Wind field parallelization based on Python multiprocessing to reduce forest fire propagation prediction uncertainty

G. Sanjuan, T. Margalef and A. Cortés

Computer Architecture and Operating Systems Department
Universitat Autònoma de Barcelona, Spain.
{gemma.sanjuan, tomas.margalef, ana.cortes}@uab.cat

Abstract. Forest fires provoke significant loses from the ecological, social and economical point of view. Furthermore, the climate emergency will also increase the occurrence of such disasters. In this context, forest fire propagation prediction is a key tool to fight against these natural hazards efficiently and mitigate the damages. However, forest fire spread simulators require a set of input parameters that, in many cases, cannot be measured and must be estimated indirectly introducing uncertainty in forest fire propagation predictions. One of such parameters is the wind. It is possible to measure wind using meteorological stations and it is also possible to predict wind using meteorological models such as WRF. However, wind components are highly affected by the terrain topography introducing a large degree of uncertainty in forest fire spread predictions. Therefore, it is necessary to introduce wind field models that estimate wind speed and direction at very high resolution to reduce such uncertainty. Such models are time consuming models that are usually executed under strict time constrains. So, it is critical to minimize the execution time, taking into account the fact that in many cases it is not possible to execute the model on a supercomputer, but must be executed on commodity hardware available on the field or at control centers. This work introduces a new parallelization approach for wind field calculation based on Python multiprocessing to accelerate wind field evaluation. The results show that the new approach reduces execution time using a single personal computer.

Keywords: Wind field parallelization, Forest Fire Spread Simulation, Python multiprocessing

1 Introduction

Forest fire propagation prediction is a key tool to fight against such disasters. Several simulators [6][2] have been developed to provide hints on fire evolution to guide the field means and control centres to fight against these events. Most of these simulators are based on Rothermel's model [10], which is a semi-empirical model that takes into account the terrain topography, the vegetation conditions and the meteorological conditions to provide forest fire expected propagation. So, this model actually requires a set of parameters that, in many cases, are

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50436-6_41

not single values, but they have values over all the terrain where the fire is going on. For example, the type of vegetation changes over the terrain or the moisture contents of the same kind of vegetation can change over the terrain according to sun exposition. So, for these parameters, a field of values at high resolution is required to calculate the fire propagation. A very particular case is the wind. This parameter actually has two components: speed and direction, and it presents several features that makes it a very particular case:

- Wind speed and wind direction are, jointly with slope, the parameters that most significantly affect fire propagation [1]. Therefore, an accurate estimation of such values is critical for forest fire propagation prediction.
- The meteorological conditions change quickly and, some meteorological model, such as WRF [13], is required to estimate beforehand the values of the meteorological variables and, in particular, wind speed and direction at a surface level.
- The meteorological wind at surface level that can be measured on meteorological stations or estimated by meteorological models is affected by terrain topography, so that at each point of the terrain the wind values (speed and direction) could be different. This spatially varying wind values for a given area constitutes the so called wind field. To obtain this wind field for the underlying terrain, one should apply a diagnostic wind field model such as WindNinja [7]. WindNinja is a wind field model widely used in the forest fire simulation community, that can provide wind speed and wind direction, given a certain meteorological wind values at a surface level, in very high resolution, typically 30 meters.

In this context, for each forest fire propagation simulation step, it is necessary to evaluate several values for the meteorological wind at surface level and, for each one of these meteorological winds, it is necessary to calculate the corresponding wind field. Then, once those wind fields are obtained, they must be introduced to the forest fire propagation model to obtain the forest fire propagation prediction [4]. This scheme is represented in figure 1. It must also be considered that the spatial resolution of the involved models, particularly wind field model, must be very high. So, the whole coupled system involves several components that must solve complex systems of equations at a very high resolution, what implies large computing requirements and long computation times. These tight needs are critical in real time emergency situations where the response time is a key factor for efficient and effective actuation. In most cases a trade off between accuracy and time must be reached to provide useful predictions in operational time. Therefore, applying high performance techniques to accelerate model execution and reduce prediction time, also contributes to increase the map resolution, reducing the uncertainty and providing more reliable predictions. So, all the efforts to accelerate the involved models have a direct impact in improving the quality and effectiveness of forest fire propagation prediction.

