Acceleration of Digital Radar Landmass Simulation on Multi-core CPU and GPGPU Computer

Taieb Lamine Ben Cheikh
École Polytechnique de Montréal
Montréal, Canada
taieb.lamine-ibnecheikh@polymtl.ca

Pascal Guillemette
CAE Inc.
Montréal, Canada
pascal.guillemette@cae.com

Gabriela Nicolescu
École Polytechnique de Montréal
Montréal, Canada
gabriela.nicolescu@polymtl.ca

ABSTRACT

Digital Radar Landmass Simulation (DRLMS) for the purpose of training radar operators is a challenging computationally-intensive task. To improve fidelity, the databases are continuously increasing in resolution and density. Moreover, the radar simulation involves increasingly sophisticated physics-based models. One of them is the application of the radar antenna radiation pattern illuminating a particular region composed of landmass, sea clutter, precipitation and targets. To achieve real-time, a number of approximations are used in current simulations, which includes modeling a narrow antenna radiation pattern and employing a coarse sampling of the illuminated region. However, this harms the simulation fidelity.

To avoid these approximations while respecting the real-time constraints, this paper proposes a multi-level parallelization approach of landmass simulation applicable to multi-core Central Processing Unit (CPU) and General-Purpose Graphics Processing Unit (GPGPU). At the first level, the processing is divided into two parallel pipelines: (1) the power accumulation, which involves database processing and (2) the antenna pattern convolution. At the second level, the convolution is divided into several threads running in parallel. Two implementations are compared: one on multi-core CPU and one on GPGPU.

As benefits for training, we improve considerably the simulation performances as we are capable to apply a more detailed radar antenna pattern and support more complex databases, both contributing to more realistic radar images. The improved DRLMS shows respectively, a speedup of 12x on multi-core CPU running 16 threads, and a speedup of around 250x on a contemporary high-end graphics card over a one-thread execution on CPU.

ABOUT THE AUTHORS

Taieb Lamine, Ben Cheikh is a Postdoctoral fellow at École Polytechnique de Montréal. Currently, he is working on a MITACS project on the acceleration of CAE Radar simulator on modern high performance computers. He received his Ph.D. degree in Computer Engineering from École Polytechnique de Montréal in 2015. He has formerly received his B.Eng. and M.Sc. degrees in Electrical Engineering from École Nationale d'Ingénieurs de Sfax respectively in 2005 and 2006.

Pascal Guillemette. Graduated in Physics at the Université du Québec à Trois-Rivières in 1997. He obtained a M.Sc. degree in Meteorology at McGill University (Montréal) in 2000. Since then he joined CAE Inc. where he currently works as a Subject Matter Expert in radar simulation.

Gabriela Nicolescu is professor at École Polytechnique de Montréal in the Department of Software and Computer Engineering. She obtained her B.Eng. degree in Electrical Engineering from Polytechnica University of Bucharest in 1998 and her Ph.D. degree in 2002 from Institut National Polytechnique de Grenoble. Her research interests are related to the design methodologies, programming models and security for advanced heterogeneous systems on chip integrating advanced technologies. She co-authored more than 160 papers including journal articles, conference papers, books, book chapters and patents.

Acceleration of Digital Radar Landmass Simulation on Multi-core CPU and GPGPU Computer

Taieb Lamine Ben Cheikh
École Polytechnique de Montréal
Montréal, Canada
taieb.lamine-ibnecheikh@polymtl.ca

Pascal Guillemette
CAE Inc.
Montréal, Canada
pascal.guillemette@cae.com

Gabriela Nicolescu
École Polytechnique de Montréal
Montréal, Canada
gabriela.nicolescu@polymtl.ca

INTRODUCTION

To provide efficient flight training in critical environments for both civil and military aviation, the flight simulator community is continuously improving the fidelity of the models. Radar simulations are among those which can benefit from performance improvements to increase realism, fidelity, and hence training effectiveness. This is particularly important for the case of military Full Mission Simulators (FMS) where some crew members are dedicated to operate these sensors and analyze the data produced. The objective of this study is to improve realism and fidelity of a computer-based radar simulation using multiple processing threads on multi-core Central Processing Unit (CPU) and/or general-purpose graphics processing unit (GPGPU).

Digital Radar Landmass Simulation (DRLMS) is particularly important for the air-to-ground radars and this aspect represents one of the biggest challenges to the radar simulation engineers due in part to the large size of the databases (Bair, 1996). This processing can take advantage of hardware with high computational power. With the advent of multi-core CPUs and massive parallel platforms such as GPUs, it is now possible to increase the simulation fidelity while maintaining the real-time user interactivity. But this could be guaranteed only by an efficient utilization of the hardware computation resources offered by these parallel platforms.

