GPU simulation with Opticks: The future of optical simulations for LZ

Oisín Creaner[1,6,*], *Simon* Blyth[2], *Sam* Eriksen[3], *Lisa* Gerhardt[1], *Maria Elena* Monzani[4,5], and *Quentin* Riffard[1]

[1]Lawrence Berkeley National Laboratory (LBNL), Berkeley, CA 94720-8099, USA
[2]Institute of High Energy Physics, Chinese Academy of Sciences, 19B Yuquan Road, Shijingshan District, Beijing, Postal code: 100049, China
[3]University of Bristol, H.H. Wills Physics Laboratory, Bristol, BS8 1TL, UK
[4]SLAC National Accelerator Laboratory, Menlo Park, CA 94025-7015, USA
[5]Kavli Institute for Particle Astrophysics and Cosmology, Stanford University, Stanford, CA 94305-4085 USA
[6]Dublin Institute for Advanced Studies, 31 Fitzwilliam Place, D02 XF86 Dublin 2, Ireland

Abstract.
The LZ collaboration aims to directly detect dark matter by using a liquid xenon Time Projection Chamber (TPC). In order to probe the dark matter signal, observed signals are compared with simulations that model the detector response. The most computationally expensive aspect of these simulations is the propagation of photons in the detector's sensitive volume. For this reason, we propose to offload photon propagation modelling to the Graphics Processing Unit (GPU), by integrating `Opticks` into the LZ simulations workflow. `Opticks` is a system which maps Geant4 geometry and photon generation steps to NVIDIA's `OptiX` GPU raytracing framework. This paradigm shift could simultaneously achieve a massive speed-up and an increase in accuracy for LZ simulations. By using the technique of containerization through `Shifter`, we will produce a portable system to harness the NERSC supercomputing facilities, including the forthcoming Perlmutter supercomputer, and enable the GPU processing to handle different detector configurations. Prior experience with using `Opticks` to simulate JUNO indicates the potential for speed-up factors over 1000× for LZ, and by extension other experiments requiring photon propagation simulations.

1 Introduction

LUX-ZEPLIN (LZ) is a direct-detection Dark Matter experiment. It consists of a Time Projection Chamber (TPC) contained within a vacuum-insulated cryostat and surrounded by an Outer Detector (OD). The TPC consists of a 7-tonne active mass of Liquid Xenon (LXe) and two arrays of photomultiplier tubes (PMTs) at the top and bottom of the apparatus. The OD uses 17 tonnes of organic liquid scintillator loaded with Gadolinium (GdLS), also observed by a suite of PMTs as illustrated in Figure 1 [1–3].

The primary objective of the experiment is to detect Weakly Interacting Massive Particles (WIMPs) through their interactions with the LXe target mass [2]. Particle interactions in the

[*]e-mail: creanero@gmail.com

active region cause ionisation, excitation and heat, which can produce photons by the prompt scintillation (S1) and secondary electroluminescence (S2) processes [1]. Differences in incoming particle type, energy and position produce different combinations of position, time and energy partition between the S1 and S2 signals. These differences allow for discrimination between particle interactions. Some Neutrons and Gammas produce an additional scintillation signal in the GdLS which allows them to be excluded [3].

WIMP signals are expected to be rare ($\sim 10^{-4}\text{kg}^{-1}\text{day}^{-1}$), while background signals from both local radioisotopes and cosmogenic sources are typically $\sim 10^6$ times more frequent [4]. Substantial mitigation efforts including shielding, careful selection of detector materials, and a deep underground location are used to reduce this factor in LZ [2]. Despite these mitigation factors, the background rate is still expected to dominate the WIMP signal [5]. Accurate simulations and offline analyses of events and background are thus essential for the construction of the background model used in LZ [6].

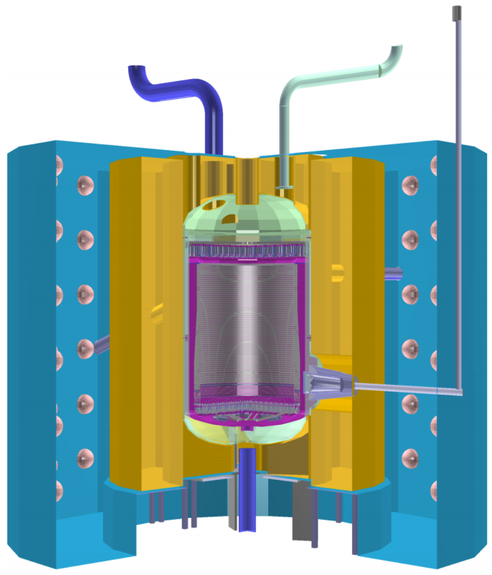

Figure 1. A schematic of the LZ detector. The components, from the inside outwards are: the TPC (magenta), the cryostat (green) the OD (yellow) and a water tank (blue). (Taken from [6]).

