Comparative Study of the Execution Time of Parallel Heat Equation on CPU and GPU

Safa Belhaous, Soumia Chokri, Sohaib Baroud, and Mohamed Mestari

***Abstract*—Parallelization has become a universal technique for computing an intensive scientific simulation to shorten the execution time of complex problems. It consists of bringing together the power of several thousand processors to perform complex calculations at high speed. The choice of the runtime environment to execute parallel programs significantly influences the execution time. For this reason, this article aims to materialize the impact of computing architectures on the performance of parallel implementations. To better achieve this contribution, we have implemented the heat equation executed on CUDA platform and we have compared the results with those of SkelGIS implementation from the literature. Through the results of the experiments, we demonstrated that the execution time of the CUDA implementation on graphics processing unit (GPU) is almost 100X faster for very large meshes compared to the other implementations.**

***Index Terms*—CUDA, GPU, Parallel implementation, Parallel architecture, Heat equation.**

I. Introduction

ACCORDING to [1], the computing architecture must present high-performance functionalities on a large scale, so as not to be an obstacle to the performance and efficiency of the program. Because if the execution environment does not evolve and the deployment time of the application is long, the profitability of the program is reduced by the time lost by the execution of the environment. Computing architectures are today more than ever under pressure to better serve scientific simulations and new applications requiring workloads. Processors or Central Processing Unit(CPU) focus on individual tasks and speed of execution. This makes it particularly well suited for tasks ranging from serial processing to running databases. Then the CPU focuses on processing a task as fast as possible. However, graphics processors or Graphics Processing Unit(GPU) have evolved to also become more generalist parallel processing processors to process the maximum possible tasks or a task for a maximum of data without exceeding a margin of time. In order to benefit from the services of these two types of processors, NVIDIA proposed in 2006 a new parallel programming platform called CUDA (Compute Unified Device Architecture) which supports the joint CPU and GPU execution of an application [2]. CUDA is based on the C language with a handful of extensions of certain keywords allowing heterogeneous programs[3]. CUDA offers simple APIs to manage devices, memory, and more[4].

Parallelization of scientific simulations consist to ameliorate performance on many different sets of computers[5]. Indeed many simulations have been some obstacles to be parallelized because of lack of time to program new parallel algorithm or lack of powerful hardwires such as GPUs. The well-known scientific simulation is the heat equation [6] because it is used generally in various scientific fields such as diffusion processes in physics, parabolic partial differential equations in mathematics, Black-Scholes option pricing in finance, and Brownian motion in probability [7]. The heat equation is the standard example of physical processes which can be modeled as a parabolic partial differential equation (PDE) [8], initially introduced in 1811 by Jean Baptiste Fourier. This problem can occur in one-dimensional [9], two-dimensional [10], three-dimensional bioheat equation [11] or n-dimensional [12] physical objects. Solving the heat equation problems of paramount importance in physics, applied mathematics, engineering, and medicine. In a Cartesian mesh (two-dimensional mesh), the prediction of the temperature distribution with the boundary conditions will be easier if the size of the domain is very small (see Figure 1(a)). In this figure, the redness indicates the temperature at each point, while the blue dots indicate the cold region. Over time, the heat diffuses into the cold area and the hot area gradually cools down until the grill reaches a uniform temperature. However, to iteratively predict the interior values (i, j) considering a Cartesian mesh of domain size 10000x10000, with 5000 iterations, the simulation will be more complicated as the size of the grid increases, and the the results will be less precise. This is why parallel implementations of the heat equation become so necessary to benefit from parallel architectures and have good results.

In the literature, there is many parallel implementations proposed to solve the heat equation in many domains [13]. The majority of these implementations were implemented using CUDA. However, each paper presented differently his contribution in depending on the handled problem and its mathematical modeling. For example in [10], the authors proposed a parallel solution of the three-dimensional heat equation through three implementations, namely, CUDA, MPI, and Open Multi-Processing (OpenMP). The proposed implementations consist in computing the temperature values on a discrete number of nodes in the mesh. The results obtained were compared with general-purpose CFD software to describe the relationship between the grid size and the execution time of each implementation. The authors concluded that as the computational domain

Manuscript received August 14, 2021; revised November 22, 2021. Date of publication December 21, 2021. Date of current version December 21, 2021.

Authors are with the SSDIA laboratory, ENSET, Hassan II University, Mohammedia, Morocco (e-mail of corresponding author: safaabelhaous@gmail.com).

Digital Object Identifier (DOI): 10.24138/jcomss-2021-0133

increases, from 100x100x100 to 500x500x500, the running time gradually increases. Based on these results, the GPU runtime of the CUDA implementation was the best compared to MPI and OpenMP. The resolution of the three-dimensional heat equation was not limited to this contribution, there is another one that used alternating direction implicit (ADI) [14] method to split the multidimensional into the three steps. The authors called these steps X-sweep, Y-sweep, and Z-sweep. In this paper, the heat equation was implemented using ADI-CUDA on GPU. To evaluate the proposed contribution, the authors compared the execution time obtained by ADI-CUDA implementation with those obtained by ADI implementation. Through this comparison, ADI-CUDA was the fastest for all grid sizes, from 10x10x10 to 50x50x50, except the first one and the speedup up to 11X. The ADI implementation took a long time because the resolution of multidimensional mesh was computationally expensive. Nevertheless, the CUDA-ADI implementation was the most rapidly due to the use of the local memory of the thread on the GPU. For solving the heat equation in parallel, [15] proposed a parallel model called SIPSim (Structured Implicit Parallelism on scientific Simulations). The implementation of this model called SkelGIS was programmed using C++ Libraries for resolving two cases of simulations: simulations on two-dimensional meshes particularly heat equation and network simulations. To evaluate the performance of SkelGIS, the authors compared the execution time of the proposed implementations with MPI. As a result, SkelGIS had better performance than the MPI version to solve a large mesh size (20000x20000 and 5000 iterations) up to 2048 cores. According to all the experiments carried out, where the mesh size was smaller than 20000, MPI rarely had better performance than SkelGIS. The authors concluded that SkelGIS was quite scalable when an expensive simulation needs to be calculated.

