Project Summary: Deploying GPU algorithms through SONIC

PI: Prof. Philip Chang (University of Florida)
Postdoc: Kelci Mohrman (University of Florida)
Project duration: Sep. 2023 - Aug. 2024

1 Overview

Starting with a brief overview, explain the goal of the project, the problem you aimed to address in your R&D project, and the method/process. What was the expected impact on the HL-LHC CMS S&C operations?

The goal of the project [1] was to implement a version of the Line Segment Tracking (LST) algorithm [2] with the SONIC framework [3] in order to enable flexible and efficient GPU usage. Because reconstruction tasks constitutes the largest fraction of CMS data processing, it is important to understand the resource requirements and to explore options for improving the efficiency of these steps. To this end, CMS is exploring reconstruction algorithms that are designed to make use of GPU resources. These include LST, which is a tracking algorithm that takes advantage of double-layer design of the HL-LHC outer tracker in order to perform hit correlations in a parallel way with GPUs. With more algorithms requiring GPU resources, it is important to understand the resource requirements and strategies for ensuring efficient deployment and usage. The SONIC framework provides the ability to make use of GPUs "as a service", enabling GPUs to be factored out of CPU machines. With this approach, the GPU-based servers may be remote from the CPU-based servers, allowing for more flexibility in the usage of GPU resources.

Our project thus aimed to demonstrate the successful implementation of the LST algorithm with SONIC, to demonstrate inter-site runs (with client CPUs and server GPUs at difference sites), and to explore the performance. If successful, this would help to improve the flexibility and efficiency of the resource utilization of the LST algorithm and contribute to a better understanding of the computational needs of the CMS experiment for the HL-LHC.

2 Techniques

What techniques did you use to address the problem? If more than one avenue were taken, what were the other techniques? And why were the chosen techniques expected to perform better than the alternative or baseline methods?

The SONIC framework was the main tool used in this project. We developed a custom LST backend for the LST algorithm, using the SONIC team's `TestIdentity` repository as an example setup [4]. We extracted relevant pieces of the LST `TrackLooper` repository [5] (at the `cuda_branch`, as described in Section 4) from its ROOT dependencies, and compiled these within the singularity environments provided in the `TestIdentity` setup.

On the client side, we created a modified version of the LST producer that passes the LST inputs to the server as a flat vector of doubles. On the server side, the inputs are decoded

and passed into the LST algorithm. The outputs of the LST algorithm are passed back to the client as a flat vector, which are decoded and saved to ROOT files (as in the standard non-SONIC LST setup). A summary of this workflow is presented in Figure 1.

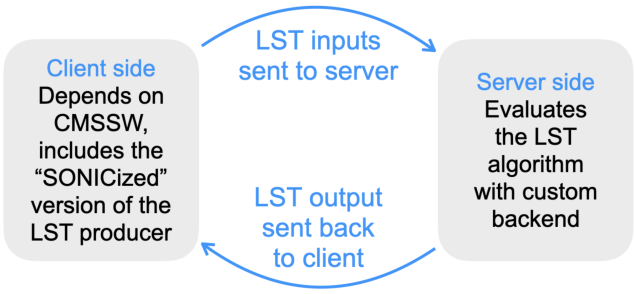

Figure 1: Schematic representation of the SONIC+LST workflow.

3 Outcome

Explain the outcome of the project, including alternative paths and their estimated impact as well. Was the outcome as expected as described in 3.? If not, why not?

In this project, we were able to successfully implement a version of LST with the SONIC framework, explore the performance, and demonstrate successful runs across multiple sites. The details of these main outcomes are summarized below:

- **Successful implementation of LST with SONIC:** As described in Section 2, a working version of the LST algorithm was implemented and run with the SONIC framework. Figure 2 shows an example efficiency plot produced with the SONIC implementation of LST. Instructions for setting up and running the SONIC+LST implementation may be found here [6].

