Parallel implementation of a simplified semi-physical wildland fire spread model using OpenMP

D. Álvarez[1,*], D. Prieto[1], M.I. Asensio[1,2], J.M. Cascón[2,3], and L. Ferragut[1,2]

[1] Departamento de Matemática Aplicada, Universidad de Salamanca, Casas del Parque 2, Salamanca, 37008, Spain
[*]{daalle,dpriher}@usal.es
[2] I. U. de Física Fundamental y Matemáticas, Universidad de Salamanca, Casas del Parque 1, Salamanca, 37008, Spain
{mas,ferragut}@usal.es
[3] Departamento de Economía e Historia Económica, Universidad de Salamanca, Edificio FES, Campus Miguel de Unamuno, Salamanca, 37007, Spain
casbar@usal.es

Abstract. We present a parallel 2D version of a simplified semi-physical wildland fire spread model based on conservation equations, with convection and radiation as the main heat transfer mechanisms. This version includes some 3D effects. The OpenMP framework allows distributing the prediction operations among the available threads in a multicore architecture, thereby reducing the computational time and obtaining the prediction results much more quickly. The results from the experiments using data from a real fire in Galicia (Spain) confirm the benefits of using the parallel version.

Keywords: OpenMP, Parallel computing, Performance, Wildland fire model

1 Introduction

Wildland fires caused by natural or human factors remain a major threat to our forests. Therefore, the real-time simulation of wildland fire spread has direct applications in prevention, fire-fighting planning, and prescribed burn planning.

There are numerous mathematical models designed to predict the spread and spatial behaviour of wildland fire events according to the nature of their construction. The types of models range from empirical models based on statistical correlations of observed fire behaviour, to theoretical or physical models based on the fundamental understanding of the physics and chemistry that govern combustion and heat transfer, covering a wide range of intermediate models: semi-physical, semi-empirical, etc. The names and classifications vary depending on the author; see for example [18, 19, 22–24]. The empirical models are simple, but applicable only to systems in which the conditions are identical to those used

in formulating and testing the models. Physical models are much more complex; their computational cost is high and their validation is extremely difficult, although they may be extrapolated to a wide variety of fire situations. Advances in computational power and spatial data analysis have improved the computational efficiency of complex models, whereby simplified physical or semi-physical models can now provide reliable and effective simulations, being considered a serious alternative to the widely used semi-empirical models.

The simulation response time of a wildland fire spread model is critical. The prediction provided by the model must obviously be much faster than the real evolution of the simulated fire event if it is to be useful for fire-fighting planning. The results of the simulation should also be reliable. Data assimilation is a technique that improves the accuracy of the predictions [8]. This method is used to feed current data into a model while it is still in simulation mode by using sequential statistical estimation. Data assimilation uses statistical methods to periodically adjust the model state, incorporating new data, with the aim of improving the simulation's accuracy. The computational cost of data assimilation is high, as it involves a large number of simulations. Data assimilation has been used previously in fire spread simulation; see for example [10] and [16]. In addition, a models's validation process includes certain steps that again involve numerous simulations, such as sensibility analysis, and especially parameter adjustment.

Today, thanks to the myriad of different types and varietiesy of data received from satellite images, real-time sensors and many other sources, the models have become more complex, with the main aim being to improving simulation accuracy. Nevertheless, a serious drawback of using such a large amount of data is the time required to compute them all and generate a prediction under demanding time constraints. It is thus of the utmost importance to significantly reduce model's simulation response time.

Thanks to advances in computation, it is becoming more common to find models exploiting multicore architectures, regardless of the model's nature. This enables computational-intensive models to generate simulations in competitive time. Techniques such as OpenMP allow fully exploiting the performance of a single machine, as it enables all the cores to be used at the same time [14]. Moreover, the GPU's advantages can exploited to improve this performance using CUDA [4]. In addition, MPI is a powerful framework that allows a task distribution over several resources in a cluster for the best performance. It is common to find a hybrid OpenMP - MPI integration [7] that achieves satisfying results in the shortest amount of time.

