A Multi-GPU Approach to Fast Wildfire Hazard Mapping

Donato D'Ambrosio[1], Salvatore Di Gregorio[1], Giuseppe Filippone[1], Rocco Rongo[1], William Spataro[1], and Giuseppe A. Trunfio[2]

[1] Department of Mathematics and Computer Science, University of Calabria, 87036 Rende (CS), Italy
[2] DADU, University of Sassari, 07041 Alghero (SS), Italy

Abstract. Burn probability maps (BPMs) are among the most effective tools to support strategic wildfire and fuels management. In such maps, an estimate of the probability to be burned by a wildfire is assigned to each point of a raster landscape. A typical approach to build BPMs is based on the explicit propagation of thousands of fires using accurate simulation models. However, given the high number of required simulations, for a large area such a processing usually requires high performance computing. In this paper, we propose a multi-GPU approach for accelerating the process of BPM building. The paper illustrates some alternative implementation strategies and discusses the achieved speedups on a real landscape.

Keywords: GPGPU, Cellular Automata, Wildfire Simulation, Wildfire Susceptibility, Hazard Maps.

1 Introduction

Among the several tools recently developed to support fire hazard management, there are the so-called *burn probability maps* (BPMs), which attempt to provide an estimate of the probability of a point in a landscape to be burned under certain environmental conditions. To cope with the nonlinear interactions between the many factors that determine the fire behaviour, models for simulating wildfire spread are increasingly being used to build BPMs [1, 2]. In particular, the typical approach is based on carrying out a high number of simulations (e.g. many thousands), under different weather scenarios and ignition locations [1].

In order to obtain reliable results in reasonable time, such an approach must be based on fast and accurate simulation models operating on high-quality high-resolution remote sensing data (e.g., Digital Elevation Models, vegetation description, etc). Among the different wildfire simulation techniques [3], those based on Cellular Automata (CA) [4–7] represent an ideal approach to build a BPM. This is because they provide accurate results and can often perform the same simulations in a fraction of the run time taken by different methods [6].

However, because of the required high number of explicit fire propagations, even using the most optimized algorithms, the building of simulation-based BPMs for a large area often results in a highly intensive computational process. For example, building a high-resolution BPM for a regional territory typically requires the use of high performance computing (HPC).

Among the different HPC alternatives is the recently emerging General-Purpose computing on Graphics Processing Units (GPGPU), in which multicore Graphics Processing Units (GPU) perform computations traditionally carried out by the CPU. In this paper, we apply GPGPU, in conjunction with a wildfire simulation model, to the process of BPM computation.

In particular, the proposed approach is based on a CA simulation model which represents a suitable trade-off between accuracy and speed of execution. The adopted parallel computation consists of the iterative simultaneous simulation of a number of wildfires with GPGPU, in order to cover the whole area under study. In the paper we illustrate two different implementation strategies together with a multi-GPU approach. In addition, we discuss some numerical results obtained on a real Mediterranean landscape, which is historically characterized by a high incidence of wildfires.

The paper is organized as follows. In the next section we outline the main characteristics of the adopted CA simulation model and illustrate some details of the typical approach for BPM computation. Then, in section 3 we present some introductory elements of the adopted GPGPU approach. In section 4 we outline the proposed parallel approaches and in section 5 we investigate some of their computational characteristics. The paper ends with section 6 in which we draw some conclusions and outline possible future work.

2 Simulation-Based Burn Probability Mapping

2.1 The Wildfire Simulation Model

As mentioned above, CA methods for simulating wildfires can be highly optimized from the computational point of view. For this reason they are well suited for the process of building BPMs. In particular, in this study we use the CA-based wildfire simulation model described in [8] (to which the reader is referred for the details).

The adopted simulator is based on the Rothermel fire model [9], which provides the heading rate and direction of spread given the local landscape and wind characteristics. An additional constituent is the commonly assumed elliptical description of the spread under homogeneous conditions (i.e. spatially and temporally constant fuels, wind and topography) [10]. Under the above hypothesis, given the assumption of homogeneity at the cell level, the CA transition function uses the elliptical model for producing the complex patterns that correspond to the fire spread in heterogeneous conditions.

In brief, the two-dimensional fire propagation is locally obtained by a growing ellipse (see Figure 1) having the semi-major axis along the direction of maximum spread, the eccentricity related to the intensity of the so-called *effective wind* and one focus acting as a 'fire source' [5, 6]. At each CA step the ellipse's size is increased according to both the duration of the time step and maximum rate of spread. Afterwards, a neighbouring cell invaded by the growing ellipse is considered as a candidate to be ignited by the spreading fire. In case of ignition, a new ellipse is generated according to the amount of overlapping between the invading ellipse and the ignited cell.

