LLNL-CONF-2000813

Lawrence Livermore National Laboratory

Progress Porting LLNL Monte Carlo Transport Codes to the AMD Instinct™ MI300A APU

International Conference on Mathematics and Computational Methods Applied to Nuclear Science and Engineering (M&C 2025)

April 27-30, 2025

Disclaimer

LLNL-CONF-2000813

Progress Porting LLNL Monte Carlo Transport Codes to the AMD Instinct™ MI300A APU

Shawn A. Dawson*, Ryan C. Bleile, Patrick S. Brantley, Evan S. Gonzalez, M. Scott McKinley, Matthew J. O'Brien, Mike M. Pozulp, Alex P. Robinson, Max Yang

Lawrence Livermore National Laboratory, Livermore, CA

[leave space for DOI, which will be inserted by ANS]

ABSTRACT

The Lawrence Livermore National Laboratory Monte Carlo Transport Project has extended a prior port of production code from Nvidia GPUs on the Sierra computer to AMD GPUs on the El Capitan computer. El Capitan is equipped with AMD Instinct™ MI300A APUs. The port included use of the HIP language extension for El Capitan rather than the prior CUDA extension. Obstacles related to generating robust code were encountered and many issues resolved. Additionally, run time performance issues have been encountered, many of which have also been resolved in collaboration with the vendor and tool chain developers. Monte Carlo speedups are 2.5x for neutron transport and 2.3x for thermal photon transport codes when comparing one node of El Capitan to one node of Sierra. When comparing El Capitan to a large-scale Xeon based CPU computer, Monte Carlo speedups are 3.1x for neutron transport and 3.0x for thermal photon transport. Additionally, for a less idealized, four node simulation, with domain decomposition and streaming particles, El Capitan shows a speedup of over 3.1x when compared to Sierra and 4.3x compared to a Xeon computer.

Keywords: Monte Carlo, LLNL, El Capitan, HIP, MI300

1. INTRODUCTION

Lawrence Livermore National Laboratory (LLNL) will deploy the fourth Advanced Technology System (ATS) computer named El Capitan (ATS-4) in 2025 [1]. El Capitan will utilize AMD Instinct™ MI300A APUs (Accelerated Processing Unit) [2]. LLNL codes which were previously ported to the Sierra (ATS-2) computer which utilized Nvidia V100 GPUs (Graphics Processing Unit) are now being ported to the MI300A APUs. The LLNL Monte Carlo Transport Project develops two production codes: Mercury, a Monte Carlo (MC) particle transport code [3], and Imp, an implicit Monte Carlo (IMC) [4] thermal photon transport code [5]. These codes are ported to distributed and shared memory CPU architectures using MPI and OpenMP for the host CPUs and to GPUs. The port to ATS-2 Nvidia GPUs utilizing the CUDA language extension was presented at the M&C 2023 conference [6]. The MI300A APUs are similar to GPUs from a software development perspective but require the use of Heterogenous-compute Interface for Portability (HIP) aware compilers rather than CUDA. In addition to the use of HIP, the port required modifications to the memory model and code enhancements necessary for robustness and performance.

*dawson6@llnl.gov

The goal of the work presented in this paper is to run calculations on ATS-4 and compare node to node performance relative to ATS-2 as well as to a current Commodity Technology System-2 (CTS-2) machine. We previously reported [6] the performance improvements obtained on ATS-2 compared to an older CTS-1 computing platform; we no longer use CTS-1 for performance comparisons. As both ATS-2 and CTS-2 machines are in daily use at LLNL, these provide the baseline to which we will compare the performance of the Monte Carlo transport codes on El Capitan (ATS-4).

Enabling Monte Carlo transport codes on GPU architectures is currently an active research area. Two recent examples include porting the Shift Monte Carlo code to Nvidia GPUs [8] and the OpenMC Monte Carlo code to different GPU architectures [9]. Pozulp et al. [6] reported on the porting of the Mercury and Imp Monte Carlo codes to Nvidia GPUs, demonstrating speedups compared to traditional CPU based clusters.

The remainder of this paper is organized as follows. In Sec. 2, we describe the architecture and software stack for CTS-2, ATS-2 and ATS-4. In Sec. 3, we describe the high-level code design of the LLNL Monte Carlo codes and general modifications we have made for the initial ATS-4 port. We then describe in Sec. 4 code modifications we have made to improve the robustness of compiled code produced by the compiler toolchain. In Sec. 5, we describe investigations and changes in the Monte Carlo codes aimed at improving performance on the AMD MI300A architecture. We then present particle processing throughput performance assessments for three test problems in Sec. 6. We conclude and offer suggestions for future work in Sec. 7.

