Core Allocation Policies on Multicore Platforms to Accelerate Forest Fire Spread Predictions

Tomàs Artés(✉), Andrés Cencerrado, Ana Cortés, and Tomàs Margalef

Computer Architecture and Operating Systems Department,
Universitat Autònoma de Barcelona, Campus UAB, Edifici Q,
08193 Bellaterra, Spain
{tomas.artes,andres.cencerrado,ana.cortes,tomas.margalef}@uab.cat
http://caos.uab.es

Abstract. Software simulators are developed to predict forest fire spread. Such simulators require several input parameters which usually are difficult to know accurately. The input data uncertainty can provoke a mismatch between the predicted forest fire spread and the actual evolution. To overcome this uncertainty a two stage prediction methodology is used. In the first stage a genetic algorithm is applied to find the input parameter set that best reproduces actual fire evolution. Afterwards, the prediction is carried out using the calibrated input parameter set. This method improves the prediction error, but increments the execution time in a context with hard time constraints. A new approach to speed up the two stage prediction methodology by exploiting multicore architectures is proposed. A hybrid MPI-OpenMP application has been developed and different allocation policies have been tested to accelerate the forest fire prediction with an efficient use of the available resources.

Keywords: Forest fire · Simulation · Data uncertainty · Hybrid MPI-OpenMP · Evolutionary computation · Resource assignment · Multicore architecture

1 Introduction

Some natural hazards involve serious consequences from the environmental, economic and social point of view. Therefore, it is critical to react as soon as possible to minimize their effects. This work focuses on forest fire spread prediction. This kind of natural disasters are among the most worrisome in southern European countries. To deal with this hazard, models which describe the forest fire spread have been developed and implemented in simulators. In the case of forest fire, the Rothermel model [1] is one of the most used and proven. FARSITE [2] is a widely used simulator which implements this model. To perform a simulation, it requires a set of parameters that describes the environment where the fire is taking place. These input parameters may present several difficulties: some of them are not uniform along the forest fire scenario, others can present a temporal variability, others must be estimated by interpolated measures, others must

R. Wyrzykowski et al. (Eds.): PPAM 2013, Part II, LNCS 8385, pp. 151–160, 2014.
DOI: 10.1007/978-3-642-55195-6_14,

be obtained from complementary models. This fact results in a certain degree of uncertainty in input data that provokes a lack of accuracy in the prediction. A calibration method has been used to reduce the uncertainty of the input data set [3]. In this method the prediction is based on two stages where firstly a calibration is carried out and afterwards the prediction is done. The first stage consists on searching the input parameters set that best reproduces the actual fire behavior. This search is carried out applying a Genetic Algorithm (GA) [4] over a certain number of iterations. Once the preset number of iterations is reached, the best set of parameters is used to carry out the prediction for the following time step. Although this prediction approach provides better results in terms of quality degree, it is more computing demanding. The increase in the time needed to obtain a satisfactory prediction result could not always be a feasible approach when dealing with emergencies. For this reason, the calibration stage has been implemented using a message passing Master/Worker paradigm to take advantage of parallel/distributed systems. The two stage prediction scheme was originally designed to run each fire spread simulation using a single core. In previous studies, it has been stated that every fire spread simulation lasts shorter or longer depending on the particular setting of the input parameters. Actually, the time needed to complete every generation of the calibration stage is bounded by the worker that lasts longer. Therefore, the just mentioned master/worker scheme using a single core per worker could eventually generate unbalance among workers. For this reason, the kernel of the used simulator (FARSITE) has been parallelized using OpenMP in order to reduce execution time. In this context, it is not worthy dedicating the same amount of resources to short and long simulations. Therefore, a methodology to characterize FARSITE which allows us to assess, beforehand, the execution time of a given scenario has been developed. This ability enables the possibility of designing core allocation policies to minimize the total execution time of the prediction scheme and, to use the available resources more efficiently. In Sect. 2, the two stage methodology and its implementation are described. Subsequently, in Subsect. 2.1 the results of the kernel parallelization are detailed. In Sect. 3, the methodology used to detect slower kernel executions is briefly presented. The results of the hybrid MPI-OpenMP with different core allocation policies are shown in Sect. 4. Finally, conclusions and future work are described in Sect. 5.

