







The Monte Carlo Computational Summit – October 25 & 26, 2023 – Notre Dame, Indiana, USA

Joanna Piper Morgan^{a,b} , Alexander Mote^{a,c} , Samuel Lee Pasmann^{a,d} ,
Gavin Ridley^e, Todd S. Palmer^{a,f}, and Kyle E. Niemeyer^{a,b} 

^aCenter for Exascale Monte Carlo Neutron Transport, Corvallis, OR, USA; ^bSchool of Mechanical, Industrial, and Manufacturing Engineering, Oregon State University, Corvallis, OR, USA; ^cSchool of Electrical Engineering and Computer Science, Oregon State University, Corvallis, OR, USA; ^dDepartment of Aerospace and Mechanical Engineering, University of Notre Dame, Notre Dame, IN, USA; ^eDepartment of Nuclear Science and Engineering, Massachusetts Institute of Technology, Cambridge, MA, USA; ^fSchool of Nuclear Science and Engineering, Oregon State University, Corvallis, OR, USA

ABSTRACT

The Monte Carlo Computational Summit was held on the campus of the University of Notre Dame in South Bend, Indiana, USA on 25–26 October 2023. The goals of the summit were to discuss algorithmic and software alterations required for successfully porting respective code bases to exascale-class computing hardware, compare software engineering techniques used by various code teams, and consider the adoption of industry-standard benchmark problems to better facilitate code-to-code performance comparisons. Participants reported that identifying and implementing suitable Monte Carlo algorithms for GPUs continues to be a sticking point. They also report significant difficulty porting existing algorithms between GPU APIs (specifically Nvidia CUDA to AMD ROCm). To better compare code-to-code performance, participants decided to design a C5G7-like benchmark problem with a defined figure of merit, with the expectation of adding more benchmarks in the future. The participants also identified the need to explore the intermediate and long-term future of the Monte Carlo neutron transport community and how best to modernize and contextualize Monte Carlo as a useful tool in modern industry. Overall the summit was considered to be a success by the organizers and participants, and the group shared a strong desire for future, potentially larger, Monte Carlo summits.

KEYWORDS

Monte Carlo; neutron transport; GPU; HPC; radiation transport; exascale

1. Motivation

The last decade has seen significant efforts to develop Monte Carlo neutral particle transport codes to better utilize accelerated compute platforms (i.e.,

GPUs) (Brantley et al. 2017; Hamilton and Evans 2019; Pereira et al. 2013), which represent the bulk of computational power in modern high-performance computers (HPC) and a significant amount in consumer-grade hardware (Tramm, Yoshii, and Romano 2023). Developing algorithms and methods that can take advantage of GPUs can provide significant speedups to users, allowing for more rapid prototyping and analysis of nuclear systems at ever greater fidelity.

The parallelism paradigm on GPUs is considerably different from that on traditional CPUs, meaning that algorithms that have been optimized for CPU-based performance have no guarantee of being equally efficient on GPUs. In fact, this has been the experience with some code bases: initially seeing slow down when first porting to GPUs (Pozulp et al. 2023). This situation is only further complicated by an API environment that is far more fractured and disparate than the codified one seen on CPUs with shared memory parallelism from OpenMP and distributed memory parallelism from MPI. Different vendors require the use of different proprietary toolchains that often do not translate well to other vendors' hardware. Even support for seemingly basic operability on one vendor's system cannot be guaranteed on other systems. While there is support for open-source standards on multiple vendors' hardware (e.g., OpenCL), the functionality of these libraries often lags critically behind vendors' own APIs. As a result, effectively porting to GPUs can require making major alterations to both underlying algorithms and to software engineering structures of a given code. Different teams at different institutions have made different design choices for both basic algorithms and software engineering structures to achieve the same goal: computing ever more complex problems faster than ever.

As some codes have achieved the speedup promised by GPU-enabled stochastic transport others have had a more difficult time (Pozulp et al. 2023). Each software and algorithmic design choice has its advantages and concessions; while some initially promising results have provided (at best) lackluster performance improvements, others have been shown to be invaluable to Monte Carlo performance on GPUs. Finding a place where the Monte Carlo transport community could have frank discussions about what does and does not work was needed and provided the motivation for the Monte Carlo Computational Summit.

2. Introduction

The Monte Carlo Computational Summit was held on the campus of the University of Notre Dame in South Bend, Indiana on 25–26 October 2023. The goals of the summit were threefold:

1. Discuss algorithmic and software alterations required for successfully porting respective code bases to exascale-class computing hardware with heterogeneous architectures (i.e., systems with both CPUs and GPUs);
2. Compare software engineering techniques used by various code teams and significant roadblocks encountered; and
3. Consider adopting industry-standard benchmark problems to better assess the comparative performance of various codes.

The meeting was purposefully small, with 31 attendees from national laboratories and universities (listed in [Table 1](#)) representing much of the stochastic neutral particle transport community in the United States. The summit was organized by the Center for Exascale Monte Carlo Neutron Transport (CEMeNT) with the support of Alex Long (LANL) and Steven Hamilton (ORNL).

3. Presentations

This section provides a brief overview and key highlights of the eight presentations given during the summit. [Table 2](#) shows the title of each presentation along with the affiliated presenters, institutions, software, and ID listed in the [presentation repository](#) (McClarren and Morgan 2023). [Table 3](#) shows the names of the codes discussed, their availability, distribution license (if any), and primary repository URL if publicly available.

3.1. Advances in the Shift GPU solver

Steven Hamilton, Oak Ridge National Lab

Shift is a continuous-energy CPU and GPU transport code, maintained at Oak Ridge National Laboratory (Pandya et al. 2016). Shift was originally

Table 1. List of participating institutions in alphabetical order and the number of people attending from each.

Institution	Number of Participants
Argonne National Lab (ANL)	4
Dept. of Energy/NNSA	1
Lawrence Livermore National Lab (LLNL)	5
Los Alamos National Lab (LANL)	4
Massachusetts Institute of Technology (MIT)	1
Naval Nuclear Lab (NNL)	2
North Carolina State University (NCSU)*	2
Oak Ridge National Lab (ORNL)	5
Oregon State University (OSU)*	3
Sandia National Lab (SNL)	1
Seattle University (SU)*	1
University of Illinois; Urban-Champagne (UIUC)	1
University of Notre Dame (ND)*	2
Total	31

*denotes CEMeNT university.

Table 2. Presentations given at the 2023 Monte Carlo Summit.