2. CTS-2, ATS-2 and ATS-4 HARDWARE AND SOFTWARE STACK

CTS-2 – Each node of CTS-2 is equipped with two 56-core Intel Xeon Platinum 8480+ processors and 256 GB memory. These are 4th generation Intel Xeon CPUs. The programming model is MPI for both intra-node and inter-node communication. OpenMP threads may be used on node to reduce the MPI ranks and allow shared memory usage across threads.

ATS-2 – Each node of ATS-2 is equipped with four Nvidia V100 GPUs and two IBM Power9 CPUs each with 22 cores. There is 256 GB memory available to the CPUs, and 16 GB of high bandwidth memory (HBM2) memory for each of the GPUs. The memory is separate between the CPU and GPU. The programming model is MPI and OpenMP for the CPUs and the Nvidia language extension CUDA to run kernels and move data to and from the GPU.

ATS-4 – Each node of ATS-4 is equipped with four AMD Instinct^{TM} MI300A APUs. The MI300 APU is a multi-chiplet design containing six GPU chiplets and three CPU chiplets. The six GPU chiplets are presented as a single GPU. The three CPU chiplets appear as 24 x86-64 cores. There is 128 GB of high bandwidth memory (HBM3) shared between the CPU and GPU chiplets, for a total of 96 CPU cores, four GPUs, and 512 GB memory per node. There is a single memory space accessible by both the GPU and CPUs. The programming model is MPI and OpenMP for the CPUs and the AMD language extension HIP to run kernels on the GPU [1,2].

The ATS-4 single memory model stands in contrast to ATS-2 which had separate memory systems for the CPU and the GPU sections of the machine. The single address space and larger GPU accessible memory on ATS-4 are notable advancements. The single address space avoids costly memory migrations between CPU and GPU memory and allows for different algorithms; the increase in memory allows for larger data sets on the GPU.

Heterogeneous-compute Interface for Portability (HIP) is a programming language extension developed by AMD [7]. HIP is designed with a CUDA-like API to ease porting of existing codes which use a CUDA compiler. The LLNL compiler used for this port is the LLVM/Clang compiler with HIP extensions, provided by vendor partners Hewlett Packard Enterprises (HPE) and Advanced Micro Devices (AMD). Furthermore, HIP is the interface to the ROCm (Radeon Open Compute) platform which interacts with the AMD MI300A

APUs. Currently, we are using Clang version 17.0.6 with HIP extensions which support ROCm version 6.2.1 on the El Capitan hardware.

A typical run utilizes four MPI ranks per node. Each MPI rank uses HIP to target the parallelism inherent in the GPU, or OpenMP to access 24 CPUs. In practice, on the MI300A, the performance of a single GPU is superior to the use of 24 CPUs, thus the GPU parallelism is preferred. However, the ability to run a heterogeneous GPU and OpenMP code is a useful feature explored in the next sections.

3. CODE DESIGN AND GENERAL MODIFICATIONS

The Monte Carlo codes are quite flexible in their use of parallelism and algorithms and support history- and event-based tracking algorithms [6], controllable via a run time option. As the code may use either the GPU (via HIP) or the CPU (via OpenMP), a run time option to switch from the GPU to the CPU also exists. This flexibility to choose where to run a simulation (GPU or CPU) and the algorithm (event- or history-based) allows one to compare runs and to verify statistically equivalent results.

The option to compile with GPU and OpenMP allows for codes to efficiently use OpenMP based libraries which are not ported to the GPU. In addition, the ability for the code to run kernels on the CPU within the Monte Carlo code itself provides a fallback when compiler robustness issues are encountered in a GPU kernel. We can selectively remove individual kernels from the GPU until issues are addressed.

The Monte Carlo codes use the single address space on the MI300A by using the memory allocation call ‘hipMalloc’ for all memory which will be used on the GPU. The CPU may still access this memory as well, as the memory is single address space. Performance considerations involving memory location and caching levels favor the use of hipMalloc over standard C malloc for memory used in GPU kernels. The codes may use the LLNL Umpire memory pool manager [10], which reduces the actual number of hipMalloc, hipFree, and related system calls, improving overall run performance.

In any type of parallel computing design (such as either OpenMP or GPU threads), contention for the same memory location or cache line leads to reduced performance. In Mercury and Imp, two common data structures exhibit such memory contention: tallies and the particle vault.

We reduce tally memory contention by providing limited replication of the tallies. For an OpenMP capable code, the replication level is the number of OpenMP threads determined at run time. This replication allows for OpenMP parallelism to avoid memory contention. For the GPU, there is not enough memory to replicate the tallies for GPU threads; nevertheless, a limited replication reduces memory contention and improves performance. The number of tally replicas may be specified at run time based on observed performance results and memory utilization.