2 Hybrid MPI-OpenMP Master/Worker Prediction Scheme

A simulator independent data-driven prediction scheme is used to calibrate the input data set provided to the simulator [3]. For this purpose, a previous calibration step is introduced, as can be seen in Fig. 1. So, the input data set used for the prediction stage is calibrated in this first stage for each prediction step. Based on the hypothesis that the meteorological conditions will not suddenly change from the calibration stage to the prediction stage, the calibrated data set could be used to produce a more accurate prediction.

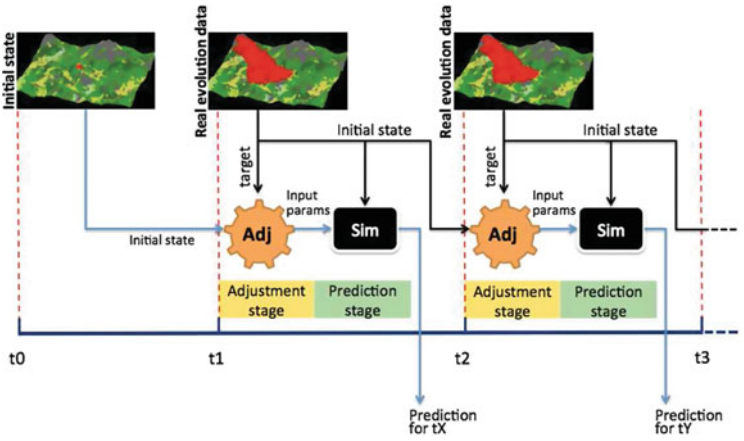


Fig. 1. 2 stage prediction method

Because of their outstanding results within this framework [5] this work is based on the use of GA as calibration technique. The algorithm starts from an initial random population of individuals, each one representing a scenario to be simulated. An individual is composed of a number of different genes that represent input variables such as dead fuel moistures, live fuel moistures, wind speed and direction, among others. Each individual is simulated and it is evaluated comparing the predicted and real fire propagation by estimating the fitness function described in Eq. 1. This fitness function computes *the symmetric difference between predicted and real burned areas.*

$$Difference = \frac{UnionCells - IntersectionCells}{RealCells - InitCells} \tag{1}$$

In Eq. 1, *UnionCells* is the number of cells which describe the surface burned considering predicted fire and the real fire. *IntersectionCells* is the number of cells burned in the real map and also in the predicted map, and *RealCells* are the cells burned in the real map. *InitCells* is the number of cells burned at the starting time. This difference takes into account the wrong burned cells and the mistaken for burned cells. According to this fitness function the whole population is ranked and the genetic operators *selection*, *elitism*, *mutation* and *crossover* are performed over the population, producing an evolved population which will have, at least, the best individual of the last generation (elitism). The new population is then evaluated in the same way. This iterative process allows us to find a good input parameter set, but it involves high computational cost due to the large amount of simulations required. Therefore, it is essential to speed up the execution keeping the accuracy of the prediction. For this reason, an implementation of the two stage methodology has been developed using High Performance Computing techniques. Since the GA fits the Master/Worker paradigm, an MPI implementation has been developed. At the first stage, the

master node generates an initial random population which is distributed among the workers. Then, the workers simulate each individual and evaluate the fitness function. The errors generated by the workers are sent to the master which sorts the corresponding individuals by their error before applying the genetic operators and producing a new population. This iterative process is repeated a fixed number of times. The last iteration (generation) contains a population from which the best individual is taken as the best solution, and then it is used in the prediction stage. Since every simulation can be carried out in a parallel way, the individual whose simulation takes longer determines the elapsed time for that particular generation. In order to shorten simulation times, FARSITE has been analyzed with profiling tools such as OmpP [6] and gprof [7] to determine which regions of the code could be parallelized with OpenMP. The result of such analysis determined the particular loops that could be parallelized using OpenMP pragmas. The results of such parallelization have been presented in [8]. The parallelized loops represents about 60 % of one iteration execution time. It means that 40 % of the iteration execution time is sequential and it implies that the speed up is not linear, but is limited by such sequential part.