Title	Presenter	Affiliation	Primary Code	ID
Advances in the Shift GPU solver	Steven Hamilton	ORNL	Shift	2
Implicit Monte Carlo at LANL	Alex Long	LANL	Jayenne	3
Progress and Challenges				
MCATK GPU Prototype Code	Tim Burke	LANL	MCATK	N/A [†]
LLNL Monte Carlo Transport	Scott McKinley &	LLNL	Mercury	4
GPU Status	Shawn Dawson			
High Energy Physics Monte Carlo on GPU: <i>Celeritas</i>	Seth Johnson	ORNL	Celeritas	6
MC/DC + Harmonize: JIT Compilation	Ilham Variansyah &	CEMeNT	MC/DC	5a &
& Asynchronous GPU scheduling	Braxton Cuneo	(OSU/SU)		5b
OpenMC at Exascale: Performance Portable	John Tramm	ANL	OpenMC	N/A [†]
Monte Carlo Particle Transport on Intel, NVIDIA, and AMD GPUs				
Experience Adapting OpenMC for CUDA*	Gavin Ridley	MIT	OpenMC	N/A [†]
Monte Carlo Research at NNL	Paul Burke	NNL	MC21	7

*Presenter absent, [†]presentations not made available for inclusion in the repository.

Table 3. Software presented at the Monte Carlo Summit, their availability, licenses, and primary URLs.

Software name	Availability (license, if any)	Primary URL
Shift	Distributed <i>via</i> RSICC	rsicc.ornl.gov
Jayenne	Not publicly distributed	–
MCATK	Not publicly distributed	–
Mercury	Not publicly distributed	–
OpenMC CPU	Open source (specialized)	github.com/openmc-dev/openmc
OpenMC GPU Argonne	Open source (MIT)	github.com/exasmr/openmc
OpenMC GPU MIT	Open source (MIT)	github.com/gridley/openmc/tree/cuda
MC21	Not publicly distributed	–
MC/DC	Open source (BSD-3)	github.com/CEMeNT-PSAAP/MCDC
Harmonize	Open source (BSD-3)	github.com/CEMeNT-PSAAP/harmonize
Celeritas	Open source (Apache 2.0)	github.com/celeritas-project/celeritas

written as a history-based CPU code with a rich set of features for modeling entire reactor systems. Many of these features have been ported to a separate GPU event-based code (Hamilton et al. 2021). Within the event-based scheme, the CPU essentially acts as an event dispatcher, calling events and passing the relevant data to the GPU. This means that the GPU and CPU back-ends share much of the same code, save for some data structure optimization and particle history re-implementation.

Thanks to the minimal code alterations required for GPU implementation, the Shift development team has had success porting many CPU features to the GPU code including:

- On-the-fly Doppler broadening,
- Thermal resonance upscattering,
- Domain decomposition,
- Isotopic depletion,
- Hybrid deterministic methods, and
- General geometry support.

Shift alternates between calling CUDA and the Heterogeneous-Compute Interface for Portability (HIP) for compiling to Nvidia and AMD GPUs, respectively. HIP supports compilation on both Nvidia and AMD GPUs; however, in practice the ability to compile for Nvidia devices seems to be broken more often than it is functional. To avoid splitting up the code base between CUDA and ROCm APIs, the Shift development team uses a Cmake macro to replace relevant function calls. The Shift team has identified that most (but not all) algorithmic optimizations for one GPU vendor will benefit the other, but AMD GPUs seem to be more sensitive to register usage and occupancy. The number of registers that the compiler determines are needed by compute kernels is widely different depending on ROCm versions. This incurs random performance variations between versions of the ROCm compiler stack, meaning there is no guarantee that newer compiler versions will bring better performance, and in fact may result in worse runtimes without careful optimization. The Shift team reports that every time they get a new ROCm version they have to reevaluate register use on a kernel-by-kernel basis, taking up valuable development time. Overall, much more time is spent porting the build than the code itself. Performance comparisons revealed that one Graphics Compute Die (GCD) of an AMD MI250X (which has two GCDs in total) was about equal to the performance of one Tesla V100, which in turn equates to about 150–200 CPU cores.

Going forward, the Shift development team will continue to make optimizations for the existing event-based GPU code and will continue to port more features to their GPU version.

3.2. Implicit Monte Carlo at LANL—progress and challenges

Alex Long, Los Alamos National Laboratory

Jayenne is a C++ implementation of the Implicit Monte Carlo (IMC) method described by Fleck and Cummings (1971) to solve thermal radiation transfer problems. Jayenne is called by the user using a Fortran API as part of a larger hydrodynamics code package (Thompson et al. 2021). The hydrodynamic “host” code passes in the relevant temperatures and opacities. Jayenne was first written at Los Alamos over 25 years ago, where it is still actively maintained and developed. This presentation covered the recent effort to port Jayenne from CPUs to GPUs, which required major refactoring.

The code was moved away from object-oriented design patterns and toward a functional programming design. Several additional changes were made, including removal of virtual and recursive functions and alterations to the “contractor” class which contained many shared pointers—causing

significant slowdown. The Jayenne team focused on porting the most-used features first. Other algorithmic and software optimizations implemented include:

- Simplifying particle data structure to 128 bytes and cell tallies to be only 88 bytes;
- Allocating one particle per thread with particles immediately loaded into shared memory;
- Removing all virtual and recursive functions;
- Implementing various algorithm acceleration techniques after an un-accelerated version (e.g., Discrete Diffusion Monte Carlo (DDMC), Random Walk acceleration); and
- Using Cmake macros for to avoid splitting a code base between Nvidia and ROCm versions.

These optimizations resulted in $9\times$ speedups in IMC on smaller transport-dominated problems and $6\times$ speedups on larger problems. There were more limited speedups on standard user runs ($3.6\times$) where problems are collision dominated and many particles do not even leave a cell. Additionally, speedup above $8\times$ is of limited utility without also porting the host hydrodynamic codes to the GPU.

Going forward, the Jayenne development team will work on porting initialization routines (so more time can be spent in transport kernels where GPU speedup can be found), simplifying required integral computations, and porting more features from the CPU version of the code to GPU.

3.3. MCATK GPU prototype code

Tim Burke, Los Alamos National Laboratory

Monte Carlo Application Toolkit (MCATK) is a Monte Carlo particle transport code from Los Alamos National Lab (Adams et al. 2015). The current GPU strategy involves moving the ray-tracing component of transport onto the GPU through a package called MonteRay (Sweezy 2018). Ray tracing operations are batched then executed in unison on the GPU. In radiography applications (where there is a significant amount of streaming) they report an $80\times$ performance increase. In simulations with few detectors they see a $1.5\text{--}2\times$ increase in performance. MonteRay has single precision capabilities.

A prototype Monte Carlo particle transport code is built from stand-alone feature libraries that can be linked together. Some of these libraries may eventually be open-sourced. The initial port uses a history based algorithm with both multi-group and continuous energy nuclear data and they

see a 10–20 \times performance increase on GPUs (Nvidia Tesla V100 vs. Intel Xeon Broadwell nodes). MCATK developers note that implementing an event-based algorithm may improve this further. In the GPU port, recursion and virtual functions are eliminated. It implements C++ standards 14 and 17 in GPU kernels.

