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

Santa Claus

August 15, 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

1 Introduction

Semiconductor processing steps may involve exposure of light or the implementation of atoms using focused beams of Ions. These processes often take place at the scale of nanometers which make both process design exceedingly difficult. The surface structure of semiconductors is often too complex for analytical models and quality control often requires expensive equipment such as electron microscopes.

Due to the increase of computing power within the last decades it has now become feasible to simulate many steps of the production process. However modern hardware is still far from being able to simulate every single atom in a focused ion beam or every single photon from a light source. Therefore simulations usually resemble an approximation using a limited amount of virtual rays or atoms. However in general simulations usually benefit from an increase in computed elements (i.e. the more the better).

Many simulations involve rays and ray-casting of some sort which is a very common technique in the gaming and movie industry. These types of computations are in fact so important that many devices offer dedicated graphical processing units (GPUs) for those problems.

An important part of every simulation is the choice of underlying data structure. One option is OpenVDB which can be used to efficiently store high resolution volumes. [4]

OpenVDB was initially developed for the CGI and movie industry. Due to it's flexibility it is also possible to adapt it for use in semiconductor process simulation [2]. In a recent article NVIDIA published a benchmark that presents a significant speed-up when using NanoVDV.

Table 1: Benchmark results published by NVIDIA. The benchmark setup and source code are undisclosed. [1]

	OpenVDB (TBB)	NanoVDB (TBB)	NanoVDB (CUDA)	CUDA Speed-Up
Level Set	148.182	11.554	2.427	5x
Fog Volume	243.985	223.195	4.971	44x
Collision	-	120.324	10.131	12x

However in 2021 the project introduced NanoVDB which is compatible with common graphics APIs such as CUDA, OpenCL, OptiX, etc. [3].

For the process simulations the results for level-set raytracing are of significant importance. According to Tab. 1 NanoVDB on a GPU should be 60x faster compared to a multithreaded implementation using OpenVDB. (Execution time of 2.427ms vs 148.182ms).

However since the benchmark setup and source code are no published it is not clear if the same increase in performance can be achieved for other applications. Therefore the goal of this paper is to verify these results using a benchmark that is tailored to typical applications within the semiconductor process simulation.

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. 2 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 2: Hardware used for the benchmark. Prices may fluctuate due to current events. Power consumption represents the absolute maximum ratings according to the vendor

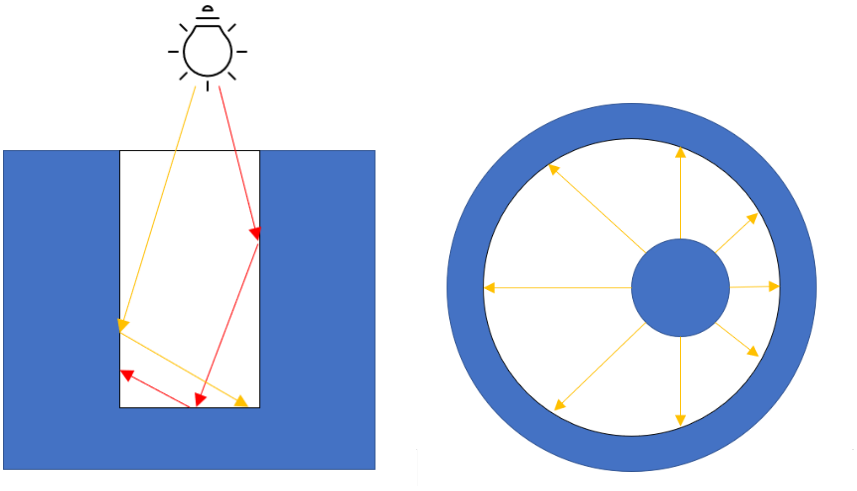
	Price	Power Consumption	Cores
Intel Xeon 6248	€ 3.300	105W	20 Cores; 40 Threads
NVIDIA Tesla T4	€ 2.500 - 3.000	70W	2.560 CUDA-Cores

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



(a)

(b)

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.

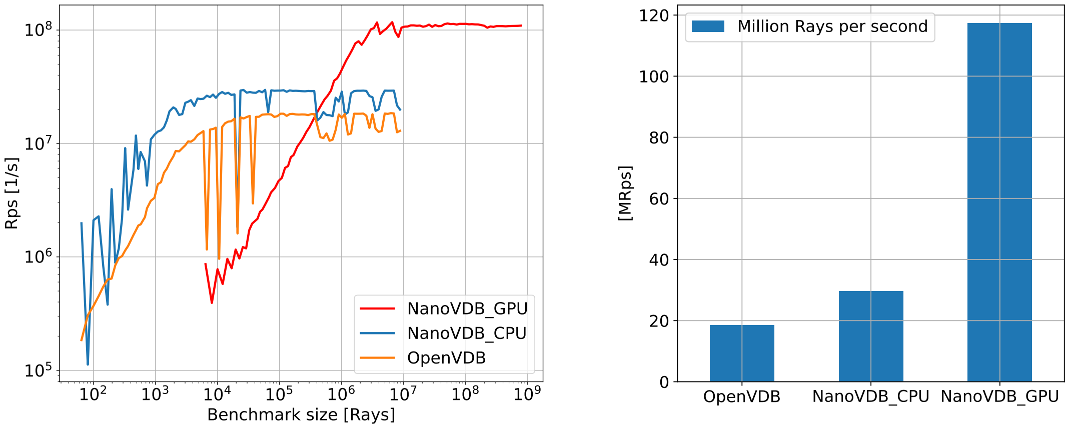
3 Results

3.1 Performance

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

3.2 Accuracy

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

4 Discussion and Outlook

As shown in Fig. 2 NanoVDB does provide better performance compared to OpenVDB on both platforms. Since both frameworks use different data structures significant adaptations to the code base are required for a project to migrate to NanoVDB. The expected increase in performance for CPU based systems can be considered too small to justify the amount of necessary work.

However for machines with a dedicated GPU a switch to NanoVDB can be very attractive especially for large simulations. Furthermore no adaptations to the codebase are required to launch kernels on either CPUs or GPUs. This also allows to combine both platforms to perform calculations on both in parallel.

With its current implementation the main limiting factor of NanoVDB seems to be the kernel for ray intersections which is not well optimized GPUs as it includes 4 branches (2 if and 2 while). Branches are usually no problem for CPUs due to their branch-prediction functionality. However on GPUs with a SIMT [3] Architecture 1 branch may force multiple threads to idle until all branches in a thread block are merged again.

Since 2018 NVIDIA is also selling GPUs with hardware accelerated raytracing capabilities (RTX 20X0 and 30X0 series). Due to its proprietary license the exact functionality of this technology is not part of the public domain. But a recent analysis [5] shows that a performance increase of up to one order of magnitude is possible in certain scenarios. However additional research and testing is required to determine of NanoVDB is compatible with this technology.

[3] Single Instruction Multiple Thread

References

[1] Wil Braithwaite and Ken Museth. *Accelerating OpenVDB on GPUs with NanoVDB*. 2020. URL: https://developer.nvidia.com/blog/accelerating-openvdb-on-gpus-with-nanovdb/ (visited on 03/19/2022).

[2] Paul Ludwig Manstetten. "Efficient flux calculations for topography simulation". PhD thesis. Wien, 2018.

[3] 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.

[4] 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.

[5] VV Sanzharov, Vladimir A Frolov, and Vladimir A Galaktionov. "Survey of nvidia rtx technology". In: *Programming and Computer Software* 46.4 (2020), pp. 297–304.