Performance analysis of GPU accelerated raytracing for narrow-band level sets using NanoVDB

Hiti Mario, 01327428

April 12, 2022

Abstract

OpenVDB and NanoVDB are data structures which were initially developed for image and movie rendering. However it is possible to use the provided frameworks within the context of semiconductor process simulation. This paper aims to benchmark the raytracing performance of NanoVDB and OpenVDB for said application in a worst-cast scenario on a scientific computing cluster. NanoVDB, which is a mini-version of OpenVDB, is also compatible with common graphics APIs. Therefore the behavior on GPUs is examined as well.

Contents

2 Methodology

The benchmark is designed to be baseline for future applications that are using NanoVDB for narrow-band level-sets and raytracing within the context of Semiconductor process simulation. Therefore no specific problem is chosen but the worst-case scenario in a typical application is modelled.

Source code, build instructions and measurement data are available at [1]:

`https://github.com/hitimr/SelectedTopicsCompElectronics`

2.1 Hardware setup

The benchmark is performed on a single node of a scientific cluster provided by TU Wien. The node consists of the devices listed in Tab. 1 which are both used for the benchmark.

CPUs and GPUs are different platforms in terms of architecture and design which makes a fair comparison with regards to their technical aspects difficult. However both devices are similar in cost of acquisition and operating expenses (i.e. power usage). Furthermore both platforms are marketed towards scientific computing.

Table 1: Hardware used for the benchmark. Prices may fluctuate due to current events. Power consumption represents the absolute maximum ratings according to the vendor

	Price	Power Consumption	Cores
Intel Xeon 6248	€ 3.300	105W	20 Cores; 40 Threads
NVIDIA Tesla T4	€ 2.500 - 3.000	70W	2.560 CUDA-Cores

2.2 Simulation environment

A common problem in Semiconductor process simulation is light being cast into a trench with semi-reflective walls as shown in Fig. 1 (left). To simplify the program and enforce a worst-case scenario the following modifications are performed:

- Rays leaving the bounding box (i.e. shooting into the sky) are cheaper to compute but do not contribute to the simulation. In order to prevent these edge cases rays are cast onto the inner surface of a hollow sphere.
- The point source is replaced with a volumetric source. Otherwise every ray would start within the same voxel which would lead to a beneficial memory access pattern.
- Depending on the reflecting angle, rays may cover different distances. Therefore the inner sphere (ray source) is offset to create a distribution of distances.
- Rays are shuffled in memory before being passed to the kernel to prevent beneficial memory access patterns.
- Any ray reflection on the surface is equivalent to having 2 separate rays (inbound and outbound) at the intersection point. Therefore reflections do not need to be modelled.

[1]Pre-Release version. The project is not finished yet

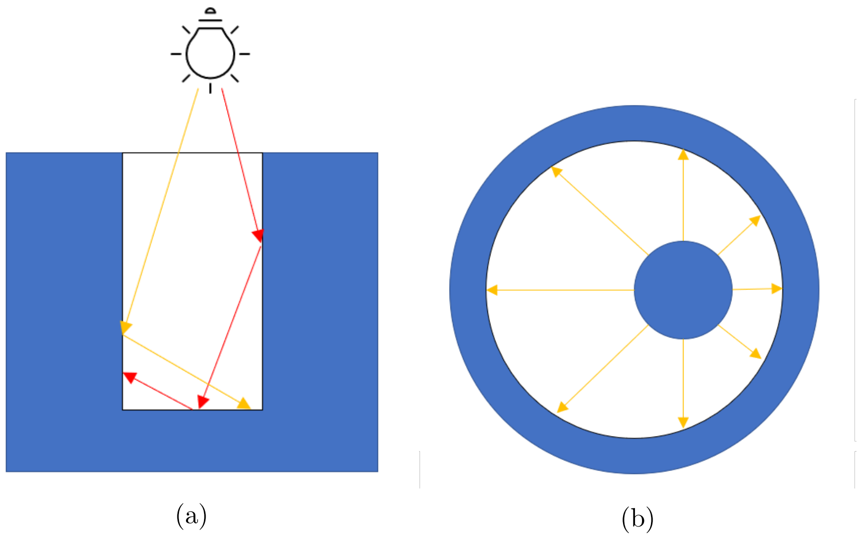


Fig. 1: Benchmark setup. **a** Trench being illuminated by a light source. Every ray has a chance of being reflected or absorbed. **b** 2D cross-section of the modified setup. For the benchmark rays are distributed evenly across the surface of the inner sphere.