The organization of this paper is as follows: Section II describes the main contribution of this paper. Section III presents the mathematical modeling of the heat equation. Then, the section IV illustrates the CUDA platform to clarify fundamental components and their impact. Section V is devoted to the parallel implementations using CUDA and SkelGIS. The results and discussions are presented in section VI. Finally, the last section contains the conclusion.

II. Our Contribution

The main contribution of this paper consists to illustrate the impact of the runtime environment on the efficiency of parallelization. Because the runtime environment must allow communications that can be a critical factor in performance [1]. For example, in GPU, the communication between CPU and GPU, or the data movement, is an important part of the calculation. So, many researches evaluated the performance of CPU-GPU communication [16], [17]. In this paper, the impact of the runtime environment was evaluated by comparing two parallel implementations of the heat equation executed on two different runtime environments. The fist parallel implementation [15] used a new library SkelGIS to resolve the heat equation. This implementation was executed on a supercomputer. For the second parallel implementation, we had implemented it using CUDA on GPU. To better achieve the comparison, the experiences of the first implementation were respected, namely the mesh size started from 5120x5120 to 20000x20000. By comparing these two different environments, the main goal was to know the gain and the performance of each one.

III. Mathematical Modeling of the Heat Equation

The heat equation allows a great improvement in the mathematical modeling of the phenomena of temperature propagation in various fields such as physics, medicine and industry. The importance of this equation has aroused great interest from many researchers who have continuously put their efforts into solving this equation using many numerical methods, depending on their field of study. Rainer Kress defined the heat equation [18] as follow: " *The temperature distribution u in a homogeneous and isotropic heat-conducting medium with conductivity k, heat capacity c and mass density ρ satisfies the partial differential equation*". This equation is written in the following form:

$$\frac{\partial U}{\partial t} = \frac{\partial^2 U}{\partial x^2} + \frac{\partial^2 U}{\partial y^2} [15] \quad (1)$$

where the function U(x,y,t) designates the temperature at point (x,y) and at the time iteration t. It should be noted that the heat equation needs to be discretized through modern numerical methods according to [19], in particular those dedicated to the resolution of nonlinear PDEs. The most appropriate method for solving two-dimensional heat equation is the FDM [20]. This numerical method is based on the derivation from Taylors polynomial and used a regular mesh, typically a Cartesian grid (see Figure 1(b)) to discretize the spatial domain. The discretization step consists in replacing the continuous operators with their discrete approaches. The discretization of the continuous PDE is always replaced by a numerical approximation as it appeared in figure 2. In this particular circumstance, the word "discrete" implies that the numerical solution is defined only in a finite number of points of the physical space. Then, the calculation step is done through a numerical implementation of the approximate equation.

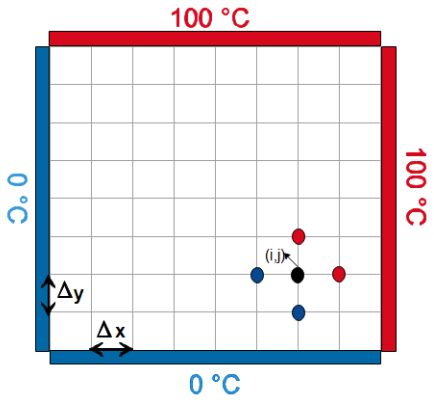
There are three schemes of FDM [22], firstly, the forward time centered space (FTCS) or explicit scheme which is considered as the least complex method based on the mathematical discretization for solving PDE through algebraic equations [10]. Then, the backward time centered space (BTCS) or implicit scheme is more complicated than the precedent scheme. Finally, Crank-Nicolson scheme which combined between both precedent scheme. Therefore, this paper aimed to use the explicit scheme [21] to approximate the derivative at each mesh point in order to obtain the discrete-space model.

The type of mesh considered in this paper is Cartesian, with $x_i = i\Delta x, y_j = j\Delta y$ and $t_n = n\Delta t$. To discretize the partial derivatives in time of the left hand side of (1), we use the right hand side approximation, then we have :

$$\frac{\partial U(x_i, y_j, t_n)}{\partial t} \approx \frac{U(x_i, y_j, t_n + \Delta t) - U(x_i, y_j, t_n)}{\Delta t} \quad (2)$$

