GPU-BASED PARALLEL COMPUTING IN REAL-TIME MODELING OF ATMOSPHERIC TRANSPORT AND DIFFUSION OF RADIOACTIVE MATERIAL

Marcelo C. dos Santos[1], Claudio M. N. A. Pereira[1], Roberto Schirru[2] and André Pinheiro[2]

[1] Instituto de Engenharia Nuclear (IEN / CNEN - RJ)
Rua Hélio de Almeida, 75
21941- 906 Rio de Janeiro, RJ
jovitamarcelo@gmail.com, cmnap@ien.gov.br

[2] Programa de Engenharia Nuclear – COPPE
Universidade Federal do Rio de Janeiro
Av. Horácio Macedo, 2030
21941-901 Rio de Janeiro, RJ
schirru@lmp.ufrj.br, apinheiro99@gmail.com

ABSTRACT

Atmospheric radionuclide dispersion systems (ARDS) are essential mechanisms to predict the consequences of unexpected radioactive releases from nuclear power plants. Considering, that during an eventuality of an accident with a radioactive material release, an accurate forecast is vital to guide the evacuation plan of the possible affected areas. However, in order to predict the dispersion of the radioactive material and its impact on the environment, the model must process information about source term (radioactive materials released, activities and location), weather condition (wind, humidity and precipitation) and geographical characteristics (topography). Furthermore, ARDS is basically composed of 4 main modules: Source Term, Wind Field, Plume Dispersion and Doses Calculations. The Wind Field and Plume Dispersion modules are the ones that require a high computational performance to achieve accurate results within an acceptable time. Taking this into account, this work focuses on the development of a GPU-based parallel Plume Dispersion module, focusing on the radionuclide transport and diffusion calculations, which use a given wind field and a released source term as parameters. The program is being developed using the C ++ programming language, allied with CUDA libraries. In comparative case study between a parallel and sequential version of the slower function of the Plume Dispersion module, a speedup of 11.63 times could be observed.

1. INTRODUCTION

The Emergency Preparedness (EPs) at Nuclear Power Plants (NPPs) is an aspect that has a vital role in granting the safety of the people that lives near NPPs. In light of a radiologic emergency, in special, one with a radioactive material release, the EP defines protective actions, as

equences and

on should be

employed, several factors are taken into account, and if the accident involves radioactive material release, an accurate forecast of the dispersion of the radioactive material is an essential factor to help the decision making process and even more crucial in case of an evacuation, to successfully guide people away from the possible affected areas.

Therefore, in order to predict the transport and diffusion of radioactive material, Atmospheric Radionuclide Dispersion Systems (ARDS) have been used [1]. These systems, to function effectively and be able to assist the decision-making process in real time in an emergency situation, need to process input information about source term (the radioactive materials released, activities and location), meteorological conditions (wind, humidity and precipitation) and geographical characteristics (topography, terrain, soil, etc.). Moreover, the ARDS is composed of 4 main modules: Source Term, Wind Field, Plume Dispersion and Dose Calculation. Among these, the modules of Wind Field and Plume Dispersion are the ones that demand greater computational processing to reach the desired results in working time. For this reason, one of the challenges when using ARDS is to make the system generate accurate and quality information within an acceptable time. Since the quality of the information provided by an ARDS is key to successful decision making during an emergency. In order to achieve adequate accuracy in desirable time the use of a high-performance computer system is required to run the ARDS and especially its heavier modules.

On that premise, this work focuses on the Plume Dispersion module. This module determines the dispersion of radioactive material in the atmosphere in the event of an accident. The model applied by the module is able to work with NPPs located in complex regions, and to make this possible, it is implemented a three-dimensional wind field, generated by the Wind Field module, which appropriately incorporates the effects of the topography of the NPP's region [2]. Besides that, the model has the fundamental concept that the atmospheric dispersion of a plume resulted from a continuous radioactivity release, can be simulated by the dispersion of a succession of radioactive clouds released at intervals of time containing the same total amount of activity [3]. Those radioactive clouds are considered as point sources in a region of interest, that is subdivided into several volumes (cells) distributed in a three-dimensional grid. In an away that the model can have its precision influenced, not only by the spatial resolution but also by the refined treatment of the plume. In spite of that, the use of a thin grid model implies in a computational cost that can impose practical limitations.