3.4. LLNL Monte Carlo Transport GPU status

Scott McKinley & Shawn Dawson, Lawrence Livermore National Laboratory

Mercury (Brantley et al. 2016) is a feature rich GPU- and CPU-capable particle transport code for neutrons, photons, and light element charged particles. Mercury has a sibling thermal radiative transport code, Imp (Brantley et al. 2019), which uses an implicit Monte Carlo method. Imp shares roughly 80% of its code with Mercury, so advances in hardware implementation provide benefits to both code bases. The LLNL Monte Carlo Transport Project has spent several years of development optimizing Mercury and Imp to meet the computing potential of new, GPU-centric architectures like Sierra and El Capitan. The earliest versions of the GPU code struggled to reach desired computational speedup values (McKinley et al. 2019). Ongoing optimization efforts have resulted in improved GPU performance (Pozulp et al. 2023), although optimization efforts continue. Currently, Mercury and Imp are performant on CPU and Nvidia architectures and are being extended to AMD architectures as El Capitan, an AMD-based machine at LLNL, is being delivered.

This architecture portability is achieved *via* “loop abstraction”, a process where macros are inserted around parallel loops. The source code in the loop is captured as a lambda expression. This lambda may be launched on the GPU with the standard chevron syntax, or on the host CPUs using OpenMP, or even run sequentially. The macro allows the compiler to generate code for all three scenarios. Where to execute is deferred to run time, and is controllable by the end user. This method also offers easy iteration and profiling during development, allowing programmers to simply modify a macro when implementing new ideas. This technique is similar to one used by RAJA (Beckingsale et al. 2019), a portability library developed at LLNL; while the loop abstraction macros may be ported to RAJA in the future, these methods were developed independently.

When porting their code to the GPU, the LLNL Monte Carlo Transport team decided to simply run their existing multiprocessor code on GPUs, use profiling to detect areas of low performance, and then optimize those areas. This method allowed for some developments that are rare among other transport codes, such as their use of managed memory within the GPU. Normally, managed memory allocation calls are considered too slow

to be performant. However, Mercury and Imp use a library called Umpire (Beckingsale et al. 2020) to create a memory pool and amortize the cost of these calls, allowing managed memory to be performant.

Tallying in Mercury and Imp uses atomics. These atomics can cause memory contention, especially in tallies with low memory usage. To mitigate this, tally data structures are replicated such that memory contention does not reduce performance. This often requires full replication on CPUs due to the slower operation of these atomics.

Mercury’s history-based particle tracking method on CPUs performed poorly on GPUs. The code initially used a monolithic kernel that used all registers on the GPU and suffered performance losses due to divergence. So an event-based algorithm was added. This was one of many performance improvements that the development team considered, including:

- A “sidecar” that preserves simulation data between kernel launches,
- Single data structure for mesh variables,
- Asynchronous loops for independent events, and
- Link Time Optimization within Nvidia’s compiler (providing 20–35% speedup on multiple problems).

These improvements led to a $5\times$ speedup on four GPUs compared against 36 CPUs on Sierra.

Development of Mercury and Imp is still ongoing, and there are several challenges the development team is currently facing as they prepare for the MCGIDI, the nuclear data interface used in Mercury is currently being ported to AMD GPUs, and contains many functional structures that are challenging to optimize on GPUs. LLNL’s HPC Toolkit was being used to profile, diagnose, and debug these issues, but the toolkit had problems with MCGIDI, leading to a roadblock in debugging; LLNL’s developers are currently addressing these challenges. The development team has also identified specific areas of difficulty with the port of Mercury and Imp from CUDA to ROCm:

- Link Time Optimization takes far longer with ROCm;
- The initial port of working CUDA code did not work for all GPU kernels with the ROCm compiler. Some kernels receive segmentation faults within the GPU kernel at runtime which are difficult to diagnose;
- NVCC is still used to compile CUDA code, and HIP is only used to compile ROCm code (much like the situation faced by the Shift developers);

- Certain optimization techniques in ROCm seem to create segmentation fault issues in Imp, with individual kernels needing to be compiled without optimization; and

Overall the Mercury and Imp developers report a successful port to Nvidia GPUs using the NVCC compiler seeing a $5\times$ node-to-node speedup over 36-core Intel Xeon type nodes. They will continue their port to other GPU platforms and continue to try to increase performance of their GPU Monte Carlo codes.

3.5. High energy physics Monte Carlo on GPUs: Celeritas

Seth Johnson, Oak Ridge National Laboratory

Celeritas is a Monte Carlo transport code focused on finding a simulation that matches experimental detector output used in the upgraded High Luminosity Large Hadron Collider (HL-LHC) at CERN (Tognini et al. 2022). These simulations compute histories that represent analog physical realizations of a particle moving through the detector region of the HL-LHC, referred to as a “track”. Currently, these simulations use the Geant4 transport code (Agostinelli et al. 2003) to compute these realizations; however, the massive throughput of data from HL-LHC experiments requires accelerated computing on GPUs. Celeritas is designed to achieve this by restructuring the data types and job kernels to run more efficiently on GPUs before passing them to Geant4. HL-LHC simulations are much more geometry intensive than neutron transport because of the high-energy charged particles being simulated, which requires the inclusion of magnetic fields, continuous slowing down, and multiple scattering. The simulations being run for the HL-LHC also require a higher degree of accuracy, at scales from millimeters to tens of meters; these problems combine to make these simulations a challenge to implement on GPU architectures.

Celeritas achieves this through a number of new implementations. Data structures are allocated in chunks on the CPU and passed to the GPU. This allows the GPU compiler to make better memory optimizations for the problem before running the main particle transport loop. This loop is written to handle large batches of tracks with an arbitrary number of kernels, using masking to hide unnecessary kernels on tracks where they are not utilized. This loop runs faster on a per-kernel basis, but sorting overhead reduces this speedup as the number of tracks increases. This also limits Celeritas to only tracking one set of particles at a time, an “event-granular parallelism” that can conflict with the track-granular parallelism expected by Geant4. The complex geometries used in these simulations cause Celeritas to run with far fewer particles on GPU due to

implementation issues the VecGeom library (Wenzel, Apostolakis, and Cosmo 2020). The Oak Ridge Advanced Nested Geometry Engine (ORANGE) has also been considered to alleviate the geometry bottle neck on GPUs.

Despite these issues, Celeritas shows performant speedup when running on GPUs compared to CPUs. On a range of benchmark problems, Celeritas runs 30–40 \times faster on GPUs than CPUs; these results were obtained with both ORANGE and VecGeom, although using ORANGE delivered higher performance metrics. The GPU saturates during the beginning of these simulations due to electromagnetic showering increasing the number of particle tracks, but these saturated steps of the transport loop account for less than 20% of the overall simulation. Celeritas also shows promise running on CPUs, simulating a detector problem about 20% faster than pure Geant4. Electromagnetic physics problems are still difficult to run on GPUs, but show a roughly 2 \times speedup compared to CPU runs. With problem geometry still proving to be a bottleneck on these problems, Oak Ridge National Laboratory and CERN are both working on developing methods for GPU architectures, which will allow these charged geometries to be simulated more quickly. GPU performance is also currently limited by the amount of code running on the GPU as opposed to the host CPU; improving data structures and problem geometry for GPUs will likely lead to larger speedups as a result.

