USAGE OF GPGPU COMPUTING IN LEO SATELLITE NETWORK SIMULATIONS

Mikko Majamaa, Janne Kurjenniemi, Lauri Sormunen, Jani Puttonen
Magister Solutions Ltd., Sepänkatu 14 C 16, FIN-40720 Jyväskylä, Finland
Email: {firstname.lastname@magister.fi}

Abstract

Low Earth Orbit (LEO) satellite constellations have been under intense research and development activities in recent years. These satellite networks aim to tackle the latency problems associated with the Geostationary Earth Orbit (GEO) satellites while still providing wide area coverage in case of large constellations. Satellite movement, non-uniform, and evolving user demand as well as coordination with other systems and regulatory constraints, sets new problems for large LEO constellations. Satellite payload flexibility and reconfigurability will be key features of on-going and future constellation projects. Development of these new systems set also new requirements for simulation tools used to develop Radio Resource Management (RRM) algorithms and there will be a need to run large scale simulation with high number of satellite and end users. These kinds of simulations will be complex and time-consuming, thus new solutions are needed to speed-up the simulations. With the aid of graphics processing unit (GPU) in general-purpose computing one can execute programs exploiting GPU's massive parallelism to reduce simulation time. The objective of this paper is to study the usage of GPGPU in satellite network simulations and evaluate the gains in simulator run-time performance.

1. Introduction

1.1 Satellite network simulator

The developed satellite network simulator is used to evaluate Low Earth Orbit (LEO) satellite network performance and to develop new radio resource management (RRM) algorithms. The studied scenario will be similar as in *ESA project AO9272: Flexible Resource Allocation Techniques for NGSO Constellations* ([1]) and the purpose of the RRM algorithm is to provide satellite network capacity throughout the globe so that in crowded areas, e.g. large cities, there are enough radio resources for all the users, also called user terminals (UT). On the other hand, the RRM algorithm needs to make sure that no excess radio resources are provided to areas where they are not needed. In other words, the task of the RRM algorithm is to balance radio resources throughout the globe taking into account non-uniform user distribution as well as non-uniform traffic demand. In the studied system, there are 198 LEO satellites (n_{SATs} = 198) that provide radio resources for varying amount of UTs (n_{UTs}), baseline being 150 000.

The simulator's main loop works in the following steps:

1. Propagate the satellites positions.

2. Compute link budgets (a measure of signal's strength that takes into account all the losses and gains of the signal from the transmitter to the receiver) for every UT to every satellite.

3. Pass the link budget data to the RRM algorithm. RRM algorithm allocates and optimizes radio resources to the UTs taking into account non-uniform traffic and user distribution.

4. Compute interference between different UTs allocated to same frequency bands.

5. Compute signal-to-noise-and-interference ratios (SINR) for the UTs.

6. Compute throughput based on calculated SINRs.

Output of the network simulator includes several statistics to evalute satellite network performance e.g. cumulative distribution functions (CDF) of the count of the satellites visible to the UTs, the capacity of the satellites' resources used and the SINRs and throughputs for the UTs.

1.2 GPGPU computing

Numba [2] is chosen to be used in the implementation because existing parts of the simulator are written in Python. Numba is an open-source just-in-time (JIT) compiler that can be used to translate a subset of Python code into machine code. It can also be used as an interface to CUDA, i.e. one can write a subset of Python code and Numba translates it to be executed on GPU. Writing GPU accelerated code with Numba is very similar to writing CUDA C.

1.3 Implementation

Usage of GPGPU is implemented to the link budget and SINR computations using Numba as an interface for computation on GPU. In the system, interference between the satellites is assumed to be insignificant, so the link budgets and SINRs can be computed in parallel. When the link budget computation is called in the simulation's main loop, the host (CPU) invokes threads to be executed on the device (GPU) so that each thread is responsible for computing a link budget for a UT related to a satellite. A result matrix in which an element C_{jk} corresponds to the j^{th} UT's link budget related to the k^{th} satellite is the output. The result matrice's dimensions are n_{UTs} x n_{SATs}. So, there are approximately 30 million threads invoked when the baseline of n_{UTs} is used.

2 Simulation results

Simulations are run on Microsoft Azure cloud computing environment with a setup that uses a half of an NVIDIA Tesla K80 card, the half has 2496 processor cores [3], and six cores of Xeon E5-2690 v3 (Haswell) processor.