Considering that situation, this work proposes the development of a GPU-based parallel computational model to accelerate the calculation of radioactive transport and diffusion in order to be able to use a thin grid structure, for application in atmospheric dispersion systems. The program is being developed using the C ++ programming language, allied with the Computing Platform and Programming model, CUDA.

This paper has the following structure. Section 2 consists of a description of the model used by the Plume Dispersion module to do the transport and diffusion of radionuclides calculation. A general outlook at the General-Purpose Computing on Graphics Processing Units Systems, GPU architecture, and CUDA programming is presented in section 3. Section 4 explains the parallelization process. The results are analyzed at section 5 and concluding remarks appear in section 6.

2. THE TRANSPORT AND DIFFUSION OF RADIONUCLIDES MODEL

The Plume Dispersion module calculates the average concentration of radionuclides in a region, based on the atmospheric transport and dispersion of radionuclides due to an advective/diffusive process [3]. The equation 1 expresses this process.

$$\frac{\partial \mathrm{C}}{\partial \mathrm{t}} = -\nabla \cdot \vec{\mathrm{U}}\mathrm{C} + \nabla \cdot \overleftrightarrow{\mathrm{K}} \cdot \nabla \mathrm{C} \tag{1}$$

Where $\frac{\partial C}{\partial t}$ is the rate of change of radionuclides concentration in a location, $-\nabla \cdot \vec{U}C$ is the increase or decrease because of the advection and $\nabla \cdot \overleftrightarrow{K} \cdot \nabla C$ is the decrease or increase due to the diffusion.

In order to solve the equation 1, the model applied, considers that the three dimensional region of interest is divided into several smaller sub regions (called nodes or cells), where the wind, diffusion and concentration fields can treated as varying linearly inside them.

Besides that, the model implemented by the module, achieve a solution for the equation 1 based in a Gaussian source model approach but with no uniform wind and diffusion fields, with both fields in a vertical and horizontal sheer. Equation 2 express the solution.

$$C = \frac{Q\,(zH)^{1/4}\,U_o^{\,1/2}}{4_\pi^{\,1/2}\,K_1^{\,1/2}\,K_o^{\,\eta^{2/3}}} \quad \mathrm{EXP}\left[\frac{-\,K_0/K_1\,\zeta^2 + z^2 + H^2}{4K_o\,\eta/U_o}\right] I_{-1/4}\left[\frac{zh}{2K_o\,\eta/U_o}\right] \tag{2}$$

Where:

- z = The vertical distance (meters)
- y = The crosswind distance (meters)
- U = Wind speed = $U_{o^{z^{1/2}}}$ (m/s)
- U_o = The reference velocity at $z = 1$
- K_z = Vertical diffusivity = $K_{o^{z^{1/2}}}$ $(m^2/2)$
- K_y = Horizontal diffusivity = $K_{1^{z^{1/2}}}$ $(m^2/2)$
- K_o = The reference vertical diffusivity at $z = 1$
- K_1 = The reference horizontal diffusivity at $z = 1$
- H = The effective plume height (meters)
- Q = The Source strength (μg/s)
- I = The modified Bessel function of the first kind
- C = The Concentration $(\mu g/m^3)$

3. HETEROGENEOUS COMPUTING: GPGPU

Heterogeneous computing is a computational system schema in which applications are executed in a computational node, composed of distinct architectures with different capacities and ways of executing instructions. Currently, there are several types of heterogeneous systems, however, the system that has stood out is the General-Purpose Computing on Graphics Processing Units (GPGPU), and it uses the Graphic Processor Unit (GPU), previously only used in the execution of graphical applications, to process general purpose applications that once were only executed by the Central Processor Unit (CPU).

In the last years, several engineering applications are using the GPGPU system and achieving optimum results. An example of that type of application is the one developed by Pinheiro et al [2], where a parallel version, using GPU, of the Wind Field module of an ARDS was elaborate, achieving a speed up of 40 times compared to the sequential version. Another similar program developed using the same architecture was the Heimlich [4], in this, the speed up with respect to the sequential version was 125 times. Furthermore, Pereira et al [5] show that it is possible to achieve a speed up of 2000 times when using a Multi-GPU system to run a neutron transport simulation problem.