This paper is organized as follows: Section 2 presents an overview of the fire model optimized in this research. Section 3 details the technique used in the parallel version of the model using the OpenMP framework. Experiments and the performance study of the proposed parallel model are discussed in Section 4. Finally, the conclusions and future research are summarised in Section 5.

2 The model

The simplified semi-physical wildland fire spread model used in this research has been proposed by Ferragut *et al.* [11]. This model is based on principles of energy and mass conservation and takes into account convection and radiation as the main heat transfer mechanisms. Although the model's equations are defined on a 2D domain, the model considers some 3D effects, such as non-local radiation from the flames above the vegetal layer, for tackling with the effect that wind and slope have over flame tilt. The model also uses a multivalued operator representing enthalpy to consider the influence of fuel moisture content and heat absorption by pyrolysis. The model has been modified with several improvements, see [9, 10], with the current model version being called *Physical Forest Fire Spread* (PhFFS) [20]. The version we present here computes the radiation term by numerical integration [11], and provides a maximum improvement in computational cost with the use of parallel computation.

The model's non-dimensional simplified equations are,

$$\partial_t e + \beta \mathbf{v} \cdot \nabla e + \alpha u = r \quad \text{in } S \times (0, t_{max}), \tag{1}$$

$$e \in G(u) \quad \text{in } S \times (0, t_{max}), \tag{2}$$

$$\partial_t c = -g(u)c \quad \text{in } S \times (0, t_{max}). \tag{3}$$

We complete the problem with homogeneous Dirichlet boundary conditions and the following initial conditions,

$$u(x, y, 0) = u_0(x, y) \quad \text{in } S, \tag{4}$$

$$c(x, y, 0) = c_0(x, y) \quad \text{in } S. \tag{5}$$

The spatial domain S represents the surface where the fire occurs, defined by the mapping,

$$\begin{aligned} S: \ d &\longmapsto \mathbb{R}^3 \\ (x, y) &\longmapsto (x, y, h(x, y)) \end{aligned}$$

where $h(x, y)$ is a known function representing the topography of surface S and $d = [0, l_x] \times [0, l_y] \subset \mathbb{R}^2$ is a rectangle representing the projection of surface S. Topography is one of the data layers that the model needs, and takes from a *Geographical Information System* (GIS), see Fig. 1.

The unknowns, $e = \frac{E}{MCT_\infty}$, dimensionless *enthalpy*, $u = \frac{T - T_\infty}{T_\infty}$, dimensionless *temperature of the solid fuel* and $c = \frac{M}{M_0}$, dimensionless *mass fraction of solid fuel*, are bidimensional variables defined in $S \times (0, t_{max})$, where t_{max} is the time of study. The physical quantities E ($J\,m^{-2}$), T (K) and M ($kg\,m^{-2}$) are *enthalpy*, *temperature* of the solid fuel and *fuel load* respectively.

M_0 ($kg\,m^{-2}$) is the *initial solid fuel load*, that defines the initial condition and it is provided by two data layers, fuel load and fuel type, which the model also takes from a GIS; see Fig. 1. Fuel load layer shows where there is fuel and where there is not (roads, rivers, barren areas, firewalls set by firefighters...). Fuel type layer shows the spatial distribution of different fuel types, as initial

