Implementation of Radio wave propagation loss calculation using RT core

Shinya HASHINOKI[†] ,Satoshi OHSHIMA
Takahiro KATAGIRI, Toru NAGAI

Nagoya University

Poster Presentation poster103s1

hashinoki@hpc.itc.nagoya-u.ac.jp[†]

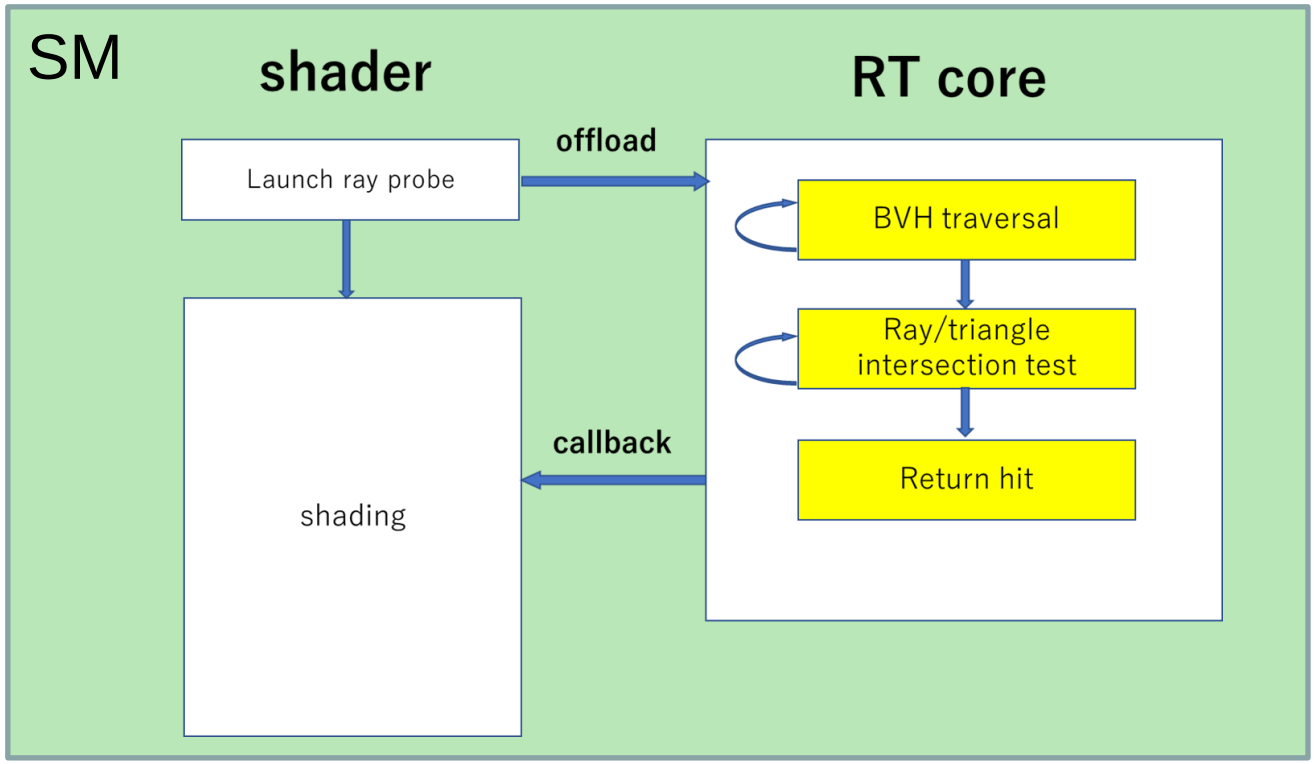
Background and objectives

- In recent years, GPUs have shown remarkable performance improvements and are used as computation accelerators in various fields such as numerical simulation and machine learning.
- Some recent GPUs are equipped with dedicated hardware that enables high-speed ray tracing processing.
- Ray tracing processing can be applied to applications other than image rendering.
  - ✓ We are planning to use it in the field of computational science.