The particle vault data structure stores particles to be processed for tracking or that have already been processed [6]. We are not currently replicating the particle vault data structure, and thus rely on atomic memory operations when accessing the particle vault. An atomic operation is an indivisible operation on a memory location. The operation must complete without interruption and ensure the memory will not be read or written by any concurrent process or thread We had to account for atomic operations on the GPU on ATS-4, which will be discussed in Sec. 5.

To enable code compilation with either CUDA or HIP, we wrap each call with a simple C++ define. The design of HIP to match the CUDA interface easily facilitated this approach. For instance, the source code to determine the number of GPU devices is GPU_GET_DEVICE_COUNT() which maps to either cudaGetDeviceCount() if using CUDA or hipGetDeviceCount() with HIP.

We also updated our memory allocation abstraction to account for the HIP compiler. Prior to HIP, we had a set of memory management wrappers based on C++ defines. These defines could either allocate memory for the CPU or the GPU, with or without Umpire. However, this simplistic approach had issues compiling for the GPU device with early versions of the ROCm compiler. We redesigned the memory allocation into

a set of templated memory allocation functions, which provided a general code improvement. The use of templated memory management functions allows for both CPU and GPU kernels to be properly generated by the ROCm compiler.

4. CODE MODIFICATIONS TO IMPROVE CORRECTNESS AND ROBUSTNESS

After the above modifications allowed the code to compile, we encountered runtime robustness issues with the GPU kernels. These issues do not manifest as traditional coding errors, but rather as memory run time faults with various ROCm specific HSA (Heterogeneous System Architecture) memory errors. Other than verifying which kernel encountered the memory error, debugging tools such as the rocgdb debugger were of limited use.

After significant investigation and in collaboration with the vendor and compiler developers, we determined that the compiler itself was generating incorrect code. Several causes of incorrect code generation existed, but a primary one was the size of the stack in the GPU kernels. Monte Carlo codes are large and have deep call stacks, resulting in kernels with large 'stack sizes' which exceeded the expectations of the compiler developers and ultimately faulty code.

A contributing factor to these mis-compiles is the use of relocatable device code (RDC) as part of the link time optimization (LTO) performed by the tool chain. When linking multiple libraries, many of which contain GPU coding, the CPU invokes device routines from multiple libraries. Additionally, GPU device code in one library may invoke device code from another library on the GPU. As an example, Mercury's use of the General Interaction Data Interface (GIDI) library [11] for collision sampling and physics on the GPU is troublesome to the compiler.

We worked around the code correctness errors in several ways. Although we cannot bypass the RDC, we could reduce the quantity of RDC calls by moving most of the GIDI kernels which were implemented in C++ source files into header files, which effectively inlined the HIP code into Mercury itself. This modification removed calls using function pointers with inlined lambda functions, allowing the compiler to better predict stack size and generate more correct code.

Another code refactor involved the removal of recursion in device code. The ROCm tool chain at this time struggles to produce correct code when significant recursion is present, possibly due to the inability to predict the stack size needed. The code compiles but dies at run time with HSA memory errors. We have refactored much of our code to eliminate recursion, although this continues to be an ongoing effort.

Code robustness issues were also encountered at higher (O2) optimization levels. Code runs again died with HSA memory errors. To work around this issue, we identified functions which contributed to HSA memory errors at higher optimizations. We then inserted C pragmas in the source code, before these functions, which informed the compiler to bypass optimizations of the problematic routines. This modification allows for optimizations in other parts of the code while avoiding the compiler produced errors during the optimization of certain functions.

Up until the release of ROCm 6.2.1, we heavily relied on the above techniques to obtain a robust code. However, as we shared our efforts with the vendor, they made updates to the tool chain specifically for GPU code generation. By ROCm version 6.2.1, which was released in October 2024, we were able to remove most of the lower optimization pragmas and successfully run many simulations and tests.

A final note code robustness and the AMD compiler tool chain as it is used by the Monte Carlo code is that we do not on believe we have addressed all robustness issues. While we have large test suites for both Imp and Mercury that run correctly on ATS-4, end users will have more complicated use cases which will require additional vetting. In collaboration with our vendor partners, we continue testing to identify and resolve mis-compiles and improve code robustness. Resolving these issues is a complicated problem. The tool chain and associated run time libraries are used worldwide. As a result, any change to help Monte Carlo

codes with large kernels and deep call stacks needs to be checked with respect to correctness and performance implications in other codes in a multitude of organizations.