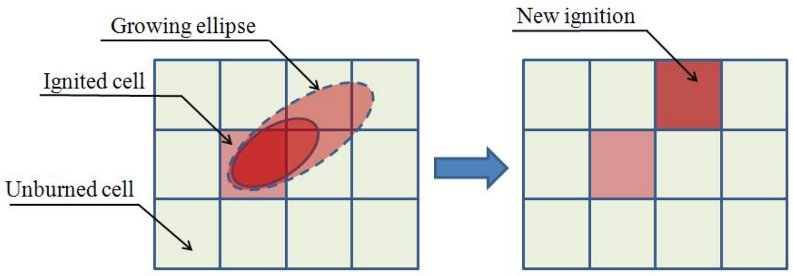

Fig. 1. Growth of the ellipse locally representing the fire front. The forward spread is incremented at each CA step according to the local spread rate and to the size of the time step.

A relevant feature of the model consists of the dynamic adaptation of the duration of each time step. In practice, the step size is computed on the basis of the minimum amount of time that elapses before the fire may have traveled from a cell on the current fire front to a neighbouring cell [4, 6, 7].

2.2 A Simulation-Based Approach for Building BPMs

In the latest years, the use of hazard maps based on the explicit simulation of natural phenomena has been increasingly investigated as an effective and reliable tool for supporting risk management [11, 1, 2, 12].

In the case of wildfire, the most general approach for computing a BPM on a landscape [1, 2] consists of a Monte Carlo approach in which a high number of different fire spread simulations are carried out, sampling from suitable statistical distributions the random variables relevant to the fire behaviour. For example, the wind direction for each simulated fire can be sampled in a range corresponding to the typical directions of severe wind for the area. At the end of the process, the local risk is computed on the basis of the frequency of burning.

The technique for computing the BPMs adopted in this study is based on a prefixed number n_f of simulation runs, where each run represents a single simulated fire. The adopted weather scenario (i.e. wind and fuel moisture content) is stationary and corresponds to extreme conditions for the area with regards to relevant historical fires. A regular grid of ignition locations is adopted, which corresponds to the assumption of a uniform ignition probability for each point of the landscape. Also, all the simulated fires have the same duration. The latter is selected considering the duration of historical fires in the regions under study. All the other relevant characteristics are kept constant during the simulations.

Once these have been carried out, the resulting n_f maps of burned areas are overlaid and cells' fire frequency are used for the computation of the fire risk. In particular, a *burn probability* $p_b(\mathbf{c})$ for each cell $\mathbf{c}$ is computed as:

$$p_b(\mathbf{c}) = \frac{f(\mathbf{c})}{n_f}; \qquad (1)$$

where $f(\mathbf{c})$ is the number of times the cell $\mathbf{c}$ is ignited during the n_f simulated fires. The burn probability for a given cell is an estimate of the likelihood that a cell will burn given a single random ignition within the study area and given the assumed conditions in terms of fire duration fuel moisture and weather.

According to the procedure described above, the number n_f of simulation runs depend on the resolution of the grid of ignition points. However, as shown in the application example discussed later, the number of fire simulation needed for achieving a good BPM accuracy can be considerably high in case of study areas with great extensions.

3 The Compute Unified Device Architecture

A natural approach to deal with the high computational effort related to construction of the BPMs is the use of parallel computing. Among the different parallel architectures and computational paradigms, the recently emerging GPGPU is particularly suited for accelerating CA-based simulations [13, 14].

Modern GPUs are multiprocessors with a highly efficient hardware-coded multithreading support. The key capability of a GPU unit is thus to execute thousands of threads running the same function concurrently on different data. Hence, the computational power provided by such an architecture can be fully exploited through a fine grained data-parallel approach when the same computation can be independently carried out on different elements of a dataset.

The GPGPU platform investigated in this paper is the one provided by nVidia, which consist of a group of Streaming Multiprocessors (SMs) able to support a limited number of co-resident concurrent threads, which share the SM's limited memory resources. Furthermore, each SM consists of multiple Scalar Processor (SP) cores. In order to program the GPU, the C-language Compute Unified Device Architecture (CUDA) [15] is used. In a typical CUDA program, sequential host instructions are combined with parallel GPU code. The idea underlying this approach is that the CPU organizes the computation (e.g. in terms of data pre-processing), sends the data from the computer main memory to the GPU global memory and invokes the parallel computation on the GPU. After, and/or during the latter, the CPU invokes the copying of the computed results into the main memory for post-processing and output purposes.