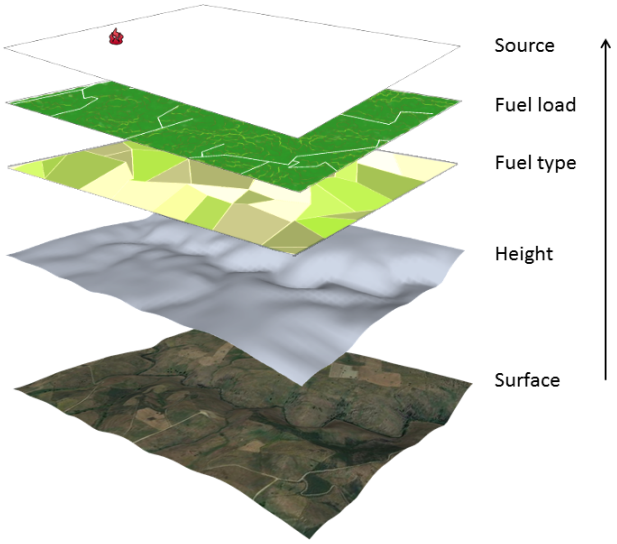

Fig. 1. Layers gathered from a surface as input data by the model.

fuel load and other model input variables depend on fuel type. The model can be fed by different fuel classifications, such as the well-known BEHAVE fuel models [2], the more recent Scott and Burgan dynamic fuel models [21], or even monitored data whenever available. C $(J\,K^{-1}\,kg^{-1})$ is the *heat capacity of solid fuel*, that also depends on fuel type. The temperature initial condition locates the fire source by another data layer (see Fig. 1).

T_∞ (K) is a reference temperature given by the *ambient temperature*, which together with wind direction and intensity are the meteorological data the model uses. The model considers wind effect in two different ways: through the convective term itself, and through the flame tilt caused by wind that affects the radiation term. The convective term, $\beta\mathbf{v}\cdot\nabla e$ in the energy conservation equation represents the energy convected by the gas pyrolyzed through the elementary control volume, where the *surface wind velocity*, $\mathbf{v}$, is re-scaled by a *correction factor* β. For a detailed explanation of the model parameter β see [20]. Wind velocity $\mathbf{v}$ can be collected from meteorological stations located close to the fire, or it can be computed by a wind model [5, 6, 12].

The model also takes into account the energy lost by natural vertical convection through the term αu in the energy conservation equation. This term is related to the *natural convection coefficient* $\alpha = \frac{H[t]}{MC}$, where H $(J\,s^{-1}\,m^{-2}\,K^{-1})$ is a model parameter representing natural convection, and $[t]$ is a time scale.

The expression $e \in G(u)$ represents a multivalued operator that models the influence of solid fuel moisture content, and depends on *fuel moisture content* M_v (kg of water/kg of dry fuel), on *dimensionless pyrolysis temperature* u_p, both related to fuel type, and on *latent heat evaporation* Λ_v.

The right-hand side of the mass conservation equation represents the loss of solid fuel due to combustion. $g(u)$ is null when $u < u_p$ and constant γ otherwise, with this constant being inversely proportional to the *half time of combustion* $t_{1/2}(s)$ of each type of fuel, $\gamma = \frac{\ln 2[t]}{t_{1/2}}$.

The right-hand side of the energy conservation equation describes the thermal radiation reaching surface S from the flame above the layer,

$$r = \frac{[t]}{MCT_\infty} R \tag{6}$$

R represents the incident energy at a point $\mathbf{x} = (x, y, h(x, y))$ of surface S due to radiation from the flame above the surface per unit time and per unit area, obtained by summing the contribution of all directions $\boldsymbol{\Omega}$, that is

$$R(\mathbf{x}) = \int_{\omega=0}^{2\pi} I(\mathbf{x}, \boldsymbol{\Omega})\boldsymbol{\Omega} \cdot \mathbf{N} \, d\omega, \tag{7}$$

where ω is the solid angle and $\mathbf{N}$ is the unit outer vector normal to surface S. This term is computed by numerical integration (see [11]) in the following 3D domain,

$$D{=}\{(x, y, z){:}\ x, y \in d, h(x, y){<}\ z{<}h(x, y) + \delta\},$$

and each contribution depends on *flame height* F and *flame temperature* T_f for each type of fuel, through the differential equation that describes total radiation intensity I at any position along a given path Ω in a grey medium, which may be written, ignoring scattering, as

$$\frac{dI}{ds} + a(s)I(s) = a(s)I_b(s). \tag{8}$$

I_b is the black body total radiation intensity, and is governed by the Stefan-Boltzmann law, corresponding to the integral over all wavelengths of the emissive power of a black body

$$I_b = \frac{\sigma}{\pi} T^4, \tag{9}$$

where $\sigma = 5.6704 \times 10^{-8} Js^{-1}m^{-2}K^{-4}$ is the Stefan-Boltzmann constant, and temperature T reaches flame temperature T_f. a is the *radiation absorption coefficient* inside the flame, and it is the third model parameter.

