Large Forest Fire Spread Prediction: Data and Computational Science

Tomàs Artés, Ana Cortés, and Tomàs Margalef

Computer Architecture and Operating Systems Department
Universitat Autònoma de Barcelona, Spain.

Abstract

The accurate prediction of forest fire propagation is a crucial issue to minimize its effects. So, several models have been developed to determine the forest fire propagation beforehand. Such models require several input parameters that, in some cases, cannot be known precisely in a real emergency. So, a Two-Stage methodology was developed to calibrate the input parameters to improve the quality of the prediction. This methodology was based on Genetic Algorithms which require the execution of many simulations. Moreover, when the fire is large some input parameters cannot be considered uniform among the whole fire and extra models must be introduced. One of these non-uniform parameters is wind. So, in this work a wind field model is introduced. This model implies more computation time and response time is the main constraint. The prediction must be provided as fast as possible to be useful, thus it is necessary to exploit all available computing resources. So a Hybrid MPI-OpenMP application has been developed to reach a response in the shortest possible time. This work focuses on reducing the execution time of a worker in a MPI Master/Worker structure analyzing the simulation software parts which compose the Fire Simulator System.

Keywords: error function, forest fire, simulation

1 Introduction

Natural hazards are significant problems that every year cause significant losses around the world. Floods, tsunamis, hurricanes, earthquakes or forest fires are some of these hazards. A good prediction of the behavior of the hazards is a crucial point to fight against them to minimize the damages. In the particular case of forest fire, many researchers from different disciplines have developed models to represent this phenomena in order to be able to provide a prediction of the future behavior of an emergency [13][2]. These models need several input parameters and in many cases such parameters are difficult to determine or even estimate in a real emergency occurrence. Thus, a methodology based on the observation of real fire propagation...
was developed in order to calibrate the input parameters according to real observation data. Such tuned parameters are then used on the next prediction step to drive the forecast [1]. The developed methodology took the fire behavior during a recent past time interval \((t_{\text{past}} - t_{\text{current}})\) and then, it searches for the values of the input parameters (wind speed, wind direction, moisture contents of the death fuel, moisture contents of the live fuel, vegetation features, and so on) that best reproduce this observed behavior. To carry out this search previous works rely on Artificial Intelligence techniques such as Genetic Algorithms (GA) [6], Case-based Reasoning [14], and also on statistical methods as the one described in [3]. The values of the parameters that best reproduce the behavior of the fire were then used as input parameters to predict the propagation during the near future time interval \((t_{\text{current}} - t_{\text{future}})\). This scheme requires the execution of many independent scenarios in an iterative way. Such parallel structure responds naturally to a Master/Worker programming paradigm using MPI [11]. The Master process generates the individuals of each generation, distributes the individuals among the Workers, waits for the results of each individual, ranks the individuals according to their results and generates the population for the next iteration. The Workers receive the individuals, evaluate the propagation, evaluate the error function corresponding to each individual and return these error values to the Master. This Master/Worker approach clearly provides better performance in terms of execution time since the workload caused by the execution of the huge number of fire spreads simulations is distributed among the available execution nodes. However, when fires are large the parameters are not uniform along the whole terrain and they present an spatial distribution. One of such parameters is the wind. Therefore, a model must be introduced to estimate the wind along the terrain. Next section describes how the prediction model works when coupling both models wind field model and forest fire propagation model. Section 3 shows the analysis performed to determine the performance scalability problems denoted by the worker process and the proposed improvements in terms of OpenMP [5] parallelization are also reported. Section 5 shows some experimental results and finally, in section 6 the main conclusions are reported.

2 Coupling wind field and forest fire propagation models

As we have previously mentioned, we rely on the Two-Stage prediction scheme to forecast fire spread evolution, which aims on calibrating the input parameters needed by the fire simulator to provide better quality results. Previous works [7] considered these parameters uniform along the whole evaluated map. That is, a single value for each parameter was obtained in the calibration process and used in the prediction stage without considering the dimensions of the fire. This assumption was acceptable when the methodology was applied to non-very large forest fires providing great improvements in the prediction quality but it becomes unfeasible when dealing with large scale forest fires that could burn hundreds or even thousands of hectares. It is well known that wind speed and wind direction have a great impact in fire propagation and, furthermore, their values vary according to the topography of the terrain. Therefore, it is mandatory to incorporate a wind field model in the above described Two-Stage prediction method to consider the influence of the topography in the wind values that will be fitted into the fire simulator when dealing with large forest fires. So, in the Calibration process the meteorological wind parameters of each scenario must be used to calculate the wind field corresponding to that particular scenario and then the propagation simulator must consider the wind speed and direction particular for each cell of the map. It implies that each worker process requires the calculation of the effective wind field for a given scenario before the fire spread simulation itself could be executed. This worker modification implies a significant increase in
the computing requirements. Both models, wind field model and fire spread model, use a cell map to represent the terrain/domain where the phenomenon takes place and, in order to have relevant results, it is desirable to use a high resolution mesh. This high resolution mesh requires a very detailed wind field which implies long computing time. Simplifying, the wind field model evaluates wind speed and wind direction for each cell and, the fire propagation model evaluates the fire spread direction and fire rate of spread at each cell.

