Accelerating Wild Fire Simulator Using GPU

C. Carrillo(✉), T. Margalef, A. Espinosa, and A. Cortés

Computer Architecture and Operating Systems Department,
Universitat Autònoma de Barcelona, Barcelona, Spain
{carles.carrillo,tomas.margalef,antoniomiguel.espinosa,
ana.cortes}@uab.cat

Abstract. In the last years, forest fire spread simulators have proven to be very promising tools in the fight against these disasters. Due to the necessity to achieve realistic predictions of the fire behavior in a relatively short time, execution time may be reduced. Moreover, several studies have tried to apply the computational power of GPUs (Graphic Processors Units) to accelerate the simulation of the propagation of fires. Most of these studies use forest fires simulators based on Cellular Automata (CA). CA approaches are fast and its parallelization is relatively easy; conversely, they suffer from precision lack. Elliptical wave propagation is an alternative approach for performing more reliable simulations. Unfortunately, its higher complexity makes their parallelization challenging. Here we explore two different parallel strategies based on Elliptical wave propagation forest fire simulators; the multicore architecture of CPU (Central Processor Unit) and the computational power of GPUs to improve execution times. The aim of this work is to assess the performance of the simulation of the propagation of forest fires on a CPU and a GPU, and finding out when the execution on GPU is more efficient than on CPU. In this study, a fire simulator has been designed based on the basic model for one point evolution in the FARSITE simulator. As study case, a synthetic fire with an initial circular perimeter has been used; the wind, terrain and vegetation conditions have been maintained constant throughout the simulation. Results highlighted that GPUs allow obtaining more accurate results while reducing the execution time of the simulations.

Keywords: Wild fire simulator · Fire front propagation · GPU

1 Introduction

The impact and the damage caused by Forest Fires has been increasing significantly over the last years. In the last decades, several fire spread models have been developed and implemented in computing simulators to help control centers in taking the adequate decisions. However, wildfires are complex systems characterized by a stochastic behaviour, with a large number of involved variables.

J. M. F. Rodrigues et al. (Eds.): ICCS 2019, LNCS 11540, pp. 521–527, 2019.
https://doi.org/10.1007/978-3-030-22750-0_46

Accurate simulators tend to take longer execution times. So, their effectiveness in real-time prediction is reduced. In order to improve the performance of fire spread simulators, several strategies have been developed to reduce the execution time without altering the accuracy of the simulations. In this context, some studies apply multicore architectures by increasing computational power, raising the number of CPUs, [1–3]. At the same time, the increase in the computational power of the Graphical Processing Units (GPUs) has turned them into an ideal tool for the modelling of complex systems. Different works have been carried out to apply the computational capacity of GPUs to accelerate the simulation of forest fire behavior [5,8,9,11,13]. These works have focused on the application on simulators based on Cellular Automata (CA). The main problem is that the simulators based on the CA approach have low intrinsic accuracy. Simulators based on the Huygens principle, or Elliptical Wave Propagation, have higher precision than those based on CA; however, their execution time is higher. In the present work, we focused on FARSITE (*Fire Area Simulator*) [6], which is a forest fire simulator based on the Elliptical Wave Propagation. Two different parallelizations are proposed; on the one hand, we have extracted the FARSITE simulation kernel and implemented it in OpenMP (*Open Multi-Processing*) [4], which is a set of compiler directives, library routines, and environment variables that can be used in any multicore CPU. On the other hand, we used CUDA (*Compute Unified Device Architecture*) to execute the simulation kernel in GPU. The aim of this work is to evaluate the performance of the two parallel strategies and analyse when the execution of one is more efficient than the other. As a first approach to the problem, a synthetic fire is used, which consists of a circular front in flat terrain, with constant wind speed, wind direction and the vegetation conditions throughout the simulation. To be able to compare the different executions (GPU and CPU) the simulations have been performed with different time propagation, in order to analyze in which conditions the execution in the CPU is more efficient than the execution in GPU. This paper is organized as follows. In Sect. 2 the principal characteristics of FARSITE are presented. Section 3 details the methodology used. Section 4 presents the experimental results and, finally, Sect. 5 summarizes the main conclusions and future work.