3.6. MC/DC + Harmonize: JIT compilation & asynchronous GPU scheduling

Ilham Variansyah & Braxton Cuneo, Center for Exascale Monte Carlo Neutron Transport

Monte Carlo Dynamic Code (MC/DC) is built and maintained by researchers associated with the Center for Exascale Monte Carlo Neutron Transport (Variansyah et al. 2023). MC/DC is fully open source (under a BSD-3 clause license) and written completely in Python with a compilation structure to support highly performant execution on CPU- and GPU-based architectures. It is written specifically to enable rapid deployment and testing of novel time-dependent transport algorithms and variance reduction techniques on heterogeneous architectures by a diverse user base in an academic setting.

While development is ongoing and some functionality currently has only experimental support, MC/DC has the following features:

- Multigroup physics, including capture, isotropic scattering, and fission (prompt and delayed);

- Continuous energy physics (at room temperature via point-wise data generated from NJOY);
- Geometry tracking via surface-tracking, quadric CSG surfaces, multi-level lattices, and time-dependent planar surfaces;
- Fixed-source (time-dependent) and k -eigenvalue simulation modes;
- CPU parallel support with MPI;
- GPU parallel support with Numba-CUDA (via Harmonize);
- Domain decomposition; and
- Total reproducibility on any architecture in any simulation mode via hash-based Random Number Generator (RNG) seeding.

MC/DC couples a fully transient neutron transport algorithm to a novel (for the field) development scheme. In particle transport the physics kernels are compute kernels. This is different from other simulation types that can often rely on exterior packages to offload most of the compute complexity and handle optimizations for heterogeneous architectures (e.g., PetSC (Balay et al. 2023), MFEM (Anderson et al. 2021)). Thus, development of particle transport Monte Carlo codes has traditionally involved compiled languages due to speed limitations and requirements of heterogeneous architectures and their APIs. This can often make development difficult for users outside of a specific lab or field, or those developing on completely different systems, even if theoretically the same architecture. MC/DC remedies this by using an acceleration and abstraction scheme based on the Numba compiler for Python that supports compilation to CPU and GPU hardware targets (Lam, Pitrou, and Seibert 2015). At runtime these compute kernels are lowered into intermediate representations in the LLVM compiler library and compiled to a specific hardware target. These compute kernels are then bound and executed by a larger Python script. The MC/DC team studied other acceleration and abstraction techniques (e.g., PyKokkos (Al Awar et al. 2021), PyOpenCL, and PyCUDA (Klöckner et al. 2012)) but determined that Numba was the least intrusive and easiest to use, while still providing significant speed-up for Monte Carlo applications (Morgan, Palmer, and Niemeyer 2022).

MC/DC has a full unit, regression, and verification test suite built in and continuously integrated using GitHub Actions. One verification test is a time-dependent Kobayashi dog-leg benchmark problem (Kobayashi, Sugimura, and Nagaya 2001). It uses the seven-group C5G7 cross section data (Hou et al. 2017) and a dog-leg duct filled with water. The initial condition is an isotropic pulse of particles in the highest-energy group. A fissionable region is located in the duct. The thermal flux is tallied in three dimensional space and time. In a strong-scaling analysis, the MC/DC team found that, after a constant 43 s compilation time, the Numba compiler provides a $462\times$ speedup over pure Python mode.

MC/DC is written in pure Python so that the code is extensible and easily interpreted to users, numerical methods developers, and most maintainers. When run in pure Python mode (i.e., no acceleration) MC/DC's kernels can be developed and implemented as in any other Python package. The Numba compiler can be turned on if greater performance is required to run various transport algorithms directly on CPU with compiled binaries that approach compiled code speeds. While this does restrict use of some Python features (e.g., dictionaries, dynamic typing, support for all libraries) the code is still being developed entirely in Python—just a smaller subset of the language. When targeting for GPUs, Python-based functions are compiled to an intermediate representation that can be passed to the Harmonize compute framework for further compilation and execution.

Harmonize (Cuneo and Bailey 2023) is an asynchronous GPU scheduler that can take divergent code bases and dynamically reschedule when kernels get executed such that they binned together. It effectively becomes a wrapper around compute kernels to execute them in a way that decreases thread divergence on streaming architectures, thus increasing performance. From the Monte Carlo transport perspective this can be thought of as on-the-fly event-based scheduling; however, Harmonize can be generally implemented by any GPU program.

Using these acceleration and abstraction techniques, MC/DC has been able to go from nothing to a fully fledged transient continuous energy transport code with completely novel methods written by relatively novice developers in about two years. MC/DC developers have done successful explorations into quasi-Monte Carlo (Variansyah and McClarren 2022b), hybrid iterative techniques for k -eigenvalue simulations (Pasmann et al. 2023a, 2023b, 2024), transient population control techniques (Variansyah and McClarren 2022a), automatic transient weight window production, hash-based random number generation, global uncertainty quantification (Clements et al. 2023), residual Monte Carlo methods, and machine learning techniques for dynamic node scheduling—among other active areas of transient methods exploration. As the benefit of some of these novel algorithms is proven in MC/DC, developers look to implement in other software packages, further pushing transient radiation transport methods into the future.

3.7. OpenMC at exascale: performance portable Monte Carlo particle transport on Intel, Nvidia, and AMD GPUs

John Tramm, Argonne National Laboratory

OpenMC is a Monte Carlo open source full-featured neutron and photon transport code (Romano and Forget 2013). For years, OpenMP threading

has served as OpenMC's shared-memory parallelism model, with MPI available for distributed memory parallelism. Tramm et al. (2022) recently refactored a branch of OpenMC to utilize OpenMP5 target offloading technology. This interface allows for shared-memory multiprocessing on multiple types of CPU and GPU platforms; as such, it is considered a "portable" interface that does not require modification to run on its supported architectures. This portability was the main motivation for the development of OpenMC, as there are many different architectures in use across the U.S. national laboratories, particularly in the realm of GPU programming. Device offloading was a new feature in OpenMP when development for OpenMC began; this was noted as a considerable risk for the development team, as offloading is used extensively in Monte Carlo codes. It was mentioned in discussion that this risk led to disappointing performance levels of several other OpenMP based (non-Monte Carlo) codes, and that OpenMC was fortunate enough to have a direct line of contact to OpenMP developers to help debug issues that arose during development, which aided in overcoming this risk.