Several efforts have been devoted to reduce the execution time of the models involved in the fore fire spread prediction process and also to improve the accu-

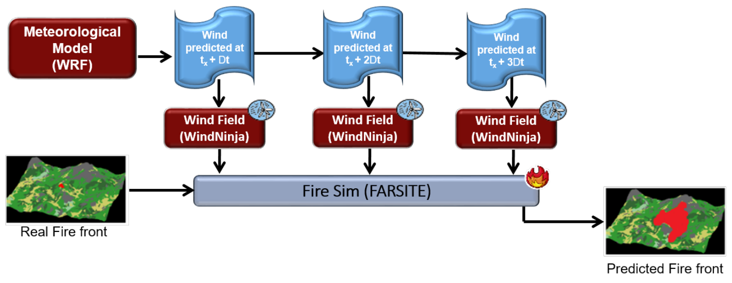


Fig. 1. Coupling Meteorological, Wind Field and Forest Fire Propagation models

racy of the results delivered by those model [8][12][11][3][5]. In particular, this work is focused on the wind field model. As it has been previously mentioned, in this work the WindNinja wind field model is used. More precisely, a new parallelization approach based on Python multiprocessing has been done. This new parallel approach has been compared to previous parallelization schemes based on MPI (Message Passing Interface). Since one of the goals of this work is being able to developed a forest fire spread prediction system that could be brought closer to the field where the firefighter's command is taken operational decisions, as execution platform one has selected commodity hardware. The parallelization scheme proposed is based on a map partitioning strategy that has been proven to work well for this kind of problem [11] .

The rest of this paper is organised as follows. Section 2 is devoted to describe WindNinja wind field model. In section 3 different WindNinja parallel implementations are introduced. Section 4 shows some preliminary results comparing the different parallel implementations described in the previous section and, finally section 5 presents the main conclusions.

2 WindNinja wind field simulator

As it has been mentioned above, wind speed and wind direction are critical parameters to determine forest fire propagation. In particular, meteorological wind at surface level is modified by terrain topography, so that there is a spatial distribution of wind values along the terrain map. This wind field must be determined to effectively predict forest fire propagation because there exist several high resolution wind phenomena such as, for example, wind speedup over ridges or flow channeling in valleys, that cannot be forecast otherwise. In this context, WindNinja [7] is a wind field simulator that takes the meteorological wind at a surface level and the terrain topography to determine wind speed and wind direction at each point of the underlying map grid at a given resolution, usually around 30 meters. WindNinja is based on mass conservation equations that are used to generate the system of equations. In order to solve this system, the Conjugate Gradient method with Preconditioner is used (PCG). PCG is an iterative method that can only be applied when the sparse matrix representing the system is symmetric, positive define and real. It uses a matrix M as a preconditioner, which determines the convergence of the system. The native solver implementation of WindNinja includes $SSOR$ and $Jacobi$ as preconditioners. The $SSOR$ preconditioner is used by default. Furthermore, WindNinja includes an OpenMP parallelization, so that the PCG can exploit the parallelism by using the available cores in the system nodes.

WindNinja can be divided into five basic blocs, as is shown in figure 2, where each bloc corresponds to one particular phase of the wind field generation process. The functionality of each one of these five phases are subsequently described:

1. Discretization of the terrain map into a mesh.
2. Application of the mass conservation equations to each point of the mesh to generate the system of equations represented as $Ax = b$.
3. Generation of the CRS (Compressed Row Storage) format to store the sparse matrix A.
4. Application of the Preconditioned Conjugate Gradient (PCG) method to solve the system of equations [9].
5. Construction of the resulting wind field.

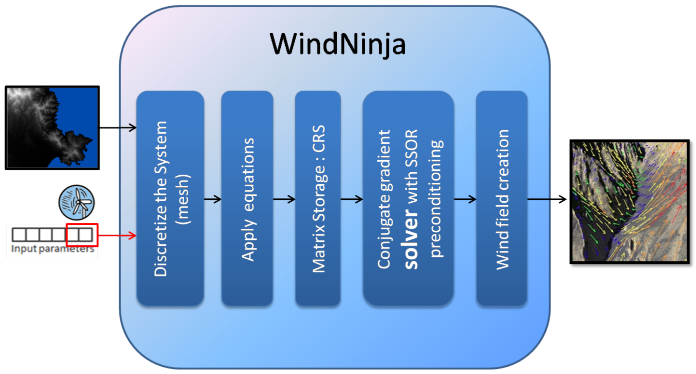

WindNinja
Input parameters
Discretize the System (mesh)
Apply equations
Matrix Storage : CRS
Conjugate gradient solver with SSOR preconditioning
Wind field creation

Fig. 2. WindNinja System

In the next section, the description of three parallel implementations of one single WindNinja execution are introduced. As it is subsequently explained, the proposed paralellization will not affect the way WindNinja works because they will not imply any change on WindNinja's code. In fact, the proposed parallel approaches could also benefit from the OpenMP parallelization included in WindNinja.