- **Performance studies to explore scaling of SONIC+LST:** With the SONIC+LST setup, we tested running with multiple concurrent instances of LST `cmsRun` jobs and measured the runtime. For these trials, we were using the "step3" `cmsRun` configuration (as described in the LST readme [5]) and tested 1, 8, 16, 32, and 64 concurrent `cmsRun` jobs. We also tested with 4 threads (i.e. setting the `numberOfThreads` parameter in the `cmsRun` configuration file to 4); this is conceptually similar to simply launching more concurrent `cmsRun` jobs manually (except that in this case multiple threads would be able to share in one `cmsRun` and CMSSW handles the scheduling). For comparison, we also ran a similar configuration of the standard non-SONIC setup. However, it should be noted that this is not a fully equivalent comparison because, as explained in Section 4, the SONIC implementation is based upon an older version of the LST algorithm. As we scaled to larger numbers of concurrent jobs, we encountered crashes related to memory (as documented in the Sep 9, 2024 Tracking POG presentation [7]). The results of these timing runs are summarized in Figure 3. For these trials, there

was not an indication that the GPU is saturated (monitoring the `nvidia-smi` output during runs, we observed GPU usage from approximately 0-60%). It is thus likely that the scaling behavior observed in Figure 3 is influenced by other factors. Further studies would be required to fully understand this scaling behavior (as described in Section 5).

- **Inter-site runs:** The performance studies described above were run with both client and server at the Purdue Tier 2 site, but we were able to set up and run with the server and client at the University of Florida (UF) Tier 2 site as well. Furthermore, we were able to demonstrate successful runs with the client at Purdue the server at UF, as well as with the client at UF and the server at Purdue. It would be beneficial to explore the timing of these inter-site runs, as discussed in Section 5.

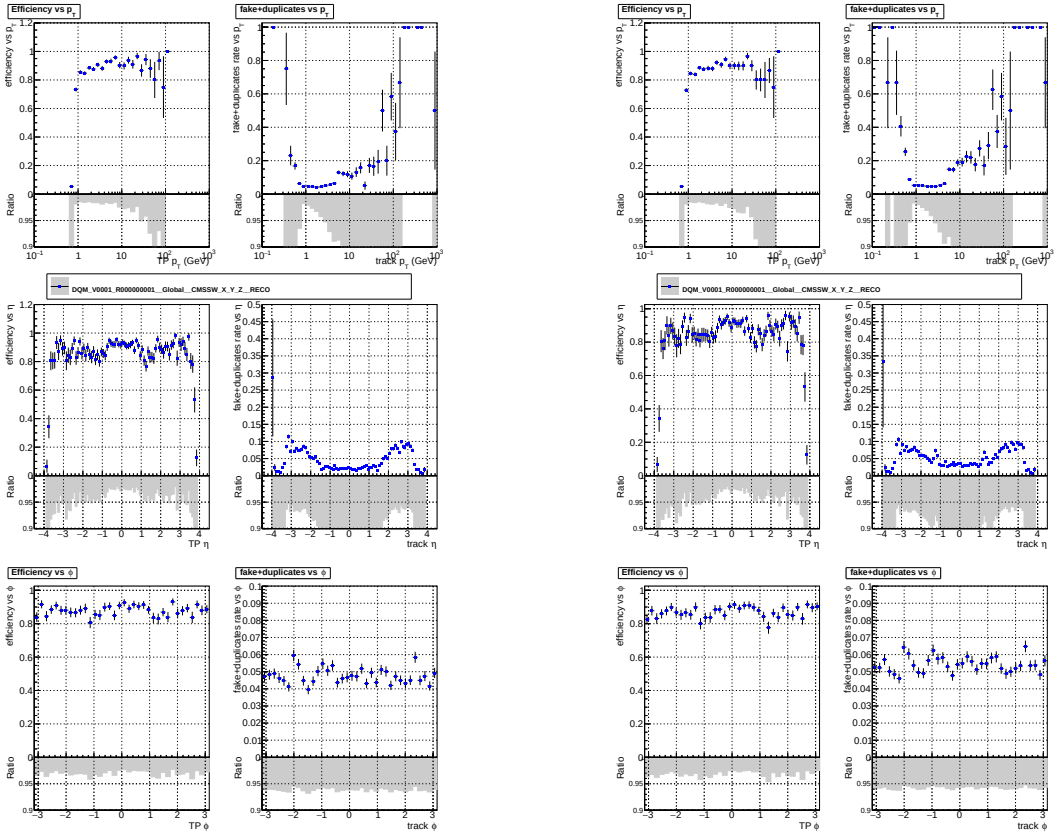

Figure 2: Efficiency plots (produced by the LST validation plotting script) for the standard non-SONIC run (left) and a run with the SONIC+LST workflow (run for 100 events). The non-SONIC setup is based on the master branch of LST (as of summer 2024, based on `CMSSW_14_1_0_pre3_LST_X`). As described in the text, the SONIC workflow is based on an older version of LST (from the `cuda_branch` branch). Although we do not expect identical results (because of the LST version differences), we can see there are no clear signs of significant qualitative differences in the validation plots.