Simulator architecture or deployment constraints may or may not permit the use of a specific GPU. Hence, our role is to provide a flexible parallel solution that offers the ability to target various platforms and optimize the partitioning of the computation workload on the resources. While other studies are targeting a specific hardware, our main contribution is to provide an approach offering more flexibility and ease of use to the domain expert. Thanks to this approach, the domain expert will run the simulation on either multi-core CPU or GPU or both without the need of any adjustments. In this work, we propose a multi-level approach that allows mapping of different types of parallelism (task- and data-parallelism) at different granularity levels of a given application on both multi-core CPU and GPU. With this strategy, several parallel implementations may coexist and could be enabled or disabled at runtime depending on the available computation resources. While this approach could be used to parallelize many applications that follow the nested task- and data-parallelism pattern, we will focus here on DRLMS as example. Parallel implementation of DRLMS on multi-core CPU reaches a speedup of 12x on 16 cores and 250x on GPU compared to a serial execution.

After a brief review of radar concepts, parallel hardware platforms and programming models, we detail the DRLMS processing. Then, the multi-level approach to parallelize DRLMS is introduced, followed by a presentation of the performance improvements. The last section gives the conclusion and other potential applications of this method.

BACKGROUND

Radar Principles

Radar uses electromagnetic waves to detect and/or track significant objects, depict the landmass, identify areas of precipitation (rain, snow, etc.), monitor airborne or sea-surface traffic, etc. Radio frequency pulses are emitted from an antenna and propagate through space. The orientation of the antenna as well as its radiation pattern determines the amount of energy sent in a particular direction. The antenna will receive the energy that is reflected (echoes) by objects in the environment. Some of these objects will affect the propagation, such as the presence of precipitation

which can attenuate the pulse, or the presence of mountains which can block it completely. This will make other objects behind more difficult or impossible to detect.

Antenna Radiation Pattern

The main purpose of the radar antenna is to determine the angular direction of the detected objects (Skolnik, 1990). During transmission, it concentrates the energy into a directive beam and plays an equivalent role at reception, capturing more of the signal from that direction. To achieve a high resolution, a very narrow beam is ideal. However, mechanical and electromagnetic constraints are such that antennas have a non-negligible beamwidth and also leak radiation in other directions called side lobes (see Figure 1). This creates ambiguity as reflectors from other directions can contaminate the signal coming from the direction the antenna is pointing at. On a radar display this will make the targets (landmass, ships, aircrafts, etc.) appear blurred in azimuth. From the point of view of a radar operator, this is an undesired effect. In simulation, this phenomenon must be modeled for realism, at an additional computational cost.

The radiation pattern depends on the physical characteristics of the antenna and the wavelength of the transmitted signal. Different beam shapes are used depending on the purpose of the radar, such as pencil beam, fan beam or cosecant squared beam (Skolnik, 1990).

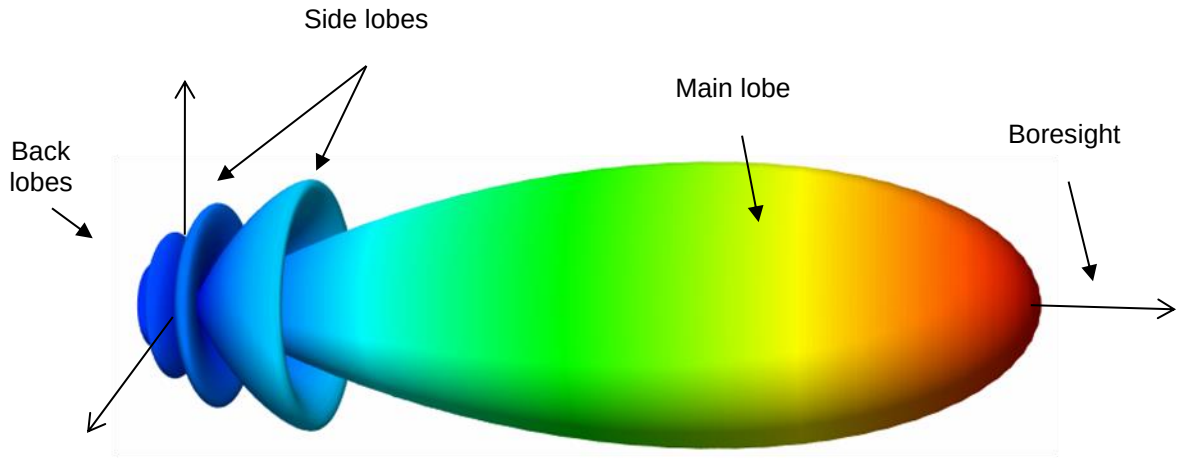

Figure 1. Typical radar antenna radiation pattern represented as gain vs. direction. The antenna is at the origin of the coordinate system.

Parallel Hardware Platforms

We focus this work on two parallel hardware platforms: (1) multi-core CPUs and (2) GPGPU.

Multi-core CPUs: this type of platform refers to general-purpose processors integrating multiple cores in the same die. In general, these cores are identical and they are based on x86 architecture. Current multi-core CPUs are limited to the order of tens of cores running tens of threads. Nevertheless, multi-core CPU is considered as a convenient platform to accelerate compute-intensive applications thanks to the programming flexibility (Lee et al., 2010).