`BACCARAT` is the LZ code for simulating particles and their interactions. `BACCARAT` uses `Geant4` to track particles and identify the points at which incoming particles interact with the detector [6]. Within `BACCARAT`, there are two options for simulating photon propagation: full and fast mode.

In full mode, photons and electrons generated by these interactions are fully simulated in `Geant4` using the `G4S1Light` and `G4S2Light` modules developed by LZ which track each photon within the detector. From these simulated tracks, individual photon hits on the PMTs are recorded. These are then passed to the `DER` (Detector Electronics Response) module which simulates the response of the PMTs. This has been used in Mock Data Challenges (MDCs) to produce data for analysers to practice on. In the fast mode, energy deposits are passed to the `NEST` (Noble Element Simulation Technique) module which uses detector-averaged quantities to generate S1 and S2 signals [6].

Full optical tracking using GEANT4 has been estimated to consume >95% of CPU time used in LZ simulations. As a result, for the generation of large-scale datasets, the fastNEST package is used to compute signal sizes which can be encoded into maps giving the probability of each outcome. These results, however, do not contain information on the times of interactions or specific photon hits on PMTs [6].

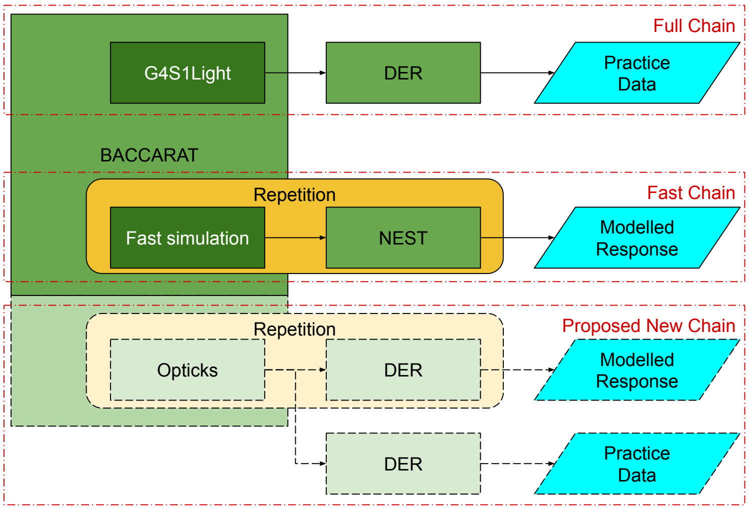

Figure 2. A schematic of the current LZ and proposed simulation workflow. In the current system, BACCARAT users can choose through a macro file whether to generate output through the fast or full chains. In the Full Chain, G4S1Light and G4S2Light simulate photon propagation and pass these to DER which simulates the detector response, and can be used for MDCs. In the Fast Chain, energy deposits are passed to NEST to create a map of probabilities of detector response. In the proposed system, both chains could be replaced with one based on Opticks [6].

We propose to replace both the full and the fast simulation chain with a single system which combines the speed of the fast-chain with the fidelity of the full-chain as illustrated in Figure 2.

2 GPU simulations

GPUs evolved to perform real time 3D graphics rendering. Ray-tracing is one means by which graphics can be rendered. Ray-tracing leverages the parallelism inherent in GPU-based systems to calculate the intersection between many simulated rays of light and simulated objects in a virtual world [7]. The similarity between this commercially available use of GPUs and the photon propagation simulations required for particle physics experiments suggests that with a suitable framework, substantial speed-up may be achieved in the latter case.

In the case of LZ, a replacement of the photon propagation simulations with a GPU-based solution may allow for large-scale datasets to be generated with full fidelity modeling of photon hits in the detector.

2.1 Opticks: GPU Accelerated Optical Photon Simulation using NVIDIA OptiX

`Opticks` is an open-source project, described in detail in Blyth's works [8–10] which are summarised briefly in this subsection. `Opticks` replaces `Geant4` optical photon simulation with an equivalent implementation using GPU accelerated ray tracing from NVIDIA `OptiX`.

`OptiX` is a general-purpose ray-tracing framework developed by NVIDIA to give optimal performance on NVIDIA's GPUs [11–14]. `OptiX` works on the premise that most ray-tracing algorithms require only a small set of operations [11]. `OptiX` allows users to build applications to control the generation of rays and their interactions with surfaces [15].

`Opticks` translates the elements of the `Geant4` context that are relevant for photon simulation into appropriate forms and uploads them to the GPU. These include the detector geometry, its optical properties, and the optical photons to be simulated [10].