5. CODE MODIFICATIONS TO IMPROVE MI300A PERFORMANCE

Two of the larger performance issues encountered by the Monte Carlo codes on the MI300A were GPU stack-based kernel launch overhead and atomic operations.

When a kernel requiring large stack-sizes is started on the GPU, there is a cost associated with this transfer of execution from the CPU to the GPU and allocation of stack memory. The costs are ensuring the executable code itself is loaded onto the GPU device and that required stack memory and data is available on the GPU. If this total launch overhead is too large, it can dominate the actual time spent executing the kernel and limit performance. On the Nvidia GPU, the overhead was in the range of 7 microseconds based on performance analysis done by our team. The average MI300A kernel launch latency is 6 microseconds without stack allocation, but as we profiled our initial port on the MI300A, we saw overheads of over 1 millisecond for our larger tracking kernels. As we can easily launch hundreds of kernels per cycle, this large increase in launch overhead significantly limited performance. It is worth noting that the overhead of many of our smaller kernels, outside of particle tracking, had overhead of 6 microseconds. The determining factor was the size of the kernels – kernels with deeper stack sizes exhibited the 1 millisecond overheads.

We worked with the vendor and compiler developers to address the launch overhead issue. Similar to the code robustness issues described above, much of the launch overhead issues have been improved with the release of ROCm version 6.2.1. However, depending on the run mode, the current launch gap can be as high as 150 microseconds or as low as 4 microseconds, leading to varying performance. While a significant improvement from earlier ROCm releases, we continue to work with the vendor and compiler developers to decrease the launch overhead across all run modes.

We encountered contention-related performance issues with the use of atomic operators on the MI300A. Performance issues were particularly evident in event-based algorithms, but not all the GPU kernels in the event-based algorithms were slow – the common factor was the events which interacted with the particle vault data structure, and which relied on calls to atomicAdd(x,1), which increments a value by one. As noted in Sec. 3, we do not currently replicate the particle vault. The atomicAdd() operation on the MI300A out-of-the-box appears to be more sensitive to contention in comparison to Nvidia GPUs.

We reduced overall atomic contention for calls such as atomicAdd(x,1) by implementing Warp Level Aggregations [12]. A warp is a group of GPU threads which perform the same instruction at the same time; on NVIDIA GPUs, the warp width is 32 lanes, while on AMD GPUs the warp width is 64 lanes. Within a warp, atomic updates to the same memory address must be serialized into separate memory transactions for each lane performing the atomic update, drastically reducing performance. Contention between warps on different compute units (CU) within a GPU may also reduce effective memory bandwidth. To address the former issue, we implemented warp level aggregates for our atomic increments. The function counts the number of lanes in a warp performing the increment, then a single lane performs the atomicAdd() using this value. The use of this augmentation increases the overall performance of event-based test cases by a factor of three. Other calls to atomicAdd(x,y) which add a number other than one exist in the code and have not yet been modified to use this technique, which may provide added benefit.

As we continue porting the code to the MI300A, we are profiling the code and are considering if we can use different types of GPU memory or limited replication of certain fields in the particle vault to further improve GPU performance. We believe there are additional avenues for performance improvements.

6. MONTE CARLO MI300A PERFORMANCE ASSESSMENTS

In this section, we present performance assessments on the CTS-2, ATS-2, and ATS-4 computer platforms for Imp and Mercury. These computers are detailed in Sec. 2. We compiled with the CUDA compiler NVCC 11.8 on ATS-2, Clang 14 on CTS-2, and ROCm compiler 6.21 on ATS-4.

6.1. Imp Crooked Pipe Results

We used Imp version 5.43.0.mcs50.imp14 with the Crooked Pipe test problem [13], an idealized thermal photon transport test problem in which a boundary temperature source induces radiative heat flow down a pipe with a bend. Similar to [6], we ran Imp for 100 timesteps of variable size and calculated the temperature at five fiducial points in a 2D RZ mesh geometry with 5,000 zones, 1 domain, constant gray opacity, constant specific heat, and a 300 eV black body source. The geometry is shown in Fig. 1(a).