4 Challenges

Was there a particularly difficult obstacle you encountered during this project? How did you navigate through it, and were there any pivotal breakthrough moments? What are some

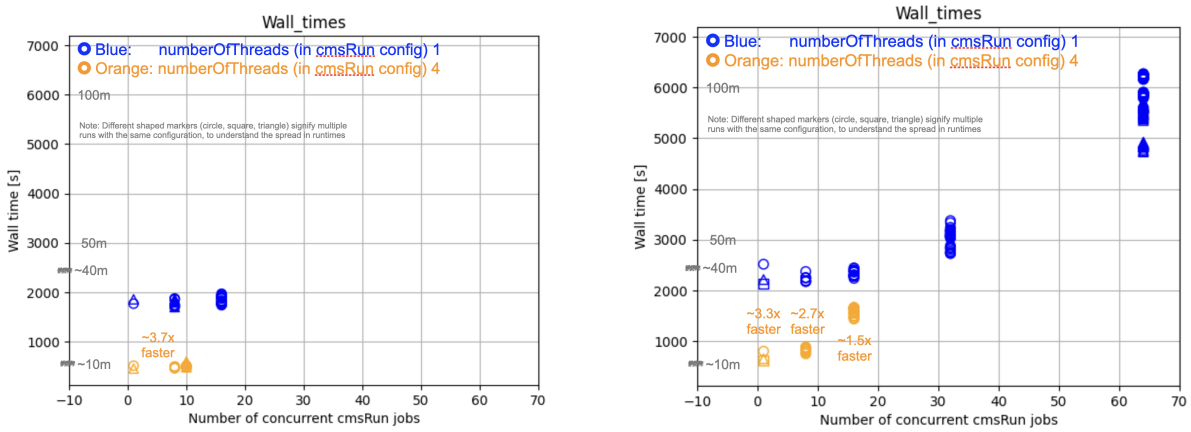

Figure 3: Timing runs with standard non-SONIC implementation of LST (left), and with the SONIC+LST workflow (right). The runs are processing 300 $t\bar{t}$ events. In blue, we show the results with a single thread (`numberOfThraeds` equal to one in the `cmsRun` configuration file), while the orange show the results with four threads. The different shaped markers (circle, triangle, and square) indicate different runs with the same configuration (to provide information about the spread of runtimes). These trials were run at the Purdue Tier 2 site, with the client and server at the same node. The node (with 128 hyper-threaded cores, and 512GB of RAM, and T4 GPU) was reserved for these trials (i.e. there would be no jobs from other users running on these nodes). For these runs, there were not indications that the GPU was saturated.

gained knowledge that others who may tackle similar problems in the future would want to know about beforehand?

One of the main challenges of this project was that the LST code was (and still continues to be) under active development, so the LST code changed significantly throughout the duration of this project. The most impactful of these changes was a move from `CUDA` to the `alpaka` library. However, `alpaka` is not available in the singularity environment used for the backend (provided in the example backend repository [4]). Because of this, the LST backend developed for this project is based on an outdated version of LST (from before the code moved away from `CUDA`). This makes validation of the SONIC implementation difficult, since the LST algorithm used in the SONIC setup is different from the current main branch of LST (which we were using for comparison).

Even though a full validation could not be performed, we were able to run the SONIC implementation through steps to create DQM validation plots, and compare these against plots created from the master LST code (as shown in Figure 2). While exact agreement is not expected (for the reasons explained above), we were able to observe that there were no clear signs of significant qualitative differences. However, in the future, would be important to work to update the LST backend to be in line with the current `alpaka` based master LST.

5 Next steps

What do you recommend for the future if the problem were to be tackled again, or to make improvements to your work?

