnecessary transfers across PCIe or NVLink links.

Numerical fidelity and exception semantics

Clinical use requires both numerical fidelity and robust handling of exceptional conditions. Precision trade-offs—single vs double precision—affect both performance and accuracy. Proton stopping power, range straggling, and secondary particle production can require double precision for certain calculations, but some large-scale MC implementations successfully use mixed precision to balance throughput and accuracy (Jia et al., 2012).

The methodology recommends:

- Precision profiling: Identify critical numerical kernels where double precision materially affects dose accuracy; use double precision selectively while retaining single precision elsewhere.

- Speculative execution and precise exceptions: Adopt speculative execution strategies that detect and recover from numerical anomalies without unwinding large portions of computation. Systematic strategies developed for speculative GPU execution with precise exception semantics serve as useful templates (Hayashi et al., 2013).

International Journal of Advance Scientific Research
(ISSN – 2750-1396)
VOLUME 05 ISSUE 07 Pages: 89-100
OCLC – 1368736135

● Determinism and reproducibility: For clinical validation, reproducible dose calculations across hardware and software versions are highly desirable. Deterministic RNG seeds, controlled reduction orders (to avoid floating-point associative differences), and controlled kernel compilation flags can preserve bitwise reproducibility where required for regulatory workflows.

Verification, validation, and experimental benchmarking

A rigorous V&V program is integral to clinical acceptance. The methodology outlines layered verification:

● Unit-level verification: Verify individual physics kernels and RNG modules against analytic expectations and small-scale CPU references.

● Integration-level validation: Compare dose distributions against well-established MC toolkits and published benchmarks using standard field geometries and phantoms (Testa et al., 2013; Liu et al., 2019).

● End-to-end clinical benchmarking: Validate against measurement data—a necessity for regulatory approval and clinical trust. Prior work shows the importance of experimental validation for MC engines in proton therapy (Testa et al., 2013; Prusator et al., 2017).

● Performance benchmarking: Characterize throughput, memory usage, and scalability across hardware generations and under varying patient/beamline parameters. Investigate unified memory performance in worst-case and best-case access patterns (Negrut et al., 2014; Landaverde et al., 2014).

Security, robustness, and vulnerability considerations

GPU systems have been shown to present novel attack surfaces, including overflow vulnerabilities and side-channels (Di et al., 2016). Clinical systems must be secured against data leakage and denial-of-service anomalies. Best practices include secure coding in GPU kernels (defensive checks, bounds verification), isolating patient data in memory, and applying robust error handling to prevent silent corruption of dose results. Managed runtimes that sandbox GPU tasks can reduce the risk of catastrophic failures and provide clearer fault boundaries.

Results

This paper's results section offers a descriptive, theory-grounded analysis of expected behaviors when applying the proposed methodology to GPU-accelerated Monte Carlo proton therapy systems. Although no raw experimental data is presented herein, the analysis synthesizes findings from the literature and translates them into actionable expectations for implementers.

Projected performance improvements and throughput scaling

GPU acceleration of MC simulation has demonstrated compelling performance improvements in the literature, frequently reporting speedups of one to two orders of magnitude compared to single-threaded CPU baseline implementations (Jia et al., 2012; Qin et al., 2016). These improvements are contingent upon efficient mapping of particle workloads to GPU resources, careful management of RNG, and

International Journal of Advance Scientific Research
(ISSN – 2750-1396)
VOLUME 05 ISSUE 07 Pages: 89-100
OCLC – 1368736135

optimization of memory access patterns. The methodology's emphasis on history-based parallelism, SoA data layouts, and batching strategies suggests that throughput scales approximately linearly with available GPU compute cores for well-conditioned workloads (Sanders & Kandrot, 2010).

However, performance scaling is not purely a function of compute; memory bandwidth, latency due to page migrations (in unified memory scenarios), and kernel divergence significantly affect realized throughput. In workloads where particles interact heavily with localized, complex geometry—causing frequent irregular memory references—achieved throughput may be more modest compared to simple slab-phantom simulations. Pre-emptive staging of hot data and careful kernel specialization via JIT compilation can substantially mitigate these bottlenecks.