The numerical solution of Equations (1,2,3) is based on $P1$ finite element approximation on a regular mesh for spatial discretisation and a Crank-Nicolson finite difference scheme for time discretisation, combined with a Euler half-step for the radiation term, and the characteristic method for the convective term. For each time step, the corresponding discretised expressions for temperature, enthalpy and fuel load can be computed separately for each spatial node, so their calculation can be parallelised. The radiation term (Equation 7) is computed with numerical integration, solving the intensity ordinary differential (Equation 8) with a second-order BDF finite difference method. In order to reduce computation time, the equations are only resolved using the *active nodes* placed

around the perimeter where the fire occurs, as the operations only affect the nodes involved in the fire front, and not all the nodes in the domain.

3 Parallel model implementation

Modern computer architectures are commonly designed with multicore-processors to work at the same time. So it is increasingly common to find programs working in parallel mode, taking advantage of this feature. OpenMP [15] is an *Application Programming Interface* (API) that provides all the language extensions needed for developing parallel applications. While the MPI framework [17] is the leading technique for parallel programming in highly-parallel systems or clusters, OpenMP framework also provides the potential to develop high-performance applications, as it allows creating threads for working simultaneously on the same computer.

Our model is developed in C++, and has three main parts:

1. *Preprocessing()*. This function initializes the model and reads the data from the initial domain, including layers, input variables and model parameters.
2. *Solve()*. This function computes the simulation from the previously acquired data.
3. *Postprocessing()*. This function saves the simulation results.

Firstly, the sequential version of the model was analysed using the **Gprof** tool [13] to locate the most time-consuming functions in the code. The analysis reports an 84.1% execution time for FireSimulator:Solve() function. As this function represents the bulk of the program, achieving a better performance focused on improving this function using OpenMP. We therefore analyse the callgraph report looking for loops potentially parallelizables in the *Solve()* function . Some of these methods present a considerable number of shared variables or other inconveniences in the loop (such as loop breaking) which mean they cannot become a parallel region. After a careful review of the analysis, the parallelisable code represents approximately an 85% execution time, and the remaining 15% corresponds to the sequential code.

As Amdahl's law [1] concludes, the theoretical speed-up in latency of a program is limited by the part of the application that cannot be parallelised. The minimum execution time we could then obtain using a parallel version of the program is shown in Equation (10), where t_{seq} represents the execution time in sequential mode, and N_{th} the number of threads used. In addition, Equation (11) expresses the maximum speed we could obtain. When the number of threads N_{th} tends to infinity, the limit of the speed-up is reached, and the equation value is $1/0.15 = 6.67$.

$$t_{par}(N_{th}) = 0.15 * t_{seq} \; + \; \frac{0.85}{N_{th}} * t_{seq} \tag{10}$$

$$S_{N_{th}} = \frac{t_{seq}}{0.15 * t_{seq} + \frac{0.85}{N_{th}} * t_{seq}} = \frac{1}{0.15 + \frac{0.85}{N_{th}}} \tag{11}$$

After recoding the *for* loops into the model algorithms for independent iterations, the OpenMP language directives have been used to achieve parallelism. In this way, the **#pragma omp parallel for** directive enables the distribution of a task among the available threads speeding-up the execution. The scheduling of the iterations is *dynamic*: each thread executes a chunk of iterations then requests another chunk until none remains to be distributed. This method allows accessing consecutive elements in arrays containing the input variables. Besides that, the data-sharing attribute clause is *shared*; so the array of input variables and the results are shared among all threads. This configuration leads to a negligible increment in the global memory when the application runs in parallel mode.