In CUDA, the GPU activation is obtained by writing device functions in C language, which are called *kernels*. When a kernel is invoked by the CPU, a number of threads (e.g. typically several thousands) execute their code in parallel on different data. According to the nVidia approach to GPGPU, threads are grouped into blocks and executed on the SMs.

The GPU can access different types of memory. For example, to each thread block can be assigned a certain amount of fast shared memory (which can be used for some limited intra-block communication between threads). Also, all threads can access a slower but larger global memory which is on the device board but outside the computing chip. The device global memory is slower if compared with the shared memory but it can deliver significantly higher bandwidth than the main computer memory. The latter is typically linked to the GPU card through a relatively slow bus. As a results, the parallel computation should be organised in such a way to minimize data transfers between the host and the device.

Even though the GPUs global memory offers a considerably higher memory bandwidth compared to CPUs host memory, it still requires hundreds of clock cycles to start the fetch of a single element. However, the massively threaded architecture of the GPU is used to hide such memory latencies by rapid switching between threads: when a thread stalls on a memory fetch, the GPU switches to a different thread. To fully exploit such latency hiding strategy, it is usually beneficial to use a large number of blocks.

In the following, the concepts outlined above are applied to the GPGPU-based parallelization of the procedure for building BPMs previously described.

4 GPGPU-Based Wildfire Hazard Mapping

The procedure described in the following is based on accelerating through GPGPU the execution of the many CA simulations required by the BPM. As in the sequential version, the CA simulation model requires two memory regions, which will be called CA_{cur} and CA_{next}, representing the *current* and *next* states for the cells respectively. For each CA step, the neighbouring values from CA_{cur} are read by the local transition function, which performs its computation and writes the new state value into the appropriate element of CA_{next}.

More in particular, accordingly to the recent literature in the field, in our implementation most of the automaton data (i.e. both the current and next memory areas mentioned above) is stored in the GPU global memory. This involves: *(i)* initialising the current state through a CPU-GPU memory copy operation (i.e. from host to device global memory) before the beginning of the simulation and *(ii)* retrieving the final state of the automaton at the end of the simulation through a GPU-CPU copy operation (i.e. from device global memory to host memory). Also, at the end of each CA step a device-to-device memory copy operation is used to re-initialise CA_{cur} with CA_{next}.

In order to speed up the access to memory, an array with the size corresponding to the total number of cells was allocated in the CPU memory for each of the CA substates. All of such arrays were then mirrored in the GPU together with some additional auxiliary arrays (e.g. for storing the neighbourhood structure and the model parameters).

A key step in the parallelization of a sequential code for the GPU architecture according to the CUDA approach, consists of identifying all the sets of instructions that can be grouped in CUDA kernels. In particular, in the CA model for wildfire simulation adopted in this study (see section 2), two main CUDA kernels have been developed:

- the kernel implementing the fire propagation logic (i.e. the cell-level mechanism of fire contagion);
- the kernel for dynamically adapting the time-step duration. Since this involves finding the minimum of all allowed time-step sizes among the cells on the current fire front, such kernel simply implements a standard parallel reduction (PR) algorithm.

The above kernels can be executed independently of each other on the different cells of the automaton. However, the GPGPU parallelization object of this study raises several issues.

First, in the whole automaton, only the cells belonging to the current fire front perform actual computation. Hence, launching one thread for each of the automaton cells would result in a certain amount of dissipation of the GPU computational power. In other words, a high percentage of threads would be uselessly scheduled. In addition,

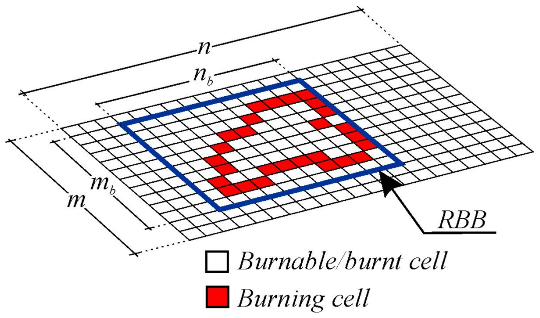

Fig. 2. The RBB of a fire in a $n \times m$ raster. Only the cells inside the RBB, which is recomputed at each CA step, are mapped into the CUDA grid of threads.

given the small size of most fires, the number of active threads generated by the simulation of a fire is usually too low to allow the GPU to effectively activate the latency hiding mechanism.

For the above reasons, two main strategies were adopted. In particular, besides the most straightforward parallel implementation, labelled as *WCAM*, in which the CUDA kernels operate on the whole automaton, we have developed an additional implementation in which the grid of threads is dynamically computed during the simulation in order to keep low the number of computationally irrelevant threads.