The solids that make up a geometry can be translated to `Opticks` primitives by the `G4Opticks` class. `G4Opticks` goes through the `Geant4` volume tree and converts its components into `Opticks` equivalents. It does so using constructive solid geometry (CSG) modelling. In CSG, more complex shapes are created from primitives by Boolean operations such as intersection, union and so on. The more complex the geometry, the longer the initialization time, so `Opticks` stores a serialization of the geometry as a `geocache` — a collection of many `.npy` [16], `.txt` and `.json` [17] files. This avoids repeated processing [8].

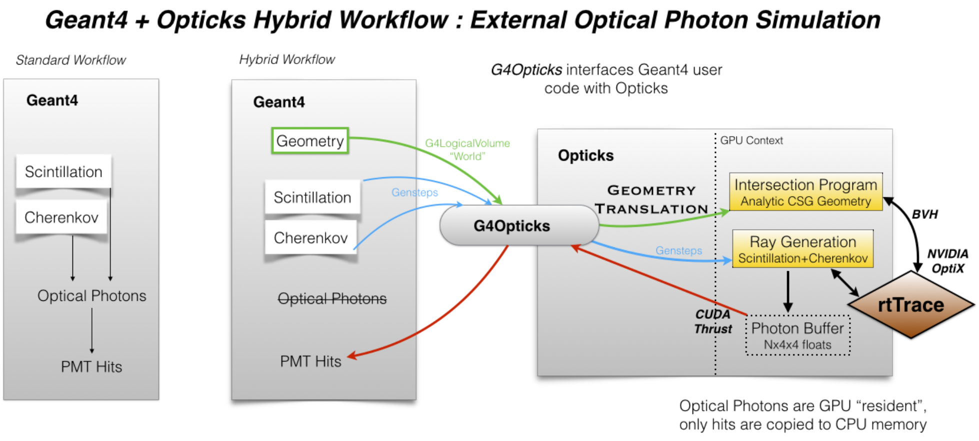

Figure 3. An illustration of how `OptiX` integrates `OptiX` into a particle physics workflow. The `Geant4` geometry and photon generation steps are translated by `Opticks` and passed over to `OptiX`. At initialization the `Geant4` geometry is translated into a GPU appropriate form and uploaded to the GPU. During event processing `genstep` data structures are collected and copied to the GPU where the optical photon generation and propagation are performed using NVIDIA `OptiX` ray tracing. Once the `OptiX` ray-tracing is complete, only photon hits on the PMTs are copied to the CPU for further processing. Taken from [10].

The Boolean operations used for geometry input form a binary tree structure. Ensuring that this tree structure is a balanced one — i.e. for each node, the sub-trees are of equal height, thus minimising the height of the overall tree — allows for faster intersection time and more efficient tree serialization [8].

Instead of generating photons in a loop as is done in `Geant4`, a `genstep` data structure is used which includes the number of photons to be generated and a line segment upon which to generate them. By default, `Geant4` has no interface to `genstep`. Therefore, the classes which represent scintillation and Cherenkov processes in `Geant4` were modified to collect `genstep` structures. `Opticks` then launches parallel GPU threads to propagate these photons with NVIDIA `OptiX` GPU ray-tracing. `OptiX` implements optical photon generation and propagation using the GPU-accelerated ray-geometry intersection provided by NVIDIA `OptiX`. `Opticks` includes a configurable number of steps to propagate these rays through [9].

Tests comparing the performance of `Opticks` with a `Geant4`-based photon simulation of another detector — the Jiangmen Underground Neutrino Observatory (JUNO) detector [18] — recorded a speed-up factor of 1660 in optical photon simulation [8]. Additionally, improved performance was observed with very simple analytic geometries when compared with the JUNO geometry, which suggested that there is potential for improvement by optimisation of geometry modelling.

The current stable build of `Opticks` is based on NVIDIA `OptiX` version 6.5. `OptiX` version 7.0 was released in 2019, and includes a new low-level API which allows greater flexibility for Multi-GPU systems and larger datasets [14]. `OptiX` version 7 does not provide backwards compatibility to code, including `Opticks`, which were built for previous versions of `OptiX`. An ongoing project to allow `Opticks` to use the latest `OptiX` version and features is ongoing between the authors and NVIDIA and discussed elsewhere in these proceedings [19].

3 Integrating with LZ

A number of challenges exist in relation to the integration of `Opticks` to the LZ workflow. These impact the inputs and outputs of the `Opticks` and the corresponding outputs of `BACCARAT` and inputs to `DER` [6] it will interact with.

As shown in Subsection 2.1, the preferred mechanism for photon generation in `Opticks` is to modify the Cherenkov and scintillation processes to use the `genstep` structure. However, if `Opticks` is to be incorporated as a modular component of the LZ workflow, changes which would impact other modules should be avoided. Additionally, initial experiments in defining the LZ TPC in terms of the ~10 primitive shapes for which `Opticks` provides intersection functions have proven difficult [20–22]. Finally, the definition of the LZ geometry currently used generates a highly unbalanced tree structure which is known to create inefficiencies [8, 9, 20–22]. Ongoing work will focus on the slowest solids by changing the implementation to address these inefficiencies.