GPGPUs: the application of GPUs is no longer restricted to graphics applications. During the last years, many compute-intensive applications were accelerated on GPGPUs (Che et al., 2008). The current GPUs are seen as general-purpose many-core platforms that integrate a large number of cores distributed on a number of streaming multiprocessors (SM). Moreover, the GPU platform is able to run a large number of simultaneous threads, which offers further parallelism.

Parallel Programming Models

In order to program parallel hardware platforms, we use specific parallel programming models. The programming models allow the programmer to express the parallelism of the application without the need to write a low-level multithreaded code. The programming models show certain architecture features such as the parallelism level, the type of parallelism, and the abstraction degree of the components' functions. Parallel programming models are implemented as a set of languages, extensions of existing languages, libraries and tools to map applications on parallel hardware.

OpenMP: is a standard shared-memory programming model (Dagum and Menon, 1998). It is designed as an API used to explicitly enable multithread execution on multi-core CPUs. The main feature of OpenMP is the ease of use by providing the capability to incrementally parallelize a sequential program. Moreover, it is capable of implementing both task and data parallelism models.

CUDA and OpenCL: Among the most popular programming models for GPUs are Compute Unified Device Architecture (CUDA) developed by NVIDIA (NVIDIA, 2007) to program their GPUs, and Open Computing Language (OpenCL) developed by Khronous (Stone et al., 2010) which targets many GPU platforms including NVIDIA GPUs and AMD ATI GPUs. Both CUDA and OpenCL are extensions of the C language and implement a particular runtime to manage the computation on GPU. CUDA and OpenCL adopt the same philosophy for their runtime models. Threads in both programming models are organized as a hierarchy of 3D grids and 3D blocks in order to match the dataset organization. Threads belonging to the same block are assigned to the same streaming multiprocessor. While CUDA is a vendor-specific programming model, OpenCL is generic and supports several parallel platforms. The higher flexibility of OpenCL compared to CUDA comes with an overhead in term of lines of code and sometimes a slightly lower performance when running on NVIDIA GPUs. In this work, we implement two parallel versions of the DRLMS on GPU, one using CUDA and the other using OpenCL in order to offer respectively the best performance when targeting NVIDIA GPUs, and the flexibility in term of implementation.

DRLMS PROCESSING

With in mind the notions of the previous section, the key to improve performance is to identify and group calculations that can be done in parallel (or not) in the radar simulation. We decompose the simulation to express parallelism, considering the following observations from radar point of view:

- objects in the environment can modulate or block the power reaching other objects beyond, but on the same azimuth;
- the antenna pattern will blend objects that are at the same range;

Thus, the first point indicates that power calculations will depend on results from closer ranges, but will be independent in azimuth. The second point suggests the opposite for the modeling of the radiation pattern effects. Therefore, the approach proposed here decomposes DRLMS in two stages: the power accumulation stage (Accumulation) and the antenna pattern convolution stage (Convolution).

Accumulation stage

The main steps of the accumulation are shown in Figure 2. As the radar platform moves, its new position is fed to the simulation, which sets the origin of the illumination. Based on this location, the simulation maps a particular region (landmass) represented as tiles of terrain elevation (digital elevation model) and culture (points, lines, surfaces and 3D models or polygons) that are extracted from a database. Then, the respective reflectivity parameters (dielectric properties, orientation, directivity, etc.) of the surfaces, building structures, streets, trees, terrain, moving targets, etc. are extracted. The echoes and attenuations caused by precipitation are also added at this stage. The returns will be represented as a 2D array of samples. The first dimension represents the azimuth angles ranging from 0 to 360 degrees sampled according to a given angle resolution. The second dimension represents the range bins where the number of range bins defines the range resolution of the radar. Each array element contains the power reflected by landmass and precipitation assuming the illumination source is an isotropic antenna with a gain of 1. Figure 3(a) is an example of such power returns where the intensity has been converted in shades of grey.