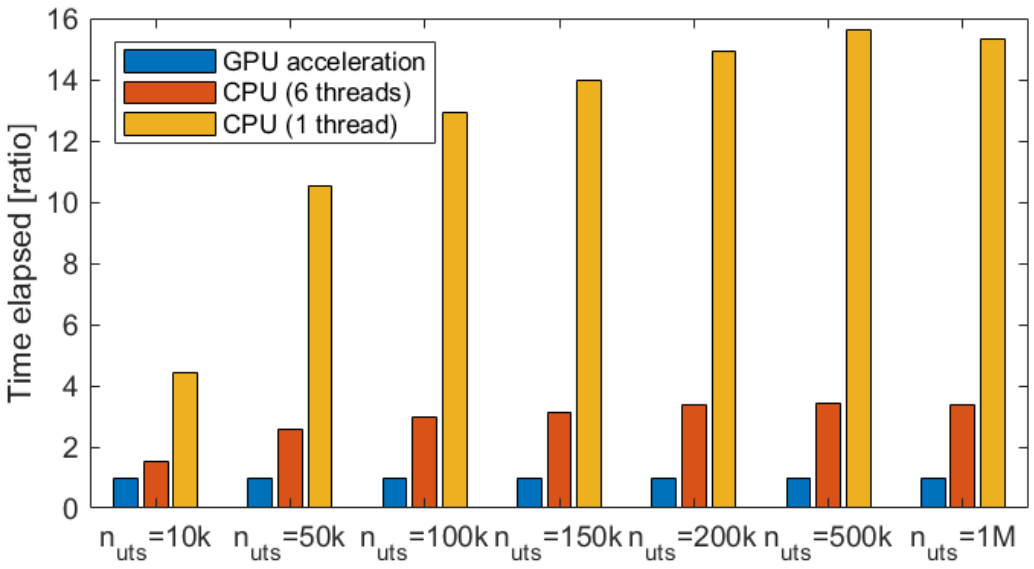

Figure 1. Performance times of different implementations, divided by the performance time of implementation using GPU acceleration, with varying number of UTs.

Performance time divided by the performance time of corresponding GPU accelerated implementation of one threaded implementation, six threaded implementation and GPU accelerated implementation are found in Figure 1. These implementations are run with varying amounts of UTs in the system (10k, 50k, 100k, 150k, 200k, 500k and 1M) for 720 simulations steps, while the step size is ten seconds, so the total simulated time is two hours.

A maximum speed-up using GPU acceleration of 15.6x is achieved compared to the sequential implementation when there are 500k UTs in the system.

3. Conclusions

The objective of this paper is to study the usage of GPGPU in LEO satellite network simulations and evaluate the performance in simulator run-time. The results suggest that there is a great potential in using GPGPU in system simulations. The significance of this fact is due to that system simulations can be time consuming but with the aid of GPGPU one can accelerate simulations and increase performance of the development of the final product and thus save time and money.

The greatest performance boost from the GPU accelerated implementation compared to the sequential implementation was achieved when 500k UTs in the system was used. Originally, the speed-up achieved was 15.6x and with the use of shared memory the speed-up increased to 16.0x. Initially, in the link budget computations with the baseline of 150k UTs a speed-up of 40.8x was achieved. After optimizing the program using shared memory, a speed-up of 45.1x in the link budget computations compared to the sequential implementation was achieved and a speed-up of 1.11x compared to the previous GPU-based implementation. As it is suggested in [4], the shared memory should be made of good use and is a simple way to optimize a program that uses GPGPU.

In the first implementation an assumption was made that the interference between the satellites and the beams of one satellite are insignificant. In final paper results will be updated with new interference model taking into account interference between satellites and beams allocated to same frequency band. It is expected that performance gain from GPGPU computing will increase with the introduction of interference calculation compared to CPU computing with one core. Additionally, in the final paper satellite network performance will be also analysed to be able to quantify the achieved gains from GPU computing with realistics satellite network performance evaluation simulations.

4. References

[1] https://artes.esa.int/funding/flexible-resource-allocation-techniques-ngso-constellations-artes-3a095

[2] “Numba documentation”. Accessed on: April 1, 2020. [Online]. Available: http://numba.pydata.org/numba-doc/latest/index.html.

[3] “TESLA K80 GPU ACCELERATOR”. Accessed on: April 3, 2020. [Online]. Available: https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/teslaproduct-literature/Tesla-K80-BoardSpec-07317-001-v05.pdf.

[4] C. A. Navarro, N. Hitschfeld-Kahler, and L. Mateu. “A survey on parallel computing and its applications in data-parallel problems using GPU architectures”. Communications in Computational Physics. Vol. 15. 2. 2014, pp. 285–329.