Further, the output format of `Opticks` PMT hits is not identical to the outputs from `BACCARAT` which are used as inputs to `DER`. As a result, some translation will be needed to ensure compatibility. Finally, it is desirable to have an optional system for running performance metrics on `Opticks` when used for LZ.

To this end, it is proposed to develop a lightweight API which will:

- Translate the outputs of `BACCARAT` particle physics simulations to `genstep` structures.
- Translate the `Opticks` outputs to suitable inputs for `DER`.
- Enable timing, memory consumption and throughput metrics on request.

4 Supercomputer Implementation

Data simulation and processing for LZ is carried out at the National Energy Research Scientific Computing (NERSC) center [23]. The supercomputer facilities at NERSC provide

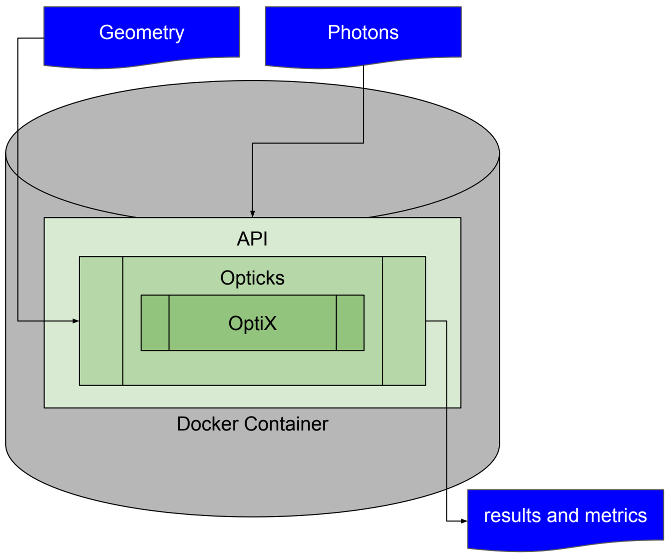

Figure 4. Schematic of the layered operational structure used to integrate `Opticks` with the LZ and NERSC workflows. LZ geometry and photon generation are passed to an API which transforms these inputs into a format suitable for `Opticks`, which acts as a wrapper for `OptiX` raytracing. To operate on NERSC systems, a `Shifter` container is used to provide access to libraries and dependencies.

the volume of computing capabilities required to carry out the large number of simulated LZ events required to provide confidence in their predictions [3, 6, 23]. The system currently used by LZ simulations is Cori, a Cray XC40, with peak performance of about 30 petaflops [24]. However, Cori's production system does not have GPU nodes, and therefore `Opticks` is unable be used on this system to leverage the `OptiX` GPU capabilities [25]. In 2018, 18 nodes were added to Cori, each with several V100 GPU accelerators. These nodes are intended for porting, development, and testing; and are being used for early `Opticks` testing [26].

The forthcoming Perlmutter supercomputer will be a production-scale GPU-accelerated system, including 1500 GPU-accelerated nodes with 4 NVIDIA A100 Tensor Core GPUs each for a total of 6000 cores [27]. It is expected that software built and tested on CoriGPU will work with minor changes on Perlmutter. Expected changes include configuration settings such as the CUDA Compute Capability setting, which is 8.0 for Perlmutter's A100s and 7.0 for CoriGPU's V100s [26–28]. Current development effort is focused on ensuring portability through containerisation as discussed in Section 5 [29].

There are limitations on the supercomputer systems at NERSC which makes direct use of `Opticks` impractical. Security is a significant concern with all NERSC systems. As a result, standard users will never have root/administration privileges, so unlike on a standalone computer, there are restrictions on what the user is permitted to do. For example, users are not typically permitted to install software via package managers like `yum`, and will have strict restrictions on the paths they have read and write access to. In addition, the versions of software or libraries that are supported and available are large but finite [30, 31].

`Opticks`, on the other hand, is dependent on a large number of libraries and dependent software [25, 32]. Many of these libraries are version-dependent and the calls to them in Opticks will not work with earlier (e.g. Boost) or later (e.g. OptiX) versions [33]. Opticks is currently changing over from Python2 to Python3 [34], and while Python 3.7.8 is preferred, there remains some potential for issues to arise [35]. Additionally, at runtime, Opticks — unless run with the --production flag set — writes output to some locations within its build environment, but which a NERSC user might not have write access to [25]. Finally, Opticks scripts frequently specify paths in which installed components of the system are expected to be found, either explicitly (e.g. /usr/local) or implicitly (e.g.$HOME) [32]. This makes it unsuitable for direct use on the supercompter systems.