A parallel implementation of the algorithms in our model requires identifying both major operations: matrix initialisation and basic matrix operations such as addition, subtraction or multiplication. Accordingly, each thread retrieves a chunk of elements, performs the assigned tasks and saves the result in a final shared matrix where all the elements are independent.

The sequential *Solve()* function is outlined in Algorithm 1. The *for* loops included in the following members of the *Solve()* function have been distributed among the available threads in a parallel version for a better performance.

Algorithm 1 Sequential *Solve()* function in the model. Sections 1 to 7 have been parallelised into threads.

Solve() {
1. Comp. of the initial values in the domain: {- *1.Parallel loop* -}
   - u_0: Dimensionless solid fuel temperature.
   - c_0: Dimensionless fuel mass fraction.
   - e_0: Dimensionless enthalpy.
2. For each simulation epoch:
   (a) Comp. of the node convection in the 2D domain. {- *2.Parallel loop* -}
   (b) Comp. of the 3D temperature for radiation comp. {- *3.Parallel loop* -}
   (c) Set active nodes in 2D domain. {- *4.Parallel loop* -}
   (d) Comp. of the radiation (for each active node):
      i. Solve the numerical integration (Eq. 7). {- *5.Parallel loop* -}
      ii. Solve the intensity ODE (Eq. 8). {- *6.Parallel loop* -}
      iii. Update values u, e, c. {- *7.Parallel loop* -}

}

4 Experiments and results

4.1 Real case study

The topographic data used in our experiments were taken from an area in Osoño (Galicia), situated in the northwest of Spain, where a real fire occurred in August,

2009. The wildfire ignited at 3.45 p.m. (local time) on August 17^{th}, 2009. Despite the firefighters' efforts to bring the fire under control in the early hours, it was not stabilised until the following day at 3.45 a.m. Finally, the fire was extinguished at 9.10 p.m. on August 18^{th}, burning a total area of $224\,ha$.

The simulation area is a surface of $3.315\,m$ x $2.740\,m$, where the altitude ranges from $540\,m$ to $680\,m$ above sea level. The average slope ranges from 6.56% at the initial point to 2.86% at the end. The fire spread over an irregular surface with discontinuous slopes, watersheds, river basins and even firebreaks. Fuel data were collected from the IFN4 database, with fuel type distribution according to BEHAVE classification [3]. Weather data were gathered from a nearby weather station $3.750\,m$ from the fire ignition, providing ambient temperature and wind data. The initial wind speed was about $11.45\,km/h$, increasing to $17.26\,km/h$, and moving from the west to the north.

The input layers in the domain are therefore transformed into an array of 663×548 nodes, as the model considers a 5 metres cell size. The model must operate over four layers, with more than $363,000$ nodes per layer, to solve the system of equations detailed in Section 2. Moreover, the real input variables and parameters detailed above must be resolved by the model, along with the numerical integration in the nodes placed around the fire area to provide the simulation result.

4.2 Performance analysis/evaluation

The main objective pursued in this research is to evaluate the potential benefit of using the parallel version of the model rather than the sequential version. To achieve this, our parallel version has been developed as detailed in Section 3 and compiled by GCC 5.4.0 using the *-O3* compilation flag with OpenMP 4.0. The experiments have been run on a workstation with a Xeon Broadwell Processor E5-1650 v4 at 3.60 GHz with six cores and 12 threads, 15 MB of cache memory and a memory of 16 GB of DDR4/2400 MHz ECC Reg. To prevent the processor from handling the frequency and obtain an adequate correlation of the data, the *Intel SpeedStep* and *Turbo Boost* technologies have been disabled.