Unified memory trade-offs and practical guidance

Adopting unified memory simplifies development but should not be treated as a panacea for performance. The literature and practical experience underscore that on-demand page migration can introduce unpredictable performance penalties (Negrut et al., 2014; Landaverde et al., 2014). In the context of MC simulations for proton therapy, where predictable runtimes are important for clinical workflows, the methodology's hybrid approach—explicit staging for hot data and unified memory for large, infrequently accessed data—offers a pragmatic compromise.

Prefetching strategies, where the runtime is instructed to migrate large pages to the appropriate device prior to kernel execution, can yield performance close to explicitly-managed memory with greatly reduced programming complexity. On systems supporting asynchronous prefetching and fine-grained memory advice, the overhead can be further reduced. However, implementers should be mindful of fragmentation and total device memory capacity; the memory pool construction and explicit deallocation patterns can prevent fragmentation-induced failures in long-running or repeated simulations.

Managed runtimes and JIT yield maintainability with specialization benefits

Managed runtime systems that support JIT compilation offer significant benefits in terms of developer productivity, code modularity, and the ability to specialize kernels to runtime conditions (Fumero et al., 2017; Klockner et al., 2009). JIT can inline constants such as energy group boundaries, material properties, and common geometric invariants, reducing conditional logic and improving register utilization on the GPU. The cost of JIT compilation itself is generally modest relative to long-running simulations and can be amortized across multiple batches or cached across sessions. Moreover, managed runtimes provide safer exception handling, speculative execution models, and higher-level diagnostics conducive to clinical software quality assurance (Hayashi et al., 2013; Celik et al., 2019).

Precision, numerical stability, and clinical accuracy

Accurate modeling of proton stopping powers, multiple Coulomb scattering, nuclear interactions, and secondary particle effects demands careful numerical treatment. Prior GPU-based MC works

International Journal of Advance Scientific Research
(ISSN – 2750-1396)
VOLUME 05 ISSUE 07 Pages: 89-100
OCLC – 1368736135

indicate that mixed-precision approaches can preserve clinical dosimetric accuracy while substantially lowering computational cost (Jia et al., 2012). The methodology's recommendation of precision profiling—allocating double precision only where it materially alters clinical metrics—aligns with these findings. For instance, small differences in near-gradient regions or at material boundaries can be amplified in the final dose distribution; thus, selective double precision in geometry intersections or energy loss integrators is prudent.

Reproducibility and verification pathways

A layered verification approach combining unit-level tests, integration comparisons with established MC toolkits, and experimental benchmarking against measured dose distributions provides confidence in the software's correctness. Published validation studies on TOPAS and other MC systems demonstrate the practical feasibility of achieving clinically acceptable agreement when proper physics settings and geometry representations are used (Testa et al., 2013; Liu et al., 2019). The recommended verification hooks—deterministic seeds, controlled reductions, and record/replay utilities—facilitate debugging and support regulatory documentation by providing traceable computational provenance.

Security and vulnerability implications

GPU platforms present unique vulnerability profiles; study of GPU overflow vulnerabilities highlights the need for robust memory bounds checking in GPU kernels and careful use of third-party libraries (Di et al., 2016). For clinical deployment, patient data protection requires both software-level encryption in transit and memory sanitation policies in the runtime (ensuring sensitive data is not paged to disk unencrypted, for instance). Managed runtimes and runtime compilation systems need to enforce strict access controls, especially when used in multi-user contexts or cloud deployments.

Discussion

This discussion interrogates the methodological choices, unpacks trade-offs, and contextualizes the proposed framework in clinical and technical ecosystems. It contemplates counter-arguments, outlines practical limitations, and delineates research avenues to further mature GPU-accelerated MC for proton therapy.