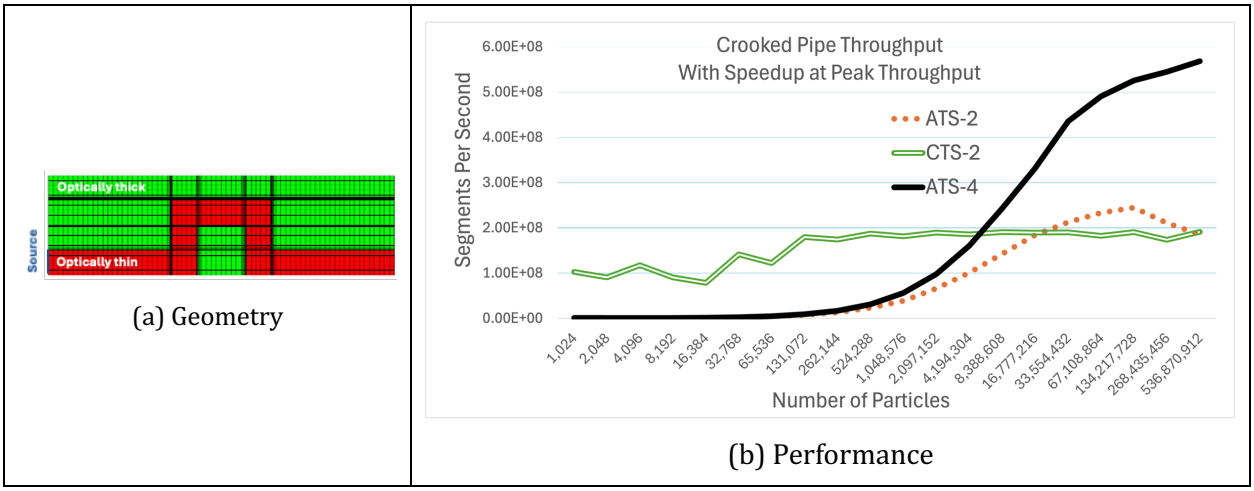

Figure 1. Crooked pipe test problem geometry and performance.

We ran a saturation throughput study on a single node of each platform, using the algorithm which delivered the best results for each platform. For the GPU-based ATS-2 and ATS-4 machines, the event-based tracking algorithm provided the best performance. For CTS-2, the history-based tracking was more performant. We began with 2^{10} (1,024) particles and increased the number of particles in powers of two. This study replicates the throughput study in paper [6] but with the current code and different hardware. We plotted segments-per-second, which is the number of segments computed divided by the time spent tracking particles. A sequence of segments constitutes a Monte Carlo particle history.

Figure 1(b) shows that throughput on CTS-2 saturates at 2^{19} (524,288) particles achieving 187.8 million segments/second. At M&C 2023 we reported [6] that the earlier commodity system CTS-1, with 36 Xeon E5-2695 CPUs, saturated at 2^{17} (131,072) particles and achieved 37.6 million segments/second. The CTS-2 system can handle larger workloads and increases performance 5.0x compared to CTS-1.

Throughput on ATS-2 saturates at 2^{27} (134,217,728) particles achieving 245.1 million segments/second. At M&C 2023 we reported [6] that ATS-2 also saturated at 2^{27} particles at 222.3 million segments/second. Modifications to the code for the port to ATS-4 have not degraded Imp performance on ATS-2.

Throughput on ATS-4 continued to improve up to 2^{29} (536,870,912) particles achieving 568.7 million segments/second. For these runs, we stopped the throughput study on the ATS-4 at 2^{29}.

Comparing node to node performance, the throughput of ATS-4 achieves a 3.0x speedup over CTS-2, and more than 2.3x speedup over ATS-2. ATS-4 always outperforms ATS-2, while larger workloads - 2^{23} (8,388,608) particles per node for this test problem - are necessary to make effective use of the ATS-4 MI300A compared to the CTS-2 Xeon CPUs.

6.2. Mercury Godiva In Water Results

We used Mercury version 5.43.0.mcs50.mer27 with the Godiva in Water test problem [14]. We used GIDI library version 3.31.26 with this build. Godiva in Water is a Godiva critical sphere surrounded by water. We used Mercury to calculate the k-eigenvalue in a 3D Cartesian mesh octant geometry with 27,000 zones, 1 domain, and continuous energy nuclear data. The geometry is shown in Fig. 2(a).