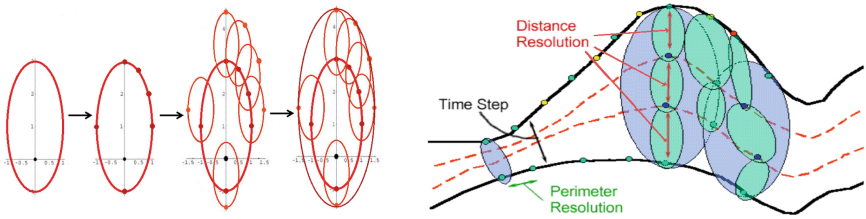
2 FARSITE Forest Spread Simulator

FARSITE is a simulator which spreads the front of the fire resolving Rothermel's equation [12]. The Rothermel's model is formulated in the following way:

$$R = R_0 \cdot (\overrightarrow{n} + \overrightarrow{\phi}_w + \overrightarrow{\phi}_s) \tag{1}$$

where R_0 represents the rate of spread in a particular point with no wind and no slope, $\overrightarrow{n}$ is the normal direction to the fire perimeter on that particular point, $\overrightarrow{\phi}_w$ is the wind factor and $\overrightarrow{\phi}_s$ the slope factor. In the Elliptical Wave Propagation, the perimeter of the fire is divided into series of points, [10]. To obtain the evolution of the fire perimeter, an ellipse is generated for each point. The shape of the ellipses is determined by the local characteristics at each point.

In this way, the new perimeter is obtained by joining the obtained ellipses, see Fig. 1(a).



(a) Elliptical wave propagation from t_1 to t_2.

(b) Graphic representation of time step (Black), Perimeter Resolution (Green) and Distance Resolution (Red), [7].

Fig. 1. Forest Fires Spread Simulator. (Color figure online)

In FARSITE there are three different parameters which have a direct impact on the resolution and, therefore, on the execution time [7], see Fig. 1(b):

- **Time Step:** The time step is the maximum amount of time that the conditions at a given point are assumed constant so that the position of the fire front can be projected.
- **Perimeter Resolution:** The perimeter resolution determines the maximum distance between points used to define the fire perimeter. The perimeter resolution controls the ability of a fire perimeter to respond to heterogeneities occurring at a fine scale.
- **Distance Resolution:** The distance resolution is the maximum projected spread distance from any perimeter point. This distance cannot be exceeded in a time step before new local data are used to compute the spread rate.

The precision of FARSITE is directly proportional to the number of points in which the fire front is split. The higher the number of points in the fire perimeter (low *Perimeter Resolution*), the more detail can be reproduced the fire fronts behaviour, consequently, the accuracy of the simulation will be better; therefore, the execution time is longer.

3 Parallelization of Forest Fire Simulator

We extracted the FARSITE simulation kernel and re-implemented in parallel into the FARSITE body. When the fire is propagated in serial, at each time iteration the propagation of the points is done sequentially. Consequently, when the number of points to expand increases, the execution time also increases proportionally. So, simulations with high resolutions provide long execution times, which limits their use in real situations. In the parallel implementations, the

point propagation was carried out in parallel. At each time iteration, each thread computes the spread of a single point. When the evolution of all points is finished, the threads are synchronised, and the spread in the next time iteration is performed. All calculations were performed in double precision.

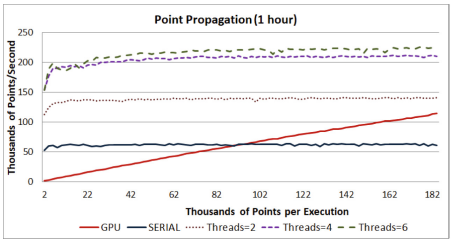
In order to parallelize the code on CPU, we have re-written the simulation kernel code thoroughly with OpenMP. For implementing the code on GPU accelerators, the simulation kernel code has been re-written with CUDA. All data are copied at the beginning of the simulation from the *Host* to *Device*. However, the perimeter data is copied from the *Device* to the *Host* at the final of each time iteration. Each thread only computes the evolution of one single point. We are interested in the evolution of the throughput in a series of simulations with increasing the number of points; we look for the number of points on which the execution in the GPU is faster than the CPU or when the number of propagated points per second is higher in the GPU than in the CPU.

As a first approximation, a synthetic fire was used with an initial circular perimeter. In this particular case, it has been considered a flat terrain, with homogeneous vegetation and constant wind speed and wind direction during the whole simulation.

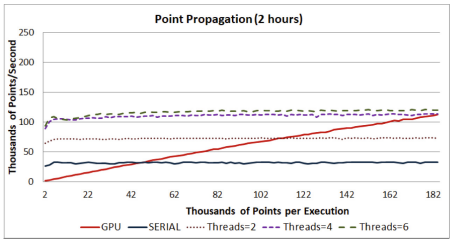
4 Experimental Study and Results

All calculations reported here were performed using a single GPU and single CPU; we measured the serial CPU performance using a single core, 2, 4 and 6 cores. As execution platform, we have used an Intel(R) Xeon(R) CPU E5-2620 v3 @ 2.40 GHz, with 6 cores and for the GPU simulations, a GeForce RTX 2080 Ti with 4352 CUDA cores was used. The tested propagation times were *1*, *2*, *5* and *10* h. The *Perimeter Resolution* was modified in each execution by increasing two thousand points at each simulation, so the extreme cases are $2,000$ and $184,000$ points. The higher the number of points, the higher resolution employed for the simulation.