By taking data on the Osoño fire as input data, each test was performed at least five times, increasing the number of the threads from one (sequential) to 12, as this is the maximum number of threads supported by the workstation. For the parallel time analysis, only the execution time of the *Solve()* function is considered. Table 1 summarises the average execution time collected from one to four hours of fire spread simulation time.

The results gathered from the experiments show how the execution time depends on the number of threads involved. Hence, whereas the sequential execution of one hour using one thread (sequential) took 93.8 seconds, this time decreased according to the number of threads configured in the simulation until the best time of 17.5 seconds was reached when 11 threads were used. In this case, the execution time running in parallel t_{par} among 11 threads is up to 18.75% of the sequential runtime t_{seq}.

Table 1. *Solve()* function execution time in seconds obtained for every hour simulated.

	Number of threads										
Hour	Seq.	2	3	4	5	6	7	8	9	10	11	12
1	93.8	51.8	37.3	29.8	28.5	24.2	21.0	19.8	19.5	19.0	17.5	18.0
2	229.6	130.2	86.4	74.6	62.4	54.3	48.4	46.7	44.4	43.7	41.7	42.4
3	434.4	228.5	157.2	122.6	107.9	88.8	82.8	77.8	75.2	73.8	73.0	72.5
4	653.3	355.1	241.7	186.0	154.8	137.1	126.1	118.3	113.5	111.3	109.4	108.8

Speed-up, which is defined as the ratio of sequential time t_{seq} to parallel execution time t_{par}, is detailed in Table 2.

Table 2. *Solve()* function Speed-up obtained from one to four hours of fire spread simulation.

	Number of threads										
Hour	Seq.	2	3	4	5	6	7	8	9	10	11	12
1	1	1.81	2.51	3.15	3.28	3.87	4.46	4.72	4.79	4.92	5.33	5.19
2	1	1.76	2.66	3.08	3.67	4.22	4.74	4.91	5.17	5.25	5.50	5.41
3	1	1.90	2.76	3.54	4.02	4.89	5.24	5.58	5.78	5.89	5.95	5.99
4	1	1.84	2.70	3.51	4.22	4.73	5.14	5.52	5.75	5.87	5.97	6.00

The model's parallelisation recorded a considerable difference in performance depending on the number of threads configured in the simulation, as detailed in Table 2. This table shows how the execution time for one hour of simulation significantly improves when the first six threads were used and therefore, a 74.15% (3.87) speed-up increase over the sequential time was raised. The remaining improvement, around 6%, was achieved when the rest of the threads were used: from six threads with 74.15% (3.87) over the sequential time to 11 threads with 81.25% (5.33). However, the results between 10 - 12 threads may fluctuate by less than 1%, and are not therefore meaningful because the operating system also keeps running, and affects the optimal number of threads for the best results.

As detailed in Section 3, the theoretical maximum speed-up we could obtain when the number of threads tends to infinity is defined in Equation 11, and its value is $1/0.15 = 6.67$. The results gathered from four hours of fire spread simu-

lation using 12 threads provide a 6.00 x maximum speed-up over the sequential time, which is not far from the theoretical speed.

Furthermore, in order to check how the increase in threads influences the increase in speed-up , the *parallel efficiency* $E_{N_{th}} = (t_{seq}/t_{par})/N_{th}$ (%) is evaluated, where N_{th} is the number of threads involved.