3 WindNinja parallelization

In a previous work [11], a map partitioning strategy was developed to divide the underlying terrain into equal size regions, with the aim of evaluating the complete wind field as a composition of a set of smaller wind fields coming from those previous regions. In particular, the map partitioning strategy divides the map in square parts introducing an overlapping halo on each one to avoid border effects. The optimal overlapping size has been proven to be 25 cells of the original mesh. Figure 3 illustrates this map partitioning scheme for two different map division, $2x2$ and $4x4$ map partitioning configurations.

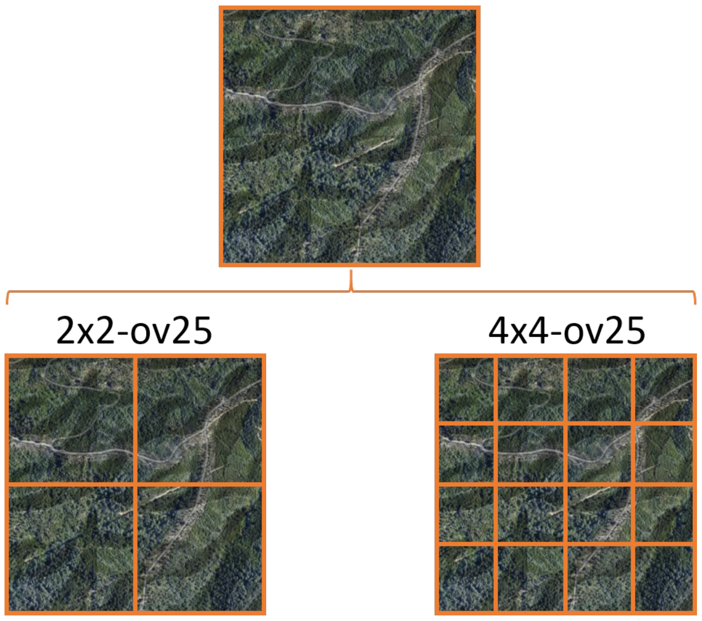
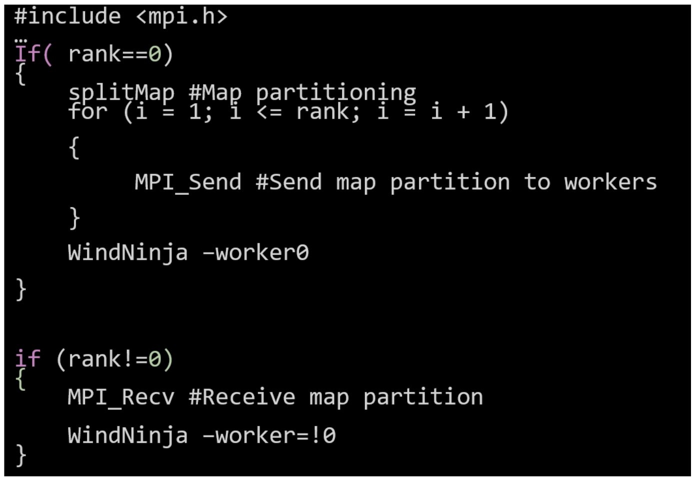


Fig. 3. Map partitioning

Once the map has been divided in partitions, the wind field corresponding to each partition can be calculated in parallel generating as many wind field maps as divisions of the map we have. Once all these computation has been done, a joining process is done in order to merge those wind field maps into one single wind field map. The main advantage of this approach is that the individual wind field map for a given region could be evaluated independently of the others, therefore, a straight forward parallelization scheme consists of executing each wind field calculation in a different computing element using the so called Master/Worker parallel programming paradigm. In the following subsections, we present three different parallel implementation of this strategy: MPI for C++, MPI for Python, and Python Multiprocessing.