	100	100	100	100	100	100	100	100	100	100	
0	50,89	71,06	80,25	84,7	86,6	86,6	84,7	80,25	71,06	50,9	100
0	30,99	51,02	62,89	69,5	72,4	72,4	69,49	62,89	51,02	31	100
0	21,15	37,64	48,96	55,9	59,1	59,1	55,88	48,96	37,64	21,1	100
0	15,33	28,31	37,99	44,3	47,4	47,4	44,28	37,99	28,31	15,3	100
0	11,41	21,46	29,31	34,6	37,2	37,2	34,6	29,31	21,46	11,4	100
0	8,527	16,18	22,31	26,5	28,7	28,7	26,54	22,31	16,18	8,53	100
0	6,264	11,94	16,56	19,8	21,4	21,4	19,79	16,56	11,94	6,26	100
0	4,403	8,415	11,7	14	15,2	15,2	14,02	11,7	8,415	4,4	100
0	2,804	5,366	7,475	8,97	9,74	9,74	8,968	7,475	5,366	2,8	100
0	1,364	2,612	3,642	4,37	4,75	4,75	4,372	3,642	2,612	1,36	100
	0	0	0	0	0	0	0	0	0	0	

(a)



(b)

Fig. 1. Cartesian mesh, (a) with heat transfer, (b) with boundaries condition.

The right hand side of (1) is approximated with the central approximation of second derivatives to get :

$$\frac{\partial^2 U(x_i, y_j, t_n)}{\partial x^2} \approx \frac{U(x_i + \Delta x, y_j, t_n) - 2U(x_i, y_j, t_n) + U(x_i - \Delta x, y_j, t_n)}{\Delta x^2} \tag{3}$$

$$\frac{\partial^2 U(x_i, y_j, t_n)}{\partial y^2} \approx \frac{U(x_i, y_j + \Delta y, t_n) - 2U(x_i, y_j, t_n) + U(x_i, y_j - +\Delta y, t_n)}{\Delta y^2} \tag{4}$$

Both the time and space derivatives are replaced by finite differences, we are using $U(x_i, y_j, t_n) = U^n_{i,j}$ to simplify the writing. Then, Equation (1) is approximated with :

$$\frac{U^{n+1}_{i,j} - U^n_{i,j}}{\Delta t} = \frac{U^n_{i+1,j} - 2U^n_{i,j} + U^n_{i-1,j}}{\Delta x^2} + \frac{U^n_{i,j+1} - 2U^n_{i,j} + U^n_{i,j-1}}{\Delta y^2} \tag{5}$$

Rearranging Equation (5) by supposing that $\Delta x = \Delta y$ we obtain the equation as follows:

$$U^{n+1}_{i,j} = (1 - 4\lambda)U^n_{i,j} + \lambda(U^n_{i+1,j} + U^n_{i-1,j} + U^n_{i,j+1} + U^n_{i,j-1}) \tag{6}$$

where $\lambda = \frac{\Delta t}{\Delta x^2} \leq 0.5$.

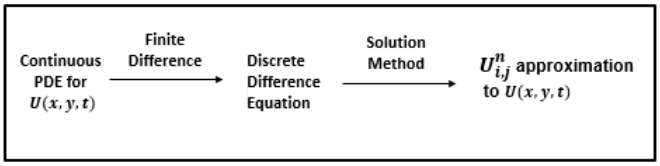


Fig. 2. Relationship between continuous and discrete problems[21].

IV. CUDA Platform

This section presents a detailed study on the CUDA platform used including a description of its parallel architecture, a presentation of its essential components, and above all an overview of the impact shown by the purpose of the software.

A. CUDA Architecture

The CUDA architecture is based on three hierarchical parts that help the programmer to use efficiently all the computing capacities of the graphics card. The CUDA architecture divides a hierarchical structure into grids, blocks, and threads, as shown in figure 4. CUDA names the function which will be executed in parallel *kernel*. This function is always started by the CPU and executed by the GPU. When running a *kernel* a set of threads is assigned and organized into one, two, or three-dimensional thread blocks. Each block in a grid contains the same number of threads, which will be sent to different cores of the same streaming multiprocessor (SM) and will be executed in parallel (see figure 3); while different blocks can also be executed on different SMs in parallel. The blocks are grouped in a one, two, or three-dimensional grid. Blocks are organized in an array (1D or 2D or 3D) of threads, each block has a unique (blocked) block ID in a grid. Threads execute the instructions specified by the kernel function; each thread

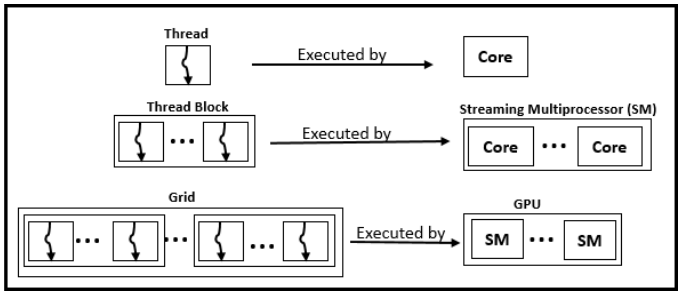


Fig. 3. Grid execution on GPU.