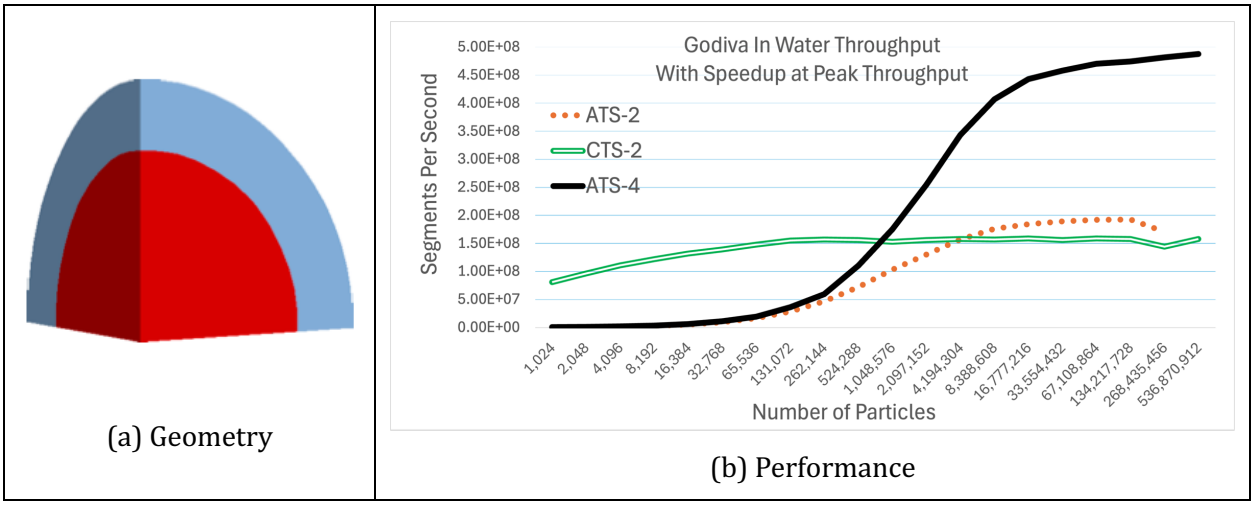

Figure 2. Godiva in water test problem geometry and performance.

Similar to the Imp testing in section 6.1, we ran a saturation throughput study on a single node of each platform, using the algorithm which delivered the best results for each platform. We used event-based tracking for both ATS-2 and ATS-4, and history-based tracking for CTS-2. We computed segments-per-second as defined in Sec. 6.1.

Figure 2(b) shows that throughput on CTS-2 saturates at 2^{17} (131,072) particles achieving 155.0 million segments/second. At M&C 2023 we reported that the earlier commodity system CTS-1, saturated at 2^{14} (16,384) particles and achieved 27.8 million segments/second. For this simulation, CTS-2 system can handle larger workloads and increases performance 5.6x compared to CTS-1

Throughput on ATS-2 saturates at 2^{26} (67,108,864) particles achieving 191.9 million segments/second. At M&C 2023 we reported [6] that ATS-2 saturated at 2^{27} (134,217,728) particles at 222.3 million segments/second. Modifications to both the GIDI library and Mercury necessary for the port to ATS-4 resulted in a 13.6% performance degradation on ATS-2. We analyzed the contributing factors by building the current Mercury code, but using the same build of GIDI used for the M&C 2023. The results show that 4% of the performance degradation is attributable to the GIDI refactor necessary for robust code on ATS-4. The remaining 9% is attributable to code modifications in the Mercury source code.

Throughput on ATS-4 continued to improve up to 2^{29} (536,870,912) particles achieving 487.7 million segments/second. Comparing node to node performance, the throughput of ATS-4 achieves a 2.5x speedup over ATS-2 and a 3.1x speedup over CTS-2.

6.3. Imp N210808 Results

We used Imp version 5.43.0.mcs50.imp14 with the N210808 test problem based on the National Ignition Facility N210808 shot that was the first inertial fusion experiment to exceed the Lawson criterion for ignition [15]. The geometry represents the conditions of a partially imploded capsule and is shown in Fig. 3(a). The test problem uses a 3D domain-decomposed mesh with 5,418,900 zones and 16 domains requiring particle communication across domain boundaries. A minimum of one thermal emission particle per zone implies at least 5.4 million particles per run. The simulation uses a time- and frequency-dependent photon source on the outer boundary. The test was run on four nodes of each platform, with a variable time step and 100 time steps.