In such an approach, labelled as *RBBM* and represented in Figure 2, the smallest rectangular bounding box (RBB) that includes any cells on the current fire front is computed at each CA step using the efficient CUDA atomic instructions. Then, all kernels required by the CA step (i.e. the PR for the time step adaptation and the transition function) are mapped on such RBB.

Another measure adopted in this study consists of handling many fires simultaneously. In particular, clusters of fires are simulated up to covering the entire area under study (i.e. until all the required n_f fires are propagated). Each cluster is composed of a block of fires originated by spatially-contiguous ignition points taken from the regular grid of fire ignitions.

Obviously, such an approach requires: (*i*) the use of an additional array for storing an independent combustion state of each cell for each simultaneous fire; (*ii*) the computation of a RBB that is common to all the simultaneous fires.

It is worth noting that since the efficiency of the *RBBM* approach depends on the actual distribution of the simultaneous fires over the automaton, we choose from contiguous ignition points the fires that are simulated together.

In order to carry out many simulations simultaneously, a particular grid of threads is used is such a way that for each cell a separate thread handles the different simultaneous fires (see Figure 3). This corresponds to a three-dimensional grid in which the vertical dimension is associated with the simultaneous fires. It is also worth noting that the alternative approach, in which a single thread per cell iterates on the different simultaneous fires, would generate a frequent divergence between the threads of the same warp, with a consequent decline of efficiency.

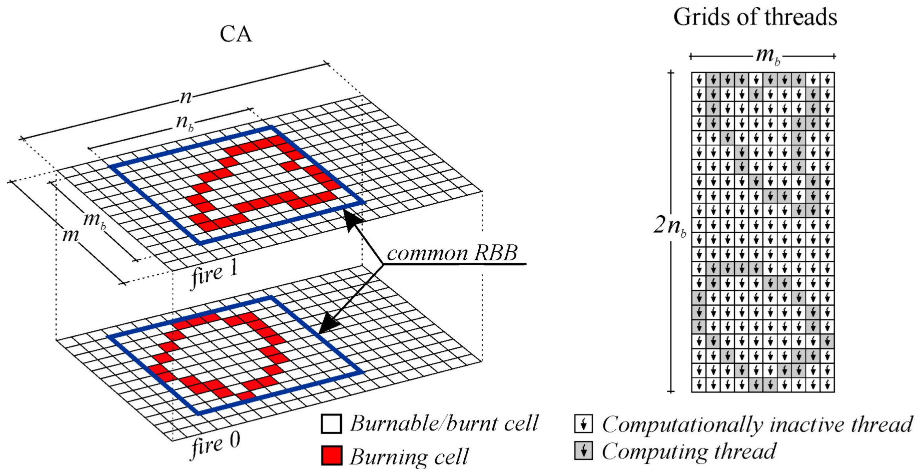

Fig. 3. The *RBBM* mapping of the CA transition function into a CUDA grid of threads in case of the two simultaneous fires shown on the left. The kernel is mapped on a grid in which a separate copy of the RBB is associated with each of the different simultaneous fires. This corresponds to a three-dimensional grid, with the base represented by the RBB and the vertical dimension corresponding to the fires.

Before starting the BPM construction, a pre-processing sequential phase takes place in which:

– for each cell the maximum rate of spread, its direction and the local ellipse eccentricity (see Figure 1) are computed using the proper model equations [9, 16]. Such pre-computed quantities determine, together with the landscape topography, the wildfire spread at the cell-level.
– the maximum time-step size for each cell is computed and stored in an array in order to speed-up the time-step adaptation during the CA iterations.
– a data structure $\mathcal{C}$ is built, which contains all the clusters of contiguous ignition points corresponding to fires that must be simulated together (see Figure 4).

It is worth noting that the first two steps above make sense because the weather conditions are considered stationary, which is a common assumption for computing BPMs.

As explained above, the required CA simulations are carried out operating on clusters of fires iteratively extracted from the data structure $\mathcal{C}$. In particular, in the GPGPU wildfire simulation, each CA step essentially consists of:

– executing the CUDA kernel that finds the current time step size for each simultaneous fire;
– executing the CUDA kernel that implements the propagation mechanism (and updates the RBB in the RBBM approach);
– activating a device-to-device memory copy operation to re-initialise CA_{cur} with CA_{next};
– incrementing the current time for each simultaneous fire by the corresponding time step size.