The Master/Worker scheme used to implement the GA must be modified to introduce the wind field model in each worker, see figure 1. Therefore, the worker process must evaluate the wind field, the forest fire propagation model and the error function before returning the result to the Master process. This computation represents a significant amount of time, and as it has been mentioned above, time is the most relevant constraint. Therefore, to reduce the response time as much as possible the multi-core features of current processors can be exploited. So, the wind field model and forest fire propagation model has been analyzed to be parallelized using OpenMP shared memory paradigm. OpenMP presents the additional advantage that is quite simple to introduce because it only requires some parallel directives to parallelize a serial version of the application. In this way, the application becomes a hybrid MPI-OpenMP application with a MPI Master/Worker structure and OpenMP parallelism inside the Worker processes. In the next section, the analysis of each involved model and the scalability of each model considering different number of cores are shown.

3 Analysis of involved models

The integration of the wind field model implies that each worker process must evaluate the wind field before the forest fire propagation can be calculated. So, the internal structure of each Worker process involves three main parts, the wind field estimation, the propagation calculation and the error evaluation (see the red box in figure 1). It is necessary to execute each worker in the shortest possible time to reach the best possible calibration and carry out the prediction accomplishing the real time constraints. Therefore, the main goal is to reduce

Figure 1: Coupling Wind Field model and Fire Propagation model
the execution time instead of focusing on having a good efficiency. In the application context, parallel code which allows to reduce the execution time taking advantage of non used hardware is welcome.

3.1 Analysis Testbed

As it has been mentioned, the aim of this work consists of reducing the execution time of whole prediction process by focusing on reducing the worker process execution time. For this purpose, we rely on multi-core platforms with the objective of assigning more than one core to each worker what is supposed to reduce the worker’s execution time. In order to take advantage of this capacity, it is mandatory to determine whether it is worthy or not to run the simulations in more than one core. Previous works [4] studied the impact of the input parameters values into the simulation execution time. Figure 2 shows the histogram of the simulation’s execution time obtained when 12000 different scenarios where evaluated for a certain topography using FARSITE simulator [8]. As we can see, there are a non negligible number of input configurations that provides execution times higher than 100 seconds. In order to properly evaluate our contributions in terms of time savings, it is worthy to select test input parameters sets whose associated simulation time is high enough to detect the most relevant performance problems and, consequently, be able to provide parallel solution which represents a significant time reduction.

For this reason, we have selected as a test bed terrain, a real terrain located at the north-east part of Spain (Catalonia). In particular, we have selected Cap de Creus landscape because it is the most affected area in Catalonia. The different maps used, such as elevation, fuel or slope, are raster maps where each map has 1004304 cells with a 30m$^2$ of surface represented per cell. Thus, the simulation runs in a scenario of 3012, 912ha. The fire perimeters resolution remains constant when working with bigger maps, this fact allows the random appearance of individuals in the GA with large fire perimeters. Since larger perimeters become slower individuals, in large scale forest fire simulations, the GA generations could be delayed by the slower execution of a worker with a large perimeter. As a test input set of parameters, we have selected several

![Figure 2: Histogram of execution time of FARSITE with 12000 randomly generated input parameter set.](image)
sets whose associated simulation time is higher than 350 seconds in order to consider higher workloads.

The execution platform that has been used for the performance analysis experiments is a Dell Presicion T7500 with two Intel Xeon E5645 (6 cores per CPU) and 96GB of DDR3 SDRAM at 1333Mhz in 12 DIMMS. This test bed platform allows us to analyze the scalability of the proposed solution from 1 core up to 12 cores.

3.2 Wind field Model

In this work, we use WindNinja [12] to compute terrain effects on wind flow. WindNinja requires elevation data for the modeling area, a domain-mean initial wind speed and wind direction, and specification of the dominant vegetation in the area.