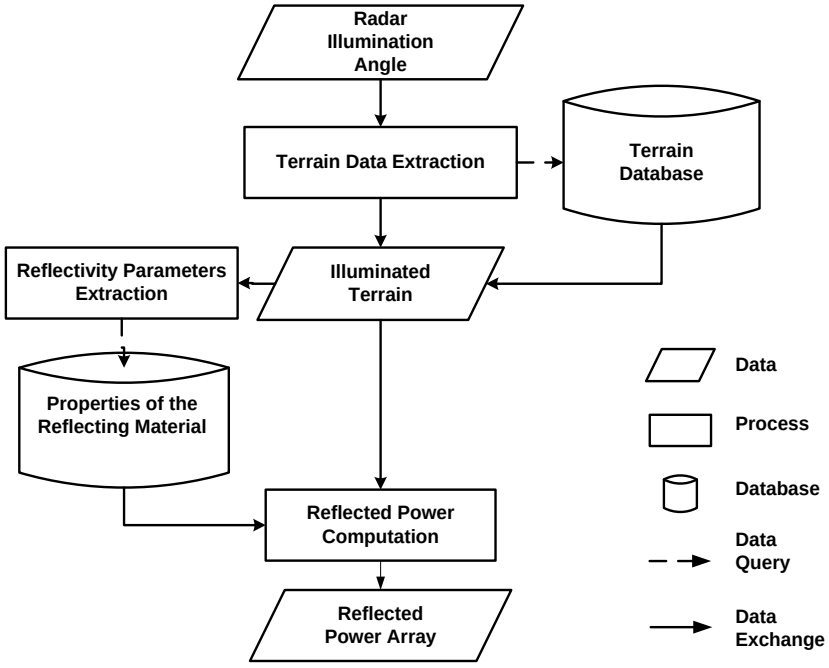

Figure 2. Accumulation process diagram

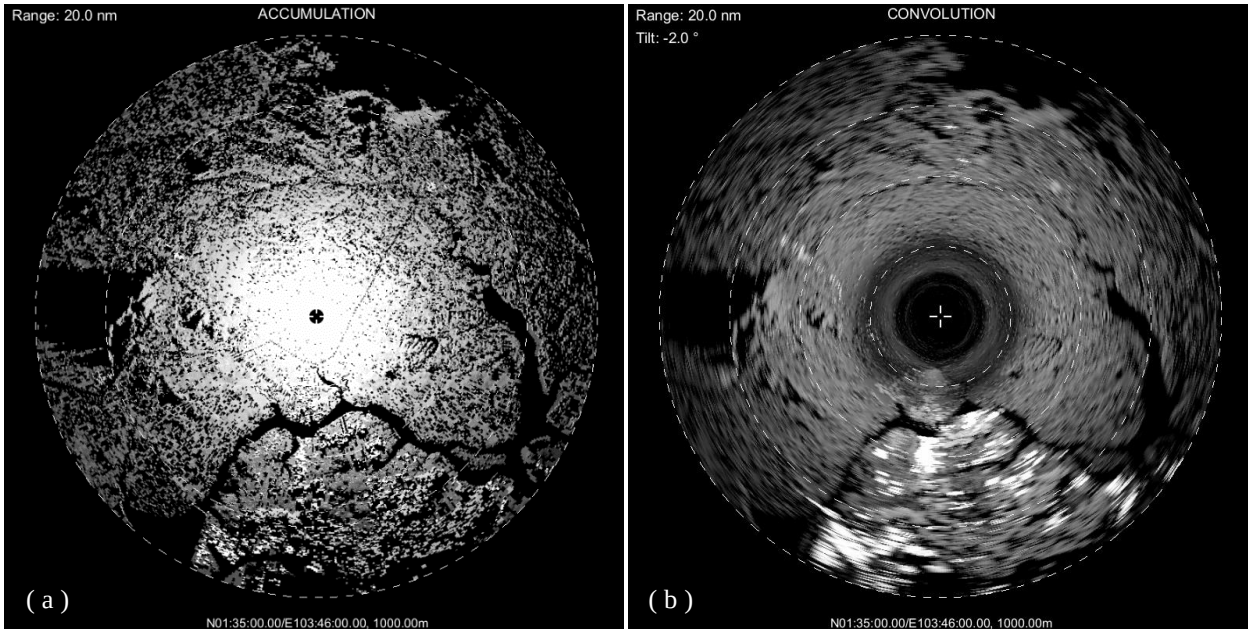

Figure 3. (a) accumulation array for an isotropic antenna, (b) convolution of a sin(x)/x antenna pattern with a 3-degree beamwidth on the accumulation array, after a complete scan.

Convolution Stage

The antenna radiation pattern is modeled in both elevation (*El*) and azimuth (*Az*). For a given antenna orientation, the antenna pattern is applied on the surrounding samples at the neighbor azimuth angles for each range bin (r_i) using Equation 1.

$$P_{out}(r_i) = \sum_{Az=0}^{360} P_{in}(r_i, Az) \cdot G^2(Az, El(r_i)) \quad (1)$$

The result of this convolution is an array of powers (P_{out}) indexed by range at the specified azimuth. This process is repeated for each azimuth angle as the antenna scans around. Figure 3(b) is an example of the convolution after a complete scan (360 degrees) where the intensity has been converted into shades of grey. The antenna radiation pattern was given a $\sin(x)/x$ shape (such as illustrated in Figure 1) with a main lobe beamwidth of 3 degrees. The $\sin(x)/x$ function, where x is the angular distance from the boresight, is a widely-used approximation for common radar antennas, but this parallelization solution remains applicable for any antenna pattern with any beamwidth without additional cost.