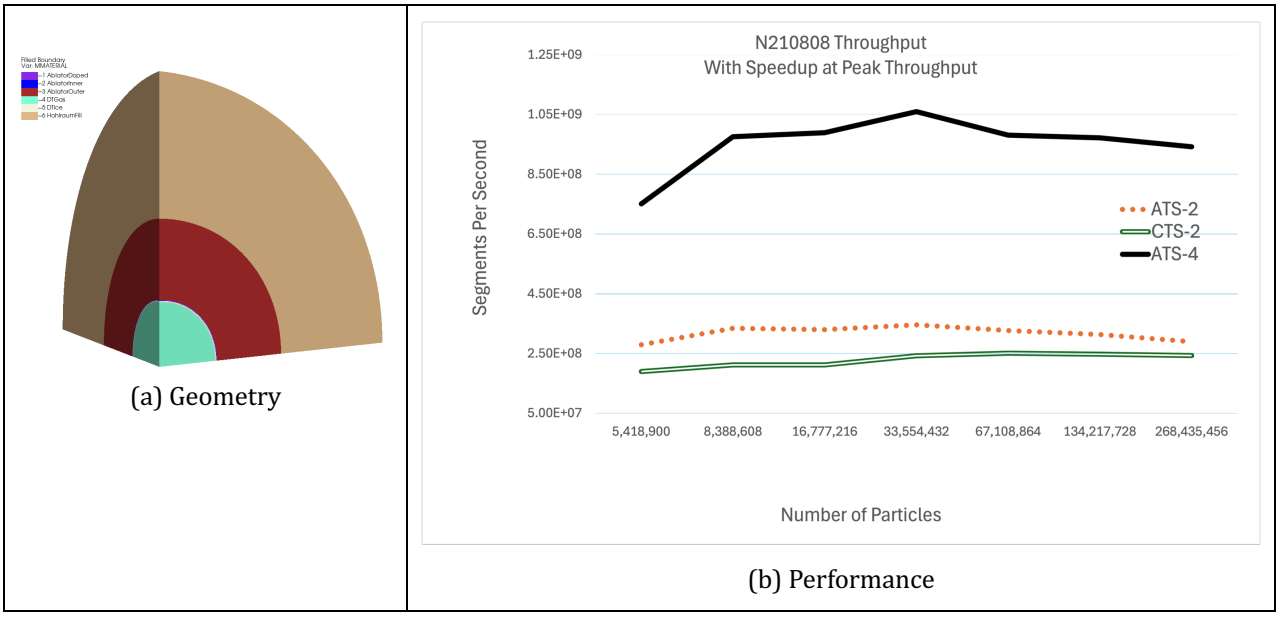

Figure 3. N210808 test problem geometry and performance.

We ran a saturation throughput study on four nodes of each platform, using the algorithm which delivered the best results for each platform. We used event-based tracking for both ATS-2 and ATS-4, and history-based tracking for CTS-2. We tracked segments-per-second as defined in Sec. 6.1. Figure 3(b) presents the results. At 2^{25} (33,554,432) particles, CTS-2 achieves 243.0 million segments/second, ATS-2 achieves 346.0 million segments/second, and ATS-4 achieves 1060.0 million segments/seconds. ATS-4 provides speedups of 3.1x vs. ATS-2 and 4.3x vs. CTS-2.

Each cycle of the calculation is composed of multiple parts which includes initialization, tracking, and finalization routines related to the Monte Carlo simulation. Each cycle also includes calls to other libraries and MPI communication calls to send and receive particles across boundaries. While Imp has a load balancing algorithm, at this smaller scale run, load imbalance still exists resulting in processes waiting to receive particles many times within each cycle. An informative metric is the ratio of the time spent in 'Tracking' vs. the total cycle time. At the lower particle count, this ratio is lower than at the higher particle count for which particle tracking dominates the cycle time. As the particle count increases, this ratio increases as does the segments per second. These values are shown in Table I. To increase the performance of Imp and Mercury across a range of workloads, it is also important to improve the performance of coding outside of tracking.

Table I: ATS-4 Imp N210808 – Influence of Particle Count on Tracking Ratio and Segments/Sec

Number of Particles	2^{23}	2^{24}	2^{25}	2^{26}
Tracking Ratio	0.30	0.40	0.48	0.65
Segments / Second	9.21×10^{8}	9.45×10^{8}	1.03×10^{9}	1.04×10^{9}

Another factor which impacts performance is the time step size. Shorter time steps result in less elapsed time and fewer collisions per GPU kernel launch. Since the N210808 test problem allows for a variable time step, we observe that within a single simulation, portions of the simulation with larger time steps exhibited a higher speedup than portions with a lower time step. As end users run simulations with a range of time steps, this also argues for continuing to improve the performance of coding outside of particle tracking. Table II shows the tracking ratio within a single run of 2^{25} particles at various individual cycles from 1 to 100. As the time step increases from 1×10^{-13} to 5×10^{-12} seconds during the run, the tracking ratio increases, leading to improved performance as more work is performed per GPU kernel launch.

Table II: ATS-4 Imp N210808 – Influence of Time Step on Tracking Ratio

Cycle	1	25	50	75	100
Time Step	1.0×10^{-13}	9.8×10^{-13}	5.0×10^{-12}	5.0×10^{-12}	5.0×10^{-12}
Tracking Ratio (cumulative)	.05	.15	.34	.44	.48

7. CONCLUSIONS AND FUTURE WORK

The port of the LLNL Monte Carlo Imp and Mercury codes to the AMD Instinct™ MI300A APU, El Capitan architecture, and ROCm was more challenging than expected due primarily to the larger kernels and deep call stacks with the Monte Carlo kernels. We ported the code while the vendor was actively improving the ROCm compiler based on feedback from our code and many other projects at LLNL. This close collaboration benefited both the vendor and customer. While we have demonstrated measurable node to node speedups compared to Nvidia GPUs and Xeon based clusters, we believe there is still opportunity for performance improvements both inside and outside of the tracking portions of Monte Carlo transport codes. GIDI robustness and performance is of particular interest. We will continue to profile, identify bottlenecks, and refactor code to increase performance. Tool chain issues impacting performance (launch overhead) and robustness (runtime memory errors) remain which we are working to resolve in collaboration with HPE and AMD.