In a depleted pin-cell benchmark study OpenMC on CPU performs about the same as other Monte Carlo transport codes on dual socket Intel Xeon Platinum 8260 nodes with 48 cores in total. In that same study on GPU the OpenMC development team reports speedups (as compared to a CPU runtime) of $5.4\times$ on a single Nvidia A100 GPU, $7.0\times$ on a single AMD MI250X GPU, and a $12.8\times$ on a single Intel Ponte Vecchio GPU. Intel GPUs use the OpenMP compiler, which is supported by internal portability codes such as Kokkos and RAJA. RAJA developers mentioned in discussion that, as long as the compiler is mature enough to be publicly deployed, implementations of the same code using SYCL, HIP, and OpenMP backends have shown to be equivalently performant.

These performance metrics were achieved through a number of optimization methods implemented by the development team. Thankfully, very little specialized optimization was required to port the preexisting Monte Carlo code into OpenMP, aside from some by-device optimizations for sorting particles. The major areas of optimization mentioned in this presentation were tallies and cross-section lookups.

Cross-section lookups were by far the most expensive kernel due to non-coherent memory access. The cost of this kernel was reduced with two optimizations. The first was to sort particles before executing the kernel. This sorted lookup results in faster performance because these loaded values are often shared between threads in a single warp. The second was to remove the microscopic cross section cache and perform extra lookups on GPU. This was faster due to the reduction of data required to move between the CPU and GPU as well as on board high bandwidth GPU

memory. However, the microscopic cross-section caching method is still more performant on CPUs, and is still implemented on those architectures. Additionally, the development team found that launching multiple MPI ranks per GPU seemed to saturate the GPU more quickly, leading to significant speed increases; in particular, GPU performance on Intel GPUs is roughly double that of AMD or Nvidia GPUs.

During the development of OpenMC, some computing trends were noted that could inform future development. High-bandwidth memory processors did not seem to improve the performance of OpenMC, and do not seem to be a necessary avenue for future development. It is also worth noting that CPU performance on Monte Carlo simulations seems to double every five years, while GPU performance doubles every two years. When accounting for power consumption, the future seems even more bleak for CPUs; while GPU performance still manages to double every three years, CPU performance is seemingly stagnant. During discussion, it was also mentioned that AI accelerators have shown promising performance boosts; however, they are currently much more difficult to design software for than conventional GPU accelerators, which could lead to trouble as development gets more intricate.

3.8. Experience adapting OpenMC to CUDA

Gavin Ridley, Massachusetts Institute of Technology

In parallel to efforts at Argonne National Laboratory to adapt OpenMC for GPU computation, a smaller thrust at the Massachusetts Institute of Technology focused on a port using the more widely adopted CUDA toolchain (Ridley and Forget 2021). The software development sought to minimize the source code change between the CPU-based code and the GPU-based code, potentially incurring a performance cost. About 50,000 lines of C++ source code comprise the core operations which must be accelerated by GPU.

This work employed the same event-based tracking technique as the OpenMP-focused work, and provided simulation results identical to the CPU and OpenMP GPU code within the usual floating point associativity differences. In contrast, the conceptual modification of the source code was minimized. Because OpenMC extensively expresses abstraction with polymorphic data structures, and CUDA can use polymorphism at some hard-to-quantify performance cost, the code used polymorphism. In addition, a few core STL data structures were replaced with implementations that can compile as CUDA device code. In particular, we created an implementation of `std::unique_ptr` that can maintain separate copies of data on both the host and device in a way that maintains separate virtual tables on each.

A threadsafe implementation of `std::vector` was also written to minimize the required programming effort.

The custom smart pointer implementation was necessitated by the fact that objects exhibiting polymorphism must be constructed on the GPU device. The nuclear data readers in OpenMC run on CPU, so the code copied the nuclear data structures to device first, then executed move constructors called in-place to set up device virtual tables.

In contrast, because the OpenMP port could not support polymorphic design patterns, the source code required substantial reworking. All polymorphic behavior had to be replaced with switch-case statements: an error-prone approach which led to at least one persistent and challenging bug. The OpenMP port also requires extensive manual memory management, whereas the MIT version employed CUDA unified memory to automatically manage memory. At the same time, because the nuclear data memory was not manually distributed, the CUDA code likely exhibited a substantial performance degradation due to a higher frequency of uncoalesced memory accesses. This confounds quantification of the performance degradation experienced due to higher software abstraction in this approach.

We also experimented with a structure-of-array approach rather than array-of-structures for particle data, but found only modest performance gains compared to the array-of-structure approach, around 5% faster calculation rates at best.

After applying the cache-less microscopic cross section trick mentioned in the prior OpenMC section and by Tramm et al. (2022), along with a variety of other optimizations such as trading occupancy for reduced thread divergence, the CUDA-based code achieved a performance about a factor of two worse than the OpenMP code. Given the widespread use of Monte Carlo transport codes, our recommendation for Monte Carlo particle transport code developers going forward is to use flat data structures without polymorphism. Since optimal performance is important here, the extra programmer effort required to avoid software abstraction pays off in the long run.

3.9. Monte Carlo research at NNL

3.9.1. Paul Burke, Naval Nuclear Laboratory

MC21 is the Naval Nuclear Laboratory's primary Monte Carlo code. It is used internally for reactor physics, criticality safety, and shielding analysis and is written primarily as a production design tool (Griesheimer et al. 2015). MC21 has support for:

- Continuous-energy transport;

- Fixed source and eigenvalue calculations;
- Neutron and photon transport capabilities, including coupled transport;
- Flexible constructive solid geometry system with support for hierarchical model construction;
- In-line feedback effects (depletion, T/H, Xenon, eigenvalue search (with movable geometry), neutron and photon heating);
- Generalized tally capability (via user-defined “phase filters” to control which particles are allowed to score each tally);
- In-line decay source; and
- Weight-window variance reduction (with support for coupled radiation simulations).

These features allow MC21 to model problems as geometrically complex as the advanced test reactor with full geometry and physics.

GPU support in MC21 is only available in a developmental fork and parallelism schemes are governed by machines available at NNL. MC21 currently uses both OpenMP and MPI parallelism, primarily targeting Intel Xeon-type machines. Using this tact, simulations have been performed at the peta-scale. When approaching GPU porting, the MC21 development team has focused on minimizing code touched by a port and maintaining compatibility with in-line feedback, searches, and tally systems. The team started by selectively porting only bottle-necking kernels beginning with macroscopic cross section lookups for tracking (Klinvex et al. 2021). Tracking and tallying is done on the CPU, then when a macroscopic cross section is needed that particle is banked and tracking is paused. This continues until a sufficient number of cross section lookup calls is batched then they are executed all at once on the GPU. While mini-app results were promising, when implemented in the full code only a 10% *speedup* in the most-lookup-bound case was found with a 30% *slowdown* for more representative simulations.

Other GPU developments have primarily involved ray tracing capabilities both for MC21 simulations and for visualizations in NNL’s internal visualizer MCVIZ. The two code stacks use the same kernel, code-named Rayzor. Again here the desire is to limit code alterations required, and enable compatibility with the full feature-rich code bases. This is different from other development teams which often allow for divergent code bases and different features for different architecture targets.