PARALLELIZATION OF DRLMS

Several parallelism levels may be exploited in the processing involved in DRLMS. At the top level, it is decomposed in two parallel tasks (Accumulation and Convolution) where the task-parallelism model is expressed. At the mid-level, the coarse-grain data processing for the convolution task is analyzed and expressed following data-parallelism model. At the bottom level, finer-grain data-parallelism is exploited by decomposing further the convolution task in elementary data processing. Since multi-core CPU and GPU show many differences regarding the architectural aspects, the proposed hierarchical parallelism representation is adopted to be suitable for both parallel platforms. Multi-core CPU is a control-oriented architecture integrating a limited number of cores which makes this architecture more efficient for coarse-grain task-parallelism and coarse-grain data-parallelism (see Figure 4). GPU, on the other hand, is a hierarchical data-oriented architecture, which is composed of a fair number of streaming multiprocessors, which integrates in turn a large number of cores. Therefore, both coarse-grain and fine-grain data-parallelism are well supported by such architecture (see Figure 4).

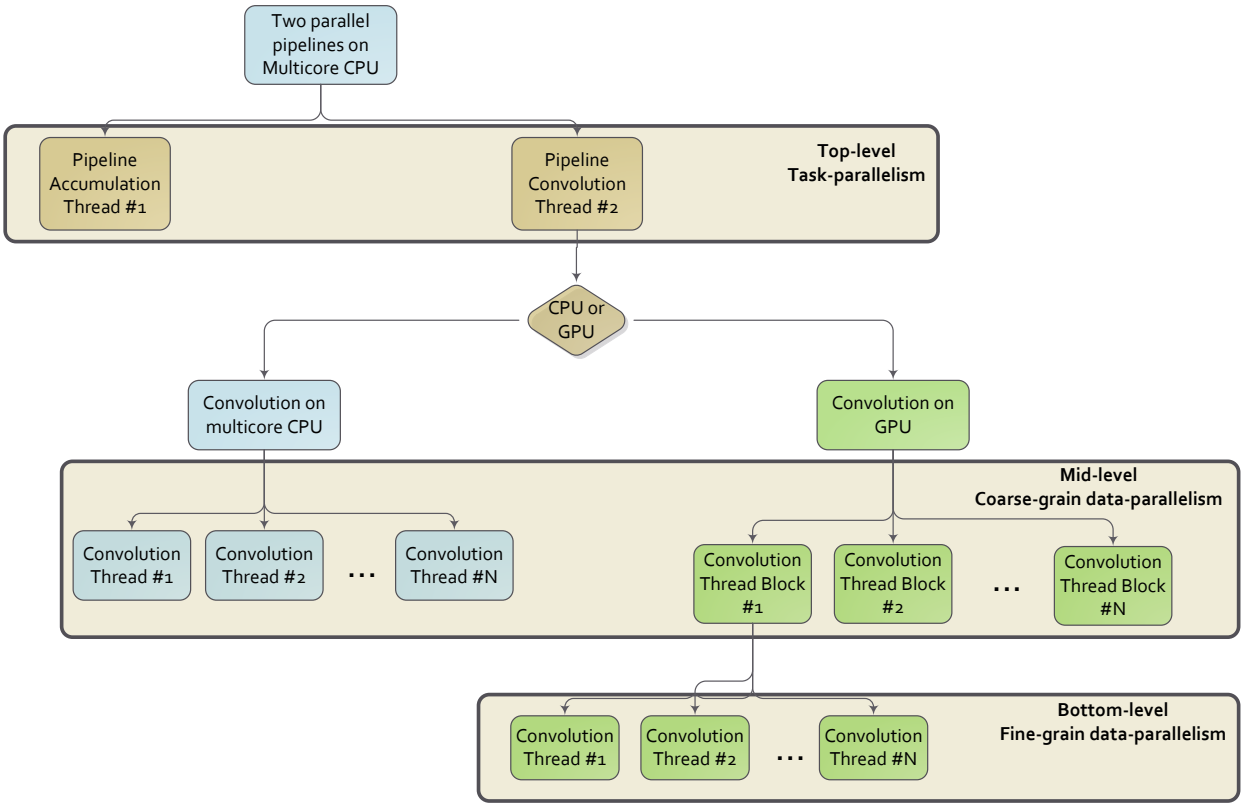

Figure 4. Parallelization hierarchy of DRLMS

In order to accelerate the DRLMS processing, we proceed in two steps:

1) To hide the Accumulation processing time, we overlap the Accumulation and the Convolution by running the two stages in parallel in two separate CPU threads following functional parallelism model. This is similar to the approach proposed by (Cavallaro and Gordini, 2013). By doing this, the convolution will run on one disk of power while the accumulation can process a new disk of power. To keep the two stages running asynchronously, we implement a double buffer mechanism for each stage. One of the main advantages of this approach is that the computational cost of convolution is now independent of the content of the database. In practice, some parts of a database may be populated with a lot of complex 3D objects such as in urban areas vs. rural areas. With a serial implementation, the computation time required for an azimuth will depend on the amount of these features hit by the radar beam in this direction. This results in an uneven scan speed on the operator's display, unless sleep time is introduced to balance processing time, which is a waste of computational resources. Figure 5(a) shows the sequence diagram of a serial execution of the accumulation followed by the convolution stage and Figure 5(b) shows the sequence diagram of the parallel pipeline running the accumulation and the convolution. In the latter diagram, the Accumulation stage and the Convolution stage are overlapped. If we consider that each colored disk represents a new accumulated data and each respective colored sector represents the convolved power, we can note that thanks to this parallelization, the DRLMS is now able to scan faster than serial implementation. Furthermore, since in the accumulation stage power levels from one azimuth are independent of those of other azimuths, these can be treated in parallel if needed.