3.1. GPU Architecture

The GPU used in the development of this work was the GeForce GTX-1070, this GPU is part of the Pascal architecture which is the sixth generation of NVIDIA devices capable of using CUDA technology. The GTX-1070 is partitioned into 3 Graphics Processing Cluster (GPCs), where each GPC, in turn, has 5 Streaming Multiprocessors (SMs) with 128 CUDA processors in each SM. At the hardware level, the SM is the main component of the GPU, because it is where the work is actually implemented through the CUDA processors that perform the mathematical operations for the Threads. In exemplification criteria, a thread can be understood as a pixel in an image. Also, to control the execution of CUDA processors the SMs have Warp Schedulers responsible for organizing the execution process in groups of 32 threads called Warps. Each Warp is allocated in the Dispatcher Unit where the instructions are executed. Figure 1 shows an SM schematic of the GTX-1070.

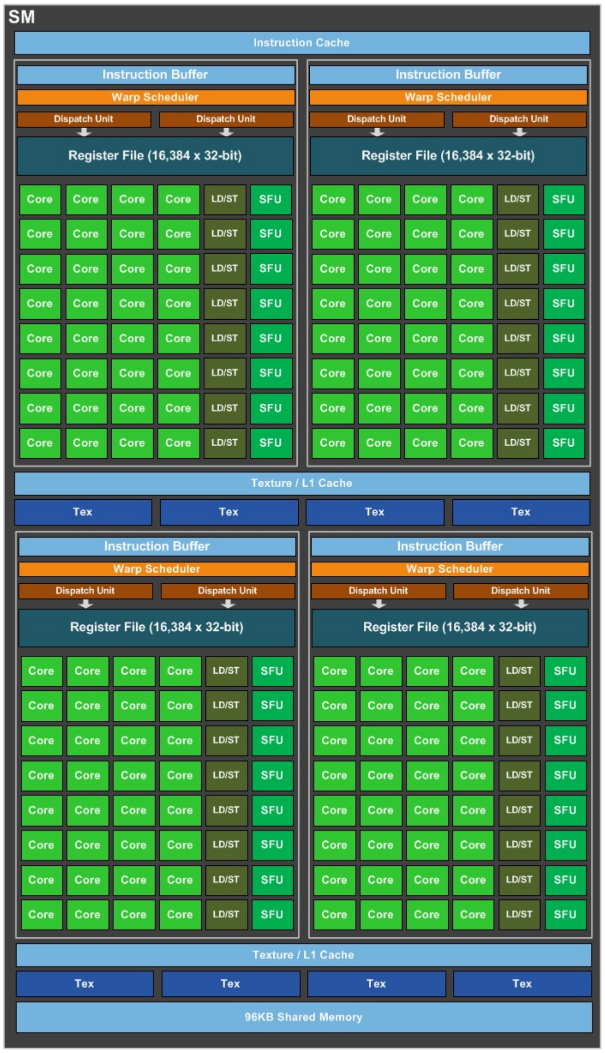


Figure 1: The NVIDIA GeForce GTX-1070's Stream Multiprocessor architecture. The SM is divided into 4 parts, each containing a warp scheduler responsible for distributing the threads, in groups of 32 threads, called warps, to its execution units.

3.2. CUDA: Compute Unified Device Architecture

The GPGPU presented in Chapter 3 has been changing the way high-performance applications are structured. Since the cores of a GPU are architected to work using the Single Instruction Multiple Data (SIMD) programming model, where an instruction is employed in multiple data in parallel. Therefore, when developing an application that will be executed by a GPU, it must be structured based on parallel programming. For this reason, this work uses the Compute Unified Device Architecture (CUDA), which is a platform and parallel programming model developed by NVIDIA, as the main foundation for the project. The CUDA platform was chosen primarily because of its flexibility, by allowing the use of general-purpose languages, as Fortran, C / C ++ or Python. Among these, CUDA C / C ++ was the one employed in the development.

3.3. CUDA C/C++ Programming Model

CUDA C / C ++ works as an extension of C / C ++ languages adding features that facilitate parallel development. One of these features is the ability to define functions in C / C ++, named kernels, that when invoked are executed concurrently N times by N CUDA processors. In this perspective, the programming model employed in CUDA is strongly integrated with the NVIDIA GPU hardware. When a kernel is started, an information requesting the creation of a grid of threads is sent from the CPU to the GPU, within this grid the threads are grouped in blocks, as shown in figure 2. Upon receiving the information, the GPU uses the Work Distribution Unit to distribute the threads blocks to the SMs, looking for those that have sufficient resources to perform the task. This process has the objective of intensifying the parallelism in the GPU, evenly distributing the threads among the SMs.