The native parallel version of WindNinja is implemented using OpenMP and it is available for linux. WindNinja offers two main ways to exploit OpenMP parallelism. On the one hand, there is the possibility to calculate different wind fields based on different general winds on the same topographic map. On the other hand, WindNinja could also exploit the multi-threading parallelism by distributing the calculation of a single wind field among different threads. In this work, we are interested in this second approach.

After running the test simulation described in section 3.1 using 1 core up to 12 cores, the results show that WindNinja exhibits a bad scalability. In particular, the time incurred when using 1 core is 300 seconds, whereas the execution time using 12 cores was 150 seconds. Although this time reduction represents a non-negligible time saving, it denotes performance problems. For this reason, WindNinja was deeply profiled in order to find the cause of such performance penalty in terms of scalability. The WindNinja profile was done using OmpP and the obtained results show that most execution time is spent in linear solver functions, which work with compressed spare row (CSR) matrix. The most time consuming functions are two functions which compute the matrix vector multiplication (38.23% of the total execution time) and the scalar vector multiplication (35.24% of the total execution time). This kind of matrix representation needs auxiliary data structures in order to traverse the matrix, that fact makes harder the parallelization. In some of that functions, some SIMD SSE4.2 instructions have been introduced obtaining an improvement, over a 20% when using 4 cores as it is shown in figure 3. Therefore, this new capability was introduced in WindNinja, although a wider analysis of the mathematical libraries used remains open to diagnose more carefully the real problem.

3.3 Forest Fire Propagation Simulator

The Forest Fire Propagation Simulator used was FARSITE [8]. The available FARSITE Linux version is a serial code, therefore, any improvement in terms of parallelizing this simulator is a good contribution. For this reason, we concentrated on reducing the execution time of FARSITE by taking advantage of the multi-core capability of the muti-core nodes. FARSITE has been analyzed using gprof [10] under different workloads to find the most time consuming functions. This analysis showed that the exclusive execution time is quite distributed all over the functions except two functions. CrossThread::Cross() function was consuming over a 21.86% and GetPerimeter1Value() over a 14.88% of inclusive execution time. We could try to parallelize this functions using OpenMP but with a poor improvement due to inclusive execution time of the functions. So, we search over the callgraph for other functions with parallelizable loops which include the calls to those functions. We found a function, Intersections::CrossCompare(), which includes over a 60% of inclusive execution time and is in a higher level in the callgraph.
but in the same branch where the heavy inclusive execution time functions are located. This function includes a loop that can be parallelized using OpenMP.

The theoretical maximum speed up is limited by the part of the application which can be parallelized. Since the parallel part is around 60% of the total execution time, the parallel version would achieve a theoretical minimum of 40% of the execution time. In figure 4, it can be observed that the obtained results are quite close to the predicted ones. After the parallelization of FARSITE, OmpP [9] profiling tool has been used to get serial and parallel region execution time. The serial part of FARSITE lasts around 130 seconds. And the parallelizable part lasts around 239 seconds on a single core. On 2 cores the parallel part is reduced to 121 seconds. Using 4 cores, the time was reduced to 61 seconds and using 12 cores the parallel part lasted less than 25 second. Figure 4(b) depicts the just mentioned speed up results. Therefore, if we consider all together the serial plus the parallel part of FARSITE, the obtained execution time was reduced from 360 seconds to less than 160 seconds. This result is quite good considering the degree of parallelization introduced (over 60%). Bearing in mind that the perimeters are represented with points and lines, such as polygon, and considering that the functions dealing with those points traverse the perimeters checking the points to ensure that the shape is coherent then, for each time step control, the cross between outer perimeters and the point correctness is needed. This task in done by correcting the crossed perimeter by merging perimeters and control the outer points of the perimeter. Finally, to control the shape coherency, the function uses loops which traverse all fires, the perimeters of the fire and the points of the perimeter. So, it can be concluded that working with high precision and large forest fires, which propagate quickly, the complexity of these functions will cause slower executions. Thus, the perimeter size is related to the execution time and it could happen that some input parameters in the GA produce a fast fire propagation with larger perimeters and slower executions. The new parallel version of FARSITE has been validated with different large scale forest fires providing exact output files than the ones obtained using the original serial version of FARSITE.