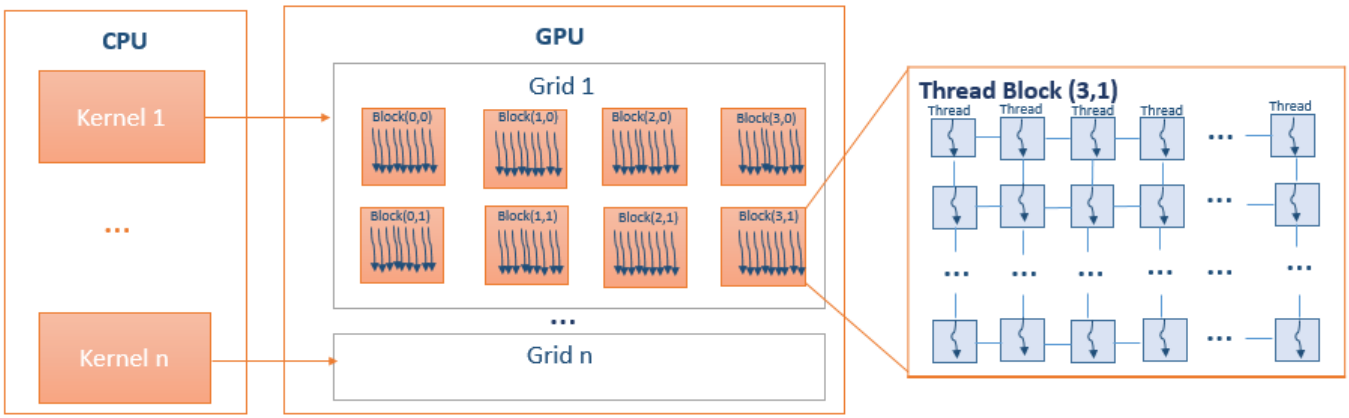


Fig. 4. CUDA architecture: thread, block and grid.

has a unique thread ID (ThreadIdx) in a block. This article used the NVIDIA GeForce GTX 750Ti GPU which is based on the maxwell architecture with five SM, each comprising 128 streaming processors (SP). In order to run a kernel on the GPU, the CPU allocates the necessary memory on the GPU and then transfers the data to the allocated memory. Then the CPU can call and run the kernel on the GPU. Finally, the results are returned to the CPU.

CUDA threads can access data from several types of memories while they are running, as shown in figure 5. Each thread in a block has its own registers used to store data relating to threads, frequently used by the thread such as the counter. Registers represent a quick space in the memory hierarchy but have little space. Each block in a grid has fast shared memory, but its size is smaller than the overall memory; it is commonly used for communication between threads of the same block. There is also constant memory. This is read-only memory space accessible by all threads in the grid. All threads in a grid have access to global memory; it is the slowest memory but has the largest size. This memory allows communication between all the blocks of the grid and the transfer of data between CPU and GPU.

B. Software Impact

The paper [10] developed a three-dimensional code to numerically solve the heat conduction equation on three parallel platforms which are MPI, OpenMP, and CUDA. The paper performed a comparison of computing time on the three parallel computing platforms over the serial program in different grid sizes ranging from 100x100x100 up to 500x500x500. The results of this article showed that the time taken by CUDA was always the best compared to that of MPI and OpenMP, for example in the case of large grid CUDA executed the program in 109.36(s) on the other hand, MPI in 511.27(s) and OpenMP in 454.8(s).

The use of CUDA is not limited to solving the heat equation [14][15][11] but several works that have used CUDA to solve other problems namely the parallel implementation of the hybrid intuitionistic fuzzy edge detection algorithm [23], the routing based on the neural network thanks to the parallelism exploit [24], the calculation of the correlation function of ensembles of Pseudo-random sequences formed automatically by CUDA program [25], and optimization of tasks that require massively parallel calculations to produce an effective implementation of the integral image algorithm[26].

V. Parallel Implementations

This section presents two different implementations of the two-dimensional heat equation. The parallel implementation via the CUDA platform is the contribution of this paper, Compared with the SkelGIS implementation that was published in the literature[15].

A. CUDA Implementation

This section presents the CUDA implementation of the two-dimensional heat equation. According to (6), the parallel implementation [27] for solving the heat equation will be more practical using CUDA (see algorithm 1).

From the algorithm 1, the symbols $Temp_{old}$, $Temp_{new}$ represent in (6), respectively, $U^n_{i,j}$, $U^{n+1}_{i,j}$. The symbols i,j represent x and y. For the neighbours $Temp_{old}[east]$, $Temp_{old}[west]$, $Temp_{old}[north]$, and $Temp_{old}[south]$ represent, respectively, $U^n_{i+1,j}$, $U^n_{i-1,j}$, $U^n_{i,j+1}$, and $U^n_{i,j-1}$. A CUDA program is sorted into a host program, comprising at least one successive thread executing on the host CPU, and at least one parallel kernel which is suitable for execution on a parallel processing device as a GPU. The parallel implementation of the heat equation using CUDA starts with many initialization such as the size of the mesh (with x and y), the number of Jacobi iterations, the temperature condition and declaration of pointers to Host (CPU) and peripheral memories (GPU). Then, for the computation, the code contains two allocations: an allocation of an array on the Host and a memory allocation on the Device to copy the data from the Host to the Device. In addition, lines 7 and 8 of the algorithm 1 consist in assigning a 2D distribution of CUDA ”threads” within each CUDA ”block” and calculating the number of blocks in the CUDA ”grid”. The main loop of the program is the Laplace function (see Algorithm 1) which calculates the values of the heat equation at each Jacobi iteration. The Laplace function,