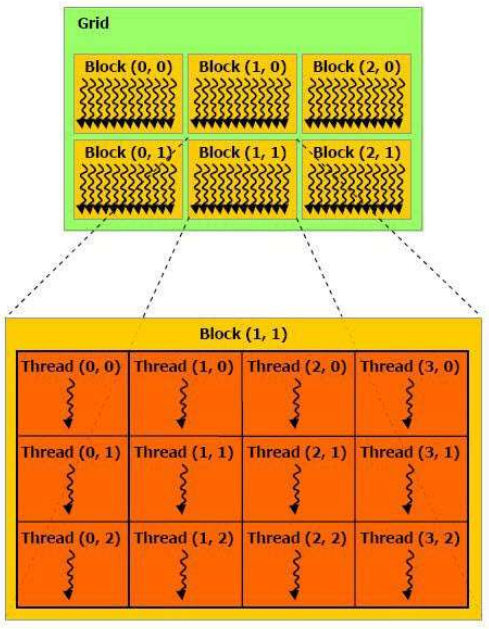
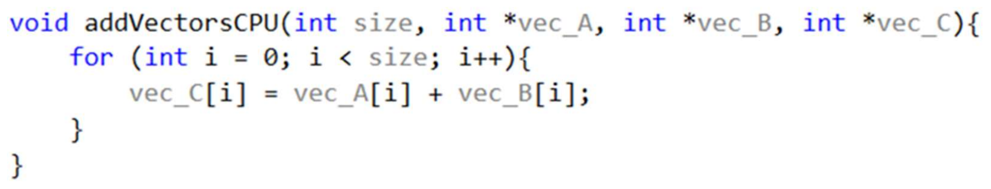


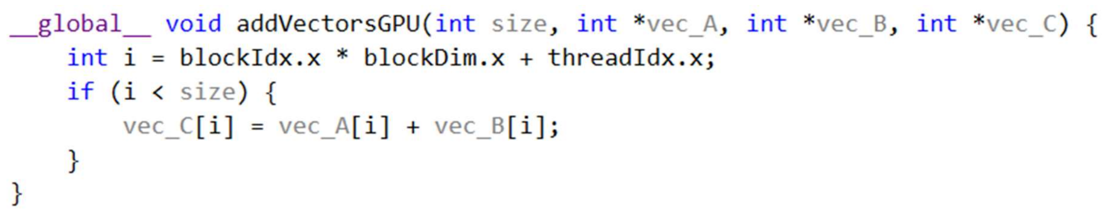
Figure 2: A grid containing 6 blocks, where each block consists of 12 threads.

In order to illustrate the basic structure of a program in CUDA C / C ++, figure 3 and 4 present a function that adds 2 vectors, one version in C / C ++ and other in CUDA C / C ++, respectively.



```
void addVectorsCPU(int size, int *vec_A, int *vec_B, int *vec_C){
    for (int i = 0; i < size; i++){
        vec_C[i] = vec_A[i] + vec_B[i];
    }
}
```

Figure 3: C function that adds 2 vectors.



```
__global__ void addVectorsGPU(int size, int *vec_A, int *vec_B, int *vec_C) {
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    if (i < size) {
        vec_C[i] = vec_A[i] + vec_B[i];
    }
}
```

Figure 4: CUDA function that adds 2 vectors.

Note that, instead of the for loop (Figure 3) that sequentially adds each element of the vector, the CUDA function (Figure 4) adds only position i, with an if statement to o ensure there are no out-of-bounds memory accesses. That occurs because, in CUDA, the CUDA processors work in parallel, with each one independently performing the sum operation for a position i of the vectors. Thus, for each thread be able to process a resultant vector element, it is necessary that the identification of the global index occurs in advance. Figure 5 exemplifies the indexing pattern in CUDA.