5 Containerisation

The solution to this challenge is to use containerisation [29]. Containerisation allows for software to be standardised and portable [36, 37]. Docker is the leading containerisation system currently in use [36]. Software, libraries, and data can be bundled together into a Dockerfile [38], which can be built into an image which can be published or stored in a public or private repository such as Dockerhub [39]. At runtime, containers are generated from these images which share the host OS kernel and therefore can be more lightweight and portable than Virtual Machines [36].

Shifter is the preferred solution for containerisation at NERSC [37, 40, 41]. Shifter is built upon Docker [37, 38] and uses much the same structure of Dockerfile, image and container [38, 40].

The user can never be root in Shifter [40], whereas the default in Docker is for the user to be root [38]. Therefore, Shifter users have reduced permissions relative to Docker users. Furthermore, Shifter images are mounted as read-only, which prevents the user from running any software that requires write access mounted in the container [40]. On the other hand, Shifter permits access to NERSC parallel file systems and allows directories in these file systems to be mounted to the container in locations the user specifies [40]. Therefore, the user may mount a path they have read/write access to on a location they would not normally be able to use to simulate a suitably write-enabled environment.

6 Implementation: opticks_on_shifter

In order to allow Opticks to run on CoriGPU, a Shifter image called opticks_on_shifter has been produced [29, 42, 43]. Using the base image nvidia/cuda:11.0-devel-centos7 [44], this container was created by following the Opticks install instructions [32] and converting the relevant commands into a Dockerfile [42]. This Dockerfile was then built progressively in a test environment called Maeve. Maeve is equipped with three NVIDIA Titan X GPUs, and allows for iterative development using Docker to install libraries which would not be permitted on CoriGPU. Development on Maeve was halted after the external build tools were installed and before the opticks-full command was called to complete the installation of Opticks itself. This is because the final build stages are hardware-aware and the Titan X GPUs on Maeve do not match the V100s on CoriGPU nor the A100s on the forthcoming Perlmutter.

This prepared environment was then pushed as an image to dockerhub [43] and pulled to CoriGPU. On CoriGPU, a Shifter container was created by launching this image. The Opticks-specific contents of this were then copied to the NERSC $SCRATCH file system and the container was closed. The container was then relaunched with the copied contents on $SCRATCH mounted over the corresponding locations in the image. The opticks-full command was then called to complete the installation of Opticks.

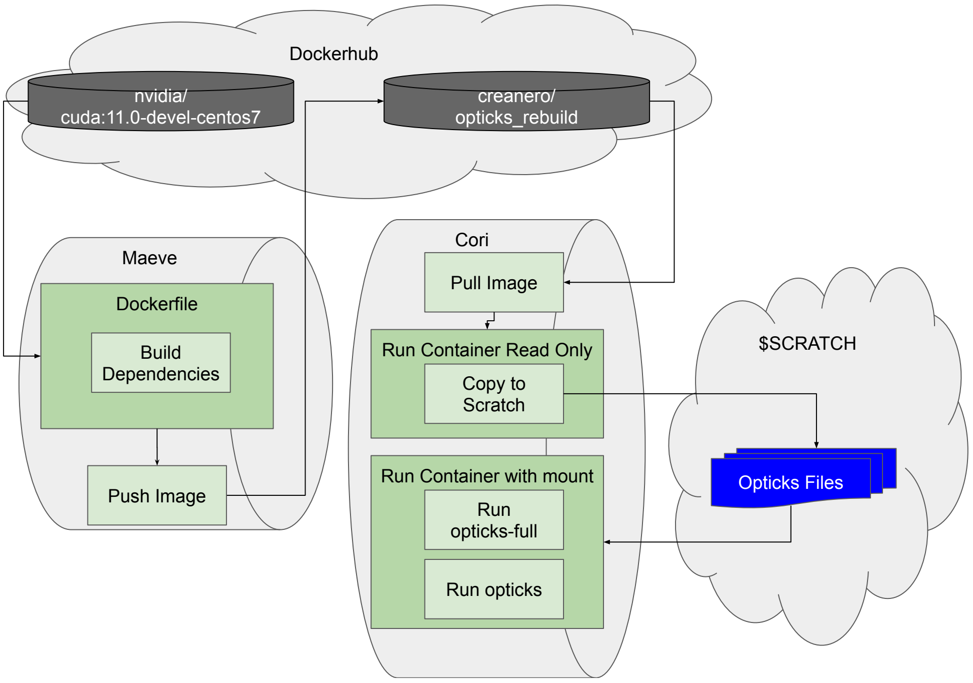

Figure 5. A schematic of the process by which the image `opticks_on_shifter` was created. Using NVIDIA's `cuda:11.0-devel-centos7` as a base, a `Dockerfile` was created to build the `Opticks` dependencies on `maeve`. An image containing these dependencies was uploaded to Dockerhub. That image was pulled to Cori and the relevant contents copied to `$SCRATCH`. Then the container could be run with that contents mounted from `$SCRATCH`.