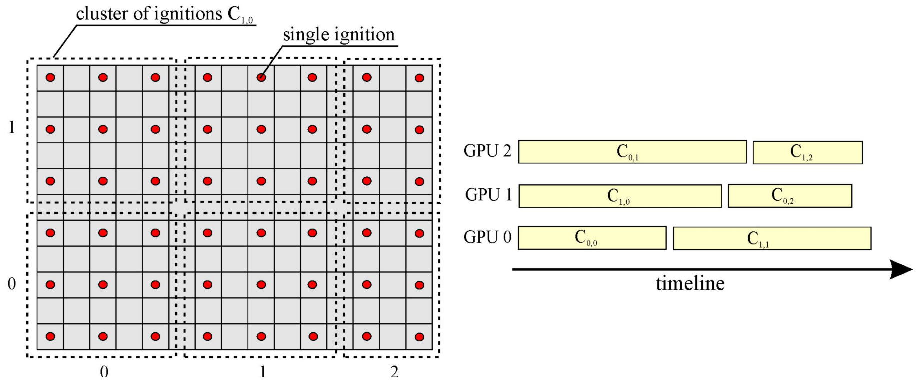

Fig. 4. An hypothetical example of regular grid of ignition points used for building a BPM. Contiguous ignitions are grouped into clusters and the corresponding fires are simulated simultaneously. In the multi-GPU implementation each cluster is assigned to the first available GPU.

The above steps are iterated until the current time reaches the desired final time for each fire of the current cluster.

4.1 The Multi-GPU Implementation

In the multi-GPU implementation, which is based on the *Pthreads* library, to each available GPU in the system we permanently associate a separate CPU thread using the specific CUDA function *cudaSetDevice*. In addition, a separate copy of the automaton is mirrored in each GPU.

After such initialization phase, the process for the BPM computation is concurrently carried out by each CPU thread as follows:

1. extraction, using the mutual exclusion mechanism provided by *PThreads*, of the first available cluster C of fire ignitions from the data structure $\mathcal{C}$ (see Figure 4) mentioned above.
2. execution of a CUDA kernel to set the ignition points included in its current cluster C into the automaton cells stored in its GPU global memory;
3. execution, in its associated GPU, of the CA steps described above, until the current time reaches the desired final time;
4. at the end of the simulations, execution of a further CUDA kernel on the whole automaton to update an array $\mathbf{f}_d$ in which each element represents the number of times that a cell has been burned since the beginning of the process;
5. if there are still elements in $\mathcal{C}$, extraction of a new cluster of ignitions and iteration of the above steps $1 - 4$; otherwise, the CPU thread ends.

When all the CPU threads have completed their cycles of simulations, all the arrays $\mathbf{f}_d$ are moved to the host memory and collected into a single array $\mathbf{f}_n$. The latter, divided by the total number of simulations n_f, gives an estimate of the burn probability for each cell of the automaton.

Given that the CPU threads are essentially independent on each others, the above procedure can achieve a satisfactory level of efficiency. However, a limiting factor of the scalability originates from the exchange of data between GPUs and CPU at each CA step (for example the duration of the steps or the RBB).

It is worth noting that, in a general case of heterogeneous landscape, the time required for the simulation of each cluster $C \in \mathcal{C}$ cannot be predicted in advance. In fact, it depends on the specific conditions of the area covered by C (e.g. fuel and terrain characteristics or local wind vector). However, according to the above scheduling procedure, as soon as a GPU finishes its current cluster computation, it fetches the next available cluster of fires. This can be seen in Figure 4, where an hypothetical example of scheduling is represented. Therefore, if $\mathcal{C}$ contains enough clusters (which is always the case for large areas) the workload of each GPU is automatically balanced. In other words the BPM computation runs flawlessly even in case of GPUs with different computational power.

5 Results on a Real Landscape

The application presented here concerns an area of the Ligury region, in Italy. The landscape, shown in Figure 5 was modelled through a Digital Elevation Model composed of 461×445 square cells with side of $40\,m$. In the area, the terrain is relatively complex with an altitude above sea level ranging from 0 to 250 m. The fuel bed, depicted in Figure 5, was based on the land cover map from the CORINE EU-project. In particular, the CORINE land-cover codes were mapped on the standard fuel models used by the CA. Plausible values of fuel moisture content were obtained from literature data. Also, a North direction open-wind vector, having an intensity of $20\,km\,h^{-1}$, was used for producing the wind field. A duration of 10 hours was adopted for all simulated fires. Over the area, a regular grid of 92×89 ignition points was superimposed, leading to 7391 fires to simulate and to the BPM shown in Figure 6.