We implemented the computation of radio propagation loss using the ray tracing acceleration hardware (RT core) of the NVIDIA GPU by using OptiX programming environment and evaluated its performance by comparing with CPU using OpenMP and GPU using OpenACC.

RT core

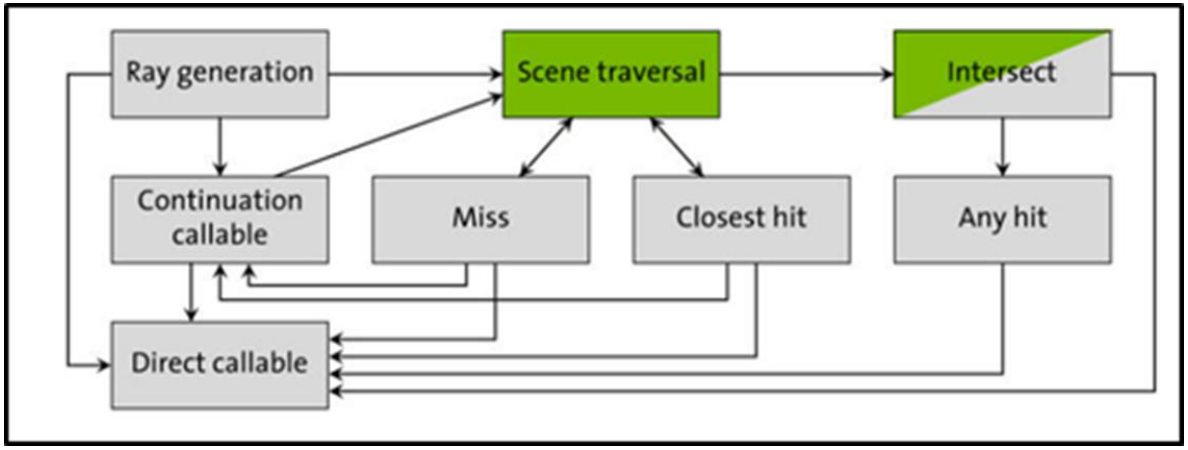
RT core is hardware that accelerates ray tracing processing. RT core takes care of some of the calculations for the ray tracing process, which speeds up the overall process.



OptiX 7

The NVIDIA OptiX 7 API is an application framework for achieving optimal ray tracing performance on the GPU.

- provides a simple, recursive, and flexible pipeline for accelerating ray tracing algorithms