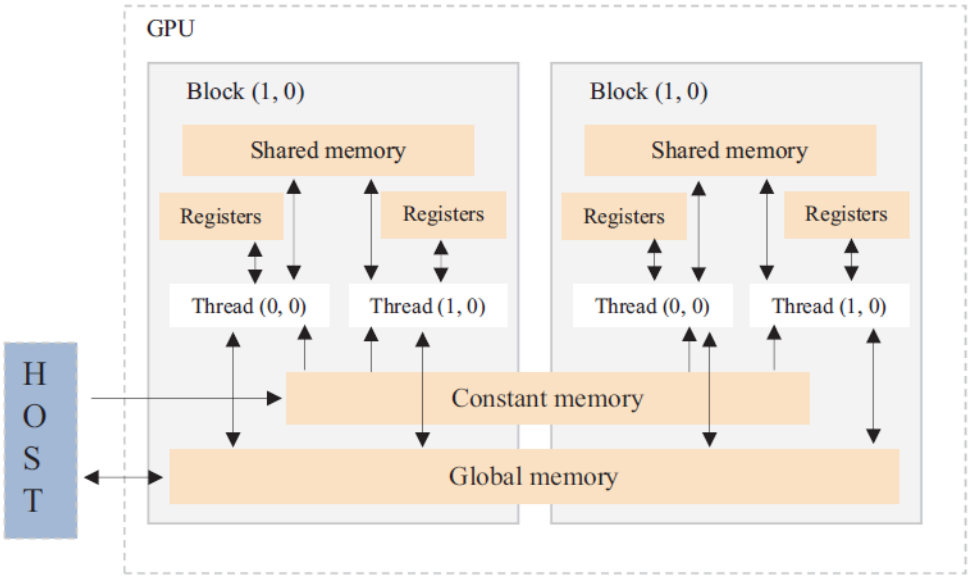


Fig. 5. CUDA Memories ([11]).

which predicts the propagation of heat, has been declared as a kernel.

Algorithm 1 : CUDA implementation of the heat equation (6)

1: Identify mesh size and a number of Jacobi iterations.
2: Initialize the condition of temperature
3: Declaration of the pointer to Host (CPU) memory and pointers to Device (GPU) memory.
4: Allocate and initialize an array on the host for pre-computation.
5: Allocate Memory on Device.
6: Copy data from CPU memory to GPU memory.
7: Assign a 2D distribution of CUDA "threads" within each CUDA "block".
8: Calculate the number of blocks in CUDA "grid".
9: **function** LAPLACE($^*Temp_{old}, ^*Temp_{new}$)
10: $i = blockIdx.x * blockDim.x + threadIdx.x$;
11: $j = blockIdx.y * blockDim.y + threadIdx.y$;
12: $current = i + j * NX$;
13: $north = i + (j + 1) * NX$;
14: $south = i + (j - 1) * NX$;
15: $east = (i + 1) + j * NX$;
16: $west = (i - 1) + j * NX$;
17: $\lambda = 0.05$;
18: **if** $i\langle 0$ && $i\langle NX - 1$ && $j\langle 0$ && $j\langle NY - 1$ **then**
19: $Temp_{new}[this] = (1 - 4\lambda)\ Temp_{old}[this] + \lambda(Temp_{old}[east] + Temp_{old}[west] + Temp_{old}[north] + Temp_{old}[south])$
20: **end if**
21: **end function**
22: Copy the final array from Device to Host.
23: Release the Allocated Memory on Device and host.

A kernel runs a consecutive scalar program on all parallel threads. The threads of the same block cooperate via shared memory and synchronization barriers. Each thread sorts how much it changes and updates the value in the corresponding memory location. When all the threads have finished updating their point, we move on to the next iteration. Finally, the program copies the final table from the device to the host and frees the memory allocated on the device and the host.

B. SkeLGIS Implementation

SkelGIS is a library proposed by [15] to furnish a programming interface based on C++ libraries, to hide parallelism for a non-computer scientist and to generate parallel code. The author chose the C++ language to implement SkelGIS due to its advanced characteristics to enable an important abstraction level while producing efficient codes. For producing the final program, SkelGIS uses MPI. SkelGIS library relies on the Structured Implicit Parallelism on scientific Simulation model (SIPSim) to solve PDEs. This model is an implicit parallel solution that hides parallelism to offer to users an easy development tool based on sequential programming. In order to generate the final parallel program, SIPSim takes care of three important tasks, namely the distribution of data, the execution of the algorithm, and communications.In this model, these tasks came true through four main components such as distributed data structures (DDS), distributed data mapped to DDS (DPMap), applicators/operations, and interfaces.

The DDS represented the data structure of the mesh and to make access to the elements of the mesh more efficient. DDS used a partitioning approach to distribute mesh between processors. The partitioning aimed to resolve automatically the sub-element of the problem. The utilization of the load balancing of data between processors was for achieving a good efficiency. The load-balancing [28] problem has been handled through many theses of the literature.

The DPMap aimed to map data on the model when some DDS was available. Data mapped on DDS defined the quan-

tities to simulate, like the concentration, the pressure, and the flow. DPMaps Instantiations are the most widely used by users.

Appliers and operations were developed as functions to follow interchanges between processors before executing the sequential program of the user. The parallelization of this program followed the Bulk Synchronous Parallel (BSP) model [29] that is generally utilized to implement SPMD programs. According to this model, each processor used the numerical data stored in its local memory.

The interfaces provided sequential interfaces for the implementation of the mesh. The authors proposed three types of interfaces: iterators, get/set and neighborhoods. In the SIPSim model, an iterator was almost as assimilated as an iterator in C ++ STL that iterated through a DDS. The user instantiated the iterator to manipulate a set of mesh values and the get/set interface was used to handle the data values. The proposed parallelism model is flexible because it is made of various data structures. To evaluate the performance of this model, SkelGIS was executed to solve two different aspects of simulations : simulations of two-dimensional mesh and multiphysics simulations connected through a network. Regarding the scalability of the proposed tool, the experiments was performed on two different clusters.