3.1 MPI C++ parallelization

An MPI C++ application was developed to distribute the partitions of the map among the nodes of the system to execute them independently. As it has been mentioned, for each partition one WindNinja execution has been performed using the corresponding region of the map of that partition. The size of the overlapped halo has set to 25 cells what has been proven to be enough to avoid border effects. In this implementation a set of processes is created when launching the application with MPIRUN and each process calculates the wind field for each partition of the terrain map. Moreover, the OpenMP parallelization integrated in WindNinja has also been used. Actually this implementation is more feasible on a cluster with several nodes, but nowadays desktop computers or even laptops can run this kind of hybrid application efficiently. The resulting implementation is shown in Figure 4.

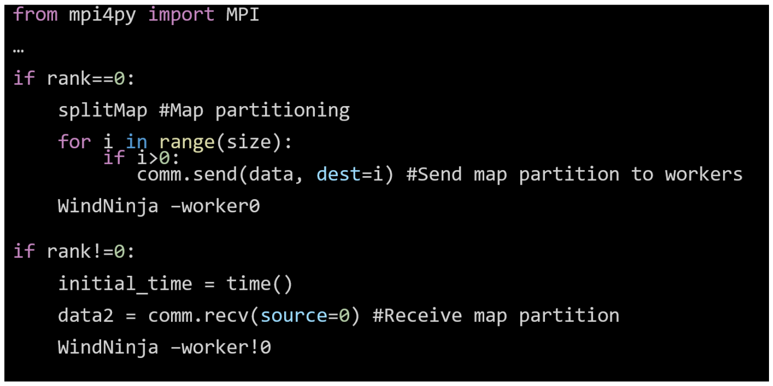


```
#include <mpi.h>
…
if( rank==0)
{
    splitMap #Map partitioning
    for (i = 1; i <= rank; i = i + 1)
    {
         MPI_Send #Send map partition to workers
    }
    WindNinja -worker0
}

if (rank!=0)
{
    MPI_Recv #Receive map partition
    WindNinja -worker=!0
}
```

Fig. 4. MPI C++ parallelization

3.2 MPI Python parallelization

Currently there is a clear trend to extend the use of Python programming language on many different areas. So, many libraries and current applications have been adapted to be used on Python programs. One particular case is MPI. MPI for Python provides MPI bindings for the Python language, allowing programmers to exploit multiple processor computing systems. mpi4py is constructed on top of the MPI-1/2 specifications and provides an object oriented interface which closely follows MPI-2 C++ bindings. So, the same map partitioning approach has been implemented using this MPI binding. The implementation can be represented as shown in Figure 5.



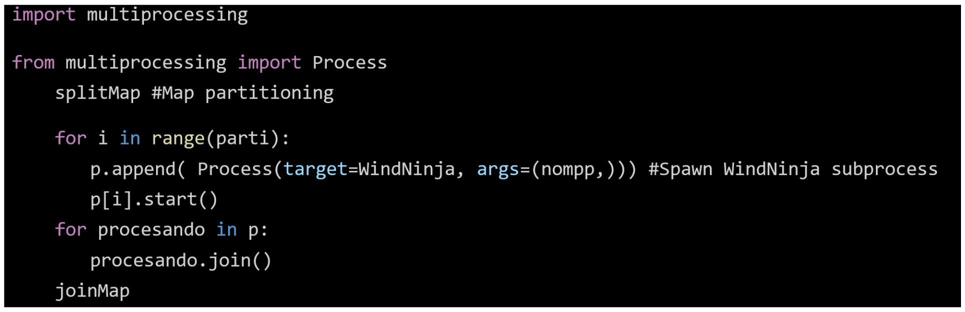
```
from mpi4py import MPI
…
if rank==0:
    splitMap #Map partitioning
    for i in range(size):
        if i>0:
           comm.send(data, dest=i) #Send map partition to workers
    WindNinja -worker0

if rank!=0:
    initial_time = time()
    data2 = comm.recv(source=0) #Receive map partition
    WindNinja -worker!0
```

Fig. 5. MPI Python parallelization

This approach is very similar to the previous one, but it is using Python as programming language and the MPI binding. The terrain map is split into a set of partitions and the wind field for each partition is calculated by each worker process. In this case, it is also possible to exploit the OpenMP WindNinja parallelization to calculate the wind field for each partition.