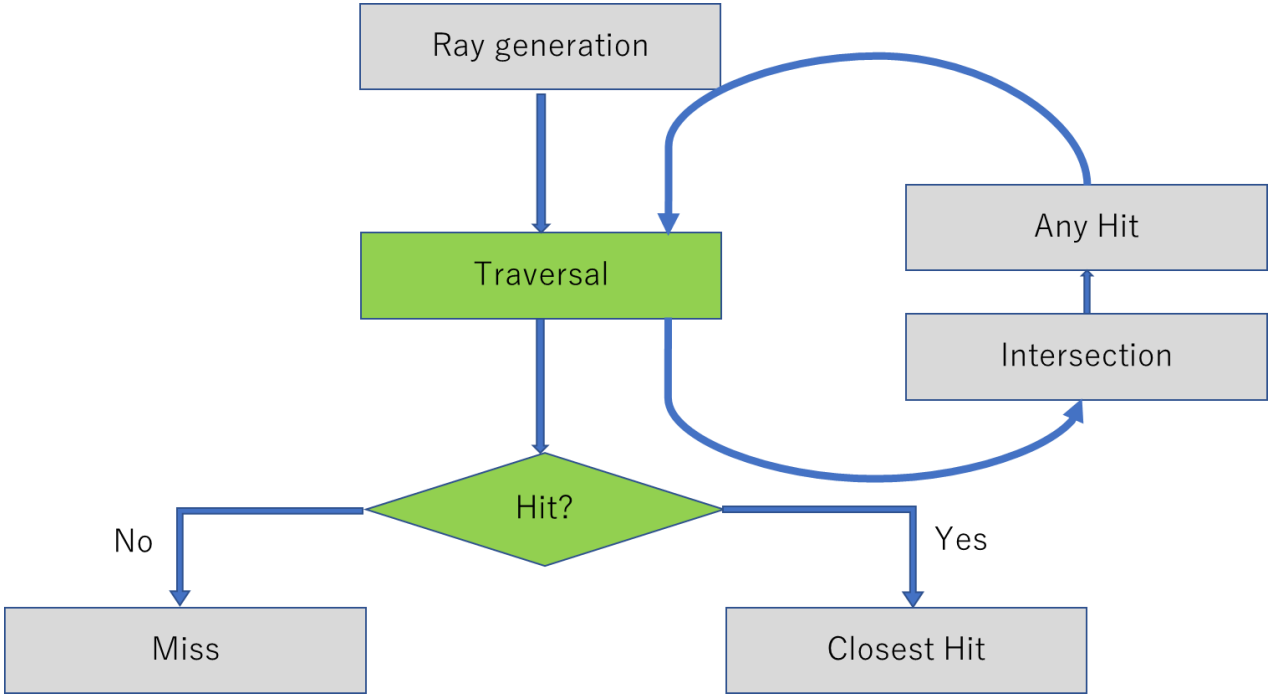


OptiX 7 programming model

NVIDIA https://raytracing-docs.nvidia.com/optix7/index.html (Accessed Nov. 22th 2021)

OptiX 7

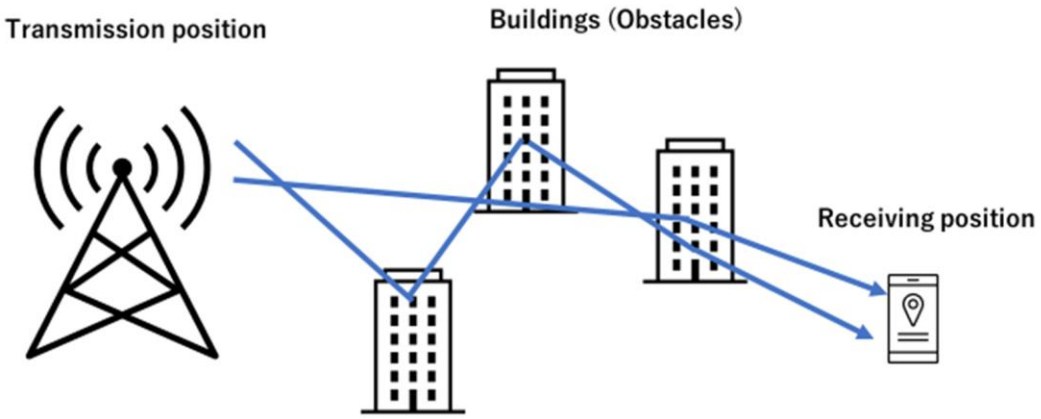
For example, OptiX 7 can be applied to an algorithm that performs the following ray tracing calculations inside it.