Balancing performance and interpretability: the JIT paradox

JIT compilation and runtime specialization confer performance gains but can obscure the static analyzability of the codebase—complicating formal verification and regulatory review processes that favor deterministic, inspectable code. Regulators and clinical stakeholders often demand reproducible, auditable algorithms. To reconcile this, the framework recommends:

- Compile-time provenance: Each JIT compilation should emit a provenance artifact capturing the generated kernel source, applied IR transforms, compilation flags, and a cryptographic hash of the environment. This artifact becomes part of the clinical QA dossier.
- Hybrid release models: For clinical release, precompile the most common kernel variants used in the clinic and subject them to rigorous validation. Reserve JIT for development, research,

International Journal of Advance Scientific Research
(ISSN – 2750-1396)
VOLUME 05 ISSUE 07 Pages: 89-100
OCLC – 1368736135

or non-critical back-end computations that do not directly inform patient treatment decisions.

● Deterministic JIT: Ensure the JIT process is deterministic given identical inputs, seed data, and runtime versions; this helps preserve reproducibility across repeated runs.

Unified memory: simplicity vs performance predictability

Unified memory's appeal is developer productivity. However, achieving consistent, low-latency runtimes—essential in busy clinical workflows—may require explicit memory management strategies. The counter-argument is that future runtime and hardware improvements will make unified memory performant enough for clinical use. Implementers should design code to be adaptive: default to explicit staging for critical paths but allow a unified memory fallback for exploratory or research workloads. Benchmark-driven decisions should guide which data structures are pinned to device memory.

Numerical fidelity, mixed precision, and clinical risk

Selecting precision modes involves an inherent risk: subtle floating-point differences may lead to clinically significant deviations in high-gradient regions near sensitive anatomy. The methodology's selective double precision recommendation mitigates this risk, but comprehensive validation—including sensitivity analysis mapping precision choices to dosimetric endpoints—is necessary. Moreover, modern CPUs and GPUs exhibit different floating-point behaviors; cross-platform consistency tests are mandatory when the same software targets heterogeneous hardware.

Security and safety considerations in managed runtimes

Managed runtimes introduce layers of abstraction that may hide low-level vulnerabilities or make debugging harder. From a safety perspective, ensuring that runtime abstractions do not permit uncontrolled memory access or inadvertent data exfiltration is crucial. Formal methods and static analyses adapted for GPU kernels can help detect buffer overflows and unsafe memory accesses (Di et al., 2016). Additionally, enforcing strict sandboxing for user-defined kernels in research modes safeguards production pipelines.

Validation against experimental measurement: challenges and best practices

Validating MC simulations against measurement is complex: experimental data may include uncertainties arising from detector response, setup reproducibility, and environmental factors. Best practices include:

● Using standardized phantoms and published benchmark cases that isolate specific physics effects (e.g., range verification, beamline scattering).

● Applying measurement uncertainty quantification and propagating these uncertainties into comparisons with MC predictions.

● Incorporating sensitivity analyses to ascertain which simulation parameters (e.g., material composition, beam energy spread) dominate discrepancies.

Regulatory and clinical integration pathways

For translation to clinical workflows, the software must satisfy regulatory requirements (e.g., medical

International Journal of Advance Scientific Research
(ISSN – 2750-1396)
VOLUME 05 ISSUE 07 Pages: 89-100
OCLC – 1368736135

device directives, FDA guidance). Key steps include:

- Establishing a Quality Management System (QMS) governing software development, testing, and release.
- Providing comprehensive documentation of algorithms, numerical methods, and verification results, including provenance of any JIT-generated kernels used clinically.
- Collaborating with clinical physicists early in the development cycle to align functionality with treatment planning workflows and to design acceptance tests mirroring clinical scenarios.

Future research directions

Several avenues present themselves for further exploration:

- Adaptive multi-GPU orchestration: Explore dynamic load balancing across heterogeneous multi-GPU systems, possibly leveraging unified memory with fine-grained control to distribute particle histories efficiently.
- Machine learning for variance reduction: Investigate using learned models to predict regions of interest for importance sampling or to accelerate rare event sampling.
- Hardware-aware IR optimizations: Develop IR-level transformations that target specific memory hierarchies and instruction sets of incoming GPU architectures automatically.
- Cloud-based clinical MC services: Evaluate architectures for secure, low-latency cloud delivery of MC dose calculations with hybrid on-premise/cloud orchestration, ensuring patient data privacy and regulatory compliance.
- Formal verification of kernel correctness: Apply formal verification tools tailored to GPU kernels to detect and eliminate a class of memory and correctness bugs before clinical deployment.