3.3 Python multiprocessing parallelization

Multiprocessing is a package that supports spawning processes using an API similar to the threading module. It allows the programmer to fully leverage multiple processes on a given machine. The implementation can be represented as shown in Figure 6.



```
import multiprocessing

from multiprocessing import Process
    splitMap #Map partitioning

    for i in range(parti):
       p.append( Process(target=WindNinja, args=(nompp,))) #Spawn WindNinja subprocess
       p[i].start()
    for procesando in p:
       procesando.join()
    joinMap
```

Fig. 6. Python Multiprocessing parallelization

In this case, no MPI is used, but the Python package to manage processes is used directly. The same master-worker parallel programming paradigm that has been used before, is also used in this approach. In this case, it is also possible to exploit the OpenMP WindNinja parallelization to calculate the wind field for each partition.

4 Experimental results

As it has been stated in the introduction of this paper, one of the objectives that we faced up when doing this work, was to be able to deploy a WindNinja parallel version that could be executed on the field during and on going event. That is, we are interested in having prediction systems that could be used in an operational fashion, instead of designing very complex forecast system that requires access to high cost computational resources either in economic terms but also in real time connectivity access possibilities. Typically, forest fires that occur in complex terrains with difficult access to the burn area, are the ones that clearly require the evaluation of the underlying wind field at high resolution because the shape of the landscape directly affects wind speed and wind direction at different points of the terrain. Since in these situations the connectivity of the computing systems in not always guaranteed, commodity hardware that could be available on the field, such a single laptop, could became an extremely useful tool. However, one cannot ignore that time constrains and accuracy of the results could not be dismissed. Thus, the experimental results outlined in this section have been obtained using a commodity hardware, in particular a single laptop based on an Intel i7-7700 processor that has 4 cores with hyperthreading, 8MB of cache memory and works at a base frequency of 3.60GHz. The system has 16 GB-DDR4 RAM.

As a test cases, we have used two map sizes having $775x775$ and $1500x1500$ cells each. The resolution in both cases has been set to 30 meters and different map partitionings have been also selected. Windninja 3.5.3 version has been used for all the experiments reported.

The first experiment that has been conducted corresponds to executing WindNinja in its native version with SSOR preconditioner and using its OpenMP implementation considering 1, 2, 4 and 8 threads. Tables 1 and 2 shows, respectively, the execution time and the speed up obtained when evaluating the windfield of the $775x774$ cells map. This experiment has been done as a basic parallel reference using the native OpenMP parallelization and it has been used to normalize the comparison with the proposed parallel approaches.

These execution times are not extremely large, but if it is considered that the wind field calculation is only one of the steps of the fire propagation prediction and, moreover, it must be executed for each value of the meteorological wind, it is mandatory to reduce the execution time as much as possible.

In order to analyze the effect of the different parallelization schemes described in section 3, one have first conducted an experiment that uses the map size of $775x775$. Two partitioning have been used: a $2x2$ and a $4x4$ with an overlap of 25 cells in both cases. So, in the first case, there are 4 partitions and in the second one, there are 16 partitions. For each case, Windninja has been executed using 1, 2 and 4 threads. The results are summarized in Table 3. These results can be

Map Size	Memory (GB)	Execution Time (seconds) # threads			
		1	2	4	8
775x775	7	205	157	136	135

Table 1. WindNinja memory requirements and Execution times using native SSOR preconditioner

Map size	SpeedUp # threads		
	2	4	8
775x775	1.30	1.50	1.52

Table 2. WindNinja speed up using native SSOR preconditioner

compared with the results shown in Table 1 that reproduces the execution time for the original WindNinja implementation for a $775x775$ cells map. It must be considered that the hardware used to run the application is just a laptop with a single processor with 4 cores and hyperthreading (8 threads). It can be observed that, for this map size, the Python multiprocessing implementation with a 2x2 partition and 4 OpenMP threads reaches an execution time of 50 seconds which represents less than 25% of the original time of 205 seconds and is clearly faster than the MPI C++ and the MPI for Python implementations. Actually, using just one OpenMP thread the execution time is just 60 seconds, which is a significant execution time improvement. However, in the case of 4x4 partitioning all the implementations are around 50 seconds (from 47 to 53) independently from the number of OpenMP threads. This is due to the fact that we start from 16 partitions and the amount of actual work involved to solve each partition is really small, and the number of cores of the system and threads support is too small. But, in any case, the execution time reached is quite good.