2.1 Evaluating the Hybrid Scheme

In order to have an snap shot of the Hybrid Master/Worker scheme potential, we performed two preliminary test experiments. For this purpose, we used as a terrain, a geographical zone of high risk of forest fire located in the North-East of Spain, *Cap de Creus*. In this terrain, a synthetic fire has been simulated using as input setting information provided by the local meteorological centre SMC (Catalan Meteorological Service) obtained from the Automatic Weather Station Network (XEMA). The vegetation types has been obtained from CORINE land cover data base [9]. The obtained fire spread was used as a real fire for comparison purposes. The computational platform used was a 32 nodes IBM x3550 cluster where each computing node has two dual core Intel Xeon 5160 and 12 GB of memory at 667 Mhz. The first experiment consists of executing the hybrid Master/Worker prediction scheme using a random initial population of 25 individuals, which was evolved 10 generations using one core for each individual. The resulting evolved populations obtained at each iteration of the GA, are recorded and reused again but using 4 core per individual. This way, both cases use the same individuals at each iteration and they only differ in the cores assigned to each individual. Figure 2 depicts the calibration errors evolution through obtained over the evolution of the population and shows the elapsed time to execute the 25 individuals of each iteration. As it was expected, the execution time is reduced significantly when the number of cores is increased. For example, the total execution time is reduced from 72693 s to 27174 s. The error evolution is consistent because the error for the next generation must be equal, as least, to the last generation. This is due to the elitism genetic operator. The best individual is introduced without changes to the next population to evaluate. If crossover or mutation operator does not create a new individual better than the individual chosen by elitism, there are no error improvement. This fact

produces a stair like behaviour when plotting the error for each generation. Bearing in mind the results obtained in this preliminary study, to increase the number of cores assigned to FARSITE simulations (individuals of the GA algorithm) could not always be a benefit in terms of final execution time. In particular, for those short simulations (several seconds) that will not bound the duration of a given generation, it will not be useful to allocate more than one core. Therefore, in order to be able to determine how many cores to assign to each FARSITE simulation (individual evaluation in the GA scheme), it is necessary to be able to asses before running the simulation, its execution time. In the next section, we describe a methodology to characterize FARSITE that allows us determining in advance the duration of a given FARSITE simulation for a certain input settings. In particular, the estimated execution time is defined by and interval time called class.

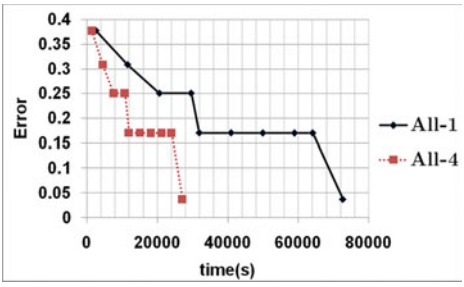


Fig. 2. Execution time and error considering 1 and 4 cores

3 FARSITE Characterization

As it has been mentioned, the execution time of a single simulation on the same map and simulating the same time can vary from seconds to several minutes or even hours depending on the input settings of the fire spread simulator. Consequently, in order to design core allocation strategies, we need to be able to anticipate the execution time of a certain input setting without the necessity of running that simulation. For this purpose, we propose a real time strategy to rapidly assess for a given input scenario (simulator input setting) an execution interval time where the corresponding FARSITE simulation time will fit. This strategy has been successfully tested for several forest fire spread simulators as is reported in [10]. To be operative during a real hazard, this execution-time estimation of a given scenario must be inferred as quickly as possible, keeping the cost of carrying out this operation to a minimum, in terms of time needed. For this reason, we rely on the field of Artificial Intelligence to be able to automatically learn from stored knowledge, so as to provide smart decisions. In particular, we use Decision Trees to extract this knowledge from certain database. The FARSITE characterization (or simulator kernel characterization) is fulfilled

by means of carrying out large sets of executions (on the order of thousands) counting on different initial scenarios (different input data sets), and then, applying knowledge-extraction techniques from the information they provide, i.e. we record the execution times from the experiment, and then we establish a classification of the input parameters according to the elapsed times they produced. Specifically, we follow this sequence of steps:

- *Training database building*: Currently, we work with training databases composed of 12000 different scenarios.
- *Determination of execution time classes*: The whole training database is executed, and every par [scenario, execution time] is recorded. After this, the histogram of execution times is analyzed. Identifying the local minimums of the histogram the upper and lower boundaries of each *execution time class* are determined. Figure 3 depicts the histogram of execution times obtained for FARSITE. From the analysis of this histogram the resulting classes are the following:
  - Class A: ET $\leq$ 270 s.
  - Class B: 270 seconds $<$ ET $\leq$ 750.
  - Class C: 750 seconds $<$ ET $\leq$ 1470 s.
  - Class D: 1470 seconds $<$ ET $\leq$ 3600 s.
- *Decision Tree building*: once we have determined how many classes we will consider, then we are ready to build the Decision Tree. For this purpose, we rely on the C4.5 algorithm, specifically, the J48 open source Java implemented in the Weka data mining tool [11].

These steps are carried out off-line, in a preparation phase before any real emergency is under analysis. Once this methodology has been followed, only one final step remains: the application of the built Decision Tree with the scenario describing the ongoing fire, in order to assess in advance the execution time its simulation will produce. This action supposes a negligible cost, in terms of time overhead (on the order of a few seconds) and enables the ability of deciding at real-time how many cores allocate to each simulation depending on its estimated execution time class. The results of applying Decision Trees were validated in [12], where it is demonstrated that we reach 96.4 % correct classifications. In the following section, we reported an experimental study where different core allocation strategies have been analyzed in terms of speedup an efficiency.

4 Experimental Study

By means of the simulator kernel Characterization described in Sect. 3, we are able to identify those individuals that will last longer in a given iteration of the Hybrid Master/Worker prediction scheme. This fact allows us to take advantage of the OpenMP parallel version of FARSITE, determining a real-time core-allocation strategy for each individual in a generation in order to save as much

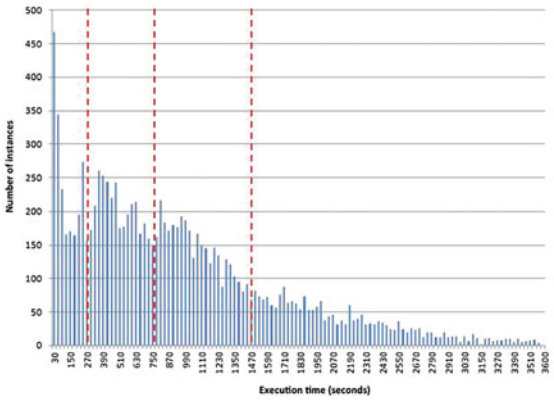


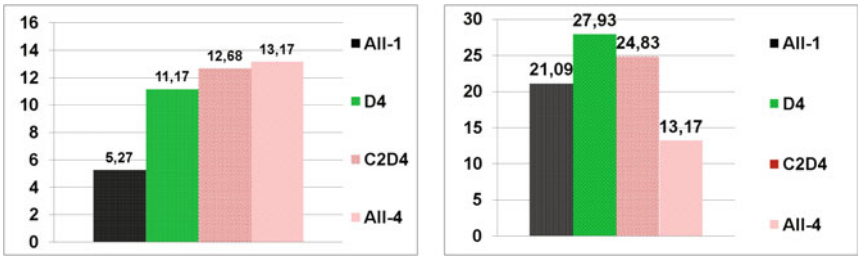
Fig. 3. Histogram of execution times using FARSITE. vertical dotted lines indicate the defined classification boundaries.

overall execution time as possible. Thus, in this section, the benefits of applying the FARSITE characterization together with the developed Hybrid Master/Worker prediction scheme are evaluated. For this purpose, we conducted an experimental study consisting of applying different core-allocation policies and analyzing the obtained speedup and efficiency. The execution platform and the terrain where the experiments have been performed are the same that the ones used in Sect. 2. The experiments carried out consist of executing the Hybrid Master/Worker prediction scheme using 9 different initial populations each one composed of 25 individuals and the GA iterates 10 generations. As it was done in the preliminary study reported in Sect. 2.1, the intermediate populations obtained at each generations for the case of one single core allocation per simulations, have been recorder to be able to reproduce the same evolution with different allocation policies which are listed following:

- *All-1*: The basic case, where each individual is allocated to one core.
- *C2D4*: Individuals belonging to class C are allocated to 2 cores. Individuals belonging to class D are allocated to 4 cores. The rest of individuals are allocated to one core.
- *D4*: Individuals belonging to class D are allocated to 4 cores. The rest of individuals are allocated to one core.
- *All-4*: All the individuals are allocated to 4 cores. It is worth mentioning that, for this experimental study, the workers are limited to execute only an individual per generation. This fact reduces the efficiency because parts of the workers are waiting for the slowest simulation to end.