4 Worker Execution time

Once we have analyzed the two main components of the worker workflow (WindNinja and FARSITE), we studied the behavior of the worker process as a whole. As it has been mentioned,
the GA generates different sets of inputs parameters, which represent the individuals of a given population. The execution time of the fire simulator depends on the provided input parameter set. The execution time of the workload for a given individual could stand out from the rest individuals of the generation causing a slower generation time, see histogram of FARSITE execution for different individuals in figure 2. These kind of individuals could make a generation slower. Thus, it is very convenient to be able to execute high workload individual faster than others. For this reason, all the workloads used in the present work represent a similar workload to the slower individuals that could be produced in the GA as has been commented in section 3.1. Since the time used to compute the error is negligible compared to the rest of the worker execution time, we only consider the time spent by WindNinja and FARSITE. As can be seen in figure 5(a), the total execution time for the workload considered, has been reduced from 650 seconds to 300 seconds. Furthermore, it can be observed that is faster to run two different serial individuals in two different cores than two parallel individuals using two cores per individual in a sequential way. One individual executed on 2 cores lasts more than 460 seconds and two of such executions take more than 900 seconds, while two serial individuals executed concurrently on 2 separate cores lasts 650 seconds. However, normally there are more cores than individuals per generation, so the method described represents a significant improvement.

5 Experimental results of the hybrid application

In the Master/Worker scheme, the execution time of the last worker represents a barrier which determines the generation time. Thus, reducing the worker execution time can represent a significant improvement. As it has been shown in previous section, exploiting the multicore feature of actual processors reduces the worker execution time and, therefore should accelerate the two stage prediction methodology. In order to determine the actual improvement on the two stage prediction methodology time using multi-core processors under a hybrid MPI-OpenMP Master/Worker structure, an experimental test has been carried out. Since the Forest Fire propagation simulator has a strong variability in the execution time (see Figure 2) this has been the only component of the worker considered by now. The main point is that the execution time of one worker can vary depending on the input parameters of the forest fire simulator and this variable behavior can produce unexpected results that should be analyzed. So, a set of executions of the two stage prediction system considering the Forest Fire has been launched. An initial population of 6 individuals is generated by the Master process and then the GA evolves.
the population for 10 generations. The 10 populations are stored in order to be used with a different number of cores (1 and 4). The experiments were executed on an IBM cluster where each node is a x3550 with 2 x Dual-Core Intel(R) Xeon(R) CPU 5160 @ 3.00GHz 4MB L2 (2x2) and 12 GB Fully Buffered DIMM 667 MHz. Figure 5(b) shows accumulated execution time of 10 generations using 1 and 4 cores for each worker. The final generation time represents the total execution time of the calibration stage. Thus, using 1 core for each worker the execution of the ten generations lasts 1623.83 seconds and using 4 cores for each worker the execution time is 887.04 seconds. This first comparison shows that exploiting the 4 cores of the architecture can reduce the execution time a 45% of the single core execution time. That could mean a significant help to avoid the hard time constraint which is implicit in the application context.

It can be observed that the execution time of each generation is not constant, but varies significantly from generation to generation. This fact is due to the variable execution time of forest fire propagation simulator. In this case, it can be observed that generations 2, 3, 4 and 9 lasts significantly more time than the other generations. Using 4 cores reduces the execution time of each generation, but since the genetic algorithm has a random component it is necessary to repeat the same experiment with different initial populations. Then, the experiment has been repeated 15 times using different initial populations in the GA, that means a total number of 150 generations. These generations have been executed with 1 core and 4 cores. The average of the improvement for all the generations is a 38.12% with a standard deviation of 10.41.

The improvement in the generation time can be compared to the serial execution time. As it is shown in figure 6(a) the percentage of improvement increases with the serial execution time. When the serial execution time is very short (few seconds) the improvement is negligible. When the execution time is about 10 seconds, the improvement is around 15-20%. The improvement increases up to 40% when the serial execution time is around 30 seconds. For 100 seconds hereinafter improvement begins to level off over the 45%, but after that the improvement is almost constant. The multi-core nodes are able to reduce the execution time up to 45% of the total time. As it has been stated in section III.C FARSITE has a serial part of 40% of its executing time and 60% that can be parallelized. Then, if the parallel part is executed on 4 cores, that part could last around 15% of the total execution time (60/4) and, therefore, the total execution time on 4 cores could be around 55% of the total execution time (40%+15%). Thus, the expected improvement in the execution time using 4 cores could be 45% which is confirmed by the experimental results when the worker time is significant. In figure 6(b), the deviations of the actual improvement compared to the expected ones are shown.

Figure 5: Worker execution time (a) and Comparison between calibration time using 1 and 4 cores per worker (b).
executions are longer than 30 seconds the deviation is below 0.1 and the improvement is over 40%. When an individual is faster than 10 seconds is better not use more than 1 core and use more cores for higher workloads. This fact introduces a way to alleviate the problem of load balancing in the GA using a Master/Worker scheme.