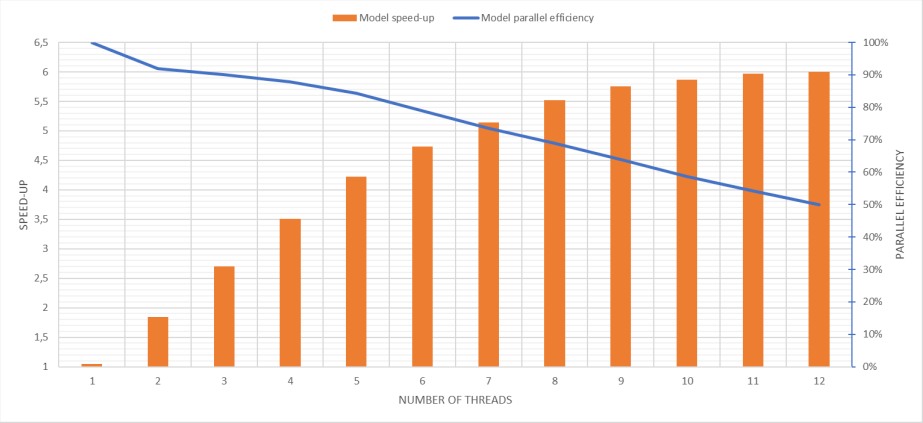

Fig. 2. Speed-up and parallel efficiency for four hours of fire spread simulation.

Fig. 2 summarises the speed-up and parallel efficiency of four hours of fire spread simulation. In this case, the speed-up increases linearly until six threads that matches with the number of the cores of the workstation. A slight increase in the speed-up has then been observed according to the number of threads (from 7 to 12) configured in the runtime. This effect is due to *Hyper-Threading* technology that allows running the concurrent scheduling of two processes per core. At this point, efficiency decreases from around 80% using six threads to 50% when 12 threads are involved. As Amdahl's law describes, when the number of threads increases, a parallel overhead appears causing a deterioration of speed-up and parallel efficiency. This drop in performance is due to the OpenMP programming methodology's loop-scheduling overhead when the model's loops are parallelised.

5 Conclusions and further research

This research describes a parallel version of a simplified semi-physical wildland fire spread model using OpenMP. This model's sequential version is analysed and improved using the **#pragma omp parallel for** directive to develop a parallel version. In this way, all the independent loop iterations can be distributed among the available threads to speed-up the execution. The experimental results for a simulation of a real fire in Osoño show an improvement in the speed-up when

the parallel version of the model is used. Nevertheless, performance efficiency decreases in step with the number of threads involved in the simulation. However, our improved parallel model computes simulations much more quickly: one hour of fire spread can be simulated in less than 50 seconds, and four hours in about four minutes.

Although our model is under continuous development, we are currently working on improving the performance reported in this work by using the CUDA framework to run the model on GPU. Future work on the model will also include a new approach for working on a cluster or in a highly-parallel system combining OpenMP and MPI frameworks. Furthermore, we are also studying other models, including wind models [12], which may exploit parallelisation capabilities for a better performance.

Acknowledgement

This work has been partially supported by the Department of Education of the regional government, the Junta of Castilla y León, Grant contract: SA020U16. The authors are also grateful to Arsenio Morillo Rodríguez chief of the *forest prevention and valorization area* of the regional government, the *Xunta de Galicia*, for his technical support providing all the necessary information about the *Osoño* fire.

References

1. G.M. Amdahl. Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities. In *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference*, AFIPS '67 (Spring), pages 483–485, New York, NY, USA, 1967. ACM.
2. H.E. Anderson. Aids to Determining Fuel Models for Estimating Fire Behavior. General Technical Report INT-122, U.S. Department of Agriculture, Forest Service, Intermountain Forest and Range Experiment Station, 1982.
3. P.L. Andrews. *BEHAVE: fire behavior prediction and fuel modeling system-BURN subsystem, Part 1.* U.S. Dept. of Agriculture, Forest Service, Intermountain Research Station Ogden, UT, 1986.
4. B. Arca, T. Ghisu, W. Spataro, and G.A. Trunfio. GPU-accelerated Optimization of Fuel Treatments for Mitigating Wildfire Hazard. *Procedia Computer Science*, 18:966 – 975, 2013.
5. M.I. Asensio, L. Ferragut, and J. Simon. A convection model for fire spread simulation. *Applied Mathematics Letters*, 18(6):673–677, 2005. Special issue on the occasion of MEGA 2003.
6. J.M. Cascón, Y.A. Engdahl, L. Ferragut, and E. Hernández. A reduced basis for a local high definition wind model. *Computer Methods in Applied Mechanics and Engineering*, 311:438 – 456, 2016.
7. A. Cencerrado, T. Artés, A. Cortés, and T. Margalef. Relieving Uncertainty in Forest Fire Spread Prediction by Exploiting Multicore Architectures. *Procedia Computer Science*, 51:1752 – 1761, 2015.
8. G. Esvensen. *Data assimilation, The Ensemble Kalman Filter.* Springer, 2009.