Radio wave propagation loss calculation

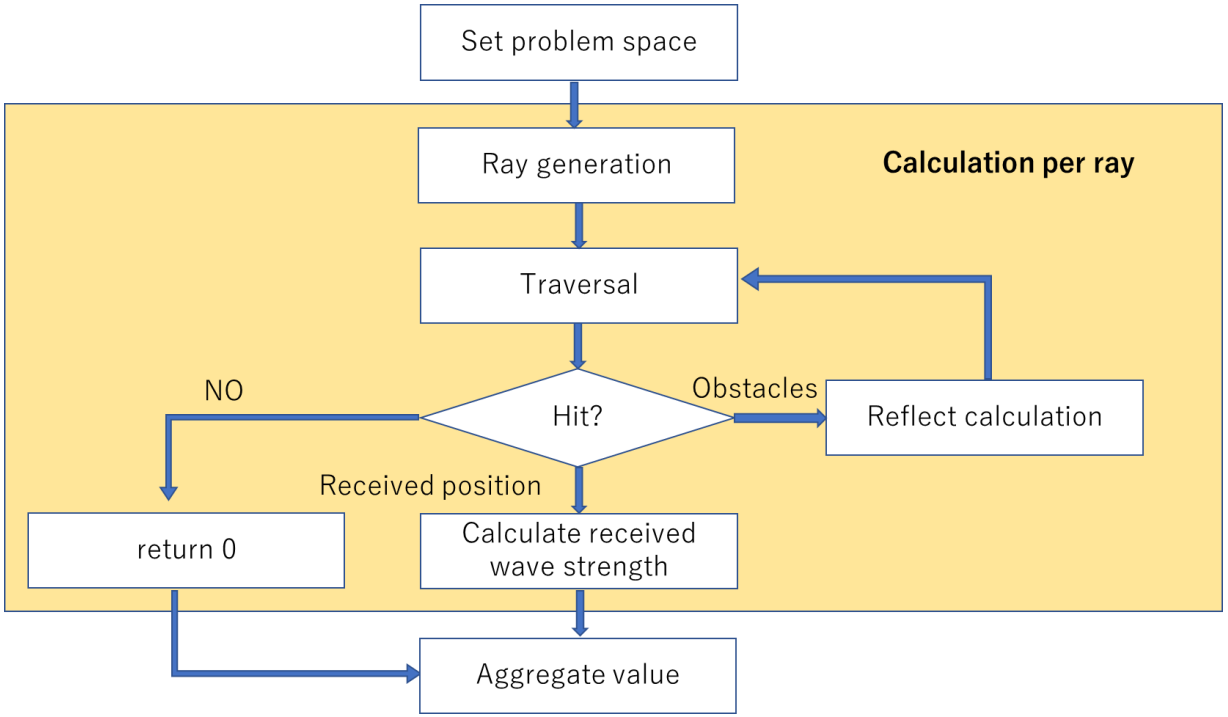
The radio wave propagation loss calculation is used to estimate the radio wave propagation characteristics by simulation.

- With the transmit power as input, the receive power after radio wave propagation can be obtained.
- can speed up this calculation by treating the transmitted radio waves as rays and applying a ray tracing process



Implementation

The radio wave propagation loss calculation proceeds in the following flow. This was implemented by applying the OptiX programming model.



OptiX program example

Data structure settings and data transfer were described in a cpp file.

```
int main( int argc, char* argv[] )
{
    std::string outfile;
    int         width  = 20000u;   //1024;
    int         height = 10000u;   //768;
    int         depth  = 8u;
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
try
    {
        char log[2048]; // For error reporting from OptiX creation functions
        //
        // Initialize CUDA and create OptiX context
        //
        OptixDeviceContext context = nullptr;
        {
            // Initialize CUDA
            CUDA_CHECK( cudaFree( 0 ) );

            CUcontext cuCtx = 0;  // zero means take the current context
            OPTIX_CHECK( optixInit() );
            . . .
launch      OPTIX_CHECK( optixLaunch( pipeline, stream, d_param,
            sizeof( Params ), &sbt, width, height, /*depth=*/1 ) );
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
}
```

Set problem size

Set problem space

cpp file source code

The user program described in the previous slide was written in a cu file.

```
#include <optix.h>

#include "wave_compute.h"
#include <cuda/helpers.h>
    . . .

extern "C" __global__ void __raygen__rg()
{
    . . .
}

extern "C" __global__ void __miss__ms()
{
    . . .
}

extern "C" __global__ void __closesthit__ch()
{
    . . .
}
```