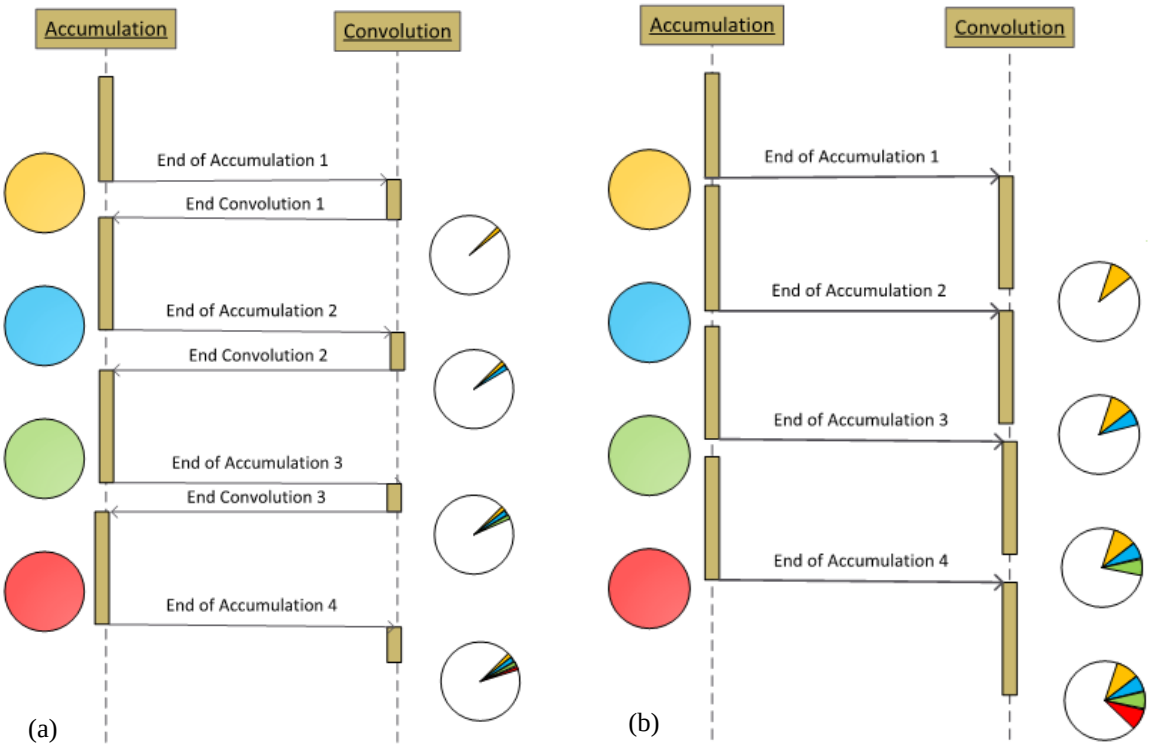

Figure 5. Sequence diagram of DRLMS: (a) serial implementation, (b) parallel implementation

2) We parallelize the Convolution on the remaining multi-core CPU threads or on the GPU. In this stage, range bins do not impact other range bins. Therefore, all range bins can be calculated in parallel. The power at each range bin belonging to a given azimuth is computed using Equation 1. The antenna gain is a function of the azimuth angle and the elevation angle. This type of parallelism is known as data-parallelism. Since the number of available CPU cores (24) is less than the number of bins (512, 1024, 2048 or 4096), each CPU thread must process a set of range bins (see Figure 6a). On the other hand, since the number of GPU threads is way larger than the number of

range bins, one level of parallelism is not sufficient to take advantage of the full computation power of the GPU. A two-level parallelism approach is considered:

- the first level is to decompose the range bins along thread blocks as each thread block will process a ring of subset of range bins;
- in the second level, each ring assigned per thread block is divided on the threads belonging to that block as each thread will compute a partial convolution along a single sector of this ring (see Figure 6b). Finally, all partial convolution results are summed by one thread of each thread block to form the output power at a given range bin.

In this paper, the multi-level parallelization of DRLMS was implemented as follow: the task-level parallelism is implemented as two CPU threads using the **`parallel sections`** directive of OpenMP and the data-parallelism is implemented as a multi-threaded processing on multi-core CPU using the **`parallel for`** directive of OpenMP while the data-parallelism on GPU is implemented as two versions one using CUDA and the other using OpenCL for the sake of programming flexibility.