In the ray tracing algorithm, Rayzor replaces recursive and polymorphic call stacks with nested for loops *via* a disjunctive normal form expression, decreasing thread divergence. Any arbitrary expression can be converted into disjunctive normal form and typically model components are very close to this form already. Also Rayzor will use a bounding volume

hierarchy scheme as opposed to a constructive solid geometry representation. This is a commonly used algorithm for ray tracing in which the search space is divided in two branches at each level of the tree, with known bounding boxes. This is different from constructive solid geometry as tracking continues through bodies and does not stop at the surface. It also requires component hierarchy where each component with children needs its own bounding volume hierarchy. Generally they have identified that a GPU port requires removal of polymorphism and recursions, a move to *structures of arrays* for data storage restructuring of compilation units.

Going forward the MC21 development team has identified a primary performance milestone for significant user workflow improvement as two orders of magnitude of speedup. Speedups of lesser magnitude represent a quality-of-life improvement but do not disrupt the current workflow dynamics of diffusion analyses during the initial design phase with Monte Carlo results only needed when moving toward production. Developers from the NNSA laboratories (LANL, LLNL, and SNL) discussed a similar paradigm with their users only willing to upgrade to a new version of a particular software if there was at least a $10\times$ speedup, with a $100\times$ speedup to change actual the design process.

The MC21 development team will continue to approach a GPU port by minimizing the amount of code touched and maturing full feature compatibility. This is due to the aforementioned nature of the MC21 use-case (being for reactor design) and the high cost associated with re-verification of models, in addition to novel methods. The primary areas of development include focusing on hybrid acceleration schemes and visualization/mesh homogenization while continuing down a piecemeal-port for GPU offloading, with the primary goal to get users accelerated functionality through smaller, single-use applications.

4. Standardization of test problems

Although these codes are largely designed to solve similar types of problems, the various benchmarks and challenge problems used to report performance make it difficult for users, code designers, and others to reach a consensus when comparing the performance of different codes. For this reason, a discussion was facilitated for standardizing these test problems such that their results could be more easily compared across architectures, code bases, and HPC systems.

Several figures of merit were discussed, including segments per second and particles per second. It was noted that particles per second could work as a standard figure of merit if all codes solved the same benchmark problem with the same features, whereas segments per second could be more

easily compared across disparate benchmarks. However, each figure of merit comes with its own problems; namely, segments per second is difficult to track, especially when additional features like variance reduction and implicit capture are added, while particles per second is not as universal as it seems, with particle weights and splitting handled differently between code bases. Adding a wide range of figures of merit may be the best compromise available, giving different code bases with different features a number of potential points of comparison to use.

Along with deciding on one or more figures of merit, summit members wished to find, or build, a benchmark problem that could be simulated using as many code bases as possible. It quickly became clear that this idea would be difficult due to the different specializations each of these codes pursued—benchmarks that work well on OpenMC may not work at all on Celeritas, for example. As a result, an idea was floated on the other end of the spectrum: a benchmark would instead be created for each type of problem the collective code bases were designed to solve. Several existing repositories for collecting problems and runtime results were posited as a candidate for this idea, including an open-source repository being developed by the American Nuclear Society's Mathematics and Computations Division, and a catalog maintained by ICSBP, although the latter was deemed too complex for use by most codes.

The majority of the challenge problems discussed were neutron transport problems. Some discussion arose around whether the summit members wanted a *benchmark problem*, which measures the correctness of a code base, or a *challenge problem*, which tests the performance of a code base at the limits of its operational capacity. Developers from NNSA labs recommended a Godiva sphere criticality test while those from Oak Ridge proposed a continuous energy reactor transport problem.

By the end of the discussion, the majority of summit members determined a time-dependent continuous-energy form of the C5G7 challenge problem (Hou et al. 2017) to be a suitable candidate for a standardized test problem. A unified figure of merit was identified: average reaction tallies across each pin region of the problem; the problem would be modified by developers at Oak Ridge National Laboratory to allow for this figure of merit to be reported. Additional features, such as multi-group and implicit capture, are also to be added as toggle-able options to the base problem; however, the base problem was chosen as the standard for all Monte Carlo codes for the reporting of comparable figures of merit.

5. Outlook

The major takeaways from this summit were concerned with collaboration. It is clear from the efforts made by the lab members present that GPU

development still represents a formidable but necessary challenge, especially with upcoming AMD and Intel architectures present in the exascale-class supercomputers of the future. The use of multiple co-functional code bases among the national labs and academia remains a useful tool for comparing problem-solving methodologies, as well as verifying and validating results. To aid in these comparisons, a more standard set of generalized benchmark problems should be produced and distributed with the performance results open and available. Additionally, a discussion arose around the future of Monte Carlo and its place in the national labs, as well as public and private industry development. Those present at the summit agreed that the Monte Carlo development community, and perhaps the neutronics community at large, should develop a report projecting how these codes can continue to develop in the near future, and how best to modernize and contextualize Monte Carlo as a useful tool in modern industry.

Moreover, the environment created by the summit was noted by all attendees to be productive and refreshing. Multiple summit members voiced a desire for more events like this in the future, where developers can share their advancements and insights through presentations and discussion rather than publications. Attendees were also glad to receive immediate feedback from one another, and there was a good deal of useful ideation and deliberation over both days of the summit. Overall, both the organizers and guests came away from the event feeling it was a great success. These successes have led members of CEMeNT to consider making the Monte Carlo Computational Summit a recurring event and expanding to include other developers outside of the Center and national labs.

Acknowledgments

We would like to thank the attendees, presenters, and organizers including co-chairs Alex Long and Steven Hamilton. We would also like to thank the University of Notre Dame and ND Energy for hosting the Monte Carlo computational summit. Joanna Piper Morgan has previously had a co-op position at Advanced Micro Devices (AMD) and previously interned at Los Alamos National Laboratory where she worked on MCATK. All other authors declare no conflicts of interest.

Disclosure statement

No potential conflict of interest was reported by the author(s).

Funding

This work was supported by the Center for Exascale Monte-Carlo Neutron Transport (CEMeNT) a PSAAP-III project funded by the Department of Energy, grant number: DE-NA003967.