For the numerical experiments, we used a workstation based on an Intel Xeon X5660 (2.80 GHz) 6-Core CPU and equipped with two GPUs, namely a nVidia Tesla C2075 and a nVidia Geforce GTX 680 graphic card. The latter card belongs to the new nVidia's Kepler GPU architecture while the former is endowed with the older Fermi GPUs. The Tesla C2075 was used in single precision floating point and with the ECC disabled. In Table 1 we report some of the relevant characteristics of the two GPU devices. To quantify the speedup of the parallel implementations, we also included in the same CUDA program the C-language sequential versions of the *WCAM* and *RBBM* approaches described above. In particular, the program allows the user to select the desired version of the algorithm, as well as whether to use the available GPUs or only the CPU.

It is worth noting that, in the sequential case only one fire at a time was propagated since the advantages of simulating multiple fires are not significant. Specifically, using only the CPU for simulating the 7391 independent fires required by the case-study BPM, the *WCAM* approach took $14167.4\,s$ while the *RBBM* strategy took $2082.6\,s$.

Using the adopted GPU devices in both single and multi-GPU mode, the BPM was built with the *WCAM* approach and a variable number of simultaneous fires. According to the results shown in Figure 7, the two GPUs working together achieved the lowest elapsed time of $197.8\,s$, simultaneously simulating 81 fires. As can be seen, the simultaneous simulation of many fires was beneficial since it allowed a significant computing

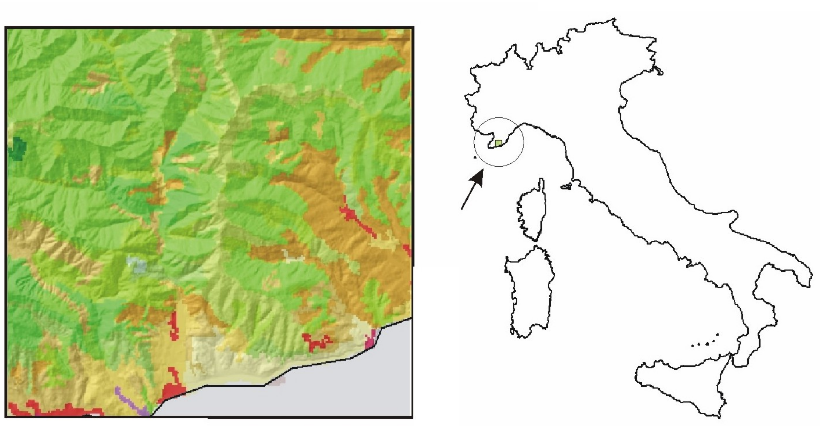

Fig. 5. The landscape under study: a $18km \times 18km$ area in Ligury, Italy. Colors refer to the standard CORINE land-cover data.

Table 1. Some relevant characteristics of the adopted GPGPU hardware for all carried out experiments. Note that GFLOPs refers to the theoretical peak performance in single-precision floating point operations.

	GTX 680	Tesla C2075
SM count	8	14
CUDA cores	1536	448
Clock rate [MHz]	1006	1150
Bandwidth [GB/s]	192.3	144.0
GFLOPs	3090.4	1030.4

time decrement. However, for all the GPU configurations, incrementing the size of the clusters in $\mathcal{C}$ above a certain threshold did not lead to significant advantages. This is due to the fact that, once the number of computationally-relevant threads is sufficient for enabling latency hiding, a further increase of simultaneous fires is not necessary.

Interestingly, using the two GPUs together lead to a speedup ranging between 1.82 and 1.90 over the faster GPU, namely the GTX 680. Particularly relevant is the speedup over the sequential run, which attained the value of 71.6.

Figure 7 also shows the times taken by the parallel *RBBM* approach as a function of the number of simultaneous fires. According to the graph, in this case the joint computational effort of the two GPUs gave the lowest elapsed time of $33.8\,s$, corresponding to a parallel speedup of 61.6.

In the *RBBM* runs, using the two GPUs together lead to a speedup over the faster GPU (i.e. the GTX 680) ranging between 1.72 (196 simultaneous fires) and 1.89 (16 simultaneous fires). Not surprisingly, given the multi-GPU scheduling scheme described in section 4.1, in case of GPUs with different computational power the use of small clusters of fires (i.e. the use of more clusters) gives the opportunity to better exploit the faster GPU.