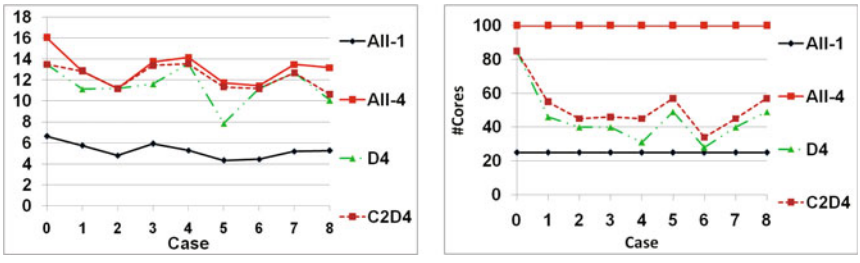
It is worth mentioning that, for this experimental study, the workers are limited to execute only an individual per generation. This fact reduces the efficiency because some workers are waiting for the slowest simulation to end but it ensures seeing the effects of the different core-allocations. Figure 4(a) shows the results obtained. This figure presents the average speedup (sequential execution

time/parallel execution time) obtained by applying the 4 different allocation policies. As it was expected, policy *All-4* was the most favorable in terms of speedup and it is useful to compare to the speedup reached by the other policies. It can also be observed that policy *C2D4* is the most close to the maximum speedup obtained by policy *All-4*. Policy *D4* gets a slightly lower value than policy *C2D4*. This means that there are cases in which C individuals execution time become the slowest ones executed with one core and this fact determines the execution time. Thus, in such cases, allocating C individuals to 2 cores would be suitable. However, if the efficiency (speedup/number of cores) is considered, *All-1* strategy presents a very poor efficiency since it needs 100 cores to be executed. So, it results in a very poor resource utilization. Figure 4(b) shows the average efficiency for each policy. Between the extreme policies *All-1* and *All-4*, executions of policy *D4* are using significantly less cores than policy *C2D4* achieving a minimally less speedup. It can be observed that assigning strictly 4 cores to D individuals (policy *D4*) provides better efficiency than policy *C2D4* where C individuals are assigned 2 cores. Figure 5(a) depicts the obtained speed up values for each case. As can be seen in Fig. 5(a), policies *D4* and *C2D4* almost reach the same speed up as the one obtained when allocating every simulation to 4 cores. In some cases, such as 1, 3 and 5, shortening class D individuals may cause that class C individuals become the lengthiest simulation at each generation. This



(a) Average Speedup. (b) Average Efficiency.

Fig. 4. Speedup and Efficiency for the 4 scheduling policies.



(a) Speedup per case (b) Core usage per case

Fig. 5. Speedup and core usage for the 4 allocation policies.

can be observed in Fig. 5(b). The benefit obtained by the application of these simple policies can also be analyzed. For instance, considering cases 4 and 6, we are able to obtain almost the same speedup as the most favorable case, in terms of execution time (policy *All-4*) by adding only 3 and 6 cores, respectively. In general terms, policy *C2D4* requires only 7 cores extra to be implemented, but the gain is relevant.