Algorithm 2 : Main function in SkelGIS to solve heat equation[15]

```
1:  include SkelGIS library.
2:  function MAIN(int argc, char** argv)
3:      Initialisation of the library by INITSKELGIS;
4:      Declaration of the head with HEAD type;
5:      Initialisation of the head;
6:      DMatrix < double, 1 > m(head, 0);
7:      m.setGlobalMiddleValue(1);
8:      DMatrix < double, 1 > m2(head, 0);
9:      for (i=0;i<100;i++) do
10:         Called Laplacien function;
11:         DMatrix < double, 1 > m3(head, 0);
12:         m = m2;
13:         m2 = m3;
14:     end for
15:     ENDSKELGIS;
16: end function
```

Algorithm 2 illustrates the SkelGIS program to solve the two-dimensional heat equation. From the equation (2), the results of the computation at time n+1 was based on these at time n. According to line 6 and line 8, the code necessitate two matrices to recover the input and output of the function. There is a main loop (lines 9 to 14) that consist of manipulate the size of the mesh and call the Laplacian function to compute in each time iteration the numerical value of the equation (2).

VI. Results and Discussions

The parallel implementation of the heat equation was executed on an NVIDIA GeForce GTX GPU with 1280 CUDA cores and 10606 GB. Concerning the accompanied CPU is an Intel i7-4770k with 4 cores at a speed of 3.50 GHz and 8 threads. Table II presents the execution times performed to solve the heat equation according to four experiments (see table I). The first experiment was a mesh of 5120x5120 with 5000-time iterations. In this experiment, the CUDA program was only executed in 8.85(s) on GPU with 1280 cores, but MPI and SkelGis with 2048 cores executed, respectively, in 67.49(s) and 103.363(s). In the second experiment, the domain size remains the same as the precedent experiment with a change in the number of iterations which will become 20000. The execution time was multiplied by 4 as the case of the number of iterations. In the two last experiments, the time iterations stay fixed and the domain size will be changed. As a result, the domain size had more effect on the result than the time iterations. In addition, always the CUDA program was executed in the best time, for example in the first experiment SkelGIS was executed by 2048 cores but CUDA was executed with just 1280 cores, SkelGIS took a long time 103.363(s) compared to CUDA 8.85(s).

The main objective of this comparison was to evaluate the efficiency of the three platforms by solving the heat equation under the same experiments. In general, the domain size had a great influence on the execution time. The results of SkelGIS Library presented in table II was published by [15]. The results of SkelGIS and MPI were similar compared to those obtained by CUDA. We can concluded some important results. Firstly, the SkelGIS version becomes less efficient than the MPI version in the three experiments. But, for the last experiment, SkelGIS had better performances than the MPI. This point confirms that SkelGIS is very compatible when a costly simulation should be computed. The figure 6 shows the evolution of the execution time of each implementation according to the four experiments presented in the table I. The graphical curves on the left show a comparison between CUDA and other implementations running in an environment containing 1024 cores. According to the presented results, whereas the size of the experiments increases, whereas the execution time increases very significantly. On the other hand, the CUDA implementation shows its performance and its speed of execution, which does not exceed 100(s). In this case, SkelGIS was better than MPI. The graphical curve on the right shows the execution in an environment containing 2048 cores for the case of MPI and SkelGIS. However the CUDA implementation always runs on an environment containing only 1280 cores. There is a great similarity between the results presented through these graphical curves and the previous ones. The only clear difference is that this time MPI is faster than SkelGIS. However, in both cases, the CUDA implementation running on a GPU gave the best results.

TABLE I
Experiments evaluated on CUDA, MPI and SkelGIS implementations

Experiments	Time iterations	Domain size
EXP1	5000	5120x5120
EXP2	20000	5120x5120
EXP3	5000	10000x10000
EXP4	5000	20000x20000

TABLE II
COMPARISON BETWEEN EXECUTION TIME CUDA, SKELGIS AND MPI

	CUDA (s)	SkelGIS Library (s)([15])		MPI [15]	
	Cores=1280	Cores=2048	Cores=1024	Cores=2048	Cores=1024
EXP1	8.85	103.363	155.85	67.49	144.26
EXP2	35.24	407.73	602.66	273.785	557.621
EXP3	34.62	3196.71	5127.93	2930.48	5739.552
EXP4	141.11	10986.8	19821.3	11867.4	22299.1