Origin and direction of every ray along with a ground truth are precomputed and passed to three different ray intersection kernels:

- OpenVDB (CPU)
- NanoVDB (CPU)
- NanoVDB (GPU)

The OpenVDB kernel servers as a baseline for comparison. Both NanoVDB kernels are identical but launched on different platforms.

Only the time required to calculate intersections is measured. Memory management, data transfer, ray generation, result verification, etc. is not included. After the benchmark is complete the number of calculated rays per second is derived using

$$Rps = \frac{ray\ count}{time} = [\frac{1}{s}] \tag{1}$$

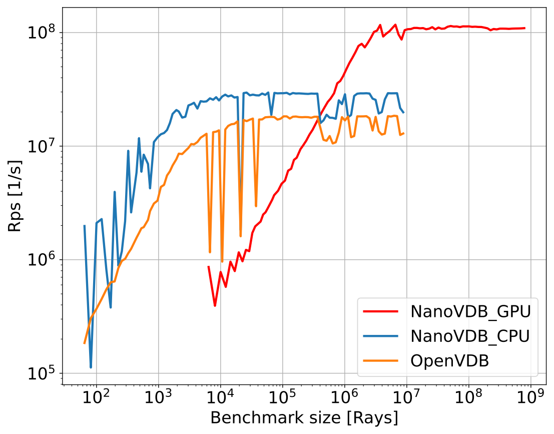
The benchmark is repeated while increasing the number of rays until no further increases in *Rps* is observed. After each iteration the resulting intersections are compared to a pre-computed ground truth to assure the correctness of the results. The asymptotic behaviour of the resulting performance curve is used to estimate a potential performance gain for switching to NanoVDB or GPUs.

3 Results

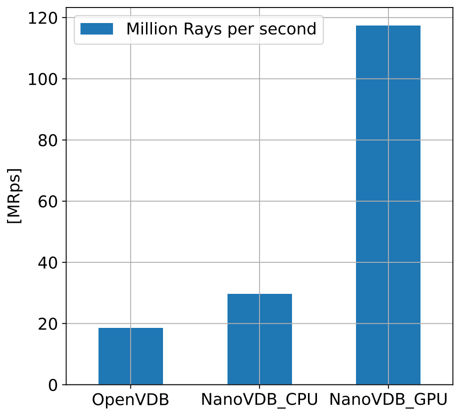
Fig. 2 shows the achieved performance for different problem sizes on all three kernels. NanoVDB achieves overall better results on the CPU compared to OpenVDB. Above one million rays the GPU starts to overtake both CPU kernels.

OpenVDB achieves up to 18.5 MRps [2]. NanoVDB consistently outperforms OpenVDB and reaches up to 29.7 MRps. For problems with 1 million rays or more the GPU kernel overtakes both CPU implementations and achieves up to 117.4 MRps. Therefore the switch from OpenVDB to NanoVDB increased performance by a factor of 6.3.

Furthermore both CPU implementations suffer from random drops in performance while GPU results are more consistent.



(a)



(b)

Fig. 2: **a** measured performance across different problem sizes. **b** Best results for each kernel

References

[1] Ken Museth. “NanoVDB: A GPU-Friendly and Portable VDB Data Structure For Real-Time Rendering And Simulation”. In: *ACM SIGGRAPH 2021 Talks*. SIGGRAPH ’21. Virtual Event, USA: Association for Computing Machinery, 2021. ISBN: 9781450383738. DOI: 10.1145/3450623.3464653. URL: https://doi.org/10.1145/3450623.3464653.

[2] Ken Museth et al. “OpenVDB: An Open-Source Data Structure and Toolkit for High-Resolution Volumes”. In: *ACM SIGGRAPH 2013 Courses*. SIGGRAPH ’13. Anaheim, California: Association for Computing Machinery, 2013. ISBN: 9781450323390. DOI: 10.1145/2504435.2504454. URL: https://doi.org/10.1145/2504435.2504454.

[2] 1 MRps = 10^6 Rays per second