6 Conclusions

In the context of forest fires, when a fire starts, every second counts. Using a GA with Master/Worker MPI scheme we can obtain good predictions, thanks to the tune of the parameters sets which are not available in real emergency cases. Such a calibration method has a high computational cost and, furthermore, the results must be delivered under hard time constraint. Nowadays, multi-core architectures provide more computing resources that must be exploited. The way that we present to improve the tuning method is based on adding multi-core parallelism at the worker level using OpenMP. Therefore, the time spend on each worker can be significantly reduced. Thus, we can reduce the execution time of each generation of the GA and, in turn, two-stage prediction scheme perform faster. Consequently, the GA can iterate more generations in the same time, which implies a better calibration. This fact gives the opportunity to get more accurate calibrated results increasing the generations in the GA. Also, we show that the use of profiling tools and OpenMP applied to coupled models can be useful to optimize or provide new multi-core capabilities to serial models. Finally, it can be concluded that the multi-core capability introduces a way to manage the load unbalance. Using a characterization of the worker, which classifies the workers depending on the different input parameters set used in the GA and using a decision tree, is now possible to provide more or less cores improving the utilization of the hardware and reducing significantly the total execution time.

Acknowledgments

This research has been supported by MINECO-Spain under contract TIN2014-53234-C2-1-R and by the Catalan government under grant 2014-SGR-576.
References

[1] Baker Abdalhaq, Ana Cortés, Tomàs Margalef, and Emilio Luque. Enhancing wildland fire prediction on cluster systems applying evolutionary optimization techniques. *Future Generation Computer Systems*, 21(1):61–67, 2005.

[2] F.A. Albini, Intermountain Forest, Utah) Range Experiment Station (Ogden, and United States. Forest Service. *Estimating wildfire behavior and effects*. General technical report INT. Dept. of Agriculture, Forest Service, Intermountain Forest and Range Experiment Station, 1976.

[3] G. Bianchini, M. Denham, A. Cortés, T. Margalef, and E. Luque. Wildland fire growth prediction method based on multiple overlapping solution. *Journal of Computational Science*, 1(4):229–237, 2010.

[4] Andrés Cencerrado, Ana Cortés, and Tomàs Margalef. Genetic algorithm characterization for the quality assessment of forest fire spread prediction. *Procedia Computer Science*, 9(0):312–320, 2012. Proceedings of the International Conference on Computational Science, ICCS 2012.

[5] B. Chapman, G. Jost, and R. Van Der Pas. *Using OpenMP: portable shared memory parallel programming*, volume 10. The MIT Press, 2007.

[6] M. Denham, A. Cortés, and T. Margalef. Computational steering strategy to calibrate input variables in a dynamic data driven genetic algorithm for forest fire spread prediction. *Computational Science–ICCS 2009*, pages 479–488, 2009.

[7] Mónica Denham, Ana Cortés, Tomàs Margalef, and Emilio Luque. Applying a dynamic data driven genetic algorithm to improve forest fire spread prediction. In Marian Bubak, Geert van Albada, Jack Dongarra, and Peter Sloot, editors, *Computational Science – ICCS 2008*, volume 5103 of *Lecture Notes in Computer Science*, pages 36–45. Springer Berlin / Heidelberg, 2008. 10.1007/978-3-540-69389-5_6.

[8] M. A. Finney. *FARSITE, Fire Area Simulator–model development and evaluation*. Res. Pap. RMRS-RP-4, Ogden, UT: U.S. Department of Agriculture, Forest Service, Rocky Mountain Research Station. 1998.

[9] K. Fürlinger and M. Gerndt. : A profiling tool for openmp. *OpenMP Shared Memory Parallel Programming*, pages 15–23, 2008.

[10] Susan L. Graham, Peter B. Kessler, and Marshall K. McKusick. *gprof: a call graph execution profiler*. *SIGPLAN Not.*, 39(4):49–57, April 2004.

[11] William Gropp. Mpich2: A new start for mpi implementations. In Dieter Kranzlmüller, Jens Volkert, Peter Kacsuk, and Jack Dongarra, editors, *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, volume 2474 of *Lecture Notes in Computer Science*, pages 37–42. Springer Berlin / Heidelberg, 2002.

[12] Brian Potter and Bret Butler. Using wind models to more effectively manage wildfire. *Managementtoday*, page 40, 2009.

[13] R.C. Rothermel. *How to predict the spread and intensity of forest and range fires*. Intermountain Forest and Range Experiment Station Ogden, 1983.

[14] Kerstin Wendt, Ana Cortés, and Tomàs Margalef. Knowledge-guided genetic algorithm for input parameter optimization in environmental modelling. *Procedia Computer Science*, 1(1):1367–1375, 2010. Proceedings of the International Conference on Computational Science, ICCS 2010.