ORCID

Joanna Piper Morgan  <http://orcid.org/0000-0003-1379-5431>

Alexander Mote  <http://orcid.org/0000-0001-5099-0223>

Samuel Lee Pasmann  <http://orcid.org/0000-0003-1391-1471>

Kyle E. Niemeyer  <http://orcid.org/0000-0003-4425-7097>

References

- Adams, T., S. Nolen, J. Sweezy, A. Zukaitis, J. Campbell, T. Goorley, S. Greene, and R. Aulwes. 2015. Monte Carlo application toolkit (MCATK). *Ann. Nucl. Energy*. 82:41–7. doi:[10.1016/j.anucene.2014.08.047](https://doi.org/10.1016/j.anucene.2014.08.047).
- Agostinelli, S., J. Allison, K. Amako, J. Apostolakis, H. Araujo, P. Arce, M. Asai, D. Axen, S. Banerjee, G. Barrand, et al. 2003. Geant4—A simulation toolkit. *Nucl. Instrum. Methods Phys. Res., Sect. A*. 506 (3):250–303. doi:[10.1016/S0168-9002\(03\)01368-8](https://doi.org/10.1016/S0168-9002(03)01368-8).
- Al Awar, N., S. Zhu, G. Biros, and M. Gligoric. 2021. A performance portability framework for Python. *Int. Conf. Supercomput.* 467–78. ACM. doi:[10.1145/3447818.3460376](https://doi.org/10.1145/3447818.3460376)
- Anderson, R., J. Andrej, A. Barker, J. Bramwell, J.-S. Camier, J. Cervený, V. Dobrev, Y. Dudouit, A. Fisher, T. Kolev, et al. 2021. MFEM: A modular finite element methods library. *Comput. Math. Appl.* 81:42–74. doi:[10.1016/j.camwa.2020.06.009](https://doi.org/10.1016/j.camwa.2020.06.009).
- Balay, S., S. Abhyankar, M. F. Adams, S. Benson, J. Brown, P. Brune, K. Buschelman, et al. 2023. PETSc/TAO Users Manual. Technical Report ANL-21/39 – Revision 3.20. Argonne National Laboratory.
- Beckingsale, A., M. J. McFadden, J. P. S. Dahm, R. Pankajakshan, and R. D. Hornung. 2020. Umpire: Application-focused management and coordination of complex hierarchical memory. *IBM J. Res. Dev.* 64 (3/4):00:1–00:10. doi:[10.1147/JRD.2019.2954403](https://doi.org/10.1147/JRD.2019.2954403)
- Beckingsale, D. A., J. Burmark, R. Hornung, H. Jones, W. Killian, O. Pearce, A. J. Kunen, P. Robinson, B. S. Ryujin, and T. R. W. Scogland. 2019. RAJA: Portable performance for large-scale scientific applications. In *IEEE/ACM international workshop on performance, portability and productivity in HPC (P3HPC)*, 71–81. Denver, CO, IEEE/ACM.
- Brantley, P. S., R. C. Bleile, S. A. Dawson, N. A. Gentile, M. S. McKinley, M. J. O. Brien, M. M. Pozulp, et al. 2017. LLNL Monte Carlo transport research efforts for advanced computing architectures. Technical Report. Lawrence Livermore National Lab (LLNL), Livermore, CA (United States).
- Brantley, P., N. Gentile, M. Lambert, M. McKinley, M. O'Brien, and J. Walsh. 2019. A new implicit Monte Carlo thermal photon transport capability developed using shared Monte Carlo infrastructure. In *Proceedings of the American Nuclear Society; Mathematics and Computation (M&C) 2019*, Portland, Oregon, USA.
- Brantley, P., S. Dawson, S. McKinley, O. Matt, D. Peters, M. Pozulp, G. Becker, K. Mohror, and A. Moody. 2016. LLNL Mercury Project Trinity Open Science Final Report. Technical Report. Lawrence Livermore National Lab. <https://www.osti.gov/biblio/1252640>.
- Clements, K., G. Geraci, A. J. Olson, and T. Palmer. 2023. Global sensitivity analysis in Monte Carlo radiation transport. In *International Conference on Mathematics and Computational Methods Applied to Nuclear Science and Engineering*, Niagara Falls, Ontario, Canada.
- Cuneo, B., and M. Bailey. 2023. Divergence reduction in Monte Carlo neutron transport with on-GPU asynchronous scheduling. *ACM Trans. Model. Comput. Simul.* 34 (1):1–25. doi:[10.1145/3626957](https://doi.org/10.1145/3626957).

- Fleck, J. A., and J. D. Cummings. 1971. An implicit Monte Carlo scheme for calculating time and frequency dependent nonlinear radiation transport. *Comput. Phys.* 8 (3):313–42. doi:[10.1016/0021-9991\(71\)90015-5](https://doi.org/10.1016/0021-9991(71)90015-5).
- Griesheimer, D. P., D. F. Gill, B. R. Nease, T. M. Sutton, M. H. Stedry, P. S. Dobreff, D. C. Carpenter, T. H. Trumbull, E. Caro, H. Joo, et al. 2015. MC21 v.6.0 – A continuous-energy Monte Carlo particle transport code with integrated reactor feedback capabilities. *Ann. Nucl. Energy.* 82:29–40. doi:[10.1016/j.anucene.2014.08.020](https://doi.org/10.1016/j.anucene.2014.08.020)
- Hamilton, S. P., and T. M. Evans. 2019. Continuous-energy Monte Carlo neutron transport on GPUs in the Shift code. *Ann. Nucl. Energy.* 128:236–47. doi:[10.1016/j.anucene.2019.01.012](https://doi.org/10.1016/j.anucene.2019.01.012).
- Hamilton, S. P., T. M. Evans, K. E. Royston, and E. D. Biondo. 2021. Domain decomposition in the GPU-accelerated Shift Monte Carlo code. *Ann. Nucl. Energy.* 166 (1):108687. doi:[10.1016/j.anucene.2021.108687](https://doi.org/10.1016/j.anucene.2021.108687)
- Hou, J. J., K. N. Ivanov, V. F. Boyarinov, and P. A. Fomichenko. 2017. OECD/NEA benchmark for time-dependent neutron transport calculations without spatial homogenization. *Nucl. Eng. Des.* 317:177–89. doi:[10.1016/j.nucengdes.2017.02.008](https://doi.org/10.1016/j.nucengdes.2017.02.008)
- Klinvex, A., K. E. Remley, N. Petrillo, D. P. Griesheimer, P. E. Burke, and A. Bird. 2021. An asynchronous GPU-enabled cross-section lookup algorithm for Monte Carlo transport simulations. In *Proceedings of the American Nuclear Society; Mathematics and Computation (M&C) 2021*, Raleigh, North Carolina.
- Klöckner, A., N. Pinto, Y. Lee, B. Catanzaro, P. Ivanov, and A. Fasih. 2012. PyCUDA and PyOpenCL: A scripting-based approach to GPU run-time code generation. *Parallel Comput.* 38 (3):157–74. doi:[10.1016/j.parco.2011.09.001](https://doi.org/10.1016/j.parco.2011.09.001).
- Kobayashi, K., N. Sugimura, and Y. Nagaya. 2001. 3D radiation transport benchmark problems and results for simple geometries with void region. *Prog. Nucl. Energy.* 39 (2):119–44. www.elsevier.com/locate/pnucene. doi:[10.1016/S0149-1970\(01\)00007-5](https://doi.org/10.1016/S0149-1970(01)00007-5)
- Lam, S. K., A. Pitrou, and S. Seibert. 2015. Numba: A LLVM-based Python JIT compiler. In *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, Austin, TX, ACM. <https://dl.acm.org/doi/10.1145/2833157.2833162>.
- McClarren, R., and J. P. Morgan. 2023. 2023 Monte Carlo Computational Summit. doi:[10.5281/zenodo.10207764](https://doi.org/10.5281/zenodo.10207764).
- McKinley, M., R. Bleile, P. Brantley, S. Dawson, M. O'Brien, M. Pozulp, and D. Richards. 2019. Status of LLNL Monte Carlo transport codes on Sierra GPUs. In *Proceedings of the American Nuclear Society; Mathematics and Computation (M&C) 2019*, Portland, Oregon, USA.
- Morgan, J. P., T. S. Palmer, and K. E. Niemeyer. 2022. Explorations of Python-based automatic hardware code generation for neutron transport applications. In *Transactions of the American Nuclear Society*, vol. 126, 318–20. Anaheim, CA, USA: American Nuclear Society. doi:[10.13182/T126-38137](https://doi.org/10.13182/T126-38137).
- Pandya, T. M., S. R. Johnson, T. M. Evans, G. G. Davidson, S. P. Hamilton, and A. T. Godfrey. 2016. Implementation, capabilities, and benchmarking of Shift, a massively parallel Monte Carlo radiation transport code. *Comput. Phys.* 308:239–72. doi:[10.1016/j.jcp.2015.12.037](https://doi.org/10.1016/j.jcp.2015.12.037)
- Pasmann, S., I. Variansyah, C. T. Kelley, and R. G. McClarren. 2024. Mitigating spatial error in the iterative-Quasi-Monte Carlo (iQMC) method for neutron transport simulations with linear discontinuous source tilting and effective scattering and fission rate tallies. <http://arxiv.org/abs/2401.04029>.
- Pasmann, S., I. Variansyah, C. T. Kelley, and R. McClarren. 2023a. iQMC: Iterative quasi-Monte Carlo with Krylov linear solvers for k-eigenvalue neutron transport simulations.