6.1 Advantages and Disadvantages

The approach used here has a number of advantages and disadvantages. Some of these are inherent to containerisation, particularly with `Shifter`, while others are specific to this system or others similar to it.

The advantages of this approach are:

- Containerisation inherently allows for portability, particularly relevant as this system will have to be ported to Perlmutter for production use.
- Locking in the correct versions of dependent software at the docker image stage ensures the requirements for `Opticks` will always be met.
- Even if compatibility-breaking upgrades are pushed by software vendors, they will not affect the controlled environment of the container.

The drawbacks of the system are:

- If there were any exploits or vulnerabilities in the software upon which this system depends, these will be persisted in the container even after the vendor provides the necessary patches.
- When the decision is made to upgrade the software or its dependencies, then the upgrade process will have to go back to an early stage of this process and this may be laborious.

- Some bars to portability such as the hardware-aware components of `opticks-full` may not be apparent.

7 Performance Benchmarking

The measure of success for any new system is whether it outperforms existing systems. We require the proposed system to be faster than the full-chain and have greater predictive value than the fast-chain described in Section 1. The ideal outcome would be to match or outperform the fast-chain for speed and the full-chain for fidelity. In the case of the result being between these two extremes, some combined metric would be needed to determine the utility of the system. This latter remains an open point of research.

Speed is relatively straightforward to measure. The initial approach will be to measure the wall-clock time for the completion of a simulation of some fixed collection of photons through the LZ geometry. Current predictions of success in this factor are promising. As stated in Subsection 2.1, the comparison of `Opticks` with `Geant4` for the JUNO experiment, gave a speed-up factor of 1660 for optical simulations [8].

The `BACCARAT` fast chain simulations achieved a speed-up factor of 20 by comparison with the full chain in LZ testing [6]. In practical terms, the overall speed-up factor for LZ with GPU optical propagation simulations are expected to give similar performance to the fast chain, as optical photon propagation represents $\gtrsim$95% of the CPU time in full chain simulations. Thus, the remaining $\lesssim$5% factors may be expected to come to dominate, giving a speed-up factor of 20.

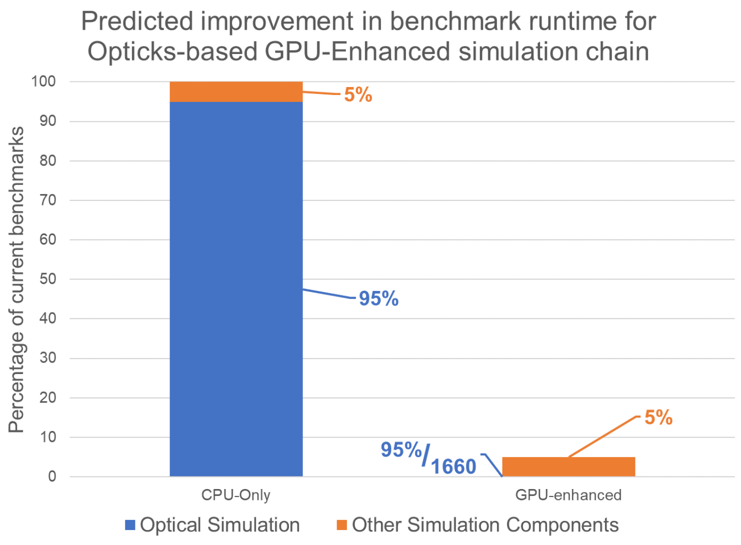

Figure 6. Illustrative example of the expected reduction in runtime for the GPU-enchanced simulation chain by comparison with the existing full-chain.

`Opticks` includes validation testing in which one executable calls both `Geant4` and `Opticks` simulations [8]. Identical input photons and random seeds are passed to each system, allowing the outputs to be a near-perfect match [8]. Use of this validation testing in the case of the JUNO geometry demonstrated that the system was vulnerable to fragile CSG modelling, but with this corrected, discrepancies were <0.25% for mis-aligned photon histories and <0.05% of deviant photons within matched histories [8]. Applying this system to the LZ geometry will permit similar metrics to be taken. Should performance for LZ closely match the experience with JUNO, then this will be considered a success. The increased complexity of the current LZ model may increase its vulnerability to CSG modelling errors or issues such as photons with grazing incidence. Iterative development of a more refined model of the LZ detector is expected to mitigate these concerns.