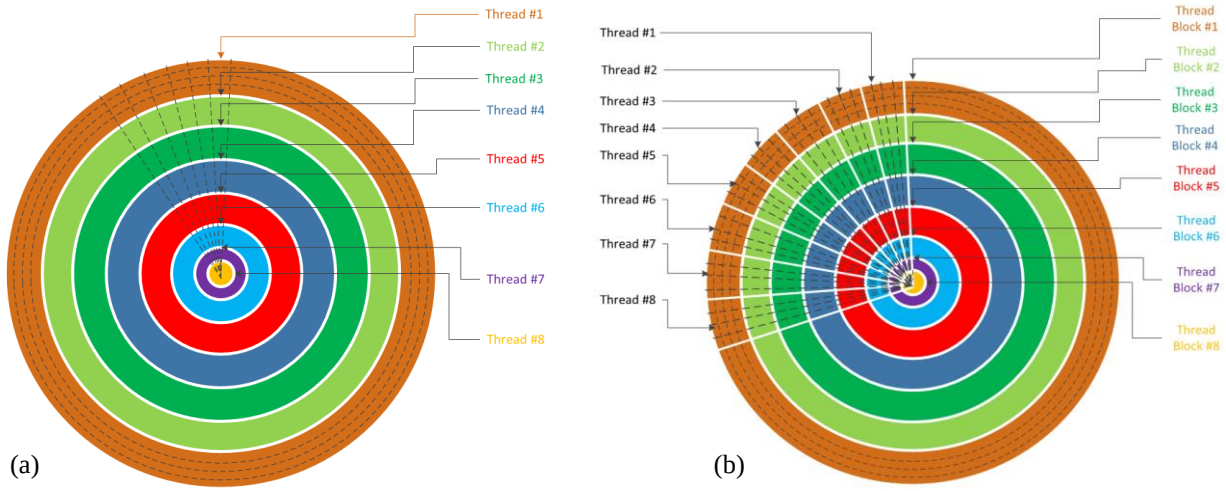

Figure 6. Parallelization of convolution on (a) multi-core CPU, (b) GPU.

EXPERIMENTS AND RESULTS

We have conducted experiments on a desktop computer integrating both a multi-core CPU and a GPU with the specifications listed in Table 1.

Table 1. Hardware Platform Specifications

Parallel Platform	**Multi-core CPU**	**GPU** (refer to NVIDIA 2016)
Manufacturer	Intel	NVIDIA
Model	Xeon E5-2620 v2	GTX 1080 (GP104)
# of processors	2	20 SM
# of cores	12	2560
Base Clock	2100 MHz	1603 MHz
Maximum # of threads	24	40960 (2048x20)
Maximum # of thread Blocks	N/A	640 (32x20)
Global Memory Size	16 GB	8 GB
Shared Memory Size	N/A	96 KB per SM

Experimental Results

In our experiments, we run DRLMS with range resolutions from 512 to 4096 bins and azimuth resolutions of 0.25 and 0.5 degree. Even though it is not always required depending on the type of radar, the convolution is applied on 360 degrees in azimuth in order to work with the worst case as a baseline. The execution times of the convolution at different resolutions are given in Table 2. The execution time of the serial convolution of a whole disk at low resolution running on one thread is around 11 s, which is not practical for a real-time simulation while the parallel version on 16 cores can take only 1 s, which is suited for real-time simulator. The GPU takes only 1.5 s to produce the whole 360-degree convolution for 4096 range bins and 0.25 degree of azimuth resolution. The performance of GPU outperforms the 16-core CPU by a speedup of 22x and 1-core CPU by a speedup of 250x. We show also that the GPU scales better than multi-core CPU with the computation complexity by offering a higher speedup when the resolution is higher.

Table 2. Acceleration Performances on Multi-core CPU and GPU

	CPU 1 core	CPU 2 cores	CPU 4 cores	CPU 8 cores	CPU 16 cores	GPU* (CUDA)
512 bins, 0.5° azimuth resolution						
Execution Time (s)	11.5	5.8	3.1	1.7	1.1	0.3
Speedup (x)	1.0	1.9	3.7	6.7	10.3	40.0
2048 bins, 0.5° azimuth resolution						
Execution Time (s)	46.8	23.0	12.2	6.4	4.3	0.5
Speedup (x)	1.0	2.0	3.8	7.2	10.8	100.0
2048 bins, 0.25° azimuth resolution						
Execution Time (s)	188.6	94.3	47.5	26.6	16.6	1.1
Speedup (x)	1.0	2.0	3.9	7.1	11.4	174.6
4096 bins, 0.25° azimuth resolution						
Execution Time (s)	374.4	187.2	93.6	54.0	33.1	1.5
Speedup (x)	1.0	2.0	4.0	6.9	11.3	247.6

* Performances obtained with OpenCL did not significantly differ from those obtained with CUDA.