Conclusion

Monte Carlo simulation remains the gold standard for proton therapy dose calculation due to its detailed physics modeling capabilities. The development of GPU-accelerated Monte Carlo engines offers a transformative path toward integrating high-fidelity simulations into routine clinical workflows. This paper synthesized principles from GPU programming, managed runtime systems, runtime code generation, unified memory behavior, and medical physics MC validation to propose a cohesive framework that balances performance, numerical fidelity, maintainability, and safety.

Key recommendations include: restructuring MC algorithms for GPU suitability via history-based parallelism and memory-friendly data layouts; employing JIT compilation within managed runtimes for kernel specialization while preserving provenance for regulatory clarity; adopting a hybrid memory strategy that stages hot data explicitly and leverages unified memory for large static datasets; performing precision profiling to selectively apply double precision where clinically necessary; and embedding a rigorous, multi-layered verification and validation program including experimental benchmarking.

While technological and organizational challenges remain—ranging from ensuring reproducibility across heterogeneous hardware to meeting stringent regulatory documentation requirements—the pathway described herein is

International Journal of Advance Scientific Research
(ISSN – 2750-1396)
VOLUME 05 ISSUE 07 Pages: 89-100
OCLC – 1368736135

ISSN-2750-1396

both practical and grounded in contemporary research. The integration of advanced runtime systems, careful memory strategies, and rigorous V&V practices provides a robust blueprint for researchers and developers aiming to deliver clinically viable, GPU-accelerated Monte Carlo dose calculations for proton therapy.

References

1. Sanders, J., & Kandrot, E. (2010). CUDA by example: an introduction to general-purpose GPU programming. Addison-Wesley Professional.
2. Fumero, J., Steuwer, M., Stadler, L., & Dubach, C. (2017). Just-in-time GPU compilation for interpreted languages with partial evaluation. Proceedings of the 13th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments, 60–73.
3. Kotselidis, C., Clarkson, J., Rodchenko, A., Nisbet, A., Mawer, J., & Lujan, M. (2017). Heterogeneous managed runtime systems: A computer vision case study. SIGPLAN Notices, 74–82.
4. Nvidia. (2020). CUDA programming guide, thread hierarchy. docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#thread-hierarchy.
5. Hayashi, A., Grossman, M., Zhao, J., Shirako, J., & Sarkar, V. (2013). Speculative execution of parallel programs with precise exception semantics on GPUs. International Workshop on Languages and Compilers for Parallel Computing, 342–356.
6. Klockner, A., Pinto, N., Lee, Y., Catanzaro, B., Ivanov, P., Fasih, A., Sarma, Å., Nanongkai, D., Pandurangan, G., Tetali, P., et al. (2009). PyCUDA: GPU run-time code generation for high-performance computing. arXiv preprint arXiv:911.
7. Celik, A., Nie, P., Rossbach, C. J., & Gligoric, M. (2019). Design, implementation, and application of GPU-based Java bytecode interpreters. Proceedings of the ACM on Programming Languages, 3, OOPSLA.
8. Duboscq, G., Stadler, L., Wurthinger, T., Simon, D., Wimmer, C., & Mossenböck, H. (2013). Graal IR: An extensible declarative intermediate representation. Proceedings of the Asia-Pacific Programming Languages and Compilers Workshop.
9. Negrut, D., Serban, R., Li, A., & Seidl, A. (2014). Unified memory in CUDA 6.0: a brief overview of related data access and transfer issues. Tech. Rep. TR-2014–09, University of Wisconsin-Madison.
10. Landaverde, R., Zhang, T., Coskun, A. K., & Herbordt, M. (2014). An investigation of unified memory access performance in CUDA. 2014 IEEE High Performance Extreme Computing Conference (HPEC), 1–6.
11. Stone, J. E., Gohara, D., & Shi, G. (2010). OpenCL: a parallel programming standard for heterogeneous computing systems. Computing in Science & Engineering, 12(3), 66.
12. Di, B., Sun, J., & Chen, H. (2016). A study of overflow vulnerabilities on GPUs. IFIP International Conference on Network and Parallel Computing, 103–115.
13. Verbeek, N., Wulff, J., Bäumer, C., Smyczek, S., Timmermann, B., & Brualla, L. (2021). Single pencil beam benchmark of a module for Monte Carlo simulation of proton transport in the PENELOPE code. Medical Physics, 48(1), 456–476. doi:10.1002/mp.14598