8 Conclusions

This paper describes the potential for `Opticks` to fill the role of a GPU-based optical photon propagation model for LZ. LZ's existing system, `BACCARAT`, uses two separate simulation paths, the fast chain and the full chain [6]. In simulations of the JUNO experiment, `Opticks` has been shown to successfully integrate NVIDIA's GPU-based `OptiX` ray-tracing framework with `Geant4` particle physics simulations [8].

A key challenge of this work is integrating `Opticks` with the LZ framework and the NERSC computing systems. The approach to solve this is a layered, modular one, where `BACCARAT` outputs are passed through an API wrapper script and `Opticks` outputs are passed back through the same system [29]. Development of this API and its integration with `BACCARAT` are important next steps for this project. Iterative development of a robust CSG model of the LZ detector is also ongoing.

In order to fit the requirements of the NERSC computing ecosystem, a `Shifter` [37] container has been developed that allows `Opticks` to run on CoriGPU. `opticks_on_shifter` works by bundling `Opticks` with its dependencies and then mounting those on a writable partition on the NERSC NFS. This allows the final setup of hardware-aware software components to be carried out on the hardware on which the software must run. This should assist with portability from the CoriGPU development environment to the Permutter production system.

Predictions based on prior tests of `Opticks` on JUNO indicate reason to be optimistic that the new system has the potential to closely match the full chain for fidelity of modelling, and closely match the fast chain for speed.

References

[1] The LZ Collaboration, D. Akerib, C. Akerlof, D. Akimov, A. Alquahtani, S. Alsum, T. Anderson, N. Angelides, H. Araújo, A. Arbuckle et al., Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment **953**, 163047 (2020)

[2] The LZ Collaboration, D.S. Akerib, C.W. Akerlof, D.Y. Akimov, S.K. Alsum, H.M. Araújo, X. Bai, A.J. Bailey, J. Balajthy, S. Balashov et al., *LUX-ZEPLIN (LZ) conceptual design report* (2015), `1509.02910`

[3] B.J. Mount, S. Hans, R. Rosero, M. Yeh, C. Chan, R.J. Gaitskell, D.Q. Huang, J. Makkinje, D.C. Malling, M. Pangilinan et al., *LUX-ZEPLIN (LZ) technical design report* (2017), `1703.09144`

[4] E. Aprile, T. Doke, Reviews of Modern Physics **82**, 2053–2097 (2010)

[5] The LZ Collaboration, D. Akerib, C. Akerlof, S. Alsum, H. Araújo, M. Arthurs, X. Bai, A. Bailey, J. Balajthy, S. Balashov et al., Physical Review D **101** (2020)

[6] The LZ Collaboration, D. Akerib, C. Akerlof, A. Alqahtani, S. Alsum, T. Anderson, N. Angelides, H. Araújo, J. Armstrong, M. Arthurs et al., Astroparticle Physics **125**, 102480 (2021)

[7] NVIDIA, *NVIDIA RTX^{TM} platform*, https://developer.nvidia.com/rtx (2021), accessed: 2021-02-17

[8] S. Blyth, *Meeting the challenge of JUNO simulation with Opticks: GPU optical photon acceleration via NVIDIA® OptiXTM*, in *EPJ Web of Conferences* (EDP Sciences, 2020), Vol. 245, p. 11003

[9] S. Blyth, *Opticks: GPU Optical Photon Simulation for Particle Physics using NVIDIA® OptiXTM*, in *EPJ Web of Conferences* (EDP Sciences, 2019), Vol. 214, p. 02027

[10] S. Blyth, *Opticks: GPU Optical Photon Simulation for Particle Physics using NVIDIA® OptiX^{TM}*, in *Journal of Physics: Conference Series* (IOP Publishing, 2017), Vol. 898, p. 042001

[11] S.G. Parker, J. Bigler, A. Dietrich, H. Friedrich, J. Hoberock, D. Luebke, D. McAllister, M. McGuire, K. Morley, A. Robison et al., Acm transactions on graphics (tog) **29**, 1 (2010)

[12] NVIDIA, *NVIDIA OptiX 7.2*, https://raytracing-docs.nvidia.com/optix7/index.html (2020), accessed: 2021-05-05

[13] NVIDIA, *NVIDIA OptiX 7.2 – Programming Guide*, https://raytracing-docs.nvidia.com/optix7/guide/index.html (2020), accessed: 2021-02-17

[14] K. Morley, *How to Get Started with OptiX 7*, https://developer.nvidia.com/blog/how-to-get-started-with-optix-7/ (2020), accessed: 2021-02-17

[15] NVIDIA, *NVIDIA OptiX 6.5 – Programming Guide*, https://raytracing-docs.nvidia.com/optix6/guide_6_5/index.html (2020), accessed: 2021-02-17

[16] S. Van Der Walt, S.C. Colbert, G. Varoquaux, Computing in science & engineering **13**, 22 (2011)