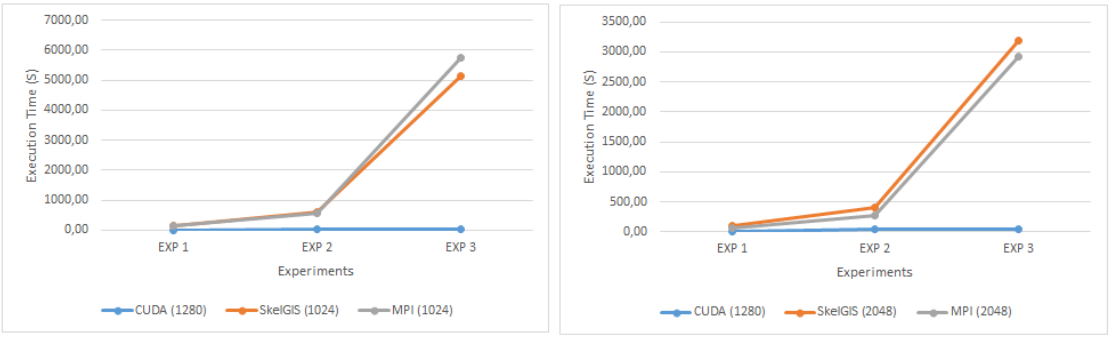


Fig. 6. Progress of the execution time, according to the three implementations

Finally, we notice that the difference between execution times obtained using the CUDA version and those using the SkelGIS version is very vast. Always the execution time of the CUDA code was faster than the SkelGIS and MPI, especially in the two last experiences. We deduce that the GPU with CUDA offers high performance at a very low cost because the GPU approach is a general-purpose unit for resolving a given complex problem. GPU computing aims to reach the highest performance for data-parallel problems via a massive parallel program that is carried out on the GPU. However, SkelGIS was slow because its overheads were brought by additional interfaces and objects used to hide the parallelism. This library was executed on a Curie supercomputer in CEA's Very Large Computing Centre (TGCC). The hardware of the cluster contained a Sandy Bridge processor with 2.7 GHz, 16 cores/Node, and 64 GB as the memory size by the node.

VII. CONCLUSION

In this paper, we focused on the effectiveness of parallel computing for solving scientific simulations on GPU. We chose the heat equation as an application simulation. For solving this differential equation, the explicit scheme of the FDM was used for the discretization. Then, the numerical implementation of the discrete form of the equation was done using CUDA. We performed a comparative study based on the literature to assess the proposed implementation and its performance. Through the execution of the different implementations (CUDA, SkelGIS and MPI), using several experiments, we deduce that our proposed implementation gave efficient results, thanks to the use of the GPU architecture. In spite of the parallelism for the other implementations of the literature, the chosen execution environment badly influenced the results.

ACKNOWLEDGMENT

This project has received funding from:

- The European Union's Horizon 2020 research and innovation program under the Marie Skłodowska-Curie grant agreement No 777720.
- The National Center for Scientific and Technical Research Morocco grant agreement No 10UH2C2017.

REFERENCES

[1] C. Coti, "Runtime environments for parallel applications communicating by pass messages for large-scale systems and worksheets," Ph.D. dissertation, Paris Sud University - Paris XI (In French), 2009.

[2] NVIDIA, "Cuda c++ programming guide," May 2021. [Online]. Available: https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html

[3] Nvidia, "Programming Guide : CUDA Toolkit Documentation." [Online]. Available: https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html

[4] ——, "Introduction to cuda c," September 2010. [Online]. Available: https://www.nvidia.com/content/GTC-2010/pdfs/2131_GTC2010.pdf

[5] A. Zhang, T. Li, Y. Si, R. Liu, H. Shi, X. Li, J. Li, and X. Wu, "Double-layer parallelization for hydrological model calibration on HPC systems," *Journal of Hydrology*, vol. 535, pp. 737–747, 2016. [Online]. Available: http://dx.doi.org/10.1016/j.jhydrol.2016.01.024

[6] X. Ji, T. Cheng, D. Li, and Q. Wang, "Solving large-scale three-dimensional heat equations on CUDA," *ICACTE 2010 - 2010 3rd International Conference on Advanced Computer Theory and Engineering, Proceedings*, vol. 2, pp. 38–42, 2010.

[7] B. S. Choi, C. Kim, H. Kang, and M. Y. Choi, "General solutions of the heat equation," *Physica A: Statistical Mechanics and its Applications*, vol. 539, 2020.

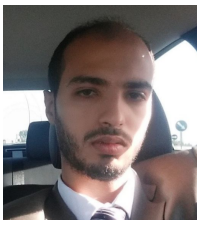
[8] H.-D. Tran, T. Bao, and T. T. Johnson, "Discrete-Space Analysis of Partial Differential Equations," in *ARCH18. 5th International Workshop on Applied Verification of Continuous and Hybrid Systems*, vol. 54, 2018, pp. 185–173.

[9] H. W. Liu and L. B. Liu, "A non-polynomial spline method for solving the one-dimensional heat equation," in *2008 International Symposium on Information Science and Engineering, ISISE 2008*, vol. 2, no. 8, 2008, pp. 782–785.