Ray generation

Miss

Closest hit

cu file source code

Performance Evaluation

The experimental environment is as follows

- CPU: **Intel(R) Core(TM) i7-7700 CPU @ 3.60GHz**
- GPU: **NVIDIA GeForce RTX 2080 super**
  - ✓ Driver Version : 470.82.00, CUDA version : 11.4
- OS : **Ubuntu 20.04.3 LTS**

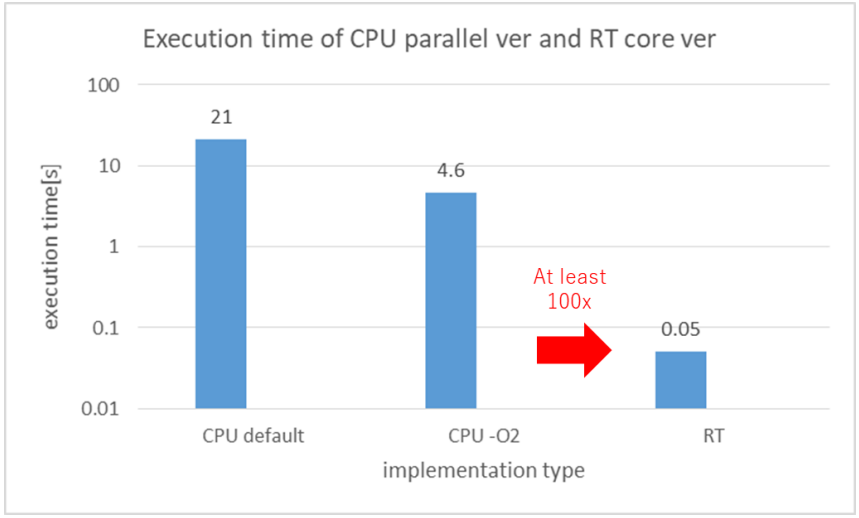
The radio propagation loss calculation was implemented on the GPU + RT core version using OptiX and the performance was compared with the CPU thread parallel version using OpenMP and the GPU version using OpenACC.

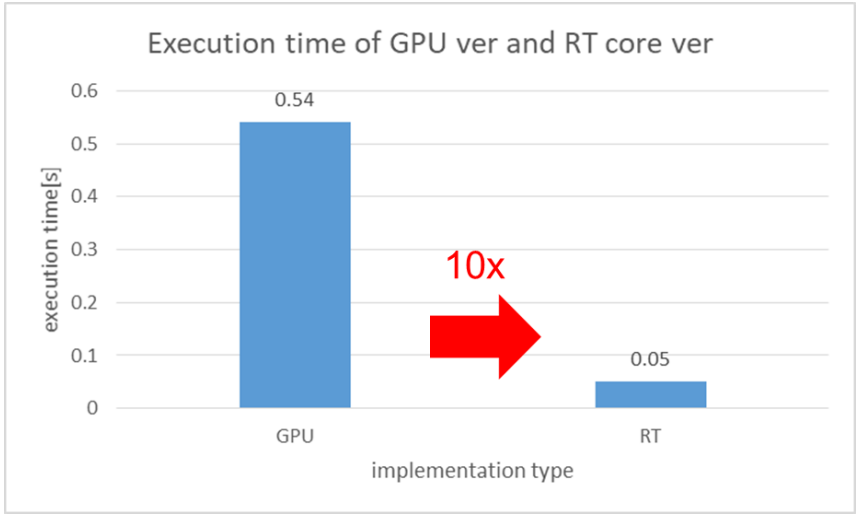
- ➢ Parallelism is performed for each ray, and its size (problem size) is represented in two dimensions as (x, y).