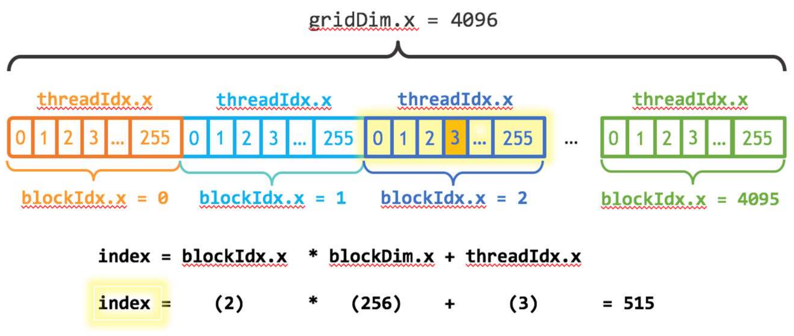
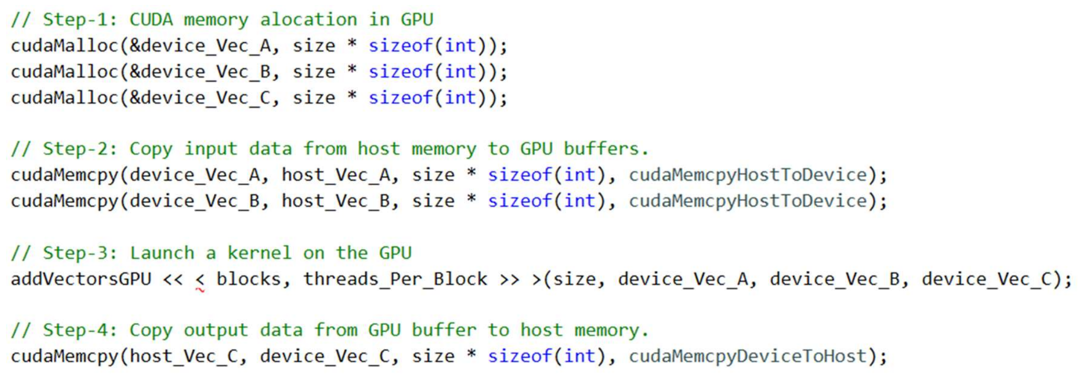


Figure 5: An example of the CUDA Indexing Pattern, where: gridDim.x (is the number of blocks), blockDim.x (is the number of threads in each block), blockIdx.x (is the index of the current block within the grid), and threadIdx. X (is the index of the current thread inside the block) [6].

Moreover, in CUDA, the CPU and GPU have distinct memory spaces, that characteristic requires a data transference between the two memories before and after the kernel execution. Figure 6 shows the data transference steps.



```
// Step-1: CUDA memory alocation in GPU
cudaMalloc(&device_Vec_A, size * sizeof(int));
cudaMalloc(&device_Vec_B, size * sizeof(int));
cudaMalloc(&device_Vec_C, size * sizeof(int));

// Step-2: Copy input data from host memory to GPU buffers.
cudaMemcpy(device_Vec_A, host_Vec_A, size * sizeof(int), cudaMemcpyHostToDevice);
cudaMemcpy(device_Vec_B, host_Vec_B, size * sizeof(int), cudaMemcpyHostToDevice);

// Step-3: Launch a kernel on the GPU
addVectorsGPU << < blocks, threads_Per_Block >> >(size, device_Vec_A, device_Vec_B, device_Vec_C);

// Step-4: Copy output data from GPU buffer to host memory.
cudaMemcpy(host_Vec_C, device_Vec_C, size * sizeof(int), cudaMemcpyDeviceToHost);
```

