Implementing and Investigating Performance of VBL++ on GPU vs CPU

Aditi Thanekar, Mentor: Jordan Penner

Lawrence Livermore National Laboratory(LLNL), University of California, Riverside, University of California, San Diego

At LLNL, the National Ignition Facility(NIF) uses an internally developed comprehensive nonlinear optical modeling software called Virtual Beamline(VBL++) to model every optical component in a laser chain. VBL++ is a computationally intensive numerical code, simulating the interaction of an electric field with different materials and capturing energy and damage metrics. Thus far, this software has been primarily geared towards the CPU, so there has been no prior comparison of its performance on GPU to the equivalent on CPU.

INTRODUCTION

VBL++ is the premier simulation modeling tool used to set up experiments on the NIF, used to design and operate modern high peak and average power laser systems. Capability to run simulations on GPU could potentially save laser physicists(our users) valuable time and allow batch jobs to run faster. Understanding how the VBL++ algorithms run on GPU will allow the developers to better understand where there are performance bottlenecks. This also benefits current releases of the software that use CPU parallelism and will allow us to explore if using supercomputing resources with GPUs instead of CPU-specific clusters could be more helpful to our users. Additionally, collecting a metric greatly help us understand when the addition of more threads no longer results in speedup (Amdahl's Law), and compare the GPU speed to the CPU version that is currently in use.

METHODS

**Note: GPU = device and CPU = host*

Although there is existing GPU code for VBL++, it had not been tested or kept up to date. I used a Debug version of the code without compiler optimizations to bring the device code to a working state on a small test case(see Figure 2), to show proof of concept that we can use GPUs and how its speed compares to CPUs.

1. **Simulation Test Case Set Up**
   - Simplest possible configuration in VBL++ showing linear propagation and illustrating use of FFTs.

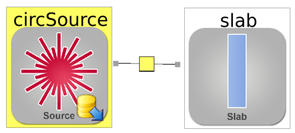


Figure 1. This test case shows a circular source connected to a slab, and has excludes set to true. It captures the effect of electric field generation, linear propagation, and surface interaction. This test case was used to run the simulation and collect statistics shown in Figure 2 and Table 1. This visualization is from the VBL++ GUI.

2. **Implementation**
   - Discovered segmentation faults caused by device loops incorrectly calling functions on host pointers
   - Created static functions to safely manage data from host pointers within device code, and allocate on unified memory
   - Extensively refactored code to prevent invalid memory access
   - Added use of CuFFTW instead of FFTW when using device

3. **Measuring Timings** – CPU and GPU from start to finish
   - 1024 x 1024 x 256 samples = **268,435,456 total samples**
   - 5 runs per trial, done on CPU(serial and parallel) and GPU

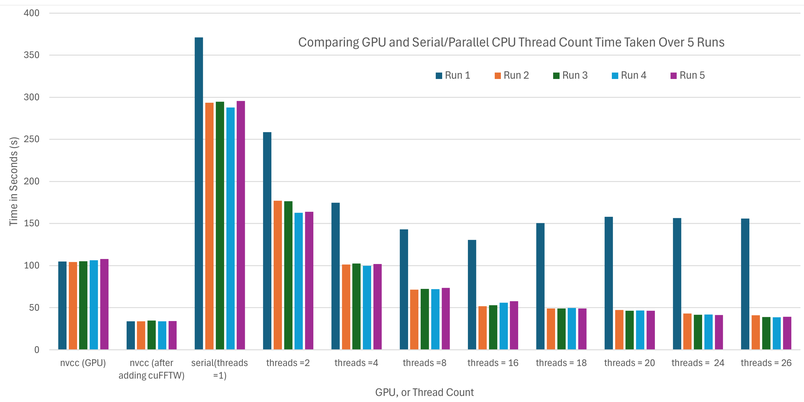
TECH STACK

VBL++ is primarily based upon C++, and utilizes third-party libraries. Its GPU loops and memory allocation are based on LLNL projects RAJA and Umpire, and is compiled with gcc for CPU or Nvidia's nvcc for GPU.

Table 1. Comparing Amount of Speed Up on GPU vs CPU Parallel Threaded versions with Test Case in Figure 1

Run	Time(s) on Serial CPU (1 thread)	Time(s) on parallel CPU (16 threads)	Time Taken(s) on GPU	Relative Speedup (CPU parallel vs GPU)
1	371.225	130.604	33.711	3.87x
2	293.571	51.663	33.883	1.52x
3	294.756	52.795	34.555	1.53x
4	287.913	55.963	33.812	1.66x
5	295.532	57.733	34.015	1. 70x

Figure 2. Time in Seconds for 5 Consecutive Runs on Debug Mode for a simple test case(shown in Figure 1), tested on nvcc(GPU) before and adding CuFFTW, serial CPU(1 thread), and on different amounts of parallel CPU threads



HURDLES ENCOUNTERED

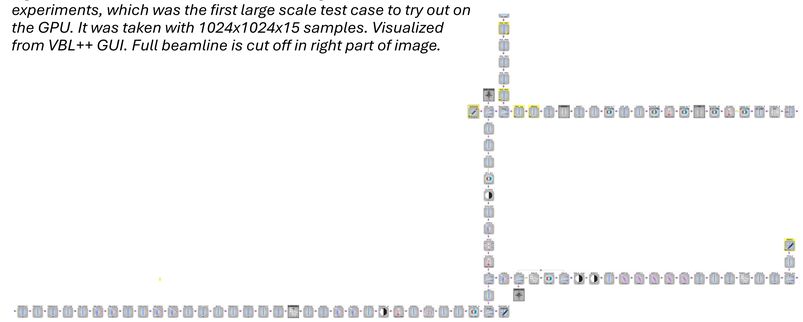
- GPU code had been implemented but not maintained through successive versions, and had logical errors and memory faults in every RAJA loop
- Large amounts of refactoring after changing getters and setters to be static
- Debug version did not properly reflect the Release version, which contains compiler optimizations, leading to bugs not identified in initial testing
- Testing out large scale test cases takes time (3 hour simulations).

RESULTS

- First successful test case run of VBL++ on GPU (nvcc)
- 3.1x average speedup on GPU via use of CuFFTW compared to prior FFTW
- Validated use of unified memory- modifiable from host and device
- GPU runs beat every CPU initial run, and CPU cached run(run after first run)
- CPU initial run time improves with more threads, but plateaus after 16 threads
- Every CPU run after the 1st initial run was faster, due to caching

VBL++ GPU RUN AT SCALE

Figure 3. A full-scale NIF test case simulating a beamline used in experiments, which was the first large scale test case to try out on the GPU. It was taken with 1024x1024x15 samples. Visualized from VBL++ GUI. Full beamline is cut off in right part of image.



CONCLUSIONS

- VBL++ successfully runs on GPUs and is faster than CPUs in all tested cases, even while CPUs were allowed to cache parameters
- Fast Fourier Transforms(FFTs) are a computational bottleneck, but the use of CuFFTW provides faster processing when run on a GPU.
- Setting CPU thread count to the maximum may not be optimal, as improvement from additional threads reaches a limit (Amdahl's Law)
- VBL++ GPU build(nvcc) produced correct results on NIF-scale test case(Figure 3), and further optimization is likely to yield additional performance gains

FUTURE GOALS

- Targeting ODE solvers to optimize Sundials library for GPUs
- Further profiling to find our slowest spots of code and speed them up
- Enable VBL++ GUI to have an option to run on GPU supercomputing cluster and build for a Release version

Lawrence Livermore National Laboratory

LLNL-POST-2009682-REV-1

Prepared by LLNL under Contract DE-AC52-07NA27344.