775x775	2x2 # Threads				4x4 # Threads			
	1	2	4	8	1	2	4	8
MPI C++	115	92	91	93	48	48	50	48
MPI for Python	108	85	81	85	47	47	49	52
Python Multiprocessing	60	53	50	57	49	48	53	52

Table 3. WindNinja Execution time for a 775x775 cells map (in seconds)

Then a $1500x1500$ cells map with a resolution of 30 meters has been considered. This map cannot be solved in the laptop used due to memory limitations (24 GBs required), so that the map partitioning strategy also solves this problem and allows to calculate the wind field. In this case, also a $2x2$ and a $4x4$ partitioning have been used, and 1, 2, 4 and 8 OpenMP threads have been tested. The results are summarized in Table 4.

1500x1500	2x2				4x4			
	# Threads				# Threads			
	1	2	4	8	1	2	4	8
MPI C++	262	222	225	225	187	179	169	153
MPI for Python	232	192	183	172	175	167	157	149
Python Multiprocessing	223	183	175	164	137	139	130	124

Table 4. WindNinja Execution time for a 1500x1500 cells map (in seconds)

In this case, the Python Multiprocessing implementation is also the faster one and reduces the execution time up to 164 seconds for the $2x2$ partition and 124 seconds for the $4x4$ partition using 8 OpenMP threads in both cases.

5 Conclusions

Wind field calculation is a critical issue to provide reliable forest fire propagation predictions. However, in the case of emergency fighting there are several constraints that should be considered. These constraints include propagation prediction time and hardware availability:

- The propagation predictions must be provided well in advance to allow the control centres to manage the field means to take the appropriate actions on the adequate time.
- It is not feasible to use extremely powerful computers by the control centres, or even the field means, but it is more feasible that they have commodity hardware available.

So, the WindNinja wind field simulator has been parallelized using three approaches based on map partitioning and tested on a single laptop. All parallel implementations reduces the execution time significantly, although the Python Multiprocessing implementation is the one that reaches the best execution time, especially for large maps. Using this parallelization it is more feasible to integrate WindNinja in an operational forest fire propagation prediction system that could be used on the field, reducing the uncertainty in propagation predictions.

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. Abdalhaq, B., Cortés, A., Margalef, T., Luque, E.: Accelerating optimization of input parameters in wildland fire simulation, Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 3019. Springer, Berlin, Heidelberg (2004)
2. Andrews, P.L.: Current status and future needs of the behaveplus fire modeling system. International Journal of Wildland Fire. **23**, 21–33 (2014). https://doi.org/10.1071/WF12167, `https://doi.org/10.1071/WF12167`
3. Brun, C., Margalef, T., Cortés, A., Sikora, A.: Enhancing multi-model forest fire spread prediction by exploiting multi-core parallelism. Journal of Supercomputing **70**(2), 721–732 (2014)
4. Brun, C., Artés, T., Margalef, T., Cortées, A.: Coupling wind dynamics into a dddas forest fire propagation prediction system. Procedia Computer Science **9**, 1110–1118 (2012)
5. Carrillo, C., Margalef, T., Espinosa, A., Cortés, A.: Accelerating wild fire simulator using GPU, Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 11540 LNCS. Springer, Berlin, Heidelberg (2019)
6. Finney, M.A.: Farsite: Fire area simulator—model development and evaluation. Research Paper RMRS-RP-4 Revised **236** (1998)
7. Forthofer, J.M., Shannon, K., Butler, B.W.: Simulating diurnally driven slope winds with windninja. In: 8th Symposium on Fire and Forest Meteorological Society (2009)
8. Ihshaish, H., Cortés, A., Senar, M.A.: Parallel multi-level genetic ensemble for numerical weather prediction enhancement. In: Procedia Computer Science. vol. 9, pp. 276–285 (2012)
9. Nocedal, J., Wright, S.J.: Conjugate gradient methods. Springer (2006)
10. Rothermel, R.: A mathematical model for predicting fire spread in wildland fuels (US Department of Agriculture, Forest Service, Inter- mountain Forest and Range Experiment Station Ogden, UT, USA 1972)
11. Sanjuan, G., Brun, C., Margalef, T., Cortés, A.: Wind field uncertainty in forest fire propagation prediction. In: Procedia Computer Science. vol. 29, pp. 1535–1545 (2014)
12. Sanjuan, G., Margalef, T., Cortés, A.: Hybrid application to accelerate wind field calculation. Journal of Computational Science **17**, 576–590 (2016)
13. Skamarock, W.C., Klemp, J.B., Dudhia, J., Gill, D.O., Barker, D.M., Wang, W., Powers, J.G.: A description of the advanced research wrf version 2. Tech. rep., DTIC Document (2005)