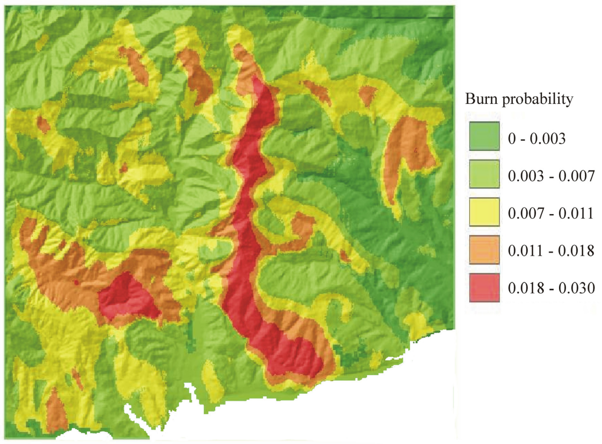

Fig. 6. The BPM obtained for the landscape under study

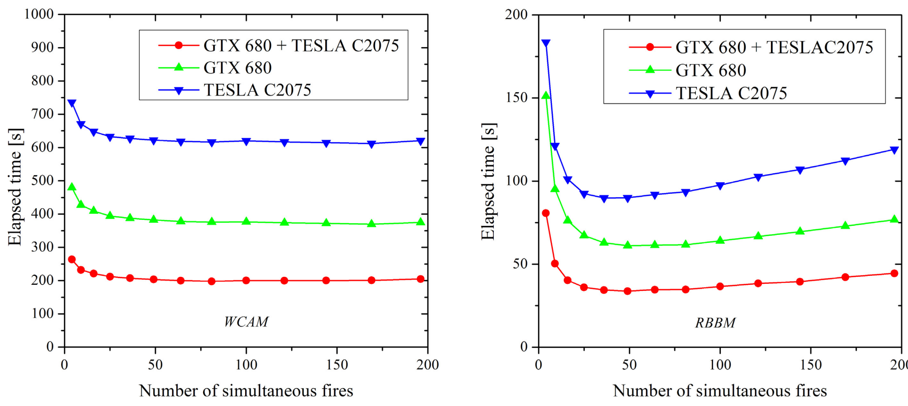

Fig. 7. Time taken by the *WCAM* and *RBBM* approaches for simulating the 7391 independent fires required by the BPM shown in Figure 6. The elapsed time is plotted as a function of the number of fires that are simultaneously simulated by the GPU.

As expected, the simultaneous simulation of many fires was much more beneficial than in the $WCAM$ case. The reason is that, having available a relatively limited number of simultaneous fires, the percentage of active cells is higher in a small RBB

Table 2. Best elapsed times (in seconds) for the computation of the BPM shown in Figure 6. In brackets is the optimum number of simultaneous fires.

	WCAM	RMBB
CPU	14167.4	2082.6
GTX 680	369.5 (169)	61.1 (49)
Tesla C2075	612.2 (169)	89.8 (36)
GTX 680 + Tesla C2075	197.8 (81)	33.8 (49)

Table 3. Best parallel speedup achieved with the used GPGPU devices

	WCAM	RMBB
GTX 680	38.3	34.1
Tesla C2075	23.1	23.2
GTX 680 + Tesla C2075	71.6	61.6

than in the entire automaton. However, also in this case it is not advantageous to simulate more than a certain number of simultaneous fires.

For clarity reasons, a summary of the elapsed times achieved for solving the test problem are shown in Table 2. Moreover, the corresponding parallel speedups are shown in Table 3. As can be seen, even using a single GPU lead to a significant acceleration of the BPM computation.

6 Conclusions and Future Work

The parallel speedups of the multi-GPU based BPM computation procedure presented in this paper are very satisfactory. The main advantage of such a parallelization lies in enabling the building of BPMs for very large areas (e.g. at a regional level), which otherwise may not be possible by adopting a standard computation. However, starting from the application described above, several research directions can be explored.

For example, the multi-GPU approach may also be adopted for the automatic planning of risk-mitigation interventions on the landscape (i.e. the so-called *fuel treatments*) [17]. Moreover, the fast simulation of a number of wildfires can make easier and more accurate the assimilation of real-time sensor data [18]. However, in this case a new approach with non-stationary weather conditions should be developed. This would require computing with the GPU also the fire characteristics at the cell level at the price of an increased computing time, though obtaining higher speedups.

Also, the GPGPU parallelization strategies investigated in this paper can be exploited, with the required adjustments, to other research and application areas. In fact, the algorithm for building BPMs is somewhat similar to those adopted for dealing with other natural hazards, such as the risk induced by debris or lava flows (e.g. [11, 19]).

Eventually, another possible direction of research consists of making the GPGPU approach available in more general libraries for supporting CA modelling and simulation, such as the one presented in [20].