Discussion

While the multi-core CPU offers an acceptable performance improvement of the simulation, it is only applicable for real-time low and mid-resolution simulation. This is explained by the low number of threads that can run in parallel on such platform. Moreover, the achieved speedup on multi-core CPU does not scale well with the data parallelism granularity (high number of range bins and azimuth resolution) due to the overhead for managing the running threads (see Table 2). On the other hand, the GPU offers significant performance improvement suited for real-time high resolution simulation. The huge number of light managed threads that can run in parallel on GPU is well suited for large parallel data processing. A parallel application could take the maximum of the GPU when the processing/data access ratio is more significant. This is also shown in Table 2 where the number of range bins and azimuth resolution is increasing. This explains the good scalability of the GPU with the large data parallelism.

Although the GPU provides high performance, it is limited to data-parallelism while the multi-core CPU is essential to implement the task-parallelism (overlapping data extraction and data processing). Moreover, the higher performance provided by the GPU compared to the CPU comes with a cost of more programming and debugging effort to port the dependencies on the GPU and to manage the data exchange between CPU and GPU.

Besides these considerations other practical aspects must be taken into account when deciding whether to opt for a multi-core CPU or a GPGPU approach. The need for additional CPU resources pushes towards the GPGPU solution. For instance, the CPU time savings could be applied to the simulation of a track-while-scan function or a terrain-following model. Both would use the result of the convolution as an input. On the other hand, adding one GPGPU-

capable graphics card can have an impact on the cost of a simulator. The cost increases not only for the part itself, but also for the effort of maintaining documentation and schematics, managing obsolescence, etc. for one computer in the computing complex of a full mission simulator.

CONCLUSION

In this work, we provide a multi-level approach to implement a nested task- and data-parallel application on both multi-core CPU and GPU. This approach is experimented with the parallel implementation of DRLMS as part of a training simulator. In particular, this approach enables the efficient utilization of available computing resources of both CPU and GPU cores to accelerate DRLMS. As results, we show that we improve considerably the simulation performances as we are capable to simulate high resolution DRLMS at real-time on GPU while applying a realistic radar antenna radiation pattern. By combining these two strategies: 1) splitting the landmass simulation in two main processes, accumulation and convolution, and 2) parallelizing the convolution, we obtain a regular scan rate even when scanning over a densely or unevenly populated database. The parallelization of DRLMS on multi-core CPU running 16 threads shows a speedup of 12x while the parallelization on GPU shows a speedup of 250x.

In the future, we plan to improve further the performances by parallelizing the accumulation stage. That would allow handling databases with higher densities. To accomplish this, the following scheme would be used:

- parallelized accumulation on multi-core CPU;
- multi-level parallelization of convolution on GPGPU.

This would make an optimal usage of the computing resources of a standard multi-core PC equipped with a GPGPU.

Besides radar, other sensor simulations could use this multi-level approach. Underwater acoustics is probably the most similar example because of the emission and reception, with various beam shapes, of energy that can bounce on the ocean floor. Our multi-level approach could also be used to implement applications in the image processing field where the data loading and data processing could run in parallel as two overlapping pipeline while the data processing could run in parallel on each separate image block of pixels and separate pixels.

REFERENCES

Bair, G. L. (1996), *Airborne Radar Simulation*, Technical Report, Camber Corporation.

Cavallaro, S. and Gordini, R. (2013), Surface clutter model for real-time simulation of coherent radar modes, in *Proceedings of the Interservice/Industry Training, Simulation and Education Conference (I/ITSEC)*, Orlando, FL.

Che, S., Boyer, M., Meng, J., Tarjan, D., Sheaffer, J.W. and Skadron, K. (2008), A performance study of general-purpose applications on graphics processors using CUDA, *Journal of Parallel Distributed Computers*, vol. 68, no. 10, pp. 1370–1380.

Dagum, L. and Menon, R. (1998), OpenMP: an industry standard API for shared-memory programming, *Computational Science & Engineering*, IEEE, vol. 5, no. 1, pp. 46–55.

Lee, V., Kim, C., Chhugani, J., Deisher, M., Kim, D., Nguyen, A., Satish, N., Smelyanskiy, M., Chennupaty, S. and Hammarlund, P. (2010) Debunking the 100x GPU vs. CPU myth: An evaluation of throughput computing on CPU and GPU, in *37th Annual International Symposium on Computer Architecture*, Saint-Malo, France.

NVIDIA (2007) *Compute unified device architecture programming guide*, Retrieved June 23, 2017 from http://docs.nvidia.com/cuda/cuda-c-programming-guide.

NVIDIA (2016), *NVIDIA GeForce GTX 1080 - Whitepaper*, NVIDIA Corporation, Retrieved June 23, 2017 from https://developer.nvidia.com/introducing-nvidia-geforce-gtx-1080.

Stone, J.E., Gohara, D. and Shi, G. (2010), OpenCL: A parallel programming standard for heterogeneous computing systems, *Computing in science & engineering*, vol. 12, no. 1-3, pp. 66–73.

Skolnik, M. I. (1990) – editor in chief, *Radar Handbook, 2nd ed.*, Boston, MA. : McGraw-Hill.