The next next steps that could be investigated are summarized below:

- **Update the LST backend to be synchronized with the current LST main branch:** As introduced in Section 4, the LST backend was implemented with `CUDA`. The master version of LST has since been updated to use `alpaka`, so the SONIC LST backend should be updated to be synchronized with the updated LST algorithm. This will involve including `alpaka` in the server singularity environment.

- **Study timing and performance of the SONIC+LST implementation:** It would be important to better understand the timing studies summarized in Figure 3. For example, it would be good to understand what is causing the SONIC+LST runtime to increase with increasing number of concurrent `cmsRun` instances. One way to explore this would be to replace the LST backend with a simple pass-through (as opposed to the LST algorithm evaluationa0, and compare the runtime and scaling. Additionally, it could be useful to rerun the timing studies with a workflow that runs only the LST algorithm (instead of the more realistic "step3" workflow). Furthermore, it could be useful to explore tools for profiling the jobs in order to better understand what fraction of time is being spent on each step of the workflow. It would also be important to understand how these performance metrics compare between the SONIC and non-SONIC setups; however, in order to explore this aspect of the comparison, it would be useful to first update the SONIC implementation to the current master LST (based on `alpaka`), as described in the previous point.

- **Study the performance of inter-site runs:** While this project demonstrated successful inter-site runs, an important next step would be to explore the timing and performance of these runs. A preliminary observation (of a test run with the client at Purdue and the server at UF) indicated that the runtime is significantly longer (by about a factor of four) compared to a run in which the client and server were located on the same node at the same site. It would be important to perform repeated runs, carefully measure the timing, and explore the scaling with increasing concurrency of `cmsRun` jobs.

- **Integrating with main LST/CMSSW:** Moving forward, it would be important to explore options for integrating the SONIC+LST implementation into the main LST codebase.

6 Skills learned

What kinds of skills did you learn through the project? Do you think this experience has helped you understand how to develop and lead a research project?

This project allowed me to gain increased experience with a variety of general technical skills, including the usage of singularity environments, Triton servers, and GPU based workflows. Regarding experience and technical skills specific to CMS, I also gained more experience working with CMSSW and learned more about CMS tracking algorithms. Through frequent

discussions with mentors and PIs, I also had to opportunity to learn more about how to plan, organize, and execute technical research projects within CMS.

7 Presentations

How was your work presented to the wider CMS community? Could you provide a list of meetings, or conferences where the work was presented?

The work was presented primarily at CMS Tracking POG meetings:

- Sep. 9, 2024: 'LST with SONIC framework'" [7]
- Jun. 3, 2024: "Project status update: LST with the SONIC framework" [8]
- Feb. 12, 2024: "Project status update: LST with the SONIC framework" [9]
- Oct. 23, 2023: "Project introduction and plans: LST with the SONIC framework" [10]

References

[1] USCMS R&D Proposal "Deploying GPU algorithms through SONIC". https://uscms-software-and-computing.github.io/assets/pdfs/Kelci-Mohrman.pdf.

[2] Philip Chang et al. Segment Linking: A Highly Parallelizable Track Reconstruction Algtorithm for HL-LHC. *J. Phys. Conf. Ser.*, 2375(1):012005, 2022.

[3] Aram Hayrapetyan et al. Portable acceleration of CMS computing workflows with coprocessors as a service. 2 2024.

[4] "TritonCBE/TestIdentity/". https://github.com/yongbinfeng/TritonCBE/tree/main/TestIdentity#instructions-to-run-patatrack-aas-at-purdue.

[5] "SegmentLinking/TrackLooper". https://github.com/SegmentLinking/TrackLooper/tree/master?tab=readme-ov-file#build-tracklooper.

[6] "Setting up and running SONIC+LST". https://gist.github.com/kmohrman/015d7fa8f807099ff61e41e074e7124a.

[7] "LST with SONIC framework". https://indico.cern.ch/event/1443183/#50-update-on-soniclst-developm.

[8] "Project status update: LST with the SONIC framework". https://indico.cern.ch/event/1418266/#35-line-segment-tracking-using.

[9] "Project status update: LST with the SONIC framework". https://indico.cern.ch/event/1374894/#25-lst-running-on-gpus-as-a-se.

[10] "Project introduction and plans: LST with the SONIC framework". https://indico.cern.ch/event/1337451/#5-lst-running-on-gpus-as-a-ser.