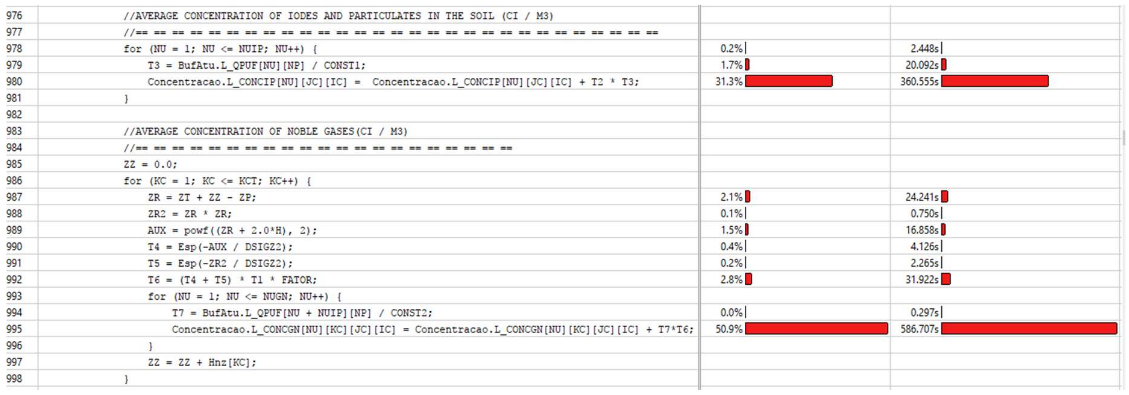
Figure 6: A piece of code with the basic steps to execute a kernel in CUDA. Those are: 1 - GPU memory allocation, 2 – Input data transferred from CPU memory to GPU memory, 3 – Kernel launching and 4 - Output data transferred back to the CPU memory.

It is important to point out, that the transference process between memories can be a handicap to the application speed up. Due to the fact, the greater the amount of data to transfer, the longer the transfer time will be and if the processing time is small when compared to transfer time, the speedup in the parallelism may be reduced.

4. THE PARALLELIZATION PROCESS

The Plume Dispersion program is divided into several functions that are executed by the main function *Tradif*. In order to define which parts of the program were the most relevant to the parallelization process, a performance profiler was used.

The result generated, displayed at figure 7, indicated that the slower parts were at the function *Average_Concentration_Calculation*, which is responsible to first, identify the concentration points on a plane, under the influence of the radioactive plume radius and then calculate the average concentration of Iodo and particulate matter, and noble gases.



```
//AVERAGE CONCENTRATION OF IODES AND PARTICULATES IN THE SOIL (CI / M3)
//== == == == == == == == == == == == == == == == == == == == == == == == == == == == == ==
for (NU = 1; NU <= NUIP; NU++) {                                                               0.2%      2.448s
    T3 = BufAtu.L_QPUF[NU][NP] / CONST1;                                                       1.7%     20.092s
    Concentracao.L_CONCIP[NU][JC][IC] =  Concentracao.L_CONCIP[NU][JC][IC] + T2 * T3;         31.3%    360.555s
}

//AVERAGE CONCENTRATION OF NOBLE GASES(CI / M3)
//== == == == == == == == == == == == == == == == == == == == == ==
ZZ = 0.0;
for (KC = 1; KC <= KCT; KC++) {
    ZR = ZT + ZZ - ZP;                                                                          2.1%     24.241s
    ZR2 = ZR * ZR;                                                                              0.1%      0.750s
    AUX = powf((ZR + 2.0*H), 2);                                                                1.5%     16.858s
    T4 = Esp(-AUX / DSIGZ2);                                                                    0.4%      4.126s
    T5 = Esp(-ZR2 / DSIGZ2);                                                                    0.2%      2.265s
    T6 = (T4 + T5) * T1 * FATOR;                                                                2.8%     31.922s
    for (NU = 1; NU <= NUGN; NU++) {
        T7 = BufAtu.L_QPUF[NU + NUIP][NP] / CONST2;                                             0.0%      0.297s
        Concentracao.L_CONCGN[NU][KC][JC][IC] = Concentracao.L_CONCGN[NU][KC][JC][IC] + T7*T6; 50.9%    586.707s
    }
    ZZ = ZZ + Hnz[KC];
}
```

Figure 7: Piece of the function *Average_Concentration_Calculation*. The red bars show the CPU utilization and time spent to execute that specific part of the program.

The sequential version of the function was controlled by five loops, the two external loops, represented the spatial dimensions (X, Y) of the plane, and were responsible for determining the distance from a concentration point to the center of the radioactive plume. The next inner loop was used in the calculation of the average concentration of Iodo and particulate matter (NUIP) at a position (I, J) at the ground level. The final two inner loops, one to add the spatial dimension (Z) to the plane and the other to control the calculation of the average concentration of noble gases (NUGN) at a position (I, J, K) of the three-dimensional grid. The number of iterations of each of these loops was X = 1072, Y = 688, NUIP = 32, Z = 4, NUGN =14.