References

1. Carmel, Y., Paz, S., Jahashan, F., Shoshany, M.: Assessing fire risk using Monte Carlo simulations of fire spread. Forest Ecology and Management 257(1), 370–377 (2009)
2. Ager, A., Finney, M.: Application of wildfire simulation models for risk analysis. In: Geophysical Research Abstracts. EGU2009-5489, EGU General Assembly, vol. 11 (2009)
3. Sullivan, A.: Wildland surface fire spread modelling, 1990-2007. 3: Simulation and mathematical analogue models. International Journal of Wildland Fire 18, 387–403 (2009)
4. Lopes, A.M.G., Cruz, M.G., Viegas, D.X.: Firestation - an integrated software system for the numerical simulation of fire spread on complex topography. Environmental Modelling and Software 17(3), 269–285 (2002)

5. Trunfio, G.A.: Predicting wildfire spreading through a hexagonal cellular automata model. In: Sloot, P.M.A., Chopard, B., Hoekstra, A.G. (eds.) ACRI 2004. LNCS, vol. 3305, pp. 385–394. Springer, Heidelberg (2004)
6. Peterson, S.H., Morais, M.E., Carlson, J.M., Dennison, P.E., Roberts, D.A., Moritz, M.A., Weise, D.R.: Using HFIRE for spatial modeling of fire in shrublands. Technical Report PSW-RP-259, U.S. Department of Agriculture, Forest Service, Pacific Southwest Research Station, Albany, CA (2009)
7. Trunfio, G.A., D'Ambrosio, D., Rongo, R., Spataro, W., Di Gregorio, S.: A new algorithm for simulating wildfire spread through cellular automata. ACM Transactions on Modeling and Computer Simulation 22(1), 1–26 (2011)
8. Avolio, M.V., Di Gregorio, S., Lupiano, V., Trunfio, G.A.: Simulation of wildfire spread using cellular automata with randomized local sources. In: Sirakoulis, G.C., Bandini, S. (eds.) ACRI 2012. LNCS, vol. 7495, pp. 279–288. Springer, Heidelberg (2012)
9. Rothermel, R.C.: A mathematical model for predicting fire spread in wildland fuels. Technical Report INT-115, U.S. Department of Agriculture, Forest Service, Intermountain Forest and Range Experiment Station, Ogden, UT (1972)
10. Alexander, M.: Estimating the length-to-breadth ratio of elliptical forest fire patterns. In: Proc. 8th Conf. Fire and Forest Meteorology, pp. 287–304 (1985)
11. Rongo, R., Spataro, W., D'Ambrosio, D., Avolio, M.V., Trunfio, G.A., Di Gregorio, S.: Lava flow hazard evaluation through cellular automata and genetic algorithms: an application to Mt Etna volcano. Fundamenta Informaticae 87(2), 247–267 (2008)
12. Rongo, R., Lupiano, V., Avolio, M.V., D'Ambrosio, D., Spataro, W., Trunfio, G.A.: Cellular automata simulation of lava flows - applications to civil defense and land use planning with a cellular automata based methodology. In: Proceedings of SIMULTECH 2011 (2011)
13. Filippone, G., Spataro, W., Spingola, G., D'Ambrosio, D., Rongo, R., Perna, G., Di Gregorio, S.: GPGPU programming and cellular automata: Implementation of the SCIARA lava flow simulation code. In: 23rd European Modeling and Simulation Symposium (EMSS), Rome, Italy, September 12-14 (2011)
14. D'Ambrosio, D., Filippone, G., Rongo, R., Spataro, W., Trunfio, G.: Cellular automata and GPGPU: an application to lava flow modeling. International Journal of Grid and High Performance Computing 4(3), 30–47 (2012)
15. CUDA C Programming Guide: v. 3.2 (2010)
16. Anderson, H.: Predicting wind-driven wildland fire size and shape. Technical Report INT-305, U.S Department of Agriculture, Forest Service (1983)
17. Ager, A.A., Vaillant, N.M., Finney, M.A.: A comparison of landscape fuel treatment strategies to mitigate wildland fire risk in the urban interface and preserve old forest structure. Forest Ecology and Management 259(8), 1556–1570 (2010)
18. Xue, H., Gu, F., Hu, X.: Data assimilation using sequential monte carlo methods in wildfire spread simulation. ACM Trans. Model. Comput. Simul. 22(4), 1–25 (2012)
19. Crisci, G.M., Avolio, M.V., Behncke, B., D'Ambrosio, D., Di Gregorio, S., Lupiano, V., Neri, M., Rongo, R., Spataro, W.: Predicting the impact of lava flows at Mount Etna, Italy. Journal of Geophysical Research: Solid Earth 115(B4) (2010)
20. Blecic, I., Cecchini, A., Trunfio, G.A.: A general-purpose geosimulation infrastructure for spatial decision support. Transactions on Computational Science 6, 200–218 (2009)