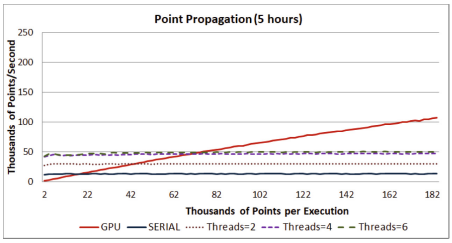
Figure 2 displays the number of points per second of the CPU implementations and the GPU implementation for the different propagation time. In Fig. 2(a) we can see that above $132,000$ perimeter points, the GPU implementation is more efficient than the Serial implementation. However, for this propagation time, all the OpenMP implementations are faster than the GPU application. It can observe that we are in front of a compute-bound problem, so the CPU is quickly saturated in all cases (below $8,000$ points), while the number of propagated points per second grows linearly in the GPU implementation. For 1 h of propagation time, the OpenMP implementations compute more points per second, which means that the OpenMP implementations are more efficient than the GPU implementation. Moreover, we see that the maximum number of propagated points per second decrease faster for all CPU implementations when the propagation time increase than for the GPU. Figure 3(a) shows the maximum propagated points per second for all implementations. It can be seen how the maximum of propagated points per second for each implementation decreases when the time of propagation is increased.



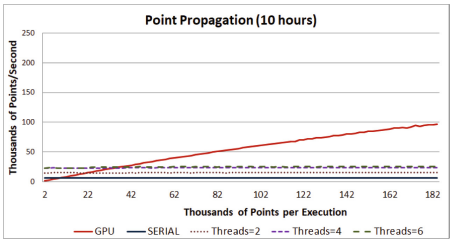
(a) 1 hour of fire propagation.



(b) 2 hour of fire propagation.



(c) 5 hour of fire propagation.



(d) 10 hour of fire propagation.

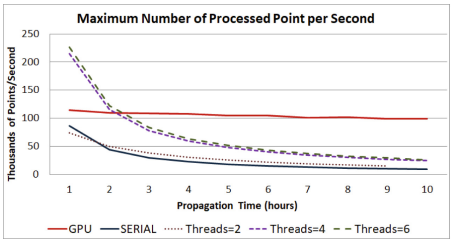
Fig. 2. Points per second depending on the number of perimeter points by the CPU and GPU.

Table 1. Number of points from which the execution in the GPU is more efficient than the CPU execution in Serial and with 2, 4, and 6 cores.

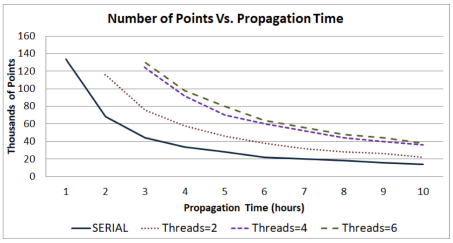
Propagation time	Number of points			
	Serial	2 Cores	4 Cores	6 Cores
1 h	132,000			
2 h	66,000	112,000		
5 h	28,000	44,000	70,000	74,000
10 h	14,000	22,000	34,000	38,000

Figure 3(b) shows the number of points from which the efficiency of the GPU is higher than the efficiency of the CPU. This number depends on the propagation time of the fire. The longer the time propagation of the simulated fire is, the less number of perimeter points is necessary so that the execution in the GPU is faster than the CPU (Table 1).

In Fig. 4 we can see the speed up of the GPU implementation against OpenMP implementation with 6 threads when the fire front is split in 184,000 perimeter points. We saw that, when we simulated propagation time below two hours, the execution in the CPU is faster when 6 threads are used. Nonetheless, above this propagation time, the execution of the GPU implementation is the fastest one.



(a) Number maximum of points per second depending on the propagation time.



(b) Number of points from which the GPU is faster than the CPU.

Fig. 3. Execution performance of the different Forest Fire Spread implementations.

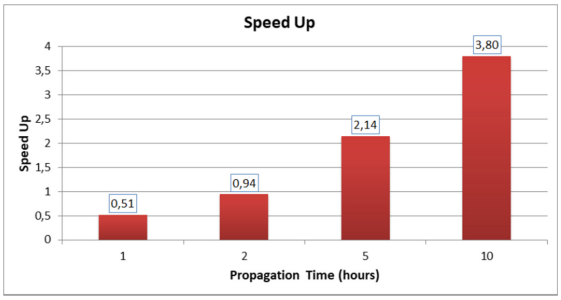


Fig. 4. Speed Up of the GPU implementation versus OpenMP implementation with 6 threads when 184,000 perimeter points are used.