[10] V. Sivanandan, V. Kumar, and S. Meher, "Designing a parallel algorithm for Heat conduction using MPI, OpenMP and CUDA," in *2015 National Conference on Parallel Computing Technologies, PARCOMPTECH 2015*, 2015, pp. 1–7.
[11] A. Bousselham, O. Bouattane, M. Youssfi, and A. Raihani, "3D brain tumor localization and parameter estimation using thermographic approach on GPU," *Journal of Thermal Biology*, vol. 71, pp. 52–61, May 2018.
[12] S. Wanchak, *ON THE BESSEL HEAT EQUATION IN N-DIMENSIONAL RELATED TO DIAMOND BESSEL OPERATOR*. Nova Science Publishers, 2020, no. 9.
[13] T. Chakkour, "Parallélisation de l ' équation de la chaleur sous mpi et fftw," *HAL Id : hal-01700612*, 2018.
[14] Y. D. Bhadke, M. R. Kawale, and V. Inamdar, "Development of 3D-CFD code for heat conduction process using CUDA," in *2014 International Conference on Advances in Engineering and Technology Research, ICAETR 2014*, 2014, pp. 0–4.
[15] H. Coullon and S. Limet, "The sipsim implicit parallelism model and the skelgis library," *Concurrency and Computation: Practice and Experience*, vol. 28, no. 7, pp. 2120–2144, may 2016.
[16] D. Seo, S.-g. Kim, H. Eom, and Y. Y. Heon, "Performance evaluation of cpu-gpucommunication depending on thecharacteristic of co-located workloads," *International Journal on Computer Science and Engineering (IJCSE)*, vol. Vol. 5 No. 05, p. 280–285, 2013.
[17] T. B. Jablin, P. Prabhu, J. A. Jablin, N. P. Johnson, S. R. Beard, and D. I. August, "Automatic cpu-gpu communication management and optimization," *SIGPLAN Not.*, vol. 46, no. 6, p. 142–151, jun 2011. [Online]. Available: https://doi.org/10.1145/1993316.1993516
[18] R. Kress, "The heat equation," in *Linear Integral Equations*, vol. 82. Springer, Berlin, Heidelberg, 1989, pp. 378–387.
[19] L. Gavete, F. Ureña, J. J. Benito, A. García, M. Ureña, and E. Salete, "Solving second order non-linear elliptic partial differential equations using generalized finite difference method," *Journal of Computational and Applied Mathematics*, vol. 318, pp. 378–387, 2017.
[20] S. Belhaous, M. Bentaleb, S. Chokri, A. Naji, and M. Mestari, "Implementation of parallel algorithm for heat equation using SkelGIS library, CUDA and SISAL," in *Proceedings of the 2018 International Conference on Optimization and Applications, ICOA 2018*, 2018.
[21] G. W. Recktenwald, "Finite-Difference Approximations to the Heat Equation," 2004. [Online]. Available: http://dma.dima.uniroma1.it/users/lsa_adn/MATERIALE/FDheat.pdf
[22] A. Cerovský Ana Dulce André Ferreira Teacher and A. Dias dos Santos, "Application of the Finite Difference Method and the Finite Element Method to Solve a Thermal ," Porto, march 2014.
[23] E. Yalçin, H. Badem, and M. Güneş, "Cuda-based hybrid intuitionistic fuzzy edge detection algorithm," *IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, pp. 1–6, 2015.
[24] S. Zoican, R. Zoican, and D. Galatchi, "Neural network routing implementation using cuda technology," *13th International Conference on Advanced Technologies, Systems and Services in Telecommunications (TELSIKS)*, pp. 275–278, 2017.
[25] D. N. Shubin and E. M. Lobov, "Using cuda technology to form ensembles of pseudo-random sequences and calculating their correlation functions," *Systems of Signal Synchronization, Generating and Processing in Telecommunications (SINKHROINFO)*, pp. 1–5, 2017.
[26] M. Afif, Y. Said, and M. Atri, "Efficient implementation of integrall image algorithm on nvidia cuda," *International Conference on Advanced Systems and Electric Technologies (IC-ASET)*, pp. 1–5, 2018.
[27] A. Joshi, "2d heat conduction-solving laplace's equation on the cpu and the gpu," 2013. [Online]. Available: http://www.joshiscorner.com/2013/12/2d-heat-conduction-solving-laplaces-equation-on-the-cpu-and-the-gpu
[28] S. Chokri, S. Baroud, S. Belhaous, M. Bentaleb, M. Mestari, and M. El Youssfi, "Heuristics for dynamic load balancing in parallel computing," in *Proceedings of the 2018 International Conference on Optimization and Applications, ICOA 2018*, 2018.
[29] L. G. Valiant, "A bridging model for multi-core computing," *Journal of Computer and System Sciences*, vol. 77, no. 1, pp. 154–166, 2011.

Safa Belhaous received the master degree in Distributed Information Systems from the Higher Normal School of Technical Education (ENSET) in Morocco, in 2015. She is currently preparing a thesis, in SSDIA Laboratory, titled Modeling and implementation of implicit parallelism for scientific simulations. Her current research interests include Heat equation, parallel computing, A star search algorithm, smart cities.

Soumia Chokri obtained his Master degree in Distributed Information Systems from the Higher Normal School of Technical Education (ENSET), Hassan II University, Mohammedia, Morocco, in 2015. She is currently a PhD candidate in Dynamic load balancing for parallel and Distributed computing based on graph partitioning at ENSET. Her research interests include Parallel Computing, Dynamic Load balancing, Artificial Intelligence and Neural Networks, HPC.



Sohaib Baroud obtained his master degree in Geographical information system and territory management from Hassan II university and actually prepares a PHD thesis in Applying artificial intelligent techniques for analysing remotely sensed images. His current research interests include neural networks, parallel computing, optimization, remote sensing,and image processing, geographical information system. Since 2010 he participated to numerous studies concerning mapping and diagnosis of several territories in Morocco.

Mohammed Mestari is Co-founder of the International Neural Network Society Morocco Chapter (INNS Morocco Chapter), Co-founder of the IEEE Computational Intelligence Morocco Chapter, and Member of the IEEE Transactions on Neural Network and Learning Systems (IEEE TNNLS). His current research interests include neural networks for image classification, data-driven approach, neural networks hardware implementation, high-speed and/or low-power techniques and systems for neural networks, and theoretical issues directly related to hardware implementation of techniques based on the principle of decomposition coordination for optimal control and trajectory planning for an Unmanned Aerial Vehicle (UAV) and a robot. He has more than 120 scientific publications regarding both theory and applications in various domains of Artificial Intelligence and Robotic.