International Journal of Advance Scientific Research
(ISSN – 2750-1396)
VOLUME 05 ISSUE 07 Pages: 89-100
OCLC – 1368736135

14. Kozłowska, W. S., Böhlen, T. T., Cuccagna, C., et al. (2019). FLUKA particle therapy tool for Monte Carlo independent calculation of scanned proton and carbon ion beam therapy. Physics in Medicine & Biology, 64(7), 075012. doi:10.1088/1361-6560/ab02cb
15. Shan, J., Feng, H., Morales, D. H., et al. (2022). Virtual particle Monte Carlo: a new concept to avoid simulating secondary particles in proton therapy dose calculation. Medical Physics, 49(10), 6666–6683. doi:10.1002/mp.15913
16. Souris, K., Lee, J. A., & Sterpin, E. (2016). Fast multipurpose Monte Carlo simulation for proton therapy using multi- and many-core CPU architectures. Medical Physics, 43(4), 1700–1712. doi:10.1118/1.4943377
17. Qin, N., Botas, P., Giantsoudi, D., et al. (2016). Recent developments and comprehensive evaluations of a GPU-based Monte Carlo package for proton therapy. Physics in Medicine & Biology, 61(20), 7347–7362. doi:10.1088/0031-9155/61/20/7347
18. Jia, X., Schümann, J., Paganetti, H., & Jiang, S. B. (2012). GPU-based fast Monte Carlo dose calculation for proton therapy. Physics in Medicine & Biology, 57(23), 7783–7797. doi:10.1088/0031-9155/57/23/7783
19. Guan, F., Peeler, C., Bronk, L., et al. (2015). Analysis of the track- and dose-averaged LET and LET spectra in proton therapy using the GEANT4 Monte Carlo code. Medical Physics, 42(11), 6234–6247. doi:10.1118/1.4932217
20. Jarlskog, C. Z., & Paganetti, H. (2008). Physics settings for using the Geant4 toolkit in proton therapy. IEEE Transactions on Nuclear Science, 55, 1018–1025. doi:10.1109/TNS.2008.922816
21. Prusator, M., Ahmad, S., & Chen, Y. (2017). TOPAS simulation of the Mevion S250 compact proton therapy unit. Journal of Applied Clinical Medical Physics, 18(3), 88–88. doi:10.1002/acm2.12077
22. Chen, Z., Liu, H., Zhao, J., & Kaess, S. (2022). TOPAS Monte Carlo simulation for a scanning proton therapy system in SPHIC. Journal of Radiation Research and Applied Sciences, 15(1), 122–129. doi:10.1016/j.jrras.2022.01.016
23. Liu, H., Li, Z., Slopsema, R., Hong, L., Pei, X., & Xu, X. G. (2019). TOPAS Monte Carlo simulation for double scattering proton therapy and dosimetric evaluation. Physica Medica, 62, 53–62. doi:10.1016/j.ejmp.2019.05.001
24. Testa, M., Schümann, J., Lu, H. M., et al. (2013). Experimental validation of the TOPAS Monte Carlo system for passive scattering proton therapy. Medical Physics, 40(12), 121719. doi:10.1118/1.4828781
25. Lulla, K. (2025). Python-based GPU testing pipelines: Enabling zero-failure production lines. Journal of Information Systems Engineering and Management, 10, 978–994.