5 Conclusions and Future Work

The computational capabilities of GPUs make them ideal for the simulation of any complex system. In our case, we have focused on the study of forest fire propagation simulators based on the Elliptical Wave Propagation, in particular, FARSITE. In this work, a synthetic fire has been used, with constant wind and vegetation conditions throughout the simulation. The obtained results demonstrated that the use of the GPU open a new way of approaching forest fire spread simulation in the sense that we expect to get more accurate results and, at the same time, faster and, therefore operationally simulation time.

According to the study carried out, for long fire propagation simulations, the GPU implementation is more efficient than the OpenMP implementation. Moreover, GPU is better than CPU when we face compute-bound. We also highlighted that the number of propagated points per second by the GPU is much higher than the number of spread points per second in the CPU in all cases, so the efficiency of the GPU is higher than the CPU.

Future work will be oriented to increase the efficiency of the GPU implementation and use real fires to determine under in which conditions it is better to do the propagation of the fire front in the GPU.

Acknowledgments. This research has been supported by MINECO-Spain under contract TIN2017-84553-C2-1-R and by the Catalan government under grant 2017-SGR-313.

References

1. Artés, T., Cencerrado, A., Cortés, A., Margalef, T.: Core allocation policies on multicore platforms to accelerate forest fire spread predictions. In: Wyrzykowski, R., Dongarra, J., Karczewski, K., Waśniewski, J. (eds.) PPAM 2013. LNCS, vol. 8385, pp. 151–160. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-55195-6_14
2. Brun, C., Margalef, T., Cortés, A., Sikora, A.: Enhancing multi-model forest fire spread prediction by exploiting multi-core parallelism. J. Supercomput. **70**(2), 721–732 (2014). https://doi.org/10.1007/s11227-014-1168-z
3. Cencerrado, A., Artés, T., Cortés, A., Margalef, T.: Relieving uncertainty in forest fire spread prediction by exploiting multicore architectures. In: Proceedings of the International Conference on Computational Science, ICCS 2015, Computational Science at the Gates of Nature, Reykjavík, Iceland, pp. 1752–1761, 1–3 June 2015. https://doi.org/10.1016/j.procs.2015.05.380
4. Dagum, L., Menon, R.: OpenMP: an industry standard api for shared-memory programming. Comput. Sci. Eng. **5**(1), 46–55 (1998)
5. D'Ambrosio, D., Di Gregorio, S., Filippone, G., Rongo, R., Spataro, W., Trunfio, G.A.: A multi-GPU approach to fast wildfire hazard mapping. In: Obaidat, M.S., Filipe, J., Kacprzyk, J., Pina, N. (eds.) Simulation and Modeling Methodologies, Technologies and Applications. AISC, vol. 256, pp. 183–195. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-03581-9_13
6. Finney, M.A.: Farsite: Fire area simulator–model development and evaluation. FResearch Paper RMRS-RP-4 Revised 236, Research Paper RMRS-RP-4 Revised (1998)
7. Farsite tutorial website (2007). http://fire.org/downloads/farsite/WebHelp/using_farsite_help.htm
8. Gregorio, S.D., Filippone, G., Spataro, W., Trunfio, G.A.: Accelerating wildfire susceptibility mapping through GPGPU. J. Parallel Distrib. Comput. **73**(8), 1183–1194 (2013). https://doi.org/10.1016/j.jpdc.2013.03.014
9. Hoang, R.V.: Wildfire Simulation on the GPU. Ph.D. thesis, university of Nevada (2008)
10. Knight, I., Coleman, J.: A fire perimeter expansion algorithm-based on huygens wavelet propagation. Int. J. Wildland Fire **3**, 73–84 (1993)
11. Ntinas, V.G., Moutafis, B.E., Trunfio, G.A., Sirakoulis, G.C.: Parallel fuzzy cellular automata for data-driven simulation of wildfire spreading. J. Comput. Sci. **21**, 469–485 (2017). https://doi.org/10.1016/j.jocs.2016.08.003
12. Rothermel, R.: A mathematical model for predicting fire spread in wildland fuels. Technical Report INT-GTR-115. (Ogden, UT) (1972)
13. Sousa, F.A., dos Reis, R.J.N., Pereira, J.C.F.: Simulation of surface fire fronts using fireLib and GPUs. Environ. Model. Softw. **38**, 167–177 (2012). https://doi.org/10.1016/j.envsoft.2012.06.006