- In *International Conference on Mathematics and Computational Methods Applied to Nuclear Science and Engineering*, Niagara Falls, Ontario, Canada. doi:[10.1080/00295639.2022.2143704](https://doi.org/10.1080/00295639.2022.2143704)
- Pasman, S., I. Variansyah, C. T. Kelley, and R. McClarren. 2023b. A quasi-Monte Carlo method with Krylov linear solvers for multigroup neutron transport simulations. *Nucl. Sci. Eng.* 197 (6):1159–73. doi:[10.1080/00295639.2022.2143704](https://doi.org/10.1080/00295639.2022.2143704).
- Pereira, C. M. N. A., A. C. A. Mól, A. Heimlich, S. R. S. Moraes, and P. Resende. 2013. Development and performance analysis of a parallel Monte Carlo neutron transport simulation program for GPU-cluster using MPI and CUDA technologies. *Progress in Nuclear Energy (New Series)* 65:88–94. doi:[10.1016/j.pnucene.2013.02.008](https://doi.org/10.1016/j.pnucene.2013.02.008).
- Pozulp, M., R. Bleile, P. Brantley, S. Dawson, M. McKinley, A. Robinsonand, M. Yang, and M. O'Brien. 2023. Progress porting LLNL Monte Carlo transport codes to Nvidia GPUs. In *International Conference on Mathematics and Computational Methods Applied to Nuclear Science and Engineering*, Niagara Falls, Ontario, Canada.
- Ridley, G., and B. Forget. 2021. Design and optimization of GPU capabilities in OpenMC. In *ANS Winter Conference*, Washington D.C.
- Romano, P. K., and B. Forget. 2013. The OpenMC Monte Carlo particle transport code. *Ann. Nucl. Energy.* 51 (C):274–81. doi:[10.1016/j.anucene.2012.06.040](https://doi.org/10.1016/j.anucene.2012.06.040)
- Sweezy, J. E. 2018. A Monte Carlo volumetric-ray-casting estimator for global fluence tallies on GPUs. *Comput. Phys.* 372:426–45. doi:[10.1016/j.jcp.2018.06.032](https://doi.org/10.1016/j.jcp.2018.06.032).
- Thompson, K. G., M. A. Cleveland, A. R. Long, K. P. Long, R. T. Wollaeger, B. R. Ryan, and T. M. Kelley. 2021. Jayenne physics manual, revision 1.0 – An implicit Monte Carlo code for thermal radiative transfer. Technical Report. Los Alamos National Lab. (LANL), Los Alamos, NM.
- Tognini, S. C., P. Canal, T. M. Evans, G. Lima, A. L. Lund, S. R. Johnson, S. Y. Jun, V. R. Pascuzzi, and P. K. Romano. 2022. Celeritas: GPU-accelerated particle transport for detector simulation in High Energy Physics experiments. Technical Report. Fermi National Accelerator Facility, Oak Ridge National Lab. doi:[10.48550/arXiv.2203.09467](https://doi.org/10.48550/arXiv.2203.09467).
- Tramm, J. R., K. Yoshii, and P. K. Romano. 2023. Power at your fingertips: Assessing the performance of a Monte Carlo neutron transport mini-app on consumer laptop GPUs. In *The International Conference on Mathematics and Computational Methods Applied to Nuclear Science and Engineering (M&C 2023)*, Niagara Falls, Ontario, Canada.
- Tramm, J., P. Romano, J. Doerfert, A. Lund, P. Shriwise, A. Siegel, G. Ridley, and A. Pastrello. 2022. Toward portable GPU acceleration of the OpenMC Monte Carlo particle transport code. In *Proceedings of the international conference on physics of reactors – PHYSOR*, 2734–2743. Pittsburgh, PA.
- Variansyah, I., and R. G. McClarren. 2022a. Analysis of population control techniques for time-dependent and eigenvalue Monte Carlo neutron transport calculations. *Nucl. Sci. Eng.* 196 (11):1280–305. doi:[10.1080/00295639.2022.2091906](https://doi.org/10.1080/00295639.2022.2091906).
- Variansyah, I., and R. G. McClarren. 2022b. An effective initial particle sampling technique for Monte Carlo reactor transient simulations. In *International Conference on Physics of Reactors*, Pittsburgh, Pennsylvania, USA.
- Variansyah, I., J. P. Morgan, K. E. Niemeyer, and R. G. McClarren. 2023. Development of MC/DC: A performant, scalable, and portable Python-based Monte Carlo neutron transport code. In *International Conference on Mathematics and Computational Methods Applied to Nuclear Science and Engineering*, Niagara Falls, Ontario, Canada.
- Wenzel, S., J. Apostolakis, and G. Cosmo. 2020. A VecGeom navigator plugin for Geant4. *EPJ Web Conf.* 245:02024. doi:[10.1051/epjconf/202024502024](https://doi.org/10.1051/epjconf/202024502024).