Then, to start the parallelization process, a CUDA version of the function was developed using the methodology explained in chapter 3.3. The two external loops were removed, and the calculations of the average concentrations were transformed in two device functions (functions that can be invoked only by the kernel and executed by the GPU). One to calculate the average concentration of Iodo and particulate matter, other to estimate nobles gases average concentration. In the device functions, the loops that controlled the average concentration calculations (NUIP, Z, NUGN) were preserved, with a slight change being made at the indexation process, as showed in the chapter 3.3 figure 5, of the arrays that hold the concentrations. Note that, in this approach, the average concentration is calculated for several concentration points of the grid in parallel.

5. RESULTS

In order to evaluate the performance of the parallel model developed for the Plume Dispersion module, a hypothetical case of release of radioactive material to the atmosphere in the area around the Brazilian Nuclear Power Plant of Angra dos Reis was selected. The Nuclear Power Station (NPP) is at sea level, surrounded by mountains and the sea.

The sequential and parallel versions of the *Average_Concentration_Calculation* function have been executed, and the execution times of both are displayed in table 1. The time in the parallel version includes kernel execution, GPU memory allocation, and data transfer between GPU and CPU.

Table 1: Execution time of the versions of the function

Function Name	CPU time(s)	GPU time(s)[a]	Speedup
Average_Concentration_ Calculation	1112.1	95.62	11.63

a. Considering GPU kernel + GPU memory allocation + Memory transfer between GPU and CPU.

Taking into consideration the modifications made in the function were made using only the basic methods of the CUDA platform, a speedup of 11.63 is a good result. The speedup obtained is similar to the function *Stability_Interpolation* from Wind Field module, which achieved 10.2 [2]. This show that even in distinct modules (Wind Field and Plume Dispersion) of the ARDS the parallelization methodology is consistent, with similar results being achieved in functions of both modules.

6. CONCLUSIONS

In this work, the CUDA C/C ++ was employed to develop a version in parallel of the function (*Average_Concentration_Calculation*) part of the Plume Dispersion program, applied in the context of an Atmospheric Radionuclide Dispersion system (ARDS). Besides that, as a scenario to test and compare the execution time of both versions, sequential(C/C++) and parallel (CUDA), of the function, the vicinity's of Angra dos Reis Brazilian NPP was considered. The parallel version achieved a speedup of 11.63.

This result demonstrates that the parallelism methodology employed in this work and in Pinheiro et al [2], could be used to develop an integrated real-time ARDS for highly refined 3D-grids. Since, in both works, similar results were achieved using the same parallelization method, for distinct modules of the ARDS. The plan for future works is the development of parallel versions for the other modules of the ARDS to be used by the Brazilian NPPs.

REFERENCES

1. S.-U. Park, A. Choe, and M.-S. Park, "Atmospheric Dispersion and Deposition of Radionuclide (137Cs and 131I) Released from Fukushima Dai-ichi Nuclear Power Plant". *Computational Water, Energy and Environmental Engineering*, **Vol. 2**, n. 2, pp. 61-68 (2013).
2. A. Pinheiro, F. Desterro, M. Santos, C. Pereira, and R. Schirru, "GPU-Based Parallel Computation in Real-Time Modeling of Atmospheric Radionuclide Dispersion". *Advances in Intelligent Systems and Computing*, **Vol. 497**, pp. 323-333 (2017).
3. A. Fabrick, R. Sklarew, J. Wilson, "Point Source Model Evaluation and Development Study – The Grid Model IMPACT (Integrated Model for Plumes and Atmospherics in Complex Terrain)". Technical report (Contract A5-058-87), California Energy Resources Conservation and Development Commission. A Science Applications, Inc (1987).
4. A. Heimlich, A. C. A. Mol, and C. M. N. A. Pereira, "GPU-based Monte Carlo simulation in neutron transport and finite differences heat equation evaluation". *Progress in Nuclear Energy*, **Vol. 53**, n. 2, pp. 229-239 (2011).
5. C. M. N. A. Pereira, A. C. A. Mól, A. Heimlich, S. R. S. Moraes, and P. Resende, "Development and performance analysis of a parallel Monte Carlo neutron transport simulation program for GPU-Cluster using MPI and CUDA technologies". *Progress in Nuclear Energy*, **Vol. 65**, pp. 88-94 (2013).
6. "An Even Easier Introduction to CUDA," https://devblogs.nvidia.com/parallelforall/even-easier-introduction-cuda/ (2017).