ACKNOWLEDGMENTS

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. We thank Ramesh Pankajakshan of LLNL and the following AMD and HPE employees for their help on this porting effort: Steve Abbott, Brendon Cahoon, Austin Ellis, Austin Kerbow, Eduardo Ponce and Michael Rowan.

REFERENCES

[1] A. Smith et al., "Realizing the AMD Exascale Heterogeneous Processor Vision: Industry Product," *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, Buenos Aires, Argentina, 2024, pp. 876-889, doi: 10.1109/ISCA59077.2024.00068.

[2] A. Smith et al., "11.1 AMD Instinct™ MI300 Series Modular Chiplet Package – HPC and AI Accelerator for Exa-Class Systems," *2024 IEEE International Solid-State Circuits Conference (ISSCC),* San Francisco, CA, USA, 2024, pp. 490-492, doi: 10.1109/ISSCC49657.2024.10454441.

[3] R. Procassini, D. Cullen, G. Greenman, C. Hagmann. "Verification and Validation of Mercury: A Modern, Monte Carlo Particle Transport Code." In *Proceedings of MC2005*. Chattanooga, TN, USA (2005).

[4] J. A. Fleck and J.D. Cummings. "An implicit Monte Carlo scheme for calculating time and frequency dependent nonlinear radiation transport." *Journal of Computational Physics*. volume 8, pp. 313-342 (1971).

[5] P. Brantley, N. Gentile, M. Lambert, M. McKinley, M. O'Brien, J. Walsh. "A New Implicit Monte Carlo Thermal Photon Transport Capability Developed Using Shared Monte Carlo Infrastructure". In *Proceedings of M&C 2019*, pp 564-577. Portland Or. USA (2019)

[6] M. Pozulp, R. Bleile, P. Brantley, S. Dawson, M. McKinley, M. O'Brien, A. Robinson, M. Yang. "Progress Porting LLNL Monte Carlo Transport Codes to Nvidia GPUs." In *Proceedings of M&C 2023*, Niagara Falls, Ontario Canada (2023)

[8] S. Hamilton and T. Evans. "Continuous-energy Monte Carlo neutron transport on GPUs in the Shift code." *Annals of Nuclear Energy*, volume 128, pp. 236–247 (2019).

[9] J. Tramm, et al. "Performance Portable Monte Carlo Particle Transport on Intel, NVIDIA, and AMD GPUs." https://arxiv.org/abs/2403.12345 (2024).

[10] D. Beckingsale, M. Mcfadden, J. Dahm, R. Pankajakshan and R. Hornung, "Umpire: Application-Focused Management and Coordination of Complex Hierarchical Memory" in *IBM Journal of Research and Development*. 2019. doi: 10.1147/JRD.2019.2954403

[11] B. Beck, C. Mattoon, and G. Gert. "GIDI+: A GNDS 2.0 suite of C++ APIs to access nuclear and atomic data for use in radiation transport codes." *EPJ Web of Conferences*. Vol. 284. EDP Sciences, 2023.

[12] S. G. D. Gonzalo, S. Huang, J. Gómez-Luna, S. Hammond, O. Mutlu and W. -m. Hwu, "Automatic Generation of Warp-Level Primitives and Atomic Instructions for Fast and Portable Parallel Reduction on GPUs," *2019 IEEE/ACM International Symposium on Code Generation and Optimization (CGO),* Washington, DC, USA, 2019, pp. 73-84, doi: 10.1109/CGO.2019.8661187.

[13] F. Graziani and J. LeBlanc. "The Crooked Pipe Test Problem*." LLNL Technical report*, UCRL-MI-143393 (2000).

[14] D. E. Cullen, C. J. Clouse, R. Procassini, and R. C. Little. "Static and Dynamic Criticality: Are They Different?" *LLNL Technical report*, UCRL-TR-201506 (2003).

[15] H. Abu-Schawareb, et al. "Lawson Criterion for Ignition Exceeded in an Inertial Fusion Experiment." *Phys. Rev. Lett*. **129**, 075001 (2022).