5 Conclusions and Future Work

Nowadays, the existing forest fire spread simulation tools provide us with more accurate results as well as the ability to use more complex simulation features. However, the predictions based on a single-simulation strategy still presents the serious disadvantage of not being able to effectively deal with the uncertainty related to the context of an environmental emergency management. In this paper, we describe two kind of uncertainties which may lead us to inaccurate predictions: uncertainty in the values of the input parameters of the simulator, and uncertainty in the time needed to deliver a fire spread prediction. In order to deal with the former, we describe a two-stage prediction framework based on the iterative calibration of the input parameters according to the actual evolution of the fire. The later issue is tackled by means of an Artificial Intelligence technique (specifically, Decision Trees) to classify in advance the different sets of input parameters according to the expected execution time they will produce. However, while these methods clearly benefits the consecution of our goal, the need arises to address the problem of efficient computational resources exploitation. The proposed calibration technique for the two-stage prediction framework, GA, presents outstanding results as regards the improvement in the quality of our predictions, but it is very computational demanding, since it needs the execution of multiple simulations at the same time. Taking advantage of the fact that we are able to estimate how long a simulation will last before its execution, we propose to rely on this technique to decide how to allocate the different simulations to the available resources. The results obtained in the reported experimental study demonstrate the great benefit we obtain, mainly in terms of absolute time savings in the prediction process, by means of the application of simple core-allocation policies. This allows us to set out the main question of our ongoing work: the implementation of an intelligent scheduling system for the optimization of the available resources in order to enhance the quality of our predictions as quick as possible. For instance, it would be an important advantage to be able to dynamically group the fastest simulations in subsets of computational resources, allocating the slowest ones to other dedicated subsets, according to the specific needs of each case. Ongoing work is also related to add to our methodologies the capability to automatically adapt to new computing resources appearance in real time. The results obtained open up these new challenges with good expectations and a guaranteed background.

Acknowledgements. This research has been supported by MICINN-Spain under contract TIN2011-28689-C02-01.

References

1. Rothermel, R.C.: A mathematical model for predicting fire spread in wildland fuels. Director (INT-115), 40 p (1972)
2. Finney, M.A.: 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)
3. Abdalhaq, B., Cortés, A., Margalef, T., Luque, E.: Enhancing wildland fire prediction on cluster systems applying evolutionary optimization techniques. Future Gener. Comput. Syst. **21**(1), 61–67 (2005)
4. Denham, M., Wendt, K., Bianchini, G., Cortés, A., Margalef, T.: Dynamic data-driven genetic algorithm for forest fire spread prediction. J. Comput. Sci. **3**(5), 398–404 (2012). (Advanced Computing Solutions for Health Care and Medicine)
5. Denham, M., Wendt, K., Bianchini, G., Cortés, A., Margalef, T.: Dynamic data-driven genetic algorithm for forest fire spread prediction. J. Comput. Sci. **3**(5), 398–404 (2012)
6. Fürlinger, K., Gerndt, M.: ompP: a profiling tool for OpenMP. In: Mueller, M.S., Chapman, B.M., de Supinski, B.R., Malony, A.D., Voss, M. (eds.) IWOMP 2005 and IWOMP 2006. LNCS, vol. 4315, pp. 15–23. Springer, Heidelberg (2008)
7. Graham, S.L., Kessler, P.B., McKusick, M.K.: gprof: a call graph execution profiler. SIGPLAN Not. **39**(4), 49–57 (2004)
8. Artés, T., Cencerrado, A., Cortés, A., Margalef, T.: Relieving the effects of uncertainty in forest fire spread prediction by hybrid mpi-openmp parallel strategies. Procedia Comput. Sci. **18**(2013), 2278–2287 (2013). (International Conference on Computational Science)
9. Bossard, M., Feranec, J., Otahel, J.: CORINE land cover technical guide Addendum 2000. Technical report No 40, European Environment Agency, Kongens Nytorv 6, DK-1050 Copenhagen K, Denmark (2000)
10. Cencerrado, A., Rodríguez, R., Cortés, A., Margalef, T.: Urgency versus accuracy: dynamic driven application system natural hazard management. Int. J. Numer. Anal. Model. **9**, 432–448 (2012)
11. Holmes, G., Donkin, A., Witten, I.H.: Weka: a machine learning workbench. In: Proceedings of the 1994 Second Australian and New Zealand Conference on Intelligent Information Systems 1994. pp. 357–361. IEEE (1994)
12. Cencerrado, A., Cortés, A., Margalef, T.: On the way of applying urgent computing solutions to forest fire propagation prediction. Procedia Comput. Sci. **9**, 1657–1666 (2012)