9. L. Ferragut, M.I. Asensio, J.M. Cascón, and D. Prieto. *A Simplified Wildland Fire Model Applied to a Real Case*, pages 155–167. Springer International Publishing, Cham, 2014.
10. L. Ferragut, M.I. Asensio, J.M. Cascón, and D. Prieto. A Wildland Fire Physical Model Well Suited to Data Assimilation. *Pure and Applied Geophysics*, 172(1):121–139, 2015.
11. L. Ferragut, M.I. Asensio, and S. Monedero. Modelling radiation and moisture content in fire spread. *Communications in Numerical Methods in Engineering*, 23(9):819–833, 2007.
12. L. Ferragut, M.I. Asensio, and J. Simon. High definition local adjustment model of 3d wind fields performing only 2d computations. *International Journal for Numerical Methods in Biomedical Engineering*, 27(4):510–523, 2011.
13. S.L. Graham, P.B. Kessler, and M.K. Mckusick. Gprof: A Call Graph Execution Profiler. *SIGPLAN Not.*, 17(6):120–126, June 1982.
14. E. Innocenti, X. Silvani, A. Muzy, and D.R.C. Hill. A software framework for fine grain parallelization of cellular models with OpenMP: Application to fire spread. *Environmental Modelling & Software*, 24(7):819 – 831, 2009.
15. M. Itzkowitz, O. Mazurov, N. Copty, and Y. Lin. An OpenMP runtime API for profiling. Sun Microsystems, Inc.. OpenMP ARB White Paper. Available online at `http://www.compunity.org/futures/omp-api.html`.
16. J. Mandel, L.S. Bennethum, J.D. Beezley, J.L. Coen, C.C. Douglas, M. Kim, and A. Vodacek. A wildfire model with data assimilation. *Mathematics and Computers in Simulation*, 79:584–606, 2008.
17. MPI Forum. Message Passing Interface (MPI) Forum Home Page. `http://www.mpi-forum.org/` (Dec. 2009).
18. E. Pastor, L. Zárate, E. Planas, and J. Arnaldos. Mathematical models and calculation systems for the study of wildland fire behaviour. *Progress in Energy and Combustion Science*, 29(2):139–153, 2003.
19. G.L.W. Perry. Current approaches to modelling the spread of wildland fire: a review. *Progress in Physical Geography*, 22(2):222–245, 1998.
20. D. Prieto, M.I. Asensio, L. Ferragut, and J.M. Cascón. Sensitivity analysis and parameter adjustment in a simplified physical wildland fire model. *Advances in Engineering Software*, 90:98 – 106, 2015.
21. J.H. Scott and R.E. Burgan. Standard Fire Behavior Fuel Models: A Comprehensive Set for Use with Rothermel's Surface Fire Spread Model. General Technical Report RMRS-GTR-153, U.S. Department of Agriculture, Forest Service, Rocky Mountain Research Station, 2005.
22. A.L. Sullivan. Wildland surface fire spread modelling, 1990-2007. 1: Physical and quasi-physical models. *International Journal of Wildland Fire*, 18(4):349–368, 2009.
23. A.L. Sullivan. Wildland surface fire spread modelling, 1990-2007. 2: Empirical and quasi-empirical models. *International Journal of Wildland Fire*, 18(4):369–386, 2009.
24. A.L. Sullivan. Wildland surface fire spread modelling, 1990-2007. 3: Simulation and mathematical analogue models. *International Journal of Wildland Fire*, 18(4):387–403, 2009.