Execution time comparison

The key conditions for performance evaluation are as follows

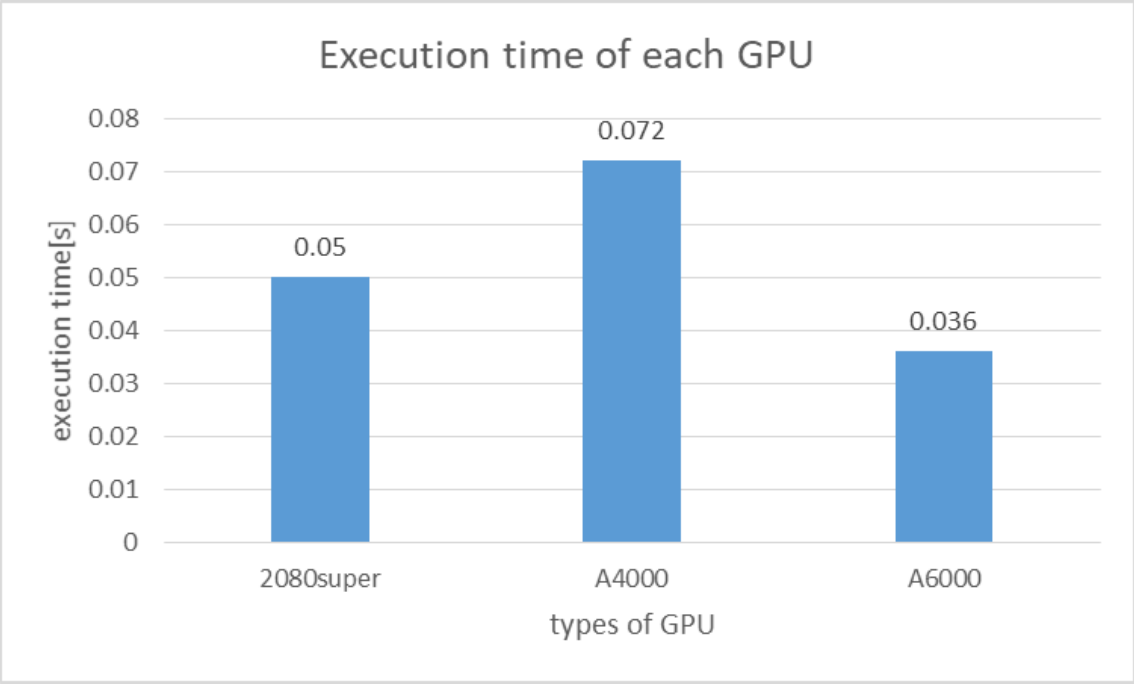
- The problem size is fixed to (20000, 10000).
- CPU parallel ver was measured with compiler (g++) optimization (-O2) and the default one. The number of threads is 8.
- The resource allocation for the GPU version was set to 48 for gang and 32 for vector.





Performance of the implementation using OptiX on various GPUs

	RT cores	speed	notes
GeForce RTX 2080 super (driver version 470.82.00)	48 (gen1)	1815 MHz	CPU: Intel(R) Core(TM) i7-7700 OS: Ubuntu 20.04.3 LTS
NVIDIA RTX A4000 (driver version: 470.57.02)	48 (gen2)	1560 MHz	CPU: Xeon W2255 OS: Ubuntu 20.04.3 LTS
NVIDIA RTX A6000 (driver version: 470.74)	84 (gen2)	1800 MHz	CPU: EPYC7313 OS: Ubuntu18.04.6 LTS



- The problem size is fixed to (20000, 10000)

Concluding remarks and future work

- A radio propagation loss calculation using RT cores was implemented using the OptiX framework.
- Implementation using RT cores was compared with parallel execution of CPU and GPU versions.
  - The RT core version was at least 100 times faster than the CPU version.
  - The RT core version was 10 times faster than the GPU version.

Future work

While OpenMP and OpenACC, which were used for parallel execution of the CPU and GPU versions, can be programmed by simply inserting directives, OptiX requires a description based on a programming model, and closing this gap is a future issue.