[17] D. Crockford, *The application/json media type for javascript object notation (json)* (2006)

[18] Y.F. Li, *Overview of the Jiangmen Underground Neutrino Observatory (JUNO)*, in *International Journal of Modern Physics: Conference Series* (World Scientific, 2014), Vol. 31, p. 1460300

[19] S. Blyth, *Integration of JUNO simulation framework with Opticks : GPU accelerated optical propagation via NVIDIA® OptiX™*, in *PLACEHOLDER: Proceedings of vCHEP 2021* (2021)

[20] S. Eriksen, H. Flecher, B. Krikler, L. Kreczko, *Search for dark matter: Optical photon simulations at LZ* (2018), oracle's High Performance Cloud for Research and Innovation, `https://www.youtube.com/watch?v=mKxVEpB9IQo`

[21] S. Eriksen, *Using Optical Photon Simulations at LUX-ZEPLIN in the search for Dark Matter* (2019), physics Postgraduate Conference, `https://www.bris.ac.uk/physics/media/postgraduate/Postgraduate\%20Conference\%20Schedule.pdf`

[22] S. Eriksen, *LUX-ZEPLIN Optical Photon Simulations Using GPUs* (2019), sTFC High Energy Physics Summer School, `https://conference.ippp.dur.ac.uk/event/785/page/70-posters`

[23] NERSC, *National energy research scientific computing center*, `https://www.nersc.gov` (2021), accessed: 2021-02-18

[24] NERSC, *Cori*, `https://www.nersc.gov/systems/cori/` (2021), accessed: 2021-02-18

[25] S. Blyth, *Opticks: GPU Accelerated Optical Photon Simulation using NVIDIA OptiX*, `https://simoncblyth.bitbucket.io/opticks/index.html` (2021), accessed: 2021-02-18

[26] NERSC, *Cori GPU nodes*, `https://docs-dev.nersc.gov/cgpu/` (2021), accessed: 2021-02-18

[27] NERSC, *Perlmutter*, `https://www.nersc.gov/systems/perlmutter/` (2021), accessed: 2021-02-18

[28] NVIDIA, *CUDA GPUs | NVIDIA Developer*, `https://developer.nvidia.com/cuda-gpus` (2021), accessed: 2021-02-18

[29] O. Creaner, M.E. Monazi, L. Gerhardt, Q. Riffard, S. Eriksen, *Optical sims using GPUs*, `https://www.nersc.gov/assets/Uploads/1400-Opticks-on-CoriGPU.pdf` (2020), accessed: 2021-02-18

[30] NERSC, *NERSC documentation: Math libraries*, `https://docs.nersc.gov/development/libraries/` (2021), accessed: 2021-02-18

[31] NERSC, *NERSC documentation: Applications*, `https://docs.nersc.gov/applications/` (2021), accessed: 2021-02-18

[32] S. Blyth, *Opticks install instructions*, `https://simoncblyth.bitbucket.io/opticks/docs/install.html` (2017), accessed: 2021-02-18

[33] S. Blyth, *Opticks externals*, `https://simoncblyth.bitbucket.io/opticks/docs/externals.html` (2017), accessed: 2021-02-18

[34] G. Van Rossum et al., *Python Programming Language.*, in *USENIX annual technical conference* (2007), Vol. 41, p. 36

[35] S. Blyth, *Opticks testing, geocache creation, python setup*, `https://simoncblyth.bitbucket.io/opticks/docs/testing.html` (2017), accessed: 2021-02-18

[36] Docker, *What is a container?*, `https://www.docker.com/resources/what-container`, accessed: 2021-02-18

[37] D.M. Jacobsen, R.S. Canon, Proceedings of the Cray User Group pp. 33–49 (2015)

[38] Docker, *Docker documentation*, `https://docs.docker.com/`, accessed: 2021-02-18

[39] Docker, *Docker hub - container image library*, `https://www.docker.com/products/docker-hub`, accessed: 2021-02-18

[40] NERSC, *Using shifter at NERSC*, `https://docs.nersc.gov/development/shifter/how-to-use/` (2021), accessed: 2021-02-18

[41] NERSC, *Shifter: User defined images*, `https://www.nersc.gov/research-and-development/user-defined-images/` (2020), accessed: 2021-02-18

[42] O. Creaner, *opticks-on-shifter*, `https://gitlab.com/luxzeplin/sim/opticks-on-shifter` (2021), accessed: 2021-02-18

[43] O. Creaner, *opticks-rebuild*, `https://hub.docker.com/r/creanero/opticks-rebuild` (2021), accessed: 2021-02-18

[44] NVIDIA, *NVIDIA CUDA*, `https://hub.docker.com/r/nvidia/cuda/` (2021), accessed: 2021-02-18